Partitioning Strategies for Modular Neural Networks

Timothy Bender, V. Scott Gordon, and Michael Daniels

***Abstract*—We observe the effects of a variety of splitting strategies for partitioning the input domain in a self-splitting modular neural network applied to the two-spiral classification problem, and assisted by a special-purpose visualization tool. The observations motivate the development of an improved strategy, consisting of a series of binary splits along the boundaries of trained areas, and a particular weight initialization strategy. The work is leading to fewer networks and better generalization for this application, when backpropagation is used.**

I. Self-Splitting Neural Networks

THE Self-Splitting Neural Network (SSNN) is a particular version of a *modular* neural network, and as such divides the input domain into partitions where a separate network is assigned to each partition. In Figure 1, the inputs are routed to one of the networks (here shown as either Domain A or Domain B), depending in which portion of the input space those input values are contained. Whenever one network is assigned to those inputs, it and it alone produces the corresponding output values. Modular networks were described by various researchers [1,2,3,4].

A variety of methods have been used for partitioning the input domain. Chen et al. used divide-and-conquer to carve up the input space heuristically based on features of the problem being solved [5]. Olsen-Darter and Gordon used a modular configuration of 120 neural networks for a vehicle control application, but the division of the input domain was determined by experimentation [6].

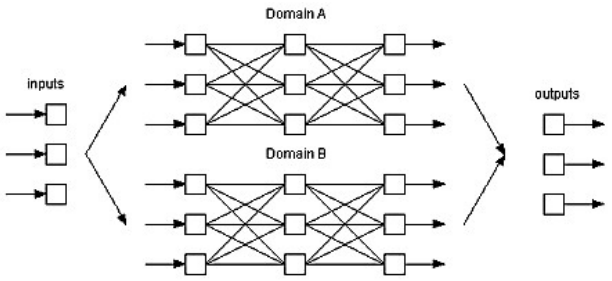

Fig. 1. Modular Network Architecture

Manuscript received December 14, 2008.

Timothy Bender and V. Scott Gordon are with the Computer Science Department at California State University, Sacramento, CA 95829 USA (e-mail: tbender85@gmail.com, gordonvs@ecs.csus.edu).

Michael Daniels is a recent graduate of California State University, Sacramento (e-mail: mike@standarderror.net).

The *self-splitting* modular approach, described by Gordon and Crouson, divides the problem by examining the results of training [7]. It starts by attempting to train a single network. If it fails to learn its entire training set, it tries to extract a trained portion. Untrained portions are assigned to new networks, as shown in Algorithm 1.

Algorithm 1 – Self splitting algorithm

1. attempt to solve the training set
2. if success, the input domain is solved ... stop.
3. if failure:
   (a) select a variable and value(s) for splitting
   (b) assign new network(s) to each sub-domain
   (c) distribute training cases by sub-domain
   (d) recursively apply to each network, until all networks learn their subset of training cases.

Gordon and Crouson tested two splitting methods: *centroid* and *trained region*. Splitting by centroid is done by splitting the domain in half on the variable with the most values. Splitting by trained region is done by sorting each input in turn, finding the largest contiguous chunk of solved training cases that spans an entire dimension, and then dividing the input domain into three partitions (Figure 2). In both methods, whenever a network solves a region, the weights used in the solution are stored along with the range of values defining the sub-domain. We use the terms "trained region" and "chunk" interchangeably.

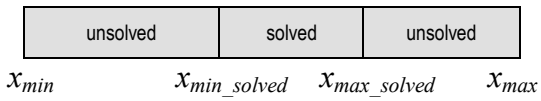

Fig. 2. Splitting by trained region for input variable x

After the entire input domain has been partitioned into solved sub-domains, the resulting set of neural networks can be tested on other (untrained) cases. Here, the trained multi-net acts as a modular network; inputs are routed to the particular network that was trained on values belonging to the same sub-domain.

The SSNN was tested on a variety of hard problems and was shown to be effective at learning large training sets while displaying good generalization. Splitting by trained region was generally the more effective. Although standard backpropagation was used, other training methods have been proposed in this context [8,9].

978-1-4244-3553-1/09/$25.00 ©2009 IEEE

II. Visualization

We observe the effects of various splitting strategies using a visualization tool developed previously and described in [10], and tuned for use with the well-known two-spiral classification problem [11]. The input domain is portrayed as a rectangular space, with one input variable extending across the X-axis, and the other along the Y-axis. When a trained subdomain is identified and splitting occurs, the region is illustrated as a smaller rectangle within the larger one. As training proceeds, the user can watch the input domain being partitioned. The tool, available at `http://ecs.csus.edu/~ssnn`, runs on any platform with Java version 1.5 or higher and Java Web Start version 1.5 or higher.

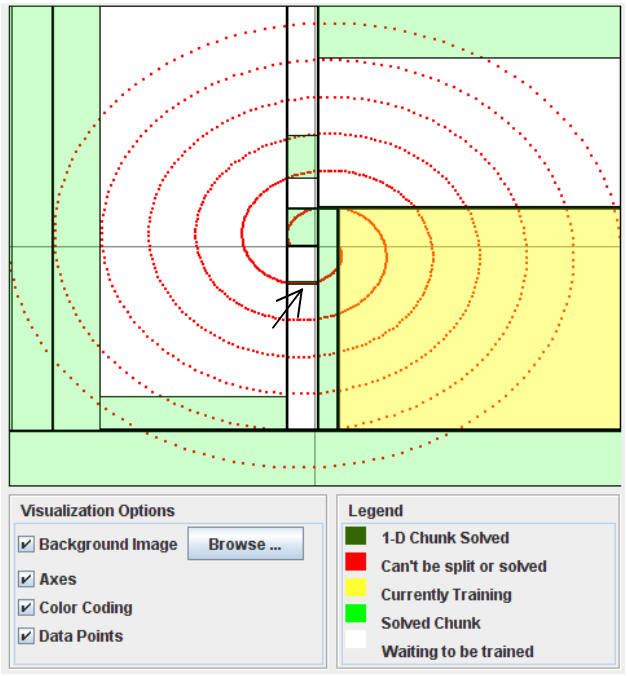

Fig. 3. Visualization Tool

An example of the visualization pane is shown in Figure 3. The red dots represent training points. Green areas have completed training, and therefore a trained neural network has been assigned to each green rectangle. The yellow rectangle is currently training. When training is complete, if it is successful, the entire pane will be covered with green rectangles.

III. Existing Splitting Strategies

A variety of splitting strategies have been examined for this and other modular neural networks.

As mentioned earlier, Chen et al. used divide-and-conquer to carve up the input space heuristically based on features of the problem being solved [5]. They build a tree structure that is used to determine which subnetwork is applicable to a given set of inputs.

By contrast, Olsen-Darter and Gordon used splitting to focus precision on problematic areas of a vehicle steering problem [6]. Those areas of the problem domain that had not completely trained were manually subdivided.

Gordon and Crouson considered two self-splitting strategies: *split by centroid* and *split by trained region*, on a variety of test problems [7], as mentioned in section 1. Their results indicated that *split by trained region* led to fewer networks and better generalization in most cases.

However, when tracking the progress of *split by trained region* using the visualization tool described earlier in section 2, we observed that many of the resulting trained regions were small. In some cases, a trained region would appear as a very narrow band, such as when only a single value in a given dimension was completely solved. An example of such a region is shown in Figure 3 – the small horizontal segment indicated by the arrow. Such regions might be better utilized by combining the small band with a neighboring region, such as the white region above it or the white region below it, and retraining. Our first goal is thus:

- ✓ *Combine small trained regions with neighboring untrained regions.*

Interestingly, setting a minimum size on an allowable trained region is not sufficient to guarantee that small trained regions would be avoided. For example, a large trained region may have been identified, but by extracting it a very small *unsolved neighboring region* is also isolated. Such cases might be better served by combining the solved region with the narrow neighboring unsolved region, and attempting to train the very slightly larger region as a single unit. A second goal is thus:

- ✓ *Combine small untrained regions with neighboring trained regions.*

A third observation we made was that, in many cases, a solved chunk is nearly, but not quite, formed. That is, sometimes a large area of points is solved but not enough to form a complete chunk across an entire dimension. It is reasonable to surmise that if a partial chunk was very close to being solved, it would be worth identifying that partially-solved chunk in the hopes that in isolation it could train completely. Thus our third goal is:

- ✓ *Exploit large groups of solved points whether or not they fit the definition of a solved chunk.*

For the aforementioned reasons, we became interested in developing more effective ways of partitioning the domain, so as to encourage the creation of larger and fewer partitions. We also are interested in confirming whether, in so doing, fewer partitions leads to better generalization, as intuition would suggest.

IV. Developing New Splitting Strategies

Motivated by the observations from section 3, above, we reconsidered the splitting strategies used in the SSNN. In particular, we sought splitting strategies that would encourage smaller regions to combine with neighboring regions before being isolated in a finished modular net.

Interestingly, *split by centroid*, although less effective than *split by trained region* at solving problems, does have the advantage of generating only two, rather than three, partitions. The notion of combining the strengths of both methods into a single new strategy suggests the following approach:

- *Use chunks to find binary split points, and*
- *Define solved regions more flexibly*

We consider each of the above strategies in turn.

A) Using Chunks to find Binary Split Points

We seek here a modified version of *split by trained region* such that, after training, the largest solved chunk is identified. However, instead of removing the largest chunk immediately as a trained module (typically resulting in a ternary split as described in section 1), a boundary is selected based on the solved chunk, and a binary split is performed at that boundary.

A solved chunk, as defined earlier, spans a definable range of values across one dimension and therefore gives rise to two possible boundaries. Consider the following simple example, taken from [9]:

case #	input A	input B	success
6	6.0	3	X
2	8.5	4	
3	12.0	1	X
7	12.0	6	X
1	15.5	2	X
4	19.0	0	X
5	19.0	1	

Fig. 4. *Split by trained region,* example.

As described in [9], a standard *split by trained region* searched all dimensions and has identified the shaded region as the largest solved chunk, isolating it as a module. There exist two boundaries, one above and one below, and those remaining two chunks (shown in white), would be placed on a queue and each assigned to their own (new) neural networks for subsequent training.

One way of converting this into a binary split would be to simply select one of the two boundaries on which to split, rather than performing the above ternary split. Say that the upper boundary was chosen. In that case, a binary split would be made as follows:

case #	input A	input B	success
6	6.0	3	X
2	8.5	4	
3	12.0	1	X
7	12.0	6	X
1	15.5	2	X
4	19.0	0	X
5	19.0	1	

Fig. 5. Chunk-based binary split, example.

That is, the two points above the solid line make up the first partition, and the five points below the line comprise the second partition. The remainder of the process could then proceed as for centroid splitting, since neither partition has been completely solved. That is, each partition is then placed on a queue for later training.

However, two differences are worth noting. First, there are two boundaries from which to choose. Second, after splitting, one of the two partitions will contain a solved chunk *and we have weight values that solve that chunk.*

Which boundary should we select? Since our goal here is ultimately to produce larger solved chunks, one idea is to try to increase the likelihood of successfully continuing to grow the already-solved partition. That would entail *retaining the weight values it had learned*, and combining it with the smaller of its two neighbors. Stated differently, we would select the boundary where the associated *untrained* region is the *largest*, thus combining the solved partition with its smallest neighbor.

In the example shown in Figure 4, both of the untrained regions are of size two, so it wouldn't matter which boundary was chosen. It is also worth noting that while both *split by centroid* and our newly-defined *chunk-based binary splitting* are binary splitting strategies, they split along different points in the domain.

B) Retaining Weight Values in Unsolved Regions

Our analysis above led us naturally to adding another element to our splitting strategy, namely, retaining trained weight values in the queue of unsolved regions. In so doing, *training proceeds rather than starts anew*, which has the appealing property that training a region containing an already solved chunk is unlikely to result in a smaller solved chunk. There is a greater likelihood that the solved chunk would be made larger. Strictly speaking, this is not absolutely guaranteed when using standard backpropagation, which works by minimizing error, not by maximizing chunk size. However, in the case of other learning algorithms such as Particle Swarm Optimization (PSO) or neighbor annealing [8], where *chunk size* is used as the fitness measure, chunk-based binary splitting is *guaranteed to never do worse than split by trained region*, because it always starts with the same weights that were used to create the solved chunk. Thus, chunks are always encouraged to grow larger before being permanently isolated within a final neural module.

C) *Defining Solved Chunks more Flexibly*

We observed earlier that, in many cases, a solved region is nearly, but not quite, formed. Chunk-based approaches described up to now define a solved region which extends across an entire range of values within a dimension. However, it is worth noting that one of the reasons this was done, was so that a solved region could be removed and designated as solved. Now that we have offered an alternative form of splitting in which the resulting partitions are not completely solved, we can consider more flexible definitions of "chunk."

Area-based splitting does this by locating the largest hypervolume of solved training points with well-defined boundaries, whether or not they extend across an entire dimension. For each dimension there are two possible boundaries on which to split. As previously described for chunk-splitting, the most promising boundary is the one which defines the largest unsolved region, therefore that is where we split. Thus in a similar manner as before, the algorithm is driven by finding the largest solved area, and then the binary split which encloses that solved area within the smallest region. The process is described step-by-step in Algorithm 2.

Algorithm 2 – Area-Based Binary Splitting

1. Initialize weights from previous attempt. Train.
2. Re-initialize to random weights. Train.

Assuming that neither (1) or (2) solve the problem:

3. Choose from (1) or (2),
   the network with the most solved points.

4. For every pair of solved points (s1,s2):
   (a) determine region R bound by (s1,s2).
   (b) count the number of failure points F in R.
   (c) calculate the area A of the region R.

5. Select from (4) the region with max(A) where F=0.

6. For each input dimension:
   (a) consider binary split at max(R) into regions X,Y where X contains R.
   (b) consider binary split at min(R) into regions X,Y where X contains R.
   (c) select from (a), (b) split site with largest area Y.

7. Select from (6) split site with largest area Y.

8. Place regions X, Y on queue with learned weights.

We chose a simple exhaustive procedure for finding large solved areas. We consider all pairs of solved training points and check to see whether every point in the intervening region across each input dimension is solved.

As in chunk-based splitting, the weights learned from the previous training attempt are stored along with each partition, so as to grow larger solutions from existing solutions. Again, neither partition is completely solved.

Additionally, partitions are required to contain at least three training points. If this cannot be achieved, or if there is no solved area of size > 0, a centroid split is performed.

We found it beneficial to also attempt training with random initialized weights. We will show later that, although in most cases it is better to start with trained weights, in some cases randomized weights produced better results. Whichever method results in the most solved points is saved and the process repeats.

V. Comparison of Splitting Strategies

We ran each splitting algorithm on the two-spiral problem to see which would solve the problem with the fewest networks, and with the best generalization. We compared two methods, the original *split by trained region* versus our newly-developed *area based binary splitting*. We did not consider *split by centroid*, because previous studies had concluded that it generally had inferior results when compared to *split by trained region.*

Standard backpropagation was used for training each network, with the following settings:

- One hidden layer containing `4` nodes
- Learning rate = `0.3`
- Momentum = `0.85`
- Criteria for success = `0.4`
- `1058` training cases
- `1056` testing cases
- Bias (threshold) units with value `1`
- Maximum # iterations = `1,000,000`

The results are averaged over 10 runs each, and are summarized in Table 1.

Splitting Strategy	Networks to solve	Iterations to solve	Generalization
split by trained region	35.8	24268876	98.3 %
area-based binary splitting	23.5	53414664	98.5 %

Table 1. Comparison of splitting strategies on the 2-Spiral Problem

Clearly, the new *area based* splitting algorithm produces fewer networks on average for this problem than does the previously-defined *split by trained region*.

It is, however, also true that area based splitting took roughly twice as long to solve the training set, this is primarily due to the fact that we attempt training twice per

region. This step may ultimately prove unnecessary. We tracked the number of times each initialization was chosen, with randomized weights being chosen only about 25% of the time. But it is possible that this is because it was always chosen second, and that in many cases using pre-defined weights led to a completely trained region simply because it was attempted first. We can explore this further, but in this study we were more concerned with producing fewer networks and improving generalization, than we were in decreasing training time.

VI. Generalization and Visualization

Performance on the testing dataset is inconclusive with respect to generalization. Both splitting strategies correctly solve over 98% of the testing cases. Since the original splitting strategy generalized so well, there is little margin in which to discern a numerical difference.

However, the visualization tool contains an additional feature (not yet incorporated into the online version) which affords a stronger visualization of the functions implemented by the networks in each solved chunk. It does this by illustrating each rectangular region as an image where the output range 0..1 is modeled in greyscale. The greyscale value is computed for each pixel by obtaining the output of the neural network at each of the corresponding input values. The resulting pane shows the entire input domain as modeled by the modular network, rather than just at the points in the training data.

The greyscale visualization works in real time, so it is possible to watch backpropagation model a region over time as the greyscale figure bends its shape to match the superimposed portion of the training data.

Visualization leads us to believe that our effort at reducing the number of networks is well-directed.

Although we cannot convey the visual impact of real-time visualization in a static printed article such as this, we do offer the final greyscale models of the worst and best training runs (that is, resulting in the most and least number of networks) for consideration, shown below in Figures 6 and 7. Figure 6 is for a successful modular network that resulted in 41 networks. Figure 7 shows a successful modular network that resulted in 17 networks.

Neither model is perfect, but the one on the right, with fewer networks, displays what appears to be a closer approximation to the original spiral function. Some of the rectangular regions contain nicely curve-fit models, strikingly so along the bottom and righthand sides. While both images contain numerous jagged areas, the one at the left is more chaotic in this regard.

VII. Future Work

Several logical next steps are evident.

First, we need to expand our application test suite to include additional difficult problems, particularly ones with higher input dimensionality. We plan to start with the remainder of the test suite from [7], and the vehicular steering problem from [6]. Some changes to the visualization tool will be necessary to accommodate the added dimensions, such as selecting two for projection.

We also need to incorporate other training methods which utilize fitness, such as PSO or annealing. Preliminary results using neighbor annealing produced solutions with as few as 14 networks, *even with the older splitting strategies* [10]. We are hopeful that even fewer networks can be achieved using area-based splitting.

Eventually, more sophisticated clustering methods should be examined. Clustering would enable us to identify areas with a high percentage of solved points, rather than confining us to completely-solved regions. This might provide better boundaries on which to split.

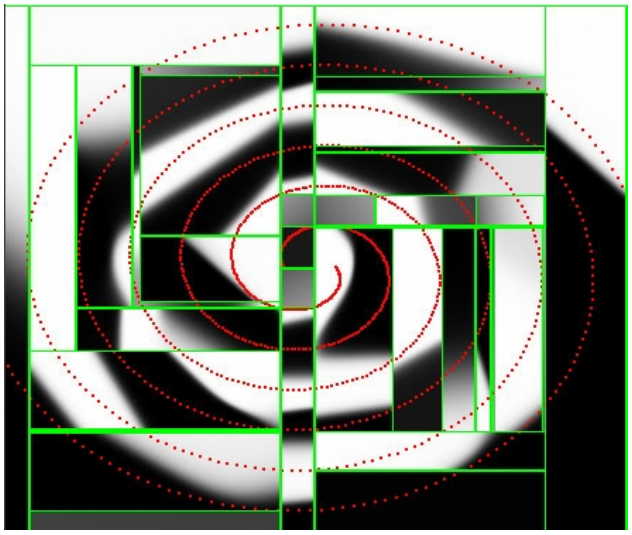

Fig. 6. 41-Network Solution

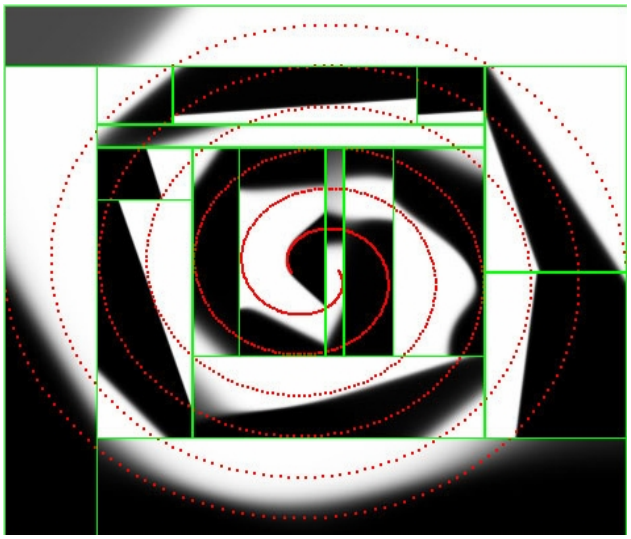

Fig. 7. 17-Network Solution

VIII. Conclusion

We described splitting strategies that have been used in modular neural networks, focusing on those used in the self-splitting modular network. We also used a visualization tool to observe the behavior of the best of these strategies, a ternary splitting method called *split by trained region*, in an effort to highlight ways that an improved splitting strategy might lead to the production of fewer networks and better generalization.

Observations using the tool led to the identification of several specific goals for improved splitting. The goals identified were: (1) use solved chunks to find binary split points, (2) retain weight values in unsolved regions, and (3) define solved chunks more flexibly.

We were able to achieve goals (1) and (2) by performing a binary split along the boundaries of trained regions, rather than the prior approach of isolating a completely solved area immediately. By combining this with initializing weights to previously trained values, we were able to define a splitting algorithm that encouraged solved chunks to grow larger before being isolated. We were able to achieve goal (3) by redefining solved chunks as based on hypervolume, and removing the requirement that the chunk be solved across a complete dimension.

The result was a new splitting algorithm we dubbed *area-based binary splitting*. We compared this new strategy with the earlier *split by trained region* approach by comparing the average number of networks produced, and how well the resulting modular networks generalized. We found that area-based splitting produced fewer networks on average, and generalized at least as well.

In order to corroborate whether fewer networks results in better generalization, we used the visualization tool to compare the models resulting from a solution with a high number of networks versus a solution with a low number of networks. Although this analysis is qualitative, the solution with the smaller number of networks appeared to model the application more accurately.

Acknowledgment

We thank James Boheman II, Marcus Watstein, Derek Goering, and Brandon Urban for helping to build the SSNN visualization tool upon which this research relied.

References

[1] R. A. Jacobs and M. I. Jordan, *A Modular Connectionist Architecture for Learning Piecewise Control Strategies*. Proc. of the American Control Conference, pp 343-351, 1991.

[2] S. D. Whitehead, J. Karlsson, and J. Tenenberg., *Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging.* Robot Learning, MIT Press, 1992.

[3] A. Sharkey, *On Combining Neural Networks*, Connection Science, 8(3/4): 299-314, 1996.

[4] *Combining Artificial Neural Networks – Ensemble and Modular Multi-Net Systems.* ed. A. Sharkey, Springer-Verlag, 1999.

[5] K. Chen, L. Yang, X. Yu, and H. Chi, *A Self-Generating Modular Neural Network Architecture for Supervised Learning*, Neurocomputing 16: 33-48, 1997.

[6] M. Olsen-Darter and V. Gordon, *Vehicle Steering Control Using Modular Neural Networks*, IEEE International Conference on Information Reuse and Integration (IRI), pp 374-379, 2005.

[7] V. Gordon and J. Crouson, *Self-Splitting Modular Neural Network – Domain Partitioning at Boundaries of Trained Regions*, 2008 International Joint Conference on Neural Networks (IJCNN 2008).

[8] V. Gordon, *Neighbor Annealing for Neural Network Training*, 2008 International Joint Conference on Neural Networks (IJCNN), Hong Kong, 2008.

[9] V. Gudise and G. Venayagamoorthy, *Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks*, Proceedings of the 2003 IEEE Swarm Intelligence Symposium (SIS'03) pp 110-117.

[10] V. Gordon, M. Daniels, J. Boheman II, M. Watstein, D. Goering, and B. Urban, *Visualization Tool for a Self-Splitting Modular Neural Network*, submitted to the 2009 International Joint Conference on Neural Networks (IJCNN 2009).

[11] M. White, *Two-Spirals Benchmark Data Set Generator*, 26 April 2006: http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/neural/bench/cmu/bench.tgz