PERCEPTRON DELTA RULE

E. Fersini

INTRODUCTION

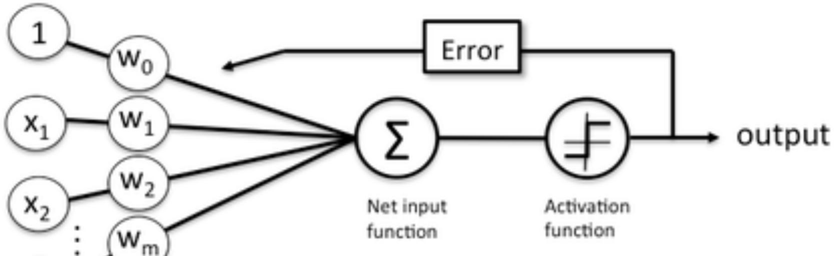
- Both **Adaline** and the Perceptron are (single-layer) neural network models. The Perceptron is one of the oldest and simplest learning algorithms, and Adaline is an improvement over the Perceptron.

COMMON FEATURES

- What Adaline and the Perceptron have in common
  - they are classifiers for binary classification
  - both have a linear decision boundary
  - both can learn iteratively
  - both use a threshold function

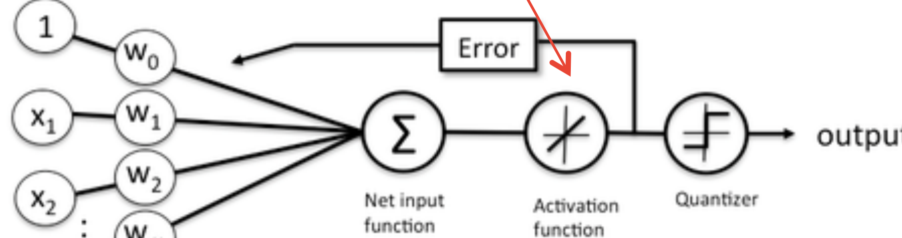
DIFFERENCES

- The Perceptron uses the class labels to learn model coefficients
- Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more “powerful” since it tells us by “how much” we were right or wrong



Perceptron rule.

It could be the identity function



Adaline.

LEARNING

- Both learning algorithms can actually be summarized by 4 simple steps – given that we use gradient descent for Adaline:

  1. Initialize the weights to 0 or small random numbers.
  2. For each training sample:
     3. Calculate the output value
     4. Update the weights

We write the weight update in each iteration as:

$$w_j := w_j + \Delta w_j.$$

- In order to learn the optimal model weights **w**, we need to define a cost function that we can optimize. Here, our cost function J(·)

$$J(\mathbf{w}) = \frac{1}{2} \sum_i \left(y^{(i)} - \phi(z)_A^{(i)} \right)^2,$$

- which is is the sum of squared errors (we multiply by 1/2 to make the derivation easier)
  - y(i) is the label or target label of the i th training point x(i)
  - φ(·) is the identity function so that φ(z)=z

IN PRACTICE

1. Initialize the weights to 0 or small random numbers.
2. For k epochs (passes over the training set)
   1. For each training sample x(i)
      1. Compute the predicted output value y*(i)
      2. Compare the predicted continuous y*(i) to the actual output y(i)
      3. Compute the "weight update" value
      4. Update the "weight update" value
   2. Update the weight coefficients by the accumulated "weight update" values

IN PRACTICE

1. Initialize the weights to 0 or small random numbers.
2. For k epochs
   1. For each training sample $x^{(i)}$
      1. $\varphi(z(i))A = y^*(i)$
      2. $\Delta w_{(t+1),j} = \eta\ (y(i) - y^*(i))x^{(i)}_j$
      3. $\Delta wj := \Delta wj + \Delta w(t+1),j$
   2. $w := w + \Delta w$

Performing this global weight update can be understood as "updating the model weights by taking an opposite step towards the cost gradient scaled by the learning rate η"

$$\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}),$$

where the partial derivative with respect to each wj can be written as

$$\frac{\partial J}{\partial w_j} = -\sum_i \left(y^{(i)} - \phi(z)_A^{(i)}\right)x_j^{(i)}.$$

TRAINING DATASET CREATION

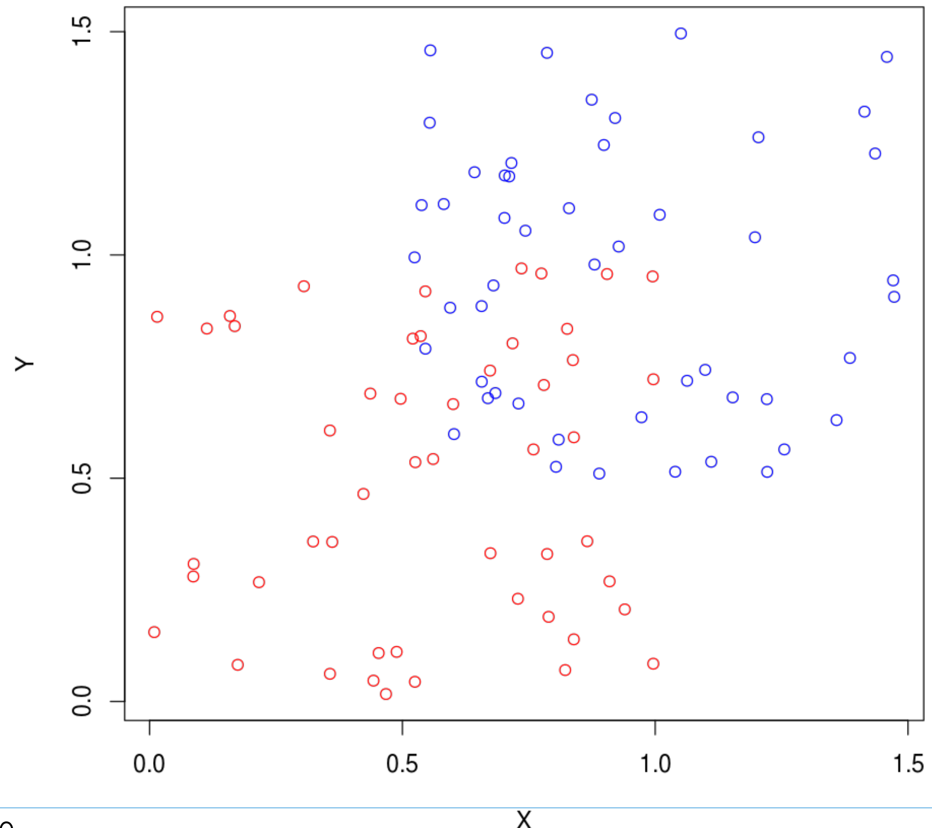
- Define a few parameters to generate the data to be used to train an Adaline algorithm.
  - We assume two groups with label -1 and 1 and the two classes are balanced.
  - We define N to be the number of data points in each group. The (x, y) coordinates for each data points are randomly created from a uniform distribution.
  - For group with label -1, the coordiate of each data points are randomly created between 0 and 1.
  - For group with label -1, the coordinate of each data points are randomly created between (x_offset, 1+x_offset) and (y_offset, 1+y_offset).
    - A larger x_offset and y_offset (for example 1.0) can create linearly seperable dataset, while a smaller x_offset and y_offset (for example 0.2) can create not linearly seperable dataset.

TRAINING SET CREATION

```
N = 50 # total number of data points each group
x_offset = 0.5 # group seperation on x axis
y_offset = 0.5 # group seperation on y axis

g1_x = runif(N, min = 0, max = 1)
g1_y = runif(N, min = 0, max = 1)

g2_x = runif(N, min = 0+x_offset, max = 1+x_offset)
g2_y = runif(N, min = 0+y_offset, max = 1+y_offset)

g_x = c(g1_x, g2_x)
g_y = c(g1_y, g2_y)
group = c(rep(-1,N), rep(1,N))

print(g_x)
print(g_y)
print(group)
```

PLOT

```
plot(g_x, g_y, type='n', xlab='X', ylab='Y')
points(g1_x, g1_y, col='red')
points(g2_x, g2_y, col='blue')
```



ADALINE PARAMETERS

- The initial weights can be randomly choosen.
  - M is the number of epoches to run
  - eta is the learning rate
  - th is the accuracy threshold to stop.
- The training will stop if we reach the number of epoches or the accuracy is larger than the accuracy threshold.
- If the two groups are not perfectly linearly seperatable, then perceptron is never going to stop. So we add two stop critera here:
  1. number of epochs
  2. the threshold of accuracy, such that the program is going to stop when either criteria meet.

ADALINE PARAMETERS

```
w0 = 0.1 # initial weitht
w1 = 0.2 # initial weight
w2 = 0.3 # initial weitht

M = 15          # number of epochs to run
eta = 0.005       # learning rate
th = 0.9        # threshold to stop
```

ASSIGNMENT

- Today we will start by implementing the Perceptron algorithm:

```
> function(data, eta) {
   .........
   .........
   .........
   .........
   .........
   .........

   return(w)
}
```

Now you should define the **DELTA RULE**

Show the behaviour of the perceptron using different **eta**