Transfer Learning Techniques for Deep Neural Nets

STEVEN MICHAEL GUTSTEIN

Department of Computer Science

APPROVED:

Olac Fuentes, Ph.D., Chair

Eric Freudenthal, Ph.D., Co-Chair

Nigel Ward, Ph.D.

Wendy Francis, Ph.D.

Patricia D. Witherspoon, Ph.D.
Dean of the Graduate School

Transfer Learning Techniques for Deep Neural Nets

by

STEVEN MICHAEL GUTSTEIN

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

May 22, 2010

Acknowledgments

The genius differs from us men in being able to endure isolation, his rank as a genius is proportionate to his strength for enduring isolation.
- Kierkegaard

I, on the other hand, don't endure isolation well at all. So, I would like to offer some thanks to people without whom I wouldn't have produced this thesis.

I would like to thank my advisors, Olac Fuentes, for guiding my reading and offering me suggestions, support and encouragement, even when my results were less than spectacular; and Eric Freudenthal, for his technical guidance and aid in helping me obtain 3 very productive summer internships. Without Eric, I never would have come to El Paso and completed this dissertation. I am also appreciative of my other two committee members, Nigel Ward and Wendy Francis for our conversations and their comments and insights. The faculty of the Computer Science Department at the University of Texas at El Paso also has my appreciation for their unstinting willingness to answer questions that all too frequently were asked in the hallway, or with no warning other than my knock on their doors. Since my first studies of Computer Science were at New York University, I would like to thank the faculty there for helping to hone in me the tenacity and discipline needed for independent research.

Finally, I would like to thank my family, without whose love and support this never would have been started.

Abstract

Inductive learners seek meaningful features within raw input. Their purpose is to accurately categorize, explain or extrapolate from this input. Relevant features for one task are frequently relevant for related tasks. Reuse of previously learned data features to help master new tasks is known as 'transfer learning'. People use this technique to learn more quickly and easily. However, machine learning tends to occur from scratch.

In this thesis, two machine learning techniques are developed, that use transfer learning to achieve significant accuracy for recognition tasks with extremely small training sets and, occasionally, no task specific training. These methods were developed for neural nets, not only because neural nets are a well established machine learning technique, but also because their modularity makes them a promising candidate for transfer learning. Specifically, an architecture known as a convolutional neural net is used because it has a modularity defined both by the fact that it is a deep net and by its use of feature maps within each layer of the net.

The first transfer learning method developed, **structurally based transfer** relies on the architecture of a neural net to determine which nodes should or should not be transferred. This represents an improvement over existing techniques in terms of ease of use.

The second technique takes a very different approach to the concept of training. Traditionally, neural nets are trained to give specific outputs in response to specific inputs. These outputs are arbitrarily chosen by the net's trainers. However, even prior to training, the probability distribution of a net's output in response to a specific input class is not uniform. The term **inherent bias** is introduced to refer to a net's preferred response to a given class of input, whether or not that response has been trained into the net. The main focus of this work will involve using inherent biases that have not been trained into the net.

If a net has already been trained for one set of tasks, then it's inherent bias may already provide a surprisingly high degree of accuracy for other, similar tasks that have not yet been encountered. Psychologists refer to this is as **latent learning**. The accuracies obtainable in such a manner are

examined, as is the use of structurally based transfer in conjunction with latent learning. These methods provide significant recognition rates for very small training sets.

Table of Contents

Chapter 1

Introduction

1.1 Inductive Learning and the Need for Biases

Edna St. Vincent Milay once observed that "Life is not one damn thing after another. It's the same damn thing again and again." As disheartening as this may appear, repeatedly experiencing the same 'damn things' is what makes inductive learning possible. Inductive learning attempts to find consistent patterns among our experiences in order to form hypotheses about their essential features, causes and consequences. It is one of the major ways by which we learn. Were life a series of unrelated, once-in-a-lifetime events, it would be impossible to form sensible hypotheses about the world. Without the $apriori$ assumption that there are learnable, consistent relations between events it makes no sense to form explanatory hypotheses at all. However, for almost every set of experiences, the set of accurate, explanatory hypotheses can be uncountably large. A more detailed explanation of why this is so and just how uncountably large the number of accurate hypotheses can be is given in Appendix A.

1.1.1 Requirements for Inductive Hypotheses

So, rather than seeking a uniquely correct inductive hypothesis, an inductive learner will have to satisfy itself with one that is sufficiently useful. Such an hypothesis must have 3 characteristics:

Accurate Inductive learning attempts to learn generalizable rules from previous experience. The first requirement of any hypothesis generated in this way is that it accounts for that previous experience with sufficient accuracy. However, perfect accuracy is not a requirement. Data from past experience may be corrupt and most likely has only finite accuracy itself. As long

as the errors made by an inductive hypothesis when reproducing past experience impose acceptably small penalties, that hypothesis may be deemed sufficiently accurate for use.

Practical The hypothesis must be simple enough to use without incurring too high a cost with regard to computation time and memory space required. For example, a straight line over an arbitrary region may be approximated by a formula of the type $y = Ax + b$, or by an infinite Fourier series. The Fourier series requires much more memory in order to retain the various coefficients and much more time to perform a summation for the series.

General One should expect that the hypothesis will give results that are reasonably accurate even when presented with new data. One of the goals of inductive learning is to find generalizable patterns from specific data. Without generalization, learning becomes a never ending process of adding to a look-up table of past experience. Generality is the most difficult characteristic to determine, because one cannot know how robust an hypothesis will be until it is applied in new situations.

1.2 Biases

1.2.1 Relative and Absolute Biases

Favoring certain hypotheses using a criterion other than accuracy is referred to as 'bias' [33]. Ockham's razor, which dates back to the 14^{th} century English friar William of Ockham, states that the simplest hypothesis is the most preferable. Although the motivation was born of a mix of practical experience and religious belief. Ockham's razor also favors practical over impractical hypotheses.

Dietterich & Kong [16], distinguish between 'relative' bias, where certain hypotheses are preferred over others without actually excluding any, and 'absolute' bias, which does exclude certain hypotheses. Other terms for relative biases include 'preference' and 'search' bias. Similarly, 'restrictive' and 'language' bias are alternate terms for 'absolute' bias [32]. Due to the exceedingly large size both of hypotheses that may be considered and of those that are consistent

with a given set of experiences in even the simplest of cases, inductive learning needs to employ both types of biases in order to discover useful hypotheses.

Ockham's razor is an example of a relative bias. It does not exclude any overly complex hypotheses with many built-in assumptions, it merely prefers to avoid them. Einstein's argument against Quantum Mechanics - "God does not play dice with the universe" is an example of an absolute bias. It restricts the set of acceptable hypotheses to those which are deterministic.

The danger with absolute biases is that they can eliminate sets of hypotheses that are both very practical and very general. An example of this can be seen in the history of astronomy. When Copernicus first developed his helio-centric model of the solar system, he assumed that planets would follow circular orbits around the sun. Although his model was certainly more practical than the Ptolemaic geo-centric model of the solar system with its epicycles and deferents, it was not as accurate. However, when Kepler expanded the set of allowable functions for describing the orbits of the planets to include ellipses, he obtained a model that was more general, more practical and more accurate than the Ptolemaic model.

1.2.2 Imposing a Bias

In machine learning, biases are generally introduced in two ways. The first is through choice of the learning algorithm employed. This can entail an absolute bias in terms of hypotheses the algorithm can learn, a relative bias in terms of the way the algorithm will search its space of allowable hypotheses, or both.

The other common method of introducing prior knowledge or biases is by controlling the examples used for inductive learning. Abu-Mostafa [1] demonstrated that by adroitly choosing examples, one could ensure that desired invariants, would be correctly incorporated by a given machine learning algorithm. This is an absolute bias, since it only allows hypotheses that contain the desired invariances.

Both of these methods are static. The physical world requires us to learn continuously. As we learn, we dynamically acquire biases for how we will approach the learning of new tasks [53].

The success of this technique is evidenced by the fact that, generally, when we attempt to master a new skill or solve a new problem, the endeavor becomes easier if we are able to use relevant information from other contexts.

Application of hypotheses acquired in the process of mastering one task in order to provide a useful bias for the learning of another can be referred to by any one of several terms, such as 'meta-learning', 'knowledge transfer', 'transfer learning' or 'inductive transfer'. Additionally, these transferred hypotheses may be thought of as 'skills', the ability to recognize certain 'features' or even more broadly, just as 'knowledge'.

Transfer learning embodies the bias that similar tasks may be mastered with similar solutions. This is a relative bias towards generalizable hypotheses. When one has a set of similar solutions for similar problems, it becomes easy to think of them as comprising a set of special cases of some more general hypothesis. Furthermore, if the existing hypotheses are already 'practical', in the sense described in §1.1.2, then the preference for new hypotheses to be similar to the existing ones is a relative bias for 'practical' hypotheses. One may also view it as 'practical' to maintain a small set of similar hypotheses, as opposed to a large set of dissimilar ones.

The benefit sought with transfer learning is improved learning with smaller sample sizes and decreased time, since there will be less to relearn. Finding ways to employ this technique in machine learning will involve not only developing methods to actually transfer knowledge, but also learning tasks in such a way as to make them more transferable.

Recently, there has been renewed interest in the neural net community for training deep networks. This interest was spawned by advantages offered by deep networks and the development of architectures and training techniques that make deep nets more practical to use. A deep network is one composed of many layers, as opposed to the traditional network, which merely contains an input layer, a hidden layer and an output layer. One advantage of a deep network is that it requires exponentially fewer nodes than a shallow net in order to approximate several common functions [6]. This demonstrates a bias for practicality.

In terms of transfer learning, the increased modularity of deep networks makes them good candidates for such a method. They should learn concepts as a hierarchy of simpler functions.

These simpler functions should be more amenable to transfer for use in learning a new function. This exemplifies a bias for generality and in an implicit sense for practicality.

1.3 Thesis Outline

The work to be presented in this thesis will investigate techniques for using transfer learning to enable neural nets to learn new tasks with improved accuracy and very small training sets. The specific task to be examined will be recognition of handwritten characters. This problem permits the use of very simple, binary images that still retain enough variation to necessitate machine learning techniques.

A convolutional neural net (CNN) will be used to recognize the handwritten characters. The reasons for this are two-fold. Firstly, these nets represent one of the most successful methods to solve this particular problem and the reason for this success implies that they should generalize very well to vision, auditory or other tasks with strong local correlations in the input data. Secondly, this neural net architecture has both the vertical modularity of deep nets and a horizontal modularity that make it a good candidate for transfer learning techniques.

Because the work to be presented here involves both transfer learning and deep learning methods, the next two chapters will survey work that has already been done in these fields. The fourth chapter will detail experiments that both demonstrate how the structure of a deep net can be used to help localize the nodes that will be most useful for transfer learning and how transfer learning techniques possess increased utility when given noisy training data.

The fifth chapter will detail experiments using a neural net's inherent bias to enable learning with small training sets. The term 'inherent bias' is being introduced to describe a neural net's preferred output to a given input, possibly without any specific training for that input. This is particularly useful when a neural net has several nodes in its output layer. The experiments detailed in the fifth chapter will demonstrate how even in the absence of specific training, the output nodes have an inherently preferred response to various classes of input. If these responses are associated with the desired input class, particularly in conjunction with transfer learning where

the net has already been trained to recognize similar classes, then not only will surprisingly high recognition accuracies be achieved prior to training, but this effect will be enhanced if the net is allowed to train even on very small training sets. Additionally, methods of best determining a net's inherent bias will be discussed.

The sixth, and final, chapter will place these results in context and suggest avenues both for future research and practical applications.

Chapter 2

Review of Transfer Learning Techniques

2.1 Chapter Overview

The main contribution of this thesis is two new techniques for transfer learning. This chapter will present a review of existing methods. It will review and explain techniquess that use 'representational transfer', 'functional transfer', combinations of those two methods and techniques that concentrate on mapping the raw data to a different space and developing a distance metric to differentiate among several classes.

These methods will concentrate on the creation of internal representations of raw input data that are useful for multiple tasks, and on how to reuse existing internal representaions of raw input data for new tasks. In later chapters, it will be seen that the transfer learning techniques introduced in this thesis offer the advantages of ease of use and greater potential of enabling a learner to keep learning new things even after its initial training period.

At its heart, transfer learning is an unsupervised learning technique. There is no teacher to provide a label saying 'use/don't use' a particular previously learned feature in order to help master a new task. There are only features that have come from other learning experiences and thereby bias the learner towards using them as relevant features for the new task(s). The teacher, if present, has no bias regarding how transfer should occur. The only feedback given by a teacher concerns the learner's final answer, not the internal means and representations used to obtain it. The learner is free to use or to ignore any previously learned hypotheses.

Transfer learning techniques can involve using distance metrics, such as will be seen in the task clustering algorithm [54], siamese nets [8, 12] and use of the canonical distortion measure [4, 5]. Or they may merely involve searching for features that will be relevant for several tasks. This

will be shown to be the case with procedures such as discriminability based transfer [36], multi task learning [5, 10] and knowledge based cascade correlation [38, 41, 42].

Whichever method is used, the learner develops an internal means of replacing or augmenting the initial data. Transfer of information between two sets of tasks can be one-way, such as when the internal representation developed for one set of tasks influences the internal representation of the other set. This is referred to as *Representational Transfer* [44, 48]. Or, it can be mutual, such as when all sets influence their mutual internal representation.This is referred to as *Functional Transfer* [44, 48].

One-way, or representational transfer tends to involve direct copying of internal parameters from one learner to another. Mutual, or functional transfer tends to involve learning several tasks simultaneously, in order to force the learner to acquire a more broadly applicable internal representation of the input data. The rest of this chapter will survey various techniques that use representational transfer, functional transfer or both of these methods.

2.2 Representational Transfer

2.2.1 Discriminability Based Transfer

An early example of a representational transfer technique is Pratt's discriminability based transfer. This method uses portions of a previously trained neural net as a seed for training a new neural net to learn a new task [36]. Prior to Pratt, when transfer learning with neural nets was attempted via a literal transfer of the weights between the input and hidden layers, results tended to be worse than when no transfer at all was attempted [36].

Pratt observed that each hidden node of a neural network serves the same function as a decision tree node, which is to divide the input data. The goal of training is to teach each node to divide the data based on some useful criteria. However, nodes with large input weights will tend to learn slowly when trained using a backprop-like technique, since most inputs will drive them to the flat region of their response functions (i.e. either strongly firing or strongly not firing).

When these nodes already use relevant criteria to the new problem, this is useful, since there is a need for them not to change. Yet, when these nodes employ irrelevant criteria, they will then bias the net towards making classification decisions based upon irrelevant features, which is decidedly detrimental.

In order to identify useful nodes for transfer, Pratt used the *Mutual Information* metric, which she referred to as the IM metric. This metric was also used by Quinlan when evaluating hyper-planes for decision tree induction [37].

The IM Metric is sometimes referred to as *Discriminability*, thus giving *Discriminabilty Based Transfer* its name [36]. Given two variables, A and B, the mutual information metric, IM(A,B), measures the decrease in entropy of the distribution of values for one random variable given the value of the other random variable. The IM metric for A and B may be calculated as follows:

$$IM(A,B) = H(A) - H(A|B) = H(A) - (H(A,B) - H(B)) \tag{2.1}$$

where,

$H(A)$ = Entropy associated with A
$H(B)$ = Entropy associated with B
$H(A|B)$ = Entropy associated with A given B
$H(A,B)$ = Entropy associated with the joint distribution of A and B

In order to select the relevant nodes, Pratt first used training data to determine the IM value of each hidden node with respect to the target node. Then, the nodes whose IM value indicated that they did not define sufficiently relevant hyper-planes would have their input weights reinitialized to small values. The reinitialized nodes would respond more readily to error signals from a backprop-like technique, and thus learn more easily. Meanwhile, hidden nodes that responded to relevant features did not need to be relearned and were immediately available for the output nodes. This resulted in learning that was both faster than without transfer and asymptotically

about as accurate or more accurate than learning without any transfer.

2.2.2 Explanation-Based Neural Network (EBNN)

Another method utilizing a representational approach to transfer learning is use of Explanation Based Neural Nets (EBNN) [31]. EBNN's were developed to help a robot navigate its way through a known environment. If a robot is to recognize objects in a room through which it navigates, then it will have to recognize those objects at varying distances. Mitchell, O'Sullivan & Thrun built an EBNN for a robot that used two neural nets. One net (i.e. Net_1) was used to determine what it would 'sense' when it advanced by 1 meter and the other (i.e. Net_2) was used to detect whether or not it was directly in front of a door. They used these two nets to develop a single neural net, which would recognize when the robot was 1 meter from a door.

A composition of Net_1 and Net_2 was used to provide target values for Net_{1+2}, which would then recognize doors from 1 meter away. For a given input vector, X,

$$Net_{1+2}(X_{in}) = Net_2(Net_1(X_{in})). \tag{2.2}$$

Representational transfer was employed by using the first derivatives of the Net_1 and Net_2 to determine the first derivative of the Net_{1+2} function:

$$\frac{\delta Net_{1+2}}{\delta X_{in}} = \frac{\delta Net_2}{\delta Net_1}\frac{\delta Net_1}{\delta X_{in}} \tag{2.3}$$

The derivative information was incorporated into the training of the net using the TangentProp training method [49].

Using this technique, it was possible to train a robot to reliably recognize a door from 1 meter away with far fewer training examples than training from scratch, but with similar asymptotic accuracy. This procedure may easily be bootstrapped to allow a robot to recognize a door, or some other object, at a wide range of distances.

2.2.3 Cascade Correlation and Knowledge Based Cascade Correlation

There is a certain intuitive appeal to the idea of constructing nets from sub-nets. It's an idea that fits in nicely with a general engineering preference for modularity in solutions. However, to create an EBNN, one must know in advance which sub-nets to train. This is a more extreme example of the general problem faced by neural net designers of how to determine an appropriate architecture for a neural net. One way to finesse this general problem is to determine the net's architecture as it trains. Knowledge based cascade-correlation (KBCC) is an extension of Fahlman & Lebiere's cascade-correlation (CC) [19], which builds the architecture of a neural net as the net is trained. It also searches among previously trained nets to find those that will aid in the mastery of the task for which the current net is being trained.

There are two main concepts behind cascade correlation. The first is the cascade architecture, where the neural net is constructed as it is being trained, by adding one hidden node at a time. The second is the method which creates, trains and chooses hidden nodes for insertion into the net.

Initially, the net has no hidden nodes, only input and output nodes. So, the first round of training involves no back-propagation and can use a single layer training technique, such as the Widrow-Hoff rule. Next, a set of candidate hidden nodes are created, each of which gets its input from the input nodes. The outputs of these candidate hidden nodes are trained to correlate with the error output. Whichever one has the greatest correlation (or anti-correlation) will be inserted into the net. The other candidate nodes will be discarded. Then the newly inserted node's output will become an input to the output node for the net. The weight of this new connection will be adjusted so that the output error for the entire net is minimized. As this process repeats, each hidden node receives input from all previously added hidden nodes, as well as the original input. They also each supply their output to the output node. This 'cascading' of hidden nodes gives the technique its name.

One of the advantages claimed for cascade-correlation is avoiding the "herd effect". Fahlman & Lebiere consider a case where a neural net needs to learn two sub tasks in order to master its

main task. They refer to these tasks as 'Task A' & 'Task B'. If Task A is appreciably easier to learn than Task B, then there will be a strong preference for *ALL* the hidden nodes to learn Task A. Very few, if any, nodes will focus on Task B. Only once Task A has been redundantly mastered will hidden nodes attempt to learn Task B. Of course, not only will they be less responsive to backprop due to their, presumably, larger weights [36], but also they will have a tendency to *ALL* attempt to master Task B, thus *ALL* forgetting Task A. This may result in a long period of indecision before the hidden nodes divide the two tasks among themselves. By training one hidden node at a time, cascade-correlation avoids this problem.

The main augmentation that KBCC makes to cascade correlation is that instead of being restricted to merely adding a single hidden node to a neural net, this method also attempts to add entire trained sub-nets [41]. This allows for a much greater instantaneous transfer of knowledge, along with creating a net that has a greater inherent degree of modularity.

KBCC has been successfully used for vowel recognition, gene-splice junction identification and has demonstrated faster training times than Multi-Task Learning in some experiments [38, 40, 52].

2.3 Functional Transfer

2.3.1 Multi-Task Learning (MTL)

Multi-Task Learning (MTL) is, perhaps, the most widely used transfer learning technique. As such, it is a seemingly reasonable benchmark to use for other transfer techniques. However, it is a functional rather than representational technique. So, rather than being given a ready made internal representation for input data that will be useful for several tasks, it must create such a representation from scratch. This may make comparisons between it and representational transfer techniques, like KBCC, of limited utility. MTL was first introduced by Caruana in the mid to late 90's [9, 10]. It entails training a net in several tasks, simultaneously.

Caruana performed experiments both using the 1D-ALVINN road image simulator, developed

by Pomerleau [35] and using a simulator referred to as "1D-DOORS", which trained a neural net to locate doorknobs and identify door types in images of doors taken with a robot-mounted camera. In each case MTL outperformed Single Task Learning (STL) with respect to the accuracies for the various tasks.

Caruana [11] attributes this improvement to 5 main effects:

Data Amplification An effective increase in sample size due to information from related tasks. There are three forms of this amplification:

- Statistical Data Amplification - When different tasks rely upon the same feature (e.g. F) in a noisy data set, it is easier to average through the noise in order to learn that feature.
- Sampling Data Amplification - When different tasks rely upon the same feature (e.g. F) in a noise-free data set, this feature may still be difficult to learn. This would be the case if F were a non-linear function of the input. However, the different error signals produced by each task with respect to F will extract more information about F from each training sample, thus giving the net more feedback from which to learn F.
- Blocked Data Amplification - When a neural net has the option of learning any one of a set of features that are sufficient, though not necessary, for mastering a given task, it will concentrate on learning the feature with the simplest internal representation. This can result in a complex feature being 'blocked' by a feature that is easier to represent. If different tasks rely upon some of the same complex features, but not upon all of the same simple features, then samples which block the learning of the difficult feature for one task won't block it for all the others. This will increase the number of samples that give the net an opportunity to learn the shared difficult feature. Caruana regards this as an extreme case of 'Sampling Data Amplification'. It is also another way to overcome the 'herd' effect mentioned earlier with respect to cascade correlation.

Attribute Selection When a feature depends upon only a small subset of the input, it will be difficult to learn. This is because the rest of the input will function as noise with respect to

that feature. If several tasks that rely on a particular feature train simultaneously, then in a manner analogous to 'Statistical Data Amplification', that feature will be easier to learn.

Eavesdropping A given feature (e.g. F) may be used in a simple way for one task (e.g. T_1), but in a complex way for a second task (e.g. T_2). When this is the case, a net learning just T_2 will have a difficult time learning F. However, if the net trains simultaneously on T_1 and T_2, then it will be as though the portions of the net focused upon T_2 are able to eavesdrop on those focused upon T_1, thus enhancing the net's ability to use F in order to learn T_2. This effect seems very similar to the data amplification effects mentioned above.

Representation Bias By forcing the net to learn an internal representation that is broadly applicable to multiple related tasks, it is more likely to achieve an internal representation that captures the essential regularities of each task. This will then result in a net that is extremely well suited for Representational Transfer techniques. It will also make Latent Learning for neural nets, as explained in Chapter 5, possible.

Overfitting Prevention By having more varied uses for a given feature, a neural net is given more varied error signals. This will help prevent the net from overfitting.

2.3.2 η MTL

Any transfer learning technique needs to ensure that only *relevant* knowledge is transferred. Otherwise a bias will be introduced that will cause the learner to favor hypotheses that make determinations based upon irrelevant features. Such hypotheses are likely to be faulty.

Discriminability Based Transfer limited itself to relevant knowledge by using a pre-test of each hidden nodes' IM metric with respect to the new task; Explanation Based Neural Nets used prior knowledge provided by humans who chose which nets to combine and Knowledge Based Cascade-Correlation constructs a net by seeing which previously trained nets or newly created nodes seem to correspond to the most informative unlearned feature or sub-task. If MTL is to be a truly effective technique, there needs to be a way to ensure that it only attempts functional

transfer among tasks that are mutually relevant and that such transfer is in some way regulated by the degree to which the tasks are mutually relevant.

A series of papers variously by Silver, Mercer, Poirer & McCracken introduced the η MTL training technique, which regulates the degree of functional transfer [43, 44, 45, 47, 46]. ηMTL simultaneously learns a 'main' task along with several 'auxiliary' tasks. It was originally designed for a 3-layer (i.e. input layer, hidden layer and output layer) feed-forward neural net, where each output node corresponded to a single binary task. The learning rate of the output node associated with each task (i.e. η) is dependent upon its degree of relatedness to the main task.

One may easily construct a case where there is a main task and the other tasks are being learned purely for the sake of mastering that task. For example in medical diagnosis, a net could be trained to make a diagnosis using easily acquired test results as input. This learning could be aided by functional transfer if simultaneously the net were trained to predict the results of other, more expensive, tests that could be used to diagnose similar ailments. Now, in this instance, one would really only be interested in learning to correctly diagnose a particular condition. Learning to predict the results of the other diagnostic tests is only useful insofar as it helps with the main task.

Obviously, the net should be more sensitive to the tests that most strongly correlate to the diagnosis being sought. Additionally, the net's learning process should not focus upon one of the auxiliary tasks to the detriment of the main task. Neural nets are biased towards learning simple linear hypotheses over more complex non-linear ones. If one of the auxiliary tasks has a simple linear representation, then the net will be strongly biased towards learning the auxiliary task even at the cost of learning the main task. To counteract this, ηMTL ties the learning rate (i.e. η) of each output node to its similarity to the main task. This way, when the net tries to use backprop to learn its many tasks, it will favor learning those tasks that are most similar to the main one.

The similarity between the main task and each of the auxiliary tasks is determined in two ways. The first of these ways is a static method. Using the IM metric that Pratt borrowed from Quinlan [36, 37], one can find the degree to which knowledge of the result of an auxiliary task

reduces the entropy associated with the main task. The greater the reduction in entropy is, the more similar the two tasks are. It should be noted that this measure is created before training begins and it is not modified as training proceeds.

The second method is dynamic. It uses a measure that is modified as training proceeds. After each training epoch, one finds the IM metric for each hidden node with respect to each task. If there are j hidden nodes and $k+1$ tasks, then a j dimensional vector can be associated with each of the $k+1$ tasks. Using the notation from Silver's Ph.D. thesis [43], the hidden vector of IM factors for the n^{th} task, $0 \leq n \leq k$, will be referred to as MIHV$_n$.

The magnitude of the $cosine$ between the vectors MIHV$_0$ & MIHV$_n$ is a measure of similarity between the 0^{th} task (i.e. the main one) and the n^{th} task (i.e. an auxiliary one). Vectors that are strongly parallel or anti-parallel will correspond to tasks that rely upon the same features to similar degrees.

It is necessary to guard against training too closely to an auxiliary task that has been poorly learned, but seems very similar to the main task. So, the measure of relatedness used to determine the degree of emphasis placed upon learning the n^{th} auxiliary task is

$$rel_n = \frac{cs_n}{E_n + \psi} \tag{2.4}$$

where,

c is an arbitrary scaling parameter
s_n is the magnitude of the $cosine$ between the vectors MIHV$_0$ & MIHV$_n$
E_n represents an error measure over task n,
ψ is a small additive term to protect against division by 0

In his thesis, Silver argued that neither a static measure of relatedness, such as Pratt's IM metric which is created prior to training, nor a dynamic measure of relatedness, as described above which evolves as training occurs, is ideal [43]. Therefore, to capture the best of both worlds, he used a hybrid measure of relatedness, which was the product of the static & dynamic

measures of relatedness:

$$R_n = S_n \times D_n, \tag{2.5}$$

where,

R_n corresponds to the hybrid measure of relatedness between the 0^{th} and n^{th} tasks, which is used to modify the learning rate (i.e. η) for the n^{th} task
S_n corresponds to the static measure of relatedness between the 0^{th} and n^{th} tasks
D_n corresponds to the dynamic measure of relatedness between the 0^{th} and n^{th} tasks

Experiments performed on artificial data sets have shown that using these metrics for task relatedness does, indeed, provide a better method of transfer learning [43]. Furthermore, ηMTL was later used successfully to help train a net to diagnose coronary artery disease [46].

2.3.3 Generating Pseudo-Tasks for Transfer Learning

Of course, it may not always be possible to find a set of related tasks for the problem in which one is interested. One approach to overcome this minor difficulty is detailed by Ahmed et al. [3] - just invent your own tasks. Or, in their particular case - pseudo-tasks.

Ahmed et al. were attempting to use a convolutional neural net (CNN) for object recognition. This architecture will be explained in more detail in section 3.5.5, which details the properties of this neural net architecture.

The process used to generate the pseudo-tasks was first to pre-process the training images with a set of Gabor filters with 4 orientations and 16 scales. This pre-processing was not necessary, but it made their technique more robust. Then, 2D patches were selected at random from the training images. A pseudo-task then became finding the maximum value obtained by using this patch as a 2D filter over an image. The maximum value, I believe, is the maximum value obtained for any placement of the filter on an image. The net was then trained for the pseudo-tasks of

calculating these values. The use of Gabor filters was Ahmed et al.'s way of incorporating their prior knowledge of the sorts of features that would likely be important into their net.

They used this net to perform object recognition with the Caltech-101 dataset, and using the Face Recognition Grand Challenge data set (FRGC 2.0), they performed gender and ethnicity recognition. They consistently achieved markedly better (though in one case comparable) results than others had published for those same problems.

2.3.4 Kernel Methods

Kernel methods are a very general class of learning algorithms for single task problems. Because these methods have been so successful and popular, there is a strong incentive to find ways of extending them to multi-task problems and, thus, derive some of the benefits of transfer learning. Some recent work in creating multi-task kernels has been done by Evgeniou, Micchelli and Pontil [18, 30].

When given a training set, $\{\mathbf{x_i}, y_i : 1 \leq i \leq m, \mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d, y \in \mathcal{Y} = \{-1, 1\}\}$, kernel methods attempt to learn a function, $f(\mathbf{x}) \in \mathcal{H}_K$, where $\mathcal{H}_K$ is a reproducing kernel Hilbert space (RKHS) with a kernel, K. The function, f, is intended to correctly associate each x_i with the corresponding y_i, thereby classifying them. The approach used is to minimize the cost functional,

$$\frac{1}{m}\sum_{i=1}^{m} L(y_i, f(\mathbf{x_i})) + \gamma||f||^2 \tag{2.6}$$

where the first term measures the average error f has over the sample set of data. The lower the error, the greater the accuracy of f over the sample set. The second term is a measure of how general f is expected to be as a classifier. The smaller the second term, the greater the belief that f will correctly classify data that was not in the original training set.

According to Wahba's *representer theorem*, as cited and expanded upon by Evgeniou et al. [18], the f that minimizes the cost functional can be expressed as

$$f(\mathbf{x}) = \sum_{j=1}^{m} c_j K(\mathbf{x_j}, \mathbf{x}) \tag{2.7}$$

where,

$c_j \in \mathcal{R}$;

$\mathbf{x_j}$ is one of the $1 \leq j \leq m$ training samples;

$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$;

$K(\mathbf{x_j}, \mathbf{x})$ is the kernel function.

Now, consider a function, Φ, which maps from $\mathcal{X} \subseteq \mathbb{R}^d$ to $\mathcal{W} \subseteq \mathbb{R}^{d_2}$, where $\mathcal{W}$ is a Hilbert space. If we let Φ map $\mathbf{x} = (x_1, x_2, ..., x_d)$ to the countably infinite space

$$(x_1, ..., x_d, x_1x_1, x_1x_2, ..., x_dx_d, x_1x_1x_1, ...)$$

then Φ provides a space of all functions $f : \mathcal{X} \to \mathbb{R}$, defined as $f(\mathbf{x}) = \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle$ for $\mathbf{w} \in \mathcal{W}$ and $\mathbf{x} \in \mathcal{X}$. The norm of f is $\langle w, w \rangle_{\mathcal{W}}$. This space, according to Evgeniou et al. [18] is the RKHS, H_K, that was being sought. Its kernel, K is $K(\mathbf{x}, \mathbf{t}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{t}) \rangle$.

The cost functional to be minimized is now,

$$R(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, \langle \mathbf{w}, \Phi(\mathbf{x}) \rangle) + \gamma \langle w, w \rangle_{\mathcal{W}} \tag{2.8}$$

The weight vector, since that is generally used term for $\mathbf{w}$, that will minimize $R(\mathbf{w})$ has the form

$$\mathbf{w} = \sum_{j=1}^{m} c_j \Phi(\mathbf{x}_j) \tag{2.9}$$

However, this is still single task learning. There is no knowledge transfer among tasks. In order to achieve such transfer, it is necessary to derive a multi-task kernel with a corresponding cost functional.

The cost functional chosen to do this by Evgeniou et al. [18] is

$$S(\mathbf{w}) = \frac{1}{nm} \sum_{l=1}^{n} \sum_{j=1}^{m} L(\mathbf{y}_{jl}, f_l(\mathbf{x}_{jl})) + \gamma J(\mathbf{w}'\mathbf{B}), \tag{2.10}$$

where,

m is the number of samples per task;

n is the number of tasks;

d is the dimensionality of the input space;

$\mathbf{w}$ is the weight vector from before, but now $\mathbf{w} \in \mathbb{R}^p$, where $p \geq nd$;

B is a matrix that maps between $\mathbb{R}^p$ and $\mathbb{R}^{nd}$. It contains the terms that link different tasks to each other.

$J(\mathbf{w}'\mathbf{B})$ is analogous to $||f||^2$, except it also contains terms linking different tasks to each other, due to its dependence upon B.

The linkage terms provide the mechanism for transfer learning. For each task, l,

$$f_l(\mathbf{x}) = (\mathbf{w}'\mathbf{B_l}\mathbf{x}) \tag{2.11}$$

Because this limits the mappings of $\mathbf{x}$ to linear transformations, this technique will only work for problems that are already linearly separable. The corresponding kernel is referred to as a *linear multi-task kernel*. It's expressed as

$$K((\mathbf{x}, l), (\mathbf{t}, q)) = \mathbf{x}'\mathbf{B_l'}\mathbf{B_q}\mathbf{t} \tag{2.12}$$

where $\mathbf{x}$ and $\mathbf{t}$ are input vectors; l and q are specific tasks; and B_l' and B_q are the sub-matrices of B that specifically apply to tasks l and q.

Now, using the representer theorem, the minimizing weight vector has the form

$$\mathbf{w} = \sum_{j=1}^{m}\sum_{l=1}^{n} c_{jl} B_l x_{jl} \tag{2.13}$$

so, for each task, q,

$$f_q(\mathbf{x}) = \sum_{j=1}^{m}\sum_{l=1}^{n} c_{jl} K((\mathbf{x_{jl}}, l)(\mathbf{x}, q)) \tag{2.14}$$

Now, all that's left to do is to choose a specific loss function, L and a specific form of the function $J(\mathbf{w}'\mathbf{B})$. The loss function chosen by Evgeniou et al. [18] was the hinge loss function,

$$L(\mathbf{y}, f(\mathbf{x})) = max(0, 1 - yf(\mathbf{x})) \tag{2.15}$$

They also included the additional constraints,

$$y_{jl} f(\mathbf{x}_{jl}) \leq 1 - \xi_{jl} \tag{2.16}$$

and

$$\xi_{jl} \geq 0, \tag{2.17}$$

thus, using the loss function for support vector machines (SVM), probably the best known kernel method for inductive learning.

In order to create linkages between the various tasks the matrix product of $\mathbf{B_l'B_q}$ in the kernel function $K((\mathbf{x}, l), (\mathbf{t}, q))$ is replaced by

$$(1 - \lambda + \lambda n \delta_{lq})\mathbf{Q} \tag{2.18}$$

where n is the total number of tasks and λ provides the linkage between tasks.

The function $J(\mathbf{w'B})$ now takes the form

$$J(\mathbf{w'B})) = \frac{1}{n}(\sum_{l=1}^{n} \|\mathbf{w'B}_l\|^2 + \frac{1-\lambda}{\lambda} \sum_{l=1}^{n} \|\mathbf{w'B}_l - \frac{1}{n} \sum_{q=1}^{n} \mathbf{w'B}_q\|^2) \tag{2.19}$$

where the first term reduces to the maximum margin requirement for a single task SVM, thereby demonstrating our bias for perceived generality and the second term imposes a requirement that related tasks have decision boundaries that are similar to each other. The smaller λ is the greater the degree of similarity. When $\lambda = 1$, the tasks are learned independently.

Although work remains to be done, extending this technique to non-linear kernels, results obtained by Evgeniou et al. [18] do confirm expectations that the advantages of transfer learning techniques become more apparent as the number of related tasks increases and the number of training samples per class decreases.

2.4 Combinations of Representational and Functional Transfer

So far, the examined methods have tended to use either representational or functional transfer, but not both. However, it is not necessary to treat representational vs. functional transfer as an either/or proposition. Some transfer learning techniques advantageously use both methods to learn new tasks and to create favorable biases for learning new tasks.

2.4.1 Connectionist Glue

One of the earliest demonstrations of transfer learning was also one of the first to use both the representational and functional approach. The technique was called *Connectionist Glue* and was published by Waibel et al. in 1989 [57]. Waibel and his collaborators were trying to train a neural net to recognize the phonemes corresponding to 'b','p','d','t','g' and 'k'. Their initial approach was to train one large net to learn all the phonemes simultaneously. However they found the required training time and training set size did not scale well as the number of phoneme classes increased. So, they trained two smaller nets. One of these nets was trained to discriminate among 'b','d' and 'g', while the other was trained to discriminate among 'p','t' and 'k'. These two nets were then 'glued' together and retrained as a single net.

It should first be made clear that the tasks of recognizing each phoneme are all related to each other. Although the phonemes are all distinct sounds, they all are 'stop' consonants. In order to produce these phonemes, the air in the vocal tract is stopped, held and then suddenly released. 'b','d' and 'g' are all voiced consonants, while 'p','t' and 'k' are each unvoiced consonants. Furthermore, 'b' and 'p' are both articulated as bilabial stops (i.e. the stop is made with both lips), 'd' and 't' are both articulated as alveolar stops (i.e. the stop is made with the tip of the tongue against the alveolar ridge), while 'g' and 'k' are both articulated as velar stops (i.e. the stop is made with the back of the tongue against the soft palate).

There seems to be no shortage of features to be shared among the tasks and a particular

dearth of features that are only relevant to a single task. This should have had the makings of a set of tasks that would be ideally suited for MTL techniques. The need to initially train two separate nets, rather than one large net seems to contradict the results expected to be obtained with MTL. After explaining their experiment in more detail, an explanation for this discrepancy will be offered in terms of the *herd effect* [19]. This effect was previously described in Section 2.1.3.

The net architecture used by Waibel et al. had 4 layers - an input layer, 2 hidden layers and an output layer. One sub-net was trained to recognize the voiced consonants (i.e. 'b','d' & 'g'); the other was trained to recognize the unvoiced consonants (i.e. 'p', 't' & 'k'). The method of gluing the two sub-nets together involved freezing the connections between the first hidden layer and the input layer of the two sub-nets, adding a new second hidden layer and a new output layer for the joint net and then allowing training to commence. Additionally, three variations on the glued first hidden layer were examined

The first variation involved training a third net to discriminate between voiced and unvoiced consonants. The first hidden layer of the voiced/unvoiced discriminating net was then added to the glued net and treated just like the first hidden layers of the other *b,d,g* discriminating sub-net and the *p,t,k* discriminating sub-net. It was felt this would give the glued net access to additional relevant features, which would help it train.

The second variation mimicked the first variation with the addition of some uninitialized nodes to the first hidden layer. The uninitialized nodes were then free to train. This was an attempt to allow the net to discover its own extra features.

The final variation involved fine tuning the net after the initial round of training by unfreezing the connections between the first hidden layer and the input layer, then performing a final round of training. Results were only reported for the net where both free nodes and nodes to distinguish between voiced and unvoiced phonemes were included in the first hidden layer.

Waibel et al. compared their 4 glued nets' ability to correctly classify the 6 phonemes with a net that would base its choice on whichever of the two original sub-nets was more confident of its classification. As one would expect, the four glued nets all dramatically outperformed the

unglued net. Additionally, three of the four glued nets also outperformed by 0.1% - 0.3%, a net which was trained from scratch on all 6 phoneme classes; the under performing glued net only underperformed by 0.1%. However, the glued nets needed only slightly more than $1/5$ of the training time as the net which learned all the classes at once.

This technique clearly uses functional transfer, both in the formation of the original sub-nets and in the way these sub-nets are allowed to interact when forming the second hidden layer of the glued net. It also uses representational transfer in the manner by which it creates the first hidden layer of the glued net. This layer has inherited the internal representation of the first hidden layer directly from the two original nets.

However, one still has to wonder why this gluing was necessary. Earlier work was discussed describing the advantages of functional transfer for learning related tasks [9, 10, 11], wherein the claim was made that simultaneous learning of related tasks is more effective than learning them one at a time. Why did Waibel et al. not share in this experience?

The cascade-correlation experiments performed by Fahlman & Lebiere's [19] contain a possible explanation. Part of the advantage for cascade-correlation was claimed to come from its avoidance of the *herd effect*. This phenomenon is encountered when there are several tasks to be learned by a hidden layer, if one sub task is markedly easier to learn than the others, then all the nodes will rush to learn that sub task. Once they have done so, they will all rush to learn some other task, thus unlearning the first task. This will cause them all to again attempt to relearn the first task. It seems plausible that distinguishing between voiced and unvoiced consonants could be such an easy task to learn, that they successfully blocked the learning of determining the stop style (i.e. bilabial, alveolar or velar) for each task. If so, this would be indicative of a class of problem for which MTL must be used cautiously and in a modified fashion.

2.4.2 Task Rehearsal

In order to use functional transfer for sequentially learning tasks, one needs a way to both efficiently access and maintain a source of training examples, as well as retaining previously learned

tasks. The task rehearsal method achieves both goals [45]. This method makes use of a one neural net to provide short term learning (i.e. the *Short Term Learning Net*) and a set of single task nets (i.e. the *Domain Knowledge Nets*) to provide storage for previously learned tasks. Acquisition of a new task occurs in the short term learning net, while storage and retrieval of already mastered tasks is provided by the domain knowledge net.

When faced with a training set for a new task, the domain knowledge nets determine which previously learned tasks are most similar to the new task to be learned (it should be noted that the measure of relatedness used in [45] is slightly different than that used in [43]). Then, a short term learning net is created to learn the new task along with the previously mastered old tasks using the ηMTL technique. The new task is treated as the primary task to learn and the old tasks are regarded as auxiliary tasks.

However, the training sets used to retain knowledge of the old tasks are *virtual*. Rather than store every training example from every task ever mastered, the task rehearsal method uses the new training set and appropriate nets from the domain knowledge nets to generate virtual training examples with which to train the short term learning net.

These virtual training examples are generated by taking training data from the new task's training set and getting the response of the domain knowledge nets for transferred tasks to this data. The responses of the domain knowledge nets are then used as target values for the corresponding nodes of the short term learning net. Once the new task has been learned, the portions of the short term learning net, which are responsible for the new task, are saved in the set of domain knowledge nets.

This method was initially investigated using two synthetic task domains and one real world task domain (i.e. prediction of coronary artery disease). As the number of training samples for a new task decreased, but the number of previously mastered tasks increased, the improvement in learning from using task rehearsal became more marked. Experiments were also performed which demonstrated the necessity of using the ηMTL technique for training the Short Term Learning net. When standard MTL learning was used, the learning of the new task was noticeably harmed by negative transfer [46].

2.5 Learning Distance Metrics

When confronted with a task of distinguishing among a set of mutually exclusive classes, one extremely effective knowledge transfer technique is to map the input data to a suitable feature space and then to learn a distance metric. This class of methods very clearly blends both functional and representational transfer. Creation of the metric utilizes functional transfer. Use of the metric for new classes utilizes representational transfer.

2.5.1 Canonical Distortion Measure (CDM)

An impressive demonstration and explanation of distance metrics can be seen in Baxter's development and use of the *Canonical Distortion Measure* [4, 5]. Baxter examined the case of trying to differentiate amongst a large set of distinct classes. His approach was to learn a function that would map the input space to a feature space and then to learn a distance metric for that feature space. This feature space is a shared internal representation for the set of classes. He demonstrated that the upper bound on the number of examples per task needed to learn this feature space would scale as $O(a + \frac{b}{n})$, where a is the number of examples required to learn a task if the correct basis set for the feature space were already known, b is a measure of the complexity of the feature space available to the learner and n is the number of tasks given to the learner. Finally, Baxter also demonstrated that the Canonical Distortion Measure is the optimal measure to use for 1 nearest neighbor classification.

The CDM may be explained as follows:

If F is the set of functions that map the input space into the feature space, then if each $f \in F$ can be expressed as

$$f(x) = \sum_{i=1}^{k} w_i \phi_i(x), \tag{2.20}$$

where,

ϕ_i is the i^{th} feature basis vector,

w_i is the weight associated with it,

then,

$$\rho(x, x') = (\Phi(x) - \Phi(x'))W(\Phi(x) - \Phi(x')) \tag{2.21}$$

where,

ρ is the *Canonical Distortion Measure*,

W is a kXk matrix of weights,

Φ is the function that projects the input space into the feature space, and

x' is an input vector with a known class label.

The classification given to x is the classification of whichever x' results in the smallest $\rho(x, x')$.

A demonstration of the power of this approach was given in [5]. Using a data set of printed Japanese kanji provided by the CEDAR Group at SUNY Buffalo, a neural net was initially trained to discriminate among 400 Japanese kanji out of a set of 3,018. The other characters were all identified as being not in the 'chosen' 400.

This is a clear example of functional transfer. However, having created a sufficiently general internal representation, it was now possible to differentiate amongst all the character classes using 1-nearest neighbor (1-NN) classification by just learning the hidden layer representations for the new classes. Baxter's reuse of the existing hidden layer to provide an internal representation with which to distinguish among the new classes is an instance of representational transfer. So, CDM employs both functional and representational transfer.

The success of this method is seen by the fact that it had a misclassification rate of only 7.5%, which compares favorably with the 4.8% misclassification rate reported by the CEDAR group on the same data set [50], especially since no optimizing or adjusting of the input was used.

2.5.2 Siamese Nets

There is a specialized neural net architecture, that implicitly uses Baxter's CDM method. A siamese architecture [8, 12] trains a pair of neural nets, that have shared weights, to produce similar outputs for all inputs of the same class and dissimilar outputs for inputs of differing classes. The Euclidean distance between the outputs of the two nets is then used to determine if their two inputs belong to the same class. This process is equivalent to learning a set of ϕ functions for the CDM where W is the identity matrix. Also, rather than looking for the x' that minimizes $\rho(x, x')$, it merely determines if $\rho(x, x')$ falls below some maximum threshold for x and x' to be considered as belonging to the same class. This technique has been successfully used for both face recognition and handwriting verification [8, 12].

The use of many classes to develop the ϕ functions corresponds to functional transfer. However, the ability to use the ϕ functions and distance metric to recognize when a new class is encountered and to recognize its members corresponds to representational transfer.

2.5.3 Task Clustering (TC) Algorithm

Another technique, which learns distance metrics for a set of tasks is the Task Clustering Algorithm [54]. Unlike the previous distance metric examples given, this method does not assume each task represents identification of one member from a set of disjoint classes. It attempts to learn an appropriate distance measure for each class by using a globally weighted version of the Euclidean distance:

$$dist_d(x, x') = \sqrt{\sum_i d^{(i)}(x^{(i)} - x'^{(i)})^2}, \tag{2.22}$$

where

x is the vector we want to classify;

x' is one of a number of data samples with known classification and

d is a learned vector that determines the relative weight of each input dimension.

Each class will, of course, have its own optimal d vector.

Transfer is accomplished by grouping tasks together on the basis of how well their learned distance metrics work for each other. Then, the various tasks are clustered together and a new distance metric is learned for the cluster. This is an example of functional transfer.

When the training set for a new class is presented, one can decide which task cluster is the best for the new task and then use its d vector for the new task. The recommended method for determining which d vector is best for the new task is to see which one maximizes the ratio between inter- and intra-class distance in the training data for the new class. There is a clear conceptual similarity here to ηMTL, because transfer is being limited to sufficiently *relevant* tasks.

The key to grouping similar classes together lies in the *Task Transfer Matrix*, C. Each element, $c_{(n,m)}$, of C represents the expected accuracy for task n when using the optimal d parameter for class m. Similar tasks can reasonably be expected to have similar distance metrics over their inputs.

The next step is to partition the set of classes into *Task Clusters*, so that the sum of the average of the $c_{(n,m)}$'s for each task cluster is maximized. The maximization of this functional provides the necessary measure of task relatedness.

This method was developed to serve as a learning algorithm to help a mobile robot learn perceptual tasks as it moved about in the physical world. It was found that not only does this method discover task clusters that are 'meaningful', but also that it successfully selects relevant knowledge to transfer.

2.6 Zero-data Learning

One of the contributions of this thesis is recognition of the utility of a net's inherent biases and latent learning for transfer learning. Both of these terms will be explained in more detail in Chapter 5, however, a brief description of these two terms would be as follows:

Inherent bias refers to the output a neural net tends to have for a given class, whether or not it has been trained to have that particular output. Latent learning refers to the way when a net is trained to have specific outputs for a set of related classes, then its inherent biases for other, similar classes, become much more accurate in differentiating among those classes. In this thesis, a net's inherent bias was treated as something that must be learned by the net's trainer/user.

During the summer when these results were first obtained, a similar, though more activist method was developed by Larochelle et al. [28]. In their approach, which they called 'Zero-data Learning', a description function, as determined by the function's designers (Larochelle et al. did not limit themselves to neural nets), was learned for a set of classes. This function would be learned by training the net on a subset of those classes, providing the net with easily distinguishable output. Then, with knowledge of the prediction function in hand, one would only need a description of the new class, not an actual sample, in order to determine what the output of the description function would be when presented with a member from the new class.

The key differences between their research and the work presented here are the need for significant prior knowledge of the set of classes when designing a descriptor function for zero-data learning and the superior demonstrated generality of the zero-data learning technique, which is not limited to neural nets. Although I expect that the use of inherent biases in conjunction with latent learning could be generalized beyond neural nets, I have made no attempt to do so.

2.7 Summary of Transfer Learning Techniques

The methods of transfer learning that have been examined are focused mainly on how and what to transfer. Their transfer methodology may involve a one way transfer from one set of tasks to another, as with representational transfer; they may involve mutual transfer among tasks, as with functional transfer; or they may use both one way and mutual transfer. The transfer learning techniques developed in this thesis will make use of both functional and representaional transfer. As will be seen in chapters 4 and 5, the transfer learning methods developed will use functional transfer to provide a more robust basis for representational transfer.

However, neither of these methods has truly focused upon how to learn a task in such a way as to make that task more easily transferable when needed for some, as yet, unspecified new task. A learning method that modularizes learning should more readily lend itself to transfer learning, since the modules of learning will be more discrete and identifiable. Deep methods provide such an advantage. Of the methods presented above, EBNN, KBCC and Connectionist Glue make use of deep learning techniques for transfer. They all involve deep neural nets that are attached to each other, in toto, for the sake of learning a new task. None of these techniques attempt to use only part of a net for transfer. So, although these methods use deep learning techniques, they don't take full advantage of the available modularity.

Chapter 3

Review of Deep Learning Techniques

3.1 Chapter Overview

The focus of this thesis is developing transfer learning techniques for neural nets. Deep architectures offer significant opportunity for transfer learning. Their layered representations are expected to be very general and 'low-level' when close to the raw input and get progressively specific and 'high level' as one approaches the output layer. This expectation is one of the reasons behind DARPA's interest in Deep Learning [2].

This chapter will begin by explaining some of the general advantages of deep learning techniques over shallow learning techniques. It will then proceed to survey existing deep learning techniques. The main contribution of this thesis does not lie in development of a new deep learning technique, but rather in the adaptation of existing deep learning tecniques for neural nets, as displayed in deep architectures, for transfer learning.

3.2 Advantages of Deep Learning over Shallow Learning

Deep learning methods utilize many layers of learning modules. These modules take can take various forms from logical gate functions to nodes of neural nets. A learner that is shallow and broad has much greater opportunity for parallel calculations than one that is narrow and deep. Breadth carries with it a benefit in speed of calculation and creates a preference for broad, shallow learners. In fact, the Cybenko theorem states that a 3 layer feed forward neural net can model arbitrary decision regions to an arbitrarily high degree of accuracy [15]. So, why would one ever want to give up the potential speed of a broad, shallow net by using a deep net? The

answer lies in the need to balance the amount of time required for a calculation along with the amount of space (i.e. memory) required for a calculation.

A common example of this trade-off is provided by the parity function, PAR, for a set of n Boolean variables, $\{x_1, x_2, ..., x_n\}$. This function has a value of 1 when an odd number of x's are true and 0 otherwise. To construct a 2 layer shallow net, using modules that are only capable of the basic logical functions (i.e. AND, OR, and NOT), would require $O(2^n)$ such modules. Essentially, one node would be needed for each possible combination of inputs in order to determine whether an odd or even number of the $x's$ were true. This is because when calculating PAR($x_1, ..., x_n$) , a change in any x_k will change the value of PAR($x_1, ..., x_n$). So, in order to calculate this function in one step, one must individually consider every possible set of input values. However, the calculation would only need $O(1)$ time, since each of the $O(2^n)$ modules would calculate simultaneously, making it necessary just to perform an OR on their outputs.

It is possible to take advantage of the fact that PAR($x_1, ..., x_n$)= PAR(PAR($x_1, ..., x_{n-1}$), x_n). by using those same logical modules to mimic a 2 input XOR function (i.e. AND ($OR(x, y)$, NOT $(AND(x, y))$)). These XOR modules can be stacked to create a circuit of depth $O(n)$ with only $O(n)$ modules that calculates PAR($x_1, ..., x_n$). The first layer would calculate PAR(x_1, x_2), the second would calculate PAR(PAR(x_1, x_2), x_3), and so on, until PAR(PAR($x_1, ..., x_{n-1}$), x_n) was calculated. The time required for this calculation is $O(n)$, instead of $O(1)$. In essence, the deep circuit has paid for a reduction in the scaling of physical resources needed for calculation, from an exponential increase to a linear one, by accepting an increase in the scaling of temporal resources needed for calculation, from independence with respect to the problem size to a linear increase in required temporal resources with respect to the problem's size.

An additional reason to forgo the potential speed of a shallow net for the space saving of a deep net is the difficulty in fully realizing this time savings. Unless one is going to build an actual circuit to instantiate a neural net, it is difficult to gain the full speed benefits of the parallelism offered by shallow nets. Most conventional machines don't have the hardware to make effective use of a shallow net's parallelism. So, with fewer units, deep nets will perform fewer calculations

and may, therefore, still be faster than shallow ones on conventional machines.

Clark and Thornton [13] advocate deep learning approaches with a meta-learning argument that deep learners are more easily trained. They observe that certain statistical regularities are more readily noticed when a learner manages to appropriately 'recode' input data. For example, given a vector X, one can imagine creating a function $X' = G(X)$, where 'statistical regularities' are more easily observed from the X' representation than the X representation.

As an example of this, they describe a 2D parity problem where x_1 and x_2 can take the values $\{0,1,2,3\}$. The output vector, Y, takes values of 0 for even parity (i.e. $x_1 + x_2$ is even) and 1 for odd parity. It is clear that $P(Y = 0|x_1) = P(Y = 1|x_1) = 0.5 \; \forall \; x_1$ and that a similar statement could be made for x_2. So, neither variable independently gives any information concerning the parity of the sum. However, what if $G(X)$ returns the vector X' where $x'_1 = x_1 + x_2$? Then, $P(Y = 0|x'_1 = 0) = 1$, $P(Y = 1|x'_1 = 1) = 1$, $P(Y = 0|x'_1 = 2) = 1$, etc. It will clearly be easier to make accurate generalizations from x'_1, than by looking for the correct pattern with x_1 and x_2. In fact, Clark and Thornton [13], cite other researchers' results that demonstrate just how difficult it is for a standard neural net to learn the parity problem when generalization is required.

As useful as it is to have this new representation, it just raises the question of how to determine the function $G(x)$. Must it be hand coded or are there generalizable techniques for finding functions to provide optimal 're-coding' (to use Clark and Thornton's term) for the input data? Some generalizable techniques that are suggested for 're-coding' include an incremental learning approach, and a modular approach that involves reuse of previously learned features.

The incremental approach involves having a learner train on progressively more difficult concepts. The key thought behind this approach is that the simpler concepts will enable the learner to focus on the most significant statistical regularities and be less likely to incorrectly learn irrelevant, chance correlations. Then, once the simple task has been mastered, it will provide a useful bias for the learner if it is given a slightly more complex task to master. Clark and Thornton use Elman's results on grammar acquisition [17] as an illustration of the utility of training with progressively more complex tasks.

3.3 Layered Learning

Reuse of previously learned features for improved learning for some complex tasks has been incorporated into the stream-to-layers (STL), many layered learning technique created by Utgoff and Stracuzzi [55]. The initial motivation for this method came from the observation that for people to acquire certain concepts, it is, at times, necessary for them to have already mastered simpler ones. The purpose of STL is to create a mechanism by which a learner can organize concepts in terms of each other, as they are mastered.

When STL is implemented, it creates a learner capable of attacking several tasks simultaneously. As simpler tasks are mastered, they are recruited to help, if possible, in the learning of more complex tasks. This results in a deep learner, since many layers will be used to master a sufficiently large and complex set of related tasks. So, STL may be seen as taking advantage of the ability of deep learners to learn new representations (in the sense of Clark and Thornton [13]) for raw data to master an otherwise difficult task.

The SCALE algorithm was subsequently developed by Stracuzzi as an improvement to STL [51]. One of the weaknesses of STL is that **all** previously learned tasks are brought to bear when learning something new. As the learner acquires more knowledge, it will begin to experience a combinatorial explosion of concepts available to 'aid' it in learning a new one. This explosion makes the unmodified STL scale poorly.

SCALE alleviates this by restricting the data space and concept space from which the learner is allowed to form hypotheses. These restrictions are designed to result in hypotheses that are more readily discovered and rely upon fewer previously learned concepts. It's also an example of the general bias for practical hypotheses (or, perhaps, in this case, meta-hypotheses). The advantage claimed for SCALE over STL is not improved accuracy, but improved usability.

3.4 Cascade Correlation Techniques

Both cascade correlation [19] and knowledge based cascade correlation [38, 40, 42] were discussed earlier in section 2.2.3 in terms of transfer learning techniques. However, they are also deep net techniques that bear much in common with the layered learning approach. They each use previously mastered concepts as input to help in learning a new concept.

Knowledge based cascade correlation is, perhaps, a little more similar to the layered learning techniques discussed above, because it takes well-defined prior knowledge, in the form of pre-existing sub-nets that have already been trained for other tasks.

Regular cascade correlation, by contrast, determines when it needs to learn a supplemental task to aid the primary task, learns a supplemental task, in an unsupervised manner, and then attempts to use this newly learned task to help in learning its primary one.

The reappearance of these techniques in the 'deep net' section of this survey is intended to illustrate how readily deep net techniques lend themselves to knowledge transfer.

3.5 Deep Belief Nets and Autoencoders

Autoencoders and deep belief nets (DBN) are yet two more deep learning technique that rely upon modular learning. Both of these techniques train a deep neural net one layer at a time.

Deep belief nets (DBN), as introduced by Hinton et al. [25, 26, 27] , take a generative approach to learning in addition to a discriminative approach. A generative model attempts to learn a probability distribution for both observed variables and hidden variables. A discriminative approach merely attempts to learn the conditional probability distribution of the label variables, given the target variables. Two paradigmatic examples of these approaches are naive Bayes classifiers, which are a generative learning technique, and logistic regression classifiers, which are a discriminative technique. When applied to image recognition, a generative technique attempts to learn $P(image, label)$, the joint probability distribution over the domain of input images and labels, as opposed to merely learning $P(label|image)$, the probability distribution that a certain

output label is appropriate, given the input image.

Generally, it has been felt that discriminative learning techniques tend to perform better than generative techniques. This is commonly attributed to the fact that discriminative techniques attempt to learn only as much about a probability distribution as is necessary to assign most probable labels to data. This is reflected by the fact that the asymptotic error achievable by a discriminative classifier is superior to that of a generative classifier drawn from the same parametric family of models for an underlying probability distribution [34]. However, the generative learning technique will tend to achieve its asymptotic performance more quickly [34]. DBN's attempt to gain the best of both worlds by first taking a generative approach to learning and then fine tuning with a discriminative approach.

Because DBNs are descended from Boltzmann machines, it is necessary first to understand Boltzmann machines.

3.5.1 Boltzmann Machines

A Boltzmann machine is a neural net with two types of binary stochastic nodes - hidden and visible. A visible node corresponds either to external input data or to directly observable, visible information. Hidden nodes correspond to variables that are not directly observed and whose values must be inferred. These nodes are linked by symmetric connections (i.e. the direct effect of node A on node B is the same as the direct effect of node B on node A). Any node may be connected to any other node.

The state of a Boltzmann machine is characterized by the vector of values that describe its nodes. With this state, the following energy is associated:

$$E(\mathbf{s}) = -\sum_{i} a_i s_i - \sum_{i,j:i<j} s_i W_{ij} s_j \tag{3.1}$$

where,

$\mathbf{s}$ is the state vector for the Boltzmann machine

s_i represents the binary state, $\{0,+1\}$ of the i^{th} node.

$s_i W_{ij} s_j$ represents the contribution to the energy made by the interaction between the states of the i^{th} and j^{th} node

$a_i s_i$ represents the energy contribution solely due to the state of the i^{th} node.

Both the a_i and W_{ij} terms will be referred to as weights of the machine. Additionally, the weights are symmetric (i.e. $W_{ij} = W_{ji}$). So, the second summation term only needs to account once for each pair of nodes.

These machines get their name from the fact that the probability of their being in any given state is determined by the Boltzmann probability distribution. So, letting S represent the set of all possible $\mathbf{s}$ vectors,

$$P(\mathbf{s}) = e^{-E(\mathbf{s})} / \sum_{\mathbf{s} \in S} e^{-E(\mathbf{s})} \tag{3.2}$$

Because each node can only be in the discrete states $\{0,1\}$, they have the following probabilities of being in either state:

$$\forall j \neq i, p(s_i = 1 | s_j) = \frac{1}{1 + e^{-a_i - \sum_{j \neq i} W_{ij} s_j}} \tag{3.3}$$

$$\forall j \neq i, p(s_i = 0 | s_j) = 1 - \frac{1}{1 + e^{-a_i - \sum_{j \neq i} W_{ij} s_j}} \tag{3.4}$$

The goal in training a Boltzmann machine is to adjust the weights so that the probability distribution of the training data vectors will be equal to the equilibrium state probability distribution of the visible nodes.

In other words, if at time $t = 0$, each node in the Boltzmann machine were randomly initialized and every Δt seconds, each node were to redetermine its state according to the above probabilities, then the Boltzmann machine would eventually achieve a steady state. The goal of training is to find the set of weights such that the steady state probability of generating any particular set of visible nodes is the same as finding the corresponding data vector in the training set. This makes Boltzmann machines a generative technique, as previously stated.

If we consider V to represent the set of possible $\mathbf{v}$ vectors for the visible nodes and H to represent the set of possible $\mathbf{h}$ vectors for the hidden nodes then the marginal probability for a given visible vector, $\mathbf{v}$, being generated by the Boltzmann machine is

$$p(\mathbf{v}) = \sum_{\mathbf{h} \in H} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{s} \in S} e^{-E(\mathbf{s})}} \tag{3.5}$$

It is easiest to maximize this probability by maximizing its log with respect to the weights. This is done with a gradient ascent technique. The gradient of the log probability is composed of two terms. The first term comes from the numerator of the probability and involves a sum over all possible states for the **hidden** nodes when the visible nodes are set to correspond to the data vector. The second term comes from the denominator of the probability and involves a sum over all possible states for **all** nodes. This results in the gradient of the log probability taking the following form, where *Data* refers to moments for the distribution of actual data and *Model* refers to moments from the distribution of the free running equilibrium distribution of the Boltzmann machine :

$$\forall i \neq j, (\frac{\partial log(p(\mathbf{v}))}{\partial W_{s_i,s_j}})_{Data} = \langle s_i s_j \rangle_{Data} - \langle s_i s_j \rangle_{Model} \tag{3.6}$$

$$(\frac{\partial log(p(\mathbf{v}))}{\partial a_{s_i}})_{Data} = \langle s_i \rangle_{Data} - \langle s_i \rangle_{Model} \tag{3.7}$$

In order to determine both the $Data$ and $Model$ terms, the Boltzmann machine must be run in two phases, commonly referred to as the **positive** and **negative** phases. In the positive phase, the visible units are clamped to the values that correspond to a particular data vector and the machine is allowed to run freely until it reaches an equilibrium state. At that point, one estimate $\langle s_i s_j \rangle_{Data}$ by sampling. In the negative phase, no units are clamped and the machine is allowed to run freely until it reaches an equilibrium. Similarly, at that point, one can estimate $\langle s_i s_j \rangle_{Model}$ by sampling. Then, gradient ascent can be used to modify weights in the usual manner.

Heuristically, one may think of this technique as making the probability distribution of the model match that of the data by forcing the moments of the model distribution to match the moments of the data distribution. A probability distribution can be characterized by its moments. If

two probability distributions share all the same moments, then they are identical. Equation 3.7 tries to minimize the difference between the first moment of the probability distribution of states for the Boltzmann machine and the probability distribution of the external data. If the interaction matrix, W, allowed for self-interaction, then Equation 3.6 would minimize the difference between the all the components of the second moments of those two distributions, instead of almost all of them.

The main problem with Boltzmann machines is that they can take an exceedingly long time to train. One way to alleviate this is a restricted Boltzmann Machine (RBM).

3.5.2 Restricted Boltzmann Machines

In a restricted Boltzmann machine, only symmetric connections between visible nodes and hidden nodes are allowed. Visible nodes do not directly affect each other, and hidden nodes do not directly affect each other. This results in a machine that can be represented with a bipartite graph. It also results in the following simplification of the gradient of the log probability:

$$(\frac{\partial log(p(\mathbf{v}))}{\partial W_{v_i,h_j}})_{Data} = \langle v_i h_j \rangle_{Data} - \langle v_i h_j \rangle_{Model} \tag{3.8}$$

$$(\frac{\partial log(p(\mathbf{v}))}{\partial a_{v_i}})_{Data} = \langle v_i \rangle_{Data} - \langle v_i \rangle_{Model} \tag{3.9}$$

It should be noted that now, that the visible nodes are conditionally independent of each other, given the hidden nodes, and that the hidden nodes are conditionally independent of each other given the visible nodes. So, whereas in the earlier gradient expressions, interactions between all $\binom{(V+H)}{2}$ pairs of nodes needed to be considered, now only interactions between $V \times H$ pairs of nodes need to be considered. This reduction in equations results in a significant speed up in training. Furthermore, because the visible nodes are conditionally independent of each other, given the hidden nodes (and vice-versa), each set of nodes may be updated in parallel, using the following formulae:

$$p(v_i|\mathbf{h}) = \sum_{\mathbf{h}\in H} \frac{e^{-E(v_i,\mathbf{h})}}{\sum_{\mathbf{h}\in H} e^{-E(v_i,\mathbf{h})}} \tag{3.10}$$

$$p(h_j|\mathbf{v}) = \sum_{\mathbf{v}\in V} \frac{e^{-E(\mathbf{v},h_j)}}{\sum_{\mathbf{v}\in V} e^{-E(\mathbf{v},h_j)}} \tag{3.11}$$

This leads to an alternation between updating the visible nodes and then updating the hidden nodes as the RBM seeks equilibrium.

3.5.3 Deep Belief Nets

Deep belief nets may be thought of as sequentially-trained, stacked RBMs. The first RBM consists of a layer of visible nodes, which contain the input data, and a layer of hidden nodes. Although, the number of connections between nodes has been reduced drastically by the use of the RBM, training can still be an onerous process. This is because the terms in equations 3.8 and 3.9 both involve averages.

Direct calculation of these averages involves some unwieldy summations over all possible node configurations, so these term are estimated by sampling. Using sampling for the first term in each equation is easy. One merely assumes that the distribution of the training data accurately reflects the actual data likely to be encountered, then uses sampling from that data to determine $\langle v_i \rangle_{Data}$ and $\langle v_i h_j \rangle_{Data}$.

Finding $\langle v_i \rangle_{Model}$ and $\langle v_i h_j \rangle_{Model}$ is more difficult, because one does not have direct access to the model's underlying probability distribution. So, in order to approximate sampling from this distribution, one samples from the data distribution to initialize the visible nodes. Then the machine is allowed to run for several iterations with the visible nodes being used to update the hidden nodes and then the hidden nodes being allowed to update the visible nodes, until the RBM is judged to have achieved an equilibrium state. Since this equilibrium is only achieved asymptotically, strictly speaking, one would need to let the RBM run for an infinite number of iterations to realize it. Yet, generally around 100 iterations will give a sufficiently accurate approximation to the equilibrium distribution [26].

It is possible to speed this process up significantly by only allowing the RBM to run for one iteration before sampling to determine the averages under the model's distribution. Although this is not a terribly good approximation of the model's distribution, the process of trying to minimize the difference between the data's distribution and the RBM's distribution after being allowed to run for only one cycle does cause the ultimate RBM distribution (i.e. the approximation of the model's distribution) to approach the data's distribution. This now transforms equations 3.8 and 3.9 to

$$(\frac{\partial log(p(\mathbf{v}))}{\partial W_{v_i,h_j}})_{Data} \approx \langle v_i h_j \rangle_{Data} - \langle v_i h_j \rangle_1 = 0 \tag{3.12}$$

$$(\frac{\partial log(p(\mathbf{v}))}{\partial a_{v_i}})_{Data} \approx \langle v_i \rangle_{Data} - \langle v_i \rangle_1 = 0 \tag{3.13}$$

This approximation gives sufficient accuracy while being far more practical than the exact solution.

With this method, the RBM's that constitute the DBN may be trained sequentially. This training starts at the layer that interacts directly with the external input. Once the base level RBM has been trained, the original hidden layer becomes the visible layer for the next level RBM. Its values are probabilistically determined by data in the the original input layer. Training of this RBM then proceeds in the same manner as for the previous machine. The goal of this training is to create a deep machine that generates data with the same probability distribution as the actual data. As long as each new RBM added to the stack has at least as many hidden nodes as the previous layer, the lower bound of the difference between the actual probability distribution of the data generated by the DBN will decrease.

Additional discriminative training occurs when the lower level of the uppermost RBM is modified by adding extra nodes to correspond to labels for the data. When the top most RBM is trained, all the nodes of the lower level are generated as the end result of a bottom up chain of probabilistically determined nodes from lower layers, except for the extra label nodes. Their values are set to the desired label for the original input data.

Once the entire DBN has been constructed, it is fine tuned by allowing the connections be-

tween nodes to cease being symmetric (i.e. the effect of node A on node B is no longer required to be the same as the effect of node B on node A), except for those at the very top RBM. The weights at the very top RBM remain symmetric and are not adjusted. The weights that go from a lower layer (i.e. closer to the original data) to a higher layer will be referred to as 'recognition' weights, since they are used by the DBN to convert raw input into a corresponding internal label. Weights that go from a higher layer to a lower one will be referred to as 'generative' weights, since they will be used by the DBN to convert an internal label back into a Platonic member of the class of raw data corresponding to that label (i.e. given the label, they 'generate' a possible input)

The fine tuning has two phases, an 'up-pass' and a 'down-pass'. In the up-pass, the visible nodes at the lowest layer of the DBN will have their values set to correspond to a member of the set of training data. Then, the recognition weights will be used to probabilistically set the state for each higher layer. Once this has happened, the generative weights will be adjusted so as to increase the probability of each layer being generated by the layer below.

The down-pass is consistently carried out in a reversed fashion. The only difference is in how the top level is initialized. The preferred method is to let it be initialized during the up-pass and then allow the top most RBM to run through a few cycles before allowing the down-pass to proceed. At this point, the generative weights are used to probabilistically set the state for each succeeding, lower layer. Then the recognition weights are adjusted to increase the probability of each layer being generated by the layer above.

Hinton et al. [26, 27] used this technique, to learn to recognize the set of handwritten numbers. They achieved an error rate of 1.25%, which is notably better than any technique which neither uses prior knowledge nor augments the training set with additional geometrically distorted numbers from the training set. Other techniques, that operate with these same limitations, achieve error rates ranging from 1.4% for an SVM, to 1.51% for a neural net, to 2.8% for a nearest neighbor technique [27] .

3.5.4 Autoencoders

Autoencoders [25, 26, 39] are exactly like deep belief nets, except for the fact that their aim is to reduce the dimensionality of the data. They do this by steadily reducing the number of hidden nodes in each RBM that gets added to the stack. Then, for fine tuning, instead of the up-pass/down-pass technique, the stacked RBMs are unfolded to form a single feed-forward net. This net is then trained with a standard backprop method, where the error term is the difference between the output and the input.

The result is a two-part machine. The bottom part, which can be thought of as an encoder, takes raw input and compresses it into an encoded form as expressed by the hidden layer of the top-most RBM. The upper part, which can be thought of as a decoder, takes the compressed output of the encoder and reproduces the original input.

By training their stacks of RBMs one at a time, both DBNs and autoencoders use a modular approach towards recoding in the sense described by Clark and Thornton [13].

3.5.5 Convolutional Neural Nets

The last deep learning method to be examined will be convolutional neural nets (CNN). In addition to using a deep learning method, these feed-forward nets use architectural restrictions to limit the hypothesis space to be searched. CNN's limit themselves to hypotheses that display [29]:

1. Shift Invariance
2. Moderate Insensitivity to Rotations
3. Moderate Insensitivity to Geometric Distortions
4. Small Scale, Local Correlations

These limitations are imposed by placing several types of restrictions on the connections nodes are allowed to form. Firstly, each node has a locally restricted input region. Rather than connecting each node to all the nodes in the layer beneath it, a node is only given connections to a small adjacent grouping of nodes in the layer beneath it. The effect of this is to cause the net to seek out only local correlations in the layer beneath it to use as significant features.

Furthermore, the layers of the net are divided into several feature maps (See Figure 3.1). Nodes in the same feature map have the same set of input weights. So, they all react to the same set of features. Additionally, the spatial relationship between two nodes in a given feature map will be replicated in the receptive fields associated with those nodes in the layer below. It should be noted that a given node may have receptive fields in several feature maps in the layer immediately below it. When this happens, all nodes in the same feature map as the given node will have receptive fields in the same set of feature maps from the layer below. Finally, the spatial relationships among these receptive fields will be the same in each of the lower feature maps. Therefore, each such map will detect and locate a given feature or combination of features from the previous layer.

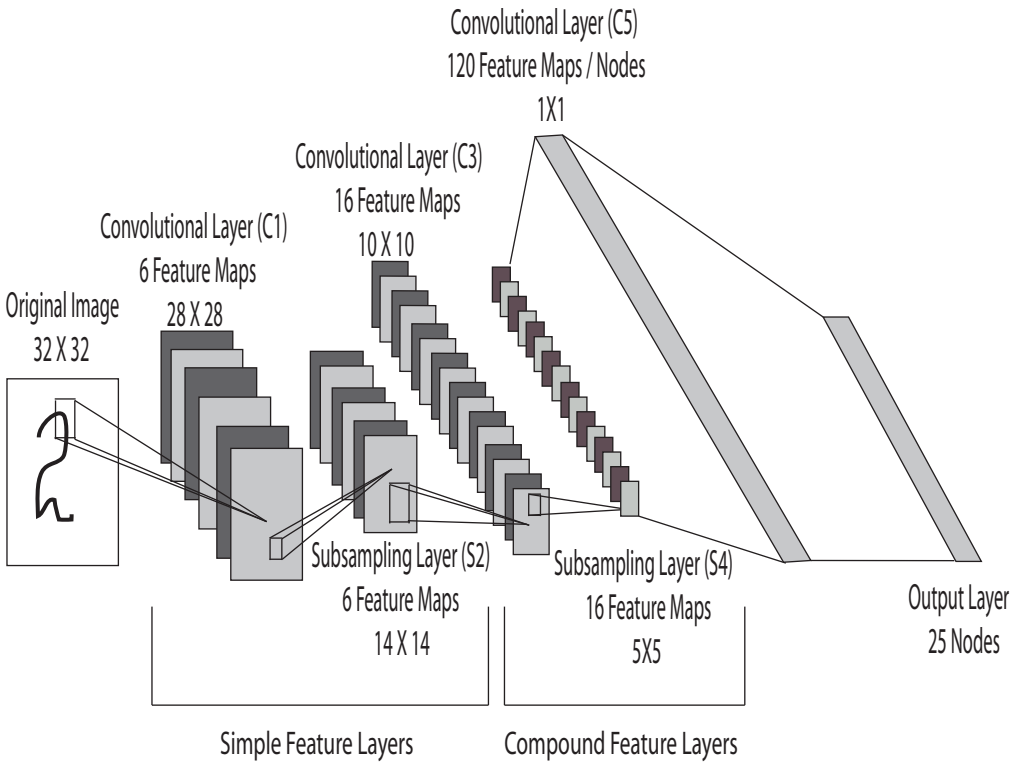

Figure 3.1: Example of convolutional neural net. This example is based on LeNet5, as first used by LeCun et al. [29]. One should note the presence of local receptive fields and several feature maps in each layer.

There are two standard types of layers found in a CNN: convolutional and sub-sampling layers. In the convolutional layers (C-layers) of the net, adjacent nodes will have overlapping receptive fields with the same relative spatial positions. So, if the input feature maps have dimensions of $X \times Y$ and the size of the receptive field is $I \times J$, then the size of the feature maps in the convolutional layer is $(X - I + 1) \times (Y - J + 1)$. This is because one can only fit $(X-I+1)(Y-J+1)$ receptive fields of size $I \times J$ on an $X \times Y$ grid, when those receptive fields are used to cover that grid in a convolutional fashion - as if one were performing the convolution of an $I \times J$ kernel over a discrete $X \times Y$ grid.

Due to all these restrictions, the calculation of a convolutional node's responses is slightly more complicated than calculating the responses of a standard node in a neural net. Normally, the output of a given node is a non-linear function of a weighted sum of the inputs to that node. The inputs are generally the outputs of other nodes or raw data. The weights normally depend only upon the given node and the source of a specific input. A convolutional node, as stated before, has restrictions concerning from which nodes it may take input and the weights it's allowed to assign to those inputs. To gain a better understanding of how this would work, let's consider the following example:

Assume that a given feature map, m, in a convolutional layer receives input from a set of feature maps, $C(m)$. This set may consist of raw input data or at least one feature map from the prior layer of the net. Let us also assume that the receptive field of nodes in m is I columns by J rows. Then, for feature map k, $k \in C(m)$, let $n^k_{x,y}$ represent the output of a node (or raw data element) from the (x, y) position in feature map k. Also, let $w^{k,m}_{i,j}$ describe the weight attached to the signal from a node (or raw data element) , associated with the location (i, j) of the receptive field on feature map k, for the input to node, $n^m_{x,y}$. Then, the total input to node $n^m_{x,y}$ is

$$TotalInput(n^m_{x,y}) = \sum_{k \in C(m)} \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} w^{k,m}_{i,j} n^k_{x+i,y+j} + w^{0,m} \tag{3.14}$$

where $w^{0,m}$ is a constant bias term. Here, it can be seen that each node of feature map m uses the same set of weights for the input it receives from its receptive field in feature map $k \in C(m)$.

In the other type of layer, a sub-sampling layer (S-layer), each feature map is connected to exactly one feature map of the prior layer. Additionally, the receptive fields of nodes in a sub-sampling map do not overlap. So, if we assume the input feature maps have dimensions of $X \times Y$ and the size of the receptive field is $I \times J$, then the size of the feature maps in the sub-sampling layer is $(XdivI) \times (YdivJ)$. Furthermore, there is only one multiplicative free parameter and one additive free parameter associated with the input from the receptive field. So, the total input to a node $n^m_{x,y}$ in the (x, y) position in a sub-sampling feature map m is:

$$TotalInput(n^m_{x,y}) = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} w^{1,m} n^1_{Ix+i,Jy+j} + w^{0,m} \tag{3.15}$$

where, $w^{1,m}$ represents the weights assigned to all the inputs from the specific input feature map for the m^{th} feature map.

The response function for nodes in either a convolutional layer or a sub-sampling layer is:

$$n^m_{x,y} = A \times tanh(S \times TotalInput(n^m_{x,y})) \tag{3.16}$$

where A = 1.7159 and S = 2/3. These numbers were chosen to optimize the training behavior of the net. Since, ultimately, nodes would be trained to have responses of +/- 1, it was deemed useful to let those values be fixed points of the response function (i.e. f(1) = 1 and f(-1) = -1). It also kept the gain close to 1 near the fixed points, so that the these fixed points were neither too attractive, nor too repellent, which made for easier training [29].

Another important aspect of the architecture of CNN's is that it forces the early layers to act as feature extractors. The convolutional layers will reflect the presence of a particular local feature, or local combination of features, wherever they occur in maps in the prior layer. Whereas, the sub-sampling layers give CNNs a decreased sensitivity to minor rotations and distortions of an image. This helps make them robust with respect to unimportant variations. Furthermore, because the net has such a modular structure by virtue of having both many layers and several feature maps in each layer, it is an ideal candidate for knowledge transfer investigations.

It is, in fact, this modular structure will be the basis of the knowledge transfer experiments discussed in the next chapter. The transfer technique that will be explored is referred to as *structural transfer*, because it relies upon the structure of a CNN in order to transfer knowledge obtained in one set of tasks in order to aid learning another set of tasks.

Chapter 4

Structurally Based Knowledge Transfer

4.1 Chapter Overview

In this chapter, the first of the two techniques developed for transfer learning will be introduced: Structurally Based Knowledge Transfer (SBKT). The work detailed has been published in [21, 22, 23].

4.1.1 Motivation For Structurally Based Knowledge Transfer

One of the key issues in any attempt to aid learning through knowledge transfer is determining just what should be transferred. Sometimes, this is easy. For instance, when learning Spanish, it's advantageous to have already mastered Italian. The words, rhythms and grammar of the two languages are extremely similar. However, knowledge of circus arts, such as juggling, tight rope walking and sword swallowing, would be of scant use when learning Spanish.

When attempting to use transfer learning techniques with neural nets, this problem is exacerbated by the fact that it's rather hard to understand the particular contribution of a given hidden node to a given problem. As a result, representational transfer learning techniques with neural nets have relied upon exposing the net to the new class of input before deciding just what should be transferred. Examples of this are Schultz and Rivest's knowledge based cascade correlation [41, 42], which judges whether the output of an entire trained net would be useful to help a new neural net master a task, and Pratt's discriminability based transfer technique [36], which judges the expected relevance of each hidden node of an existing neural net to a new task.

Both of these techniques employ relatively large training sets to determine the utility of any

net or node to a new problem. However, it is not always easy or even possible to obtain large sets of labeled data. Additionally, if one were to envision a learner that was interacting with the physical world, one would prefer for that learner to be able to initiate and accomplish learning on the basis of a single experience. The odds of long-term survival are enhanced if one only needs to witness a single tiger attack in order to recognize and avoid tigers.

The experiments described in this chapter use a convolutional neural net (CNN), as developed by LeCun et al [29]. The motivation for choosing this net is the fact that it is a deep net.

The reader should recall from section 3.2 that part of Clark and Thornton's argument [13] for deep learners was the ability to 'recode' data so as to better identify significant statistical regularities. These recodings occur in the shallower layers of the learner. Given this situation, it seemed likely that if a learner were trained to discriminate among several members of a subset of a set of classes, then the statistical regularities that were effective to discriminate among those classes would also be useful to discriminate among other classes from the main set. This observation was the inspiration for structurally based knowledge transfer (SBKT). If a deep neural net were trained to perform a set of related tasks, then it seems likely that a Clark-Thornton style recoding occurs and that this recoding captures similarities shared by all members of the set of tasks. If there are other tasks, in the same task set, which have not been learned, then it should be possible to use the input recoding in the lower layers to more easily learn new tasks from that same task set.

The decision of which internal parameters to use for transfer learning is, therefore, made using only the structure of the net. Nodes that are sufficiently close to the input data will have their weights preserved and so transfered to the new task. Nodes that are not will have their weights reinitialized and be retrainined for the new task.

4.1.2 Summary of Structurally Based Knowledge Transfer Experiments

In order to test whether this idea was correct and to determine where a good boundary layer separating the transfered and reinitialized nodes lay, I performed a set of experiments measuring

the efficacy of using an SBKT method to help a CNN learn to discriminate among a subset of the set of standard Western alpha-numeric characters. The images of these characters that I used came from NIST Special Database 19 [20], which is comprised of several samples of handwritten characters from several 'writers'. Some samples from this dataset are given in Figure 4.1

Figure 4.1: Sample characters from the NIST Special Database 19.

The first set of experiments performed involved training a CNN to discriminate among a subset of the set of standard Western alpha-numeric characters. Then, lower layers of the net (i.e. those closer to the initial input) were frozen and higher layers of the net were reinitialized. Finally, the freshly reinitialized nodes were retrained to enable the net to discriminate among different characters of the Western alpha-numeric characters. Comparisons were made among the various accuracies obtained for small training sets as the boundary was varied between the layers whose knowledge was transferred and those that were retrained.

I also performed a second set of experiments, using the same methodology, with the additional feature of adding background noise to the data images. The goal here was not merely to observe the efficacy of this technique in the presence of noise. Instead, the goal was to make a point about transfer learning that frequently seems to be overlooked. Much of the literature spends time discussing the importance of finding relevant features to transfer in the active sense. Equally, and perhaps more important is filtering through features to be ignored.

Sensory data contains a plethora of information. In general very little of it is relevant to any specific task. Consider the standard problem of face recognition. Most images of faces, unless they have been carefully scaled and cropped, contain many background features that are totally unrelated to the actual face one wants to recognize. In addition, there may be shadows on the face that are also irrelevant to the problem of recognition. All of these factors come into play before even considering variations in pose and expression, which may actually contain informa-

tion relevant to recognizing a person. Since the irrelevant features in an image are frequently so plentiful, being able to filter them out of the decision process is fairly important. So, in order to demonstrate the increased utility of transfer learning when there is significant variation in the raw data, that is irrelevant to the given task, experiments were performed to observe the benefits of transfer learning when learning from noisy data.

The importance of recognizing what is irrelevant to a given problem was demonstrated by Viola and Jones [56]. They did this by developing a technique for face detection that concentrated on learning to recognize characteristics of image regions that eliminated them from consideration of displaying a face. Then, in true Sherlock Holmes style, once they had "eliminated everything that was impossible, anything that remained, however improbable, is the truth.". In the case of Viola and Jones, this meant once they had eliminated all regions which they knew did not contain a face, any faces would lie in the remaining regions.

4.2 Noise-free Structurally Based Knowledge Transfer

The neural net architecture used for these experiments was a CNN that was closely based upon the LeNet5 architecture used by LeCun et al. [29]. For these experiments, the most important aspect of this architecture is the fact that it is deep.

4.2.1 Experiments

In order to examine transfer learning, I first found a family of problems that were amenable to transfer. The chosen family of problems was recognition of handwritten characters. The variations in individuals' handwriting both among different people and among different instances of the same individual writing the same character are sufficiently great to make this class of problems well suited for our purposes. Furthermore, the complete set of distinguishing features should be learnable from a reasonably sized subset of characters. This should greatly facilitate the use of knowledge transfer from a subset of characters that have already been learned to a new set of characters that one wants to learn.

Acquisition of transferable knowledge was achieved by first training the net to recognize 20 classes of handwritten characters from NIST Special Dataset 19. The net was initially trained with about 400 samples per character of the characters 'I', 'Q' through 'Y' and 'a' through 'i'. The target output for each class was a randomly assigned a 20 bit random vector. It was felt that since the average inter-class distance between these vectors would be greater than the average inter-class distances among classes with with standard 1 of N coded targets, the random-bit encoding should result in improved performance for the net. Later experiments revealed that this is not a particularly large effect. The recognition accuracy achieved was slightly over 94%. Although it is possible to achieve higher accuracies, it was felt that this accuracy was sufficiently high for our purposes. Because this net acts as the 'source' of the knowledge we want to transfer, it will be referred to as the 'source net'. The net to which knowledge is being transferred will be referred to as the 'target net'. Both nets will have the same architecture.

Once in possession of the trained source net, the next task was to attempt transfer learning. Following in the footsteps of Pratt's discriminability based transfer (DBT) [36], this transfer was effected by directly copying nodes from the source net to the target net. However, whereas DBT needs to expose the transferred nodes to the new dataset, in order to identify both the ones with meaningful reactions that should be preserved and the ones that should have their weights randomly reinitialized for training, structurally based knowledge transfer (SBKT) does not use such exposure. SBKT has an absolute bias that nodes at, or below, a given layer possess transferable knowledge and nodes above that given layer do not. So, it will only transfer nodes at, or below, that level and limit training to a set of reinitialized nodes (i.e. all the nodes) above that layer.

To gain an initial appreciation of how best to utilize SBKT, the bottom n layers of the source net were copied over to the target net, where $0 \leq n \leq 5$. Transferred weights were kept fixed. Experiments were run beginning with $n = 5$ and culminating with $n = 0$. This last scenario corresponds to the case of no transfer learning and serves as a useful benchmark. $n = 6$ was not attempted, since this would correspond to transferring all the weights from the source net and then not allowing any training. For obvious reasons, this was not done. For less obvious

$Layer$	$Free\ Parameters$	$\%of\ Total$	$CumulativeFree\ Parameters$	$Cumulative\%\ FreeParameters$
F6	2,420	4.63	2,420	4.63
C5	48,120	92.09	50,540	96.72
S4	32	0.06	50,572	96.78
C3	1,516	2.90	52,088	99.68
S2	12	0.02	52,100	99.70
C1	156	0.30	52,256	100.00

Table 4.1: Number of free parameters associated with each layer of the net and cumulative number of free parameters for each layer and all layers above it. When 5 layers are retained for transfer learning, only the layer above the C5 level is allowed to retrain. This corresponds to 2,420 free parameters, or 4.63% of the total parameters of the net.

reasons, which will be explained in the next chapter, that was an unfortunate decision. It will be explained, in the next chapter, how to usefully transfer knowledge without the need for any additional training, if one keeps the entire net.

For each value of n, 5 learning trials were performed with training set sizes of 1, 5, 10, 20 and 40 samples per class. The testing sets had 1,000 characters (i.e. 50 samples per class). As the value of n decreased, the net gained more free parameters, thus increasing its capacity. However, this increase in number of free parameters is very sharply spiked at $n = 5$, as shown in Table 4.1. When $n = 5$, the net can only train with about 4.6% of the free parameters it normally has.Yet, when $n = 4$, the net has about 96.7% of its weights available to be trained. So, the transition from relying on transfer to relying upon training was not a smooth and gradual one.

A set of summarized results as given in Figure 4.2 shows that structurally based knowledge transfer (SBKT) is an effective means of transfer learning. Although the benefits decrease as the size of the training sets increase. This is not unexpected. Larger training sets are able to convey more information than smaller ones and may, therefore, convey some of the same knowledge that

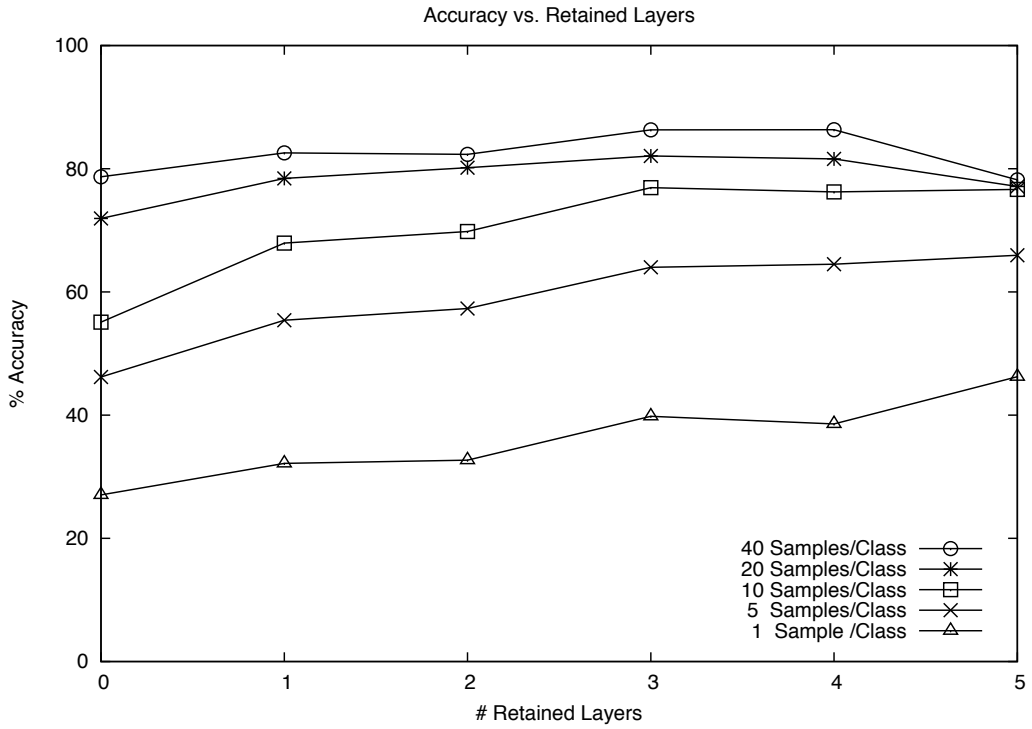

Figure 4.2: Comparison of learning curves showing accuracy vs. number of retained layers for various numbers of samples per class in the training set. Curves show results, from top to bottom, for 40, 20, 10, 5 and 1 samples per class. Each point on a curve represents the average of 5 trials on a testing set with 1,000 samples per class

is being transferred. Furthermore, this graph also shows more information is being transferred by the C-layers, than by the S-layers. Although it is reasonable to assume that this is because the C-layers are more involved with detection of actual features, whereas the S-layers serve mainly to smooth out noise, one cannot rule out the more prosaic answer that the C-layers simply have many more free parameters than the S-layers. Lastly, it is hard not to notice that retaining 5 layers, as opposed to 4, gives a large advantage for the smallest of training sets, a modest advantage for the next larger datasets and is actually deleterious for the largest of training sets.

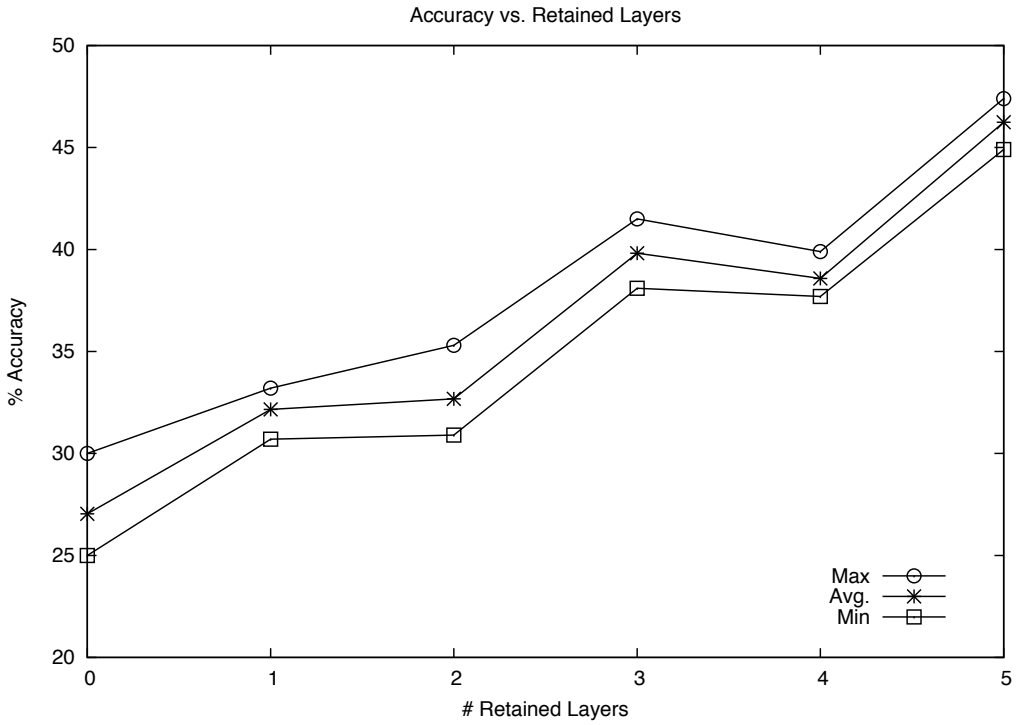

Figure 4.3: Comparison of learning curves showing accuracy vs. number of retained layers for 1 sample per class in the training set. Curves show the minimum, average and maximum accuracies achieved over 5 trials on a testing set with 1,000 samples per class

Examination of Figures 4.3, 4.4 and 4.5 shows, more readily than Figure 4.2, the magnitude of the benefit that SBKT can achieve for each training set size. These figures also highlight the changing shape of the learning curves as the increase in training set size lessens the relative importance of transfer learning to net capacity (i.e. number of free weights).

One question that this set of experiments did not investigate was how to judge the quality of the training set used for transfer. The issue to be considered is how completely the information to be transferred covers the information needed for mastering the new task. For example, Spanish

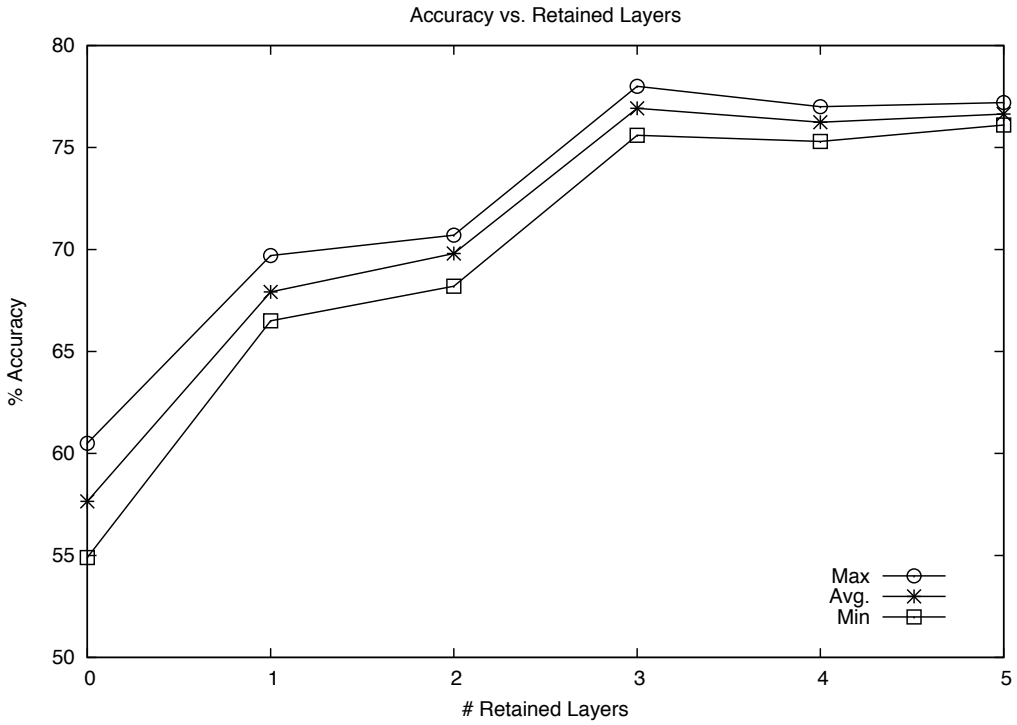

Figure 4.4: Comparison of learning curves showing accuracy vs. number of retained layers for 10 samples per class in the training set. Curves show the minimum, average and maximum accuracies achieved over 5 trials on a testing set with 1,000 samples per class

and Italian are very similar languages. A speaker of either language is familiar with most of the phonemes of the other. However, neither language contains the click consonants of the !Kung language. Although knowledge of one spoken language is helpful when learning another spoken language, if the target language has phonemes that are missing from the source language then new phonemes will need to be learned.

An analogous situation for character recognition could involve something like learning to differentiate among the characters ‘X’, ‘K’, ‘A’, ‘T’ and ‘E’ and then using this skill to help learn

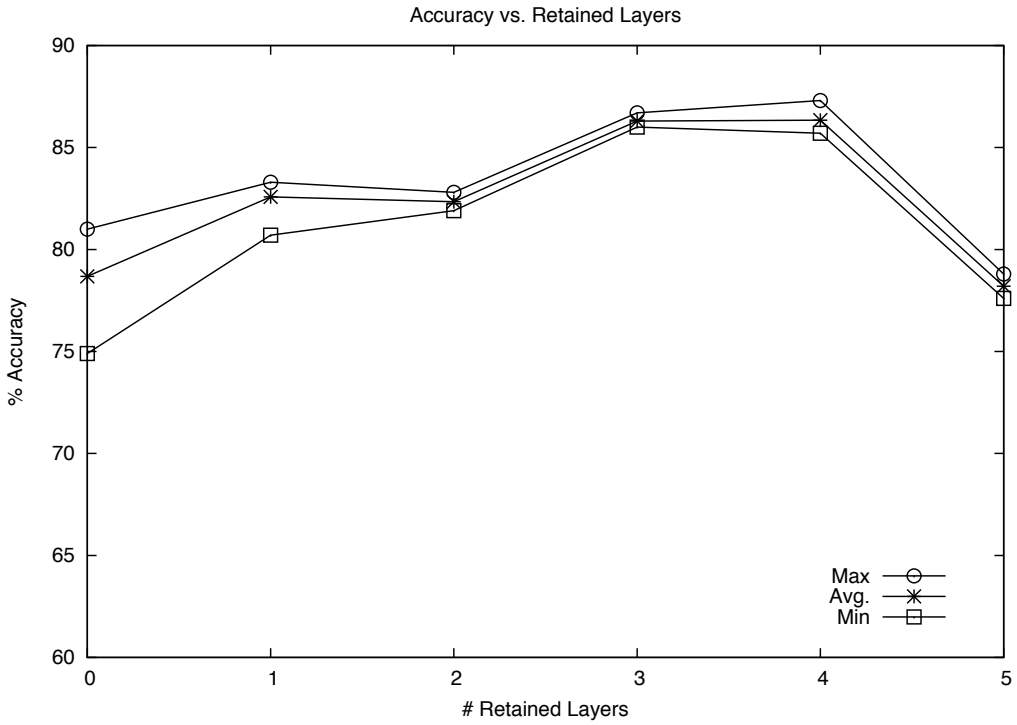

Figure 4.5: Comparison of learning curves showing accuracy vs. number of retained layers for 40 samples per class in the training set. Curves show the minimum, average and maximum accuracies achieved over 5 trials on a testing set with 1,000 samples per class

to differentiate among ‘C’, ‘O’, ‘G’, ‘S’ and ‘J’. The first set gives no exposure to characters with any degree of curvature. This lack will need to be overcome without the benefit of transfer. In order to ensure a wide variety of learned features to transfer, Baxter, one of the early pioneers of using transfer learning in neural nets, chose to demonstrate his technique using machine printed Japanese kanji. This way he could use 400 different character classes to create a robust internal representation using knowledge transfer [4, 5]. Furthermore, since the characters used were machine printed, there should have been much less intra-class variation. So, if our source net

had been better trained or possessed the ability to discriminate among a larger set of characters, it is reasonable to believe that greater benefits from transfer learning would have been observed. Additionally, techniques like Thrun and O'Sullivan's task clustering algorithm [54] could be used to help determine an optimal source set to be used to help learn a target problem.

Nevertheless, even with a relatively small number of classes in the source training set, structurally based knowledge transfer can achieve significant benefits when used to learn new tasks that possess sufficient relevance to the older ones.

4.3 Structurally Based Transfer Learning in the Presence of Background Noise

The point of this second set of experiments was not to examine the robustness of SBKT in the presence of background noise, but instead to highlight its increased utility in the presence of background noise. The goal of this set of experiments is to emphasize the benefits of transfer learning when much of the raw input data is irrelevant to the task being learned. For instance, when learning to drive a car with an automatic transmission, one quickly learns to pay attention to the responsiveness of the steering wheel, brake pedal and accelerator. The responsiveness of the tuning and volume controls of a radio are irrelevant to driving the car (unless of course the radio is tuned to a punk rock station - that must be changed immediately). When one then tries to transfer the skills needed to drive a car with an automatic transmission to driving one with a manual transmission, the steering wheel, brake and accelerator pedals retain their relevance. Similarly, the radio controls retain their irrelevance. Any learner that knows the radio is irrelevant to driving a car with a manual transmission will not need to waste any time determining this for itself. So, when a learner is presented with a data source that is rich in irrelevant information to the task at hand, the benefits of a transfer learning technique will be enhanced.

Unfortunately, the inner workings of neural nets are difficult to decipher. Although there are many ways to train a neural net to master a given task and to verify its mastery, there are none to

make the method it uses intelligible for humans. As hard as it is to determine where a neural net is focusing its attention, it should be just as hard, if not harder, to determine what it is actively ignoring.

In order to show that a learner can filter irrelevant information, the experiments that will be discussed present a learner with a great deal of irrelevant information with respect to the given task. The greater the degree of irrelevant information, the greater the need to ignore it. If transfer learning techniques also help learners avoid wasting their time considering irrelevant data, then the benefits of transfer learning should increase as a learner is forced to sift through more information in order to find the relevant portions. However, there will be a price to pay for this approach. Because the source net will be forced to learn from a noisy data set, it will be harder to acquire information to transfer.

Bearing these caveats in mind, the same source and target character sets as before were used. However, this time, three versions of these sets were used - a 0% noise set, a 10% noise set and a 20% noise set. Because the images used are binary images of white characters on a black background, noise was introduced by randomly turning background pixels from black to white. For the 10% noise set, background pixels had a 10% chance of changing and for the 20% noise set, there was a 20% chance. The background pixels in the 0% noise set were not changed.

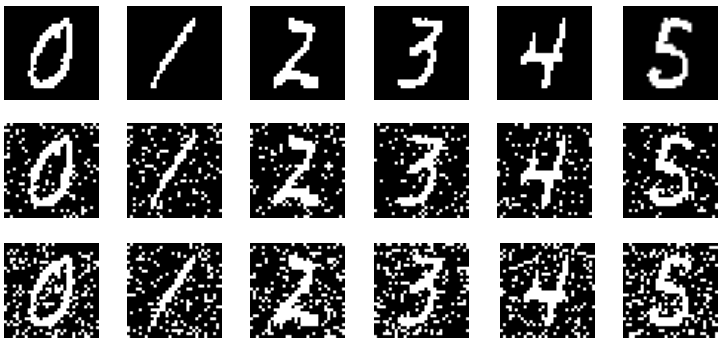

Figure 4.6: Comparison of data with no noise added, 10% noise added and 20% noise added.

Summarized results for the three sets of experiments are shown in Figures 4.7, 4.8 and 4.9. A cursory comparison of these three charts confirms the expectation that as the data gets noisier, learning becomes more difficult. One may also observe that, once again, the most significant

increases in accuracy are associated with retention of a convolutional layer, particularly C1 and C3. The occasional dips in accuracy associated with retention of the S-layers (i.e. S2 and S4) are somewhat puzzling. Since these layers are associated with the local averaging of features, one may speculate that at times they average too severely, and information which was of lesser importance to the source net was thrown out even though the target net has greater need of that information. Or, perhaps, the problem of averaging too zealously over the input also existed in the source net, but was never isolated in the fashion done here.

More noticeable however, are the occasional drops in accuracy found when all the layers through C5 are retained as opposed to just retaining the layers through S4. Since C5 is a convolutional layer, retaining it should only aid learning the target task by transferring relevant features. However, as apparent in Table 4.1, when layers C1-C5 are retained, fewer than 5% of the net's original free parameters are available to be trained. Furthermore, all learning must take place within a single layer, which limits the net's capacity even further. In spite of these effects, there are still significant benefits for using transfer learning, especially for noisy data. The attainable improvements in accuracy can be as high as about 20%. However, the major benefits of transfer are contained in the 3.2% of the net's free parameters at the bottom 3 layers of the net (C1-C3).

When comparing the gains in accuracy achieved by transfer learning in the presence of noise, it is important to be aware of the fact that there are two competing effects. The first is that with noisy data, any accurate information will have increased utility in aiding learning. The second effect is that it becomes much harder for the source net to gain accurate information to transfer, when this information must be extracted from noisy data sets. So, although one would expect to see increasing benefits for transfer learning as the data sets used for training become noisier, there should be a point where these increases do not appear, because the amount of useful information gathered by the source net has also decreased due to the noise.

There is yet another variable to be adjusted when making these comparisons, and that is the number of samples per class in the training sets used for the target net. Clearly, in the degenerate case where the training set has 0 samples per class, there is no benefit for transfer learning since the target net needs to be given something to learn. Equally clearly, if the target training set has an

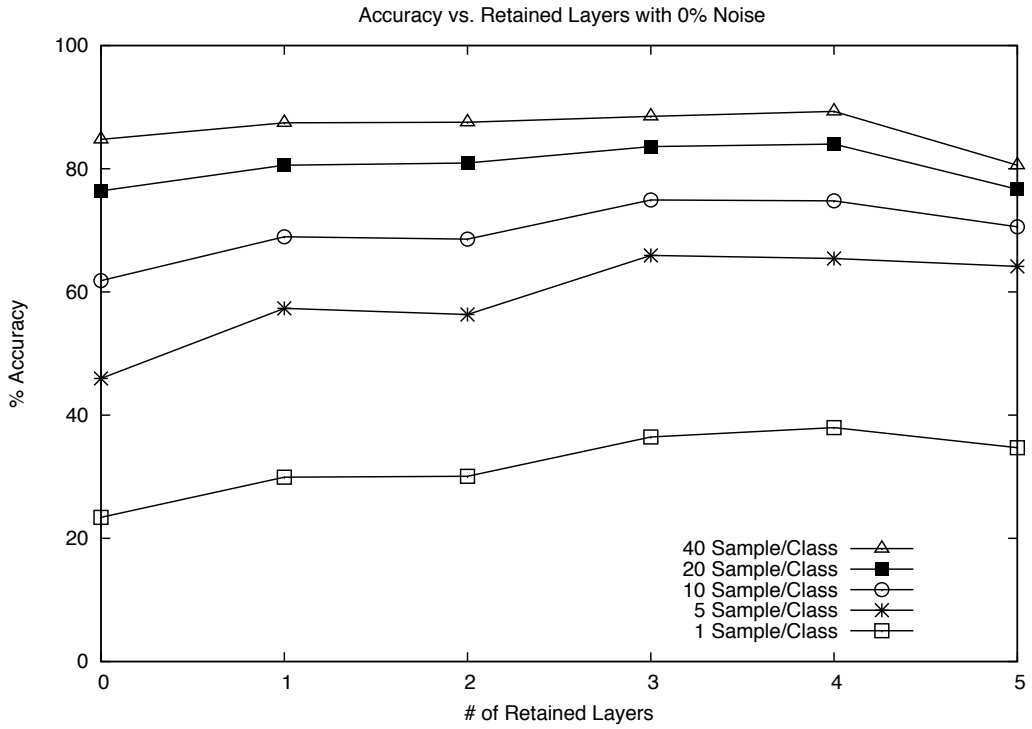

Figure 4.7: Comparison of learning curves showing accuracy vs. number of retained layers for various numbers of samples per class for the 0% noise set. training set. Curves show the results, from top to bottom, of 40, 20, 10, 5 and 1 sample per class. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

infinite number of samples per class, transfer learning will not improve the accuracy of the target net, since all the information being transferred will be duplicated in the infinite training set. This will lead to a peak benefit for some intermediate sized training set for the target net. If transfer learning truly does become more important in the presence of noisy data, then this peak benefit should occur for larger training set sizes as the amount of noise increases and should decay more slowly as the training set size increases for noisier data sets.

In Figures 4.10, 4.11 and 4.12, it can be seen that the maximum benefit from SBKT for

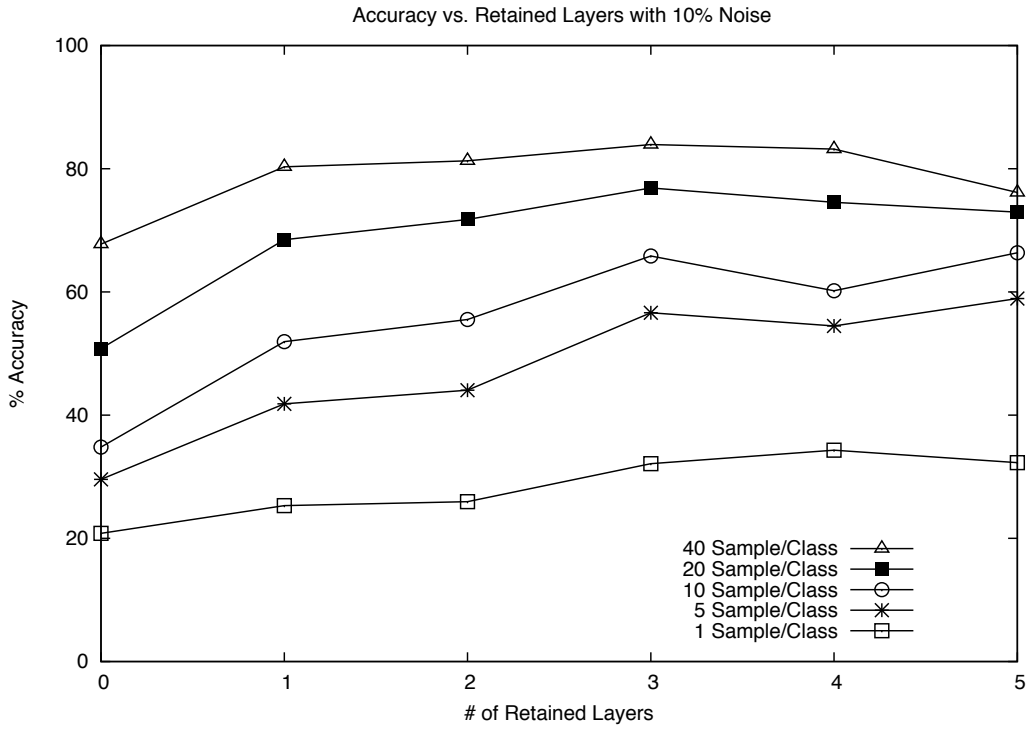

Figure 4.8: Comparison of learning curves showing accuracy vs. number of retained layers for various numbers of samples per class for the 10% noise set. training set. Curves show the results, from top to bottom, of 40, 20, 10, 5 and 1 sample per class. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

this particular neural net architecture tends to occur when the first 3 layers (i.e. C1, S2 & C3) are retained. What is also clear is how much harder it is to learn in the presence of noise, as evidenced by the drops in accuracy when no layers are retained.

The way in which the maximum benefits of SBKT, for the particular architecture being used, vary with degree of noise and samples per class is shown in Figure 4.13. In this figure, one can see indications that as it becomes more difficult for the source net to acquire knowledge to transfer, less knowledge is transferred. However, it also becomes increasingly difficult to acquire

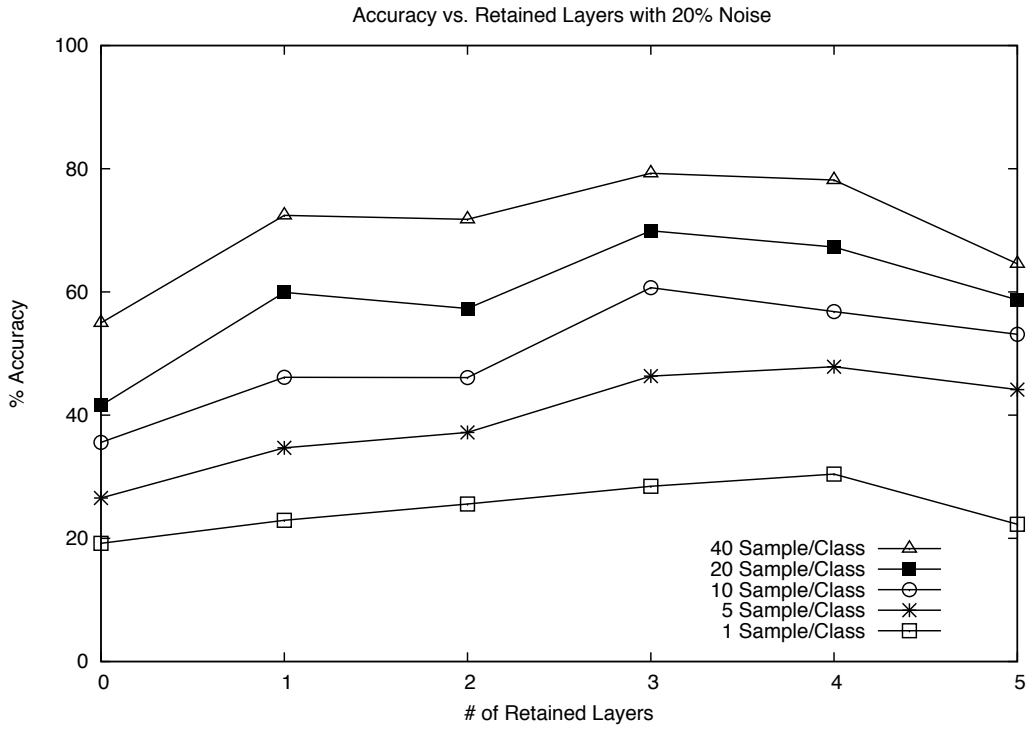

Figure 4.9: Comparison of learning curves showing accuracy vs. number of retained layers for various numbers of samples per class for the 20% noise set. training set. Curves show the results, from top to bottom, of 40, 20, 10, 5 and 1 sample per class. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

the transferred knowledge from scratch. This can be seen by the persistence of the transfer learning benefit even as the training sets grow larger. This added difficulty increases the value of any successfully transferred information.

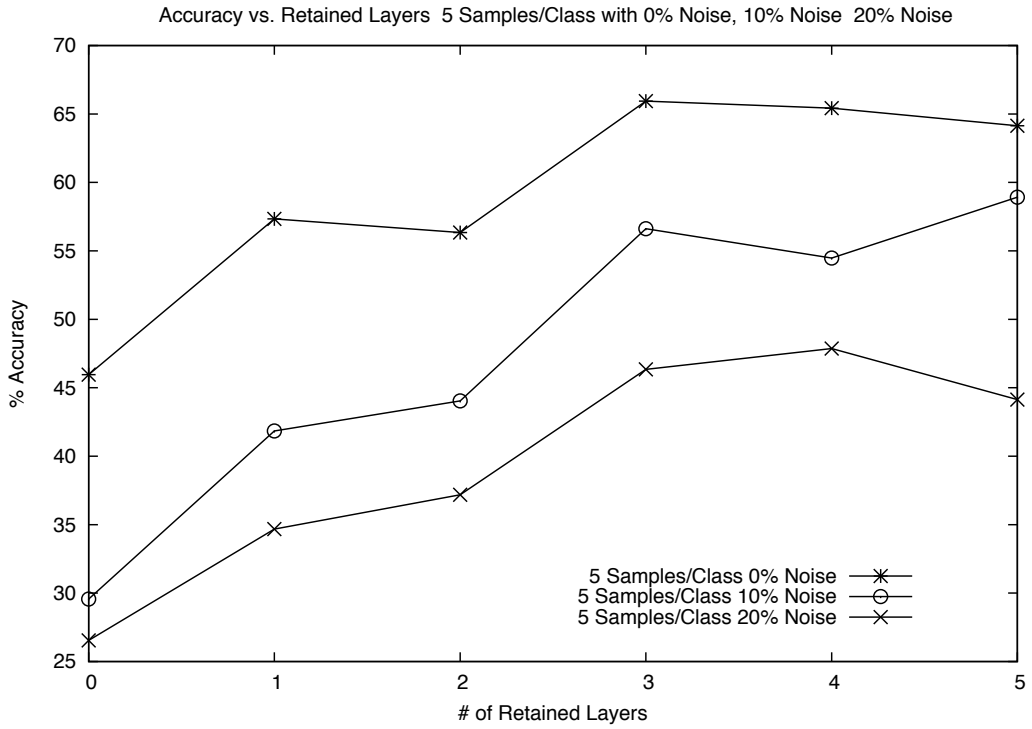

Figure 4.10: Comparison of learning curves showing accuracy vs. number of retained layers for 5 samples/class for 0% noise, 10% noise and 20% noise. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

4.4 Summary

The experiments discussed in this chapter demonstrate not only that it is possible to effectively transfer knowledge using only a neural net's architecture to determine which weights to transfer, but also that appreciable benefits are achievable even when the source net has mastered only relatively few tasks. Additionally, results shown here give a strong indication that transfer learning aids learning not only by helping the learner focus on relevant features, but also by helping it ignore irrelevant ones. The more irrelevant data for a specific task to which a learner is exposed,

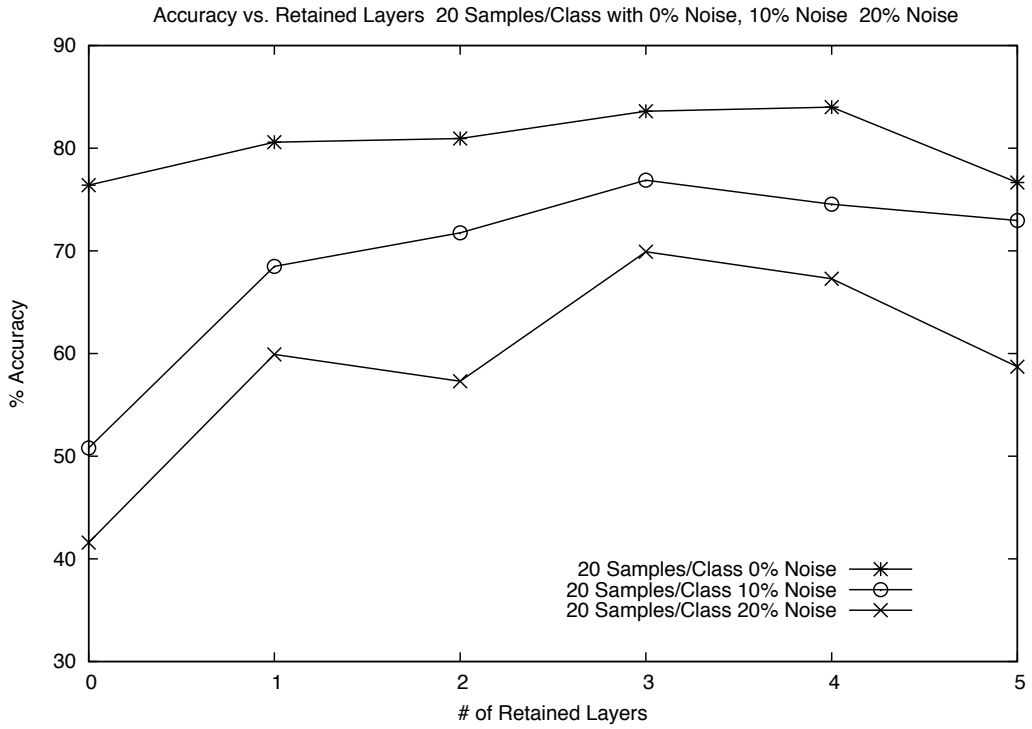

Figure 4.11: Comparison of learning curves showing accuracy vs. number of retained layers for 20 samples/class for 0% noise, 10% noise and 20% noise. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

the more important it is for that learner to ignore those features.

However, there is also territory that has yet to be explored. For instance, when transferring data, this incarnation of SBKT only uses the vertical structure of the net to aid transfer. The horizontal structure of the net is completely ignored. It is reasonable to ask the question what are the best ways to recognize which feature maps at a given level are worth transferring and which should be reinitialized for learning a new task, rather than just assuming that they all should be either transferred or reinitialized. Pratt's research suggested one way [36], but it would be useful

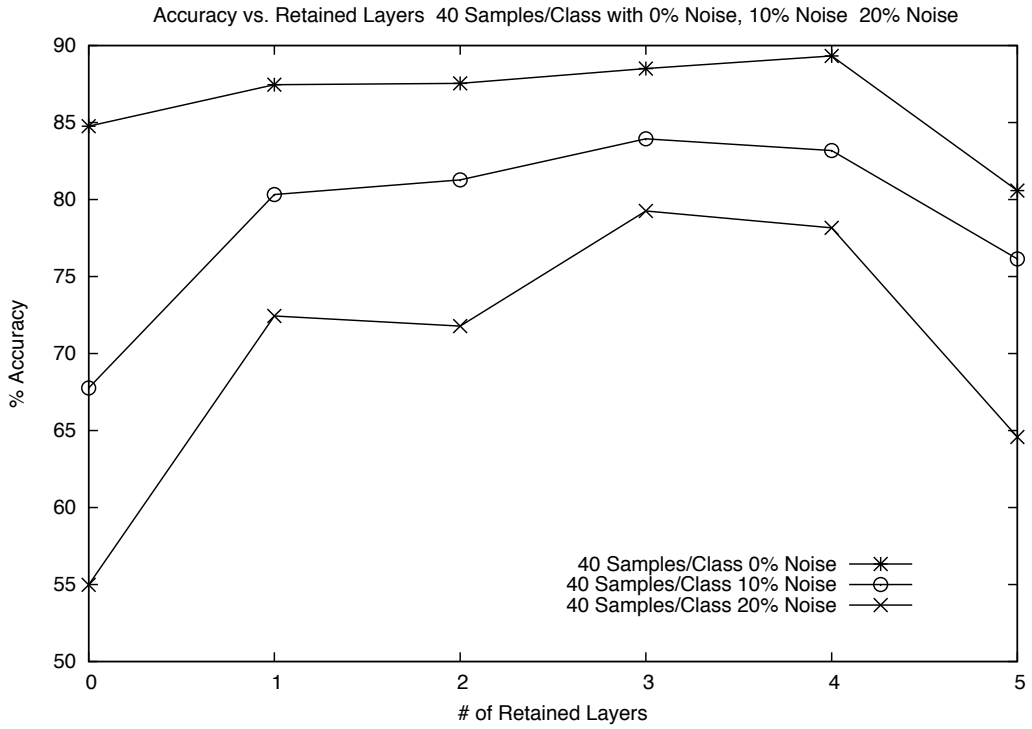

Figure 4.12: Comparison of learning curves showing accuracy vs. number of retained layers for 40 samples/class for 0% noise, 10% noise and 20% noise. Each point represents the average accuracy achieved over 5 trials on testing sets with 1,000 samples per class

to develop other techniques that are less labor intensive. Also, methods to allow the architecture of the net to change as it's being given new tasks would likely also be useful. It is, for example, easy to imagine a case where the source net only needed n feature maps at a given level to learn a set of tasks, but where the target net will need not only those n maps, but also an additional m maps. Techniques that are able to evaluate, or at least intelligently consider such possibilities, would be useful.

Lastly there is the question of just where should one pick a vertical cut-off for transfer. A

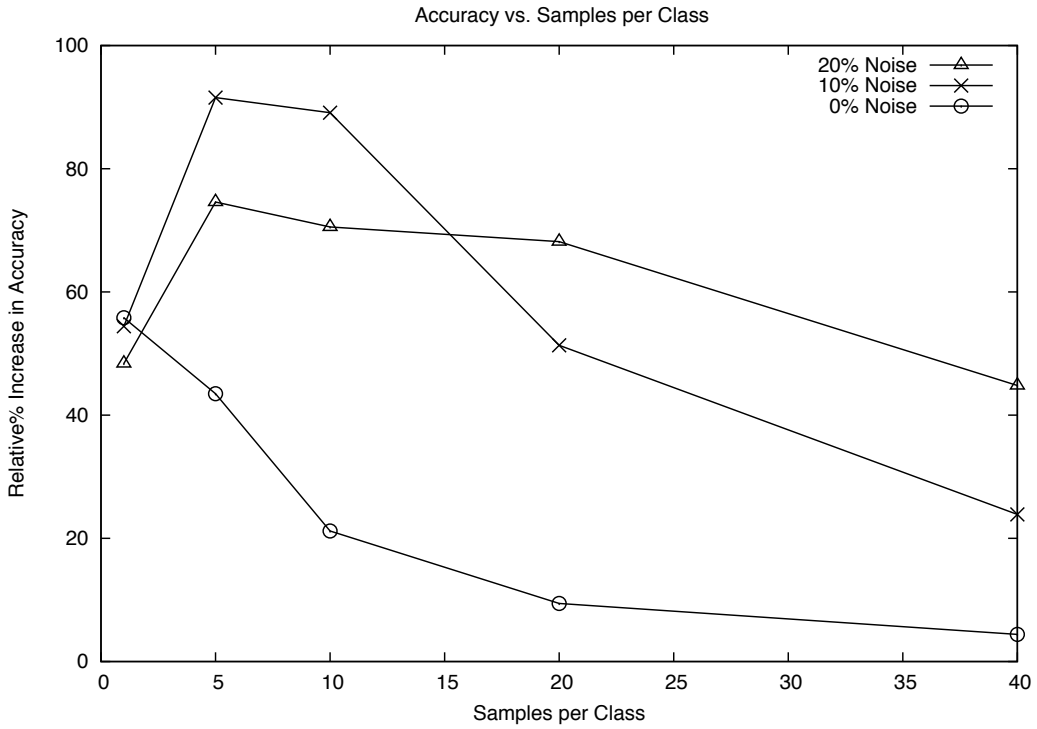

Figure 4.13: Relative % of improvement in learning accuracy when transferring the knowledge contained in layers C1- C3 vs. training from scratch without any knowledge transfer. Comparisons are made between the average accuracies achieved over 5 trials on testing sets with 1,000 samples per class

naive approach would be to take all the layers except for the top most layer. However, as the size of the target net's training set increases this is clearly not the optimal approach. So, the question remains are there techniques that would allow for this decision to be made dynamically? Perhaps, by allowing SBKT to judge which parts of the net should be transfered (and which should not) after some training has occurred? Or, perhaps by using a modified form of Optimal Brain Damage [14] to add new trainable nodes and feature maps at each level and prune away the ones that do not provide any additional benefit?

The observation that inspired the experiments discussed in the next chapter was that when retaining all layers except the topmost, for extremely small target training set sizes one would obtain optimal benefit from transfer, however, as the target training set sizes increased, the accuracy obtainable seemed to approach an asymptotic limit. Furthermore, this asymptotic limit was lower than the limits obtainable for larger training sets with less of the source net transferred. Although a greatly reduced net capacity undoubtedly played a part in this result, it seemed possible that this was not the entire story.

When using SBKT, or any other form of transfer learning, one is introducing a bias to the learner. In the case of SBKT, this is an absolute bias. Yet, when choosing target outputs arbitrarily, one is assuming that although the target net has a bias as to how to identify features to be used for character identification, it has no bias for preferred output representations of these characters. When only the top layer (i.e. the output layer) is allowed to retrain this does not seem particularly likely. This may partially explain the fact that even though transferring the penultimate layer was optimal with small training sets for new tasks, it was sub-optimal for moderate to large training sets. Possibly, larger training sets gave nets with larger capacities more opportunity to lean a 'non-inherent' output representation.

Perhaps, if the net already possesses an inherent bias for how it will represent new classes, this bias could be used as a transfer learning mechanism to aid learning. The next chapter will detail our investigation to develop a technique natural bias of a net to aid learning. This should both improve the accuracy and ease of learning. Furthermore, it should greatly reduce, if not eliminate, the risk of catastrophic forgetting when attempting sequential learning with neural nets.

Chapter 5

Inherent Bias and Latent Learning

5.1 Chapter Overview

In this chapter, the second technique developed for transfer learning will be introduced. This technique, latent learning, is reliant upon the inherent bias of a net, a concept which is also a contribution of this thesis. Some of the work presented here has already been published [24], some has been accepted for publication and some has yet to be submitted for publication in a more public venue.

What makes inherent bias and latent learning unique is the way they turn traditional training of neural nets on its head. Normally, such training involves modifying the behavior of a neural net so that it responds to input by producing output that is both consistent and meaningful. The output is meaningful because the trainer/user has decided what output is desired for what input. However, even before training, a neural net will produce output in response to an input. Frequently, this output would be useable by the trainer/user for the task of interest, if only it were known. Inherent bias and latent learning concentrate on determining the existing responses of a neural net to various classes of input instead of modifying those responses in a manner chosen arbitrarily by a trainer.

Inherent bias is the term I've coined to describe the set of outputs that exist for given classes of input before the net has been trained to have any specific responses for those classes. In this chapter, the method used to determine inherent bias is discussed, in addition to some alternative methods that will be the subject of future research. It will be seen that the inherent bias of a net that has only been randomly initialized will still be able to discriminate among a set of classes with an accuracy that is far greater than random guessing would be expected to achieve.

Latent learning is generally used to refer to tasks that are learned without any rewards or punishments to incentivize the learner. Usually, these tasks are mastered through passive observation - the way a child may learn to make coffee by watching his parents make coffee each morning. The child might not be asked to make coffee; the child might not even desire coffee, but when the time comes to make coffee, the child may find the knowledge already there. Here, latent learning is used to refer to tasks that are implicitly learned while being explicitly trained for a related set of tasks. Specifically in the experiments described in this chapter, when a set of responses for a set of handwritten letters is explicitly learned, a set of responses for handwritten digits is implicitly learned. These implicitly learned responses are the net's inherently biased responses for those classes.

The inherent bias produced by latent learning provides greater accuracy than that created by a randomly initialized net. It should be noted that in this case, learning has occurred without task-specific training. Training takes place when behavior is modified in order to produce desired responses to specific stimuli. If a net's inherent bias is used to discriminate among classes, then its existing behavior is learned by the user, not modified by the user. One may, perhaps, think of the net's operator as being trained instead of the net.

Furthermore, as will be shown in this chapter, significantly greater accuracy may be achieved with the addition of supervised learning, even if only a very small portion of the net is permitted to train. The use of both inherent bias and latent learning, in the sense described above, for neural nets is an original contribution of this thesis.

5.2 Inherent Bias

Standard methods of training neural nets involve training the net to give a specific output for a specific, arbitrarily chosen, class of input. The response function of neural net nodes will have two target values. One target is associated with a positive response and usually has a value of $+1$. The other target value is associated with a negative result and usually has a value of 0 or -1. Generally, when the net is intended to differentiate among several mutually exclusive classes,

there is one output node assigned to respond positively to each class. This is commonly referred to as 1-of-N coding. The net is then trained to enforce this arbitrary set of responses. However, before any training occurs, if the net is exposed to an input, the nodes will respond. There will be some subset of the output nodes that have a response more suggestive of the positive response target value, and others whose response is more suggestive of the negative output value. This raises the question of whether that response carries sufficient information to be useful.

In order for the inherent response to be useful, two conditions need to be true:

1. For a given input class, the probability distribution must be sufficiently localized to associate a distinct output pattern(s) with that input class. In this thesis, a singe output pattern will be assumed. Since the net (at least the type used in this thesis) has deterministic responses, the randomness of the probability distribution is caused by variations in the specific samples from each class.

2. The output pattern(s) associated with each input pattern must be easily distinguishable.

In order to determine these output patterns, a net can be presented with a number of samples of each class to be identified. Although, there are many ways to look at these responses and decide what the inherent response is for a given class, the initial approach taken was merely to declare the inherent response for each unit node to be $sgn(\text{average output})$. So, if, on average, a particular output node responded with a positive value to a given class, the inherent response was deemed to be $+1$. Likewise, if the average response were a negative value, the inherent response was deemed to be -1.

A more exacting approach would be to consider the degree to which the response of a given output node was correlated with a given class. This would allow the interpretation of the output node as voting whether or not a particular input belonged to a given class. The weight associated with that vote would be determined by how strongly correlated that node's response was to the given class. Investigations along these lines are planned for the future.

The neural net used to examine the potential usefulness of inherent bias was the same LeNet5-style architecture as used in previous experiments. This net is shown in Figure 5.1.

The initial experiments involved first randomly initializing the state of the neural net and then exposing it to small numbers of samples from the classes to be identified. These classes were the handwritten digits '0'-'9', as supplied by NIST Special Database 19. The experiments were performed using training sets of 1, 5, 10, 20, 40, 80 and 160 samples per class. Then using the inherent response, the accuracy of the net on a test set with 100 samples per class (i.e. 1,000 characters) was measured.

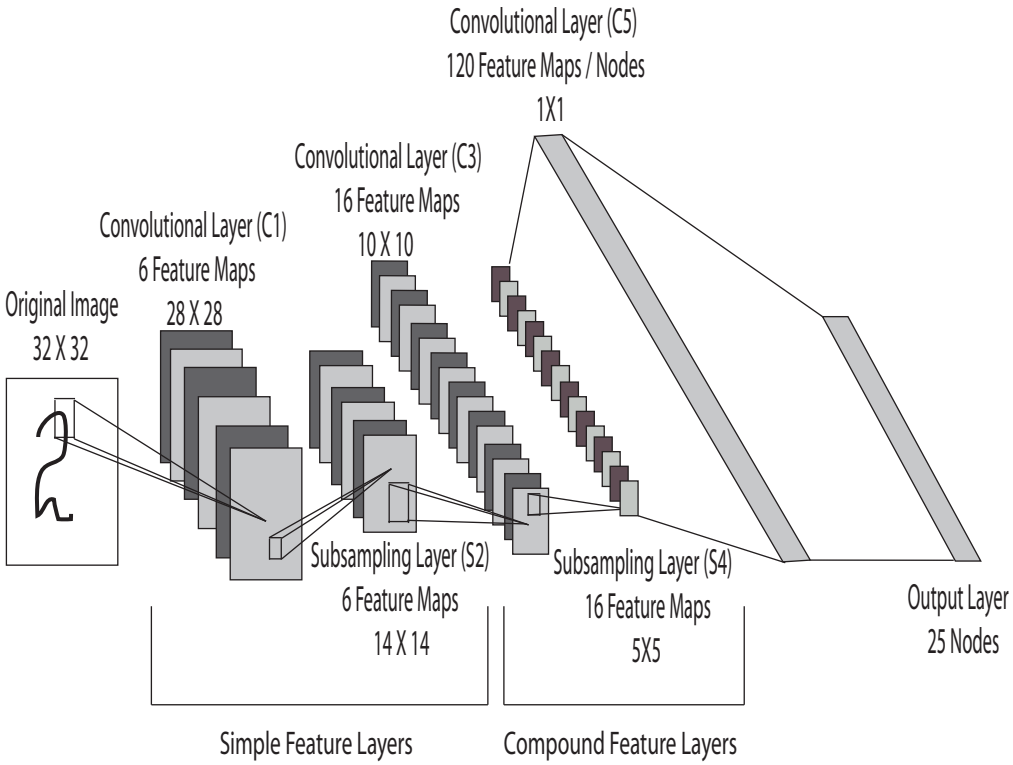

Figure 5.1: The LeNet5 style neural net used for the experiments described in this chapter

Each training set was evaluated with its own, individual test set. There were 5 pairs of training/testing sets for each training set size. Two randomly initialized nets for each training/testing set were created. This gave 10 accuracy measurements for the inherent bias at each training set size. An accuracy measurement reflects the percentage of samples that are correctly classified by the net. Minimum, average and maximum accuracies observed for each training set size are

shown below in Figure 5.2 . It should be noted that random guessing would be expected to give an accuracy of only 10%. What is more remarkable about the results achieved with the nets' inherent biases is that they were achieved in the **complete absence of any training**.

The closest thing to 'training' that occurs is the learning the correspondences beteween individual classes and outputs of the neural net. Although it is possible to claim that this represents a 'training' of the classifier/post-processor, I would argue that all that is occuring is observation and recording of average responses to classes of input. No behaviors are changed, they are merely given labels. It is 'training' only in the same degenerate sense that gathering samples for the k-nearest neighbor classification method is 'training'.

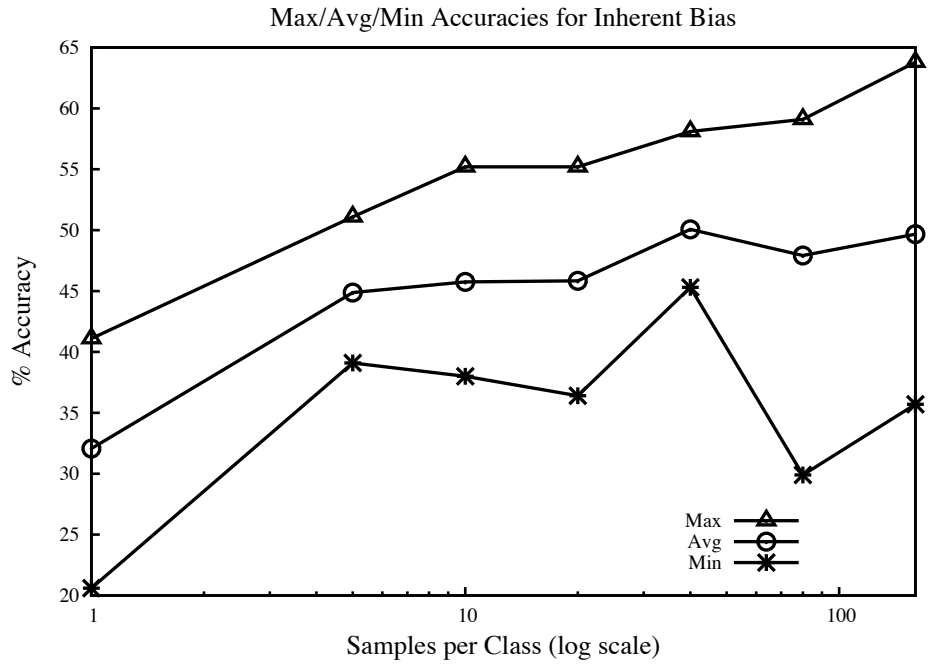

Figure 5.2: Maximum, average and minimum accuracies in discrimination among the characters '0'-'9', for nets with random initialization and **no training**. The log-scale x-axis gives the sample sizes used to determine each net's inherent bias.

As stated at the beginning of this chapter, to train something, whether an animal or a neural net, means to somehow modify its behavior for a desired purpose. In the case of neural nets, it generally means getting the net to give a specific response to a specific type of input. This did not occur. The net's inherent responses were not modified, instead, they were learned. Any 'training', which took place was experienced by the user of the net, who learned to interpret which responses the net would give to which inputs. Unfortunately, although the average accuracies achieved are far better than one could hope for by random guessing, they are too low and too variable to be practically useful. However, these results to suggest that a net already possessing some knowledge of the problem domain, could have a much more accurate set of inherent biases.

5.3 Latent Learning

In order to create a net with better inherent responses for the desired tasks, the next set of experiments took advantage of *transfer learning* - the process of using learning acquired from one set of tasks to help learn another set of tasks. In these experiments, the net was first trained to discriminate among the following set of 25 handwritten characters from NIST Special Database 19 - 'A', 'B', 'C', 'D', 'E', 'F', 'H', 'I', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'Y', 'a', 'b', 'd', 'e', 'g', 'h', 'q' , 'r', 't'. These characters were chosen mainly due to the number of samples that the data set contained of each of them. I was able to create training and testing sets with 775 samples/character and 20 samples/character, respectively. The letters 'O' and 'o' were excluded, since they were deemed too close to the number '0' to use for transfer learning experiments.

The hope in doing this is that once the net has learned a set of encodings for one set of characters, its inherent responses for other, similarly formed, characters will be more reliable and distinguishable. Such a shaping of a net's inherent bias would be an example of *latent learning*. The next set of experiments demonstrate that while training a net to give a specific set of responses to one set of characters, it acquires a set of latent responses to other characters to which it has had **absolutely no exposure**.

Initially, the net was trained to discriminate among the 25 character classes mentioned above

by producing a specific output response. This output response will be referred to as the 'target point' for that class. The specific target point for each character was stochastically determined by giving each node a 50% probability of being assigned a $+1$ or a -1 target value for that class. Input images would then be classified based upon which target point was closest, in a Euclidean sense, to the output pattern they produced. The net was trained using a training set of 775 samples per class. The testing set was much smaller and consisted of only 20 samples per class. The accuracy achieved on the training set was 95% and that achieved on the testing set was 92%. Then, using the new set of inherent responses for the handwritten digits '0' - '9', the previous experiments were repeated. The average accuracies achieved are compared to those without latent learning in Figure 5.3. The use of latent learning results in both increased accuracy and a more consistent level of accuracy, than that provided by an inherent bias that is uninformed by any latent learning. This may be seen in Figure 5.4, which shows far less variance in the achievable accuracies than those obtained by only inherent bias, in Figure 5.2.

Although these results represent an improvement, they still are not useful for practical purposes. However, if one is limited to training sets as small as the ones being used to determine the net's inherent bias, it is reasonable to ask whether traditional supervised training would provide accuracies superior to those of latent learning. If not, then would traditional supervised learning even manage to outperform the use of a net's inherent bias in the absence of latent learning?

To make this comparison, a randomly initialized net was trained using the standard supervised learning approach on all 5 of our training/testing sets for the character set '0'-'9'. The ultimate accuracies achieved by each net were determined with validation sets, which like the testing sets had 1,000 characters (i.e. 100 samples/class).

As can be seen in Figure 5.5, only in the case of training sets with 1 sample/class did pure latent learning outperform standard supervised learning. In fact, a closer look at all the accuracies achieved would show that only about 20% of the time would supervised learning achieve a result that was even marginally better than the **worst** result obtained from pure latent learning. However, with only moderate increases in sample size, supervised learning outperforms pure latent learning. This is not unexpected and even a little relieving. It makes sense that just as a person,

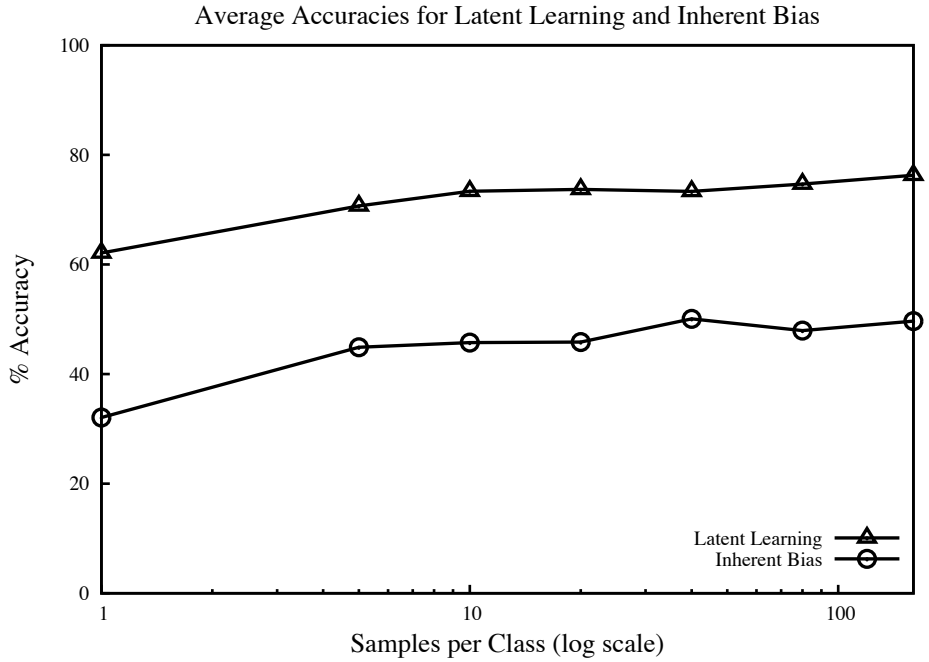

Figure 5.3: Comparison of average accuracies achieved using latent learning vs. inherent bias without latent learning. The x-axis gives the sample sizes used to determine each net's inherent bias.

a neural net that puts actual effort into mastering a specific task can perform it better than one which does not.

The main reason to insist on not training a net that has taken advantage of latent learning would be to avoid *catastrophic forgetting* - the tendency of a neural net to forget old skills and tasks, when being trained for new ones. The prime purpose of the experiments being discussed is to use information already contained in a neural net to help it more quickly learn a new set of tasks. Determining how best to eliminate or mitigate catastrophic forgetting is a matter for future investigations. However, it seems likely that training a net in a manner consistent with behavior it already possesses will reduce the effect of catastrophic forgetting.

If the neural net is allowed to train, in addition to having its desired output for each class of

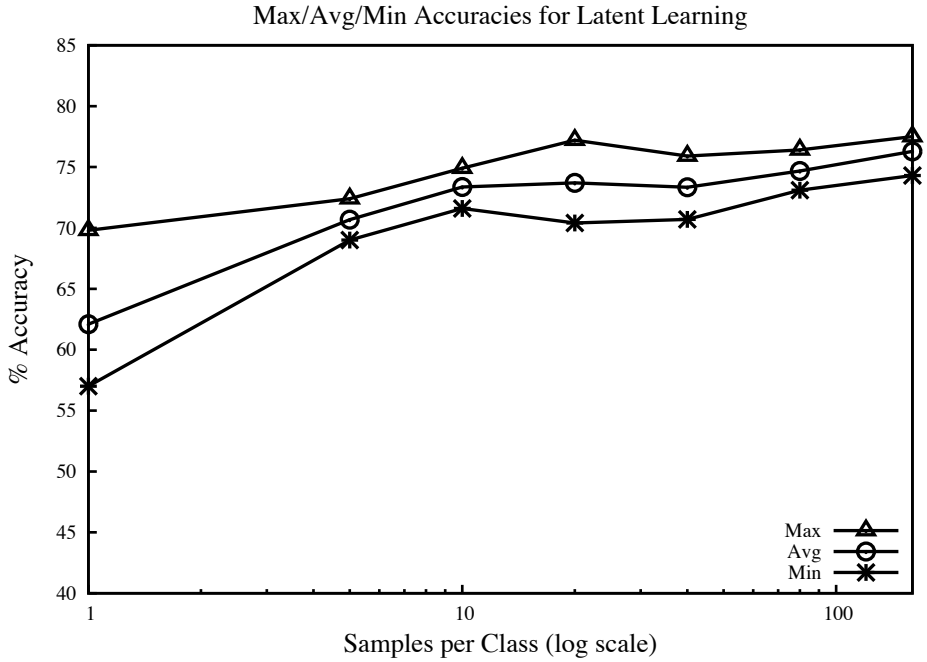

Figure 5.4: Maximum, average and minimum accuracies achieved with pure latent learning. The x-axis gives the sample sizes used to determine each net's inherent bias.

input determined with latent learning, then not only would we expect higher accuracy rates for smaller training sets, but also it might not be necessary to train the entire net. Perhaps, it would be sufficient to allow only the very uppermost layer to train. Experiments were performed where only the 25 output nodes were trained. The free parameters associated with these nodes account for just 5.7% of the free parameters of the net. The gains sought here are not only learning with smaller training sets, but also using nets that are very small - in this case only 3,025 free parameters as opposed to 52,861. An additional, anticipated future benefit of only allowing part of the net to train is a reduction in the catastrophic forgetting effect.

Figure 5.6 demonstrates that by allowing even only this small portion of the net to train, there is a significant improvement in accuracy. Furthermore, the advantages of latent learning are

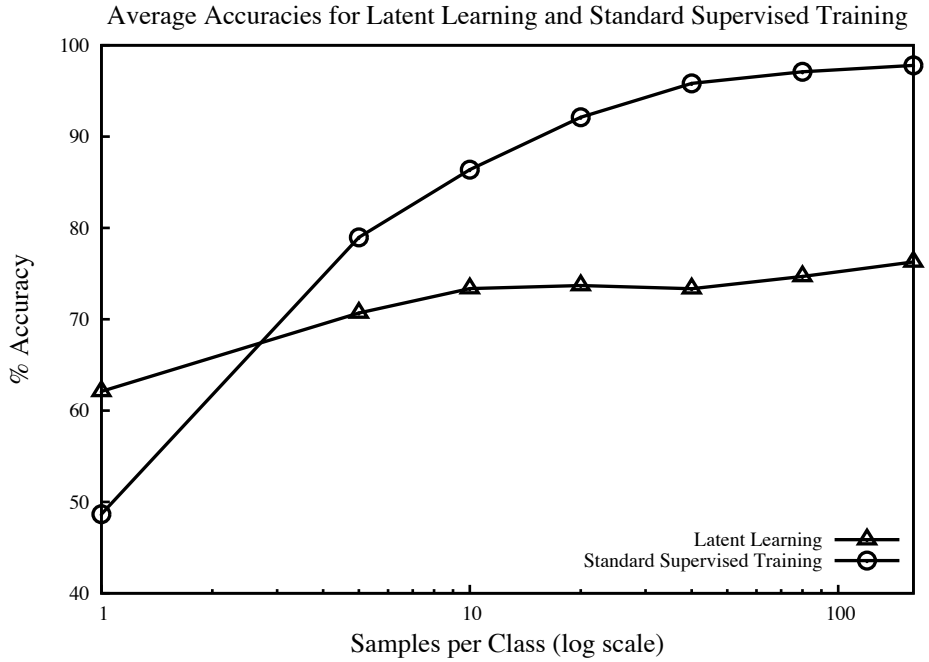

Figure 5.5: Comparison between average accuracies achieved for latent learning without task-specific training and the standard supervised learning approach. The x-axis gives the sample sizes used to determine each net's inherent bias.

more evident, since the increase in accuracy over supervised training without latent learning is larger and persists for even larger training sets. Ultimately, the net that used latent learning and partial supervised training didn't do quite as well as the net which employed standard supervised learning. However, the difference in asymptotic performance between the two nets is fairly small - only about 1.5%. This discrepancy is most likely due to the difference in the number of free parameters in the two nets.

So, for a final comparison, consider the performance achieved by the latent learning net when its allowed to train in its entirety. The results are shown in Figure 5.7. Here it can be seen that a net which uses latent learning with supervised training will perform better than a net which uses traditional supervised training alone for small training sets. However, ultimately the two nets will

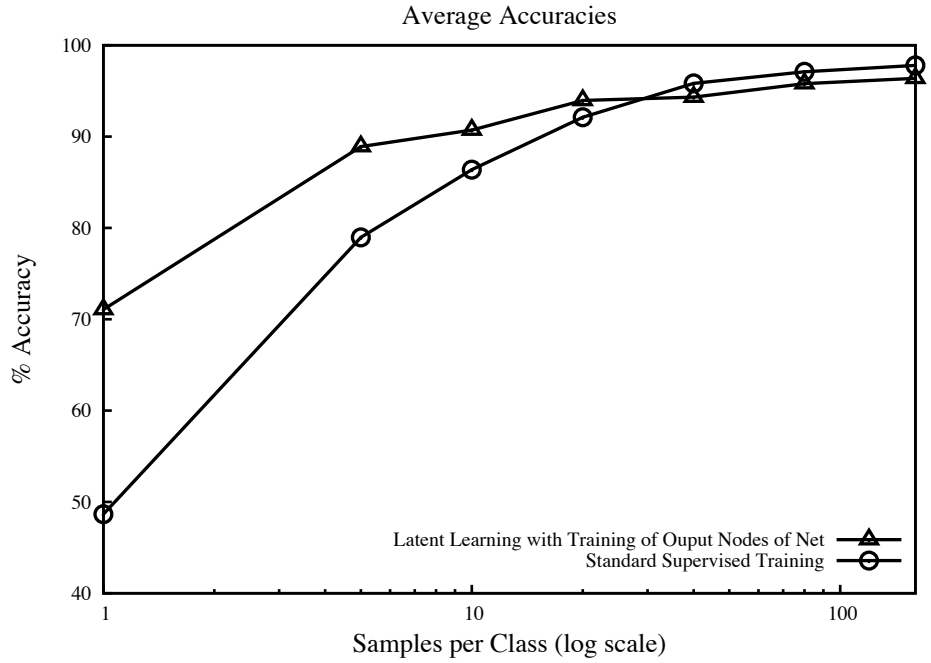

Figure 5.6: Comparison between average accuracies achieved with latent learning with task-specific training limited to the output nodes to train versus the standard supervised learning approach. The x-axis gives the sample sizes used to determine each net's inherent bias.

achieve the same asymptotic degree of accuracy as the size of the training sets increases.

5.4 Training Latent Learners

5.4.1 Considerations for Training a Latent Learner

Although latent learning shows promise as a technique both for transfer learning and life long learning (i.e. the ability to continually learn new tasks, without forgetting old ones), it does raise a unique set of concerns. Primarily, how does one design and train a net for a task that has not

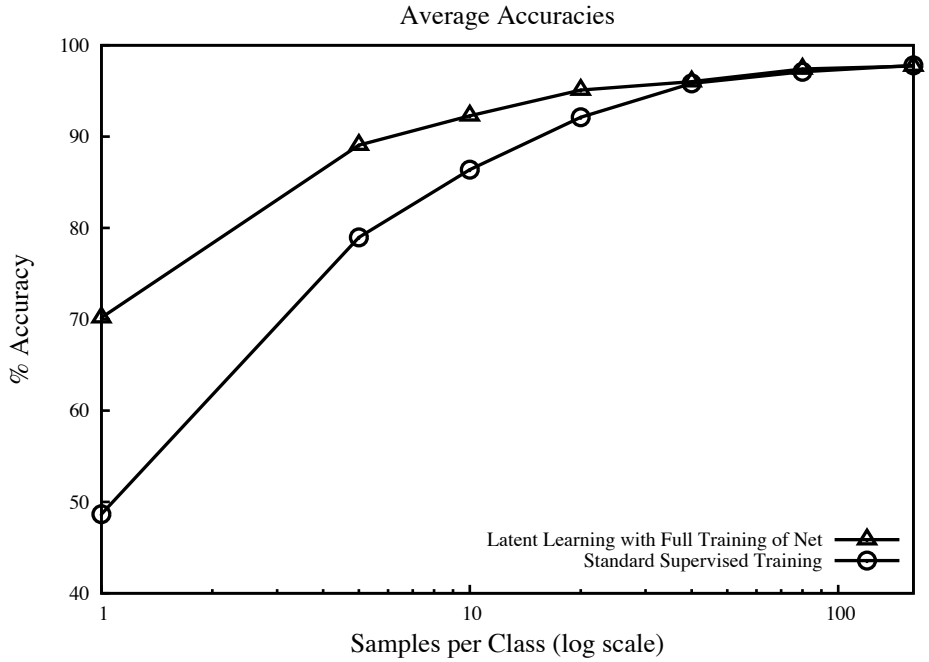

Figure 5.7: Comparison between average accuracies achieved with latent learning with task-specific training for the full net to train versus the standard supervised learning approach. The x-axis gives the sample sizes used to determine each net's inherent bias.

been completely determined?

Supervised training of a neural net entails using a labeled training set, to adjust the internal parameters of the net. The adjustments can be made either after each sample is presented to the net, or after the entire batch of samples is presented to the net. Each adjustment tends to be very small in relation to the parameter being adjusted. After adjustments have been made for each member of the training set, whether one at a time or all at once, this training process can be repeated. The intent is that after each complete pass through the training set, the net will more accurately label the members of the training set.

To the degree that the training set is statistically representative of the samples liable to be

encountered subsequent to training, the learning that takes place is likely to generalize, allowing the net to correctly label data to which it has never been exposed. This is the goal of supervised training. However, if one allows the net to train too much, it will begin to place too much emphasis on features that are unique to the particular training set and that do not generalize to all possible samples to which the net might eventually be exposed. This is known as *overtraining*.

To guard against overtraining, one makes use of a set of labeled examples, known as a *validation set*. The internal parameters of the net (i.e.weights) are **not** specifically adjusted to optimize the net's ability to label members of this set. However, this set is used to provide an indication of when overtraining is occurring. If the net's performance continues to improve on the training set, but starts to degrade on the validation set, then one may conclude overtraining is occurring. As a result, a common method of training a neural net is to allow it to continue to train on the training set, until it becomes clear that its performance on the validation set has been optimized. The version of the net that had the highest accuracy in labeling the validation set is chosen as being most likely to provide the best overall performance on samples to which it has not been exposed. Then, a third labeled data set, known as a *testing set*, is used to provide an estimate of the net's expected accuracy when exposed to samples it has never seen before.

When one knows the problem for which the net will be used, this is perfectly reasonable. But, a latent learner does not know the problems with which it will be faced. With each training epoch, one would expect the inherent bias of the net to change. Possibly, in terms of which characteristic output will be associated with each class, definitely in terms of what response will be given to each specific input. When should one stop training on the initial set of tasks, in order to preserve the latent learning on some unknown set of tasks? To what degree will latent 'forgetting' occur as one trains on the set of known tasks.

A second issue in creating a latent learner involves the number of nodes needed for the output layer. Generally, when considering the number of nodes needed for a neural net, most of the concern centers around the number of hidden nodes. The number of nodes for the input layer is governed by the dimensionality of the raw data, whereas the number of nodes for the output layer tends to be governed by the desired number of output responses. Since our approach to latent

learning not only doesn't know in advance how many distinct responses are desired, other than for the first set of learned tasks, but also depends inherent bias to provide labels for new tasks, it is no longer clear how many output nodes are desirable.

5.4.2 Empirical Results from Training a Latent Learner

In order to begin to see how long a net should be trained in order to preserve any latent learning that it may acquire and how many output nodes it should have, the following sets of experiments were performed.

First, 25 nets were trained to discriminate among the following set of 25 handwritten characters from NIST Special Database 19 - 'A', 'B', 'C', 'D', 'E', 'F', 'H', 'I', 'P', 'R', 'S', 'T', 'U', 'V', 'W', 'Y', 'a', 'b', 'd', 'e', 'g', 'h', 'q' , 'r', 't'. The architecture of these nets was essentially the same as that of the net used for the original set of latent learning experiments, except for the number of output nodes.

The 25 nets were divided into 5 sets of 5 nets, with 25 output nodes, 100 output nodes, 400 output nodes, 1600 output nodes, and 6400 output nodes, respectively. These numbers were chosen in the following way:

In the first set of latent learning experiments, the latent learning was provided by training the net on a set of 25 classes, so it made sense to have 25 output nodes. Initially, when deciding to investigate various numbers of output nodes, I realized that because the distance between two randomly created target points scales as the square root of the number of output nodes, the expected distance between two such points doubles when the number of output nodes quadruples. This was the reasoning behind choosing to also examine nets with 100, 400, 1600 and 6400 output nodes. Upon subsequent reflection, I felt that the ratio of output nodes to input nodes is, perhaps, a more useful parameter. With this in mind, the ratios of output nodes to input nodes examimed were approximately 0.02, 0.10, 0.39, 1.56 and 6.25.

Each net was initialized with a different set of random weights and had a different set of randomly generated target points for the 25 handwritten characters.

The initial set of results in Figures 5.8 - 5.12 show to what degree latent learning occurred with respect to the set of 10 handwritten digits, after each epoch of training on the set of 25 handwritten characters. The 0^{th} epoch refers to the nets when they have been randomly initialized and nothing more. It serves as a benchmark for how much accuracy can be obtained solely by unsupervised clustering with inherent bias.

Figures 5.8 - 5.12 also show the average accuracy achieved by the 5 sets of nets with a stated number of output nodes when the number of samples per class used to determine the latently learned target points for the hand written digits are 1, 5, 10 & 160 samples per class. The most important result from these graphs is that the efficacy of latent learning nearly always peaks at the first training epoch. This raises the question, for future work, whether latent forgetting can be avoided by merely rearranging the order in which training samples are presented to a net during training, whether no training sample should ever be seen more than once in a series of training epochs or whether there's some optimal relation between number of labels and the number of samples in the training set for the non-latently learned labels.

It should also be noted that most of the benefit from latent learning could be achieved using only 10 samples per digit to determine the net's inherent responses. This, however, is most likely a function of our specific experiment. Were either the set of classes learned through supervised training or the set of classes learned through latent learning to be larger, possess a higher dimensionality or be, in some sense, more similar to each other, then one would expect more samples would be necessary to determine the inherently biased response for each class.

The experiments performed in this section essentially had 4 variables of interest:

1. % accuracy differentiating digits
2. Number of training epochs to differentiate characters
3. Number of output nodes
4. Number of samples per digit

Figures 5.8 - 5.12 showed how % accuracy differentiating digits varied with the number

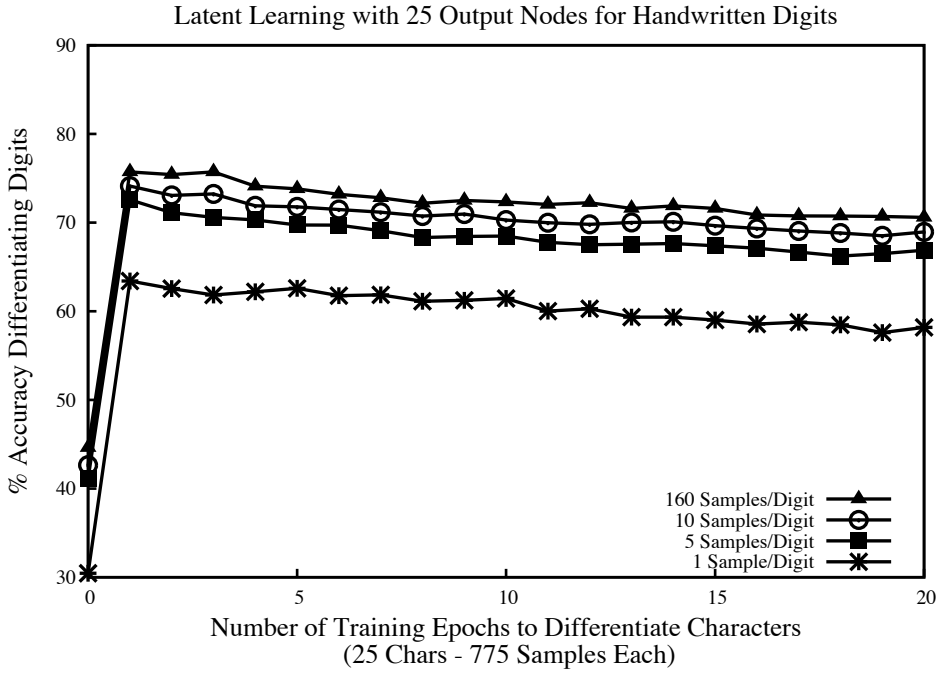

Figure 5.8: Accuracy achieved by latent learning for differentiating among handwritten digits as neural net with 25 output nodes is trained to differentiate among 25 handwritten characters with 775 samples per character. The ratio of output nodes to input nodes is about 0.02

of training epochs to differentiate characters, for a different numbers of samples per digit while holding constant the number of output nodes of the net. Figures 5.13 - 5.19 show the same data, only with different numbers of output nodes, while holding the number of samples per digit constant. The important aspect of these graphs is to note that once the number of output nodes exceeded the number of input nodes (i.e. 1,024, since the raw images were 32 X 32 pixels), there was no further benefit from using a pure latent learning technique.

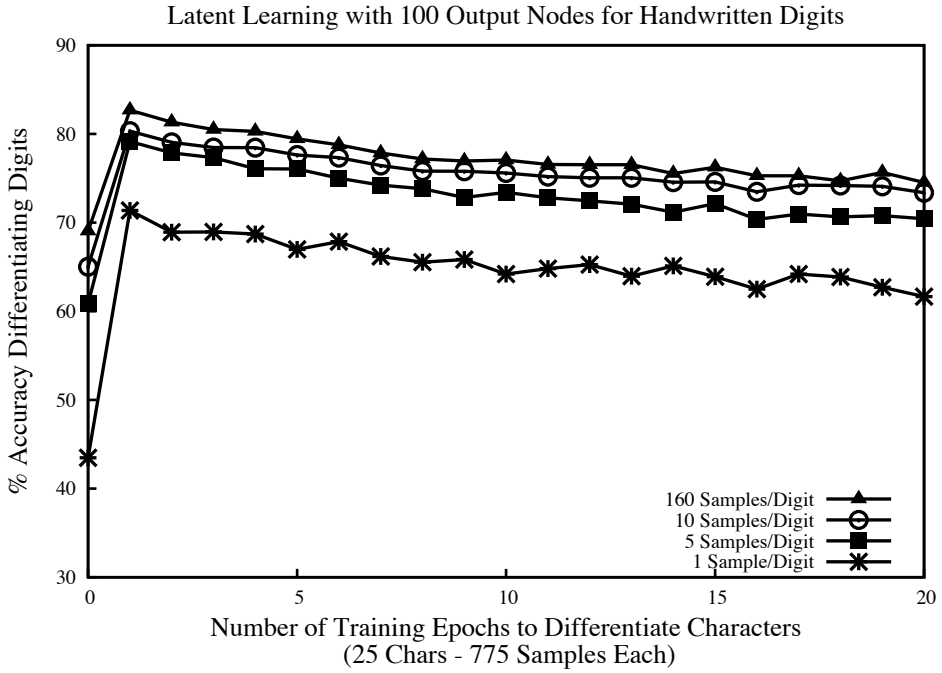

Figure 5.9: Accuracy achieved by latent learning for differentiating among handwritten digits as neural net with 100 output nodes is trained to differentiate among 25 handwritten characters with 775 samples per character. The ratio of output nodes to input nodes is about 0.10

5.4.3 Summary

In this section, a technique has been introduced for allowing neural nets to take advantage of latent learning. The fundamental idea behind this technique is that when a neural net learns to differentiate among some members of a set of related classes, it acquires a logically consistent, internal representation for the entire set of classes. This gives it a latent representation for members of that set to which it has not been exposed. Taking advantage of this latent representation is a form of transfer learning, which not only enables learning with very small training sets, but also holds the promise of allowing neural nets to maintain the ability to learn new tasks without catastrophic forgetting.

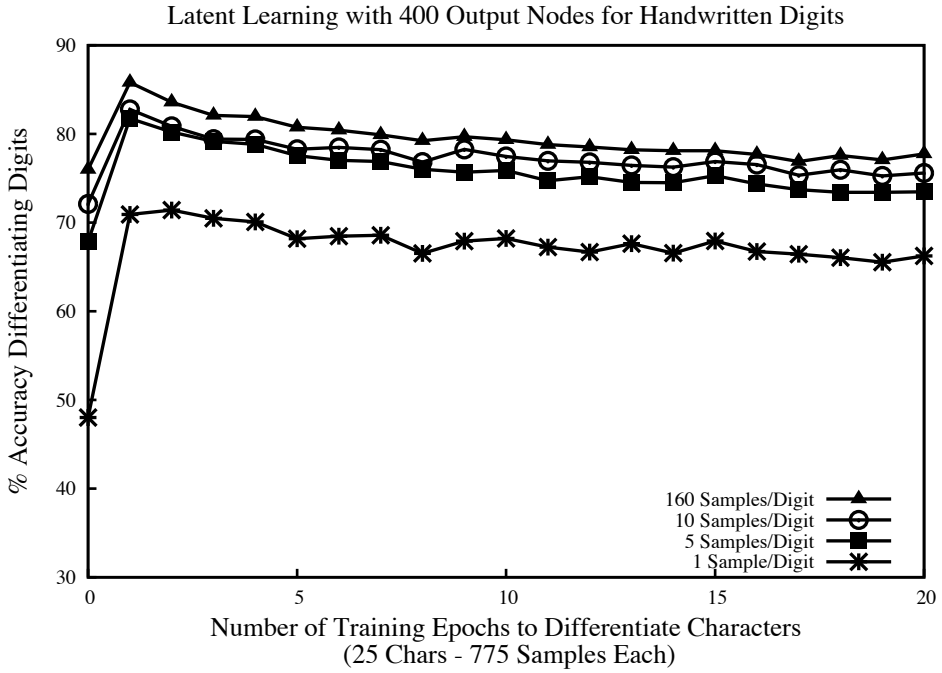

Figure 5.10: Accuracy achieved by latent learning for differentiating among handwritten digits as neural net with 400 output nodes is trained to differentiate among 25 handwritten characters with 775 samples per character. The ratio of output nodes to input nodes is about 0.39

The methods shown here are still very rudimentary. However, they do demonstrate latent learning with neural nets, which has not been explicitly shown previously. They also demonstrate that latent learning enables the productive use of very small training sets. Currently, one of the main difficulties with supervised learning is the need for copious amounts of labeled data. Common ways of mitigating this need include semi-supervised learning, which uses unlabeled data to supplement the learning provided by labeled data, and multi-task learning, which uses simultaneous learning of related tasks to enhance the acquisition of each individual class. Latent learning extends multi-task learning by learning related tasks for which the net is not specifically trained. This greatly reduces the need for labeled data samples.

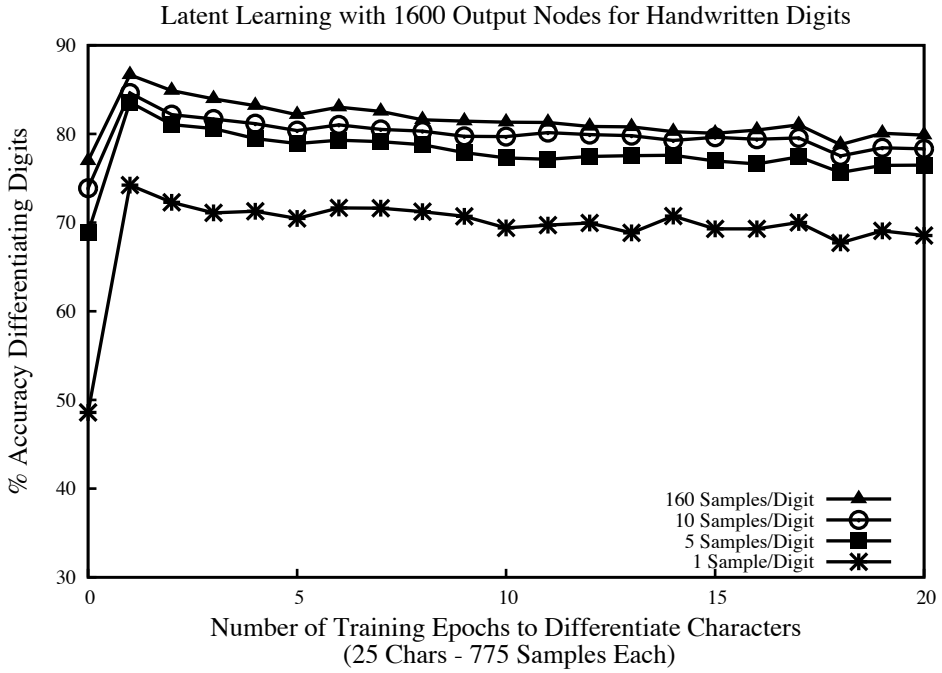

Figure 5.11: Accuracy achieved by latent learning for differentiating among handwritten digits as neural net with 1600 output nodes is trained to differentiate among 25 handwritten characters with 775 samples per character. The ratio of output nodes to input nodes is about 1.56

There has not been a demonstration of true life long learning. For true life long learning, once a net has mastered a set of tasks,A, and has then latently learned a set of tasks, B, it should then demonstrate mastery over the set of tasks $A \cup B$. My preliminary investigations indicate that instances of classes from the set of tasks, B, show a strong tendency to be mislabeled as a member from the set of tasks, A.

There has not been a demonstration of the elimination of catastrophic forgetting. Preliminary investigations show that when the net is allowed to undergo supervised training for the set of tasks, B, in addition to latent learning by virtue of being trained for the set of tasks, A, that its accuracy with respect to the set of tasks in A decreases. Decreases of about 15% in accuracy have

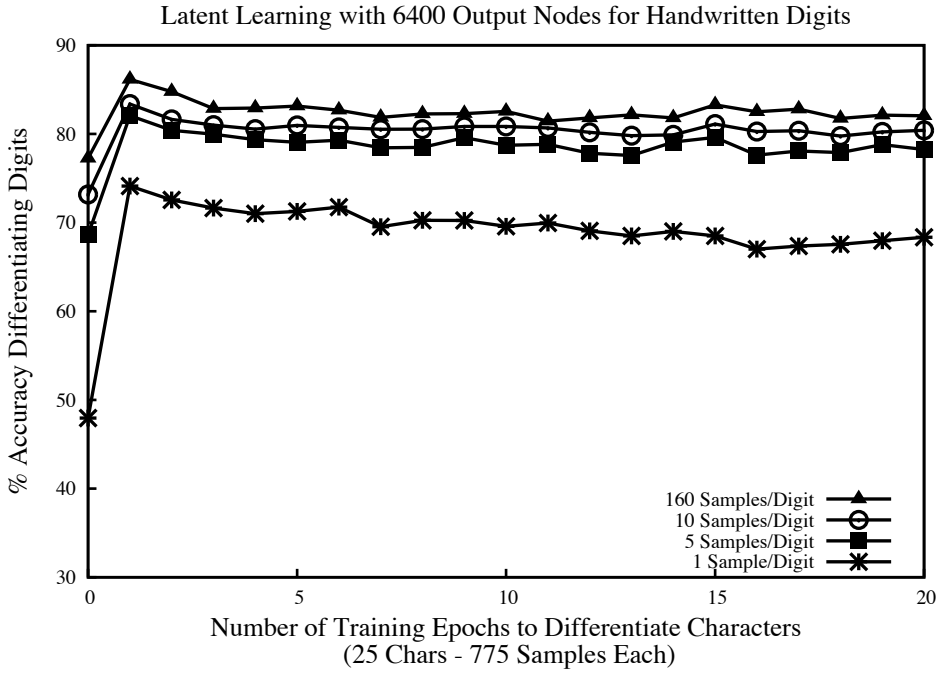

Figure 5.12: Accuracy achieved by latent learning for differentiating among handwritten digits as neural net with 6400 output nodes is trained to differentiate among 25 handwritten characters with 775 samples per character. The ratio of output nodes to input nodes is about 6.25

been observed. Although the amount of forgetting reflected by this is far from catastrophic, it is still far too large to be ignored.

In spite of these deficits, latent learning has been shown to be a promising technique for learning with neural nets. The areas where its utility have not been demonstrated are a reflection of the novelty of the attempt to use latent learning, rather than an inherent limitation of latent learning.

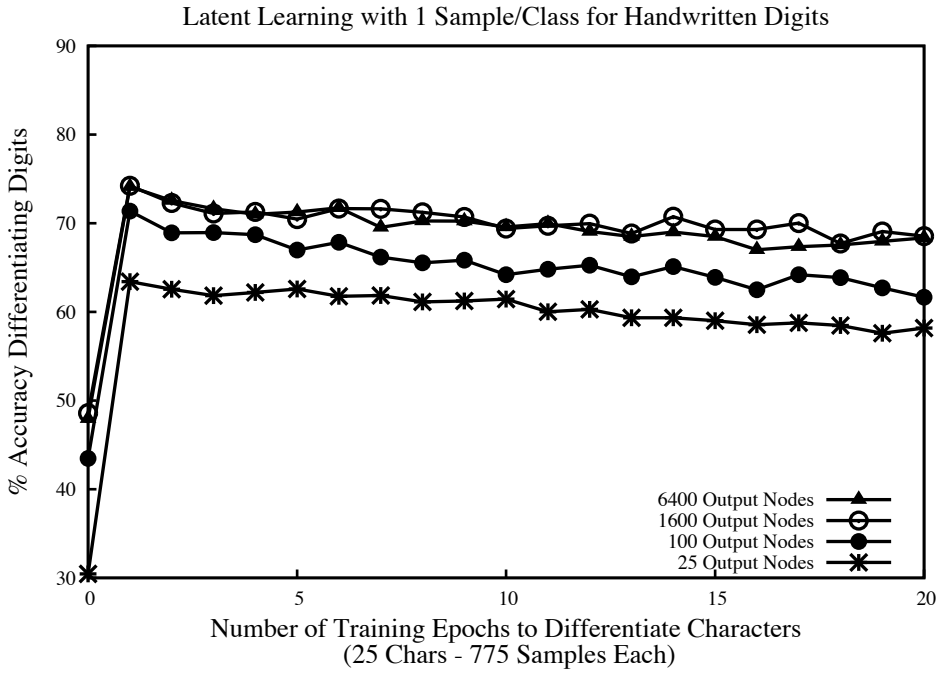

Figure 5.13: Accuracy achieved by latent learning for differentiating among handwritten digits with only 1 sample per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

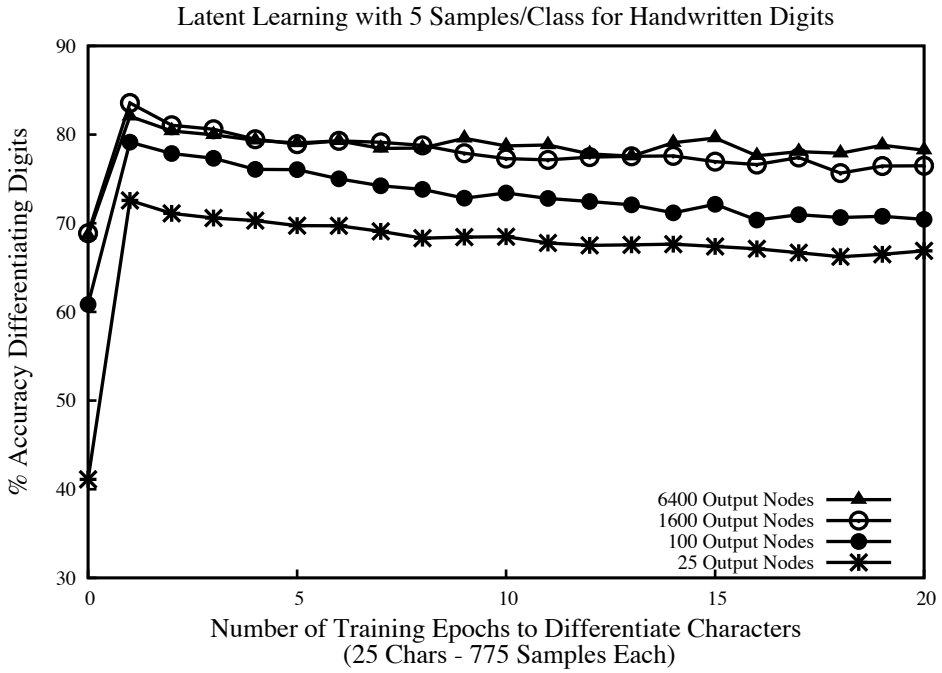

Figure 5.14: Accuracy achieved by latent learning for differentiating among handwritten digits with only 5 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

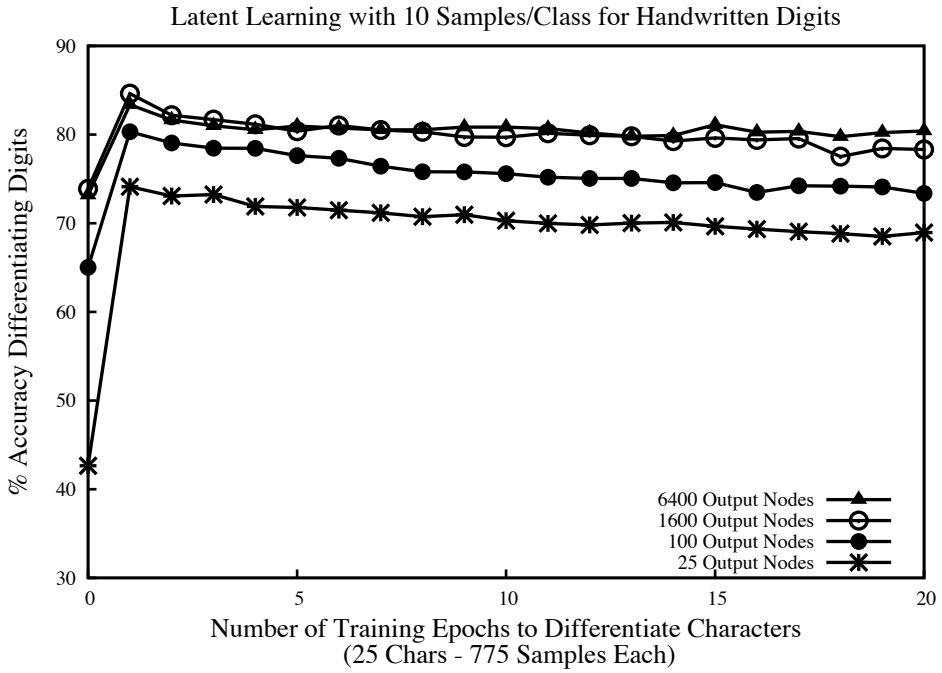

Figure 5.15: Accuracy achieved by latent learning for differentiating among handwritten digits with only 10 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

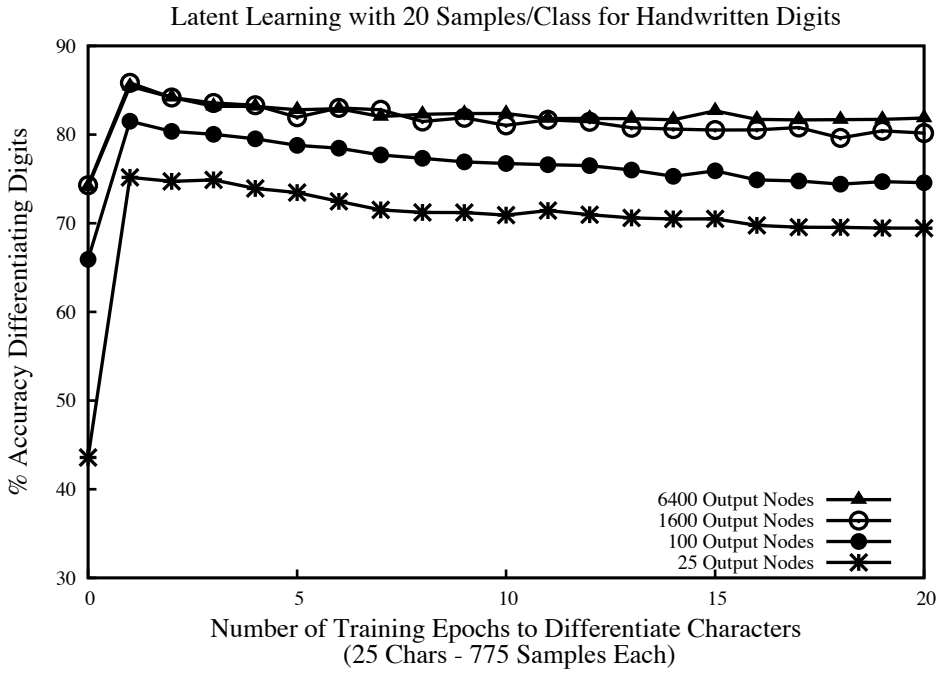

Figure 5.16: Accuracy achieved by latent learning for differentiating among handwritten digits with only 20 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

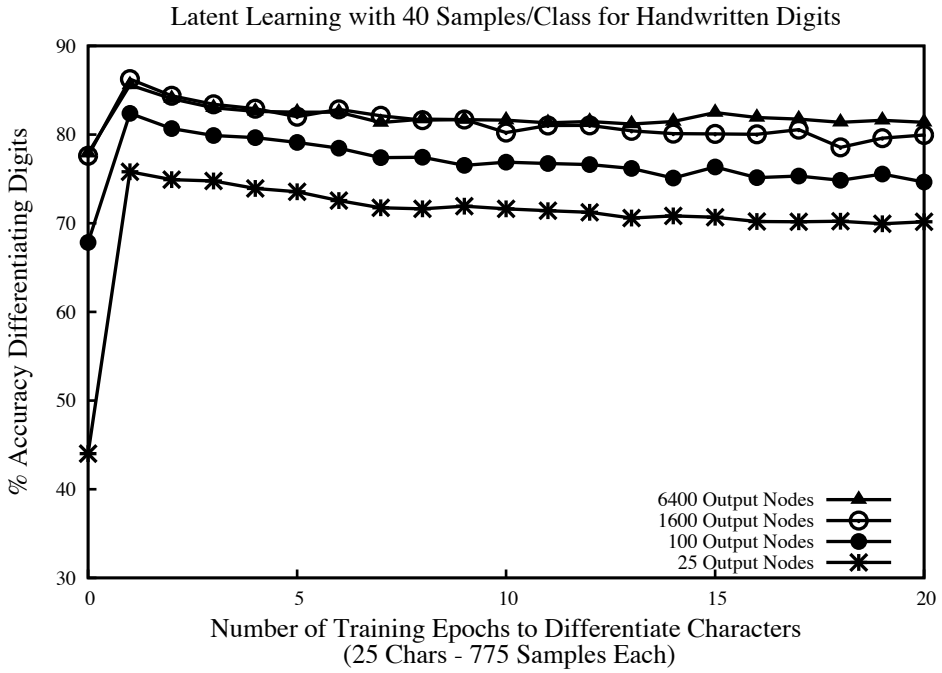

Figure 5.17: Accuracy achieved by latent learning for differentiating among handwritten digits with only 40 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

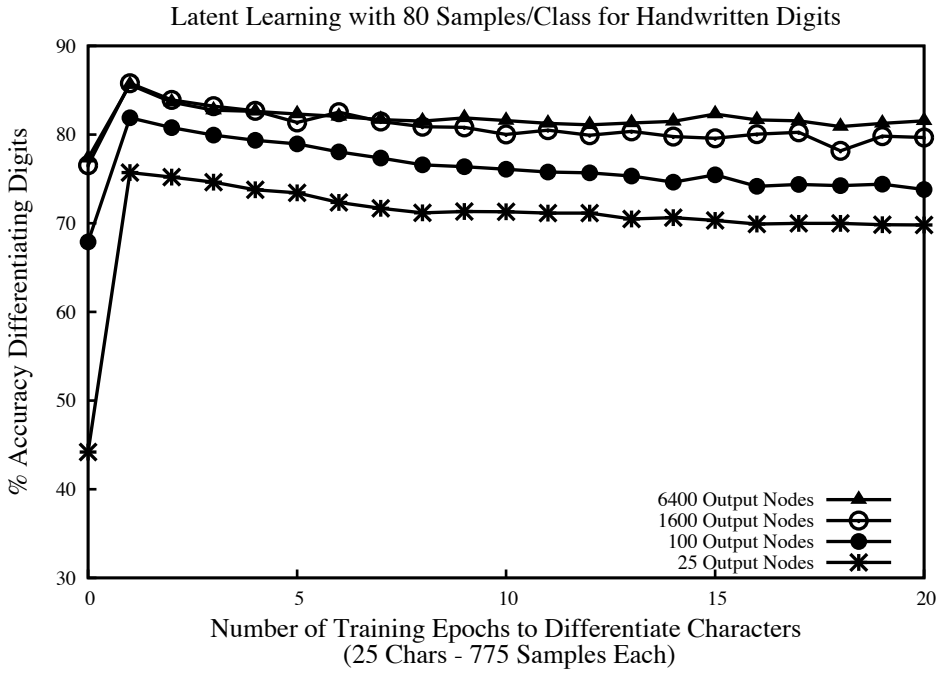

Figure 5.18: Accuracy achieved by latent learning for differentiating among handwritten digits with only 80 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

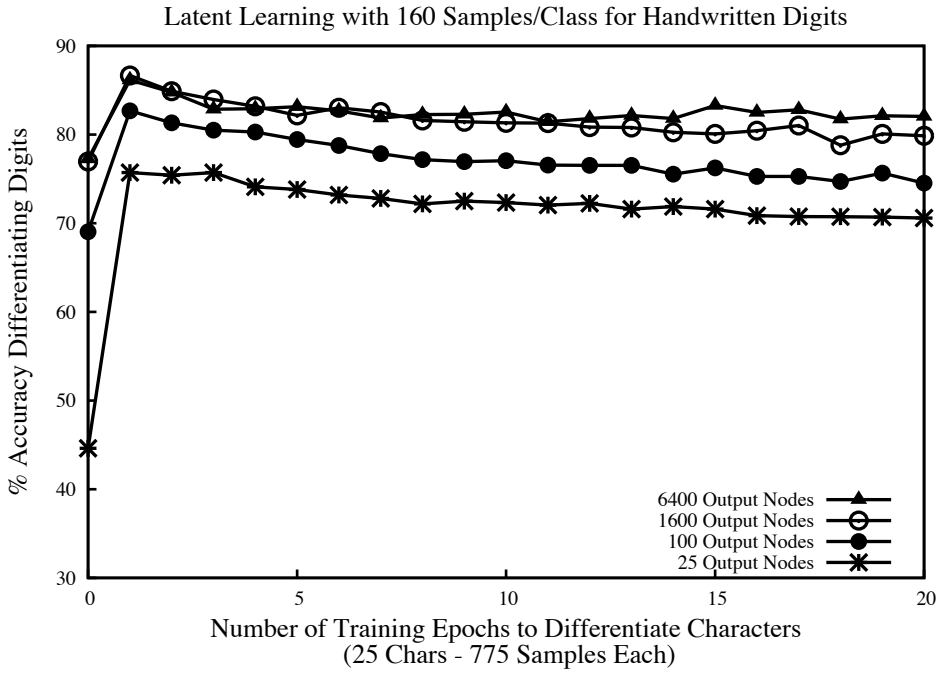

Figure 5.19: Accuracy achieved by latent learning for differentiating among handwritten digits with only 160 samples per digit as neural nets with various numbers of output nodes are trained to differentiate among 25 handwritten characters with 775 samples per character. Results are given for output to input node ratios of 6.25, 1.56, 0.10 and 0.02.

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

'Reinventing the wheel' is rarely regarded as a productive use of time. Yet, when neural nets are trained, it is nearly always what happens. The main reason for this situation is the difficulty in interpreting the solutions found by neural nets for given tasks. Without understanding what part any individual node plays in enabling a net to perform a given task, it is, to put it mildly, quite difficult to determine which nodes should be transferred to create a net that will master a new task more quickly, or even how to perform such a transfer. Overcoming this handicap has many obvious benefits, such as not needing very large, labeled training sets, obtaining faster training times and never losing the ability to learn new things.

In this thesis, two approaches for reusing existing nets to learn new skills were examined. The first approach, structurally based knowledge transfer, involved using the 'deep' quality of a deep net to help determine which nodes could be reused for new tasks. It was observed that when such a net was trained to perform a set of tasks, then the nodes closest to the raw input could be reused to help learn a new set of related tasks. Unlike earlier attempts, the choice of which nodes to transfer is made without any direct consideration of how those nodes respond to the new set of tasks. As neural nets develop the ability to engage in life long learning, it seems likely that reusing lower layers of nets in this way will be a productive technique.

The other approach that I developed, latent learning, was based upon the observation that neural nets will respond to inputs prior to any training. After confirming that these responses not only contained information about the inputs, but contained sufficient information to distinguish among them in useful ways, experiments were performed to optimize these inherent responses.

The approach taken by these experiments was to first train the net to distinguish among a set of classes, A, and then to examine the inherent bias for a similar set of classes, B. As expected, these inherent biases were far more accurate than those obtained from a net that had never been trained for anything. This was due to latent learning.

Latent learning occurs when something is learned without being specifically taught - i.e. no positive or negative reinforcement is used to enable learning. There is no evidence of the knowledge acquired, until it is specifically needed by the learner. Usually, this is seen by a learner passively observing the environment. In our case, it occurred by assuming some degree of logical consistency for a neural net's output patterns for different inputs.

It seemed evident that when a neural net is used to differentiate among a set of classes, that it will perform best when the degree of similarity between any two classes is reflected in the degree of similarity between the net's representation of them. The greater the degree to which this holds true for a particular neural net, the easier it should be to train and use the net for such differentiation. Furthermore, once such training has occurred, the net will have a strong tendency to represent other classes from that set of classes in a self-consistent manner. This tendency is a form of latent learning, since the 'consistent' representations have been learned without specific training. More generally, it is a form of transfer learning, since the knowledge gained by learning to differentiate among the first set of classes was used to learn to differentiate among the second set of classes. The experiments performed in this thesis are, to my knowledge, the first to explicitly demonstrate latent learning in neural nets.The potential advantages promised by latent learning make it a very promising area for further research.

6.2 Future Directions

6.2.1 Better Determining Inherent Biases

The approach we used for determining a net's inherent bias for a given class of input was to take a set of n samples from a particular class. Then the average response of each output node

to that class was considered. If the average output of a node was negative, its inherent bias for that class was assumed to be -1, otherwise the inherent bias was assumed to be $+1$. The set of inherent bias of the output nodes formed the target point for that class. The similarity of an input to a particular class was determined by the Euclidean distance between the output vector that the input generated and the target point for that particular class.

When determining the target point, if a node's average output has a magnitude that's fairly small, less than 0.5 for example, then it seems probable that assigning it a value of $+/-1$ in the target vector may not be the right thing to do. It's entirely possible for that particular node to have a non-indicative response with respect to a particular class. Therefore, it might be a better decision not to include that node in the target vector for that class.

Another indication of how relevant a particular output node might be for deciding if a given input belongs to a particular class would be to observe the variance of its responses to sample members of that class. An output node with a highly varying response with respect to input from a particular class is most likely not a good indicator for that class.

Rather than deciding in a binary fashion whether a node should be used to determine when a given input should be associated with a particular class, a node's influence in that decision could be weighted. This weighting could be reflected in the calculation of the Euclidean distance between the target point for a particular class and the output produced by a current input.

Usually, this distance is calculated as follows:

$$\sum_i (y^i_{target} - y^i_{output})^2 \tag{6.1}$$

where, y^i_{target} is the i^{th} coordinate of the target vector and y^i_{output} is the i^{th} coordinate of the output vector. Instead, the distance could be calculated with the minor modification of a weighting factor:

$$\sum_i \nu^i (y^i_{target} - y^i_{output})^2 \tag{6.2}$$

This weighting could be provided by the actual correlation between that node's output and

a binary variable $+/-1$ indicating whether a particular input did/did not belong to the given class. This correlation would be measured during the 'training' phase, when the inherent bias is estimated.

6.2.2 Creating Better Inherent Biases

All the latent learning experiments performed in this thesis had the first set of classes start with a stochastically constructed set of desired responses (i.e. target points). This is clearly not optimal. There is no guarantee that the differences in similarity between classes will be faithfully represented in the Euclidean distances between their associated target vectors.

In order to create an initial set of representations that are more self consistent, it might be a good idea to train the net's representations of the initial set of classes in an unsupervised manner. Two possible techniques for doing this include using Siamese nets [8, 12] for learning the initial set of classes or using a greedy layerwise training technique for an auto encoding net, similar to techniques developed by Hinton [25, 26, 27]. In both approaches, the net learns an internal representation that helps to minimize a loss function that does not have parameters arbitrarily determined by a user. Siamese nets have a loss function that tries to give inputs from the same class internal representations that are as similar to each other as possible, if not identical, while trying to ensure that inputs from different classes have very different internal representations. Auto encoding nets try to compress an input in such a way that the original input can be recreated from the compressed form. In both cases, an internal representation should form where the similarities and dissimilarities between classes will be mirrored in those between the net's representation of those classes. This should create better inherent bias for latent learning.

Another approach would be to train the net on the initial set of classes and then use a confusion matrix to see which classes are most commonly confused. At this point extra nodes could be added with the express purpose of disambiguating those classes.

Finally, a loss function could be introduced that attempts to decorrelate the output of internal nodes at the same level of the net. Fahlman and Lebiere [19] had observed the 'herd effect'

almost 20 years ago. This effect is the tendency of nodes to learn the same feature at the same time. Their approach to avoid this and have the nodes learn uncorrelated features led to their Cascade-Correlation training approach, which worked without backpropagation and would practically construct a net one node at a time. More recently, Bergstra and Bengio [7] attempted to create a loss function that would force internal nodes to be uncorrelated with each other. However, their technique relies upon batch training and monitoring the actual behavior of each node through an entire epoch in order to keep the nodes uncorrellared. This slows down training for large datasets.

Another possible approach would be to consider the weight parameters of a node to be a vector. Then, a loss function could be created that puts 'pressure' on these vectors to be as nearly mutually perpendicular as possible. If the 'weight' vectors for two different nodes are parallel/anti-parallel or nearly so, then those two nodes will have highly correlated/anti-correlated output. By discouraging such parallelism, correlation between nodes becomes less likely. If such a loss function proved too expensive to compute, then, perhaps instead of starting weights with random valuess, initialization could be used to ensure that the weight vectors would be less parallel to each other.

6.2.3 Catastrophic Forgetting

One of the major problems with neural net is the fact they experience catastrophic forgetting. Whenever a net which has been trained for one task(s) is trained for a new task(s), it loses the ability to perform the first task(s). Because our use of the net's inherent bias was, in some sense, fully consistent with what the net had already learned, it was hoped that even after some traditional supervised training for the classes that were latently learned, that the net would retain its ability to distinguish among the first set of classes among which it learned to discriminate.

Initial results showed a decease in accuracy for distinguishing among the first set of classes that was about 15% (i.e. a drop from accuracy in the low/mid 90%'s to accuracies of about 79% - 80%). Although this drop is hardly catastrophic, it is not negligible. Some approaches to

reducing this effect include:

1. Having the net retrain on idealized versions of classes it already has learned while training on the new, latently learned classes

2. Placing limits on the degree to which a node's input weight vector is allowed to change when learning a new class. The main concern would be to make certain that this vector did not rotate too much.

3. A net's architecture is generally fixed. It might be advantageous to add new nodes when learning additional classes. This way, it also becomes more reasonable to 'freeze' some of the old nodes, so that their behavior does not change. Furthermore, when the new nodes learn new features from the additional classes, these nodes may be a source of latent learning for the old set of classes, just as the original nodes were a source of latent learning for the additional classes.

6.2.4 Self Initiated Learning

Latent learning suggests a path for life long learning and for learning with very few data samples. By using a net's latent responses to guide its responses for new classes of data one should be able to eliminate, or at least greatly reduce the degree of catastrophic forgetting that takes place when learning to recognize new classes. Furthermore, by taking advantage of existing responses, the net will require fewer labeled samples in order to master recognitiion of new classes of input. This ability, in conjunction with structurally based transfer, should be able to greatly reduce training time necessary for neural nets with little to no loss in accuracy. Additionally, a sophisticated latent learner should be able to recognize for itself when it encounters a member of some class it hasn't seen before. One can imagine this occurring when an input generates a response that is too different from existing responses to belong to any known class, yet somehow shows evidence of possessing a sufficient number of important features to be a new class, much in the way a child, raised with only a knowledge of animals native to a desert will immediately recognize a seal

as a new class of animal. Such recognition should enable a machine to automatically decide to train itself, without any human intervention. A learner that needs few examples in order to learn, and which is capable of realizing when it needs to learn can be placed into strange environments without the need for human supervision.

6.2.5 Demonstrating Techniques On Other Problems

All the work presented in this thesis has involved the problem of recognizing handwritten characters. However, this was not due to an overweening interest in this problem for its own sake. This problem was only of interest as a test bed for the methods of transfer learning that were studied.

In order for structurally based knowledge transfer and latent learning to be truly significant techniques, they will have to demonstrate their efficacy on other problems. Such problems could come from the some of the recognition tasks in the field of biometrics, such as face, finger print, retinal, voice or gait recognition. Other suitable problems would include accent recognition, text classification or even various types of object recognition. With more effort and imagination, it should be possible to use variants of these techniques to help robots learn new physical skills.

Undoubtedly, there are several additional applications that have not been listed here. The wide variety of potential applications indicates not only the possible benefits of latent learning techniques, but also the importance of transfer learning techniques for future machine learning applications.

References

[1] Yasser Abu-Mostafa. Learning from hints. *Journal of Complexity*, 10(1):165–178, 1994.

[2] Defense Advanced Research Projects Agency. DARPA-BAA-09-40: Deep Learning, 2009.

[3] Amr Ahmed, Kai Yu, Wei Xu, Yihong Gong, and Eric Xing. Training hierarchical feed-forward visual recognition models using transfer learning from pseudo-tasks. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 69–82, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[5] Jonathan Baxter and Peter L. Bartlett. The canonical distortion measure in feature space and 1-NN classification. In *Neural Information Processing Systems Conference*, 1997.

[6] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In Leon Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*, 2007.

[7] James Bergstra and Yoshua Bengio. Slow, decorrelated features for pretraining complex cell-like networks. In Yoshua Bengio, Dale Schuurmans, Christopher Williams, John Lafferty, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 (NIPS'09)*, 2009.

[8] Jane Bromley, James W. Bentz, Leon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Sackinger, and Roopak Shah. Signature verification using a siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4), 1993.

[9] Rich Caruana. Learning many related tasks at the same time with backpropagation. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 657–664. The MIT Press, 1995.

[10] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[11] Rich Caruana. *MultiTask Learning*. Ph.D. in Computer Science, Carnegie Mellon, 1997.

[12] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proc. of Computer Vision and Pattern Recognition Conference*. IEEE Press, 2005.

[13] Andy Clark and Chris Thornton. Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences*, 20:57–90, 1997.

[14] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[16] Thomas G. Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Department of Computer Science, Oregon State University, Corvalis, Oregon, 1995.

[17] J.L. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.

[18] Theodoros Evgeniou, Charles A. Micchelli, and Massimiliano Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637, 2005.

[19] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[20] Patrick Grother. NIST Special Database 19, 2002.

[21] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. Knowledge transfer in deep convolutional neural nets. In *FLAIRS Conference*, pages 104–109, 2007.

[22] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. Knowledge transfer in deep convolutional neural nets. *International Journal on Artificial Intelligence Tools*, 17(3):555–567, 2008.

[23] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. The utility of knowledge transfer for noisy data. In *FLAIRS Conference*, pages 59–64, 2008.

[24] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. Training to a neural net's inherent bias. In *FLAIRS Conference*, 2009.

[25] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[26] Geoffrey E. Hinton. The next generation of neural nets. Invited Google Talk - url:http://www.youtube.com/watch?v=AyzOUbkUf3M, 2007.

[27] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006.

[28] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI*, pages 646–651, 2008.

[29] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[30] Charles A. Micchelli and Massimiliano Pontil. Kernels for multi–task learning. In *Neural Information Processing Systems Conference*, 2004.

[31] T. M. Mitchell and S. B. Thrun. Explanation-based neural network learning for robot control. In C. L. Giles, S. J. Hanson, and J. D. Cowan, editors, *Advances in Neural Information Processing Systems 5, Proceedings of the IEEE Conference in Denver*, San Mateo, CA, 1993. Morgan Kaufmann.

[32] Tom Mitchell. *Machine Learning*. WCB/McGraw-Hill, 1997.

[33] Tom M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, New Jersey, 1980.

[34] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, pages 841–848, 2001.

[35] Dean A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance*. Kluwer Academic Publishers, Norwell, MA, USA, 1993.

[36] Lorien Y. Pratt. Discriminability-based transfer between neural networks. In *Neural Information Processing Systems Conference*, pages 204–211, 1992.

[37] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An artificial intelligence approach*, pages 463–482. Morgan Kaufmann, 1993.

[38] Francois Rivest and Thomas Schultz. Application of knowledge-based cascade-correlation to vowel recognition. In *IEEE International Joint Conference on Neural Network 2002*, pages 53–58. IEEE Society Press, 2002.

[39] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11, 2007.

[40] Thomas Schultz, Francois Rivest, L. Egri, and J.P. Thivierge. Knowledge-based learning with KBCC. In *Proceedings of the Fifth International Conference on Development and Learning ICDL 2006*, 2006.

[41] Thomas R. Shultz and François Rivest. Knowledge-based cascade-correlation. In *International Joint Conference on Neural Networks (5)*, pages 641–646, 2000.

[42] Thomas R. Shultz and François Rivest. Knowledge-based cascade-correlation: using knowledge to speed learning. *Connection Science*, 13(1):43–72, 2001.

[43] Daniel Silver. *Selective Transfer of Neural Network Task Knowledge*. Ph.D. in Computer Science, University of Western Ontario, 2000.

[44] Daniel Silver and Robert Mercer. Selective functional transfer: Inductive bias from related tasks. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC2001)*, pages 182–189, 2001.

[45] Daniel Silver and Robert Mercer. The task rehearsal method of life-long learning: Overcoming impoverished data. In Bruce Spencer Robin Cohen, editor, *Advances in Artificial Intelligence, 15th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2002)*, volume 2338, pages 90–101, 2002.

[46] Daniel Silver and Robert Mercer. Sequential inductive transfer for coronary artery disease diagnosis. In *Proceedings of International Joint Conference on Neural Networks (International Joint Conference on Neural Networks-07)*, pages 2635 – 2641, 2007.

[47] Daniel Silver and Ryan Poirier. The sequential consolidation of learned task knowledge. In *Advances in Artificial Intelligence, 17th Conference of the Canadian Society for Computational Studies of Intelligence (AI 2004)*, pages 217–232, 2004.

[48] Daniel L. Silver. The parallel transfer of task knowledge using dynamic learning rates based on a measure of relatedness. *Connection Science*, 8(2):277–294, 1996.

[49] Patrice Simard, Yann LeCun, John S. Denker, and Bernard Victorri. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*, pages 239–27, 1996.

[50] S.N. Srihari, T. Hong, and Z. Shi. Cherry blossom: A system for reading unconstrained handwritten page images. In *Symposium on Document Image Understanding Technology*, 1997.

[51] David J. Stracuzzi. Scalable knowledge acquisition through memory organization. In *Helsinki University of Technology*, pages 57–64, 2005.

[52] J.P Thivierge and Thomas R. Shultz. Finding relevant knowledge: KBCC applied to splice-junction determination. In *International Joint Conference on Neural Networks*, pages 1401–1405, 2002.

[53] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Neural Information Processing Systems Conference*, pages 640–646, 1995.

[54] Sebastian Thrun and Joseph O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *International Conference on Machine Learning*, pages 489–497, 1996.

[55] Paul E. Utgoff and David J. Stracuzzi. Many-layered learning. *Neural Computation*, 14(10):2497–2529, 2002.

[56] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, pages 511–518, 2001.

[57] Alexander Waibel, Hidefumi Sawai, and Kiyohiro Shikano. Modularity and scaling in large phonemic neural networks. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, volume 37, pages 1888–1898, 1989.

Appendix A

On the Set Size of Accurate Inductive Hypotheses

Consider the simple, yet common, task of 2-dimensional curve fitting. A learner is given a finite set of (x, y) points, where $x \in \mathbb{R}^1$ and $y \in \mathbb{R}^1$. Within the set of pairings, no two pairs share a common x-value, although several may share a common y-value. The learner is then asked to find a hypothesis that accurately describes the underlying process which generated those points. This hypothesis takes the form of a function $y = f(x)$, where the accuracy of the function will be judged by how well it manages to generate the correct y for a given x.

In order to understand how many functions, $f(x)$, would serve as an accurate hypothesis, consider the following:

When given a set of objects, the size of that set is referred to as the *cardinality* of that set. If the set has a finite number of elements, then its cardinality is equal to that finite number. However, it is possible for sets to be infinite. For example, the set of counting numbers, which are also referred to as 'natural' numbers - $\mathcal{N} = \{0, 1, 2, 3, ...\}$ is clearly infinite. Yet, it is also *countable*, because one can create an algorithm which will, eventually, list every element of this set. The cardinality of the set of counting numbers is referred to as $\aleph_0$. No countable set has a larger cardinality than $\aleph_0$ nor does any infinite set possess a smaller cardinality than $\aleph_0$.

The set of real numbers has a greater cardinality than the set of natural numbers. Although it is possible to create a mapping so that for every natural number there is a unique real number associated with it, one cannot create a mapping such that for every real number there is a unique natural number associated with it. For this reason the set of real numbers is considered larger than the set of natural numbers and uncountable. The cardinality of the set of real numbers is

referred to as $\aleph_1$. Additionally, the power set of a set (i.e. the set of all subsets of a given set) with cardinality $\aleph_1$ has cardinality $\aleph_2$ (Note: This is actually generalizable. Any set with cardinality $\aleph_\omega$ has a power set with cardinality $\aleph_{\omega+1}$).

These observations about infinite sets, their countability and cardinality are important because the set of all possible functions, $y = f(x)$, where $x \in \mathcal{R}^1$ and $y \in \mathcal{R}^1$, has cardinality $\aleph_2$, since a one-to-one mapping may be created between it and the power set of real numbers. Similarly, the set of all functions, $y = f(x)$, which are consistent with a given finite set of (x,y) points also has cardinality $\aleph_2$. So, when attempting to find a function, $y = f(x)$, that is consistent with a set of data points, (x,y), not only is the set of possible hypotheses uncountably large, so too is the set of reasonable hypotheses. If we were to impose a fairly mild restriction, born of experience for most natural processes, that only continuous functions would be considered as explanatory hypotheses, then the set of possible hypotheses would have cardinality $\aleph_1$ - still infinite and still uncountable.

It is only when we begin to restrict the hypothesis space to a specific family, or families, of functions with arguments of finite precision that the space of hypotheses becomes countable. Yet, both the set of possible hypotheses and the set of accurate hypotheses remain infinite.

Even in the extremely restricted case of attempting to learn a Boolean function, the hypothesis space can still be quite large. Consider a Boolean function where the domain consists of N Boolean inputs and a single Boolean output. The hypothesis space for this function consists of 2^{N+1} Boolean functions. For each example encountered, only one eligible function is removed. So, the set of hypotheses that are completely consistent with experience for a Boolean function grows exponentially with number of inputs but shrinks only linearly with the number of given examples. This is in spite of the extreme restrictions on the hypothesis space. Clearly, when searching for explanatory hypotheses that agree with a particular set of data, frequently there is an embarrassment of possibilities.