Deep Neural Networks for Voice Control

by Loren Peter Lugosch

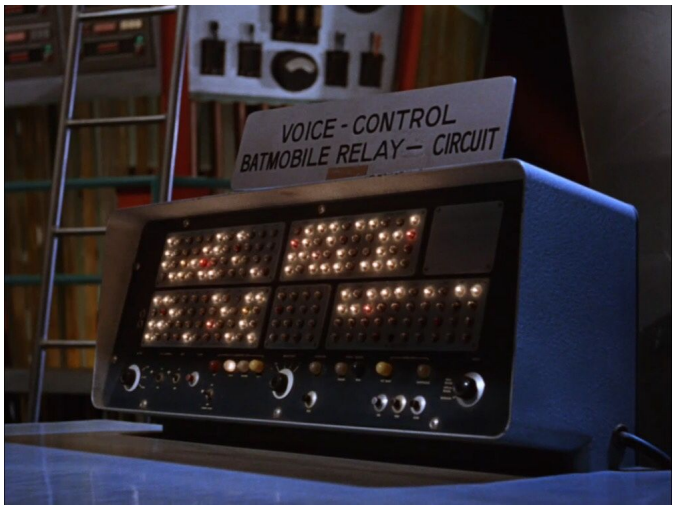
Department of Electrical and Computer Engineering

McGill University

Montreal, Quebec, Canada

Spring 2023

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Doctor of Philosophy



ROBIN
I get it! You're gonna try to remote control the Batmobile circuit via the Batcave relay link.

BATMAN
Right, Robin. It's our only chance.
(speaking into radio)
Batman to Batcave. Voice-Control Batmobile Relay-Circuit, switch on.

The VOICE-CONTROL BATMOBILE RELAY-CIRCUIT begins beeping.

BATMAN
Batmobile Ejection Seat, fire.

The beeping gets faster. Then the VOICE-CONTROL BATMOBILE RELAY-CIRCUIT goes silent. BATMAN's radio down-chirps dejectedly.

ROBIN
What the heck! The failure signal.

BATMAN
Circuit's on the blink, I'm afraid.

ROBIN
How can it be?

BATMAN
Human mechanisms are made by human hands, Robin. None of them is infallible. It's a lesson which must be faced.

— Batman, Season 1 Episode 28: "The Pharaoh's In A Rut"

ABSTRACT

Voice control systems enable people to control their computers by speaking to them. After a review of the state-of-the-art in sequence modeling, speech recognition, and language understanding using deep learning, this thesis describes a number of contributions to the art of voice control. The first contribution is a study of large-scale semi-supervised learning through pseudo-labeling for massively multilingual speech recognition. The second contribution is a study of the use of autoregressive models for conditional computation with neural networks, using speech recognition as a test case. The third contribution is a method for training end-to-end spoken language understanding models using speech synthesis. The fourth contribution is a crowdsourced dataset, Timers and Such, for spoken language understanding involving numbers, along with baseline experimental results and open-source software infrastructure for using the dataset. The fifth contribution is our part in the design and implementation of SpeechBrain, an open-source software toolkit for speech processing. Finally, using some of the tools and techniques developed earlier in the thesis, we propose a simplified and unified approach to voice control in which the entire traditional pipeline, composed of an automatic speech recognition subsystem, a natural language understanding subsystem, and human-programmed control logic, is subsumed within a single deep neural network.

ABRÉGÉ

Les systèmes de commande vocale permettent aux utilisateurs de contrôler leur ordinateur en leur parlant. Après un examen de l'état de l'art en matière de modélisation de séquences, de reconnaissance vocale et de compréhension du langage à l'aide de réseaux de neurones profonds, cette thèse décrit un certain nombre de contributions à l'art du contrôle vocal. La première contribution est une étude de l'apprentissage semi-supervisé à grande échelle par pseudo-étiquetage pour la reconnaissance vocale massivement multilingue. La deuxième contribution est une étude de l'utilisation de modèles autorégressifs pour le calcul conditionnel avec des réseaux de neurones, en utilisant la reconnaissance de la parole comme cas test. La troisième contribution est une méthode d'entraînement de modèles de compréhension de langue parlée de bout en bout à l'aide de la synthèse vocale. La quatrième contribution est un ensemble de données crowdsourcé, Timers and Such, pour la compréhension du langage parlé impliquant des nombres, ainsi que des résultats expérimentaux de base et une infrastructure logicielle open source pour l'utilisation de l'ensemble de données. La cinquième contribution est notre rôle dans la conception et la mise en œuvre de SpeechBrain, une boîte à outils logicielle open source pour le traitement de la parole. Enfin, en utilisant certains des outils et techniques développés plus tôt dans la thèse, nous proposons une approche simplifiée et unifiée du contrôle vocal dans laquelle l'ensemble du pipeline traditionnel, composé d'un sous-système de reconnaissance automatique de la parole, d'un sous-système de compréhension du langage naturel et logique de contrôle, est subsumé dans un seul réseau de neurones profonds.

ACKNOWLEDGMENTS

First, thanks to my supervisors Brett Meyer and Derek Nowrouzezahrai: to Brett for teaching me what research is and how to do it in his ECSE 541 class back in Fall 2015, for his Nintendo-infused friendship (e.g., completing my *Animal Crossing: New Horizons* fruit inventory), and for his refreshing insistence that every AI problem be seen as a multi-objective optimization problem; to Derek for his boundless energy, "connectionism" (by which I mean both his neural network optimism and his ability to facilitate connections between researchers), and good cheer, which was inspiring to me even during the worst days of the pandemic; and to both for giving me a long leash and writing me a never-ending stream of recommendation letters. Thanks also to Jackie Cheung and Ioannis Psaromiligkos for taking the time to be on my committee.

In the Reliable Silicon Systems Lab, thanks to Jarul Mehta, Derek Yu, and Guillaume Richard for letting me intrude on their channel for detecting intrusions on channels; Alex Orzechowski for giving me an outlet for procrastination by assisting with his demand forecasting work; and Adithya Lakshminarayanan, Marihan Amein, and Paul Xiong for enjoyable coffee chats on neural architecture search.

At Mila, my biggest thanks go to Mirco Ravanelli for hooking me up with an internship there when I emailed him out of the blue, for later insisting on me joining the SpeechBrain project despite my waffling, and for our years of fruitful collaboration; Kyle Kastner, João Felipe Santos, and Dima Serdyuk for, along with Mirco, welcoming me into the audio fold when I first arrived; the incomparable Olexa Bilaniuk for keeping me abreast of the latest exotic hardware accelerators, debugging my dynamic programming gradients, and helping me with his vast Linux knowledge on many occasions, including once spending an entire evening remotely guiding me through the painful process of building `ffmpeg` on the Mila cluster; Shawn Tan, Sasha Luccioni, and Joseph Viviano for delightful machine learning pun exchanges; Ethan Caballero for preaching to me the good word of scaling laws; David Yu-Tung Hui for sharing his extensive language grounding Google Doc with me; and Yoshua Bengio, Doina Precup, and Simon Ramstedt for giving me helpful feed-

back when I presented my voice control ideas at the RL Sofa as an RL noob. Thanks also to my other colleagues on the SpeechBrain team — Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Cem Subakan, Nauman Dawalatabad, Abdelwahab Heba, Jianyuan Zhong, Ju-Chieh Chou, Sung-Lin Yeh, Szu-Wei Fu, Chien-Feng Liao, Elena Rastorgueva, François Grondin, William Aris, Hwidong Na, Yan Gao, and Renato De Mori — for the pleasure of our weekly meetings.

During my time as an engineer at Fluent.ai before the PhD, Vikrant Singh Tomar and Sam Myer took me from knowing a little bit about MFCCs and HMMs to knowing how to actually build practical spoken language understanding systems. They also taught me the importance of considering the system as a whole, as opposed to just the transcription part — to which this entire dissertation is a testament. Thanks also to Arash Rad, Christoph Conrads, Farzaneh Fard, Robert Johnson, Luis Rodríguez Ruiz, and Mohamed Mhiri for the joy of our time spent together there trying to trick wakeword detectors ("OK, Poodle") and tussling with each other for GPU time.

Thanks to Gabriel Synnaeve, Ronan Collobert, and Tatiana Likhomanenko for supervising (or should I say "semi-supervising"?) me during my three-time-zone-spanning Summer 2021 internship at Facebook AI Research. An additional thanks to Ronan for, afterwards, coping with the bot that deleted my directory and continuing to train our models, right up to the ICASSP deadline, to Vineel Pratap for open-sourcing our final model and Flashlight code, and to Chan Woo Kim and Patrick von Platen for the PyTorch port.

Thanks to the McGill Engineering Doctoral Award (MEDA), the Fonds de Recherche du Québec Nature et Technologies (FRQNT), and the National Sciences and Engineering Research Council (NSERC) for funding my work.

Thanks to Montreal friends Harsh Aurora, Andrey Tolstikhin, Margot Silberblatt, Scott Dagondon Dickson, Dylan Watts, Winee Lutchoomun, Dirk Dubois, Patrick Ignoto, Sean Grogan, and Marwan Kanaan for the picnics, presentation parties, canoe camping, sugar shacking, and other delightful debauchery.

And thanks to Ed Schipul for letting me work at Tendenci way back in Summer 2011, despite my being comically underqualified for it at the time, and to JMO, Glen Zangirolami, Eloy Zuniga, and Jenny Qian, working with whom taught me Python, Linux, Git, and other

skills I used every day during my PhD.

Finally, thanks to my family for everything.

If I forgot to acknowledge you here, reader, forgive me; know that you have nonetheless been absorbed implicitly into these pages, like a training example into the weights of a deep neural network.

CONTRIBUTION OF AUTHORS

All data, plots, and text are my own original work, unless otherwise specified within the text.

LIST OF ABBREVIATIONS

- AM — acoustic model
- ASR — automatic speech recognition
- CER — character error rate
- CNN — convolutional neural network
- CTC — connectionist temporal classification
- FBANK — Mel filterbanks
- GMM — Gaussian mixture model
- GPU — graphics processing unit
- HMM — hidden Markov model
- IPL — iterative pseudo-labeling
- LM — language model
- NLU — natural language understanding
- RNN — recurrent neural network
- SLU — spoken language understanding
- WER — word error rate

LIST OF FIGURES

Contents

Chapter 1

Introduction

This thesis considers the problem of **voice control**. We define "voice control" as the task of mapping a speech signal to a sequence of actions that the speaker wants to be performed. We can define "actions" rather broadly: for instance, one action a voice control system might take is to respond to the speaker in natural language, in which case the system is referred to as a "dialog" system. Other examples of actions a system might take include actuating physical devices (in response to e.g. "Open the pod bay doors, HAL"), setting values in a computer application (e.g. "Set a timer for five minutes"), retrieving information from a knowledge base (e.g. "Who's the Prime Minister of Canada?"), or pondering an existential question (e.g. "Siri, what am I doing with my life?").

Current voice control systems work well for simple tasks like setting timers or opening pod bay doors, but in many ways they fall short of the capabilities and flexibility we might expect from an actual human assistant. For instance, the thesis author tried asking the following question of his iPhone voice assistant:

> Est-ce que je peux parler Franglais when I make a Siri request?

which was transcribed as

> Squish Papale throng lay when I make a Siri request

and pawned off onto a Google search instead of being understood. In another instance, the following unfruitful dialog occurred:

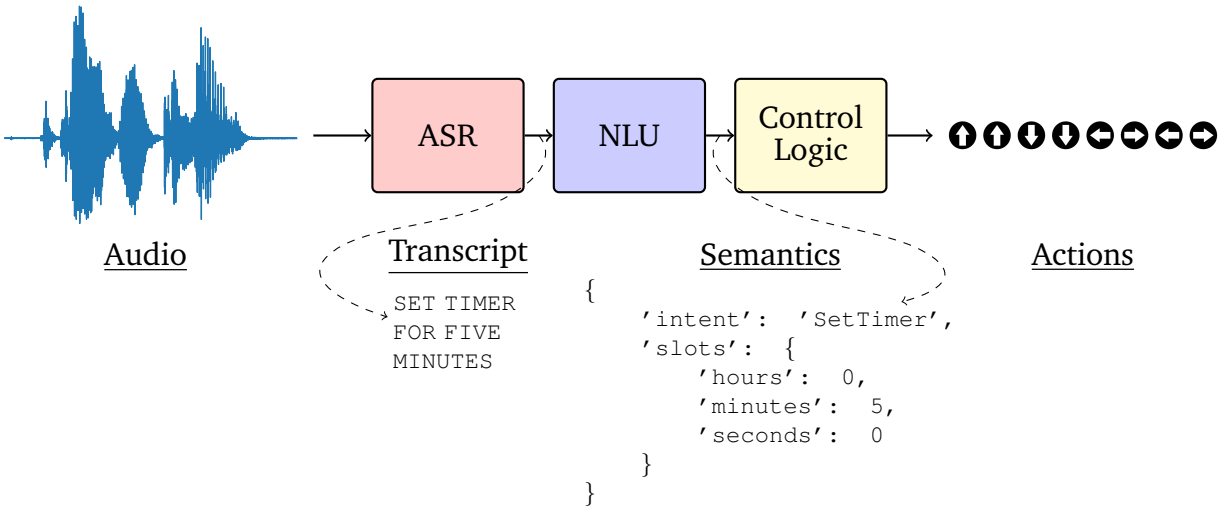


Figure 1.1: The voice control pipeline.

[*Loren:*] Create a note called “Brainstorming” for me.

[*Siri:*] OK, I’ve created a note called “Brainstorming For Me”.

[*Loren:*] Change the title of the note to “Brainstorming”.

[*Siri:*] What do you want me to add?

[*Loren:*] I don’t want to add anything. Just change the title of the note to “Brainstorming”.

[*Siri:*] OK, I added “I don’t want to add anything just change the title of the note to brainstorming” to your note “Brainstorming For Me”.

How might we go about solving these problems? Let’s start by examining the way voice control is currently implemented. A diagram of a typical voice control system is shown in Fig. 1.1. The system is composed of an automatic speech recognition (ASR) module, a natural language understanding (NLU) module, and a control logic module. The ASR module transcribes the speech signal, producing a sequence of words called the “transcript”. The NLU module parses the transcript to extract the meaning of the utterance in the form of a data structure referred to as the “semantics”. The control logic module inserts the values of the semantics into some application, resulting in some actions being performed.

Many of the flaws of state-of-the-art voice control systems implemented using the pipeline in Fig. 1.1 stem from the fact that they are ultimately *rule-based systems* — what Batman referred to in the epigraph of this thesis as "human mechanisms [...] made by human hands". Conventional wisdom once held that advanced artificial intelligence (AI) would be explicitly programmed by humans: in the 1983 movie *Star Wars: Episode VI — Return of the Jedi*, for instance, the droid C-3P0 says "it's against my programming to impersonate a deity!" The philosopher Hubert Dreyfus has argued that the idea of implementing intelligence by programming rules goes all the way back to Plato, in whose dialogues Socrates confronts various Athenians to ask what rules they follow when determining, e.g., what is "just" [8].

But it seems that, *pace* Socrates, we humans do not follow rules most of the time: instead, we act on unconscious knowledge and pattern recognition. *Machine learning* attempts to emulate humans' unconscious ability to find and act on patterns. In machine learning, instead of having rules programmed by a human, a computer learns its program from experience, using examples of the correct behavior, or using a teacher that gives rewards for good behavior. It has gradually become apparent that the solutions to some AI problems are simply too difficult for humans to program and might be solved using machine learning instead. ASR and NLU in the conventional voice control pipeline are two such problems: we train an ASR system by showing it examples of correctly transcribed audio signals, and we train an NLU system by showing it examples of correctly semantically labeled sentences.

Moreover, the *extent* to which ASR and NLU are learned rather than programmed has grown. The AI pioneer Fred Jelinek is famously reported to have said that "every time I fire a linguist, the performance of our speech recognizer goes up." Whereas earlier ASR systems required orchestrating a dazzling variety of techniques — Mel frequency cepstral coefficient feature extraction, Gaussian mixture models, simple framewise neural network acoustic models, hidden Markov models, forced alignment, phonetic pronunciation dictionaries, context-dependent triphones, state tying, weighted finite state transducers, search algorithms, n-gram language models, hypothesis rescoring using recurrent neural network language models — it is now possible to train a *deep neural network* to map directly from

the speech waveform to the letters of the transcript and attain near state-of-the-art results. Deep neural networks — for the reader who has been living under a rock — are brain-like statistical models that extract many layers of features from an input to implement a desired function with minimal engineering required. With the growth of data through the Internet and the growth of compute power through Moore's Law, we now have enough of both ingredients to make many interesting AI capabilities possible using deep learning.

Unlike ASR and NLU, control logic (the third box in Fig. 1.1) is not implemented using machine learning because writing the program to do what a user wants is generally trivial given the semantics. But are we using the right semantics? Suppose that a user makes a request slightly more complicated than the one in Fig. 1.1, like "set a timer for five minutes — and beep twice when there's one minute left". Unless the designer of the NLU system has thought to include that possibility in the semantic structure, there is no chance the system will be able to handle it (and indeed no voice assistant that we have tested has been able to handle this particular example). It is also questionable whether the division of labor in Fig. 1.1 is correct in the first place. In the same way that ASR can now use a single end-to-end-trained neural network instead of a collection of separately trained or programmed modules, could we not use a single deep neural network to implement both ASR and NLU at the same time, or even go all the way and implement the entire pipeline, from audio to actions? What advantages and disadvantages might these end-to-end approaches have compared to the conventional pipeline?

This thesis makes a number of contributions to the art of voice control. We begin with ASR, then move on to spoken language understanding (SLU; = ASR + NLU), and finally consider the problem as a whole. Throughout the thesis, we try to apply a simple problem-solving method:

1. If labeled data is unavailable for a task, make it (or fake it).
2. Then train a deep neural network to implement the task directly.

The thesis runs as follows:

- In Chapter 2, we provide background knowledge on machine learning and the speech and language technology used in subsequent chapters.

- In Chapter 3, we show how to make use of unlabeled audio for ASR in a challenging multilingual setting. Work described in this chapter was published as:
  [9] **L. Lugosch**, T. Likhomanenko, G. Synnaeve, R. Collobert, "Pseudo-Labeling for Massively Multilingual Speech Recognition", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 7687-7691, Singapore, May 2022.
  Contribution of co-authors: T. Likhomanenko helped with code and computing infrastructure and the use of slimIPL; G. Synnaeve helped with code and edited the paper and suggested experiments; R. Collobert debugged code and ran experiments on behalf of the thesis author and edited the paper and suggested experiments.

- In Chapter 4, we show how autoregressive models can be used to reduce the computational cost of neural networks for streaming applications, using ASR as a test case. Work described in this chapter was published as:
  [10] **L. Lugosch**, D. Nowrouzezahrai, B. H. Meyer, "Surprisal-Triggered Conditional Computation with Neural Networks", *arXiv*, June 2020.
  Contribution of co-authors: D. Nowrouzezahrai and B. H. Meyer edited the paper and suggested experiments.

- In Chapter 5, we move from ASR to SLU, and show how the lack of labeled audio for end-to-end SLU can be overcome using speech synthesis. Work described in this chapter was published as:
  [11] **L. Lugosch**, B. H. Meyer, D. Nowrouzezahrai, M. Ravanelli, "Using Speech Synthesis to Train End-to-End Spoken Language Understanding Models", *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 8499-8503, Barcelona, Spain, May 2020.
  Contribution of co-authors: B. H. Meyer, D. Nowrouzezahrai, and M. Ravanelli edited the paper and suggested experiments.

- In Chapter 6, we fill a gap in existing open source SLU datasets with the design and creation of Timers and Such, an SLU dataset for numeric commands. Work described in this chapter was published as:

[12] **L. Lugosch**, P. Papreja, M. Ravanelli, A. Heba, T. Parcollet, "Timers and Such: A Practical Benchmark for Spoken Language Understanding with Numbers", *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS) — Track on Datasets and Benchmarks*, pp. 1-11, (virtual), December 2021.
Contribution of co-authors: P. Papreja implemented the recording website; M. Ravanelli edited the paper and implemented the ASR encoder; A. Heba and T. Parcollet implemented the LibriSpeech ASR recipe.

- In Chapter 7, we describe SpeechBrain, a PyTorch toolkit for speech processing, and the state-of-the-art SLU recipes we contributed to it. Work described in this chapter was published as:
  [13] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, **L. Lugosch**, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J. Chou, S. Yeh, S. Fu, C. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. De Mori, Y. Bengio, "SpeechBrain: A General-Purpose Speech Toolkit", *arXiv*, June 2021.
  Contribution of co-authors: SpeechBrain is a large software project with many contributors; most of the functionality described in Section 7.1 was designed and implemented by other contributors. The thesis author's main contributions are the SLU recipes.

- In Chapter 8, we move beyond SLU and sketch the outline of a fully neural end-to-end voice control system, with no hardcoded semantics or control logic, and describe some preliminary experimental work in this direction.

- Chapter 9 concludes.

Chapter 2

Background

Abstract

We assume that the reader is familiar with linear models for regression and classification and how they can be trained through maximum likelihood estimation using stochastic gradient descent. We do not assume that the reader knows anything about sequence modeling, deep learning, speech recognition, or language understanding, and in this chapter provide enough background on those topics for the rest of the thesis to make sense.

2.1 Inputs and outputs

2.1.1 Waveforms

Speech is represented in computers as a digital signal. The signal is recorded by sampling the output voltage of a microphone at a regular interval (say, 16,000 times per second) and using an analog-to-digital converter to store the audio sample using a finite amount of memory (say, 16 bits). Some voice-controlled devices, such as Google Home [14], use multiple microphones to record a multi-channel signal. In this thesis, we consider only the single-microphone setup with a one-channel output. Fig. 2.1 shows an example of the type of speech signal, or waveform, that we will deal with.

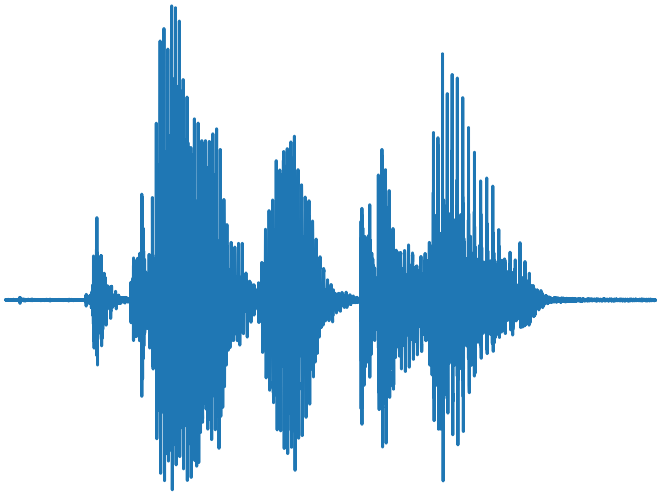


Figure 2.1: A 16 kHz recording of the thesis author saying the phrase “it’s all Greek to me”.

2.1.2 Spectrograms

Speech signals can be understood using the “source-filter” production model: the speaker’s vocal cords produce a simple “excitation” (a sequence of pulses for voiced sounds, or random noise for unvoiced sounds), which is then filtered by the vocal tract to produce the acoustic signal [15, 16]. The vocal tract, much like the Internet, can be seen as a series of tubes, where the dimensions of each tube control a set of resonant frequencies called “formant frequencies”. Fig. 2.2 shows English vowel sounds plotted by their first two formant frequencies. The vowels are nicely separated in this space: for instance, the /i/ sounds (top left) can be separately perfectly from the /u/ sounds (bottom left) using a linear classifier.

For that reason, speech is often processed in the frequency domain. Formant frequencies, and other useful features, can be extracted from a spectrogram representation. A spectrogram is computed by extracting overlapping segments (often 25 ms long, and shifted by 10 ms) from the raw waveform, computing the windowed short-time Fourier transform (STFT) [17], and taking the log magnitude of each resulting complex value. The spectrogram computed from the speech signal in Fig. 2.1 is shown in Fig. 2.3.

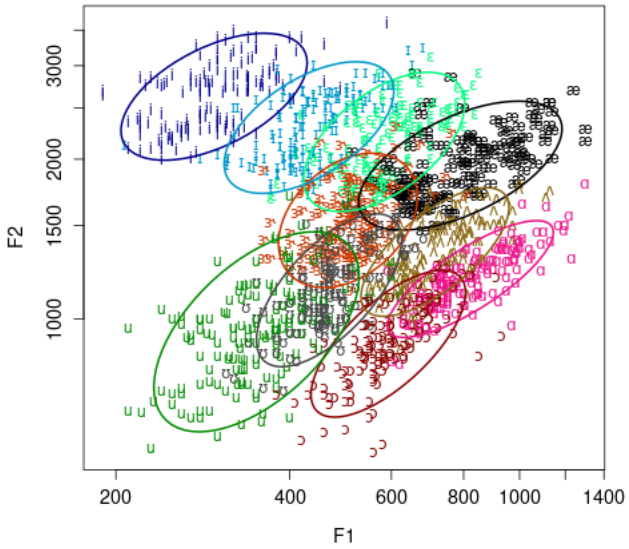


Figure 2.2: Scatter plot of English vowel sounds from 76 speakers [1]. (Plot generated using phonTools [2].)

2.1.3 Filterbanks

While some ASR systems use the spectrogram as input [18], it is more common to first compress the spectrogram using a set of Mel scale filterbanks. The Mel scale is a logarithmic mapping from frequencies to units that are thought to match human auditory perception [19]. Mel filterbank (or FBANK) features are extracted by computing a set of triangular windows spaced across the frequency spectrum according to the Mel scale and adding up the energy within each window. A sequence of FBANK features is shown in Fig. 2.4.

Older speech recognizers go a step further and compute *cepstral* features called Mel frequency cepstral coefficients (MFCC) using the inverse discrete cosine transform. ("Cepstral" terminology inverts "spectral" terminology: *spectral* $\rightarrow$ *cepstral*, *frequency* $\rightarrow$ *quefrency*, *filter* $\rightarrow$ *lifter*.) MFCC features are less correlated than FBANK features, which is useful for Gaussian mixture model (GMM)-based acoustic models, but not for neural network acoustic models, for which FBANK features yield better performance [20].

Some more recent speech models, such as wav2letter [21] and wav2vec [22], do not

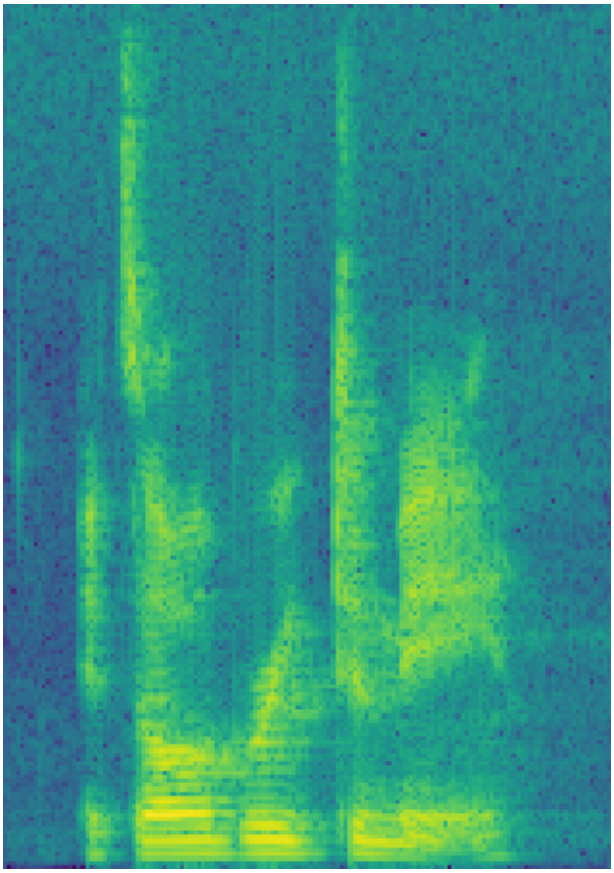


Figure 2.3: Spectrogram of the thesis author saying “it’s all Greek to me” (computed using `speechbrain.processing.features.spectral_magnitude`).

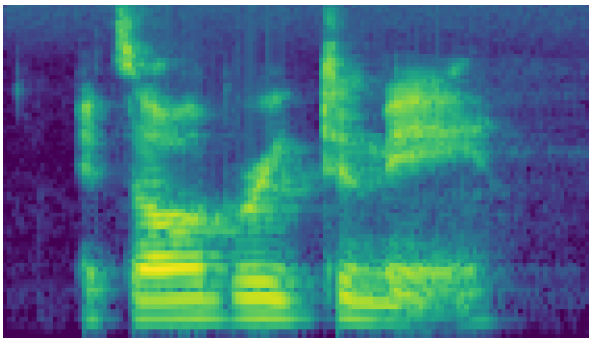


Figure 2.4: FBANK representation of the thesis author saying “it’s all Greek to me” (computed using `speechbrain.processing.features.Filterbank`).

perform any kind of hardcoded feature extraction and instead learn to extract features directly from the raw waveform [23, 24]. SincNet [25] is a sort of hybrid approach: it uses sinc filters with trainable frequency parameters to process the input signal. Hardcoded FBANK features are still a strong baseline and are used throughout much of this thesis.

2.1.4 Phonemes and graphemes

Phonemes are speech sounds that distinguish words. For instance, the English words

heed, hid, head, had, hod, hawed, hood, who'd, hud, heard

are each composed of three phonemes: /h/, a vowel sound, and /d/.

Linguists draw a distinction between *phonemes* (which are meaning-contrastive within a specific language) and *phones* (which are not). Speech technology researchers often use the two terms interchangeably, a practice that has been criticized by some [26]. In this thesis we too will cheerfully ignore this linguistic distinction and use the word "phoneme" the way it is used in the speech technology literature, trusting that the reader will be able to infer the sense of the word in context, the way one might infer the sense of other such highly polysemic words as "model", "neuron", "decoder", and "batch".

Whereas phonemes are spoken, graphemes — a.k.a., letters, or characters — are written: "who'd" consists of the graphemes "w", "h", "o", "'", and "d". A single grapheme does not necessarily correspond to a single phoneme, nor vice versa. Languages differ in orthographic regularity: whereas Spanish has a nearly perfect correspondence between phonemes and graphemes, English orthography is particularly irregular: e.g. in the words

rough, cough, dough, bough, through

the grapheme sequence "ough" corresponds to a different phoneme sequence in each case (/rʌf/, /kɑf/, /doʊ/, /bæʊ/, and /θru/, respectively, using the International Phonetic Alphabet (IPA)).

The output of a speech recognizer ultimately needs to be graphemes. Because of orthographic irregularity, it is often significantly more difficult for a machine learning system to learn to "spell" than to learn to recognize phonemes [27, 28].

2.1.5 Words and subwords

In English and many other languages, written words are separated by spaces. In other languages like Chinese, there are no spaces in writing, so words are not explicitly indicated. A space between written words usually does not correspond to silence between spoken words.

Words are composed of meaning-bearing units called morphemes: e.g., the English word "learning" is composed of the free morpheme "learn" and the bound morpheme "-ing". Words are not always formed by concatenating morphemes: a morpheme composed of a sequence of phonemes (e.g. Arabic "ktb": "to write") may be interleaved with other phonemes to form different words (e.g. "kitab": "book"; "yaktubu": "he writes").

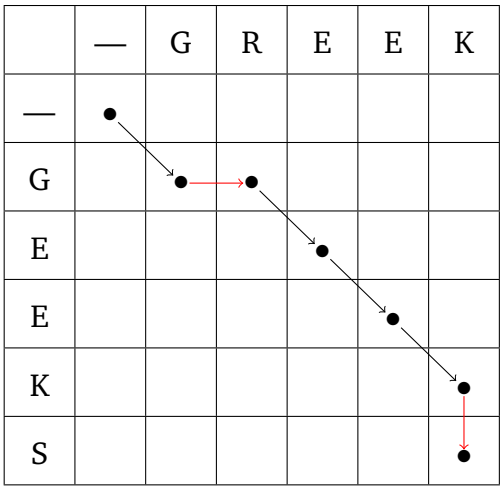
Morphemes, or other subword units such as syllables or common grapheme subsequences, may be easier for a machine learning system to process than whole words. A tokenizer may be used to split words into smaller units called tokens. Tokenization can use human-built rules [29], or the tokens may be derived automatically from a text corpus using a tool such as SentencePiece [30].

2.1.6 Word error rate

Measuring the performance of an ASR system requires comparing the sequence of words output by the system with the sequence of words in the reference transcript. Because the hypothesis length and reference length may not be the same, it is not possible to straightforwardly compare each word in the hypothesis with the corresponding word in the reference. Instead, word error rate (WER) is used to measure performance. WER is defined as

$$\text{WER} = \frac{\text{ED}(\text{reference}, \text{hypothesis})}{|\text{reference}|} \times 100, \tag{2.1}$$

where ED denotes the edit distance (a.k.a. Levenshtein distance [31]) between the reference and the hypothesis in terms of words, and $|\text{reference}|$ denotes the number of words in the reference. Edit distance is defined as the minimum number of insertions, deletions, and substitutions needed to change the reference into the hypothesis. An insertion adds a



(a) Possible path through edit table.

G	R	E	E	K	—
	↓ D				↓ I
G	—	E	E	K	S

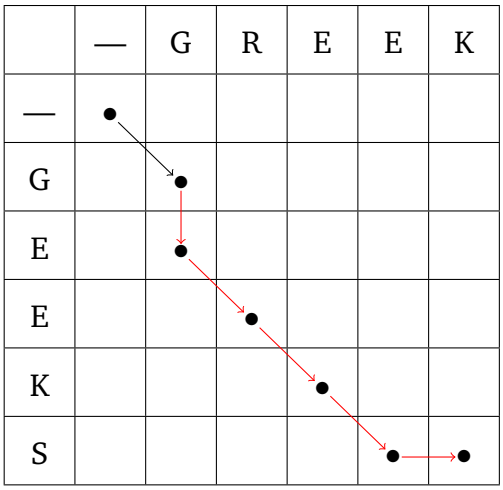
(b) Corresponding alignment.

Figure 2.5: An alignment between “GREEK” and “GEEKS” with cost 2.

word to the transcript (e.g. “It’s all Greek to me” → “It’s all too Greek to me”); a deletion removes a word (e.g. “It’s all Greek to me” → “all Greek to me”); a substitution changes a word (e.g. “It’s all Greek to me” → “It’s all geeks to me”). It is also possible to split the transcripts into characters instead of words, in which case using the edit distance will compute character error rate (CER).

It is not immediately obvious how to compute edit distance, since there are many possible ways to change one sequence into another sequence. We can use a table to visualize the space of possible transformations. Fig. 2.5 shows an edit table (using characters as the base unit) for the reference sequence “GREEK” and hypothesis sequence “GEEKS”, with a path through the table (Fig. 2.5a), along with the corresponding reference-hypothesis alignment (Fig. 2.5b). The cost of the alignment is 2 because the alignment uses two edit operations: one deletion and one insertion. Fig. 2.6 shows another alignment with a (worse) cost of 5.

It is inefficient to compute edit distance by enumerating over all possible alignments, so a dynamic programming algorithm [32] is used (Algorithm 1). The algorithm loops over each cell of the edit table computing the minimum cost of all possible alignments leading to that cell, and finally the entry in the lower-right cell is returned (e.g. Fig. 2.7a). An optimal alignment can also be recovered by noting the operation that would be used to



(a) Possible path through edit table.

G	—	R	E	E	K
	↓ I	↓ S	↓ S	↓ S	↓ D
G	E	E	K	S	—

(b) Corresponding alignment.

Figure 2.6: An alignment between “GREEK” and “GEEKS” with cost 5.

enter each cell (breaking ties using some arbitrary ordering when computing an argmin) and backtracking through the table of operations starting from the last cell (Fig. 2.7b).

2.2 Models and algorithms

2.2.1 Autoregressive models

This thesis deals mostly with sequence data. One important technique for handling sequence data is the autoregressive model, a technique that has been referred to as “a universal unsupervised learning algorithm” [33]. Autoregressive models take advantage of the chain rule of probability to model arbitrary data distributions over sequences, in the following way. Let $\mathbf{x} = \{x_1, x_2, \ldots, x_T\}$ denote a length-T data sequence of interest: for example, a sequence of words. By repeatedly applying the chain rule

$$p(A, B) = p(A|B) \cdot p(B), \tag{2.2}$$

Algorithm 1: Compute edit distance (and optimal alignment)

Input: $\mathbf{r}$ (reference sequence of length T), $\mathbf{h}$ (hypothesis sequence of length U)
Output: Edit distance, optimal alignment
Initialize cost table $\mathbf{C}$ of shape $(U+1) \times (T+1)$ with zeros
Initialize operation table $\mathbf{O}$ of shape $(U+1) \times (T+1)$ with K
(K=keep, I=insertion, D=deletion, S=substitution)
$\mathbf{C}[0,0] := 0$; $\mathbf{O}[i,0] :=$ K
for $i = 1$ *to* U **do**
  $\mathbf{C}[i,0] := i$; $\mathbf{O}[i,0] :=$ I
for $j = 1$ *to* T **do**
  $\mathbf{C}[0,j] := j$; $\mathbf{O}[0,j] :=$ D
for $i = 1$ *to* U **do**
  for $j = 1$ *to* T **do**
    if $\mathbf{r}[j-1] = \mathbf{h}[i-1]$ **then**
      $\text{cost}_\text{S} := \mathbf{C}[i-1, j-1]$; op := K
    else
      $\text{cost}_\text{S} := \mathbf{C}[i-1, j-1] + 1$; op := S
    $\text{cost}_\text{I} := \mathbf{D}[i-1, j] + 1$; $\text{cost}_\text{D} := \mathbf{D}[i, j-1] + 1$
    if $cost_S < cost_I$ *and* $cost_S < cost_D$ **then**
      $\mathbf{C}[i,j] := \text{cost}_\text{S}$
    else if $cost_D < cost_I$ **then**
      $\mathbf{C}[i,j] := \text{cost}_\text{D}$; op := D
    else
      $\mathbf{C}[i,j] := \text{cost}_\text{I}$; op := I
    $\mathbf{O}[i,j] :=$ op
Backtrack through the operation table to get alignment
$i := T$; $j := U$; $\mathbf{a} := \emptyset$
while (i,j) *!=* $(0,0)$ **do**
  Prepend $\mathbf{O}[i,j]$ to $\mathbf{a}$
  if $\mathbf{O}[i,j] = K$ *or* $\mathbf{O}[i,j] = S$ **then**
    $i := i-1$; $j := j-1$
  else if $\mathbf{O}[i,j] = I$ **then**
    $i := i-1$
  else if $\mathbf{O}[i,j] = D$ **then**
    $j := j-1$
return $\mathbf{C}[U,T]$, $\mathbf{a}$

	—	G	R	E	E	K
—	0	1	2	3	4	5
G	1	0	1	2	3	4
E	2	1	1	1	2	3
E	3	2	2	1	1	2
K	4	3	3	2	2	1
S	5	4	4	3	3	**2**

(a) Edit table filled with partial alignment costs.

	—	G	R	E	E	K
—	**K**	D	D	D	D	D
G	I	**K**	**D**	D	D	D
E	I	I	S	**K**	D	D
E	I	I	I	K	**K**	D
K	I	I	I	I	I	**K**
S	I	I	I	I	I	**I**

(b) Edit table with operations used to enter each cell and backtracking path indicated.

Figure 2.7: Results of running the dynamic programming algorithm for edit distance.

we can write the joint distribution $p(\mathbf{x}) = p(x_1, x_2, \ldots, x_T)$ as the product of the conditional distributions of each of the x_t terms, that is,

$$p(\mathbf{x}) = \prod_t p(x_t | \underbrace{x_{t-1}, x_{t-2}, \ldots, x_1}_{\text{a.k.a. "}\mathbf{x}_{<t}\text{"}}). \tag{2.3}$$

For instance, let $T = 4$. Then the autoregressive chain rule decomposition yields

$$p(\mathbf{x}) = p(x_1, x_2, x_3, x_4) \tag{2.4}$$

$$= p(x_4|x_3, x_2, x_1) \cdot p(x_3|x_2, x_1) \cdot p(x_2|x_1) \cdot p(x_1). \tag{2.5}$$

An autoregressive model uses a single $p_\theta(x_t|\mathbf{x}_{<t})$ with trainable parameters θ to implement all the conditional distributions in Equation 2.3. (It is also possible to use a distinct model to implement each conditional distribution, but such a model does not have the desirable capability of generalizing to arbitrarily long sequences.)

Given a training set of data sequences, autoregressive models can be trained using

maximum likelihood estimation [34, 35]:

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\log p_\theta(\mathbf{x})]. \tag{2.6}$$

Autoregressive models are generative models: that is, they can be used to sample new data sequences similar to those in the training set. It is easy to sample from the distribution defined by the model by repeatedly sampling the next x_t from $p_\theta(x_t|\mathbf{x}_{<t})$ (Algorithm 2) and concatenating, though more sophisicated sampling algorithms also exist [36].

Algorithm 2: Sampling from an autoregressive model

$\mathbf{x} := \emptyset$
while *not bored* **do**
  Sample $x \sim p_\theta(x|\mathbf{x})$
  Append x to $\mathbf{x}$

The "universality" of autoregressive models lies in the fact that they can be applied to *any* data modality: any data structure can be represented as a sequence of bytes, so any data structure can be modeled and generated autoregressively. Video data can be treated as sequences of pixels, audio data can be treated as sequences of microphone samples, text data can be treated as sequences of characters — and autoregressive models have been trained for all of these modalities [37, 38, 39] and many more [33, 40, 41, 42, 43, 44], with varying degrees of success.

2.2.2 n-gram models

Until the invention of recurrent neural networks [45, 46] (see Section 2.2.7), it was not clear how an autoregressive model of the form $p_\theta(x_t|\mathbf{x}_{<t})$, with an unlimited history over previous x_t terms, could actually be implemented. Instead, many models approximate Equation 2.3 by assuming that x_t can be predicted using only the previous $n-1$ terms:

$$p(\mathbf{x}) \approx \prod_t p(x_t|x_{t-1}, x_{t-2}, \ldots, x_{t-(n-1)}). \tag{2.7}$$

A model that implements $p(x_t|x_{t-1}, x_{t-2}, \ldots, x_{t-(n-1)})$ using a lookup table is known

"0-gram" model (all characters equiprobable):
XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD
QPAAMKBZAACIBZLHJQD.

1-gram model:
OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA
NAH BRL.

2-gram model:
ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TU
COOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

3-gram model:
IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF
DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

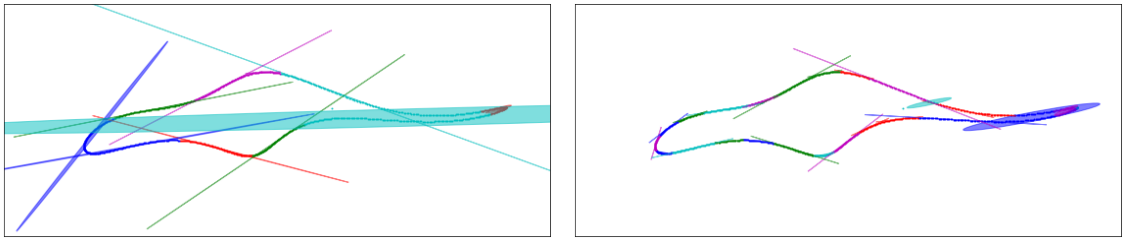
Figure 2.8: n-gram model samples for increasing n (from [3]).

as an n-gram model. Maximum likelihood estimation for an n-gram model has a simple closed-form solution: set each entry of the lookup table equal to the relative frequency of the corresponding n-gram in the training data. For large n, many entries of the lookup table will never appear in the training data, and so will be assigned 0 probability by the model. To help the model generalize to unseen n-grams, "smoothing" of various forms can be applied [47, 48]. $O((\text{size of vocabulary})^n)$ bytes of storage are required to implement an n-gram model, so only small n may practically be used.

Claude Shannon famously showed in [3] how increasingly realistic English text can be generated by fitting character-level n-gram models of increasing n to a text corpus and sampling from the model using Algorithm 2. Some of the text samples from [3] are shown in Fig. 2.8. n-gram models are of course not only useful for generation; they can be used to score a sequence by computing $p(\mathbf{x})$ using the chain rule.

2.2.3 Gaussian mixture models

Lookup tables, such as are used in n-gram models, can be used to model distributions over a finite set of discrete outcomes, but not over an infinite set of real-valued outcomes where $x \in \mathbb{R}^d$, e.g. for audio signals. One simple model for arbitrary continuous distributions is



(a) GMM with 10 components. (b) GMM with 25 components.

Figure 2.9: Fitting an elephant using a GMM. (Elephant data taken from [4]; GMM implementation provided by scikit-learn [5].)

the Gaussian mixture model (GMM), which expresses the likelihood of a data point x as

$$p_\theta(x) = \sum_z p_\theta(x, z) \tag{2.8}$$

$$= \sum_z p_\theta(x|z) \cdot p_\theta(z), \tag{2.9}$$

where

$$p_\theta(x|z) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_z|}} \exp\left(-\frac{1}{2}(x - \mu_z)^T \Sigma_z^{-1}(x - \mu_z)\right), \tag{2.10}$$

$$p_\theta(z) = \rho_z, \tag{2.11}$$

and the parameters $\theta = \{\rho_z \in \mathbb{R}, \mu_z \in \mathbb{R}^d, \Sigma_z \in \mathbb{R}^{d\times d} | z = 1, \ldots, k\}$ are constrained such that $p_\theta(x|z)$ and $p_\theta(z)$ are valid probability distributions. Since Gaussian distributions (Equation 2.10) are simple blobs, any more complex distribution may be modelled by a GMM [49] by forming the distribution using a number of these blobs, like a pointillist painting. (This does not necessarily mean a GMM can model a given distribution *efficiently*: an intractably large number of blobs may be needed [50, 51].) Fig. 2.9 shows the use of GMMs with different k (numbers of Gaussian components) to fit an elephant-shaped distribution.

Maximum likelihood estimation for a GMM has no closed-form solution, so iterative optimization methods like gradient descent need to be used. The method conventionally employed for training GMMs is the Expectation-Maximization (EM) algorithm [52]. The EM algorithm is based on the observation that if the mixture component z that generated x

were known for each x, there would be a simple closed-form solution for the parameters of each $p_\theta(x|z)$ using the Gaussian maximum likelihood estimator. But we do not know which z corresponds to which x, or even what the value of k should be: only the x values are visible, and the z values are "hidden" or "latent". The EM algorithm alternates between estimating the latent variables (the E-step) and re-estimating the parameters given the estimates of the latent variables (the M-step).

Algorithm 3: The EM algorithm (for GMMs)

Given $x_1, x_2, \cdots \sim p(x)$
while *not bored* **do**
  # E-step
  for *each* x_i *and* z **do**
    $w_{(z,i)} := \frac{p_\theta(x_i|z)\rho_z}{\sum_{z'} p_\theta(x_i|z')\rho_{z'}}$ # $= p_\theta(z|x_i)$
  # M-step
  for *each* z **do**
    $\rho_z := \sum_i w_{(z,i)}$
    $\mu_z := \frac{\sum_i w_{(z,i)} x_i}{\sum_i w_{(z,i)}}$
    $\Sigma_z := \frac{\sum_i w_{(z,i)}(x_i-\mu_z)(x_i-\mu_z)^T}{\sum_i w_{(z,i)}}$

2.2.4 Hidden Markov models

The hidden Markov model (HMM) [53] takes the idea of a latent variable introduced with GMMs and extends it to handling sequences. Instead of observing a single x, we observe a sequence $\mathbf{x} = \{x_1, x_2, \ldots, x_T\}$, which depends on an underlying correlated sequence of latent variables $\mathbf{z} = \{z_1, z_2, \ldots, z_T\}$. An HMM models a system that is in a particular state (z_t) at any given time — one out of N possible states — and produces an output (x_t) at each timestep that depends on the state. At each timestep, the system randomly jumps to a new state. The model makes two further assumptions:

1. The state at time t depends only on the state at time $t-1$.

2. The output at time t depends only on the state at time t.

Hence

$$p_\theta(\mathbf{z}) = \prod_t p_\theta(z_t|\mathbf{z}_{<t}) = \prod_t p_\theta(z_t|z_{t-1}) \tag{2.12}$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_t p_\theta(x_t|\mathbf{z}, \mathbf{x}_{<t}) = \prod_t p_\theta(x_t|z_t) \tag{2.13}$$

and

$$p_\theta(\mathbf{x}) = \sum_{\mathbf{z}} p_\theta(\mathbf{x}, \mathbf{z}) \tag{2.14}$$

$$= \sum_{\mathbf{z}} p_\theta(\mathbf{x}|\mathbf{z}) \cdot p_\theta(\mathbf{z}) \tag{2.15}$$

$$= \sum_{\mathbf{z}} \prod_t \underbrace{p_\theta(x_t|z_t)}_{\text{emission model}} \cdot \underbrace{p_\theta(z_t|z_{t-1})}_{\text{transition model}} . \tag{2.16}$$

The parameters θ are composed of the parameters of the emission model $b_s(x_t)$ (for continuous x_t, a GMM), the transition model A (a square matrix where $A_{s,s'} = p_\theta(z_t = s|z_{t-1} = s')$, the probability of jumping from state s' to state s), and a set of initial state priors π (where π_s denotes the probability of starting in state s).

Directly summing over all possible $\mathbf{z}$ to compute $p_\theta(\mathbf{x})$ in Eq. 2.16 is intractable, since there are N^T possible state sequences. Instead, we can use a dynamic programming algorithm (similar to Algorithm 1 used for computing edit distance) to compute the sum in just $O(N^2T)$ time: the Forward algorithm (Algorithm 4).

Algorithm 4: The Forward algorithm (for HMMs)

for $s = 1 \to N$ **do**
  $\alpha_{s,1} := b_s(x_1) \cdot \pi_s$
for $t = 2 \to T$ **do**
  for $s = 1 \to N$ **do**
    $\alpha_{s,t} := b_s(x_t) \cdot \sum_{s'} A_{s,s'} \cdot \alpha_{s',t-1}$
$p_\theta(\mathbf{x}) := \sum_s \alpha_{s,T}$
return $p_\theta(\mathbf{x})$

The Forward algorithm computes, in addition to $p_\theta(\mathbf{x})$, a set of "forward probabilities" α for each state and timestep, where $\alpha_{s,t} = p_\theta(x_1, x_2, \ldots, x_t, z_t = s)$. Running the algorithm in reverse (the "Backward algorithm") yields analogous "backward probabilities" β. The Forward-Backward algorithm combines $\alpha_{s,t}$ and $\beta_{s,t}$ to compute $\gamma_{s,t}$, the probability of being in state s at time t. $\gamma_{s,t}$ can be plugged into the EM algorithm (the E-step) to train the model.

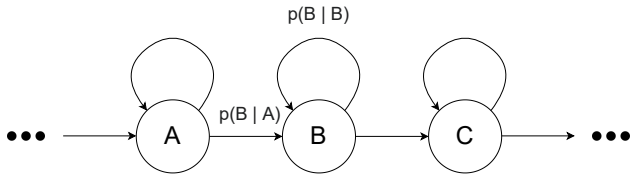


Figure 2.10: 3 states of a left-to-right HMM, with two transition probabilities indicated.

Inferring the most likely state sequence $\mathbf{z}$ from the observed sequence $\mathbf{x}$ is similar to computing $p_\theta(\mathbf{x})$: instead of summing over state sequences, we take the maximum of their probabilities. The Viterbi algorithm [54] (Algorithm 5) modifies the Forward algorithm to compute the maximum, and then, like the edit distance algorithm, backtracks through the dynamic programming table to find the argmax state sequence.

Algorithm 5: The Viterbi algorithm (for HMMs)

for $s = 1 \to N$ **do**
  $\delta_{s,1} := b_s(x_1) \cdot \pi_s$
  $\psi_{s,1} := 0$
for $t = 2 \to T$ **do**
  for $s = 1 \to N$ **do**
    $\delta_{s,t} := b_s(x_t) \cdot (\max_{s'} A_{s,s'} \cdot \delta_{s',t-1})$
    $\psi_{s,t} := \operatorname{argmax}_{s'} A_{s,s'} \cdot \delta_{s',t-1}$
$z_T^* := \operatorname{argmax}_s \delta_{s,T}$
for $t = T\text{-}1 \to 1$ **do**
  $z_t^* := \psi_{z_{t+1}^*,t+1}$
$\mathbf{z}^* = \{z_1^*, \ldots, z_T^*\}$
return $\mathbf{z}^*$

2.2.5 Acoustic models and language models

While HMMs are naturally unsupervised learners of $p(\mathbf{x})$, it is possible to shoehorn them into a supervised learning setup by forcing certain states to be associated with certain output labels. This setup has been used for applications like speech recognition and machine translation [55, 56].

In an HMM speech recognizer [57, 58], each phoneme is represented using a left-to-right HMM (Fig. 2.10). A left-to-right HMM has a chain of states in which transitions are allowed from one state only to one other state or to itself. The HMM for a word is formed by concatenating the HMM for each phoneme of the word stored in a lexicon

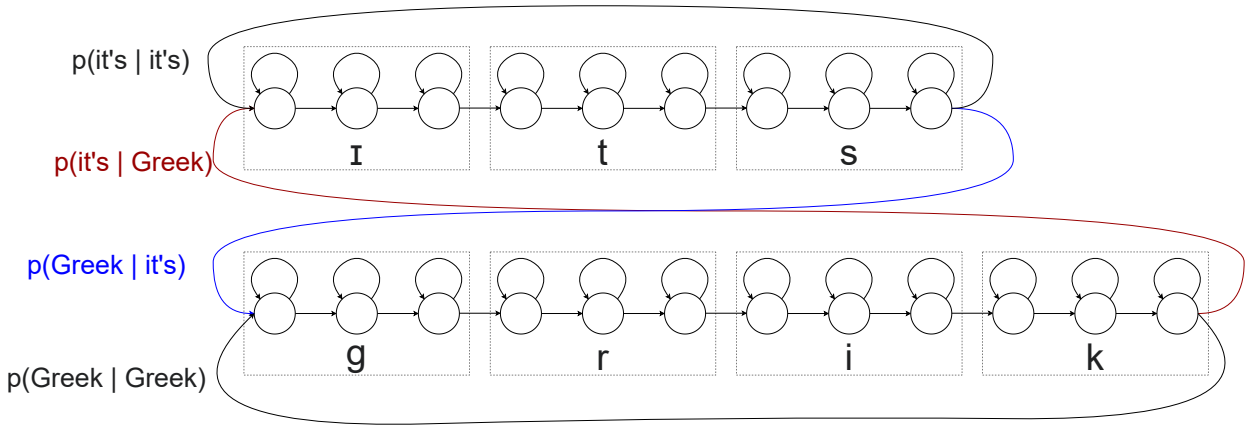


Figure 2.11: 2-word composite HMM for "it's" and "Greek" with 3 states per phoneme, with bigram word transition probabilities.

(or "pronunciation dictionary" or "pronunciation model"). A composite HMM (Fig. 2.11) can be formed by connecting the HMM for each word in the vocabulary to every other word, where the transition probability between two words is the probability according to a bigram LM trained on an external text dataset. A trigram LM can be used instead by creating a virtual HMM corresponding to each word in the context of each other word, and so forth for higher-order n-gram models. The size of the composite HMM grows exponentially with n, since a sub-HMM needs to be formed for each n-gram. Decoding[1] a speech signal works by running the Viterbi algorithm and returning the sequence of words corresponding to the sub-HMMs traversed by the best path.

Training the HMM uses a variant of the EM algorithm:

1. Form the left-to-right HMM corresponding to the transcript for each example.
2. Generate a "flat" alignment for each example, with an equal number of timesteps for each state in the HMM.
3. Estimate the GMM parameters given the alignments.
4. Realign data using the Viterbi algorithm (or Forward-Backward algorithm) with the new GMM parameters.

[1]The use of "decoding" to refer to the Viterbi algorithm and other search algorithms for sequence models is due to early researchers such as Fred Jelinek, who originally worked on digital communication systems and ported the terminology of error-correcting codes to speech recognition and machine translation.

5. (Back to 3. until bored.)

A human annotator can manually align the data instead of using the Viterbi algorithm, but it is extremely precise and time-consuming work to align phonemes with the audio signal — only a small number of small datasets, like TIMIT [59], have manual alignments. It is much more common to use the EM algorithm as described above to align the data.

HMM ASR has an aesthetically pleasing interpretation in terms of Bayes' rule: since we are trying to find the most likely transcript given the audio, we can formulate the problem in terms of the probability assigned by an acoustic model and a language model [60, 57, 61]:

$$\underset{\text{transcript}}{\operatorname{argmax}}\, p(\text{transcript}|\text{audio}) = \underset{\text{transcript}}{\operatorname{argmax}}\, \frac{p(\text{audio}|\text{transcript}) \cdot p(\text{transcript})}{p(\text{audio})} \tag{2.17}$$

$$= \underset{\text{transcript}}{\operatorname{argmax}}\, \underbrace{p(\text{audio}|\text{transcript})}_{\text{acoustic model}} \cdot \underbrace{p(\text{transcript})}_{\text{language model}}, \tag{2.18}$$

where the acoustic model is an HMM and the language model is an n-gram LM. Intuitively, while an acoustic model should assign the same score to "`I ATE FOOD`" and "`I EIGHT FOOD`" for a given audio, a language model should assign a higher score to the former sentence. The term "acoustic model" is also often used more generally for any kind of model where the input is audio: for instance, end-to-end models that directly estimate $p(\text{transcript}|\text{audio})$ are also referred to as "acoustic models" (we will sometimes use this terminology in the thesis), and even purely unsupervised models of $p(\text{audio})$ have been referred to as "acoustic models" [62].

Practical decoding diverges from the pleasant Bayes' rule interpretation in a number of ways: the LM score needs to be multiplied by a scaling factor to get good performance; the transcript corresponding to the Viterbi path may not actually have the highest probability because another transcript might have multiple paths whose probabilities sum to a higher value; a pruned version of the Viterbi algorithm must be used because the state space is too large to consider all possible states and transitions. Toolkits like Kaldi [63] efficiently implement decoding by taking advantage of the fact that HMMs, n-gram models, pronunciation dictionaries, and other optional components can be regarded as special

cases of a powerful gadget called a finite state transducer (FST) [64]. FSTs assign a score to a mapping between an input string and an output string, and can be composed with each other to form more sophisticated mappings. The acoustic model, language model, and lexicon can be welded together into a single FST, compressed, composed with an FST corresponding to the GMM scores for each input audio frame to form a search graph, and decoded using the Viterbi algorithm.

There are a number of issues with HMM speech recognition: these systems require very complicated software implementation; the hardcoded pronunciation dictionary requires some linguistic expertise to develop and does not allow for learning speakers' unanticipated pronunciations; only simple small n-gram LMs can be used without dramatically expanding the size of the search graph; and the strong probabilistic assumptions of GMMs and HMMs are not realistic for speech. For a single model that aims to solve these problems (for ASR and beyond), we now turn to neural networks.

2.2.6 Neural networks

Neural networks were first conceived in 1943 [65] and have experienced many waves of hype and disillusionment since.[2] The first true implementation of an artificial neural network was the 1957 Perceptron [67], a linear model with a hard binary decision at the output. Minsky and Papert showed in 1969 [68] that certain problems (e.g., the "XOR problem") could not be solved using a Perceptron but could be solved by a "multi-layer Perceptron", with the output of some Perceptrons fed as input to other Perceptrons. It was not known how such a multi-layer model could be efficiently trained until the backpropagation paper [69], which showed that a multi-layer model, with differentiable outputs instead of the Perceptron's hard decision output, could be trained by computing the gradient using the chain rule and running gradient descent. (Backpropagation, like many algorithms, has a long history of multiple invention [70].)

Despite a few early successes like TD-Gammon [71], the Graph Transformer Network check reading system [72], the SENNA parser [73], and CTC handwriting recognition

[2] In 2010, professors at MIT considered removing neural networks from their AI course to make room for more important topics [66].

[74], neural networks were initially not widely used because they were found to be finicky [75] and often did not perform much better than other machine learning models. Renewed widespread interest came in 2012 when AlexNet [76], a very large and deep neural network, won the challenging ImageNet [77] image classification competition by a large margin compared to more handcrafted systems.

Neural networks are suitable for implementing end-to-end learned systems because they lack many of the flaws of other machine learning algorithms. Unlike n-gram models, HMMs, and FSTs, they can model arbitrarily long-term dependencies in a sequence without an exponential increase in model size. Unlike linear models, they can fit non-linear patterns and decision regions without feature engineering. Unlike support vector machines, they can learn complex feature extractors suited to the task at hand, rather than relying on a human-designed kernel. Unlike nearest neighbors models, they can absorb large datasets without any increase in runtime complexity. Unlike decision trees, they can learn online and can be backpropagated through, enabling new applications like generative adversarial networks [78] and gradient-based meta-learning [79, 80]. Systems that could not possibly have been implemented using one of those other models, such as we will see in this thesis, can be implemented with ease using neural networks. In what follows, we describe the neural network variants and training tricks we will use in the thesis.

The basic feedforward network

Feedforward networks — also called "multi-layer perceptrons (MLP)", though unlike the original Perceptron they do not use non-differentiable hard decision outputs — map an input vector to an output vector by multiplying it with a sequence of trainable weight matrices. The input to the network must be a vector: if the data is discrete (e.g., letters), it must first be mapped to vectors using a lookup table (or "embedding").

Each matrix multiplication is followed by a nonlinear function applied elementwise to the product, so a "hidden layer" computes:

$$h_{out} = \sigma(W h_{in} + b), \tag{2.19}$$

where $h_{in} \in \mathbb{R}^{d_{in}}$, $h_{out} \in \mathbb{R}^{d_{out}}$, $W \in \mathbb{R}^{d_{out} \times d_{in}}$ (the "weights"), and $b \in \mathbb{R}^{d_{in}}$ (the "biases"). A hidden layer with d_{out} outputs is sometimes said to have d_{out} "hidden units" or "neurons". Hidden layer outputs both before and after the nonlinearity are referred to as "activations". The output layer is a simple linear transformation of the last hidden layer's activations, $W_{out}h_L + b_{out}$. Without the nonlinearity, a feedforward network would be equivalent to a linear model; including nonlinearities allows the model to fit more complex functions. The nonlinearity $\sigma(\cdot)$ used in some models is the (logistic) sigmoid function

$$y = \frac{1}{1 + \exp(-x)}. \tag{2.20}$$

Another often used sigmoidal nonlinearity is the hyperbolic tangent (tanh) function. (Both the logistic and tanh functions are "sigmoidal", but more often "sigmoid" is used to mean "logistic sigmoid".)

Sigmoid functions have been found to make gradient-based training for deeper networks more difficult because their derivative is only non-zero within a small input range, resulting in "vanishing gradients" for the weights of earlier layers [81, 82]. More recent models use the rectified linear unit (ReLU) nonlinearity [76]

$$y = \max(0, x) \tag{2.21}$$

or "leaky" ReLU

$$y = \max(\alpha x, x) \tag{2.22}$$

instead, where α is frozen at some small positive value like 0.1.

Backprop and SGD

Neural networks are trained by minimizing a loss function using stochastic gradient descent (SGD). Often the loss function is the negative log-likelihood, which for a classification problem is the negative log of the softmax [83] output for the target label. To compute the gradient of the loss using the chain rule of calculus, the backpropagation (or "backprop") algorithm is used [69]. Backprop, like the Forward and Viterbi algorithms [84],

is a dynamic programming algorithm: it avoids needless computations for the chain rule by reusing earlier computations. The algorithm works as follows: given a computational graph — a cycle-free directed graph, where each node of the graph computes a function of the outputs of the nodes connected to it — we compute the forward pass by computing the output of each node, given its inputs; then, starting from the last node, we compute the derivative of the final output with respect to the output of each node by working backwards through the graph using the chain rule. A more thorough treatment of backprop and computational graphs in general can be found in Chapter 6 of the Deep Learning textbook [49].

Once the gradient g of all parameters θ has been computed for a random minibatch of inputs and targets using backprop, the SGD update for θ is

$$\theta := \theta - \eta_b g, \tag{2.23}$$

where η_b denotes the learning rate for the bth minibatch. It often helps to apply "momentum", so that instead of using the raw gradient to update the parameters, a smoothed exponential moving average m is used:

$$m := \beta m - \eta_b g \tag{2.24}$$

$$\theta := \theta + m, \tag{2.25}$$

where $\beta =$ e.g. 0.9. Sometimes the component-wise square of the gradient g^2 is also used, in e.g. the Adagrad [85] and Adam [86] update rules. More detail on the different variants of SGD used for neural networks can be found in [87].

2.2.7 Deep neural networks

The importance of depth

While shallow neural networks with a single hidden layer are universal approximators (as are decision trees, GMMs, and many other models), deep neural networks with many layers work better. Using a shallow neural network can be likened to writing a program

without using any function calls. In a program with no function calls, every time a certain function is performed, all of its instructions must be copied out anew, resulting in a very large and unwieldy program. Using function calls allows code to be reused for a more efficient program. Likewise, a feature computed by one hidden layer in a neural network may be reused by a subsequent hidden layer to form more complex hierarchical features: if one layer finds edges in an image, a subsequent layer can find objects made out of those edges. [88] gives a more concrete example where depth helps: the d-input parity function (given d bits, compute whether there is an even or odd number of 1s) can be computed using $O(d^2)$ parameters with a 1-layer network, $O(d)$ parameters with a network with $O(\log_2 d)$ layers, and $O(1)$ parameters with an "infinitely deep" recurrent network. Networks with 10s, 100s, or even 1000s of layers routinely attain state-of-the-art results on benchmark tasks.

Other important tricks

In addition to being deep, neural networks need a few more tricks to work well.

- **Weight initialization:** The gradient is computed by multiplying a sequence of Jacobian matrices. If the initial random weights are too small, the gradient will be too small, and training will not proceed; if too large, the gradient will be too large, and training will diverge. The weight matrices can be initialized so that the distribution of magnitudes of the activations does not change from layer to layer, which helps convergence. Which initial distribution should be used depends on the nonlinearity used in each hidden layer [81, 89].

- **Normalization:** Just as normalizing inputs to mean 0 and variance 1 is common practice for many machine learning models, normalizing activations *within* a neural network can improve performance as well. Batch norm [90] uses the statistics across a minibatch to do normalization, whereas layer norm normalizes layer outputs within a single example [91]. Batch norm sometimes interacts in unexpected and unpleasant ways with other sources of randomness during training [92], so other types of normalization have grown more popular.

- **Residual connections** — so-named because of their resemblance to the additive adjustments made to minimize residual errors in algorithms like gradient boosting [93] — are shortcut connections between the input and output of a hidden layer [94, 95]. In other words, instead of computing $\sigma(Wh_{in} + b)$, a residual block computes $\sigma(Wh_{in} + b) + h_{in}$. Each layer of a residual network makes a small additive change to the previous layer's output, similar to unrolled iterative algorithms [96]. Residual connections are crucial for training very deep networks and transformer networks.

- **Dropout:** In dropout, some of the outputs of a layer are randomly set to zero during training [97]. Similarly, layer drop [98] randomly skips an entire layer in a residual network, sending h_{in} instead of $\sigma(Wh_{in} + b) + h_{in}$ to the next layer. Both techniques regularize the model, and layer drop enables using a smaller subnetwork instead of the entire network for more efficient inference.

- **Learning rate scheduling:** SGD converges to a global minimum for convex optimization problems if $\sum_{b=1}^{\infty} \eta_b = \infty$ and $\sum_{b=1}^{\infty} \eta_b^2 < \infty$, e.g. if $\eta_b = \frac{1}{b}$ [99]. For non-convex neural network losses, there is no such convergence guarantee, but similar learning rate schedules have been found to work well. Simply decreasing the learning rate once in the middle of training is a strong baseline [100]. In fact, virtually any learning rate schedule — even one set using the pixel values from a picture of Geoff Hinton's face [101] — outperforms a fixed learning rate. For transformer architectures, it seems to be important to use a "warmup" schedule, where the learning rate starts small, gradually ramps up to some maximum value, and then is scheduled as normal [102, 103].

- **Data augmentation:** Small changes to the input can produce new training examples. For audio data, this may take the form of adding noise, slightly speeding up or slowing down the recording [104], or deleting portions of the frequency spectrum (SpecAugment) [105]. For text data, when using a tokenizer, subword regularization [106] takes advantage of the fact that a given text sequence can be tokenized in multiple ways, and randomly selects different tokenizations during training.

- **Early stopping:** It is good practice to train a neural network for as long as possible and keep a checkpoint of the weights for the epoch or update with the best validation performance. The actual metric of interest (e.g., WER) should be used for checkpointing rather than the loss (e.g., negative log-likelihood), since the metric of interest sometimes does not correlate well with the loss [107].

Vivid demonstrations of the utility of these tricks were given in [108] and [109], which revisited the classic feedforward network language model of [110] and the digit recognizer of [111], respectively, and updated them to use modern techniques without changing the rest of the setup, greatly improving both systems' performance and highlighting the enormous progress in the intervening years since those systems were published.

Convolutional networks

The basic feedforward network maps a vector to another vector. If the input is instead a *sequence* of vectors, possibly of variable length, the network can be applied to each input vector in the sequence. However, doing so does not take advantage of the context in which each vector appears — for instance, to determine whether "stick" is a noun or a verb, one ought to look at the neighboring words.

A convolutional layer [112] instead takes as input a window of neighboring vectors from the input sequence, and multiplies it by a weight matrix. The same weight matrix is used for each window. The weight matrix can thus be thought of as a set of finite impulse response filters convolved with the input sequence, where each filter produces a scalar output. The dimension of each input vector is referred to as the number of "input channels", and the number of filters in the weight matrix is the number of "output channels". An 80-dimensional FBANK sequence could be processed by a convolutional layer with 80 input channels and an arbitrary number of output channels, using an arbitrarily long filter.

Convolutional neural networks (CNNs) are built up using convolutional layers, each followed by a nonlinearity, e.g. $\mathrm{ReLU}(\cdot)$. The input sequence is thereby transformed into an output sequence with the same length, and a possibly different dimension for each vector. Padding (with e.g. zeros) must be appended to the beginning and end of the input

sequence at each layer so that each vector has enough neighboring vectors for the filter to be applied.

A layer within a CNN can be used to make the input sequence shorter in two ways: stride and pooling. "Stride" refers to the number of steps a filter moves forward as the filter is scanned from the beginning of the sequence to the end. If the stride is 1, the filter moves forward by one vector each timestep, and the number of output vectors is the same as the number of input vectors; if the stride is 2, the input length is halved because the filter is only applied to every other input, and so on. "Pooling" refers to aggregating the values of each neighboring vector within the same channel: e.g., max pooling applied to the (1-channel) sequence [1, 2, 3, 2] would return 3. Pooling can be applied globally, producing a single output vector, or like a convolutional layer, by scanning the pooling operation across the input sequence.

In some cases, the input sequence may always have some fixed length or maximum length, in which case it is possible to concatenate the inputs into a single vector and just use a normal feedforward "fully-connected" network. However, it may still make sense to use a convolutional layer on fixed-length sequences because of the the layer's "inductive bias", which refers to built-in assumptions made by a model that would otherwise have to be learned from scratch. A convolutional layer has the inductive bias of "translational equivariance", which means the same pattern may appear in multiple different places in the input, and if the pattern is shifted in the input, the model should produce the same output, different only by a shift. Convolutional layers also have the inductive bias of "locality", which means neighboring vectors are more likely to be correlated. The inductive biases of convolutional networks make them especially well suited for image data, where a 2-D convolution may be used instead.

Recurrent networks

A convolutional network can only find relationships between inputs that are close enough to appear together within its "receptive field". To enable finding relationships between inputs that may be arbitrarily far apart in the input sequence, we need to equip the network with memory. A recurrent neural network (RNN) [46] uses feedback connections to store

information between timesteps, thereby allowing information to be kept in memory and combined with new information for an arbitrary amount of time.

The basic recurrent network computes

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b), \tag{2.26}$$

where x_t is the tth vector in the input sequence, h_t is the "(hidden) state" of the RNN, and h_0 is initialized to some (trainable) vector.

RNNs are theoretically very exciting because they can implement general-purpose digital computers [113]. Whether this is learnable or feasible using reasonably sized RNNs and SGD is a different question [114]. In practice the basic RNN does not work well, due to the vanishing gradient problem [82]. The solution researchers have found is to use an additive update to the state, as in residual networks, and a "gating mechanism" to (differentiably) allow or disallow information to be added to the state. The Long Short-Term Memory (LSTM) architecture [115] uses the following gated state update:

$$i_t = \text{sigmoid}(W_{ix} x_t + W_{ih} h_{t-1} + b_i) \tag{2.27}$$

$$f_t = \text{sigmoid}(W_{fx} x_t + W_{fh} h_{t-1} + b_f) \tag{2.28}$$

$$g_t = \tanh(W_{gx} x_t + W_{gh} h_{t-1} + b_g) \tag{2.29}$$

$$o_t = \text{sigmoid}(W_{ox} x_t + W_{oh} h_{t-1} + b_o) \tag{2.30}$$

$$c_t = f_t \cdot c_t + i_t \cdot g_t \tag{2.31}$$

$$h_t = o_t \cdot \tanh(c_t) \tag{2.32}$$

where W_*, b_* are the weights and biases for the various gates (input gate i, forget gate f, cell gate g, and output gate o). The putative function of each gate is not that important for our purposes; more detail can be found in e.g. [116]. Because the sigmoid activation is between 0 and 1, it can either completely allow another activation to pass to another part of the network (=1), or completely block it (=0), or somewhere in between. The differentiability of the sigmoid function allows backpropagating through the gating function. The "cell" state c_t receives an additive update, and the hidden state h_t is typically used as

the output for downstream modules.

The simpler Gated Recurrent Unit (GRU) [116], which we will also use, has the following update:

$$r_t = \text{sigmoid}(W_{rx}x_t + W_{rh}h_{t-1} + b_r) \tag{2.33}$$

$$z_t = \text{sigmoid}(W_{zx}x_t + W_{zh}h_{t-1} + b_z) \tag{2.34}$$

$$n_t = \tanh(W_{nx}x_t + b_{nx} + r_t \cdot (W_{nh}h_{t-1} + b_{nh})) \tag{2.35}$$

$$h_t = (1 - z_t) \cdot n_t + z_t \cdot h_{t-1}, \tag{2.36}$$

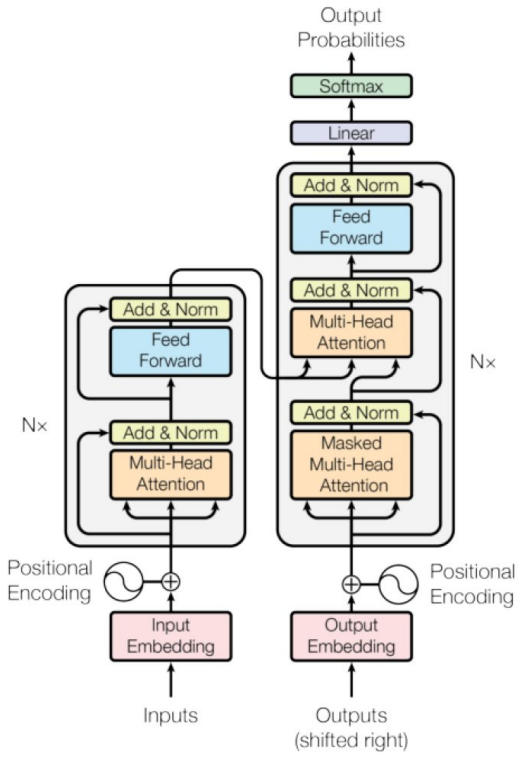
where again W_*, b_* are the weights and biases.

Like convolutional layers, recurrent layers can be stacked up to create deep networks and trained with backpropagation, with the hidden state sequence from one RNN layer forming the input sequence for another RNN layer [117]. It also may help to use bidirectional RNNs [118], where a forward RNN and backward RNN process the input sequence in different directions and concatenate their outputs. Whereas a unidirectional RNN can process a streaming sequence of inputs, a bidirectional RNN cannot: it must wait until the sequence has ended before the backward RNN can begin processing, making it unsuitable for real-time applications.

Transformers

Convolutional networks are fast to train because they can be implemented using parallel matrix multiplications, but they have no long-term memory unless their filters are made very large. Recurrent networks have "infinite" memory, but are slow to train because each timestep cannot be computed in parallel. Transformers [119] (or "transformer networks" or "self-attention networks") cleverly attempt to get the best of both worlds using an "attention" operation.

There are many variants of attention for neural networks [121]. All of them in some way allow one part of the network to "focus on", or "attend to", some subset of signals produced by another part of the network. Transformers use key-value attention, which works as follows. The sequence of input vectors is treated as a single matrix X, which



(a) The original encoder-decoder transformer model (from [119]).

(b) A Tatooine moisture vaporator (from [120]).

Figure 2.12: Two suspiciously similar-looking objects.

is multiplied by three weight matrices, W_q, W_k, and W_v, to obtain the queries Q, keys K, and values V, respectively. The queries are multiplied by the keys to get a score for each timestep, which for each query indicates how relevant each timestep is to that query. The scores are normalized using a softmax operation to form attention weights that add up to 1. (If one input score is much larger than any other, its attention weight will be 1, and all others' will be 0, thereby focusing attention only on that timestep.) It is easy to mask out certain timesteps, so that they do not contribute at all to the output, by setting their attention weights to 0 — this is used when training autoregressive models implemented using transformers to prevent the model from looking at future timesteps while still parallelizing computation over time.

The values are weighted by the attention weights and added up to get a single output vector for each timestep:

$$\text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.37}$$

where d_k is the dimension of the keys (and queries), and the $\sqrt{d_k}$ term helps the magnitude of the activations not grow too large. This describes a single attention head: multi-headed attention uses multiple heads, each with its own set of query, key, and value weight matrices, to compute a different attention pattern, and the output of each head is concatenated. Because the queries are computed from the same input sequence as the keys and values, this operation is called "self-attention"; the queries can also come from somewhere else (i.e., another neural network), in which case the operation is just called "attention" (or "cross-attention").

A transformer layer uses multi-headed self-attention, followed by feedforward layers (i.e., convolutional layers with a filter length of 1), along with layer norm and residual connections. Without any further modifications, a transformer layer is permutation equivariant, meaning that the order of the inputs in a sequence does not matter — which we usually do not want. ("The lion eats the man" does not mean the same thing as "The man eats the lion".) To distinguish between timesteps, transformers add position embeddings to each timestep. Absolute position embeddings simply use a lookup table with one entry per

time index, which does not allow for arbitrarily long input sequences. Sinusoidal position embeddings (computed using the sin and cos of the current timestep) in theory generalize to longer sequences than were observed in training. In practice sinusoidal embeddings do not generalize well, a problem which ALiBi embeddings [122] attempt to solve by adding an increasing bias over timesteps. Other commonly used position embeddings are relative [123] and CAPE [124].

Transformers are more general than convolutional networks: while multi-headed self-attention can implement a convolution [125], a convolutional layer cannot implement a global operation as a self-attention layer can. The generality of transformers comes at a cost: whereas convolutional networks come with built-in translational equivariance and locality, transformers must learn these notions from scratch if they are needed. However, datasets in some domains have grown so large that the inductive biases have actually begun to *hurt* performance, with the Vision Transformer [126] ultimately outperforming CNNs at scale for image classification. The downside of transformers is that the self-attention operation has $O(T^2)$ complexity. Processing very long sequences requires modified versions of the architecture, using e.g. downsampling and time-restricted self-attention [127].

The transformer architecture has enabled large performance gains in virtually every application domain for which large datasets exist [128, 6, 126]. However, transformers won't be the last word in neural network architecture. If the goal of AI is general-purpose agents that can handle infinitely long incoming streams of data, some kind of memory will be necessary for their architecture — consider a simple task like remembering the number of times an event has occurred in the past. The self-attention operation can't be extended into indefinitely into the past without eventually running out of memory and computational resources on any realizable physical computer. The block-recurrent transformer [129] is a recent good example of an attempt in the direction of stateful neural architectures that keep the benefits of transformers.

An excellent resource for understanding the transformer architecture is [130], an annotated version of the transformer paper, with PyTorch code implementing each piece of the architecture alongside the original text descriptions.

2.2.8 Neural networks for speech recognition

Now that we have introduced the various neural network architectures used in the thesis, we can discuss how to use these architectures to implement ASR and NLU.

Neural networks have been used for speech recognition for decades [131],[3] but only recently came to dominate the field. The first forays of neural networks into ASR were so-called "hybrid" neural network hidden Markov model systems [132, 131], in which almost all of the GMM-HMM machinery was kept intact, except for the emission model. Using the alignments from a trained GMM system, a neural network in a hybrid system is trained to predict the HMM state of the alignment at a given timestep, given a window of FBANK or MFCC audio frames centered at that timestep. At test time, the softmax output of the neural network is fed into the HMM decoder to transcribe the audio. Because the softmax computes $p(\text{HMM state}|x_t)$, but the Viterbi algorithm expects $p(x_t|\text{HMM state})$ from the emission model, the softmax posteriors are first divided by the "state priors" $p(\text{HMM state})$, which can be computed by counting the number of times each HMM state occurs in all the alignments. (To correctly compute $p(x_t|\text{HMM state})$ using Bayes' rule, we would also need to multiply by $p(x_t)$, but the neural network cannot compute this value, and the rest of the computation is invariant to it anyways.) Like the GMM before it, the trained neural network can be used to realign the data and compute new state priors, so that a new neural network can be trained, and so forth, iteratively.

Over the years, more sophisticated networks were used as emission models, such as 2-D convolutional networks [133], time-delay neural networks (TDNNs) [134], and bidirectional LSTMs [135]. What we would really like, though, is a neural network that can be trained to do the task of interest *directly* without any additional HMM machinery. The problem is that the networks we have described cannot produce variable-length outputs: they can only produce an output sequence that is the same length as the input sequence, or shorter by some fixed factor (e.g., divided by 2 if using a stride of 2). But in ASR and other problems, the correct output may be shorter or longer than the input by some arbitrary factor that cannot be anticipated. We will now see how to deal with this issue.

[3] Bourlard and Morgan in [131] write (in 1994): "Since we use large networks (with hundred of thousands of parameters), overtraining was a real problem." How quaint.

2.2.9 Sequence-to-sequence models

Speech recognition can be regarded as one of many AI problems in which the goal is to transform one sequence (the speech signal) into another sequence (the transcript), where the input and output sequence lengths are not known in advance. Such problems can be solved using the sequence-to-sequence model, an ingenious generalization of the autoregressive model (Sec. 2.2.1) that enables mapping arbitrarily long input sequences to arbitrarily long output sequences using neural networks. Given an input sequence $\mathbf{x}$ of length T and an output sequence $\mathbf{y}$ of length U, sequence-to-sequence (or "seq2seq", or "encoder-decoder") models compute the probability of $\mathbf{y}$ given $\mathbf{x}$ as

$$p_\theta(\mathbf{y}|\mathbf{x}) = \prod_u p_\theta(y_u|\mathbf{y}_{<u}, \mathbf{x}). \tag{2.38}$$

Sequence-to-sequence models were invented independently and simultaneously by multiple groups [136, 137, 138]. These first variants of the model function by encoding the input sequence into a single vector of fixed length. An encoder RNN consumes the input sequence, vector by vector, and the final state vector of the encoder RNN after all inputs are consumed is the encoding (Fig. 2.13). A decoder RNN then predicts each output vector given all the previous output vectors and the encoding. The decoder is another RNN: the encoder vector can be input to the decoder RNN at each timestep, in addition to the previous output, or it can be used as the initial state of the decoder RNN. The first input to the decoder is a special "`SOS`" (start-of-sequence) token, and an "`EOS`" (end-of-sequence) token is appended to $\mathbf{y}$ during training.

The authors of [139] found that sequence-to-sequence models with a fixed-length vector encoding did not generalize well to long sequences, especially sequences longer than those in the training set. The fixed-length encoding forms a bottleneck through which it may be difficult to send all the information about the input necessary for the decoder to predict the output. The solution proposed in [140] uses attention to query all the encoder outputs based on the decoder state, greatly improving performance for longer sequences. (A similar "sliding window" attention had been proposed earlier in [141].) Gehring *et al.* showed afterwards in [142] that RNNs are not necessary for sequence-to-sequence

learning: CNNs can be used in both encoders and decoders in conjunction with the attention mechanism instead, speeding up training by parallelizing computation over encoder and decoder timesteps. The encoder-decoder transformer model originally described in [119] improved the performance of sequence-to-sequence learning even further by using self-attention in the encoder and decoder instead of RNNs or CNNs.

Maximum likelihood training of sequence-to-sequence models is sometimes referred to as "teacher forcing" [143] because the correct previous outputs are fed to the decoder, as opposed to the model's own predictions. Another training strategy, scheduled sampling [144], attempts to reduce the discrepancy between the way the model works during inference and the way the model is trained by occasionally feeding a sample from $p_\theta(y_{u-1}|\mathbf{y}_{<u-1}, \mathbf{x})$ into the decoder instead of y_{u-1} during training. In a similar vein, methods like reward-augmented maximum likelihood (RAML) [145] and minimum word error rate (MWER) training [146] attempt to reduce the discrepancy between maximum likelihood training (which, for discrete outputs, implicitly considers all output sequences other than the truth to be equally wrong) and the way the model is actually evaluated by instead minimizing a loss based on the metric ultimately used for evaluating the model (WER, BLEU score, etc.). Similar techniques are employed in text generation models for learning from human feedback [147].

2.2.10 Beam search

Inference with sequence-to-sequence models requires finding

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} \log p_\theta(\mathbf{y}|\mathbf{x}). \tag{2.39}$$

Just as there are too many possible state sequences to perform a brute force exhaustive search for the most likely state sequence in an HMM, it is too expensive to solve Equation 2.39 directly. However, unlike HMMs, sequence-to-sequence models do not make any probabilistic assumptions that would allow for an efficient exact search algorithm like the Viterbi algorithm. Instead, an approximate search needs to be used.

A reasonable first approximation to Equation 2.39 is greedy decoding (Fig. 2.13).

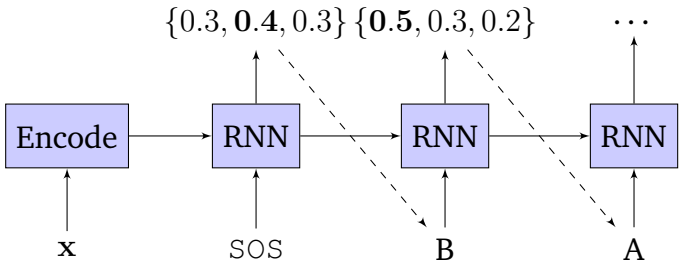


Figure 2.13: Greedy decoding for encoder-decoder model.

In greedy decoding, the argmax of the decoder's output distribution is selected at each decoding timestep and appended to $\mathbf{y}^*$, until the decoder predicts EOS.

It is easy to show that greedy decoding does not always find the output sequence with the highest score. Suppose that in the first decoding timestep $p_\theta(a|\mathbf{x}) = 0.4$ and $p_\theta(b|\mathbf{x}) = 0.6$. Greedy decoding would select "b". But now suppose that in the next decoding timestep $p_\theta(aa|\mathbf{x}) = 0.4$, $p_\theta(ab|\mathbf{x}) = 0.0$, $p_\theta(ba|\mathbf{x}) = 0.35$, and $p_\theta(bb|\mathbf{x}) = 0.25$. Greedy decoding, having already selected "b", would then select "a" to form the output "ba" — but a higher-scoring sequence is "aa", since 0.4 is greater than 0.35.

In beam search [58, 148], the decoder instead maintains a set of B hypotheses and their probabilities, with greedy decoding as the special case where $B = 1$. (The list is called the "beam", though sometimes the hypotheses themselves are called "beams".) At each timestep, the decoder extends each hypothesis with every possible output label and computes the probability of each new extended hypothesis (= the probability of the old hypothesis multiplied by the softmax probability of the new label). There are now BL hypotheses, where L is the number of labels. If we do not prune the beam, we will quickly run out of memory as the size of the beam grows exponentially with each step. To prune the beam, we keep only the top B highest scoring hypotheses at each step. This continues until every hypothesis ends in EOS (Algorithm 6). Fig. 2.14 shows the hypotheses returned by beam search for an attention-based RNN sequence-to-sequence model trained to insert missing vowels into disemvoweled text.

A few additional tricks can improve beam search. First, decoding will sometimes loop forever, never producing EOS (especially at the beginning of training, when the model does not yet have a good estimate of $p(\text{EOS}|\mathbf{y}_{<u}, \mathbf{x})$), so some maximum number of decoding

Algorithm 6: Beam search (for sequence-to-sequence models)

```
beam_prev := {∅}
while not (all hypotheses in beam_prev end in EOS or maximum number of decoding
 steps reached) do
    beam_next := {}
    for h in beam_prev do
        if h ends in EOS then
            Add h to beam_next
        else
            for l in labels do
                h_extended := concat(h, l)
                score(h_extended) := log p_θ(h_extended|x)
                Add h_extended to beam_next
    beam_prev := top B hypotheses in beam_next
return beam_prev
```

```
input: t's ll Grk t m.
truth: It's all Greek to me.
guess: it's all Gurk to me.
beam:
       it's all Gurk to me.  (score = -3.27)
       t's all Gurk to me.   (score = -3.32)
       it's all Gark to me.  (score = -4.23)
       t's all Gark to me.   (score = -4.26)
       t's all Grike to me.  (score = -4.49)
       it's all Gurk at me.  (score = -4.49)
       it's all Grike to me. (score = -4.50)
       t's all Gurk at me.   (score = -4.52)
```

Figure 2.14: Output of a beam search of width 8.

timesteps can be set to ensure that decoding terminates. Second, additional terms can be added to Equation 2.39 [149] — for example, if there is an additional set of $\mathbf{y}$-only data, we can train a model for $p(\mathbf{y})$ (in ASR, this would correspond to a language model), and instead search for

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}} \log p_\theta(\mathbf{y}|\mathbf{x}) + \alpha \log p_{\theta_{\text{other}}}(\mathbf{y}), \tag{2.40}$$

where α can be set to minimize error using a validation set [150]. Finally, given the set of hypotheses returned by beam search, it is possible to rescore those hypotheses using a second model that computes either $p(\mathbf{y})$ or $p(\mathbf{y}|\mathbf{x})$, possibly one for which beam search is slow but computing $p(\cdot)$ is fast (e.g., a large feedforward decoder) [28].

2.2.11 Connectionist Temporal Classification

Encoder-decoder models are theoretically flexible enough to handle any sequence-to-sequence problem, including ASR [151, 152, 153], but there are sometimes reasons to use a model with more specific domain assumptions. One such reason is that the attention operation does not allow for streaming inference, where outputs are predicted in real-time as inputs arrive; the encoder must have access to the entire input sequence before the decoder can attend to it. Another reason is that attention does not take advantage of the fact that the alignment between input and output sequences is *monotonic* for ASR, meaning that if word A comes after word B in the transcript, word A must come after word B in the audio signal — attention-based encoder-decoder models are more difficult to train for ASR as a result [154].

Connectionist Temporal Classification (CTC) [155, 156] is a sequence model that takes advantage of monotonic alignments and can be used for streaming inference. Like encoder-decoder models, CTC does not require a label for each timestep: the model itself learns the alignment. CTC models can be thought of as "encoder-only" sequence-to-sequence models, or encoder-decoder models where the decoder is a simple linear output layer instead of an autoregressive model. At each timestep, a CTC model uses a neural network encoder to assign a probability to each of the labels and a "blank" symbol, which corresponds to "no output". Fig. 2.15 shows a subset of the softmax outputs of a trained character-level CTC model (M-CTC-T; see Chapter 3) for an example utterance.

In CTC, an alignment $\mathbf{a}$ is defined as a length T sequence of labels and blanks. The per-timestep softmax probabilities define the probability of $\mathbf{a}$ as

$$p_\theta(\mathbf{a}|\mathbf{x}) = \prod_t p_\theta(a_t|\mathbf{x}), \tag{2.41}$$

where $p_\theta(a_t|\mathbf{x})$ represents the softmax output for symbol a_t at time t.

The probability of an output sequence $\mathbf{y}$ is defined as the sum of the probabilities of all

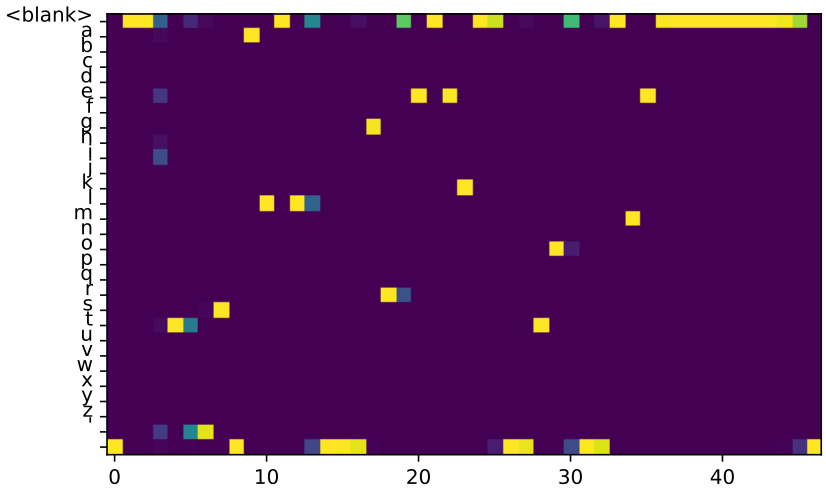


Figure 2.15: CTC model softmax output representation of the thesis author saying the phrase "it's all Greek to me".

possible alignments between $\mathbf{x}$ and $\mathbf{y}$:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{a}\in\mathcal{B}^{-1}(\mathbf{y})} p_\theta(\mathbf{a}|\mathbf{x}), \tag{2.42}$$

where $\mathcal{B}^{-1}(\mathbf{y})$ represents the set of valid alignments. The function $\mathcal{B}(\cdot)$ "collapses" an alignment to a label sequence by removing repetitions and then removing blanks: e.g.,

$$\mathcal{B}(\_\texttt{GGR\_EE\_E\_\_\_KK}) = \mathcal{B}(\_\texttt{GR\_E\_E\_K}) = \texttt{GREEK}. \tag{2.43}$$

The model is trained by minimizing the negative log-likelihood. Computing $-\log p_\theta(\mathbf{y}|\mathbf{x})$ by computing Eq. 2.42 directly is intractable; the CTC Forward algorithm, like the Forward algorithm for HMMs (Sec. 2.2.4), computes the sum with dynamic programming, using the recursion

$$\alpha_{s,t} = \begin{cases} (\alpha_{s,t-1} + \alpha_{s-1,t-1}) \cdot p_\theta(a_t = l_s|\mathbf{x}) & \text{if } l_s = l_{s-2} \text{ or } l_s = \text{blank} \\ (\alpha_{s,t-1} + \alpha_{s-1,t-1} + \alpha_{s-2,t-1}) \cdot p_\theta(a_t = l_s|\mathbf{x}) & \text{otherwise} \end{cases} \tag{2.44}$$

for $t = 1 \to T$ and $s = 1 \to 2U + 1$, where $\mathbf{l} = \{\text{blank}, y_1, \text{blank}, y_2, \ldots, \text{blank}, y_U, \text{blank}\}$. At the end of the recursion, $p_\theta(\mathbf{y}|\mathbf{x})$ is equal to $\alpha_{2U,T} + \alpha_{2U+1,T}$.

Because the output probabilities produced by a CTC network tend to be very "peaky" [157], an extremely simple greedy algorithm can be used to decode: take the argmax of $p_\theta(a_t|\mathbf{x})$ for each timestep and collapse using $\mathcal{B}(\cdot)$. (The reader is invited to try applying this algorithm to Fig. 2.15.) Greedy decoding often works well, but it is not guaranteed to find the most likely label sequence. Like encoder-decoder models, CTC models can also be decoded using beam search [28, 158]. Beam search is more complicated for CTC because multiple alignments per hypothesis need to be considered. The algorithm works by computing the per-timestep softmax outputs $\mathbf{s}$ using the neural network and updating a list of B hypotheses at each timestep, where two probabilities are maintained for each hypothesis: p_b (the probability of all alignments for that hypothesis ending in a blank) and p_{nb} (the probability of all alignments ending in a label). A simplified version of the CTC beam search from [159] is given by Algorithm 7.

Algorithm 7: Beam search (for CTC)

```
beam_prev := {∅}
p_b(∅) := 1
p_nb(∅) := 0
for t = 1 → T do
    beam_next := {}
    for h in beam_prev do
        for l in labels ∪ blank do
            if l = blank then
                p_b(h) := s_t[blank] · (p_b(h) + p_nb(h))
                Add h to beam_next
            else
                h_extended := concat(h, l)
                if l = last label of h then
                    p_nb(h_extended) := s_t[l] · p_b(h)
                    p_nb(h) := s_t[l] · p_nb(h)
                else
                    p_nb(h_extended) := s_t[l] · (p_b(h) + p_nb(h))
                if h_extended not in beam_prev then
                    p_b(h_extended) := s_t[blank] · (p_b(h_extended) + p_nb(h_extended))
                    p_nb(h_extended) := s_t[l] · p_nb(h_extended)
                Add h_extended to beam_next
    beam_prev := top B hypotheses in beam_next, sorted by p_b + p_nb
return beam_prev
```

Both greedy decoding and beam search decoding for character-level CTC models are

"open-vocabulary", which means the system can produce arbitrary words that are not necessarily found in a lexicon. It is also possible to constrain the decoder to only output in-vocabulary words by proposing an extension only if the resulting output sequence is a prefix of some word in the lexicon — e.g., if the lexicon contains only the words "IT'S", "ALL", and "GREEK", the hypothesis "IT'S ALL GREE" can only be extended by "K", and not by "N" or any other character. Checking whether an extension is a prefix can be implemented efficiently by storing the lexicon in a trie and only proposing extensions that are children of the trie node corresponding to the hypothesis [160]. A language model can be integrated into decoding [159] by multiplying the probability of the hypothesis by $p^{\text{LM}}(\text{new word}|\text{previous words})$ whenever a new word is formed by extending the hypothesis with a space label.

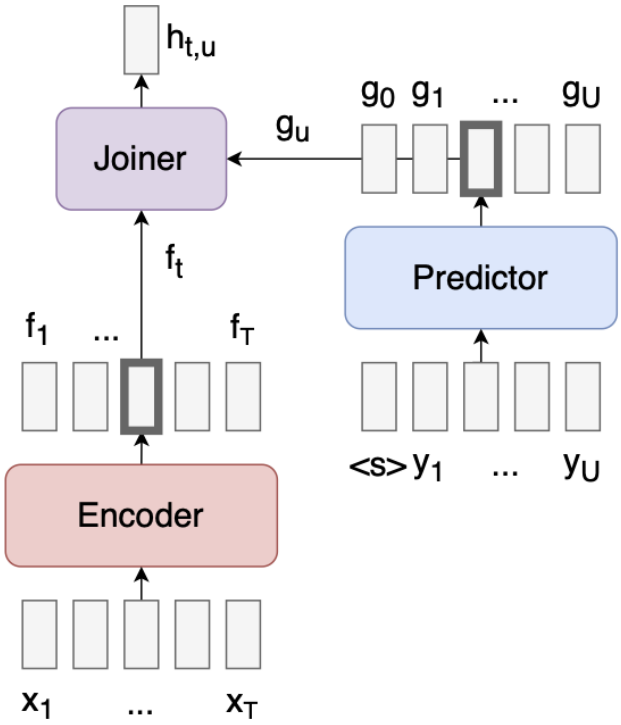
There are many similarities between HMMs and CTC [161]. HMM systems with a neural network emission model can in principle be trained in an end-to-end fashion similar to CTC by backpropagating the negative log-likelihood through the HMM Forward algorithm (Algorithm 4). Some early work demonstrating the use of this idea to improve a phoneme recognition system for TIMIT was described in [162, 163]. However, it seems to be very difficult to train neural network-HMM hybrids from scratch in this way. The CTC blank symbol has been found to be essential for training without alignments: training from a random initialization without a blank symbol diverges [164, 165, 166]. Like HMMs, CTC models can also be composed with FSTs for decoding [167, 168, 169], or even during training, backpropagating through the decoding graph into the neural network [72, 170, 171, 172].

2.2.12 Transducers

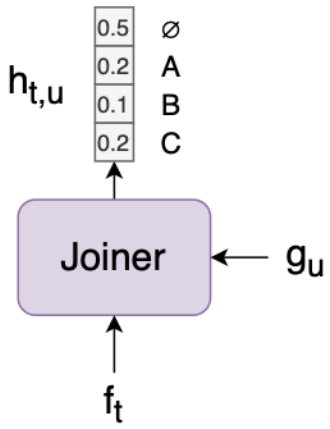
While CTC models are suitable for problems with a monotonic input-output alignment, they have two theoretical flaws compared to encoder-decoder models:

1. The output sequence length must be less than or equal to the input sequence length, since only one label or blank can be decoded per input timestep.

2. Each output a_t is conditionally independent of each other given the input $\mathbf{x}$ (i.e.,



(a) Architecture with encoder, predictor, and joiner.



(b) Joiner output for a 3-label alphabet.

Figure 2.16: The Transducer model.

$p_\theta(a_t|\mathbf{a}_{<t}, \mathbf{x}) = p_\theta(a_t|\mathbf{x})$). The famous example of this being an issue was given in [152]: if the model is decoding a recording in which the speaker says "triple A", both "`TRIPLE A`" and "`AAA`" are correct transcriptions, but if the first decoded output is "`T`", the next output must be "`R`", which requires remembering which outputs have previously been predicted.

The Transducer[4] [175, 176] (or "RNN-Transducer", or "RNN-T", though the model need not be implemented using RNNs) elegantly solves both problems, while retaining some of CTC's advantages over encoder-decoder models. It solves 1) by changing the interpretation of a label from "output this symbol and transition to the next timestep" to simply "output this symbol"; and 2) by adding a predictor network and joiner network to the encoder network.

[4]We prefer to capitalize it to disambiguate from "(finite state) transducers", though the naming convention is still in flux. "Neural transducer" is also sometimes used [173], though there is unfortunately already a collision with [174].

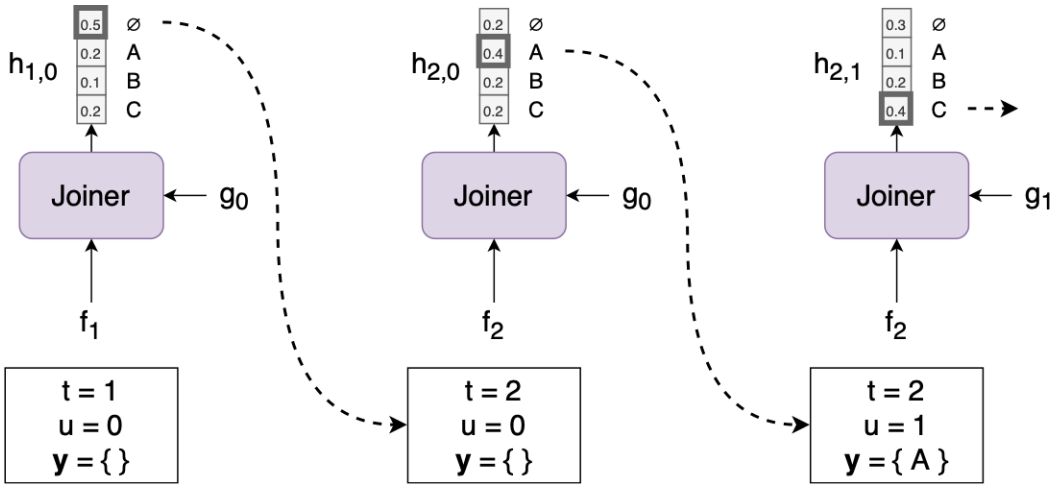
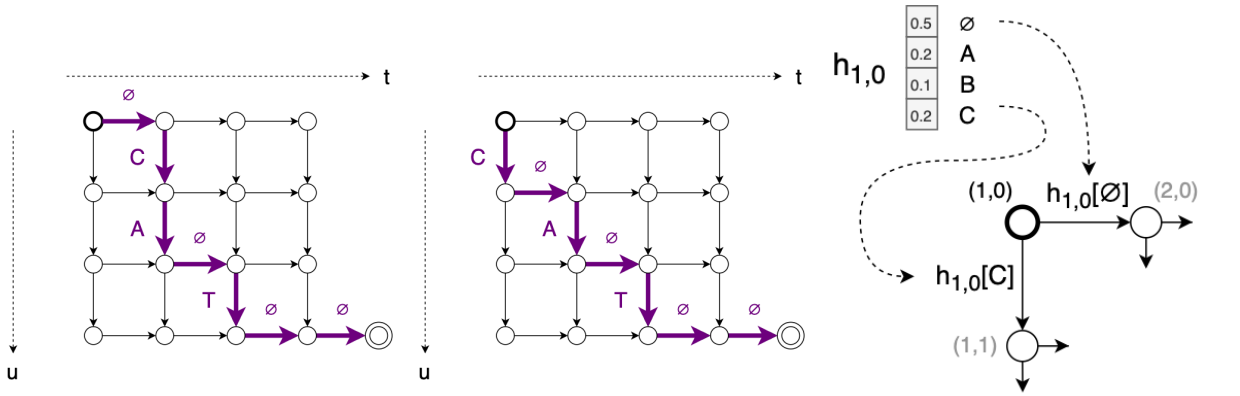


Figure 2.17: Illustration of Transducer decoding.

Fig. 2.16a shows a diagram of the Transducer. The predictor is autoregressive, but does not take $\mathbf{x}$ as input (unlike an encoder-decoder model), so it is easy to pre-train it using $\mathbf{y}$-only examples [177, 178, 179]. The joiner (not present in the original formulation but added in [117]) is a simple feedforward network that combines the encoder vector f_t and predictor vector g_u and outputs a softmax $h_{t,u}$ over all the labels, as well as a "null" output $\varnothing$ (Fig. 2.16b). The null $\varnothing$ differs from the CTC blank in that *only* $\varnothing$ indicates "move to the next timestep", whereas in CTC every other label also has that interpretation. Greedy decoding for the Transducer works by computing f_t using the encoder, computing g_u using the predictor, computing $h_{t,u}$ using f_t and g_u, taking the argmax of $h_{t,u}$, and appending the argmax to the output (if it is a label) or moving to the next timestep (if it is a blank) (Fig. 2.17).

Similar to CTC, Transducer models assign a probability to any alignment, and the full negative log-likelihood is computed by marginalizing over all possible alignments. The probability of an alignment is the product of the weights of the corresponding edges of the alignment graph (Figs. 2.18a and 2.18b), where the edge weights are computed using the corresponding joiner output (Fig. 2.18c). The Transducer Forward recursion computes, for each node (t, u) of the graph, the probability of all possible alignments ending at that node:

$$\alpha_{t,u} = \alpha_{t-1,u} \cdot h_{t-1,u}[\varnothing] + \alpha_{t,u-1} \cdot h_{t,u-1}[y_u], \tag{2.45}$$



(a) $\mathbf{a} = \{\varnothing, C, A, \varnothing, T, \varnothing, \varnothing\}$ (b) $\mathbf{a} = \{C, \varnothing, A, \varnothing, T, \varnothing, \varnothing\}$

(c) Mapping from joiner outputs to edge weights.

Figure 2.18: Transducer alignments illustrated using the alignment graph.

and then $p_\theta(\mathbf{y}|\mathbf{x})$ can be computed using the last node of the graph as:

$$p_\theta(\mathbf{y}|\mathbf{x}) = \alpha_{T,U} \cdot h_{T,U}[\varnothing]. \tag{2.46}$$

Computing $\alpha_{t,u}$ for all nodes may consume a very large amount of memory. Suppose that $T = 1000$, $U = 100$, $L = 1000$ labels, and the batch size $B = 32$ (realistic numbers for ASR). Then just to store $h_{t,u}$ for all (t, u) pairs to run the Forward(-Backward) algorithm, we need a tensor of size $B \times T \times U \times L = 3,200,000,000$, or 12.8 GB using single-precision floats, not including the memory for the intermediate joiner activations for each (t, u). Simplifying the training of Transducer models is an active area of research: some examples include using forced alignment via the Viterbi algorithm with a second model to reduce the size of the alignment graph [180, 181, 182], using simple addition instead of a feedforward network so that the joiner operation becomes separable [183], and broadcasting the encoder and predictor outputs for each example in the batch separately to avoid wasted padding [184]. Transducers are only memory-intensive during training; during decoding, only a small amount of memory is required to store previous outputs.

The flaws of CTC that Transducers were designed to correct are often not actually a dealbreaker in practice. The output sequence length for ASR is almost always smaller than the input sequence length, unless a dramatic amount of stride/pooling is used in

the encoder, so the ability to produce a longer output sequence is not terribly important; and while CTC models *explicitly* consider the outputs to be conditionally independent, they *implicitly* learn a language model over the outputs [6], especially when using deep encoder networks with high parameter counts [185]. (In applications outside of ASR, the Transducer's assumptions can in fact be important, e.g. for speech synthesis [186].) Empirically, however, Transducers often outperform CTC models and encoder-decoder models: they have attained state-of-the-art performance on the challenging LibriSpeech benchmark [187] and are at present the *de facto* neural architecture used for streaming ASR in devices such as Google's Pixel phone [188].

2.2.13 Natural language understanding: Intents and slots

The natural language understanding (NLU) stage of the conventional voice control pipeline infers the meaning of the transcript. But what is the meaning of "meaning"? As Collobert *et al.* put it in [189]:

> *Will a computer program ever be able to convert a piece of English text into a data structure that unambiguously and completely describes the meaning of the natural language text? Among numerous problems, no consensus has emerged about the form of such a data structure.*

Instead, more specific NLU tasks are defined, and the designers of NLU systems derive the appropriate semantic structures therefrom. The GLUE benchmark [190] provides a useful sample of various NLU tasks, including:

- linguistic acceptability: whether a sentence is a grammatical English sentence;
- sentiment analysis: whether the text has a positive or negative sentiment;
- semantic equivalence: whether two sentences have the same meaning, or "how similar" (on a scale from 1 to 5) they are; and
- entailment: given a premise sentence and hypothesis sentence, predict whether the premise entails the hypothesis.

Each of these tasks can be treated as a classification task, where the output is a discrete label. Such tasks are typically solved by extracting some kind of fixed-length feature vector from the text sequence and using it as the input to a traditional classifier. A bag-of-words [191], which is a vector containing the number of times each word in the vocabulary appears in the sentence, is a remarkably difficult-to-beat baseline representation. Neural networks can also be used to extract a representation, with different types of networks used in different ways. Convolutional networks use global pooling to aggregate the output activations into a single feature vector [73, 192, 193]. Recurrent networks can use global pooling or take the final hidden state as the feature vector [194]. Transformers can also use pooling; another common method is to feed a special token ("[CLS]") into the model along with the input sequence, and use the output representation at that token's position as the feature vector [195], thereby allowing the model itself decide how to aggregate information using the self-attention mechanism instead of it being hardcoded.

Other NLU tasks require a sequential output. Some of these tasks are described in [189]:

- part-of-speech tagging: labeling each word of a sentence as noun, verb, adjective, adverb, etc.;
- chunking: labeling sections of a sentence as noun phrase, verb phrase, etc.;
- named entity recognition: labeling sections of a sentence as categories such as "person" or "location"; and
- semantic role labeling: labeling the semantic role of each syntactic constituent of the sentence.

Each of these tasks assigns a label to each word. The IOB ("inside, outside, beginning") tagging format [196] is commonly used to label words within a "chunk" (i.e., a noun phrase, a named entity, a verb argument): words at the beginning of a chunk are labeled "B (chunk type)", words thereafter in the chunk are labeled "I (chunk type)", and "filler" words not within a meaningful chunk are labeled "O" — for instance, "find, recent, comedies, by, james, cameron" might be labeled as "O, B-date, B-genre, O, B-director, I-director"

[197]. These tasks have traditionally been solved using conditional random fields (CRF), a discriminative variant of the HMM [198], using the Viterbi algorithm for decoding. Neural networks can be used with [199, 200] or without [201, 202] Viterbi decoding.

For voice control, NLU usually takes the form of intent detection (a classification task) and slot filling (a sequence labeling task, formulated using IOB outputs) [203, 197]. The downsides of treating the problem in this way are that it is somewhat inflexible; it requires word-aligned labels, which require more effort to create and might not be available for some problems; it assumes that the intents and slots are conditionally independent of each other given the input; and most importantly, it cannot directly solve tasks like question answering and machine translation, where an arbitrarily long unaligned output might be required ("Siri, how do you say 'It's all Greek to me' in French?").

In contrast, an encoder-decoder model trained directly to output the information of interest (an approach that is used by e.g. T5 [204, 205]) does not have these flaws. (Encoder-decoder models were first used in [206] for intent detection and slot filling, but with separate decoders for intents and slots and using word-aligned slot labels.) In Chapters 5, 6, and 7 we use a simple and flexible encoder-decoder model that predicts any semantic structure, with no *a priori* information about the intents or slot structure, by splitting the semantics into characters and predicting each character autoregressively. A similar approach is used in [207] and [208], but with a hardcoded set of intents and slots baked into the decoder.

2.2.14 Sesame Street and unsupervised pre-training

Labeling data using human labor is expensive and time-consuming, so in most domains one finds much more unlabeled data than labeled data; for example, YouTube contains millions of hours of unlabeled audio, whereas the largest labeled audio datasets only have on the order of thousands of hours [209]. With larger datasets, more computational power, and improved neural network recipes has come more interest in unsupervised learning algorithms that can take advantage of the ocean of unlabeled data to train a feature extractor potentially useful for many different downstream tasks.

The first great success of unsupervised pre-training was the deep belief network [210, 88], a neural network trained layer-by-layer as a sequence of latent variable models. Deep belief net pre-training greatly improved neural network acoustic models [50, 211], drawing heightened interest in neural networks from industrial speech recognition groups [51]. Unsupervised pre-training fell out of fashion for a time when it became clear that purely supervised training with improved architectures and simple backpropagation could succeed without the need for a separately programmed pre-training step [212, 76].

Another line of research on unsupervised pre-training developed in parallel in NLU. In [73], a convolutional network was trained with multi-task learning on a number of supervised text sequence labeling tasks, as well as a "language modeling" task in which a word in a sentence is possibly replaced with a random other word, and the model is trained to guess using the surrounding context whether the word is replaced. Since no human labels are required for this task, the authors were able to use large unlabeled text corpora (Wikipedia) in addition to the small supervised training sets. Isolated-word models like word2vec [213] pre-trained word embeddings using similar contextual tasks, yielding improved performance in models that reused these embedding weights.

Subsequent work pre-trained LSTMs as conventional language models (predicting the next word given previous words), followed by fine-tuning on downstream supervised learning tasks [214, 194]. The Generative Pre-Trained (GPT) model [215] applied the same idea with transformers, enabling larger models and faster training. To take advantage of both forward and backward contexts, ELMo [216] combined the representations of LMs operating forwards and backwards on the text, instead of just a forward LM. BERT [195] (followed by ERNIE [217], both named after Sesame Street characters, as a nod to ELMo) improved upon ELMo by using a "Cloze" task [218]: instead of predicting the next word using only the previous words, certain words in the input are replaced with a "[`MASK`]" token, and a "bidirectional" (full context, not forward or backward) transformer is trained to predict the masked word given the rest of the sentence. The BERT model outperformed existing work on all GLUE tasks by a large margin.

The success of BERT and other pre-trained models [219] for text led to unsupervised pre-training being revisited for other domains. In audio, Autoregressive Predictive Coding

[220] used an autoregressive LSTM (as in [214, 194]) trained on FBANK features as a pre-trained feature extractor. wav2vec [22] (a play on "word2vec") instead used the raw waveform as input, processing it using a two-tier (causal) convolutional network: one tier with a number of layers downsampling the input sequence, and a second tier that solves a contrastive predictive coding (CPC) task [221]. In CPC, the model is trained to predict, for a collection of samples, which one actually comes next in the sequence, and which are randomly sampled from elsewhere. Contrastive estimation is thought to be an easier task than full autoregressive modeling, and therefore possibly better suited for representation learning, because it does not require reconstructing all the information in the input [222]. In wav2vec 2.0 [7], the authors improve upon wav2vec by using a transformer for the second tier, quantizing the outputs of the first tier by mapping each vector to the nearest vector in a trainable codebook, and using the quantized vectors as training targets for a contrastive loss. wav2vec 2.0 dramatically reduced the amount of labeled data required to train a supervised ASR model. The model became popular with practitioners when Hugging Face provided scripts for finetuning it for ASR using CTC, during an event in which the company invited participants to train ASR models for a number of low-resource languages [223].

Feature extractors obtained through unsupervised (or supervised) pre-training can have their parameters frozen, so that they are not updated when training on the downstream task, or unfrozen, in which case they are updated, possibly using a smaller learning rate [224]. The advantage of a frozen feature extractor is reduced memory consumption and gradient computation during training, and possibly more stable training. The advantage of fine-tuning the pre-trained parameters is that the model may be able adapt better to the downstream task. We will use both techniques in this thesis.

A final note on terminology: what we have here called "unsupervised" is also sometimes called "self-supervised". Though the term "self-supervised" is old [225], it has only recently become more popular: the authors of the BERT and wav2vec papers both refer to their work as "unsupervised", whereas the authors of wav2vec 2.0 and other subsequent work use the term "self-supervised".

2.3 Software

It is often said that the astonishing progress in AI of the last decade has been due to the confluence of larger datasets, improved learning algorithms, and faster computing hardware [226]. Another important factor is open-source software. The experiments described in this thesis make use of a variety of open-source machine learning software tools, especially PyTorch [227]. In addition to allowing programmers to avoid reimplementing commonly used models and algorithms [5, 228], machine learning software has a number of other benefits:

1. **Efficiency:** Building on low-level libraries like the Basic Linear Algebra Subroutines (BLAS) and cuDNN [229], libraries like numpy [230] and (and gnumpy [231]) make it possible to efficiently use modern CPUs, and hardware accelerators like GPUs [212], in a high-level language without writing careful low-level code [232].

2. **Reproducibility:** It is not always easy to read a paper and exactly reproduce the authors' results. While there have been efforts like the "Reproducibility Checklist" [233] to make reproducing experiments easier, often the experiment code itself is the simplest and clearest description of the experiment. There is a strong ethos of "release your experiment code!" in machine learning research, which is helped by standard libraries written in high-level languages and tools like Jupyter Notebooks [234] and Google Colab that provide an interface that interleaves code with textual descriptions and persistent visualizations [235]. It has also become common practice to release the results of experiments as pre-trained models, which machine learning libraries make easy to reuse.

3. **Automatic differentiation:** Tools like Autograd [236] derive the backward pass of a computation from a program for the forward pass, allowing programmers to define sophisticated forward passes without needing to program backpropagation by hand [237]. This essentially cuts the work programmers have to do to implement gradient-based learning in half (or more, since deriving the backward pass of an operation by hand is error-prone).

Deep learning frameworks like Torch [238], Theano [239], TensorFlow [240], PyTorch [227], and JAX [241] combine these concerns with an emphasis on neural networks. Higher-level frameworks like fastai [242] and SpeechBrain (see Chapter 7) implement additional features on top of deep learning frameworks, such as the commonly used tricks described in Section 2.2.7 above.

Chapter 3

Pseudo-Labeling for Massively Multilingual Speech Recognition

Abstract

Semi-supervised learning through pseudo-labeling has become a staple of state-of-the-art monolingual speech recognition systems. In this chapter, we extend pseudo-labeling to massively multilingual speech recognition with 60 languages. We propose a simple pseudo-labeling recipe that works well even with low-resource languages: train a supervised multilingual model, fine-tune it with semi-supervised learning on a target language, generate pseudo-labels for that language, and train a final model using pseudo-labels for all languages, either from scratch or by fine-tuning. Experiments on the labeled Common Voice and unlabeled VoxPopuli datasets show that our recipe can yield a model with better performance for many languages that also transfers well to LibriSpeech.

3.1 Semi-supervised learning

In Chapter 2, we discussed unsupervised learning methods that train a feature extractor on unlabeled data and subsequently train on labeled data. In this chapter, we discuss an alternative (complementary [243, 244]) way of using unlabeled data, semi-supervised

learning, which uses both labeled and unlabeled data to train a model [245, 246].

A few assumptions about the data distribution are reasonable for many domains:

1. Points belonging to the same cluster found in the data are likely to have the same label.

2. More confident predictions on unlabeled data by a model trained on labeled data are more likely to be correct.

3. "Low-density separation": the decision boundary for a classification problem lies in a low-density region of input space.

4. Augmentations or small changes to the input do not change the correct output. (A scenario where this assumption would not apply is if the label is "noisy/not noisy".)

Semi-supervised learning methods try to take advantage of these assumptions to learn from the unlabeled data. For example, Assumption 2 can be combined with Assumption 4 by training on augmented versions of unlabeled examples for which the supervised model predicts confidently for the nonaugmented versions.

3.2 Self-training

A number of semi-supervised learning algorithms of varying complexity and with different assumptions have been proposed [247]. One very simple method that has been found to work well across domains, dataset sizes, and models is self-training, or pseudo-labeling [248, 249]. Let D_L denote a labeled dataset of $\{\mathbf{x}, \mathbf{y}\}$ pairs and D_U denote unlabeled data with $\mathbf{x}$ only. In self-training, a supervised model is trained on D_L and used to generate pseudo-labels for (some subset of) D_U for use in "pseudo-supervised"[1] learning.

Self-training can be regarded as an instance of the EM algorithm (Section 2.2.3), with labels in the place of latent variables, where the latent variables are known for some data points and not for others.[2] In hard pseudo-labeling, each data point is assigned to exactly

[1] Pseupervised?

[2] Cf. Chapter 3 of [246].

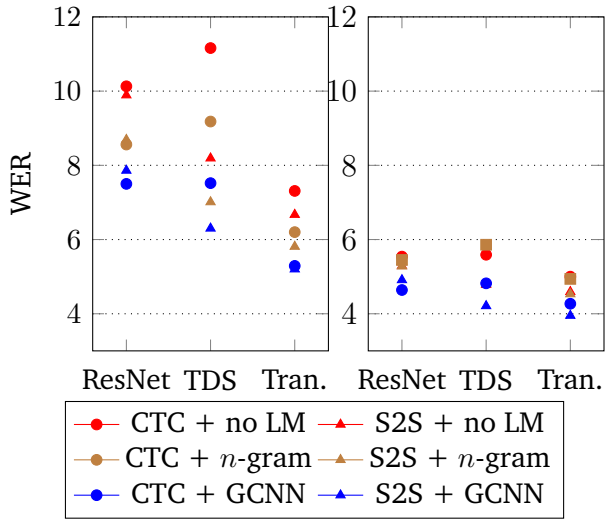


Figure 3.1: Supervised performance (left) vs semi-supervised performance (right) of different models on LibriSpeech. (Figure reproduced from [6].)

one latent z in the E-step (as in the Viterbi algorithm); in soft pseudo-labeling [250], a distribution over z is used. Hard pseudo-labels are often easier to work with because they require less memory and can be stored in a text file and easily inspected by a human.

3.2.1 Iterative pseudo-labeling

The simplest form of self-training trains a single supervised model to convergence with D_L, uses it to generate pseudo-labels for D_U, and then trains a model on the labeled and pseudo-labeled data to convergence using the original supervised learning method [251]. In ASR [252], better results can be obtained by using data augmentation (such as SpecAugment [105] or speed perturbation [104]) on the unlabeled audio, generating better pseudo-labels using beam search decoding with an external language model, and filtering out pseudo-labels with a length not expected relative to the length of the input signal (e.g., we would not expect a one-word transcript for a 30-second long recording). Fig. 3.1 from [6] shows the effect of combining the 1,000 hour labeled LibriSpeech dataset with 60,000 hours of pseudo-labeled LibriVox data: WER dramatically improves for the semi-supervised models across all types of acoustic models and language models, and the gap between different neural network architectures is reduced.

Given that a model trained on both labeled and pseudo-labeled data has better per-

formance than the original supervised model, might it not produce better pseudo-labels? This is the idea behind iterative pseudo-labeling (IPL) [253, 254]. In IPL, a sequence of models is trained on pseudo-labels generated by previous models, and iterations can continue until validation performance stops improving. The IPL setup is somewhat similar to old-school HMM ASR recipes (Section 2.2.5 and 2.2.8), but with no constraints on the type of models and labels used in each round. If the same neural network architecture is used in each round, it is possible to continue training the model by initializing its weights to the weights from the previous round; but in some instances, training new weights from scratch has been found to work better.

3.2.2 Continuous pseudo-labeling and slimIPL

IPL works by pseudo-labeling the entire unlabeled dataset before training, and subsequently does not update the pseudo-labels until convergence. Just as it is sensible to use online learning algorithms like SGD instead of batch learning algorithms for large-scale datasets [255], it is sensible to use online or continuous pseudo-labeling instead of batch IPL for semi-supervised learning [256, 257]. Online pseudo-labeling methods avoid "stale" pseudo-labels from an older model for better results by immediately using the best pseudo-labels as the model improves.

However, online methods seem to be more unstable, often diverging or producing empty transcripts. Two strategies have been developed to deal with the instability of online pseudo-labeling:

1. Generating pseudo-labels using a second model whose weights are an exponential moving average (EMA) of the (more quickly) changing weights of the model being trained [258, 259]. Additional memory is required to store the EMA weights.

2. Using a cache of pseudo-labels generated from previous minibatches. Memory is required to store the cache, but only one copy of the model is required.

The caching method is used by slimIPL [100], which uses greedy decoding instead of beam search decoding to produce pseudo-labels (Algorithm 8). Using greedy decoding speeds

up pseudo-label generation and, by not using a language model, avoids letting the acoustic model overfit to the language model, which has been observed in the basic self-training setting [6]. slimIPL yields state-of-the-art results for LibriSpeech with additional unlabeled data, using fewer GPU hours, even outperforming methods that *do* use a language model to generate pseudo-labels.

Algorithm 8: slimIPL

Train acoustic model $\mathcal{M}_\theta$ on D_L with augmentation for M updates
while *cache is not full at size* C **do**
  Draw a random batch of $\mathbf{x}$ from D_U
  Generate pseudo-label $\hat{\mathbf{y}}$ using $\mathcal{M}_\theta$ with greedy decoding
  Store $\{\mathbf{x}, \hat{\mathbf{y}}\}$ in cache
  Train $\mathcal{M}_\theta$ on D_L with augmentation for 1 update
Decrease dropout of $\mathcal{M}_\theta$
while *not bored* **do**
  Train $\mathcal{M}_\theta$ on D_L with augmentation for N_L updates;
  for N_U *updates* **do**
    Draw a random batch $B = \{\mathbf{x}, \hat{\mathbf{y}}\}$ from the cache
    w.p. p, remove B from cache and add new $\mathbf{x}' \in D_U$ and its pseudo-label $\hat{\mathbf{y}}'$ generated by $\mathcal{M}_\theta$ to cache
    Apply augmentation to batch B and make an optimization step to update $\mathcal{M}_\theta$

3.3 Multilingual ASR

One of the long-term goals of automatic speech recognition (ASR) research is a single system that can transcribe speech in any language [260, 261]. Such a multilingual system would be simpler to maintain than a collection of monolingual models, enable users to comfortably speak any language without needing to tell the system which language to expect in advance, and share knowledge between all languages for improved performance.

Given that pseudo-labeling has become a key ingredient of state-of-the-art monolingual ASR systems, we propose in this chapter to go beyond the monolingual setting and demonstrate the use of pseudo-labeling to improve a massively multilingual speech recognizer trained on all 60 languages of the Common Voice dataset [262] simultaneously. First, we show that self-training on all unlabeled data in the multilingual VoxPopuli dataset [263] at

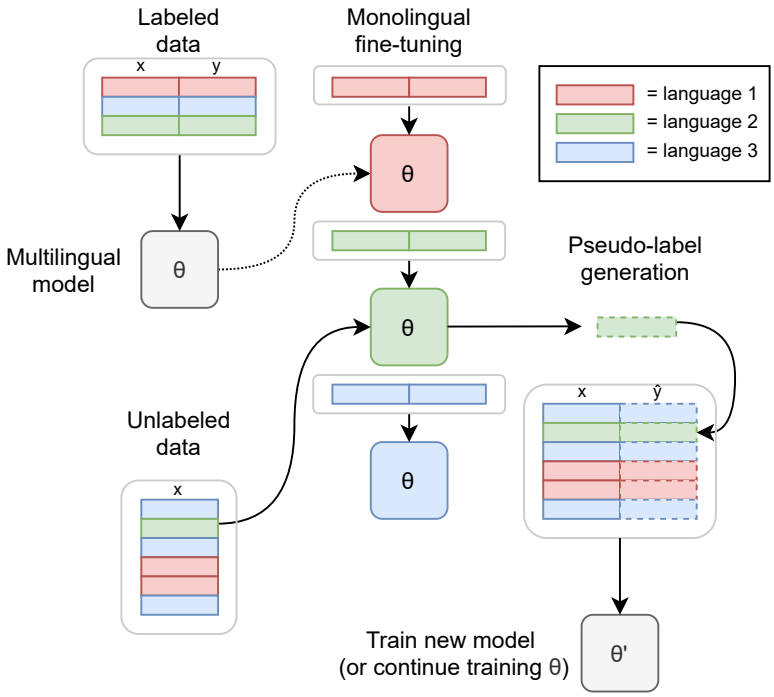


Figure 3.2: Illustration of our method: to produce better pseudo-labels for a given language, we first fine-tune the multilingual model on that language.

once tends to produce poor PLs for low-resource languages, and instead propose a simple recipe (Fig. 3.2) in which the model is first fine-tuned for a particular language before pseudo-labeling. Next, we compare a number of methods for training with the generated PLs, and find that training a larger model from scratch on all labeled and pseudo-labeled data, followed by fine-tuning on labeled data, works best. Finally, we show that the use of pseudo-labeled data improves out-of-domain generalization through experiments on LibriSpeech [264]. Unlike much previous work on this topic, our experiments use only open-source data, and we release our code and models for those who would like to experiment with them further.[3]

[3]For Flashlight code, model checkpoints, and a Colab notebook showing how to perform inference, see: `https://github.com/flashlight/wav2letter/tree/main/recipes/mling_pl`. The PyTorch version can be found at `https://huggingface.co/speechbrain/m-ctc-t-large`.

3.4 Model

The model used in our experiments (Fig. 3.3) is identical to the neural network used for LibriSpeech in [100], except for the output layer(s). The input to the encoder is a sequence of 80-dimensional log mel filterbank frames, extracted using 25 ms Hamming windows every 10 ms from the 16 kHz audio signal. The encoder has a single convolutional layer with a filter length of 7 and a stride of 3, followed by 36 transformer layers with 4 heads, feedforward dimension 3072, and self-attention dimension 768, using the relative position embeddings of [123]. The output of the encoder is fed to a CTC [155] head and a language identification (LID) head. The CTC head is a linear layer with 8065 outputs: one for each character (most of which are Chinese characters), including punctuation, space, and the CTC <blank> symbol. The CTC head is shared across all languages: it is a "joint" multilingual model, using the terminology of [261]. The LID head is a linear layer with 60 outputs (one per language), followed by mean-pooling to aggregate the variable-length sequence of output vectors into a single vector of logits. The LID head outputs are only used during training: during inference, standard decoding algorithms can be applied to the CTC head outputs. The model — which we refer to as M-CTC-T ("multilingual CTC transformer") — is implemented and trained using Flashlight [160].

While we do not perform explicit empirical comparisons with other multilingual models in the literature (as the focus of this work is on pseudo-labeling), it is worth noting that M-CTC-T is significantly simpler than existing multilingual models, forgoing the use of language- or language-family-specific parameters, decoders, and tokenizers. We are not the first to use an encoder-only CTC architecture for multilingual ASR [265, 266, 267], but we believe we are the first[4] to demonstrate this for *massively* multilingual end-to-end ASR. Previous work on this topic [268, 269, 270, 271, 272] has instead used more sophisticated sequence transduction models with autoregressive decoder networks [153, 151, 152, 175, 188], citing the flaw of CTC's conditional independence assumption. In practice, CTC models implemented using modern neural network architectures are able to learn strong implicit language models [6, 100] and achieve state-of-the-art results for the

[4]We learned afterwards of an unpublished similar model based on XLSR-53 [267]: `https://t.co/yrr7pfuVVo`

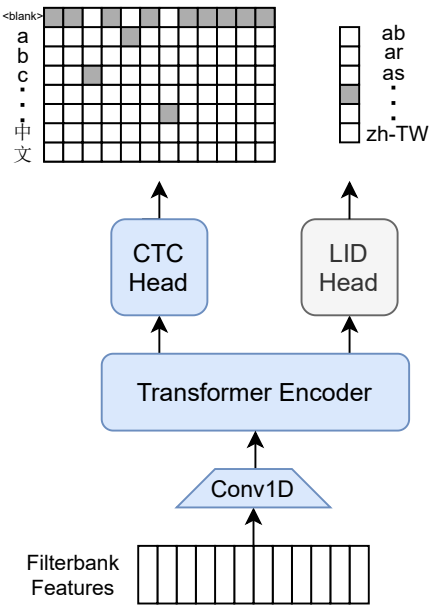


Figure 3.3: Illustration of M-CTC-T, with optional language identification head (Sec. 3.6) shown in grey.

low-resource setting [7, 100]. For those reasons, we focus on CTC models here.

3.5 Data

The model is trained using the December 2020 release (6.1) of Common Voice (CV) [262], which has 3.6k hours of training data. CV is a continuously growing multilingual speech dataset recorded online by volunteer speakers. The 60 constituent languages vary greatly in the amount of available data: 7 languages have more than 100h of data, and 10 languages have less than 1h of data. We do not remove punctuation and capitalization from the CV transcripts, as this makes it easier to replicate our setup[5] and learning speed was not noticeably impacted. We downsample all audio to 16 kHz.

In addition to CV, we use VoxPopuli (VP) [263], a very large scale (384k hours) unlabeled multilingual dataset of European languages. The dataset is split into 23 languages. 19 of the 23 VP languages are in CV (Czech, German, Greek, English, Spanish, Estonian, Finnish, French, Hungarian, Italian, Lithuanian, Latvian, Maltese, Dutch, Polish,

[5]While there have been attempts to standardize the formatting of transcripts for Common Voice for English [273], most reported results use an ad-hoc normalization scheme, and so cannot readily be compared.

Portuguese, Romanian, Slovenian, and Swedish): we use only those 19 languages for semi-supervised learning.

3.6 Supervised training

We train supervised models on CV for ∼500k updates. The hyperparameters and training procedure are identical to those used in [100], except we use 2 SpecAugment [105] time masks instead of 10 (using 10 masks was found to cover too much of the shorter CV audio), and the learning rate is halved just once, at 250k updates. We do not use the language balancing technique of [269, 261] to sample languages evenly (which we found easily overfit to the low-resource languages), or curriculum learning as in [261]. In addition to the base model (275M params), we also train larger models (1.06B params) by doubling the feedforward and self-attention dimensions of the transformer layers. The base models are trained on 16 GPUs with dynamic batching using 200s of audio per batch per GPU, and the large models are trained using 64 GPUs with 50s of audio per GPU, resulting in the same effective batch size.

Following [274], we add an LID loss, so that the loss ℓ used for training is $\ell = \ell_{\text{CTC}} + \gamma \cdot \ell_{\text{LID}}$, where ℓ_{CTC} represents the CTC loss, ℓ_{LID} represents the LID loss (the cross-entropy between the LID head outputs and the one-hot language label for a given utterance), and γ is a hyperparameter. We trained models on CV with $\gamma \in \{0, 0.1, 1, 10\}$: $\gamma = 1$ yielded the best results, with 2.6% absolute improvement in average validation character error rate (CER) over the baseline with $\gamma = 0$ (no LID), using greedy decoding. Some examples of greedy decoding outputs for the base supervised model are shown in Fig. 3.4 and Fig. 3.5.

3.7 Semi-supervised training

To train on the unlabeled data in VP, we use slimIPL with a cache size of 1000, replacement probability 0.1, and $\lambda = 10$ (ratio of unlabeled batches to labeled batches).

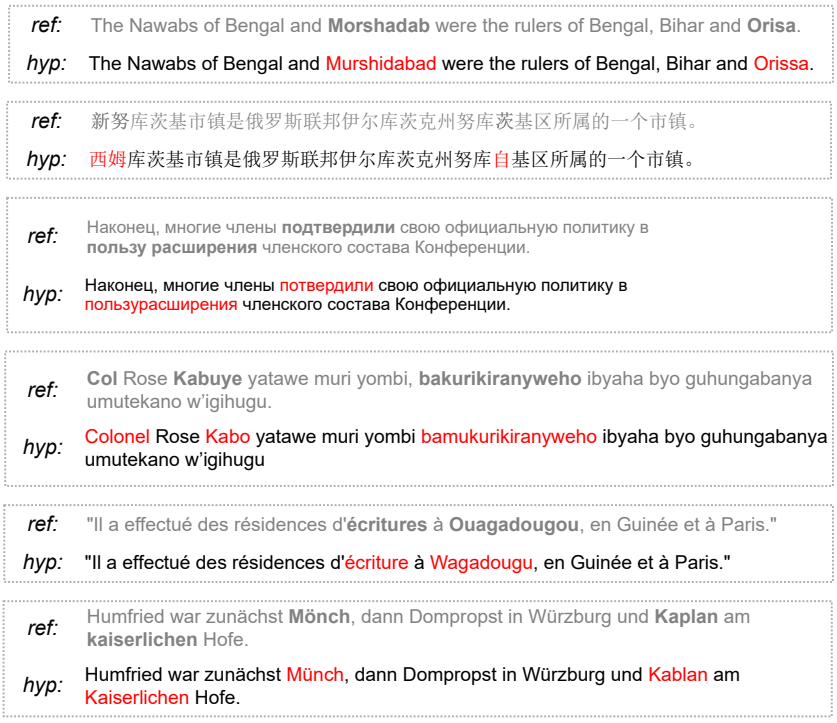


Figure 3.4: Example greedy decoding outputs from the base supervised model for 6 utterances from the validation sets of some of the higher-resource CV languages: English, Chinese (China), Russian, Kinyarwanda, French, and German.

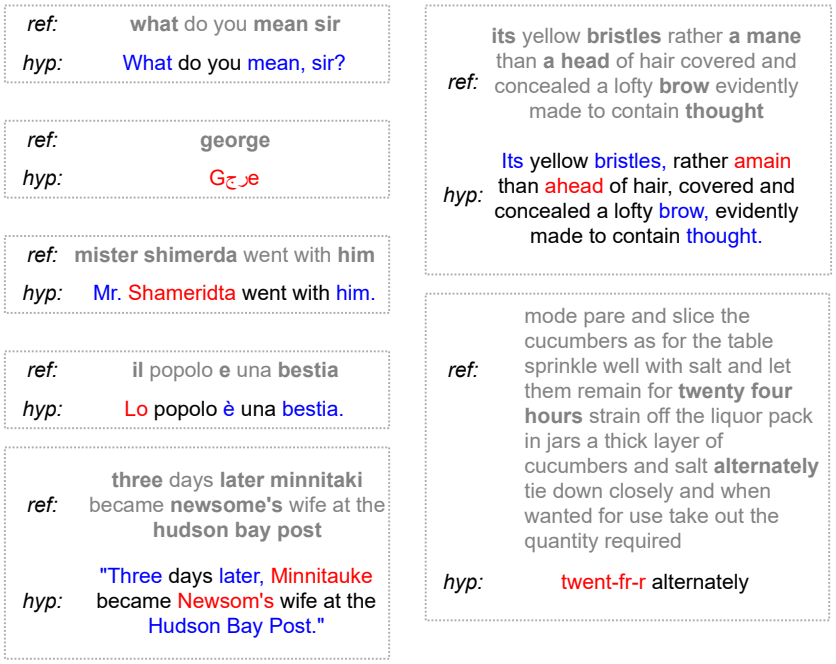


Figure 3.5: Examples of LibriSpeech dev-clean outputs with greedy decoding for base supervised model, trained only on CV, not on LibriSpeech. (Substitutions are colored: red = genuine error, blue = punctuation/truecasing counted as error.) Note that the model almost correctly transcribes the unusual Italian sentence in dev-clean, unlike a typical LibriSpeech model (cf. [7, Table 12]). The supervised model fails to transcribe the final, longer utterance — see Section 3.7.2.

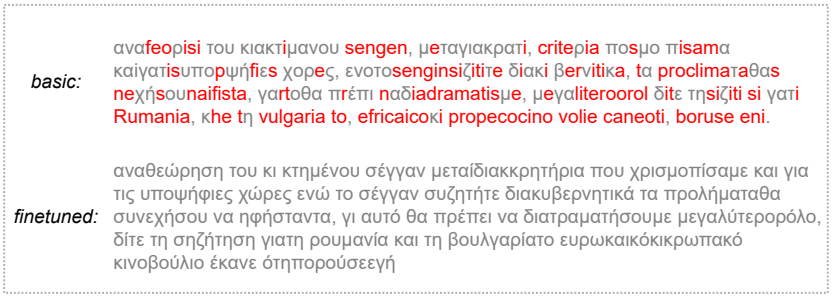


Figure 3.6: Pseudo-labels for an utterance from the Greek subset of VP with basic slimIPL (top) or with slimIPL after monolingual fine-tuning (bottom). Red letters are Latin characters.

Table 3.1: (Semi-)supervised learning results with slimIPL for the CV Greek data given different training sets.

Labeled	Unlabeled	Valid CER	Test CER
CV All	–	53.2	47.8
CV Greek	–	30.6	33.6
CV Greek	VP Greek	23.9	25.1
CV Greek	VP English[6]	24.3	28.4
CV All $\rightarrow$ CV Greek	–	9.9	9.6
CV All $\rightarrow$ CV Greek	VP Greek	8.7	8.5

3.7.1 Fine-tuning before pseudo-labeling

The simplest way to perform semi-supervised learning would be to pool the unlabeled data for all languages, as we do for the labeled data, and run slimIPL. We found that doing so led to poor PLs for low-resource languages, such as Greek, which has only 2.75h of training data (see top of Fig. 3.6 — the transcript has a mix of Greek and Latin characters).

Instead, to produce PLs for a VP language, we first fine-tune the trained multilingual model by training only on CV data for that language for 10k updates, and then run slimIPL using the corresponding VP data (bottom of Fig. 3.6). The same effect could also be achieved by generating PLs using a monolingual model, but our proposed approach yields better results by taking advantage of multi-task learning (Table 3.1).

After training slimIPL models for all 19 languages in (CV languages $\cap$ VP languages),

[6]See Sec. 3.10 for a more detailed discussion of semi-supervised learning with language mismatch.

we generate a final set of PLs[7] for all unlabeled VP utterances using the appropriate slimIPL models. We filter out all utterances for which the PL length is 0 or >630 (maximum label length supported by the CTC loss implementation). The PLs for all languages can then be pooled and used either by continuing training the non-fine-tuned multilingual model checkpoint with all available CV and VP data, or by training a new model on that data from scratch. When training a model from scratch, we found it necessary for convergence to lower the learning rate from 0.03 to 0.01 and to delay SpecAugment until 50k updates; we also lower the learning rate when using VP data to fine-tune the base model already trained on CV.

Distilling the per-language fine-tuned models' knowledge back into a single final model is similar to the recently proposed multi-task self-training (MuST) [275]. In MuST, a separate teacher model is trained for each task and used to pseudo-label every available training example, and a general student model with one head for each task is then trained on all the pseudo-labels. The difference here is that our final model only performs one "task", since we use a single shared CTC head over all languages, and the model itself must determine which language is being spoken.

3.7.2 Avoiding collapse: cropping warmup period

Another difficulty arose from the fact that the utterances of VP (average duration of 30s) are much longer than those of CV (average duration of 5.3s). The model trained only on CV generates mostly empty transcripts for VP, a commonly observed failure mode for out-of-domain audio or utterances longer than those observed during training [258, 124, 276]. Semi-supervised learning failed as a result, usually collapsing to generating all blanks even for the labeled data. To acclimate the model to the longer VP utterances, we use a warmup period of 10k updates during which we crop unlabeled audio into 10s segments before running the acoustic model, then stitch the resulting logit sequences back together and decode to obtain PLs. The model is then trained on the original uncropped utterance using those PLs. Cropping the utterances results in poor pseudo-labels, so after a number of

[7] Pseudo-labels can be found at `https://dl.fbaipublicfiles.com/wav2letter/mling_pl/all_pseudo_labeled.lst`

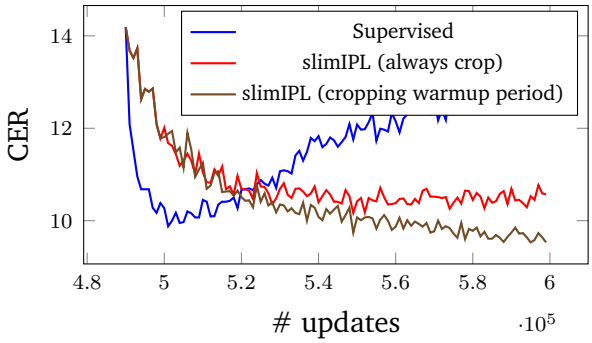


Figure 3.7: Validation CER for CV Greek (after training on CV All) with supervised fine-tuning or semi-supervised fine-tuning with VP Greek via slimIPL, using either a cropping warmup period or always cropping.

updates, we stop cropping the unlabeled utterances during pseudo-labeling. This warmup period approach works better than simply always cropping (Fig. 3.7).

3.8 Performance on Common Voice

Table 3.2 lists the performance of the multilingual model averaged over all CV languages in various settings.[8] Table 3.3 reports the same information for CV languages that are in VP. All results for CV are reported using greedy decoding in terms of character error rate (CER), as suggested in [262].

In addition to the base model (trained only on CV), we report performance when the VP audio with the final set of PLs is added back into the training set, either by fine-tuning the model already trained on CV ("+ all PLs (fine-tune)") or by training a model from scratch on CV+VP ("+ all PLs (from scratch)"). We only report results for the large model when training it from scratch on CV+VP, as the large model overfit to CV after a few epochs (see Fig. 3.8, "CV (large)"). Test CER is measured by selecting the checkpoint with the best average validation CER across all languages. While performance is degraded on average (Fig. 3.8), it is greatly improved for the VP languages (Fig. 3.9a), with the best results achieved training a larger model from scratch.

The degradation for CV languages on average can be explained by the fact that VP

[8]Detailed per-language training logs and decoded outputs for all 60 languages can be found at `https://dl.fbaipublicfiles.com/wav2letter/mling_pl/supplementary.zip`.

Table 3.2: CER averaged over all CV languages.

Model	Valid CER	Test CER
Base model	26.8	28.8
+ all PLs (fine-tune)	27.6	29.7
+ all PLs (from scratch, base)	38.0	39.9
$\hookrightarrow$ fine-tune on CV only	26.6	28.2
+ all PLs (from scratch, large)	33.0	34.9
$\hookrightarrow$ fine-tune on CV only	21.4	23.3
Monolingual baseline	33.8	35.5
Supervised fine-tuning	10.6	11.4

is much larger than CV, leading to an imbalance in favor of the 19 languages in (CV languages $\cap$ VP languages). If we then fine-tune the models trained on CV+VP on *only* CV ("$\hookrightarrow$ fine-tune on CV only"), they not only still have improved performance over the base model when averaging over the VP languages, but also close the gap when averaging over all CV languages.

We also train a monolingual model for each language separately using the same hyper-parameters as the multilingual model, and report the performance of those models along with the performance of the multilingual model when fine-tuned using only labeled data for that language ("supervised fine-tuning") or, when unlabeled data is available (Table 3.3), using both labeled and unlabeled data for that language ("slimIPL fine-tuning"). For monolingual models, or multilingual models with monolingual fine-tuning, the test CER is measured using the checkpoint with the best validation CER. There is still a large gap between the base model and fine-tuned models (see e.g. Greek in Table 3.1), but the gap is reduced for the VP languages when training on the pseudo-labeled data.

Fig. 3.10 shows the performance of the multilingual model being improved by pseudo-labeled data for three low-resource CV languages. In Fig. 3.11, for three high-resource languages, performance is worse when fine-tuning on CV+VP and much worse when training a new model from scratch on CV+VP, but the performance gap is closed by fine-tuning the larger model on CV only.

There is a straightforward explanation for why the model trained from scratch on CV+VP initially performs so much worse on Catalan and Kabyle, before the model is fine-

Table 3.3: CER averaged over languages in (CV languages ∩ VP languages).

Model	Valid CER	Test CER
Base model	24.4	24.8
+ all PLs (fine-tune)	17.5	17.9
+ all PLs (from scratch, base)	15.0	15.6
↪ fine-tune on CV only	13.8	14.0
+ all PLs (from scratch, large)	11.5	12.0
↪ fine-tune on CV only	10.1	10.6
Monolingual baseline	25.1	26.8
Supervised fine-tuning	7.7	8.3
slimIPL fine-tuning	6.9	7.5

tuned only on CV: those languages are not in VP, so the amount of training data observed by the model for those languages is dwarfed by the amount of training data observed for the VP languages. However, English is among the VP languages, so it is surprising that the performance of English is also worse for the model trained from scratch on CV+VP, and that performance becomes worse when the model trained on CV is fine-tuned on CV+VP. It is worth noting that VP data is somewhat noisy: much of it is spoken by interpreters attempting to translate, in real-time, what is being said by another speaker in another language—sometimes stumbling over a word or repeating themselves. The domain mismatch between this type of speech, as opposed to the prompted speech in CV, may explain the performance gap. Even though performance is degraded for the English subset of CV, the use of VP data does improve the model's ability to process English in a new domain, as we show in the next section.

3.9 Transferring to LibriSpeech

To see how well the multilingual models perform on out-of-domain audio, we evaluate them on LibriSpeech in Table 3.4. Word error rate (WER) is reported both using greedy decoding and using a beam search for

$$\underset{\mathbf{y}}{\operatorname{argmax}} \ \log p_\theta(\mathbf{y}|\mathbf{x}) + \alpha \log p^{\mathrm{LM}}(\mathbf{y}) + \beta|\mathbf{y}|, \tag{3.1}$$

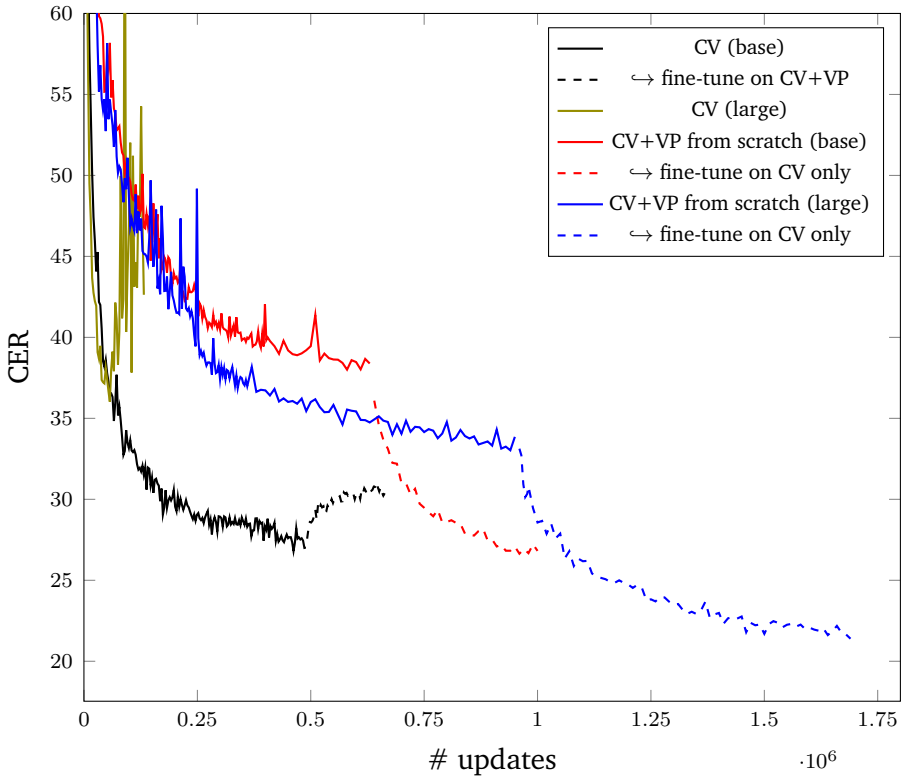
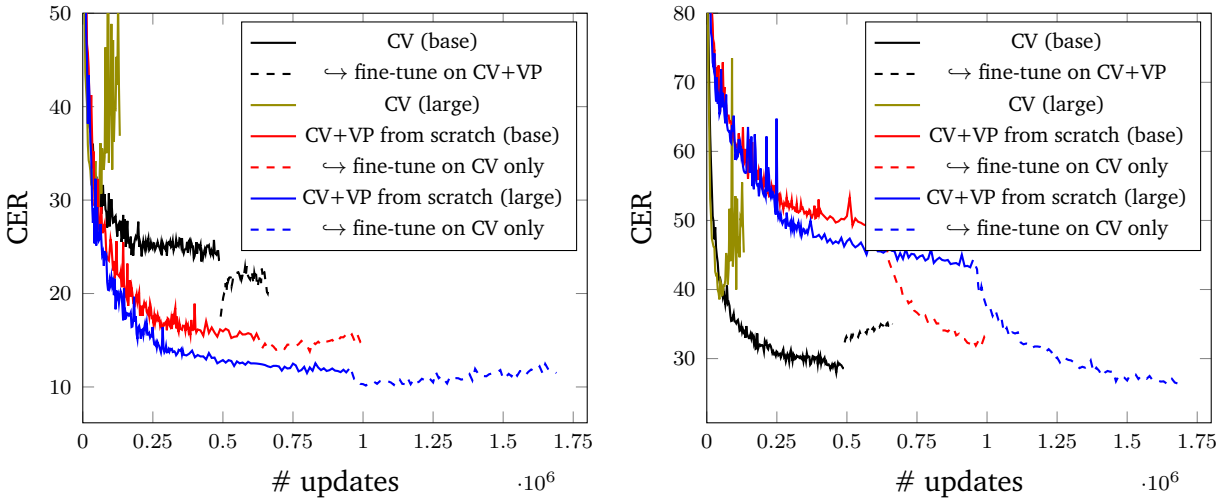


Figure 3.8: Validation CER curves for CV averaged over all languages for various training settings.

where $p_\theta(\mathbf{y}|\mathbf{x})$ is the probability of transcript $\mathbf{y}$ given input audio $\mathbf{x}$ according to the acoustic model, $p^{\text{LM}}(\mathbf{y})$ is the probability of $\mathbf{y}$ according to an external 4-gram word-level LM trained on the LibriSpeech LM corpus, $|\mathbf{y}|$ denotes the length of $\mathbf{y}$, and α, β are set using a small grid search on the dev sets. We find that the multilingual model fine-tuned with all VP PLs performs much better on LibriSpeech across all settings. It can be seen from Table 3.5, in which test-other is split by the duration of utterances, that the improvement is due mostly to the model's ability to process longer sequences acquired from training on the longer VP utterances (see Sec. 3.7.2).

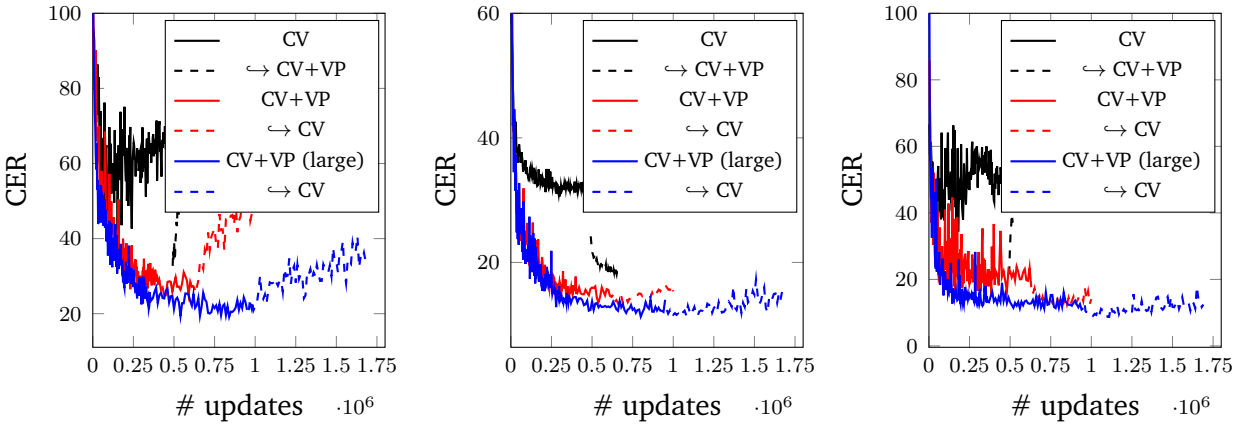
We also demonstrate the base model's transfer capability by fine-tuning it either on the 100h or 960h subset of LibriSpeech (Table 3.4, "CV $\rightarrow$ LS-{100,960}"). During fine-tuning, instead of 2 SpecAugment masks (Sec. 3.6), we use 10 masks, as in [100], which we found yielded better performance. With fine-tuning on LibriSpeech, performance is greatly improved for the 100h setup over the 100h-only training, while with 960h performance is similar or slightly worse. We have not yet made these comparisons for the CV+VP models,



(a) CV languages ∩ VP languages.

(b) CV languages \VP languages.

Figure 3.9: Validation CER curves for CV when averaging over the subset of languages in VP (left) and the subset of languages *not* in VP (right).

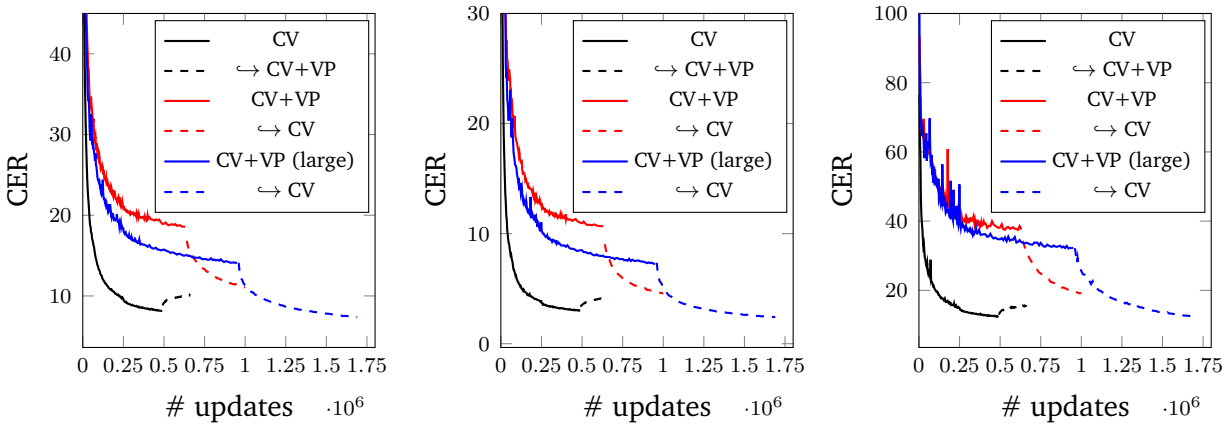


(a) Greek — 2.75h

(b) Finnish — 0.55h

(c) Lithuanian — 1.18h

Figure 3.10: Validation CER curves for the base and large multilingual models' performance on three low-resource CV languages with a corresponding subset in VP.



(a) English — 897h (b) Catalan — 444h (c) Kabyle — 109h

Figure 3.11: Validation CER curves for the base and large multilingual models' performance on three high-resource CV languages: English ($\in$ VP), Catalan ($\notin$ VP), Kabyle ($\notin$ VP).

but our other results suggest that similar benefits may be observed.

3.10 Pseudo-labeling the wrong language

To our surprise, we found that training a monolingual speech recognizer by pseudo-labeling the *wrong language* could also improve test performance. Fig. 3.12 shows the validation CER of CV Greek when no unlabeled data, unlabeled data in the right language (VP Greek), and unlabeled data in the wrong language (VP English) is used.

This result may not currently be of much practical interest, since we can easily train a

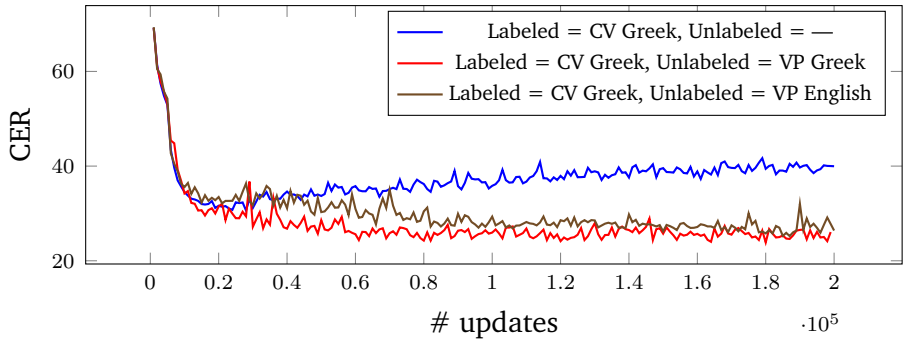


Figure 3.12: Validation CER for CV Greek for purely supervised monolingual training on CV Greek, using VP Greek as unlabeled data for slimIPL, or using VP English as unlabeled data. It's all Greek to me, indeed.

Table 3.4: LibriSpeech WER for different training sets (275M parameter model).

Data	LM	Dev WER		Test WER	
		clean	other	clean	other
CV	-	59.7	60.1	62.0	62.8
	4-gram	33.7	34.3	37.6	37.7
CV	-	34.1	41.7	33.5	42.5
→ CV+VP	4-gram	8.8	15.9	9.0	16.8
CV+VP	-	39.7	47.9	39.0	49.4
→ CV	4-gram	10.1	17.8	10.4	19.5
CV	-	4.8	13.7	5.1	13.6
→ LS-100	4-gram	3.3	9.7	3.8	9.9
CV	-	3.0	7.5	3.1	7.4
→ LS-960	4-gram	2.1	5.3	2.6	5.8
LS-100	-	6.2	16.8	6.2	16.8
	4-gram	4.1	12.4	4.5	12.7
LS-960	-	2.7	6.8	2.8	6.9
	4-gram	2.0	5.1	2.6	5.7

Table 3.5: WERs for test-other split over audio duration.

Data	LM	Duration			
		<10s	10-15s	15-20s	>20s
CV	-	46.6	83.6	99.1	99.9
	4-gram	15.6	54.7	93.3	98.7
CV	-	43.5	38.6	41.3	47.7
→ CV+VP	4-gram	17.0	15.2	17.2	20.8
CV+VP	-	48.2	47.3	52.8	63.5
→ CV	4-gram	17.9	18.8	23.0	33.3

better monolingual Greek speech recognizer through other methods (Table 3.1). Still, we believe it may be useful for understanding how and why semi-supervised learning works, and we hope to explore the phenomenon for more language pairs in the future.

3.11 Conclusion

We have demonstrated the use of pseudo-labeling to improve an end-to-end joint model for massively multilingual ASR with Common Voice. Fine-tuning a multilingual model with semi-supervised learning on each language of VoxPopuli separately, and then training on all VoxPopuli pseudo-labels combined, i) significantly improves the performance of the model for those 19 languages, ii) helps the model generalize to a new domain (LibriSpeech), and iii) enables training a larger model than was possible with Common Voice alone without overfitting. Many interesting questions and problems remain, such as reducing the gap between the performance of the multilingual model on its own and after fine-tuning on a particular language, improving performance for languages without unlabeled data, integrating language models into the PL generation process, and running iterative pseudo-labeling instead of a single round with all languages. The method we have employed requires knowledge of which language is spoken in the unlabeled audio: overcoming this requirement, so that even more data in the wild can be used, would also be worth exploring.

Chapter 4

Surprisal-Triggered Conditional Computation with Neural Networks

Abstract

Autoregressive neural network models have been used successfully for sequence generation, feature extraction, and hypothesis scoring. This chapter presents yet another use for these models: allocating more computation to more difficult inputs. In our model, an autoregressive model is used both to extract features and to predict observations in a stream of input observations. The surprisal of the input, measured as the negative log-likelihood of the current observation according to the autoregressive model, is used as a measure of input difficulty. This in turn determines whether a small, fast network, or a big, slow network, is used. Experiments on two speech recognition tasks show that our model can match the performance of a baseline in which the big network is always used with 15% fewer FLOPs.

4.1 Introduction

In "Thinking, Fast and Slow", Daniel Kahneman hypothesizes that human cognition operates in one of two modes: "System 1" cognition, which is fast, automatic, and effortless, and "System 2" cognition, which is slow, deliberate, and effortful [277, 278]. What de-

termines whether System 1 or System 2 is active at a given time is roughly the current level of cognitive ease: most of the time System 1 dominates, and only when something breaks down and the environment becomes difficult to predict or control does System 2 activate. An example of experimental support for this hypothesis, or at least for the weaker hypothesis that environmental surprisal controls cognitive effort in some way, can be found in studies of reading time: words that are surprising (in the sense that a statistical language model assigns lower probability to them), as well as the words that follow, require more time for human subjects to read, suggesting that more effort is being used [279, 280, 281, 282].

In contrast to human cognition, the deep neural networks used in artificial intelligence typically do not perform any less computation for any input: the model always multiplies the input by the same sequence of weight matrices to compute an output, no matter how difficult or easy the input may be. This seems like a waste of energy. As neural networks have gotten bigger [283, 284] and more expensive to run [285, 286], it has become more pressing to find ways to address this waste. The question this chapter asks is: can we emulate human cognition to improve the computational efficiency of neural networks?

To try to answer this question, we present a simple model of conditional computation for neural networks that mirrors the System 1/System 2 division of labor. The model is depicted in Figure 4.1. An autoregressive neural network model is used to process a stream of input observations, both to extract features and to predict the next observation. A small, fast network, loosely analogous to System 1, runs most of the time, minimizing the amount of computation that must be performed on average, and a big, slow network, loosely[1] analogous to System 2, runs only when the autoregressive component is unable to accurately predict the current input.

In addition to resembling certain theories about human cognition, our model can be thought of as taking advantage of the "low-density separation" assumption of semi-supervised learning [246, p.7] (Section 3.1): namely, that the optimal decision boundary for a classifier lies in a low-density region of the input space. The autoregressive model

[1]Here we are not considering more "logical" operations, like sequential reasoning and symbol manipulation; rather, we are focusing on the aspects of this model relevant to the control of computational resources.

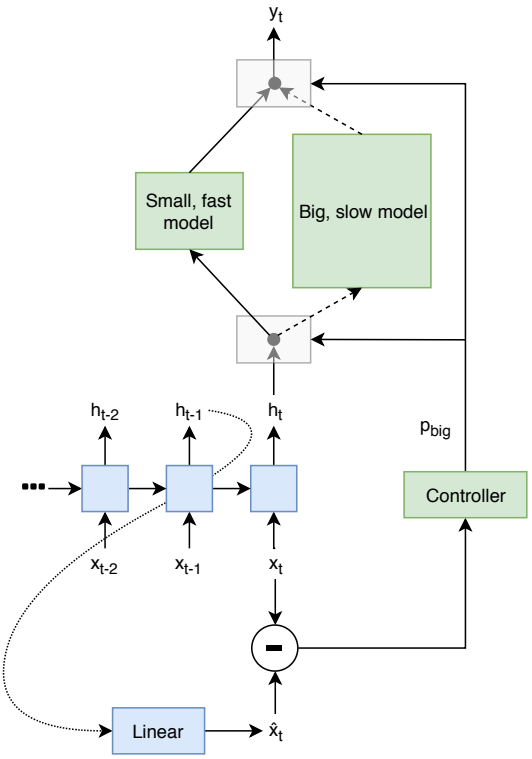


Figure 4.1: Architecture of the proposed system. Here the autoregressive component of the model (blue) is depicted as an RNN.

in our setup explicitly detects when the input is in such a region. Under the low-density separation assumption, the less surprising the input is, the farther away the input is from the complex decision boundary defined by the big network. Hence, it may be expected that less error will be incurred when approximating that decision boundary using the small network—though for now, we have no formal proof of how good this approximation might be.

In our experiments, we observe an improved tradeoff between computation and accuracy using our model: it can perform as well as or even better than models that always use the big network at lower cost. The improvement is consistent over a variety of hyperparameter settings for two datasets, which gives strong evidence that surprisal is a useful inductive bias for conditional computation. Another interesting contribution our work makes is that while other work has used autoregressive models *either* as pre-trained feature extractors *or* for predicting future observations, our work seems to be the first to show that it is possible and worthwhile to use them for *both* functions at the same time.

4.2 Related work

Conditional computation In conditional computation, only a fraction of a model's parameters are used to process any given input. Some machine learning algorithms, like decision trees, natively support conditional computation, but neural networks do not. In the last few years, researchers have begun thinking more about how to incorporate conditional computation into neural networks [287, 288].

Perhaps the two most commonly used approaches to conditional computation with neural networks are the early exiting and score margin techniques. In early exiting, a classifier is attached to an intermediate layer of a network and trained to determine whether stopping at that layer will result in a misclassification; if a misclassification is not predicted, subsequent layers are not computed [289, 290, 291, 292, 293]. In the score margin approach, a small model is used to compute scores for each class in a classification problem; if the margin between the largest score and the second largest score is below a certain threshold, then a bigger model is used [294, 295, 296, 297, 298].

In other approaches, the model itself learns when to use its various components. This often takes the form of a mixture-of-experts [299, 300], in which a learned controller is used to select the experts relevant for a given input [301, 302, 303, 304, 305, 306], possibly hierarchically [307, 308, 309, 310]. The binary decision of selecting or not selecting an expert is not differentiable, so it is not possible to perform standard backpropagation in a mixture-of-experts. Instead, these approaches treat the expert selection as a policy, and use reinforcement learning to train the policy. To avoid the difficulties of reinforcement learning, often a soft approximation to the hard selection decision is used during training [287, 311, 312, 313, 314, 315, 316, 317, 318].

The model we propose strikes a balance between the more "innate" and the more "learned" approaches to conditional computation. It is hard-wired to use surprisal to determine when to use the bigger network, but this measure of surprisal is learned in an unsupervised way. Unlike more innate approaches, our model makes few assumptions about the nature of the problem or domain: only that it is sensible to express the input as a sequence of observations, which is true of many data modalities, like audio, text, and

video. Unlike more learned approaches, our model is more suitable for scenarios where there is very limited labeled training data (as long as there is sufficient unlabeled data available to train the autoregressive model).

Surprisal-based models and autoregressive models Surprisal is a useful notion for many tasks, such as anomaly detection [319] and comparing the difficulty of modeling different languages [320]. Using surprisal as an input feature in a neural network model has been explored in the past: for instance, in [321], He He *et al*. generate puns[2] using a model with surprisal-based features. [322] uses prediction error as an input to the model, though not for the purposes of conditional computation. In [323, 324], surprisal is used to determine whether to apply zoneout to units in LSTMs.

Autoregressive neural network models have not only been used to predict future observations or as generative models; they have also been used more recently with great success as unsupervised pre-trained feature extractors [214, 325, 326, 194, 327, 328, 219]. This seems to work well because accurately predicting the future necessitates extracting informative features from the past [329, 330]. Hence, we use the autoregressive model in our setup not only for measuring surprisal, but also to preprocess the input, which amortizes the additional cost incurred by running the autoregressive model.

Other related ideas Our model is similar to certain techniques in data compression and signal processing. In arithmetic coding, less effort is allocated to less surprising inputs, in the sense that shorter bitstrings (less bandwidth) are assigned to more predictable symbols [331, 332]. In linear predictive coding, a linear filter is used to predict the next sample of the input signal from previous samples, and the filter coefficients and error signal are transmitted instead [333, 334, 221]. According to the predictive coding hypothesis in neuroscience, human brains communicate information in a similar way: not as the raw signals themselves but rather in the form of prediction error [335, 336].

The Neural Sequence Chunker model proposed in [337] is very similar to our model, in that the prediction error of an autoregressive model is used to control a subsequent model;

[2]Nominative determinism.

but in that work, the subsequent model simply does not process predictable inputs at all, so as to reduce the input sequence length, whereas here we assume that every input must result in a corresponding output, which is often the case in sequence modeling. Another idea related to our model is the Expert Gate [338], which uses the reconstruction error of a set of autoencoders to select an expert in a mixture-of-experts to use. In [339], a reinforcement learning agent uses both a habitual controller and a planning-based controller and arbitrates between them using state prediction error and reward prediction error. Similarly, the Variational Bandwidth Bottleneck of [340] uses a notion of channel capacity to determine whether to run an expensive model-based planner. Our model is somewhat simpler and more broadly applicable than these approaches; it can be used outside of reinforcement learning and makes no constraints on the exact nature of the big and small networks.

4.3 Model architecture

Here we describe how our model works in more detail. Overall, the input to the model is a sequence of observations $(x_1, x_2, x_3, \dots)$, and for each timestep t, the model produces an output y_t.

4.3.1 Autoregressive model

An autoregressive model (Sec. 2.2.1) expresses the joint distribution of a sequence $p(x_1, x_2, x_3, \dots)$ as the product of the conditional distributions of the elements of the sequence given the previous elements:

$$p(x_1, x_2, x_3, \dots) = \prod_t p(x_t | x_{t-1}, x_{t-2}, \dots), \tag{4.1}$$

where $p(x_t | x_{t-1}, x_{t-2}, \dots)$ can be estimated using a neural network [46, 341, 342].

The surprisal of an observation is defined as the negative log-likelihood of that observation under the distribution defined by the autoregressive model. For real-valued inputs, a reasonable choice for the distribution is an isotropic Gaussian with variance 1, in which

case surprisal is equivalent (minus a constant term) to the squared error between the model's prediction $\hat{x}_t$ and the actual observation x_t:

$$\text{Surprisal}(x_t) = -\log p(x_t|x_{t-1}, x_{t-2}, \dots) \approx \frac{1}{2}||x_t - \hat{x}_t||_2^2. \tag{4.2}$$

We use a neural network encoder to compute a feature vector $h_t = f(x_t, x_{t-1}, \dots)$ and a linear model to compute the prediction $\hat{x}_t$ of the observation x_t given the feature vector from the previous timestep h_{t-1}. In our experiments, we use RNNs to implement the encoder, but other causal neural network layers, like causal convolutions [142] and masked self-attention [119], could be used as well.

Because the autoregressive model is unsupervised, it could be trained either using the input data for the target task or on a larger source of unlabeled data in the same domain. For example, in our experiments, we train conditional computation models on the small TIMIT dataset, but we train the autoregressive part on the much larger LibriSpeech dataset.

It is natural to ask whether it is worthwhile to backpropagate through the entire model, including the autoregressive model, when training on the downstream task [224]. In the experiments for this chapter, however, we simply keep the weights of the autoregressive model frozen. The reason is that if its weights are trained along with the rest of the model, it may not remain autoregressive, in which case it will not accurately compute surprisal. It should be possible to jointly train the entire model for better performance by adding a term to the final loss function that encourages the autoregressive model to remain autoregressive, but then this term would require its own regularization strength, which would mean another hyperparameter to be tuned, adding further complexity and variability to our experiments.

4.3.2 Controller

The controller uses the surprisal of the current observation x_t to compute the probability of sampling the big network p_{big}:

$$p_{big} = \text{sigmoid}(w \cdot \text{Surprisal}(x_t) + b), \tag{4.3}$$

where w and b are scalars.

We may want the distribution of p_{big} to have a certain mean (to achieve a certain computational budget) and variance (to ensure that both networks have the chance to be sampled for any given input, instead of only greedily sampling one or the other). We can train the controller so that p_{big} has a specified mean μ and variance σ^2 by minimizing the following loss function with respect to the controller parameters w and b:

$$\mathcal{L}_{controller} = \frac{1}{2}(\hat{\mu} - \mu)^2 + \frac{1}{2}(\hat{\sigma}^2 - \sigma^2)^2, \tag{4.4}$$

where $\hat{\mu}$ and $\hat{\sigma}^2$ are the sample mean and variance of p_{big}. Alternately, one could specify a target budget in terms of FLOPs, and set w and b so that the resulting expected amount of computation is equal to this budget. Like the autoregressive encoder, training the controller can be done either using the target dataset/environment or using a separate stream of unlabelled data.

4.3.3 Big and small networks

We use simple fully-connected neural networks to implement the big and small networks. An interesting aspect of the model is that if gradients are not backpropagated into the autoregressive model, non-gradient-based learners like decision trees could easily be used here, similar to the way that evolutionary algorithms are used in conjunction with neural world models in [327].

One caveat for the big and small networks is that it may not be straightforward to implement them using stateful models. For example, if we were to use RNNs as the big and small networks, where the state vector for the small RNN is not the same size as the state vector for the big RNN, it would not be possible to switch between these networks without introducing some additional machinery. A workaround that makes it easier to use stateful models is described in the next subsection. In the future, it could be interesting to find ways to overcome this limitation, possibly using models like Neural ODEs [343] that can maintain state across arbitrary timespans.

4.3.4 Pre-net and post-net

It is optionally possible to sandwich the conditional part of the model between a non-conditional "pre-net" and "post-net", as was done in the Sparsely Gated Mixture-of-Experts of [313]. In this case, the output of the autoregressive model h_t is instead fed to the pre-net, and the output y_t is taken from the post-net. This could be used to more easily add state, for example, since the pre-net and post-net can be implemented using RNNs.

4.4 Experiments

There are a number of questions one might ask about our model. How much computation does it save? Is surprisal actually a good heuristic for effort allocation? In other words, do we get better results using surprisal-triggered sampling than if we were to learn a controller or to just sample the big or small networks at random? Is our decision to use the features computed by the autoregressive model instead of the original inputs justified? How robust are the results to the choice of hyperparameters? We ran experiments to answer each of these questions.

4.4.1 Datasets

We use two small speech recognition datasets for our experiments[3]: TIMIT and Mini-LibriSpeech. TIMIT [344] is a 3-hour dataset with hand-aligned phoneme labels. Mini-Librispeech[4] is a 5-hour subset of the 960-hour LibriSpeech dataset [264] with transcripts but no phoneme labels; we used the Montreal Forced Aligner [345] to obtain label sequences from the transcripts.

[3] Our PyTorch [227] experiment code can be found online at `https://github.com/lorenlugosch/conditional-computation-using-surprisal`.

[4] `https://www.openslr.org/31/`

4.4.2 Setup

The small network in these experiments is a fully connected leaky ReLU layer with 512 hidden units. The big network is a fully connected layer with 2048 hidden units followed by another fully connected layer with 512 hidden units. The pre-net is a bidirectional GRU [346] with 256 hidden units in each direction and 50% dropout [347]. The post-net uses the same architecture as the pre-net, followed by a fully connected layer with $(n + 1)$ outputs, where n is the number of phonemes (39 for TIMIT, 41 for Mini-LibriSpeech) and 1 is for the CTC "blank" symbol.

The autoregressive model has two unidirectional GRU layers with 512 hidden units and 50% dropout, each followed by a fully connected layer with 512 hidden units. The inputs to the autoregressive model are sequences of 80-dimensional filterbank frames extracted using a 25 ms Hamming window every 10 ms from the 16,000 Hz audio signal. We skip every second frame [348], since this made our experiments much faster to run at a small cost in accuracy. We train the autoregressive model with maximum likelihood estimation on the full 960 hours of LibriSpeech.

The controller is trained for TIMIT to have mean 0.5 and variance 0.04 for p_{big}, which is the mean and variance that result from a standard normal distribution for the input to the sigmoid. Having this distribution ensures that p_{big} does not saturate at 0 or 1 and the range in between 0 and 1 is covered, so a given input observation is not always presented to only the big network or only the small network. For Mini-LibriSpeech, we instead train the controller to have mean 0.65 for p_{big} to bias it more towards using the big network, since its validation set has much lower surprisal. We use one pass of SGD through the training set of the task of interest to train the controller.

The number of parameters (which for this architecture happens to translate exactly to the number of FLOPs[5] performed by the network for each input timestep) for each part of the model is shown in Table 4.1. Each model is trained using CTC [155] for 50 epochs. The validation performance is measured at the end of every epoch or every 5 epochs (we do

[5]A FLOP can be defined in different ways. We use the convention that one multiply-accumulate = one FLOP. (FLOP count does not always precisely correspond to an actual useful metric, like latency or power consumption, but we leave more realistic evaluations like these for the future.)

Table 4.1: Parameter counts for model used in main experiments.

Component	Number of parameters
Autoregressive model	3.05M
Pre-net	1.18M
Small network	0.26M
Big network	2.10M
Post-net	1.20M
Controller	2

this for Mini-LibriSpeech to speed up our experiments, since decoding its validation set is much slower than training for one epoch). The model checkpoint with the best validation phoneme error rate over the course of training is used for the test set [107]. The test set is decoded using a beam search of width 10. Each model is trained with 5 random seeds, and we report the mean and standard deviation of results over the 5 trials.

4.4.3 Comparison with existing conditional models

We first compare our model with existing techniques for conditional computation. Early exiting is not applicable here because our experiments use CTC models, not classifiers, so it is not possible to implement the misclassification predictor; likewise, the score margin technique is not applicable because there are more intermediate stages between the big and small networks and the softmax output. We therefore compare with a simple baseline model that is identical in every respect, except that a learned controller is used instead of using surprisal. The learned controller is a feedforward gating network which takes as input h_t and outputs s_t, a binary decision to select the big network or small network. The network has a single hidden layer with 80 hidden units so that this controller is roughly the same size as the linear model used in the autoregressive model to estimate $\hat{x}_t$ (which is 80-dimensional) from h_{t-1}. We use the straight-through estimator [287]: in other words, we treat the threshold function used to compute s_t as the identity function during back-propagation so that the controller receives a non-zero gradient. The overall loss function

Table 4.2: Results for TIMIT and Mini-LibriSpeech. (For all tables, rows in grey are Pareto-optimal.)

Model	TIMIT		Mini-LibriSpeech	
	PER	Avg. FLOPs per input	PER	Avg. FLOPs per input
Random controller	20.52% ± 0.28%	6.63M	26.16% ± 0.35%	6.62M
Learned controller	**19.91%** ± 0.58%	6.63M	25.93% ± 0.32%	6.50M
Surprisal-based controller	20.00% ± 0.15%	**6.41M**	25.80% ± 0.10%	**6.43M**
Big network only	20.09% ± 0.24%	7.54M	**25.69%** ± 0.30%	7.54M
Mid-sized network	20.29% ± 0.44%	6.62M	25.89% ± 0.36%	6.62M

used for training this model is:

$$\mathcal{L} = \mathcal{L}_{CTC} + \lambda \cdot \sum_t (s_t - 0.5)^2, \tag{4.5}$$

where the second term encourages the model to use the big network roughly half of the time, and $\lambda = 0.001$ (chosen using a grid search in $\{0.1, 0.01, 0.001, 0.0001, 0.00001\}$).

The results of the experiment—the phoneme error rate (PER) and average FLOPs per input timestep for the test set of TIMIT and Mini-LibriSpeech—are shown in Table 4.2. The learned controller and the surprisal-based controller have similar performance; however, the surprisal-based model has lower variance in PER and a lower cost in FLOPs for both datasets. The performance of the learned controller model also seems to be sensitive to the more opaque hyperparameter λ: when λ is set to 0.0001 instead of 0.001, its PER and FLOPs for TIMIT increase to 23.97% and 6.99M, respectively (see Sec. 4.8). In contrast, the μ and σ^2 target hyperparameters for the surprisal-based controller are easy to interpret and do not need an expensive grid search to be set. This is an important consideration because when training extremely large models—where conditional computation may prove especially useful—there is often only enough budget for a small number of training runs [128].

Table 4.3: Results of ablation study for TIMIT.

Autoregressive features?	Surprisal-based during training?	Surprisal-based during testing?	PER	Avg. FLOPs per input
✗	✗	✗	22.81% ± 0.27%	**2.91M**
✗	✗	✓	22.49% ± 0.28%	5.75M
✗	✓	✗	22.73% ± 0.44%	**2.91M**
✗	✓	✓	22.58% ± 0.05%	5.75M
✓	✗	✗	20.52% ± 0.28%	6.63M
✓	✗	✓	20.52% ± 0.17%	6.41M
✓	✓	✗	20.28% ± 0.17%	6.63M
✓	✓	✓	**20.00%** ± 0.15%	6.41M
Small network only			20.61% ± 0.24%	5.70M
Big network only			20.09% ± 0.24%	7.54M

4.4.4 Ablation study

Our model essentially makes three independent design choices: it uses features computed by the autoregressive model instead of the original inputs, it samples the big network according to surprisal (as opposed to just randomly with probability 0.5) during training, and it samples the big network according to surprisal during testing. We trained models where each of these choices is ablated and report the results, as well as the baseline results when just the small network or just the big network is used. The results[6] of the ablation study with TIMIT are shown in Table 4.3.

First, we find that across all experiments, using the features computed by the autoregressive model results in significantly lower PER than using the original input features. This is perhaps not surprising, but it does independently confirm the efficacy of filterbank-based autoregressive models as pre-trained feature extractors for speech recently proposed in [328]. For this reason, we use the autoregressive features in subsequent experiments and when using the big network only or the small network only (the last two rows of Tables 4.3 and 4.4).

Next, we find that surprisal-based sampling yields Pareto-optimal results, with perfor-

[6]Note that the FLOP counts are lower for the models that do not use autoregressive features because the input-to-hidden weight matrices in the pre-net have input dimension 80 (the dimension of the filterbank features) instead of 512 (the dimension of the autoregressive features). Also, the FLOP counts for the first and third rows are lower by 3.05M because for them the autoregressive model does not need to be run at all during test time.

Table 4.4: Results of ablation study for Mini-LibriSpeech.

Surprisal-based during training?	Surprisal-based during testing?	PER	Avg. FLOPs per input
✗	✗	26.16% ± 0.35%	6.62M
✗	✓	26.04% ± 0.26%	6.43M
✓	✗	26.44% ± 0.06%	6.62M
✓	✓	25.80% ± 0.10%	6.43M
Small network only		26.31% ± 0.33%	**5.70M**
Big network only		**25.69%** ± 0.30%	7.54M

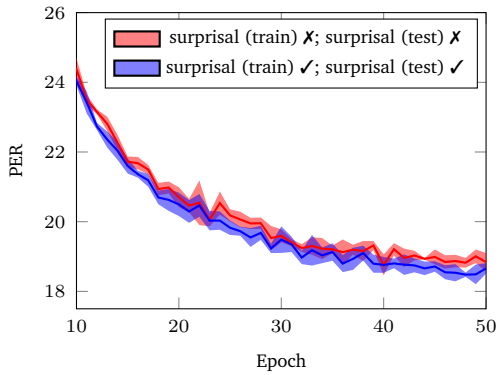
mance closer to or even slightly outperforming the models that only use the big network compared to the models that do not use surprisal. Similar results are obtained for Mini-LibriSpeech (Table 4.4) and when using other neural network architectures (Sec. 4.5). Also, if only the models not using autoregressive features are considered (the first four rows of Table 4.3), the model with surprisal-based sampling during both training and testing is still Pareto-optimal.

Fig. 4.2 shows the validation PER of models with and without surprisal-based sampling at test time over the course of training: the models with surprisal-based sampling during training and testing consistently outperform those without over time, despite making slightly less use of the big model (due to the discrepancy between the surprisal levels for the training set and for the test set).

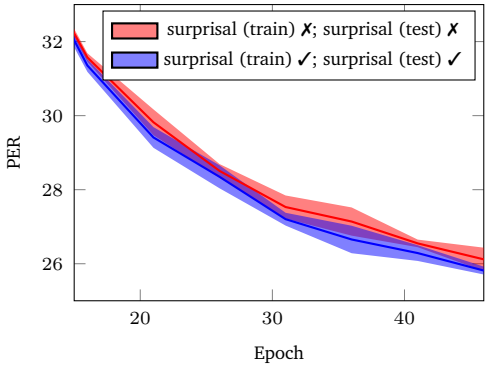
Whether it is crucial that surprisal is used during training is less clear. When surprisal is not used during training, the models that use it during testing do perform slightly better than those that do not for Mini-LibriSpeech (PER of 26.16% vs. 26.04%) and TIMIT when not using autoregressive features (22.81% vs. 22.49%), but not for TIMIT when using autoregressive features (both 20.52%). Section 4.6 further explores the effect of a mismatch between train time and test time.

4.5 Effect of varying hyperparameters

We re-ran the ablation experiment on TIMIT using three other neural architectures different from the one used in our main experiments, listed here in more concise PyTorch-like



(a) Validation PER for TIMIT.



(b) Validation PER for Mini-LibriSpeech.

Figure 4.2: Validation PER over the course of training for models with and without surprisal. Similar curves are obtained with a mismatch between train and test (not shown here for clarity of presentation).

notation. For these models, the controller trained to have mean 0.65 for p_{big}, instead of 0.5. The results show that using surprisal for effort allocation at test time improves PER across different hyperparameter settings, and that the pre-net, post-net, and recurrent layers are not essential for the idea to work. Using different hyperparameters for the autoregressive model would also be worthwhile to explore, but as training it is time-consuming, we have restricted the autoregressive model to a single configuration.

Table 4.5: Model 1 results for TIMIT.

Surprisal-based during training?	Surprisal-based during testing?	PER	Avg. FLOPs per input
✗	✗	26.94% ± 0.48%	6.52M
✗	✓	26.89% ± 0.20%	6.49M
✓	✗	28.86% ± 0.31%	6.52M
✓	✓	27.51% ± 0.39%	6.49M
Small network only		27.72% ± 0.53%	**5.96M**
Big network only		**26.75%** ± 0.13%	7.07M

Model 1:

Pre-net:

```
Conv1D(length 11, 512 filters)
LeakyReLU(0.125)
```

Small network:

```
Linear(512, 40)
LogSoftmax()
```

Big network:

```
Linear(512, 2048)
LeakyReLU(0.125)
Linear(2048, 40)
LogSoftmax()
```

Post-net: (none)

Table 4.6: Model 2 results for TIMIT.

Surprisal-based during training?	Surprisal-based during testing?	PER	Avg. FLOPs per input
✗	✗	26.76% ± 0.11%	6.52M
✗	✓	26.30% ± 0.36%	6.49M
✓	✗	28.16% ± 0.27%	6.52M
✓	✓	26.54% ± 0.32%	6.49M
Small network only		27.60% ± 0.60%	**5.96M**
Big network only		**25.91%** ± 0.41%	7.07M

Model 2:

Model 2 is the same as Model 1, except we add `Dropout(0.5)` to the end of the pre-net. (We found that the variants of Model 1 trained with surprisal easily overfit, which we believe happens because it is easier for the big and small networks to model either only surprising inputs or only unsurprising inputs separately than a mixture of both types.)

Model 3:

Pre-net: (none)

Small network:

```
Dropout(0.5),
Conv1D(length 11, 40 filters)
LogSoftmax()
```

Table 4.7: Model 3 results for TIMIT.

Surprisal-based during training?	Surprisal-based during testing?	PER	Avg. FLOPs per input
✗	✗	32.30% ± 0.66%	4.62M
✗	✓	31.14% ± 0.54%	4.54M
✓	✗	34.07% ± 0.58%	4.62M
✓	✓	30.37% ± 0.31%	4.54M
Small network only		35.47% ± 0.68%	**3.28M**
Big network only		**28.04%** ± 0.55%	5.96M

Big network:

```
Dropout(0.5),
Conv1D(length 11, 512 filters)
LeakyReLU(0.125)
Linear(512, 40)
LogSoftmax()
```

Post-net: (none)

4.6 Effect of varying controller bias

To investigate the effect of a mismatch between train time and test time further, we ran an experiment with Mini-LibriSpeech in which the bias parameter of the controller of models trained with surprisal is gradually increased from (average bias value learned when training the controller) to (that value + 4) in uniform increments. This increases the probability of using the big network from what it was during training.

The resulting sweep of mean test PERs with standard deviations is shown in Figure 4.3. Using the big network more does decrease PER up to a point, but when the bias is increased to the point where the big network is always sampled, the conditional model does not attain the same PER as the model that only ever uses the big network during training and testing. In fact, PER begins to increase, which may be because the big network is exposed to a distribution of more unsurprising inputs than it was shown during training.

In general, though, always using the big network may not be expected to yield the

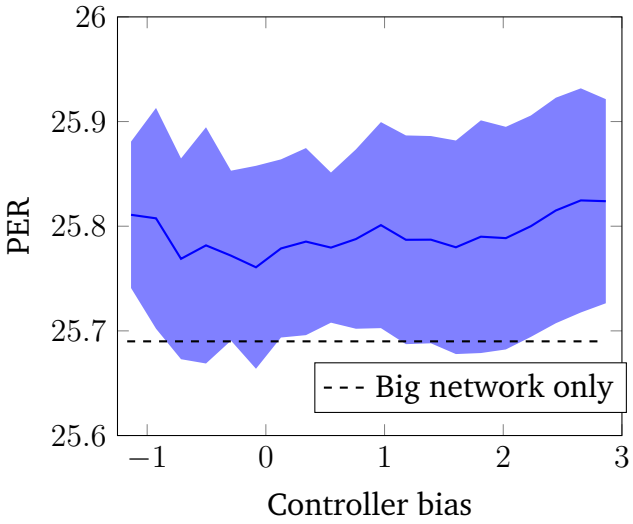


Figure 4.3: Test PER for Mini-LibriSpeech when increasing the controller bias.

best performance; using both the big and small networks at different times may have an ensemble effect, which may explain how the conditional models in our main experiments were able to slightly outperform the model using only the big network for TIMIT.

4.7 Effect of deterministic execution

In our main experiments, the big network is selected stochastically with probability p_{big}, rather than deterministically when p_{big} is greater than 0.5. This makes the comparison with randomized controllers in the ablation study more appropriate and removes the possibility of certain unlucky edge cases—for example, if p_{big} is just barely less than 0.5 for the entirety of an input sequence, the big network would never be sampled in deterministic execution.

However, it is sometimes desirable for the execution of a model to be deterministic, e.g. to make software testing easier and to obviate the need for random number generation when implementing the model in on a resource-limited device. We therefore also report the test results when the big network is selected deterministically in Tables 4.8 and 4.9. It appears that when training stochastically, deterministic execution at test time causes a small increase in PER, though this is not reflected in the test loss. Also, it appears crucial to train stochastically, as otherwise overfitting occurs more easily, possibly because the big and small networks always see the same input observations.

Table 4.8: Results comparing deterministic and stochastic execution for Mini-LibriSpeech.

Stochastic during training?	Stochastic during testing?	Train loss	Test loss	Test PER	Avg. FLOPs per input
✗	✗	69.96 ± 0.41	62.80 ± 0.76	$26.45\% \pm 0.23\%$	6.43M
✗	✓	69.72 ± 0.43	63.69 ± 0.36	$26.48\% \pm 0.14\%$	6.43M
✓	✗	73.76 ± 0.31	62.15 ± 0.60	$25.98\% \pm 0.26\%$	6.43M
✓	✓	74.30 ± 0.52	62.18 ± 0.64	**25.80%** $\pm 0.10\%$	6.43M

Table 4.9: Results comparing deterministic and stochastic execution for TIMIT.

Stochastic during training?	Stochastic during testing?	Train loss	Test loss	Test PER	Avg. FLOPs per input
✗	✗	16.19 ± 0.14	24.61 ± 0.52	$21.00\% \pm 0.19\%$	6.41M
✗	✓	16.21 ± 0.08	25.11 ± 0.24	$20.81\% \pm 0.24\%$	6.41M
✓	✗	17.58 ± 0.20	23.88 ± 0.31	$20.08\% \pm 0.22\%$	6.41M
✓	✓	17.55 ± 0.26	24.04 ± 0.22	**20.00%** $\pm 0.15\%$	6.41M

(It is possible to train a model either stochastically or deterministically, and then test it both stochastically and deterministically, thus avoiding redundant training runs. The train loss is only different between the first and second rows and between the third and fourth rows because it did not occur to us at the time of this experiment to implement this optimization.)

4.8 More detail on the learned controller baseline

When comparing the performance of surprisal-based controllers with learned controllers in Section 4.4.3, we noted that the learned controllers' performance is sensitive to the hyper-parameter λ used in the loss function. This observation is illustrated for Mini-LibriSpeech in Figure 4.4.

Of course, this result should be taken with a grain of salt: the space of possible models for implementing the learned controller is vast, and there may be others that are more robust. Still, the fact that a gating network using the straight-through estimator—the simplest and best-performing approach to learned conditional computation considered in [287]—behaves this way suggests that other learned conditional computation methods,

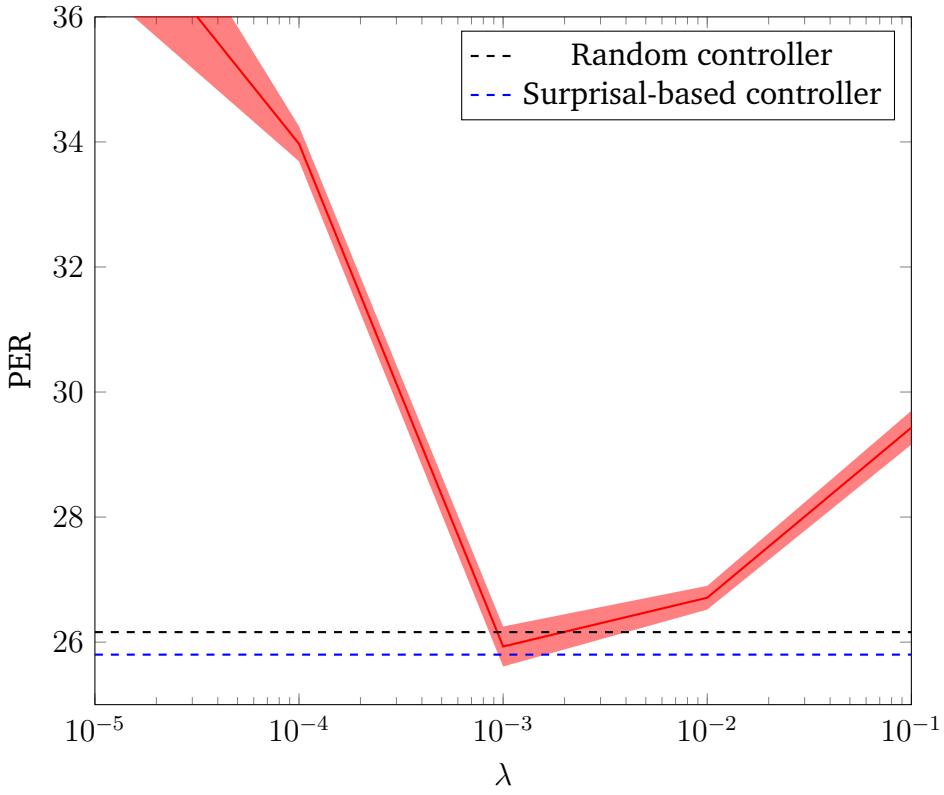


Figure 4.4: Test PER for Mini-LibriSpeech for models with a learned controller as a function of λ.

such as those based on reinforcement learning, may encounter similar difficulties.

4.9 Conclusion

We have shown that it is possible to use the surprisal of an autoregressive model to determine whether to use a big neural network or a small neural network for processing a stream of inputs, using the big network only for more difficult inputs and thereby reducing overall computation—in one instance reducing FLOP count by 15% at no cost in accuracy. We also find that while a baseline with a learned controller can achieve similar results, our model has lower variance and is less sensitive to hyperparameters. This suggests that the simple inductive bias of surprisal may make it easier to train much larger conditional models, where rounds of hyperparameter tuning are more expensive.

To return to the analogy with human cognition given in the introduction, a valid criticism of our model is that it does not take into account the expected reward resulting

from using more cognitive effort. Given a difficult task, a human decision maker might give more thought to the task if there is a big enough potential reward but might also decide not to waste energy if the reward is small [349]. In this work, we have effectively assumed a uniform potential benefit for using the big network, which turned out to be a reasonable assumption for the tasks we considered; in the future, we hope to compare our technique with a more data-driven evaluation of costs and benefits in a complete reinforcement learning setup. Also, here we have only run experiments in which there is single task to be performed; another interesting setting to study might be multi-tasking, in which an agent must allocate a limited computational budget among various tasks simultaneously [350].

Chapter 5

Using Speech Synthesis to Train End-To-End Spoken Language Understanding Models

Abstract

End-to-end models are an attractive new approach to spoken language understanding (SLU) in which the meaning of an utterance is inferred directly from the raw audio without employing the standard pipeline composed of a separately trained speech recognizer and natural language understanding module. The downside of end-to-end SLU is that in-domain speech data must be recorded to train the model. In this chapter, we propose a strategy for overcoming this requirement in which speech synthesis is used to generate a large synthetic training dataset from several artificial speakers. Experiments on two open-source SLU datasets confirm the effectiveness of our approach, both as a sole source of training data and as a form of data augmentation.

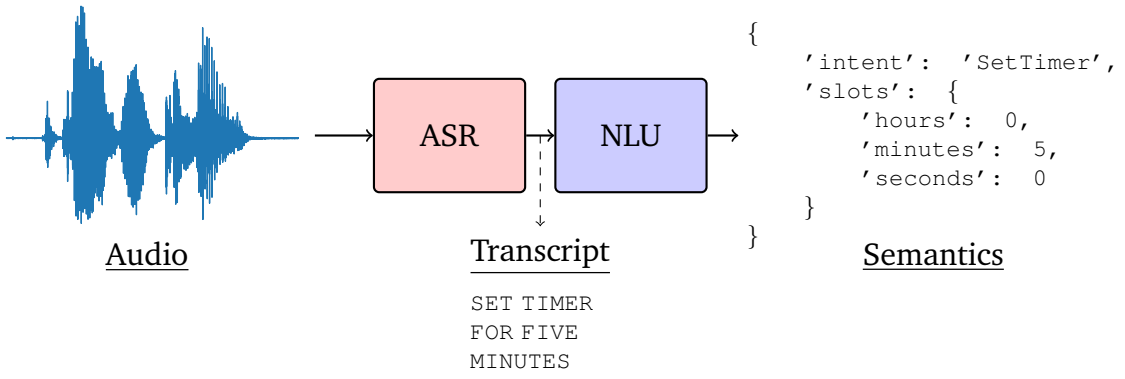


Figure 5.1: The conventional SLU pipeline, in which an automatic speech recognition (ASR) module transcribes the input speech, and a natural language understanding (NLU) module infers the semantics from the ASR transcript.

5.1 Spoken language understanding

We now turn from ASR to the next stage of the voice control pipeline: semantics. We give a brief overview of the state-of-the-art in spoken language understanding (SLU) before describing our contribution.

5.1.1 Decoupled SLU models

The designers of SLU systems have assumed from the beginning [351, 352, 353, 354] that SLU needs to be implemented using a decoupled ASR module and NLU module (Fig. 5.1). A good open-source example of a decoupled SLU system is the Snips Voice Platform (SVP), described in [355]. The SVP uses an hybrid ASR system, with a TDNN-LSTM neural network acoustic model and an n-gram language model trained on queries collected for the domain of interest. After extracting a number of handcrafted features from the ASR transcript, the NLU module uses a linear classifier to predict the intent, and, given the intent, selects the appropriate CRF to do slot filling using BILOU labels (similar to the IOB labels described in Section 2.2.13). The baseline system for the SLURP dataset (Chapter 7) is very similar but instead uses a BERT-based feature extractor to produce the input to the CRF.

5.1.2 End-to-end SLU models

Decoupled systems have the usual software engineering advantages of interpretability and separation of concerns. They were the natural choice for SLU at a time when it was (rightly) assumed that there were no machine learning models capable of learning the complex mapping required to implement the entire SLU pipeline from input to output. However, now that neural networks (and other models [356]) have become more powerful, researchers have begun experimenting with *end-to-end* SLU models that map the audio to semantics without an explicit intermediate text step. End-to-end models have a number of advantages over the conventional decoupled setup:

- Instead of two decoding steps — the ASR search algorithm and the NLU search algorithm — an end-to-end SLU model requires only one decoding step at the output. Removing the intermediate decoding step can significantly speed up inference and simplify the software implementation.

- Whereas ASR models are optimized for WER (which does not always correlate well with semantic accuracy [357]), end-to-end SLU models have all their parameters optimized directly for the actual end task of interest. (It is possible to backpropagate through the NLU module into the ASR parameters, but that requires backpropagating through a search algorithm, which is slow and possibly non-differentiable.) By optimizing for semantic accuracy instead of WER, the model can implicitly learn to give more priority to recognizing words that are more relevant to the SLU task, as opposed to less informative words like “the” and “please”.

- Operating directly on audio enables using information contained in the speech signal other than the transcript, such as prosody [358, 359, 360]. (Consider the difference in meaning between “*I* didn’t kill him” and “I didn’t kill *him*”.)

Similar considerations have been made for text: processing a document visually (as a human does) may work better than a two-stage pipeline with text decoding followed by interpretation [361].

An early instance of transcription-free SLU is the "self-taught vocal interface" described in [362, 363, 356, 364], which uses nonnegative matrix factorization (NMF) [365] to model the relationship between acoustics and semantics. In their system, the variable-length audio input is encoded into a fixed-length acoustic feature vector using a "bag-of-words"-like operation: a phoneme-level acoustic model (a GMM or neural network) computes the per-timestep probability of each phoneme, and $p(z_t = a) \cdot p(z_{t-\text{delay}} = b)$ is summed over time for each pair of phonemes (a, b), yielding a "soft count" of how many times a follows b in the utterance. The labels for the utterance are also encoded as a fixed-length vector, with a binary value for each semantic slot (e.g., activate, deactivate, lights, temperature control, kitchen, bedroom, ...). The acoustic feature vectors $v_a \in \mathbb{R}^{d_a} \geq 0$ and semantic feature vectors $v_s \in \mathbb{R}^{d_s} \geq 0$ for each training example are collected into matrices V_a and V_s, respectively, and a single data matrix $V \in \mathbb{R}^{(d_a+d_s)\times n}$ is formed by concatenating V_a and V_s, where n is the number of training examples. NMF attempts to find $W \in \mathbb{R}^{(d_a+d_s)\times k} \geq 0$ and $H \in \mathbb{R}^{k\times n} \geq 0$ such that

$$V \approx WH, \tag{5.1}$$

where $W(=\text{concat}(W_a, W_s))$ represents a set of k recurring latent patterns in the data, and H represents, for each training example, which patterns are "active". The semantics for a new utterance can be inferred by extracting the acoustic feature vector v_a, finding h such that

$$v_a \approx W_a h, \tag{5.2}$$

computing $v_s = W_s h$, and decoding the semantics from v_s. While this model avoids ASR decoding, inference is still somewhat slow because it requires solving a convex optimization problem using an iterative algorithm to find h.

Subsequent work has used neural networks instead [366]. The first work to demonstrate fully end-to-end SLU (intent classification) without any sort of ASR outputs or training targets was [367]. Their system uses a bidirectional LSTM to encode the audio, followed by max pooling over time to create a feature vector and a feedforward network to classify the utterance. While the end-to-end model underperformed the decoupled models

in [367], the end-to-end models were much smaller and faster, which has led to some interest in using end-to-end SLU models for embedded systems. [368] extended the end-to-end setup to predicting IOB-style slots using Transducer models. In [208], a number of different types of sequence-to-sequence models were trained for end-to-end SLU on a large production dataset, directly predicting the output dictionary as in Fig. 5.1, and were found to outperform decoupled systems.

5.1.3 Transfer learning

An end-to-end SLU model must implicitly learn to do both ASR and NLU, so the learning problem is potentially more difficult than for decoupled ASR and NLU models — the model does not even know to begin with that the same word spoken by two different people has the same meaning, whereas a decoupled system has this invariance baked in because the transcript is the same for both speakers.

To make training easier, a number of pre-training strategies for SLU models have been proposed. [369] use an auto-encoding task to initialize the model, training a decoder RNN to reconstruct the input audio from the final hidden state of an encoder RNN. [370] pre-train the first stage of the SLU model on grapheme ASR targets. In [371], both word targets and phoneme targets (which help with learning word targets) are extracted using Viterbi forced alignment [345] and used to pre-train the encoder[1] — in fact, three papers describing similar ASR-based pre-training methods appeared at the same conference [372, 373, 374]. In addition to ASR pre-training, BERT-style NLU pre-training has also been explored, solving a Cloze task on text and/or speech [375, 376, 377, 378, 379].

5.2 The problem: labeled audio data

Because the input to an end-to-end model is speech and not text, end-to-end models cannot learn directly from text data. This means that new audio data must be recorded to train the

[1]The original motivation for using word-level outputs in [371] was to be able to use word2vec embeddings as additional targets, thereby helping the model to generalize to synonymous words not encountered in the SLU training audio. But there was not enough time to implement this before the conference paper submission deadline. (Personal communication with the first author of [371].)

model for every new SLU domain or application. In contrast, the conventional ASR-NLU pipeline can be trained just once on a generic speech corpus to learn the mapping from speech to text, and subsequently only on text data. Thus, end-to-end SLU can be more difficult to implement in practice than conventional SLU because audio is more expensive and time-consuming to obtain than text data.

We propose a method for reducing, or avoiding entirely, the need to record audio data to train an end-to-end SLU model. Given a dataset of semantically labeled text data, we use a generic speech synthesizer, or text-to-speech (TTS), to read out these texts, thus generating an audio dataset that can be used for training the model. The ability to use synthetic data greatly lowers the barrier to entry for people who want to develop an SLU model for a new application: even if the accuracy of a model trained on synthetic speech is not satisfactory for end users, it may be good enough to allow fast prototyping of voice interfaces without waiting on the slow, expensive process of recording real speakers. Our method is useful not only when no real data is available: it also acts as data augmentation by exposing the model to more speaking styles and more ways of pronouncing the same phrases.

Our main contributions in this chapter are as follows:[2]

- We show that it is possible to train an end-to-end SLU model using only synthetic speech and achieve high accuracy on a test set of real speech.
- We run experiments using synthetic speech to augment an existing dataset of real speech and show that this augmentation can significantly improve accuracy, especially when few real speakers are available.

5.3 Related work

Our method is closely related to the idea of using speech synthesis to generate training data for end-to-end ASR [380, 381]. In end-to-end ASR, instead of using a separate acoustic model, language model, and pronunciation model, a single sequence-to-sequence model

[2]The PyTorch code for our experiments is available online at `https://github.com/lorenlugosch/end-to-end-SLU`.

predicts the transcript from the audio [28]. Because the language model in end-to-end ASR is only implicit and not decoupled from the rest of the model, it is difficult to train on standalone text data, so it does not easily handle certain types of utterances that are not well represented in the training audio, such as numeric sequences [382, 383]. To help the model recognize these domain-specific types of utterances, they can be synthesized and added to the training set.

Outside of speech recognition, backtranslation (or back-training [384]) is another technique in a similar vein often used for data augmentation in machine translation [385, 386]. In backtranslation, given three languages A, B, and C and paired data for (A, B) and (B, C), synthetic paired data for (A, C) is generated by translating the B text in (B, C) data into language A using a model trained on (A, B) data, and vice versa. If we think of the three modalities of audio, text, and semantics as three "languages", then our proposed technique is just backtranslation from semantically labeled text into audio. The back-training method proposed here is also akin to the self-training method in Chapter 3: instead of synthesizing $\hat{\mathbf{y}}$ from $\mathbf{x}$, we now synthesize $\hat{\mathbf{x}}$ from (a related) $\mathbf{y}$.

Another related idea is "sim2real" transfer in robotics [387]. In sim2real transfer, a policy is learned in a simulated environment, avoiding the risks involved in physically operating a robot, such as breaking the robot or harming humans in the environment. The speed of simulation can also give the robot more experiences than would be possible in a limited amount of time in the real world. Likewise, fast speech synthesis can allow generating more audio than would be possible with a human speaker, due to time constraints or fatigue for the speaker.

5.4 Proposed method

The method proposed in this chapter is simple. Two ingredients are required: 1) a text dataset, where each example consists of a transcript (e.g., "turn it up a couple notches") and corresponding semantic label (e.g., `{"intent": "ChangeVolume", "slots": [{"action": "increase"}, {"amount": "two"}]}`), and 2) a TTS for the language in which the transcripts are written. The TTS is used to synthesize each transcript.

The label assigned to the synthesized audio is the label for the transcript used to synthesize the audio. If the TTS has multiple speakers, each speaker is used to synthesize the transcript, so that multiple training examples per transcript are generated. A subset of the available speakers can be used for a given transcript if it is too expensive to use all speakers. If spoken training examples from real speakers are available, the real and synthetic datasets can be concatenated to form a single larger dataset. An end-to-end SLU model can then be trained using the generated dataset.

We have identified three criteria that are important for choosing the TTS:

1. *Multi-speaker:* In the past, we have found that having multiple speakers in the training set is crucial to achieving high test accuracy in end-to-end SLU. We anticpated that this would also be the case when using synthetic speakers.

2. *"Everyday" voices:* Commercial TTS voices typically speak in refined "actor speech", which is pleasant for the listener. But this type of speech sounds very different from the casual speech in which most people naturally speak to voice interfaces. To avoid this mismatch, casual, everyday voices should be used to synthesize training data.

3. *Open-source:* Like most researchers, we have a limited budget and want to perform research that is easy to reproduce, so we avoid commercial services like Google's Cloud TTS.

For our experiments, the TTS that best met these criteria was Facebook's VoiceLoop [388]. We used the pre-trained US English model included with the VoiceLoop repo, which has 22 synthetic speakers trained using the VCTK dataset [389]. We have listened to some of the synthesized audios selected at random and found the VoiceLoop speech to sound fairly natural. However, the synthesized speech does have some flaws: it contains audible vocoder artifacts, punctuation is ignored, and in some instances the model did not correctly pronounce the input text. Despite these imperfections, the synthesized speech works quite well for training, as we will show.

5.5 Experiments

To test our method, we run a number of experiments on two open-source SLU datasets.

5.5.1 Datasets

For the main set of experiments, we use the Fluent Speech Commands dataset [371]. Fluent Speech Commands is a dataset of 30,043 English audios with 77 speakers, each labeled with "action", "object", and "location" slots. There are 248 distinct sentences, each spoken by multiple speakers in both the training set and validation/test sets.

We also use the Snips SLU Dataset [390], more specifically the "smart lights" near-field subset of the dataset. This dataset is smaller and more challenging than Fluent Speech Commands: it contains numbers and has only 1,660 audios, each corresponding to a different sentence, so the model is tested entirely on sentences it has never heard before and must generalize to them to achieve high accuracy. Also, the number of slots varies across sentences: for example, the sentence "Could you turn the lights on please?" has the label {"intent": "SwitchLightOn", "slots": []} with no slots, but the sentence "Turn the flat light to twelve" has the label {"intent": "SetLightBrightness", "slots": [{"entity": "house_room_unique", "slot_name": "room", "text": "flat"}, {"entity": snips/number", "slot_name": "brightness", "text": "twelve"}]} with two slots. The dataset is intended to be split into five folds for cross-validation and has multiple speakers, but the splits and speaker identities are not included in the dataset.

5.5.2 Models

We use encoder-decoder models in our experiments. The encoder is a deep neural network with multiple convolutional layers and recurrent layers, with max-pooling in some layers to reduce the sequence length. The encoder is pre-trained using the LibriSpeech ASR dataset [391], and the encoder parameters are unfrozen over the course of SLU training; more details on how the pre-training and unfreezing are done are given in [371]. The decoder

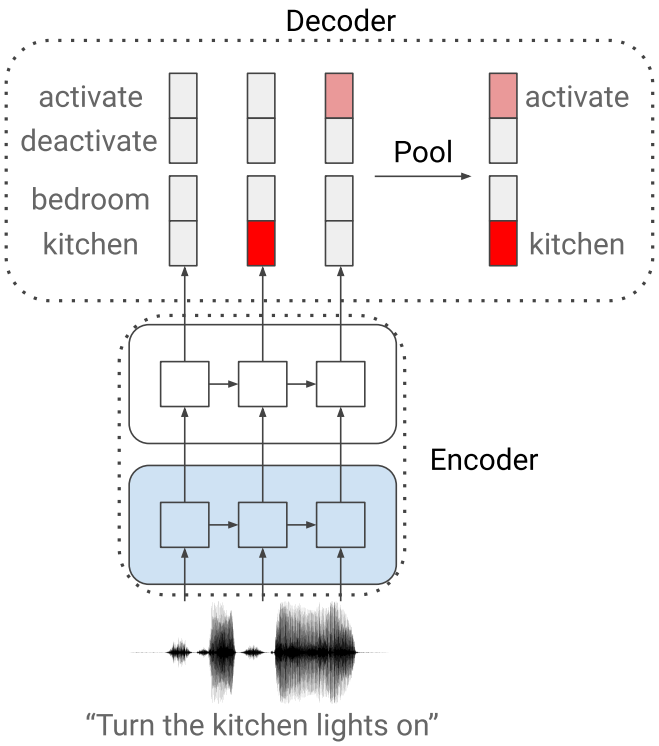


Figure 5.2: Model with max-pooling decoder. The portion of the model shaded in blue is pre-trained using an ASR task.

for Fluent Speech Commands is a linear classifier applied to the output of the encoder at each timestep separately, followed by global max-pooling to convert the variable-length sequence of vectors of slot scores into a single vector (Fig. 5.2). For simplicity, we use the same hyperparameters and transfer learning methodology as were used in the best performing model in [371] across all experiments.

For the Snips SLU Dataset, since the number of slots varies across utterances, it is not possible to use the simple max-pooling decoder with a fixed-length output. Instead, we use an attention-based autoregressive decoder [140], as was proposed for SLU in [208] (Fig. 5.3). The decoder uses two gated recurrent unit (GRU) layers of 256 hidden units each [116], with key-value attention [119], and sequentially predicts the semantic label string, character by character, using a beam search.[3] We trained autoregressive encoder-

[3]The thesis author's beam search implementation (`https://github.com/lorenlugosch/end-to-end-SLU/blob/e94bd479de1a82663c38363979309089acebcd36/models.py#L558`) incorrectly selects the top B extensions for each hypothesis before collecting all extended hypotheses, unlike the standard beam search described in Section 2.2.10. This means there are B^2 hypotheses instead of BL hypotheses before pruning. Effectively this just amounts to using a smaller beam width. *Mea culpa.*

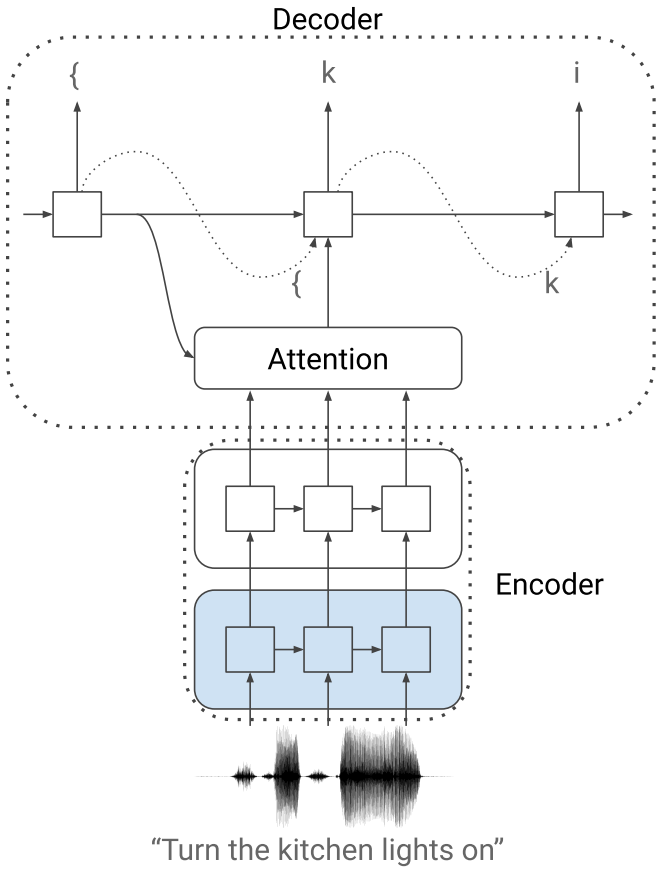


Figure 5.3: Model with autoregressive decoder used for the Snips SLU Dataset.

decoder models on Fluent Speech Commands and used the test accuracy to determine the hyperparameters used in the models for the Snips SLU Dataset.

5.5.3 Results for purely synthetic training sets

We first present results for models trained using only synthetic speakers. We used all 22 synthetic VoiceLoop speakers to synthesize all sentences in Fluent Speech Commands[4]. To quantify how many speakers are needed to achieve good accuracy, we train models using the data from one speaker, two speakers, and so on, and report the resulting accuracy. The accuracy is measured on the test set of real speakers in Fluent Speech Commands. Note that we only report test results on real speech, never on synthetic speech.

Not every speaker is equally high-quality or useful for training, so the randomly chosen subset of speakers can have a big impact on test accuracy, in addition to other sources of

[4]The synthesized dataset can be downloaded here: `https://zenodo.org/record/3509828`

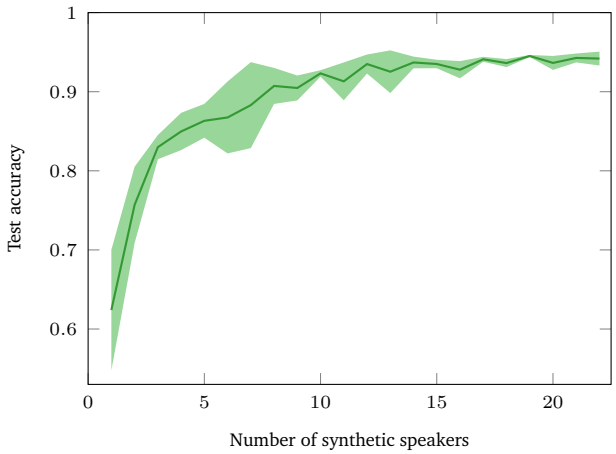


Figure 5.4: Test accuracy on Fluent Speech Commands as a function of the number of synthetic speakers.

stochasticity, like the initial model weights and the order in which training examples are presented. To reduce the variance of the results, we run each experiment five times using different random seeds, and record the mean and standard deviation (shown as shading in our plots).

Fig. 5.4 shows the test accuracy as a function of the number of speakers. The accuracy increases sharply up to about 15 speakers, and plateaus afterwards, with a very slight upward trend. The conclusion we draw is that one should use all available synthetic speakers if possible, but if synthesis is expensive, or if the resulting dataset is too large to train on exhaustively, it may make sense to incrementally add new synthetic speakers and stop when adding more speakers does not improve accuracy much. In subsequent experiments when using synthetic speakers, we use all 22 available synthetic speakers.

5.5.4 Results combining real and synthetic speech

We next present results for when the model is trained using real speech and augmented with synthetic speech. We simulate the scenario where only a few real speakers are available by selecting a random subset of speakers from the full training set. The experiments here take longer to run since there are more speakers, so we run each experiment just three times instead of five times.

Fig. 5.5 shows the results, presented alongside the accuracy when all 22 synthetic

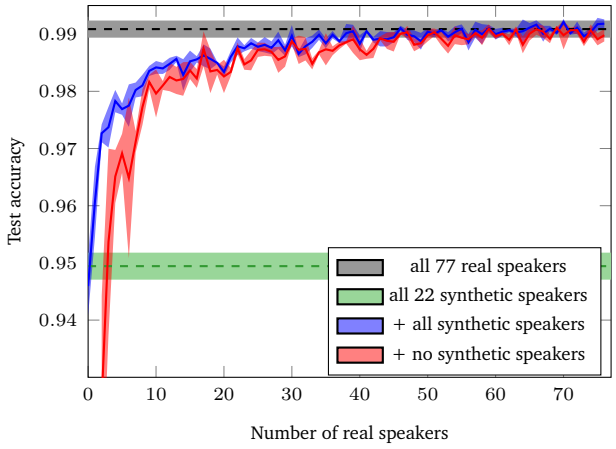


Figure 5.5: Test accuracy on Fluent Speech Commands as a function of the number of real speakers.

speakers are used (green bar on bottom) and the accuracy when all 77 real speakers are used (grey bar on top). Unsurprisingly, real speech is more useful than synthesized speech. The model trained using only real speech is about 4% more accurate than the model trained using only synthetic speech (99.1% $\pm$ 0.1% versus 94.9% $\pm$ 0.2%). Also, with only three real speakers and no synthetic speakers, the model already performs better than when using all 22 synthetic speakers.

Up to 40 real speakers, including the synthetic speakers in the training set results in better accuracy in 38 out of 40 cases. When using more than 40 real speakers, it is less clear from our experiments if including synthetic speakers is helpful. We measured the difference in accuracy across the number of real speakers with more than 40 real speakers; the accuracy was 0.07% higher on average when synthetic speakers were included. The difference is not significant, but it at least suggests that it is not harmful to include synthetic speakers even when a large number of real speakers is available.

Finally, we present results for the more challenging Snips SLU Dataset. Again, we synthesize each sentence using all 22 speakers, which boosts the size of the training set for each fold from 1,328 audios to 30,544 audios. The autoregressive model requires many more SGD updates to fit a dataset than the simpler max-pooling model, so we upsample the real-only dataset so that an equivalent number of updates are taken each epoch for that dataset as for the dataset with synthetic speakers. The model is able to overfit the dataset without synthetic speakers; we therefore record the best test accuracy achieved

Table 5.1: Cross-validation results for Snips SLU Dataset.

Data type	**Best accuracy**	**Best loss**
Real	$65.5\% \pm 2.9\%$	2.81 ± 0.42
Real + synthetic	$71.4\% \pm 1.4\%$	1.67 ± 0.16

over the course of training for each fold, instead of the final test accuracy. To use another metric to confirm that this improvement is not just a fluke for this small dataset, we also record the best loss. Table 5.1 reports these results: both the best accuracy and best loss are significantly better when synthetic speakers are included.

5.6 Conclusion

In this chapter, we have shown that it is possible to use synthetic speech to train an end-to-end SLU model. Including synthesized speech in the training set improves accuracy across a variety of settings, in some cases by a large amount. Our results strongly suggest that practitioners should try our method to augment their datasets.

In the future, we hope to find ways to reduce the gap between the performance of a model trained solely on synthetic speech and a model trained on a comparable amount of real speech. Also, our method is limited to high-resource languages, like English, for which it is possible to train a high-quality TTS. While our method should become more useful for low-resource languages as more data becomes available, it would be interesting to find a way to avoid this restriction.

Chapter 6

Timers and Such: A Practical Benchmark for Spoken Language Understanding with Numbers

Abstract

This chapter introduces Timers and Such, a new open source dataset of spoken English commands for common voice control use cases involving numbers. We describe the gap in existing spoken language understanding datasets that Timers and Such fills, the design and creation of the dataset, and experiments with a number of ASR-based and end-to-end baseline models, the code for which has been made available as part of the SpeechBrain toolkit.

6.1 The need for Timers and Such

Spoken language understanding (SLU) research has begun to emphasize the importance of both *testing* and *training* SLU systems end-to-end on audio. *Testing* on audio is important because an independently trained automatic speech recognition (ASR) system and natural language understanding (NLU) system will not necessarily work well when combined [392, 390]. *Training* SLU systems end-to-end on audio is likewise worthwhile because it

can make the NLU model more robust to transcription errors [393], and because it enables training a single neural network to perform the entire SLU pipeline without an intermediate search step, a technique with advantages over ASR-based approaches, as we saw in Chapter 5.

Experiments involving end-to-end training and testing of SLU models require audio data. Over the last few years, a number of open source audio datasets have been released to enable high-quality, reproducible end-to-end SLU research. The **Snips SLU Dataset** [390] is a small dataset of English and French commands for a smart home setting, such as controlling smart lights, speaker volume, and music selection. **Fluent Speech Commands** [371] is a somewhat larger, though simpler, dataset of similar English smart home commands. The most recently released **SLURP** dataset [394] is an even larger and much more semantically complex multi-domain SLU dataset.

An important feature missing from these datasets is a thorough coverage of *numbers*. Numbers are necessary for many SLU domains, especially for very common use cases like setting timers and converting units of measurement while cooking. While there do exist datasets of digits spoken in isolation [395, 396, 397], and the Snips SLU Dataset and SLURP do have a small number of commands involving simple numbers, there does not to our knowledge exist any open source SLU dataset that covers more general multi-digit numbers (e.g. “13.57”, “-21.4”) spoken in context. The dataset introduced here—**Timers and Such**—fills this gap, with each command containing one or two numbers with one or more digits.

One of the original motivations for the development of end-to-end SLU models was the need for more compact models that can easily fit on resource-limited devices and operate without an Internet connection [367]. Whereas existing SLU datasets focus mostly on Internet-connected smart home commands or queries that require an Internet search, Timers and Such is composed only of commands that can be executed without the need for the Internet. This makes the dataset ideal for training or testing a simple offline voice assistant. While the baselines described in this chapter all use rather comfortably large neural networks ($>$100 million parameters), we hope that researchers and developers working on machine learning for edge devices will improve upon our models in terms of

storage requirements and computational complexity; we believe they will find Timers and Such to be a challenging and interesting test case for their models.

The dataset should also be useful for researchers working on representation learning for audio and language to use as a downstream test task, as Fluent Speech Commands has been [398, 399]. While in the past we have found supervised ASR-based pre-training to be essential for getting good results with end-to-end SLU models, we believe unsupervised feature extractors may ultimately prove to be a better general-purpose solution for SLU and other audio tasks [400, 7].

A final, more mundane motivation for Timers and Such was the need for an SLU dataset that could easily be downloaded programmatically using tools like `wget` or `curl`, similar to MNIST or LibriSpeech.[1] Fluent Speech Commands requires users to sign up on a web page, and the Snips SLU dataset requires filling in an online form and waiting to be approved. In contrast to these, Timers and Such is hosted on Zenodo[2] under the very permissive CC0 license, and the experiment code[3] we provide downloads the dataset if it is not already present in the location specified by the user. These features should lower the barrier to entry for anyone interested in training or testing their first SLU model.

In what follows, we outline the design and creation of Timers and Such, describe some baseline models for the dataset, discuss their experimental performance, and end by listing some ideas for future research.

6.2 Dataset design

The dataset has four intents, corresponding to four common offline voice assistant uses: `SetTimer`, `SetAlarm`, `SimpleMath`, and `UnitConversion`. The semantic label for each utterance is a dictionary with the intent and a number of slots. An example of a command and its corresponding semantics is shown in Listing 6.1.

[1] SLURP, released after the start of this work, can also be downloaded programmatically.

[2] The dataset can be found at `https://zenodo.org/record/4623772`.

[3] The code can be found at `https://github.com/speechbrain/speechbrain/tree/develop/recipes/timers-and-such`.

```
("what's 37.67 minus 75.7",
{
  'intent': 'SimpleMath',
  'slots': {
    'number1': 37.67,
    'number2': 75.7,
    'op': ' minus '
  }
})
```

Listing 6.1: A `SimpleMath` command and its label dictionary.

The prompts to be recorded by speakers were generated using a script written by the thesis author with a simple "grammar" that produced a few variations of set phrases for each of the four intents ("set a timer for...", "set timer for...", "start timer for..."). Random numbers were inserted from a range that made sense for the given intent (for instance, when converting temperatures, temperatures less than 0 Kelvin were not used).[4]

A better way to collect different ways of phrasing commands than introspection is to place speakers in a voice control scenario (or have them imagine themselves in one) and ask them what they would say to have the system complete a certain task. This method was used to create part of the closed source Facebook dataset in [367] and the open source SLURP [394]. However, this approach is complicated to set up and much more taxing on speakers. Given that our speakers were volunteers, we decided instead to simply prompt them with randomly generated phrases for each of the intents, similar to the approach used in Mozilla's Common Voice project [401].

6.3 Preliminary small-scale study

A preliminary version of Timers and Such was made between November 2019 and October 2020. 11 colleagues recorded themselves reading a list of prompts, some using the thesis author's laptop, and others using their own computers. The thesis author then segmented these audio files into the individual commands and split the resulting 271 audios into a training set with 144 audios (4 speakers), a dev set with 72 audios (2 speakers), and a test set with 55 audios (5 speakers). Models trained on this small dataset were found

[4]The script for generating prompts can be found at `https://gist.github.com/lorenlugosch/5df9e30227aa5c67ff51cd28271414f0`.

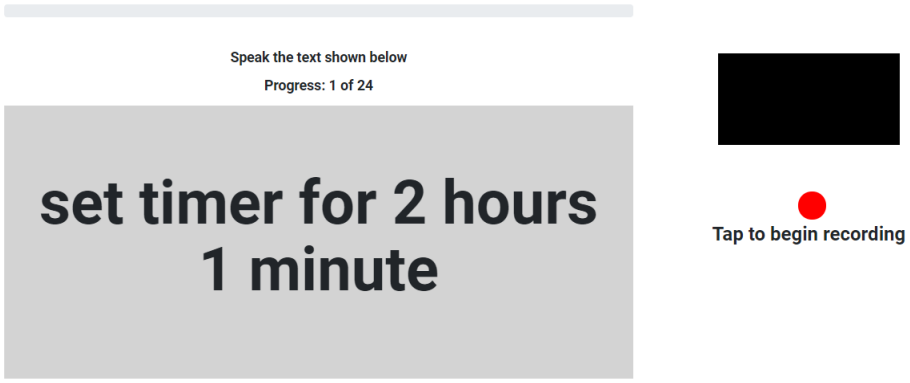


Figure 6.1: The recording interface used by speakers.

to have high variability in performance for the test set, which was hypothesized to be because of the small test set size. (This actually seems not to have been the real reason; see Sec. 6.5.4.) To make a dataset that could be used more reliably to train and compare SLU models, we decided to reach out to a larger pool of speakers by asking volunteers online to donate their voices.

6.4 Data collection

6.4.1 Recording website

The second author of the paper built a website to allow speakers to record themselves reading prompts. Speakers using the website were first asked for their age, gender, and spoken English proficiency. For each demographic field, users also had the option to respond "Prefer not to say". After giving their consent to have their demographic information and recordings released in a publicly available dataset, speakers used the interface shown in Fig. 6.1 to record a set of 24 randomly generated prompts.

6.4.2 Speaker recruitment

Starting on February 18, 2021, we advertised the project and recording website on various social media platforms (Twitter, LinkedIn, Reddit, Hacker News, Facebook). In response to

this advertisement, 89 sessions were recorded from the first day until March 12, 2021.

Whether the 89 recorded sessions correspond to exactly 89 different speakers is unknown. We neglected to ask speakers in the recording instructions not to record more than one session. Because speakers were (deliberately) not asked to provide any information that would uniquely identify them, such as their name or email address, there is no way to ascertain whether two sessions correspond to the same speaker (as is the case for recording platforms like Common Voice's, which allow a speaker to record without entering any personally identifiable information). To avoid an overlap between speakers in the training set and the test set, we examined the demographic information provided by speakers (age, gender, fluency) and selected only sessions with a unique demographic triple to be in the test set. Assuming speakers provided their demographic information truthfully, this means there are no speakers from the test set in the training set.

6.4.3 Data preprocessing and cleaning

All recordings were converted from their original formats to single-channel 16,000 Hz .wav files for compatibility with the acoustic model used in our baseline experiments.

Data cleaning for the smaller set of audios collected during the preliminary small-scale study was done manually by the thesis author. The 271 audios collected in the preliminary study were assigned to the `dev-real` subset. Those speakers were not asked for their demographic information, so that information is not provided for this split.

For the larger set of audios recorded using the recording website, we used a more automated form of cleaning: the audios were transcribed using an acoustic model (described in Sec. 6.5.1), and the word error rate (WER) between each prompt and transcript was computed. Audios for which the ASR transcript was empty or looked significantly different from the prompt were listened to and kept or deleted as appropriate. (A simple automatic decision rule that was found to yield nearly the same subset was to select all audios with WER less than 100%.) After this cleaning procedure, the remaining 1,880 audios were split into `train-real` and `test-real` subsets. A .csv file for each subset ({`train-real, dev-real, test-real`}`.csv`) lists, for each utterance, the .wav filename, the semantic

label dictionary, the session ID (≈ speaker ID), and the transcript.

6.4.4 Synthetic data

As in Chapter 5 [11], we used VoiceLoop [388] to synthesize a large set of audios from 22 synthetic speakers. (The VoiceLoop model is trained on the VCTK dataset [389].) That set was split by speaker into the `train-synth` (192,000 audios), `dev-synth` (24,000 audios), and `test-synth` (36,000 audios) subsets. As for the data from the real speakers, we include a .csv file ({`train-synth, dev-synth, test-synth`}`.csv`) listing the filename, semantics, speaker ID (a number 1 to 22 indicating which VoiceLoop synthetic speaker was used), and transcript. The VoiceLoop speech synthesizer is deterministic: running it on the same prompt twice produces the same audio signal. As a result, some of the rows in the .csv file describing the synthetic subset are redundant: they point to the same audio file with the same labels. We have not removed the redundant rows because we found that doing so led to an unbalanced training set: for example, there were many more instances of "set alarm for `<hour>` `<minute>` AM" than of "set alarm for `<hour>` AM", so models trained on this unbalanced dataset tended to hallucinate an erroneous value for the `<minute>` slot for the latter type of utterance. (Alternately, users can rebalance the data in a different way, if they choose, using e.g. `pandas.DataFrame.drop_duplicates()` on the filename column of the .csv file.) We encourage users of Timers and Such not to think of the synthetic subset as *fixed* (except to avoid unfair comparisons between two models differing in some other respects), but rather to try adding more synthetic speakers and using improved speech synthesis techniques.

6.4.5 Dataset statistics

The overall statistics for both the real and synthetic subsets of Timers and Such after data cleaning are listed in Table 6.1. At 2,151 non-synthetic utterances, Timers and Such is a fairly small dataset, but like TIMIT (6,300 utterances [59]) and the Snips "smart lights" dataset (1,660 utterances [390]), we have found the dataset nonetheless very useful for experimentation. It is more challenging than Fluent Speech Commands (which can be

Table 6.1: Timers and Such speaker counts and recording statistics. (*Speaker counts are approximate; see Section 6.4.2.)

Split	# of speakers*	# of audios	# hours
`train-synth`	16	192,000	132.2
`dev-synth`	2	24,000	15.8
`test-synth`	3	36,000	23.5
`train-real`	74	1,640	1.9
`dev-real`	11	271	0.3
`test-real`	10	240	0.3
`all-real`	95	2,151	2.5

Table 6.2: Speaker gender statistics. (`dev-real` demographics not included; see Section 6.4.3.)

Split	Man	Woman	Non-Binary	(Prefer not to say)
`train-real`	54	17	0	3
`test-real`	5	4	1	0

treated as a simple classification problem and for which accuracy as high as 99.7% has been achieved [378]), but it is smaller and simpler than SLURP. By training only on text or synthetic speech, and testing on all available real audio, it is possible to obtain a relatively large test set (cf. the LibriSpeech `test-clean` subset with 2,620 audios).

6.5 Baseline models

Here we describe extensive experiments with a set of baseline neural network models for Timers and Such. All experiments are conducted using the open source SpeechBrain [13] toolkit.

Table 6.3: Speaker English proficiency statistics.

Split	Native speaker	Fluent	Somewhat fluent	(Prefer not to say)
`train-real`	20	42	9	3
`test-real`	4	2	4	0

Table 6.4: Speaker age ranges. (See `train-demographics.csv` and `test-demographics.csv` for more granularity.)

Split	18-25	26-35	36-45	46+	(Prefer not to say)
`train-real`	11	41	6	1	15
`test-real`	3	5	2	0	0

6.5.1 Acoustic model and language models

The baseline ASR models use an acoustic model trained on the 960-hour LibriSpeech English ASR dataset [391]. The acoustic model is an autoregressive attention-based sequence-to-sequence model [140, 152] that achieves 3.08% WER on the `test-clean` subset of LibriSpeech. The encoder of the acoustic model is a convolutional recurrent deep neural network (CRDNN) that extracts 40-dimensional FBANK features from the input signal and has two 2-D convolutional layers that downsample the input sequence by a factor of 4 in the time dimension, followed by four bidirectional LSTM layers and two fully-connected layers. The decoder is a GRU network that uses the location-aware attention mechanism of [151] to process the encoder outputs. The encoder outputs are additionally passed through a linear CTC [155] head; during training, the output of the CTC head is used to compute an auxiliary CTC loss term [154]. Both the CTC head and the autoregressive decoder have 1000 outputs for a 1000-token SentencePiece [30] BPE vocabulary.[5] (This acoustic model was chosen because it was the best performing English acoustic model in SpeechBrain at the time when these experiments were conducted.)

The acoustic model transcribes the input signal $\mathbf{x}$ using a beam search for

$$\begin{aligned}\underset{\mathbf{y}}{\operatorname{argmax}} \quad & \log p_{\text{AM}}(\mathbf{y}|\mathbf{x}) \\ & + \alpha \log p_{\text{CTC}}(\mathbf{y}|\mathbf{x}) \\ & + \beta \log p_{\text{LM}}(\mathbf{y}) \\ & + \gamma c(\mathbf{x}, \mathbf{y}),\end{aligned}$$

[5] More detailed hyperparameters for the acoustic model can be found at `https://github.com/speechbrain/speechbrain/blob/develop/recipes/LibriSpeech/ASR/seq2seq/hparams/train_BPE_1000.yaml`.

where $p_{\text{CTC}}(\mathbf{y}|\mathbf{x})$ is the likelihood of transcript $\mathbf{y}$ according to the CTC head [154], $p_{\text{LM}}(\mathbf{y})$ is the likelihood according to an external language model (LM), $c(\mathbf{x}, \mathbf{y})$ is a coverage penalty term [402], and α, β, γ were set to minimize WER on the LibriSpeech dev sets.

The default LM is an LSTM trained on the LibriSpeech language modeling resources.[6] In addition to the default LibriSpeech LM (LS LM), we also trained an LSTM LM on the Timers and Such training set transcripts (TAS LM). For ASR-based baseline models, we present results both using the LS LM and TAS LM.

6.5.2 SLU models

We provide code, pre-trained models, and results for a traditional decoupled SLU model and (using the terminology suggested by Haghani et al. in [208]) two types of "end-to-end" models: a multistage model and a direct model.

The **decoupled** model uses a sequence-to-sequence model to map the transcript to the semantics. During training (and when decoding the validation set), the ground-truth transcripts are used as the input, and during testing, the transcripts produced by the LibriSpeech acoustic model are used. For all models, the semantic dictionaries are treated as raw sequences of characters and split using a 51-token SentencePiece tokenizer.

The **multistage** model likewise uses a sequence-to-sequence model to map the transcript to the semantics, but instead of training on the ground-truth transcripts, it is trained on the ASR transcripts. The transcripts are not precomputed: rather, each minibatch of audio signals is transcribed on the fly during training, which simplifies the implementation of our experiments. In theory, transcribing training examples on the fly should also make the NLU model more robust, as it is exposed to more types of transcription errors resulting from different noise samples (e.g. from dropout, batch normalization, data augmentation) across minibatches—though we have not compared the results with simply training on a single set of precomputed ASR transcripts, and leave this as an avenue for other researchers to explore. The downside of on-the-fly transcription is that the inherently sequential ASR beam search becomes a bottleneck on training step time. Using the

[6] `https://www.openslr.org/11/`

default ASR beam width of 80, the time for one epoch on `train-synth` was about 12 hours (compared with about 0.5 hours for the decoupled model). Reducing the ASR beam width to 1 reduced the time for one epoch to about 2.5 hours. The results presented below use an ASR beam width of 1 for the multistage model.

The **direct** model uses a single sequence-to-sequence model to map audio directly to semantics, without an intermediate ASR search step. Compared to the multistage model, the direct model is significantly faster both in training and decoding, at about 1.5 hours per epoch with `train-synth` instead of 2.5 hours. Pre-training using related ASR or NLU tasks has consistently been found to improve the performance of direct models [371, 375, 403, 404, 405], so we pre-train the encoder here as well. In our experiments described in previous papers, the encoder of the direct model was pre-trained using force-aligned phoneme and word labels [371, 11]. The pre-training strategy used in our baselines here is somewhat simpler: we extract the encoder from the LibriSpeech acoustic model and use it as a feature extractor in the direct SLU model. Another difference is that we do not backpropagate into the pre-trained encoder and leave its weights frozen, which greatly reduces training time and memory consumption. A more thorough ablation study and comparison of pre-training strategies would be worthwhile to conduct, but we leave that for the future, since the point here is just to establish some reasonable baseline models for this dataset.

While the SLU models do use a beam search to produce the output sequence, there are a number of differences between the SLU decoder and the ASR decoder. The SLU beam search does not use a coverage penalty (which was found to hurt performance both for Timers and Such and for the SLURP dataset) or an external "language model" over the space of output dictionaries. Instead of location-aware attention (which assumes a monotonic alignment between input and output sequences), the SLU decoder uses a simple one-headed key-value attention mechanism. The SLU models also do not use an auxiliary CTC head: whereas CTC's assumptions (monotonic alignments; output length $<$ input length) make sense for ASR, they generally do not hold for SLU, unless the dataset has word-aligned slot labels (Timers and Such does not). Other hyperparameters for these models were not optimized and chosen simply by copying the decoder hyperparameters

from the LibriSpeech recipe, which were optimized for the validation set of that dataset.

6.5.3 Experiments

For all baseline models, we provide results for three composite training sets: `train-real` only (trained for 50 epochs), `train-real` plus `train-synth` (trained for 2 epochs), and `train-synth` only (trained for 2 epochs). For all three training sets, we measure performance on `test-real` and `test-synth`. When training on `train-synth` only, we additionally report performance for `all-real`, a subset obtained by combining all the real data in `train-real`, `dev-real`, and `test-real`. (We do not test models trained on `train-real` on `all-real` because `all-real` contains `train-real`. For the same reason, we use `dev-synth`, not `dev-real`, to select the model checkpoint from the epoch with the best validation performance when testing on `all-real`.)

As in previous work, we report performance in terms of accuracy, where an output is deemed "correct" if all predicted slots and slot values are correct. Bastianelli et al. in [394] have argued for the use of metrics more informative than simple accuracy when evaluating end-to-end SLU models. They propose SLU-F1, a metric based on word-level and character-level edit distance between the model's output and the true labels. The SLU-F1 metric sensibly penalizes errors like "pizzas" $\rightarrow$ "pizza" less than errors like "pizzas" $\rightarrow$ "fries". It is unclear, though, whether character-level edit distance is suitable for the numeric commands of Timers and Such: should "11" $\rightarrow$ "111" (character error rate of 50%) be regarded as less of an error than "11" $\rightarrow$ "22" (character error rate of 100%) when setting a cooking timer in minutes? For this reason, we do not recommend using character-level error to evaluate systems for this task. As a compromise, we also suggest reporting "SLU WER", an easy-to-compute metric that treats the space-delimited output of the SLU model and the true output dictionary as regular sequences of words and simply computes the usual WER metric. Note that no "normalization" of the outputs (e.g., "twelve and a half", "twelve point five" $\rightarrow$ "12.5") is necessary before evaluating, since the labels are always written in the correct numeric format.

Table 6.5: Results (mean and stdev. over 5 random seeds) for all baseline models. See Sec. 6.5.3 for the definition of "SLU WER".

		test-real		test-synth	
Model	Training set	Accuracy	SLU WER	Accuracy	SLU WER
Decoupled (LS LM)	train-real	$24.1\%_{\pm 1.1\%}$	$34.4\%_{\pm 3.3\%}$	$16.1\%_{\pm 1.4\%}$	$33.2\%_{\pm 8.7\%}$
	(both)	$31.4\%_{\pm 4.3\%}$	$26.5\%_{\pm 5.0\%}$	$22.5\%_{\pm 2.1\%}$	$25.2\%_{\pm 2.5\%}$
	train-synth	$32.3\%_{\pm 3.9\%}$	$26.5\%_{\pm 2.5\%}$	$23.7\%_{\pm 1.6\%}$	$24.2\%_{\pm 0.7\%}$
Decoupled (TAS LM)	train-real	$43.5\%_{\pm 2.0\%}$	$20.3\%_{\pm 3.5\%}$	$34.6\%_{\pm 1.2\%}$	$18.5\%_{\pm 3.8\%}$
	(both)	$46.8\%_{\pm 2.1\%}$	$16.5\%_{\pm 2.2\%}$	$38.4\%_{\pm 1.3\%}$	$15.2\%_{\pm 0.9\%}$
	train-synth	$49.1\%_{\pm 2.3\%}$	$16.3\%_{\pm 1.1\%}$	$39.9\%_{\pm 0.7\%}$	$13.9\%_{\pm 0.8\%}$
Multistage (LS LM)	train-real	$55.5\%_{\pm 3.4\%}$	$10.1\%_{\pm 0.6\%}$	$43.1\%_{\pm 2.9\%}$	$10.8\%_{\pm 0.8\%}$
	(both)	$67.8\%_{\pm 1.4\%}$	$7.4\%_{\pm 0.4\%}$	$79.4\%_{\pm 0.4\%}$	$3.2\%_{\pm 0.1\%}$
	train-synth	$66.6\%_{\pm 0.8\%}$	$7.7\%_{\pm 0.8\%}$	$79.1\%_{\pm 0.2\%}$	$3.2\%_{\pm 0.0\%}$
Multistage (TAS LM)	train-real	$64.0\%_{\pm 3.3\%}$	$7.4\%_{\pm 0.9\%}$	$51.5\%_{\pm 2.9\%}$	$8.7\%_{\pm 0.7\%}$
	(both)	$72.6\%_{\pm 1.6\%}$	$5.9\%_{\pm 0.1\%}$	$85.4\%_{\pm 0.2\%}$	$2.4\%_{\pm 0.0\%}$
	train-synth	$72.2\%_{\pm 1.4\%}$	$6.2\%_{\pm 0.4\%}$	$85.4\%_{\pm 0.3\%}$	$2.4\%_{\pm 0.1\%}$
Direct	train-real	$\mathbf{81.6}\%_{\pm 5.4\%}$	$\mathbf{2.6}\%_{\pm 1.1\%}$	$70.0\%_{\pm 5.7\%}$	$15.2\%_{\pm 19.1\%}$
	(both)	$77.5\%_{\pm 1.6\%}$	$3.3\%_{\pm 0.4\%}$	$\mathbf{96.7}\%_{\pm 0.3\%}$	$\mathbf{1.1}\%_{\pm 0.0\%}$
	train-synth	$68.0\%_{\pm 5.5\%}$	$8.9\%_{\pm 3.4\%}$	$96.4\%_{\pm 0.2\%}$	$\mathbf{1.1}\%_{\pm 0.0\%}$

6.5.4 Results

A few trends in the results shown in Table 6.5 are worth noting.

- **The direct model and multistage TAS LM work best.** This is perhaps unsurprising, since these two models effectively have the most opportunity to train on the downstream SLU task.

- **The direct model "overfits" to synthetic speech.** It seems that because the direct model has access to the raw speech features instead of a transcript, it can learn the idiosyncratic pronunciations of the speech synthesizer and achieve much better performance than the ASR-based models (96.7% vs. 85.4%). This model still performs well on the real test data—we mention this simply to explain why this model suddenly performs so much better for the synthetic test data.

- **Test accuracies and SLU WERs[7] have high variability.** Some test accuracies have a

[7]The 19.1% stdev. in SLU WER for the direct model on test-synth is due to a single outlier random

Table 6.6: Baseline results for the `all-real` set.

Model	Training set	`all-real`	
		Accuracy	SLU WER
Decoupled (LS LM)	`train-synth`	$26.8\%_{\pm 3.3\%}$	$29.0\%_{\pm 2.2\%}$
Decoupled (TAS LM)	`train-synth`	$44.6\%_{\pm 2.4\%}$	$17.3\%_{\pm 1.1\%}$
Multistage (LS LM)	`train-synth`	$64.6\%_{\pm 0.7\%}$	$7.2\%_{\pm 0.2\%}$
Multistage (TAS LM)	`train-synth`	$\mathbf{69.9}\%_{\pm 0.9\%}$	$\mathbf{6.0}\%_{\pm 0.2\%}$
Direct	`train-synth`	$68.9\%_{\pm 5.4\%}$	$8.2\%_{\pm 3.4\%}$

standard deviation as high as 5.7%. We observed this phenomenon with the preliminary version of Timers and Such and suspected that the variance was because of the smaller test set size (55 audios). However, this does not seem to be the explanation here, since `all-real` (Table 6.6) has 2,151 audios and still has highly variable test accuracy (stdev. of 3.3%, 2.4%, 0.7%, 0.9%, 5.4%). We will not venture further here to diagnose this problem; instead, we leave it as a problem for future research on this dataset to solve.

6.5.5 Computing resource usage

Training and testing all the SLU models across all random seeds, models, and training set compositions required about 233 GPU-hours on an Nvidia Quadro RTX 8000 GPU. Additionally, the LibriSpeech acoustic model was trained using one Nvidia Tesla V100 GPU for 194 hours, and the LibriSpeech LM was trained using 4 V100s for about 84 hours.

However, we hasten to note for those with limited computing resources interested in experimenting with Timers and Such that i) the pre-trained LibriSpeech models are available online and are downloaded automatically by the recipes, and ii) training a *single* model on Timers and Such can be done relatively quickly, at around a minute per epoch for the direct recipe when training on `train-real`. The decoupled recipe can also be sped up significantly by using a larger batch size during training, since the input is text instead of speech and requires less memory. Note also that all the recipes have also been

seed for which the decoder produced many infinitely looping outputs ("`unit1 unit1 unit1 unit1...`").

successfully tested on an older 12 GB Nvidia Tesla K80 GPU without any hyperparameter modifications.

6.6 Risks and rewards

A risk of recording speech data is that a malicious actor could use the data to imitate the speaker and use the speaker's voice for purposes the speaker did not intend [406]. Similar to Common Voice, it is unlikely that this could happen to the speakers of Timers and Such, since they did not provide any information that could uniquely identify them.

On the whole, we think Timers and Such will be a great benefit to the research community and (indirectly) to users of voice interfaces. Speech datasets are often recorded by professional speakers in clean conditions unlike the conditions in which voice interfaces are typically used. This leads to brittle, overfitted models that break when applied to real-world speech [273]. Timers and Such will contribute to research and development of more robust models that can understand speech in a variety of accents and conditions.

6.7 Conclusion

Timers and Such is a new dataset of numeric commands that should be useful for SLU researchers, hackers aiming to train their own offline voice assistant, and researchers developing new representation learning methods for audio and language [398, 399, 400, 7] looking for another downstream task to test on. Some directions for the future of Timers and Such we hope to see worked on include: diagnosing and fixing the high variability of test performance; exploring the acoustic model architecture (e.g., using a CTC model or Transducer model [175]); speeding up the multistage approach, e.g. by using transfer learning to initialize a multistage model using a decoupled model; improving the performance of the direct model on `all-real`; using an ASR dataset with a more diverse set of accents and recording conditions, like Common Voice [401]; using different tokenizers or other hand-crafted output labels; improving the speech synthesis (using systems such as the RTVC multispeaker TTS [407, 408] to add even more synthetic speakers) and balance

between real and synthetic training data; and enabling streaming inference [409, 410], which cannot be performed with the baseline models as-is, due to their global attention mechanism.

Chapter 7

SpeechBrain: A General-Purpose Speech Toolkit

Abstract

This chapter discusses our contributions to SpeechBrain, a general-purpose open-source PyTorch speech toolkit. The chapter is divided into two sections: in the first, we give a brief introduction to the toolkit; in the second, we focus on our contribution most relevant to the thrust of this thesis: state-of-the-art recipes for spoken language understanding.

7.1 Introduction to SpeechBrain

As discussed in Section 2.3, open-source software is important for the progress of AI research. Existing speech toolkits like Kaldi [63] and HTK [411] are written in C++, which makes fast prototyping of new systems more difficult than when using a higher-level language like Python. Most of speech processing is done using deep neural networks, or can be expressed using neural network primitives like convolutions (much of signal processing), so it makes sense to take advantage of the Python deep learning ecosystem to implement a general-purpose toolkit for speech problems. By combining recipes and functionalities for many different speech problems within a single Python toolkit, SpeechBrain

makes it easy to prototype new applications and combine state-of-the-art systems — e.g., backpropagating through a speech recognizer into a speech enhancement system [412].

SpeechBrain is an efficient, easy-to-use,[1] `pip`-installable PyTorch-based library. It takes care many of the fiddly algorithms, tricks, and implementation details described in Chapter 2, and many more. Below we describe some of the useful features of SpeechBrain. (Note that the functionality described in this section was contributed by many people other than the thesis author, and it is described in fuller detail in the SpeechBrain paper [13].)

Training loop A SpeechBrain training run defines and creates a "Brain" object, and `Brain.fit()` trains the model for a specified number of epochs, with validation after each epoch. It is also possible to define an "epoch" as a certain number of SGD updates, rather than a complete pass through the training set. What happens in each training step, evaluation step, forward computation step, and loss computation step can be overridden by the programmer, as well as what happens at the beginning and end of each epoch.

Lobes "Lobes" in SpeechBrain (like lobes in an animal brain) implement sets of commonly-used functions and modules, such as the FBANK feature extraction pipeline and various neural network encoders and decoders. Unlike older toolkits [63], which pre-compute the features for the entire dataset before training, SpeechBrain computes features on-the-fly, enabling a larger and more sophisticated set of data augmentations to be performed.

Variable-length sequence management Speech and text are variable-length sequences. To be able to perform the same operations operations on each example in a minibatch for efficiency, SpeechBrain dataloaders take care of padding each example out to the same length. The actual lengths of the inputs are passed between modules and used to ensure the correctness of certain operations (e.g., input normalization computes the mean and variance using only the non-padded inputs). The lengths are normalized to the range $[0, 1]$ so that the programmer does not need to worry about whether each module changes the

[1] Colab notebook tutorials for learning to use SpeechBrain can be found on the toolkit website: `https://speechbrain.github.io/`. A reader with basic knowledge of Python and the neural network concepts in Chapter 2 can easily follow these tutorials.

```
from speechbrain.pretrained import EncoderDecoderASR

asr_model = EncoderDecoderASR.from_hparams(
    source ="speechbrain/asr-transformer-transformerlm-librispeech",
    savedir ="pretrained_models/asr")
asr_model.transcribe_file("its_all_greek_to_me.wav")
>>> ["IT'S ALL GREEK TO ME"]
```

Listing 7.1: Pre-trained model example usage.

absolute lengths, e.g., through pooling.

Pre-trained models Inference with pre-trained models for various applications can be implemented using the `pretrained` module. Listing 7.1 shows an example of loading a pre-trained speech recognizer and using it to transcribe an audio file.

Checkpointing and fault tolerance If training stops prematurely (e.g., due to a SLURM job ending on a cluster computer), re-running the training command will resume training from the latest checkpoint. A checkpoint is a periodically saved collection of files containing objects such as the current model parameters and the minibatch index. A checkpoint can also be saved at each validation step, where the programmer provides the metric (e.g., WER) used to determine whether a new "best" set of parameters has been found.

Optimizers The Brain class initializes optimizers for the parameters and uses them to update the weights at each step. The default optimizer setup can be overridden, e.g. to use different optimizers for different parts of the model. The optimizer state is saved in the checkpoint along with other state variables, so that restarting training correctly implements optimizer updates that have memory.

Recipes SpeechBrain comes with state-of-the-art recipes for training and inference for many popular datasets, like LibriSpeech, Common Voice, TIMIT, and VoxCeleb.[2] Each recipe uses a Python file to run the experiment and a YAML file to control the experiment hyperparameters. Different YAML files can be used in conjunction with the same Python file.

[2] `https://github.com/speechbrain/speechbrain/tree/develop/recipes`

```
("Make a calendar entry for brunch on Saturday morning with Aaronson.",
{
  "scenario": "calendar",
  "action": "create_entry",
  "entities": [
    {"type": "event_name", "filler": "brunch"},
    {"type": "date", "filler": "Saturday"},
    {"type": "timeofday", "filler": "morning"},
    {"type": "person", "filler": "Aaronson"}
  ]
})
```

Listing 7.2: A SLURP transcript and its label dictionary.

7.2 Recipes for SLU

The thesis author contributed code for various components, such as the pre-trained model interface, forced alignment, grapheme-to-phoneme conversion, and ASR recipes for the AISHELL-1 Chinese speech dataset [413], as well as high-level design decisions, documentation, and user testing, to the toolkit. Here we specifically describe the contribution most relevant to the thesis, namely the SLU recipes. In addition to the decoupled, multistage, and direct end-to-end SLU recipes for Timers and Such described in the previous chapter, we have recipes for two more datasets: SLURP and Fluent Speech Commands.

7.2.1 SLURP

SLURP [394] is much larger and more semantically challenging than other SLU datasets, with more distinct tokens, bigrams, trigrams, Lexical Sophistication [414], Corrected Verb Sophistication [415], and Mean Segmental Text-to-Token Ratio [416]. Whereas Timers and Such and Fluent Speech Commands can be solved with perfect accuracy given the transcript (i.e., there is no real NLU challenge, given the original splits), SLURP cannot. A SLURP example is shown in Listing 7.2.

The creators of SLURP intended for the dataset to be a challenge for end-to-end SLU models. They write:

> *SLURP is not only bigger, but also a magnitude more challenging than previous datasets. The purpose of this new data release is not to provide yet another bench-*

Table 7.1: Performance on SLURP (audio as input).

Model	`scenario` (accuracy)	`action` (accuracy)	`intent` (accuracy)	**Word-F1**	**Char-F1**	**SLU-F1**
HerMiT [394]	85.69	81.42	78.33	69.34	72.39	70.83
CTI [378]	—	—	**86.92**	—	—	**74.66**
Ours (CRDNN)	82.15	77.79	75.64	62.35	66.45	64.34
Ours (wav2vec 2.0)	**89.49**	**86.40**	85.34	**72.60**	**76.76**	**74.62**

mark dataset, but to provide a use-case inspired new challenge, which is currently beyond the capabilities of SOTA E2E approaches (due to scalability, lack of data efficiency, etc.).

We have tested several SOTA E2E-SLU systems on SLURP, including (Lugosch et al., 2019b) [371 in this thesis] which produces SOTA results on the FSC corpus. However, re-training these models on this more complex domain did not converge or result in meaningful outputs. Note that these models were developed to solve much easier tasks (e.g. a single domain). Developing an appropriate model architecture is left for future work.

SLURP performance is measured using accuracy for the "scenario" and "action" labels, as well as "intent" (= "scenario" + "action"), and using Word-F1, Char-F1, and SLU-F1 measures [394] for the slots. The SLURP authors provide a decoupled baseline using HerMiT [200]. The HerMiT architecture uses a stack of LSTMs pre-trained using ELMo, each followed by a self-attention layer and a CRF decoded using the Viterbi algorithm. In addition to HerMiT, we compare our recipes with the Continuous Token Interface (CTI) [378]. CTI trains a transformer encoder with RoBERTa NLU pre-training [417] as an intent classifier and CRF. The inputs to the encoder are either the softmax outputs of an autoregressive ASR decoder or the gold transcript during training; at test time the ASR decoder softmax outputs are used.

We train an encoder-decoder model similar to the one described in Chapter 6, with a frozen pre-trained LibriSpeech CRDNN encoder followed by a randomly initialized LSTM encoder and an attention-based autoregressive decoder. We train a 58-token SentencePiece tokenizer on the serialized semantic dictionaries and use the tokenized dictionaries as the

Table 7.2: Performance on SLURP (gold transcripts as input).

Model	**scenario** (accuracy)	**action** (accuracy)	**intent** (accuracy)
HerMiT [394]	90.15	86.99	84.84
CTI [378]	—	—	87.73
Ours	**91.45**	**89.46**	**88.68**

output targets. The model is trained for 20 epochs, and the learning rate is lowered by a factor of 0.8 whenever error on the validation set increases.[3] Our model performs reasonably well, but worse than both HerMiT and CTI. When the CRDNN encoder is replaced with a wav2vec 2.0 encoder fine-tuned for ASR on LibriSpeech, the encoder-decoder beats the CRF models across most metrics.[4]

To what extent can the performance differences be attributed to the use of an encoder-decoder model instead of a CRF, and to what extent to the difference in (implicit) ASR capability? To find out, we train an encoder-decoder model on the ground-truth transcripts instead of the audio. The architecture is identical, except instead of the pre-trained audio encoder, we use a simple text embedding layer before the LSTM encoder. The transcripts are tokenized using the 1000-token ASR tokenizer (Section 6.5.1) and fed into the encoder. Our simple NLU model significantly outperforms HerMiT and CTI when those systems are applied to the gold transcripts, and unlike either baseline uses no additional NLU pre-training and does not take advantage of the word-aligned CRF slot labels or the list of intents and slot values.

7.2.2 Fluent Speech Commands

We implemented a recipe for the Fluent Speech Commands dataset [371] using the same CRDNN model and hyperparameters as for SLURP. This recipe outperforms our previous results for the dataset, while CTI slightly outperforms our recipe (Table 7.3). Fluent Speech Commands is a simple dataset and is essentially "solved" in its original form.

[3] `https://github.com/speechbrain/speechbrain/blob/develop/recipes/SLURP/direct/hparams/train.yaml`

[4] Thanks to Boumadane Abdelmoumene for running the wav2vec 2.0 experiment.

Table 7.3: Performance on Fluent Speech Commands (original splits).

Model	**Accuracy**
Ours (Chapter 5)	99.2
CTI [378]	**99.7**
Ours (SpeechBrain)	99.6

7.3 Conclusion

SpeechBrain is a powerful and easy-to-use toolkit for speech and other sequence data. Our SLU recipes demonstrate using it to develop state-of-the-art systems using the toolkit's sequence dataloaders, pre-trained model interface, and built-in neural modules and decoders. In the next chapter we will see a somewhat more exotic use for SpeechBrain.

Chapter 8

Towards End-to-End Voice Control Through Gesture Imitation

Abstract

In this chapter, we introduce an imitation learning framework which attempts to implement the idea floated at the beginning of this thesis: fully learned end-to-end voice control, from audio to actions. We discuss the advantages of agent-based voice control over hard-coded control logic, and the design and creation of our system.

8.1 Disadvantages of handcrafted semantic representations

In Chapters 1 and 2, we argued that the conventional way of doing language understanding in voice control — intent classification and slot filling using human-defined labels — is insufficient for implementing useful general-purpose assistants. Consider the semantics defined in our own Timers and Such dataset in Chapter 6: any request that cannot be construed as one of `SetTimer`, `SetAlarm`, `SimpleMath`, and `UnitConversion` will be handled incorrectly. Even more sophisticated systems, such as Amazon Alexa [418] or SLU systems built using SLURP (Chapter 7), still have this flaw because they are implemented the same way. We would like to build voice control systems that can generalize to *new* intents, not just the ones observed in training.

There are other flaws with conventional voice control apart from the inability to generalize to new types of requests. Even if one somehow got ahold of the "correct" data structure for representing semantics — a hypothetical data structure containing "sufficient statistics" for understanding any possible request — and amassed a sufficiently large corpus labeled with such semantics, one would still need to program the control logic. While programming the logic may be easy for a restricted set of intents like setting timers and doing math, it becomes more difficult and time-consuming for a broader domain. Another problem is streaming inference. Handling an utterance like "set a timer for five minutes — no, sorry, four minutes" is awkward with the conventional setup. A really useful assistant would be capable of reacting in real-time to what the speaker is saying, the way a human can, instead of the clunky "say wakeword/push to talk $\rightarrow$ make your request, then stop talking $\rightarrow$ wait for the assistant to react" interactions one has to go through today.

In this chapter we attempt to show how these desiderata may be met using a neural network agent radically different from the conventional voice control pipeline. We begin by discussing an alternative paradigm for machine learning — reinforcement learning and imitation learning — in which an agent learns to take actions to interact with an environment, which we believe is more suitable for the problem of voice control. We review existing agents for mapping language to actions, categorizing methods by their "action space". We then introduce an architecture for an agent that uses "gestures" as its action space to control a smartphone in real-time in response to spoken commands. Using Timers and Such, we create a dataset of demonstrations following the spoken commands and attempt to train an agent on it. We find that the agent has difficulty learning and end with some suggestions of ways in which it could be improved.

8.2 Agents: reinforcement learning and imitation learning

Since voice control is about performing actions that the speaker wants, it makes sense to think of it as a reinforcement learning (RL) problem. In RL, an agent learns to interact with

its environment to maximize some long-term reward. Usually the RL problem is described using the Markov decision process (MDP) formalism. In an MDP, at each timestep t, the agent is in a given state s_t and takes an action a_t. The environment responds with a reward r_t, and the state changes to s_{t+1}, where $r_t, s_{t+1} \sim p(r_t, s_{t+1}|s_t, a_t)$. Interactions can continue indefinitely, or until an "episode" is complete. The goal of RL is to find a policy $\pi_\theta(s_t)$ — a mapping from the current state to the action that should be taken — such that the expected sum of rewards (the return) is as large as possible. In an MDP with a small number of discrete states and actions, there are efficient algorithms for finding the optimal policy [419]. If the input space is large or continuous (images, audio, text, ...), a function approximator needs to be learned instead. The flexibility and online learning capabilities of neural networks make them a natural choice for implementing the function approximator.

RL methods are classified into "model-based" and "model-free" methods. Model-based methods learn and use a model of the environment $p(r_t, s_{t+1}|s_t, a_t)$ to plan a trajectory of actions that maximizes return. Model-free methods try to learn a policy $\pi_\theta(s_t)$ that works well without learning an explicit model of the environment. Model-free methods are in turn classified into value-based methods (which attempt to learn the expected return from starting in a given state and/or taking a given action in that state) and policy gradient methods (which attempt to maximize the return directly by pushing up the probability of the actions taken in an episode in proportion to some function of the return).

RL requires a teacher to give the agent a reward signal. Sometimes the reward signal can be generated automatically, as in a video game, where "win/lose" or "points scored" can be computed easily from the game state. Sometimes the reward requires a human teacher, as in the case where the reward is "how well-written is this text?". If no reward signal is available, an alternative approach called imitation learning (IL) can be used. In IL, a human expert performs actions in the environment of interest, and the agent learns to copy the expert's demonstrations. IL reduces the RL problem to a (typically much easier) supervised learning problem by learning to predict the taken action given the state. Sometimes this process is referred to more specifically as "behavior cloning", and "imitation learning" is used to refer to a broader set of techniques, e.g. DAgger [420].

8.3 Action spaces for voice control

8.3.1 RAM reads and writes

Think of the microprocessor running your AI program as an agent, and the external devices it controls and communicates with as its environment. At the lowest level of abstraction, the only actions this agent can take are reading from and writing to locations in random access memory (RAM), since modern computers use memory-mapped I/O. One could conceivably train a voice control agent to operate a computer using a humongous softmax output layer over all possible memory locations, using IL or RL with a reward given based on whether the agent's memory accesses ultimately do what the user wants.

This approach is not as farfetched as one might think. In [421], Sygnowski and Michalewski train an agent to play Atari 2600 games using only information stored in RAM. The Neural Turing Machine [114] and Differentiable Neural Computer [422] are able to learn algorithms for copying, sorting, and graph traversal using a small (= 128 slots) external memory. But computers for implementing sophisticated behavior will require large memory spaces — even 1 MB of RAM would correspond to millions of possible actions. In RL, generalization to new actions is obtained using e.g. Gaussian policy parameterizations, which assign similar probability mass to nearby points in action space. For a large, discrete, unstructured action space like RAM, it is not clear how to generalize across actions — two nearby memory locations might correspond to two very different devices or pieces of information — so a higher level of abstraction than raw RAM locations may be necessary for an agent to be able to learn.

8.3.2 Handcrafted high-level actions

Instead of low-level memory accesses or microprocessor instructions, some work has attempted to train a model to predict higher-level actions from the language input [423, 424, 425, 426], such as API calls [427] — a good review of such approaches can be found in [428]. There is a long history of using RL specifically for dialog systems [429, 430, 431], where the actions taken are commonly used routines for the application in question, like

`ListHotelOptions` or `BookFlight`, and the user rewards the system for making it through an episode of dialog turns successfully.

The disadvantage of these approaches is that the higher the level of abstraction, the more engineering effort needs to be expended to implement the actions — at one extreme, we effectively return to the original "hard-coded semantics and control logic" setup, which is precisely what we wanted to avoid by using machine learning. Ideally we would reuse as much existing code as possible, or generate the code automatically instead.

8.3.3 Large language models and code generation

Another possible action space involves the use of models that have recently become known collectively under the name "large language models (LLMs)" [432] (or the sleek and vaguely sci-fi-ish "foundation models" [433], or the charmingly simple "big models" [434]). Neural network LMs prior to the invention of transformers in 2017 were known to be capable of generating semi-coherent text for a sentence or two using (variants of) Algorithm 2 [435], but their generations quickly devolved into nonsense. Shortly after the invention of transformers, [436] showed that transformer LMs could generate long, coherent text by training on Wikipedia articles. The quality of the samples was found to be strongly related to the number of parameters in the model [437] — when generating the article for "Abraham Lincoln", for example, the mean squared error of the estimates for Abraham Lincoln's birth and death dates in the generated text decreased as the size of the model increased [438]. OpenAI's GPT-2 [330] — sequel to GPT (Section 2.2.14) — used a 1.5 billion parameter transformer LM trained on a large dataset scraped from the Internet and produced even more impressive generations. GPT-3 [128], weighing in at 175 billion parameters, was found to be capable not only of long, coherent generations, but also of "few-shot"/"in-context" learning, whereby prompts provide the model with "training data" on-the-fly. Fig. 8.1 shows the use of prompting with GPT-3 to infer labels for examples from Timers and Such.

A text generator may at first not seem useful for producing actions, since the outputs are text and not actions. But much of the text scraped from the Internet contains code. GitHub

```
The following are examples of transcripts and their semantic labels.

("what's 37.67 minus 75.7",
{
  'intent': 'SimpleMath',
  'slots': {
    'number1': 37.67,
    'number2': 75.7,
    'op': ' minus '
  }
})

("convert 79 degrees Kelvin to Fahrenheit",
{
  'intent': 'UnitConversion',
  'slots': {
    'unit1': 'Kelvin',
    'unit2': 'Fahrenheit',
    'amount': 79
  }
})

("set the timer for 1 minute and 56 seconds",
{
  'intent': 'Timer',
  'slots': {
    'minutes': 1,
    'seconds': 56
  }
})

("it's all Greek to me",
{
  'intent': 'Idiom',
  'slots': {}
})
```

Figure 8.1: Few-shot NLU using GPT-3 (`text-davinci-003`). Unbolded lines are written by the thesis author, bolded lines are written by the model.

[*Loren:*] Write a Python function called "set_timer" that parses the user's input and sets a timer for the amount of time they ask for. For example, if I call set_timer("set timer for an hour and a half"), a timer should be set for 1 hour, 30 minutes, and 0 seconds, and if I call set_timer("timer for 10m 30s"), a timer should be set for 0 hours, 10 minutes, and 30 seconds.

[*ChatGPT:*] Here is one way you could implement a set_timer function in Python:

```
import re
import time

def set_timer(input_string: str):
    # Parse the input string to get the time values
    hours_match = re.search(r"(\d+) hours?", input_string)
    minutes_match = re.search(r"(\d+) minutes?", input_string)
    seconds_match = re.search(r"(\d+) seconds?", input_string)
    hours = int(hours_match.group(1)) if hours_match else 0
    minutes = int(minutes_match.group(1)) if minutes_match else 0
    seconds = int(seconds_match.group(1)) if seconds_match else 0

    # Calculate the total number of seconds that the timer should run for
    total_seconds = 3600 * hours + 60 * minutes + seconds

    # Sleep for the specified number of seconds
    time.sleep(total_seconds)
```

Figure 8.2: Code generation using ChatGPT.

Copilot, based on OpenAI's Codex [439], is trained on publicly available code from GitHub. The tool has become very popular among programmers as a way to autocomplete portions of their code. Another method, InstructGPT [440], trains a GPT model on unlabeled text as usual, then applies supervised fine-tuning (human labelers respond to prompts, and their responses are used as training targets), trains a reward model on samples from the fine-tuned GPT model ranked by human labelers, and then uses RL to train the GPT model using the reward model [147]. Fig. 8.2 shows an example of using ChatGPT, a system based on InstructGPT, to generate code for implementing part of a voice assistant.[1]

8.3.4 Human gestures

Instead of imitating actions in the form of text outputs, it is possible to imitate *gestures*. By "gestures", we mean the non-code actions that humans take to control computers: e.g., typing on a keyboard, touching a touchscreen, clicking a mouse, or moving one's limbs in a virtual reality setup. Like code generation, gesture imitation can reuse already-composed high-level actions: by implicitly making use of the code invoked through the gestures a user makes. Gesture imitation has the advantage over code generation that only a relatively small portion of the population knows how to write code to control computers, whereas a much larger portion of the population knows how to control computers using interfaces like touchscreens — so there is potentially a much larger source of training data that could be gathered. Gestures may also generalize across computers, operating systems, and programming languages in a way that code cannot: moving a mouse does the same thing on Windows and on Mac, though the underlying code looks different.

A good recent example of gesture imitation is [441]. Using the screen and the Document Object Model (DOM) as inputs, the team behind [441] train an neural network agent to control a computer using keyboard and mouse inputs. The task the agent solves is the browser-based MiniWob++ task [442], where the agent must complete online goals like booking a flight from a text instruction. Previous work on MiniWob++ uses hard-coded higher-level actions (e.g., "Click(Near(Text("Bob")))"); in [441], the actions instead are

[1] Unfortunately the code does not work correctly.

in the form [action type, (x, y) cursor coordinates, keyboard key index, task field index], where the action type is one of {mouse (move, click, double click, press, release, wheel up, wheel down), keyboard (key press, emit text), no-op}. Actions and observations are taken at 30 Hz. The agent is trained using a combination of IL and RL, and it attains human-level performance on the task.

Similar approaches have had some success in the domain of robotics. BC-Z [443] collects a large dataset of recordings of video and gestures made by human operators using a robot arm in response to commands like "pick up the ceramic bowl". Using pre-trained language models to embed the text command, an agent is trained to imitate the operators' gestures in response to the commands. In "Grounding Language in Play" [444, 445], a clever alternative way of collecting data for gesture imitation is introduced. The experimenters ask human operators to play with a robot arm, using it to manipulate objects with no particular goal in mind, and recordings of the video and gestures made during these play sessions are made. Human labelers then annotate each recorded play example with their answer to "what instruction would you give the agent to get from first frame to last frame?" The annotations can then be *inverted* to be used as the input commands by training a neural network to map from the annotation text to the gestures performed by the robot. In their follow-up work [446], real-time voice control of the robot is enabled by periodically embedding the decoded ASR transcript using CLIP [447] and feeding it to the neural network agent.

8.4 Android gesture imitation

Following this work on gesture imitation, we propose to do something similar for controlling an Android smartphone as a testbed for end-to-end voice control. The action space is simple (touching or lifting a finger on the 2-D plane), but unlike the robot arm or MiniWob++ interface, there is potentially a much larger set of tasks that can be performed using an Android phone — one can even open up an SSH connection and remotely log in

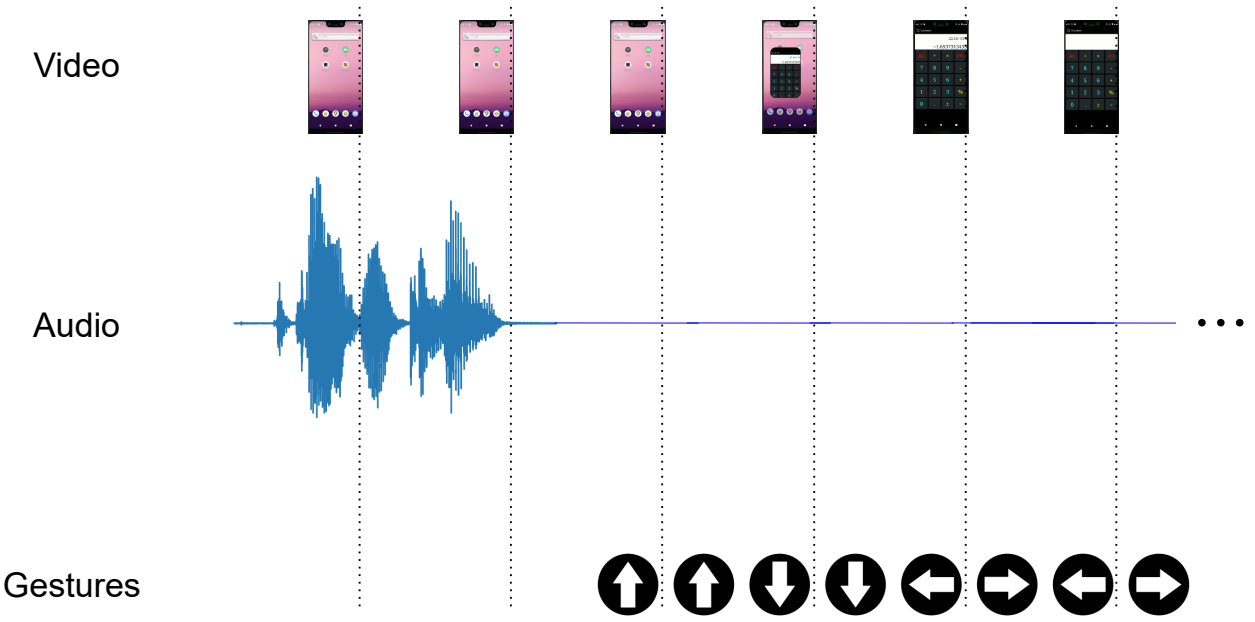


Figure 8.3: Frame-level view of voice control implemented by human gestures on a smartphone. See Section 8.5 for a description of the gesture action structure.

to and control a more powerful computer.[2] Using a smartphone as the environment may also allow for interesting comparisons with existing phone-based voice assistants.

We are not the first to propose using Android as an environment: AndroidEnv [448] provides a Python platform for running RL experiments on an Android emulator, using touchscreen gestures as the action space, with the ability to define a "task" that computes a reward signal and manages the beginnings and ends of episodes. We did not use AndroidEnv for our work because it is not currently capable of recording video and gestures for the purposes of imitation learning. AndroidEnv uses the Android Debug Bridge (ADB) to send individual commands to the emulator, which we found to be too high-latency for our purposes.

Fig. 8.3 shows how voice control looks from the perspective of an agent using the phone. At each timestep, the agent observes an audio frame and a video frame, and takes actions in the form of touchscreen gestures. There are only actions and states/observations in this setup, not rewards. One could derive a reward signal from whether the final frame of an episode is "the same" as the final frame of the demonstration — which would require

[2] An eccentric colleague of ours performed much of his graduate research in this way while living in his van or in the woods in British Columbia, using a solar charger to power the phone.

Figure 8.4: Screenshot of the home screen of the emulated phone used in experiments.

some engineering to determine what counts as “the same” (for example, should the clock time at the top of the screen matter?). It would also be possible to use RL from human feedback to fine-tune the agent. In this chapter we only consider imitation learning.

8.5 Data generation using Timers and Such

We generate training data for imitation learning on the smartphone using Timers and Such as the source of natural language commands. The thesis author listened to all 2,151 non-synthetic examples in Timers and Such and acted out the command spoken in each example using the emulated phone — for instance, for the command “set the timer for 1 minute and 56 seconds”, we open the “Clock” app on the home screen, tap on the “Timer” tab, and enter 1’56” by tapping the appropriate numbers. The phone is a Google Pixel 3 XL API 30 (Android 11.0 x86), emulated on the thesis author’s laptop using Android Studio.

Fig. 8.4 shows the home screen of the emulated device. We set up the "Simple Calculator" app[3] on the home screen alongside the "Clock" app. (We also have a "Unit Converter" app on the home screen, but we use the Google searchbar for unit conversion commands instead for simplicity. An icon for the default APK is also shown on the home screen, but we don't use it for anything.)

The training data[4] is generated by synchronously collecting audio frames, video frames, and gestures. We use PyAudio[5] to record the microphone in real-time at a sampling rate of 16,000 Hz. Samples are returned in chunks from the audio buffer: we use a chunk size of 1,600 samples, which corresponds to 0.1s (10 Hz). Every time an audio frame is returned, we record a video frame by taking a screenshot using MSS[6] — hence the video has a 10 Hz frame rate. The touchscreen gestures are recorded by reading directly from `/dev/input/event2`. There are only two types of gestures: touching the finger to the screen and lifting the finger from the screen. Gestures in Android are registered as a sequence of 16-byte "events" (Fig. 8.5): a Unix timestamp, followed by 3 (= signals an event containing information) or 0 (= signals that there are no more events for this gesture), then one of {57 (= an event indicating the type of gesture), 53 (= event indicating an x-coordinate), 54 (= event indicating an x-coordinate), or 58 (= event indicating the pressure with which the screen is touched — on our emulator, only 0 or 1024)}, and finally a signed 4-byte integer (the x- or y- coordinate, pressure, or 0/-1 for touch/lift, respectively). If the x- or y-coordinate is the same as the previous gesture's, an event for that coordinate is not generated.

Raw video is too expensive to store without compression. Downsampling by a factor of 2, and no more, was possible without making any text in the videos illegible. Because many frames do not change much from the previous frame (see Fig. 8.3), a simple way to compress the video is to save only the indices and values of changed pixels. Sparse coding combined with the highest level of `zip` compression gives us the best results (Table 8.1).

[3] `https://apkpure.com/simple-calculator/com.veronicaapps.veronica.simplecalculator`

[4] The data can be found at `https://zenodo.org/record/7529704`. The code for recording each episode can be found at `https://github.com/lorenlugosch/AGI/blob/main/record.py`.

[5] `https://pypi.org/project/PyAudio/` — when collecting the demonstration dataset, audio recording is only used for synchronization, since we already have the Timers and Such audio.

[6] `https://pypi.org/project/mss/`

```
(1666801105, 913352, 3, 57, 0)
(1666801105, 913352, 3, 53, 32084)
(1666801105, 913352, 3, 54, 32678)
(1666801105, 913352, 3, 58, 1024)
(1666801105, 913352, 0, 0, 0)
(1666801105, 927943, 3, 58, 0)
(1666801105, 927943, 3, 57, -1)
(1666801105, 927943, 0, 0, 0)
```

Figure 8.5: Events for two gestures: touching the bottom-right corner of the screen (first five events), followed by lifting the finger (last three events). The first two fields are seconds and fractions of a second in Unix time.

Table 8.1: File size for a single typical episode video using different compression methods.

Method	**Size**
Uncompressed	296 MB
Downsampled	74 MB
Downsampled + `zip -9`	3 MB
Downsampled + sparse encode	13 MB
Downsampled + sparse encode + `zip -9`	232 KB
Downsampled + MP4	262 KB

However, MP4 decoding was found to be much faster than decoding with our method, so we use MP4 instead, so that dataloading does not become a bottleneck during training. MP4 is lossy, so this introduces a small amount of artifact noise, but never enough to obscure what is happening on the screen.

8.6 Agent architecture

8.6.1 Transducer

Our agent, which we will refer to using the innocent name Android Gesture Imitator (AGI), needs to be capable of taking more than one action per time step, as we have tried to illustrate in Fig. 8.3. (This can happen when dragging the finger across the touchscreen, which is registered as a sequence of "touch" gestures without corresponding "lift" gestures, but also sometimes when tapping quickly.) We therefore implement the AGI baseline using

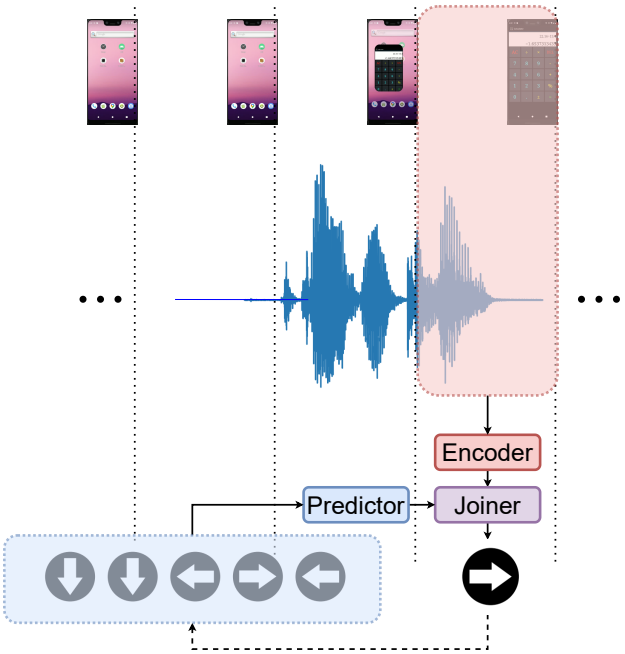


Figure 8.6: Streaming Transducer for gesture imitation.

a Transducer (Section 2.2.12) neural network architecture (Fig. 8.6).

The Transducer has an encoder, predictor, and joiner network. The encoder trunk is a unidirectional 3-layer LSTM with 1024 hidden units per layer and dropout 0.15 applied after each layer. We choose LSTMs so that the model can run online with no modifications, but other streaming networks could also be used [188]. The input to the encoder is a video frame and a set of audio frames. Audio frames contain 1,600 samples, which corresponds to 10 FBANK frames (extracting FBANK features with a 10 ms stride, as we have done throughout the thesis). Thus there are 10 FBANK frames for each video frame. Each 630 $\times$ 300 (downsampled) video frame is encoded using a single 2-D convolutional layer with 3 input channels, 128 output channels, a stride of 7, and a filter length of 7, followed by global max pooling, to produce a much smaller feature vector of size 128. The audio features and video features are concatenated into a single vector and passed through a single linear layer before being fed into the encoder trunk LSTM. Unlike [446], we do not perform ASR decoding, which gives us the advantages of end-to-end SLU discussed in the previous chapters.

The predictor is a 1-layer LSTM with 1024 hidden units, which takes the (decoded or teacher-forced) gestures as input. The encoder and predictor feed into the joiner network,

a feedforward ReLU network with 1024 hidden units followed by a linear transformation to the output activations.

8.6.2 Output parameterization

The output layer of the joiner has a slightly different interpretation from the joiner softmax described in Section 2.2.12: it has a sigmoid output for the "null" $\varnothing$ output (instead of having a softmax over $\varnothing$ and a set of labels); a sigmoid output for the gesture type (touch or lift); and real-valued outputs for an x-coordinate and y-coordinate, normalized to be between 0 and 1. The $\varnothing$ output is equivalent to "no op": move to the next timestep without making a gesture.

Real values (a Gaussian parameterization) are not the only possible way to code the output coordinates. Another option is tile coding [449, 450, 451], which partitions output space into (possibly overlapping) sections and uses their indices as a discrete set. Other options include mixtures of Gaussians (Section 2.2.3) or mixtures of logistics [452].

8.7 Training

The agent can be trained on the episode dataset by maximizing the likelihood of the demonstration gestures.[7] Since the input is a sequence of audio and video frames, and output is a sequence of gestures, one could imagine training a sequence-to-sequence model to output the gestures without an alignment. However, in this instance, the alignment matters because the time at which a gesture is made matters: for example, making a gesture before the required screen has loaded will not register correctly.

Instead of using the full-sum Transducer loss (marginalizing over all possible alignments), we maximize the likelihood of the ground-truth alignment from the demonstration. The probability of a single alignment can be computed using the joiner outputs along the edges of the alignment graph corresponding to the correct (t, u) pairs. Fig. 8.7 shows how the encoder output vectors and predictor output vectors are gathered to form the

[7] Our SpeechBrain code for AGI can be found at `https://github.com/lorenlugosch/AGI`.

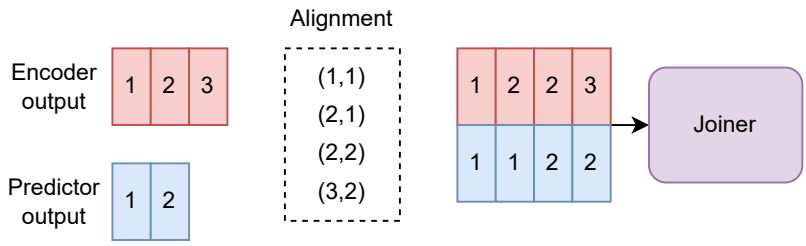


Figure 8.7: Aligned Transducer encoder and predictor outputs.

input to the joiner for a particular alignment during training.

8.8 Decoding

We use a greedy decoding algorithm (similar to the one in Section 2.2.12) to run the agent. At each timestep, we run the encoder, predictor, and joiner to compute the sigmoid probability for $\varnothing$: if it is greater than 0.5, we move to the next timestep; else, we check the sigmoid probability for the gesture type ("touch" or "lift") and output the appropriate gesture. If the gesture is a "touch" gesture, we output the predicted (x, y) coordinates. It might be possible to use Monte Carlo tree search (MCTS) instead of greedy decoding for action selection; we leave exploring this to the future.

8.9 Evaluation

In increasing order of difficulty, four ways in which the agent can be evaluated are

1. **Teacher forcing:** given the correct alignment, measure the accuracy of the predicted gesture types and the mean squared error of the predicted coordinates.

2. **Decoding (video from demonstration):** instead of conditioning on the correct alignment, use greedy decoding. The videos from demonstrations are not the correct visual inputs in this case, since the visual results of the agent's actions will diverge from the demonstration. This scenario is more difficult than when the alignment is provided, but the agent can still "cheat" by observing that a gesture has been made in the demonstration video when deciding whether to output a gesture.

3. **Decoding (video from environment):** the actual scenario in which the agent would work. This requires the agent to interact with the emulator environment in real-time, since the Android emulator itself (not an app within the emulator) does not currently allow for single-step execution. We have found that the computer on which we have conducted our experiments was not fast enough to run even a much smaller baseline network in real-time without skipping input frames. Careful low-level implementation on a faster machine might be necessary to run this type of experiment.

4. **Decoding (blind):** provide no video input. Because this scenario does not require feedback from the environment, it is possible to evaluate it without interacting with the emulator in real-time. Though this scenario is more challenging, it is worth noting that in the case of Timers and Such commands, much of the time the agent only needs the very first video frame to be able to determine whether to exit the current app and open another app.

We have trained a number of baseline agents on the demonstration dataset. We use the Adam optimizer with a learning rate of 3e-4, higher than which led to training diverging. Because the video data consumes a large amount of memory, we have only been able to train using a small batch size of 4 on the Mila cluster. In addition to the architecture described above, we have tested a baseline in which dropout is applied to the video features, a baseline in which the video stream is dropped entirely, and a baseline in which the (character-level, one-hot-encoded) transcript is fed into the encoder instead of the audio.

So far we have not had success with the baseline agents, in any of the different evaluation modes. We find that with teacher forcing, while the loss goes down and "no op" is correctly predicted for most frames, the agent does not do a good job predicting gesture types or coordinates — by which we mean that the predicted gesture coordinates are often not even in the correct upper/lower half or left/right half of the screen. When decoding with or without video input, we find that the agent often does not even start by correctly predicting a "touch" gesture (it is impossible to make a "lift" gesture by lifting the finger without having first made a "touch" gesture).

It is not clear what steps need to be taken for the agent to successfully learn the task. It

could be that the dataset is too small, or that it is necessary to pre-train the encoder for the different modalities in some way using larger external datasets. We leave this exploration for future work.

8.10 Conclusion

We have not yet achieved the ideal of a fully end-to-end deep neural network voice control system — but with the environment, dataset, code, and conceptual framework introduced by this chapter, we hope that future researchers will be able to make "AGI" a reality. We conclude with a few simple suggestions that others might wish to try.

- **More data:** To enable large-scale data collection using crowdsourcing, it would be necessary to replicate the thesis author's recording setup in a way easy for anyone with a computer to use. This could make the dataset a lot larger and provide more trajectories for the same commands and a more diverse set of "gesture styles" for the agent to learn from. Different users training on different emulated devices could enable the agent to generalize to new devices. This setup could also be used to implement the "learning from play" setup in [444] mentioned earlier.

- **Improved loss functions:** Training may be helped by introducing some of the auxiliary loss terms seen earlier, such as the predictor-only autoregressive loss, or an encoder-only transcription loss (CTC). In addition to training on the aligned gestures, the full-sum Transducer loss might help (at the cost of more memory consumption), since learning to match the exact frame in which the demonstrator performs a gesture might be difficult initially. Finally, a CLIP-like loss, in which the agent must learn to guess which gesture sequence corresponds to which audio sequence, might serve to teach the agent the different "intents", without the help of hard-coded semantics. Some work might need to be done to reduce memory consumption to make it possible to use the Transducer loss or use large enough minibatches for the CLIP loss to work.

- **Other imitation learning methods:** DAgger and Upside-Down RL [453, 42] may

also be interesting to explore, in addition to our simple behavior cloning setup.

- **Conditional computation:** State-of-the-art language understanding models seem to require a large number of parameters to store all the world knowledge necessary for accurate prediction. Sophisticated language understanding might not be necessary for AGI to learn the Timers and Such tasks, but it certainly will be for more difficult tasks. Running a model as large as GPT-3 at 10 Hz with low latency will be challenging. Conditional computation, such as the streaming variety we introduced in Chapter 4, might be useful for enabling very large models to do real-time voice control.

- **Alternate training targets:** A limitation of directly imitating gestures is that implementing some functionality using a sequence of gestures takes more time than simply executing the underlying APIs that get called. An alternative worth exploring is to trace which APIs get called by the user's gestures and learn to imitate those calls instead. The disadvantage of imitating API calls is that timing information conveyed by the gesture trajectories might be important for certain functionalities.

Chapter 9

Conclusion

The thesis author recently found himself pretending to be a robot for his 8-year-old nephew, responding with pedantic literality to spoken commands for the young man's amusement and edification. The point was to demonstrate how "computers are stupid and you need to program them to tell them exactly what to do or they won't do it correctly". With each passing year this lesson grows less true. Deep neural networks are now capable of being trained to usefully understand spoken and written language by taking advantage of large labeled and unlabeled datasets in various ways. This thesis has described some of our small contributions to this larger trend, our view of the current limitations of the technology, and our hopes for the future.

Starting with ASR, in Chapter 3 we ported the successful semi-supervised learning recipes used in end-to-end ASR from the monolingual setting to the multilingual setting, greatly improving the performance of a massively multilingual speech recognizer using unlabeled audio with only a few simple modifications to existing techniques. This work resulted in the creation of the open-source M-CTC-T acoustic model, a step towards practical speech recognizers with the code-switching capability alluded to in Chapter 1. In Chapter 4 we explored a very different way to use unlabeled audio for ASR, and a new use for unsupervised pre-training and autoregressive models in general, namely as the front-end of a conditional computation system for neural networks. We showed that using the autoregressive model both to extract features from the input, and to select a bigger downstream network to process surprising inputs, could solve an ASR task with the same performance

as the bigger network using fewer FLOPs.

Next, we turned from ASR to SLU and discussed end-to-end SLU, where a single model maps from audio to semantics instead of composing distinct ASR and NLU models. We advocated the use of end-to-end SLU for its flexibility, simpler software implementation, faster decoding, and ability to optimize all parameters jointly for the actual task of interest. But training and testing SLU models is challenging due to the variability of speech compared to text, the lack of inductive biases conferred by a text representation, and the scarcity of labeled data. Our work in this thesis has mitigated some of these challenges. In Chapter 5, we showed that synthesizing speech from multiple speakers could enable end-to-end SLU without recording new audio or modifying the neural network architecture, and using synthetic speech as data augmentation improved upon our previous state-of-the-art results for the Fluent Speech Commands dataset. In Chapter 6, we described the design and creation of the open-source Timers and Such SLU dataset, along with a set of extensively tested pre-trained LMs, NLU models, SLU models, and code for using the dataset that should be useful for experiments on other SLU benchmarks and beyond. In Chapter 7 on the SpeechBrain toolkit, we described our end-to-end and non-end-to-end SLU recipes in SpeechBrain. In so doing we showed that the SLURP dataset, previously thought to be too challenging for end-to-end models, could in fact be solved with them, the first such proof using a public dataset with non-trivial semantic difficulty. Subsequent state-of-the-art recipes for SLURP have followed our lead in using end-to-end models [454].

Finally, in Chapter 8 we noted the disadvantages of both conventional and end-to-end SLU setups and expressed our hope of getting past the bottleneck of text and handcrafted labels entirely using a more agentic approach. We discussed a number of possible architectures for voice control based on reinforcement learning and imitation learning, more or less handcrafted high-level actions, LLMs and code generation, and directly imitating a human demonstrator in the environment of interest. Focusing on gesture imitation, we created and released a dataset of trajectories for following Timers and Such commands in an Android environment and SpeechBrain code for training agents to control the environment, in the hope that other researchers will solve this task or use it as inspiration for a

more tractable task.

We conclude by listing some ideas for future work.

- The cross-lingual semi-supervised learning experiment described in Section 3.10, in which unlabeled data for the wrong language is used in self-training, raises many questions. Under what circumstances does this work? How different can the languages be, and how much labeled and unlabeled data is required? Answering these questions would be useful for both for practitioners interested in getting as much use out of their unlabeled data as possible and for researchers working on theories of semi-supervised learning.

- Speech synthesizers have improved considerably since we ran the experiments described in Chapters 5 and 6 [455]. It would be worth revisiting these experiments using newer state-of-the-art TTS models and varying the speaking style in addition to the speaker identity.

- Much of the work describing mixture-of-experts models has found them to be unstable during training [456], as we found with even the simple 2-expert model in Chapter 4. Designing auxiliary training targets for learned expert selection, possibly based on surprisal, may help stabilize mixtures-of-experts.

- We only briefly discuss the use of LLMs for end-to-end voice control in Section 8.3.3, but we feel that this is the most promising avenue for improving voice control in the short term. In the long term, we believe the constraints of text-based models in turn-based interactions will make alternative approaches more attractive. Gesture imitation is one such approach. The gesture imitation agent proposed in Chapter 8 may already be capable of useful behavior with some tuning. A missing ingredient that may be necessary is a large-scale dataset of gesture data from human-computer interaction for unsupervised pre-training. Figuring out how to create such a dataset is itself a great challenge.

Bibliography

[1] G. E. Peterson and H. L. Barney, "Control methods used in a study of the vowels," *The Journal of the acoustical society of America*, vol. 24, no. 2, pp. 175–184, 1952.

[2] S. Barreda, "phontools: Functions for phonetics in r," *R package version 0.2-2.1*, 2015.

[3] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[4] J. D. Cook, "How to fit an elephant," June 2011. [Online]. Available: https://www.johndcook.com/blog/2011/06/21/how-to-fit-an-elephant/

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[6] G. Synnaeve, Q. Xu, J. Kahn, T. Likhomanenko, E. Grave, V. Pratap, A. Sriram, V. Liptchinsky, and R. Collobert, "End-to-end ASR: from supervised to semi-supervised learning with modern architectures," *ICML SAS Workshop*, 2020.

[7] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *NeurIPS*, 2020.

[8] H. L. Dreyfus, "From Socrates to expert systems: The limits of calculative rationality," *Bulletin of the American Academy of Arts and Sciences*, vol. 40, no. 4, pp. 15–31, 1987.

[9] L. Lugosch, T. Likhomanenko, G. Synnaeve, and R. Collobert, "Pseudo-labeling for massively multilingual speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 7687–7691.

[10] L. Lugosch, D. Nowrouzezahrai, and B. H. Meyer, "Surprisal-triggered conditional computation with neural networks," *arXiv preprint arXiv:2006.01659*, 2020.

[11] L. Lugosch, B. H. Meyer, D. Nowrouzezahrai, and M. Ravanelli, "Using speech synthesis to train end-to-end spoken language understanding models," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 8499–8503.

[12] L. Lugosch, P. Papreja, M. Ravanelli, A. Heba, and T. Parcollet, "Timers and such: A practical benchmark for spoken language understanding with numbers," *NeurIPS Datasets and Benchmarks*, 2021.

[13] M. Ravanelli, T. Parcollet, P. Plantinga, A. Rouhe, S. Cornell, L. Lugosch, C. Subakan, N. Dawalatabad, A. Heba, J. Zhong, J.-C. Chou, S.-L. Yeh, S.-W. Fu, C.-F. Liao, E. Rastorgueva, F. Grondin, W. Aris, H. Na, Y. Gao, R. D. Mori, and Y. Bengio, "SpeechBrain: A general-purpose speech toolkit," 2021, arXiv:2106.04624.

[14] B. Li, T. N. Sainath, A. Narayanan, J. Caroselli, M. Bacchiani, A. Misra, I. Shafran, H. Sak, G. Pundak, K. K. Chin *et al.*, "Acoustic modeling for google home." in *Interspeech*, 2017, pp. 399–403.

[15] G. Fant, "The source filter concept in voice production," *STL-QPSR*, vol. 1, no. 1981, pp. 21–37, 1981.

[16] D. O'shaughnessy, *Speech communications: Human and machine (IEEE)*. Universities press, 1987.

[17] F. J. Harris, "Windows, harmonic analysis and the discrete fourier transform," NAVAL UNDERSEA CENTER SAN DIEGO CA, Tech. Rep., 1976.

[18] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*. PMLR, 2016, pp. 173–182.

[19] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *The journal of the acoustical society of america*, vol. 8, no. 3, pp. 185–190, 1937.

[20] A.-r. Mohamed, G. Hinton, and G. Penn, "Understanding how deep belief networks perform acoustic modelling," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 4273–4276.

[21] R. Collobert, C. Puhrsch, and G. Synnaeve, "wav2letter: an end-to-end convnet-based speech recognition system," *arXiv preprint arXiv:1609.03193*, 2016.

[22] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv preprint arXiv:1904.05862*, 2019.

[23] D. Palaz, R. Collobert, and M. M. Doss, "End-to-end phoneme sequence recognition using convolutional neural networks," *NeurIPS Deep Learning Workshop*, 2013.

[24] N. Zeghidour, N. Usunier, G. Synnaeve, R. Collobert, and E. Dupoux, "End-to-end speech recognition from the raw waveform," *Interspeech*, 2018.

[25] M. Ravanelli and Y. Bengio, "Speaker recognition from raw waveform with sincnet," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 1021–1028.

[26] R. K. Moore and L. Skidmore, "On the Use/Misuse of the Term 'Phoneme'," *Interspeech*, 2019.

[27] F. Eyben, M. Wöllmer, B. Schuller, and A. Graves, "From speech to letters-using a novel neural network architecture for grapheme based ASR," in *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*. IEEE, 2009, pp. 376–380.

[28] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *International conference on machine learning*. PMLR, 2014, pp. 1764–1772.

[29] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens *et al.*, "Moses: Open source toolkit for statistical machine translation," in *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, 2007, pp. 177–180.

[30] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing," *arXiv preprint arXiv:1808.06226*, 2018.

[31] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[32] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[33] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*, 2020, pp. 1691–1703.

[34] R. A. Fisher, "On the mathematical foundations of theoretical statistics," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 222, no. 594-604, pp. 309–368, 1922.

[35] S. M. Stigler, "The epic story of maximum likelihood," *Statistical Science*, pp. 598–620, 2007.

[36] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *ICLR*, 2020.

[37] D. Weissenborn, O. Täckström, and J. Uszkoreit, "Scaling autoregressive video models," in *International Conference on Learning Representations*, 2019.

[38] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[39] T. Mikolov, M. Karafiát, L. Burget *et al.*, "Recurrent neural network based language model." in *In INTERSPEECH 2010,*. Citeseer, 2010.

[40] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: generating realistic graphs with deep auto-regressive models," in *International conference on machine learning*. PMLR, 2018, pp. 5708–5717.

[41] C. Nash, Y. Ganin, S. A. Eslami, and P. Battaglia, "Polygen: An autoregressive generative model of 3d meshes," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7220–7229.

[42] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15 084–15 097, 2021.

[43] T. H. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–38, 2020.

[44] R. Bonatti, S. Vemprala, S. Ma, F. Frujeri, S. Chen, and A. Kapoor, "PACT: Perception-action causal transformer for autoregressive robotics pre-training," *arXiv preprint arXiv:2209.11133*, 2022.

[45] F. Pineda, "Generalization of back propagation to recurrent and higher order neural networks," in *Neural information processing systems*, 1987, pp. 602–611.

[46] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[47] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *1995 international conference on acoustics, speech, and signal processing*, vol. 1. IEEE, 1995, pp. 181–184.

[48] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE transactions on acoustics, speech, and signal processing*, vol. 35, no. 3, pp. 400–401, 1987.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[50] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE transactions on audio, speech, and language processing*, vol. 20, no. 1, pp. 14–22, 2011.

[51] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[52] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[53] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[54] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[55] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: Parameter estimation," *Using Large Corpora*, p. 223, 1994.

[56] S. Vogel, H. Ney, and C. Tillmann, "HMM-based word alignment in statistical translation," in *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics*, 1996.

[57] J. K. Baker, "Trainable grammars for speech recognition," *The Journal of the Acoustical Society of America*, vol. 65, no. S1, pp. S132–S132, 1979.

[58] B. T. Lowerre, *The Harpy speech recognition system*. Carnegie Mellon University, 1976.

[59] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM," 1993.

[60] F. Jelinek, "Continuous speech recognition by statistical methods," *Proceedings of the IEEE*, vol. 64, no. 4, pp. 532–556, 1976.

[61] L. R. Bahl, F. Jelinek, and R. L. Mercer, "A maximum likelihood approach to continuous speech recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 179–190, 1983.

[62] J. Droppo and O. Elibol, "Scaling laws for acoustic models," *arXiv preprint arXiv:2106.09488*, 2021.

[63] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The Kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*, no. CONF. IEEE Signal Processing Society, 2011.

[64] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[65] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[66] P. Winston, "MIT 6.034 Artificial Intelligence - Lecture 12a: Neural Nets," Fall 2015. [Online]. Available: https://www.youtube.com/watch?v=uXt8qF2Zzfo

[67] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[68] M. Minsky and S. Papert, "Perceptrons." 1969.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[70] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[71] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[72] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[73] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 160–167.

[74] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2008.

[75] G. B. Orr and K.-R. Müller, *Neural networks: tricks of the trade*. Springer, 1998.

[76] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[78] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[79] Y. Bengio, S. Bengio, and J. Cloutier, *Learning a synaptic learning rule*. Citeseer, 1990.

[80] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[81] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[82] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[83] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[84] J. Eisner, "Inside-outside and forward-backward algorithms are just backprop (tutorial paper)," in *Proceedings of the Workshop on Structured Prediction for NLP*, 2016, pp. 1–17.

[85] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[86] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[87] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[88] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, 2006.

[89] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[90] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[91] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[92] A. Irpan, "On the perils of batch norm," https://www.alexirpan.com/2017/04/26/perils-batch-norm.html, 2017.

[93] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[94] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[95] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[96] K. Greff, R. K. Srivastava, and J. Schmidhuber, "Highway and residual networks learn unrolled iterative estimation," *arXiv preprint arXiv:1612.07771*, 2016.

[97] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[98] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," *arXiv preprint arXiv:1909.11556*, 2019.

[99] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[100] T. Likhomanenko, Q. Xu, J. Kahn, G. Synnaeve, and R. Collobert, "slimIPL: Language-Model-Free Iterative Pseudo-Labeling," *Interspeech*, 2021.

[101] E. Jang, C. Raffel, I. Gulrajani, and D. M. de Almeida, "Aesthetically pleasing learning rates," https://blog.evjang.com/2018/04/aesthetic-lr.html, 2018.

[102] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu, "On layer normalization in the transformer architecture," in *International Conference on Machine Learning*. PMLR, 2020, pp. 10 524–10 533.

[103] P. Xu and S. Prince, "Tutorial #17: Transformers iii training," https://www.borealisai.com/research-blogs/tutorial-17-transformers-iii-training/, 2022.

[104] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Sixteenth annual conference of the international speech communication association*, 2015.

[105] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A simple data augmentation method for automatic speech recognition," *Interspeech*, 2019.

[106] T. Kudo, "Subword regularization: Improving neural network translation models with multiple subword candidates," *arXiv preprint arXiv:1804.10959*, 2018.

[107] A. Hannun, "Training sequence models with attention," 2017. [Online]. Available: https://awni.github.io/train-sequence-models/

[108] S. Sun and M. Iyyer, "Revisiting simple neural probabilistic language models," *arXiv preprint arXiv:2104.03474*, 2021.

[109] A. Karpathy, "Deep neural nets: 33 years ago and 33 years from now," https://karpathy.github.io/2022/03/14/lecun1989/, 2022.

[110] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[111] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[112] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The handbook of brain theory and neural networks*, 1998, pp. 255–258.

[113] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 440–449.

[114] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[115] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[116] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *NeurIPS Workshop on Deep Learning*, 2014.

[117] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2013, pp. 6645–6649.

[118] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[119] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[120] G. Lucas, *Star Wars: A New Hope*, 1977, vol. 4.

[121] L. Weng, "Attention? attention!" *lilianweng.github.io*, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-06-24-attention/

[122] O. Press, N. A. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," *arXiv preprint arXiv:2108.12409*, 2021.

[123] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *NAACL*, 2018.

[124] T. Likhomanenko, Q. Xu, R. Collobert, G. Synnaeve, and A. Rogozhnikov, "CAPE: Encoding relative positions with continuous augmented positional embeddings," *NeurIPS*, 2021.

[125] A. Vaswani and A. Huang, "Stanford CS224N: NLP with Deep Learning - Lecture 14: Transformers and Self-Attention," Winter 2019. [Online]. Available: https://www.youtube.com/watch?v=5vcj8kSwBCY

[126] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[127] D. Povey, H. Hadian, P. Ghahremani, K. Li, and S. Khudanpur, "A time-restricted self-attention layer for asr," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 5874–5878.

[128] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner,

S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[129] D. Hutchins, I. Schlag, Y. Wu, E. Dyer, and B. Neyshabur, "Block-recurrent transformers," *arXiv preprint arXiv:2203.07852*, 2022.

[130] A. M. Rush, "The annotated transformer," in *Proceedings of workshop for NLP open source software (NLP-OSS)*, 2018, pp. 52–60.

[131] H. A. Bourlard and N. Morgan, *Connectionist speech recognition: a hybrid approach*. Springer Science & Business Media, 1994, vol. 247.

[132] H. Bourlard and C. J. Wellekens, "Links between markov models and multilayer perceptrons," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1. Morgan-Kaufmann, 1988. [Online]. Available: https://proceedings.neurips.cc/paper/1988/file/0777d5c17d4066b82ab86dff8a46af6f-Paper.pdf

[133] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[134] V. Peddinti, D. Povey, and S. Khudanpur, "A time delay neural network architecture for efficient modeling of long temporal contexts," in *Sixteenth annual conference of the international speech communication association*, 2015.

[135] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[136] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1700–1709.

[137] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[138] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[139] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[140] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR*, 2015.

[141] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[142] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1243–1252.

[143] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[144] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," *Advances in neural information processing systems*, vol. 28, 2015.

[145] M. Norouzi, S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans *et al.*, "Reward augmented maximum likelihood for neural structured prediction," *Advances In Neural Information Processing Systems*, vol. 29, 2016.

[146] R. Prabhavalkar, T. N. Sainath, Y. Wu, P. Nguyen, Z. Chen, C.-C. Chiu, and A. Kannan, "Minimum word error rate training for attention-based sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4839–4843.

[147] N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano, "Learning to summarize with human feedback," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3008–3021, 2020.

[148] R. Reddy *et al.*, "Speech understanding systems: A summary of results of the five-year research effort," *Department of Computer Science. Carnegie-Mellon University, Pittsburgh, PA*, vol. 17, p. 138, 1977.

[149] K. Murray and D. Chiang, "Correcting length bias in neural machine translation," *arXiv preprint arXiv:1808.10006*, 2018.

[150] C. Gulcehre, O. Firat, K. Xu, K. Cho, L. Barrault, H.-C. Lin, F. Bougares, H. Schwenk, and Y. Bengio, "On using monolingual corpora in neural machine translation," *arXiv preprint arXiv:1503.03535*, 2015.

[151] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," *NeurIPS*, 2015.

[152] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 4960–4964.

[153] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition." in *Interspeech*, 2017, pp. 939–943.

[154] S. Kim, T. Hori, and S. Watanabe, "Joint CTC-attention based end-to-end speech recognition using multi-task learning," in *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2017, pp. 4835–4839.

[155] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006, pp. 369–376.

[156] A. Hannun, "Sequence Modeling with CTC," *Distill*, 2017, https://distill.pub/2017/ctc.

[157] A. Zeyer, R. Schlüter, and H. Ney, "Why does CTC result in peaky behavior?" *arXiv preprint arXiv:2105.14849*, 2021.

[158] L. Lugosch, S. Myer, and V. S. Tomar, "DONUT: CTC-based query-by-example keyword spotting," *NeurIPS Workshop on Interpretability and Robustness in Audio, Speech, and Language*, 2018.

[159] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng, "First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs," *arXiv preprint arXiv:1408.2873*, 2014.

[160] J. Kahn *et al.*, "Flashlight: Enabling innovation in tools for machine learning," *arXiv preprint arXiv:2201.12465*, 2022.

[161] A. Zeyer, E. Beck, R. Schlüter, and H. Ney, "CTC in the Context of Generalized Full-Sum HMM Training," *Proc. Interspeech 2017*, pp. 944–948, 2017.

[162] Y. Bengio, R. De Mori, G. Flammia, and R. Kompe, "Global optimization of a neural network-hidden markov model hybrid," *IEEE transactions on Neural Networks*, vol. 3, no. 2, pp. 252–259, 1992.

[163] Y. Bengio, "Artificial neural networks and their application to sequence recognition," Ph.D. dissertation, McGill University, 1991.

[164] T. Bluche, H. Ney, J. Louradour, and C. Kermorvant, "Framewise and CTC training of neural networks for handwriting recognition," in *2015 13th international conference on document analysis and recognition (ICDAR)*. IEEE, 2015, pp. 81–85.

[165] T. Bluche, "Deep neural networks for large vocabulary handwritten text recognition," Ph.D. dissertation, Paris 11, 2015.

[166] T. Bluche, C. Kermorvant, H. Ney, and J. Louradour, "La CTC et son intrigant label 'blank'," 2016.

[167] Y. Miao, M. Gowayyed, and F. Metze, "Eesen: End-to-end speech recognition using deep RNN models and WFST-based decoding," in *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*. IEEE, 2015, pp. 167–174.

[168] A. Laptev, S. Majumdar, and B. Ginsburg, "CTC variations through new WFST topologies," *arXiv preprint arXiv:2110.03098*, 2021.

[169] A. Heba, T. Pellegrini, J.-P. Lorré, and R. Andre-Obrecht, "Char+ CV-CTC: combining graphemes and consonant/vowel units for CTC-based ASR using Multitask Learning," in *20th Annual Conference of the International Speech Communication Association (INTERSPEECH 2019)*, 2019, pp. 1611–1615.

[170] P. Rastogi, R. Cotterell, and J. Eisner, "Weighting finite-state transductions with neural context." Association for Computational Linguistics, 2016.

[171] R. Collobert, A. Hannun, and G. Synnaeve, "A fully differentiable beam search decoder," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1341–1350.

[172] A. Hannun, V. Pratap, J. Kahn, and W.-N. Hsu, "Differentiable weighted finite-state transducers," *arXiv preprint arXiv:2010.01003*, 2020.

[173] A. Zeyer, A. Merboldt, R. Schlüter, and H. Ney, "A new training pipeline for an improved neural transducer," *arXiv preprint arXiv:2005.09319*, 2020.

[174] N. Jaitly, D. Sussillo, Q. V. Le, O. Vinyals, I. Sutskever, and S. Bengio, "A neural transducer," *arXiv preprint arXiv:1511.04868*, 2015.

[175] A. Graves, "Sequence transduction with recurrent neural networks," *ICML Rep. Learn. Workshop*, 2012.

[176] L. Lugosch, "Sequence-to-sequence learning with Transducers," Nov 2020. [Online]. Available: https://lorenlugosch.github.io/posts/2020/11/transducer/

[177] K. Rao, H. Sak, and R. Prabhavalkar, "Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 193–199.

[178] F. Boyer, Y. Shinohara, T. Ishii, H. Inaguma, and S. Watanabe, "A study of transducer based end-to-end asr with espnet: Architecture, auxiliary loss and decoding strategies," in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2021, pp. 16–23.

[179] J.-J. Jeon and E. Kim, "Multitask learning and joint optimization for transformer-rnn-transducer speech recognition," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6793–6797.

[180] J. Mahadeokar, Y. Shangguan, D. Le, G. Keren, H. Su, T. Le, C.-F. Yeh, C. Fuegen, and M. L. Seltzer, "Alignment restricted streaming recurrent neural network transducer," in *2021 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2021, pp. 52–59.

[181] Y. Wang, Z. Chen, C. Zheng, Y. Zhang, W. Han, and P. Haghani, "Accelerating RNN-T Training and Inference Using CTC guidance," *arXiv preprint arXiv:2210.16481*, 2022.

[182] H. Hu, R. Zhao, J. Li, L. Lu, and Y. Gong, "Exploring pre-training with alignments for rnn transducer based end-to-end speech recognition," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7079–7083.

[183] F. Kuang, L. Guo, W. Kang, L. Lin, M. Luo, Z. Yao, and D. Povey, "Pruned RNN-T for fast, memory-efficient ASR training," *arXiv preprint arXiv:2206.13236*, 2022.

[184] J. Li, R. Zhao, H. Hu, and Y. Gong, "Improving rnn transducer modeling for end-to-end speech recognition," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 114–121.

[185] J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde, "Jasper: An end-to-end convolutional neural acoustic model," *Interspeech*, 2019.

[186] J. Chen, X. Tan, Y. Leng, J. Xu, G. Wen, T. Qin, and T.-Y. Liu, "Speech-T: Transducer for Text to Speech and Beyond," *Advances in Neural Information Processing Systems*, vol. 34, pp. 6621–6633, 2021.

[187] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," *arXiv*, 2020.

[188] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6381–6385.

[189] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[190] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[191] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, no. 1, pp. 43–52, 2010.

[192] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," *Advances in neural information processing systems*, vol. 28, 2015.

[193] V. Pratap, Q. Xu, T. Likhomanenko, G. Synnaeve, and R. Collobert, "Word order does not matter for speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 7202–7206.

[194] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *ACL*, 2018.

[195] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North*

American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[196] L. A. Ramshaw and M. P. Marcus, "Text chunking using transformation-based learning," in *Natural language processing using very large corpora*. Springer, 1999, pp. 157–176.

[197] H. Weld, X. Huang, S. Long, J. Poon, and S. C. Han, "A survey of joint intent detection and slot filling models in natural language understanding," *ACM Computing Surveys (CSUR)*, 2021.

[198] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[199] R. Collobert, "Deep learning for efficient discriminative parsing," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 224–232.

[200] A. Vanzo, E. Bastianelli, and O. Lemon, "Hierarchical multi-task natural language understanding for cross-domain conversational AI: HerMiT NLU," *arXiv preprint arXiv:1910.00912*, 2019.

[201] D. Hakkani-Tür, G. Tür, A. Celikyilmaz, Y.-N. Chen, J. Gao, L. Deng, and Y.-Y. Wang, "Multi-domain joint semantic frame parsing using bi-directional RNN-LSTM." in *Interspeech*, 2016, pp. 715–719.

[202] G. Mesnil, Y. Dauphin, K. Yao, Y. Bengio, L. Deng, D. Hakkani-Tur, X. He, L. Heck, G. Tur, D. Yu *et al.*, "Using recurrent neural networks for slot filling in spoken language understanding," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 3, pp. 530–539, 2014.

[203] Y.-Y. Wang and A. Acero, "Discriminative models for spoken language understanding." in *Interspeech*, 2006.

[204] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer." *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[205] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" *arXiv preprint arXiv:2002.08910*, 2020.

[206] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," *arXiv preprint arXiv:1609.01454*, 2016.

[207] S. Rongali, L. Soldaini, E. Monti, and W. Hamza, "Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing," in *Proceedings of The Web Conference 2020*, 2020, pp. 2962–2968.

[208] P. Haghani, A. Narayanan, M. Bacchiani, G. Chuang, N. Gaur, P. Moreno, R. Prabhavalkar, Z. Qu, and A. Waters, "From Audio to Semantics: Approaches to end-to-end spoken language understanding," *IEEE Spoken Language Technology Workshop (SLT)*, 2018.

[209] H. Liao, E. McDermott, and A. Senior, "Large scale deep neural network acoustic modeling with semi-supervised training data for youtube video transcription," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, 2013, pp. 368–373.

[210] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[211] N. Jaitly and G. Hinton, "Learning a better representation of speech soundwaves using restricted boltzmann machines," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2011, pp. 5884–5887.

[212] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[213] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[214] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in neural information processing systems*, 2015, pp. 3079–3087.

[215] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training."

[216] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 2227–2237.

[217] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu, "ERNIE: Enhanced language representation with informative entities," *arXiv preprint arXiv:1905.07129*, 2019.

[218] W. L. Taylor, ""Cloze procedure": A new tool for measuring readability," *Journalism quarterly*, vol. 30, no. 4, pp. 415–433, 1953.

[219] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, 2019, pp. 5754–5764.

[220] Y.-A. Chung and J. Glass, "Generative pre-training for speech with autoregressive predictive coding," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 3497–3501.

[221] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[222] A. Anand, "Contrastive self-supervised learning," 2020, https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html.

[223] P. von Platen, "Xlsr-wav2vec2 fine-tuning week for low-resource languages," 2021, https://discuss.huggingface.co/t/open-to-the-community-xlsr-wav2vec2-fine-tuning-week-for-low-resource-languages/4467.

[224] M. Peters, S. Ruder, and N. A. Smith, "To tune or not to tune? Adapting pretrained representations to diverse tasks," *4th Workshop on Representation Learning for NLP*, 2019.

[225] J. Schmidhuber, "Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments," 1990.

[226] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[227] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[228] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for high performance and scientific computing*, vol. 14, no. 9, pp. 1–9, 2011.

[229] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[230] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[231] T. Tieleman, "Gnumpy: an easy way to use GPU boards in python," *Department of Computer Science, University of Toronto*, 2010.

[232] H. He, "Making deep learning go brrrr from first principles," 2022. [Online]. Available: https://horace.io/brrr_intro.html

[233] J. Pineau, "The machine learning reproducibility checklist," 2020, https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf.

[234] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, *Jupyter Notebooks-a publishing format for reproducible computational workflows.*, 2016, vol. 2016.

[235] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[236] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Autograd: Effortless gradients in numpy," in *ICML 2015 AutoML workshop*, vol. 238, no. 5, 2015.

[237] A. Griewank and A. Walther, *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[238] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.

[239] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, vol. 4, no. 3. Austin, TX, 2010, pp. 1–7.

[240] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: a system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 2016, pp. 265–283.

[241] R. Frostig, M. J. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," *Systems for Machine Learning*, vol. 4, no. 9, 2018.

[242] J. Howard and S. Gugger, "fastai: a layered API for deep learning," *Information*, vol. 11, no. 2, p. 108, 2020.

[243] Q. Xu, A. Baevski, T. Likhomanenko, P. Tomasello, A. Conneau, R. Collobert, G. Synnaeve, and M. Auli, "Self-training and pre-training are complementary for speech recognition," in *ICASSP*, 2021.

[244] J. Du, É. Grave, B. Gunel, V. Chaudhary, O. Celebi, M. Auli, V. Stoyanov, and A. Conneau, "Self-training improves pre-training for natural language understanding," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 5408–5418.

[245] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[246] O. Chapelle, B. Schölkopf, and A. Zien, Eds., *Semi-Supervised Learning*. Cambridge, MA: MIT Press, 2006. [Online]. Available: http://www.olivier.chapelle.cc/ssl-book/

[247] A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. Goodfellow, "Realistic evaluation of deep semi-supervised learning algorithms," *Advances in neural information processing systems*, vol. 31, 2018.

[248] H. Scudder, "Probability of error of some adaptive pattern-recognition machines," *IEEE Transactions on Information Theory*, vol. 11, no. 3, pp. 363–371, 1965.

[249] D.-H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," in *ICML Workshop on Rep. Learning*, 2013.

[250] T. Likhomanenko, R. Collobert, N. Jaitly, and S. Bengio, "Continuous soft pseudo-labeling in ASR," *arXiv preprint arXiv:2211.06007*, 2022.

[251] J. Kahn, A. Lee, and A. Hannun, "Self-training for end-to-end speech recognition," in *ICASSP*, 2020.

[252] S. Thomas, M. L. Seltzer, K. Church, and H. Hermansky, "Deep neural network features and semi-supervised training for low resource speech recognition," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6704–6708.

[253] Q. Xu, T. Likhomanenko, J. Kahn, A. Hannun, G. Synnaeve, and R. Collobert, "Iterative pseudo-labeling for speech recognition," *Interspeech*, 2020.

[254] D. S. Park, Y. Zhang, Y. Jia, W. Han, C.-C. Chiu, B. Li, Y. Wu, and Q. V. Le, "Improved noisy student training for automatic speech recognition," *Interspeech*, 2020.

[255] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[256] Y. Chen, W. Wang, and C. Wang, "Semi-supervised ASR by end-to-end self-training," *Proc. Interspeech 2020*, pp. 2787–2791, 2020.

[257] D. Berrebbi, R. Collobert, S. Bengio, N. Jaitly, and T. Likhomanenko, "Continuous pseudo-labeling from the start," *arXiv preprint arXiv:2210.08711*, 2022.

[258] Y. Higuchi, N. Moritz, J. L. Roux, and T. Hori, "Momentum pseudo-labeling for semi-supervised speech recognition," *Interspeech*, 2021.

[259] V. Manohar, T. Likhomanenko, Q. Xu, W.-N. Hsu, R. Collobert, Y. Saraf, G. Zweig, and A. Mohamed, "Kaizen: Continuously improving teacher using exponential moving average for semi-supervised speech recognition," in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2021, pp. 518–525.

[260] T. Schultz and A. Waibel, "Multilingual and crosslingual speech recognition," in *DARPA Workshop on Broadcast News Transcription and Understanding*, 1998.

[261] V. Pratap, A. Sriram, P. Tomasello, A. Hannun, V. Liptchinsky, G. Synnaeve, and R. Collobert, "Massively multilingual ASR: 50 languages, 1 model, 1 billion parameters," *Interspeech*, 2020.

[262] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common Voice: A Massively-Multilingual Speech Corpus," *LREC*, 2020.

[263] C. Wang, M. Rivière, A. Lee, A. Wu, C. Talnikar, D. Haziza, M. Williamson, J. Pino, and E. Dupoux, "VoxPopuli: A large-scale multilingual speech corpus for representation learning, semi-supervised learning and interpretation," *ACL*, 2021.

[264] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "LibriSpeech: an ASR corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[265] M. Müller, S. Stüker, and A. Waibel, "Phonemic and graphemic multilingual CTC based speech recognition," *arXiv*, 2017.

[266] S. Tong, P. N. Garner, and H. Bourlard, "Cross-lingual adaptation of a CTC-based multilingual acoustic model," *Speech Communication*, vol. 104, pp. 39–46, 2018.

[267] A. Conneau, A. Baevski, R. Collobert, A. Mohamed, and M. Auli, "Unsupervised cross-lingual representation learning for speech recognition," *arXiv*, 2020.

[268] O. Adams, M. Wiesner, S. Watanabe, and D. Yarowsky, "Massively multilingual adversarial speech recognition," 2019.

[269] A. Kannan, A. Datta, T. N. Sainath, E. Weinstein, B. Ramabhadran, Y. Wu, A. Bapna, Z. Chen, and S. Lee, "Large-scale multilingual speech recognition with a streaming end-to-end model," *Interspeech*, 2019.

[270] J. Cho, M. K. Baskar, R. Li, M. Wiesner, S. H. Mallidi, N. Yalta, M. Karafiat, S. Watanabe, and T. Hori, "Multilingual sequence-to-sequence speech recognition: architecture, transfer learning, and language modeling," 2018.

[271] B. Li, Y. Zhang, T. Sainath, Y. Wu, and W. Chan, "Bytes are all you need: End-to-end multilingual speech recognition and synthesis with bytes," 2018.

[272] W. Hou, Y. Dong, B. Zhuang, L. Yang, J. Shi, and T. Shinozaki, "Large-scale end-to-end multilingual speech recognition and language identification with multi-task learning," in *Interspeech*, 2020.

[273] T. Likhomanenko, Q. Xu, V. Pratap, P. Tomasello, J. Kahn, G. Avidov, R. Collobert, and G. Synnaeve, "Rethinking Evaluation in ASR: Are Our Models Robust Enough?" *arXiv preprint arXiv:2010.11745*, 2020.

[274] S. Toshniwal, T. N. Sainath, R. J. Weiss, B. Li, P. Moreno, E. Weinstein, and K. Rao, "Multilingual speech recognition with a single end-to-end model," in *ICASSP*, 2018.

[275] G. Ghiasi, B. Zoph, E. D. Cubuk, Q. V. Le, and T.-Y. Lin, "Multi-task self-training for learning general representations," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 8856–8865.

[276] C.-C. Chiu, A. Narayanan, W. Han, R. Prabhavalkar, Y. Zhang, N. Jaitly, R. Pang, T. N. Sainath, P. Nguyen, L. Cao *et al.*, "RNN-T models fail to generalize to out-of-domain audio: Causes and solutions," in *SLT Workshop*, 2021.

[277] D. Kahneman, *Thinking, fast and slow*. Macmillan, 2011.

[278] K. E. Stanovich and R. F. West, "Individual differences in reasoning: Implications for the rationality debate?" *Behavioral and brain sciences*, vol. 23, no. 5, pp. 645–665, 2000.

[279] R. Levy, "Expectation-based syntactic comprehension," *Cognition*, vol. 106, no. 3, pp. 1126–1177, 2008.

[280] ——, "Integrating surprisal and uncertain-input models in online sentence comprehension: formal techniques and empirical results," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*. Association for Computational Linguistics, 2011, pp. 1055–1065.

[281] I. F. Monsalve, S. L. Frank, and G. Vigliocco, "Lexical surprisal as a general predictor of reading time," in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2012, pp. 398–408.

[282] J. L. Hoover, M. Sonderegger, S. T. Piantadosi, and T. J. O'Donnell, "The plausibility of sampling as an algorithmic theory of sentence processing," 2022.

[283] D. Amodei and D. Hernandez, "AI and Compute," *OpenAI blog https://blog.openai. com/ai-and-compute*, vol. 31, 2018.

[284] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing*, 2019.

[285] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," *57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

[286] O. Sharir, B. Peleg, and Y. Shoham, "The Cost of Training NLP Models: A Concise Overview," *arXiv preprint arXiv:2004.08900*, 2020.

[287] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[288] A. Davis and I. Arel, "Low-rank approximations for conditional feedforward computation in deep neural networks," *arXiv preprint arXiv:1312.4461*, 2013.

[289] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[290] S. Tan and K. C. Sim, "Towards implicit complexity control using variable-depth deep neural networks for automatic speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5965–5969.

[291] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "DeeBERT: Dynamic Early Exiting for Accelerating BERT Inference," *ACL*, 2020.

[292] D. Stamoulis, T.-W. Chin, A. K. Prakash, H. Fang, S. Sajja, M. Bognar, and D. Marculescu, "Designing adaptive neural networks for energy-constrained image classification," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[293] S. Scardapane, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Why should we add early exits to neural networks?" *arXiv*, 2020.

[294] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2015, pp. 124–132.

[295] A. Gruenstein, R. Alvarez, C. Thornton, and M. Ghodrat, "A cascade architecture for keyword spotting on mobile devices," *NeurIPS - Workshop on Machine Learning on the Phone and other Consumer Devices*, 2017.

[296] H. Tann, S. Hashemi, and S. Reda, "Flexible deep neural network processing," *arXiv preprint arXiv:1801.07353*, 2018.

[297] D. J. Pagliari, E. Macii, and M. Poncino, "Dynamic bit-width reconfiguration for energy-efficient deep learning hardware," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[298] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[299] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.

[300] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.

[301] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, "Conditional computation in neural networks for faster models," *ICLR*, 2016.

[302] N. Léonard, "Distributed conditional computation," 2015.

[303] A. W. Yu, H. Lee, and Q. V. Le, "Learning to skim text," *arXiv preprint arXiv:1704.06877*, 2017.

[304] A. Odena, D. Lawson, and C. Olah, "Changing model behavior at test-time using reinforcement learning," *ICLR Workshop*, 2017.

[305] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[306] J. Dean, "The deep learning revolution and its implications for computer architecture and chip design," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 8–14.

[307] D. Eigen, M. Ranzato, and I. Sutskever, "Learning factored representations in a deep mixture of experts," *ICLR Workshop*, 2014.

[308] K. Cho and Y. Bengio, "Exponentially increasing the capacity-to-computation ratio for conditional computation in deep learning," *arXiv preprint arXiv:1406.7362*, 2014.

[309] C. Rosenbaum, T. Klinger, and M. Riemer, "Routing networks: Adaptive selection of non-linear functions for multi-task learning," *ICLR*, 2018.

[310] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori, "Adaptive neural trees," *ICML*, 2019.

[311] A. Graves, "Adaptive computation time for recurrent neural networks," *arXiv preprint arXiv:1603.08983*, 2016.

[312] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1039–1048.

[313] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *ICLR*, 2017.

[314] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang, "Skip RNN: Learning to skip state updates in recurrent neural networks," *NeurIPS Time Series Workshop*, 2017.

[315] Y. Jernite, E. Grave, A. Joulin, and T. Mikolov, "Variable computation in recurrent neural networks," *ICLR*, 2017.

[316] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama, "Adaptive neural networks for efficient inference," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 527–536.

[317] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser, "Universal transformers," *ICLR*, 2019.

[318] A. Bapna, N. Arivazhagan, and O. Firat, "Controlling computation versus quality for neural sequence models," *arXiv preprint arXiv:2002.07106*, 2020.

[319] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan, "Likelihood ratios for out-of-distribution detection," in *Advances in Neural Information Processing Systems 32*, 2019.

[320] S. J. Mielke, R. Cotterell, K. Gorman, B. Roark, and J. Eisner, "What kind of language is hard to language-model?" *ACL*, 2019.

[321] H. He, N. Peng, and P. Liang, "Pun generation with surprise," *NAACL*, 2019.

[322] K. M. Rocki, "Surprisal-driven feedback in recurrent networks," *arXiv preprint arXiv:1608.06027*, 2016.

[323] K. Rocki, T. Kornuta, and T. Maharaj, "Surprisal-driven zoneout," *NeurIPS - Continual Learning and Deep Networks Workshop*, 2016.

[324] T. Alpay, F. Abawi, and S. Wermter, "Preserving activations in recurrent neural networks based on surprisal," *Neurocomputing*, vol. 342, pp. 75–82, 2019.

[325] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *ICLR*, 2017.

[326] A. Radford, R. Jozefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment," *arXiv preprint arXiv:1704.01444*, 2017.

[327] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," *NeurIPS*, 2018.

[328] Y.-A. Chung, W.-N. Hsu, H. Tang, and J. Glass, "An unsupervised autoregressive model for speech representation learning," *Interspeech*, 2019.

[329] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using LSTMs," in *International Conference on Machine Learning*, 2015, pp. 843–852.

[330] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[331] J. J. Rissanen, "Generalized Kraft inequality and arithmetic coding," *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198–203, 1976.

[332] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[333] G. A. Frantz and R. H. Wiggins, "Design case history: Speak & Spell learns to talk," *IEEE Spectrum*, vol. 19, no. 2, pp. 45–49, 1982.

[334] D. O'Shaughnessy, "Linear predictive coding," *IEEE Potentials*, vol. 7, no. 1, pp. 29–32, 1988.

[335] Y. Huang and R. P. Rao, "Predictive coding," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 2, no. 5, pp. 580–593, 2011.

[336] A. Clark, "Whatever next? Predictive brains, situated agents, and the future of cognitive science," *Behavioral and Brain Sciences*, vol. 36, no. 3, pp. 181–204, 2013.

[337] J. Schmidhuber, "Neural sequence chunkers," 1991.

[338] R. Aljundi, P. Chakravarty, and T. Tuytelaars, "Expert gate: Lifelong learning with a network of experts," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3366–3375.

[339] F. S. Fard and T. Trappenberg, "A novel model for arbitration between planning and habitual control systems," *Frontiers in Neurorobotics*, vol. 13, p. 52, 2019.

[340] A. Goyal, Y. Bengio, and M. B. S. Levine, "The variational bandwidth bottleneck: Stochastic evaluation on an information budget," *ICLR*, 2020.

[341] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[342] H. Larochelle and I. Murray, "The neural autoregressive distribution estimator," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 29–37.

[343] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in neural information processing systems*, 2018, pp. 6571–6583.

[344] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," *NASA STI/Recon technical report n*, vol. 93, 1993.

[345] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger, "Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi." in *Interspeech*, 2017, pp. 498–502.

[346] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *NeurIPS 2014 Deep Learning and Representation Learning Workshop*, 2014.

[347] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[348] H. Sak, A. Senior, K. Rao, and F. Beaufays, "Fast and accurate recurrent neural network acoustic models for speech recognition," *Interspeech*, 2015.

[349] F. Lieder, A. Shenhav, S. Musslick, and T. L. Griffiths, "Rational metareasoning and the plasticity of cognitive control," *PLoS computational biology*, vol. 14, no. 4, p. e1006043, 2018.

[350] A. R. Otto, S. J. Gershman, A. B. Markman, and N. D. Daw, "The curse of planning: dissecting multiple reinforcement-learning systems by taxing the central executive," *Psychological science*, vol. 24, no. 5, pp. 751–761, 2013.

[351] A. Newell, "A tutorial on speech understanding systems," *Speech recognition*, pp. 4–54, 1975.

[352] D. R. Reddy, *Speech recognition: invited papers presented at the 1974 IEEE symposium*. Elsevier, 1975.

[353] D. H. Klatt, "Review of the ARPA speech understanding project," *The Journal of the Acoustical Society of America*, vol. 62, no. 6, pp. 1345–1366, 1977.

[354] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty," *ACM Computing Surveys (CSUR)*, vol. 12, no. 2, pp. 213–253, 1980.

[355] A. Coucke, A. Saade, A. Ball, T. Bluche, A. Caulier, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril *et al.*, "Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces," *arXiv preprint arXiv:1805.10190*, 2018.

[356] B. Ons, J. F. Gemmeke, and H. Van hamme, "The self-taught vocal interface," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2014, no. 1, pp. 1–16, 2014.

[357] Y.-Y. Wang, A. Acero, and C. Chelba, "Is word error rate a good indicator for spoken language understanding accuracy," in *2003 IEEE workshop on automatic speech recognition and understanding (IEEE Cat. No. 03EX721)*. IEEE, 2003, pp. 577–582.

[358] E. Shriberg, A. Stolcke, D. Hakkani-Tür, and G. Tür, "Prosody-based automatic segmentation of speech into sentences and topics," *Speech communication*, vol. 32, no. 1-2, pp. 127–154, 2000.

[359] T. Tran, *Neural models for integrating prosody in spoken language understanding*. University of Washington, 2020.

[360] T. Tran and M. Ostendorf, "Assessing the use of prosody in constituency parsing of imperfect transcripts," *arXiv preprint arXiv:2106.07794*, 2021.

[361] E. Salesky, D. Etter, and M. Post, "Robust open-vocabulary translation from visual text representations," *EMNLP*, 2021.

[362] V. Stouten, K. Demuynck, and H. Van hamme, "Discovering phone patterns in spoken utterances by non-negative matrix factorization," *IEEE Signal Processing Letters*, vol. 15, pp. 131–134, 2008.

[363] L. ten Bosch, L. Boves, H. Van hamme, and R. K. Moore, "A computational model of language acquisition: the emergence of words," *Fundamenta Informaticae*, vol. 90, no. 3, pp. 229–249, 2009.

[364] V. Renkens, V. Tomar, and H. Van hamme, "Incrementally learn the relevance of words in a dictionary for spoken language acquisition," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 144–150.

[365] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[366] J. Poncelet, V. Renkens, and H. Van hamme, "Low resource end-to-end spoken language understanding with capsule networks," *Computer Speech & Language*, vol. 66, p. 101142, 2021.

[367] D. Serdyuk, Y. Wang, C. Fuegen, A. Kumar, B. Liu, and Y. Bengio, "Towards end-to-end spoken language understanding," *ICASSP*, 2018.

[368] S. Thomas, H.-K. J. Kuo, G. Saon, Z. Tüske, B. Kingsbury, G. Kurata, Z. Kons, and R. Hoory, "RNN transducer models for spoken language understanding," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7493–7497.

[369] Y. Qian, R. Ubale, V. Ramanarayanan, and P. Lange, "Exploring ASR-free end-to-end modeling to improve spoken language understanding in a cloud-based dialog system," *ASRU*, 2017.

[370] Y.-P. Chen, R. Price, and S. Bangalore, "Spoken language understanding without speech recognition," *ICASSP*, 2018.

[371] L. Lugosch, M. Ravanelli, P. Ignoto, V. S. Tomar, and Y. Bengio, "Speech model pre-training for end-to-end spoken language understanding," *Interspeech*, 2019.

[372] N. Tomashenko, A. Caubrière, and Y. Estève, "Investigating adaptation and transfer learning for end-to-end spoken language understanding from speech," in *Interspeech 2019*. ISCA, 2019, pp. 824–828.

[373] S. Bhosale, I. Sheikh, S. H. Dumpala, and S. K. Kopparapu, "End-to-end spoken language understanding: Bootstrapping in low resource scenarios." in *Interspeech*, 2019, pp. 1188–1192.

[374] A. Caubrière, N. Tomashenko, A. Laurent, E. Morin, N. Camelin, and Y. Estève, "Curriculum-based transfer learning for an effective end-to-end spoken language understanding and domain portability," *Interspeech*, 2019.

[375] P. Wang, L. Wei, Y. Cao, J. Xie, and Z. Nie, "Large-scale unsupervised pre-training for end-to-end spoken language understanding," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7999–8003.

[376] M. Kim, G. Kim, S.-W. Lee, and J.-W. Ha, "St-bert: Cross-modal language model pre-training for end-to-end spoken language understanding," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7478–7482.

[377] Y. Qian, X. Bianv, Y. Shi, N. Kanda, L. Shen, Z. Xiao, and M. Zeng, "Speech-language pre-training for end-to-end spoken language understanding," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7458–7462.

[378] S. Seo, D. Kwak, and B. Lee, "Integration of pre-trained networks with continuous token interface for end-to-end spoken language understanding," *arXiv:2104.07253*, 2021.

[379] A. Bapna, Y.-a. Chung, N. Wu, A. Gulati, Y. Jia, J. H. Clark, M. Johnson, J. Riesa, A. Conneau, and Y. Zhang, "Slam: A unified encoder for speech and language modeling via speech-text joint pre-training," *arXiv preprint arXiv:2110.10329*, 2021.

[380] J. Li, R. Gadde, B. Ginsburg, and V. Lavrukhin, "Training neural speech recognition systems with synthetic speech augmentation," *arXiv preprint arXiv:1811.00707*, 2018.

[381] A. Rosenberg, Y. Zhang, B. Ramabhadran, Y. Jia, P. Moreno, Y. Wu, and Z. Wu, "Speech recognition with augmented synthesized speech," *arXiv preprint arXiv:1909.11699*, 2019.

[382] C. Peyser, H. Zhang, T. N. Sainath, and Z. Wu, "Improving performance of end-to-end ASR on numeric sequences," *Interspeech*, 2019.

[383] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," *ICASSP*, 2019.

[384] D. Kulshreshtha, R. Belfer, I. V. Serban, and S. Reddy, "Back-training excels self-training at unsupervised domain adaptation of question generation and passage retrieval," *EMNLP*, 2021.

[385] A. Poncelas, D. Shterionov, A. Way, G. M. d. B. Wenniger, and P. Passban, "Investigating backtranslation in neural machine translation," *Conference of the European Association for Machine Translation (EAMT)*, 2018.

[386] M. Wiesner, A. Renduchintala, S. Watanabe, C. Liu, N. Dehak, and S. Khudanpur, "Pretraining by backtranslation for end-to-end ASR in low-resource settings," *arXiv preprint arXiv:1812.03919*, 2018.

[387] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *European Conference on Artificial Life*. Springer, 1995, pp. 704–720.

[388] Y. Taigman, L. Wolf, A. Polyak, and E. Nachmani, "VoiceLoop: Voice fitting and synthesis via a phonological loop," *ICLR*, 2018.

[389] C. Veaux, J. Yamagishi, K. MacDonald *et al.*, "CSTR VCTK corpus: English multi-speaker corpus for CSTR voice cloning toolkit," *University of Edinburgh. The Centre for Speech Technology Research (CSTR)*, 2017.

[390] A. Saade, A. Coucke, A. Caulier, J. Dureau, A. Ball, T. Bluche, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril, and M. Primet, "Spoken language understanding on the edge," *NeurIPS Workshop on Energy Efficient Machine Learning and Cognitive Computing*, 2019.

[391] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "LibriSpeech: an ASR corpus based on public domain audio books," *ICASSP*, 2015.

[392] D. Hakkani-Tür, F. Béchet, G. Riccardi, and G. Tur, "Beyond ASR 1-best: Using word confusion networks in spoken language understanding," *Computer Speech & Language*, vol. 20, no. 4, pp. 495–514, 2006.

[393] C. Wang, S. Dai, Y. Wang, F. Yang, M. Qiu, K. Chen, W. Zhou, and J. Huang, "Arobert: An asr robust pre-trained language model for spoken language understanding," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 1207–1218, 2022.

[394] E. Bastianelli, A. Vanzo, P. Swietojanski, and V. Rieser, "SLURP: A Spoken Language Understanding Resource Package," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[395] R. G. Leonard and G. Doddington, "Tidigits ldc93s10," *Web Download. Philadelphia: Linguistic Data Consortium*, 1993.

[396] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[397] Z. Jackson, C. Souza, J. Flaks, Y. Pan, H. Nicolas, and A. Thite, "Jakobovski/free-spoken-digit-dataset: v1.0.8," 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1342401

[398] A. Tamkin, M. Wu, and N. Goodman, "Viewmaker networks: Learning views for unsupervised representation learning," *ICLR*, 2021.

[399] Y.-A. Chung, C. Zhu, and M. Zeng, "Semi-supervised speech-language joint pre-training for spoken language understanding," *NAACL*, 2021.

[400] S. Pascual, M. Ravanelli, J. Serra, A. Bonafonte, and Y. Bengio, "Learning problem-agnostic speech representations from multiple self-supervised tasks," *Interspeech*, 2019.

[401] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common Voice: A massively-multilingual speech corpus," *LREC*, 2020.

[402] A. Kannan, Y. Wu, P. Nguyen, T. N. Sainath, Z. Chen, and R. Prabhavalkar, "An analysis of incorporating an external language model into a sequence-to-sequence model," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 1–5828.

[403] Y. Huang, H.-K. Kuo, S. Thomas, Z. Kons, K. Audhkhasi, B. Kingsbury, R. Hoory, and M. Picheny, "Leveraging unpaired text data for training end-to-end speech-to-intent systems," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7984–7988.

[404] B. Sharma, M. Madhavi, and H. Li, "Leveraging acoustic and linguistic embeddings from pretrained speech and language models for intent classification," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 7498–7502.

[405] Y.-A. Chung, C. Zhu, and M. Zeng, "SPLAT: Speech-Language Joint Pre-Training for Spoken Language Understanding," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 1897–1907.

[406] A. Ovadya and J. Whittlestone, "Reducing malicious use of synthetic media research: Considerations and potential release practices for machine learning," *arXiv preprint arXiv:1907.11274*, 2019.

[407] Y. Jia, Y. Zhang, R. J. Weiss, Q. Wang, J. Shen, F. Ren, Z. Chen, P. Nguyen, R. Pang, I. L. Moreno *et al.*, "Transfer learning from speaker verification to multispeaker text-to-speech synthesis," *NeurIPS*, 2018.

[408] C. Jemine, "Master thesis: Real-time voice cloning," 2019.

[409] M. Mhiri, S. Myer, and V. S. Tomar, "A low latency ASR-free end to end spoken language understanding system," *Interspeech*, 2020.

[410] N. Potdar, A. R. Avila, C. Xing, D. Wang, Y. Cao, and X. Chen, "A streaming end-to-end framework for spoken language understanding," *IJCAI*, 2021.

[411] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, "The HTK book," *Cambridge university engineering department*, vol. 3, no. 175, p. 12, 2002.

[412] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio, "Batch-normalized joint training for dnn-based distant speech recognition," in *2016 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2016, pp. 28–34.

[413] H. Bu, J. Du, X. Na, B. Wu, and H. Zheng, "Aishell-1: An open-source mandarin speech corpus and a speech recognition baseline," in *2017 20th conference of the oriental chapter of the international coordinating committee on speech databases and speech I/O systems and assessment (O-COCOSDA)*. IEEE, 2017, pp. 1–5.

[414] B. Laufer, "The lexical profile of second language writing: Does it change over time?" *RELC journal*, vol. 25, no. 2, pp. 21–33, 1994.

[415] K. Wolfe-Quintero, S. Inagaki, and H.-Y. Kim, *Second language development in writing: Measures of fluency, accuracy, & complexity*. University of Hawaii Press, 1998, no. 17.

[416] W. Johnson, "Studies in language behavior: A program of research," *Psychological Monographs*, vol. 56, no. 2, pp. 1–15, 1944.

[417] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[418] T. Kollar, D. Berry, L. Stuart, K. Owczarzak, T. Chung, L. Mathias, M. Kayser, B. Snow, and S. Matsoukas, "The Alexa meaning representation language," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*, 2018, pp. 177–184.

[419] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[420] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[421] J. Sygnowski and H. Michalewski, "Learning from the memory of atari 2600," in *Workshop on Computer Games, International Workshop on General Intelligence in Game-Playing Agents*. Springer, 2017, pp. 71–85.

[422] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, 2016.

[423] S. R. Branavan, H. Chen, L. Zettlemoyer, and R. Barzilay, "Reinforcement learning for mapping instructions to actions," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2009, pp. 82–90.

[424] S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, 2011, pp. 1507–1514.

[425] D. Chen and R. Mooney, "Learning to interpret natural language navigation instructions from observations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, 2011, pp. 859–865.

[426] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio, "BabyAI: A platform to study the sample efficiency of grounded language learning," *arXiv preprint arXiv:1810.08272*, 2018.

[427] M. Sulír and J. Porubän, "Natural mapping between voice commands and APIs," *Open Computer Science*, vol. 11, no. 1, pp. 135–145, 2021.

[428] A. Szlam, J. Gray, K. Srinet, Y. Jernite, A. Joulin, G. Synnaeve, D. Kiela, H. Yu, Z. Chen, S. Goyal *et al.*, "Why build an assistant in minecraft?" *arXiv preprint arXiv:1907.09273*, 2019.

[429] V. Rieser and O. Lemon, "Natural language generation as planning under uncertainty for spoken dialogue systems," in *Empirical methods in natural language generation*. Springer, 2009, pp. 105–120.

[430] J. D. Williams and S. Young, "Partially observable markov decision processes for spoken dialog systems," *Computer Speech & Language*, vol. 21, no. 2, pp. 393–422, 2007.

[431] V. Rieser and O. Lemon, *Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation*. Springer Science & Business Media, 2011.

[432] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[433] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[434] S. Yuan, H. Zhao, S. Zhao, J. Leng, Y. Liang, X. Wang, J. Yu, X. Lv, Z. Shao, J. He *et al.*, "A roadmap for big model," *arXiv preprint arXiv:2203.14101*, 2022.

[435] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," https://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[436] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences," *ICLR*, 2018.

[437] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[438] Noam Shazeer, "Neural Language Models: Bigger is Better," https://www.youtube.com/watch?v=9P_VAMyb-7k.

[439] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[440] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *arXiv preprint arXiv:2203.02155*, 2022.

[441] P. C. Humphreys, D. Raposo, T. Pohlen, G. Thornton, R. Chhaparia, A. Muldal, J. Abramson, P. Georgiev, A. Santoro, and T. Lillicrap, "A data-driven approach for learning to control computers," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9466–9482.

[442] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang, "Reinforcement learning on web interfaces using workflow-guided exploration," *arXiv preprint arXiv:1802.08802*, 2018.

[443] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 991–1002.

[444] C. Lynch and P. Sermanet, "Language conditioned imitation learning over unstructured data," *arXiv preprint arXiv:2005.07648*, 2020.

[445] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet, "Learning latent plans from play," in *Conference on robot learning*. PMLR, 2020, pp. 1113–1132.

[446] C. Lynch, A. Wahid, J. Tompson, T. Ding, J. Betker, R. Baruch, T. Armstrong, and P. Florence, "Interactive language: Talking to robots in real time," *arXiv preprint arXiv:2210.06407*, 2022.

[447] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8748–8763.

[448] D. Toyama, P. Hamel, A. Gergely, G. Comanici, A. Glaese, Z. Ahmed, T. Jackson, S. Mourad, and D. Precup, "AndroidEnv: a reinforcement learning platform for Android," *arXiv preprint arXiv:2105.13231*, 2021.

[449] J. Albus, "Brains, behavior, and robotics," *McGraw-Hill*, 1981.

[450] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, vol. 8, 1995.

[451] S. Whiteson, "Adaptive tile coding for value function approximation," Tech. Rep., 2007.

[452] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[453] R. K. Srivastava, P. Shyam, F. Mutz, W. Jaśkowski, and J. Schmidhuber, "Training agents using upside-down reinforcement learning," *arXiv preprint arXiv:1912.02877*, 2019.

[454] H. Xu, F. Jia, S. Majumdar, H. Huang, S. Watanabe, and B. Ginsburg, "Efficient sequence transduction by jointly predicting tokens and durations," *arXiv preprint arXiv:2304.06795*, 2023.

[455] J. Kim, J. Kong, and J. Son, "Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5530–5540.

[456] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.