

EE543 - Neurocomputers and Deep Learning

Ahmet Taha Albayrak

December 2021

Chapter 1

From Biological Neuron To Artificial Neuorn Model

Chapter 2

Recurrent Neural Networks and Neurodynamis

Chapter 3

Neural Networks as Associative Memory

Chapter 4

Learning in Feed Forward Neural Networks

4.1 Perceptron Convergence Procedure

Perceptron was introduced by Frank Rosenblatt in the late 1950's with a learning algorithm. Perceptron may have continuous valued inputs. It works in the the same way as the formal artificial neuron defined previously Its activation is determined by equation: $\alpha = w^T u + \theta$

Now, consider such a perceptron in N dimensional space, the equation $w^T u + \theta = 0$ that $w_1 u_1 + w_2 u_2 + \dots + w_n u_n = 0$ defines a hyperplane. This hyperplane divides the input space into two parts such that at one side, the perceptron has output value +1, and in the other side, it is -1.

A perceptron can be used to decide whether an input vector belongs to one of two classes say classes A and B. The decision rule may be set as to respond as class A if the output is +1 and as class B if the output is -1. The perceptron forms two decision regions separated by the hyperplane. The equation of the boundary hyperplane depends on the connection weights and threshold

Connection weights and the threshold in a perceptron can be fixed or adapted using a number of different algorithms. The original convergence procedure developed by Rosenblatt for adjusting weights is provided in the following

- Step 1: Initialize weight and threshold
- Step 2: Present New Input and Desired output
- Step 3: Calculate actual output
- Step 4: Adapt weights
- Step 5: Repeat steps 2-4 no error occurs

In [Rosenblatt, 1959] it is proved that if the inputs presented from the two classes are linearly separable, that is they fall on opposite side of some hyperplane, then the perceptron convergence procedure always convergence in time. Furthermore, it positions the final decision hyperplane such that it separates the sample of class A from those of class B

One problem with the perceptron convergence procedure is that decision boundary may oscillate continuously when the distribution overlap or the classes are not linearly separable.

4.2 LMS Learning Rule

A modification to the perceptron convergence procedure forms the Least Mean Square (LMS) solution for the case that the classes are not separable. This solution minimizes the mean square error between the desired output and the actual output of the processing element. The LMS algorithm was first proposed for Adaline (Adaptive Linear Element) in [Widrow and Hoff 60].

The output function of the Adaline can be represented by the identity function as $f(a) = a$. So the output can be written in terms of input and weights as $x = f(a) = \sum_{j=0}^N w_j u_j$ where the bias is implemented via a connection to a constant input u_0 which means the input vector and the weight vector are of space $R^{(N+1)}$ instead of R^N . The output equation of Adaline can be written as: $x = w^T u$ where w and u are weight and input vectors respectively having dimension $N + 1$.

Suppose that we have a set of input vectors $u^k, k = 1, 2, \dots, K$, each having its own desired output value y^k . The performance of the Adaline for a given input value u^k can be defined by considering the difference between the desired output y^k and the actual output x^k , which is called error and denoted as ε^k . Therefore, the error for the input u^k is as follows: $\varepsilon^k = y^k - x^k = y^k - w^T u^k$. The aim of LMS learning is to adjust the weights through a training set

u^k, y^k $k = 1, 2..K$ such that the mean of square of the errors is minimum. The mean square error is defined as $\langle (\varepsilon^k)^2 \rangle = \lim_{K \rightarrow \infty} 1/K \sum_{k=1}^K (\varepsilon^k)^2$ where the notation $\langle . \rangle$ denotes the mean value.

The mean square error can be written as $\langle (\varepsilon^k)^2 \rangle = \langle (y^k - w^T u^k)^2 \rangle = \langle (y^k)^2 \rangle + w^T \langle u^k \times u^k \rangle w - 2 \langle y^k u^k \rangle w$ where T denotes transpose and \times denotes outer vector product. Defining input correlation matrix R and a vector P as $R = \langle u^k \times u^k \rangle = \langle u^k u^{(kT)} \rangle$ and $P = \langle y^k u^k \rangle$ results in $e(w) = \langle (\varepsilon^k)^2 \rangle = \langle (y^k)^2 \rangle + w^T R w - 2P^T w$. The optimum value w^* for weight vector corresponding to the minimum of the mean square error can be obtained by evaluating the gradient of $e(w)$.

The point which makes gradient zeros gives us the value of w^* . That is $\nabla e(w)|_{w=w^*} = \frac{de(w)}{dw}|_{w=w^*} = 2Rw^* - 2P = 0$. Here the gradient is $\nabla e(w) = [\frac{de}{dw_1} \frac{de}{dw_2} \dots \frac{de}{dw_n}]^T$ and it is a vector extending in the direction of the greatest rate of change. The gradient of a function evaluated at some point is zero if the function has a maximum or minimum at that point. The error function is of the second degree. So it is a paraboloid and it has a single minimum at point w^* .

4.3 Stepest Decent Algorithm

The analytical calculation of the optimum weight vector for a problem is rather difficult in general. Not only does the matrix manipulation get cumbersome for the large dimensions, but also each component of R and P require knowledge of the statistics of the input signal [Freeman 91]. A better approach would be to let the Adaline Linear Combiner to find the optimum weights by itself a search over the error surface.

Instead of having a purely random search, some intelligence is added to procedure such that weight vector is changed by considering the gradient of $e(w)$ iteratively [Widrow 60], according to formula known as **delta rule**. $w(t+1) = w(t) + \Delta w(t)$ where $\Delta w(t) = -\eta \nabla e(w(t))$. In the formula η is a small positive constant

For the real valued scalar function $e(w)$ on a vector space $w \in R^N$, the gradient $\nabla e(w)$ gives the direction of the steepest upward slope, so the negative of the gradient is the direction of the steepest descent. We have considered the linear output function in the derivation of the optimum weight w^* for the minimum error. However in the general case, we should consider any nonlinearity $f(.)$ at the output of the neuron. It should be noted that in such a case

the error surface is no more paraboloid, so it may have several local minima. For an input u^k applied at the time t , $(e^k(t))^2$ can be used as an approximation to $\langle \varepsilon(k)^2 \rangle$ where $\varepsilon^k(t) = y^k - f(a^k) = y^k - f(w(t)^T u^k)$. Therefore, we obtain, $\nabla \langle (\varepsilon^k)^2 \rangle \simeq \nabla (\varepsilon^k(t))^2 = \nabla (y^k - f(a^k))^2$ with differentiable function $f(\cdot)$ having derivative $f'(\cdot)$, it becomes $\nabla (y^k - f(a))^2 = -2\varepsilon^k(t)f'(a)\nabla a$

Since $\nabla a^k = \nabla w(t)^T u^k = u^k$ the weight update formula becomes $w(t+1) = w(t) + 2\eta\varepsilon^k(t)f'(a)u^k$

- Step 1: Apply an input vector u^k with the desired output value y^k to the neuron's input.
- Step 2: By considering u^k and using the current value of the weight vector determine the value of the activation vector a^k where $a^k = w(t)^T u^k$
- Step 3: Determine the value of the derivation of the output function using the current value of activation a^k that is $f'(a^k) = \left. \frac{f(a)}{a} \right|_{a=a^k}$.
- Step 4: Determine the value of error $\varepsilon^k(t)$ as $\varepsilon^k(t) = y^k - f(a^k)$
- Step 5: Update the weight vector with respect to following update formula $w(t+1) = w(t) + 2\eta f'(a^k)\varepsilon^k(t)u^k$.
- Step 6: Repeat steps 1-5 until $\langle \varepsilon^k(t) \rangle$ reduces to an acceptable level.

Notice that the iterative weight update by the **delta rule** is derived by assuming constant u^k . Therefore, it tends to minimize the error with respect to applied u^k . In fact, we require the average error to be minimized. This implies that $\frac{de}{dw_j} = \frac{1}{K} \sum_{k=1}^K \frac{d(\varepsilon^k)^2}{dw_j} = \frac{1}{K} \sum_{k=1}^K \frac{2\varepsilon^k d\varepsilon^k}{dw_j}$

Therefore, the net change in w_j after one complete cycle of pattern presentation is expected to be $w_j(t+K) = w_j(t) - \eta \frac{1}{K} \sum_{k=1}^K \frac{2\varepsilon^k d\varepsilon^k}{dw_j}$. However, this would be true that if the weights are not updated along a cycle but only at the end. By changing the weights as each pattern is presented, we depart to some extent from gradient descent in e . Nevertheless, provided the learning rate is sufficiently small, this departure will be negligible and **delta rule** will implement a very close approximation to gradient descent in mean squared error [Freeman 91].

4.4 The Backpropagation Algorithm : Single Layer Network