1. Perceptron Model

Here we use the Perceptron model to solve a binary classification task, in which the output contains only two classes. The decision function is a unit step function:

$$\phi(\mathrm{z}) = \begin{cases} 1 & if z \geq 0 \\ -1 & otherwise \end{cases}$$

where the net input z is calculated as

$$\mathrm{z} = w_0 x_0 + w_1 x_1 + ... + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

Here, $\boldsymbol{w}$ is the weight vector, $\boldsymbol{x}$ is the training sample inputs. w_0 is called the bias, and x_0 is set to 1.

The Perceptron Learning Rule

Repeat until converge:

Step 1: the weight vector $\boldsymbol{w}$ are initialized to some small random numbers (e.g., following a normal distribution.)

Step 2: For each training sample $\boldsymbol{x}^i$:

compute the predicted output $\hat{y}$ and update all the weights by calculating Δw_j as:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

where η is the learning rate, a value between 0 and 1. $y^{(i)}$ is the target (or the true class label) of the training set. Predicted label $\hat{y}^{(i)}$ is calculated based on the current weights for each input and the decision function.

An example of Perceptron

Now, let's work on a concrete example.

The Iris dataset consisting of 150 samples and four features is written as an matrix X:

$$\boldsymbol{X} = \begin{bmatrix} {x_1}^{(1)} & {x_2}^{(1)} & {x_3}^{(1)} & {x_4}^{(1)} \\ {x_1}^{(2)} & {x_2}^{(2)} & {x_3}^{(2)} & {x_4}^{(2)} \\ \vdots & & & \vdots \\ {x_1}^{(150)} & {x_2}^{(150)} & {x_3}^{(150)} & {x_4}^{(150)} \end{bmatrix}$$

and the targets are

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(150)} \end{bmatrix}$$

For this example, we select only two features, and will use two samples.

$$\boldsymbol{X} = \begin{bmatrix} {x_1}^{(1)} & {x_2}^{(1)} \\ {x_1}^{(2)} & {x_2}^{(2)} \end{bmatrix} = \begin{bmatrix} 6.2 & 3.4 \\ 3.9 & 3.0 \end{bmatrix}$$

and the target variables (true class labels) are

$$\boldsymbol{y} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

For the two type of outputs, we use 1, and -1 are used to indicate the two classes.

The inputs, including the weight factor, are written as:

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

and the corresponding weights are:

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

The learning process

The learning process will go through each sample one by one.

Suppose we initialize the weights randomly to $w = [w_0, w_1, w_2] = [0.2, 0.3, -0.5]$, and set $\eta = 0.1$

For the first sample: $x^{(1)} = [x_0^{(1)}, x_1^{(1)}, x_2^{(1)}] = [1, 6.2, 3.4]$, whose true class label is $y = 1$.
We first calculate the predicted output $\hat{y}$ with the decision function

$$\phi(z) = \phi(\boldsymbol{w}^T \boldsymbol{x}) = \phi(w_0 x_0 + w_1 x_1 + w_2 x_2)$$

$$\begin{aligned} \hat{y} &= \phi(w_0 x_0^{(1)} + w_1 x_1^{(1)} + w_2 x_2^{(1)}) \\ &= \phi(0.2 \times 1 + 0.3 \times 6.2 + (-0.5) \times 3.4) \\ &= \phi(0.36) \\ &= 1 \end{aligned} \tag{1}$$

Then, the weight updates are calculated as:

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

Since we are working with the first sample, the weight updates are:

$$\begin{aligned} \Delta w_0 &= \eta(y^{(1)} - \hat{y}^{(1)}) x_0^{(1)} \\ &= 0.1 \times (1 - 1) \times 1 \\ &= 0 \end{aligned} \tag{2}$$

$$\begin{aligned} \Delta w_1 &= \eta(y^{(1)} - \hat{y}^{(1)}) x_1^{(1)} \\ &= 0.1 \times (1 - 1) \times 6.2 \\ &= 0 \end{aligned} \tag{3}$$

$$\begin{aligned}\Delta w_2 &= \eta(y^{(1)} - \hat{y}^{(1)})x_2^{(1)} \\ &= 0.1 \times (1-1) \times 3.4 \\ &= 0\end{aligned} \tag{4}$$

There is no change since the predicted output is the same as the true output given the current weights.
Next, let's look at the second example: $x^{(2)} = [x_0^{(2)}, x_1^{(2)}, x_2^{(2)}] = [1, 3.9, 3.0]$, whose true class label is $y = 1$. The predicted output is:

$$\begin{aligned}\hat{y} &= \phi(w_0 x_0^{(2)} + w_1 x_1^{(2)} + w_2 x_2^{(2)}) \\ &= \phi(0.2 \times 1 + 0.3 \times 3.9 + (-0.5) \times 3.0) \\ &= \phi(-0.13) \\ &= -1\end{aligned} \tag{5}$$

We have a prediction error! So the weights will be updated. The weight updates are calculated as:

$$\begin{aligned}\Delta w_0 &= \eta(y^{(2)} - \hat{y}^{(2)})x_0^{(2)} \\ &= 0.1 \times (1 - (-1)) \times 1 \\ &= 0.2\end{aligned} \tag{6}$$

$$\begin{aligned}\Delta w_1 &= \eta(y^{(2)} - \hat{y}^{(2)})x_1^{(2)} \\ &= 0.1 \times (1 - (-1)) \times 3.9 \\ &= 0.78\end{aligned} \tag{7}$$

$$\begin{aligned}\Delta w_2 &= \eta(y^{(2)} - \hat{y}^{(2)})x_2^{(2)} \\ &= 0.1 \times (1 - (-1)) \times 3.0 \\ &= 0.6\end{aligned} \tag{8}$$

Therefore, the new weights are now $[0.2 + 0.2, 0.3 + 0.78, -0.5 + 0.6] = [0.4, 1.08, 0.1]$.
This learning process iterates through the whole sample data set, and weights are updated simultaneously while processing each sample.

2. Adaline Model

In most supervised algorithms, including Adaline, the goal is to optimize a cost function J. We can define a cost function, Sum of Squared Errors as:

$$J(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

The weight change is calculated as:

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i (y^{(i)} - \phi(z^{(i)}))x_j^{(i)}$$

Each weight is updated based on how much it contributes to the change of the cost function. Each weight update is calculated based on **all** the samples in the training set, not just based on one sample like in the Perceptron model.

Use the same initial weights $w = [w_0, w_1, w_2] = [0.2, 0.3, -0.5]$, and learning rate $\eta = 0.1$, the weight updates in the first iteration over the same two samples $i = 1, 2$ are shown below.

$$
\begin{aligned}
\Delta w_0 &= \eta \sum_i (y^{(i)} - \phi(z^{(i)})) x_0^{(i)} \\
&= 0.1 \times \left[(y^{(1)} - \phi(z^{(1)})) \times x_0^{(1)} + (y^{(2)} - \phi(z^{(2)})) \times x_0^{(2)} \right] \\
&= 0.1 \times [(1 - 0.36) \times 1 + (1 - (-0.13)) \times 1] \\
&= 0.177
\end{aligned} \tag{9}
$$

$$
\begin{aligned}
\Delta w_1 &= \eta \sum_i (y^{(i)} - \phi(z^{(i)})) x_1^{(i)} \\
&= 0.1 \times \left[(y^{(1)} - \phi(z^{(1)})) \times x_1^{(1)} + (y^{(2)} - \phi(z^{(2)})) \times x_1^{(2)} \right] \\
&= 0.1 \times [(1 - 0.36) \times 6.2 + (1 - (-0.13)) \times 3.9] \\
&= 0.8375
\end{aligned} \tag{10}
$$

$$
\begin{aligned}
\Delta w_2 &= \eta \sum_i (y^{(i)} - \phi(z^{(i)})) x_2^{(i)} \\
&= 0.1 \times \left[(y^{(1)} - \phi(z^{(1)})) \times x_2^{(1)} + (y^{(2)} - \phi(z^{(2)})) \times x_2^{(2)} \right] \\
&= 0.1 \times [(1 - 0.36) \times 3.4 + (1 - (-0.13)) \times 3.0] \\
&= 0.5566
\end{aligned} \tag{11}
$$

Therefore, by adding the updates to the current weight and they become $[0.377, 1.1375, 0.0566]$.

This weight updating process iterates till the learning converges. That is when the weights produce outputs that get closer and closer to the minimal of the cost function, or we say when it reaches the global optimal.

What Adaline and the Perceptron have in common

- They are both classifiers for binary classification
- Both have a linear decision boundary
- Both can learn iteratively, sample by sample. the Perceptron naturally, and variation of Adaline via stochastic gradient descent (refer to the last section of Chapter 2).

The differences between the Perceptron and Adaline

- 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.

https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html