Partitioned Neural Networks

Douglas P. Sutton, Martin C. Carlisle, Traci A. Sarmiento, and Leemon C. Baird III,
United States Air Force Academy
Colorado Springs, CO 80840

***Abstract* – A new method is given for speeding up learning in a deep neural network with many hidden layers, by partially partitioning the network rather than fully interconnecting the layers. Empirical results are shown both for learning a simple Boolean function on a standard backprop network, and for learning two different, complex, real-world vision tasks on a more sophisticated convolutional network. In all cases, the performance of the proposed system was better than traditional systems. The partially-partitioned network outperformed both the fully-partitioned and fully-unpartitioned networks.**

I. INTRODUCTION

Much work has been done on adaptively pruning a feedforward backprop network to improve generalization and learning speed, and algorithms to make it reorganize to become more local and change its structure [1], [2], [3], [4]. These authors show the importance of adaptively reducing the weights, obtaining the optimum network size, and decreasing susceptibility to interference in neural networks.

Most of this work has been *adaptive* (occurring during or after learning) and has used feedforward networks that are *shallow* (e.g. having only a single hidden layer). Comparatively little work has been done on improving deep networks with many hidden layers by choosing different interconnect structures *a priori* before learning begins. It appears that in humans some large-scale brain structure exists even before learning, such as the existence of two separate hemispheres and other regions that are partially partitioned from each other. Therefore it could be useful to explore the effect of partitioning on the learning speed of a simple neural network.

II. PARTITIONING

A feedforward, sigmoidal backprop neural network is typically structured with full interconnections from each layer to the next. For example, in Figure 1, network (a) has 3 inputs, 2 outputs, and 3 hidden layers with 4 neurons each. Each layer is fully interconnected to the layer above it. Such a network is a *universal approximator*, it can represent any continuous function to arbitrarily-low error given sufficient neurons in each hidden layer.

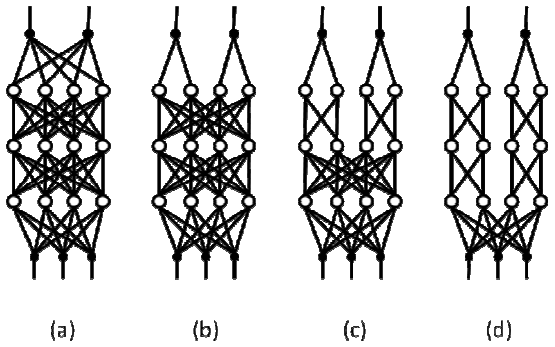

Fig. 1. An example network with 3 inputs, 2 outputs, and 3 hidden layers of 4 neurons each. Networks (a), (b), (c), and (d) have 0, 1, 2, and 3 partitioned hidden layers, respectively. Network (a) is not partitioned at all, (d) is fully partitioned, and (b) and (c) are partially partitioned.

In network (b), the top hidden layer has been *partitioned*, meaning that the neurons in its left half have outputs to only the left half of the layer above, and the neurons in its right half have outputs to only the right half of the layer above. Network (c) has its top 2 hidden layers partitioned, and (d) has all 3 of its hidden layers partitioned. We will refer to (a) as *unpartitioned,* (b) and (c) as *partially partitioned*, and (d) as *fully partitioned*.

Note that there is no (e) with the input layer partitioned. Such a network would be incapable of learning any function where the leftmost output is a function of the rightmost input. In general, partitioning can only be performed on hidden layers.

Most existing networks in common use resemble either (a) or (d), but nothing in between. For example, if the goal is to have a neural network recognize images of handwritten digits, there are two obvious approaches. One common approach is to build a network with 10 outputs. When shown an image, each output gives the degree to which the network thinks the image corresponds to that digit. Another common approach is to build 10 separate networks with 1 output each, where each one is trained to recognize just one digit. Then all of them are given the image as input, and each one outputs the degree to which that image corresponds to the digit it is looking for. These two approaches correspond to networks (a) and (d), respectively. The

intermediate networks (b) and (c) are the new, partially-partitioned networks proposed here.

For learning a difficult, real-world function, we might expect network (a) to learn faster than (d). This would be true if each output depended on certain complicated features that are calculated from the inputs, and if the two outputs needed some features in common. It has layers with many neurons to calculate the features, and then at the end it can derive the two outputs from those features. But the left half of network (d) would have to derive those same features using only half as many neurons, and so would the right half.

Conversely, if the two outputs are completely unrelated, such that there are no conceivable features that could benefit both, then network (d) might be expected to learn faster than network (a). This would be true because the left half could work on calculating the features it finds useful, with no interfering inputs coming from the right half. Similarly, the right half could perform its calculations without interference from the left. The two partitions would be completely independent, and so would not confuse each other during learning.

Note that this partitioning might help learning speed, and it might even improve which local minimum backprop will find, but it cannot improve the approximation error for optimal weights. If an oracle were to tell us the optimal weights for each network, then (a) must be at least as good as all the others. That is because no matter how well (d) performs, (a) can achieve the same performance by setting to the zero the weight for each connection that exists in (a) but not (d). Of course, no such oracle exists, so (a) may not find the best solution in practice. And that analysis ignores learning speed, so (a) may not learn the fastest, either.

We hypothesize that partially partitioned networks like (b) and (c) will tend to give faster learning than either fully partitioned or fully unpartitioned networks, for most of the difficult problems that network are typically given. This would seem to give the best of both worlds. The inputs would first feed into several fully-connected layers, allowing it to apply all its resources to learning features that are common to all the outputs. Then there would be several partitioned layers, allowing each output to learn its own, individual features and perform calculations without being confused by what is going on in the other partitions.

TABLE I
BEST PARAMETERS FOUND

Number of hidden layers partitioned	Learning rate	Momentum
0	0.002001	0.980069
1	0.001415	0.990034
2	0.002001	0.920275
3	0.002001	0.990034
4	0.002001	0.980069
5	0.002001	0.980069
6	0.002001	0.980069
7	0.002001	0.980069
8	0.001415	0.001415

These were the best parameters in multiple runs, for each amount of partitioning. These parameters were used in the experiments comparing the learning speed for each network.

III. EMPIRICAL RESULTS

To test this hypothesis, we ran a series of experiments on three difficult problems: one simple and two complex. In the simple problem, the network was required to learn a Boolean function with several inputs and 2 outputs. One of the outputs was the parity function, which is known to be difficult for neural networks, so this was a nontrivial problem. But the definition of the problem was simple to aid analysis and replication by other researchers. The complex problems were computer vision problems of recognizing either handwritten digits or fingerprints. These were large, difficult, real-world problems, designed to show how the proposed system scales up.

IV. EMPIRICAL RESULTS – SIMPLE

The initial experiment had a simple function for the network to learn. The function took a specified number of inputs, each of which was either 0 or 1, added them, and gave as its two outputs the two least significant bits of the sum. The equation was:

$$\text{Outputs} = \text{SUM(inputs)} \text{ AND } 3 \quad (1)$$

where SUM adds together all the inputs (i.e. it counts how many of the inputs are 1s rather than 0s), and AND is a bitwise AND (i.e. it takes the 2 least significant bits of the sum). Note that the least significant bit of the sum is the parity function, which is known to be a difficult function to learn for this type of neural network. The other output is different from parity but related to it, since it's related to the sum modulo 4, and sum is the count modulo 2. Thus the output consists of two different functions, each of

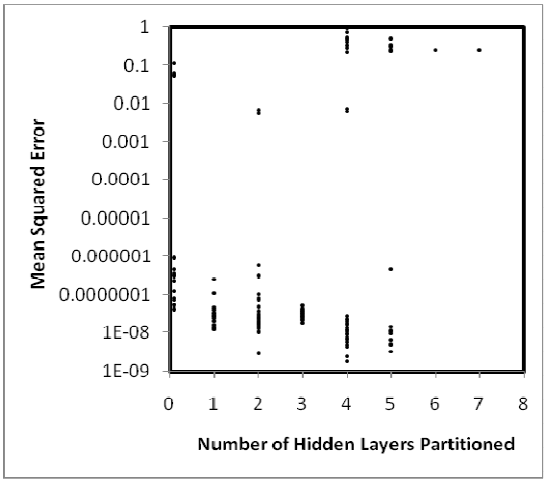

Fig. 2. Mean squared error vs. the amount of partitioning for the simple problem. The error reflects the quality of the weights learned after 50 million iterations. Each dot represents a separate run with random initial weights.

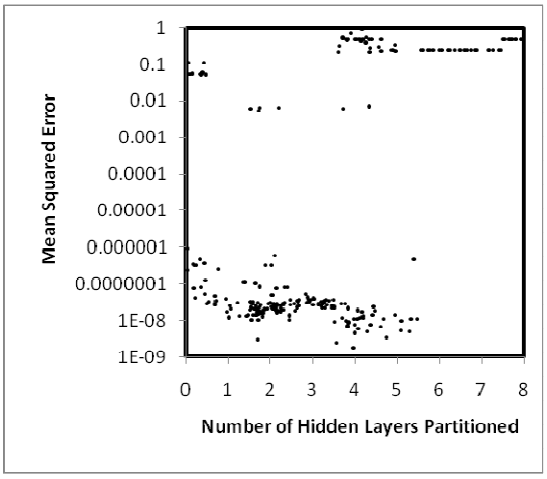

Fig. 3. The same data as Figure 2, but each dot is offset horizontally by a small random amount to make the distribution clearer. It is clear that the distribution for some partition amounts is bimodal, with each error in the upper cluster being more than 10,000 times larger than all the errors in the lower cluster.

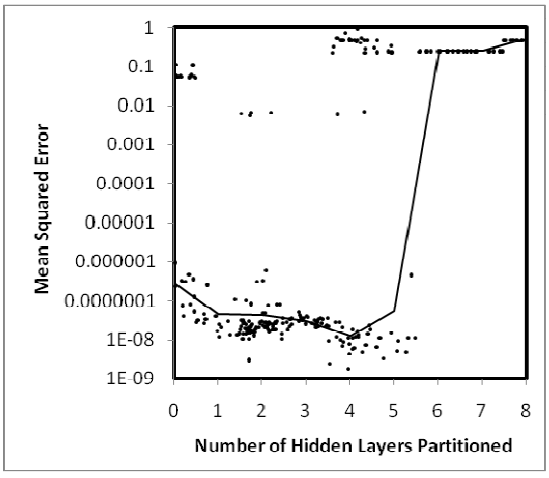

Fig. 4. A trendline superimposed over Figure 2. The line goes through the average of either all the points (for unimodal distributions) or the lower of the two clusters (for bimodal).

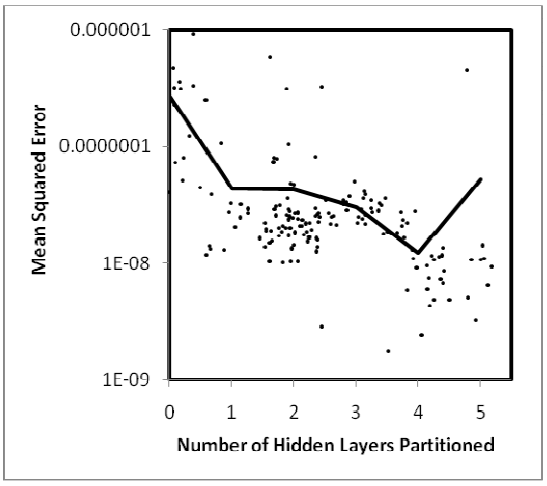

Fig. 5. A zoom into Figure 4. This image includes exactly those points that are in the lower cluster for every partition amount from 0 to 5.

which is difficult to learn, and which have some similarity to each other.

The neural network used in this experiment consisted of 8 hidden layers, with 32 neurons per layer, 7 inputs, and 2 outputs. Learning was standard backprop with momentum. Multiple runs were performed with the network partitioned by various amounts, with an average error calculated for each amount of partitioning. The error value was computed as the average total squared error in the output.

The network learning parameters were optimized independently for each amount of partitioning. Existing parameter searching code made numerous runs to find reasonable values for the learning rate and momentum for each amount of partitioning.

For each amount of partitioning, several runs were made, of fifty million iterations each, and the network was tested at the end of the run to find the mean squared error given its final weights. If the proposed system is useful, then the error should tend to be lower for partially partitioned networks (e.g. 4

hidden layers partitioned) than for both fully partitioned (all 7 layers partitioned) and fully unpartitioned (none of the layers partitioned).

Figure 2 shows the results, as mean squared error vs. the amount of partitioning. The error represents the quality of the function learned by the network after 50 million iterations of backprop, using the optimized parameters from Table 1. Each dot represents a separate run, using random initial weights.

It is clear from Figure 2 that the best results come from using a network that is half partitioned (4 of the 7 hidden layers are partitioned). It then gets worse as less and less of it is partitioned, until it is significantly worse with no partitioning. On the other hand, increasing the partitioning to 5 is slightly worse, and increasing to 6, 7, or 8 yields an enormous performance penalty, with errors 10,000 times larger.

The vertical distribution of dots in each column is interesting. It is bimodal for partition amounts of 0, 2, 4, and 5, and unimodal for 1, 3, 6, 7, and 8. This suggests that in the bimodal cases there are two main local minima, and backprop falls into one or the other with roughly equal probability. The strange mixture of some being bimodal and some not may be due to how well the learning rate and momentum were optimized for each case. The parameter optimizer that was used may have been confused by such a strongly bimodal distribution.

Figure 3 makes the distribution easier to see by moving each dot horizontally a small, random distance. It is clearly bimodal, with the upper cluster staying above about 0.01, and the lower one below 0.000001.

This is an unusual distribution. It suggests that the best way to learn this function is to run the learning procedure several times, and pick the best of the resulting networks. If this is done, then it is likely that at least one of the runs will fall into the better of the two local minima. Therefore, in those cases that are bimodal, with a cluster of high-error values and another cluster of low-error values, it is the latter which are representative of how well the best network will learn in several trials.

Figure 4 shows the same data as Figure 3, but with a trendline added. This line goes through the mean error for each amount of partitioning,. For the bimodal cases (i.e. 0, 2, 4, 5,), it uses the mean of the lower cluster (i.e. below .001), since that cluster is likely to be the result if the best of several runs is taken.

Figure 5 zooms in to show all the data from the lower clusters, and only that data. It is clear that partitioning the network in 4 of the 7 layers is better than partitioning more or fewer. And the extreme of portioning all or none (as is standard in most existing systems) is clearly worse than the partial partitioning proposed in this paper. So the proposed system appears to be very useful in this case.

Fig. 6. A few examples of the wide variety of digit images that the neural network learned to distinguish in the second experiment. This is the digit 4, but it was also trained on the digit 5.

V. EMPIRICAL RESULTS – COMPLEX

To see whether the system can scale well, it was applied to two real-world vision problems. The first was the recognition of images of handwritten digits as collected in the NIST digit databases. Specifically, the MNIST database [5] was used, which is a subset of the NIST Special Database 1 and Special Database 3.

These images include a wide variety of ways of forming the digits, such as the digit 4 having the two sides join at the top or not. They include thick lines and thin lines. They include digits with all sorts of warping and rotation. This is certainly a much larger, and more real-world problem than the one used in the previous section. Figure 6 shows just a few examples of the kind of data the network was intended to handle. Some of them would even be difficult for a human to recognize, such as the one in the lower left.

In the past, we have used convolutional neural networks [6], [7] to distinguish the 10 digits from this database with high reliability, even in the presence of substantial noise. This was therefore a useful test bed for exploring how well the partitioning approach scaled.

For this experiment, we used a network with 3 hidden layers. We used only 2 outputs, one giving

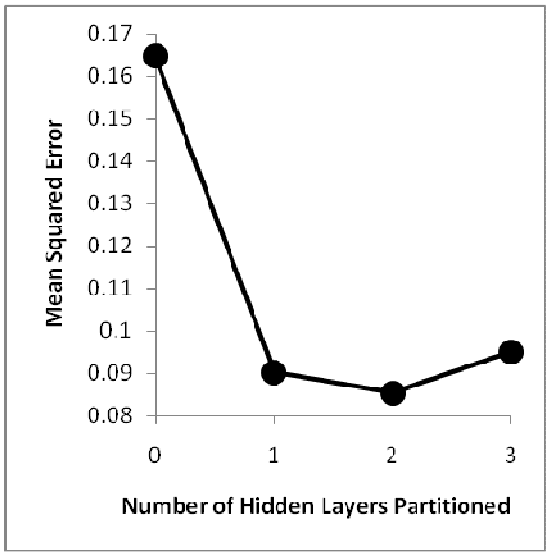

Fig. 7. Results for the digit recognition experiment. Once again, the proposed system with partial partitioning was better than the common existing approaches which partition fully or not at all.

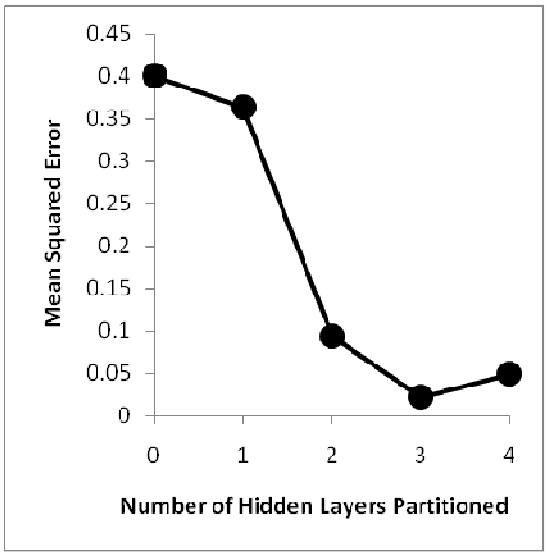

Fig. 8. Results for the fingerprint recognition experiment. Once again, the proposed system with partial partitioning was better than the common existing approaches which partition fully or not at all.

the degree to which the image was a picture of a "4", and the other giving the degree to which it was a "5". In each case, it was repeatedly trained until it seemed to have reached a local minimum. The learning rate and momentum were optimized separately for each amount of partitioning.

Figure 7 shows the results of these tests. As before, the proposed system of partial partitioning outperformed the common, existing approaches of partitioning fully or not at all.

Finally, similar experiments were conducted on a fingerprint recognition problem. There were 2 different fingers, with 100 slightly-different images of each finger in the database. The fingerprints had noise, rotation, and other flaws. The convolutional network was similar, but had 4 hidden layers rather than 3. Figure 8 shows the results for this experiment. Once again, the proposed system of partial partitioning gave better results than the traditional systems of fully partitioned or fully unpartitioned neural networks.

It is interesting to compare the results of these three problems. In all cases, the optimal partition was close to halfway (4 out of 7, then 2 out of 3, then 3 out of 4). But the two types of problems had different worst cases. For the simple generalized parity problem, the worst case was when it was completely partitioned. This was 3 or 4 orders of magnitude worse than completely unpartitioned. In the digit recognition problem and fingerprint recognition problems, those two are reversed. Completely unpartitioned was the worst, and completely partitioned was not as bad.

In all cases, the curve is a U shape that starts out high at fully unpartitioned, drops monotonically to some partial partitioning point, then rises monotonically to fully partitioned. It would be interesting to try more problems and see if this is true for most real-world problems.

VI. CONCLUSION

The proposed partitioning system appears to be a useful way to improve learning speed in deep neural networks, both for ordinary networks and for convolutional networks. Both intuition and empirical results suggest it outperforms traditional fully-partitioned and fully-unpartitioned systems.

It is reasonable to think that a partially partitioned network will outperform both a fully-partitioned and fully-unpartitioned network, because the various outputs can develop shared feature detectors in their early layers, while still privately performing their own calculations in the later layers without interference from each other.

This intuition was supported by the empirical results. The parity problem is both simple to describe and difficult for a neural network to learn. The generalization of it given here is also simple and difficult, and has two outputs, making it ideal for empirical and analytical testing of partial partitioning. The digit recognition problem and fingerprint

recognition problems are more complex. But here too, the proposed system outperformed traditional systems.

There is much room for further study. Is the optimal partitioning always halfway? Are there real-world problems where partial is worse than fully partitioned or fully unpartitioned? Is there some way to estimate the optimal partitioning? If there are many outputs, is there any advantage to make some cuts between partitions deeper than others? Can any of these results be mathematically proved, for the generalized parity function tested here? These and other questions would be interesting areas for future research.

REFERENCES

[1] Y.L. Cun, J.S. Denker, and S.A. Solla, "Optimal brain damage," Advances in Neural Information Processing Systems, Morgan and Kaufmann, 1990.

[2] A. Sankar and R.J. Mammone. "Optimal pruing of neural tree networks for improved generalization," *International Joint Conference on Neural Networks, IEEE,* pp.219-224, 1991.

[3] S.E. Weaver, L.C. Baird, and M.M. Polycarpou. "An analytical framework for local feedforward networks," *IEEE Transactions on Neural Networks*, 9:3, pp. 473-482, 1997.

[4] S.E. Weaver, L.C. Baird, and M.M. Polycarpou. " Local feedforward networks," *Adaptive Distributive Parallel Computing,* Dayton, OH, pp. 280-290, Aug 8-9.

[5] The MNIST database is available at http://yann.lecun.com/exdb/mnist/

[6] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, The Handbook of Brain Theory and Neural Networks. MIT Press, 1995.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998.