International Journal of Neural Systems, Vol. 19, No. 1 (2009) 1–9
© World Scientific Publishing Company

IMPROVING SUPERVISED LEARNING BY ADAPTING THE PROBLEM TO THE LEARNER

JOSHUA MENKE and TONY MARTINEZ
Computer Science Department, Brigham Young University,
3365 TMCB, Provo, UT, USA

While no supervised learning algorithm can do well over all functions, we show that it may be possible to adapt a given function to a given supervised learning algorithm so as to allow the learning algorithm to better classify the original function. Although this seems counterintuitive, adapting the problem to the learner may result in an equivalent function that is "easier" for the algorithm to learn. One method of adapting a problem to the learner is to relabel the targets given in the training data. The following presents two problem adaptation methods, SOL-CTR-E and SOL-CTR-P, variants of *Self-Oracle Learning with Confidence-based Target Relabeling* (SOL-CTR) as a proof of concept for problem adaptation. The SOL-CTR methods produce "easier" target functions for training *artificial neural networks* (ANNs). Applying SOL-CTR over 41 data sets consistently results in a statistically significant ($p < 0.05$) improvement in accuracy over 0/1 targets on data sets containing over 10,000 training examples.

Keywords: Supervised learning; objective function; relabeling.

1. Introduction

It is well known that no supervised learning algorithm does well over all functions,[18] however it may be possible to adapt a given function to better fit a given learning algorithm. Given a learner that is an instantiated model from a supervised learning algorithm, instead of only training the learner on the problem, we show that the problem can be "trained" on the learner simultaneously, in order to improve performance. Adapting the problem to the learner may result in an equivalent function that is "easier" for a given algorithm to learn. As a special case example, consider *rankprop*.[5] Caruana showed that given a standard classification function $f(x)$ with 0/1 targets, and a problem where the goal is learning to sort the patterns instead of directly modeling $f(x)$, there can exist a function $g(x)$ that models the sorting of the patterns by $f(x)$. Caruana showed that $g(x)$ can be "easier to learn" for backpropagation-trained *artificial neural networks* (ANNs) and was able to obtain higher accuracy training on $g(x)$ than training on $f(x)$. Rankprop is designed for single-output problems where ranking is appropriate (e.g. ranking patient priorities for admittance to the hospital). It would be desirable to develop a learning method that adapted any problem to its learner without having specific restraints like needing to sort the data in some fashion. We propose a more general approach which takes an arbitrary data set and modifies that data set to better fit the learning algorithm such that the learning algorithm attains higher classification accuracy on a test set taken from the original data set. It is a method for target relabeling that combines *self-oracle learning* (SOL), based on a training paradigm called *oracle learning*,[13] and ANN confidence measures. SOL is a proof of concept method to demonstrate the potential for adapting problems to the learner.

The main idea in oracle learning is that instead of training directly on a set of data, a learning model is trained to approximate a given *oracle's* behavior on a set of data. The oracle can be another learning model that has already been trained on the data, or it can be any given functional mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$

where n is the number of inputs to both the mapping and the *oracle-trained model* (OTM), and m is the number of outputs from both. The main difference with oracle learning is that the OTM trains on a training set whose targets have been relabeled by the oracle instead of training with the original training set labeling. Having an oracle to label data means that previously unlabeled data can also be used to augment the relabeled training set. The key to oracle learning's success is that it attempts to use a training set that fits the observed distribution of the given problem to accurately approximate the oracle on those sections of the input space that are most relevant in real-world situations. In Ref. 13 small ANNs are trained to approximate larger ANNs instead of being trained directly on the data. In addition, the smaller ANNs are trained on previously unlabeled data since the larger ANNs can serve as *oracle ANNs* to label data that did not originally have labels. Using oracle learning to reduce the size of these ANNs resulted in a 15% decrease in error over standard training and maintained a significant portion of the oracles' accuracy while being as small as 6% of the oracles' size. The *bestnets* algorithm[12] uses oracle learning to approximate multiple domain experts with a single ANN.

Although oracle learning allows for the use of unlabeled data to augment existing training sets, preliminary experiments for a previous work[13] resulted in improvements using the oracle-labeled targets instead of the original 0/1 encoding even when no unlabeled data was used. The higher accuracy suggests oracle learning may be creating an "easier function" for backpropagation to learn, but without requiring a specific meaning to the encoding as is the case with rankprop. The same principle is used in this paper, except that with the SOL methods presented, an ANN acts as its own oracle to relabel the standard training set. The set is labeled with the ANN's exact outputs on the data points as training progresses, instead of having a separate oracle model for relabeling. SOL is used to determine the difficulty level of each data point. If the ANN is struggling with certain data points, or has already learned certain points, the final targets can be adapted to reflect the current estimated difficulty of each point.

The problem with using an ANN to relabel the training set with its own outputs is that the ANN can be wrong in its predictions, and using its exact outputs as labels for the data can discard information about the true class of each data point. In order to preserve the correctness of the training set, the original 0/1 output targets are combined with the ANN's outputs using measures of the ANN's confidence in its own outputs. When the ANN is very confident, the labels are more likely to be similar to the ANN's own outputs. When the ANN is less confident, the labels will approach the original 0/1 encoding. Combining SOL with ANN confidence measures yields final targets that are customized for each data point based on the ANN's own measure of the data point's difficulty combined with the ANN's confidence in that measure. Applying *Self-Oracle Learning with Confidence-based Target Relabeling* (SOL-CTR) over 41 data sets consistently results in a statistically significant ($p < 0.05$) improvement in accuracy over 0/1 targets on the data sets containing over 10,000 training examples.

2. Background

In addition to *rankprop*, another recent work from Caruana *et al.*[4] also modifies the target function to yield improved results. Similar to previous work by Zeng and Martinez,[19] Caruana relabels synthetically created data using a large ensemble classifier. The difference here is that the SOL methods do not use synthetic data, and only use the original, base classifiers, instead of large ensembles.

Besides the aforementioned oracle learning and rankprop methods, another area related to SOL is semi-supervised learning, where the model being trained is used to label unlabeled data to improve training accuracy.[3,2] The basic approach, often known as semi-supervised learning or bootstrapping, is to train on given labeled data, classify a different set of unlabeled data, and then train using both the original and the relabeled data. This process is repeated multiple times until generalization accuracy stops increasing. In an oracle learning sense, the model trained is acting as its own oracle, similar to SOL. There are two main differences between semi-supervised learning and SOL. First, semi-supervised learning does not relabel the labeled training set, only the unlabeled data, whereas SOL relabels all the data. Second, semi-supervised learning uses the usual 0/1 encoding, whereas SOL seeks to replace the 0/1 labels and create a function that, like rankprop's, is "easier" for backpropagation-trained ANNs to learn.

Another area of research related to relabeling is Rimer's *classification-based* training algorithm, CB1.[16] The main idea behind the CB1 algorithm is to only backpropagate error when the training ANN misclassifies the current data point or does not have a large enough margin between the correct class and closest incorrect classes' output. Even then, the error is only backpropagated along the outputs that were too high or too low. This is related to SOL because it is another way to produce a potentially simpler and yet equivalent classification function for backpropagation to learn. The main difference with SOL is that it seeks to improve the targets themselves whereas the CB1 algorithm seeks to improve how the error with respect to the targets should be determined. SOL-CTR still backpropagates error on every output, although less for more confident outputs than others, whereas the CB1 algorithm will not backpropagate any error if the ANN is confident enough in its output.

Other areas that are less directly related to SOL, but still worth mentioning include using non $-0/1$ (or non $-1/+1$) targets, adaptive learning rate methods, and regularization methods like weight decay and pruning. It is common to suggest using targets other than those at the asymptotes of the transfer function (e.g., using 0.1/0.9 instead of 0/1 with a sigmoid) so that the targets can be reached through training and weights are not needlessly saturated. More formal methods[10] have also been suggested for choosing exactly where to place the targets given the transfer function. Although using targets other than 0/1 may be another way of creating an "easier" function, SOL-CTR takes this concept to an adaptive level, where the targets are customized for each output based on ANN performance rather than choosing a set of static, non-adaptive targets, that are used the same with every training data point.

Adaptive learning rate methods like *rprop*,[15] *quickprop*,[7] and *conjugate gradient methods*[17] do customize the amount of error backpropagated for a training set at each epoch in training, however the goal is generally faster convergence by taking larger steps along a predicted gradient rather than improving accuracy. SOL-CTR changes the error surface altogether to one that is hopefully easier for backpropagation to converge on, resulting in higher accuracy rather than faster convergence.

One reason SOL-CTR may work is because it leads to smaller magnitude weights, and is therefore less likely to overfit.[1] This can be compared to the weight decay[9] regularization method, where weights constantly shrink if they are not being updated. The difference is while weight decay is usually done the same on each weight, SOL-CTR will customize the affect on each weight. In addition, instead of constantly penalizing unused weights, SOL-CTR tries to only use the weights needed at a given point in training. Pruning,[14] another form a regularization, will remove unused weights altogether, whereas SOL-CTR will instead try and use the unused weights more efficiently.

3. Self-Oracle Learning with Confidence-Based Target Relabeling

3.1. *Self-oracle learning*

As mentioned in Sec. 2, SOL uses the ANN being trained as its own oracle in order to find better targets for the training data points. In its simplest form, SOL can be applied as shown in Fig. 1. The algorithm takes both a training and *hold-out* set. Here, hold-out set refers to a separate partition from the same data that is not used directly for training. The final accuracy should also be evaluated on a third set, separate from both the training and hold-out sets.

Note that the same random weights are used on each iteration of SOL. This is because SOL is

```
Procedure SOL(training set, hold-out set)
    while hold-out set accuracy increases do
      Initialize ANN to same random weights.
      while hold-out set accuracy increases do
        Train ANN one epoch on the training set
        with current labels.
      foreach data point in the training set do
        Obtain the trained ANN's output on the
        point.
        Relabel the point's targets with the
        ANN's current exact outputs.
```

Fig. 1. This gives the Brute-Force SOL algorithm which relabels the data after each training session using the previous training session's ANN.

designed to learn the correct targets for a given initial weight setting. This ensures that new targets are adapted to not only the structure of the ANN, but also its starting position. When used in this manner, SOL becomes a brute force search for better targets, based on what the ANN is outputting. The idea is that the outputs the ANN is actually able to produce better represent its capacity for fitting the given training data. Note that this can be used for concept (2-class) learning, multi-class problems, or even continuous valued functions by replacing in each case, the standard 0/1 or given continuous target, with the ANN's previous output on that data point. Applying this approach to a large, noisy OCR data set in preliminary experiments resulted in an ANN that was just as accurate as standard training, but represented a *simpler* function.

Simpler is defined in this case by the final magnitude of the ANN's weights. This measure is appropriate since the bias of backpropagation-trained ANNs is to move from simple to more complex as training progresses. This bias results from the fact that ANN weights are initialized to small, random values near 0. As the ANN trains, the weights move away from 0 to adapt to the data. A smaller average final magnitude weight implies a simpler function.[1] In the preliminary results, the self-oracle-trained ANN achieved equivalent accuracy to a 0/1 target trained ANN with 71% of the final weight magnitude, yielding a simpler function than the 0/1 targets. Since the resulting ANN is simpler, SOL may be creating a function that is "easier" for backpropagation to learn, resulting in a more efficient use of the weights. Intuitively, the ANN is acting as its own functional complexity reducer. At each stage, the target function's complexity has been reduced to one more accessible to the ANN.

The problem with using SOL as described here is that relabeling the targets exactly as output by the ANN itself means discarding relevant information about the true class of data points that are misclassified. It is possible to achieve higher accuracy by preserving the known class information in the final targets, but the question then becomes how much of the original 0/1 labels to use, and how much of the ANN's outputs should be used. The SOL-CTR approach weights each based on a heuristic used to measure the confidence of the ANN.

3.2. *ANN confidence measures*

One method for combining the outputs of the ANN with the original 0/1 labels is to weight each by the ANN's confidence. Thus the new target T for output j of data point i becomes:

$$T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j} \tag{1}$$

where $O_{i,j}$ is the value of the jth output node of the ANN given data point i, $C_{i,j}$ is the ANN's confidence that $O_{i,j}$ is correct, and $S_{i,j}$ is the original 0/1 encoding of the target outputs for data point i. The variable α is a meta-level trust value placed on the confidence measure C. If C is known to be exactly accurate, α can be set to 1. If there is some meta-level uncertainty about the parameter C, then α can be set to reflect that uncertainty. Therefore the output of each data point is trained using a target customized for that exact output and data point, based on the ANN's output and confidence in that output.

In theory, the quantity $C_{i,j}$ given above can not be measured directly since it will always be 0. This is because $C_{i,j}$ represents the evaluation of a continuous probability density function at a single point. Given a continuous density function, probabilities are measured over intervals, and here the interval and probability are both 0. Therefore, instead of trying to measure this quantity directly, a heuristic is used that measures the ANN's confidence in each class. The heuristic chosen for this paper is the *F-measure*. The F-measure for class k is determined as follows:

$$\textit{F-measure}_k = \frac{2 \times \textit{Recall}_k \times \textit{Precision}_k}{\textit{Recall}_k + \textit{Precision}_k} \tag{2}$$

where

$$\textit{Recall}_k = \frac{TruePositives_k}{TruePositives_k + FalseNegatives_k} \tag{3}$$

and

$$\textit{Precision}_k = \frac{TruePositives_k}{TruePositives_k + FalsePositives_k}. \tag{4}$$

Recall is a measure of how often a data point from a given class is recognized as being from that class, whereas *precision* is a measure of how often a data point recognized as being from a given class actually belongs to that class. The F-measure combines both recall and precision in a way that requires them both to be high and similar. Therefore, the new target is chosen based on the ANN's confidence in its outputs for a given class k as calculated by using the F-measure. The F-measure is attractive as a confidence measure because it takes into account both recall and precision, which are valid measures of an ANN's performance on a given class.

It was found in practice that when the ANN misclassifies a given example, setting the confidence $C_{i,j}$ to 0 despite the F-measure results in improved results. This suggests that the 0/1 targets are still best when the ANN is struggling to learn a data point. This is equivalent to setting $\alpha = 0$ when the ANN misclassifies an example. In addition, better results were obtained by setting $\alpha = 0.5$ when the ANN correctly classifies an example. This is most likely because the F-measure is still only a heuristic, and therefore can not be trusted completely as a measure of the probability that the ANN's exact output is correct. This results the in new target values lying between 0.5 and 1.0 for the target class, and 0.0 and 0.5 for the non-target classes. If α were always set to 1, then the targets could vary anywhere between 0 and 1 for both situations. Note that as long as the correct class had the highest output, it does not matter how high that output is. For example, if the ANN output corresponding to the correct class outputs 0.3 and that is still the highest output, the ANN has classified the example correctly. Combining these settings results in simplifying the targets when the ANN classifies correctly in order to leave more adaptive capacity in the ANN for learning the more difficult data points. The function presented becomes more complex than the pure SOL function, but uses the weights of the ANN more efficiently than pure 0/1 relabeling. The mixing of pure SOL and 0/1 targets is what leads to SOL-CTR's success. It is able to combine the merits of both to more effectively train ANNs.

Using the F-measure with SOL yields a learning algorithm that relabels targets for training ANNs with backpropagation. Unfortunately, as is, SOL requires the ANN to be entirely retrained each time the data is relabeled. Figure 2 shows a more efficient approach, *SOL-CTR by Epoch* (SOL-CTR-E). It was designed to train the ANN only once by updating the target labels after every epoch instead of after completely training the ANN.

An even more efficient method is to relabel the targets after each data point. This method is called *SOL-CTR by Pattern* (SOL-CTR-P) and can be seen in Fig. 3.

```
Procedure SOL-CTR-E(training set, hold-out set, α)
  $S_{i,j} = T_{i,j} =$
  0/1 targets for data point i and output j
  // Initialize true positives (TP)
     false positives (FP) and false
     negatives (FN)
  $TP = FP = FN = \{0\}$
  while hold-out set accuracy increases do
    Train ANN for one epoch using $T$.
    foreach data point i in the training set do
      // Store the ANN outputs for later
         relabeling
      foreach ANN output j do
        $O_{i,j}$ = ANN's $j$th output given $i$
      // Store the ANN classification
      $class = argmax(O_i)$
      // Get the true classification
      $k = argmax(T_i)$
      if $class = k$ then $TP_k = TP_k + 1$
      else
        // Increment the False positives
           for the incorrectly chosen
           class
        $FP_{class} = FP_{class} + 1$
        // Increment the False Negatives
           for what should have been the
           chosen class
        $FN_k = FN_k + 1$
    foreach class of the problem do
      $Recall_{class} = \frac{TP_{class}}{TP_{class}+FN_{class}}$
      $Precision_{class} = \frac{TP_{class}}{TP_{class}+FP_{class}}$
      $C_{i,j} = F\text{-}measure_{class} = \frac{2 \times Recall_{class} \times Precision_{class}}{Recall_{class}+Precision_{class}}$
    foreach data point i in the training set do
      foreach target j of point i do
        $T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j}$
```

Fig. 2. This presents SOL-CTR-E, which performs SOL-CTR by relabeling the entire training set after each epoch, but not within an epoch.

Notice that the only difference between SOL-CTR-P and SOL-CTR-E is that SOL-CTR-P adapts the targets *within* the training epoch whereas

```
Procedure SOL-CTR-P(training set,hold-out set,$\alpha$)
    $S_{i,j} = T_{i,j} =$
    0/1 targets for data point i and output j
    // Initialize true positives (TP)
       false positives (FP) and false
       negatives (FN)
    $TP = FP = FN = \{0\}$
    while hold-out set accuracy increases do
      foreach data point i in the training set do
        // Store the ANN outputs for later
           relabeling
        foreach ANN output j do
          $O_{i,j} =$ ANN's $j$th output given $i$
        // Store the ANN classification
        $class = argmax(O_i)$
        // Get the true classification
        $k = argmax(T_i)$
        if $class = k$ then $TP_k = TP_k + 1$
        else
          // Increment the false positives
             for the incorrectly chosen
             class
          $FP_{class} = FP_{class} + 1$
          // Increment the false negatives
             for what should have been the
             chosen class
          $FN_k = FN_k + 1$
        foreach class of the problem do
          $Recall_{class} = \frac{TP_{class}}{TP_{class}+FN_{class}}$
          $Precision_{class} = \frac{TP_{class}}{TP_{class}+FP_{class}}$
          $C_{i,j} = F\text{-}measure_{class} = \frac{2\times Recall_{class}\times Precision_{class}}{Recall_{class}+Precision_{class}}$
        foreach target j of point i do
          $T_{i,j} = \alpha C_{i,j} O_{i,j} + (1 - \alpha C_{i,j}) S_{i,j}$
        Backpropagate error using $T_i$ as the
        targets.
```

Fig. 3. This gives SOL-CTR-P, which relabels each data point using the ANN's current state. This happens within an epoch in an on-line learning fashion.

SOL-CTR-E only relabels the targets after the training epoch. In the last line of SOL-CTR-P, the newly generated targets are immediately used to train on the current pattern, whereas SOL-CTR-E will not apply the new targets until it trains for another epoch. Because the F-measure information is less accurate until the ANN's initial class accuracy trends emerge, it is better in practice to train an epoch before relabeling the data when using SOL-CTR-P. In the preliminary experiments we used to guide our research, SOL-CTR-E and SOL-CTR-P resulted in improved accuracy and average final weight magnitudes 87% lower than training with standard 0/1 targets.

3.3. *Complexity*

The complexity of each of the SOL methods is the same as standard error-backpropagation, with the following differences. Brute-force SOL takes k-times longer to train, where k is the number of full ANN trainings before hold-out set accuracy stops increasing. The SOL-CTR methods are more efficient, however, and do not suffer from this penalty. They only add a small constant amount of additional time needed to calculate the F measure. However this is smaller than the time required to update the weights of the ANN given one training pattern, and therefore does not significantly increase the training time. In terms of space requirements, both SOL-CTR methods add a space complexity equal to $3C$ where C is the number of classes or possible outputs for the ANN in a continuous-valued problem. This is used to track the true positives, false positives, and false negatives. Note that the precision, recall, f-measure, and C matrix quantities do not need to be stored throughout training, but can be calculated on the fly given the true positives, false positives, and false negatives.

4. Methods

In order to test the effectiveness of SOL-CTR, both SOL-CTR-E and SOL-CTR-P are compared to using standard 0/1 targets on 37 UCI Machine Learning Database (MLDB) problems (see Table 2, two versions of a large, proprietary, real-world OCR data set consisting of 500,000 OCR examples with 83 characters, and one large automated-speech recognition (ASR) data set consisting of over 800,000 phonemes taken from the TIDIGITS speech corpus with 199 single, bi- and tri-phoneme classes. The phoneme level labels for the ASR data set were created using a separate ANN along with known word-labels to segment and force-align the sound input. The difference between the first and second version of the OCR

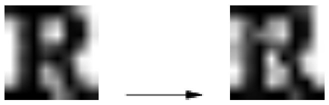

Fig. 4. This shows an R-character from the OCR data set before and after adding random noise.

data set is that noise is added to the images in the second set to increase the difficulty of the problem. Each character in the OCR set consists of an $8x8$ grid of grayscale pixel values. To obtain the noisy set, a different amount of noise was chosen for each pixel by randomly generating a number from a standard normal distribution, cubing it, dividing it by 3, and then adding it to the pixel's original value. Pixel values were then clipped to $[0, 1]$. This is repeated for each pixel on each character creating the effect seen in Figure 4. The OCR sets are broken into a 200,000 example training set, a 100,000 example hold-out set, and a 200,000 example test set. The ASR data set was broken into a 500,000 phoneme training set, a 200,000 phoneme hold-out set, and a 200,000 test set. To verify the statistical significance of the results, a 10 pair permutation test is used[11] with each data set. For the MLDB problems, stratified 10-fold cross validation[8] is used, and therefore each pair is a different fold with a slightly different training set and a different test set in addition to different initial weights. For the ASR and OCR sets, a different initial weight setting is used for each pair. Different training set and test set partitions are not used with the larger sets since the size of the data sets is large enough to capture the distribution adequately.[6] Each pair consists of one ANN trained using a relabeling method, and one ANN trained using standard 0/1 targets — both ANNs use the same number of hidden nodes: 128, learning rate: 0.01, and have the exact same random initial weight settings. Each ANN is also trained using error-backpropagation in an online fashion.

5. Results and Analysis

Table 1 gives the results of the experiments. The first column gives the relabeling method being compared to training with the standard 0/1 targets and the following columns give the results of comparing the relabeling methods on 5 data sets. The *letter* data set from the MLDB has its results shown separately because its size (20,000 data points) is significantly larger than any other data set in the MLDB. The

Table 1. % Differences in Mean Accuracy. *: p-value below 0.05, **: below 0.01.

Data/Method	SOL-CTR-E	SOL-CTR-P
MLDB	−0.0400	−0.0200
Letter	0.3000*	0.3100*
OCR	0.0520**	0.0560**
Noisy OCR	0.1872*	0.2002**
ASR	0.3140**	0.2630**

Table 2. Detailed results. The results in bold show the highest accuracy. A (*) indicates the method was not statistically worse than the other. SOL-CTR is preferred for the larger sets: ASR, OCR, NOISY-OCR, and the UCI letter set. On the other MLDB sets, the SOL-CTR method is never worse than standard.

Dataset	Standard	SOL-CTR-P
letter	0.9520	0.9551
ASR	0.8632	**0.8658**
OCR	0.9708	**0.9713**
NOISY-OCR	0.8688	**0.8709**
anneal	**0.7707**	0.764*
anneal.ORIG	**0.7718**	0.7684*
audiology	**0.7536**	**0.7536**
autos	**0.4625**	0.4313*
balance-scale	**0.8704**	0.8688*
breast-cancer	**0.7174**	0.7069*
breast-w	**0.9707**	0.9692*
colic	**0.8667**	**0.8667**
colic.ORIG	**0.6631**	**0.6631**
credit-a	0.6756*	**0.6832**
credit-g	**0.7**	**0.7**
diabetes	**0.7629**	0.759*
glass	0.5879*	**0.6026**
heart-c	0.6038*	**0.6139**
heart-statlog	0.8296*	**0.8370**
hepatitis	**0.8208**	0.8083*
ionosphere	0.8721*	**0.8749**
iris	**0.9333**	0.92*
kr-vs-kp	**0.9897**	0.9887*
lymph	**0.7786**	0.7724*
mushroom	0.9998*	**0.9999**
primary-tumor	0.466*	**0.4840**
segment	0.9628*	**0.9636**
sonar	0.7743*	**0.8124**
soybean	0.9283*	**0.9312**
splice	0.9533*	**0.9558**
vehicle	0.4915*	**0.4927**
vote	0.9517*	**0.9563**
vowel	**0.0909**	**0.0909**
waveform-5000	**0.8612**	0.8598*
zoo	**0.9009**	0.8718*

numbers given are the mean % differences in accuracy between the relabeling method and using standard 0/1 targets across the 10 pairs. The $*$'s give a measure of the statistical significance of the differences in accuracy. A single $*$ means the resulting p-value from the experiment was below 0.05 whereas two $*$'s mean the resulting p-value was below 0.01. In addition to the results summary table, Table 2 gives a detailed report compared to SOL-CTR-P only, since the results of SOL-CTR-E are not significantly different, and SOL-CTR-P is more efficient. The results show that on the smaller data sets in the MLDB, SOL-CTR gives no significant difference in accuracy. However, on the letter data set, and on the other 3 large data sets, both relabeling methods consistently yield statistically significant improvements in accuracy over using standard 0/1 targets. This suggests that SOL-CTR can successfully create an improved training function for backpropagation-trained ANNs, but perhaps only when enough data is available to effectively model the confidence of the ANNs. Although the average improvement on the large sets is modest at 0.05–0.3%, it is consistent and can be attributed to the relabeling. Further research improving the confidence measure can result in increased improvements because a more accurately modeled confidence measure means the targets can be more effectively chosen for training. If the targets are more effectively chosen, the resulting function can be even "easier" for backpropagation-trained ANNs to learn, and therefore resulting in even better accuracy.

6. Conclusions and Future Work

The main contribution of the experiment is that it serves as a proof of concept for adapting the problem to the learner. Instead of only adapting the learner to the problem, supervised learning can interactively use the feedback of a given learner to improve the problem's representation. In this manner, a given problem can be made to better fit the inductive bias of a given learner, and therefore broaden the set of problems over which a given learning algorithm performs well. For future work we will investigate methods for learning improved targets that do not require a confidence metric. We will also develop problem adaption techniques for learning algorithms besides backpropagation-trained ANNs. One way to learn targets without using a confidence measure is to iteratively refine them throughout the training process. Training would update both the weights and the targets simultaneously, attempting to minimize classification error as the objective.

Another possible area for improvement within the SOL-CTR algorithms themselves is in the choosing of the parameter α. For this work, it was tuned by hand in preliminary experiments. Future work will try brute force binary searches on hold-out data, followed with investigating more formal approaches.

References

1. P. L. Bartlett, For valid generalization the size of the weights is more important than the size of the network, in M. C. Mozer, M. I. Jordan and T. Petsche (eds.), *Advances in Neural Information Processing Systems*, Vol. 9, The MIT Press (1997), pp. 134.
2. K. P. Bennett, A. Demiriz and R. Maclin, Exploiting unlabeled data in ensemble methods, in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press (2002), pp. 289–296.
3. A. Blum and T. Mitchell, Combining labeled and unlabeled data with co-training, in *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers* (1998), pp. 92–100.
4. C. Buciluă, R. Caruana and A. Niculescu-Mizil, Model compression, in *KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, USA, 2006. ACM (2006), pp. 535–541.
5. R. Caruana, S. Baluja and T. Mitchell, Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation, in D. S. Touretzky, M. C. Mozer and M. E. Hasselmo (eds.), *Advances in Neural Information Processing Systems*, The MIT Press (Cambridge, MA) (1996), pp. 959–965.
6. T. G. Dietterich, Approximate statistical test for comparing supervised classification learning algorithms, *Neural Computation* **10**(7) (1998) 1895–1923.
7. S. Fahlman, Faster learning variations on back-propagation: An empirical study, in D Touretzky, G. Hinton and T. Sejnowski (eds.), *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, San Mateo (1989), pp. 38–51.
8. R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection (1995).
9. A. Krogh and J. A. Hertz, A simple weight decay can improve generalization, in J. E. Moody, S. J. Hanson and R. P. Lippmann (eds.), *Advances in Neural Information Processing Systems*, Vol. 4, Morgan Kaufmann Publishers, Inc. (1992), pp. 950–957.

10. Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, Effiicient backprop, in G. B. Orr and K.-R. Müller (eds.), *Neural Networks: Tricks of the Trade*, Vol. 1524 of *Lecture Notes in Computer Science*, Springer (1996), pp. 9–50.
11. J. Menke and T. R. Martinez, Using permutations instead of student's t distribution for p-values in paired-difference algorithm comparisons, in *Proceedings of the 2004 IEEE Joint Conference on Neural Networks IJCNN'04* (2004).
12. J. Menke and T. R. Martinez, Domain expert approximation through oracle learning, in *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN 2005)* (2005).
13. J. E. Menke and T. R. Martinez, Artificial neural network reduction through oracle learning, To appear in *Intelligent Data Analysis* **13**(1) (2009).
14. R. Reed, Pruning algorithms — a survey, *IEEE Transactions on Neural Networks* **4**(5) (1993) 740–747.
15. M. Riedmiller and H. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, in *Proc. of the IEEE Intl. Conf. on Neural Networks*, San Francisco, CA (1993), pp. 586–591.
16. M. E. Rimer and T. R. Martinez, Classification-based objective functions, *Machine Learning Journal* **63**(2) (2006) 183–205.
17. J. R. Shewchuk, An introduction to the conjugate gradient method without the agonizing pain, Technical report. Carnegie Mellon University, Pittsburgh, PA, USA (1994).
18. D. H. Wolpert and W. G. Macready, No free lunch theorems for search, Technical Report SFI-TR-95-02-010, The Santa Fe Institute, Santa Fe, NM (1995).
19. X. Zeng and T. Martinez, Using a neural networks to approximate an ensemble of classifiers. *Neural Processing Letters* **12**(3) (2000) 225–237.