Query-Based Learning Applied to Partially Trained Multilayer Perceptrons
Jenq-Neng Hwang, Jai J. Choi, Seho Oh, and Robert J. Marks II

Abstract—In many machine learning applications, the source of the training data can be modeled as an oracle. An oracle has the ability, when presented with an example (query), to give a correct classification. An efficient query learning procedure is to provide the good training data to the oracle at low cost. This paper presents a novel approach for query-based neural network learning. Consider a layered perceptron partially trained for binary classification. The single output neuron is trained to be either a 0 or a 1. A test decision is made by the query-based learning algorithm, say, $\hat{I}$. The set of inputs that produce an output of $\hat{I}$ forms the classification boundary. We adopted an inversion algorithm for the neural network that allows generation of this boundary. In addition, for each boundary point, we can generate the classification gradient. The gradient provides a useful measure of the steepness of the multidimensional decision surfaces. Using the boundary point and gradient information, conjugate input pairs are generated and presented to the oracle for proper classification. These new data are used to further refine the classification boundary, thereby increasing the classification accuracy. The result can be a significant reduction in the training set cardinality in comparison with, for example, randomly generated data points. An application example to power system security assessment is given.

I. INTRODUCTION

In many classification machine learning applications, the source of the training data can be modeled as an oracle. An oracle has the ability, when queried with an example, to give a correct classification. A cost, which can be very expensive (e.g., a supercomputer emulator), is typically associated with this query. The study of queries in classifier training paradigms is, therefore, a study of the manner by which oracles can provide good classifier training data at low cost.

Query-based learning requires asking a partially trained classifier to respond to the question, “What don’t you yet understand?” The response of the query is then taken as the oracle. The oracle, for a price, will respond with the correct classification for a given data point. Examples of oracles include computationally intensive simulators, costly experimentation, or a human expert. The properly classified points from the oracle are then introduced as additional training data for the classifier. The use of queries through such a systematic training data generation mechanism can be viewed as interactive learning. On the other hand, only the use of available (or randomly generated) data, is passive learning. In certain problems, when applied properly, queries can significantly increase the resulting classification accuracy with a small amount of additional complexity introduced [1].

In this paper, we consider the use of queries in the training of a partially trained multilayer perceptron. We propose to use the inversion algorithm, which allows generation of the network input (or inputs) that can produce any specified output vector. For binary classifiers, inversion of a network midway between the two classifications (i.e., 0 and 1) results in a classification boundary. This boundary is the locus of input vectors that, with respect to the neural network’s representation of the training data, is highly confusing. In addition, for each boundary point, we can generate the classification gradient. The gradient provides a useful measure of the steepness of the multidimensional decision profile. Using the boundary point and gradient information, conjugate input pairs are generated, which are supposed to carry the most important information (maximum confusion) in locating the correct decision boundaries [2], [3]. These points of confusion are presented to the oracle for proper classification. The choice of whether or not to use an oracle is dependent on the degree of training and classification complexity. Our approach to query-based learning works best when the network’s performance is being fine tuned.

The organization of this paper is as follows. In Section II, the backpropagation learning and network inversion algorithms are briefly discussed. Section III introduces the boundary search procedure from network inversion and the recursive formula for gradient computation. A simple toy problem for binary classification based on this query learning is discussed in Section IV. Finally, we apply this query learning to a power system security assessment problem in Section V.

II. LEARNING AND INVERSION OF A MULTILAYER PERCEPTRON

The forward system dynamics in the retrieving phase of an $L$-layer perceptron can be described by the following iterative equations (for $1 \leq l \leq N$, $0 \leq l \leq L - 1$):

$$u_l(l + 1) = \sum_{j=0}^{N_l} w_{ij}(l + 1) a_j(l) + \theta_j(l + 1)$$

$$a_j(l + 1) = f(u_j(l + 1))$$

where $x_j(l)(u_j(l))$ denotes the activation value (net input) of the $j$th neuron at the $l$th layer, $\theta_j(l)$ (or $w_{ij}(l)$) denotes the bias of the $j$th neuron at the $l$th layer; $w_{ij}(l)$ denotes the weight value linked between the $i$th neuron at the $l$th layer and the $j$th neuron at the $(l + 1)$th layer; and $f$ is the nonlinear activation function.

A. Backpropagation Network Learning

The learning phase of a multilayer perceptron uses the backpropagation learning rule, an iterative gradient descent algorithm designed to minimize the mean squared error $E$ between
the desired target vector \( \{ t_i \} \) and the actual output vector \( \{ a_i(L) \} \), (4), [5]:

\[
\begin{align*}
\delta_i(l) &= w_i(l) - \eta \frac{\partial E}{\partial w_i(l)} \\
&= w_i(l) - \eta \frac{\partial E}{\partial a_i(l)} \frac{\partial a_i(l)}{\partial w_i(l)} \\
&= w_i(l) - \eta \delta_i(l) \frac{\partial a_i(l)}{\partial w_i(l)}
\end{align*}
\]

(2)

where

\[
E = E(\{ w_i(l) \}, \{ a_i(l) \}) = \frac{1}{2} \sum_{i=1}^{N_i} (t_i - a_i(L))^2
\]

(3)

and the backpropagated error signal \( \delta_i(l) \) can be recursively calculated (see Fig. 1):

\[
\begin{align*}
\delta_i(l) &= \frac{\partial E}{\partial a_i(l)} \\
&= \sum_{j=l+1}^{N_o} \frac{\partial E}{\partial a_j(l+1)} \frac{\partial a_j(l+1)}{\partial a_i(l)} \\
&= \delta_j(l+1) \frac{\partial a_j(l+1)}{\partial a_i(l)} \\
&= \delta_j(l+1) \frac{\partial a_i(l)}{\partial a_i(l)} \\
&= \delta_j(l+1) \frac{\partial a_i(l)}{\partial a_i(l)}
\end{align*}
\]

(4)

with the initialization error signal \( \delta_i(L) = \frac{\partial E}{\partial a_i(L)} = -(t_i - a_i(L)) \).

B. Network Inversion

The inversion of a network will generate the input vector \( \{ a_i(0) \} \) that can produce a desired output vector. By taking advantage of the duality between the weights and the input activation values in minimizing the mean squared error \( E \) [see (3)], the iterative gradient descent algorithm can again be applied to obtain the desired input vector [6]:

\[
a_i(0) = a_i(0) - \eta \frac{\partial E}{\partial a_i(0)} = a_i(0) - \eta \delta_i(0).
\]

(5)

The idea is similar to the backpropagation algorithm, where the error signals are propagated back to tell the weights the manner in which to change in order to decrease the output error. The inversion algorithm backpropagates the error signals to the input layer to update the activation values of input units so that the output error is decreased (see Fig. 1). In order to avoid the input activation values \( \{ a_i(0) \} \) from growing without limits, a small modification of the updating rule was usually made:

\[
\begin{align*}
\eta_i(0) &= u_i(0) - \eta \frac{\partial E}{\partial u_i(0)} \\
&= u_i(0) - \eta \frac{\partial E}{\partial a_i(0)} \frac{\partial a_i(0)}{\partial u_i(0)} \\
&= u_i(0) - \eta \delta_i(0) \frac{\partial a_i(0)}{\partial u_i(0)}
\end{align*}
\]

(6)

where \( u_i(0) = f^{-1}(a_i(0)) \) is a "pseudo" net input created to allow flexible gradient descent search without limiting the dynamic ranges (e.g., usually we assume \( 0 \leq u_i(0) \leq 1 \)).
Fifty randomly selected two-dimensional training data are selected for octagonal region classification. A perspective plot of the octagonal region classification profile.

Fig. 2. (a) Fifty randomly selected two-dimensional training data are selected for octagonal region classification. (b) A perspective plot of the octagonal region classification profile.

B. Gradient Computation for Steepness of Classification Profile

After a neural network is trained, the parametric mapping relationship between the input and output is established through the weights. For each point in the input space, we can compute the gradient \( \phi_i(0) \) of each output neuron (e.g., the \( k \)th) with respect to each input neuron (e.g., the \( j \)th). At the boundary, this gradient is a measure of the steepness there. This gradient [8], [9],

\[
\phi_i(0) = \frac{\partial \phi_i(L)}{\partial \phi_i(0)}
\]  

(7)

can be recursively computed (based on a simple chain rule) from

\[
\phi_i(l) = \sum_{m=1}^{N_i} \phi_m(l+1) f'(w_m(l+1)) \phi_m(l+1),
\]

\[1 \leq l \leq L - 1\]

(8)

where the initial values are given as \( \phi_i(0) = 1 \) if \( i = j \); 0 otherwise.

Fig. 3(a) shows the magnitude of the gradients of 30 points from the 0.5 contour of Fig. 3(a), i.e.,

\[
|\text{gradient}| = \sqrt{\sum_{j} \phi_j(0)^2}.
\]

IV. CONJUGATE PAIRS FOR REFINING THE BOUNDARY

For each inverted boundary point, a conjugate training data pair based on the magnitude of gradient can be created [see Fig. 6(a)]. More specifically, two points lying on opposite sides of
Fig. 4. (a) A four-class classification problem consists of three rectangular regions. (b) After 10,000 iterations of the training, 100 classification boundary points were created for each rectangle.

Fig. 5. The magnitude of the gradients of 30 points from the 0.5 contour of Fig. 3(a).

Fig. 6. (a) For each inverted boundary point, a conjugate training data pair based on the magnitude of gradient were created. (b) Conjugate pairs which have different classifications and the boundary points whose conjugate pairs are classified in the same class.

the line passing through the boundary point and perpendicular to the boundary surface are located with their distances to the corresponding boundary point equal to \(1/|\text{gradient}|\). This idea is very similar in concept to the use of close-opposed pairs in locally trained piecewise linear classifiers [2]. This perpendicular line can be parametrically represented as:

\[
\frac{x_1 - z_1}{\partial_1(0)} = \frac{x_2 - z_2}{\partial_2(0)} = \cdots = \frac{x_n - z_n}{\partial_n(0)}
\]

Use of only the boundary points for additional training information typically leads to a biased emphasis of one side of boundary. We therefore adopt the two-sided conjugate pairs for boundary fine-tuning. The motive behind the selection of the conjugate pairs stems from our desire to increase the boundary steepness between two distinct classes by narrowing the regions of ambiguity. The oracle will provide true classification (1 or 0) for the newly generated conjugate pair query points. If all three points (the boundary point as well as the conjugate pair) fall into the same class, we neglect the conjugate pair and keep only the boundary point as part of the new training data. Otherwise, we choose all three points to be part of the new training
data. To illustrate, we continue with the octagonal example, a set of 137 training data were newly generated and used to retrain the network along with the originally randomly selected 50 training data. Fig. 6(b) shows the created conjugate pairs which have different classifications and the boundary points whose conjugate pairs are classified in the same class.

Fig. 7(a) shows the dramatically improved result after retraining based on the 187 (= 50 + 137) data points. The result of query learning is strikingly better in comparison with conventional learning using 500 points of randomly selected training data [see Fig. 7(b)]. To get the best performance, the above query-based procedure can be repetitively applied in the training so that the classification accuracy can be gradually improved.

V. APPLICATION TO POWER SYSTEM SECURITY PROBLEM

One of the main aspects of power system security is static or steady-state security. This is defined as the ability of the power system, after a disturbance such as a line break or other rapid load change, to reach a safe state that does not violate any operating constraints [10].

Defining multidimensional static-security regions for even a small power network is a computationally demanding task. It involves the solution of a nonlinear programming problem with a large number of variables and an equally large number of limit constraints which define the feasible region of operation [11]. In addition, the amount of memory required to store the security status under each probable network configuration is equally prohibitive. This leads us to the solution by artificial neural networks [12].

A three-layer perceptron consists of four inputs, two hidden layers (with 20 and 10 units separately), and one output is used to train a four-dimensional power system security assessment problem, where a four-dimensional input vector (representing the voltages of the power lines) is tested to determine whether the power plant is in a "secure" status or not.

To illustrate how well the query-based learning identifies the true boundary, a randomly selected two-dimensional slice of the four-dimensional power security assessment problem is used for graphical presentation. Fig. 8(a) shows a two-dimensional slice of the true boundary and the resulting representation boundary trained with 5000 randomly selected training data. On the other
query clarifies this confusion. The clarification is incorporated into the training process to further refine the performance of the neural network. The empirical result of our query-learning examples illustrates that, in certain cases, a much more favorable performance results as compared with that of conventional learning based on purely randomly selected training data. This can be true at even a significantly higher training data set cardinality.

ACKNOWLEDGMENT

Valuable discussions with Professors M. A. El-Sharkawi, L. Atlas, and R. Ladner at the University of Washington are gratefully acknowledged.

REFERENCES