# Network Simplification Through Oracle Learning

Joshua Menke, Adam Peterson, Mike Rimer, Tony Martinez
Computer Science Department, Brigham Young University, Provo, UT, 84602
*Email:* {josh, adam, mrimer}@axon.cs.byu.edu, martinez@cs.byu.edu

**Abstract – Often the best artificial neural network to solve a real world problem is relatively complex. However, with the growing popularity of smaller computing devices (handheld computers, cellular telephones, automobile interfaces, etc.), there is a need for simpler models with comparable accuracy. The following research presents evidence that using a larger model as an oracle to train a smaller model on unlabeled data results in 1) a simpler acceptable model and 2) improved results over standard training methods on a similarly sized smaller model. On automated spoken digit recognition, oracle learning resulted in an artificial neural network of half the size that 1) maintained comparable accuracy to the larger neural network, and 2) obtained up to a 25% decrease in error over standard training methods.**

## I. INTRODUCTION

As Le Cun, Denker, and Solla observed in [1], often the best artificial neural network (ANN) to solve a real-world problem is relatively complex. They point to the large ANNs Waibel used for phoneme recognition in [2] and that of Le Cun et al. with handwritten character recognition in [3]. "As applications become more complex, the networks will presumably become even larger and more structured." [1] The growing complexity of neural networks in real-world applications presents a problem when using them in environments with less available memory and processing power (i.e. embedded systems like handheld computers, cellular telephones, etc.). Therefore, there is a demand to create smaller, faster, neural networks that still maintain a similar degree of accuracy. The oracle learning solution involves using the most accurate available model as an oracle to train a smaller model. We propose that oracle learning will result in simpler models that 1) have accuracy comparable to their oracles, and 2) have improved results over standard training methods for the same sized model. For the following experiment, simple feed-forward single-hidden layer ANNs were used as both the oracle and the *oracle-trained network* (OTN). We propose the use of the following nomenclature for classifying OTNs within this paper:

OTN ($n \rightarrow m$)

Reads "an OTN approximating an $n$ hidden node ANN with an $m$ hidden node ANN." For example:

OTN (200 → 100)

Reads "an OTN approximating a 200 hidden node ANN with a 100 hidden node ANN." The rest of the paper describes oracle learning in terms of ANNs since the experiments deal solely with ANNs. We refer to the oracle as an *oracle ANN* (which is no different than a standard ANN, it is just used as an oracle).

One of the advantages of using an ANN as an oracle is the ability to use unlabeled training data to train smaller ANNs. In speech recognition, for example, there are more than enough data, but it is difficult and expensive to hand label them at the phoneme level. However, if an oracle ANN exists, the smaller ANN can theoretically request as many labeled data points as is necessary to best approximate the larger or oracle ANN.

The idea of approximating a more complex model is not entirely new. Domingos used Quinlan's C4.5 decision tree approach from [4] in [5] to approximate a bagging ensemble and Zeng and Martinez used an ANN in [6] to approximate a similar ensemble (both using the bagging algorithm Breimen proposed in [7]). Craven and Shevlik used a similar approximating method to extract rules [8] and trees [9] from ANNs. Domingos and Craven and Shevlik used their ensembles to generate training data where the targets were represented as either being the correct class or not. Zeng and Martinez used a target vector containing the exact probabilities output by the ensemble for each class. The following research also used vectored targets similar to Zeng and Martinez since Zeng's results supported the hypothesis that vectored targets "capture richer information about the decision making process . . ." [6].

While, previous research has focused on either extracting information from neural networks [8,9], or using statistically generated data [5,6] for training, the novel approach we propose in this paper is to use the approximated network as an oracle. The next section explains the details of the oracle learning process.

## II. ORACLE LEARNING

Oracle learning involves the following 3 steps:

A. Oracle Preparing
B. Data Labeling
C. Oracle Learning

### *A. Oracle Preparing*

The primary component in oracle learning is the oracle itself. Since the accuracy of the oracle ANN directly influences the performance of the final, simpler ANN, the

oracle should be the most accurate classifier available, regardless of complexity (number of hidden nodes). The only requirement is that the number and type of the inputs and the outputs of each ANN (the oracle and the OTN) be the same.

*B. Data Labeling*

The main step in oracle learning is to use the oracle ANN to create a very large training set for the OTN to use. Fortunately the training set does not have to be pre-labeled since the OTN only needs the oracle ANN's outputs for a given input. Therefore the training set can consist of as many data points as there are available, including unlabeled points.

The key to the success of oracle learning is to obtain as much data as possible that ideally fit the distribution of the problem. There are several ways to approach this. In [6], Zeng and Martinez use the statistical distribution of the training set to create data. Another approach is to add random jitter to the training set, again following its distribution. The easiest way to fit the distribution is to have more unlabeled *real* data. In many problems, like ASR, there are more than enough unlabeled data. Other problems where there are plenty of unlabeled data include intelligent web document classifying, optical character recognition, and any other problem where gathering the data is far easier than labeling them. The oracle ANN can label as much of the data as necessary to train the OTN at the phoneme level. Therefore, the OTN has access to an arbitrary amount of ideally distributed training data.

In detail, this step must create a target vector $\boldsymbol{t}$ for each input vector $\boldsymbol{x}$ where each $t_i$ in $t_1 \ldots t_n$ ($n$ being the number of output nodes) is equal to the oracle ANN's activation of output $i$ given $\boldsymbol{x}$. Then, the final oracle learning data point contains both $\boldsymbol{x}$ and $\boldsymbol{t}$. In order to create the points, each available pattern $\boldsymbol{x}$ (labeled or not, but not including a small labeled subset for testing) is presented as an input to the oracle which then returns the output vector $\boldsymbol{t}$. The OTN's training set then consists of every $\boldsymbol{x}$ paired with its corresponding $\boldsymbol{t}$.

As an example, the following two vectors represent the target vectors for a given input. The first vector is a standard 0-1 encoded target where the 4th class is the correct one. The second is more representative of an ANN output vector (the oracle for the following experiments) where the outputs are between 0 and 1, and the 4th class is still the highest.

(1) <0,0,0,1,0>

(2) <0.27, 0.34, 0.45, 0.89, 0.29>

Now suppose the OTN outputs the following vector:

(3) <0.19, 0.43, 0.3, 0.77, 0.04>

The standard error would simply be the difference between the target vector in (1) and the output vector in (3) which is:

(4) <-0.19, -0.43, -0.3, 0.23, -0.04>.

Whereas the oracle-trained error would be the difference between the target vector in (2) and the output in (3):

(5) <0.08, -0.09, 0.15, 0.12, 0.25>

Notice the oracle-trained error in (5) is on average lower than the standard error in (4), and therefore the OTN learns a function that may be easier for standard back-propagation.

Once again, Zeng and Martinez found the use of vectored targets to give improved accuracy over using standard targets in [6].

*C. Oracle Learning*

For the final step, the OTN is trained using the data generated in step 2, making sure to utilize the targets exactly as presented in the target vector. The OTN must interpret each element of each target vector as the correct output activation for the output node it represents given the input paired with it, hence the ANN's learning algorithm may need to be modified depending on how it handles targets. For most ANNs, classification targets are encoded in binary with the correct class as 1 and all others as 0 and hence the error is generally computed as $\{0 \mid 1\} - o_i$ where $o_i$ represents the output of node $i$. With oracle learning, the error would instead be the $t_i - o_i$ where, as stated above, $t_i$ is the $i$th element of the target vector $\boldsymbol{t}$ paired with the input $\boldsymbol{x}$. The outputs of the OTN will approach the target vectors of the oracle ANN on each data point as training continues.

## III. EXPERIMENT

One of the most popular applications for smaller computing devices (i.e. hand held organizers, cellular phones, etc.) and other embedded devices is automated speech recognition (ASR). Since the interfaces are limited in smaller devices, being able to recognize speech allows the user to more efficiently enter data. Given the demand and usefulness of speech recognition in systems lacking in memory and processing power, there is a demand for simpler ASR engines capable of achieving acceptable accuracy. Hence the following experiments seek to reduce the complexity of our current ASR engine—or more specifically, the phoneme classifying ANN portion of the engine.

The following experiments use data from the unlabeled TI digit corpus [10] for testing the ability of the oracle ANN to create accurate phoneme level labels for the OTN. The corpus was partitioned into a training set of 15,322 utterances (3,000,000 phonemes), and a test set of 1000 utterances. A small subset of the training corpus consisting of around 40,000 phonemes was labeled at the phoneme level for

training the oracle ANN. The inputs are the first 13 mel cepstral coefficients and their derivatives in a 16 ms frame extracted from wav files every 10 ms (overlapping).

It is important to mention the fact that the final measure of accuracy is performed at the word and utterance levels, not the phoneme level. In general, word and sentence accuracies are more significant in speech recognition and do not always directly correlate with phoneme accuracy. It depends on the decoding technique and / or speech model used to build phonemes into words. In fact, in preliminary experiments, the standard trained networks always had slightly better phoneme accuracies than the OTNs (for any size).

Figure 1 diagrams the basics of the ASR engine used for the experiments. The mel cepstral coefficients are fed into the ANN and the ANN phoneme outputs are decoded into words. Both the oracle ANN and the OTN are used as the neural network recognizer part of the engine when determining word and utterance accuracy.

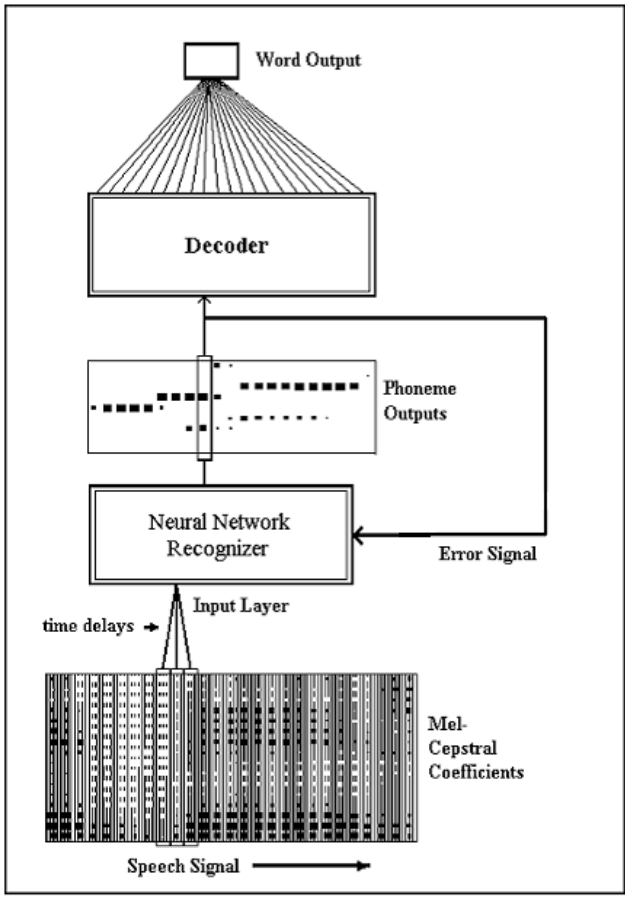


Figure 1: The basic ASR Engine

*A) Obtaining the Oracles*

The ASR engine's standard neural network recognizer is a 200 hidden node standard back-propagation-trained feed-forward network that has been tuned and optimized over time. In the following experiment, the ANN is trained directly on the phoneme labeled training data, storing the ANN weight configurations for future testing. Although the ANN most accurate on the test set (words and utterances) was chosen as the oracle ANN, any one of them was sufficient to validate oracle learning as long as the OTN achieves similar accuracy. We chose to use the most accurate ANN in order to create the most accurate OTN.

*B) Labeling the Data Set*

For the next step a large training set was created from the unlabeled data. The entire 15,000+ utterance training set was used to create a new training set consisting of the inputs from the old set combined with the target vectors from each oracle (one data set for each oracle), acquiring the target vectors as explained in B of section II (from the oracle ANN's outputs). In detail, oracle learning presents the oracle with an input pattern and then saves the activations of each output node for that input as a vector. The new OTN's training vector then consists of the original input and the new target vector.

*C) Oracle Learning*

Finally, the large OTN training set created in B is used to train an ANN half the size of the oracle (100 hidden nodes) using vectored targets instead of 0-1 targets according to the method described in section II part C. For a given training pattern, the error back-propagated was set to the difference between the oracle ANN's output node activation and the OTN's output or $t_i - o_i$ where $t_i$ is the oracle's output for class $i$ and $o_i$ is the output of the OTN net on class $i$.

To measure the effectiveness of oracle learning during the training phase, several metrics were used: the mean error with respect to the target vector, accuracy compared to the oracle ANN, and the top 100 OTN outputs compared the top 100 oracle ANN outputs. The general trend during training was for each of the metrics to improve, however, contrary to intuition, the best OTNs did not have the best values according to our metrics. It would be intuitive to believe the ANN with the least error with respect to the oracle would perform most like the oracle and hence have the best overall accuracy, but it did not. We hypothesize the reason was the phoneme-to-word decoding module did better with networks better arranging the ordering (from highest to lowest) of the output activation levels, regardless of the single highest output of the oracle ANN. The decoder considers more than just the top output, so where the next several outputs are ordered correctly, better word accuracy results. Therefore, even though one network may be more likely to have the same highest scoring phoneme as the oracle, the final ordering of the probabilities is better in a network with slightly a worse overall accuracy against the oracle.

A standard 100 hidden node network was also trained in order to compare it to the oracle learning 100 hidden node OTN (200 → 100).

After every oracle learning epoch, word and utterance accuracies were gathered and the respective OTN weights saved. The weights of the most accurate epoch were chosen as the best OTN of that particular oracle learning run.

## IV. RESULTS AND ANALYSIS

Table I reports the accuracy for each of the mentioned ANNs on the test set (the standard back-propagation-trained 200-hidden node ANN used as the oracle, the OTN (200 → 100) and the 100 hidden node standard net). Sentence accuracy refers to the percentage of times where the ASR system recognized the digits in an utterance correctly.

**TABLE I**
**ORACLE LEARNING ACCURACIES**

| Network configuration | Word % | Sentence % |
|---|---|---|
| 200 hidden nodes (standard, the oracle ANN) | 99.59 | 98.70 |
| OTN (200 → 100) | 99.56 | 98.60 |
| 100 hidden nodes (standard) | 99.41 | 98.10 |

As seen above, an OTN (200 → 100), having half as many hidden nodes than its oracle, achieves a comparable accuracy, 99.56% instead of 99.59%. The OTN (200 → 100)'s accuracy also demonstrates 25% less error than training a 100 hidden node net with the standard back-propagation approach (99.56% vs. 99.41%).

One reason for the improvement is that the OTN can train as long as necessary to over-fit on the oracle ANN's outputs using the large amount of unlabeled data and hence "sees" far more data points than the standard trained network which can only be trained with labeled data. Also the fact that the OTN (200 → 100) is learning a simpler function than the 0-1 encoding the standard 100-node network must learn may aid its improved accuracy.

## V. CONCLUSION AND FUTURE WORK

The results of the experiment support the theory that training a smaller ANN to approximate a larger ANN results in 1) a less complex network capable of accuracy comparable to its oracle, and 2) improved accuracy over standard training of smaller ANNs. An OTN with half the complexity of its oracle had significantly less error than the standard trained model, and achieved comparable accuracy to its oracle.

Future work in this area includes several more experiments. First, research will be done to determine how well even smaller ANNs perform when approximating both the original oracle and even approximating larger OTNs. It is important to determine the relation between the sizes of both the OTN and its oracle ANN. For example, does a 50 hidden node network yield better results approximating the original 200 hidden node oracle or an OTN (200 → 100)? Next, even more powerful oracles will be obtained (including mixture models, ensembles, etc.) to ascertain the robustness of using OTNs when presented with non-ANN oracles.

Preliminary results in the above areas indicate that the closer the complexity of the oracle ANN to the OTN, the better the OTN performs. For example, in one experiment, an OTN (100 → 50) achieved higher accuracies than an OTN (200 → 50). If this trend persists, the ideal size will be determined (number of hidden nodes) for an OTN to approximate even more complex oracles (mixture models, ensembles, etc.) to reveal how the complexity of an ANN relates to the complexity of non-ANN models.

Other research includes using the above complexity measures to develop a system for more accurately comparing complexity between different classifier models (i.e. ANN compared to mixture-of-gaussian ASR models). The system would be in terms of the number of hidden nodes needed to effectively approximate a given model and would be obtained by simply oracle-training ANNs of various sizes using the model being measured as the oracle. The main problem in this area would be handling the different inductive biases between the models.

The ASR engine used in the experiment uses a decoder that takes as much advantage of the order of the outputs as it does the single highest output. Therefore, in order to determine if oracle learning can be as effective in problems that do not require or lend themselves to decoding, further experiments will compare and contrast decoded and non-decoded problems to find the correlation.

## ACKNOWLEDGEMENTS

This research was funded in part by a grant from *fonix* Corp.

## VI. REFERENCES

[1] Y. Le Cun, J.S. Denker, and S.A. Solla, "Optimal brain damage ", in *Advances in Neural Information Processing Systems 2*, D.S. Touretzky, Ed., pp. 598--605. Morgan Kaufmann, San Mateo, CA, 1990.

[2] Waibel, A. (1989) "Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks" in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann.

[3] Y. Le Cun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. "Handwritten digit recognition with a back-propagation network." In David S. Touretzky, editor, *Neural Information Processing Systems*, volume 2, pages 396-404. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[4] P. Domingos, “Knowledge acquisition from examples vis multiple models”, in Proc. of the *Fourteenth International Conference on Machine Learning*, pp. 211-218, 1997.

[5] Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

[6] Zeng, Xinchuan and Tony R. Martinez, “Using a Neural Network to Approximate an Ensemble of Classifiers,” *Neural Processing Letters*, vol. 12, pp. 225-237, 2000.

[7] Breiman, L. (1996a), “Bagging Predictors”, *Machine Learning*, Vol. 24, No. 2, pp. 123-140.

[8] M. W. Craven and J. W. Shavlik, “Learning symbolic rules using artificial neural networks,” in Proceedings of the *10th International Conference on Machine Learning*, pp. 73-80, Amherst, MA. Kaufmann, 1993.

[9] M. W. Craven and J. W. Shavlik, “Extracting tree-structured representation from trained networks”, in D. S. Touretzky, M. C. Mozer and M. Hasselmo (ed.) *Advances in Neural Information Processing System 8*, pp. 24-30, MIT Press, 1996.

[10] R. Gary Leonard and George Doddington. (1993). TIDIGITS speech corpus, http://morph.lds.upenn.edu/Catalog/LDC93S10.html. Texas Instruments, Inc.