**Implementation of Delta Rule Learning**

**Bala Vinay Kumar Pappu**

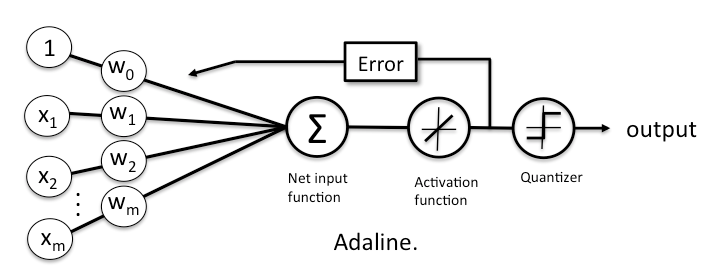
*Abstract*— The Delta rule of the adaline (also known as Widrow-Hoff” rule or Adaline rule) updates the weights based on a linear activation function rather than a unit step function. One of the biggest advantages of the linear activation function over the unit step function is that it is differentiable

Keywords— Delta rule, linear activation function, unit step function

1.Introduction

Delta rule learning is Supervised learning algorithm. developed by Widrow and Hoff, the delta rule, also called the Least Mean Square (LMS) method, is one of the most commonly used learning rules. The delta rule is also called *LMS* (*least-mean-square*) *rule* or *Widrow-Hoff rule*.

This linear activation function g(**z**)g(z) is just the identity function of the net input g(**wTx**) =wTx. In the next section, we will see why this linear activation is an improvement over the perceptron update and where the name “delta rule” comes from.



2. Gradient Descent

linear activation function allows us to define a cost function J(**w**) that we can minimize in order to update our weights. In the case of the linear activation function, we can define the cost function J(**w**) as the sum of squared errors(SSE), which is similar to the cost function that is minimized in ordinary least squares (OLS) linear regression.

J(**w**)=1/2∑i(target(i)−output(i))2output(i)∈ℝ

(The fraction 1/2 is just used for convenience to derive the gradient as we will see in the next paragraphs.)

In order to minimize the SSE cost function, we will use gradient descent, a simple yet useful optimization algorithm that is often used in machine learning to find the local minimum of linear systems.

Before we get to the fun part (calculus), let us consider a convex cost function for one single weight. As illustrated in the figure below, we can describe the principle behind gradient descent as “climbing down a hill” until a local or global minimum is reached. At each step, we take a step into the opposite direction of the gradient, and the step size is determined by the value of the learning rate as well as the slope of the gradient. 

*Fig 2: Schematic of gradient descent*

Now, as promised, onto the fun part – deriving the Adaline learning rule. As mentioned above, each update is updated by taking a step into the opposite direction of the gradient Δ**w**=−η∇J(**w**). Thus, we have to compute the partial derivative of the cost function for each weight in the weight vector: Δwj =−η ∂J/∂wj.

The partial derivative of the SSE cost function for a particular weight can be calculated we get.

Δwj=− η∑i(t(i)−o(i)) x(i)j.

Eventually, we can apply a simultaneous weight update similar to the perceptron rule:

W= w + Δw

**Although, the learning rule above looks identical to the perceptron rule, we shall note the two main differences:**

1. Here, the output “o” is a real number and not a class label as in the perceptron learning rule.
2. The weight update is calculated based on the weights incrementally after each sample, which is why this approach is also called “online” gradient descent.

3.Algorithm

For a given input vector, the output vector is compared to the correct answer. If the difference is zero, no learning takes place; otherwise, the weights are adjusted to reduce this difference. The change in weight from ui to uj is given by: dwij = r\* ai \* ej, where r is the learning rate, ai represents the activation of ui and ej is the difference between the expected output and the actual output of uj. If the set of input patterns form a linearly independent set, then arbitrary associations can be learned using the delta rule.

The gradient descent rule updates the weights after calculating the whole error accumulated from all examples, the incremental version approximates the gradient descent error decrease by updating the weights after each training example.

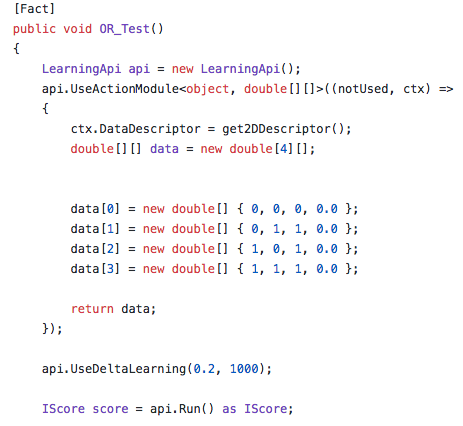
Incremental gradient descent is implemented according to the Delta rule:



4. Implementation

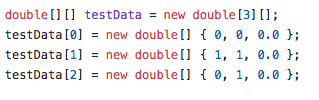
Now it is evident that we have an idea of the entire process of Delta Rule Learning and we can implement it in C# .NET core. Below is the process for implementation.

* + 1. Training:
* Provide the input data to the Delta Rule Algorithm
* Run for some required number of samples until the weights are correctly updated.

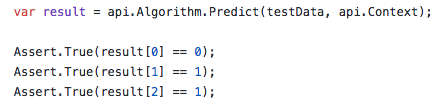


* + 1. Predicting:

After the Training, the neuron weights are updated. Once the weights are updated, we give a test data which compares with the expected output and predicts the required output.



5.Results

****

**6.** Conclusion

Implemented the Delta rule algorithm in .NET Core and implemented it in Learning API with Unit Test.

References

* [1] https://en.wikipedia.org/wiki/Delta\_rule
* [2] The Adaptive Filters, Simon Haykin
* http://uhavax.hartford.edu/compsci/neural-networks-delta-rule.html
* [3] Adaptive Filters, Bernard Widrow (http://www-isl.stanford.edu/~widrow/papers/b1971adaptivefilters.pdf)
* [4] https://en.wikipedia.org/wiki/System\_identification
* [5] https://en.wikipedia.org/wiki/Black\_box