School of Computing and Information Systems
The University of Melbourne
COMP90049 Introduction to Machine Learning (Semester 1, 2021)

Sample solutions: Week 7

1. Why is a perceptron (which uses a **sigmoid** activation function) equivalent to logistic regression?

   A perceptron has a weight associated with each input (attribute); the output is acquired by applying the activation function. This activation function can be the **step function** (as shown in the lectures) or any other (appropriate) function. If we use the **sigmoid** activation function ($\sigma(x) = f(x) = \frac{1}{1+e^{-x}}$) to the linear combination of inputs ($\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$), which simplifies to $f(\theta^T x) = \sigma(\theta^T x)$— this is the same as the logistic regression function.

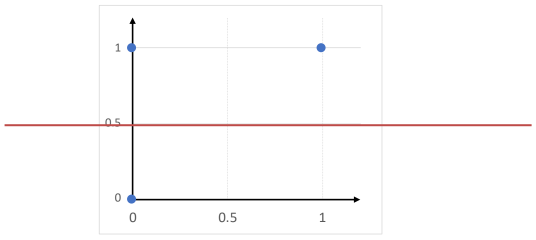
2. Consider the following training set:

(x_1, x_2)	y
(0,0)	0
(0,1)	1
(1,1)	1

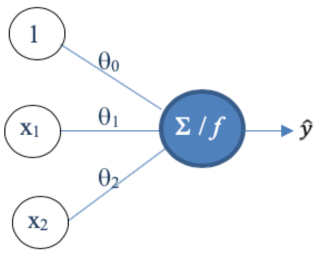
   Consider the initial weight function as $\theta = \{\theta_0, \theta_1, \theta_2\} = \{0.2, -0.4, 0.1\}$ and the activation function of the perceptron as the step function of $f = \begin{cases} 1 & if\ \Sigma > 0 \\ 0 & otherwise \end{cases}$.

   a) Can the perceptron learn a perfect solution for this data set?

   A perceptron can only learn perfect solutions for linearly separable problems, so you should ask yourself whether the data set is linearly separable. Indeed, it is. Imagine drawing the coordinates in a coordinate system, you will find that you can separate the positive example (0,0) from the negative ones ((0,1), (1,1)) with a straight line.



   b) Draw the perceptron graph and calculate the accuracy of the perceptron on the training data before training?



To calculate the accuracy of the system we first need to calculate the output (prediction) of our perceptron and then compare it the actual labels of the class.

(x_1, x_2)	$\Sigma = \theta_0 + \theta_1 x_1 + \theta_2 x_2$	$\hat{y} = f(\Sigma)$	y
(0,0)	0.2 – 0.4 x 0 + 0.1 x 0 = 0.2	*f*(0.2) = 1	0
(0,1)	0.2 – 0.4 x 0 + 0.1 x 1 = 0.3	*f*(0.3) = 1	1
(1,1)	0.2 – 0.4 x 1 + 0.1 x 1 = -0.1	*f*(-0.1) = 0	1

As you can see one of the predictions (outputs) of our perceptron match the actual label and therefore the accuracy of our perceptron is $\frac{1}{3}$ at this stage.

c) Using the perceptron *learning rule* and the learning rate of $\eta = 0.2$. Train the perceptron **for one epoch**. What are the weights after the training?

Remember the perceptron weights learning rule in iteration *t* and for train instance *i* is as follows:

$$\theta_j^{(t)} \leftarrow \theta_j^{(t-1)} + \eta\left(y^i - \hat{y}^{i,(t)}\right) x_j^i$$

For epoch 1 we will have:

(x_1, x_2)	$\Sigma = \theta_0 + \theta_1 x_1 + \theta_2 x_2$	$\hat{y} = f(\Sigma)$	y
(0,0)	0.2 - 0.4 x 0 + 0.1 x 0 = 0.2 Update θ: $\theta_0^{(1)} = \theta_0^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_0^1 = 0.2 + 0.2\ (0-1)\ 1 = 0$ $\theta_1^{(1)} = \theta_1^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_1^1 = -0.4 + 0.2\ (0-1)\ 0 = -0.4$ $\theta_2^{(1)} = \theta_2^{(0)} + \eta\left(y^1 - \hat{y}^{1,(1)}\right) x_2^1 = 0.1 + 0.2\ (0-1)\ 0 = 0.1$	1	0
(0,1)	0 – 0.4 x 0 + 0.1 x 1 = 0.1 Correct prediction → no update	1	1
(1,1)	0 – 0.4 x 1 + 0.1 x 1 = – 0.3 Update θ: $\theta_0^{(1)} = \theta_0^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_0^3 = 0 + 0.2\ (1-0)\ 1 = 0.2$ $\theta_1^{(1)} = \theta_1^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_1^3 = -0.4 + 0.2\ (1-0)\ 1 = -0.2$ $\theta_2^{(1)} = \theta_2^{(0)} + \eta\left(y^3 - \hat{y}^{3,(1)}\right) x_2^3 = 0.1 + 0.2\ (1-0)\ 1 = 0.3$	0	1

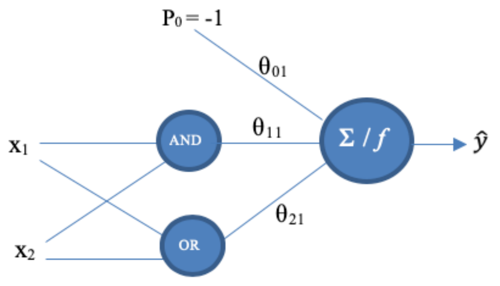
d) What is the accuracy of the perceptron on the training data after training for one epoch? Did the accuracy improve?

With the new weights we get

- for instance (0,0) with y=0: 0.2 – 0.2 x 0 + 0.3 x 0 = 0.2; f(0.2) = 1 ; **incorrect**
- for instance (0,1) with y=1: 0.2 – 0.2 x 0 + 0.3 x 1 = 0.0; f(0.5) = 1 ; **correct**
- for instance (1,1) with y=1: 0.2 – 0.2 x 1 + 0.3 x 1 = 0.1; f(0.3) = 1 ; **correct**

The accuracy of our perceptron is now $\frac{2}{3}$. So, the accuracy of the system has been improved ☺

3. Consider the two levels deep network illustrated below. It is composed of three perceptron. The two perceptron of the first level implement the AND and OR function, respectively.



Determine the weights θ_1, θ_2 and bias θ_0 such that the network implements the XOR function. The initial weights are set to zero, i.e., $\theta_{01} = \theta_{11} = \theta_{21} = 0$, and the learning rate η (eta) is set to 0.1.

Notes:

- The input function for the perceptron on level 2 is the weighted sum (Σ) of its input.
- The activation function *f* for the perceptron on level 2 is a *step function*:

$$f = \begin{cases} 1 & if \sum > 0 \\ 0 & otherwise \end{cases}$$

- Assume that the weights for the perceptron of the first level are given.

Learning Algorithm for XOR:

```
For each training example x
    $p \leftarrow (p_0, f_{AND}(x), f_{OR}(x))$
    $\hat{y} \leftarrow f(\sum_{i=0}^{n} \theta_i \, p_i)$
    $y \leftarrow target\ of\ x$
    For i = 1: n
        $\Delta\theta_i \leftarrow \eta(y - \hat{y})\, p_i$
        $\theta_i \leftarrow \theta_i + \Delta\theta_i$
```

To calculate the *output* of the two level 1 perceptron, we can use the following table (Remember since the weights for these perceptron are given, we don't need to do the iterative learning):

Train instance #	x_1	x_2	p_1	p_2	y
			$f_{AND}(x)$	$f_{OR}(x)$	$x_1\ XOR\ x_2$
1	1	0	0	1	1
2	0	1	0	1	1
3	1	1	1	1	0
4	0	0	0	0	0

Based on the results from above table, our input signals (training instances) for level 2 perceptron (XOR) are $P_i =< p_0, p_1, p_2 > = < -1\,, p_1\,, p_2 >$ and the parameters are $\theta = <\theta_{01}, \theta_{11}, \theta_{21}> = <0, 0, 0>$.

So, for the first epoch we have:

ep	*P* $<p_0, p_1, p_2>$	z = θ . *P* $\theta_0 \times p_0 + \theta_1 \times p_1 + \theta_2 \times p_2$	$\hat{y}$ *f(z)*	*y*	$\Delta\theta_{1i}$ $\eta\ (y - \hat{y})\ p_i$
1	<-1,0,1>	0 x (-1) + 0 x 0 + 0 x 1=0 Update θ to <0, 0, 0> + <-0.1,0 ,0.1>=<-0.1,0 ,0.1>	f(0)=0	1	0.1 x (1-0) x <-1, 0,1> = <-0.1, 0, 0.1>
	<-1,0,1>	(-0.1) x -1 + 0 x 0 + (0.1) x 1 = 0.2	f(0.2)=1	1	No change required
	<-1,1,1>	(-0.1) x -1 + 0 x 1 + (0.1) x 1 = 0.2 Update θ to <-0.1,0 ,0.1> + <0.1, -0.1, -0.1> = < 0, -0.1, 0>	f(0.2)=1	0	0.1 x (0-1) x <-1, 1,1> = <0.1, -0.1 ,-0.1>
	<-1,0,0>	0 x -1 + (-0.1) x 0 + 0 x 0 = 0	f(0)=0	0	No change required
2	<-1,0,1>	0 x -1 + (-0.1) x 0+ 0 x 1 = 0 Update θ to < 0, -0.1, 0> + <-0.1, 0, 0.1> = < -0.1, -0.1, 0.1>	f(0)=0	1	0.1 x (1-0) x <-1, 0,1> = <-0.1, 0, 0.1>
	<-1,0,1>	(-0.1) x -1 + (-0.1) x 0 + (0.1) x 1 = 0.2	f(0.2)=1	1	No change required
	<-1,1,1>	(-0.1) x -1 + (-0.1) x 1 + (0.1) x 1 = 0.1 Update θ to < -0.1, -0.1, 0.1> + <0.1, -0.1, -0.1> = <0, -0.2, 0>	f(0.1)=1	0	0.1 x (0-1) x <-1, 1,1> = <0.1, -0.1, -0.1>
	<-1,0,0>	0 x -1 + (-0.2) x 0 + 0 x 0 = 0	f(0)=0	0	No change required
3	<-1,0,1>	0 x -1 + (-0.2) x 0 + 0 x 1 = 0 Update θ to <0, -0.2, 0> + <-0.1, 0, 0.1> = < -0.1, -0.2, 0.1>	f(0)=0	1	0.1 x (1-0) x <-1, 0,1> = <-0.1, 0, 0.1>
	<-1,0,1>	(-0.1) x -1 + (-0.2) x 0 + (0.1) x 1 = 0.2	f(0.2)=1	1	No change required
	<-1,1,1>	(-0.1) x -1 + (-0.2) x 1 + (0.1) x 1 = 0	f(0)=0	0	No change required
	<-1,0,0>	(-0.1) x -1 + (-0.2) x 0 + (0.1) x 0 = 0.1 Update θ to < -0.1, -0.2, 0.1> + < 0.1, 0, 0> = < 0, -0.2, 0.1>	f(0.1)=1	0	0.1 x (0-1) x <-1, 0, 0> = < 0.1, 0, 0>
4	<-1,0,1>	0 x -1 + (-0.2) x 0 + (0.1) x 1 = 0.1	f(0.1)=1	1	No change required
	<-1,0,1>	0 x -1 + (-0.2) x 0 + (0.1) x 1 = 0.1	f(0.1)=1	1	No change required
	<-1,1,1>	0 x -1 + (-0.2) x 1 + (0.1) x 1 = -0.1	f(-0.1)=0	0	No change required
	<-1,0,0>	0 x -1 + (-0.2) x 0 + (0.1) x 0 = 0	f(0)=0	0	No change required

Since in the last epoch we didn't have any updated for our weights (θ_1), the algorithm converges here.

So, for our network to perform XOR function, the final θ_1 would be θ_1= < 0, -0.2, 0.1>. In other words, θ_{10} = 0, θ_{11} = -0.2 and θ_{12} = 0.1