CHAPTER 5

NEURAL NETWORK ENSEMBLE SIMULATOR

The creation of network ensembles with pre-existing neural network simulators can be laborious and time consuming. Statistical analysis of these ensembles is even more difficult, as most simulators provide only a reporting of individual networks. The creation, selection, and analysis of ensemble networks is therefore performed in an ad hoc fashion, with the statistical analysis performed either manually or by some external utility after the network results have been stored. If network ensembles are to be of any practical use, they must be efficiently generated and provide statistical feedback automatically upon creation. The Neural Network Ensemble Simulator (NNES) presented as part of this research is created for just such a purpose. NNES supports the creation and combination of unlimited students into the ensemble, while maintaining a voluminous array of statistical data about the ensemble and its student members. A complete listing of the simulator code is provided in Appendix G.
5.1 Overview of NNES

NNES is a complex simulation tool designed to support supervised learning and FFNN architectures. The simulator produces an ensemble of networks called “students” [7]. Each student represents a unique solution (network representation) of the given problem. The individual students are created by altering experiment parameters or the exemplars available for training. The students are combined by NNES to form an ensemble prediction or classification. Experiment results are then written to permanent files. See Appendix A for NNES documentation.

5.1.1 Architecture of the Model

Students within a given ensemble are trained using the same learning algorithm and architectural model. The learning algorithms currently supported by NNES are the delta rule (LMS or gradient descent) [27] and the quickprop learning algorithm [16]. Architectural models supported include the Adaline single-layer linear network [27], and the Cascade-Correlation constructive network [16].

The ensemble model relies on the assumption that each student makes a unique contribution. As pointed out in section 2.3, this requires ambiguity between students. Ambiguity may be accomplished by altering the initial weight vector, altering one or more of the modifiable training parameters, subdividing the training data, re-training over subsets of the training data, or performing k-fold cross-validation on the training and validation set.
5.1.2 Statistical Reporting

Information is maintained about each potential ensemble student during the creation process. This information includes the network’s performance on the training data, validation data and test data, along with its topology (number of units, connections, and weights). Settings for all modifiable training parameters are also maintained, as they may vary between networks.

Results from the creation process are analyzed prior to network selection and combination. The networks are evaluated for inclusion into the ensemble and their ensemble contribution factors are derived based on the combination method chosen. NNES currently supports voting and weighted contribution based on validation error.

The best and worst networks are chosen for each data set (training, validation, and test), based on correctness or mean square error ($MSE$). The statistical results, file settings, parameter settings, and ensemble contribution for these networks are permanently recorded, along with the overall ensemble results. Statistical results include number of hidden units, number of epochs required for training, MSE, sum square error ($SSE$), root mean square error ($RMSE$), bits wrong, (if binary), and percentage correct.

5.2 Implementation Details

5.2.1 Interfacing with the Cascade Neural Network Simulator

The NNES driver invokes local software components to run the experiments, create linear Adaline networks, combine students, analyze student performance, and
maintain statistical data. However, to create constructive networks NNES must invoke
Cascade-Correlation. The Cascade Neural Network Simulator (CNNS), version 1.0 by
Matt White, is chosen [121] to create the Cascor networks. An interface between the
two simulators creates communication of the run-time parameters and the network
results via system calls and temporary files. Only slight modifications are made to the
original CNNS source to create temporary files for passing network parameters,
statistical data, and predicted outputs (see Appendix B). CNNS trials are generated by
script files invoked from within the NNES run-time environment. When CNNS trials
are complete, control returns to NNES for further analysis, combination, and statistical
reporting.

5.2.2 Data Format Requirements

NNES is designed to be as compatible as possible with existing data repositories.
Upon investigation, it is determined that the PROBEN1 Benchmark Set provides one of
the most useful data repositories for neural network benchmark testing [96]. Most of the
data sets in PROBEN1 have been acquired from the highly regarded UCI Machine
Learning Database [122] and encoded for direct use by supervised learning networks.
All of the software components developed for NNES support the PROBEN1 data
format. Additionally, these components support the data format required by the 1992
Crowder version of Cascor [123].

The Cascor component of NNES relies upon the CNNS data format
requirements. CNNS was written to be compatible with the Carnegie Mellon University
Learning Benchmark Collection [118]. To accommodate Cascor learning on PROBEN1
data sets, an external translation script is written and included with the NNES software utilities. The script converts PROBEN1 formatted data to the CMU data format. Likewise, data sets in the CMU collection can be used for Cascor Ensemble creation within NNES.

An NNES experiment may access multiple data files for training, validating, and testing of an ensemble. Each student in the ensemble may itself be trained on separate data files. The separate files must follow a specified naming convention (see Appendix A), and are opened and read in turn automatically by the simulator. An option for entering data interactively is also provided.

5.3 Features of NNES

5.3.1 Algorithms

The *Adaline* (ada) feature of NNES is based on R.C. Lacher’s C source code, which implements a classic Widrow and Hoff Madaline model. Batch learning is performed on the training set until the error is below the error-tolerance parameter or until *max epochs* is reached. The Adaline component is designed to support both binary and continuous inputs and outputs.

The Cascor (*cc*) option invokes a locally modified version of CNNS. Developed by Matt White, CNNS is based on previous work by Scott Fahlman, Scott Crowder, Peter McCluskey, Conor Doherty and Michael Kingsley [121]. NNES is designed to support all of the features available through CNNS. However, for the scope of this
research the implementation and testing focus only on Cascor learning with a select number of modifiable parameters. Cascor trains on the training set until the error is below the error-tolerance (threshold) parameter or until \textit{max epochs} is reached. NNES is initially implemented to support binary and continuous-valued inputs and binary outputs. With minimal modifications to the interprocess interface, it will also support continuous-valued outputs from CNNS.

\textbf{5.3.2 Combination Methods}

This research focuses on two combination methods: voting and weighted averaging. Voting is accomplished in NNES by taking the plurality vote over all students. Each student is allowed an equal vote on the prediction or classification of each output in the test set. The vote most commonly cast marks the decision of the ensemble. Voting is the most common approach employed by linear ensembles for classification.

However, giving each student equal weight in the decision-making is not always appropriate. Some students may have overfit the training data and hence perform poorly on unobserved data. Theoretically, a method that rewards students that are better able to generalize should prove to be more advantageous. Weighted averaging supports this concept by allowing each student to contribute to the ensemble decision at unique levels. In NNES this level of contribution is based on the individual student’s performance on the validation set (subset of the available training data). For continuous-valued outputs, the ensemble prediction is merely the sum of products for each output and its contribution factor. For binary classification, the summation is passed through a binary
activation function. The NNES combiner can also be extended to handle multiple-classification problems, as outlined in section 4.2, by applying the SBCM. This feature will be incorporated into future versions of NNES.

5.3.3 Modifiable Parameters

Users of NNES select the learning algorithm and combination method for each experiment. They also indicate the type of experiment (traditional or ensemble) and the number of respective networks or students. The data source must also be carefully specified. The experimenter is then presented with a list of parameter settings and the option of accepting the defaults or specifying alternate values. Some of these parameters remain constant for the duration of the experiment (number of inputs and outputs, ensemble or traditional experiment, algorithm and method). However, users may choose to alter other select parameters between network creations, when the step command option is chosen. This is especially useful if the networks are repeatedly poorly trained and counted as “failures”, which cannot be included as ensemble students. Another benefit is the ability to alter parameters between nonlinear network creation to achieve diverse networks from common data.

Modifiable parameters for both Adaline and Cascor learning include the number of training patterns, validation patterns, and test patterns. This is usually dictated by the input file. However, PROBEN1 formatted data files allow the user of NNES flexibility to alter these parameters easily at run-time. The error tolerance and max epochs can also be modified between network trials under both algorithms. By default, all networks created under NNES begin with a unique random weight vector.
A select number of CNNS parameters are chosen to be modifiable from within the current NNES run-time environment. These parameters include the learning rate, score-threshold for training, validation, and testing, and the max output epochs for training. Modifiability is limited to only these parameters for model simplicity. In past experiments, the author of this research has found these parameters to be the most influential over multiple domains. Future versions of NNES will support a larger number of modifiable parameters for ensemble creation under the Cascor learning algorithm.

5.3.4 Future Enhancements

The focus of this research on binary classification directs the current implementation of NNES. However, the greater vision of NNES is that of a complex simulator capable of modeling multiple domain types effectively, while providing the experimenter with a vast array of options for ensemble creation, selection, combination, and evaluation. With this in mind, future enhancements to the simulator are already underway. NNES is designed with many of these features in mind, making implementation trivial.

Future versions of NNES can be expected to include opportunities for cross-validation level 1 (cv1) and cross-validation level 2 (cv2). Under cv1, cross-validation is performed on the available training set (for training and validation phases) as a method of achieving diverse networks from subsets of the training data. At a higher level, cv2 provides cross-validation over all available data (including test data), as a complete performance measure of a fixed ensemble. The necessary structures for cv1
and cv2 are already embedded in the NNES source under Adaline learning, and will be incorporated for Cascor learning as well.

The modular design of NNES components provides support for independent training, validation, testing, and prediction. These features will be made available to users of NNES in future versions. Other anticipated features include: implementation of the SBCM component, support for backpropagation, enhanced student selection process, increased Cascor parameter options, support for NNES script processing, and a GUI interface.

5.4 Validating the Model

5.4.1 Linear Problem

Basic testing of the completed NNES begins with a simple linear problem. The purpose of this test is to validate that the simulator is working correctly for the simplest case. In this scenario, there are two binary inputs and one binary output. The output is simply a linear function on (y). The training data consisted of all four possible input pairs. Training and testing is performed for all models supported by NNES, and repeated for ten trials each. Ensemble networks consist of three students, trained on identical data sets. In all cases, NNES traditional networks and ensemble networks are able to easily identify the function correctly.

The default parameters are sufficient for learning under all models, with a learning rate of 0.1, Adaline error tolerance of .01, and Cascor training threshold of 0.1.
The average number of epochs required for Adaline networks is 28, and for Cascor is 9 (Table 1). As indicated by Table 1, ensembles are not necessary for solving linear problems.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Combination</th>
<th>Epochs</th>
<th>Error Rate</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaline</td>
<td>None</td>
<td>32</td>
<td>0.00 %</td>
<td>0.0094</td>
</tr>
<tr>
<td>Adaline</td>
<td>Voting</td>
<td>17</td>
<td>0.00 %</td>
<td>0.0090</td>
</tr>
<tr>
<td>Adaline</td>
<td>Weighted</td>
<td>34</td>
<td>0.00 %</td>
<td>0.0095</td>
</tr>
<tr>
<td>Cascor</td>
<td>None</td>
<td>9</td>
<td>0.00 %</td>
<td>0.0084</td>
</tr>
<tr>
<td>Cascor</td>
<td>Voting</td>
<td>9</td>
<td>0.00 %</td>
<td>0.0064</td>
</tr>
<tr>
<td>Cascor</td>
<td>Weighted</td>
<td>9</td>
<td>0.00 %</td>
<td>0.0069</td>
</tr>
</tbody>
</table>

5.4.2 XOR Problem

The next problem to be tested on NNES is the classic XOR problem. Again, the reason for performing this test is to validate the basic functionality of the simulator. As expected, NNES successfully solves the XOR problem for all models except traditional Adaline single networks. A single Adaline network is only capable of learning three of the four possible patterns at best. With this in mind, the data are partitioned into four overlapping sets of three training patterns each. The Adaline networks generated by the different data sets are then successfully combined by NNES into an ensemble capable of correctly classifying all four XOR patterns.

The NNES default parameters are accepted for XOR learning and the maximum number of epochs is set at 2000. Traditional Adaline networks are able to correctly learn three of the four patterns on average in 151 epochs. Traditional Cascor networks
are able to learn all four patterns in 70 epochs. Cascor Ensembles responsible for only three patterns each can learn after only 14 epochs (Table 2). To extend the concept of combining linear classifiers as discussed in CHAPTER 1, Cascor Ensembles demonstrate an ability to correctly classify all four patterns when combining any three of the Cascor networks created from disjoint data sets. The combined effort in this case required only 32 total epochs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Combination</th>
<th>Epochs</th>
<th>Error Rate</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaline</td>
<td>None</td>
<td>2000</td>
<td>25.00 %</td>
<td>0.2567</td>
</tr>
<tr>
<td>Adaline</td>
<td>Voting</td>
<td>49</td>
<td>0.00 %</td>
<td>0.0000</td>
</tr>
<tr>
<td>Adaline</td>
<td>Weighted</td>
<td>98</td>
<td>0.00 %</td>
<td>0.0000</td>
</tr>
<tr>
<td>Cascor</td>
<td>None</td>
<td>70</td>
<td>0.00 %</td>
<td>0.0048</td>
</tr>
<tr>
<td>Cascor</td>
<td>Voting</td>
<td>14</td>
<td>0.00 %</td>
<td>0.0000</td>
</tr>
<tr>
<td>Cascor</td>
<td>Weighted</td>
<td>14</td>
<td>0.00 %</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

5.5 Preliminary analysis of NNES

NNES is systematically validated on simple linear and nonlinear problems. In both sets of experiments, numerous trials are performed and statistics recorded and verified. The results and derived computations are carefully checked for accuracy (see Appendix C for sample run-time results). The simulator is performing as expected and reporting appropriate results. Testing of the simulator on more complex data sets demonstrates its power as a new machine learning model. These results are presented in Chapter 6.