Extracting Rules from the Aggregate Feedforward Neural Network

Vincent A. Schmidt
C.L. Philip Chen
Wright State University
Dayton, OH, U.S.A.

Abstract

The Aggregate Feedforward Neural Network (AFFNN) is a connectionist model that learns the relationships leading to all K input attributes using a single neural network. This unified network behaves as if it is an aggregation of K distinct networks, each trained for a specific target. This paper describes the creation of the AFFNN using one of the well-known Three MONK's Problems, and then demonstrates the use of a decompositional rule extraction approach for obtaining and expressing the knowledge learned by the AFFNN.

Keywords: Neural Networks, Data Mining, Rule Generation

1 Introduction

It is increasingly common to find connectionist models performing data mining operations, discovering useful and non-obvious patterns in collections of data. Various neural network architectures already actively participate in the data mining arena. One example is the Knowledge-Based Artificial Neural Network (KBANN) system by Shavlik, Towell, Craven, and other researchers, which creates a neural network with weights and links initialized based on available domain knowledge and can extract M-of-N and rules [1, 2, 3]. Another such system is the NeuroRule system, a feedforward neural network with a single hidden layer. This system uses hyperbolic tangent activation functions along with a penalty-based error function and specialized pruning steps reduce the size of the network before applying a decompositional rule extraction algorithm [4, 5].

Although these systems are reasonably flexible, they suffer from a characteristic common to most neural networks: the desired target attribute set must be determined prior to training. For an arbitrary dataset with K distinct attributes, establishing the relationships of each of the K attributes as a function of the remaining attributes in the dataset will require training and interpreting K separate neural networks.This can be both tedious and time-consuming.

An interesting connectionist system that circumvents this problem is the Aggregate Feedforward Neural Network (AFFNN), a connectionist system that learns individual dataset attributes as a function of the remaining attributes within the same neural network [6]. The AFFNN is a system consisting of a connectionist model and a small set of specialized support functions promoting the desired behavior. The network itself is a fully-connected feedforward neural model with a single hidden layer. This network contains the same number of nodes in the input and output layers. Hidden nodes often use sigmoid activation functions while output nodes are typically linear, but this is not a strict requirement for the AFFNN.

The AFFNN associates a collection of input and output nodes with each of the K attributes in the dataset. All attributes are used in training as both the source and the target data. This network is not autoassociative, however, because the network outputs for a specific attribute are guaranteed by the AFFNN system to rely only on the remaining attributes, a feature enforced by the system's support functions. The nature of this system demonstrates that the relationships leading to all K attributes can be learned simultaneously within a single neural network. This allows rules to be extracted from the trained AFFNN system for any arbitrary attribute without requiring a priori target attribute selection required in other systems.

The following sections of this paper briefly describe the AFFNN as it supports rule extraction for the example provided.

2 AFFNN Creation and Training

The activities required to create and train an AFFNN are similar to those used for other neural models. The K attributes of the dataset should be carefully selected and encoded using some binary encoding scheme, such as discretization or thermometer encoding. The size of an encoded input vector containing all attributes defines the number of AFFNN input and output nodes. The number of nodes in the hidden layer should be determined empirically, and should be expected to be a larger when describing an AFFNN than in cases where a traditional neural network with only a single output attribute is to be generated.

The user examining output vector components corresponding to the encoding of a particular attribute A_i, $1 \leq i \leq K$ can be assured that attribute A_i did not contribute as an input to the network for the output nodes in question. (Although A_i is permitted to contribute to all other output nodes.) The AFFNN relies on a set of specialized preprocessing and postprocessing functions to maintain exclusion between attributes during training. The preprocessing function replicates each input case K times, masking out consecutive attributes. This process increases the number of input vectors by a factor of K. The preprocessed vector set is used as input cases for AFFNN training. Every input case presented to the network produces an output vector. The postprocessing function takes each set of K output vectors and uses a similar masking scheme to "collapse" them back into a single output vector, reducing the final number out output cases to match the number of original input cases. All encoded portions of postprocessed output vectors corresponding to individual attributes A_i, $1 \leq i \leq K$ are valid for every attribute. That is, the postprocessing function modifies the network outputs such that each attribute represented in the output vector is guaranteed to be a function only of the remaining attributes.

The AFFNN also uses a custom performance function during training to support the user-selected supervised training algorithm. This function intercepts intermediate values immediately before network training error is computed. Once obtained, the intermediate values are refined by clearing specific parts of the intermediate outputs that should not have an impact on network error, an operation performed using the same mask employed by the preprocessing and postprocessing functions. Thus the custom performance function ensures that only the links leading to the nodes supporting a specific attribute will have their weights modified. This feature prevents the weights leading to nodes of one output attribute from interfering with the training the weights for output nodes of another attribute. The performance function then passes these processed results to the original performance function for the standard error processing needed by the supervised training algorithm. These capabilities are described in more detail in the AFFNN references. Note that the AFFNN is not constrained to use a specific supervised training algorithm or error function; it can use many training algorithms and error functions without modification.

The resulting network output is a collection of output vectors with the same number of elements as the input vectors. The relevant portions of these vectors are the result of expressing each attribute in terms of the remaining attributes. The trained AFFNN is also capable of classifying previously unseen data. The next objective is to methodically extract a series of rules from the trained network such that these rules correctly express the relationships learned by the solution network. These rules are also capable of classifying previously unseen data.

3 Rule Extraction

Decompositional approaches are frequently used to extract rules from connectionist systems. Such algorithms use weight and bias information from within the trained network in order to create relevant rules. The technique we implemented is similar in some regards to the RX algorithm used in the NeuroRule system, in that discretized hidden node outputs are generated and considered for rule production [4, 7]. Although we currently use a custom-defined strategy and implementation, any reasonable rule extraction algorithm could potentially be used.

Figure 1 depicts an AFFNN architecture trained for an arbitrary dataset with seven attributes, where each attribute is encoded into a collection of nodes. The boxes around sets of input and output nodes in the figure indicate the number of nodes encoding each attribute. Many links were consolidated to the attribute level in the figure to promote clarity. In reality, the network is fully connected from all nodes at the input layer to the hidden layer, and from all hidden layer nodes to all nodes in the output layer. According to the definition of the AFFNN, examining nodes for a particular output attribute ensures that the same attribute did not contribute to the network training. For example, when output attribute $A7$ is the attribute of interest, only the links leading from the hidden layer to the output nodes corresponding to attribute $A7$ are examined in this layer, and the contributions of all $A7$-related links from the input layer to the hidden must be ignored. This configuration is indicated by the solid (contributing) and dashed (noncontributing) links in the figure. This view of the trained network exhibits the outputs on the basis of individual attributes, with each attribute expressed as a function of the remaining attributes. Rules can be produced describing these relationships.

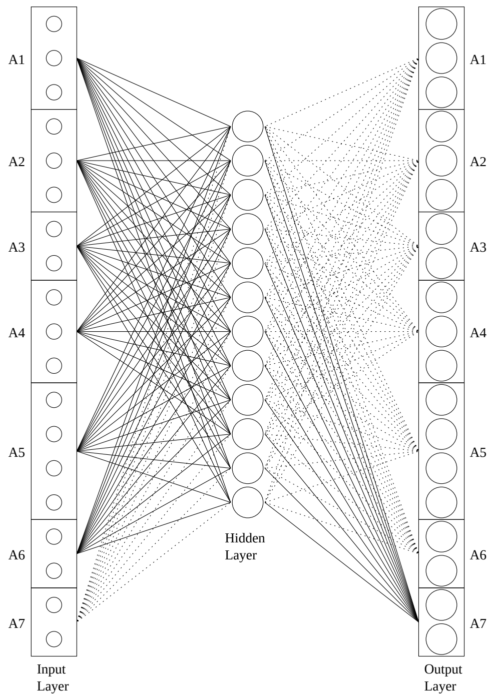

Figure 1: For a Single AFFNN Attribute

The first step of rule extraction is to identify the output node for which rules are desired, then discretize the activation values produced by the hidden nodes leading to this output. The selection of one discretized output from each hidden node defines a single test case. For small to medium-sized networks, there are not generally very many clusters of discretized activation values, so determining all combinations of values producing the desired output is a straightforward task. The valid combinations of values are called “candidate expressions” for the AFFNN system.

Candidate expressions consist of terms, where each term is one of the discretized hidden node values. Inputs applied to the input nodes produce values that are members of the discretized hidden node output sets. A combination of inputs producing all terms in a candidate expression is a valid solution for the specified output node, and is added to the solution set. Since inputs are frequently encoded when training neural networks, it can be impractical to test all combinations of the encoded inputs due to combinatorial explosion. We have devised an effective method to relax the dimensionality of such data by remapping it back into a space that more closely corresponds to the original input space before searching for valid solutions [8]. This relaxation technique has been demonstrated to significantly reduce the time and complexity of finding valid rules when using this decompositional extraction approach.

When valid solutions have been collected and simplified, it is a simple exercise to generate a series of *if-then* rules representing the knowledge expressed by the solution set. These rules are easily readable by subject matter experts and are manifested in a form capable of processing input cases independently of the trained AFFNN. Both the extracted rules and the trained AFFNN are products that can continue to be used for classifying new data.

The rule extraction algorithm represents its results using MATLAB notation. These results are encapsulated within a MATLAB function, allowing the rules to be applied against other sample data cases. The accuracy of the rules produced and encoded by the rule generator depends on the accuracy of the neural network from which they are obtained.

4 Example

The Three MONK’s Problems are fabricated datasets in which robots are described using a combination of six discrete-valued attributes [9]. These datasets were originally designed to assist in the comparison of various classification algorithms. Robots belong to the solution set of the second of these problems, problem M_2, when exactly two of the six attributes are assigned to their initial values. The attributes and their options include: head shape (round, square, octagon), body shape (round, square, octagon), smiling (yes, no), holding (sword, balloon, flag), jacket color (red, yellow, green, blue), and tie (yes, no). There are 432 exhaustive combinations of these attributes.

The AFFNN adds another attribute, in_solution_set (yes, no), for a total of 7 attributes. Simple mutually exclusive encoding extends these attributes to 19 neural network inputs, corresponding to each combination of the encoded values of all 7 attributes. The AFFNN was designed as a fully connected 19-12-19 network with sigmoid activation functions in the hidden layer and linear activation functions at the output.

The training set consists of the specified 169 training vectors for problem M_2, encoded into the 19-element input vectors. Training for 1000 epochs was completed in just over 6 seconds on a dual Pentium-II Linux-based system using a custom AFFNN MATLAB implementation. Problem M_2 was designed to portray a single relationship in which robots possessing various combinations of the original six attributes belong to the solution set. Examining AFFNN output from this perspective, there were no misclassifications in the trained network.

Figure 1 is actually the AFFNN architecture used for training and extracting rules from M_2. Attribute seven (in_solution_set) is the attribute of interest in this example, so the figure correctly reflects the node and link configuration used during rule extraction. The first node in this set represents in_solution_set=no, and the second node represents in_solution_set=yes.

Rule extraction proceeds by selecting the output node for which rules should be extracted, which in this case is the output node representing the attribute/value combination in_solution_set=yes. Discretizing and testing hidden node values as described above yields 3 candidate expressions, all based solely on values of hidden nodes 3, 6, and 7. This means that the contributions of only three AFFNN hidden nodes determines the ultimate outcome for the selected output node. The automated rule generation process examined all relevant combinations of (relaxed) inputs producing the values of each expression and generated 15 rules after simplification, the minimum number of rules required to faithfully represent the solution to problem M_2. Generated rules follow a basic structure that is easy to interpret:

```
if (
    (x1 <= 1) and
    (x2 <= 1) and
    (x3 > 1) and
    (x4 > 1) and
    (x5 > 1) and
    (x6 > 1)
) then result=true;
```

This is one of fifteen M_2 rules correctly describing the desired characteristic: exactly two of the six attributes have their initial values. Specifically, this rule states that the first and second attribute have their initial values and the other attributes do not. The MATLAB function containing the rules extracted from M_2 correctly classifies all input cases as a function of attributes $A_1..A_6$ with no misclassifications.

5 Summary

The AFFNN is a novel connectionist architecture that simultaneously learns the relationships of each of the K attributes for a given dataset with respect to the remaining attributes. The techniques used for training the AFFNN prevent the network from becoming autoassociative.

One key benefit to using the AFFNN is that rules can be extracted for any attribute once network training is completed. Since the AFFNN behaves as if it were a unified collection of individual networks, rules can be extracted from any output node independently of the rest of the outputs. This feature allows the user to defer the selection of potentially interesting outputs until after training has completed, without taking a significant performance hit (i.e. needing to create and train an entirely new network) when examining rules for other outputs and attributes.

The M_2 problem presented here is clearly a small example, but it is able to demonstrate the fundamental issues concerning the basic training and rule extraction paradigm used with the AFFNN. This problem is also useful as a benchmarking problem, since it is commonly used for connectionist and other data mining tests. (The M_2 problem has been most helpful in validation testing, to ensure nothing has been "broken" as rule extraction research continues, but it is not an ideal problem for exhaustive AFFNN training, since only a single attribute has a well-defined and popular relationship.) Solving this caliber of problem and automatically generating the corresponding rule set is a proof of concept that merely qualifies the AFFNN for additional research.

There is still work to be done as the AFFNN and its rule extraction mechanism is developed. Like other neural networks, topology configuration, activation function and training algorithm selection, and convergence issues all impact successful use of the AFFNN. In addition, rule extraction techniques are currently limited to an AFFNN trained using various forms of binary-encoded data. These and other issues remain to be addressed as research continues.

References

[1] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13:71–101, 1993.

[2] Geoffrey G. Towell and Jude W. Shavlik. Using symbolic learning to improve knowledge-based neural networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 177–182, San Jose, CA, 1992. AAAI/MIT Press.

[3] David W. Opitz and W. Shavlik, Jude. Dynamically adding symbolically meaningful nodes to knowledge-based neural networks. *Knowledge-Based Systems*, 8(6):301–311, 1995.

[4] Hongjun Lu, Rudy Setiono, and Huan Liu. NeuroRule: A connectionist approach to data mining. In *Proceedings of the 21st VLDB Conference*, Zurich, Swizerland, 1995.

[5] Rudy Setiono. A penalty function for pruning feedforward neural networks. *Neural Computation*, 9(1):185–204, 1997.

[6] Vincent A. Schmidt and C. L. Philip Chen. Defining an aggregate feedforward neural network. Submitted to ANNIE02 Conference, 2002.

[7] Hongjun Lu, Rudy Setiono, and Huan Liu. Effective data mining using neural networks. In *Proceedings of the 21st VLDB Conference*, Zurich, Swizerland, 1995.

[8] Vincent A. Schmidt and C. L. Philip Chen. Data dimensionality relaxation for neural networks trained with encoded data — an application in rule extraction. Submitted to ANNIE02 Conference, 2002.

[9] S.B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S.E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R.S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different algorithms. Technical report, Carnegie Mellon University, 1991. CMU-CS-91-197.