Appears in the Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95).

Combining the Predictions of Multiple Classifiers: Using Competitive Learning to Initialize Neural Networks*

Richard Maclin
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706 USA

Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706 USA

Abstract

The primary goal of inductive learning is to generalize well – that is, induce a function that accurately produces the correct output for future inputs. Hansen and Salamon showed that, under certain assumptions, combining the predictions of several separately trained neural networks will improve generalization. One of their key assumptions is that the individual networks should be independent in the errors they produce. In the standard way of performing backpropagation this assumption may be violated, because the standard procedure is to initialize network weights in the region of weight space near the origin. This means that backpropagation's gradient-descent search may only reach a small subset of the possible local minima. In this paper we present an approach to initializing neural networks that uses competitive learning to intelligently create networks that are originally located far from the origin of weight space, thereby potentially increasing the set of reachable local minima. We report experiments on two real-world datasets where combinations of networks initialized with our method generalize better than combinations of networks initialized the traditional way.

1 Introduction

The main goal of classification learning is *generalization* – being able to induce a concept that accurately classifies future examples. The difficulty with achieving good generalization is that a learner cannot measure generalization directly; instead, the learner relies on its inductive bias to *hopefully* produce an accurate classifier. A number of schemes have been introduced to address this ubiquitous problem, ranging from tree-pruning in decision trees [Quinlan, 1987] to the use of complexity terms in neural networks [Hinton, 1989]. In this paper we present a way to address the generalization problem for neural networks trained by backpropagation [Rumelhart *et al.*, 1986]. Our solution employs the previously explored approach [Drucker *et al.*, 1994; Hansen and Salamon, 1990; Lincoln and Skrzypek, 1990; Rogova, 1994] of training a set of networks rather than a single network and then combining the results. The novelty of our approach lies in the type of networks we combine to produce our composite classifier.

When using backpropagation, one typically minimizes a cost function. This function is a measure of the errors made by the learner, and perhaps other terms, as discussed below. For example, one straightforward cost function is the sum of squared errors between the predicted and target outputs for the training data. Backpropagation learns by making changes to the network that reduce the total cost.

However, minimizing the cost function does not guarantee optimal generalization. Generalization can be harmed by a number of factors: the learner may have an inappropriate network topology for the problem; the parameters of the learning algorithm, such as the learning rate, may be inappropriate; there may be too few examples to learn the concept; and the learner may "overfit" its model to correctly classify noisy data. Many methods have been introduced to solve these problems. Some researchers introduce penalty terms into the cost function that favor generalization [Hinton, 1989]. Others "avoid" overfitting using stopping criteria like patience [Fahlman and Lebiere, 1990] or by employing validation sets [Lang *et al.*, 1990]. Some research has focused on choosing an appropriate network by growing [Fahlman and Lebiere, 1990] or shrinking [Le Cun *et al.*, 1989] a network topology, or by exploring a number of topologies [Opitz and Shavlik, 1993]. Combining multiple networks[1] has the advantage of achieving good generalization without having to do a great deal of empirical exploration or complex coding. The obvious disadvantage is the time spent training multiple networks, but this disadvantage can be mitigated, because it is trivial to parallelize the training, since each network can be trained separately.

In Section 2 we discuss Hansen and Salamon's [1990] model of combining the predictions of multiple classifiers. A key aspect of their work is that to increase generalization, the mistakes made by the networks combined

*This research was partially supported by ONR Grant N00014-93-1-0998 and NSF Grant IRI-9002413.

[1] We will use the phrase *combining multiple networks* as a shorthand to mean *combining the **predictions** made by multiple neural networks.*

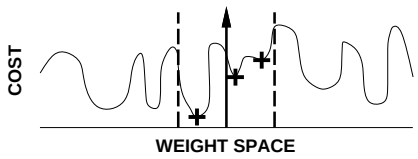

Figure 1: Consider a cost function described over a simple weight space. Assuming the initial value of the weight parameter falls between the two dashed lines, the only local minima that can be reached by gradient descent are those marked with crosses (+), even though many other local minima may exist.

must be independent. The problem, of course, is how to achieve "independent" networks. In order to achieve independence in networks, one would expect to have to greatly vary a number of aspects of the networks including (but not limited to) the set of training patterns, the topology of the networks, the learning parameters of the networks, and the method for initializing the networks. But we have investigated a method that empirically produces good results for combining networks that only requires the user to vary the method of initializing the networks. We do not mean to claim that the resulting networks are truly independent, merely that the networks initialized as we suggest are less interdependent that networks initialized in the normal manner, and therefore more effective when used in combination.

Our idea is to initialize the parameters, (i.e., weights and biases) of a neural network over a much larger range of values than normal. We do this to cause backpropagation to seek local minima not normally encountered using standard initialization (see Figure 1).

We explored a number of means to achieve our goal of a diverse set of starting points in weight space; the best one is a method employing competitive learning [Rumelhart and Zipser, 1985] to produce class prototypes that are used as hidden units in a network in a method similar to Moody and Darken's [1988; 1989]. In Section 3 we outline competitive learning, and in the following section we discuss how to use it to initialize neural networks. Section 5 shows experiments we performed on two real-world data sets, demonstrating the advantages of our approach. We then discuss future research directions, related work, and our conclusions.

2 Combining Multiple Networks

The basic idea in combining neural networks is to train a number of networks, and then somehow use the collection to increase generalization. Figure 2 shows a framework for combining multiple networks.

The choice of a function for combining predictions is important [Kearns and Seung, 1995]. Examples of combination functions include voting schemes [Hansen and Salamon, 1990], simple averages [Lincoln and Skrzypek, 1990], weighted average schemes [Perrone and Cooper, 1994; Rogova, 1994], and schemes for *training* combiners [Rost and Sander, 1993; Wolpert, 1992; Zhang *et al.*, 1992]. We chose as our method for combining networks the scheme of simply averaging the values of the corresponding output units. Results from portfolio theory suggest this method provides a good solution with minimal extra work [Mani, 1991].

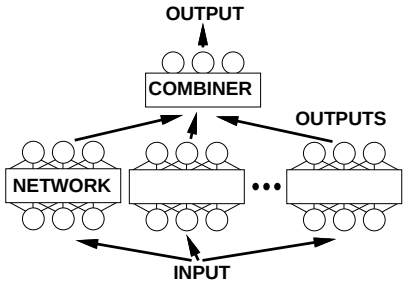

Figure 2: Basic framework for combining multiple network predictions.

Hansen and Salamon [1990] explored the value of combining multiple networks. They demonstrated that under certain assumptions, generalization will increase as more networks are combined. For each pattern where the average error rate is less than 50% for the possible trained networks, Hansen and Salamon demonstrate that in the limit the expected error on that pattern can be reduced to zero. Of course, since not all patterns will necessarily share this characteristic (e.g., outliers may be predicted at more than 50% error), the error rate over all the patterns cannot necessarily be reduced to zero. But if we assume a significant percentage of the patterns are predicted with less than 50% average error, gains in generalization will be achieved. A key assumption of Hansen and Salamon's analysis is that the networks combined should be independent in their production of errors.

Typically, the weights of a neural network are randomly initialized with small values. The problem with this approach is that this tends to confine the exploration of the network to a single area of weight space (Figure 1). Given similar training sets, this means that multiple networks may often find the same local minima. In order to produce networks that are less interdependent in their predictions, we investigated methods that avoided this bias. In particular, we focused on changing the standard means of initializing network weights to produce networks that have large initial weights.

Our first approach was the obvious solution – use a much larger random range for generating initial network weights. This works well in terms of producing large weights, but the resulting networks suffer from the *flat-spot* problem [Fahlman, 1988] – backpropagation is unable to refine these networks since the activation values of the hidden and output units tend to be very close to 0 or 1.[2] This occurs because the net input values for a network unit will often be highly positive or negative.

In our second approach we tried to randomly pick examples from the training data to act as "prototypes." We then created a hidden unit for each prototype with large positive weights for each of the "on" features of the prototype. This solution also suffered from the flat-spot problem, since in a large input space only a small number of examples will share a large set of features with another example and, hence, the hidden units are typically inactive. Using what we learned from this ap-

[2]Error is multiplied by $activation(1 - activation)$ when backpropagating, assuming a logistic activation function and a sum-of-squared-errors cost function. For units with activation near 0 or 1, this means that little error is propagated.

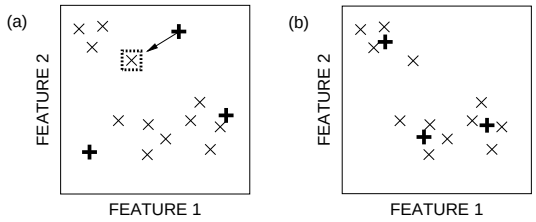

Figure 3: Phases of competitive learning (x = example, + = seed); (a) during training the seed nearest to a example is updated by moving the seed in the direction of the example; (b) after training seeds represent the centers of *clusters* of examples.

proach, we switched to an approach where we produced hidden units that represented *groups* of patterns rather than single patterns. Our solution makes use of the technique of competitive learning, which we outline next.

3 Competitive Learning

Competitive learning [Rumelhart and Zipser, 1985] is a simple unsupervised learning technique that can be used to partition a set of examples into classes. We start competitive learning by randomly creating a set of class seeds (which are simply points in a feature space). The seeds then compete for the training examples; a seed "wins" an example if it is closest to the example. When a seed wins the competition for an example, the seed is updated by moving it towards that example. Over a number of training epochs, the seeds will move to locations in feature space that represent groups (classes) of similar examples (see Figure 3).

We can view competitive learning as a technique to find a set of "seeds" which represent the average member of that class. Each seed j has a value for each input feature i which we will refer to as $w_{i,j}$. To calculate the class of an example, we first evaluate a closeness-of-fit metric for each seed, $\sum_{i \in inputs} |w_{i,j} - f_i|$, where f_i is the value of feature i in the example. The seed j with the lowest total difference is the "winner" for the example (i.e., the example is a member of this class). Learning can be done by changing the average value of input i for the winning seed by an amount $\mu(f_i - w_{i,winner})$, where μ is the learning rate (we used $\mu = 0.01$). This process is repeated for each of the examples for a number of epochs (we used 50 epochs).

4 Combining Networks Initialized via Competitive Learning

Table 1 contains our algorithm for training multiple neural networks initialized via competitive learning.

Competitive learning takes a set of examples and partitions it into J classes (where J is user defined). For the following discussion we will refer to the clusters produced by competitive learning as *subclasses*, since each represents a subclass of *one* of our output categories. We want to create a network that has a hidden unit corresponding to each of the subclasses of each category, where each hidden unit will have an activation near 1 for the examples in its subclass and 0 otherwise. Figure 4 shows a sketch of the resulting network.

Table 1: Algorithm for initializing K networks using competitive learning. J is the number of hidden units that will be in the resulting network.

Repeat K times
- Randomly choose J seeds, then use competitive learning to create subclasses for each category of examples
- Use the subclasses for all of the categories to initialize a neural network
- Train the resulting network using backpropagation

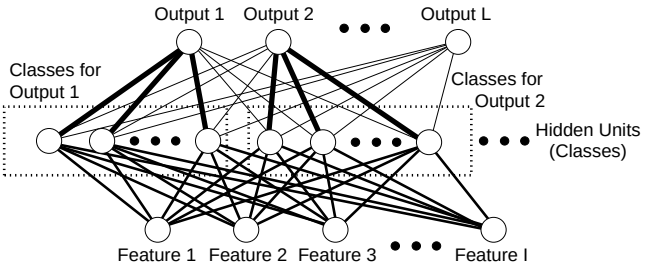

Figure 4: Network combining the results of competitive learning. Input features are connected to hidden units representing subclasses using weights determined by competitive learning. Classes are connected to their output category's unit with a large weight and to other output categories with a small random weight.

In our network we initialize the links between an output unit and the hidden units representing the subclasses of that category with large positive links (uniformly in [3.9, 4.1]), and set the bias of the output unit so that if any of the subclasses are active, the output unit will be active. We also connect the hidden units associated with a particular output unit to the other output units, using small randomly-weighted links (uniformly in [-0.1,0.1]); this is done so that backpropagation can use recognizers of subclasses defined for one output category in recognizing other output categories.

Finally, we have the question of how we create a hidden unit that recognizes a subclass defined by competitive learning. The weights $w_{i,j}$ of the seed of a competitive learning class represent the average feature values of examples in that subclass. If we assume that all of the input features are Boolean[3], each weight represents how important a particular feature is for an example to be part of that subclass. Recall that we want to create a hidden unit whose weights are dependent on the competitive learning seed's weights, and which has an average activation value near 1 for examples in that subclass and 0 for others.

The first thing we do is determine the average activation value (a) we want the unit to have when it is "on" and similarly for "off." Since we are using the sigmoidal activation function:

$$a = \frac{1}{1 + e^{-netinput}} \quad (1)$$

we can determine what value of *netinput* is needed to produce a particular average activation value by solving

[3]This assumption can be relaxed using a more complex recognition scheme similar to the fuzzy recognition scheme presented by Berenji and Khedkar [1992].

for *netinput* in Equation 1. For example, if we want our hidden unit to have an average activation value a of 0.9 when it is "on" we would need $netinput = 2.19$. Let us assume we have determined the average *netinput* values net_{on} and net_{off}, which are the required *netinput* values for our desired average activation values.

Next we create a hidden unit that has weights copied from the competitive learning seed. We use this hidden unit to calculate the average net input for the examples that are part of that subclass, and similarly the average net input for examples that are not part of the subclass (call these values $avg_{inclass}$ and avg_{other}). We then calculate a multiplier m for the weights and a bias b for the unit so that the average net input for the examples in the subclass will sum to net_{on}, and to net_{off} for those examples not in the subclass. To do this we solve for m and b in these equations:

$$m\ avg_{inclass} + b = net_{on} \quad (2)$$
$$m\ avg_{other} + b = net_{off} \quad (3)$$

We multiply the hidden unit's weights by m and set its bias to b, making the unit a detector of its subclass. We repeat this process for each subclass of each category, producing a network ready for backpropagation.

Recall that our main goal is to have a set of initial networks that are widely dispersed in weight space. The variability of our networks comes from two main sources: (i) we randomly choose the J seeds and (ii) the competitive-learning algorithm randomly sequences through the training examples. (In addition, as explained above, we randomly select hidden-to-output weights from small uniform ranges.) Due to computer-time limitations, we held the number of seeds constant in our experiments. Varying the number of seeds would increase the variability, and is a topic for future work.

5 Experiments

To judge the value of combining neural networks, we contrast four approaches:

- *single network* – train a single network. This approach is meant to represent methods that do not use multiple neural networks.
- *minimal train error* – train K networks, selecting the network that does the "best" on the training data. This is an obvious approach to using multiple networks – select the network that does best on the training data on the assumption that the same network will be the best on the test data.
- *oracle* – train K networks, selecting (using an oracle) the network that generalizes best. This approach cannot be applied in practice because there is generally no oracle that can indicate how general a solution is, but this "approach" will provide a good baseline, since it might appear on the surface that this is the best one can do.
- *combination* – train K networks and then combine their predictions.

We consider two ways of initializing the networks being trained: (a) the standard approach of randomly selecting weights and biases from a small interval centered on zero; (b) our approach that uses competitive learning.

5.1 Test domains

Our two test domains are a set of handwritten digits from Shen [1992] and the protein secondary-structure data from Qian and Sejnowski [1988]. The first data set contains 3301 8x8 digitized numerals. The problem is to recognize the digit (0-9) in the 8x8 binary image. The second data set has 21,623 subsequences of 13 amino acids (each amino acid takes 21 bits to represent, so there are 273 input units). The output for this problem is one of three categories: whether the central amino acid is part of a helix, sheet, or coil.

5.2 Methodology

We train our networks using a cost function that is the squared-error between the correct and predicted outputs. To mitigate overfitting, for the digit-recognition data we added a sum-of-weight-magnitudes complexity term to the cost function (which is equivalent to using weight decay [Hinton, 1989]), while for the protein-folding data we used validation sets [Lang *et al.*, 1990] (tests with weight decay produced poor results compared to the standard results for this data set).

However, we did not use the cost function when evaluating the trained networks. Instead, as is typical, we interpret the output unit with the highest activation as the predicted class, and measure testset error by comparing the predicted class to the correct class. (We also use this methodology to select the minimal-error classifier.)

To measure generalization we performed five 10-fold cross-validation tests on each data set. In 10-fold cross-validation, the data is randomly divided into 10 groups. We then create 10 (possibly composite) classifiers for the data – each classifier uses a different one of the 10 groups as test data, and the other nine groups as training data. (Note that we learn only from the training data; the test data is set aside during learning.) We then calculate the testset error for the 10-folds and average over the five runs. For the single-network results our classifier is, naturally, a single network. For the minimal-error classifier we train 10 networks and then choose the network that achieves the lowest error on the training set as our classifier. The oracle-based classifier uses the same 10 networks, but it "cheats" by applying the networks to the test data and then using the network that achieves the lowest error on the test data. For the combination results we average the outputs of the 10 networks to obtain a single prediction.

To initialize the networks for our competitive-learning approach, we run competitive learning for each of the different categories of outputs. For the digit data, we use competitive learning to create five subclasses for each type of digit. Thus there are 50 hidden units (five subclasses for each of 10 digits) in the networks. This number was suggested by other empirical tests on this data set [Cherkauer, personal communication]. For the protein-folding data we used 40 hidden units, which Qian and Sejnowski [1988] concluded was an effective number. We used competitive learning to produce 10 helix subclasses, 8 sheet subclasses and 22 coil subclasses (these numbers reflect the approximate distribution of the data among the three categories of outputs).

Table 2: Average testset error rates for the digit-recognition data.

	Initialization	
Method	Standard	Competitive
Single Network	11.7%	15.6
Minimal Error	11.4	12.7
Oracle	9.4	9.8
Combination	9.7	**8.8**

Table 3: Average testset error rates for the protein secondary-structure data.

	Initialization	
Method	Standard	Competitive
Single Network	37.8%	35.9
Minimal Error	38.0	36.0
Oracle	34.7	34.3
Combination	34.9	**33.8**

5.3 Results

Our results appear in Tables 2 and 3. For each of our tests we report four pairs of numbers: one for the standard backpropagation method of initialization (Standard) and one for our method of initialization (Competitive). For each data set, we always used networks with the same number of hidden units. The testset error rates reported correspond to the four approaches listed above: using a single network;[4] choosing, from the 10 networks, the network that is most accurate on the training data; choosing the network among the 10 that is most accurate on the testset; and using (for each example) the mean prediction of the 10 networks.

For both data sets the best result was achieved by our combination of networks initialized using competitive learning, even though the individual networks did not necessarily perform well. The reduction in error for our combination of networks using competitive learning (Combination/Competitive) compared to the standard approach of training a single standard neural network (Single-Network/Standard) is statistically significant at the $p < 0.05$ level (i.e., with 95% confidence) for both sets of data. Apparently the networks initialized with competitive learning are more independent in the errors, since in combination they lead to decreases in testset error. An interesting point to note is that the combination results for our approach are better than the oracle results – even though the oracle may seem to be ideal.

As a baseline for comparison, note that the best reported error rate for the digit-recognition task is 14.9%, using decision lists [Shen, 1992]. For this set of protein-folding data, the best reported error rates are 37.3% using standard neural networks [Qian and Sejnowski, 1988], 36.6% using a knowledge-based neural network [Maclin and Shavlik, 1993], and 30.7% using a case-based reasoning algorithm (and a somewhat larger data format) [Leng *et al.*, 1994]. Our results are better than all but the case-based reasoning results (which also used a different input encoding).

In Figure 5 we report error as a function of the number of networks combined. As one might expect, the error shows a sharp drop when combining the first few networks and then a gradual decrease. One general conclusion is that combining the predictions of several neural networks is a wise decision; doing so is algorithmically simple and can lead to sizable reductions in testset error compared to the traditional approach of using one trained network. This reduction holds even if one does not employ our approach to initializing networks.

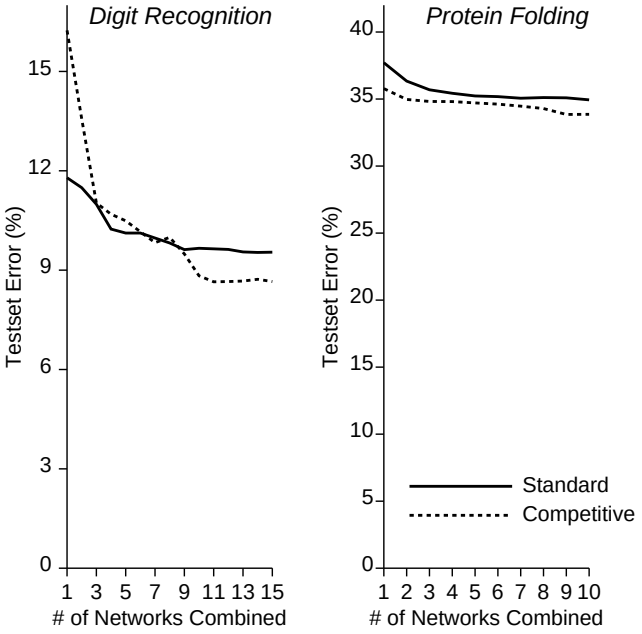

Figure 5: Error as a function of the number of networks combined using the standard initialization method and initialization using competitive learning.

6 Future and Related Work

We plan to study a number of other methods for creating independent networks.[5] Other methods we would like to try include varying the network architecture (the number of hidden units), using different training sets, and varying other training parameters (such as the learning rate). We also intend to study the question of how to determine the optimal number of networks to use in a combination. Finally, we plan to design a scheme for filtering out networks – some means to recognize networks that are likely to hurt performance.

A number of areas of research are related to the work we presented in this paper. Combining predictions has a long history in a number of fields (for an overview in the area of forecasting see Granger [1989]). In the field of neural networks, a number of researchers have looked at the advantages of combining multiple predictions. Lincoln and Skrzypek [1990] and Hansen and Salamon [1990] both explore the advantages of combining groups of networks in a simple way. Many researchers [Ghosh *et al.*, 1992; Hashem *et al.*, 1993; Perrone and Cooper, 1994; Rogova, 1994; Wolpert, 1992] have studied the problem of combining predictions in a more robust way, taking

[4] We report the average error rate of the 10 networks.

[5] The main difficulty in pursuing these other experiments is CPU time; our protein folding experiments required us to train 500 networks (5 random repeats of 10 networks for each of 10 cross-validation folds), and each of these networks takes a little more than 1 CPU day of training.

into account the confidence of the prediction. Another solution to this problem is to use a learning system, such as neural networks, to learn how to combine the results of multiple predictions. Zhang et al. [1992] and Rost and Sander [1993] use this method. Jacobs et al [1991] have taken the approach of combining multiple networks one step further, in that they attempt to evolve a set of subnetworks such that each subnetwork is good at prediction for a different region of input space.

A related approach to our method of choosing various starting points in weight space is to vary the training process to produce networks that are independent and therefore useful for combination. Examples of this approach include Reilly et al.'s multi-resolution architectures [1987], Schapire's [1990] and Drucker et al.'s [1994] boosting algorithms, Hampshire and Waibel's [1990] use of different objective functions, Baxt's [1992] method of training networks on different tasks, and Perrone's [1992] tree-structured neural networks.

Our method of combining competitive (unsupervised) and backpropagation (supervised) learning is similar to a number of other approaches [Hecht-Nielsen, 1988; Huang and Lippmann, 1988; Moody and Darken, 1989]. The main difference between our work and this previous research is that we use a hybrid unsupervised/supervised network as a method for producing networks that are very effective when used in combination, while the others focused on producing a better single network.

Although we were not aware of the close relationship when we developed our algorithm, our approach is similar to Moody and Darken's [1989]. However, our goals were different from theirs: we wanted to produce networks with large initial weights, while their focus was on partitioning the input space. Our method differs from Moody and Darken's in that we employ standard sigmoidal units rather than Gaussian units, and after our initialization phase we use supervised (backpropagation) learning throughout the network, rather than just between the hidden and output layer. Our work is also closely related to Nguyen and Widrow's [1990] which initializes a set of hidden units to each be responsible for a different region of input space; the main difference is that in our work we use competitive learning and our examples to select these regions rather than trying to select them randomly.

Our work also relates to hybrid systems that mix levels of unsupervised and supervised learning in neural networks [Hecht-Nielsen, 1988; Huang and Lippmann, 1988]. One difference in our work is that we perform our unsupervised learning among the categories separately. We also transform the results from competitive learning using our weight multiplier – producing large initial weights. Finally, we install the results of competitive learning into a standard network, so that we can use backpropagation to adjust the resulting subclasses.

7 Conclusions

A fundamental goal for inductive learners is to generalize well – that is, produce classifiers that accurately predict future examples. We present a straightforward approach for improving generalization in which we combine the predictions of several separately trained neural networks. To be productively combined, each individual network should represent solutions whose errors are largely independent of those of the other networks. Our key idea is to use competitive learning to initialize our networks. Using competitive learning allows us to create initial networks that lie far from the origin in weight space, yet do not suffer from the "flat-spot" problem that can greatly slow backpropagation training; thus, we are capable of reaching a wider variety of the local minima in this space than are reachable by the standard network initialization method.

The first step in our network-initialization algorithm is to use competitive learning to cluster the input patterns for each category into subclasses of that category. In our second step, we create a neural network where each subclass is recognized by one hidden unit. Finally, we perform backpropagation to refine the resulting network. We repeat this process a number of times to create several networks that in combination generalize well, according to our experiments on two real-world testbeds.

While we are not the first to make the point that the "train-multiple-classifiers" approach is useful, our experiments do provide additional evidence of the merit of this simple technique for improving generalization. In addition, we present a novel algorithm that improves one important aspect of this task, namely the initialization of the set of networks to be trained.

References

[Baxt, 1992] W. Baxt. Improving the accuracy of an artificial neural network using multiple differently trained networks. *Neural Comp.*, 4:772–780, 1992.

[Berenji and Khedkar, 1992] H. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3:724–740, 1992.

[Drucker *et al.*, 1994] H. Drucker, C. Cortes, L. Jackel, Y. Le Cun, and V. Vapnik. Boosting and other machine learning algorithms. In *Procs. of the 11th Int. Machine Learning Conf.*, pages 53–61, New Brunswick, NJ, 1994.

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

[Fahlman, 1988] S. Fahlman. Faster learning variations on back-propagation: An empirical study. In *Procs. of the 1988 Connectionist Models Summer School*, pages 38–51, Pittsburgh, PA, 1988.

[Ghosh *et al.*, 1992] J. Ghosh, L. Deuser, and S. Beck. Evidence combination techniques for robust classification of short-duration oceanic signals. In *SPIE Conf. on Adaptive and Learning Systems*, volume 1706, pages 266–276, Orlando, FL, 1992.

[Granger, 1989] C. Granger. Combining forecasts – twenty years later. *J. of Forecasting*, 8, 1989.

[Hampshire and Waibel, 1990] J. Hampshire and A. Waibel. A novel objective function for improved phoneme recognition using time-delay neural networks. *IEEE Trans. on Neural Networks*, 1:216–228, 1990.

[Hansen and Salamon, 1990] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.

[Hashem *et al.*, 1993] S. Hashem, Y. Yih, and B. Schmeiser. An efficient model for product allocation using optimal combinations of neural networks. In C. Dagli, L. Burke, B. Femandez, and J. Ghosh, editors, *Intelligent Engineering Systems through Artificial Neural Networks*, volume 3. ASME Press, 1993.

[Hecht-Nielsen, 1988] R. Hecht-Nielsen. Applications of counterpropagation networks. *Neural Networks*, 1:131–139, 1988.

[Hinton, 1989] G. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[Huang and Lippmann, 1988] W. Huang and R. Lippmann. Neural net and traditional classifiers. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1, pages 387–396. Morgan Kaufmann, 1988.

[Jacobs *et al.*, 1991] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[Kearns and Seung, 1995] M. Kearns and H. Seung. Learning from a population of hypotheses. *Machine Learning*, 18:255–276, 1995.

[Lang *et al.*, 1990] K. Lang, A. Waibel, and G. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3:23–43, 1990.

[Le Cun *et al.*, 1989] Y. Le Cun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 598–605. Morgan Kaufmann, 1989.

[Leng *et al.*, 1994] B. Leng, B. Buchanan, and H. Nicholas. Protein secondary structure prediction using two-level case-based reasoning. In *J. of Computational Biology*, pages 25–38, 1994.

[Lincoln and Skrzypek, 1990] W. Lincoln and J. Skrzypek. Synergy of clustering multiple back propagation networks. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 650–659. Morgan Kaufmann, 1990.

[Maclin and Shavlik, 1993] R. Maclin and J. Shavlik. Using knowledge-based neural networks to improve algorithms: Refining the Chou-Fasman algorithm for protein folding. *Machine Learning*, 11:195–215, 1993.

[Mani, 1991] G. Mani. Lowering variance of decisions by using artificial neural network portfolios. *Neural Computation*, 3:484–486, 1991.

[Moody and Darken, 1988] J. Moody and C. Darken. Learning with localized receptive fields. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 133–143, Pittsburgh, PA, 1988.

[Moody and Darken, 1989] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[Nguyen and Widrow, 1990] D. Nguyen and B. Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In *Procs. IJCNN-90*, volume 3, pages 21–26, San Diego, CA, 1990.

[Opitz and Shavlik, 1993] D. Opitz and J. Shavlik. Heuristically expanding knowledge-based neural networks. In *Procs. IJCAI-93*, pages 1360–1365, Chambery, France, 1993.

[Perrone and Cooper, 1994] M. Perrone and L. Cooper. When networks disagree: Ensemble method for neural networks. In R. Mammone, editor, *Artificial Neural Networks for Speech and Vision*. Chapman and Hall, 1994.

[Perrone, 1992] M. Perrone. A soft-competitive splitting rule for adaptive tree-structured neural networks. In *Procs. IJCNN-92*, pages 689–693, Baltimore, MD, 1992.

[Qian and Sejnowski, 1988] N. Qian and T. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *J. of Molecular Biology*, 202:865–884, 1988.

[Quinlan, 1987] J. R. Quinlan. Simplifying decision trees. *Int. Journal of Man-Machine Studies*, 27:221–234, 1987.

[Reilly *et al.*, 1987] R. Reilly, C. Scofield, C. Elbaum, and L. Cooper. Learning system architectures composed of multiple learning modules. In *Procs. ICNN-87*, pages 495–503, San Diego, CA, 1987.

[Rogova, 1994] G. Rogova. Combining the results of several neural-network classifiers. *Neural Networks*, 7:777–781, 1994.

[Rost and Sander, 1993] B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *J. of Molecular Biology*, 232:584–599, 1993.

[Rumelhart and Zipser, 1985] D. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.

[Rumelhart *et al.*, 1986] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing, Volume 1*, pages 318–363. MIT Press, 1986.

[Schapire, 1990] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[Shen, 1992] W.-M. Shen. Complementary discrimination learning with decision lists. In *Procs. AAAI-92*, pages 153–158, San Jose, CA, 1992.

[Wolpert, 1992] D. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[Zhang *et al.*, 1992] X. Zhang, J. Mesirov, and D. Waltz. Hybrid system for protein secondary structure prediction. *J. of Molecular Biology*, 225:1049–1063, 1992.