Pair Attribute Learning: Network Construction Using Pair Features

Eric K. Henderson
Cambridge University
122 Tenison Rd
Cambridge, CB1 2DW
UK

Tony R. Martinez
Brigham Young University
Computer Science Department
3361 TMCB
Provo, UT, 84602

Abstract – We present the Pair Attribute Learning (PAL) algorithm for the selection of relevant inputs and network topology. Correlations on training instance pairs are used to drive network construction of a single-hidden layer MLP. Results on nine learning problems demonstrate 70% less complexity, on average, without a significant loss of accuracy.

I. INTRODUCTION

Two important parameters must be selected when designing a neural network to solve a given classification problem. First, the number, type, and range of the inputs must be chosen. Since this is established when the training data is collected and often includes many irrelevant inputs, a subset of these inputs should be selected that optimizes performance. This is known as input (or feature) selection. Second, the network topology must be created. That is, the number and organization (e.g. interconnections) of the nodes comprising the neural network must be specified. Since both of these parameters significantly affect the network's performance, it is essential to have some means to select them appropriately.

This paper presents an algorithm called Pair Attribute Learning (PAL), which addresses both input selection and network topology specification. The PAL algorithm selects features from a training set of instances that are then used to determine the topology of a neural network for solving a classification problem. The algorithm uses a novel search strategy based on features appearing in instance pairs. Features are chosen using a rank of statistical accuracy over the training set. The selected features drive the number of nodes in a single hidden layer network, and also dictate the connections on the input layer. The resulting network can then be trained using standard techniques, such as backpropagation.

The PAL algorithm preprocesses the training data and constructs the network directly from the result – it does not require iterative constructive methods. In addition, the resulting networks are significantly less complex than those built using other techniques, while maintaining similar predictive accuracy. Experimental results on nine separate learning problems demonstrate that PAL constructed networks are 70% less complex on average than the best performing standard networks, while maintaining accuracy within 1.2%. When compared to a common heuristic network, the PAL constructed networks show a 38.8% average reduction in complexity, with a corresponding 3.1% *increase* in predictive accuracy. In addition, because the PAL algorithm addresses input selection and network topology simultaneously, it is a comprehensive solution for the application of a neural network on a particular problem.

The remainder of this paper is organized as follows. Section II presents in detail the feature selection problem and summarizes the approaches that exist in the literature. Section III explores the issue of neural network topology and surveys some of the types of solutions that have been previously applied to this problem. Section IV gives the details of the proposed algorithm. Several experiments are introduced in Section V along with the methods used to obtain the results. The results are analyzed in Section VI, and Section VII concludes with a summary and an outline of planned future work.

II. FEATURE SELECTION

For a given classification problem, if the instance distribution is not random it will contain groups or patterns of instances having the same output class. These groups can be described as a function of some set of the inputs. These will be referred to as *features* of the input space. A feature is an area of the input space where certain inputs take a certain range or value, much like geographic features on a two dimensional map can be specified using coordinate values. Instances whose input values lie within the range of the feature are *members* of the feature.

For a given non-random distribution of instances, there exists many sets of features that can reproduce it to some desired level of accuracy. The learning algorithm must discover a set of features that promises the best performance on future novel instances. Biasing the search towards more general features increases the likelihood of future accuracy because these features are the most inclusive.

Even for a small number of inputs the search for a good set of features to describe the instance distribution is extremely complex. One means of reducing the complexity of this search is to reduce the size of the input space by eliminating an input altogether. An input can usually be eliminated if it is not strongly relevant to the features used to model the distribution. Each input removed significantly reduces the complexity of the input space. Selecting a minimal, relevant *subset* of the inputs can therefore reduce

the scope of the feature search required by the learning algorithm, potentially improving training speed and accuracy.

The selection of a subset of relevant inputs is often referred to in the literature as the "feature subset selection problem" [10]. Some references use the terms *feature*, *input*, and *attribute* interchangeably. It should be emphasized that this paper distinguishes between a *feature* (as described above), and an *input*. (The term *input* and the term *attribute* are used synonymously throughout this paper). In this sense, the search for features is simply adapting the network to solve the classification problem (by selecting features to model the instance distribution). In contrast, the selection of a relevant subset of inputs is primarily concerned with removing irrelevant inputs from the representation of the problem and reducing the input space complexity.

There have been many approaches to the selection of relevant inputs proposed in the literature. These can be functionally classified as *filter* or *wrapper* algorithms [10]. A filter attempts to reduce the number of inputs independent of the learning algorithm. The filter is run in a pre-processing stage and uses some measure of relevance to determine the subset of inputs to pass to the learning algorithm [2], [4], [12]. A wrapper is used in conjunction with the learning algorithm. In this case the wrapper determines a candidate subset of inputs and then measures the relevance by running the actual learning algorithm on them [1], [6], [10].

III. Neural Network Construction

There are many heuristics based on empirical research that can help construct a neural network with satisfactory results. Usually the hidden layer is set in relation to the number of inputs and outputs of the network. If there is domain knowledge about the problem, such as the expected number and shape of features, the hidden layer can be set accordingly. The network topology can then be adjusted based on experimental results. Unfortunately, training a network on a large problem can be prohibitively expensive and using trial and error to select the topology may not be feasible.

To address this problem researchers have focused on two types of solutions. The first approach is to construct the network, usually using some iterative algorithm that starts with a small network and gradually increases the size until some desired accuracy is achieved. The second approach is to start with a very large network trained to the desired level of accuracy, and to reduce the size until some error threshold is exceeded. This is called *pruning* the network.

A. Constructive Algorithms

Many constructive techniques produce network configurations unlike the standard feedforward single hidden layer MLP [3] [7] [24]. Other constructive techniques are specific to a class of learning problems, and use alternative activations or adaptive rules [11] [16]. Constructive algorithms that result in standard network configurations often use variations on backpropagation training, or novel iterative techniques [17] [15] [20] [25] [27] [30].

The majority of constructive algorithms applicable to supervised learning of classification problems require iterative techniques. Like heuristic or trial and error approaches, for large problems, training iterative networks becomes prohibitive. In addition, many of these constructive methods produce alternative topologies that preclude the use of widely available tools. Although a few non-iterative constructive techniques exist, they typically have constraints on the type of problems they can be applied to. The PAL algorithm presented in this paper is generally applicable to all classification problems and does not rely on iterative methods to determine the network topology.

B. Pruning Algorithms

Pruning techniques can be used on individual weights (i.e. connections) or individual nodes. Some pruning methods are interactive [26], others operate after the training phase [8] [14] [18], and some algorithms incorporate the pruning into the adaptive rule itself [5] [9] [13] [21] [28] [29].

Pruning algorithms are generally successful at reducing the complexity of some networks. However, the size of the network to be pruned must first be determined, and must be large enough to easily adapt to the problem. This introduces the computational expense of first training a large network. There is also the issue of when to stop pruning (i.e. when a sufficient reduction in complexity has been achieved). In addition, pruning may only succeed in removing redundant elements, and not affect the internal representation adapted by the network, which may be hindering generalization [23].

IV. Pair Attribute Learning

This section presents a novel algorithm called Pair Attribute Learning (PAL) that addresses both feature selection and network topology. This method uses a filtering stage to select relevant features based on a statistical measure. The resulting features are used directly to construct a neural network. The network is trained using standard backpropagation and no further processing is required.

The PAL algorithm does not explicitly search for irrelevant inputs to the problem. Instead it evaluates individual features and determines which inputs are relevant to that feature. Useful features are selected based on a performance measure. These features are then used to construct a single hidden layer MLP such that each feature produces a corresponding hidden node with connections only to the inputs that were determined to be relevant to that feature. The algorithm is biased toward low order features resulting in the construction of hidden nodes with low order discriminants. This reduces the complexity of placing the discriminants while allowing the neural network to find the best fit for a given feature.

The PAL algorithm uses correlations on pairs of instances in the training set to generate the features to be

explored. This constrains the search to only those features that appear in the training set. For this to be effective, the distribution of the training data must model the actual distribution of the learning problem (a constraint shared by most learning algorithms).

The algorithm generates a feature by finding the correlation on inputs between a pair of instances that share the same class. Correlated inputs are simply inputs that have the same value (continuous values are handled using discretization). All correlated inputs are relevant in the context of the feature (since in this case the feature is defined as the correlated inputs), whereas uncorrelated inputs are not relevant. The algorithm attempts to explore all features that exist in the training data by iterating through successive pairs. Each feature is evaluated using a statistical measure based on the accuracy of the feature when used to predict the class of the training data. This is done by finding the percent of instances that correlate with the feature, within the feature's class. A penalty term is derived for instances that correlate with the feature but have a different class. Each feature is ranked based on the result and the top scoring features are selected for use in the construction of the network. The algorithm is biased toward more general features by selecting them over more specific features when both show a similar performance. This increases the likelihood that the network can generalize adequately, and is less susceptible to noise.

Once a set of features is selected, a corresponding network is constructed. A node is placed in the hidden layer for each feature in the set. Each relevant input used in the feature is connected to the node with all other inputs left unconnected. The output layer is then fully connected. This produces a network with the input layer sparsely connected to the hidden layer, assuming the pre-processing produces low order features. The network can then be trained using standard techniques such as backpropagation.

FIGURE I
A FULLY CONNECTED NETWORK (a)
A PAL CONSTRUCTED NETWORK (b)

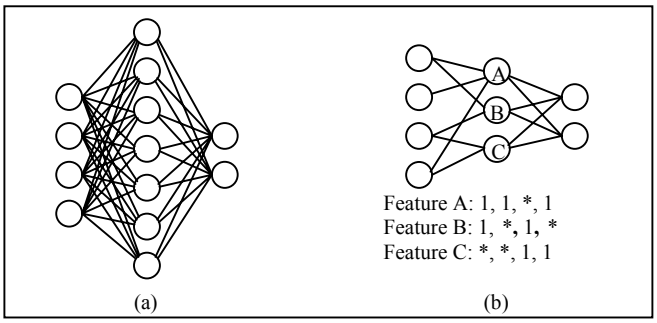


Figure I shows an example of a network constructed with the PAL algorithm. In this example, the classification problem has four inputs and two outputs. The first network (a) is a standard fully connected single hidden layer MLP with seven nodes in the hidden layer. A list of three features is shown under the second network (b). The features are given as an ordered list of 1's and *'s corresponding to the four inputs. A 1 is shown for a relevant input, and * is shown for an irrelevant input. These features were used to construct the network (b) shown in the figure.

FIGURE II
PAL PSEUDOCODE

```
begin Pair Attribute Learning

for all classes in the training data
  for all instance pairs in the class
    if instance pair has a correlated feature
      rank feature
      add the feature and rank to a list for later processing

for all classes in the training data do
  while there are more features in the list for this class and
  more features are needed
  select the feature with the highest rank for network
  construction and remove it from the list

end Pair Attribute Learning
```

Pseudocode for the feature selection phase of the Pair Attribute Learning algorithm is given in Figure II. The pseudocode consists of two main loops. The first loop iterates through all same-class pairs of instances to find features. These are then ranked and saved in a list. The second main loop selects a subset of the collected features based on rank, to be used in the construction of the network.

The rank of a feature is calculated using equation (1) shown below, where f is the feature to be ranked, c_f is the set of instances belonging to the class c of feature f, x is a training instance, m(f, x) is 1 if the feature f matches instance x and zero otherwise, n_c is the number of training instances in class c, and n is the total number of training instances. Negative ranking features are ignored.

$$r_f = \frac{1}{n_c} \sum_{x \in c_f} \mathrm{m}(f, x) \; - \frac{1}{n - n_c} \sum_{x \notin c_f} \mathrm{m}(f, x) \qquad (1)$$

The rank provides a rough measure of the coverage and accuracy of a feature. After all features are discovered and ranked for a given class, they are extracted in high-low order until the cumulative rank of all extracted features reaches an empirically determined threshold. Each extracted feature is then used in the construction of the network.

The PAL algorithm iterates through all possible pairs in the data set that share the same class. Each pair produces a feature, and each feature is ranked by checking it against all instances. This yields a worst-case time complexity that is cubic in the number of instances. Several optimizations to the algorithm reduce this time considerably in practice.

One simple but effective optimization is to skip the evaluation of redundant features. A practical learning

problem will have features that appear many times in the training data and this avoids making many redundant passes through the training set.

Another optimization is to have the algorithm discontinue searching for features if sufficient features have been found to model the distribution of a class. This has a significant benefit because good features cover many instances and therefore show up early in the search, allowing the algorithm to terminate after searching only a relatively few pairs.

A further optimization is to ignore single order features. These features are spurious for most interesting real world problems, but typically occur very frequently. By enforcing a minimum order of two on evaluated features, the algorithm avoids computing the rank for these features.

Finally, for a given instance, if a feature has been found to rank higher than some (empirically determined) threshold, the instance is no longer used to generate features (i.e. pairs). A good feature will cover many instances, and subsequent pairs using the given instance will thus be redundant. Allowing the algorithm to terminate a loop early in this case can significantly speed up execution time.

V. EXPERIMENTS

The PAL algorithm was compared to conventional methods for implementing single hidden layer MLPs using eight real-world learning problems and one artificial learning problem. The objective was to determine the complexity and accuracy of the network model constructed using the PAL algorithm, versus conventional models.

The following data sets were used: blood [22], cancer, credit, echo, iris, lenses, zoo, mushroom, and monk3. The size of each data set is listed in Table I. Except where otherwise noted, all data sets were taken from the UCI repository [19].

All results reported in this research were obtained using a modified form of ten-fold stratified cross validation, where each "holdout" set was used strictly for evaluation after training was completed. A small portion of the training instances was used to determine when to stop training.

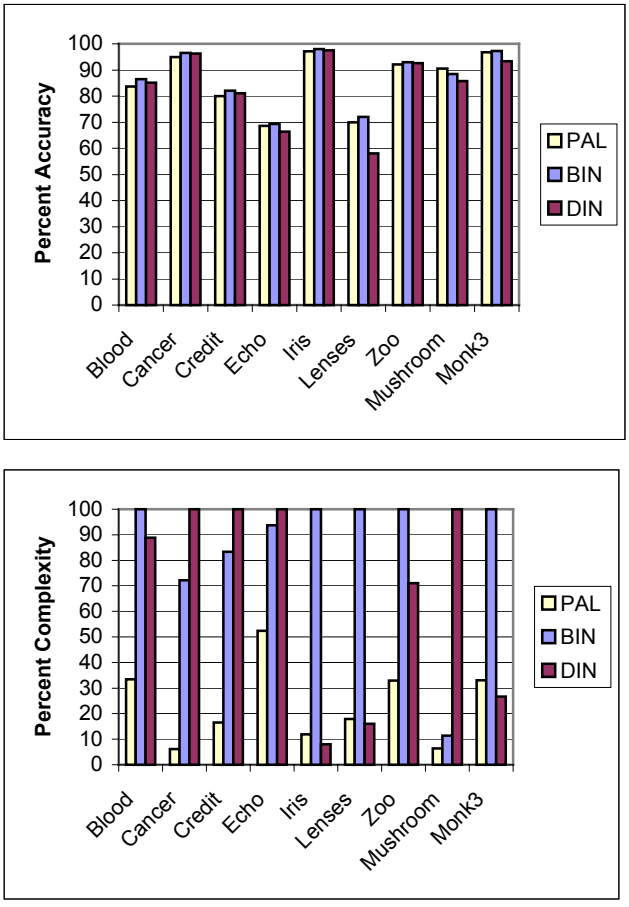
The network corresponding to the best recorded SSE, over the entire training run, was used to measure performance on the holdout set. This implies that even if the network began to overfit the training data, this would not be reflected in the test results because a previous version of the network was saved for use in the testing.

The networks were trained using standard backpropagation with no momentum. The connection weights were first randomly initialized between –0.2 and 0.2 using a uniform distribution. The networks were then trained on-line (vs. batch mode) using a learning rate of 0.2. Continuous inputs were first discretized.

In order to have the most meaningful results, the standard network, comprised of a fully connected single hidden layer MLP, was exhaustively tested to find the best performing topology to use for comparison. Standard networks were built with hidden layers from 1 to 15 nodes, and (for larger problems) 15 to 100 nodes in 5 node increments. The topology with the highest accuracy was selected for comparison with the PAL constructed network. This is referred to as the Best Iterative Network (BIN). The common heuristic of using twice the number of inputs for the size of the hidden layer was also reported for comparison. This is referred to as the Double Input Network (DIN).

Because the number of nodes does not determine the number of connections directly in a PAL constructed network, the metric used for comparing complexity was the number of connections in the network, regardless of the number of nodes. The network complexities for the best iterative topology (BIN), and the heuristic topology (DIN), are pre-determined and remain constant over cross-validation because the topologies are not a function of the training data. This is not true for the PAL algorithm, however. In this case, the PAL algorithm produces a (potentially) different topology for each mix in the cross validation because the training data will contain different examples (and thus different features). To compare network complexities, it was necessary to average the complexities of the PAL constructed topologies over the ten-fold cross-validation. These are the scores reported in the results.

FIGURE III
ACCURACY AND COMPLEXITY FOR THE PAL, BIN, AND DIN NETWORKS



VI. RESULTS

The results of running the PAL algorithm on all nine data sets, along with the results from the BIN and DIN methods, are shown in the bar charts in Figure III. The top chart shows the percent accuracy for each method on each data set. The BIN slightly outperformed the other methods on all but the mushroom data sets, where the PAL algorithm was the most accurate. The PAL algorithm was within 3.4% of the BIN method in every case. The DIN also had very similar performance to the BIN method, but suffered on the Lenses data set. The DIN method under performed PAL on all but three of the problems.

The BIN results represent the best accuracy possible given a standard single hidden layer MLP, since all (reasonable) sizes for the hidden layer were tested. Any constructive technique that produces a standard single hidden layer MLP was implicitly tested by iteratively testing all network sizes. Constructive algorithms that produce alternative topologies were not used for comparison.

The lower chart in Figure III shows the complexity of the resulting network for each method as a percentage of the most complex network. The most complex network is shown as 100%. Table I gives the actual complexities (number of connections) produced by each method, as well as the number of inputs, output classes, and training instances for each learning problem. The PAL algorithm produced significantly less complex networks than the BIN method in all cases. In three cases the DIN method produced slightly less complex networks than the PAL algorithm. These were the Iris, Lenses, and Monk3 data sets. Of the three, the accuracy of the DIN method was only better than the PAL algorithm on the Iris data set.

Overall, the PAL algorithm produced networks 70% less complex than the BIN method, and 38.8% less complex than the DIN method. The accuracy of the PAL networks was only 1.2% lower than the most accurate network (BIN), on average. These results demonstrate that using the PAL algorithm produces small, accurate networks without the computational overhead of iterative construction techniques, or the uncertainties of heuristic approaches.

One shortcoming of the PAL algorithm is that it lacks a facility for removing redundant inputs. Since there is no explicit measure of correlation on inputs within a feature, all inputs that are correlated in an instance pair are used for the feature. Redundant inputs will then appear as connections in the constructed network, adding to the network complexity.

The PAL algorithm potentially discovers all features that appear in the training set. Not every feature is useful for generalizing so some means of selecting the features to drive the network construction must exist. As explained in Section IV, the PAL algorithm ranks the features based on predictive accuracy, but a heuristic had to be empirically determined to set the threshold for feature selection.

TABLE I

NETWORK COMPLEXITIES AND GENERAL PARAMETERS

Learning Problem	Complexity			Parameters		
	BIN	DIN	PAL	Inputs	Outputs	Instances
Blood	54	48	18.1	4	2	209
Cancer	143	198	12.1	9	2	683
Credit	425	510	84.2	15	2	653
Echo	150	160	83.8	8	2	62
Iris	700	56	83.7	4	3	150
Lenses	350	56	62.7	4	3	24
Zoo	1035	736	341.6	16	7	101
Mushroom	120	1056	67.3	22	2	5644
Monk3	360	96	119	6	2	554

The threshold value was chosen to ensure that enough features could be selected to cover each class, but this was not necessarily the optimum for a given problem. This is why for certain problems the PAL network is more complex than the DIN method. More features were selected in this case than were necessary, although the accuracy did not suffer significantly.

VII. CONCLUSION AND FUTURE WORK

Two important issues relating to the implementation of a neural network to solve a classification problem were examined in this paper, namely feature selection and network topology. The choices for certain parameters of the implementation, related to feature selection and topology, can significantly impact the performance of the network. There is presently a need for better methods to address these aspects of neural network design.

The Pair Attribute Learning algorithm addresses both these issues simultaneously by using the results of a feature search to drive network construction. Features are extracted as correlations of instance pairs, and selected based on a statistical measure. A single hidden layer network is constructed with a hidden layer node inserted for each selected feature. The hidden layer node is only connected to inputs that are relevant in the feature. The output layer is fully connected, and the network is trained via standard backpropagation.

Results from nine different experiments show that the PAL algorithm constructs networks that have on average a 70% reduction in complexity when compared to the best performing standard network topology. Although the PAL networks were significantly less complex, the predictive accuracy remained on average within 1.2% of the highest recorded accuracy. This is due to the low order features selected in the first phase of the algorithm that determine the hidden layer connections, and the minimization of redundant or extraneous hidden nodes.

Some parameters of the PAL algorithm, such as the cumulative rank threshold, were determined empirically over the nine data sets used in the experiments. Future work will focus on removing dependence on these parameters, such as dependent ranking to avoid overlap. Also, further work will extend the feature search to more general features by reducing instance pair correlations (e.g. removing spurious inputs). Other extensions to be explored include optimizing the algorithm to improve execution time, provisions for problems where class outputs are underrepresented by the selected features, and sparsely connecting the output layer.

VIII. REFERENCES

[1] D. W. Aha and R. L. Bankert, "A Comparative Evaluation of Sequential Feature Selection Algorithms," D. Fisher and J. H. Lenz (eds.), *Artificial Intelligence and Statistics V*. New York: Springer-Verlag, 1996.

[2] H. Almuallim and T. G. Dietterich, "Learning with Many Irrelevant Features," *Proceedings of the Ninth National Conference on Artificial Intelligence*, 547-522. San Jose, California: AAAI Press, 1991.

[3] T. L. Andersen and T. R. Martinez, "A Dynamic Multi-Layer Perceptron Construction Algorithm," *International Journal of Neural Systems*, 11(2):145-166, 2001.

[4] C. Cardie, "Using Decision Trees to Improve Case-Based Learning," *Proceedings of the Tenth International Conference on Machine Learning*, 25-32. Amherst, Massachusetts: Morgan Kaufmann, 1993.

[5] Y. A. Chauvin, "A Back-propagation Algorithm with Optimal Use of Hidden Units," *Advances in Neural Information Processing Systems* (1), ed. D. S. Touretzky, pp. 519-526. Morgan Kaufmann, San Mateo, 1989.

[6] P. Domingos, "Context-Sensitive Feature Selection for Lazy Learners," *Artificial Intelligence Review*, 11:227-253, 1997.

[7] S. E. Fahlman C. and Lebiere, "The Cascade-Correlation Learning Algorithm," *Technical Report CMU-CS-90-100*, Carnegie Mellon University, 1991.

[8] B. Hassibi and D. G. Stork, "Optimal Brain Surgeon," *Advances in Neural Information Processing Systems* (5), eds. S. J. Hanson, J. D. Cowan, and C. L. Giles, pp. 164-171. Morgan Kaufmann, San Mateo, 1993.

[9] M. Ishikawa, "A Structural Learning Algorithm with Forgetting of Link Weights," *Technical Report TR-90-7*. Electrotechnical Laboratory, Tsukuba-City, Japan, 1990.

[10] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subset Selection Problem," *Machine Learning: Proceedings of the Eleventh International Conference*, 121-129, 1994.

[11] J. H. Kim and S. K. Park, "The Geometrical Learning of Binary Neural Networks," *IEEE Transactions on Neural Networks*, 6(1), 1995.

[12] K. Kira and L. Rendell, "A Practical Approach to Feature Selection," *Proceedings of the Ninth International Conference on Machine Learning*, 249-256. Aberdeen, Scotland: Morgan Kaufmann, 1992.

[13] J. K. Kruschke, "Creating Local and Distributed Bottlenecks in Hidden Layers of Back-propagation Networks," *Proceedings of the 1988 Connectionist Models Summer School*, eds. D. Touretzky, G. Hinton, and T. Sejnowski, pp. 120-126, Morgan Kaufmann, San Mateo, 1989.

[14] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," *Advances in Neural Information Processing Systems*, 2:598-605, 1990.

[15] M. Lehtokangas, "Modelling with Constructive Backpropagation," *Neural Networks*, 12:707-716, 1999.

[16] T. R. Martinez, B. Hughes, and D. M. Campbell, "Priority ASOCS," *Journal of Artificial Neural Networks,* 1(3), pp. 403-429, 1994.

[17] J. Moody, "Prediction Risk and Architecture Selection for Neural Networks," *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, eds. V. Cherkassky, J. H. Friedman, and H. Wechsler, NATO ASI Series F, Springer-Verlag, 1994.

[18] M. C. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat From A Network Via Relevance Assessment," *Advances in Neural Information Processing* (1), 107-115. D. S. Touretzky, Ed., 1988.

[19] P. M. Murphy, "UCI Repository of Machine Learning Databases," Machine-Readable Data Repository, Department of Information and Computer Science, University of California at Irvine, Irvine, California, 1995.

[20] D. W. Opitz and J. W. Shavlik, "Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies," *Journal of Artificial Intelligence Research* 6, pp. 177-209, 1997.

[21] D. C. Plaut, S. J. Nowlan, and G. E. Hingon, "Experiments on Learning by Back Propagation," *Technical Report CMU-CS-86-126*, Carnegie-Mellon University, Pittsburgh, 1986.

[22] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals Through Simulations*. New York: John Wiley & Sons, Inc, 2000.

[23] R. D. Reed and R. J. Marks II, *Neural Smithing – Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, Massachusetts: The MIT Press, 1999.

[24] A. Roy, L. S. Kim, and S. Mukhopadhyay, "A Polynomial Time Algorithm for the Construction and Training of a Class of Multilayer Perceptrons," *Neural Networks*, 6:535-545, 1993.

[25] R. Setiono and L. C. K. Hui, "Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm," *IEEE Transactions on Neural Networks*, 6(1), 1995.

[26] J. Sietsma and R. J. F. Dow, "Creating Artificial Neural Networks that Generalize," *Neural Networks* 4(1), pp. 67-79, 1991.

[27] J. M. Steppe, K. W. Bauer Jr., and S. K. Rogers, "Integrated Feature and Architecture Selection," *IEEE Transactions on Neural Networks*, 7(4), 1996.

[28] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Back-propagation, Weight-elimination and Time Series Prediction," *Proceedings of the 1990 Connectionist Models Summer School*, eds. D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, pp. 105-116. Morgan Kaufmann, San Mateo, 1991.

[29] D. Whitley and C. Bogart, "The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms," *Proc. Int. Joint Conf. Neural Networks*, Vol. 1, p. 134. IEEE, New York, 1990.

[30] S. Young and T. Downs, "CARVE – A Constructive Algorithm for Real-Valued Examples," *IEEE Transactions on Neural Networks*, 9(6), 1998.