Artificial Neural Networks for Analysis of Coherent X-Ray Diffraction Images

Daniel Abarbanel

Physics and Astronomy, McGill University, Montréal

August 6, 2018

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science.

Abstract

In the past decade, artificial neural networks have been demonstrated to perform complex image recognition tasks. This is due to their ability to mathematically distill raw image information into a representation that encodes higher order concepts. As a result of these successes, neural networks are being increasingly applied to the analysis of imaging data in the natural sciences. Here, we present an application of neural networks to diffraction images produced via X-ray photon correlation spectroscopy (XPCS) experiments. The light collection apparatus for these experiments is an area detector comprised of an array of charge-coupled devices (CCDs). The CCDs collect electrons excited by a monochromatic source of X-ray photons. The digital signal returned by the detector is proportional to the amount of charge collected, and thus proportional to the energy of the incident photons. Determining the location of each photon allows for calculation of the image contrast, which is useful for statistical analyses that can elucidate the structural and dynamical properties of a system being measured. However, the excited electrons have a spatial extent, and in many cases this can result in one photon activating multiple adjacent CCDs. Multiple photons incident on the detector array in the same vicinity create a connected region of activated pixels known as a droplet. A droplet contains additive interference that can further obscure the original locations of the incident photons. In this thesis, we demonstrate that neural networks can be used to process raw area detector images and determine the incident locations of the photons to a satisfactory degree of certainty. A feed-forward neural network classifier was designed and trained to systematically scan a droplet and determine a discrete probability distribution for the number of photons that have been incident within the area of each CCD. Bayesian inference was performed on these predictions to determine the most likely configuration of photons within the droplet as a whole. Count-rate histograms were determined from artificially generated images containing many droplets. These histograms were fit to a contrast parameter and were found to be in agreement with the true underlying values. We have thus demonstrated a promising method for accurately determining the contrast from experimental XPCS images.

Résumé

Au cours de la dernière décennie, il a été démontré que les réseaux de neurones artificiels peuvent accomplir des tâches complexes de reconnaissance d'images du fait de leur capacité à distiller mathématiquement, à partir de données brutes, des concepts d'ordre supérieur. Par conséquent, les réseaux de neurones sont de plus en plus appliqués, en sciences naturelles, à l'analyse de données d'imagerie. Appuyés par la technique des réseaux de neurones, nous présentons une méthode d'analyse d'images de diffractions produites par des expériences de spectroscopie de corrélation de photons X (XPCS, pour "X-ray photon correlation spectroscopy"). Dans ces expériences, les photons sont dénombrés par un détecteur de surface constitué d'un réseau de dispositifs à couplage de charge (CCD, pour "charge-coupled devices"). Les CCD détectent des électrons excités par une source monochromatique de photons X; le signal numérique enregistré par chaque CCD est proportionnel à la quantité de charge détectée, et donc proportionnel à l'énergie des photons incidents. À partir de l'emplacement d'incidence de chaque photon, le contraste de l'image peut être calculé; ce contraste peut ensuite être utilisé dans des analyses statistiques permettant d'élucider les propriétés structurelles et dynamiques du système investigué. Cependant, les électrons excités ont une certaine étendue spatiale, ce qui peut entraîner l'activation par un seul photon de plusieurs CCD adjacents. Lorsque plusieurs photons incidents sur le détecteur stimulent un groupe de CCD adjacents, ils créent une région connectée de pixels activés connue sous le nom de "gouttelette". Une gouttelette contient des interférences additives qui peuvent obscurcir davantage les emplacements d'origine des photons incidents. Dans cette thèse, nous démontrons que les réseaux de neurones peuvent être utilisés pour analyser des images brutes obtenue par des expériences de XPCS afin de déterminer les emplacements d'incidence de photons avec un degré de certitude satisfaisant. Un réseau de neurones en aval supervisé a été conçu et formé pour scanner systématiquement une gouttelette et déterminer une distribution de probabilité discrète pour le nombre de photons incidents dans chaque CCD. L'inférence bayésienne a par la suite été utilisée pour déterminer, à partir de ces prédictions, la configuration la plus probable des photons dans l'ensemble de la gouttelette. Les histogrammes de taux de comptage ont été déterminés à partir d'images de gouttelettes générées artificiellement. Ces histogrammes ont été ajustés à un paramètre de contraste et se sont avérés être en accord avec les véritables valeurs sous-jacentes. Nous avons ainsi démontré une méthode prometteuse pour déterminer avec précision le contraste d'images obtenues par des expériences de XPCS.

Preface and Statement of Originality

All the writing in this thesis is the sole work of the main author, Daniel Abarbanel (DA). In addition, all the figures were designed and generated by DA, unless explicitly stated otherwise. In Chapter 4, Nicolas Miro Fortier (NMF) provided programming assistance in training the mixture density network (Chapter 4.2.3). The remainder of the programming is the sole work of DA.

There are a number of original results in this thesis. They are summarized below:

- In Chapter 4.1, DA shows that the spatial probability distribution of electrons excited by an incident photon on a charge-coupled device (CCD) can be effectively modelled as Gaussian. This result is used to generate training data for a neural network classifier algorithm.
- In Chapter 4.2.1, DA designs and trains a neural network classifier that outputs discrete probability distributions for the number of photons that are incident within a CCD.
- In Chapter 4.2.2, DA develops a Bayesian inference method that is used to improve the accuracy of the neural network classifier designed in Chapter 4.2.1.
- In Chapter 4.2.3, DA and NMF design and train a mixture density network that outputs continuous probability distributions for the spatial coordinates of photons that are incident within a CCD.
- In Chapter 4.3, DA designs a method for generating a rich and balanced training data set using Sobol-distributed vectors. This data set is used to train the classifier in Chapter 4.2.1.
- In Chapter 4.4, DA uses the aggregated predictions of the neural network classifier to determine the speckle contrasts of a series of artificial area detector images.

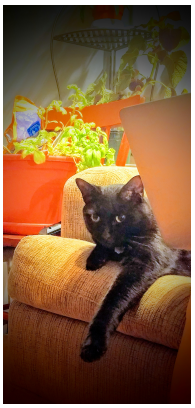
Acknowledgements

I would like to thank my supervisor Dr. Hong Guo for bringing machine learning and data science into his research group, and encouraging me to climb its hills and navigate its saddle points in search of the perfect valley. Thanks are also in order for providing accelerated computational resources on the ComputeCanada cluster. My utmost gratitude is due to Dr. Mark Sutton for putting this project into my hands, and offering experimental data, advice, and resources for various droplet algorithms that were implemented in this thesis. Additional thanks are due to Professor Guo and Professor Sutton for their editorial work on this thesis. Thanks to Raphaël Prentki for checking my math on my more stupider days, and for translating where necessary. Thanks to Nicolas Fortier for his help in training the mixture density network. Thanks to Aaron Mascaro, Matt Heffernan, Jessica Churchill, Alexander McDonald and Andreas Spielhofer for making 227 one of the best places to work and argue. Thanks to Alessandro Ambler, Kevin Murray, Zachary Hockenbery and Felix Fehse for making me look forward to every lunch/coffee break.

I would like to thank McGill University for hosting me during this degree. I would also like to thank the staff at ComputeCanada, and specifically to those that worked with the Graham and Guillimin servers. Your patience was appreciated.

Thank you to my family, and to Iara, for your part in getting me here.

And lastly, to Peppe, for all his calming languor.



Contents

1 Introduction

Machine learning is a field of computer science and statistics whose aim is to develop problem solving algorithms that are not explicitly hard-coded. Rather, the algorithms are designed to improve their performance on a given task by processing a source of data [1]. What *is* explicitly programmed is the structure of the machine learning algorithm, and the methods by which it can process data and adjust its own parameters. Machine learning algorithms enjoy a particular advantage at performing tasks that cannot be defined formally, such as recognizing an object in a picture, understanding spoken language, or navigating obstacles [2–4]. The key insight is that such tasks are relatively easy for humans to perform, and ultimately relate back to the way information is processed by, and affects the structure of, the brain. This insight motivated the inception of *artificial neural networks*, computational systems of nodes with interconnections that facilitate information flow. This type of structure was historically inspired by the biological brain, in which the neuron represents a node, and the synapses that connect neurons represent the connections between nodes [5]. One of the first models to be constructed with this philosophy was in the 1940s with the McCulloch-Pitts neuron, a simple binary predictor that would "activate" or output 1, if the weighted sum of a set of data elements was above a given threshold, and output 0 otherwise [6]:

$$f(A_1, A_2, \ldots, A_n; W_1, W_2, \ldots, W_n) = \Theta\left(\left[\sum_{i=1}^{n} W_i A_i\right] - \phi\right) \tag{1.1}$$

Here, ϕ is a chosen threshold value and Θ is the Heaviside function. The input data elements A_i are known as *features*, and the values W_i are known as the *weights*. The model is linear, and the weights were determined via trial and error [7]. In the late 1950s, the first *trainable* neuron known as the *perceptron* was developed by F. Rosenblatt [8]. The training of the perceptron used a gradient descent method similar to the ones used today [7]. Due to the inherent linearity of these early models, their inability to learn more complex

functions caused the field to stagnate until the mid 1980s, when a new philosophy known as *connectionism* gained traction. The connectionist approach was to construct learning models out of multiple artificial neurons, and introduce non-linear elements to lift the restrictions brought on my purely linear models [9]. It was during this time that the backpropagation algorithm was developed, which remains the most popular algorithm for training neural network models [10]. Through the 1990s, the field of machine learning expanded beyond the biological underpinning of its predecessors. It was at this time that the focus shifted to the broader goal of developing models with high degrees of freedom, whose structure permitted optimization from data via an iterative procedure. One such model is the Support Vector Machine (SVM), which aims to classify data into categories by generating a feature space and creating distinct regions whose borders are defined by optimized hyperplanes [11]. Neural networks went through a decline in popularity until a series of breakthroughs in training methodology in the mid 2000s [12]. The crux of the resurgence centred around *representation*: neural networks with many layers could now be reliably trained to take features from a given layer and combine them to generate new features with higher complexity. Neural networks that can filter and transform raw data through many representations in this way are known as *deep* neural networks [13]. One can illustrate this methodology by considering a deep neural network that is able to recognize the difference between hand-drawn circles and squares. The input layer is a 2D matrix of pixel values. The second layer processes the input image and generates data that represents the positions and angles of pen-strokes that occur in the original picture. The remaining layers process this information into a feature space in which data originally corresponding to circles do not overlap with data corresponding to squares, making it a successful classifier. The use of neural networks to generate predictive models using a complexity-based hierarchy of representations is known as *deep learning*, and is currently a topic of major theoretical interest [7]. In addition to research interest, deep learning models have empirically demonstrated their usefulness in fields such as speech and image recognition, language translation, game strategies, fraud

detection, and finance [14–19].

Machine learning has garnered significant interest in the sciences. It is particularly helpful in cases where there is an abundance of raw scientific data that contain unknown, implicit features with predictive power. Examples in medical science include convolutional models for cancer detection in skin histology images and spinal fracture detection from X-ray images [20, 21]. In chemistry, machine learning has been used to predict atomization energies of organic molecules, and map phase diagrams of ternary systems [22, 23]. Biological applications of deep learning include DNA sequence analysis and prediction of mutation effects [24, 25]. In physics, machine learning has a wide variety of applications including the prediction of phase transitions in Ising systems, quantum-mechanics based molecular dynamics, classification of galaxy morphologies, band gap prediction for inorganic compounds, and analysis of diffraction images [26–30]. It is on this last note that we will transition to a brief overview of diffraction experiments in physics, which are relevant to the project in this thesis.

Diffraction describes the phenomena of electromagnetic waves interacting with matter and other waves. The first scientific study of light diffraction was made by Francesco Maria Grimaldi in the mid 17^{th} century [31]. His studies were made public posthumously in 1665, and were immediately pondered by influential physicists such as Isaac Newton, Robert Hooke, and Christiaan Huygens [32]. Despite Hooke and Huygens' treatment of light as a wave propagating through a medium known as the *aether*, Newton advocated for a particle, or *corpuscular*, theory of light in his influential treatise *Opticks*, published in 1704 [33]. Nearly a century layer, Thomas Young's famous double-slit experiment presented serious criticisms of Newton's corpuscular theory, and gave legitimacy to a wave-like interpretation [34]. In the late 19^{th} century, Michelson and Morley's interferometry experiment highlighted the major deficiencies of aether theories. Einstein's subsequent work on the photoelectric effect gave birth to the modern interpretation of light as behaving like a particle and a wave; the details

of an experiment inform which interpretation is more appropriate to explain the results [35].

The first X-ray diffraction experiment was performed by Max von Laue, Walter Friedrich and Paul Knipping in 1912 [36]. In that same year, W. H. and W. L. Bragg published their famous law of diffraction within crystal layers, which earned them a Nobel prize in 1915 [37]. In 1916-17, Peter Debye, Paul Scherrer and Albert Hull developed a way to characterize crystals in the form of randomly oriented powders via diffraction measurements of incident X-ray light. This influential method, now known as *X-ray powder diffraction*, initiated a wave of experimental and theoretical research over the next 50 years [36]. X-ray diffraction experiments continue to help determine the structural properties of crystals such as atomic composition, space group, lattice constants, and degrees of impurity [38].

The invention of the laser in 1960 by Theodore H. Maiman, based on theoretical research by Charles H. Townes and Arthur L. Schawlow, allowed for a new reliable source of *coherent* light that could be widely used in diffraction experiments [39–41]. A diffraction image produced via the interaction of coherent light with a material produces an intensity profile known as a *speckle pattern*. Speckle patterns contain phase information not present in X-ray powder diffraction data [42]. While the light intensity remains the only measurable quantity in coherent diffraction experiments, the full phase information can in some circumstances be extracted, providing a rich structural description of the material being measured [43]. Coherent diffraction imaging was initially restricted to light sources in the visible spectrum which did not permit the study of materials on length scales smaller that a few hundred nanometres [44]. In 1991, Mark Sutton *et al.* demonstrated that coherent diffraction could be performed using an X-ray light source, opening the door to studies of structure and dynamics on the atomic scale [42].

The project in this thesis is to use machine learning techniques to determine the speckle *contrast* of coherent X-ray scattering images. The images are measured with an area detector, in this case consisting of an array of charge-coupled devices (CCDs). Each CCD contributes

one "pixel" to the speckle image. Contrast is a parameter that varies with the timescale of an XPCS measurement, and can thus be used to determine the dynamical properties of a system. Speckle contrast can be calculated using a histogram of area detector pixels where each bin corresponds to the integer number of photons that have been incident on a pixel. Due to the fact that CCD-based area detectors measure photons indirectly via the collection of excited charge, multiple photons incident on the area detector in close proximity will cause interference in the underlying CCD measurements. This interference can conceal the underlying photon counts of each pixel, and thus conceal the contrast.

The main results of this thesis are, in summary:

- A model was developed to describe the spatial probability distribution of electrons excited by a photon incident on a CCD. This model was used to generate artificial speckle images.
- A neural network classifier was designed to read CCD measurements from a subset of the area detector, and predict the number of photons incident on each pixel. The neural network was trained on artificial data produced by our model.
- The classifier could reliably determine the number of photon counts per pixel, and thus produce accurate histograms of photon counts for full speckle images. This in turn resulted in an accurate determination of the speckle contrast.
- In addition, a mixture density network was trained to determine probability distributions for the spatial coordinates of the incident photons within the pixels that were determined by the classifier.

These results will be published in *Artificial Neural Networks for the Determination of Speckle Contrast in Split-Pulse XPCS Measurements* by Abarbanel *et al.* [45].

The remainder of this document will be split into three main chapters: Chapter 2 will provide an overview of the machine learning techniques that are relevant to the project. Chapter

3 will describe coherent diffraction imaging techniques in more detail, and provide an in-depth description of the project. Finally, Chapter 4 will present the machine learning based research and methodology, and give detailed results.

2 Review of Machine Learning Techniques

This chapter will provide a cursory overview of the components that go into designing a machine learning algorithm. There will be an emphasis on the components of *neural networks* as they were the main tool used for analysis in Chapter 4.

2.1 Loss Functions

For supervised learning algorithms, a prediction by the algorithm is compared to an available answer via a measure of success called a *loss function*, $\mathcal{L}$ [46]. The training objective is to find a set of optimal internal variables $\boldsymbol{\theta}$ that minimize the loss function over all the available data. Here, $\mathbf{X}$ denotes a data set of features, and $\mathbf{y}$ denotes a data set of labels that are related to $\mathbf{X}$ via a function $g(\mathbf{X})$. Henceforth, a single instance of data from the features $\mathbf{X}$ will be denoted $\tilde{\mathbf{X}}$ and similarly for $\tilde{\mathbf{y}}$. Following the convention of Goodfellow *et al.*, a ML estimator for a relationship $\mathbf{y} = g(\mathbf{X}) + \boldsymbol{\epsilon}$ will be denoted as $f(\mathbf{X}; \boldsymbol{\theta})$. Here, $\boldsymbol{\epsilon}$ represents a possible noise term in the data labels that the function f will not be trained to predict. If $\boldsymbol{\theta}$ is optimized, then f will reliably approximate $g(\mathbf{X})$ for the full data set $\mathbf{X}$. As an example, consider the case of a linear least-squares regression algorithm for N ordered pairs (x, y). The variables to be determined are the slope m and intercept b for a linear function f:

$$\mathbf{y} = (\tilde{y}_1, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_N)$$

$$\mathbf{X} = (\tilde{x}_1, \tilde{x}_2, \tilde{x}_3, \ldots, \tilde{x}_N)$$

$$\boldsymbol{\theta} \equiv (m, b)$$

$$f(x; \boldsymbol{\theta}) = mx + b$$

In this case, the loss function is given by the average of the squared residuals:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\tilde{y}_i - f(\tilde{x}_i, \boldsymbol{\theta}))^2 \tag{2.1}$$

Denoting $\boldsymbol{\theta}^*$ as the internal variables that minimize $\mathcal{L}$, the objective is to determine:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \left\langle (\tilde{y}_i - f(\tilde{x}_i; \boldsymbol{\theta}))^2 \right\rangle_i \tag{2.2}$$

$$\text{or} \qquad (m^*, b^*) = \text{argmin}_{m,b} \left\langle (\tilde{y}_i - m\tilde{x}_i - b)^2 \right\rangle_i \tag{2.3}$$

In general, for a single instance of data $(\tilde{\mathbf{X}}, \tilde{\mathbf{y}})$, the per-sample loss function is denoted L and the optimal internal variables are given by the expected value of L over all training instances:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}} \left\langle L(\tilde{\mathbf{X}}, \tilde{\mathbf{y}}; \boldsymbol{\theta}) \right\rangle_{\tilde{\mathbf{X}}, \tilde{\mathbf{y}} \in \mathbf{X}, \mathbf{y}} \tag{2.4}$$

The functional form of L depends on the format of the data and the optimization strategy. For optimization of regression tasks based on mean absolute error, the loss function is given by:

$$\mathcal{L}(\boldsymbol{\theta}) = \left\langle ||f(\tilde{\mathbf{X}}; \boldsymbol{\theta}) - \tilde{\mathbf{y}}||_1 \right\rangle_{\tilde{\mathbf{X}}, \tilde{\mathbf{y}} \in \mathbf{X}, \mathbf{y}} \tag{2.5}$$

One can construct a loss function using a combination of mean squared error and mean absolute error using the Huber loss function [47]. The per-sample version is:

$$L(\boldsymbol{\theta}) = \begin{cases} \frac{1}{2}||\tilde{\mathbf{y}} - f(\tilde{\mathbf{X}}; \boldsymbol{\theta})||^2 & ||\tilde{\mathbf{y}} - f(\tilde{\mathbf{X}}; \boldsymbol{\theta})||_1 \leq \delta \\ \delta||\tilde{\mathbf{y}} - f(\tilde{\mathbf{X}}; \boldsymbol{\theta})||_1 - \frac{1}{2}\delta^2 & ||\tilde{\mathbf{y}} - f(\tilde{\mathbf{X}}; \boldsymbol{\theta})||_1 > \delta \end{cases} \tag{2.6}$$

where δ is a tuning parameter that determines on what scale the function transitions from quadratic to linear. The choice between mean-squared and mean-absolute error is context dependent. Mean-squared error permits a smooth loss function, which is convenient for

gradient based optimization, and is more appropriate in the specific case where the errors $\boldsymbol{\epsilon}$ are Gaussian distributed. Mean-absolute error is not smooth, but more robust to outliers [48]. So far, examples of loss functions for *regression* tasks have been given. We will conclude this overview by discussing loss functions that apply to classification tasks and predicting probability distributions, which are relevant to the project in this thesis.

For classification tasks in a supervised learning paradigm using artificial neural networks, the individual training features $\tilde{\mathbf{X}}$ are coupled with a class label, encoded as an n-vector $\tilde{\mathbf{Y}}$:

$$\tilde{\mathbf{Y}} \equiv (\tilde{Y}_1, \tilde{Y}_2, \ldots, \tilde{Y}_n) \tag{2.7}$$

Here, n is the number of classes, and $\tilde{Y}_i$ corresponds to the probability that $\tilde{\mathbf{X}}$ belongs to the i^{th} class. In most cases, the classes within a training data set are not ambiguous, resulting in one element in $\tilde{\mathbf{Y}}$ being equal to unity, and the rest equal to zero. This is known as *one-hot encoding*. For example, in a binary classification problem, the labels will belong to the set $\tilde{\mathbf{Y}} \in [(1,0),(0,1)]$. In other cases, the labels can correspond to probabilities of each class, allowing each element in $\tilde{\mathbf{Y}}$ to be a decimal between 0 and 1, subject to the constraint:

$$\sum_{i=1}^{n} \tilde{Y}_i = 1 \tag{2.8}$$

An ML based classification algorithm will only receive the inputs $\mathbf{X}$, and must generate a set of values that predict the labels $\mathbf{Y}$. These outputs are called *logits*, and are encoded with the same vector form as $\mathbf{Y}$. A logit generated by an ML algorithm for one training instance will be denoted $\tilde{\mathbf{P}}$. The value of $\tilde{\mathbf{P}}$ is a function of an input feature $\tilde{\mathbf{X}}$, and the function is defined by the parameters $\boldsymbol{\theta}$ of the ML algorithm:

$$\tilde{\mathbf{P}} \equiv \tilde{\mathbf{P}}(\tilde{\mathbf{X}}; \boldsymbol{\theta}) \tag{2.9}$$

In general, the logits $\mathbf{P}$ are not one-hot encoded. This is because they represent discrete probability distributions, and allow an ML algorithm to provide a degree of confidence in its prediction. By example, one may consider a training set consisting of ill-posed features and labels. This implies that there is a subset of training instances with identical inputs $\mathbf{X}_{\text{identical}}$ but with distinct labels $\mathbf{Y}_{\text{distinct}}$. One can further consider the simplest case where $\mathbf{Y}$ encodes binary classes, and the labels $\mathbf{Y}_{\text{distinct}}$ are evenly split between $(0, 1)$ and $(1, 0)$ across the data subset. In such a case, a well trained ML algorithm will output $\tilde{\mathbf{P}} = (0.5, 0.5)$ when given a set of features belonging to $\mathbf{X}_{\text{identical}}$ [49]. In cases where the features have a one-to-one mapping with the labels, a well trained algorithm will output logits that are close to one-hot (e.g. $\tilde{\mathbf{P}} = (0, 0, 0.001, 0.999)$). The errors of a sub-optimally trained algorithm can be "smoothed out" by converting $\tilde{\mathbf{P}}$ to a one-hot label under the assumption that the class prediction corresponds to the maximal element in $\tilde{\mathbf{P}}$. For example:

$$\tilde{\mathbf{P}} = (0.1, 0.3, 0.4, 0.1) \;\rightarrow\; \tilde{\mathbf{P}}_{\text{one-hot}} = (0, 0, 1, 0) \tag{2.10}$$

The ubiquitous loss function used for classification tasks, and the tool of choice for this thesis, is *categorical cross-entropy*. For an instance of input features $\tilde{\mathbf{X}}$, a class label $\tilde{\mathbf{Y}}$ and a predicted logit $\tilde{\mathbf{P}}(\tilde{\mathbf{X}}; \boldsymbol{\theta})$, the per-sample loss function takes the form:

$$L(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \tilde{Y}_i \log\Big(\tilde{P}_i(\tilde{\mathbf{X}}; \boldsymbol{\theta})\Big) \tag{2.11}$$

In the case where $\tilde{\mathbf{Y}}$ is one-hot encoded, it is clear that the loss function is minimized for $\tilde{\mathbf{P}} = \tilde{\mathbf{Y}}$. The same holds true in the general case. For ill-posed data, the full loss function $\mathcal{L}$ is minimized when $\mathbf{P}$ encodes the probability distribution of classes for a set of identical or similar inputs. Like mean-squared error loss, the categorical cross-entropy loss function is smooth, continuous and convex, allowing for gradient based optimization methods to be employed [7].

A third family of loss functions exist for neural network architectures that output continuous probability distributions, rather than discrete ones. This lifts the restriction of regression tasks, namely, that there must be a one-to-one mapping between the features $\mathbf{X}$ and the output function f. In cases where there is a continuous or discrete range of correct outputs for one feature vector $\tilde{\mathbf{X}}$, the general strategy is to design a density function with enough degrees of freedom to capture all possible solutions. After training, the probability density corresponding to the neural network output will reflect the proportion of the class labels in the training set that have similar feature vectors [50]. In such cases, the neural network is designed to output a set of parameters $\boldsymbol{\phi}$ for a chosen likelihood function G. For example, if G is chosen to be a 1D normal distribution, then $\boldsymbol{\phi}$ is a vector containing the mean μ and standard deviation σ. The loss function of choice for these problems is called *negative log-likelihood*. For a training datum $(\tilde{\mathbf{X}}, \tilde{\mathbf{y}})$, and a neural network that outputs parameters $\boldsymbol{\phi}(\tilde{\mathbf{X}})$ for an arbitrary likelihood function $G(\tilde{\mathbf{y}}; \boldsymbol{\phi}(\tilde{\mathbf{X}}))$, the per-sample loss function is [7]:

$$L(\boldsymbol{\theta}) = -\log\Big(G(\tilde{\mathbf{y}}; \boldsymbol{\phi}(\tilde{\mathbf{X}}; \boldsymbol{\theta}))\Big) \tag{2.12}$$

Continuing with the example where G is a 1D normal distribution, the loss function reduces to weighted least squares, which is consistent with the claim that mean squared loss is appropriate for training data with Gaussian distributed error [48]:

$$G(\tilde{y}; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2(\tilde{\mathbf{X}}; \boldsymbol{\theta})}} \exp\left(-\frac{(\tilde{y} - \mu(\tilde{\mathbf{X}}; \boldsymbol{\theta}))^2}{2\sigma^2(\tilde{\mathbf{X}}; \boldsymbol{\theta})}\right) \tag{2.13}$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{2}\log(2\pi) + \left\langle \log\Big(\sigma(\tilde{\mathbf{X}}; \boldsymbol{\theta})\Big) + \frac{(\tilde{y} - \mu(\tilde{\mathbf{X}}; \boldsymbol{\theta}))^2}{2\sigma^2(\tilde{\mathbf{X}}; \boldsymbol{\theta})} \right\rangle_{\tilde{\mathbf{X}}, \tilde{y} \in \mathbf{X}, \mathbf{y}} \tag{2.14}$$

Negative log-likelihood loss functions will be used to optimize a neural network that outputs a continuous probability distribution in Chapter 4.2.3.

For neural network architectures, finding the parameters $\boldsymbol{\theta}^*$ that minimize the loss function

must be done by finding a solution to the minimization equation:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = 0 \tag{2.15}$$

In practice, this minimization is done numerically. Techniques for minimization are discussed in Chapter 2.3. Additionally, it is important to avoid local minima in the loss function that correspond to sub-optimal performance. This family of strategies is called *regularization* [7] and will be discussed further in Chapter 2.4.

2.2 Feedforward Neural Networks

The main ingredients for a feedforward neural network are:

- An input $\tilde{\mathbf{X}}$. While $\tilde{\mathbf{X}}$ can be a tensor of any order n, $(A_1, A_2, ...A_n)$, it will be represented here as a vector with $A_1 A_2 ... A_n$ elements. For simplicity, we will denote the total number of input elements a_0.

- A series of m weight matrices $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_m$. m is defined as the network *depth*. The outer dimension of each weight matrix is chosen by the user, but the inner dimension must be compatible with the vector that $\mathbf{W}$ is multiplied by. For example, $\mathbf{W}_1$ must be multiplied by the input $\mathbf{X}$, so its dimension must be (a_1, a_0), where a_0 has been previously defined, and a_1 is chosen. In general, a given weight matrix $\mathbf{W}_i$ will have dimensions (a_i, a_{i-1}).

- A series of m bias vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_m$. The length of each bias vector is equal to the chosen outer dimension of the corresponding weight matrix. Thus, $\mathbf{b}_i$ will have a_i elements.

- A series of m activation functions $u_1, u_2, \ldots, u_m$. A given function u_i must be a function of a_i variables.

The internal parameters $\boldsymbol{\theta}$ is a vector containing all elements of the weight matrices and bias vectors. The algorithm that governs the data-flow through this structure is called *forward propagation*, and is given below:

Algorithm 1: Forward Propagation

- $\mathbf{H}_0 \leftarrow \tilde{\mathbf{X}}$
- ***for*** i ***in*** $(1, 2, ..., m)$ ***do***:

  $$\mathbf{H}_i = u_i(\mathbf{b}_i + \mathbf{W}_i\mathbf{H}_{i-1})$$

  end for
- $f(\tilde{\mathbf{X}}) \equiv \mathbf{H}_m$
- Given a training label $\tilde{\mathbf{y}}$, compute per-sample loss function $L(\tilde{\mathbf{y}}, f(\tilde{\mathbf{X}}; \boldsymbol{\theta}))$

The vector denoted $\mathbf{H}_0$ is the *input layer*. The vectors $\mathbf{H}_1$ - $\mathbf{H}_{m-1}$ are called *hidden layers* and $\mathbf{H}_m$ is the *output layer*. It is through the use of the functions u_i that the feed-forward neural network acquires its non-linear behaviour. This can be seen by examining the case where all activations u within a neural network are the identity operator: the calculation reduces to a series of matrix multiplications, which is a linear system. It has been shown that introducing non-linear functions allows a neural network with at least two layers to realize the user's goal of a universal function approximator [51]. We will examine some common activation functions, with emphasis on the ones that are used in this thesis. Non-linearity is an essential ingredient for all activation functions within the hidden layers.

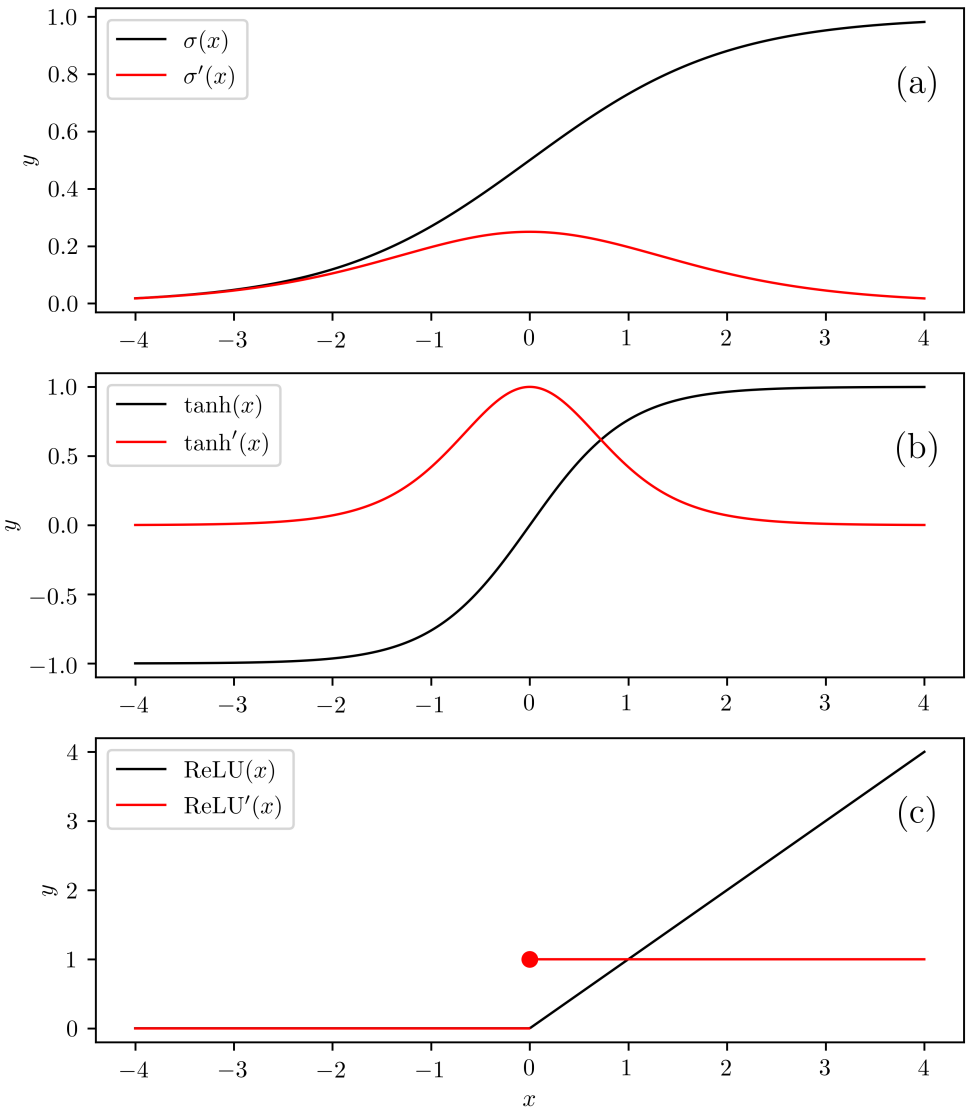


Figure 2.1: Some common activation functions for feedforward neural networks. (a): Sigmoid activation function and its derivative. (b): Tanh activation function and its derivative. (c): ReLU activation function and its derivative.

2.2.1 Sigmoid

The sigmoid activation function and its derivative are defined by [52]:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.17}$$

The sigmoid function produces values bounded by $\sigma(x) \in (0, 1)$, having a "squashing" effect on inputs with high magnitude. Sigmoid functions are used extensively as the output activation function for binary classifiers, in which the output is rounded to the nearest integer to produce a binary outcome of 0 or 1 [53]. The sigmoid function was also used as an activation for hidden layers in early machine learning research, but has fallen out of favour due to the fact that its gradient is close to zero in most of its domain, making gradient based training more difficult [54]. This is known as the *vanishing gradient problem*. Other saturating functions exist, such as tanh, given by:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.18}$$

$$\tanh'(x) = 1 - \tanh^2(x) \tag{2.19}$$

The overall shape of the tanh function is similar to the sigmoid, but the bounds are instead given by $\tanh(x) \in (-1, 1)$, and the gradient is larger for inputs close to 0.

2.2.2 Rectified Linear Unit (ReLU)

The ReLU function and its derivative are defined by:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.20}$$

$$\mathrm{ReLU}'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.21}$$

The ReLU function was designed with the biological neuron's function in mind, namely one that relays an input signal only under positive bias [55]. It is currently the *de facto* standard for hidden layer activations, and is implemented for all ML algorithms designed in this thesis [7]. The main advantages that justify its widespread use are:

- Constant gradient: In the active regime of a ReLU activated node $x \geq 0$, the gradient is always 1, preventing saturation problems that occur with other activation functions such as sigmoid and tanh.
- Sparsity: In the non-active regime $x < 0$, the ReLU function has zero gradient. This has the effect of "freezing" a neuron in a state where it always outputs 0. This can be helpful in cases where it is not known which nodes in an input or hidden layers are relevant to the task [7]. Upon successful training, the neurons that have been frozen due to the ReLU activation indicate which components of the architecture are extraneous. In cases where frozen neurons cause training problems, a similar function called a *leaky* ReLU can be employed where the gradient in the $x < 0$ regime is slightly positive [56].
- Ease of computation.

2.2.3 SoftMax

The SoftMax activation function is usually applied to an output layer, and is used to generate a logit. For a non-activated output layer $\mathbf{H} \equiv (H_1, H_2, \ldots, H_n)$, the SoftMax function $\boldsymbol{\sigma}$ is defined as:

$$\boldsymbol{\sigma}(\mathbf{H}) = \frac{1}{\sum_{i=1}^{n} e^{H_i}} (e^{H_1}, e^{H_2}, \ldots, e^{H_n}) \tag{2.22}$$

Because the exponential function is strictly positive, it is clear that all the vector elements of $\boldsymbol{\sigma}(\mathbf{H})$ will be bounded between 0 and 1. The vector elements also sum to unity, fulfilling a necessary condition of outputting a discrete probability distribution for a finite set of outcomes.

For all the discussed cases except for SoftMax, the activation functions are applied element-wise on their inputs. Sigmoid, tanh, and ReLU are plotted in Figure 2.1.

2.3 Training and Validation

So far, we have defined the architecture of a feed-forward neural network, the main functions that comprise its structure, and the necessary conditions for it to approximate a desired function. Here, we will provide an overview of the algorithms that adjust the internal parameters $\boldsymbol{\theta}$ to fulfill the objective of function approximation.

2.3.1 Backpropagation

As discussed in Chapter 2.1, optimizing a neural network comes down to finding a solution to the equation:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = 0 \tag{2.23}$$

The method of *gradient descent* is the most common for optimizing $\boldsymbol{\theta}$. The basic algorithm is [7]:

Algorithm 2: Gradient Descent with Momentum

1. Define learning rate λ
2. Define momentum α
3. Define batch size B
4. Define initial velocity $\mathbf{v}$
5. Randomly sample B instances in the training set $(\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2, \ldots, \tilde{\mathbf{X}}_B)$ and their labels $(\tilde{\mathbf{y}}_1, \tilde{\mathbf{y}}_2, \ldots, \tilde{\mathbf{y}}_B)$. If B is less than the total number of training instances, this is known as a *minibatch*.
6. Calculate loss function over the minibatch, $\mathcal{L}_B = \langle L(f(\mathbf{X}_i; \boldsymbol{\theta}), \mathbf{y}_i \rangle_i$
7. Compute gradient of loss function $\mathbf{g} = \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L}_B$

8. Update velocity $\mathbf{v} \leftarrow (\alpha\mathbf{v} - \lambda\mathbf{g})$.

9. Update $\boldsymbol{\theta}$ via an Euler step: $\boldsymbol{\theta} \leftarrow (\boldsymbol{\theta} + \mathbf{v})$.

10. Repeat steps 5-9 until convergence.

This method is called *stochastic* gradient descent (SGD) with momentum. The most basic implementation of this algorithm is to set $\alpha = 0$, resulting in a constant update for each parameter at each step. More sophisticated variants of SGD define a variable learning rate for each individual parameter, and are known as *Adaptive Learning Rate* (ALR) algorithms. Popular ALR algorithms include AdaGrad, RMSProp, and AdaM and can be found in Refs. [57], [58] and [59] respectively.

The backpropagation algorithm concerns Step 7 in Algorithm 2, namely, computing the gradient of the loss function with respect to the internal parameters. This is done via the chain rule of calculus:

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial u_1}\frac{\partial u_1}{\partial u_2}\frac{\partial u_2}{\partial u_3}\cdots\frac{\mathrm{d}u_n}{\mathrm{d}W} \tag{2.24}$$

In practice, this calculation is performed recursively. A computational class `op` implements a forward propagation function `f`, a gradient function `grad` and a backpropagation function `backprop`. These three functions are outlined more clearly below [7]:

- `op.f`: A function that computes a tensor $\mathbf{H}$, given a set of inputs $\tilde{\boldsymbol{\theta}}$. For example, a function that propagates the input layer values to the first hidden layer will have output $\mathbf{H_1}$. The inputs are the training vector $\tilde{\mathbf{X}}$, the weight matrix $\mathbf{W}_1$, and the bias vector $\mathbf{b}_1$.

- `op.grad`: Computes the gradient of each element of $\mathbf{H}$ with respect to $\tilde{\boldsymbol{\theta}}$. Each gradient is indexed by i. This function is simple to implement as `op.f` defines the functional relationship between $\mathbf{H}$ and $\tilde{\boldsymbol{\theta}}$.

- `op.backprop`: Given the gradient of $\mathcal{L}$ with respect to $\mathbf{H}$, indexed by i, computes the

gradient of $\mathcal{L}$ with respect to $\tilde{\boldsymbol{\theta}}$ via the chain rule:

$$\texttt{op.backprop}(\boldsymbol{\nabla}_{\mathbf{H}}\mathcal{L}) = \sum_i (\boldsymbol{\nabla}_{\mathbf{H}}\mathcal{L})_i \texttt{op.grad}_i \tag{2.25}$$

Because $\tilde{\boldsymbol{\theta}}$ comprises *all* inputs to a function f, the backprop function will output some partial derivatives of $\mathcal{L}$ with respect to weight and bias parameters. These derivatives are components of the target $\boldsymbol{\nabla}_{\boldsymbol{\theta}}\mathcal{L}$, and can thus be saved in a table. Other gradients computed by op.backprop are with respect to node values $\mathbf{H}'$ produced by another class op'. These gradients can be inputted into the backprop method of op' to produce more components of $\boldsymbol{\nabla}_{\boldsymbol{\theta}}\mathcal{L}$. This process is repeated until the table is filled, and the gradient of the loss function with respect to all the weights and biases has been fully computed. The basic backpropagation algorithm is then [7]:

Algorithm 3: Backpropagation for a Feed Forward Neural Network

- Define: $\texttt{op}_n$, the operation that computes the loss function $\mathcal{L}$ from the output layer $\mathbf{H}_n$. In all other cases, $\texttt{op}_m$.f computes $\mathbf{H}_{m+1}$, given inputs $\tilde{\boldsymbol{\theta}}_m \equiv (\mathbf{H}_m, \mathbf{W}_{m+1}, \mathbf{b}_{m+1})$.
- Define: table, an unpopulated array that will store the terms of the target gradient.
- Initialize gradient: $\mathbf{g} \leftarrow \texttt{op}_n\texttt{.backprop(1)}$
- ***for*** k ***in*** $(n-1, \ldots, 0)$ ***do***:

  $\mathbf{g} \leftarrow \texttt{op}_k\texttt{.backprop}(\mathbf{g})$

  Remove elements of $\mathbf{g}$ that correspond to partial derivatives of $\mathcal{L}$ with respect to parameters in $\mathbf{W}_{k+1}$ and $\mathbf{b}_{k+1}$, and insert into table.
- Return table.

2.3.2 Validation

At this point, the basic structural and algorithmic components of a feed-forward neural network have been defined. Determining when or if a neural network has been optimized is known as *validation*. The most common form of validation is to split the full training data set into a *training set*, which usually contains about 90% of the data, and a *test set*, which contains the remaining data. The training and test sets are constructed via random sampling. During training, the neural network is only exposed to instances from the training set. One pass of the full training set through the neural network during optimization is called an *epoch*. The number of updates to $\boldsymbol{\theta}$ that occur within an epoch depends on the size of the minibatch:

$$\text{training steps per epoch} = \frac{\text{size of training set}}{\text{batch size}} \tag{2.26}$$

After every epoch, the data in the test set is evaluated by the model and the loss function. Because the neural network is not trained on these instances, a decreasing loss function over the test set with each epoch indicates that the neural network is learning the correct rules of inference. If the loss function is decreasing for training instances but increasing for test instances, it means that the neural network is using its capacity to memorize the training labels [7, 60]. This is an example of *overfitting*. Techniques to avoid overfitting are summarized in Chapter 2.4.

For small training sets, the test set may be too small for it to be certain that the neural network has learned all the necessary rules of inference. In these cases, a technique known as *k-fold cross-validation* is employed: The data set is split into training and test sets k times. Because the sampling is random, this will result in k distinct training sets and test sets. The model accuracy is averaged over each training run to provide a better estimate of the performance as a whole [61].

2.4 Regularization

Machine learning architectures are often vulnerable to instabilities. An example of this can be found when the weight parameters $\mathbf{W}_i$ and $\mathbf{b}_i$ of the i^{th} layer in a basic fully connected feed-forward neural network are unbounded. In such a case, it is possible and sometimes likely that the back-propagation gradients produced in a training step will continuously drive these parameters to become larger in magnitude. This can introduce multiple problems. For example, if the network input has been normalized to small values bounded between 0 and 1, a succession of sufficiently large weight parameters in the hidden layers can produce output orders of magnitude higher. Feeding this output into an activation function that utilizes an exponential can cause numerical overflow resulting in a *NaN*. Another possibility is that a network parameter will be driven beyond the maximum floating point value allowed by the system, resulting in an *INF* parameter that will corrupt any information passing through. Evaluation of gradients in the back-propagation algorithm can also produce *NaN* values if the architecture has activation functions that are not smooth and continuous. In the case where the training process is stable, an ML architecture can still find local or global minima in the loss function that correspond to nonphysical or incorrect predictions. Another type of instability can arise when minute changes in the input correspond to substantial changes in the output of the model. Regularization is the practice of augmenting components of the architecture, such as the weights, gradient values, loss function, training time or the data set, in order to guide ML algorithms toward stability and accuracy. Common regularization techniques are discussed in this section.

2.4.1 L1 and L2 Regularization

L2 regularization penalizes ML architectures with large weight variables. The squared Frobenius norm of the weight matrices, i.e. the sum of squares of all matrix elements, is added to

the loss function $\mathcal{L}$ [62]:

$$\tilde{\mathcal{L}} = \mathcal{L} + \frac{\alpha}{2} \sum_{k=1}^{m} \sum_{i=1}^{a_k} \sum_{j=1}^{a_{k-1}} \left| W_k^{ij} \right|^2 = \mathcal{L} + \frac{\alpha}{2} \sum_{k=1}^{L} ||\mathbf{W}_k||_{\mathrm{F}}^2 \tag{2.27}$$

Here, m is the number of layers in the network, (a_k, a_{k-1}) is the dimension of the weight matrix $\mathbf{W}_k$, $\tilde{\mathcal{L}}$ is the regularized loss function, and α is an L2 hyperparameter that controls the effect of the regularization. If α is too large, the architecture will under-perform due to the strong preference to reduce weights over the bare loss function $\mathcal{L}$. As a result, α is an additional parameter that must be tuned to maximize the effectiveness of the model on the training and test data.

L1 regularization is analogous to L2, except the regularization term is the sum of absolute values of the matrix elements, rather than the Frobenius norm [62]:

$$\tilde{\mathcal{L}} = \mathcal{L} + \alpha \sum_{k=1}^{m} \sum_{i=1}^{a_k} \sum_{j=1}^{a_{k-1}} \left| W_i^{jk} \right| \tag{2.28}$$

The behavioural differences between L1 and L2 regularization are considerable. The main difference can be seen by examining the loss function near a minimum where the gradient vanishes. Labelling $\boldsymbol{\theta}$ as the set of internal parameters flattened into a vector, and $\boldsymbol{\theta}^*$ as the set of internal parameters that minimize $\mathcal{L}$, the Taylor expansion of the loss function to second order is:

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) \approx \mathcal{L}(\boldsymbol{\theta}^*; \mathbf{X}, \mathbf{y}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^{\mathrm{T}} \mathbf{H}_{\boldsymbol{\theta}}(\boldsymbol{\theta}^*; \mathbf{X}, \mathbf{y})(\boldsymbol{\theta} - \boldsymbol{\theta}^*) \tag{2.29}$$

Here, $\mathbf{H}_{\boldsymbol{\theta}}$ is the Hessian matrix of $\mathcal{L}$ with respect to $\boldsymbol{\theta}$. Following Goodfellow *et al.* in Ref. [7], we will assume further that the Hessian is diagonal, i.e. the features in $\mathbf{X}$ are uncorrelated.

In this case, the loss function takes the form:

$$\mathcal{L}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) \approx \mathcal{L}(\boldsymbol{\theta}^*; \mathbf{X}, \mathbf{y}) + \frac{1}{2} \sum_i \left[H_{ii} (\theta_i - \theta_i^*)^2 \right] \tag{2.30}$$

The minimization condition for L1 and L2 regularization are respectively:

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_{\text{L1}} = \alpha \text{sign}(\boldsymbol{\theta}) + \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L} = 0 \tag{2.31}$$

$$\boldsymbol{\nabla}_{\boldsymbol{\theta}} \tilde{\mathcal{L}}_{\text{L2}} = \alpha \boldsymbol{\theta} + \boldsymbol{\nabla}_{\boldsymbol{\theta}} \mathcal{L} = 0 \tag{2.32}$$

The ideal regularized weight parameters $\tilde{\theta}_i$ are thus given by [7]:

$$\tilde{\theta}_i = \text{sign}(\theta_i^*) \max \left(|\theta_i^*| - \frac{\alpha}{H_{ii}}, 0 \right) \qquad \text{(L1)} \tag{2.33}$$

$$\tilde{\theta}_i = \frac{H_{ii}}{H_{ii} + \alpha} \theta_i^* \qquad \text{(L2)} \tag{2.34}$$

Because the loss function is expanded about a local minimum, the Hessian elements H_{ii} are positive. This implies that for L2 regularization, if the unregularized optimal parameter θ_i^* is non-zero, the regularized parameter $\tilde{\theta}_i$ will also be non-zero. In L1 regularization, if the regularization parameter α is sufficiently large, the optimal regularized parameter can be exactly 0. Thus, L1 regularization encourages a sparse representation of the optimal solution. This is useful in cases where it is unclear if certain features of the input are important for approximating a function of interest. The weights that are driven to 0 in an L1 regularized algorithm will determine which input features can be discarded. This process is called *feature selection* [63].

2.4.2 Bootstrap Aggregating

Bootstrap Aggregating, also known as bagging, is a model averaging technique where multiple models are trained, and predictions are made by averaging the model responses [64]. Bagging methods enjoy the most success in the case where the individual model errors are uncorrelated. This can be seen by examining N models with errors ϵ_i. The total expected error $\langle E \rangle$ and its square are given by:

$$\langle E \rangle = \frac{1}{N} \sum_{i=1}^{N} \langle \epsilon_i \rangle \tag{2.35}$$

$$\left\langle E^2 \right\rangle = \frac{1}{N^2} \left(\sum_{i=1}^{N} \sum_{j=1}^{N} \langle \epsilon_i \epsilon_j \rangle \right) \tag{2.36}$$

$$= \frac{1}{N^2} \left(\sum_{i} \left\langle \epsilon_i^2 \right\rangle + \sum_{i} \sum_{j \neq i} \langle \epsilon_i \epsilon_j \rangle \right) \tag{2.37}$$

Under the assumption that the errors ϵ_i are uncorrelated ($\langle \epsilon_i \epsilon_j \rangle_{j \neq i} = 0$) and distributed about 0 with variance σ^2 [7]:

$$\left\langle E^2 \right\rangle = \frac{\sigma^2}{N} \tag{2.38}$$

If the errors are correlated, this gives:

$$\langle \epsilon_i \epsilon_j \rangle_{i \neq j} \leq \sigma^2 \tag{2.39}$$

$$\therefore \qquad \left\langle E^2 \right\rangle \leq \sigma^2 \tag{2.40}$$

This implies that for any ensemble of models with non-perfect error correlation, the prediction accuracy will be improved. The principle method for reducing error correlation between models is to train them on randomly selected subsets of the training data. In some cases, it is sufficient to select model hyperparameters randomly (see Chapter 2.5), and train all the models on the same data set [7]. Bagging is typically performed on models that do

not require extensive computational effort to train, as many models must be optimized to construct a bagging ensemble. One common example of a bagging method is performed with decision trees as the base model, known as a random forest [65, 66].

2.4.3 Dropout

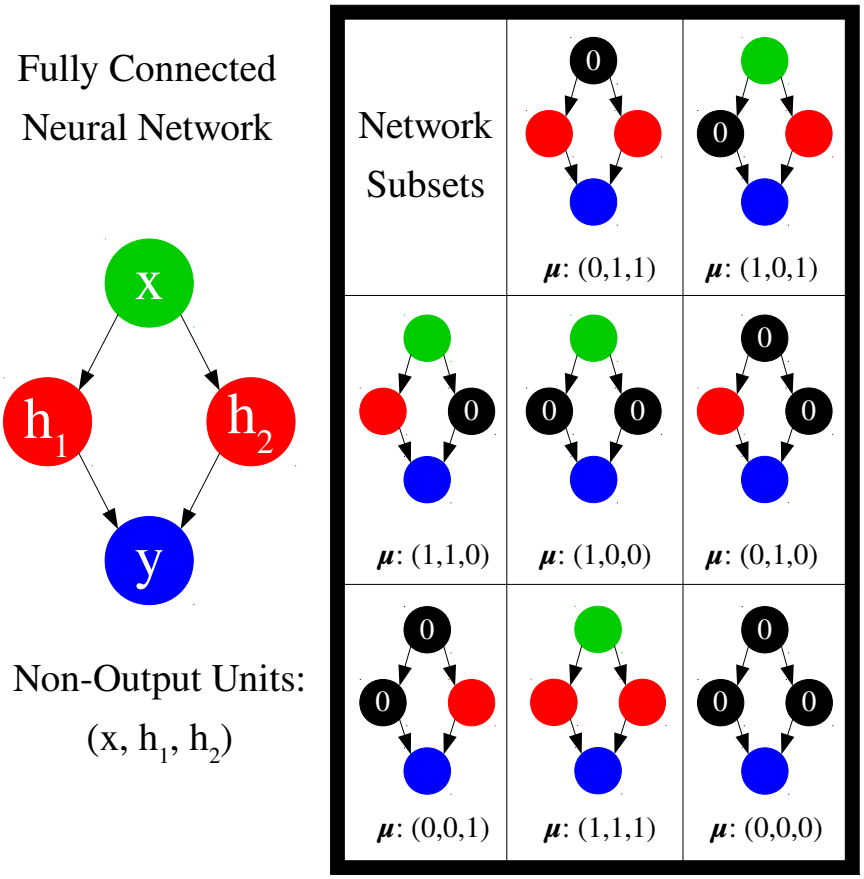


Figure 2.2: All network subsets for the simple fully connected neural network shown on the left. At training time, a binary mask $\boldsymbol{\mu}$ is multiplied element-wise with the non-output nodes of the network. A given update is performed on the weights corresponding to the neurons that remain active in the configuration. At test time, the outputs of each network subset are averaged [7, 60].

When training high-capacity models with large training sets, the standard bagging method quickly becomes impractical due to the required time and computational resources of training multiple models. The dropout method, first introduced by Srinistava *et al.* in 2014, sidesteps

this issue by instead aggregating *subsets* of a single model. In the case of an artificial neural network, a subset can be constructed by forcing a configuration of neurons in the input and hidden layers to output 0 via a binary dropout mask $\boldsymbol{\mu}$ [7,60]. For a neural network with N non-output units, the amount of unique masks that can be constructed are 2^N. Figure 2.2 illustrates this method for a small neural network.

Because of the exponential scaling of the number of neural network subsets with N, an average over the whole ensemble is usually infeasible to train. For such cases, the number of binary masks can be limited. This allows for an ensemble method that enjoys the reduced variance shown in Equation 2.38, and remains computationally tractable.

A stochastic adaptation of dropout regularization which now undergoes common use draws from the work of Warde-Farley *et al.* and Hinton *et al.* [67, 68]. Rather than selecting a number of binary masks to be used during training, every non-output node in the network is set to 0 with some chosen probability p. At test time, all the nodes in the network are active and scaled by a factor of $(1-p)$. To understand the purpose of this scaling factor, consider the simple case of a node h in a neural network whose expected value for optimal performance over the full training set is 1. Assuming a dropout value $p = 0.5$, the value of h will be equal to zero for about half of the training steps. To maintain an expected value of 1 over the course of training, the network will converge such that h has an expected value of 2 for training steps in which it remains activated. At test time, h is always activated, and thus has a sub-optimal expected value of 2, which the scaling factor of $1-p$ corrects for. This form of dropout regularization has empirical success despite a lack of a rigourous theoretical justification [7].

2.4.4 Early Stopping

Early stopping is a simple and effective regularization technique used to prevent overfitting in a model with excessive capacity [7,69]. During the course of training, the loss is periodically

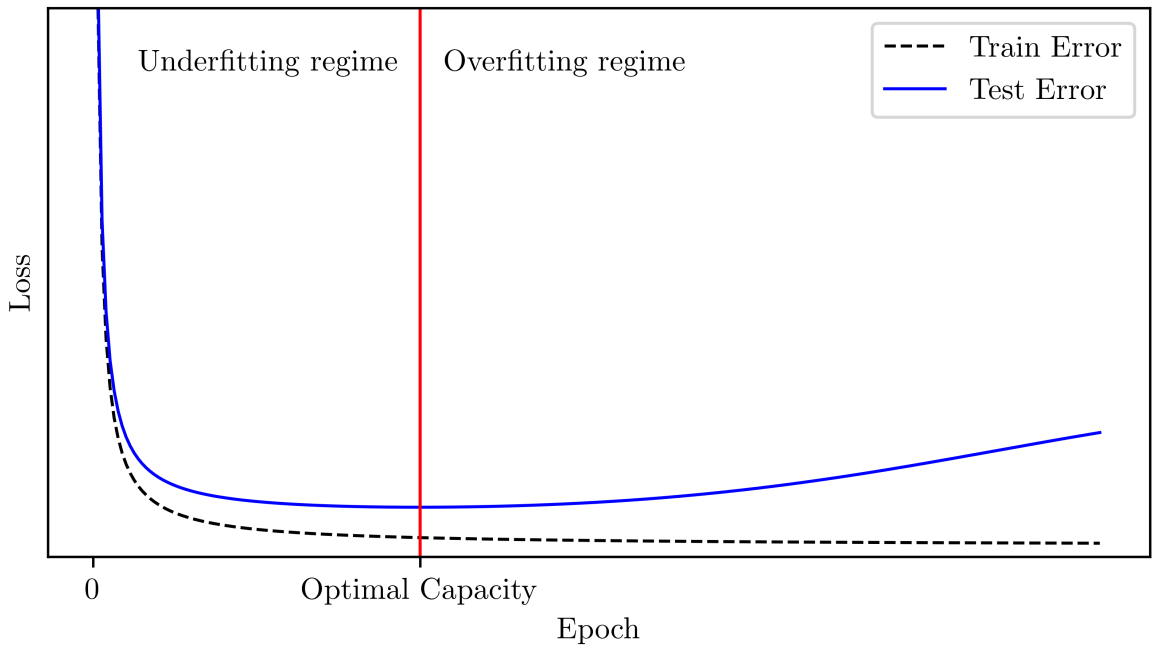


Figure 2.3: Typical behaviour of ML algorithm during training. In the underfitting regime, the loss for both training and test data is not optimal. In the overfitting regime, the loss for the training set is minimized, but the test loss begins to rise as the algorithm is now memorizing the training set at the expense of determining the desired function. The epoch given by the red line represents where training should stop. Figure sourced from Ref. [7] pp. 112.

calculated using the validation set as the model input (see Chapter 2.3), and training is halted when the test loss begins to rise. This represents the crossover point in training when the model ceases to approximate the desired function and begins to memorize the training set. Overfitting is to be avoided unless the training set completely saturates the function input space. An example of such a training set is given in Chapter 4.2.3. All other training sets in this thesis sparsely sample the input space and the early stopping regularizer is always employed. Early stopping is illustrated in Figure 2.3.

2.4.5 Artificial Data

It can often be the case in supervised learning that the data set is too small or imbalanced for an ML algorithm to effectively train from. Many techniques to grow the data set are available, but their effectiveness is problem-specific. For example, in image recognition applications,

new data can be generated by rotating, shifting, reflecting or enlarging images within the data set [70], but in other cases such transformations are not feasible due to the input data format.

A related issue for classification problems concerns the percentage of data in the training set that belong to a particular class. For example, in fraud detection the amount of non-fraudulent occurrences can outnumber fraudulent ones by 2 orders of magnitude [71, 72]. In such cases, a ML classifier may maximize its accuracy by always predicting the majority class, rendering it useless [73]. In the case where the data set is sufficiently large, the ratio of classes can be equalized by generating a data subset where the majority class has been deliberately undersampled. In cases where the data set is small, an oversampling method must be employed. One example of a robust oversampling technique is known as *SMOTE*, where members of the minority class are used to generate new members via linear combinations of their features [74].

If possible, one should collect more data to improve ML performance. If this cannot be done directly, one may develop a model for the data that can be used to generate new examples. This is especially applicable to cases of function inversion, where production of data given a set of inputs is trivial, but determining the inputs from the data is not. This is the main strategy for the project in this thesis, and will be further explored in Chapters 2.4.5 and 4.3.

2.5 Hyperparameter Optimization

Once a ML architecture with *internal* variables has been constructed, there always remain *external* parameters that fully define the optimization process. Examples of these *hyper-parameters* include batch size B and learning rate λ. Additional hyperparameters can be present if regularization techniques are used, such as α in L1 and L2 regularization (see Chapter 2.4.1), and the dropout parameter p (see Chapter 2.4.3). These hyperparameters must be optimized via an external method to guarantee success of the ML architecture. In

principle, any optimization method can be used to tune hyperparameters. Methods that have been used include global search, gradient descent, Bayesian optimization, genetic algorithms, and particle swarms [75–78]. Due to the computationally expensive task of training and validating a ML algorithm to determine the performance of a single tuple of hyperparameters, all hyperparameter optimizations in this thesis were performed using search methods.

2.5.1 Search Methods

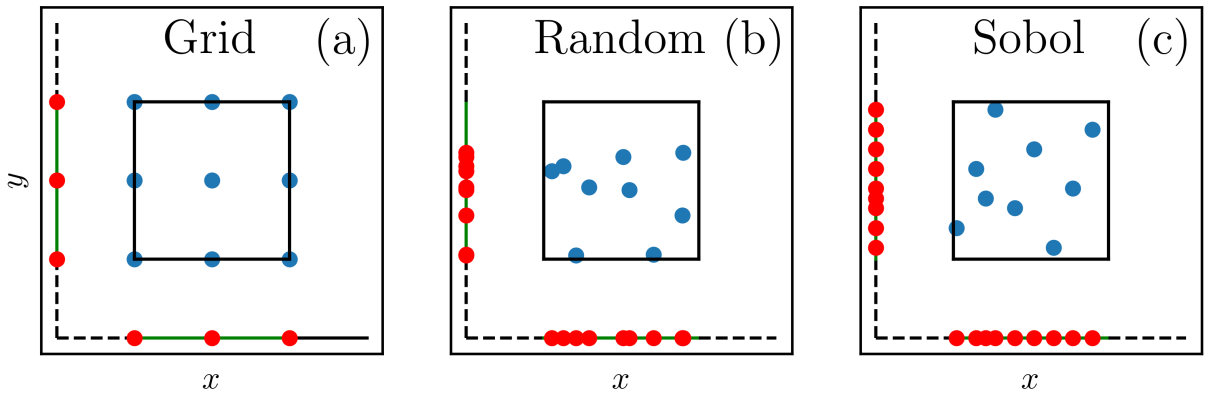


Figure 2.4: (a): Grid search vectors for hyperparameters x and y (blue). Individual x and y components are plotted in red. Space is uniformly filled, but sparse when dimensions are reduced. (b) Random search vectors. The sample space is not uniformly filled, but has improved coverage when dimensions are reduced. (c): Sobol search vectors. Both full and reduced sample spaces are uniformly filled.

The grid search method chooses an immutable set of values from the continuum of each hyperparameter, generates a set of hyperparameter tuples from the Cartesian product of the sets, and tests the ML algorithm performance over each tuple [7]. The chosen points can be distributed linearly, logarithmically or otherwise depending on the hyperparameter, and should lie within a range considered to provide the highest probability of success. The main advantage of this method is that it explores the hyperparameter space uniformly, and is simple to implement. The main disadvantage is that if one or more hyperparameters have little effect on the outcome in the explored range, the majority of the tested hyperparameter tuples are redundant. This will result in wasted computational effort. Figure 2.4a shows an

example of the grid search method.

The random search method draws an integer number of tuples from a random distribution over a region of interest in the full hyperparameter space [7]. Random search has the main advantage of avoiding redundancy in the case where some hyperparameters have a minimal effect. The main disadvantage of random search is that regions of the hyperparameter space can go unexplored. This disadvantage is compounded as the dimensionality of the hyperparameter space increases, or if the sample rate is limited. Figure 2.4b shows an example of the random search method.

To utilize the advantages and avoid the disadvantages of both grid search and random search, a pseudo-random search using Sobol sequences was implemented for the optimization of hyperparameters in this thesis. A detailed description of the algorithm that generates Sobol pseudo-random numbers can be found in Ref. [79]. In brief, Sobol-distributed vectors provide a uniform filling of the sample space, and maintain a uniform filling when collapsed along a given dimension. Figure 2.4c shows an example of the Sobol search method.

In this thesis, we will construct a neural network classifier to scan an area detector image, and classify the number of photons that were incident on each pixel. The training data will be constructed using an artifical data model, as outlined in Chapter 2.4.5. The neural network will be trained according to the methods of Chapter 2.3, specifically with the backpropagation algorithm. Regularization techniques such as dropout and early stopping will be employed. The network hyperparameters will be optimized according to the search methods discussed in Chapter 2.5.1, specifically the Sobol search method. The hyperparameters will be optimized by maximizing the network accuracy via the k-fold cross-validation method. The output of the classifer will be a series of discrete probability distributions, and will be processed using Bayesian methods to arrive at a prediction for a full image. Then, large speckle images with known contrast parameters will be generated and used to determine the efficacy of the classifier algorithm. In addition, a mixture density network will be designed to determine

continuous probability distributions for the spatial coordinates of the incident photons. This algorithm will also be trained and validated using an artificial data model.

3 X-Ray Photon Correlation Spectroscopy and Machine Learning

3.1 Background

The main goal of Dynamic Light Scattering (DLS) experiments is to study the structure and dynamics of a wide range of systems using diffraction patterns produced via their interactions with a coherent source of light. The diffraction image is also known as a *speckle pattern*. Examples of such systems include crystalline and non-crystalline solids, liquid crystals, colloid suspensions, DNA and other biological molecules, and foams [80–83]. Depending on the length and time scales of interest, the optimal wavelength for the light source can range from near-infrared to X-ray [84, 85].

Photon correlation spectroscopy (PCS) describes a family of DLS techniques that measure the temporal auto-correlation function of a speckle pattern to determine timescales of equilibrium and non-equilibrium effects such as phase transitions and diffusion, and structural properties such as particle size. To give a specific example, consider a dilute mono-disperse suspension of spherical particles with unknown radius r and diffusion coefficient D. A coherent source of monochromatic light is incident on the sample and a detector collects the light-intensity information $I(t)$ at a particular wavevector q, given by:

$$q = \frac{4\pi n}{\lambda} \sin\left(\frac{\theta}{2}\right) \tag{3.1}$$

where n is the refractive index of the suspension, λ is the wavelength of the incident light source, and θ is the detector angle. The second order intensity auto-correlation function $g_2(q, \tau)$ can be generated by performing the experiment and calculating [80]:

$$g_2(q,\tau) = \frac{\langle I(q,0)I(q,\tau)\rangle}{\langle I\rangle^2} = \frac{1}{\langle I\rangle^2} \lim_{T\to\infty} \frac{1}{T} \int_0^T I(q,t)I(q,t+\tau)\,\mathrm{d}t \tag{3.2}$$

The second order auto-correlation function $g_2(q,t)$ can be related to the intermediate scattering function $F(q,t)$ via the Siegert equation [86]:

$$g_2(q,t) = 1 + \gamma^2 \left|F(q,t)\right|^2 \tag{3.3}$$

where γ is a factor that accounts for partial coherence effects due to the incident laser beam. For $\tau = 0$, the integral simplifies to $\langle I^2 \rangle / \langle I \rangle^2$, giving $g_2(0) = 1+\gamma^2$. As τ approaches infinity, the signals $I(t)$ and $I(t+\tau)$ will de-correlate, resulting in an asymptotic auto-correlation of 1. The behaviour of $g_2(\tau)$ between these two limits can be shown to be exponentially decaying (see Ref. [80]):

$$g_2(q,\tau) = 1 + \gamma^2 e^{-Dq^2\tau} \tag{3.4}$$

The diffusion constant can be retrieved via a straightforward fitting algorithm, and the particle radius r can be determined from the Einstein-Stokes relation:

$$r = \frac{k_\mathrm{B}T}{6\pi\eta D} \tag{3.5}$$

where k_B is Boltzmann's constant, T is the temperature of the sample, and η is the known viscosity of the medium.

3.2 X-Ray Photon Correlation Spectroscopy

The project in this thesis concerns a subset of PCS known as X-ray Photon Correlation Spectroscopy (XPCS), first introduced by Sutton *et al.* in 1991 [42]. Prior to the advent of XPCS, PCS experiments were performed with light in the visible spectrum, permitting study of low-opacity materials on length scales greater than 200nm [42]. In the 18 years following the first use of intensity fluctuation spectroscopy using X-rays, it became possible to observe dynamics in high opacity systems, and on length scales of order $\sim$ 10nm [85, 87]. This

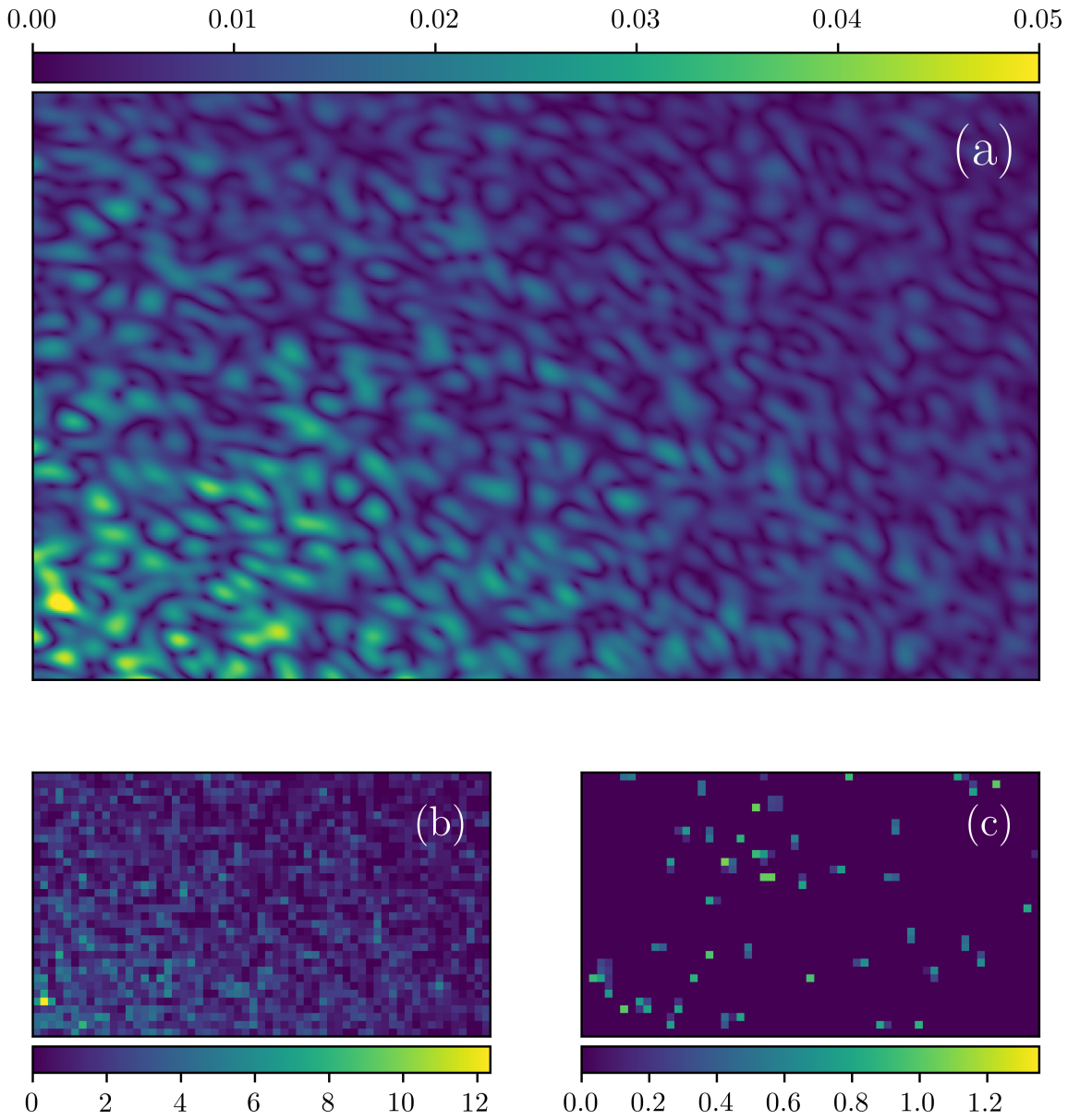


Figure 3.1: (a): Numerically simulated speckle pattern for coherent light scattering off a suspension of spherical colloids. Intensity fluctuations are roughly constant. (b): Simulated area detector readings of the speckle pattern in (a) for a 35x60 CCD array in the high intensity limit. (c): Simulated CCD readings of the speckle pattern for a low intensity X-ray pulse.

occurred in conjunction with the continued development of high-intensity X-ray synchrotron sources [88]. Observation of atomic scale dynamics using XPCS was first made in a diffusion study of Cu-Au in 2009 [89].

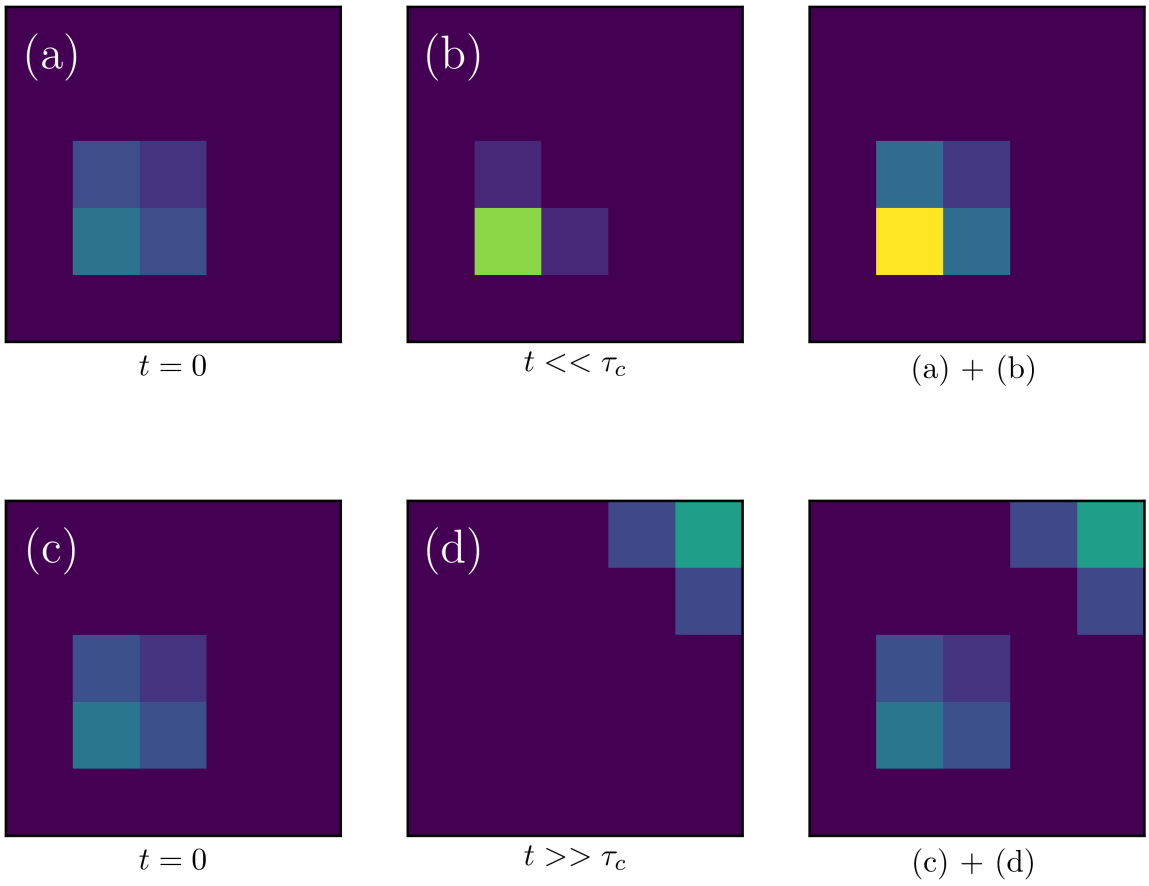


Figure 3.2: (a) and (c): First photon detected from first X-ray pulse. (b): Second X-ray detected in the case where the pulse delay is much smaller than the sample correlation time τ_c. (d): Second X-ray detected where the pulse delay is large compared to the sample correlation time. The two figures on the right show the difference in contrast between these two speckle patterns when the area detector sums both contributions together. On a large scale ($> 10^6$ pixels), these differences are statistically significant, and depend strongly on the sample dynamics.

A key consideration for XPCS studies of atomic scale dynamics is the trade-off between high and low intensity pulses. High intensity pulses provide rich speckle patterns at the expense of interference with the dynamics of the system being measured. Low intensity pulses do not interfere with the sample dynamics but result in a sparse detection of light from the area detector due to the fact that the probability of a single photon being detected is low [44,90].

Figure 3.1 shows an example of sparse light detection from an underlying electromagnetic intensity profile that exhibits speckle.

The data analyzed in this thesis is produced via an X-ray split-pulse technique, wherein two weak coherent X-ray pulses separated by a delay time τ are incident on a sample. An area detector comprised of charge-coupled devices (CCDs) collects the charge excited by the incident X-ray photons from both pulses. Each CCD outputs an integer number of counts proportional to the amount of charge collected. These counts are termed *analog digital units* (ADUs). This produces a single image corresponding to light scattered by both pulses. A key insight is that while the speckle pattern is sparsely measured, the delay time τ affects the statistical distribution of photons on the area detector. For pulse delays much smaller than the sample correlation time, a high-intensity region in the detector will tend to collect photons corresponding to both pulses. For delay times larger than the sample correlation time, each pulse will induce high intensity regions that are independent, resulting in roughly twice as many "bright spots." This is illustrated in Figure 3.2. As we will soon show, information on the dynamics of the sample for intermediate pulse delay times can be meaningfully extracted from these statistics, provided enough data is collected.

The statistical distribution of photons incident on a detector is derived by Goodman in Ref. *Statistical Optics* [91], and will be summarized here. The total energy E collected over time t within an area A is given by:

$$E = \int I(\mathbf{r}, t)\, \mathrm{d}t\mathrm{d}A \tag{3.6}$$

where $I(\mathbf{r}, t)$ is the light intensity field. In general, I is a fluctuating quantity, giving rise to a probability distribution for the fluctuations of E:

$$P(E) = \left(\frac{M}{\bar{E}}\right)^M \frac{\bar{E}^{M-1} e^{-\frac{ME}{\bar{E}}}}{\Gamma(M)} \tag{3.7}$$

where Γ denotes the Gamma function, $\bar{E}$ is the average energy and M is the number of

degrees of freedom in the measurement. The parameter M earned its name due to its limiting cases: For measurement times t exceeding the correlation time of the system τ_c, M simplifies to t/τ_c, the number of coherence intervals in time. In cases where $t << \tau_c$, M approaches unity. The explicit behaviour of M in the intermediate cases is given in Ref. [91]. The contrast of a speckle pattern, which is a quantity from which the system dynamics can be calculated [86], is given by $\beta = 1/M$. In the weak field limit, as is relevant to studies of atomic scale dynamics, one must use Mandel's formula to convert this continuous distribution to a discrete one that describes the probability of individual photons being detected within the area of a CCD [44]:

$$P(k) = \frac{\Gamma(k+M)}{\Gamma(M)\Gamma(k+1)} \left(1 + \frac{M}{\bar{k}}\right)^{-k} \left(1 + \frac{\bar{k}}{M}\right)^{-M} \tag{3.8}$$

where k denotes the number of photons detected and $\bar{k}$ is the average count-rate per CCD, or *pixel*. Because the intermediate scattering function $F(q,t)$ can be determined by the contrast factor $\beta = 1/M$ [86], the system dynamics can be extracted from a histogram of photon counts via a maximum likelihood fit to Equation 3.8.

3.3 Fitting via Maximum Likelihood

We will define the weak field limit as $\bar{k} < 0.01$. This permits us to expand Equation 3.8 to cubic order:

$$P_0 \approx 1 - \bar{k} + \frac{(M+1)\bar{k}^2}{2M} - \frac{(M+1)(M+2)\bar{k}^3}{6M^2} \tag{3.9}$$

$$P_1 \approx \bar{k} - \frac{(M+1)\bar{k}^2}{M} + \frac{(M+1)(M+2)\bar{k}^3}{2M^2} \tag{3.10}$$

$$P_2 \approx \frac{(M+1)\bar{k}^2}{2M} - \frac{(M+1)(M+2)\bar{k}^3}{2M^2} \tag{3.11}$$

$$P_3 \approx \frac{(M+1)(M+2)\bar{k}^3}{6M^2} \tag{3.12}$$

For an observed histogram of (n_0, n_1, n_2, n_3) counts, the most likely parameters for $\bar{k}, \beta = 1/M$ can be determined by minimizing a chi-square distribution, conventionally given as [92]:

$$\chi^2(\bar{k}, \beta) = \sum_{k=0}^{3} \frac{(n_i - NP_k(\bar{k}, \beta))^2}{NP_k(\bar{k}, \beta)} \tag{3.13}$$

This minimization condition assumes that each n_k is Gaussian distributed. However, this only holds if the counts in each bin are sufficiently high. Due to the fact that n_2 and n_3 will be small for $\bar{k} < 0.01$, it is more appropriate in this case to assume the bin counts are Poisson distributed if the total number of pixels N is not known. If N is known, as is the case in this thesis, the bin counts will be multinomial distributed. This gives an alternative chi-squared statistic [92]:

$$\chi_p^2(\bar{k}, \beta) = -2 \sum_{k=0}^{3} n_k \log \left(\frac{NP_k(\bar{k}, \beta)}{n_k} \right) \tag{3.14}$$

If $\bar{k}$ had already been determined by averaging the total counts, the contrast parameter β can be determined by minimizing this statistic. The most likely contrast β^* is thus:

$$\beta^* = \text{argmin}_\beta \left[-\sum_{k=0}^{3} n_k \log \left(\frac{NP_k(\beta; \bar{k})}{n_k} \right) \right] \tag{3.15}$$

One can estimate the number of pixels required to determine β to a satisfactory degree by expanding the quantities P_2 and P_1^2 to second order:

$$P_2 \approx \frac{(M+1)\bar{k}^2}{2M} \tag{3.16}$$

$$P_1^2 \approx \left(\bar{k} - \frac{(M+1)\bar{k}^2}{M} \right)^2 \approx \bar{k}^2 \tag{3.17}$$

$$\therefore \quad \frac{2P_2}{P_1^2} \approx \frac{M+1}{M} = 1 + \beta \tag{3.18}$$

$$\beta \approx \frac{2P_2}{P_1^2} - 1 \tag{3.19}$$

We can express the uncertainty in β in terms of the uncertainties in P_1 and P_2 using the error propagation formula [93]:

$$\frac{\delta\beta}{\beta} = \sqrt{\left(\frac{P_1}{\beta}\frac{\partial\beta}{\partial P_1}\frac{\delta P_1}{P_1}\right)^2 + \left(\frac{P_2}{\beta}\frac{\partial\beta}{\partial P_2}\frac{\delta P_2}{P_2}\right)^2} \tag{3.20}$$

$$= \sqrt{\left(\frac{-4MP_2}{P_1^2}\frac{\delta P_1}{P_1}\right)^2 + \left(\frac{2MP_2}{P_1^2}\frac{\delta P_2}{P_2}\right)^2} \tag{3.21}$$

Assuming near-Poisson statistics, the relative uncertainties in P_1, P_2 can be expressed as:

$$\frac{\delta P_k}{P_k} = \frac{1}{\sqrt{N_k}} = \frac{1}{\sqrt{P_k N}} \tag{3.22}$$

where N is the total number of pixels. For count rates $\bar{k} < 0.01$, P_1 is at least two orders of magnitude larger than P_2. Thus, we can approximate the relative uncertainty in β by only keeping the term corresponding to the relative uncertainty in P_2:

$$\frac{\delta\beta}{\beta} \approx \frac{2MP_2}{P_1^2}\frac{1}{\sqrt{P_2 N}} \tag{3.23}$$

$$\approx \frac{1}{\bar{k}}\sqrt{\frac{2M(M+1)}{N}} \tag{3.24}$$

To achieve 10% relative uncertainty, the number of pixels that need to be analyzed is:

$$N = \frac{2M(M+1)}{(0.1\bar{k})^2} \tag{3.25}$$

To give a numerical example, for $M = 2$ and $\bar{k} = 0.01$, the required number of pixels measurements is $N = 1.2 \times 10^7$.

3.4 Machine Learning and XPCS

The application of Machine Learning to this project stems from the fact that CCD area detectors can obscure the location of incident X-ray photons. This is because the CCDs *indirectly* measure photons via the collection of excited electrons. The integer number of ADUs returned by the detector is proportional to the energy of the incident X-ray. The distribution of electrons excited by a photon has a spatial dependence, allowing for the activation of multiple, neighbouring CCDs. A connected collection of activated pixels due to one or multiple photons is called a *droplet* [94]. Examples of droplets are shown in Figure 3.2.

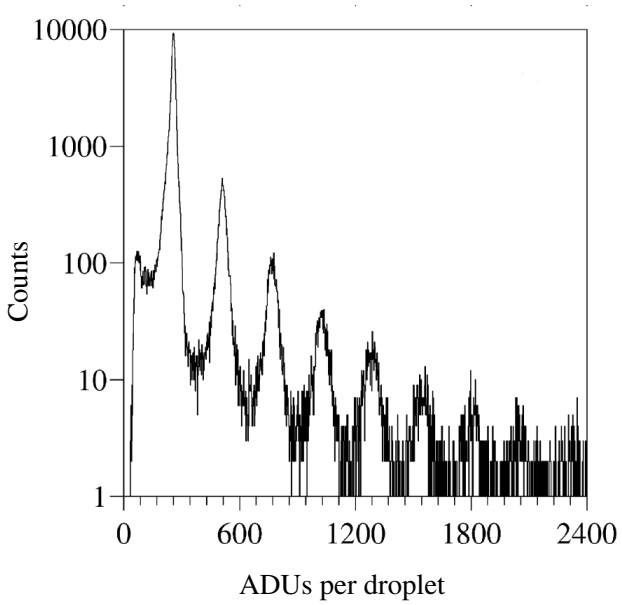


Figure 3.3: Histogram of of experimental droplet count-rates. First major peak is located at 257 counts. Peaks corresponding to integer multiples of this count-rate are clearly visible. Standard deviation in first peak is equal to 31 counts, indicating the incident X-ray count rate I has uncertainty $0.12I$. Figure sourced from Ref. [94].

Figure 3.3 shows a histogram of the summed count-rates per droplet for $AuAgZn_2$ scattering data taken with a *Princeton Instruments* "deep depletion" CCD array [44]. Strong peaks

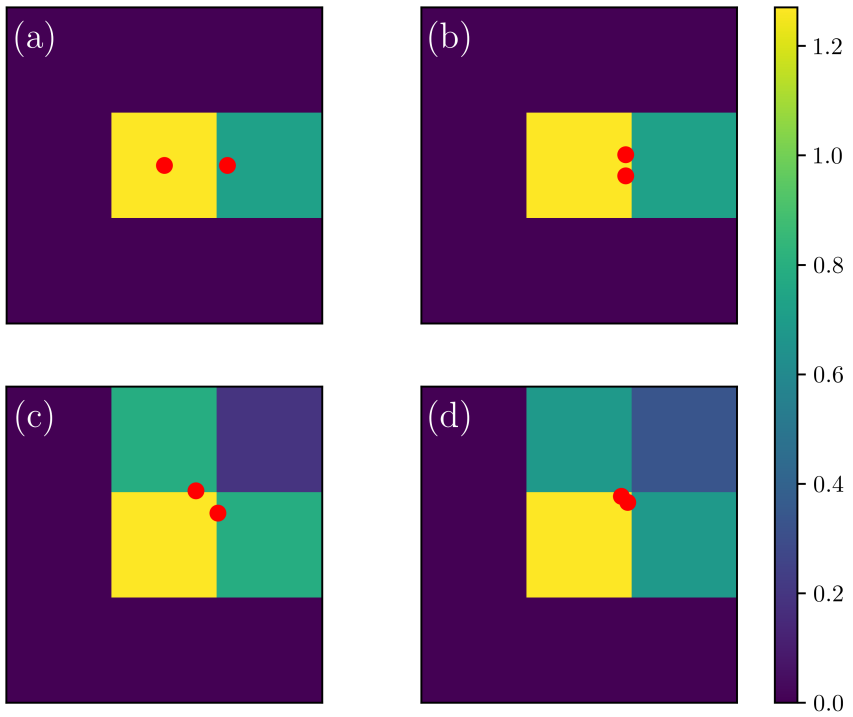


Figure 3.4: Examples of two-photon droplets. For the cases of (a) and (b), the underlying droplet patterns are identical, but can be produced by distinct photon configurations. For (c) and (d), the underlying patterns are similar but individually allow for only one photon configuration. It is the goal of this project to extract meaningful probabilistic information from the droplet in cases such as (a) and (b), and directly solving droplets for cases such as (c) and (d).

occur regularly at integer multiples of the photon energy measured in ADUs. These correspond to the integer number of X-ray photons in the droplet. For droplets with one X-ray, in this case corresponding to roughly 260 ± 30 ADUs, the location of the X-ray is simply the pixel with the maximum number of counts, and unambiguously contributes one pixel to a histogram bin corresponding to $k = 1$. Droplets with two or more photons can be ambiguous. Figure 3.4 shows several droplets containing two X-rays using a model developed in Chapter 4.1. It can be possible to have two identical droplet patterns whose photon configurations differ between one pixel with two photons, and two pixels with one photon each. In such a case, one can only calculate the posterior probabilities of these two configurations given the area detector measurements. In other cases, droplets with minor differences can individually admit different, but exclusive configurations. These issues are compounded as the number of

photons within a droplet increases. Generating posterior probabilities or directly solving for droplet configurations is in general not a straightforward task, but can have an important effect on histogram used to fit Equation 3.8 to the contrast factor β. A Machine Learning technique using feed-forward neural networks has been developed to address this problem, and will be described in the next chapter.

4 X-ray Speckle Droplet Analysis

4.1 Data Model

There are several challenges associated with direct use of the experimental droplet data to train a ML model. First, the experimentally produced droplet patterns do not have labels corresponding to the true positional coordinates of the X-ray photons. This fails to fulfill a necessary condition for supervised learning architectures. Second, the limited size of the data set prevents us from taking advantage of a neural network's increased performance with higher data volume. This problem is compounded if the data set is filtered into categories that further reduce the data volume (e.g. all droplets containing three photons). For these reasons, an artificial data approach outlined in Chapter 2.4.5 was taken. A model was developed and calibrated to the statistical attributes of the experimental data set, and used to produce a sufficiently sized training set. This section provides a description of that model.

Two models for the spatial probability distribution of the cloud of electrons excited by an incident X-ray photon were tested. The first was a normal distribution given by:

$$P_N(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu_x)^2 + (y-\mu_y)^2}{2\sigma^2}\right) \tag{4.1}$$

where (x, y) denotes the spatial coordinate of the detector array, (μ_x, μ_y) is the coordinate of the photon, and σ is the standard deviation of the spatial probability distribution. The signal returned by a detector, measured in ADUs, is proportional to the number of electrons collected by that detector. Thus, the detector signal S_d is given by the integral of Eq. (4.1)

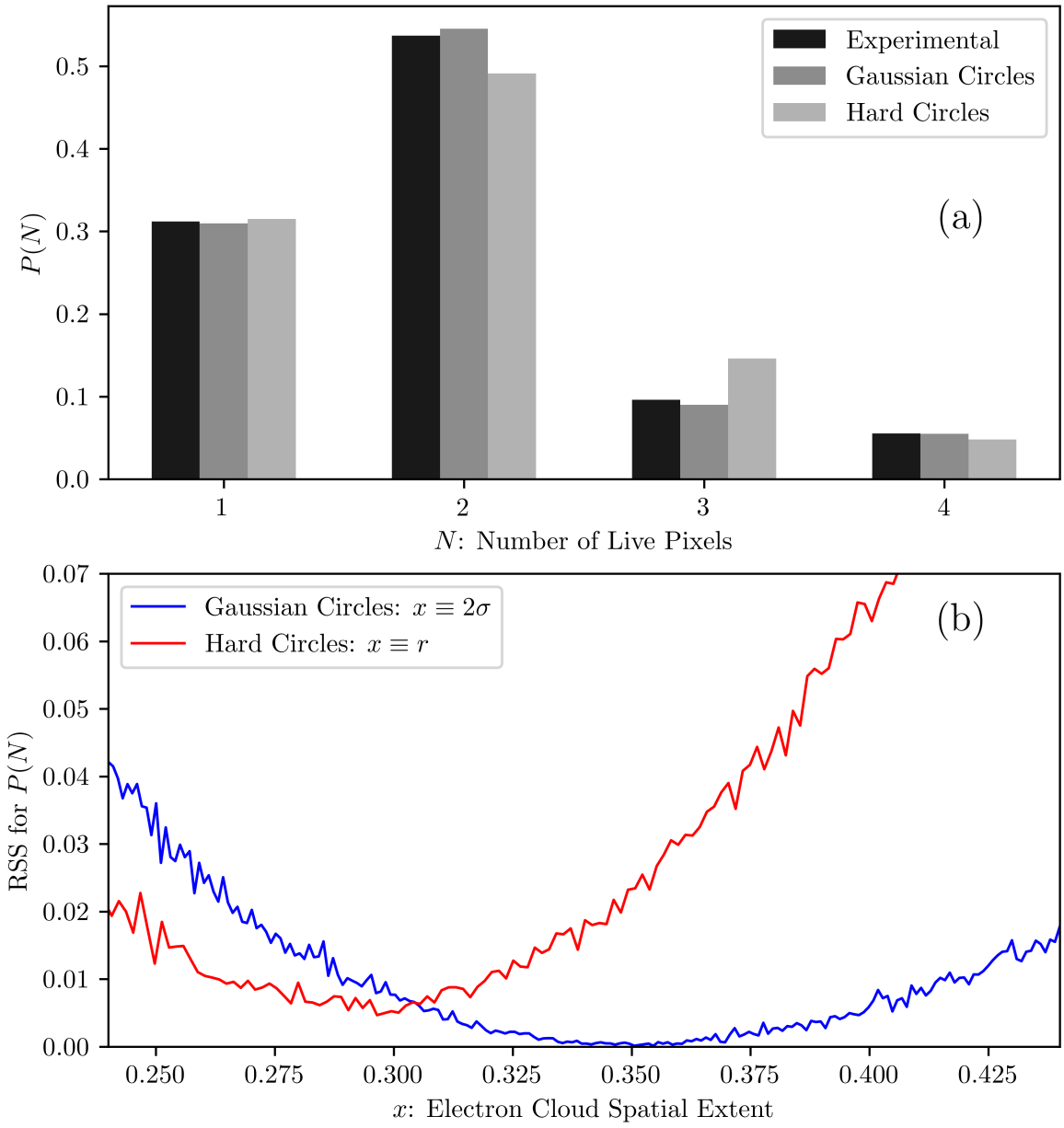


Figure 4.1: (a): Fraction of droplets consisting of one, two, three or four live pixels for experimental data, the Gaussian model with optimal σ, and the Hard Circle model with optimal r. (b): Residual Sum of Squares (RSS) of Monte Carlo generated histograms vs. experimental histogram for varying σ, r.

over the spatial extent of the detector:

$$S_d = A \int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} P_N(x, y)\, \mathrm{d}y\mathrm{d}x \tag{4.2}$$

$$= \frac{A}{4}\left[\operatorname{erf}\left(\frac{x_{\max}-\mu_x}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{x_{\min}-\mu_x}{\sqrt{2}\sigma}\right)\right]\left[\operatorname{erf}\left(\frac{y_{\max}-\mu_y}{\sqrt{2}\sigma}\right) - \operatorname{erf}\left(\frac{y_{\min}-\mu_y}{\sqrt{2}\sigma}\right)\right] \tag{4.3}$$

Here, $(x, y)_{\text{min, max}}$ denote the boundary coordinates of the square pixels. The constant of proportionality A is equal to the expected value of a photodetector measurement of one X-ray photon. For convenience, the constant of proportionality is normalized to unity in this thesis. In post-processing of the experimental data, pixels returning a value less than $0.1A$ are set to zero for noise reduction. Additionally, A can vary as much as 12% according to Figure 3.3. Both of these effects are accounted for in the model that generates the training data.

The second model tested was a uniform "hard circle" distribution given by:

$$P_U(x, y) = \frac{1}{\pi r^2} \begin{cases} 1 & \sqrt{(x-\mu_x)^2 + (y-\mu_y)^2} \leq r \\ 0 & \sqrt{(x-\mu_x)^2 + (y-\mu_y)^2} > r \end{cases} \tag{4.4}$$

where r is the radius of the circle. The detector value is given by a straightforward area integral of the segment of the circle that overlaps with the detector:

$$S_d = A \int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} P_U(x, y)\, \mathrm{d}y\mathrm{d}x \tag{4.5}$$

The spatial standard deviation σ and hard circle radius r are the only free parameters in the respective models subject to optimization. This was done by generating histograms corresponding to the number of live pixels in a single-photon droplet with varying σ and r. Each histogram was compared to a histogram of the experimental single-photon droplets via least squares optimization. The results are summarized in Fig. 4.1. The optimal model is Gaussian with $\sigma = 0.17 \pm 0.01$ pixel widths (PW). The concavity of the RSS dependence on σ indicates this the only optimal value. The exceptionally low RSS value, both in absolute value and compared to the hard circle model, indicates that a Gaussian distribution is the best model for the spatial dependence of the cloud of excited electrons. This thesis henceforth uses the Gaussian model as a *de facto* standard.

4.2 Description of Machine Learning Models

4.2.1 Blind Classifier

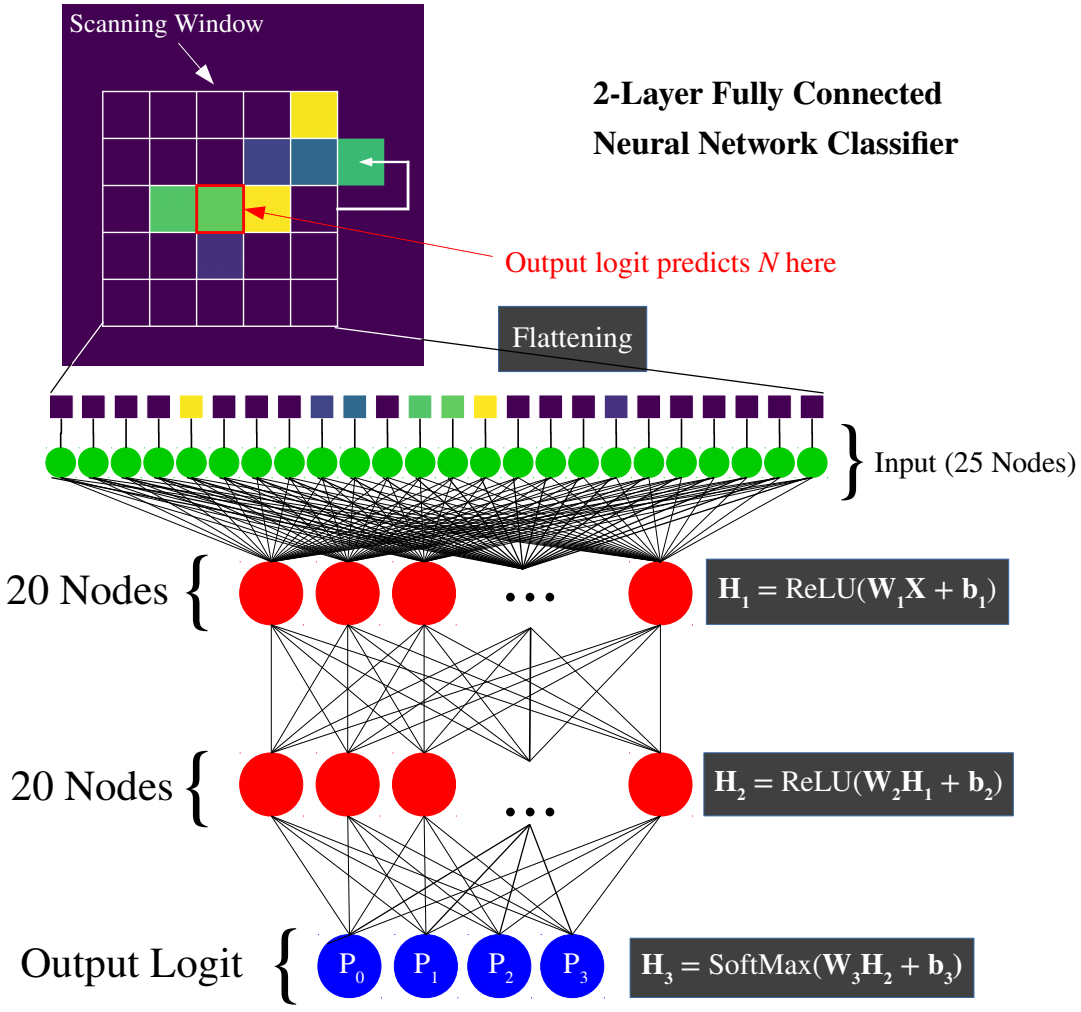


Figure 4.2: Illustration of NNC data-flow. Input is drawn directly from the droplet image (green), fed through two fully connected layers (red), and output as a discrete probability distribution (blue).

The Neural Network Classifier (NNC) is used to estimate a probability distribution over predictions of discrete variables [95]. In this case, the discrete variable is the number of X-

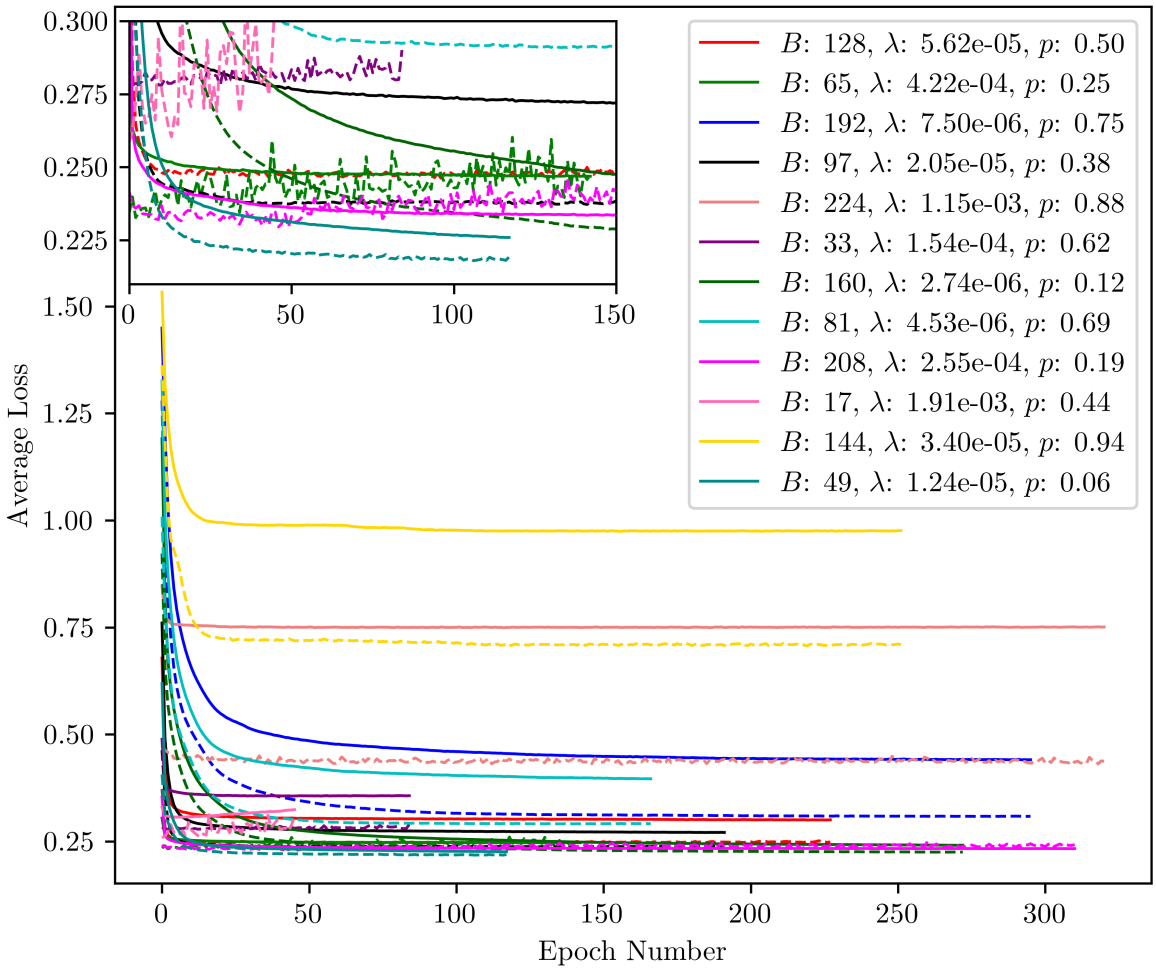


Figure 4.3: Average loss per sample using the categorical cross-entropy loss function. Solid lines denote training loss and dotted lines denote test loss. Each color corresponds to a Sobol distributed combination of batch size B, learning rate λ and dropout p. Inset graph is magnified to show that the best combination of hyperparameters corresponds to 6% dropout, learning rate $\sim$ 1e-5 and batch size $\sim$ 50.

rays N that are incident within the bounds of a single pixel. In principle, N can be arbitrarily large, but for the count rates we were considering ($\bar{k} < 0.01$), the approximation $P(N > 3) = 0$ was employed. Thus, the standard output of the NNC is a 4-vector corresponding to the class probabilities for $N \in [0 - 3]$:

$$\tilde{\mathbf{P}} = (P_0, P_1, P_2, P_3), \quad \sum_{i=0}^{4} P_i = 1 \tag{4.6}$$

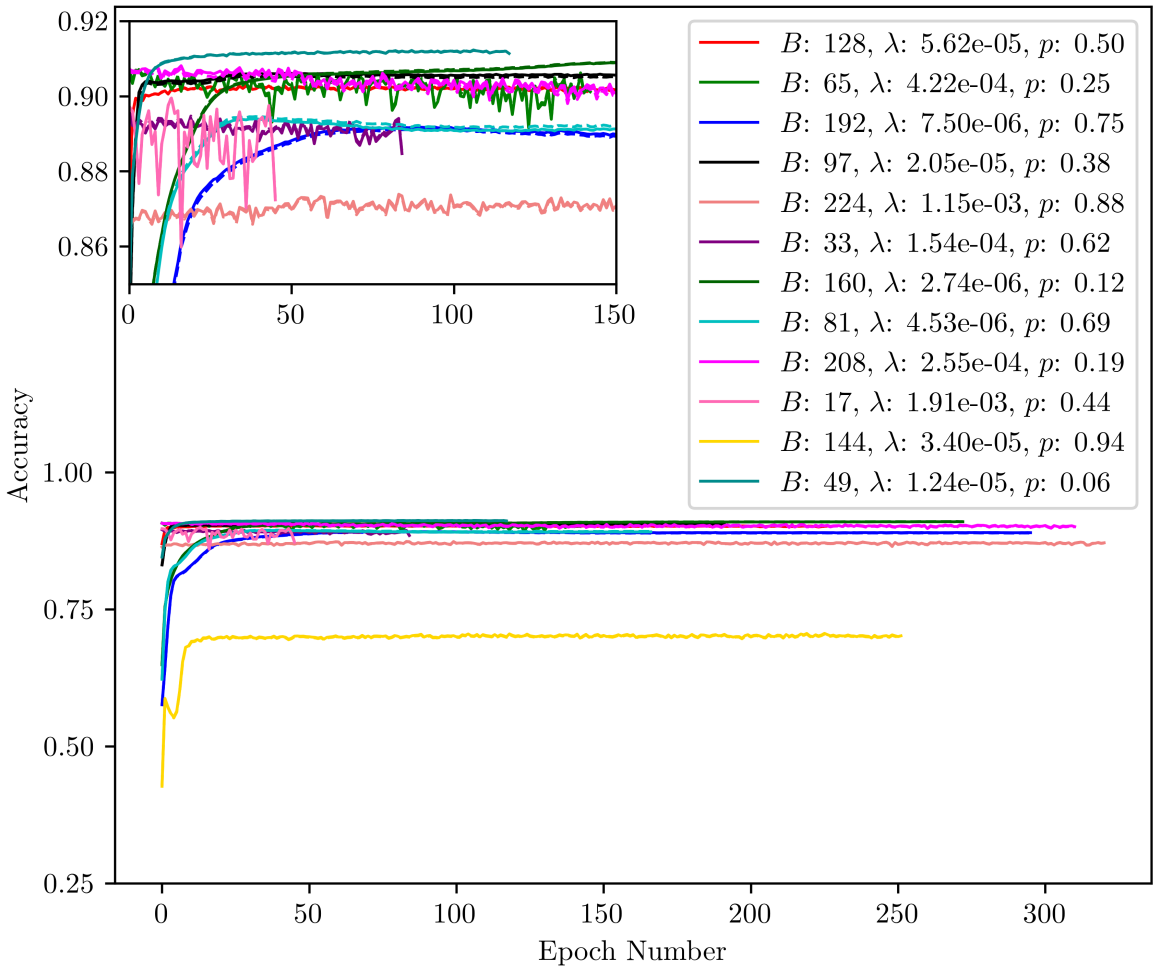


Figure 4.4: Overall classification of NNC accuracy for varying batch size B, learning rate λ and dropout p. The highest performer achieves $\sim 91\%$ accuracy. Accuracy remains consistent when stratified by class.

The total amount of information that can be supplied to the NNC as input is the image of the entire droplet. Because the input image can vary greatly in size (1-200 pixels), an NNC that is capable of processing an arbitrary droplet image would require a high amount of layer nodes, training data, and training time. To manage the overall size and complexity of the NNC, a scanning method was employed. The architecture consists of a feed-forward NNC with 25 input nodes coupled with a 5x5 scanning window that selects pixel values from the droplet image. The 25 input nodes are fed through 2 fully connected, ReLU activated hidden layers $\mathbf{H}_1$ and $\mathbf{H}_2$ with weights $\mathbf{W}_1, \mathbf{W}_2$ and biases $\mathbf{b}_1, \mathbf{b}_2$ respectively. The NNC was

trained to predict the number of X-rays that lie in the central pixel of the 5x5 grid. Thus, the final layer $\mathbf{H}_3$ with weights $\mathbf{W}_3$ and biases $\mathbf{b}_3$ produces 4 raw values given by:

$$\mathbf{H}_3 \equiv (P_0^R, P_1^R, P_2^R, P_3^R) \tag{4.7}$$

The final output logit transforms $\mathbf{H}_3$ via the SoftMax function:

$$P_i = \frac{e^{P_i^R}}{\sum_{i=0}^{3} e^{P_i^R}} \tag{4.8}$$

By scanning the grid over the image, the NNC makes a series of predictions that are then combined to determine the solution for the whole droplet. This architecture is illustrated in Figure 4.2. The NNC is *blind* in that it does not take into account previous predictions, nor is it given droplet image values that lie beyond the extent of the 5x5 grid. Several methods that address these shortcomings are discussed in Chapter 4.2.2.

The NNC evaluates its loss via categorical cross-entropy:

$$L = -\sum_{i=0}^{3} \tilde{Y}_i \log\left(\tilde{P}_i\right) \tag{4.9}$$

where $\tilde{\mathbf{Y}} \equiv (\tilde{Y}_0, \tilde{Y}_1, \tilde{Y}_2, \tilde{Y}_3)$ corresponds to a one-hot class supplied by the training set. The NNC was trained using an AdaM gradient descent optimizer. AdaM is an adaptive learning rate algorithm with three hyperparameters: A main learning rate λ, and two decay parameters ρ_1, ρ_2 [59]. A detailed description of the training set methodology is given in Chapter 4.3. Hyperparameter optimization was performed over minibatch size B, main learning rate λ, and dropout probability p. The decay parameters were held constant at TensorFlow default values of $\rho_1 = 0.9, \rho_2 = 0.999$. The optimization technique was a pseudo-random search. Rather than sampling hyperparameter values from a uniformly random distribution, a pseudo-random Sobol sequence was employed. This exploited the advantages of random

search over grid search, and mitigated the risk of missing large regions of parameter space due to a low sample rate (See Chapter 2.5). Figures 4.3-4.4 show a summary of the network performances over 12 architectures. Accuracy was tested by comparing the class corresponding to the maximum logit value $\max_i(\tilde{P}_i)$ with the class label in the training set. The best performing NNC achieved $\sim 91.2\%$ accuracy over the full training and test sets. The corresponding hyperparameters were:

$$\text{Batch size: } B = 49 \tag{4.10}$$

$$\text{Learning rate: } \lambda = 1.24 \times 10^{-5} \tag{4.11}$$

$$\text{Dropout probability: } p = 0.06 \tag{4.12}$$

Accuracy was not significantly affected by class label. 10-fold cross-validation was implemented, resulting in a normalized accuracy of 0.912 ± 0.001. It should be noted that because some droplets are ambiguous, the accuracy cannot be arbitrarily close to 1. Post-processing techniques for the raw NNC data are discussed in Chapter 4.2.2.

4.2.2 Classifier Boosters

The structure of the NNC is simple and effective, but suffers several shortcomings when applied to a droplet image. The first is that the NNC has no built-in constraints to ensure the number of photons in the droplet is correct. For example, if a given droplet whose total counts add up to 3 is processed through the NNC, the series of predictions given by $\max_i(\tilde{P}_i)$ could result in a total X-ray count of 4 which is incorrect by observation. The other main shortcoming is that accuracy over multiple predictions is diminished. For example, if a given droplet contains 6 activated pixels, the chance that the NNC predicts every pixel correctly is approximately $0.91^6 \sim 57\%$. These issues are loosely related: An incorrect prediction due to bad accuracy over the whole droplet can also miscount the number of photons.

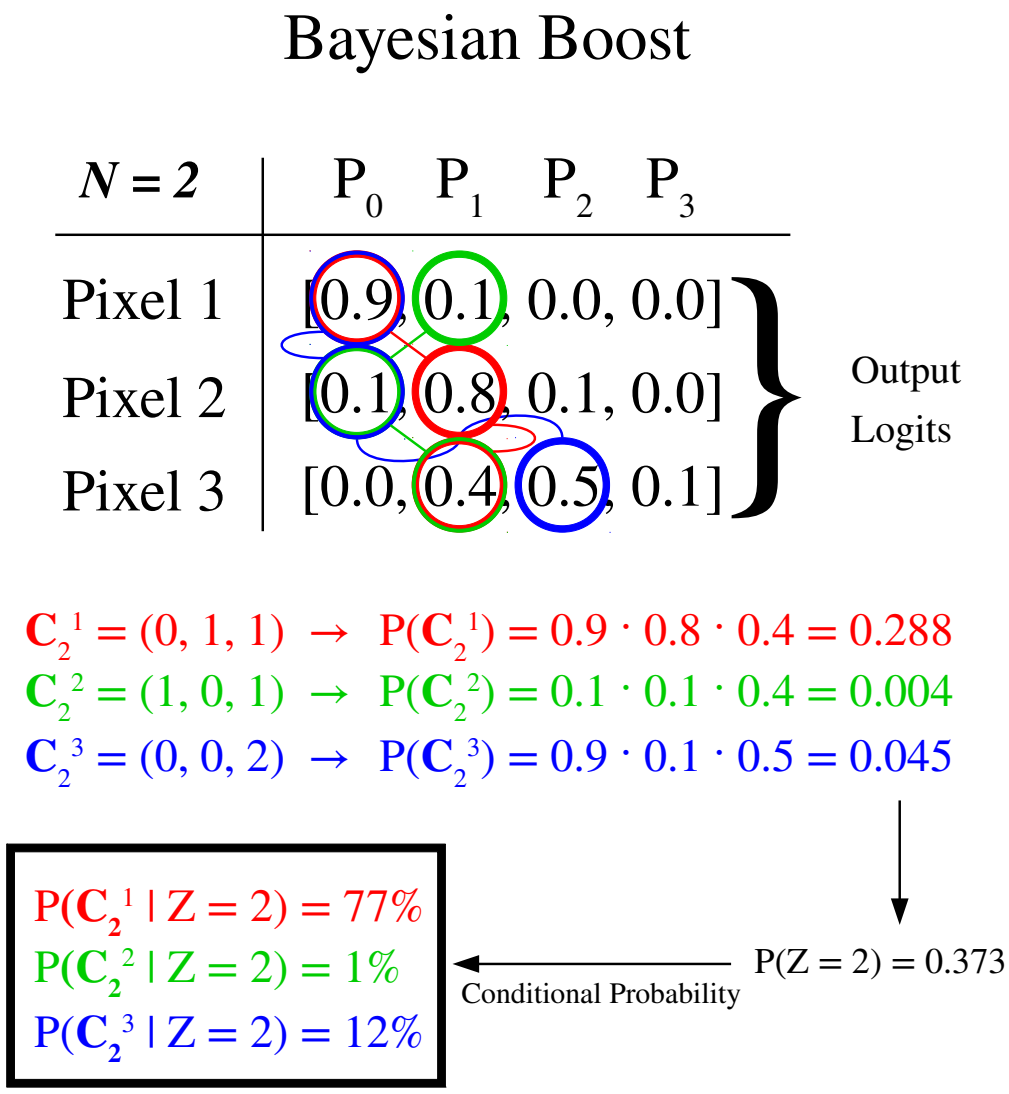


Figure 4.5: Bayesian boost procedure example for a 3-pixel droplet with 2 X-rays. Probability products for different feasible configurations are color coded. Without the boost, maximum likelihood predicts $\mathbf{C}_3 = (0, 1, 2)$ as the most likely X-ray configuration within the droplet. With the added constraint that $N = 2$, $\mathbf{C}_2 = (0, 1, 1)$ is promoted to most likely.

Several post-processing techniques or *boosters* were developed to overcome these shortcomings. The first was a procedural invocation of Bayes theorem. Defining $\mathbf{C}_Z$ as a *unique configuration* of Z X-rays within a droplet and N as the total X-ray count, Bayes theorem takes the form:

$$P(\mathbf{C}_Z|N) = \frac{P(N|\mathbf{C}_Z)P(\mathbf{C}_Z)}{P(N)} \tag{4.13}$$

More information on the format of $\mathbf{C}_Z$ can be found in Chapter 4.3. It is clear that $P(N \neq Z|\mathbf{C}_Z) = 0$ and $P(N = Z|\mathbf{C}_Z) = 1$. Thus, for a droplet containing a total of Z X-rays, Bayes theorem simplifies to:

$$P(\mathbf{C}_Z|Z) = \frac{P(\mathbf{C}_Z)}{P(Z)} \tag{4.14}$$

$$= \frac{P(\mathbf{C}_Z)}{\sum_i P(\mathbf{C}_Z^i)} \tag{4.15}$$

where i indexes every unique configuration containing Z X-rays. To estimate Equation 4.15, the probabilities provided by the output logits of the NNC were used. First, the probabilities of every configuration $\mathbf{C}_Z^i$ are calculated for a given Z by multiplying the appropriate probabilities from each logit. The sum of these probabilities is $P(Z)$, and is used to re-normalize the result. Figure 4.5 shows an example of this calculation in detail.

The Bayesian boost is effective for small droplets (less than 10 pixels), but the complexity of enumerating $P(\mathbf{C}_Z^i)$ for all i goes as $\mathcal{O}(n^2)$ where n is the number of pixels. To solve this issue, a set of conditional *lock-in* and *peeling* boosters were implemented. A lock-in booster assesses the confidence of an NNC output logit, and locks its predicted class if the confidence is sufficiently high ($> 98\%$). For example, if an output logit for an $N = 3$ X-ray droplet reads $[P_0 = 0, P_1 = 0.99, P_2 = 0.01, P_3 = 0]$, the associated pixel will be assumed to have 1 X-ray, and the Bayesian booster will be run on the remaining logits with $N = 2$. One implementation of a lock-in booster reduces the amount of pixels in the Bayesian boost algorithm by one. In practice, the lock-in booster can be used multiple times on the same droplet. A peeling booster looks for specific patterns on the edges of the droplets from which 1-ray droplets can be successively subtracted or "peeled" off. A single application of a peeling booster can remove between one and four pixels depending on the configuration of pixels and the pixel counts within the droplet. Figure 4.6 illustrates the peeling algorithm.

The final booster that was implemented concerns logits for M-ray droplets with non-zero

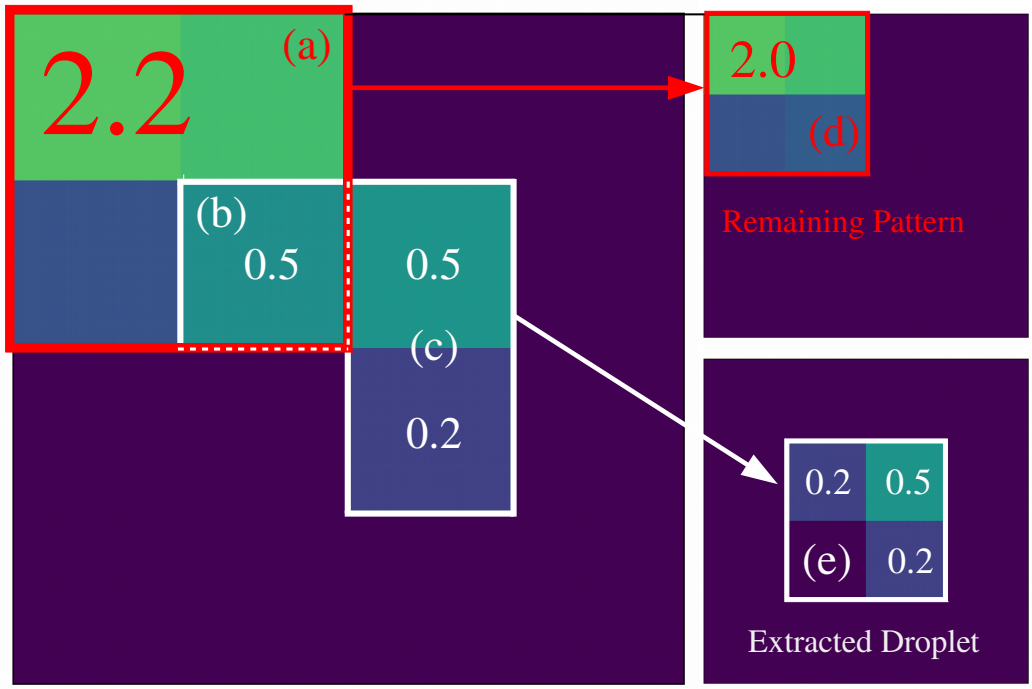


Figure 4.6: Example of a peeling booster applied to a 6-pixel droplet. A single X-ray photon has contributed to the counts in section (c). To leave behind an integer number of photons in section (a), 0.2 counts must be "peeled" away from pixel (b). The resulting extraction gives a 1-ray droplet in section (e) that can be processed via a Mixture Density Network (see Chapter 4.2.3). Section (d) shows the remainder of the droplet, now with 2 fewer pixels. The extracted droplet has 0.9 counts, but this is acceptable given the variance in the X-ray intensity, and the fact that detectors below a 10% threshold are set to zero.

values for $P(N > M)$. This can occur for pixels with high count rates, but results in an underestimation for $P(M = N)$, and to a lesser extent $P(M < N)$. To illustrate this, consider a 2-ray droplet in which one pixel has an activation of 1.7. Evaluating the pixel and its neighbours with the classifier may result in a logit such as $[0, 0.1, 0.8, 0.1]$. The Bayesian booster will discard the value corresponding to $P = 3$, resulting in a normalized logit of $[0, 1/9, 8/9, 0]$. It is likely in this case that $P = 2$ is underestimated, as the relative probability $P(2)/P(1)$ is unchanged. Better results are achieved in this case by simply adding the value of $P(3)$ to $P(2)$, and setting $P(3)$ to zero. Because the output logit can only output probabilities for $N \leq 3$, this booster is only implemented for 2-ray droplets.

4.2.3 Mixture Density Network

The Mixture Density Network (MDN) is used to estimate a conditional probability distribution over predictions of continuous variables [50]. In this case, the continuous variable $\mathbf{R}$ is the vector of 2 values corresponding to the coordinates of the X-ray in a 1-photon droplet, given an input droplet pattern $\tilde{\mathbf{X}}$. Previously, $\tilde{\mathbf{X}}$ represented a 5x5 subset of a droplet image with multiple X-rays. Here, $\tilde{\mathbf{X}}$ is a 3x3 matrix of detector values, and encompasses the entire droplet.

$$\mathbf{R}(\tilde{\mathbf{X}}) = (x, y) \tag{4.16}$$

$\mathbf{R}$ represents not just the incident pixel coordinate, but the precise location of the incident photon within that pixel. Rather than predicting $\mathbf{R}$ directly, the MDN is trained to predict $P_{\boldsymbol{\phi}}(\mathbf{R}; \tilde{\mathbf{X}}, \boldsymbol{\theta})$, where $\boldsymbol{\phi}(\tilde{\mathbf{X}}; \boldsymbol{\theta})$ are the output variables of the ML architecture with internal variables $\boldsymbol{\theta}$ corresponding to the parameters of the probability distribution P. In this ML model, the parameters $\boldsymbol{\phi}$ correspond to the means $\boldsymbol{\mu}$, scale parameters $\boldsymbol{\sigma}$, shape parameters $\boldsymbol{\beta}$ and weights $\boldsymbol{\Pi}$ of a set of M scalar basis functions with 2-dimensional input:

$$P_{\boldsymbol{\phi}}(\mathbf{R}; \tilde{\mathbf{X}}) = \sum_{i=1}^{M} \Pi_i(\tilde{\mathbf{X}}; \boldsymbol{\theta})\, G(\mathbf{R}; \boldsymbol{\mu}_i(\tilde{\mathbf{X}}; \boldsymbol{\theta}), \boldsymbol{\sigma}_i(\tilde{\mathbf{X}}; \boldsymbol{\theta}), \boldsymbol{\beta}_i(\tilde{\mathbf{X}}; \boldsymbol{\theta})) \tag{4.17}$$

Here, the choice of basis function G is the 2-dimensional separable generalized normal distribution defined by:

$$G(\mathbf{R}; \boldsymbol{\theta}) = \frac{\beta_x \beta_y}{4\sigma_x \sigma_y \Gamma(1/\beta_x)\Gamma(1/\beta_y)} \exp\left\{ -\left(\frac{|x-\mu_x|}{\sigma_x}\right)^{\beta_x} - \left(\frac{|y-\mu_y|}{\sigma_y}\right)^{\beta_y} \right\} \tag{4.18}$$

To ensure that P is normalized, $\boldsymbol{\Pi}$ is subject to an additional constraint:

$$\sum_{i=1}^{M} \Pi_i(\tilde{\mathbf{X}}; \boldsymbol{\theta}) = 1 \tag{4.19}$$

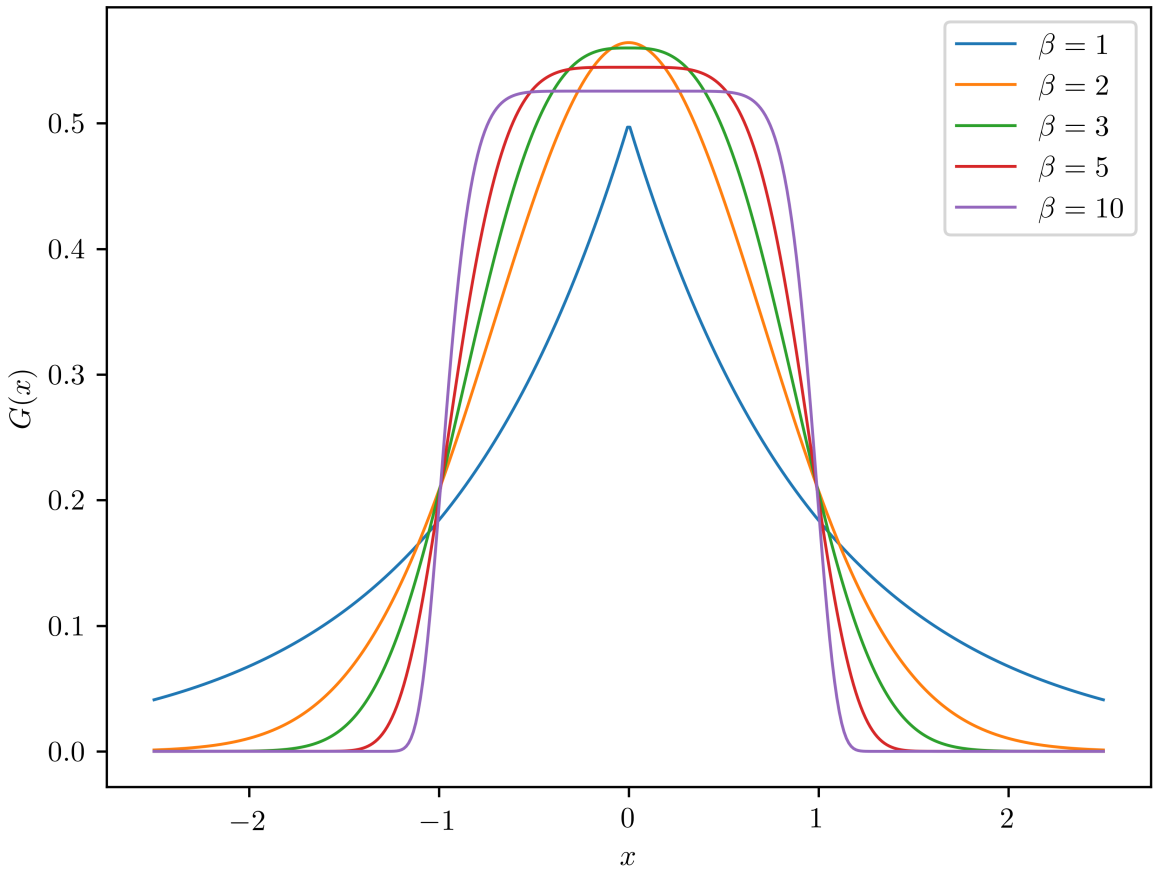


Figure 4.7: 1-D generalized normal distribution $G(x)$. $\beta = 2$ corresponds to a Gaussian distribution with standard deviation $\sigma/\sqrt{2}$. Increasing β drives the distribution toward uniform in the region $-\sigma < x < \sigma$.

Examples of the generalized normal distribution in one dimension for varying β values are given in Figure 4.7. A more general form of the multivariate generalized normal distribution promotes $\boldsymbol{\sigma}$ to a full covariance matrix $\boldsymbol{\Sigma}$, allowing for correlations between the coordinate variables x and y [96]. However, training an MDN to learn a full covariance matrix $\boldsymbol{\Sigma}$ is difficult, in part due to the numerical instability risk and computational cost of inverting it. As a result, it is common to enforce $\boldsymbol{\Sigma}$ to be diagonal [7], allowing for a representation in the form of Equation 4.18. Furthermore, this simplification is suitable for droplet decomposition: Figure 4.8 shows the detector reading S_d as a function of the incident photon coordinate and the gradient magnitude $|\boldsymbol{\nabla}_{\mathbf{R}} S_d|$. The eigenvectors of $\boldsymbol{\Sigma}$ are expected to point in the high symmetry directions of Figure 4.8b as $S_d(\mathbf{R})$ remains constant. Because $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$

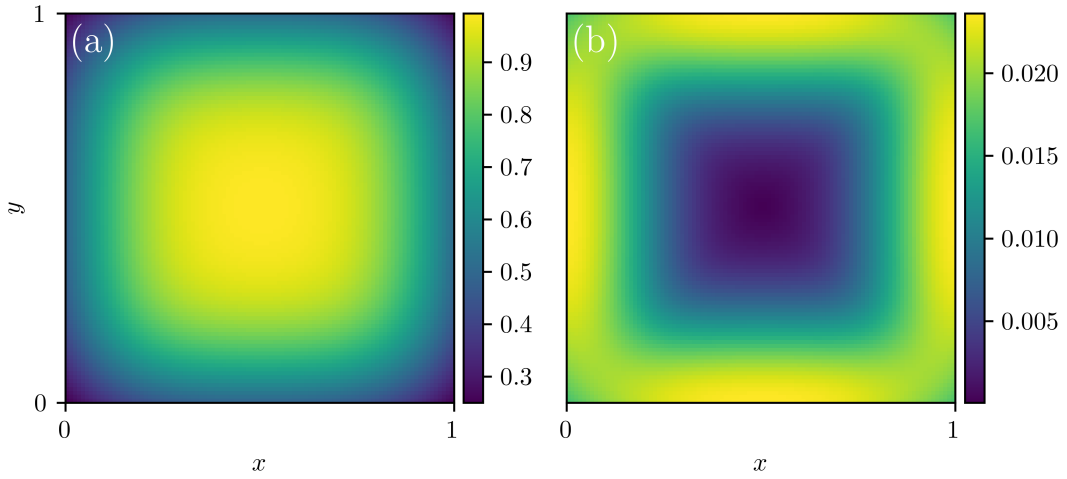


Figure 4.8: (a): Detector reading $S_d(\mathbf{R})$ for incident photon with $I = 1.0$ and $\mathbf{R} \equiv (x, y)$. (b): Gradient magnitude $|\boldsymbol{\nabla}_{\mathbf{R}} S_d(\mathbf{R})|$. Directions of high symmetry point along $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. This figure shows that the digital readings of the CCD are sensitive to photons that are incident on its edges.

correspond to the directions of symmetry, the off diagonal elements of $\boldsymbol{\Sigma}$ will be driven to 0 during training and are thus unnecessary to include as parameters. The structure of the ML architecture is a feed forward neural network with 9 inputs corresponding to the detector measurements in a 3x3 grid, 2 ReLU activated hidden layers $\mathbf{H}_1$ and $\mathbf{H}_2$ with weight matrices and bias vectors $\mathbf{W}_1, \mathbf{b}_1$ and $\mathbf{W}_2, \mathbf{b}_2$ respectively, and 6 sigmoid activated outputs corresponding to the position, scale and shape parameters. The raw, non-activated output vector $\mathbf{H}_3$ has weights $\mathbf{W}_3$ and biases $\mathbf{b}_3$ and produces raw output values given by:

$$\mathbf{H}_3 \equiv (\mu_x^R, \mu_y^R, \sigma_x^R, \sigma_y^R, \beta_x^R, \beta_y^R) \tag{4.20}$$

The weight parameters $\boldsymbol{\Pi}$ were omitted from the architecture, as the output corresponds to the parameters of a single distribution (a homogeneous mixture). The sigmoid activations

that produce the final output are:

$$\mu_{x,y} = 1 + \sigma(\mu^R_{x,y}) \in (1, 2) \tag{4.21}$$

$$\sigma_{x,y} = 0.01 + \sigma(\sigma^R_{x,y}) \in (0.01, 1.01) \tag{4.22}$$

$$\beta_{x,y} = 2 + 100\sigma(\beta^R_{x,y}) \in (2, 102) \tag{4.23}$$

where σ denotes the sigmoid function. Figure 4.9 visualizes and summarizes the data-flow of this structure. The input data for the training process consisted of 100000 1-photon droplets $\mathbf{X}$ and corresponding photon coordinates $\mathbf{R}$ split into a 9:1 ratio for training and test data respectively. The random values for $\mathbf{R}$ were uniformly distributed within the bounds of the central pixel in the 3x3 input grid. The intensity of a given X-ray was Gaussian distributed with $\mu = 1$ and $\sigma = 0.05$. Examples of training data are given in Figure 4.10. For a given training datum, the loss function was the negative log likelihood of the output MDN (Equations 4.17-4.18) evaluated at $\tilde{\mathbf{R}}$:

$$L(\boldsymbol{\theta}; \tilde{\mathbf{R}}, \boldsymbol{\phi}) = -\log\left[\frac{\beta_x\beta_y}{4\sigma_x\sigma_y\Gamma(1/\beta_x)\Gamma(1/\beta_y)}\exp\left\{-\left(\frac{|\tilde{x}-\mu_x|}{\sigma_x}\right)^{\beta_x} - \left(\frac{|\tilde{y}-\mu_y|}{\sigma_y}\right)^{\beta_y}\right\}\right] \tag{4.24}$$

The choice of bounds for $\mu_{x,y}$ in Equation 4.21 limit the photon location to within the central pixel of the input image. The calculation of the inverse of $\sigma_{x,y}$ in Equation 4.24 introduces risk of an exploding gradient instability if $\sigma_{x,y}$ approaches 0. Equation 4.22 includes a lower bound on $\sigma_{x,y}$ to avoid this. The bounds on $\beta_{x,y}$ (Equation 4.23) give the MDN architecture full ability to choose a shape distribution between Gaussian ($\beta = 2$) and uniform with steep drop-off at σ ($\beta >> 2$).

The MDN neural network was trained using an AdaM gradient descent optimizer [59] with default moment parameters. The network performance did not vary significantly with batch size and initial learning rate; the chosen values were 100 and 1e$-$4 respectively. Dropout regularization was not employed. Figure 4.11 shows the average loss and variance per datum

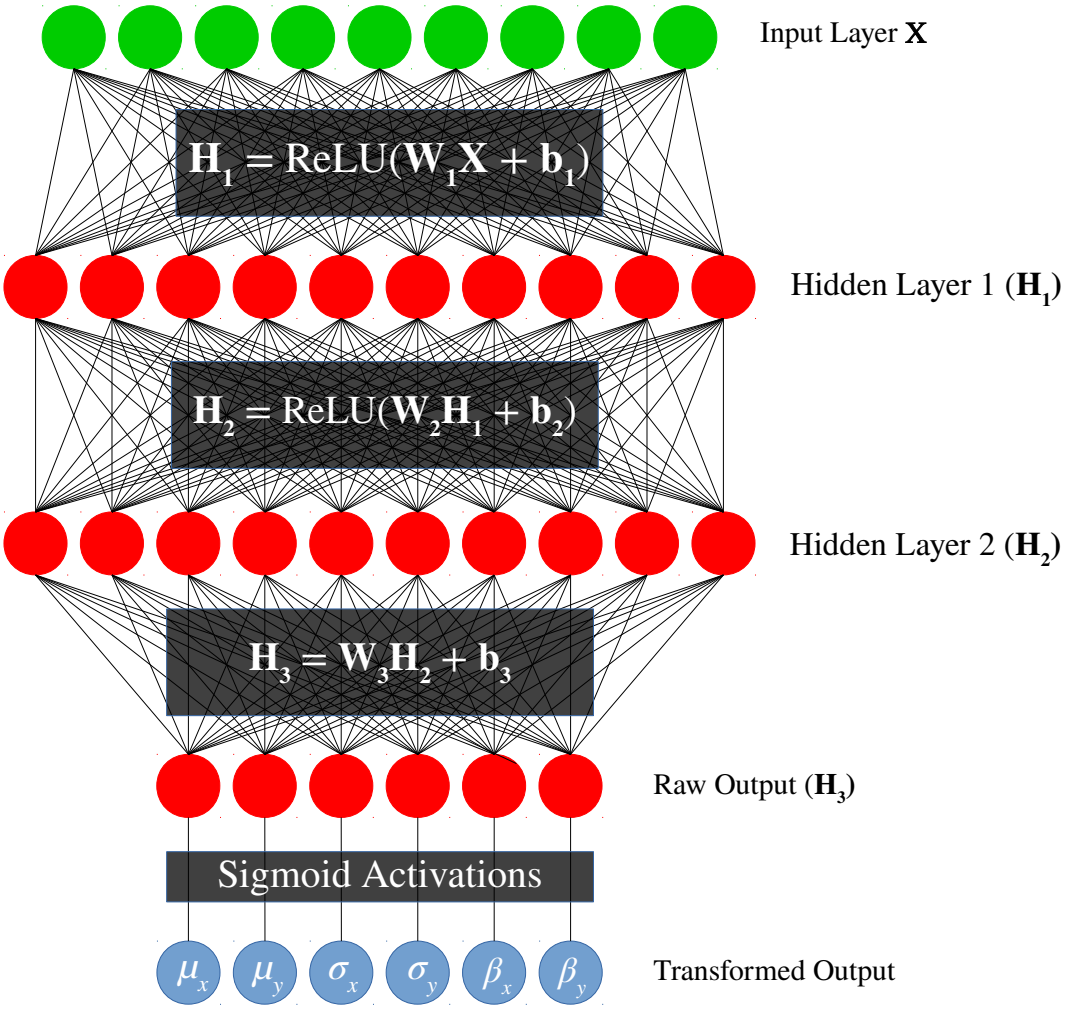


Figure 4.9: Structure of 1-photon droplet MDN. 3x3 droplet pattern is fed into input layer (green), then processed through two fully connected hidden layers (red) and returned as MDN parameters (blue). The sigmoid activations that transform the raw output are given in Equations 4.21-4.23.

for the training and test sets. The optimal state of the neural network was chosen to be at the 800^{th} epoch to optimize the trade-off between reduced loss and increased test variance. Examples of the neural network output are shown in Figure 4.12. In all cases, the value of shape parameter was of order $\beta \sim 10$, implying a uniform distribution within the bounds given by $\sigma_{x,y}$. Cross-validation was not necessary, as the data in the training and tests sets

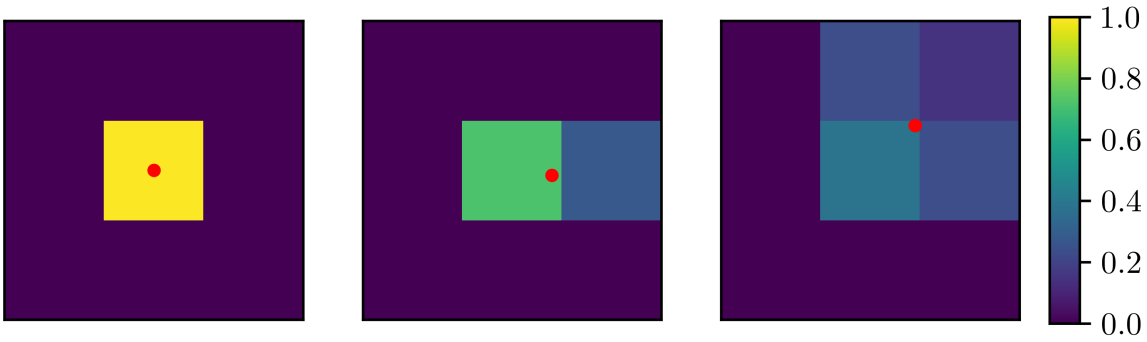


Figure 4.10: Three examples of supervised training vectors. The values in the 3x3 grid are the MDN input values, and the coordinates **R** corresponding to the positions of the red dots are used to evaluate the loss function (Equation 4.24).

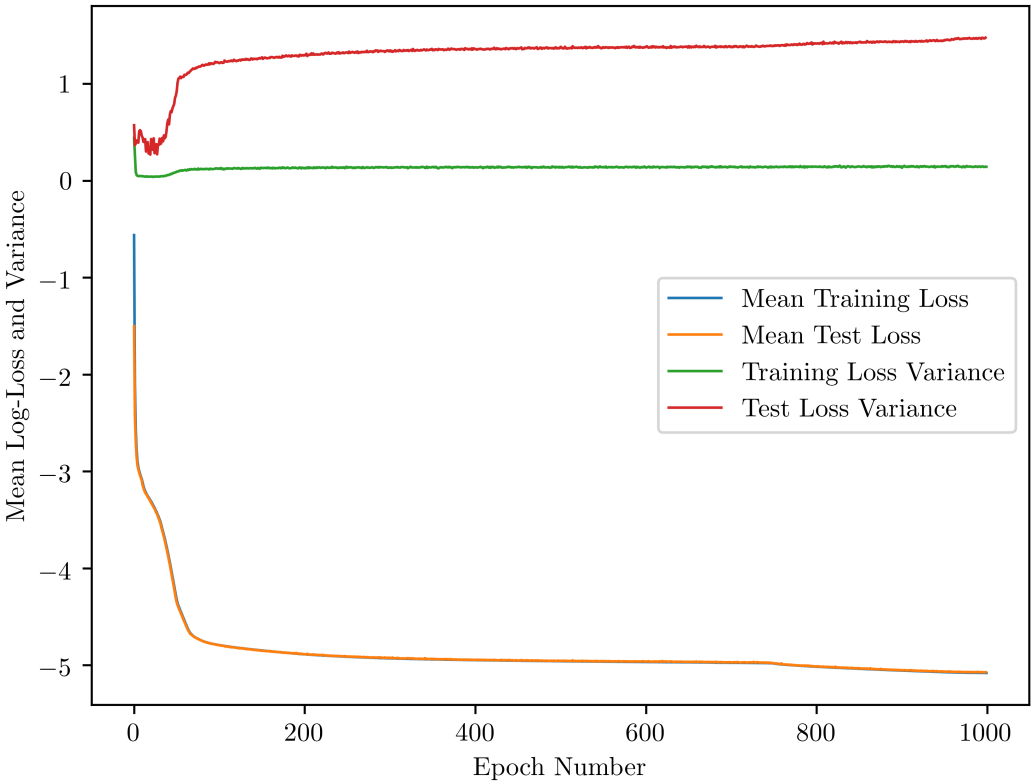


Figure 4.11: Mean loss and variance for training and test data of 1-photon droplet MDN.

spanned the entire input space.

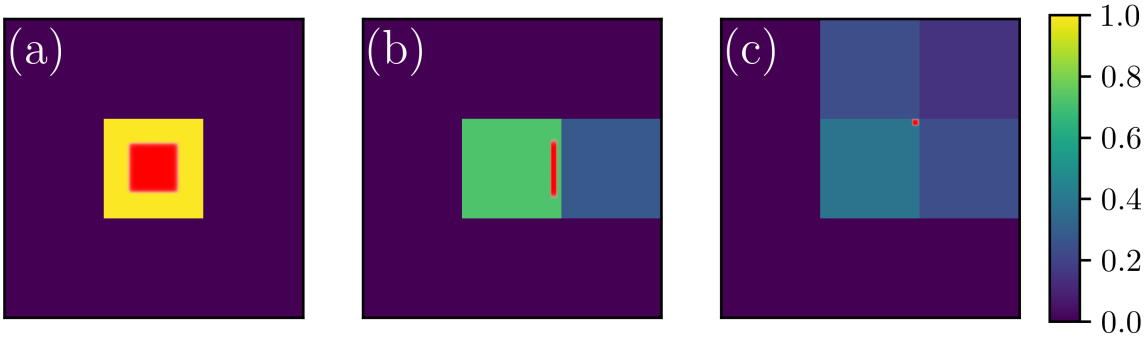


Figure 4.12: Examples of output distributions for 1-photon droplet MDN (red). The incident location of an X-ray that activates a single pixel can be anywhere within the red region shown in (a). Two pixel droplets have low uncertainty in the X-ray position along the axis of the activated pixels as shown in (b). Three or four pixel droplets allow the neural network to determine the incident photon coordinate with high certainty as shown in (c). In general, the shape and position of the predicted probability distribution for a given input will lie in a region of Figure 4.8b in which the gradient magnitude is constant.

4.3 Production of Training Data

In cases where a machine learning algorithm is applied to the inversion of an injective function (i.e. one-to-one mapping), training data should be produced such that the input space is widely and uniformly sampled. However, multiple sets of X-ray coordinates can give rise to similar droplet patterns, due to the ambiguities discussed in Chapter 3. In this non-injective paradigm, a successfully trained feed-forward neural network classifier with sufficient capacity will produce class probabilities that approximate the underlying statistical properties of the training set. However, it is not sufficient to generate training data from a statistical distribution given by Equation 3.8. For values of the average count rate $\bar{k} < 0.01$ and contrast $0 < \beta \leq 1$, P_0 is considerably greater than $P_{1,2,3}$. Thus, a training set produced directly from this distribution will be highly imbalanced. A neural network trained to maximize its accuracy will be biased to predict a class corresponding to the majority of the training set ($N = 0$ in this case). Furthermore, the classifier is only used to solve droplets of sufficient size ($N_{\text{tot}} \geq 2$). This indicates that high count rate samples should be over-represented in the training set.

(a) (N_1, N_2, N_3)	(b) Permutations (P)	(c) $\mathbf{C}_Z$ Example	(d) Training Samples
$(0,0,0)$	1	$(0,0,0,0,0,0,0,0)$	691
$(1,0,0)$	8	$(0,0,0,0,1,0,0,0)$	7,360
$(2,0,0)$	28	$(0,0,1,0,0,1,0,0)$	51,520
$(3,0,0)$	56	$(0,1,1,0,0,1,0,0)$	154,560
$(4,0,0)$	70	$(0,0,0,0,1,1,1,1)$	257,600
$(5,0,0)$	56	$(0,1,0,1,0,1,1,1)$	257,600
$(0,1,0)$	8	$(0,0,2,0,0,0,0,0)$	14,720
$(1,1,0)$	56	$(0,1,0,0,2,0,0,0)$	154,560
$(2,1,0)$	168	$(0,0,0,1,2,1,0,0)$	618,240
$(0,2,0)$	28	$(0,0,2,0,0,0,0,2)$	103,040
$(3,1,0)$	280	$(0,1,2,1,0,0,0,1)$	1,288,000
$(1,2,0)$	168	$(1,0,0,2,0,2,0,0)$	722,800
$(0,0,1)$	8	$(0,0,0,3,0,0,0,0)$	22,080
$(1,0,1)$	56	$(0,1,0,0,0,3,0,0)$	206,080
$(2,0,1)$	168	$(1,1,0,0,0,0,0,3)$	772,800
$(0,1,1)$	56	$(0,0,3,2,0,0,0,0)$	257,600
			Total: 4,889,251

Table 4.1: (a) Number of 1-photon pixels N_1, 2-photon pixels N_2 and 3-photon pixels N_3 in the surrounding ring. (b) Total number of distinct configurations within family. (c) One example of $\mathbf{C}_Z$ within the family of configurations given in (b). (d) Total number of samples generated from family.

A given training sample consists of a 5x5 grid of detector values, and a class value given by the number of X-rays N in the central pixel. Because the sample is meant to represent a portion of a droplet image, a training sample with label N can have more than N photons, as long as the additional photons are not in the central pixel. The training set was generated via a set of hierarchical categories. The first category was the number of X-rays in the central pixel, $N \in [0,3]$. The training samples were split evenly among these four classes to prevent bias in the classifier due to over-representation. Within these classes, the training samples were subdivided into categories defined by the total number of X-rays in the 8 surrounding detectors, $Z \in [0,5]$. The number of training samples in these subcategories were made proportional to Z, ensuring high count-rate droplets have more representation in the training set. The final set of subcategories were defined by the distinct configurations of Z X-rays within the 8 surrounding detectors. A given category can be defined by $(N, \mathbf{C}_Z)$

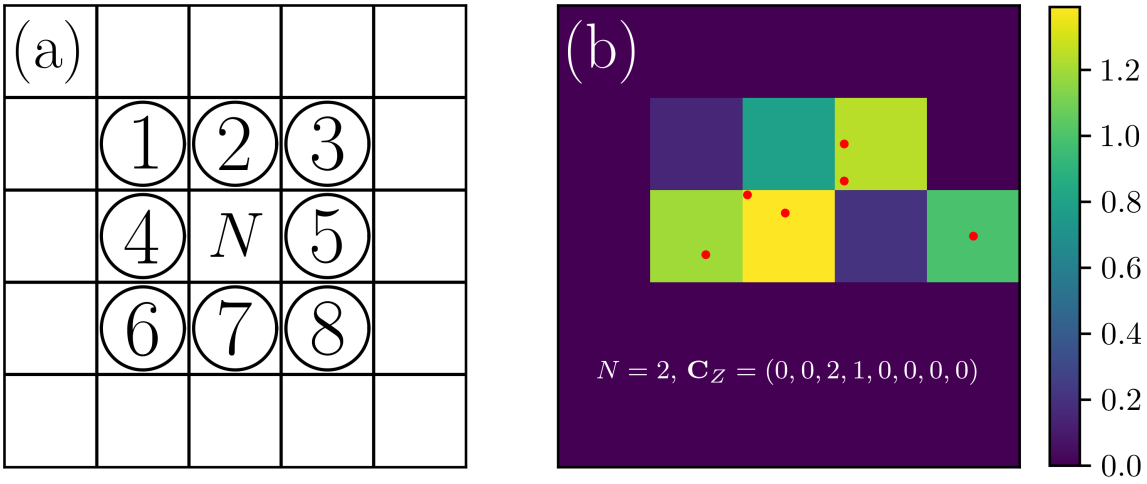


Figure 4.13: (a): Labelling convention for training data categories. (b): Example of training data. The features of the data are the pixel values in the underlying droplet. The red dots show the X-ray coordinates that were used to generate the pattern. The training label N and sub-category $\mathbf{C}_Z$ for this training datum are given according to the convention in (a). The specific class a droplet belongs to only takes into account the number of photons in the central pixel. The training features are the 25 detector values given in the grid, and the corresponding class label is the value of N.

where $\mathbf{C}_Z$ represents an 8-vector $(n_1^Z, n_2^Z, \ldots, n_8^Z)$ corresponding to the number of X-rays in pixels 1-8. $\mathbf{C}_Z$ is further constrained such that:

$$\sum_{i=1}^{8} n_i^Z = Z \tag{4.25}$$

For each unique configuration $(N, \mathbf{C}_Z)$, $230Z$ distinct samples were generated by selecting pixel coordinates $\tilde{\mathbf{R}}$ for each X-ray from a uniform distribution bounded by the edges of the selected pixels. The value of $230Z$ was chosen to bring the total count of training samples close to 5,000,000. The configurations and training samples are summarized in Table 4.1.

For each X-ray configuration enumerated in Table 4.1, between 1 and 3 additional X-rays were randomly added to the 16 pixels in the outermost ring of the training sample with empirically chosen probabilities $P(1) = 0.4, P(2) = 0.5$, and $P(3) = 0.1$. This was done to account for the fact that many experimentally generated droplets will have spatial extent beyond 3 pixels. Figure 4.13 illustrates the categorization method and shows an example of

the training data.

4.4 Model Performance

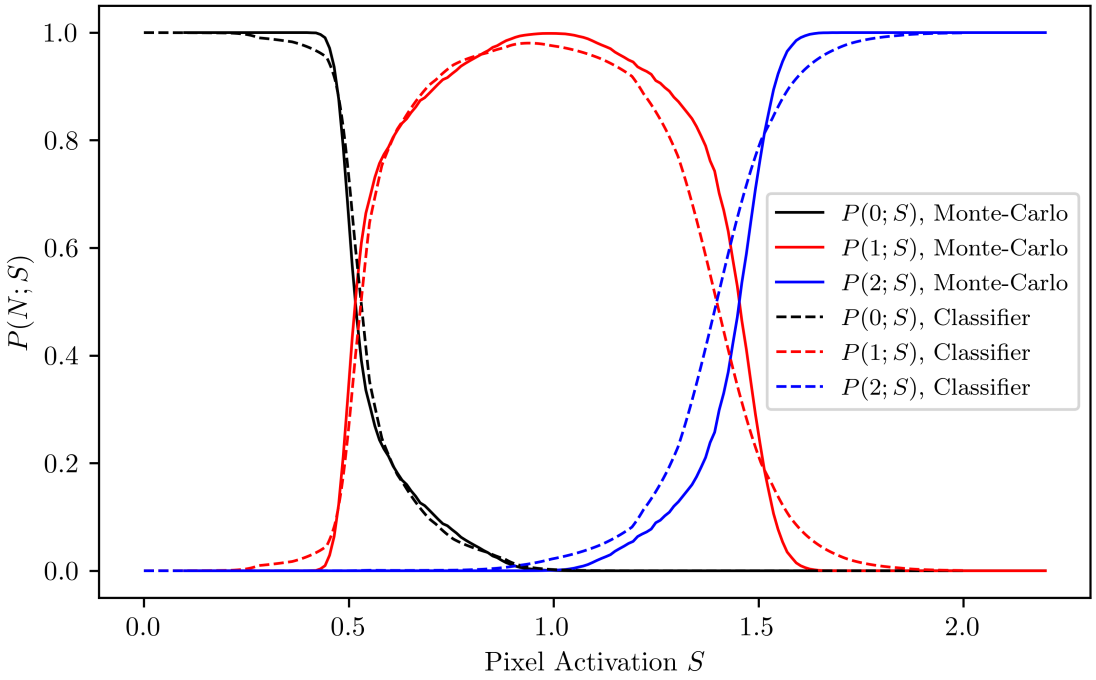


Figure 4.14: Posterior distributions for $P(N)$ for the case of all 2-ray, 2-pixel droplets defined by A, calculated using Monte-Carlo and the ML classifier.

A given droplet pattern defines a posterior probability distribution of possible pixel counts. These distributions can found via Monte-Carlo simulations, and compared to the ML classifier to verify that the classifier approximates the probability of a given X-ray configuration within the droplet. It is infeasible to generate *all* posterior distributions, as the amount of unique droplets is innumerable, so we restrict this analysis to the case of 2-ray, 2-pixel droplets. For low-count rate images, these are the most commonly found droplets that require careful analysis. Examples of droplets that fit into this category are shown in Figures 3.4a and 3.4b. A 2-ray, 2-pixel droplet contains two adjacent pixels with activation values S and $2 - S$. For a given value of S, the discrete posterior is defined as $P(N;S)$, with $N \in \{0, 1, 2\}$. Figure 4.14 shows the Monte-Carlo and classifier results for $P(N;S)$ as a

(a) Number of pixels in data set	$\bar{k}$	(c) β_{true}	(d) $\beta_{\text{predicted}}$
6.0×10^9	1×10^{-3}	0.209	0.21 ± 0.02
1.9×10^9	1.8×10^{-3}	0.254	0.24 ± 0.03
1.8×10^9	1×10^{-3}	1	1.0 ± 0.1
8.9×10^8	1×10^{-3}	1	1.0 ± 0.1
6.0×10^8	3.2×10^{-3}	0.210	0.21 ± 0.02
5.6×10^8	1×10^{-3}	1	1.0 ± 0.1
5.5×10^8	1.8×10^{-3}	0.327	0.29 ± 0.03
4.0×10^8	1×10^{-3}	1	1.0 ± 0.1
2.8×10^8	1.8×10^{-3}	1	1.0 ± 0.1
1.9×10^8	5.6×10^{-3}	0.170	0.16 ± 0.02
1.8×10^8	1.8×10^{-3}	0.866	0.82 ± 0.08
1.7×10^8	3.2×10^{-3}	0.437	0.38 ± 0.04
1.3×10^8	1.8×10^{-3}	0.777	0.74 ± 0.07
8.8×10^7	3.2×10^{-3}	0.565	0.50 ± 0.05
6.0×10^7	1×10^{-2}	0.170	0.17 ± 0.02
5.6×10^7	3.2×10^{-3}	1	1.0 ± 0.1
5.5×10^7	5.6×10^{-3}	0.481	0.46 ± 0.04
4.0×10^7	3.2×10^{-3}	1	1.0 ± 0.1
2.8×10^7	5.6×10^{-3}	0.594	0.56 ± 0.06
1.7×10^7	5.6×10^{-3}	0.835	0.68 ± 0.07
1.7×10^7	1×10^{-2}	0.352	0.35 ± 0.04
1.2×10^7	5.6×10^{-3}	1	0.93 ± 0.09
8.0×10^6	1×10^{-2}	0.0.535	0.45 ± 0.05
5.0×10^6	1×10^{-2}	0.0.636	0.53 ± 0.05
4.0×10^6	1×10^{-2}	1	0.96 ± 0.09

Table 4.2: Contrast factor $\beta_{\text{predicted}}$ determined from data sets generated using the model in Chapter 4.1. The histogram corresponding to the true class labels gives a value of β_{true} that agrees with $\beta_{\text{predicted}}$ within uncertainty. Data is presented graphically in Figure 4.17.

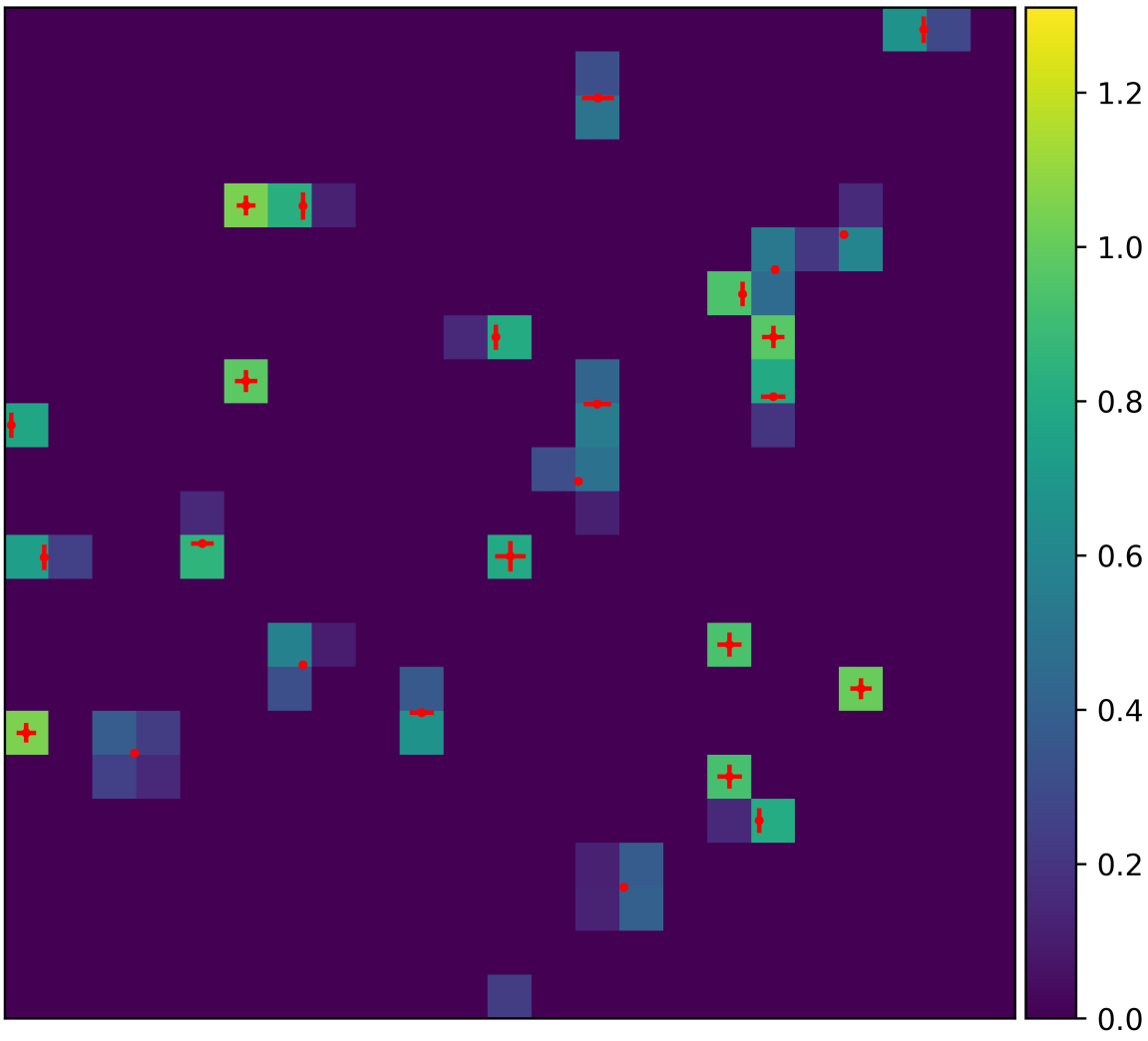


Figure 4.15: Portion of artificial speckle image processed with the blind classifier, the boosters and the MDN. Previously, we have presented the X-ray positions using a continuous probability density function (PDF). Here we use error bars along $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ to represent the extent σ of the probability distribution outputted by the MDN. The uncertainties for X-ray positions vary depending on the droplet configurations. In all cases, the shape parameter β is high, meaning that the uncertainties are "hard" and not Gaussian.

function of S for $N = 0, 1, 2$. The graphs are in good agreement, implying the classifier has learned to approximate the correct posterior distributions for this case.

Figure 4.15 shows the full capability of the system consisting of the blind classifier, the classifier boosters, and the mixture density network. The statistical parameters of the artificially generated speckle pattern were $\bar{k} = 0.05, \beta = 0.3$. The uncertainties on the X-ray positions

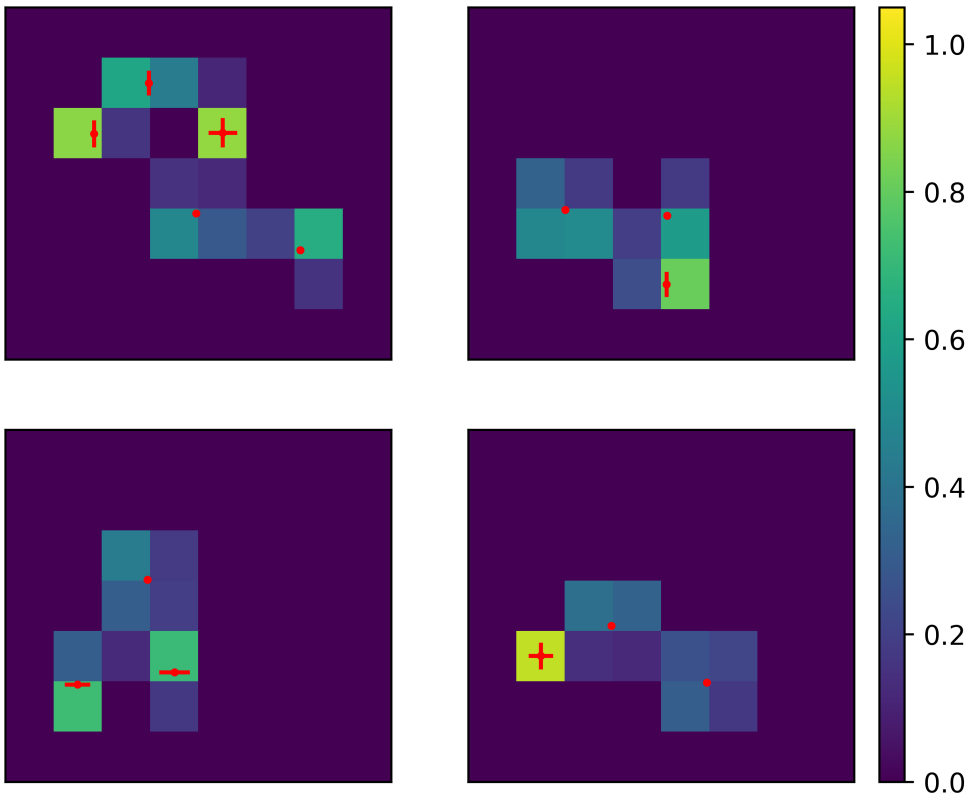


Figure 4.16: Large experimentally measured droplets processed with the full ML system. In the regime of $\bar{k} \leq 0.01$, these droplets are uncommon and represent the most challenging to decompose.

vary depending on the posterior probability distributions defined by the droplet. Figure 4.16 shows the same system applied to some experimentally measured droplets on the large end of the spectrum (> 5 pixels, > 2 X-rays).

With respect to the objective of determining a histogram that can be fit to Equation 3.8, the mixture density network is unnecessary, as the histogram only tabulates the X-ray counts within the bounds of a pixel, and not their exact incident location. Determining the average count rate $\bar{k}$ is straightforward: One can sum the total pixel counts and divide by the number of pixels. However, due to the pixel cutoff value typically around 10% of an average X-ray activation, this will systematically underestimate the true count rate. In this analysis, the average count rate was calculated by determining the integer number of photons in each droplet, and dividing the sum of those results by the total number of pixels. Table 4.2 and

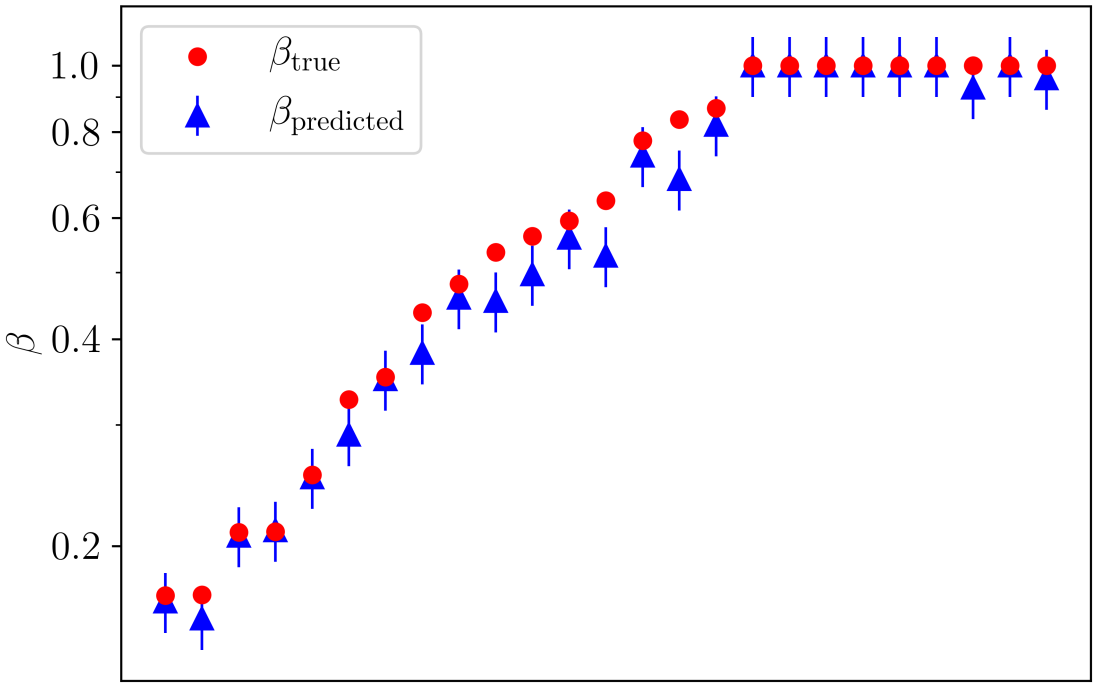


Figure 4.17: Comparison of $\beta_{\text{predicted}}$ and β_{true} by plotting in ascending order. The values of $\bar{k}$ vary between 1e-3 and 1e-2.

Figure 4.17 show the NNC predictions for the speckle contrast β from 25 data sets generated using the model in Chapter 4.1. The values of $\bar{k}$ and β that were used to generate the data sets were within the range defined by:

$$\bar{k} \in (10^{-3}, 10^{-2})$$

$$\beta \in (0.2, 1.0)$$

The range of $\bar{k}$ defines the weak field limit, and the range of β was chosen to correspond to two full coherence intervals in time. The training labels were used to generate histograms of the *true* pixel counts, and the contrast factor β_{true} was fit to these via maximum likelihood. The training features were processed by the NNC and used to generate *predicted* histograms of pixel counts, and the contrast factor $\beta_{\text{predicted}}$ was fit to these histograms. The uncertainties in $\beta_{\text{predicted}}$ were determined using Equation 3.25. The values for $\beta_{\text{predicted}}$ show good

agreement with β_{true} within uncertainty, indicating the NNC is approximating the contrast to a satisfactory degree.

5 Conclusion

In this thesis, we have developed a model for producing speckle patterns that would be typically measured in a CCD-based area detector for X-ray synchrotron sources. Our model incorporates the statistical parameters of average count rate $\bar{k}$, contrast β, average intensity I, and electron cloud spatial standard deviation σ. With this model, we were able to produce a rich training set for a neural network classifier. This enabled it to approximate the probabilities of distinct photon configurations given raw droplet data. The neural network classifier provided an accurate estimation of speckle contrast for area detector images that obscured the underlying counting statistics due to additive interference.

In general, the physical attributes of an X-ray source and an area detector can vary from system to system, as can the experimental methodology. The ML architectures designed in this thesis correspond to only one such case. To extend this algorithm to fit the needs of an arbitrary system, the following steps must be taken:

- Normalize the XPCS data by determining the average count-rate of a single photon via a histogram fit of droplet counts (See Chapter 3.4 and Figure 3.3).
- Determine the pixel cutoff value corresponding to the user's desired level of noise reduction.
- Determine the spatial standard deviation of the electrons excited by an incident photon. This can be done by fitting a histogram of the number of live pixels in a 1-ray droplet (See Chapter 4.1).
- Generate a droplet training set using the methods in Chapter 4.3.
- Train the neural network using the ideal hyperparameters found in Chapter 4.2.1. If the provided hyperparameters result in bad performance, retrain with a set of Sobol distributed hyperparameters to find the optimal algorithm.

With this done, the general algorithm to process an area detector and determine a histogram of photon counts is as follows:

1. Require average count-rate S_d.
2. Require detector image $\mathbf{A}$.
3. $\mathbf{A} \leftarrow \mathbf{A}/S_d$.
4. Initialize histogram for $(0, 1, 2, 3)$ photon counts: $\mathbf{H} = [0, 0, 0, 0]$.
5. Define N_p as the number of pixels in the detector image.
6. Use an image labelling algorithm to isolate each droplet and the coordinates of its pixels. Define $\mathbf{D}$ as a tensor containing all the droplet subsets of $\mathbf{A}$. Because the droplets form a subset of the total detector image $\mathbf{A}$, the remaining pixels will be processed later.
7. ***for*** $\tilde{\mathbf{D}}$ ***in*** $\mathbf{D}$ ***do***:
   - Initialize a matrix of zeros $\mathbf{C}$ with the same shape as $\tilde{\mathbf{D}}$. This matrix will store the predicted integer number of X-rays in each pixel.
   - Use peeling booster to draw a droplet $\tilde{\mathbf{D}}_{\text{peel}}$ out of $\tilde{\mathbf{D}}$. Add 1 to $\mathbf{C}[\text{argmax}(\tilde{\mathbf{D}}_{\text{peel}})]$. Repeat this step until no more droplets can be peeled from $\tilde{\mathbf{D}}$.
   - Generate a list of logits $\tilde{\mathbf{P}}$ by calling the NNC on every live pixel in $\tilde{\mathbf{D}}$. The NNC will also read the 24 surrounding pixels.
   - Process $\tilde{\mathbf{P}}$ with the Bayesian booster. This will produce a series of n candidate configurations $\mathbf{C}_i$ and corresponding probabilities P_i.
   - $\mathbf{C} \leftarrow \mathbf{C} + \mathbf{C}_m$ where $m = \text{argmax}(P_1, P_2, \ldots, P_n)$.
   - Take a histogram $\tilde{\mathbf{H}}$ of $\mathbf{C}$.
   - $\mathbf{H} \leftarrow \mathbf{H} + \tilde{\mathbf{H}}$

8. $\mathbf{H}[0] \leftarrow \mathbf{H}[0] + N_p - \text{size}(\mathbf{D})$. This ensures that all the remaining pixels in the image contribute to the histogram.

We have thus demonstrated that this method shows promise for accurately determining speckle contrast from experimental XPCS images, allowing for a more precise studies of the dynamical and structural properties of a wide range of systems.

Appendix

The computational programs designed in the preparation of this thesis were written in the Python 3 programming language [97]. The machine learning algorithms were designed and trained within Google's TensorFlow API [98].

The work in this thesis was performed using three computational packages developed by Daniel Abarbanel that have been made publicly available. They are:

- `produce_training_data`: This package contains the routines that produce training data according to the methods in Chapter 4.3
- `train_neural_net`: This package contains the neural network classes and training routines that were used in this thesis.
- `speckle_data_analysis`: This package contains the routines for generating artificial speckle images, and determining the corresponding histograms of photon counts using a pre-trained classifier.

In each of these computational packages, there is a notebook that demonstrates how to operate the software. The notebook is written using the Jupyter Notebook format. This requires the installation of the `python-jupyter` package, which can be found at:

`http://www.jupyter.org/`

The packages are publicly available at:

`https://abarbadan@bitbucket.org/abarbadan/speckle_ml.git`

References

[1] N. M. Nasrabadi, Pattern recognition and machine learning, Journal of Electronic Imaging **16**, 049901 (2007).

[2] P. Duygulu, K. Barnard, J. F. G. de Freitas, and D. A. Forsyth, Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary, in *European conference on computer vision*, pages 97–112, Springer, 2002.

[3] D. Bahdanau, K. Cho, and Y. Bengio, Neural machine translation by jointly learning to align and translate, arXiv preprint arXiv:1409.0473 (2014).

[4] E. Rosten, R. Porter, and T. Drummond, Faster and better: A machine learning approach to corner detection, IEEE Transactions on Pattern Analysis and Machine Intelligence **32**, 105–119 (2010).

[5] R. W. Parks, D. S. Levine, and D. L. Long, *Fundamentals of neural network modeling: Neuropsychology and cognitive neuroscience*, MIT Press, 1998.

[6] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bulletin of Mathematical Biophysics **5**, 115–133 (1943).

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, The MIT Press, 2016.

[8] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review **65**, 386–408 (1958).

[9] D. S. Touretzky and G. Hinton, Symbols among the neurons: Details of a connectionist inference architecture, in *IJCAI*, volume 85, pages 238–243, 1985.

[10] D. E. Rumelhart, G. Hinton, and R. J. Williams, Learning representations by back-propagating errors, Nature **323**, 533 (1986).

[11] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, Support vector machines, IEEE Intelligent Systems and their applications **13**, 18–28 (1998).

[12] G. Hinton, S. Osindero, and Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Computation **18**, 1527–1554 (2006).

[13] Y. Bengio and Y. LeCun, Scaling learning algorithms towards AI, Large-scale Kernel Machines **34** (1-41 2007).

[14] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhouke, P. Nguyen, T. N. Sainath, and B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, IEEE Signal Processing Magazine **29**, 82–97 (2012).

[15] 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*, pages 770–778, 2016.

[16] 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).

[17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, Mastering the game of Go without human knowledge, Nature **550**, 354–359 (2017).

[18] E. L. Paula, M. Ladeira, R. N. Carvalho, and T. Marzagão, Deep learning anomaly detection as support fraud investigators in Brazilian exports and anti-money laundering, in *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pages 954–960, IEEE, 2016.

[19] J. B. Heaton, N. G. Polson, and J. H. Witte, Deep learning for finance: deep portfolios, Applied Stochastic Models in Business and Industry **33**, 3–12 (2017).

[20] A. A. Cruz-Roa, J. E. A. Ovalle, A. Madabhushi, and F. A. G. Osorio, A deep learning architecture for image representation, visual interpretability and automated basal-cell carcinoma cancer detection, in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 403–410, Springer, 2013.

[21] H. R. Roth, Y. Wang, J. Yao, L. Lu, J. E. Burns, and R. M. Summers, Deep convolutional networks for automated detection of posterior-element fractures on spine CT, in *Medical Imaging 2016: Computer-Aided Diagnosis*, volume 9785, page 97850P, International Society for Optics and Photonics, 2016.

[22] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, Fast and accurate modeling of molecular atomization energies with machine learning, Physical Review Letters **108**, 058301 (2012).

[23] C. J. Long, J. Hattrick-Simpers, M. Murakami, R. C. Srivistava, and I. Takeuchi, Rapid structural mapping of ternary metallic alloy systems using the combinatorial approach and cluster analysis, Review of Scientific Instruments **78**, 072217 (2007).

[24] J. Zhou and O. G. Troyanskaya, Predicting effects of noncoding variants with deep learning-based sequence model, Nature Methods **12**, 931–934 (2015).

[25] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning, Nature Biotechnology **33**, 831 (2015).

[26] L. Wang, Discovering phase transitions with unsupervised learning, Physical Review B **94**, 195105 (2016).

[27] Z. Li, R. Kermode, and A. De Vita, Molecular dynamics with on-the-fly machine learning of quantum-mechanical forces, Physical Review Letters **114**, 096405 (2015).

[28] M. Benerji, O. Lahav, C. J. Lintott, F. B. Abdalla, K. Schawinski, S. P. Bamford, D. Andreescu, P. Murray, M. J. Raddick, A. Slosar, A. Szalay, D. Thomas, and J. Vandenberg, Galaxy zoo: reproducing galaxy morphologies via machine learning, Monthly Notices of the Royal Astronomical Society **406**, 342–353 (2010).

[29] J. Lee, A. Seko, K. Shitara, K. Nakayama, and I. Tanaka, Prediction model of band gap for inorganic compounds by combination of density functional theory calculations and machine learning techniques, Physical Review B **93**, 115104 (2016).

[30] A. G. Kusne, T. Gao, A. Mehta, L. Ke, M. C. Nguyen, K.-M. Ho, V. Antropov, C.-Z. Wang, M. J. Kramer, C. Long, and I. Takeuchi, On-the-fly machine-learning for high-throughput experiments: search for rare-earth-free permanent magnets, Scientific Reports **4**, 6367 (2014).

[31] F. M. Grimaldi, *Physico-mathesis de lumine, coloribus et iride*, Victorii Benatii, 1665.

[32] A. R. Hall, Beyond the Fringe: Diffraction as Seen by Grimaldi, Fabri, Hooke and Newton, Notes and Records of the Royal Society of London **44**, 13–23 (1990).

[33] I. Newton, A. Einstein, E. Whittaker, I. B. Cohen, and D. H. D. Roller, *Opticks: Or, a treatise of reflection, refractions, inflections and colours of light*, Dover, 1952.

[34] T. Young, II. The Bakerian Lecture. On the theory of light and colours, Philosophical transactions of the Royal Society of London **92**, 12–48 (1802).

[35] M. Born and E. Wolf, *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*, Elsevier, 2013.

[36] M. Etter and R. E. Dinnebier, A century of powder diffraction: a brief history, Zeitschrift für anorganische und allgemeine Chemie **640**, 3015–3028 (2014).

[37] W. L. Bragg, The structure of some crystals as indicated by their diffraction of X-rays, Proc. R. Soc. Lond. A **89**, 248–277 (1913).

[38] B. E. Warren, *X-ray Diffraction*, Courier Corporation, 1969.

[39] T. H. Maiman, Stimulated optical radiation in ruby, (1960).

[40] C. H. Townes and A. L. Schawlow, *Microwave spectroscopy*, Courier Corporation, 2013.

[41] J. Miao, Coherent diffraction imaging, Microscopy and Microanalysis **20**, 368–369 (2014).

[42] M. Sutton, S. G. J. Mochrie, T. Greytak, S. E. Nagler, L. E. Berman, G. A. Held, and G. B. Stephenson, Observation of speckle by diffraction with coherent X-rays, Nature **352**, 608–610 (1991).

[43] J. C. Dainty, *Laser speckle and related phenomena*, volume 9, Springer Science & Business Media, 2013.

[44] S. O. Hruszkewycz, M. Sutton, P. H. Fuoss, B. Adams, S. Rosenkranz, K. F. Ludwig Jr., W. Roseker, D. Fritz, M. Cammarata, D. Zhu, S. Lee, H. Lemke, C. Gutt, A. Robert, G. Grübel, and G. B. Stephenson, High Contrast X-ray Speckle from Atomic-Scale Order in Liquids and Glasses, Physical Review Letters **109**, 185502 (2012).

[45] D. Abarbanel, M. Sutton, and H. Guo, Artificial neural networks for the determination of speckle contrast in split-pulse XPCS measurements, to be published (2018).

[46] A. Zellner, Bayesian estimation and prediction using asymmetric loss functions, Journal of the American Statistical Association **81**, 446–451 (1986).

[47] P. J. Huber, The 1972 Wald Lecture robust statistics: A review, The Annals of Mathematical Statistics **43**, 1041–1067 (1972).

[48] T. Chai and R. R. Draxler, Root mean square error (RMSE) or mean absolute error (MAE)? Arguments against avoiding RMSE in the literature, Geoscientific Model Development **7**, 1247–1250 (2014).

[49] A. Y. Ng and M. I. Jordan, On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes, in *Advances in Neural Information Processing Systems*, pages 841–848, 2002.

[50] C. M. Bishop, Mixture Density Networks, Technical report, Aston University, Birmingham, 1994.

[51] G. Cybenko, Approximations by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems **2**, 183–192 (1989).

[52] A. C. Marreiros, J. Daunizeau, S. J. Kiebel, and K. J. Friston, Population dynamics: Variance and the sigmoid activation function, Neuroimage **42**, 147–157 (2008).

[53] K.-B. Duan and S. S. Keerthi, Which is the best multiclass SVM method? An empirical study, in *International Workshop on Multiple Classifier Systems*, pages 278–285, Springer, 2005.

[54] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, Efficient backprop, in *Neural Networks: Tricks of the Trade*, pages 9–48, Springer, 2012.

[55] R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, and R. J. Douglas, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit, Nature **405**, 947–951 (2000).

[56] J. Ling, A. Kurzawski, and J. Templeton, Reynolds average turbulence modemodel using deep neural networks with embedded invariance, Journal of Fluid Mechanics **807**, 155–166 (2016).

[57] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient method for online learning and stochastic optimization, Journal of Machine Learning Research **12**, 2121–2159 (2011).

[58] T. Tieleman and G. Hinton, Lecture 6.5 - RMSProp: Divide the gradient by the running average of its recent magnitude, COURSERA: Neural Networks and Machine Learning

4, 26–31 (2012).

[59] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint , arXiv: 1412.6980 (2014).

[60] N. Srivistava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research **15**, 1929–1958 (2014).

[61] J. D. Rodriguez, A. Perez, and J. A. Lozano, Sensitivity analysis of k-fold cross validation in prediction error estimation, IEEE Transactions on Pattern Analysis and Machine Intelligence **32**, 569–575 (2010).

[62] A. Y. Ng, Feature selection, L1 vs L2 regularization and rotational invariance, in *Proceedings of the twenty-first international conference on machine learning*, page 78, 2004.

[63] G. Obozinski, B. Taskar, and M. Jordan, Multi-task feature selection, Technical report, Statistics Department, UC Berkeley, 2006.

[64] T. Hothorn and B. Lausen, Double-bagging: combining classifiers by bootstrap aggregation, Pattern Recognition **36**, 1303–1309 (2003).

[65] A. Rizzoli, S. Merler, C. Furlanello, and C. Genchi, Geographical information systems and bootstrap aggregation (bagging) of tree-based classifiers for lyme disease risk prediction in Trentino, Italian Alps, Journal of Medical Entomology **39**, 485–492 (2002).

[66] M. Pal, Random forest classifier for remote sensing classification, International Journal of Remote Sensing **26**, 217–222 (2003).

[67] G. E. Hinton, N. Srivistava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint , arXiv: 1207.0580 (2012).

[68] D. Warde-Farley, I. Goodfellow, A. Courville, and Y. Bengio, An empirical analysis of dropout in piecewise linear networks, arXiv preprint , arXiv: 1312.6197 (2013).

[69] Y. Yao, L. Rosasco, and A. Caponnetto, On early stopping in gradient descent learning, Constructive Approximation **26**, 289 (2007).

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, pages 1097–1105, Curran Associates, Inc., 2012.

[71] T. Fawcett and F. J. Provost, Combining data mining and machine learning for effective user profiling, in *KKD*, pages 8–13, 1996.

[72] F. Provost and T. Fawcett, Robust classification for imprecise environments, Machine Learning **42**, 203–231 (2001).

[73] N. Japkowicz and S. Stephen, The class imbalance problem: A systematic study, Intelligent Data Analysis **6**, 429–449 (2002).

[74] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, Journal of Artificial Intelligence Research **16**, 321–357 (2002).

[75] D. MacLaurin, D. Duvenaud, and R. Adams, Gradient-based hyperparameter optimization through reversible learning, in *International Conference on Machine Learning*, pages 2113–2122, 2015.

[76] J. Snoek, H. Larochelle, and R. Adams, Practical Bayesian optimization of machine learning algorithms, in *Advances in neural information processing systems*, pages 2951–2959, 2012.

[77] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, Optimizing deep learning hyper-parameters through an evolutionary algorithm, in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 4, 2015.

[78] X. C. Guo, J. H. Yang, C. G. Wu, C. Y. Wang, and Y. C. Liang, A novel LS-SVMs hyper-parameter selection based on particle swarm optimization, Neurocomputing **71**, 3211–3215 (2008).

[79] I. M. Sobol, Distribution of points in a cube and approximate evaluation of integrals, U.S.S.R. Comput. Maths. Math. Phys. **7**, 86–112 (1967).

[80] B. J. Berne and R. Pecora, *Dynamic light scattering with applications*, John Wiley & Sons, Inc., 1976.

[81] W. van Megen and P. N. Pusey, Dynamic light-scattering study of the glass transition in a colloidal suspension, Physical Review A **43**, 5429 (1991).

[82] R. S. Dias, J. Innerlohinger, O. Glatter, M. G. Miguel, and B. Lindman, Coil-Globule transition of DNA molecules induced by cationic surfactants: A dynamic light scattering study, Journal of Physical Chemistry B **109**, 10458–10463 (2005).

[83] D. J. Durian, D. A. Weitz, and D. J. Pine, Multiple light-scattering probes of foam structure and dynamics, Science **252**, 686–688 (1991).

[84] H. Oikawa and K. Takeda, Dynamic near-infrared light scattering from blue-colored polystyrene latex, Journal of Macromolecular Science, Part B: Physics (1998).

[85] S. B. Dierker, R. Pindak, M. Fleming, I. K. Robinson, and L. Berman, X-Ray photon correlation spectroscopy study of brownian motion of gold colloids in glycerol, Physical Review Letters **75**, 449 (1995).

[86] C. Gutt, L.-M. Stadler, T. Duri, T. Autenrieth, O. Leupold, Y. Chushkin, and G. Grübel, Measuring temporal speckle correlation at ultrafast X-ray sources, Optics Express **17**, 55–61 (2009).

[87] O. G. Shpyrko, E. D. Isaacs, J. M. Logan, Y. Feng, G. Aeppli, R. Jaramillo, H. C. Kim, T. F. Rosenbaum, P. Zschack, M. Sprung, S. Narayanan, and A. R. Sandy, Direct measurement of antiferromagnetic domain fluctuations, Nature **447**, 68–71 (2007).

[88] D. M. Mills, editor, *Third-Generation Hard X-ray Synchrotron Radiation Sources: Source Properties, Optics, and Experimental Techniques*, John Wiley & Sons, Inc., 2002.

[89] M. Lietner, B. Sepiol, L.-M. Stadler, B. Pfau, and G. Vogl, Atomic diffusion studied with coherent X-rays, Nature Materials **8**, 717–720 (2009).

[90] G. Grübel, G. B. Stephenson, C. Gutt, H. Sinn, and T. Tschentscher, XPCS at the European X-ray free electron laser facility, Nuclear Instruments & Methods in Physics Research, Section B: Beam Interactions with Materials and Atoms **262**, 357–367 (2007).

[91] J. W. Goodman, *Statistical Optics*, John Wiley & Sons, Inc., 2015.

[92] S. Baker and R. Cousins, Clarification of the use of chi-square and likelihood functions in fits to histograms, Nuclear Instruments & Methods in Physics Research **221**, 437–442 (1984).

[93] J. Frederick, *Statistical methods in experimental physics*, World Scientific Publishing Company, 2006.

[94] F. Livet, F. Bley, J. Mainville, R. Caudron, S. G. J. Mochrie, E. Geissler, G. Dolino, D. Abernathy, G. Grübel, and M. Sutton, Using direct illumination CCDs as high-resolution area detectors for X-ray scattering, Nuclear Instruments & Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors, and Associated Equipment **541**, 596–609 (2000).

[95] M. D. Richard and R. P. Lippmann, Neural network classifiers estimate bayesian a posteriori probabilities, Neural Computation **3**, 461 (2008).

[96] F. Sinz, S. Gerwinn, and M. Bethge, Characterization of the p-generalized normal distribution, Journal of Multivariate Analysis **100**, 817–820 (2009).

[97] *Python Software Foundation. Python Language Reference, version 3.5. Available at http://www.python.org/.*

[98] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: A system for large-scale machine learning, in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.