Knowledge Transfer in Neural Networks: Knowledge-Based Cascade-Correlation

François Rivest

School of Computer Science
McGill University, Montréal
July 2002

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements of
the degree of Master in Science

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisisitons et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-85820-0
Our file Notre référence
ISBN: 0-612-85820-0

TABLE OF CONTENT

ABSTRACT

Most neural network learning algorithms cannot use knowledge other than what is provided in the training data. Initialized using random weights, they cannot use prior knowledge such as knowledge stored in previously trained networks. This manuscript thesis addresses this problem. It contains a literature review of the relevant static and constructive neural network learning algorithms and of the recent research on transfer of knowledge across neural networks. Manuscript 1 describes a new algorithm, named knowledge-based cascade-correlation (KBCC), which extends the cascade-correlation learning algorithm to allow it to use prior knowledge. This prior knowledge can be provided as, but is not limited to, previously trained neural networks. The manuscript also contains a set of experiments that shows how KBCC is able to reduce its learning time by automatically selecting the appropriate prior knowledge to reuse. Manuscript 2 shows how KBCC speeds up learning on a realistic large problem of vowel recognition.

RÉSUMÉ

La plupart des algorithmes d'apprentissage des réseaux de neurones ne peuvent utiliser des connaissances autres que celles contenues dans les données d'entraînement. Initialisés avec des poids aléatoires, ils ne peuvent utiliser de connaissances préexistantes telles que celles contenues dans d'autres réseaux entraînés antérieurement. Cette thèse traite de ce problème. Elle contient une revue de la littérature sur les algorithmes d'apprentissage des réseaux de neurones statiques et constructifs pertinents et des recherches récentes sur le transfert des connaissances entre les réseaux de neurones. Le premier manuscrit décrit un nouvel algorithme, nommé KBCC, qui améliore l'algorithme d'apprentissage *cascade-correlation* pour lui permettre d'utiliser des connaissances préexistantes. Ces connaissances préexistantes peuvent être fournies sous forme de réseaux de neurones entraînés. Le manuscrit contient aussi un ensemble d'expériences montrant que KBCC est capable de réduire son temps d'apprentissage en choisissant les connaissances préexistantes appropriées. Le second manuscrit montre comment KBCC accélère son apprentissage sur un problème réaliste d'envergure : la reconnaissance de voyelles.

ACKNOWLEDGEMENT

I am especially thankful to Professor Thomas R. Shultz. He gave me the opportunity to start doing real research when I was an undergraduate student, and for this, I can never thank him enough. He gave me the latitude to develop the ideas I had and that I believed in, while providing me with the necessary support and feedback to bring those ideas to life. I will always be thankful to the trust he showed in my research abilities.

I am also thankful to Professor Doina Precup for her support and guidance through my graduate years at McGill. Her broad knowledge of my field allowed for insightful, constructive supervision.

My work has also profited in one way or another from comments of fellow students David Buckingham, Reza Farivar, Jacques Katz, Sylvain Sirois, and Jean-Philippe Thivierge. Faculty members Yoshio Takane and Yuriko Oshima-Takane also contributed useful comments. The manuscripts profited from the comments of a few anonymous reviewers.

I was fortunate to be supported by a grant from the Natural Sciences and Engineering Research Council of Canada and a grant from the Fonds de la Formation de Chercheurs et l'Aide à la Recherche to Thomas Shultz and by a grant from the Centre de Recherche Informatiqe de Montréal (CRIM) in collaboration with the Fonds de la Formation de Chercheurs et l'Aide à la Recherche to me. I am thankful to M. Pierre Dumouchel, VP at CRIM, for his useful support and supervision while I was there.

Finally, I would like to thank my parents for their continuous belief in me and their encouragement in my academic aspirations. They were also always present in the last year when I needed a babysitter to be able to complete this work. I am also thankful to my wife Marie-Claude Bergeron for her continuous understanding and encouragement.

I would like to dedicate my thesis to my daughter Mélodie and to my next child who should be born soon.

CONTRIBUTION OF AUTHORS

Professor Thomas R. Shultz is the first author of the first paper and second author of the second paper. The original idea of recruiting whole, previously learned networks, in addition to single hidden units, was due to Professor Shultz. I developed the mathematics for KBCC, and implemented the algorithm, determining its details.

In the first paper, I developed KBCC and conducted the experiments and data analysis. Design of the experiments was done collaboratively with Professor Shultz.

In the second paper, professor Shultz contributed as my advisor, providing feedback and suggestions at all stages of my work.

STATEMENT OF ORIGINALITY

Knowledge-based cascade-correlation (KBCC) is an original algorithm that I developed that extends the cascade-correlation algorithm (Fahlman & Lebiere, 1991). It is a new solution to the problem of transferring knowledge in neural networks that has fewer limitations than previous work and a higher level of automation (see conclusion).

INTRODUCTION

Artificial neural networks reappeared in the 80s after being dismissed for a long period by the artificial intelligence community. It is only after they gained wide acceptance as AI tools and cognitive models that the problem of transfer of knowledge in neural networks really appeared as an important issue.

The problem became more important in the early 90s, when a large number of applications based on neural networks were developed. Considering the huge amount of time required to train them, any speed improvement that could be gained when training a second version from a first one would be an asset. Similarly, considering the lengthy process of gathering data to train a successful network, any generalization that could be gained from re-using trained neural network knowledge to supplement the training set would also be an asset. Neural networks were also widely used as models of human cognitive abilities or development. The fact that these networks start from scratch while most human learning is based on prior experience (Pazzani, 1991, Wisniewski 1995) is a real-drawback. Moreover, simple re-training of a previously trained network on a new task showed catastrophic forgetting of the prior knowledge (McCloskey & Cohen, 1989), which is also inadmissible for any model of the human brain.

The interest in the topic of knowledge transfer in neural networks reached its apogee in the mid 1990s. At NIPS 1995, there was a workshop organized by Baxter, Caruana, Mitchell, Pratt, Silver and Thrun on *Learning to Learn: Knowledge Consolidation and Transfer in Inductive Systems*.[1] The following year, there was a special issue of *Connection Science* 8(2) on *Transfer in Inductive System*.

Although a large amount of research took place, many of the problems raised by transfer of knowledge in neural systems remain unsolved. Despite some partial success, most applications and cognitive models are still built from scratch. Few solutions address the problem of changes in input encoding and output encoding from previously trained networks to new tasks. None address the unequal complexity of the various tasks a network may have to learn during its life. And many solutions are heavily restricted in

[1] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/caruana/pub/transfer.html

how prior knowledge is represented. Finally, much of the knowledge transfer must still be done by hand; for example, networks must usually be pre-selected to be reused.

This thesis is an attempt to improve knowledge transfer in neural networks by presenting a novel algorithm called *knowledge-based cascade-correlation* (KBCC). KBCC is based on Fahlman and Lebiere's (1991) *cascade-correlation* (CC) algorithm, which was shown to model many cognitive development phenomena (Shultz & Bale 2001, Buckingham & Shultz, 2000) as well as being able to deal with real applications (Yang & Honavar, 1998). KBCC inherits many of the useful properties of CC. Moreover, it has no restrictions in terms of input and output encoding of the prior knowledge. The only restriction on the form of prior knowledge in KBCC is that it be in the form of a function, and most knowledge can be represented in this way. KBCC automatically searches for relevant knowledge and uses only what is necessary. It also allows a new type of compositionality in knowledge transfer (see conclusion).

This manuscript thesis begins with a literature review of the existing work on transfer of knowledge across neural networks. Then, the first manuscript, published in *Connection Science*, describes the KBCC algorithm in detail. This description is followed by a series of experiments showing how KBCC speeds up learning by using prior knowledge as well as a series of experiments demonstrating KBCC's ability to choose the most appropriate sources and integrate them to learn a solution to the target task. This manuscript is followed by manuscript 2, published in the *Proceedings of the 2002 International Joint Conference on Neural Networks*, which documents the ability of KBCC to transfer knowledge on a large realistic problem. Finally, the conclusion compares KBCC to other approaches mentioned in the literature review, explains how KBCC addresses many important issues in knowledge transfer, and discusses further research.

LITERATURE REVIEW

Neural Networks

Artificial Neural Networks encompass a wide range of architectures. This study of transfer of knowledge in neural networks is restricted to the so-called feedforward architecture. The justification for this choice is that, so far, most of the literature on transfer of knowledge in neural networks is on this architecture. Nevertheless, some transfer methods could be ported easily to other architectures, and some cases will be mentioned. Two types of feedforward networks have been studied for transfer of knowledge: layered feedforward networks, usually trained using the backpropagation algorithm, and cascaded feedforward networks, most often built and trained using the cascade-correlation algorithm. The former is probably the most common type of neural network. The latter is the basis of KBCC, a new algorithm I devised for the transfer of knowledge. KBCC is described in detail in manuscript 1 and improved in manuscript 2.

Multilayer Backpropagation Networks

Often called backpropagation networks, this class of networks can be more precisely described as multilayer feedforward neural networks trained using backpropagation of the error signal. This section first describes the architecture of this type of network, then its general training algorithm, and, finally, some of its known properties.

Multilayer feedforward networks are made of two parts: layers of nodes, and sets of weights, as shown in Figure 1. Layers are collections of neurons. In this type of architecture, each layer feeds the next one through a set of weighted connections. Usually, every unit of a layer feeds every unit of the next layer. But a unit cannot feed any unit other than those on the next layer. If a unit was feeding a unit on a layer ahead that is not the next one, this would be a *cross-connection*. If a unit was feeding a unit on the same layer or on a previous layer, this would be a *recurrent connection*. The multilayered feedforward architecture discussed here contains neither cross-connections nor recurrent connections.

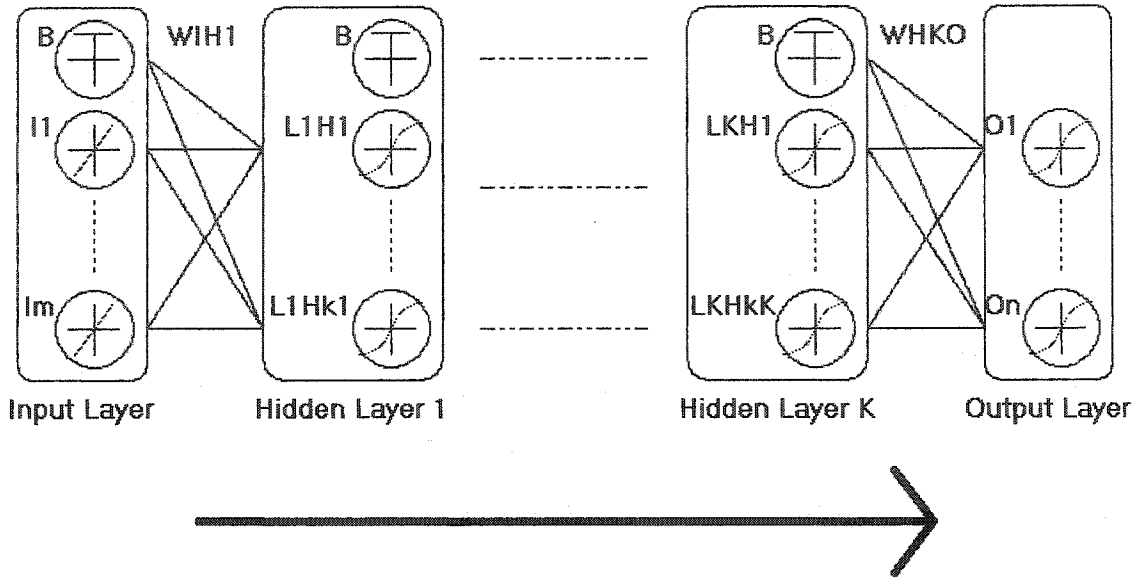


Figure 1: General multilayer feedforward neural network topology.

Usually, each layer is fed by the previous layer and by a bias unit, a constant giving the basic activity (see Table 1). For simplicity, this bias unit is often implemented as being part of the previous layer as shown in Figure 1.[1] Input units are usually linear functions, i.e. their activation is given by the input pattern. For each non-input unit, its input is given by the weighted sum of the previous layer activations times the weight vector connecting the previous layer to the unit. This weighted sum is then passed through the unit's activation function to obtain its output, or activation level, that will be an input for the next layer. The output layer activations corresponding to an input pattern are the associated output pattern of the network.

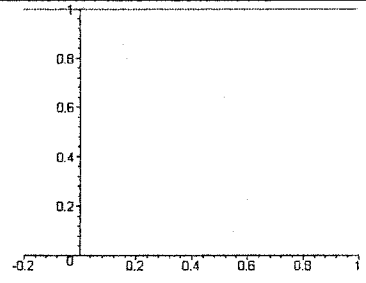
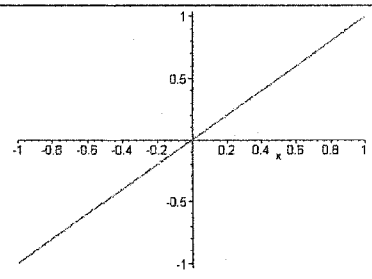
Activation functions can have many forms. In the original work on perceptrons (McCulloch & Pitts, 1943, Rosenblatt, 1962), which generally had no hidden layers, the activation function was a simple threshold function of the form $f(x)=\begin{cases}0 & if \quad x < 0\\ 1 & if \quad x \geq 0\end{cases}$.

But Minsky and Papert (1969) showed that perceptrons could only solve *linearly separable* problems. Two sets of points are linearly separable if and only if there exist a hyperplane such that all points of the first set are on one side of it and all points of the other set are on the other side of it. Moreover, this function has the disadvantage of not

[1] Note that it is equivalent to have all but the input layer fed by a single bias unit.

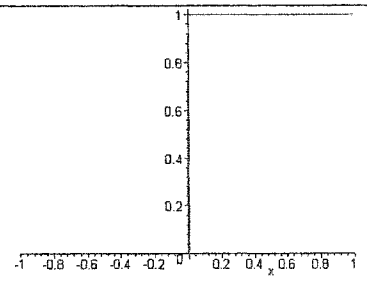
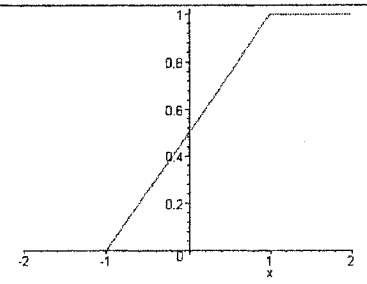
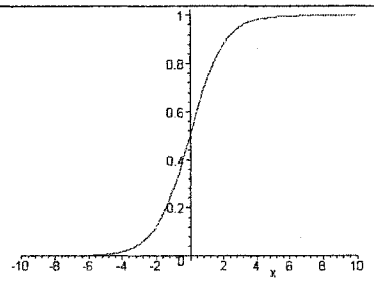
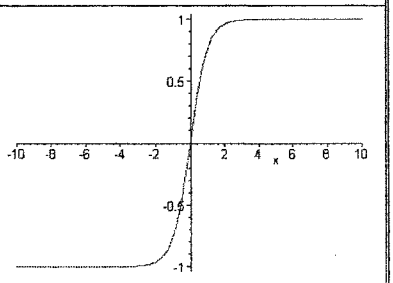
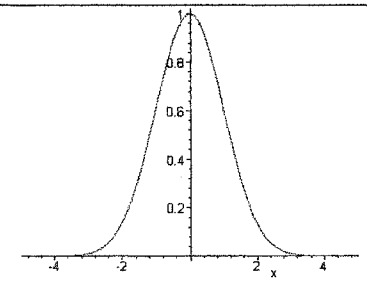
being differentiable everywhere, and, hence, does not work with gradient descent optimization algorithms. The threshold function was therefore replaced by the sigmoid $f(x)=\frac{1}{1+e^{-\alpha x}}$ and the hyperbolic tangent $f(x)=\tanh(x)$ functions[2]. These functions are sometimes called *soft* threshold functions because they are a good differentiable approximation of the threshold function, as shown in Table 1. This led to a more general architecture with multiple layers, first invented by Bryson and Ho (1969), and popularized by Rumelhart et al. (1986) and the PDP Group in the mid-1980s. Many other functions were also studied, such as the ramp (continuous piece-wise linear), the augmented ratio of squares and the Gaussian. This last one is most often used in a slightly different architecture called radial basis networks, or networks of radial basis functions. Table 1 lists these functions with their mathematical definition and their graph. Note that all the functions mentioned above can be adjusted to match the desired output range using an affine transform of the form $af(x)+b$.

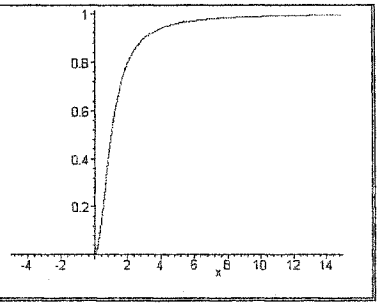
Table 1: Most common activation functions with their definition and their graph.[3]

Function Name	Function Definition	Function Graph
Bias	1	
Linear	$f(x)=x$	

[2] In the sigmoid function, the α parameter is the so-called *temperature* parameter, usually set to 1 in this architecture

[3] This list is based in part from Eberhart, Simpson and Dobbins (1996).

Threshold, step, or heaviside	$f(x)=\begin{cases}0 & if \quad x < \theta \\ 1 & if \quad x \geq \theta\end{cases}$	
Ramp or continuous piece-wise linear	$f(x)=\begin{cases}0 & if \quad x < \alpha \\ x & if \quad \alpha \leq x < \beta \\ 1 & if \quad x \geq \beta\end{cases}$	
Sigmoid or logistic	$f(x)=\frac{1}{1+e^{-\alpha x}}$	
Hyperbolic tangent	$f(x)=\tanh(x)$	
Gaussian	$f(x)=e^{\frac{(x-\mu)^2}{2\sigma^2}}$	

Augmented ratio of squares	$f(x)=\begin{cases}\dfrac{x^2}{1+x^2} & if \quad x \geq 0 \\ 0 & otherwise\end{cases}$	

Training a multilayer feedforward neural network can be viewed as a straightforward optimization problem, where the weights are the parameters to optimize, and the sum squared error of the network on the training set is the objective function. Most of the time, a simple gradient descent approach, requiring the derivative of the objective function with respect to the weights, is used. Let $N_W : \Re^{IN} \mapsto \Re^{OUT}$ be the network function with weights W, where *IN* is the number of inputs and *OUT* the number of outputs. Let $\{\vec{i}_p\} \subset \Re^{IN}$ be the set of input patterns and $\{\vec{t}_p\} \subset \Re^{OUT}$ be the corresponding set of target patterns. If $\{\vec{o}_p = N_W(\vec{i}_p)\} \subset \Re^{OUT}$ is the set of corresponding output patterns produced by the network, then the objective function is given by:

$$E = \sum_p \left\| \vec{t}_p - \vec{o}_p \right\|^2 \qquad (1)$$

where $\|\vec{v}\|$ is the standard Euclidian norm of $\vec{v}$. The update rule for the weights using gradient descent is then $W_{(t+1)} = W_{(t)} + \Delta W_{(t)}$, where $W_{(t)}$ are the weights at time *t*, $\Delta W_{(t)} = -\eta \dfrac{\partial E}{\partial W_{(t)}}$, and $\dfrac{\partial E}{\partial W_{(t)}}$ is the partial derivative of *E* with respect to the weights *W*. This is usually called the *backpropagation* algorithm (BP). It is also current practice to add a momentum term updating the weights with $\Delta W_{(t)} = -\eta \dfrac{\partial E}{\partial W_{(t)}} + \alpha \Delta W_{(t-1)}$. Assuming a ball in weight space at position *W* on the error surface *E*, the gradient gives the direction and speed with which the ball should roll down from that position, while the momentum is analog to its current speed and direction. The momentum term makes the ball roll in a direction that is in part due to the current form of the error surface and in part due to the ball's current direction.

The objective function can also be modified to constrain the weights. By adding the proper penalty term to the objective function, one can force the weights to be small (add $|W|$) or as orthogonal as possible (Golden, 1996). The objective function can also be modified to obtain invariance with respect to translation of the input by adding a penalty term based on the value of the slope (Mitchell, 1997).Finally, the objective function can also be the minimization of the cross-entropy of the output patterns (Hinton, 1989). In that case, and under certain assumptions, the output units are the probability density function of the output values, and the resulting algorithm (that does not need target patterns) is an unsupervised learner generating an Independent Component Analysis (ICA) (Comon, 1994) of the inputs. ICA is similar to Principal Component Analysis (PCA) but on higher order statistics.

Finally, the optimization is in no way limited to the gradient descent algorithm. For instance, Fahlman developed Quickprop, which uses an estimate of the second order derivative (Fahlman, 1988). One could also use the second order Levenberg-Marquardt algorithms, Shannon algorithms or general conjugate gradient approach (Golden, 1996). More distributed algorithms could also be used, like the particle swarm optimization algorithm (that requires first order derivative) or basic genetic algorithms (which do not require derivatives and whose objective functions are called fitness functions), just to name a few (Eberhart, Simpson & Dobbins, 1996).

An important known result about sigmoid-based multilayer feedforward networks is that two layers (one hidden layer of sigmoidal units with an output layer of linear units) are sufficient to approximate any continuous function to any arbitrary precision (Cybenko 1989; Hornick et al. 1989). Moreover, three layers (two hidden layers of sigmoidal units with an output layer of linear units) are sufficient to approximate any function to any arbitrary precision (Cybenko 1988).

Cascade-Correlation Networks

Multilayer feedforward networks are said to be static, because their structure is fixed at design time. A different approach is to use a network with a more dynamic structure, beginning with only a minimal number of units and connections, and letting it build its structure all by itself. There are at least two techniques that have been developed

to do this, and they appeared around the same time: splitting node networks by Wynne-Jones (1992) and cascade-correlation networks (CC) by Fahlman and Lebiere (1990). This review is limited to the cascade-correlation architecture because it is the basis of the algorithm developed in the manuscripts. Many other constructive algorithms have been developed on the basis of cascaded networks and other similar architectures, some of which, like CC, are based on the work done originally by Gallant (1986). However, these other constructive approaches are more often designed to build classifiers rather than function approximators, which limits their use.

Cascade-correlation networks differ from backpropagation networks in two ways. First, they do have cross-connections, and second, they construct their architecture automatically. In order to build the network architecture, CC alternates between two phases. In one phase, called the *output phase*, the weights feeding the output layer are trained similarly to backpropagation networks to reduce the error signal. In the other phase, the *input phase*, it is the weights feeding candidate units that might be installed in the network that are trained.

A cascade-correlation network begins in output phase with only the bias unit, the input layer and the output layer. An example of a new network would be given in Figure 2, if only the input and output layers and their connections were considered. The weights are trained to minimize the sum of squared error as in backpropagation, usually using Quickprop. When the error stagnates, the network shifts to input phase.

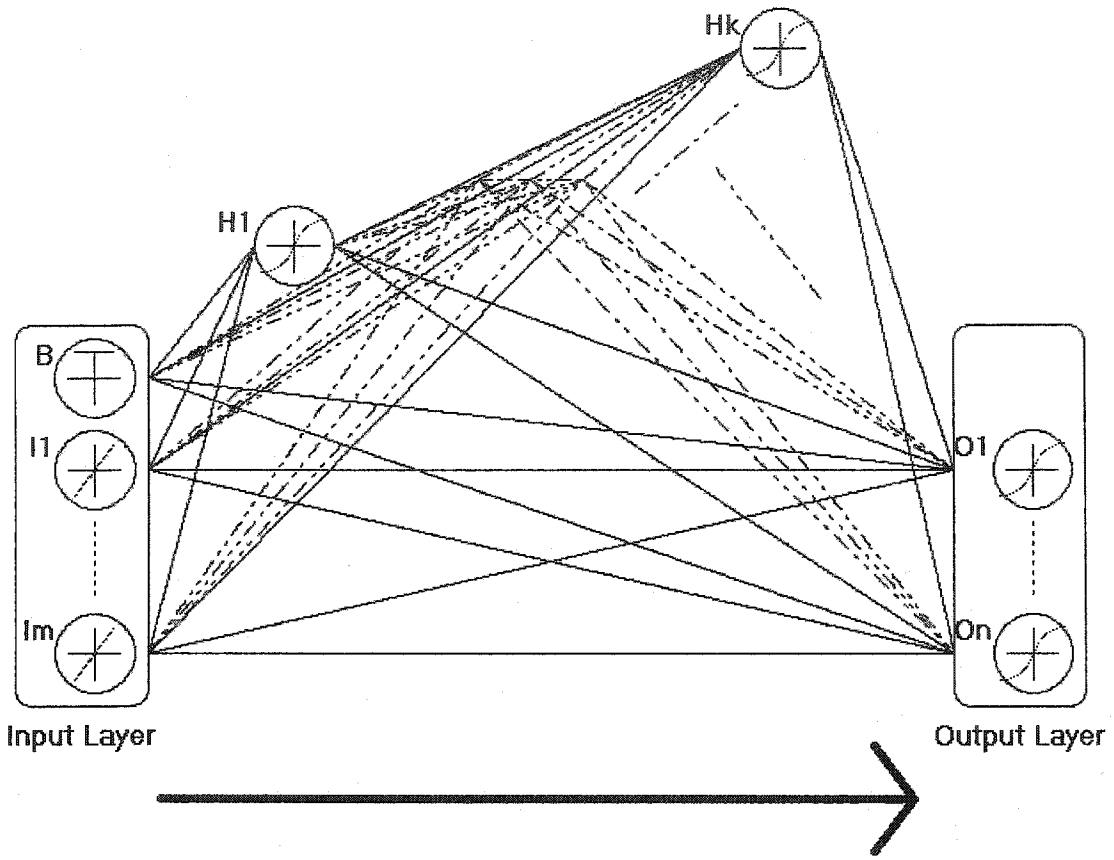


Figure 2: General cascade network topology.

The first step of the input phase is to initialize a pool of candidate units. Those units (usually sigmoid or symmetric sigmoid) are fed by all but the output units. Each candidate is trained in parallel to maximize its covariance with the error patterns. If $\{\vec{e}_p = \vec{t}_p - \vec{o}_p\} \subset \Re^{OUT}$ is the set of error patterns, and $\{\vec{v}_p\}_c \subset \Re$ the set of corresponding activations for candidate c, then its objective function is given by: [4]

$$S_c = \frac{\sum_o \left| Cov\left(\{\vec{v}_p\}_c, \{e_{p,o}\}\right) \right|}{\sum_p \left\| \vec{e}_p \right\|^2}, \text{ where } e_{p,o} \text{ is the } o^{th} \text{ element of vector } \vec{e}_p . (2)$$

Therefore, candidate units are biased toward being best at tracking the unresolved portion of the problem. When their score (the value of their objective function) stagnates, the best

[4] A generalized mathematical definition of the network topology, derivative and objective function can be found in Appendix I: KBCC Mathematics.

unit is kept and a set of output weights between the new unit and the output layer is initialized with the proper sign (based on the sign of the covariance). The other candidates are discarded. Once a candidate is recruited and installed, the network shifts to output phase again.

Whenever the network reaches the learning objective in output phase, the whole process stops. A final CC network has all the possible cross-connections and looks as shown in Figure 2. A particularity of this algorithm is that all the weights but the output ones are frozen. Hence, the same network could be used to learn another task without forgetting its knowledge stored in the hidden units (which are often considered to be feature detectors).

Although CC in its original configuration has a lot of user defined parameters, in theory it should need fewer than backpropagation networks. The reason is that CC finds the right number of hidden nodes to reach the necessary computational power all by itself. Prechelt (1997) found that Rprop, a gradient descent algorithm using an adaptive learning rate and invented by Riedmiller & Braun (1986), was less sensitive to parameter variations than Quickprop. A debate still remains about whether hidden nodes should be cascaded or not, but Baluja and Fahlman (1994) have found that having both types of candidates (some that would be appended to the top most layer and some that would create a new layer) seems an excellent tradeoff, leading to compact networks of very few layers.

Two important reviews of CC need to be mentioned. Prechelt (1997) has benchmarked different versions of CC on 42 different problems from the PROBEN1 database. Few general results were found. One general finding was that CC was superior for classification problems to an error minimization version of itself, which was superior on regression problems. Also, Waugh (1995) studied CC for function approximation, benchmarking CC, special connection schemes in CC, connections pruning in CC, and other extensions of CC. He found few positive improvements to CC, especially on phase stopping criteria and connections pruning. The literature contains more than a hundred papers on CC or variations of it.

Finally, many of the variations discussed on BP learning can also be applied to CC. The type of candidate units can be as diverse as for BP, a penalty term can be added

to the objective functions to favor certain properties, and the optimization algorithm is in no way restricted to Quickprop. There also exists a recurrent version of CC (Fahlman, 1991).

Transfer of Knowledge in Neural Networks

There are different types of transfer of knowledge in neural networks. The most important distinction is *representational* versus *functional* transfer (Pratt & Jennings 1996, Silver 2000). In representational transfer, it is the representation of the knowledge that is transferred either *directly* or *indirectly*. Direct methods are either *literal* or not. For example, a trained network can be copied, with (non-literal) or without (literal) minor changes to its weights and structures. It could also be transformed into an intermediate representation between its original and final representation (indirect method). On the other hand, in functional transfer, the representation is not directly involved. In this type of transfer, the knowledge is used to bias the training instead of the initial state of the network (Silver 2000)[5]. The learning may be biased by extra patterns, in one way or another, or the indirect use of previous learning experience.

This section will deal with both types of transfer. First, there is a review of the work on representational transfer, from direct literal transfer to non-literal methods. Then, there is a review of the work on functional transfer of knowledge. Finally, research on the transfer of symbolic knowledge into neural networks is reviewed followed by a brief summary of other related work. The new algorithm developed in this thesis, called Knowledge-based Cascade-correlation, will not be covered here, because it is described in detail in manuscript 1. KBCC transfers the knowledge representation, without limiting that knowledge to any specific representation. It will be compared with the other methods presented here in the conclusion.

Representational Transfer of Knowledge: Literal and Non-Literal Methods

The simplest way to transfer knowledge from a trained network to a new network learning a task is to literally copy the trained network and train it on the new task as

---

[5] Pratt and Jennings (1996) have a slightly different definition of representational and functional transfer. Although both definitions are not totally incompatible, I found Silver's (2000) definition more appropriate.

shown in Figure 3. Although this may sometimes accelerate the learning, it may also slow it down (Pratt 1993a) and reduce the target network accuracy (Martin, 1988, mentioned in Pratt 1993b).Moreover, the re-trained network often loses its prior knowledge, a problem called catastrophic forgetting (McCloskey & Cohen, 1989). Forcing the hidden layer to be as orthogonal (i.e., that weight vectors feeding hidden units are orthogonal) and as distributed as possible seems to help, but it does not always work (French, 1992, 1994).

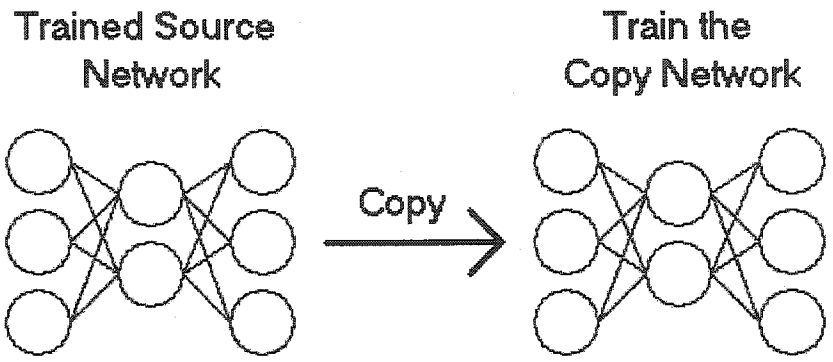


Figure 3: Literal transfer of knowledge.

Cascade-correlation is one of the first successful examples of literal transfer. Because all but its output weights are frozen once trained, only the output weights change when learning a second task. Hence, even if some new hidden units are added to its structure, the network can very rapidly recover its prior knowledge by re-adjusting the output weights properly. This is an example of transfer through *sequential learning* as shown in Figure 4.

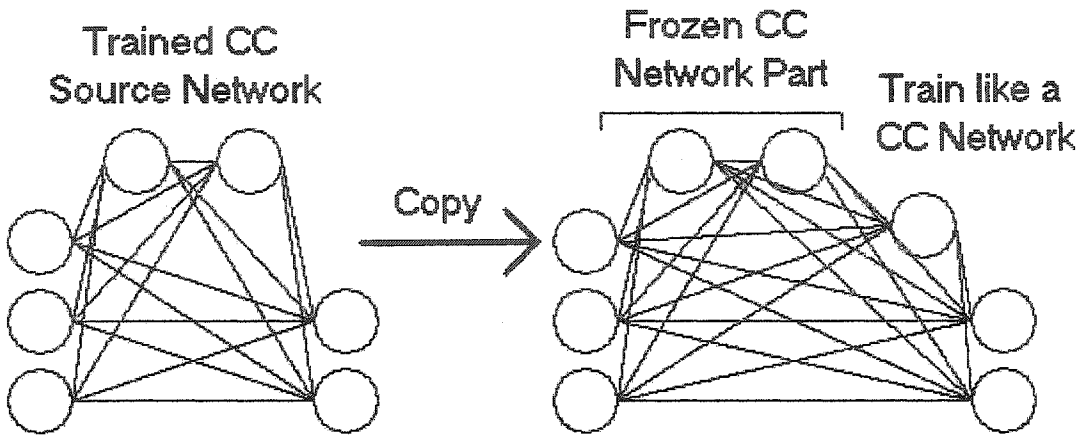


Figure 4: Literal transfer of knowledge using sequential learning on a CC network.

A similar approach was also used by Parekh and Honavar (1998). Using the knowledge-based artificial neural network (KBANN[6]) algorithm (Towell and Shavlik, 1994) to transform symbolic knowledge into a neural network, they used the inputs of the current task as well as the outputs of the knowledge-based network to generate the inputs of a constructive algorithm. Note that this technique is not limited to symbolic knowledge. Any network could be copied and used to generate extra inputs for a new network to be trained. It is also not limited to constructive algorithms, it suffices that the new network contains as many inputs as the target task does plus the number of outputs of the prior knowledge as shown in Figure 5.

[6] KBANN is discussed in the section on symbolic to neural network transfer.

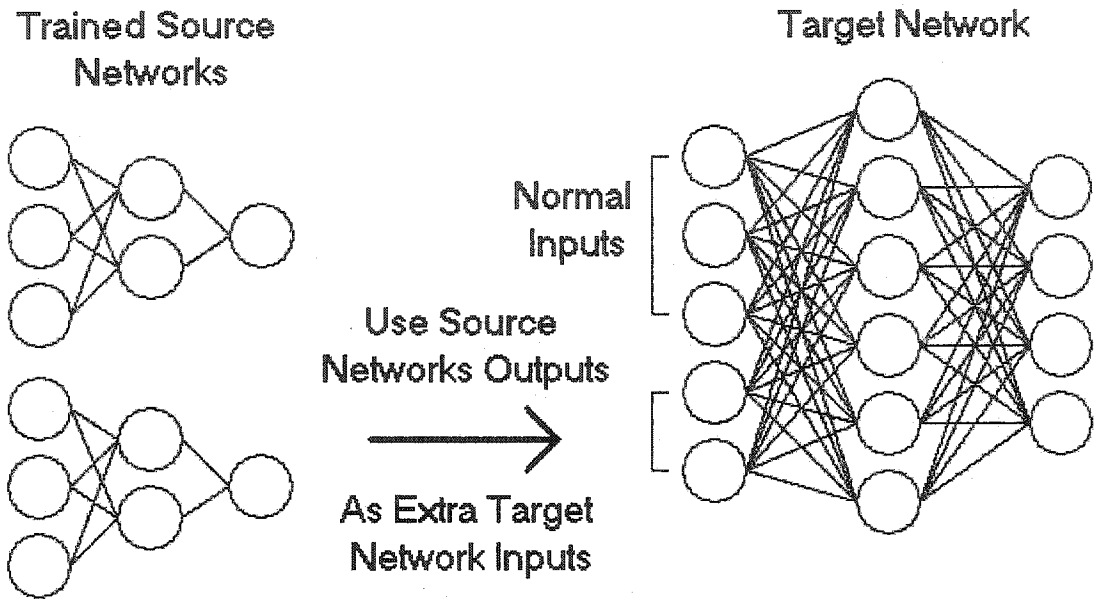


Figure 5: Literal transfer of knowledge by using network outputs as extra inputs.

Another approach close to literal copying of a backpropagation network also appears to accelerate the learning of complex tasks. Instead of training a large network on some multi-output task, the task is first decomposed into subtasks corresponding to groups of different outputs. A small network is trained for each of these subtasks. The networks are then integrated into a larger network, as shown in Figure 6. Once that knowledge is transferred, the large network may be supplemented with some extra nodes, which are trained on the full task. The whole network may then be refined through full training. This approach, often called *task decomposition,* was used first used successfully by Waibel (1989) and then re-evaluated by Pratt (Pratt, Mostow & Kamm, 1991).

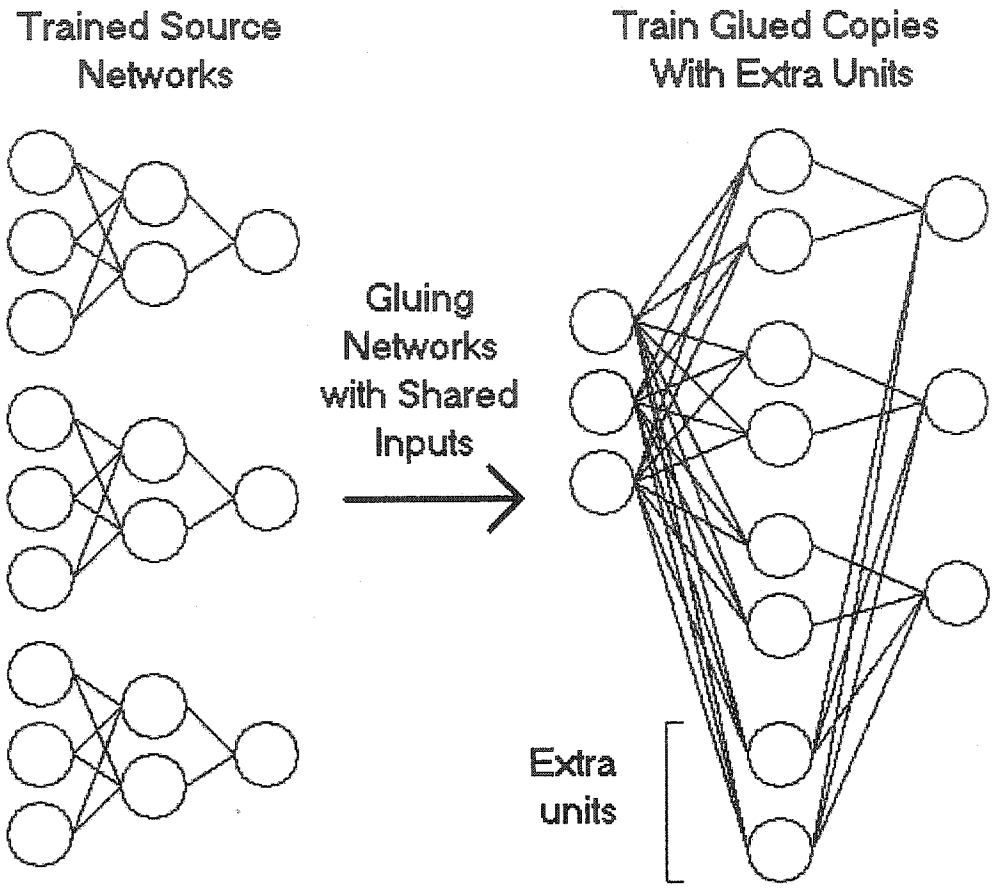


Figure 6: Transfer of knowledge by gluing trained sub-networks together.

A similar idea proposed by Hinton[7] was to copy the trained networks to initialize a *mixture of experts* (ME) system (Jacobs, Jordan, Nowlan & Hinton, 1991). In this model, a number of networks are trained in parallel, each on a subset of the training patterns. At the output of the networks, there is a gating network that decides which network output to use for a specific input pattern. This leads networks to specialize on the subset of patterns they are best at mapping. This model is shown in Figure 7.

[7] Personal discussion at McGill in February 2002.

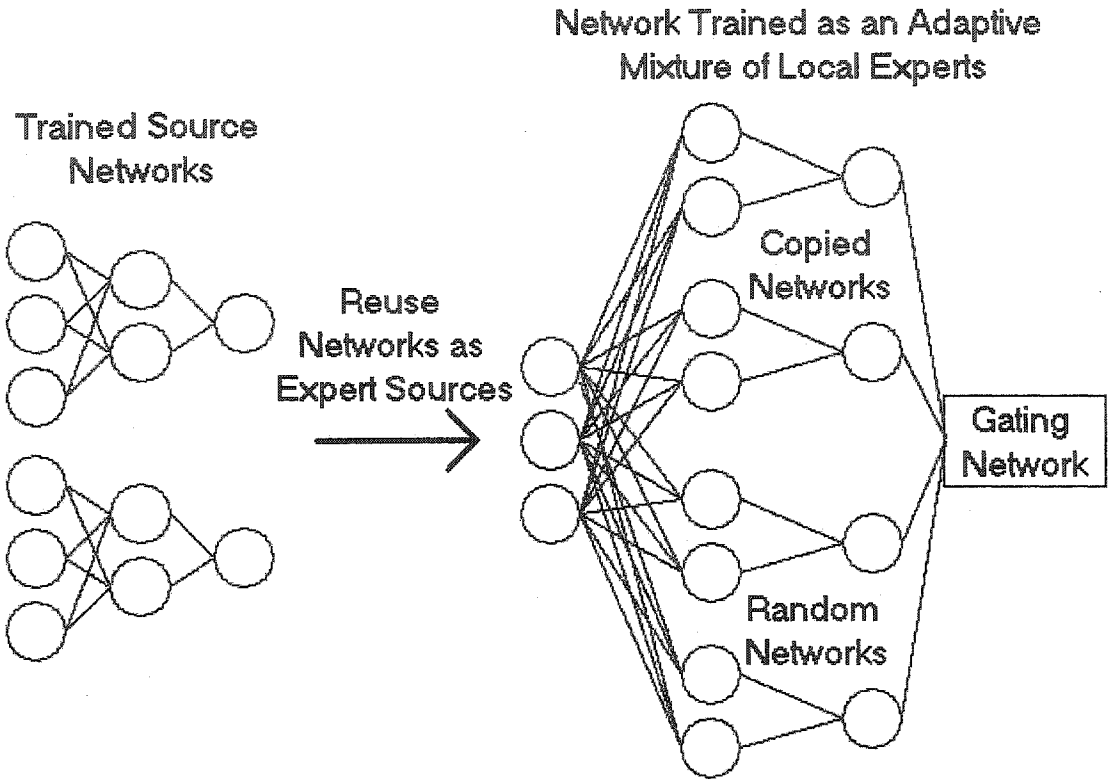


Figure 7: Literal transfer of knowledge by using networks in a mixture of experts.

When prior knowledge is literally copied to a new task, some of the hyperplanes[8] generated by the first layer of weights can be a bad fit for the new task and hence, may negatively affect the learning (when compared with a randomly initialized network) (Pratt, 1993a, 1993b). In order to solve this issue, Pratt (1993a, 1993b) devised a new algorithm called *discriminality based transfer* (DBT). The idea is to measure the discriminality of each hyperplane (unit in the first hidden layer). If a hyperplane is useful for the new task, its weights are scaled up to high values to keep it in place during learning. If a hyperplane is bad, its weights are reduced or even re-initialized to random values, assuming that a random hyperplane is more likely to become useful than a known bad hyperplane. Once this weight adjustment is accomplished, the network is trained using standard backpropagation.

---

[8] A sigmoidal unit can be viewed at the extreme as a threshold function where the weighted sum at the input determines a hyperplane. On one side of the hyperplane, the threshold unit is ON, on the other side, it is OFF.

In Pratt's work, the discriminality of a hyperplane is given by the mutual information measure, as in the construction of some decision trees. Discriminality is used to decide whether or not an existing hyperplane should be kept or changed. The mutual information is given by:

$$MI = \frac{1}{N}\left(\sum_{i=0}^{1}\sum_{j=1}^{C} x_{i,j} \log x_{i,j} - \sum_{i=0}^{1} x_i \log x_i - \sum_{j=1}^{C} x_j \log x_j + N \log N \right), \quad (3)$$

where N is the number of patterns, C the number of classes, j indexes over all classes, i indexes over the two sides of the hyperplane and x is the number of patterns in a given class (x_j), or on a given side of the hyperplane (x_i), or both ($x_{i,j}$) (from Mingers 1989 as referenced by Pratt 1993b). Figure 8 and Figure 9 show examples, on a bi-dimensional input space, of two hyperplanes, one with a poor and one with a good discriminality, respectively.

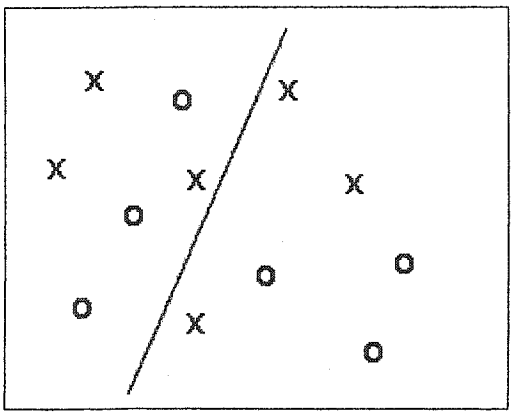


Figure 8: Hyperplane with poor discriminability.

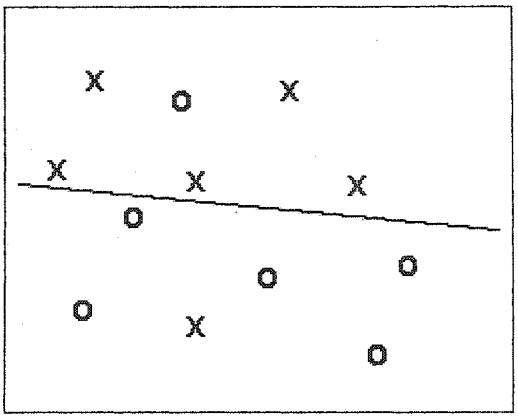


Figure 9: Hyperplane with good discriminability.

Pratt's (1993a, 1993b) results suggest that this method has all the advantages of literal transfer without the drawbacks. She also surveyed a few other techniques related to weight adjustment in knowledge transfer, but they seemed to have less success. She also surveyed other knowledge transfer techniques (Pratt & Jennings 1996), some of which were not included in this thesis.

Functional Transfer of Knowledge: MTL

A different approach is to use only the network functionality rather than its implementation. In order to do that, Silver (2000) developed a technique based on results

for multi-task learning by Baxter (1996) and Caruana (1995). The key idea is that when learning multiple tasks at the same time, the first set of weights of a two-layer neural network will build up a common representation useful to as many of the tasks as possible. In psychological terms, knowing about several problems related to the same concept gives a better grasp of that concept.

Baxter's (1996) work was mainly theoretical. He considered a multiple output network with each output corresponding to a different task and with at least a common hidden layer leading to all outputs. He showed that increasing the number of tasks reduced the necessary number of training patterns. More precisely, he showed that if $O(a)$ is the minimum number of patterns necessary to learn a single task, and if $O(a+b)$ is the minimum number of patterns to learn several tasks independently, then the network learning the tasks simultaneously requires only $O(a+b/n)$ patterns, where n is the number of tasks.

Caruana's (1995) work was more empirical, but he provided a few reasons why learning multiple related tasks should be better than single task learning. The most important one is the fact that the hidden layer has pressure from all tasks, and hence, should find a better hidden representation. His empirical results showed that *multi-task learning* networks (MTL) generalized better than *single-task learning* networks (STL).

Silver (Silver & Mercer, 1998; Silver, 2000) used MTL in conjunction with *pseudo-rehearsing* to transfer prior knowledge in new learning. He called this technique the *task rehearsal method* (TRM). Given a number of sources of knowledge for a total of S_n outputs and a target task of T_n outputs, let the new network to train have $T_n + S_n$ outputs, one for each task output, as shown in Figure 10. The input patterns are first processed through the source networks. Then, the resulting output patterns are concatenated with the target patterns to generate the training set of the new network. This process of passing new input patterns through old networks to generate target patterns is called *pseudo-rehearsal*, where normal rehearsal would imply using the original patterns only. The network thus ends up relearning all the tasks that were part of the source knowledge at the same time as the new task, while generating at the hidden layer, a set of features that are likely to be useful for more tasks (a common representation) than if the network was trained only on the new task.

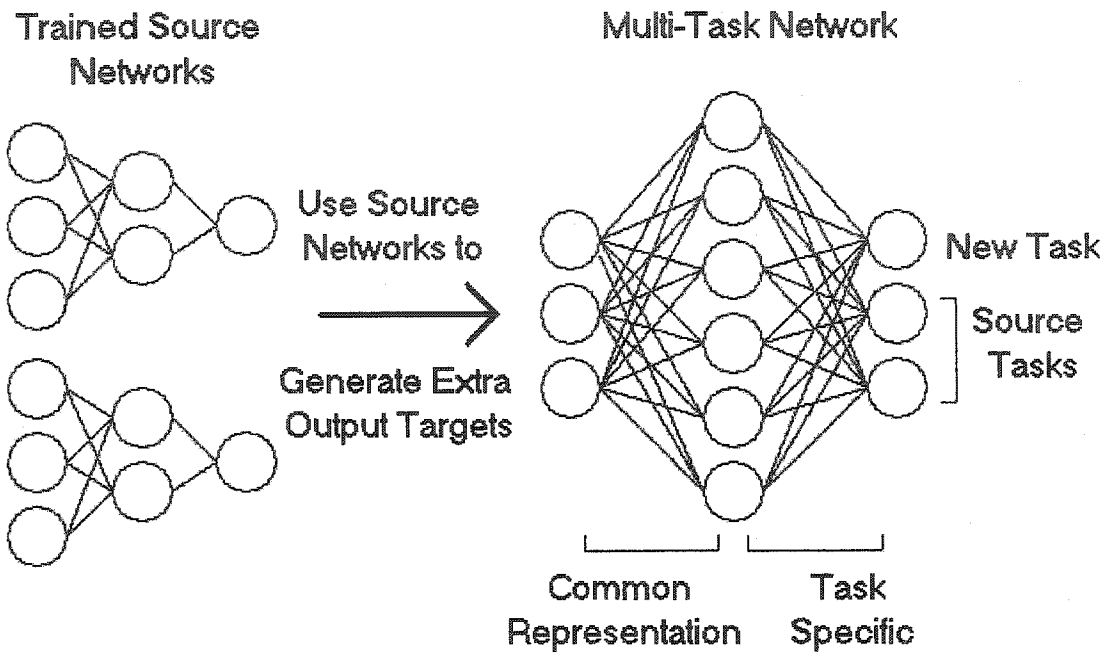


Figure 10: Functional transfer of knowledge through multi-task learning.

Silver (2000) has also shown that this procedure could be used on impoverished training sets, i.e., sets that contain too few patterns for proper learning. By generating extra input patterns and using them to generate extra outputs, the extra training that the hidden layers get from training the extra outputs (outputs matching the source networks outputs) may help the network discover the right internal representation for the target task.

Silver (Silver & Mercer, 1996, Silver, 2000) also developed a variant of multi-task learning (MTL) called ηMTL. The idea is to give each extra output a different learning rate from the target task. This is justified by the fact that the learning of those source tasks is not important in itself. What is important is the effect that learning them has on the internal representation of the network. The key is to evaluate, as training goes, how those source outputs relate to the target outputs. The more they relate, the closer their learning rate can be to the one of the target task. Unrelated tasks should be given very small learning rates, hence applying less pressure on the internal representation and reducing their potential interference. The learning rate η_k for task k is given by:

$$\eta_k = \eta \tanh\left(\frac{1/SSE_k}{d_k^2 + \varepsilon} \cdot \frac{1}{RELMIN}\right), \qquad (4)$$

where, assuming a single target task $k = 0$, η is the target task learning rate, SSE_k the sum squared error of the output k, d_k the distance between the weight vector feeding output 0 and output k, $\varepsilon > 0$ and *RELMIN* the parameter controlling the rate of decay.

Overall, knowledge transfer using pattern generation and ηMTL fills two goals, gaining *efficiency* (speed of learning) and *effectiveness* (accuracy). Silver's work has achieved some success in both of these goals.

Functional Transfer of Knowledge: Other Approaches

Another approach is the idea of *meta-learning* developed by Naik and Mammone (1992). The general idea is to have a meta-network that learns how to learn as shown in Figure 11. Here, no direct prior knowledge is available, only traces of prior learning experiences are present. In the case of Naik and Mammone (1992), the meta-network receives as input the current weights of the network under training. Its output is a prediction of the final weights of the network under training. Predicted final weights and current weights are used to estimate an optimal weight direction and distance to move in that direction. This term is added to the standard weight update rule of backpropagation. Naik and Mammone (1992) showed that when the meta-network was trained on similar tasks, the algorithm converges faster than normal backpropagation and is less sensitive to initial weight values.

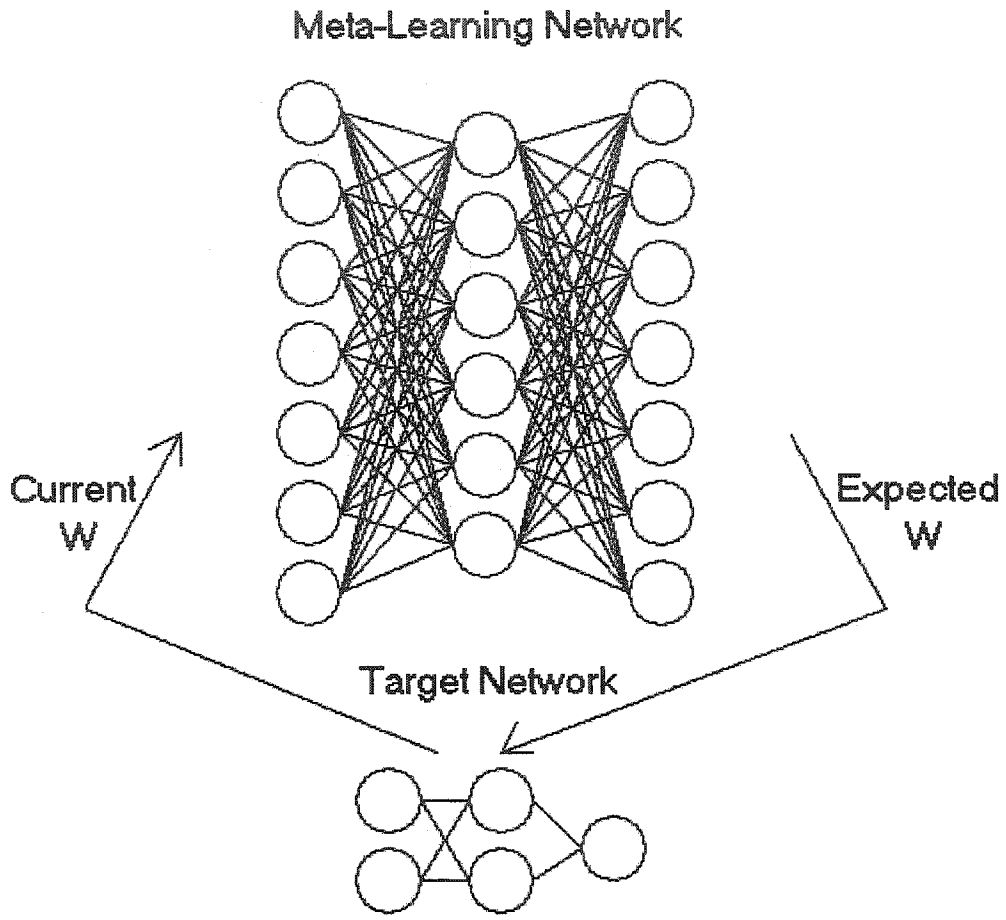


Figure 11: Functional transfer of knowledge through meta-learning.

Another dual network architecture is the idea of *self-refreshing* memory. The work of Ans and Rousset (2000) is a recent example of this older idea. Their specific network architecture is not covered here. Only the general scheme is described in this review. The model requires two similar networks, as shown in Figure 12. First, network 1 is trained on a task. Then, random input patterns are generated and processed through network 1 to generate outputs. These input-output pairs are then used to train network 2. Hence, network 2 is trained through pseudo-rehearsal of network 1. This process transfers the knowledge stored in network 1 to network 2. When network 1 needs to be trained on a new task, its training set is augmented by pseudo-rehearsed patterns from network 2. This way, network 1 learns the new task as well as its old knowledge, which it can refresh from the 'storage' represented by network 2. The main advantage of this technique is that it reduces catastrophic forgetting on sequential learning.

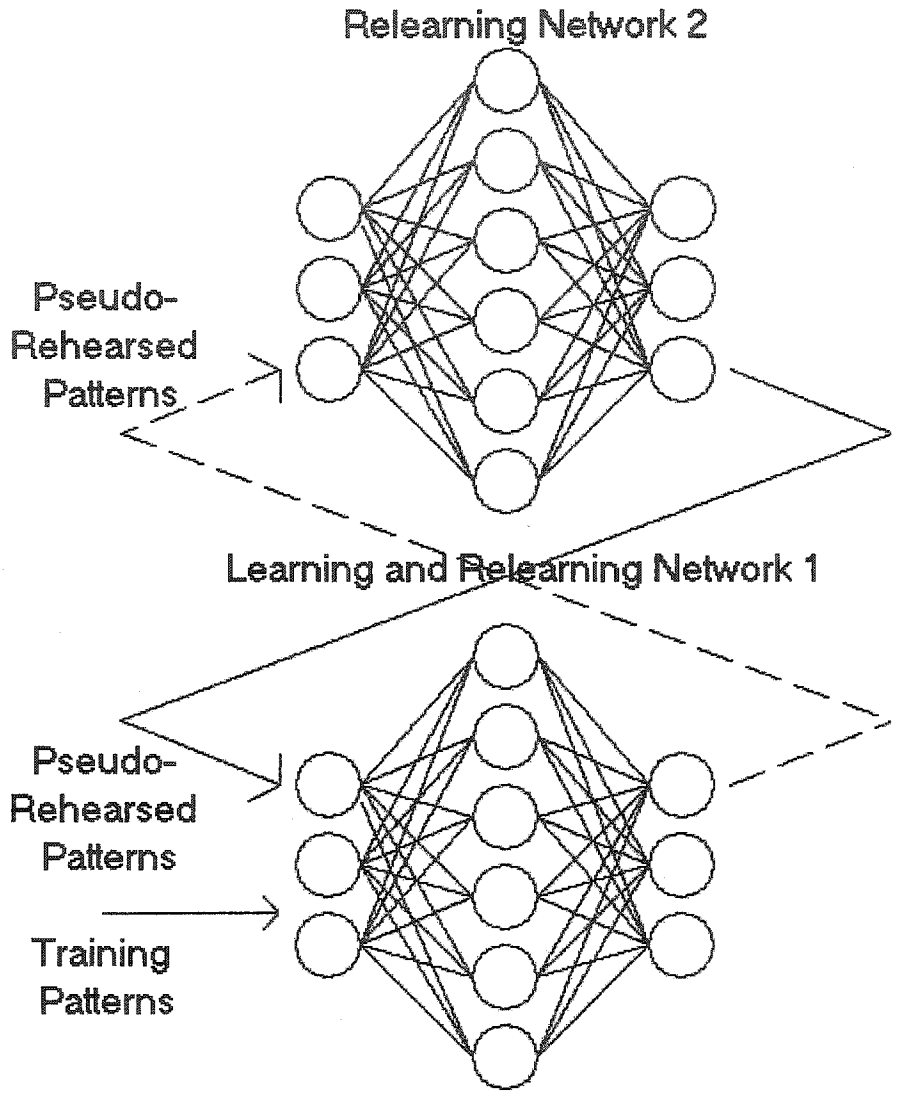


Figure 12: Functional transfer of knowledge through self-refreshing memory.

The last approach to functional transfer of knowledge covered here is called explanation based neural network (EBNN) and was develop by Thrun and Mitchell (1993). EBNN is a connectionist version of the symbolic explanation based learning algorithm (EBL). To maintain the analogy with EBL, it requires some domain theory and a standard training set for the new task. The domain theory must take the form of a differentiable function (as in KBCC, see manuscript 1). It could be a previously trained neural network or a KBANN (see next section) for example. Here, the prior knowledge will be used to explain the training data and help the target network to learn better.

Given a set of input and target patterns ($\{\vec{i}_p\}$ and $\{\vec{t}_p\}$), EBNN first processes the input patterns through the source network. This gives a set of output patterns ($\{\vec{s}_p\}$). Then it computes the derivative of the source knowledge output with respect to its input ($\{\partial\vec{s}_p/\partial\vec{i}_p\}$). These values are the explanation of each target pattern. The target network is trained to learn the target values ($\{\vec{t}_p\}$) as well as the source knowledge derivative values ($\{\partial\vec{s}_p/\partial\vec{i}_p\}$) using Tangent prop (Simard, Victorri, LeCun & Denker, 1992), as shown in Figure 13. Tangent prop is a variation of backpropagation that trains a neural network to fit not only its output to target values, but also its output derivative to target derivative values. The effect of learning derivatives is to learn existing *invariances* in the target function. The learning rate associated with the derivative of each pattern is determined by the distance between the target value and the source output ($\eta_p = 1 - \|\vec{s}_p - \vec{t}_p\|/c$), i.e., by how much the source knowledge explains the data.

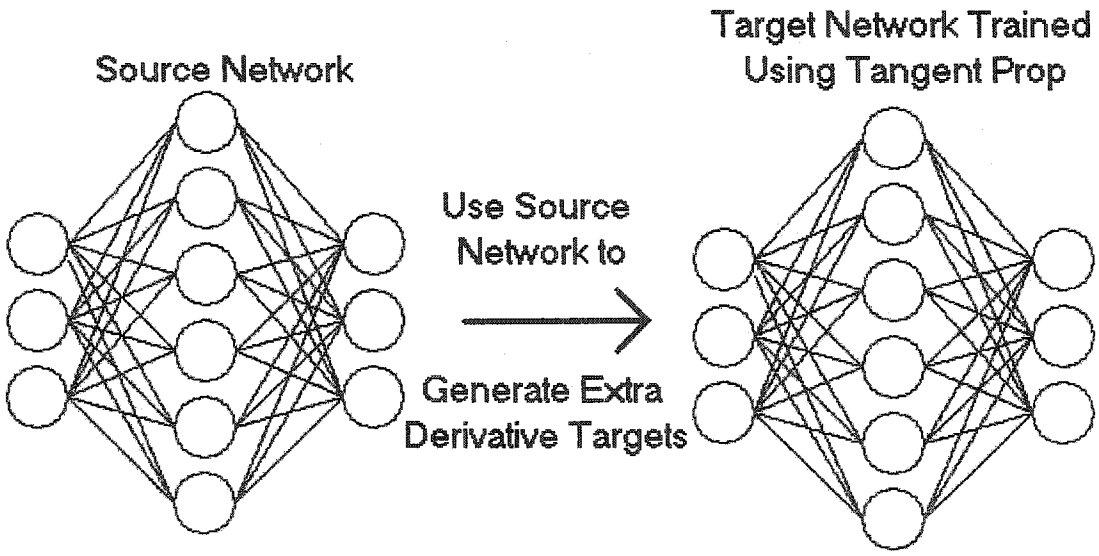


Figure 13: Functional transfer of knowledge through invariance learning.

Symbolic to Neural Network Transfer

A slightly different problem is how to transfer symbolic or rule based knowledge into a new neural network. Pseudo-rehearsing methods partially answer this problem because, if the symbolic rules can be used as functions to generate target patterns, they

can be used with MTL or a dual memory model. But this approach is based on relearning, which is not necessarily the way one may wish to reuse prior knowledge. Towell and Shavlik (1994) devised an algorithm specially made to integrate rules into a neural network named KBANN (knowledge-based artificial neural network). Their goal was to refine rule-based knowledge using neural networks as an intermediate step. They devised KBANN to transform rules into a network. The network was then trained on data for improvement and then rules were re-extracted from it using another algorithm. KBANN could be use to generate an initial network from which one could transfer knowledge to some other neural networks using the methods previously mentioned, or even KBCC (see manuscript 1). The work of Parekh and Honavar (1998) as well as Thrun and Mitchell (1993) are two examples where this technique could be applied.

MANUSCRIPT 1

Knowledge-based Cascade-correlation: Using Knowledge to Speed Learning

Thomas R. Shultz and François Rivest
Laboratory for Natural and Simulated Cognition
Department of Psychology and School of Computer Science
McGill University

http://www.tandf.co.uk. Reprinted, with permission, from:
Shultz, T.R. & Rivest F. (2001) Knowledge-based Cascade-correlation: Using Knowledge to Speed Learning. *Connection Science* 13(1):43-72.

Abstract

Research with neural networks typically ignores the role of knowledge in learning by initializing the network with random connection weights. We examine a new extension of a well-known generative algorithm, cascade-correlation. Ordinary cascade-correlation constructs its own network topology by recruiting new hidden units as needed to reduce network error. The extended algorithm, knowledge-based cascade-correlation (KBCC), recruits previously learned sub-networks as well as single hidden units. This paper describes KBCC and assesses its performance on a series of small, but clear problems involving discrimination between two classes. The target class is distributed as a simple geometric figure. Relevant source knowledge consists of various linear transformations of the target distribution. KBCC is observed to find, adapt, and use its relevant knowledge to significantly speed learning.

1. Existing Knowledge and New Learning

Learning in neural networks is typically done "from scratch", without the influence of previous knowledge. However, it is clear that people make extensive use of their existing knowledge in learning (Heit 1994; Keil 1987; Murphy 1993; Nakamura 1985; Pazzani 1991; Wisniewski 1995). Use of knowledge is likely responsible for the ease and speed with which people are able to learn new material, although interesting interference of knowledge with learning can also occur. Neural networks fail to use knowledge in new learning because they begin learning from initially random connection weights.

Here we examine a connectionist algorithm that uses its existing knowledge to learn new problems. This algorithm is an extension of cascade-correlation (CC), a generative learning algorithm that has proved to be useful in the simulation of cognitive development (Buckingham & Shultz, 1994; Mareschal & Shultz, 1999; Shultz 1998; Shultz, Buckingham, & Oshima-Takane, 1994; Shultz, Mareschal, & Schmidt, 1994; Sirois & Shultz, 1998). Ordinary CC creates a network topology by recruiting new hidden units into a feed-forward network as needed in order to reduce error (Fahlman & Lebiere, 1990). The extended algorithm, called knowledge-based cascade-correlation (KBCC), recruits whole sub-networks that it has already learned, in addition to the untrained hidden units recruited by CC (Shultz & Rivest, 2000a). The extended algorithm thus adapts old knowledge in the service of new learning. KBCC trains connection weights to the inputs of its existing sub-networks to determine whether their outputs correlate well with the network's error on the problem it is currently learning. Consistent with the conventional terminology of the literatures on analogy and transfer of learning, we refer to these existing sub-networks as *source* knowledge and to the current learning task as a *target* problem. These previously learned source networks compete with each other and with conventional untrained candidate hidden units to be recruited into the target network learning the current problem. As we discuss later, KBCC is similar in spirit to recent neural network research on transfer of knowledge, multitask learning, sequential learning, lifelong learning, input re-coding, knowledge insertion, and

modularity, but it incorporates these ideas by learning, storing, and searching for knowledge within a generative network approach.

We first describe the KBCC algorithm, show its learning speed performance on a number of learning problems that could potentially benefit from prior knowledge, and then discuss its advantages and limitations in the context of the current literature on knowledge and learning in neural networks.

2. Description of KBCC

2.1 Overview

Because KBCC is an extension of CC, it uses many of CC's ideas and mathematics. As we describe KBCC, we note particular differences between the two algorithms. Both algorithms specify learning in feed-forward networks, adjust weights based on training examples presented in batch mode, and operate in two phases: output phase and input phase. In output phases, connection weights going into output units are adjusted in order to reduce error at the output units. In input phases, the input weights going into recruitment candidates are adjusted in order to maximize a modified correlation between activation of the candidate and error at the output units. Networks in both algorithms begin with only input and output units. During learning, networks alternate between input and output phases, respectively, depending on whether a new candidate is being recruited or not. We begin with the contrasting features of networks and units, and proceed to discuss the output phase, input phase, and connection scheme.

2.2 Networks and Units

The major new idea in KBCC is to treat previously learned networks just like candidate hidden units, in that they are all candidates for recruitment into a target network. A sample KBCC network with two input units and a bias unit is pictured in Figure 1. This particular network has two hidden units, the first of which is a sub-network and the second of which is a single unit. Later hidden units are installed downstream of existing hidden units.

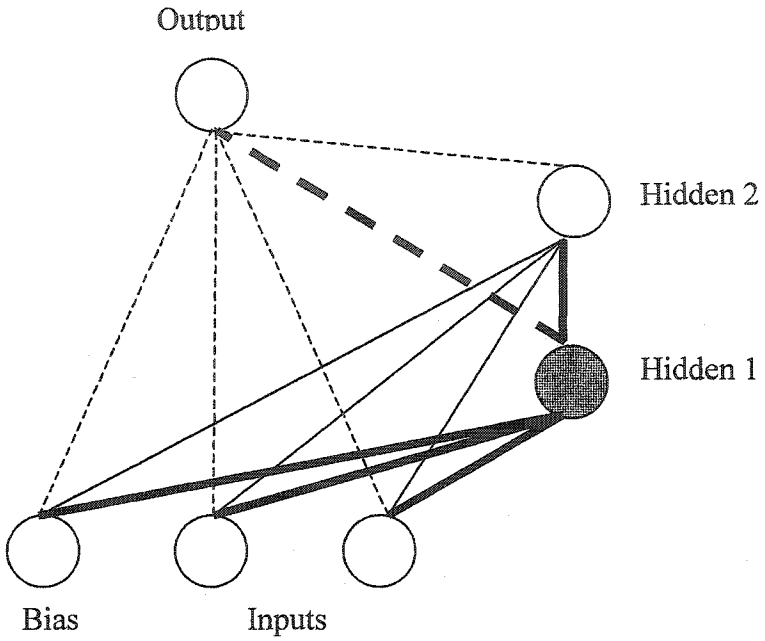


Figure 1. A KBCC network with two hidden units, the first of which is a previously learned sub-network and the second a single unit. The network is shown in the third output phase. Dashed lines represent trainable weights, and solid lines represent frozen weights. Thin lines represent single weights; thick lines represent vectors of weights entering and exiting the recruited sub-network, which may have multiple inputs and multiple outputs.

A single unit and a network both describe a differentiable function, which is what is required for learning in most feed-forward learning algorithms. In the case of a single unit, such as hidden unit 2 in Figure 1, this is a function of one variable, the net input to the unit. Net input to a unit *i* is the weighted sum of its input from other units, computed as:

$$x_i = \sum_j w_{ij} a_j \qquad (1)$$

where *i* indexes the receiving unit, *j* indexes the sending units, *a* is the activation of each sending unit, and *w* is the connection weight between units *j* and *i*.

The function for a single unit is often the centered logistic sigmoid function, in the range of -0.5 to 0.5:

$$f(x) = \frac{1}{1+e^{-x}} - 0.5 \qquad (2)$$

where x is the net input to the unit. Other activation functions used in CC and KBCC are the *asigmoid* (logistic sigmoid) and *Gaussian* functions, both in the range 0.0 to 1.0.

In the case of an existing sub-network, things are a bit more complicated because, unlike a single unit, there may be multiple inputs from each upstream unit and multiple outputs to each downstream unit, as illustrated by hidden unit 1 in Figure 1. For each such sub-network, the input weights and the output weights are each represented as a vector of vectors. Because the internal structure of a previously trained network is known, it can be differentiated in order to compute the slopes needed in weight adjustment, just as is commonly done with single hidden units.

In KBCC, there is a list of current candidate networks, referred to as a parameter called *CandidateNetworksList*.

We refer to the weights entering candidate hidden units and candidate networks as input-side weights. The input-side weights are trained during input phases as explained in section 2.5.

2.3 Notation

Before presenting the algorithm in detail, it is helpful to describe the notation used in the various equations.

$w_{o_u,o}$: Weight between output o_u of unit[1] u and output unit o.

w_{o_u,i_c}: Weight between output o_u of unit u and input i_c of candidate c.

$f'_{o,p}$: Derivative of the activation function of output unit o for pattern p.

$\nabla_{i_c} f_{o_c,p}$: Partial derivative of candidate c output o_c with respect to its input i_c for pattern p.

$V_{o,p}$: Activation of output unit o for pattern p.

$V_{o_c,p}$: Activation of output o_c of candidate c for pattern p.

$V_{o_u,p}$: Activation of output o_u of unit u for pattern p.

$T_{o,p}$: Target value of output o for pattern p.

---

[1] We use *unit* to refer to any of the bias, input, or hidden units except when otherwise stated. Hidden units include both single units and sub-networks.

2.4 Output Phase

In the output phase, all weights entering the output units, called output weights, are trained in order to reduce error. As in CC networks, KBCC networks begin and end their learning career in output phase. The weights that fully connect the network at the start of training are initialized randomly using a uniform distribution with range [*-WeightsRange, WeightsRange*]. The default value is *WeightsRange* = 1.0. A bias unit, with an activation of 1.0, feeds all hidden and output units in the network.

The output weights are trained using the *quickprop* algorithm (Fahlman 1988). The quickprop algorithm is significantly faster than standard back-propagation because it supplements the use of slopes with second-order information on curvature, which it estimates with the aid of slopes on the previous step. Quickprop has parameters for learning rate ε, maximum growth factor μ, and weight decay γ. The learning rate parameter controls the amount of linear gradient descent used in updating output weights. The maximum growth factor constrains the amount of weight change. The amount of decay times the current weight is added to the slope at start of each output phase epoch. [2] This keeps weights from growing too large. The default values for these three parameters are $\varepsilon = 0.175/n$, $\mu = 2.0$, and $\gamma = 0.0002$, respectively, where n is the number of patterns.

The function to minimize in the output phase is the sum-squared error over all outputs and all training patterns:

$$F = \sum_{o} \sum_{p} \left(V_{o,p} - T_{o,p}\right)^2 \tag{3}$$

The partial derivative of F with respect to the weight $w_{o_u,o}$ is given by

$$\frac{\partial F}{\partial w_{o_u,o}} = 2 \sum_{p} \left(V_{o,p} - T_{o,p}\right) f'_{o,p} V_{o_u,p} \tag{4}$$

The activation function for output units is generally the sigmoid function shown in Equation 2. Linear activation functions can also be used for output units. When the sigmoid function is used for output units, a small offset is added to its derivative to avoid getting stuck at the flat points when the derivative goes to 0 (Fahlman 1988). By default, this *SigmoidOutputPrimeOffset* = 0.1.

---

[2] An epoch is a batch presentation of all of the traning patterns.

An output phase continues until any of following criteria is satisfied:

1. When a certain number of epochs pass without solution, there is a shift to the input phase. By default this number of epochs *MaxOutputEpoch* = 100.
2. When error reduction stagnates for few consecutive epochs, there is a shift to the input phase. Error is measured as in Equation 3, and must change by at least a particular proportion of its current value to avoid stagnation. By default, this proportion, called *OutputChangeThreshold*, is 0.01. The number of consecutive output phase epochs over which stagnation is measured is called *OutputPatience* and is 8 by default.
3. When all output activations are within some range of their target value, that is, when $|V_{o,p} - T_{o,p}| \leq$ *ScoreThreshold* for all o outputs and p patterns, victory is declared and learning ceases. By default, *ScoreThreshold* = 0.4, which is generally considered appropriate for units with sigmoid activation functions (Fahlman 1988). The *ScoreThreshold* for output units with linear activation functions would need to be set at the level of precision required in matching target output values.

2.5 Input Phase

In the input phase, a new hidden unit is recruited into the network. This new unit is selected from a pool of candidates. The candidates receive input from all existing network units, except output units, and these input weights are trained by trying to maximize the correlation between activation on the candidate and network error. During this training, all other weights in the network are frozen. The candidate that gets recruited is the one that is best at tracking the network's current error. In KBCC, candidates include not only single units as in CC, but also networks acquired in past learning.

N is the *NumberCandidatesPerType*, which is 4 by default. Weights entering N single-unit candidates are initialized randomly using a uniform distribution with range [-*WeightsRange, WeightsRange*] as in the output phase. Again, the default value is *WeightsRange* = 1.0. For each network in the *CandidateNetworksList*, input weights for N-1 instances are also initialized. Each input-side connection of these units is initialized using the same scheme as for the basic network weights, with one exception. The exception is that one instance of each stored network has its weight matrix initialized

with weights of 1.0 connecting corresponding inputs of target networks to source networks and weights of 0.0 elsewhere. This is to enable use of relevant exact knowledge without much additional training. We call this the *directly connected* version of the knowledge source. Activation functions of the single units are generally all sigmoid, asigmoid, or Gaussian, with sigmoid being the default.

As in output phases, all of these input-side weights are trained with quickprop (with $\varepsilon = 1.0/nh$,[3] where n is the number of patterns and h is the number of units feeding the candidates, $\mu = 2.0$, and $\gamma = 0.0000$). The function to maximize is the average covariance of the activation of each candidate (independently) with the error at each output, normalized by the sum-squared error. For candidate c, the function is given by

$$G_c = \frac{\sum_{o_c}\sum_{o}\left|\sum_{p}\left(V_{o_c,p} - \overline{V}_{o_c}\right)\left(E_{o,p} - \overline{E}_{o}\right)\right|}{\#O_c \cdot \#O \cdot \sum_{o}\sum_{p} E_{o,p}^2} \qquad (5)$$

where $\overline{E}_o$ is the mean error at output unit o, and $\overline{V}_{o_c}$ is the mean activation output o_c of candidate c.

The output error at pattern p is

$$E_{o,p} = \begin{cases} \left(V_{o,p} - T_{o,p}\right) & \text{if} \quad RawError = true \\ \left(V_{o,p} - T_{o,p}\right) f'_{o,p} & otherwise \end{cases} \qquad (6)$$

G_c is standardized by both the number of outputs for the candidate c ($\#O_c$) and the number of outputs in the main network ($\#O$). By default, $RawError = false$.[4]

The partial derivative of G_c with respect to the weight w_{o_u,i_c} between output o_u of unit u and input i_c of candidate c is given by

---

[3] The $1/n$ (output phase) and $1/nh$ (input phase) fraction cannot be described as part of the objective function of their respective phases as in standard back-propagation because ε is not used in the quadratic estimation of the curve in quickprop. It is a heuristic from Fahlman (1988) to set ε dynamically.

[4] The variation of the error function $E_{o,p}$, which depends on *RawError*, comes from Fahlman's (1991) CC code (ftp://ftp.cs.cmu.edu/afs/cs/project/connect/code/supported/cascor-v1.2.shar).

$$\frac{\partial G_c}{\partial w_{o_u,i_c}} = \frac{\sum_{o_c}\sum_{o}\sum_{p} \sigma_{o_c,o}\left(E_{o,p} - \overline{E}_o\right) \nabla_{i_c} f_{o_c,p} V_{o_c,p}}{\#O_c \cdot \#O \cdot \sum_{o}\sum_{p} E_{o,p}^2} \quad (7)$$

where $\sigma_{o_c,o}$ is the sign of the covariance between the output o_c of candidate c and the activation of output unit o.

An input phase continues until either of following criteria is met:

1. When a certain number of input phase epochs passes without solution, there is a shift to output phase. By default this *MaxInputEpoch* = 100.
2. When at least one correlation reaches a *MinimalCorrelation* (default value = 0.2) and correlation maximization stagnates for few consecutive input phase epochs, there is a shift to output phase. Correlation is measured as in Equation 5, and must change by at least a particular proportion of its current value to avoid stagnation. By default, this proportion, called *InputChangeThreshold*, is 0.03. The number of consecutive input phase epochs over which correlation stagnation is measured is called *InputPatience* and is 8 by default.

When a criterion for shifting to output phase is reached, a set of weights is added from the outputs of the best candidate to each output of the network. All other candidate units are discarded, and the newly created weights are initialized with small random values (between 0.0 and 1.0), with the sign opposite to that in correlation.

2.6 Connection Scheme in KBCC

Figure 2 shows the connection scheme for a sample KBCC network with two inputs, two outputs, one recruited network, and a recruited hidden unit. The recruited network, labeled H1 because it was the first recruited hidden unit, has two input units, two output units, and a single hidden unit, each labeled with a prime (') suffix. The main network and the recruited sub-network each have their own bias unit. Figure 2 reveals that the recruited sub-network is treated by the main network as a computationally encapsulated module, receiving input from the inputs and bias of the main network and sending output to later hidden units and the output units. Other than that, the main network has no interaction with the work of the sub-network.

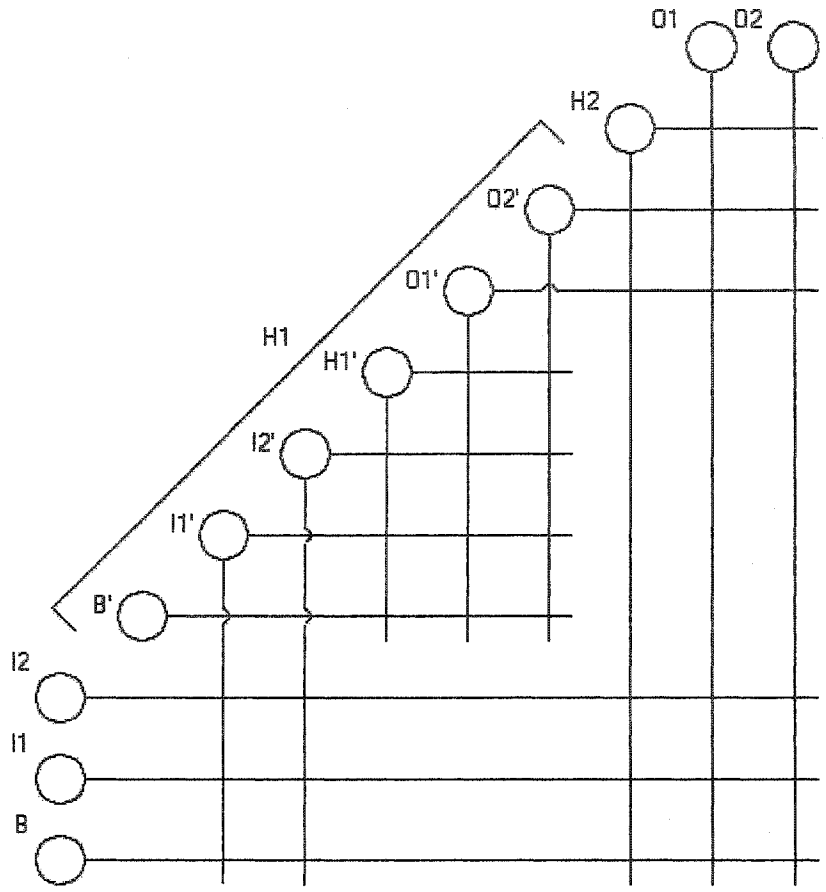


Figure 2. Connection scheme for a sample KBCC network with two inputs, two outputs, one recruited network, and a recruited hidden unit.

3. Applications of KBCC

To evaluate the behavior of KBCC, we applied it to learning in two different paradigms. One paradigm tests whether KBCC can find and use its relevant knowledge in the solution of a new problem and whether this relevant knowledge shortens the time it takes to learn the new problem. A second paradigm tests whether KBCC can find and combine knowledge of components to learn a new, more complex problem comprised of these components, and whether use of these knowledge components speeds learning. In each paradigm there are two phases, one in which source knowledge is acquired and a second in which this source knowledge might be recruited to learn a target problem. These experiments are conducted with toy problems with a well-defined structure so that we can clearly assess the behavior of the KBCC algorithm. In each problem, networks

learn to identify whether a given pattern falls inside a class that has a two-dimensional uniform distribution. The networks have two linear inputs and one sigmoid output. The two inputs describe two real-valued features; the output indicates whether this point is inside or outside a class of a particular distribution with a given shape, size, and position. The input space is a square centered at the origin with sides of length 2. Target outputs specify that the output should be 0.5 if the point described in the input is inside the particular class and -0.5 if the point is not in the class. In geometric terms, points inside the target class fall within a particular geometric figure; points outside of the target class fall outside of this figure. Networks are trained with a set of 225 patterns forming a 15 x 15 grid covering the whole input space including the boundary. For each experiment, there are 200 randomly determined test patterns uniformly distributed over the input space. These are used to test generalization, and are never used in training. We used this task to facilitate design and description of problems, variation in knowledge relevance, and identification of network solutions (by comparing output plots to target shapes). These problems, although small and easy to visualize, are representative of a wide range of classifier and pattern recognition problems. Knowledge relevance involved differences in the position and shape of the distribution of patterns that fell within the designated class (or figure). Degree of relevance was indexed by variation in the amounts of translation, rotation, and scaling. So-called *irrelevant* source knowledge involved learning a class whose distribution has a different geometric shape than the target class.

We ran 20 KBCC networks in each condition of each experiment in order to assess the statistical reliability of results, with networks differing in initial output and input weights. Learning speed was measured by epochs to learn. Use of relevant knowledge was measured by identifying the source of knowledge that was recruited during input phases.

In these experiments, networks learning a target task have zero, one, or two source networks to draw upon in different conditions. In each input phase of single source experiments, there are always eight candidates, four of them being previously learned networks and four of them being single units. In control conditions without knowledge (no source networks), all eight candidates are single units. These control networks are essentially CC networks. In conditions with two source networks in

memory, there are three candidates representing one source network, three candidates representing the other source network, and three single unit candidates. The reason for having multiple candidates for each unit and source network is to be able to provide a variety of initial input weights at the start of the input phase. This enables networks to try a variety of different mappings of the target task to existing knowledge.

4. Finding and Using Relevant Knowledge

We did two kinds of experiments to assess the impact of source knowledge on learning a target task. In one kind of experiment, we varied the relevance of the single source of knowledge the network possessed to determine whether KBCC would learn faster if it had source knowledge that was more relevant. In a second kind of experiment, we gave networks two sources of knowledge, varying in relevance to a new target problem, to discover whether KBCC would opt to use more relevant source knowledge. To assess the generality of our results, we conducted both types of experiments with three different sets of linear transformations of the input space: translation, size changes, and rotation. In all of these experiments, KBCC networks acquired source knowledge by learning one or two problems and then learned another, target problem for which the impact of the previously acquired source knowledge could be assessed. We first consider problems of translation, then sizing, and finally rotation.

4.1 Translation[5]

In translation problems, degree of knowledge relevance was varied by changing the position of two-dimensional geometric figures. The target figure in the second (or target) phase of knowledge-guided learning was a rectangle with width of 0.4 and height of 1.6 centered at (-4/7, 0) in the input space. For translation problems, we first consider the effects of single-source knowledge on learning speed in the target phase and then the knowledge that is selected when two sources of knowledge are available.

---

[5] A preliminary version of the translation results were presented in Shultz and Rivest (2000a).

4.1.1 Effects of Single-source Knowledge Relevance on Learning Speed

In this experiment, networks had to learn a rectangle (named RectL) positioned a bit to the left of center in the input space, after having previously learned a rectangle, or two rectangles, or a circle at particular positions in the input space. The various experimental conditions are shown in Table I, in terms of the name of the condition, a description of the stimuli that had been previously learned in phase 1, and the relation of those stimuli to the target rectangle (RectL). Previous, phase-1 (or source) learning created the various knowledge conditions for the new learning of the target rectangle in the target phase. Conditions included exact knowledge (i.e., identical to the target), exact but overly complex knowledge, relevant knowledge that was either near to or far from the target, overly complex knowledge that was far from the target, and irrelevant knowledge. In a control condition, networks had no knowledge at all when beginning the target task, which is equivalent to ordinary CC.

Table I: Single-source Knowledge Conditions for Translation Experiments

Name	Description	Relation to target
RectL	Rectangle centered at (-4/7, 0)	Exact
2RectLC	2 rectangles, centered at (-4/7, 0) and (0, 0)	Exact/near, overly complex
RectC	Rectangle centered at (0, 0)	Near relevant
RectR	Rectangle centered at (4/7, 0)	Far relevant
2RectCR	2 rectangles, centered at (0, 0) and (4/7, 0)	Near/far, overly complex
Circle	Circle centered at (0, 0) with radius 0.5	Irrelevant
None	No knowledge	None

A factorial ANOVA of the epochs required to reach victory yielded a main effect of knowledge condition, $F(6, 133) = 33$, $p < .0001$. Mean epochs to victory in each condition, with standard deviation bars and homogeneous subsets, based on the *LSD* post hoc comparison method, are shown in Figure 3. Means within a homogeneous subset are not significantly different from each other. Figure 3 reveals that exact knowledge, whether alone or embedded in an overly complex structure produced the fastest learning, followed by relevant knowledge, distant and overly complex knowledge and irrelevant knowledge, and finally the control condition without any knowledge.

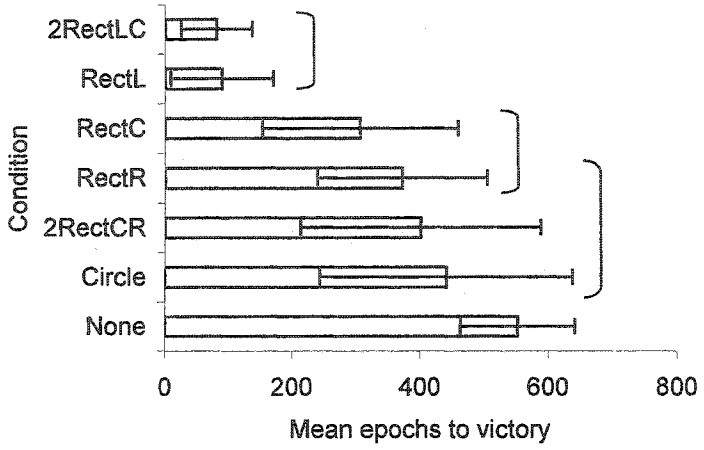


Figure 3. Mean epochs to victory in the target phase of the translation experiment, with standard deviation bars and homogeneous subsets.

Some example output activation diagrams for one representative network from this simulation are shown in Figure 4. In these plots, white regions of the input space are classified as being inside the rectangle (network response > 0.1), black regions outside of the rectangle (network response < -0.1), and gray areas are uncertain, meaning that the network gives a borderline, unclassifiable response somewhere between -0.1 and 0.1. The 225 target training patterns, forming a 15 x 15 grid covering the whole input space, are shown as points in each plot. Figure 4a shows the source knowledge learned by this network in the exact but overly complex condition. The two white regions indicate the two rectangles constituting the target class for this condition. Such shapes are somewhat irregular, even if completely correct with respect to the training patterns, because they are produced by sampling the network on a fine grid of 220 x 220 input patterns.

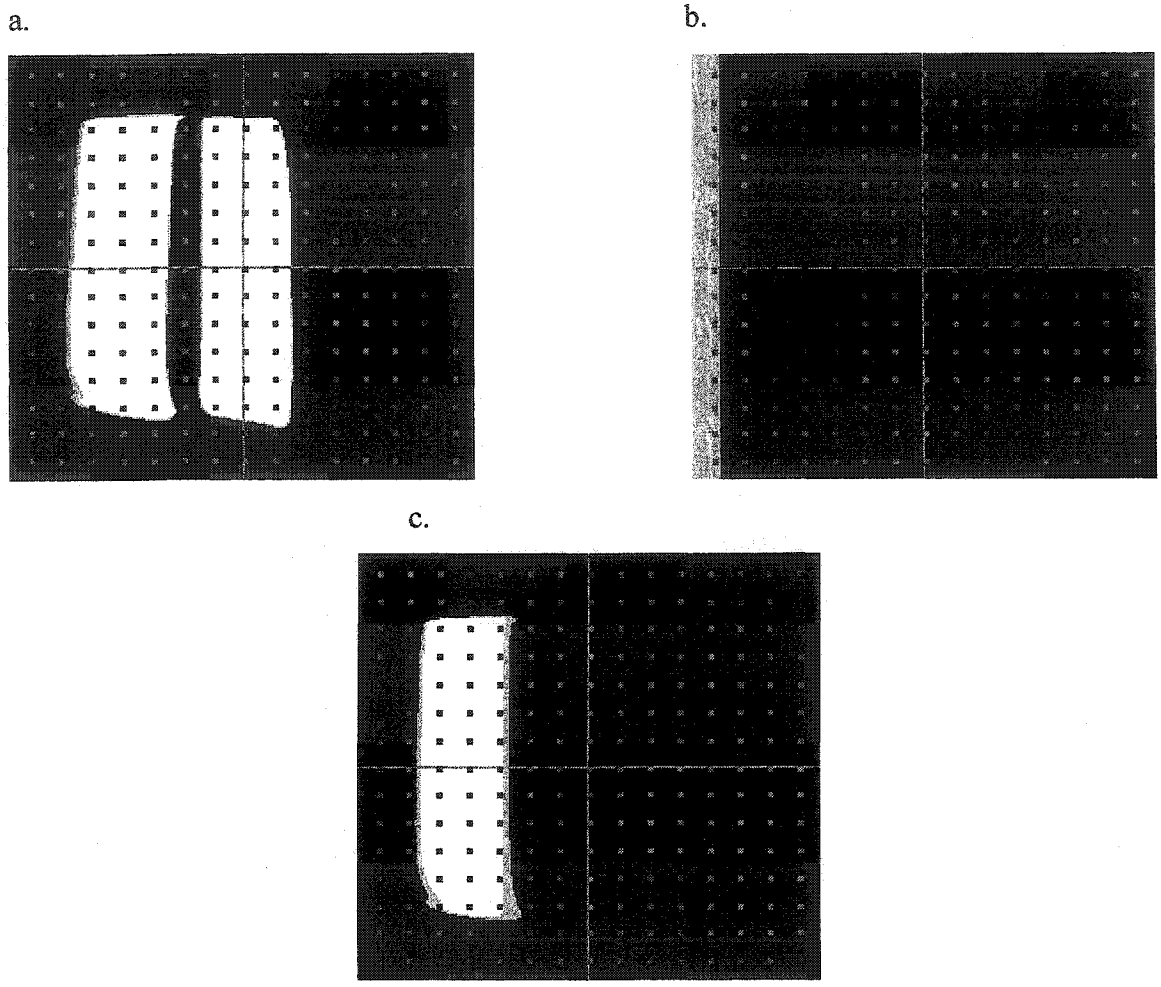


Figure 4. Output activation diagrams showing exact but overly complex source knowledge (a), the target network at the end of the first output phase (b), and the target solution at the end of the second output phase (c) after recruiting the source knowledge in (a).

Figure 4b shows this same KBCC system's output at the end of the first output phase of target training. There are no white regions in this plot because the network has learned to classify most of the input patterns as being outside of the target class; the network is, on our definition, unsure of patterns within the first column of the input space. Because only 33 of the 225 training patterns fall within the target class, this is the best that the target network can do without any hidden units. Such behavior was common for the first output phase of learning a small rectangular target in every condition of every experiment reported in this paper.

Figure 4c shows this network's final solution at the end of the second output phase of target training, after having recruited the source knowledge shown in Figure 4a. In this single-source experiment, exact but overly complex source knowledge was very effective in speeding up learning, as reported in Figure 3. Comparison of Figures 4a and 4c suggests that KBCC uses the recruited source network only when the input represented on the *x*-axis is less than about -3/14; otherwise it uses its direct input-to-output weights. Examination of the output weights from the bias unit, the *x*-axis input unit, and the recruited hidden network confirmed that this is the case. These weights are such that the recruited hidden network can raise the weighted sum feeding the output unit above 0.0 only in the region of the target rectangle. Notice how closely the shape of the final solution in Figure 4c resembles that of the exactly correct portion of the source network in Figure 4a.

Because one of the candidate source networks is initialized using direct input connections (i.e., weights that map input *i* to source network input *i* are set to 1.0 and all others to 0.0), KBCC always recruits that candidate source first in exact knowledge conditions. For 65% of these exact source networks, no further recruitment was necessary; for the remaining 35%, some additional recruitment was necessary to adjust to a few borderline patterns. A directly connected source candidate network was much less likely to be recruited in the close (25%) and far (0%) relevant conditions.

4.1.2 Selection of Relevant Knowledge from Two Sources

In this experiment, networks first learned two tasks of varying relevance to the target task of learning RectL, the rectangle placed slightly to the left of the origin. The names of the various knowledge conditions, their relations to the target, and the mean times each network was recruited during input phases of target learning are shown in Table II. Descriptions of the figures designated by each condition name were provided in Table 1. The two recruitment means in each row were compared with a *t*-test for paired samples, except in the Exact vs. Relevant condition, where there was no variation in either variable. In every other case, the mean difference was significant at $p < .001$, $df = 19$. The pattern of differences shows that target networks preferred exact knowledge, even when it was embedded in overly complex knowledge. They also preferred simple

exact knowledge to overly complex knowledge that had exact knowledge embedded within it. Interestingly, a circle source was more often recruited than a source rectangle positioned at the right. Only two single-hidden-units were recruited by the 120 networks in this experiment.

Table II: Dual-source Knowledge Conditions and Mean Networks Recruited in Translation Experiments

Name	Relation to target	Mean networks recruited			
		RectL	RectR	2RectLC	Circle
RectL, RectR	Exact vs. Relevant	1.0	0.0	n/a	n/a
RectL, 2RectLC	Exact vs. Overly complex	0.95	n/a	0.15	n/a
RectL, Circle	Exact vs. Irrelevant	1.05	n/a	n/a	0.0
RectR, 2RectLC	Relevant vs. Overly complex	n/a	0.15	1.25	n/a
RectR, Circle	Relevant vs. Irrelevant	n/a	1.25	n/a	2.55
2RectLC, Circle	Overly complex vs. Irrelevant	n/a	n/a	1.45	0.15

The results of this dual-source experiment make sense given our analysis of the single-source experiment. Exact source knowledge is preferred over inexact source knowledge because it makes a nearly perfect match when accessed with direct connections. Overly complex exact source knowledge is less apt to be recruited than is simple exact source knowledge because it correlates less well with error in the case of directly connected sources. Perhaps the circle source knowledge is recruited over the far source rectangle because the circle is closer to the target rectangle even though it is the wrong shape.

4.2 Sizing

In sizing problems, knowledge relevance was varied by changing the size of two-dimensional geometric figures. The target figure in the second phase of knowledge-guided learning was a rectangle as were the figures in several of the knowledge conditions. Rectangles were always centered at (0, 0) in the input space and always had a height of 22/14.

4.2.1 Effects of Single-source Knowledge Relevance on Learning Speed

In this experiment, several knowledge conditions varied the width of the first-learned rectangle. Because scaling the width up and scaling the width down do not

produce the same results, we included conditions with either small or large target rectangles. The various conditions, which also included irrelevant knowledge in the form of a circle and no knowledge at all, are shown in Table III.

Table III: Single-source Knowledge Conditions for Sizing Experiments

Name	Description	Relation to target RectL	Relation to target RectS
RectS	Rectangle of width 6/14	Far relevant	Exact
RectM	Rectangle of width 14/14	Near relevant	Near relevant
RectL	Rectangle of width 22/14	Exact	Far relevant
Circle	Center at (0, 0), radius 0.5	Irrelevant	Irrelevant
None	No knowledge	None	None

A factorial ANOVA of the epochs to victory when the small rectangle was the target yielded a main effect of knowledge condition, $F(4, 95) = 103, p < .0001$. The mean epochs to victory, with standard deviation bars and homogeneous subsets, based on the *LSD* post hoc comparison method, are shown in Figure 5. Relevant knowledge, regardless of distance from the target, produced faster learning than did irrelevant knowledge and no knowledge. This suggests that scaling down in size is not much affected by the amount of scaling required. The relatively few epochs required in phase 2 indicates that scaling down in size is relatively easy for these networks to learn.

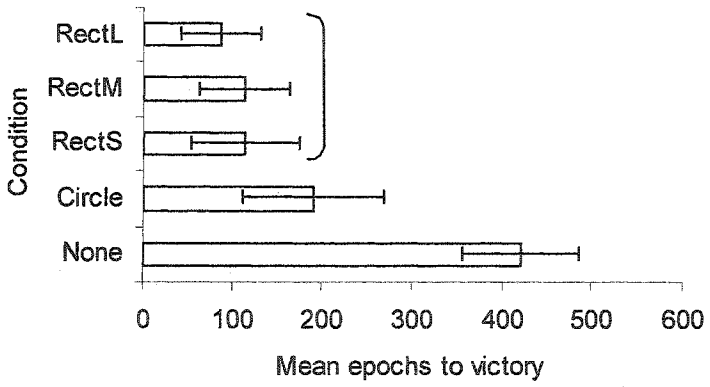


Figure 5. Mean epochs to victory in the target phase of the small rectangle condition of the sizing experiment, with standard deviation bars and homogeneous subsets.

A factorial ANOVA of the epochs to victory when the large rectangle was the target also yielded a main effect of knowledge condition, $F(4, 95) = 74, p < .0001$, but with a somewhat different pattern of results. The mean epochs to victory, with standard

deviation bars and homogeneous subsets, based on the *LSD* post hoc comparison method, are shown in Figure 6. Exact knowledge yielded the fastest learning, followed in turn by near relevant knowledge, far relevant knowledge, and finally by no knowledge and irrelevant knowledge. This means that scaling up in size gets more difficult with the amount of scaling required. In this case, irrelevant knowledge did not speed up learning, as compared to the no-knowledge control. Examination of phase-1 source acquisition results confirmed that small rectangles were easier to learn than large rectangles, in terms of both hidden units recruited and epochs to learn.

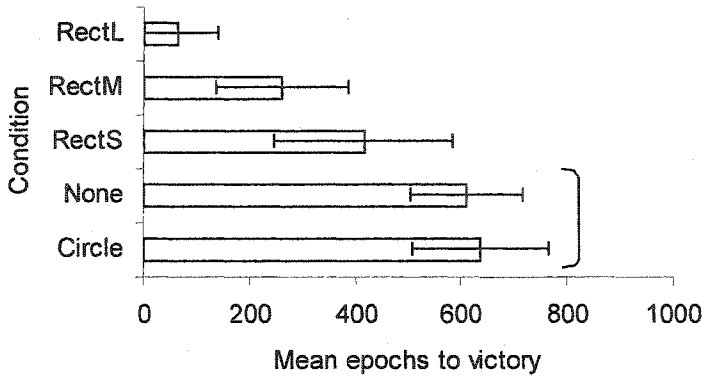


Figure 6. Mean epochs to victory in the target phase of the large rectangle condition of the sizing experiment, with standard deviation bars and homogeneous subsets.

When learning a small target rectangle, the percent of networks recruiting a directly connected source network decreased from 100% in the exact source knowledge condition to 80% in the near relevant source knowledge condition to 45% in the far relevant source knowledge condition. Figures 7 and 8 present output activation diagrams for networks learning a small target rectangle, recruiting either near relevant or far relevant directly connected source knowledge, respectively. Again, in both cases, there is a striking resemblance between the shape of the source knowledge and that of the final solution. As with translation experiments, the networks here learn to classify all patterns as being outside of the target class during the first output phase.

a. b.

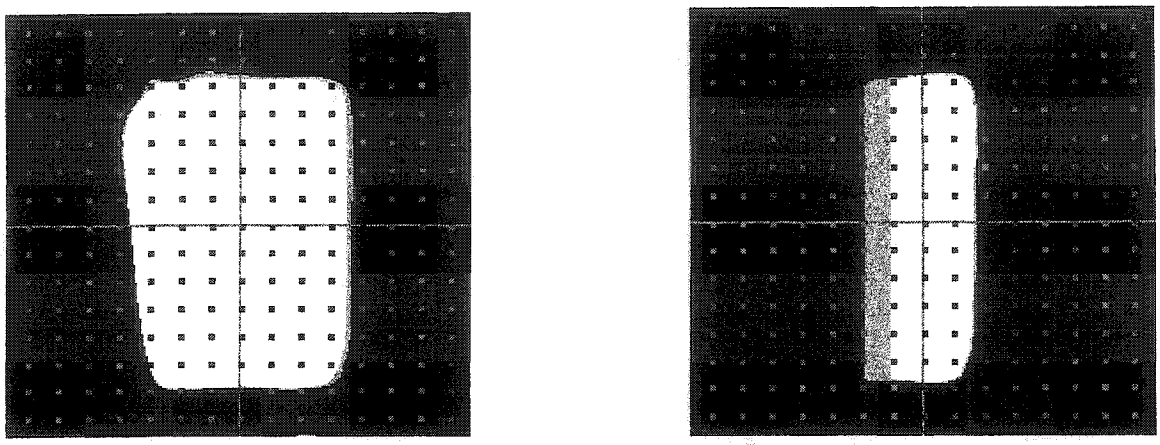


Figure 7. Output activation diagrams for a network learning a small rectangle, showing near relevant source knowledge (a) and the final target solution at the end of the second output phase (b) after recruiting the knowledge in a.

a. b.

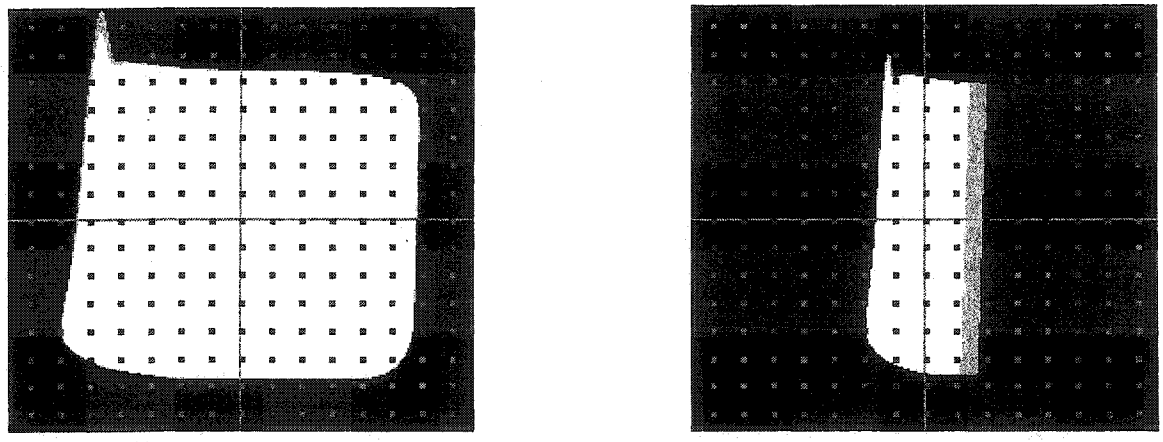
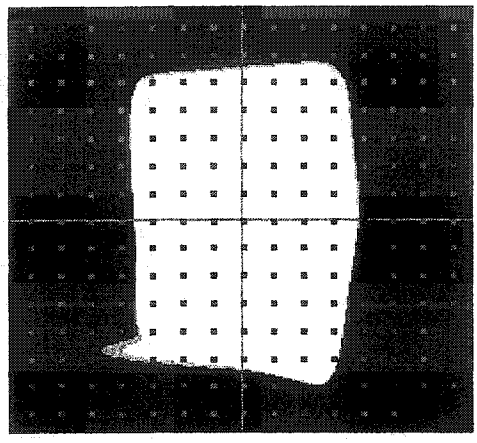


Figure 8. Output activation diagrams for a network learning a small rectangle, showing far relevant source knowledge (a) and the final target solution at the end of the second output phase (b) after recruiting the knowledge in a.

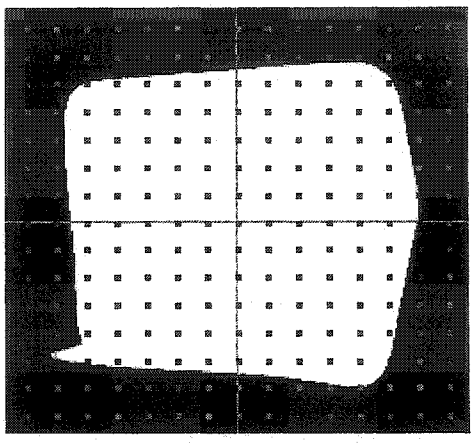
However, when learning a large target rectangle, networks do the opposite; that is, they learn to classify all patterns as being inside of the target class during the first output phase. This is because many more of the training patterns (121 of 225) fall within the target class when the target is a large rectangle. The percent of networks recruiting the directly connected source network when learning a large target rectangle was 100% in the exact and near source knowledge conditions and 75% in the far relevant source

knowledge condition. Figures 9 and 10 show output activation diagrams for networks learning a large target rectangle, with either near relevant or far relevant source knowledge, respectively.

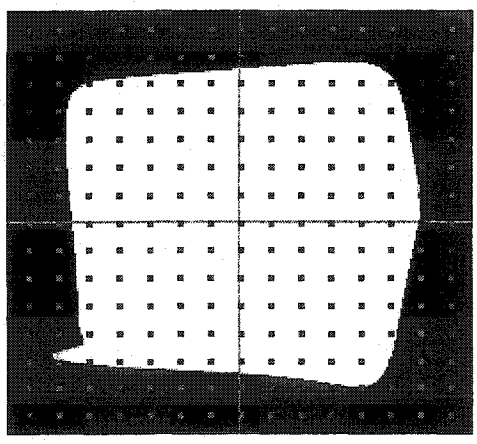
a.



b.



c.



d.

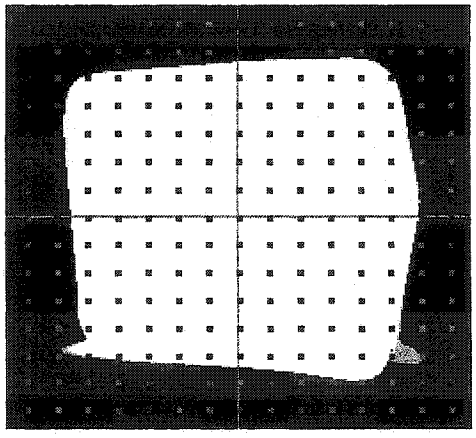


Figure 9. Output activation diagrams for a network learning a large rectangle, showing near relevant source knowledge (a) and target solutions at the end of the second (b), third (c), and fourth and final (d) output phases.

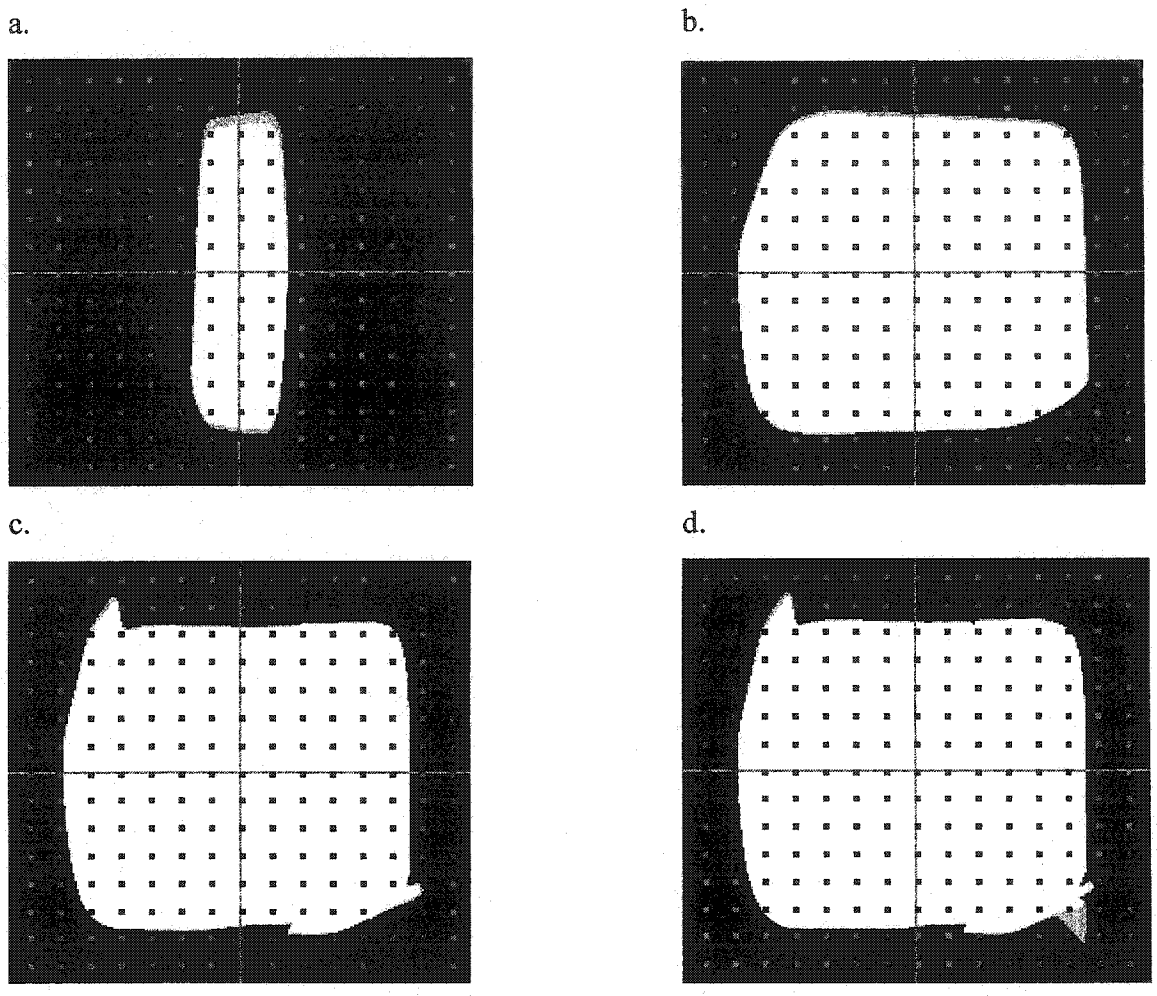


Figure 10. Output activation diagrams for a network learning a large rectangle, showing far relevant source knowledge (a) and target solutions at the end of the second (b), fifth (c), and sixth and final (d) output phases.

Any lingering error during target learning, whether scaling down to a small rectangle or scaling up to a large rectangle, involves patterns near the four corners of the target rectangle, such corners being regions of intersecting hyper-planes being learned by the network. When scaling down to learn a small target rectangle, network recruitment sharpens these corners, making target learning rather fast. In contrast, when scaling up to a large target rectangle, network recruitment smoothes these corners, thus prolonging target learning in order to re-sharpen the corners. When scaling up to a large rectangle, the amount of corner smoothing and eventual re-sharpening grows with the degree of scaling. Because no additional sharpening of corners is required when scaling down to a

small rectangle, learning speed is rather fast and does not vary with the degree of scaling required.

Mean input connection weights for the critical *x*-axis input units learned in the sizing experiment while recruiting a directly connected source network are plotted in Figure 11. Because recruiting exact knowledge requires no rescaling of inputs, these connection weights are about 1, regardless of the size of the target rectangle. With a small target rectangle, these weights increase with the amount of scaling required; with a large target rectangle, these weights decrease with the amount of scaling required. Such trends make sense because when scaling down to a small target rectangle, the inputs to the small target would need to be scaled up with larger weights in order to effectively use the larger source network. In contrast, when scaling up to a large target rectangle, the inputs to the large target would need to be scaled down with smaller weights in order to effectively use the smaller source network. During these recruitments, connection weights for *y*-axis input units were always about 1 because rectangle height was constant, and all other connection weights were about 0 because they were unimportant.

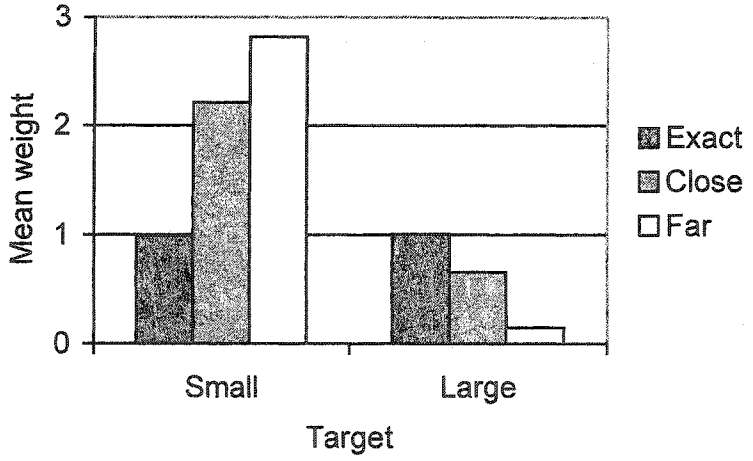


Figure 11. Mean input connection weights for *x*-axis input units learned in the sizing experiment while recruiting a directly connected source network.

4.2.2 Selection of Relevant Knowledge from Two Sources

In this experiment, networks first learned two tasks of varying relevance to the target task. The names of the various knowledge conditions, their relations to the target, and the mean times each network was recruited during input phases are shown in Table IV. The descriptions of the figures associated with each condition name were provided in Table III. The two means in each row were compared with a *t*-test for paired samples. Results are shown in the last two columns of Table IV. Exact knowledge was preferred over relevant or irrelevant knowledge. Relevant knowledge was preferred over irrelevant knowledge only when scaling down to a smaller rectangle. The large number of recruited networks in the relevant vs. irrelevant, scaling-up condition reflects the relative difficulty of learning in this condition. Again, the longer that learning continues the more recruitment is required. In this experiment, two of the 120 networks each recruited 1 single-hidden-unit, and two others recruited 2 single-hidden-units, for a total of 6. All six of these hidden units recruited were all in the scaling-up conditions of the experiment.

Table IV: Dual-source Knowledge Conditions and Mean Networks Recruited for Sizing Experiments

Name	Relation to target	Mean networks recruited			$t(19)$	$p <$
		RectS	RectL	Circle		
Target: RectS						
RectS, RectL	Exact vs. Relevant	1.05	0.60	n/a	3.33	.005
RectS, Circle	Exact vs. Irrelevant	1.00	n/a	0.45	3.58	.005
RectL, Circle	Relevant vs. Irrelevant	n/a	1.50	0.45	4.70	.001
Target: RectL						
RectL, RectS	Exact vs. Relevant	0.15	1.20	n/a	11.92	.001
RectL, Circle	Exact vs. Irrelevant	n/a	1.05	0.0	21.00	.001
RectS, Circle	Relevant vs. Irrelevant	2.65	n/a	3.40	1.25	ns

The results of this dual-source sizing experiment make sense given our analysis of the single-source sizing experiment. Exact source knowledge is preferred over inexact source knowledge because it makes a nearly perfect match when accessed with direct connections. When scaling down to a small rectangle, relevant inexact source knowledge is preferred to irrelevant source knowledge because the recruitment sharpens the critical corners of the target figure, which is rectangular like the relevant sources. In contrast,

when scaling up to a large rectangle, there is no advantage for relevant source knowledge because recruiting smoothes the critical target corners thus requiring additional re-sharpening through further learning.

4.3 Rotation

In rotation problems, knowledge relevance was varied by rotating a rectangle of size 1.5 x 0.5 or a cross comprising two such rectangles that were offset 90 degrees. All figures were centered at (0, 0). The target figure in the second phase of knowledge-guided learning was a vertically oriented rectangle (Rect90).

4.3.1 Effects of Single-source Knowledge Relevance on Learning Speed

In this experiment, networks had to learn a vertical rectangle (Rect90) after having previously learned a rectangle or a cross or a circle. The various experimental conditions are shown in Table V.

Table V: Single-source Knowledge Conditions for Rotation Experiments

Name	Description	Relation to target
Rect90	Vertical rectangle	Exact
Rect45	Diagonal rectangle	Near relevant
Rect0	Horizontal rectangle	Far relevant
Cross90	2 superposed rectangles	Exact/far, overly complex
Cross45	2 superposed rectangle forming diagonal cross	Near, overly complex
Circle	Circle centered at (0, 0) with radius 0.5	Irrelevant
None	No knowledge	None

A factorial ANOVA of the epochs to victory yielded a main effect of knowledge condition, $F(6, 133) = 20, p < .0001$. The mean epochs to victory, with standard deviation bars and homogeneous subsets, based on the *LSD* post hoc comparison method, are shown in Figure 12. Exact knowledge produced the fastest learning, followed by exact knowledge embedded in an overly complex structure and irrelevant knowledge, near and distant relevant knowledge, distant and overly complex knowledge, and finally the control condition without any knowledge.

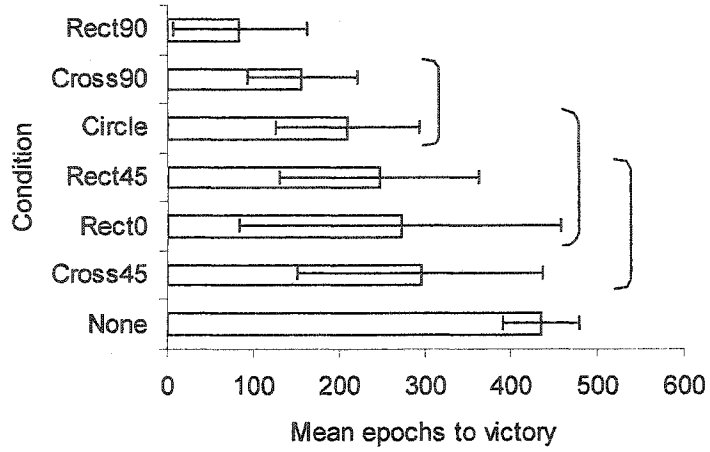
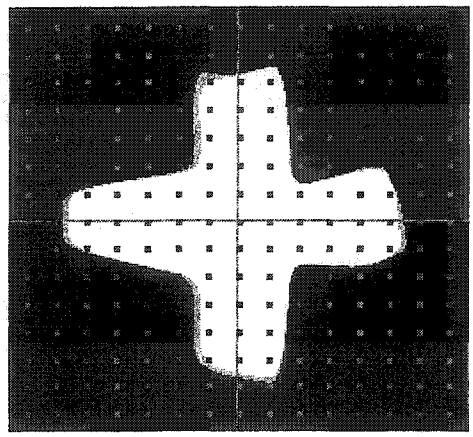


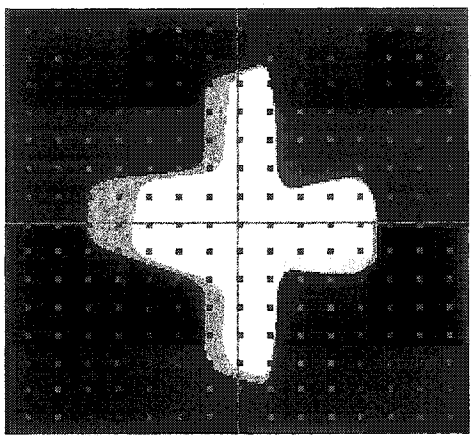
Figure 12. Mean epochs to victory in the target phase of the rotation experiment, with standard deviation bars and homogeneous subsets.

Figure 13 presents output activation diagrams for a network learning a vertical rectangle after recruiting directly connected, exact but overly complex source knowledge. The target solution is accomplished by gradually shrinking the horizontal arms of the recruited cross via more recruiting and adjusting of direct input-to-output weights. The top and bottom of the vertical portion of the recruited cross (Figure 13a) resemble those in the emerging (Figure 13b) and final target solutions (Figure 13c). The directly connected version of the source knowledge was recruited by 100% of networks in the exact knowledge condition and by 65% of networks in the exact but overly complex condition. This explains why these two conditions showed the fastest learning. In contrast, most networks in other conditions either failed to recruit the directly connected version of a knowledge source or failed to develop weights that could make effective use of that recruitment. Nonetheless, all knowledge-laden conditions were superior in learning speed to the condition without any previous knowledge. It is difficult to pinpoint the nature of all of these advantages of knowledge over no knowledge.

a.



b.



c.

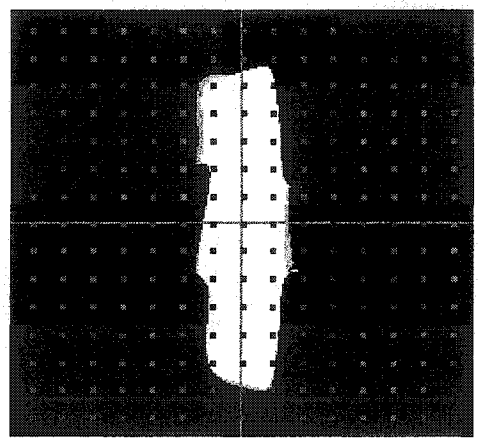


Figure 13. Output activation diagrams for a network learning a vertical rectangle after recruiting directly connected exact but overly complex source knowledge (a). Target solutions are shown at the end of the second (b) and third and final (c) output phases.

4.3.2 Selection of Relevant Knowledge from Two Sources

In this experiment, networks first learned two tasks of varying relevance to the target task of learning a vertical rectangle (Rect90). The names of the various knowledge conditions, their relations to the target, and the mean times each network was recruited during input phases are shown in Table VI. The descriptions of the figures for each condition were provided in Table V. The two means in each row were compared with a *t*-test for paired samples. In each case, except the last, the mean difference was significant, $p < .001$, $df = 19$. Exact knowledge was preferred over relevant, irrelevant, or overly complex knowledge. Exact knowledge embedded within overly complex knowledge was

also preferred over relevant, but inexact knowledge. The nominally irrelevant circle networks were recruited more often than relevant, inexact knowledge and as often as overly complex, but exact knowledge. Only one single-hidden-unit was recruited by the 120 networks in this experiment.

Table VI: Dual-source Knowledge Conditions and Mean Networks Recruited for Rotation Experiments

Name	Relation to target	Mean networks recruited			
		Rect90	Rect0	Cross90	Circle
Rect90, Rect0	Exact vs. Relevant	1.35	0.25	n/a	n/a
Rect90, Cross90	Exact vs. Overly complex	1.00	n/a	0.35	n/a
Rect90, Circle	Exact vs. Irrelevant	1.25	n/a	n/a	0.25
Rect0, Cross90	Relevant vs. Overly complex	n/a	0.35	2.05	n/a
Rect0, Circle	Relevant vs. Irrelevant	n/a	.60	n/a	2.00
Cross90, Circle	Overly complex vs. Irrelevant	n/a	n/a	1.25	1.15

The preference for recruiting exact and exact but overly complex knowledge sources can be understood in the same terms used to analyze single source experiment on rotation. Directly connected interpretations of these exact knowledge sources are often recruited because they predict error very well, and once recruited they require very little further modification.

5. Finding and Using Component Knowledge

In this paradigm, we tested whether KBCC can find and combine source knowledge of components to learn a new, more complex target problem comprised of these components, and whether use of these knowledge components speeds learning. The main component in these tasks is a 0.5 x 1.5 rectangle. The target task is a cross (Cross90) formed by two superposed rectangles (Rect0 and Rect90). The transformation used to create variation in knowledge relevance is rotation. All figures are centered at (0, 0). The various source knowledge conditions are shown in Table VII. Descriptions of the various source knowledge components were provided in Table V.

Table VII: Source Knowledge Conditions for Component Experiments

Name	Relation to target
Cross90	Exact
Rect0 & Rect90	Both components

Rect0	1st component
Rect90	2nd component
Cross45	Rotated 45 degrees
Rect45 & Rect135	Both components rotated
Rect45	1st rotated component
Rect135	2nd rotated component
Circle	Irrelevant
None	No knowledge

A factorial ANOVA of the epochs to victory yielded a main effect of knowledge condition, $F(9, 190) = 50$, $p < .0001$. The mean epochs to victory, with standard deviation bars and homogeneous subsets, based on the *LSD* post hoc comparison method, are shown in Figure 14. Exact knowledge produced the fastest learning, followed by knowledge of the two exact target components, knowledge of one of the two exact target components, all other sorts of knowledge, and finally the control condition without any knowledge.

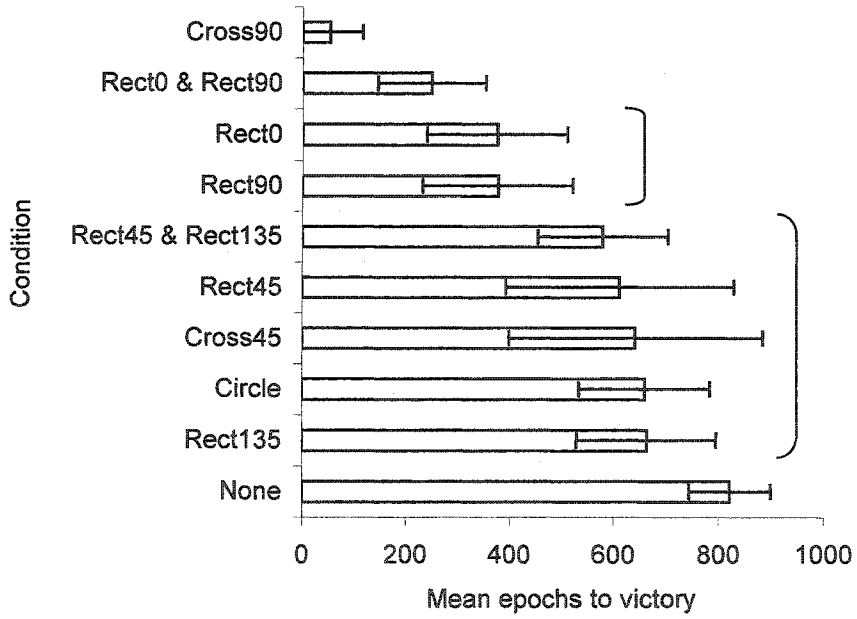


Figure 14. Mean epochs to victory in the target phase of the components experiment, with standard deviation bars and homogeneous subsets.

Number of hidden units recruited during the acquisition of source knowledge can be used as an index of complexity for single sources. These were subjected to a factorial ANOVA, yielding a main effect of knowledge condition, $F(6, 133) = 236$, $p < .0001$. The

mean number of hidden units recruited in the source acquisition phase, with standard deviation bars and homogeneous subsets, based on the Scheffe method of post hoc comparison, are shown in Figure 15. Single rectangles proved to be the simplest, followed by the circle and the cross at 90 degrees, and finally the cross at 45 degrees.

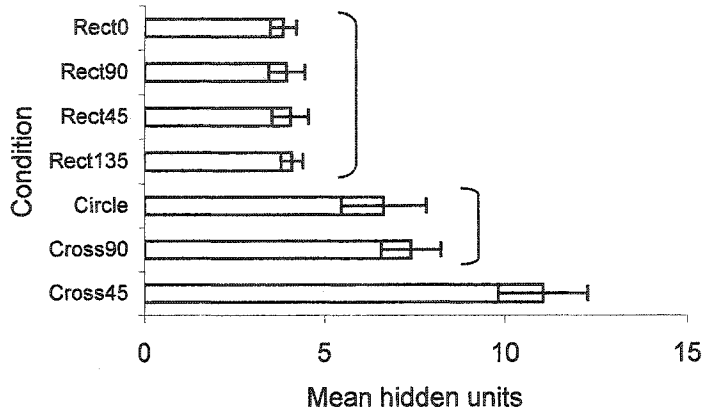


Figure 15. Mean hidden units recruited in the source-acquisition phase of the components experiment, with standard deviation bars and homogeneous subsets.

Output activation diagrams are plotted in Figure 16 for a network learning the cross target by recruiting its two basic components, the vertical and horizontal rectangles. Figures 15a and 15b show the horizontal and vertical component sources, respectively. Figures 15c, 15d, and 15e show the target network at the end of the first, second, and third output phases, respectively. As illustrated in Figure 15c, when learning the cross, the best solution without any hidden units is to classify all patterns as being outside of the target class because the target class contains only 57 of the 225 training patterns. The third output phase was the final output phase for this network. This network recruited the horizontal source rectangle first, followed by the vertical source rectangle, each in their directly connected versions.

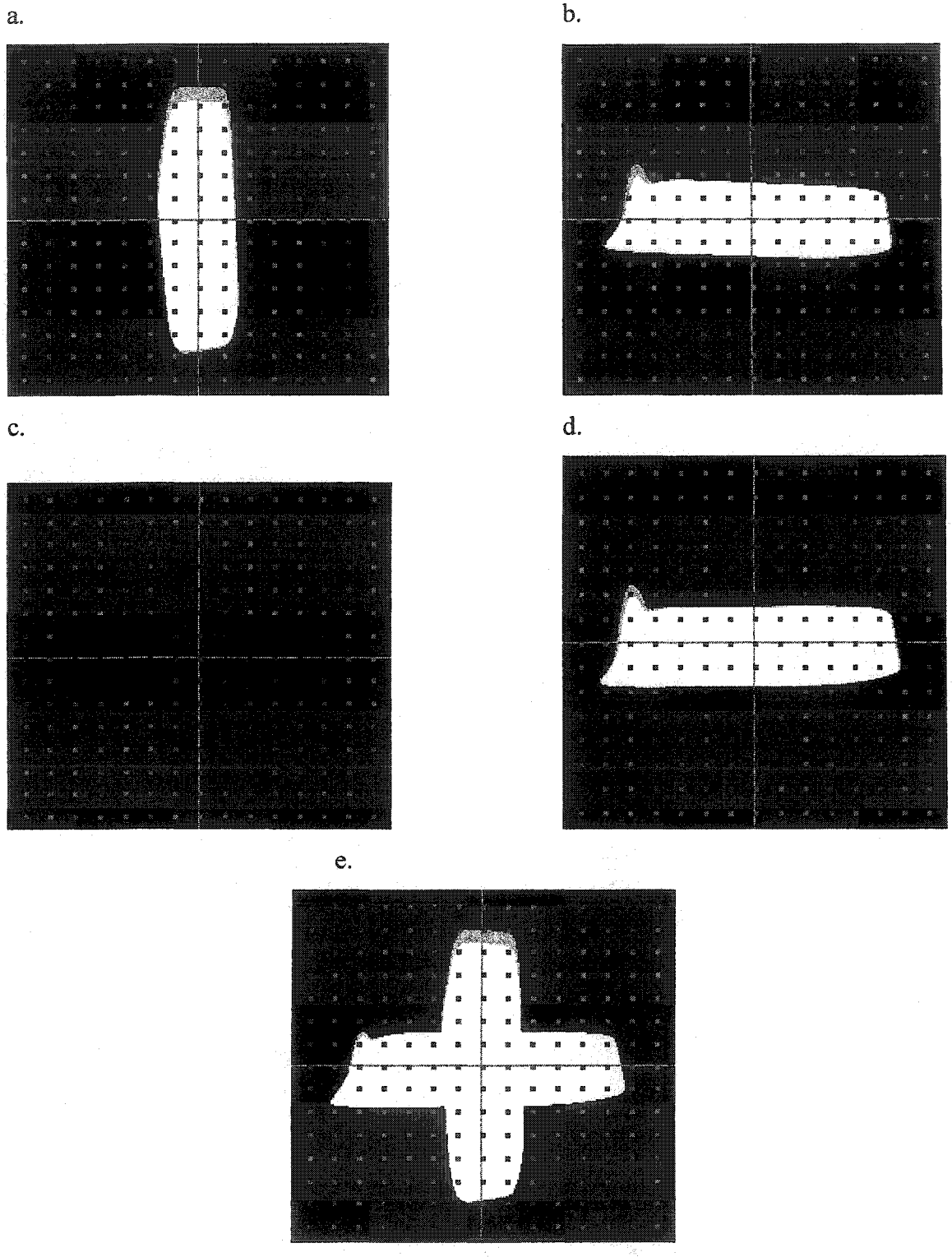


Figure 16. Output activation diagrams for a network learning a cross target by recruiting its two components (a and b). Target solutions are shown at the end of the first (c), second (d), and third (final, e) output phases.

More generally, 81% of the source networks recruited in this condition were directly connected versions. All of networks in the exact (un-rotated cross) knowledge condition recruited the directly connected source network, and 90% were finished after this single recruitment. In these recruitments (whether of independent source components or exact source knowledge), the direct input-side connection weights were around 1 for the x-axis and y-axis input units and around 0 elsewhere. Again, the shapes of the target solutions closely resembled the shapes of the recruited source knowledge. Networks in conditions with rotated source components did not, in general, exhibit such straightforward behavior as did networks with these un-rotated components.

6. Summary of Learning Speed Ups

To compare the amount of learning speed up in the various experiments, we computed several indices of speed up. For each experiment, we divided mean epochs in the no-knowledge or irrelevant-knowledge condition by mean epochs in either the exact-knowledge or best-inexact-knowledge condition. [6] Because the results were different for scaling down to a small rectangle and scaling up to a large rectangle in the sizing experiment, the indices were computed separately for the two versions of that experiment. These indices of learning speed up, shown in Figure 17, range from 1.34 in the irrelevant/best-inexact measure in the rotation experiment to 15.51 in the none/exact index in the components experiment. In general, knowledge speeds up learning substantially everywhere, but there is considerable variation in the size of these effects. Figure 17 shows tendencies for exact knowledge to be more beneficial than inexact knowledge and for irrelevant knowledge to be more beneficial than no knowledge at all. Across indices and experiments, the overall mean speed-up factor is 5.29.

[6] By "best" we mean fastest to learn.

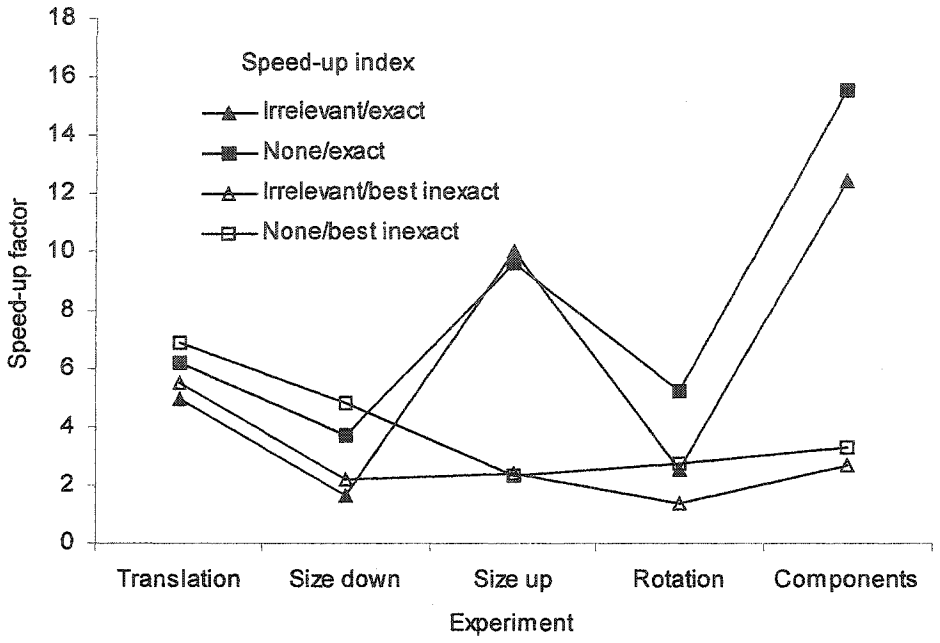


Figure 17. Speed-up factors in the various experiments in terms of mean epochs in the no-knowledge or irrelevant-knowledge condition divided by mean epochs in either the exact-knowledge or best-inexact-knowledge condition.

7. Generalization

Generalization tests with the 200 randomly determined test patterns are presented in Table VIII in terms of the mean and standard deviation of percent misclassification by KBCC networks in each experiment. The mean percent misclassification is 3%, and never exceeds 7% in any experiment, indicating good generalization performance.

Table VIII: Mean and Standard Deviation of Percent Misclassification Error on Test Problems

Experiment	Mean	SD
Rotation/single	.027	.009
Rotation/dual	.025	.010
Translation/single	.030	.013
Translation/ dual	.023	.012
Scale-up/single	.040	.012
Scale-down/single	.031	.015
Scale-up/dual	.035	.018
Scale-down/dual	.020	.011
Cross from components	.070	.018

8. Discussion

8.1 Overview of Results

The present results show that KBCC is able to find, adapt, and use its existing knowledge in the learning of new problems, significantly shortening the learning time. When exact knowledge is present, it is recruited for a quick solution. The more relevant the knowledge is, the more likely it will be recruited for solution of a new problem and the faster that new learning is likely to be. If KBCC knows the components of its new task, then it recruits and combines those components into a solution, again with a significant speed up in learning. These are the sorts of qualities one would expect in a system that effectively uses its knowledge in new learning.

With a single source of knowledge in memory, KBCC tends to learn fastest with knowledge that matches the target exactly, followed by exact but embedded knowledge, close and relevant knowledge, distant relevant knowledge, irrelevant knowledge, and no knowledge at all. When learning a multi-component target task, KBCC learns faster with knowledge of both components than knowledge of only one component. With multiple sources of knowledge, there is a tendency for KBCC to prefer to recruit sources in this same order, that is, to recruit the source that allows it to learn faster. Many of these results were traced to differential tendencies to recruit directly connected source networks. Such source networks are more likely to be recruited when the knowledge is exact or embedded, and is likely to speed learning.

Testing KBCC in the different domains of translation, sizing, and rotation provided evidence on the generality of our conclusions. Although these domains differ in their overall difficulty and in some aspects of the findings, there was considerable generality in the results. For example, in learning a small rectangle, recruiting a source rectangle sharpened the critical corners so that learning was fast and distance of relevant knowledge was irrelevant to learning speed. However, in the more difficult problem of scaling up to a larger rectangle, critical corners were smoothed by recruitment and had to be relearned, thus increasing learning time in relation to distance of the source

knowledge. Thus, the patterns of results relating knowledge relevance to learning speed may be dampened or enhanced by various manipulations, but they are rarely reversed.

8.2 A Note on Irrelevant Source Knowledge

We had termed learning a circle as an *irrelevant* source knowledge because a circle lacks the critically important corners possessed by target and source rectangles. In single-source experiments, KBCC networks recruiting circular source knowledge were generally slower to learn rectangular targets than those recruiting rectangular source knowledge. However, recruitment of circular source knowledge was typically favored over, and was faster than, recruitment of single hidden units. In dual-source experiments, recruitment of circular source knowledge was less preferred than recruitment of exact knowledge, but sometimes more preferred (in translation and rotation experiments) and sometimes less preferred (in sizing experiments) than relevant knowledge. It is interesting to speculate about the utility of supposedly irrelevant circular source knowledge in some of these comparisons.

Single hidden units carve up a problem space with uniform hyper-planes, whereas candidate networks built on circular concepts employ a more complex geometry that may be similar to that of the target concept, thus raising correlation during recruitment phases and possibly lowering error during output phases. For example, both a rectangle and a circle separate inside patterns from outside patterns. When a different-shaped source is in the same region as the target, it may become a desirable object to recruit because it can correlate quite highly with target error. Such high correlations may lead to recruitment but the need to sharpen corners to meet the additional requirements of rectangular targets may prolong learning, leading to even more recruitments (cf. the multiple numbers of circles recruited in some conditions in Tables II, IV, and VI).

8.3 Relation to Previous Work

As noted earlier, the present work on KBCC bears some relation to previous neural network research on knowledge transfer, multitask learning, sequential learning, lifelong learning, input re-coding, knowledge insertion, and modularity.

Pratt (1993) developed a technique called discriminability-based transfer that uses the weights from a previously trained network to initialize a new network. This is

probably the most obvious and straightforward idea for using knowledge in new learning. However, because it did not work as well as expected, Pratt improved the technique by re-scaling the previous network's hyper-planes so that useful ones had large weights and less useful ones had small weights.

Caruana (1993, 1997) pioneered Multitask Learning (MTL) in which he trained a network on several tasks taken from the same domain in parallel, with a single output for each task. Such networks typically learned a common hidden-unit representation, which then proved useful for learning subsequent tasks from the same domain. Baxter (1995) proved that the number of examples required for learning any one task in an MTL paradigm decreases as a function of total number of tasks learned in parallel.

Silver and Mercer (1996) developed a method of sequential learning called task rehearsal. Here, old tasks are pseudo-rehearsed during new learning, generating patterns that can be added to the those of the target task. In pseudo-rehearsal, the network generates its own target vectors, using its current weights, rather than merely accepting them from the environment (Robins 1995). Separate learning rates for each task are used to control the impact of each source task, ensuring that the most related tasks have the most impact on learning.

Thrun and Mitchell (1993) engineered a technique they called lifelong learning, in which a network meta-learns the slope of the desired function at each training example. This is essentially the derivative of the function at an example output with respect to the input attribute vector. Then, in new learning, a meta-network makes slope predictions and estimates its accuracy for each new training example. This technique seems to rely not so much on knowledge representations as on search knowledge.

Clark and Thornton (1997) discussed the importance of networks being able to re-code their input in learning difficult, so-called Type-2 problems. Type-1 problems are those that can be solved by sampling the originally coded input data. In contrast, Type-2 problems need re-coding in order to use Type-1 knowledge. Ability to do this would require some degree of incremental learning, modularity, and perhaps representational re-description (Karmiloff-Smith 1992), but no specific algorithm was proposed.

Shavlik (1994) presented the KBANN algorithm for creating knowledge-based artificial neural networks. KBANN converts a set of symbolic rules embodying domain

knowledge of a problem into a feed-forward neural network with the final rule conclusions as output units and intermediate rule conclusions as hidden units. Connection and bias weights are initialized to implement the conjunctive and disjunctive structures of the rules. Networks thus initialized with knowledge are then trained with examples to refine the knowledge. Training is typically faster than with standard networks with random weights and leads to better generalization. Following the training, symbolic rules can be extracted from the network.

Jordon and Jacobs (1994) proposed the Hierarchical Mixture of Experts (HME) architecture to decompose a problem into network modules. Distinct network modules become expert on subtasks, and cooperate on an overall solution using gating networks that learn to weight the modular expert contributions for the different parts of the problem. HME was found to learn the dynamics of a four-degree-of-freedom robot arm much faster than a multi-layer back-propagation network did.

8.4 Advantages of KBCC

In contrast to these previous methods for using knowledge in learning, KBCC uses established techniques from generative learning algorithms (Fahlman & Lebiere, 1990). KBCC recruits existing networks as well as single units as it needs them in order to increase its computational power in the service of error reduction. Treating its existing networks like untrained single units, KBCC trains weights to the inputs of existing source networks to determine whether their outputs correlate with the target network's error. In addition, KBCC trains the output weights from a recruited network in order to incorporate it into a solution of the current problem. This process of adapting old knowledge to new task allows for further recruitment of either additional networks or single units. Indeed, the same network could be recruited more than once if this proved to be desirable for learning the target task. This adaptation of the outputs of the recruited source network allows KBCC to use knowledge that is only partly relevant to the new task. The ability to adjust both incoming and outgoing weights with respect to a source network gives KBCC considerable flexibility in its use of knowledge. The fact that units and sub-networks are installed in a cascade allows KBCC to build its new learning on top of any recruited knowledge.

KBCC may be one way of reducing a complex problem to a simpler, already known problem, as recommended by Clark and Thornton (1997). When a network is recruited, this effectively reinterprets a target problem as if it were an instance of a known problem. Further recruitments and output weight adjustments then craft this reinterpretation into a solution to the target problem.

During input phases, KBCC searches for the best linear transformation of the target network's inputs in relation to its source networks. This enables a potentially large range of input re-coding schemes. All of the linear transformations that were tried in the present simulations (translations, size-scaling, and rotation) produced satisfactory results in the sense of faster learning.

A direct comparison on translation problems showed that KBCC was considerably more effective than MTL in terms of speeding up learning (Shultz & Rivest, 2000b). In contrast to KBCC networks, MTL networks did not show any benefits of knowledge in terms of increased learning speed. MTL networks had particular difficulty extracting exact knowledge from an overly complex source network. Moreover, they often failed to learn their assigned source problem and thus had to be replaced before proceeding to the target phase. The primary reason that MTL does not speed the learning of new tasks is that it requires both old and new tasks to be freshly learned in parallel. In contrast, KBCC recruits its old knowledge without having to relearn it.

Unlike many of the previous techniques for which both the inputs and outputs of the source and target task must match precisely, KBCC can potentially recruit any sort of function to use in a new task. Source network inputs and outputs can be arranged in different orders, employ different coding methods, and exist in different numbers than those in the target network. Indeed, the only real constraint on what can be recruited by KBCC is that it must be possible to find the first derivative for a recruitment object. This need for the first derivative is due to the use of the quickprop algorithm for weight adjustment. If numerical estimation or an optimization algorithm that did not require first derivatives were used, even this restriction would disappear. This extreme flexibility means that functions created by means other than KBCC itself could be recruited. The wide range of recruitment objects would appear to offer considerably more power and flexibility than most knowledge-based learners provide.

When a source network is recruited by KBCC, it is thereafter treated as a black box module. Like the modules discussed by Fodor (1983), KBCC's recruited networks are computationally encapsulated sub-systems that interact with the rest of the system only through their inputs and outputs. Although Fodor (1983) proposed that such modules are innate and operate only in particular specialized areas such as perception and language processing, it is now recognized that modules can be learned and also operate within central cognition (Karmiloff-Smith 1992).

In contrast to larger and more homogeneous networks, modular neural networks restrict complexity to be proportional to problem size, easily incorporate prior knowledge, generalize effectively, learn multiple tasks, perform robustly, and are easily extended or modified (Gallinari 1995). The solutions of modular networks should also be easier to analyze than the solutions of homogeneous networks. Whatever recruited KBCC modules do with their input would not change from the time of their initial acquisition, although it still might be challenging to determine their role in the overall solution reached by the target network. Unlike the HME approach to modularity, the sub-networks in KBCC are gradually constructed through automatic learning rather than being designed ahead of time and being simultaneously present throughout the whole of learning.

KBCC also implements a natural resistance to the retroactive interference that often plagues sequential learning in neural networks (French 1992). Because each source network is an unchanged module, it never loses its original functionality, no matter how many times and ways that it is recruited. There is also no need to relearn old tasks while learning new ones as in Silver and Mercer's (1996) task rehearsal method and in Caruana's (1993, 1997) MTL.

KBCC allows for a combination of learning by either analogy and/or induction. KBCC learns by analogy to its current knowledge whenever it can and switches to a more inductive mode if it needs to. Recruiting a network is learning by analogy, whereas recruiting a single unit is learning by induction. Both processes are seamlessly integrated in KBCC's approach to a new target task.

8.5 Future Work

In this paper, we assessed the speed up in learning that comes from recruiting existing relevant knowledge in the KBCC algorithm. KBCC should also be assessed for the possibility that it could learn a problem more deeply and generalize more effectively by virtue of recruiting such knowledge. Deeper learning and more effective generalization was not apparent in the current work because learning was allowed to proceed to completion in every condition.[7] Even without any stored knowledge, KBCC, which is then essentially equivalent to ordinary CC, is powerful enough to learn these non-linear problems by recruiting individual hidden units as needed. Assessing the impact of knowledge on the quality of learning would require impoverished training sets and/or assessments earlier in learning. This issue is the focus of work that is currently underway in our laboratory.

KBCC has so far been applied to only toy demonstration problems, albeit problems that might pose some difficulty for other learning algorithms. Use of these well understood and easy to visualize problems enables us to explore the properties of KBCC in some detail and with growing confidence that the algorithm is working appropriately. Nonetheless, it would be interesting and important to try KBCC on real and even more difficult problems and with a larger and more realistic array of source networks. A number of realistic problems have already been the focus of work on knowledge transfer in neural networks. These include problems in speech recognition, medical diagnosis, DNA pattern discovery, and chess (Pratt 1993; Silver & Mercer, 1996). In addition to exploring such realistic problems, we also plan to apply KBCC to the simulation of psychological data on the use of knowledge in learning.

A significant difficulty could be anticipated as a KBCC system accumulates extensive experience. The problem is that searching an extensive knowledge base of source networks in input (recruitment) phases would become prohibitively expensive computationally. It seems reasonable in such circumstances to focus that search on source

---

[7] This is not to say that KBCC does not generalize well. The generalization results in Table VIII show that mean misclassification error on test problems was only 3%. The point made here is that there is not yet any demonstration that recruiting existing knowledge by KBCC networks improves depth of learning, as indexed by superior generalization.

networks that could be expected to be particularly useful. Focusing these recruitment searches could perhaps be accomplished with weight-implemented heuristics such as similarity in inputs and outputs, recency of learning, and externally provided hints about what existing knowledge might be useful. In addition and perhaps more importantly, past recruitment of particular source networks for particular problems might be used to develop a task semantics that could further constrain these searches.

The current version of KBCC is able to find and use its existing knowledge in learning new tasks. This holds the promise of being able to undertake more realistic implementations of the kind of knowledge-based learning at which people excel.

Author Note

This research was supported by a grant from the Natural Sciences and Engineering Research Council of Canada. This work has benefited from comments from David Buckingham, Jacques Katz, Sylvain Sirois, Yoshio Takane, and two anonymous reviewers.

References

Baxter, J. (1995) Learning internal representations. Proceedings of the Eighth International Conference on Computational Learning Theory. Santa Cruz, CA: ACM Press.

Buckingham, D. & Shultz, T. R. (1994) A connectionist model of the development of velocity, time, and distance concepts. Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

Caruana, R. (1993) Multitask learning: A knowledge-based source of inductive bias. Proceedings of the Tenth International Machine Learning Conference. San Mateo, CA: Morgan Kaufmann.

Caruana, R. (1997) Multitask learning. Machine Learning, 28, 41-75.

Clark, A. & Thornton, C. (1997) Trading spaces: Computation, representation, and the limits of uninformed learning. Behavioral and Brain Sciences, 20, 57-97.

Fahlman, S. E. (1988) Faster-learning variations on back-propagation: An empirical study. In D. S. Touretzky, G. E. Hinton, & T. J. Sejnowski (Eds.), Proceedings of the 1988 Connectionist Models Summer School. Los Altos, CA: Morgan Kaufmann.

Fahlman, S. E. & Lebiere, C. (1990) The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), Advances in Neural Information Processing Systems 2. Los Altos, CA: Morgan Kaufmann.

Fodor, J. A. (1983) The Modularity of Mind. Cambridge, MA: MIT Press.

French, R. (1992) Semi-distributed representations and catastrophic forgetting in connectionist networks. Connection Science, 4, 365-377.

Gallinari, P. (1995) Training of modular neural net systems. In M. A. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks. Cambridge, MA: MIT Press.

Heit, E. (1994) Models of the effects of prior knowledge on category learning. Journal of Experimental Psychology: Learning, Memory, and Cognition, 20, 1264-1282.

Jordan, M. I. & Jacobs, R. A. (1994) Hierarchical mixtures of experts and the EM algorithm. Neural Computation, 6, 181-214.

Karmiloff-Smith, A. (1992) Beyond modularity: A Developmental Perspective on Cognitive Science. Cambridge, MA: MIT Press.

Keil, F. C. (1987) Conceptual development and category structure. In U. Neisser (Ed.), Concepts and Conceptual Development: Ecological and Intellectual Factors in Categorization. Cambridge: Cambridge University Press.

Mareschal, D. & Shultz, T. R. (1999) Development of children's seriation: A connectionist approach. Connection Science, 11, 149-186.

Murphy, G. L. (1993) A rational theory of concepts. The Psychology of Learning and Motivation, 29, 327-359.

Nakamura, G. (1985) Knowledge-based classification of ill-defined categories. Memory and Cognition, 13, 377-384.

Pazzani, M. J. (1991) Influence of prior knowledge on concept acquisition: Experimental and computational results. Journal of Experimental Psychology: Learning, Memory, and Cognition, 17, 416-432.

Pratt, L. Y. (1993) Discriminability-based transfer between neural networks. Advances in Neural Information Processing Systems 5. San Mateo, CA: Morgan Kaufmann.

Robins, A. V. (1995) Catastrophic forgetting, rehearsal, and pseudorehearsal. Connection Science, 7, 123-146.

Shavlik, J. W. (1994) A framework for combining symbolic and neural learning. Machine Learning, 14, 321-331.

Shultz, T. R. (1998) A computational analysis of conservation. Developmental Science, 1, 103-126.

Shultz, T. R., Buckingham, D., & Oshima-Takane, Y. (1994) A connectionist model of the learning of personal pronouns in English. In S. J. Hanson, T. Petsche, M. Kearns, & R. L. Rivest (Eds.), Computational Learning Theory and Natural Learning Systems, Vol. 2: Intersection between Theory and Experiment. Cambridge, MA: MIT Press.

Shultz, T. R., Mareschal, D., & Schmidt, W. C. (1994) Modeling cognitive development on balance scale phenomena. Machine Learning, 16, 57-86.

Shultz, T. R., & Rivest, F. (2000a) Knowledge-based cascade-correlation. Proceedings of the International Joint Conference on Neural Networks. Los Alamitos, CA: IEEE Computer Society Press.

Shultz, T. R., & Rivest, F. (2000b) Using knowledge to speed learning: A comparison of knowledge-based cascade-correlation and multi-task learning. Proceedings of the

Seventeenth International Conference on Machine Learning. San Francisco: Morgan Kaufmann.

Silver, D. & Mercer, R. (1996) The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. Connection Science, 8, 277-294.

Sirois, S. & Shultz, T. R. (1998) Neural network modeling of developmental effects in discrimination shifts. Journal of Experimental Child Psychology, 71, 235-274.

Thrun, S. & Mitchell, T. (1993) Integrating inductive neural network learning and explanation-based learning. In R. Bajcsy (Ed.), Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. San Mateo, CA: Morgan Kaufmann.

Wisniewski, E. J. (1995) Prior knowledge and functionally relevant features in concept learning. Journal of Experimental Psychology: Learning, Memory, and Cognition, 21, 449-468.

CONNECTING TEXT

The preceding manuscript introduced a new algorithm for knowledge transfer called KBCC. KBCC was also evaluated for its ability to find and use relevant knowledge to speed learning. The problems on which it was tested were only toy problems and a major issue is whether KBCC can also work well on larger, more realistic problems. The next manuscript attempts to answer this question by using KBCC on a vowel recognition problem for which results with neural networks and some other knowledge transfer algorithms are already known. The manuscript also presents some improvements to KBCC.

MANUSCRIPT 2

Application of Knowledge-based Cascade-correlation to Vowel Recognition

François Rivest and Thomas R. Shultz
Laboratory for Natural and Simulated Cognition
School of Computer Science and Department of Psychology
McGill University

Rivest F. & Shultz, T.R. (2002) Application of Knowledge-based Cascade-correlation to Vowel Recognition. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, pp. 53-58.

Abstract

Neural network algorithms are usually limited in their ability to use prior knowledge automatically. A recent algorithm, knowledge-based cascade-correlation (KBCC), extends cascade-correlation by evaluating and recruiting previously learned networks in its architecture. In this paper, we describe KBCC and illustrate its performance on the problem of recognizing vowels.

I Existing Knowledge and New Learning

Neural network algorithms rarely allow prior knowledge to be incorporated into their learning. Most start from scratch and those that do use prior knowledge require that knowledge to have a specific form, such as having the same architecture [1], being a symbolic domain theory [2], or being given as hints [3]. However, prior knowledge can often take the form of some existing classifier or function approximator and no algorithm is flexible enough to permit the integration of such a wide variety of knowledge.

It is clear that humans do not learn from scratch, but make extensive use of their knowledge in learning [4-6]. Use of prior knowledge in learning can ease and speed learning and lead to better generalization as well as interference effects. The current difficulty in using prior knowledge is arguably the major limitation in neural network modeling of human learning and cognition. In this paper, we describe and test a neural learning algorithm that implements a general mechanism of knowledge reuse.

Knowledge-based cascade-correlation (KBCC) is a fundamental extension of cascade-correlation (CC), a constructive learning algorithm that has been successfully used in many real applications [7] and in simulations of cognitive development [8-13]. CC builds its own network topology by adding new hidden units to a feedforward network in cascade fashion, i.e., new units receive inputs from each non-output unit already in the network [14]. Our KBCC extension recruits previously learned networks in addition to the untrained hidden units recruited by CC. These recruitable networks could potentially be any functional form knowledge, although being differentiable is a must. We refer to existing networks as *source* knowledge and to the current task to learn as a *target*. Previously learned source classifiers or approximators compete with each other and with standard hidden units to be recruited into the learning network.

In artificial bivariate dichotomous tasks, KBCC successfully recruited networks representing parts of a target task, equivalent-knowledge networks, and more complex networks embedding equivalent knowledge, with substantial learning speed ups [15]. KBCC was also shown to be superior to multi-task learning (MTL) in these respects [16].

II Previous Work on Knowledge and Learning

KBCC is similar to recent neural network research on transfer [1], sequential learning through multi-task learning [17], and knowledge insertion [2,18]. But KBCC is more ambitious and principled because it stores and searches for knowledge within a generative network approach and has no real limitation in the structure of the recruited knowledge.

Pratt [1] studied the idea of transferring knowledge from a source neural network to a target network through copying the network structures and parameters (weights). She found that literally copying a network could sometimes slow down the training and reduce generalization performance compared to random networks. She therefore developed a technique to re-scale the weight vector feeding hidden units. If a hidden unit has good discrimination power, its weight vector is scaled up to reduce training effects, and conversely, if the discrimination hyperplane is bad, its weight vector is scaled down, or even randomized, in order to avoid copying bad effects. This technique is limited to discrete output networks where the target task requires a network at least as big as the source network and where input and output perfectly matches the source network.

Silver and Mercer [17] developed a transfer of knowledge technique based on Caruana's multi-task learning [19]. The basic idea derives from a proof that if a network has multiple related tasks to learn, it requires fewer examples to learn them, because the hidden layer can develop a more general representation. Silver and Mercer's idea is to re-learn the prior knowledge while learning the new task, in parallel, on the same network. The target network has an output for the target task, and extra outputs to represent each source network's outputs. Prior knowledge is used to generate the desired values for these extra outputs to learn. This can be simply done by processing the input patterns through the prior knowledge, thus permitting the prior knowledge to be any sort of function. This still has a major limitation in that target inputs must match source inputs, and the new network must be big enough to learn the prior knowledge. Moreover, relearning of prior knowledge is required, which does not seem very efficient.

With a slightly different goal in mind, Towell and Shavlik [2] invented an algorithm to transform rule-based knowledge into a neural network (KBANN). The idea was to refine that knowledge in neural network form and then to later extract improved

rules. We believe that this technique could be used with KBCC, by taking rule-based knowledge and transforming it into differentiable functional form.

This kind of idea was also developed by Parekh and Honavar [18], who proposed to use KBANN in conjunction with constructive algorithms. KBANN was used to create a neural network that would serve as a basis for a constructive algorithm that could build on the source knowledge outputs and inputs. Again, this requires the same encoding for prior and new knowledge. Moreover, it does not allow composition of prior knowledge like most other approaches.

III Description of KBCC

Because KBCC is a generalization of CC, it is quite similar to CC. As in CC, candidates are installed on top of the network, just below the output; hence new units receive inputs from every non-output unit already in the network. Unlike CC, KBCC is not limited to a pool of candidate units that are univariate single-valued functions. KBCC can recruit any multivariate vector-valued component. The connection scheme in KBCC as shown below is similar to the CC connection scheme, except that a hidden unit may have a matrix of weight connections (as opposed to a single vector) at their inputs and their outputs as shown in figure 1.

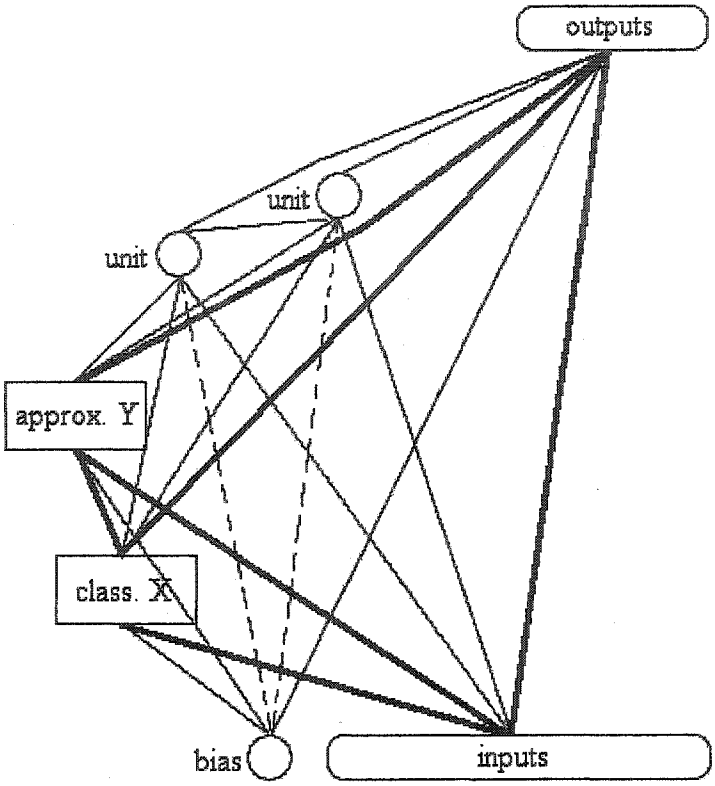


Figure 1: A KBCC network with four hidden units. The first one is an existing classifier, the second one is an existing approximator, and the last two are single sigmoid units. Dash lines represent single weights, while solid thin lines represent weight vectors, and solid thick lines weight matrices.

KBCC training is composed of two phases: In output phase, only the weights feeding the output units are trained. In input phase, only the weights feeding the candidate units (and networks) are trained.

The network begins in output phase with a set of output units fully connected to the inputs. These weights are trained to minimize the sum squared error:

$$F = \sum_{o} \sum_{p} \left(V_{o,p} - T_{o,p} \right)^2 \qquad (1)$$

Where $V_{o,p}$ is the network output o at pattern p and $T_{o,p}$ the corresponding target value. The training uses QuickProp[1] [14], a gradient based algorithm that employs the current and previous gradient to estimate the second order derivative of the objective

[1] Although training is not limited to QuickProp.

function with respect to the weights to be trained. The output phase stops either when it successfully learns the task, or when the sum squared error stagnates or a maximum number of epochs is reached, in which case the algorithm goes into input phase.

The input phase begins by initializing a pool of candidate units and networks (or other functional knowledge) with random weights from every non-output unit of the target network to the candidate inputs. These weights are then trained using QuickProp to maximize the covariance between the candidate outputs and the target network residual error:

$$G_c = \frac{\sum_{o_c}\sum_{o}\left\|Cov(V_c, E)\right\|^F}{\sum_{o}\sum_{p}E_{o,p}^2} \qquad (2)$$

Where $E_{o,p}$ is the error at output unit o for pattern p, V_c is the candidate output patterns, E the network error patterns and $\|C\|^F$ the Frobenius norm of matrix $C=Cov(V_c,E)$ defined as:

$$\|C\|^F = \sum_{i,j} C_{i,j}^2 \qquad (3)$$

Again, whenever the best score $max\{G_c\}$ stagnates, or a maximum number of epochs is reached after a minimal score, the input phase stops and the best candidate is installed into the target network by adding connections from its outputs to the target network outputs using small random values and the sign of the covariance. The other candidates are discarded.

A more detailed description of the KBCC algorithm with all the default parameter values can be found in [20].

IV Demonstration of KBCC: Peterson-Barney Vowel Recognition

We created six transfer scenarios with the Peterson-Barney vowel recognition problem from the CMU AI repository.[2] The data set can be split into three subsets based on the speaker type: male, female or child. One scenario was originally used by Pratt [1] and involved training networks on the female data and then using them as sources to train

[2] http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/speech/database/pb/pb.tgz

target networks on male data. The other scenarios are similar and complete all permutations of the three subsets.

The data set contains the two middle formants of the speech sound made by 76 speakers saying 10 different vowels twice. The speakers were 33 males, 28 females, and 15 children, all speaking English. The inputs were scaled down by 1000 resulting in input values in the range [0.0, 1.5] and [0.0, 4.0] for the first and second formants, respectively. The outputs were encoded on 10 units (one per vowel) with a value of 1.0 for the correct vowel and 0.0 for the others. A network was considered to properly classify a pattern if the output with highest activation corresponded to the target vowel.

The scenarios are constructed using the following scheme. Starting with the three subsets (male, female, child), one subset is used to train the source networks, and a different subset for training the target networks. This scheme generates six scenarios. In order to compare KBCC with CC without knowledge, we added three more scenarios where we trained CC nets on one of the subsets without any prior knowledge.

A. Experimental Setup

First, for each subset, we generated 10-fold cross-validation train/test set pairs. We trained 10 CC networks (for up to 15 hidden units) for each train/test pair for a total of 100 CC networks per subset. Those represent the three no-knowledge baseline scenarios.

We found that a good CC source network (similar to Pratt's sources) has about 10 hidden units. Since each subset is used as source in two scenarios, we have trained 200 CC networks on each of the three subsets for 10 recruitments each.

For each scenario, we have 100 CC networks per source data set. Given a scenario, for each of the 10 train/test pairs of the target data set, we trained 10 KBCC networks (for up to 15 hidden units/networks). For each of the 100 resulting KBCC networks we used a different source network.

During the training of the target networks, we evaluated the network on the data subset of their source to measure retention and on the third data subset, the one that wasn't used in their training nor in the training of their sources. For example, in the

scenario where the source networks are trained on female data and the target networks are trained on male data, the third subset is the child data.

B. Early Learning Comparison

In one of our scenarios, similar to Pratt, we used the female data (560 patterns) to train the source networks. We first found that good sources had about 10 hidden units, so we trained 100 CC sources with 10 hidden units each. We then tested these female-trained sources on the male data and obtained 52%±3% accuracy. This is quite close to Pratt's result using static back propagation networks.

Then we generated 10-fold cross-validation train/test sets. For each of the 10 train/test set pairs we trained 10 CC networks and 10 KBCC networks (each KBCC network used a different source) for up to 15 recruitments. We computed the train and test percentage correct at every epoch and analyzed the resulting learning curves.

Before the first recruitment, the linear solution scored around 71% correct on the train set and 69% on the test set. Before the second recruitment (which happened sooner in KBCC than in CC), KBCC reached 86% and 84% on the train and test sets while CC had only reached 80% and 77% on these same sets. Moreover the peak generalization of the KBCC-averaged test curves (85.5%) is reached at epoch 438 while that peak on the CC-averaged test curves (85.8%)[3] is reached at epoch 1699. Finally, we looked at the average number of epochs for each network curve to reach its peak generalization. We computed a paired sample t-test using the average for each fold. KBCC was significantly faster, taking an average of 827 epochs to reach its peak generalization while CC took 1279 epochs, with $t(9) = 4.418$ and $p < 0.005$. Both average peak values were 89% correct. Results are plotted in figure 2, also showing that the effect of the first recruitment is even stronger with child sources.

[3] These two peaks are not significantly different in percent correct.

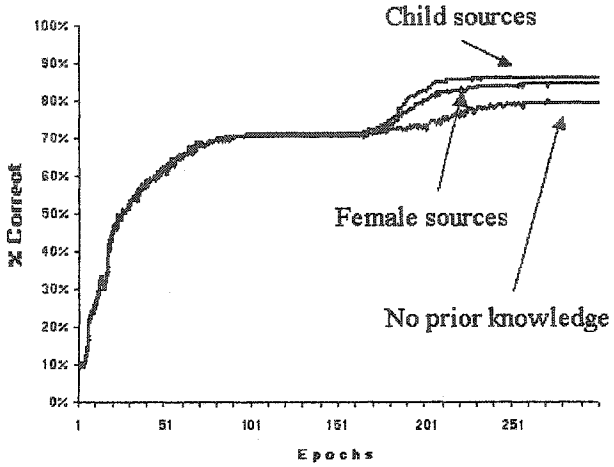


Figure 2: Averaged learning curves for the KBCC networks learning the male data. The curve reaching the lowest point is averaged over networks without prior knowledge. The one in the middle is averaged over networks using source networks trained on female data and the highest one represents networks using sources trained on the child dataset.

Similar results were obtained in the other scenarios. Each row in tables 1 and 2 represents one target situation. For each of these, the proportion correct after the first recruitment is given for every source condition. In all cases, the existence of prior knowledge shows a clear advantage in proportion correct after the first recruitment.

Table 1: Train Proportion Correct After the First Recruitment

Target	Source			
	None	Male	Female	Child
Male	80%	N/A	86%	84%
Female	81%	86%	N/A	87%
Child	76%	81%	84%	N/A

Table 2: Test Proportion Correct After the First Recruitment

Target	Source			
	None	Male	Female	Child
Male	78%	N/A	84%	82%
Female	78%	83%	N/A	83%
Child	71%	76%	80%	N/A

C. Learning Time Comparison

To evaluate the learning time, we simulated early-stopping after training. Since during training we recorded train and test set proportion correct, we could reconstruct for every target network the number of epochs it took to reach its highest generalization peak before over training. Given a target subset, we compared the learning speed of the three conditions (for example, given male data as target subset, the three conditions are without knowledge, using knowledge of female data and using knowledge of chid data).

Figures 4- 6 show the results grouped by target task. For each of those figures, we ran an ANOVA and looked at the Scheffe post hoc test. For all three targets, the prior knowledge conditions were significantly faster than the no knowledge condition at the .05 level.

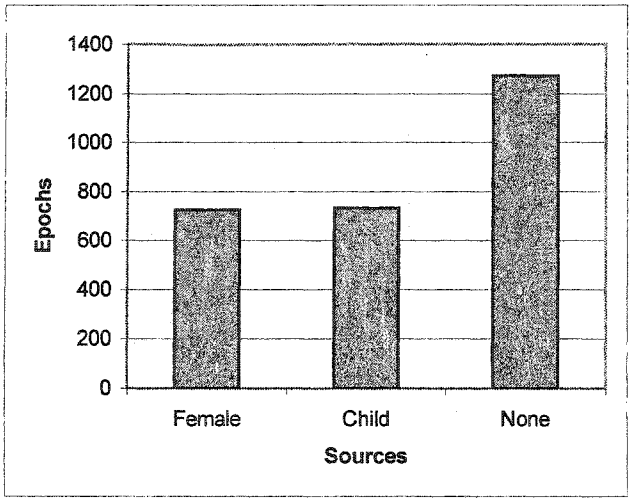


Figure 4: Mean number of epochs to learn male data in three different source conditions.

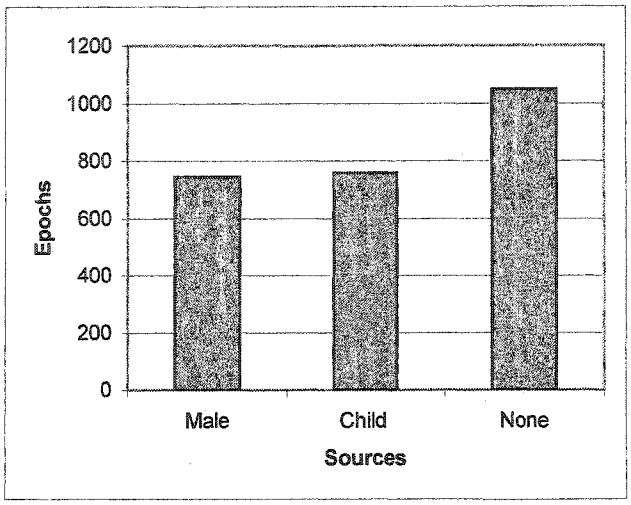


Figure 5: Mean number of epochs to learn female data in three different source conditions.

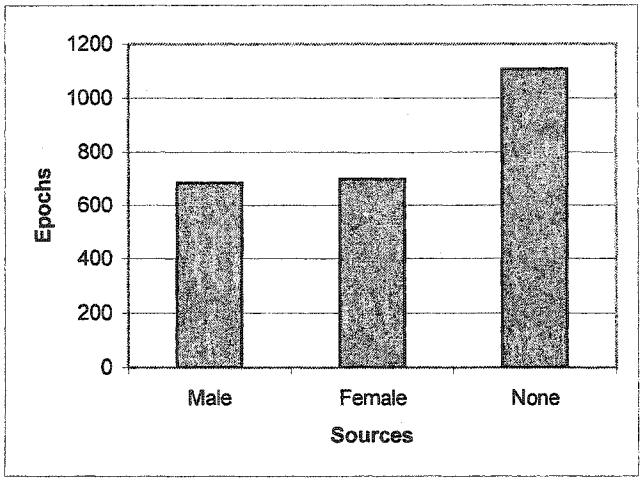


Figure 6: Mean number of epochs to learn child data in three different source conditions.

D. Learning Quality

We also compared the best train and test percent correct reached by our networks. Results are presented in Table 3. We did an ANOVA to compare the three source conditions under each target task separately. None of the three ANOVAs yielded significance at the .05 level on the Scheffe test. Hence, the quality of the final solution did not seem to be affected by prior knowledge.

Table 3: Test Proportion Correct at Highest Generalization

Target	Source			
	None	Male	Female	Child
Male	89%	N/A	88%	88%
Female	89%	89%	N/A	89%
Child	84%	85%	85%	N/A

E. Retention and 3rd Set Generalization

We compared the retention and third set generalization of the source knowledge conditions for each target task. Even though in few cases, the prior knowledge condition had a slightly significant advantage, in others it had a slight disadvantage. In most cases there was little difference.

V Discussion

These results show that KBCC is able to adapt and use its prior, related knowledge in the learning of a large and realistic new problem. Moreover, the availability of relevant knowledge significantly shortens KBCC learning time, without any loss of accuracy. Effective use of prior knowledge in new learning is the sort of quality one would like from both engineering and cognitive modeling viewpoints.

In contrast to previous methods for using knowledge in learning, KBCC has almost no restrictions on the format of prior knowledge. First, because prior knowledge is recruited into the network topology instead of being relearned, there is basically no limit to the internal complexity of the sources. Second, KBCC automatically searches for the best way to connect recruited sources in its architecture, removing any necessity for source inputs and outputs to perfectly match those of the target task.

Moreover, KBCC can use one or multiple sources to build a compositional solution. Because every candidate receives input from every previously recruited module, KBCC can combine them in a compositional way, for example, processing input data first through some classifier and then through some approximator (as shown in figure 1).

KBCC also seamlessly integrates learning by analogy with learning by induction. It learns by induction whenever it recruits single hidden units and by analogy when it recruits a previously trained source network. Finally, KBCC is consistent with the CC

algorithm that has been successful in solving many real problems and in simulating many aspects of cognitive development.

Application of KBCC to other real problems such as DNA junction-splicing is currently being studied in our laboratory. Another area under study is the effect of prior knowledge on KBCC in impoverished training environments.

B. Acknowledgments

This work was supported by a grant from the Centre de Recherche Informatiqe de Montréal to the first author and from the Fonds de la Formation de Chercheurs et l'Aide à la Recherche to each author. We are grateful for comments on this work from Doina Precup.

C. References

[1] L. Y. Pratt, "Discriminality-based transfer between neural networks," *Advances in neural information processing systems 5*, pp. 204-211. San Mateo, CA: Morgan Kaufmann, 1993.

[2] G. G. Towell and J. W. Shavlik, "Knowledge-based artificial neural networks," *Artificial Intelligence* **70**:119-165, 1994.

[3] Y. S. Abu-Mostafa, "A method for learning from hints," *Advances in Neural Information Processing Systems 5*, pp. 73-80. San Mateo, CA: Morgan Kaufmann, 1993.

[4] E. Heit, "Models of the effects of prior knowledge on category learning," *Journal of Experimental Psychology: Learning, Memory, and Cognition* **20**:1264-1282, 1994.

[5] M. J. Pazzani, "Influence of prior knowledge on concept acquisition: Experimental and computational results," *Journal of Experimental Psychology: Learning, Memory, and Cognition* **17**:416-432, 1991.

[6] E. J. Wisniewski, "Prior knowledge and functionally relevant features in concept learning," *Journal of Experimental Psychology: Learning, Memory, and Cognition* **21**:449-468, 1995.

[7] J. Yang and V. Honavar, "Experiments with the cascade-correlation algorithm," *Microcomputers Applications* **17**:40-46, 1998.

[8] D. Buckingham and T. R. Shultz, "A connectionist model of the development of velocity, time, and distance concepts," *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 72-77. Hillsdale, NJ: Erlbaum, 1994.

[9] D. Mareschal and T. R. Shultz, "Development of children's seriation: A connectionist approach," *Connection Science* **11**: 149-186.

[10] T. R. Shultz, "A computational analysis of conservation," *Developmental Science* **1**:103-126, 1998.

[11] T. R. Shultz, D. Buckingham and Y. Oshima-Takane, "A connectionist model of the learning of personal pronouns in English," *Computational learning theory and natural learning systems, Vol. 2: Intersection between theory and experiment*, pp. 347-362. Cambridge, MA: MIT Press, 1994.

[12] T. R. Shultz, D. Mareschal and W. C. Schmidt, "Modeling cognitive development on balance scale phenomena," *Machine Learning* **16**:57-86, 1994.

[13] S. Sirois and T. R. Shultz, "Neural network modeling of developmental effects in discrimination shifts," *Journal of Experimental Child Psychology* **71**:235-274, 1998.

[14] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Advances in neural information processing systems 2*, pp. 524-532. Los Altos, CA: Morgan Kaufmann, 1990.

[15] T. R. Shultz and F. Rivest, "Knowledge-based cascade-correlation," *Proceedings of the International Joint Conference on Neural Networks*, Vol. V, pp. 641-646. Los Alamitos, CA: IEEE Computer Society Press, 2000.

[16] T. R. Shultz and F. Rivest, "Using knowledge to speed learning: A comparison of knowledge-based cascade-correlation and multi-task learning," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 871-878. San Francisco: Morgan Kaufmann, 2000.

[17] D. L. Silver and R. E. Mercer, "The task rehearsal method of sequential learning," Technical Report #517, Department of Computer Science, University of Western Ontario, 1998.

[18] R. Parekh and V. Honavar, "Constructive theory refinement in knowledge based neural networks," *Proceedings of the International Joint Conference on Neural Networks*. Anchorage, Alaska, 1998.

[19] R. Caruana, "Multitask learning: A knowledge-based source of inductive bias," *Proceedings of the Tenth International Machine Learning Conference*, pp. 41-48. San Mateo, CA: Morgan Kaufmann, 1992.

[20] T. R. Shultz and F. Rivest, "Knowledge-based cascade-correlation: Using knowledge to speed learning," *Connection Science* **13**:1-30, 2001.

CONCLUSION

KBCC and Other Approaches

The Table 2 below summarizes how various neural network knowledge transfer techniques deal with the important issues in knowledge transfer.

The first two columns contain techniques for knowledge transfer discussed in the literature review and the names of their authors.

Table 2: Summary of Transfer of Knowledge Methods and Their Properties

Method		Author(s)	Input Restrictions	Output Restrictions	Architecture Restrictions	Allows Compositionality	Number of Sources
Representational Transfer	Copy		MBS	MBS	MBS	No	1
	Copy CC	Fahlman & Lebiere	MBS	MBS	Source must be a CC network	Prior knowledge can be processed before new knowledge	Life-long
	Copy as extra inputs	Generalization of Parekh & Honavar	MBS	None	Sources must be in functional form	Parallel composition	A small number
	Copy and glue	Waibel	MBS	None	Sources must be BP networks and are included in the final solution	Parallel composition	A small number
	Copy in ME	Hinton, using Jacobs & al.	MBS	MBS	Sources must be BP networks	Parallel composition	A small number
	DBT	Pratt	MBS	MBS	MBS	No	1
Functional Transfer	MTL/TRM	Baxter, Caruana, using Silver & Mercer	MBS	None	Share a common hidden layer. New network must be big enough learn prior knowledge	N/A	A small number
	ηMTL/TRM	Silver & Mercer	MBS	None	Share a common hidden layer.	N/A	A small number
	Meta-learner	Naik & Mammone	MBS	MBS	MBS	N/A	Life-long
	Dual memory model	Rousset & Ans as well as others	MBS	MBS	MBS	N/A	Life-long
	EBL	Thurn & MItchell	MBS	MBS	Sources must be in differentiable functional form	N/A	1
Rep.	**KBCC**	**Shultz & Rivest**	**None**	**None**	**Sources must be in functional form**	**Parallel and serial composition**	**A small number**

The third column indicates restrictions on the number of inputs and their meaning. MBS means *must be the same*, that is, the inputs of the sources must contain the same type of information using the same set of values as in the target task. Although target networks can always have more inputs (initializing the extra weights using random values), this has never been tried to my knowledge. Silver (2000) suggested that sources

could have extra inputs, but it remains unclear what would be the right way to set those extra input values. When the prior knowledge is a BP network, then those extra inputs could also be removed together with their weights.

The fourth column indicates restrictions on the number of outputs and their meaning. Here, MBS has the same meaning as for the inputs, although again, one could always delete or add extra outputs. No restrictions are mentioned when it is clear that, from the architecture viewpoint, the output of the source knowledge and the target task do not need to match.

The fifth column indicates architectural restrictions, such as whether the source and target networks must have the same architecture.

The sixth column describes the type of compositionality potentially involved. Parallel composition means that the outputs of the prior knowledge could be used, all on the same level, to generate the output of the target system. Single parallel composition is the degenerate case when only one source can be used. Serial composition means that the output of some prior knowledge could be used as the input of some other prior knowledge whose outputs can then be used to generate the output of the target system. Determining compositionality for functional methods requires a detailed discussion on compositionality that is outside the scope of this thesis.[1]

The last column indicates the number of sources that the system may have. This is either 1, a small number of sources (in parallel or competing), or life-long. Life-long models usually have a single source of knowledge that contains the experience of many tasks.

Discussion

Manuscripts 1 and 2 show that KBCC can speed up learning by using prior knowledge. Manuscript 1 also shows that KBCC is able to extract knowledge from more complicated sources of knowledge as well as to build complex solutions from multiple simpler sources of knowledge. Manuscript 1 also illustrates KBCC's ability to select the most relevant knowledge. Manuscript 2 shows that KBCC also works on realistic large tasks and not only on simple toy problems.

---

[1] For a discussion on compositionality in connectionist models see Gelder (1990).

As mentioned in manuscript 1, KBCC also addresses the input-recoding problem often faced by learners who have relevant prior knowledge (Clark & Thornton, 1997). Because of the weights feeding the recruited network, KBCC can re-use functions that are a linear transformation away from the needed function. These weights, along with the output weights, also solve the issue of aligning the structure of prior knowledge to the requirements of the current task (Gentner & Markman, 1993). Moreover, because KBCC can feed a source network with the output of another source network, it can also change Clark and Thornton's (1997) type 2 problems to type 1 problems by using a source network to do the transformation. This ability lets KBCC not only produce outputs that are the parallel composition of multiple source functions, but also that are the result of the serial composition ($f(g(x))$) of multiple source functions ($f(x)$ and $g(x)$).

Future Research

It remains to show that KBCC is not only efficient (in terms of speeding up learning), but also effective, that is, that it can improve neural network accuracy by using prior knowledge. This property is particularly important in the case of impoverished training sets. Experiments that study KBCC effectiveness in impoverished set conditions on tasks similar to those in manuscript 1 are currently underway.

It also remains to show that KBCC accurately models human cognitive development and knowledge transfer. This topic is also currently investigated in our Laboratory for Natural and Simulated Cognition at McGill.

Finally, one piece currently missing from KBCC is the ability to accumulate a huge amount of knowledge as in the case of life-long learning. Using a meta-network that learns to select promising source networks from a large number of sources to put in the pool of candidates seems a good option. As the number or sources increases, some consolidation mechanism like rehearsing may also become useful in maintaining only a small number of source networks. These remain temporarily open problems.

APPENDIX I: KBCC MATHEMATICS

KBCC requires the derivative of each candidate it has in its pool. The derivative of simple units like sigmoids are well known, but when it comes to complete networks, things are more complicated. In order to implement KBCC, I had to find the general formulas for the type of networks I wanted to be able to recruit. I developed these equations for multilayer feedforward networks, for cascade-correlation networks, and for knowledge-based cascade-correlation networks. The multilayer feedforward networks equations are not included here for space and because they are not directly related to KBCC. However, second order derivatives of CC and KBCC networks are included. These are necessary for second order optimization algorithms and for pruning methods like *Optimal Brain Damage* (LeCun, Denker & Solla, 1990) and *Optimal Brain Surgeon* (Hassibi & Stork, 1993).

The following sections develop general formulas to compute the outputs, the gradients, and the Hessians of CC and KBCC networks. These were originally developed using basic summations and multiplications without vectors or matrices. I found an equivalent form using vector and matrix operations, which is simpler and more elegant. Second order derivatives are defined using equations on vectors of matrices. Only a high level sketch of the full development of the formulas is presented here. Those formulas were implemented 'as is' and their computations were empirically compared to those of the symbolic mathematics program Maple. Maple was given only the symbolic description of a complex KBCC network (with sub-networks of various number of inputs and outputs) and asked to compute the first and second order derivative values. Small weights were used to avoid sigmoid saturations. Maple and my simulator agreed up to at least 10 digits on every value.

Cascade-Correlation Neural Networks

Notation

Sizes:

IN : Number of inputs.

OUT: Number of outputs.

K: Number of hidden units.

Labels:

B: Bias unit.

I: Input layer.

I_i: i^{th} unit of the input layer, $1 \le i \le IN$.

H_k: k^{th} hidden unit, $1 \le k \le K$.

O: Output layer.

O_o: o^{th} unit of the output layer, $1 \le o \le OUT$.

Functions:

f_u: Activation function of unit u.

f_u': First derivative of the activation function of unit u.

f_u'': Second derivative of the activation function of unit u.

Results:

V_u: Activation of unit u.

$\nabla_{\vec{x}} V_u$: Gradient of the activation of unit u with respect to the input variables $\vec{x}$ of the network.

$H_{\vec{x},\vec{x}} V_u$: Hessian of the activation of unit u with respect to the input variables $\vec{x}$ of the network.

Weights:

$\vec{w}_u$: Row vector of weights feeding unit u.

The input pattern presented to the network is denoted by the column vector:

$$\vec{p} = [p_1 \quad \cdots \quad p_{IN}]^T.$$

To simplify the notation, let's add a column vector $\vec{p}_u$ that is composed of all the values (in order) feeding the network unit u. $\vec{p}_u$ is defined as follows:

$$\vec{p}_{H_1} = [V_B \quad V_{I_1} \quad \cdots \quad V_{I_{IN}}]^T,$$

$$\vec{p}_{H_k} = [V_B \quad V_{I_1} \quad \cdots \quad V_{I_{IN}} \quad V_{H_1} \quad \cdots \quad V_{H_{k-1}}]^T, \text{ for } 1 < k \le K,$$

$$\vec{p}_O = \begin{bmatrix} V_B & V_{I_1} & \cdots & V_{I_{IN}} & V_{H_1} & \cdots & V_{H_K} \end{bmatrix}^T, \text{ for any output } O_o.$$

Using this notation, the gradient of $\vec{p}_u$ with respect to the input variables $\vec{x}$ of the network is given by:

$$\nabla_{\vec{x}} \vec{p}_u = \begin{bmatrix} \nabla_{\vec{x}} \vec{p}_{u_1} \\ \vdots \\ \nabla_{\vec{x}} \vec{p}_{u_{\dim(\vec{p}_u)}} \end{bmatrix}, \text{ where } \vec{p}_{u_i} \text{ is the } i^{th} \text{ component of vector } \vec{p}_u.$$

Similarly, the Hessian of $\vec{p}_u$ with respect to the input variables $\vec{x}$ of the network is given by:

$$H_{\vec{x},\vec{x}} \vec{p}_u = \begin{bmatrix} H_{\vec{x},\vec{x}} \vec{p}_{u_1} \\ \vdots \\ H_{\vec{x},\vec{x}} \vec{p}_{u_{\dim(\vec{p}_u)}} \end{bmatrix}, \text{ where } \vec{p}_{u_i} \text{ is the } i^{th} \text{ component of vector } \vec{p}_u.$$

Note here that $H_{\vec{x},\vec{x}} \vec{p}_u$ is a vector of Hessians, that is, a vector of matrices.

Activation

The activations of the network units at pattern $\vec{p}$ are given by the following equations:

$$V_B = 1.0$$

$$V_{I_i} = f_{I_i}(p_i), \text{ for } 1 \le i \le IN$$

$$V_{H_k} = f_{H_k}(\vec{w}_{H_k} \cdot \vec{p}_{H_k}), \text{ for } 1 \le k \le K$$

$$V_{O_o} = f_{O_o}(\vec{w}_{O_o} \cdot \vec{p}_O), \text{ for } 1 \le o \le OUT$$

Gradient

The gradients of the activations at pattern $\vec{p}$ with respect to the input variables $\vec{x}$ of the network are given by the following equations:

$$\nabla_{\vec{x}} V_B = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}$$

$$\nabla_{\vec{x}} V_{I_i} = \begin{bmatrix} 0 & \cdots & 0 & f'_{I_i}(p_i) & 0 & \cdots & 0 \end{bmatrix}, \text{ for } 1 \le i \le IN$$

$$\nabla_{\vec{x}} V_{H_k} = f'_{H_k}(\vec{w}_{H_k} \cdot \vec{p}_{H_k}) \cdot \begin{bmatrix} \vec{w}_{H_k} \cdot \nabla_{\vec{x}} \vec{p}_{H_k} \end{bmatrix}, \text{ for } 1 \le k \le K$$

$$\nabla_{\vec{x}} V_{O_o} = f'_{O_o}(\vec{w}_{O_o} \cdot \vec{p}_O) \cdot \begin{bmatrix} \vec{w}_{O_o} \cdot \nabla_{\vec{x}} \vec{p}_O \end{bmatrix}, \text{ for } 1 \le o \le OUT$$

Hessian

The Hessians of the activations at pattern $\vec{p}$ with respect to the input variables $\vec{x}$ of the network are given by the following equations:

$$H_{\vec{x},\vec{x}}V_B = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}$$, the topmost layer of $H_{\vec{x},\vec{x}}V_{H_k}$, for $1 \le k \le K$.

$$H_{\vec{x},\vec{x}}V_{I_i} = \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & f_{I_i}''(p_i) & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$, for $1 \le i \le IN$, each a layer of $H_{\vec{x},\vec{x}}V_{H_k}$, for $1 \le k \le K$.

$$\begin{aligned} H_{\vec{x},\vec{x}}V_{H_k} &= \nabla_{\vec{x}}\left[\nabla_{\vec{x}}V_{H_k}\right] \\ &= \nabla_{\vec{x}}\left[f'_{H_k}\left(\vec{w}_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\left[\vec{w}_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right]\right] \\ &= \nabla_{\vec{x}}\left[f'_{H_k}\left(\vec{w}_{H_k}\cdot\vec{p}_{H_k}\right)\right]\cdot\left[\vec{w}_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &\quad + f'_{H_k}\left(\vec{w}_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\nabla_{\vec{x}}\left[\vec{w}_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &= f''_{H_k}\left(\vec{w}_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\left[\vec{w}_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right]^T\cdot\left[\vec{w}_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &\quad + f'_{H_k}\left(\vec{w}_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\left[\vec{w}_{H_k}\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k}\right] \end{aligned}$$, for $1 \le k \le K$, where

$$\nabla_{\vec{x}}\left[\nabla_{\vec{x}}V_{H_k}\right] = \begin{bmatrix} \nabla_{x_1}\left[\nabla_{\vec{x}}V_{H_k}\right] \\ \vdots \\ \nabla_{x_{IN}}\left[\nabla_{\vec{x}}V_{H_k}\right] \end{bmatrix} = \begin{bmatrix} H_{x_1,x_1}V_{H_k} & \dots & H_{x_{IN},x_1}V_{H_k} \\ \vdots & \ddots & \vdots \\ H_{x_1,x_{IN}}V_{H_k} & \dots & H_{x_{IN},x_{IN}}V_{H_k} \end{bmatrix}$$ is a layer,

and $\vec{w}_{H_k}\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ should be computed as a row vector of scalars times a column vector of matrices. This gives a weighted sum of the matrices in $H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ weighted by scalars in $\vec{w}_{H_k}$, i.e., each matrix in $H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ is multiplied by the corresponding scalar in $\vec{w}_{H_k}$. The resulting matrices are then summed together element-wise to get the resulting matrix $\left[\vec{w}_{H_k}\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k}\right]$.

$$H_{\vec{x},\vec{x}}V_{O_o} = f''_{O_o}\left(\vec{w}_{O_o}\cdot\vec{p}_o\right)\cdot\left[\vec{w}_{O_o}\cdot\nabla_{\vec{x}}\vec{p}_o\right]^T\cdot\left[\vec{w}_{O_o}\cdot\nabla_{\vec{x}}\vec{p}_o\right] + f'_{O_o}\left(\vec{w}_{O_o}\cdot\vec{p}_o\right)\cdot\left[\vec{w}_{O_o}\cdot H_{\vec{x},\vec{x}}\vec{p}_o\right], \text{ for } 1 \le o \le OUT.$$

Knowledge-Based Cascade-Correlation Neural Networks

Notation

Sizes:

IN: Number of inputs.

OUT: Number of outputs.

K: Number of hidden units.

IN_k: Number of inputs of the k^{th} hidden unit, $1 \le k \le K$.

OUT_k: Number of outputs of the k^{th} hidden unit, $1 \le k \le K$.

Labels:

B: Bias unit.

I: Input layer.

I_i: i^{th} unit of the input layer, $1 \le i \le IN$.

$H_{k,o}$: Output o of the k^{th} hidden unit, $1 \le k \le K$, $1 \le o \le OUT_c$.

O: Output layer.

O_o: o^{th} unit of the output layer, $1 \le o \le OUT$.

Functions:

f_u: Activation function of non-hidden unit u.

$\vec{f}_u$: Activation vector-function of hidden unit u (a column vector).

f'_u: First derivative of the activation function of non-hidden unit u.

$\nabla\vec{f}_u$: Gradient of the activation vector-function of hidden unit u (a column vector where each row is a gradient).

f''_u: Second derivative of the activation function of non-hidden unit u.

$H\vec{f}_u$: Hessian of the activation vector-function of hidden unit u (a column vector where each layer is an Hessian).

Results:

V_u: Activation of non-hidden unit u.

$\vec{V}_u$: Activation vector of hidden unit u.

$\nabla_{\vec{x}} V_u$: Gradient of the activation of non-hidden unit u with respect to the input variables $\vec{x}$ of the network.

$\nabla_{\vec{x}} \vec{V}_u$: Column vector of gradients (one per output) of the activations of hidden unit u with respect to the input variables $\vec{x}$ of the network.

$H_{\vec{x},\vec{x}} V_u$: Hessian of the activation of non-hidden unit u with respect to the input variables $\vec{x}$ of the network.

$H_{\vec{x},\vec{x}} \vec{V}_u$: Column vector of Hessians (one per output) of the activations of hidden unit u with respect to the input variables $\vec{x}$ of the network

Weights:

$\vec{w}_u$: Row vector of weights feeding output unit u.

$\vec{w}_{u_i}$: Row vector of weights feeding the i^{th} input of hidden unit u.

$$W_u = \begin{bmatrix} \vec{w}_{u_1} \\ \vdots \\ \vec{w}_{u_{dimI_u}} \end{bmatrix}$$: Weight matrix that feeds hidden unit u.

The input pattern presented to the network is denoted by the column vector:

$$\vec{p} = \begin{bmatrix} p_1 & \cdots & p_{IN} \end{bmatrix}^T.$$

To simplify the notation, let's add a column vector $\vec{p}_u$ that is composed of all the values (in order) feeding the network unit u. $\vec{p}_u$ is defined as follow:

$$\vec{p}_{H_1} = \begin{bmatrix} V_B & V_{I_1} & \cdots & V_{I_{IN}} \end{bmatrix}^T,$$

$$\vec{p}_{H_k} = \begin{bmatrix} V_B & V_{I_1} & \cdots & V_{I_{IN}} & V_{H_{1,1}} & \cdots & V_{H_{1,OUT_1}} & \cdots & V_{H_{k-1,1}} & \cdots & V_{H_{k-1,OUT_{k-1}}} \end{bmatrix}^T,$$

for $1 < k \leq K$,

$$\vec{p}_O = \begin{bmatrix} V_B & V_{I_1} & \cdots & V_{I_{IN}} & V_{H_{1,1}} & \cdots & V_{H_{1,OUT_1}} & \cdots & V_{H_{K,1}} & \cdots & V_{H_{K,OUT_K}} \end{bmatrix}^T,$$

for any output O_o.

Using this notation, the gradient of $\vec{p}_u$ with respect to the input variables $\vec{x}$ of the network is be given by:

$$\nabla_{\vec{x}}\vec{p}_u = \begin{bmatrix} \nabla_{\vec{x}}\vec{p}_{u_1} \\ \vdots \\ \nabla_{\vec{x}}\vec{p}_{u_{\dim(\vec{p}_u)}} \end{bmatrix}, \text{ where } \vec{p}_{u_i} \text{ is the } i^{th} \text{ component of vector } \vec{p}_u .$$

Similarly, the Hessian of $\vec{p}_u$ with respect to the input variables $\vec{x}$ of the network is given by:

$$H_{\vec{x},\vec{x}}\vec{p}_u = \begin{bmatrix} H_{\vec{x},\vec{x}}\vec{p}_{u_1} \\ \vdots \\ H_{\vec{x},\vec{x}}\vec{p}_{u_{\dim(\vec{p}_u)}} \end{bmatrix}, \text{ where } \vec{p}_{u_i} \text{ is the } i^{th} \text{ component of vector } \vec{p}_u .$$

Note here that $H_{\vec{x},\vec{x}}\vec{p}_u$ is again a vector of Hessians, that is, a vector of matrices.

Activation

The activations of the network units at pattern $\vec{p}$ are given by the following equations:

$$V_B = 1.0$$

$$V_{I_i} = f_{I_i}(p_i), \text{ for } 1 \le i \le IN$$

$$\vec{V}_{H_k} = \vec{f}_{H_k}\left(W_{H_k} \cdot \vec{p}_{H_k}\right), \text{ for } 1 \le k \le K$$

$$V_{O_o} = f_{O_o}\left(\vec{w}_{O_o} \cdot \vec{p}_O\right), \text{ for } 1 \le o \le OUT$$

Gradient

The gradients of the activations at pattern $\vec{p}$ with respect to the input variables $\vec{x}$ of the network are given by the following equations:

$$\nabla_{\vec{x}}V_B = \begin{bmatrix} 0 & \dots & 0 \end{bmatrix}$$

$$\nabla_{\vec{x}}V_{I_i} = \begin{bmatrix} 0 & \dots & 0 & f_{I_i}'(p_i) & 0 & \dots & 0 \end{bmatrix}, \text{ for } 1 \le i \le IN$$

$$\nabla_{\vec{x}}\vec{V}_{H_k} = \nabla\vec{f}_{H_k}\left(W_{H_k} \cdot \vec{p}_{H_k}\right) \cdot W_{H_k} \cdot \nabla_{\vec{x}}\vec{p}_{H_k}, \text{ for } 1 \le k \le K$$

$$\nabla_{\vec{x}}V_{O_o} = f_{O_o}'\left(\vec{w}_{O_o} \cdot \vec{p}_O\right) \cdot \left[\vec{w}_{O_o} \cdot \nabla_{\vec{x}}\vec{p}_O\right], \text{ for } 1 \le o \le OUT$$

Hessian

The Hessians of the activations at pattern $\vec{p}$ with respect to the input variables $\vec{x}$ of the network are given by the following equations:

$$H_{\vec{x},\vec{x}}V_B = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix}$$, the topmost layer of $H_{\vec{x},\vec{x}}V_{H_{k,o}}$, for $1 \le k \le K$, $1 \le o \le OUT_k$.

$$H_{\vec{x},\vec{x}}V_{I_i} = \begin{bmatrix} 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & f_{I_i}''(p_i) & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$, for $1 \le i \le IN$, each a layer of $H_{\vec{x},\vec{x}}V_{H_{k,o}}$, for $1 \le k \le K$, $1 \le o \le OUT_k$.

$$\begin{aligned} H_{\vec{x},\vec{x}}V_{H_{k,o}} &= \nabla_{\vec{x}}\left[\nabla_{\vec{x}}V_{H_{k,o}}\right] \\ &= \nabla_{\vec{x}}\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\left[W_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right]\right] \\ &= \nabla_{\vec{x}}\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\right]\cdot\left[W_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &\quad + \nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\nabla_{\vec{x}}\left[W_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &= \left[W_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right]^T\cdot H\vec{f}_{H_k,o}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot\left[W_{H_k}\cdot\nabla_{\vec{x}}\vec{p}_{H_k}\right] \\ &\quad + \left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot W_{H_k}\right]\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k} \end{aligned}$$, for $1 \le k \le K$, $1 \le o \le OUT_k$, where

$$\nabla_{\vec{x}}\left[\nabla_{\vec{x}}V_{H_{k,o}}\right] = \begin{bmatrix} \nabla_{x_1}\left[\nabla_{\vec{x}}V_{H_{k,o}}\right] \\ \vdots \\ \nabla_{x_{IN}}\left[\nabla_{\vec{x}}V_{H_{k,o}}\right] \end{bmatrix} = \begin{bmatrix} H_{x_1,x_1}V_{H_{k,o}} & \dots & H_{x_{IN},x_1}V_{H_{k,o}} \\ \vdots & \ddots & \vdots \\ H_{x_1,x_{IN}}V_{H_{k,o}} & \dots & H_{x_{IN},x_{IN}}V_{H_{k,o}} \end{bmatrix}$$ is a layer,

and $\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot W_{H_k}\right]\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ should be computed as a row vector of scalars times a column vector of matrices. This gives a weighted sum of the matrices in $H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ weighted by scalars in $\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot W_{H_k}\right]$, i.e., each matrix in $H_{\vec{x},\vec{x}}\vec{p}_{H_k}$ is multiplied by the corresponding scalar in $\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot W_{H_k}\right]$. The resulting matrices are then summed together element-wise to get the resulting matrix $\left[\nabla\vec{f}_{H_{k,o}}\left(W_{H_k}\cdot\vec{p}_{H_k}\right)\cdot W_{H_k}\right]\cdot H_{\vec{x},\vec{x}}\vec{p}_{H_k}$.

$$H_{\vec{x},\vec{x}} V_{O_o} = f''_{O_o}\left(\vec{w}_{O_o} \cdot \vec{p}_o\right) \cdot \left[\vec{w}_{O_o} \cdot \nabla_{\vec{x}} \vec{p}_o\right]^T \cdot \left[\vec{w}_{O_o} \cdot \nabla_{\vec{x}} \vec{p}_o\right] + f'_{O_o}\left(\vec{w}_{O_o} \cdot \vec{p}_o\right) \cdot \left[\vec{w}_{O_o} \cdot H_{\vec{x},\vec{x}} \vec{p}_o\right] \quad \text{, for } 1 \le o \le OUT.$$

Knowledge-Based Cascade-Correlation Objective Functions

Notation

Use the same notation as in the previous section with few extra symbols.

Sizes:

P : Number of patterns

Labels:

C : Candidate unit c.

C_o : o^{th} output of candidate unit c, $1 \le o \le OUT_C$.

Functions:

∇f_{c_o} : Gradient of the activation function of candidate unit c, output o, $1 \le o \le OUT_C$.

Results:

T_o : Target activation for output unit o, $1 \le o \le OUT$.

E_o : Error for output unit o, $E_o = V_{O_o} - T_o$, $1 \le o \le OUT$.

Objective Function

KBCC objective functions are given by the following equations:

$$E = \sum_{p=1}^{P} \sum_{o=1}^{OUT} {E_o}^2$$

$$S_c = \frac{\sum_{o_c=1}^{OUT_c} \sum_{o=1}^{OUT} Cov\left(V_{C_{o_c}}, E_o\right)^2}{E} \quad \text{(from manuscript 2)}$$

where, $Cov\left(V_{C_{o_c}}, E_o\right)$ is given by:

$$Cov\left(V_{C_{o_c}}, E_o\right) = \sum_{p=1}^{P} \left(V_{C_{o_c},p} - \overline{V}_{C_{o_c}}\right)\left(E_{o,p} - \overline{E}_o\right),$$

$$\overline{V}_{C_{o_c}} = \frac{1}{P} \sum_{p=1}^{P} V_{C_{o_c}} \quad , \quad \overline{E}_o = \frac{1}{P} \sum_{p=1}^{P} E_{o,p} \, .$$

Gradient

The derivative of the objective function E with respect to the output weights of the o^{th} output unit is given by:

$$\begin{aligned}
\frac{\partial E}{\partial \vec{w}_{O_o}} &= \frac{\partial}{\partial \vec{w}_{O_o}}\left(\sum_{p=1}^{P}\sum_{o=1}^{OUT} E_{o,p}{}^2\right) \\
&= \sum_{p=1}^{P} \frac{\partial E_{o,p}{}^2}{\partial \vec{w}_{O_o}} \\
&= \sum_{p=1}^{P} \frac{\partial}{\partial \vec{w}_{O_o}}\left(V_{O_o,p} - T_{o,p}\right)^2 \\
&= \sum_{p=1}^{P} 2\cdot\left(V_{O_o,p0} - T_{o,p}\right)\cdot\frac{\partial V_{O_o,p}}{\partial \vec{w}_{O_o}} \\
&= 2\cdot\sum_{p=1}^{P} E_{o,p}\cdot\frac{\partial f_{O_o}\left(\vec{p}_{O,p}\cdot\vec{w}_{O_o}\right)}{\partial \vec{w}_{O_o}} \\
&= 2\cdot\sum_{p=1}^{P} E_{o,p}\cdot f'_{O_o}\left(\vec{p}_{O,p}\cdot\vec{w}_{O_o}\right)\cdot\vec{p}_{O,p}
\end{aligned}$$

The derivative of the objective function S_c for candidate c with respect to the candidate input weights is given by:

$$\begin{aligned}
\frac{\partial S_c}{\partial W_C} &= \frac{\partial}{\partial W_C}\left(\frac{1}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT} Cov\left(V_{C_{o_c}}, E_o\right)^2\right) \\
&= \frac{1}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT}\frac{\partial}{\partial W_C} Cov\left(V_{C_{o_c}}, E_o\right)^2 \\
&= \frac{1}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT} 2\cdot Cov\left(V_{C_{o_c}}, E_o\right)\cdot\frac{\partial}{\partial W_C} Cov\left(V_{C_{o_c}}, E_o\right) \\
&= \frac{2}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT} Cov\left(V_{C_{o_c}}, E_o\right)\cdot\frac{\partial}{\partial W_C}\sum_{p=1}^{P}\left(V_{C_{o_c},p} - \overline{V}_{C_{o_c}}\right)\left(E_{o,p} - \overline{E}_o\right)
\end{aligned}$$

Note that:

$$\sum_{p=1}^{P}\left(V_{C_{o_c},p}-\overline{V}_{C_{o_c}}\right)\left(E_{o,p}-\overline{E}_o\right)=\sum_{p=1}^{P}\left(V_{C_{o_c},p}E_{o,p}-\overline{V}_{C_{o_c}}E_{o,p}-V_{C_{o_c},p}\overline{E}_o+\overline{V}_{C_{o_c}}\overline{E}_o\right)$$

$$=\sum_{p=1}^{P}V_{C_{o_c},p}E_{o,p}-\overline{V}_{C_{o_c}}\sum_{p=1}^{P}E_{o,p}$$

$$-\overline{E}_o\sum_{p=1}^{P}V_{C_{o_c},p}+P\overline{V}_{C_{o_c}}\overline{E}_o$$

$$=\sum_{p=1}^{P}V_{C_{o_c},p}E_{o,p}-P\overline{V}_{C_{o_c}}\overline{E}_o-P\overline{V}_{C_{o_c}}\overline{E}_o+P\overline{V}_{C_{o_c}}\overline{E}_o$$

$$=\sum_{p=1}^{P}V_{C_{o_c},p}E_{o,p}-P\overline{V}_{C_{o_c}}\overline{E}_o$$

And hence that:

$$\frac{\partial}{\partial W_C}\left(\sum_{p=1}^{P}V_{C_{o_c},p}E_{o,p}-P\overline{V}_{C_{o_c}}\overline{E}_o\right)=\sum_{p=1}^{P}E_{o,p}\frac{\partial V_{C_{o_c},p}}{\partial W_C}-P\left(\frac{1}{P}\sum_{p=1}^{P}\frac{\partial V_{C_{o_c},p}}{\partial W_C}\right)\overline{E}_o$$

$$=\sum_{p=1}^{P}E_{o,p}\frac{\partial V_{C_{o_c},p}}{\partial W_C}-\sum_{p=1}^{P}\overline{E}_o\frac{\partial V_{C_{o_c},p}}{\partial W_C}$$

$$=\sum_{p=1}^{P}\left(E_{o,p}-\overline{E}_o\right)\frac{\partial V_{C_{o_c},p}}{\partial W_C}$$

Therefore:

$$\frac{\partial S_c}{\partial W_C}=\frac{2}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT}Cov\left(V_{C_{o_c}},E_o\right)\cdot\frac{\partial}{\partial W_C}\sum_{p=1}^{P}\left(V_{C_{o_c},p}-\overline{V}_{C_{o_c}}\right)\left(E_{o,p}-\overline{E}_o\right)$$

$$=\frac{2}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT}Cov\left(V_{C_{o_c}},E_o\right)\cdot\sum_{p=1}^{P}\left(E_{o,p}-\overline{E}_o\right)\cdot\frac{\partial V_{C_{o_c},p}}{\partial W_C}$$

$$=\frac{2}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT}Cov\left(V_{C_{o_c}},E_o\right)\cdot\sum_{p=1}^{P}\left(E_{o,p}-\overline{E}_o\right)\cdot\frac{\partial}{\partial W_C}f_{C_{o_c}}\left(W_C\cdot\vec{p}_{O,p}\right)$$

$$=\frac{2}{E}\sum_{o_c=1}^{OUT_c}\sum_{o=1}^{OUT}Cov\left(V_{C_{o_c}},E_o\right)\cdot\sum_{p=1}^{P}\left(E_{o,p}-\overline{E}_o\right)\cdot\nabla f_{C_{o_c}}\left(W_C\cdot\vec{p}_{O,p}\right)\cdot\vec{p}_{O,p}$$

REFERENCES

Abu-Mostafa, Y. S. (1993). A Method for Learning From Hints. In *Advances in Neural Information Processing Systems 5*, Steven Hanson, Jack Cowan, Lee Giles (eds), Morgan Kaufmann, San Mateo, CA, pp. 73-80.

Ans, B. & Rousset, S. (2000) Neural Networks with a Self-Refreshing Memory: Knowledge Transfer in Sequential Learning Tasks Without Catastrophic Forgetting. *Connection Science* **12**(1):1-19.

Baluja, S. & Fahlman, S.E. (1994). *Reducing Network Depth in the Cascade-Correlation Learning Architecture*, Technical Report #CMU-CS-94-209, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Baxter, J. (1995). Learning Internal Representations. In *Proceedings of the Eighth International Conference on Computational Learning Theory*. ACM Press, Santa Cruz, CA.

Baxter, J. (1996). Learning Model Bias. In *Advances in Neural Information Processing Systems 8*, David Touretzky, Michael Mozer, Mark Hasselmo (eds), MIT Press, pp. 169-175.

Buckingham, D. & Shultz, T.R. (1994). A Connectionist Model of the Development of Velocity, Time, and Distance Concepts. In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society. Erlbaum, Hillsdale, NJ.

Buckingham, D. & Shultz, T. R. (2000). The Developmental Course of Distance, Time, and Velocity Concepts: A Generative Connectionist Model. *Journal of Cognition and Development* **1**:305-345.

Bryson, A.E. and Ho, Y.-C. (1969). *Applied Optimal Control*. Blaisdell, New York.

Caruana, R. (1993). Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Machine Learning Conference*. Morgan Kaufmann, San Mateo, CA, pp. 41-48.

Caruana, R. (1995). Learning Many Related Tasks at the Same Time with Backpropagation. In *Advances in Neural Information Processing Systems 7*, Gerry Tesauro, David Touretzky, Todd Leen (eds), MIT Press, pp. 657-664.

Caruana, R. (1997). Multitask Learning. *Machine Learning* **28**:41-75.

Clark, A. & Thornton, C. (1997). Trading Spaces: Computation, Representation, and the Limits of Uninformed Learning. *Behavioral and Brain Sciences* **20**:57-97.

Comon, P. (1994). Independent Component Analysis: A New Concept? *Signal Processing* **36:**287-314.

Cybenko, G. (1988). *Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient.* Technical Report. Department of Computer Science, Tufts University, Medford, MA.

Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems* **2**:303-314.

Eberhart, R., Simpson, P. and Dobbins, R. (1996). *Computational Intelligence PC Tools*. AP Professional, Boston.

Fahlman, S.E. (1988). Faster-Learning Variation on Back-Propagation: An Empirical Study. In *Proceedings of 1988 Connectionist Models Summer School*, D. Touretzky, G. Hinton & T. Sejnowski (eds.), Morgan Kaufmann, San Mateo, CA, pp. 38-51.

Fahlman, S.E. & Lebiere C. (1990). The Cascade-Correlation Learning Algorithm. In *Advances in Neural Information Processing Systems 2*, David Touretzky (ed), Morgan-Kaufmann, pp. 524-532.

Fahlman, S.E. (1991). The Recurrent Cascade-Correlation Architecture. In *Advances in Neural Information Processing Systems 3*, Richard Lippmann, John Moody, David Touretzky (eds), Morgan-Kaufmann, pp. 190-198.

Fodor, J.A. (1983). *The Modularity of Mind.* MIT Press, Cambridge, MA.

French, R.M. (1992) Semi-Distributed Representations and Catastrophic Forgetting in Connectionist Networks. *Connection Science* 4:365-377.

French, R.M. (1994). Dynamically Constraining Connectionist Networks to Produce Distributed, Orthogonal Representations to Reduce Catastrophic Interference. In *Proceedings of the 16th Annual Cognitive Science Society Conference*, pp.335-340.

Gallant, S.I. (1986). Three Constructive Algorithms for Network Learning. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pp. 652-660.

Gallinari, P. (1995). Training of Modular Neural Net Systems. *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib (ed.), MIT Press, Cambridge, MA.

Gelder, T.V. (1990). Compositionality: A Connectionist Variation on a Classical Theme. *Cognitive Science* 14(3):355-384.

Genter, D. & Markman, A.B. (1993). Analogy – Watershed or Waterloo? Structural Alignment and Development of Connectionist Models of Analogy. In *Advances in Neural Information Processing Systems 5,* Steven Hanson, Jack Cowan, Lee Giles (eds), Morgan Kaufmann, pp. 855-862.

Golden, R.M. (1996). *Mathematical Methods for Neural Network Analysis and Design.* MIT Press, Cambridge, Massachusetts.

Hassibi, B. & Stork, D.G. (1993). Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In *Advances in Neural Information Processing Systems 5*,

Steven Hanson, Jack Cowan, Lee Giles (eds), Morgan Kaufmann, San Mateo, CA, pp. 164-171.

Heit, E. (1994). Models of the Effects of Prior Knowledge on Category Learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **20**:1264-1282.

Hertz J., Krogh, A. and Palmer R.G. (1991) *Introduction to the Theory of Neural Computation*. Addison Wesley, Redwood City, CA.

Hinton, G.E. (1989). Connectionist Learning Procedures. *Artifical Intelligence* **40**:185-234.

Hornick, K., Stinchcombe, M. and White. H. (1989). Multilayer Feedforward Networks Are Universal Approximators. *Neural Networks* **2**:359-366.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. & Hinton, G.E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation* **3**:79-87.

Jordan, M.I. & Jacobs, R.A. (1994). Hierarchical Mixtures of Experts and the EM Algorithm. *Neural Computation* **6**:181-214.

Karmiloff-Smith, A. (1992). *Beyond modularity: A Developmental Perspective on Cognitive Science*. MIT Press, Cambridge, MA.

Keil, F.C. (1987). Conceptual Development and Category Structure. *Concepts and Conceptual Development: Ecological and Intellectual Factors in Categorization*, U. Neisser (ed), Cambridge University Press, Cambridge.

Le Cun, Y., Denker, J.S. & Solla, S.A. (1990). Optimal Brain Damage. In *Advances in Neural Information Processing Systems 2*, David Touretzky (ed), Morgan-Kaufmann, pp. 598-605.

Mareschal, D. & Shultz, T.R. (1999). Development of Children's Seriation: A Connectionist Approach. *Connection Science* **11**:149-186.

Martin, G. (1988). *The Effects of Old Learning on New in Hopfield and Back-Propagation Nets.* Technical Report ACA-HI-019. Microelectronics and Computer Technology Corporation (MCC).

McCloskey, M. & Cohen, N.J. (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. *The Psychology of Learning and Motivation* **24**.

McCulloch, W.S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* **5**:115-137.

Mingers, J. (1989). An Empirical Comparison of Selection Measures for Decision-Tree Induction. *Machine Learning* **3**(4):319-342.

Minsky M.L. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, Massachusetts (1st ed.).

Mitchell, T.M. (1997). *Machine Learning.* McGraw-Hill, New York.

Murphy, G.L. (1993). A Rational Theory of Concepts. *The Psychology of Learning and Motivation* **29**:327-359.

Naik, D.K. & Mammone, R.J. (1992). Meta-Neural Networks that Learn by Learning. *International Joint Conference on Neural Networks 1992*, Vol. 1, pp. 437-442.

Nakamura, G. (1985). Knowledge-Based Classification of Ill-Defined Categories. *Memory and Cognition* **13**:377-384.

Parekh, R. & Honavar, V. (1998). Constructive Theory Refinement in Knowledge Based Neural Networks. In *Proceedings of the International Conference on Neural Networks*, Anchorage, Alaska, pp. 2318-2323

Pazzani, M.J. (1991). Influence of Prior Knowledge on Concept Acquisition: Experimental and Computational Results. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **17**:416-432.

Pratt, L.Y., Mostow, J. & Kamm, C.A. (1991). Direct Transfer of Learned Information Among Neural Networks. In *Proceedings of the Ninth National Conference on Artificial Intelligence 1991*, Vol. 2, Menlo Park, CA, AAAI Press & MIT Press, pp. 584-589.

Pratt, L. Y. (1993a). Discriminability-Based Transfer Between Neural Networks. In *Advances in Neural Information Processing Systems 5,* Steven Hanson, Jack Cowan, Lee Giles (eds), Morgan Kaufmann, pp. 204-211.

Pratt, L.Y. (1993b). *Transferring Previously Learned Back-Propagation Neural Networks to New Learning Tasks*, Ph.D. Thesis (Technical Report ML-TR-37), Rutgers University, Computer Science Department, NJ.

Pratt, L.Y. & Jennings, B. (1996). A Survey of Transfer Between Connectionist Networks. *Connection Science* **8**(2):163-184.

Precheltz, L. (1997). Investigation of the Cascor Family of Learning Algorithms. *Neural Networks* **10**(5):885-896.

Riedmiller, M. & Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The Rprop Algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, Vol. XX, IEEE, Piscataway, NJ, pp. 586-591.

Rivest F. & Shultz, T.R. (2002). Application of Knowledge-based Cascade-Correlation to Vowel Recognition. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, pp. 53-58.

Robins, A.V. (1995). Catastrophic Forgetting, Rehearsal, and Pseudorehearsal. *Connection Science* 7:123-146.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan, Washington.

Sirois, S. & Shultz, T.R. (1998). Neural Network Modeling of Developmental Effects in Discrimination Shifts. *Journal of Experimental Child Psychology* 71:235-274.

Shavlik, J. W. (1994). A Framework for Combining Symbolic and Neural Learning. *Machine Learning* **14**:321-331.

Shultz, T. R. (1998). A Computational Analysis of Conservation. *Developmental Science* **1**:103-126.

Shultz, T.R. & Bale, A. C. (2001). Neural Network Simulation of Infant Familiarization to Artificial Sentences: Rule-Like Behavior Without Explicit Rules and Variables. *Infancy* **2**:501-536.

Shultz, T.R., Buckingham, D., & Oshima-Takane, Y. (1994). A Connectionist Model of the Learning of Personal Pronouns in English. In *Computational Learning Theory and Natural Learning Systems, Vol. 2: Intersection between Theory and Experiment*, S.J. Hanson, T. Petsche, M. Kearns, & R.L. Rivest (eds.), MIT Press, Cambridge, MA.

Shultz, T.R., Mareschal, D., & Schmidt, W.C. (1994). Modeling Cognitive Development on Balance Scale Phenomena. *Machine Learning* **16**:57-86.

Shultz, T.R. & Rivest, F. (2000a). Knowledge-Based Cascade-Correlation. In *Proceedings of the International Joint Conference on Neural Networks,* Vol. V, pp. 641-646. IEEE Computer Society Press, Los Alamitos, CA.

Shultz, T.R. & Rivest, F. (2000b). Using Knowledge to Speed Learning: A Comparison of Knowledge-Based Cascade-Correlation and Multi-Task Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 871-878. Morgan Kaufmann, San Francisco.

Shultz, T.R. & Rivest, F. (2001) Knowledge-Based Cascade-Correlation: Using Knowledge to Speed Learning. *Connection Science* **13**:1-30.

Silver, D.L. & Mercer, R.E. (1996). The Parallel Transfer of Task Knowledge Using Dynamic Learning Rates Based on a Measure of Relatedness. *Connection Science* **8**:277-294.

Silver, D.L. & Mercer, R.E. (1998). *The Task Rehearsal Method of Sequential Learning.* Technical Report #517, Department of Computer Science, University of Western Ontario.

Silver, D. (2000). *Selective Transfer of Neural Network Task Knowledge.* Ph.D. Thesis, University of Western Ontario.

Simard, P, Victorri, B, Le Cun, Y & Denker, J (1992). Tangent Prop – A Formalism for Specifying Selective Invariances in an Adaptive Network. In *Advances in Neural Information Processing Systems 4*, John Moody, Steven Hanson, Richard Lippmann (eds), Morgan Kaufmann, San Mateo, CA, pp. 895-903.

Thrun, S. & Mitchell, T. (1993) Integrating Inductive Neural Network Learning and Explanation-Based Learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, R. Bajcsy (Ed.), Morgan Kaufmann, San Mateo, CA.

Towell, G.G. & Shavlik, J. W. (1994). Knowledge-Based Artificial Neural Networks. *Artificial Intelligence* **70**:119-165.

Waibel, A., Sawai, H. & Shikano, K. (1989). Modularity and Scaling in Large Phonemic Neural Networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37**(12):1888-1898.

Waugh, S. (1995). *Extending and Benchmarking Cascade-Correlation.* Ph.D. Thesis, Department of Computer Science, University of Tasmania.

Wisniewski, E. J. (1995). Prior Knowledge and Functionally Relevant Features in Concept Learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition* **21**: 449-468.

Wynne-Jones, M. (1992). Node Splitting : A Constructive Algorithm for Feed-Forward Neural Networks. In *Advances in neural information processing systems 4*, John Moody, Steven Hanson, Richard Lippmann (eds). Morgan-Kaufmann, pp. 1074-1079.

Yang, J & Honavar, V. (1998). Experiments with the Cascade-Correlation Algorithm. *Microcomputers Applications* 17:40-46.