NEURAL NETS HW1

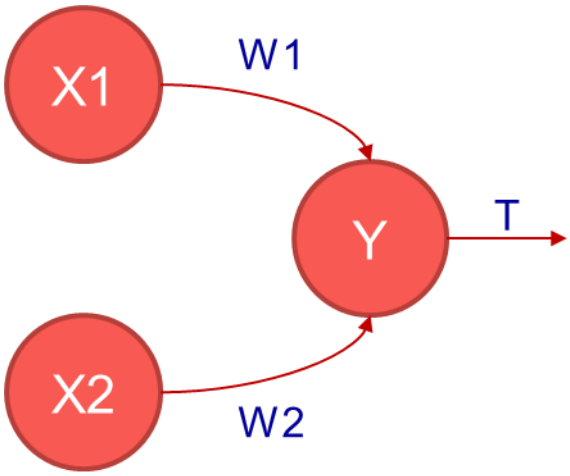
October 6, 2022

S.M. Erfan Moosavi M.

TABLE OF CONTENTS

1. Single Artificial Neuron

Since both NOR & OR get two inputs, we need a single AN with 2 inputs (and two weights). Something like this:



Since the main formula is:

$$H(w^t x - b) = \begin{cases} 1 & w^t x - b \geq 0 \\ 0 & otherwise \end{cases}$$

We can derive the following formula for our simple neurons:

$$Y = \begin{cases} 1 & X_1 W_1 + X_2 W_2 \geq T \\ 0 & X_1 W_1 + X_2 W_2 < T \end{cases}$$

Now to solve our problems, we simply put the values into the above formula.

1. *NOR Gate, Threshold: -0.5*

X1	X2	Y
0	0	1
0	1	0
1	0	0
1	1	0

Solution:

Our formula should be filled for every row of above table:

$$1: \quad 0 * W_1 + 0 * W_2 \geq -0.5$$

In case of double zeros for input, the result is always zero which is above the threshold

$$2: \quad 0 * W_1 + 1 * W_2 < -0.5$$
$$W_2 < -0.5$$

$$3: \quad 1 * W_1 + 0 * W_2 < -0.5$$
$$W_1 < -0.5$$

$$4: \quad 1 * W_1 + 1 * W_2 < -0.5$$
$$W_1 + W_2 < -0.5$$

By enforcing rule 2 & 3, rule 4 will be enforced automatically

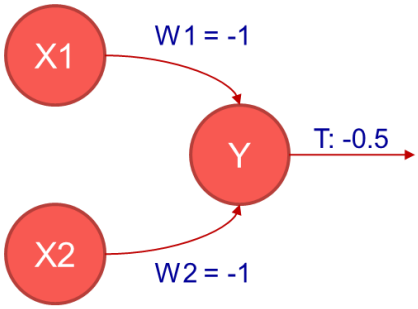
NOR Gate Weights:

$$W_1 = \{x \mid x \epsilon \mathbb{R}, \quad x < -0.5\}$$

$$W_2 = \{x \mid x \epsilon \mathbb{R}, \quad x < -0.5\}$$

Example:

$$W_1 = -0.51, -2.3, -3, -3.5, \ldots$$
$$W_2 = -0.51, -2.3, -3, -3.5, \ldots$$



3. *OR Gate, Threshold: 0.5*

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

Solution:

Our formula should be filled for every row of above table:

$$1: \quad 0 * W_1 + 0 * W_2 < 0.5$$

In case of double zeros for input, the result is always zero which is below the threshold

$$2: \quad 0 * W_1 + 1 * W_2 \geq 0.5$$
$$W_2 \geq 0.5$$

$$3: \quad 1 * W_1 + 0 * W_2 \geq 0.5$$
$$W_1 \geq 0.5$$

$$4: \quad 1 * W_1 + 1 * W_2 \geq 0.5$$
$$W_1 + W_2 \geq 0.5$$

By enforcing rule 2 & 3, rule 4 will be enforced automatically

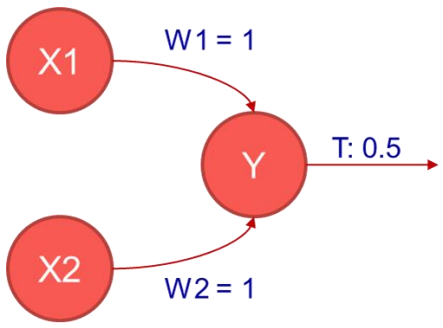
OR Gate Weights:

$$W_1 = \{x \mid x \epsilon \mathbb{R}, \quad x \geq 0.5\}$$

$$W_2 = \{x \mid x \epsilon \mathbb{R}, \quad x \geq 0.5\}$$

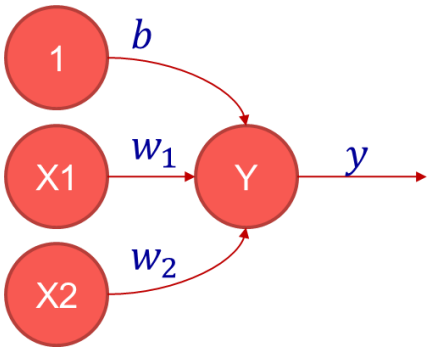
Example:

$$W_1 = 0.51, 2.3, 3, 3.5, \ldots$$
$$W_2 = 0.51, 2.3, 3, 3.5, \ldots$$



2. Adaline Neuron

X1	X2	B	D
0	0	1	1
0	1	1	0
1	0	1	0
1	1	1	0



First of all, we start by defining the learning algorithm:

$$let\ the\ weights\ \&\ bias\ learning\ algorithm\ be\ defined\ as:$$
$$w_i^{t+1} = w_i^t - \alpha\delta_i b^{t+1} = b^t - \alpha\delta_b$$
$$where\ \alpha\ is\ algorithm\ learning\ rate$$
$$and\ \delta_i\ is\ the\ gradient\ of\ cost\ function\ with\ respect\ to\ input\ i,\ which\ defined\ as:$$
$$\delta_i = \nabla\left(\frac{1}{2}c^2\right)$$
$$c = d - y$$
$$where\ y = b + x_1 w_1 + x_2 w_2$$
$$and\ d\ is\ the\ desired\ output\ of\ the\ perceptron\ for\ the\ given\ input\ x_1\ \&\ x_2$$
$$based\ on\ above\ formulation,\ \delta_i\ would\ be:$$
$$\delta_i = -cx_i = -(d - y)x_i$$
$$finally\ we\ can\ conclude\ that:$$
$$w_i^{t+1} = w_i^t - \alpha(-(d - y)x_i) = w_i^t + \alpha(d - y)x_i$$
$$b^{t+1} = b^t - \alpha(-(d - y)x_b) = b^t + \alpha(d - y)$$

Now, we can run the defined algorithm on set of input data. There are two ways for updating weights and bias:

- In the first way, we update the weights and bias with respect to just one input sample in each run (Stochastic Gradient Decent)
- In the second way, we aggregate weight updates of each sample and update weights and bias with respect to mean of those updates.

For sake of this exercise, since it's based on slides 42 to 46, we need to do it with SGD.

NOTE:

I used the term epoch at the end of this question, what I meant by epoch is:

One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Which is a definition used in the following webpage:

Epoch vs Batch Size vs Iterations | by SAGAR SHARMA | Towards Data Science

And similar definitions used here:

Difference Between a Batch and an Epoch in a Neural Network (machinelearningmastery.com)

Since one of the TAs proposed another definition for epoch on Course Telegram Group, *I just wanted to ensure whomever reading this document that I didn't used the word epoch by that definition.*

SGD:

- *Initializing:*

$$b^0 = 0.15$$
$$w_1^0 = 0.22$$
$$w_2^0 = 0.17$$
$$\alpha = 0.1$$

- *Run 1:*

$$sample\ 1: x_1 = 0\ \&\ x_2 = 0\ \&\ d = 1$$
$$y^1 = 0.15 + 0 * 0.22 + 0 * 0.17 = 0.15$$
$$b^1 = b^0 + \alpha(d - y^1) = 0.15 + 0.1(1 - 0.15) = 0.235$$
$$w_1^1 = w_1^0 + \alpha(d - y^1)x_1 = 0.22 + 0.1(1 - 0.15)0 = 0.22$$
$$w_2^1 = w_2^0 + \alpha(d - y^1)x_2 = 0.17 + 0.1(1 - 0.15)0 = 0.17$$

- *Run 2:*

$$sample\ 2: x_1 = 0\ \&\ x_2 = 1\ \&\ d = 0$$
$$y^2 = 0.235 + 0 * 0.22 + 1 * 0.17 = 0.405$$
$$b^2 = b^1 + \alpha(d - y^2) = 0.235 + 0.1(1 - 0.405) = 0.2945$$
$$w_1^2 = w_1^1 + \alpha(d - y^2)x_1 = 0.22 + 0.1(1 - 0.405)0 = 0.22$$
$$w_2^2 = w_2^1 + \alpha(d - y^2)x_2 = 0.17 + 0.1(1 - 0.405)1 = 0.2295$$

Run 3:

$$sample\ 3: x_1 = 1\ \&\ x_2 = 0\ \&\ d = 0$$
$$y^3 = 0.2945 + 1 * 0.22 + 0 * 0.2295 = 0.5145$$
$$b^3 = b^2 + \alpha(d - y^3) = 0.2945 + 0.1(1 - 0.5145) = 0.34305$$
$$w_1^3 = w_1^2 + \alpha(d - y^3)x_1 = 0.22 + 0.1(1 - 0.5145)1 = 0.26855$$
$$w_2^3 = w_2^2 + \alpha(d - y^3)x_2 = 0.2295 + 0.1(1 - 0.5145)0 = 0.2295$$

- *Run 4:*

$$sample\ 4: x_1 = 1\ \&\ x_2 = 1\ \&\ d = 0$$
$$y^4 = 0.34305 + 1 * 0.26855 + 1 * 0.2295 = 0.8411$$
$$b^4 = b^3 + \alpha(d - y^4) = 0.34305 + 0.1(1 - 0.8411) = 0.35894$$
$$w_1^4 = w_1^3 + \alpha(d - y^4)x_1 = 0.26855 + 0.1(1 - 0.8411)1 = 0.28444$$
$$w_2^4 = w_2^3 + \alpha(d - y^4)x_2 = 0.2295 + 0.1(1 - 0.8411)1 = 0.24539$$

- *After 1 epoch:* $b = 0.35894, w_1 = 0.28444, w_2 = 0.24539$

3. Neural Net Implementation

The following lines will demonstrate why I used each line of code which I filled in:

```
(1)  parameters = np.zeros((3, 1))
```

Using the zeros function of numpy module, we can create an array with desired shape. For sake of this exercise, we created a 3 by 1 zero array. [b, w1, w2]

```
(2)  X = np.concatenate([np.ones((X.shape[0], 1)), X],
  axis=1)
```

Here we used numpy concatenate function to merge a ones vertical vector with original X matrix to create a biased input. We use single bias value 1.

```
(3)  y_out = np.heaviside(X @ parameters, 0)
```

Using numpy Heaviside function, we can have binary step function functionalities. We can also define some masks to filter numbers below 0 to 0 and number above zero to 1:

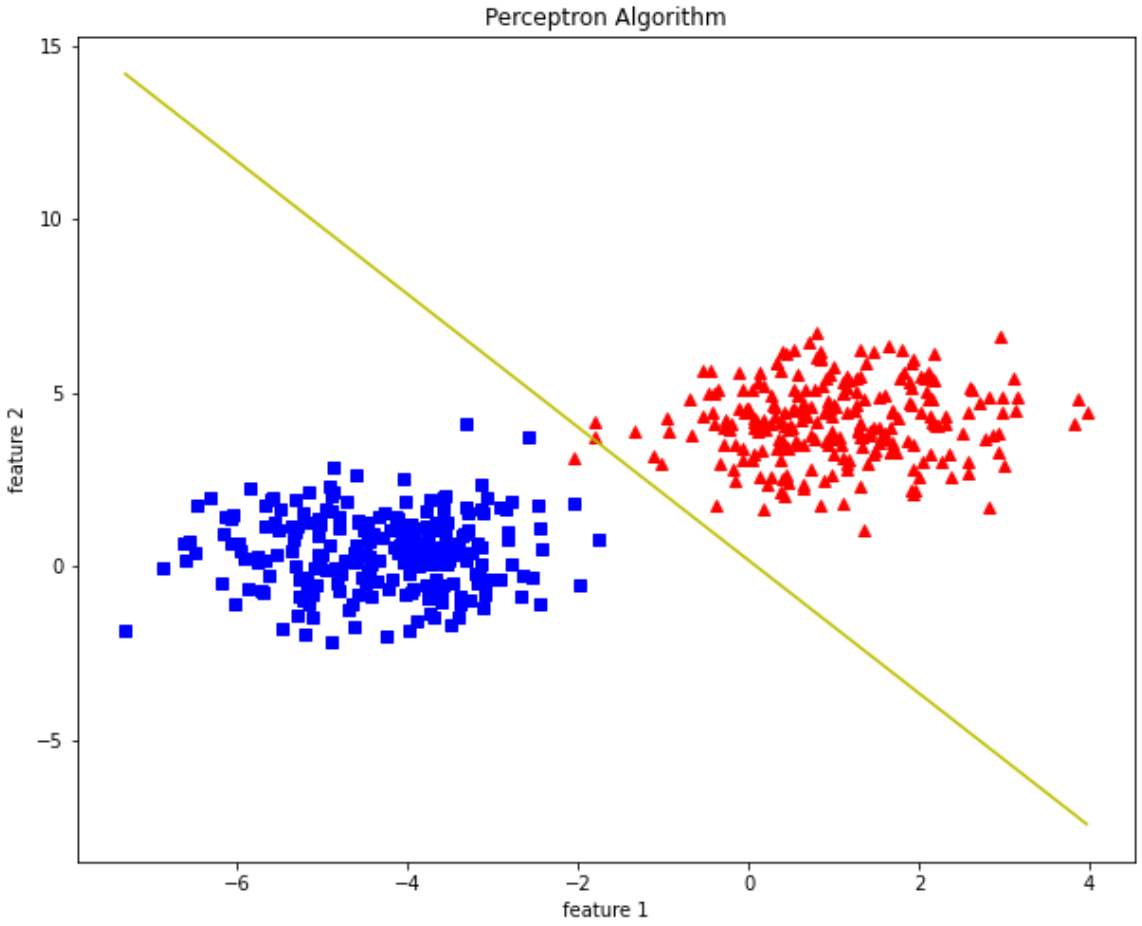
```
(4)  y_out = X @ parameters
(5)  y_out[y_out<=0] = 0
(6)  y_out[y_out>0] = 1
```

```
(7)  parameters = parameters + (lr * (y[j,0]-y_out[j,0]) *
  X[j, :].reshape(parameters.shape))
```

This line of code demonstrates the weight update procedure of learning algorithm here. Here we used delta method.

(y[j,0]-y_out[j,0]) calculates the cost and then the cost would be reduced by a factor of learning_rate which is lr.

After that, we apply the sample input values to this and then update the weight.


Perceptron Algorithm
15
10
5
0
−5
feature 2
−6
−4
−2
0
2
4
feature 1