International Journal of Engineering Research & Management Technology
ISSN: 2348-4039
May-2019 Volume 6, Issue-3
Email: editor@ijermt.org www.ijermt.org

STUDY ON PERCEPTRONS IN ARTIFICIAL NEURAL NETWORKS AND SEMANTIC NETWORK

Karan Siwach
Department of Computer Science and Engineering
ABSS Institute of Technology,
Meerut, U.P.,

ABSTRACT
Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions. ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones. Artificial neural networks (ANNs) provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions by perceptrons. Perceptrons algorithms such as back propagation gradient descent to tune network parameters to best fit a training set of input-output pairs. ANN learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies with significant perceptrons.
KEYWORDS: Neural Network, Perceptrons, Training Rule, Delta and Gradient Rule, Hypothesis space, Semantic Nets

INTRODUCTION
Perceptrons are one kind of artificial neural network that based on a unit. A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise. More precisely, given inputs x_1 through x_n, the output $o(x_1,\ldots\ldots,x_n)$ computed by the perceptron is

$$o(x_1,\ldots,x_n) = \begin{cases} 1 \ if \ Xn > 1 \\ -1 \ otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots(1.1)$$

Where $X_n = w_0 + w_1x_1 + w_2x_2 + \ldots\ldots + w_nx_n$

Here each w_i is a real-value weight, that determines the contribution of input x_i to the perceptron output. The quantity $(-w_0)$ is a threshold that the weighted combination of inputs $w_0 + w_1x_1 + w_2x_2 + \ldots\ldots + w_nx_n$ must surpass in order for the perceptron to output a 1.

Learning a perceptron involves choosing values for the weights $w_0,\ldots\ldots,w_n$ .Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors.

REPRESENTATION OF PERCEPTRONS
The perceptron representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points). The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side. The equation for this decision hyperplane is $w_i.x_i = 0$. Some sets of positive and negative examples cannot be separated by any hyperpane. Those that can be separated are called linearly separable sets of example.

A single perceptron can be used to represent many Boolean functions. For example, if we assume Boolean values of 1 (true) and -1 (false), then one way to use a two-input perceptron to implement the AND function is to set the weights $w_0 = -.8$ and $w_1 = w_2 = .5$. This perceptron can be made to represent the OR function instead by altering the threshold to $w_0 = -.3$. AND and OR can be viewed as special cases of m-of-n functions where at least m of the n inputs to the perceptron must be true. The OR function corresponds m = 1 and the AND function to m = n. Any m-of-n function is easily represented using a perceptron by setting all input weights to the same value (e.g., 0.5) and then setting the threshold w_0 accordingly.

Perceptrons can represent all of the primitive boolean functions AND, OR, NAND, and NOR. Some boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if $x_1 \neq x_2$. The ability of perceptrons to represent AND, OR, NAND, and NOR is important because every boolean function can be represented by some network of interconnected units based on these primitives. In fact, every Boolean function can be represented by some network of perceptrons only two levels deep, in which the inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage. One way is to represent the boolean function in disjunctive normal form (i.e., as the disjunction (OR) of a set of conjunctions (ANDs) of the inputs and their negations). Note that the input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight. Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

THE PERCEPTRON TRAINING RULES

Represent the learning networks of many interconnected units by understanding how to learn the weights for a single perceptron. Here the precise learning problem is to determine a weight vector that causes the perceptron to produce the correct +1 or -1 output for each of the given training examples. There are various Algorithms for solving this learning problem. Here we consider two: the perceptron rule and the delta rule. These two algorithms are guaranteed to converge to somewhat different acceptable hypotheses, under somewhat different conditions. They are important to ANNs because they provide the basis for learning networks of many units. One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule

$W_i \leftarrow w_i + \Delta w_i$ ……..(1.2)

where $\Delta w_i = \delta(t - o)x_i$ here t is the target output for the current training example, o is the output generated by the perceptron, and δ is a positive constant called the learning rate. The role of the learning rate is to moderate the degree to which weights is changed at each step. It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

Let the training example is correctly classified already by the perceptrons. In this case, (t - o) is zero, making Δw_i zero, so that no weights are updated. Suppose the perceptron outputs a -1, when the target output is + 1. To make the perceptron output a + 1 instead of - 1 in this case, the weights must be altered to increase the value of $w_i.x_i$. For example, if $x_i>0$, then increasing w_i will bring the perceptron closer to correctly classifying this example. Notice the training rule will increase w_i, in this case, because (t - o), δ, and x_i are all positive. For example, if $x_i = .8$, $\delta = 0.1$, t = 1, and o = - 1, then the weight update will be $\Delta w_i = \delta (t - o) x_i = 0.1(1 - (-1))0.8 = 0.16$. On the other hand, if t = -1 and o = 1, then weights associated with positive x_i will be decreased rather than increased.

In fact, the above learning procedure can be proven to converge within a finite number of applications of the perceptron training rule to a weight vector that correctly classifies all training examples, provided the training examples are linearly separable and provided a sufficiently small δ is used. If the data are not linearly separable, convergence is not assured.

PERCEPTRON GRADIENT AND DELTA RULE

Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable. A second training rule, called the delta rule, is designed to overcome this difficulty. If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.

The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. This rule is important because gradient descent provides the basis for the BACKPROPAGATION algorithm, which can learn networks with many interconnected units. It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.

The delta training rule is best understood by considering the task of training an unthresholded perceptron; that is, a linear unit for which the output o is given by

$o(x_v) = w_{vi}.x_{vi}$ .....(1.3)

Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold. In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the training error of a hypothesis (weight vector), relative to the training examples. Although there are many ways to define this error, one common measure that will turn out to be especially convenient is

$E(w_{vi}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$ .....(1.4)

Where D is the set of training examples, t_d is the target output for training example d and o_d is the output of the linear unit for training example d. By this definition, $E(w_{vi})$ is simply half the squared difference between the target output td and the hear unit output o_d, summed over all training examples. Here we characterize E as a function of w_{vi}, because the linear unit output o depends on this weight vector. E also depends on the particular set of training examples, but we assume these are fixed during training, so we do not bother to write E as an explicit function of these. We can apply Bayesian justification for choosing this particular definition of E. In particular, there we show that under certain conditions the hypothesis that minimizes E is also the most probable hypothesis in H given the training data.

THE HYPOTHESIS SPACE RESULT

To understand the gradient descent algorithm, it is helpful to visualize the entire hypothesis space of possible weight vectors and their associated E values, as illustrated in Figure . Here the axes w_0 and w_1 represent possible values for the two weights of a simple linear unit. The w_0, w_1 plane therefore represents the entire hypothesis space. The vertical axis indicates the error E relative to some fixed set of training examples. The error surface shown in the figure thus summarizes the desirability of every weight vector in the hypothesis space. Given the way in which we chose to define E, for linear units this error surface must always be parabolic with a single global minimum. The specific parabola will depend, of course, on the particular set of training examples. Gradient descent search determines a weight vector that minimizes E by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface. This process continues until the global minimum error is reached.

DESCENT RULE DERIVATION

The direction of steepest descent along the error surface can be found by computing the derivative of E with respect to each component of the vector w_i. This vector derivative is called the gradient of E with respect to w_i written $\Delta E(w_i)$.

$\Delta E(w_i) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots\ldots \frac{\partial E}{\partial w_n}\right]$ .....(1.5)

$\Delta E(w_i)$ is a vector itself, whose components are the partial derivatives of E with respect to each of the w_i. When interpreted as a vector in weight space, the gradient specializes the direction that produces the steepest increase in E. The negative of this vector therefore gives the direction of steepest decrease. Since the gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

$W_{vi} \leftarrow w_{vi} + \Delta w_{vi}$

Where $\Delta w_{vi} = -\delta \Delta E (w_{vi})$……(1.6)

Where δ is a positive constant called the learning rate, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that decreases E. This training rule can also be written in its component form

$\Delta w_{vi} = -\delta \frac{\partial E}{\partial w_i}$ ……..(1.7)

Which makes it clear that steepest descent is achieved by altering each component w_i, of w_{vi} in proportion to $\frac{\partial E}{\partial w_i}$.

To summarize, the gradient descent algorithm for training linear units is as follows. Pick an initial random weight vector. Apply the linear unit to all training examples, and then compute Δw_i for each weight according to Equation. Update each weight w_i by adding Δw_i , then repeat this process. Because the error surface contains only a single global minimum, this algorithm will converge to a weight vector with minimum error, regardless of whether the training examples are linearly separable, given a sufficiently small learning rate δ is used. If δ is too large, the gradient descent search runs the risk of overstepping the minimum in the error surface rather than settling into it. For this reason, one common modification to the algorithm is to gradually reduce the value of δ as the number of gradient descent steps grows.

CONCLUSION AND APPROXIMATION TO GRADIENT

Gradient descent is an important general paradigm for learning. It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever (1) the hypothesis space contains continuously parameterized hypotheses (the weights in a linear unit), and (2) the error can be differentiated with respect to these hypothesis parameters. The key practical difficulties in applying gradient descent are (1) converging to a local minimum can sometimes be quite slow (it can require many thousands of gradient descent steps), and (2) if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

We have considered two similar algorithms for iteratively learning perceptron weights. The key difference between these algorithms is that the perceptron training rule updates weights based on the error in the thresholded perceptron output, whereas the delta rule updates weights based on the error in the unthresholded linear combination of inputs. The difference between these two training rules is reflected in different convergence properties. The perceptron training rule converges after a finite number of iterations to a hypothesis that perfectly classifies the training data, provided the training examples are linearly separable. The delta rule converges only asymptotically toward the minimum error hypothesis, possibly requiring unbounded time, but converges regardless of whether the training data are linearly separable.

REFERENCES

1. Bishop, C. M. (1996). *Neural networks for pattern recognition*. Oxford, England: Oxford University Press.
2. Freeman, J. A., & Skapura, D. M. (1991). *Neural networks*. Reading, MA: Addison Wesley.
3. Fu, L. (1994). *Neural networks in computer intelligence*. New York: McGraw Hill.
4. Huang, W. Y., & Lippmann, R. P. (1988). Neural net and traditional classifiers. In Anderson (Ed.), *Neural Information Processing Systems* (pp. 387-396).
5. Williams, R., & Zipser, D. (1995). Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin & D. Rumelhart (Eds.), *Backpropagation: Theory, architectures, and applications* (pp. 433-486). Hillsdale, NJ: Lawrence Erlbaum Associates
6. Chauvin, Y., & Rumelhart, D. (1995). BACKPROPAGATION:Theory, architectures, and applications(edited collection). Hillsdale, NJ: Lawrence Erlbaum Assoc.
7. Cybenko, G. (1988). Continuous valued neural networks with two hidden layers are sufficient (Technical Report). Department of Computer Science, Tufts University, Medford, MA.