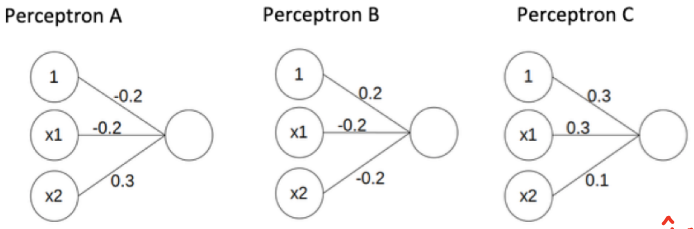
3. Consider a multiclass classification task using the three perceptrons below:



$\hat{y} = \underset{k \in Y}{\text{argmax}}\ \theta_k^T x$

Recall the training procedure for the multi-class perceptron: we predict $\hat{y}$ as the class whose parameters lead to the largest activation. During training, if a wrong class was predicted, we update the parameters activated for the wrong class (decreasing their activation), and the parameters activated for the correct class (increasing their activation).

Following this scheme, do the following:

a) *Training*: For training example $x_{train} = (0,1)$, with class **B**, give the updated weights of each perceptron after processing x. Use learning rate $\eta = 0.1$.

Perceptron A : $-0.2 + 0.3 = 0.1$

Perceptron B : $0.2 - 0.2 = 0$

Perceptron C : $0.3 + 0.1 = 0.4$

} $\hat{y} = \underset{k \in Y}{\text{argmax}}\ \theta_k^T x$

⇒ Predict C (not match with "B")

⇒ Update Perceptron B (true) & C (pred)

(B) $\theta_{y^i} \leftarrow \theta_{y^i} + \eta x^i$ move towards predicting y^i for x^i

(C) $\theta_{\hat{y}^i} \leftarrow \theta_{\hat{y}^i} - \eta x^i$ move away from predicting $\hat{y}^i$ for x^i

B : $\theta^{(0)} = \{0.2, -0.2, -0.2\}$ $\eta = 0.1$

$\theta_0^{(1)} = 0.2 + 0.1 \times 1 = 0.3$

$x = (0,1)$ $\theta_1^{(1)} = -0.2 + 0.1 \times 0 = -0.2$

$\theta_2^{(1)} = -0.2 + 0.1 \times 1 = -0.1$

C : $\theta^0 = \{0.3, 0.3, 0.1\}$

$\theta_0^{(1)} = 0.3 - 0.1 \times 1 = 0.2$

$\theta_1^{(1)} = 0.3 - 0.1 \times 0 = 0.3$

$\theta_2^{(1)} = 0.1 - 0.1 \times 1 = 0$

b) *Testing*: Using the new weights from part (a), given test example $x_{test} = (1, 1)$ with true class **C**, give the class prediction of the group of three perceptrons. Is the prediction correct? Should we change the weights?

A : $\underset{\sim}{\theta} = \{-0.2, -0.2, 0.3\}$

Perceptron A : $-0.2 - 0.2 + 0.3 = -0.1$

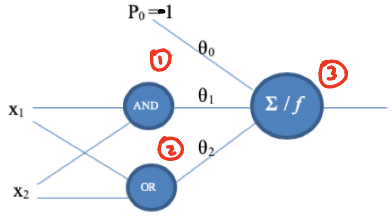
Perceptron B : $0.3 - 0.2 - 0.1 = 0$

Perceptron C : $0.2 + 0.3 + 0 = 0.5$

⇒ Predict C (Correct)

Not update (test instance)

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



Determine the weights θ_1, θ_2 and threshold θ_0 such that the network implements the XOR function. The initial weights are set to zero, i.e. $\theta_0 = \theta_1 = \theta_2 = 0$, 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 perceptrons of the first level are given. (AND, OR)

XOR: for each training example x

$p \leftarrow (p_0, f_{AND}(x), f_{OR}(x))$ (level 2)

$\hat{y} \leftarrow f(\Sigma_i \theta_i p_i) = \begin{cases} 1 & \text{if } \Sigma > 0 \\ 0 & \text{otherwise} \end{cases}$

$y \leftarrow$ target of x (XOR)

train: For $i = 1:n$

$\Delta\theta_i \leftarrow \eta (y - \hat{y}) p_i$

$\theta_i \leftarrow \theta_i + \Delta\theta_i$

(update θ) training

Output of level 1:

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

all possibilities

Inputs for level 2 perceptron: $\langle p_0, p_1, p_2 \rangle = \langle -1, p_1, p_2 \rangle$

Initial params $\theta = \langle \theta_0, \theta_1, \theta_2 \rangle = \langle 0, 0, 0 \rangle$

$\eta = 0.1$

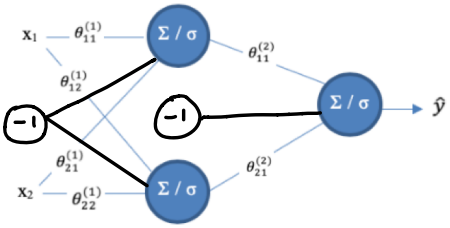
For first epoch (training):

	$\langle p_0, \overset{\text{AND}}{p_1}, \overset{\text{OR}}{p_2} \rangle$	$\Sigma = \theta_0 p_0 + \theta_1 p_1 + \theta_2 p_2$	$\hat{y} = f(\Sigma)$	y (x_1 XOR x_2)	$\Delta\theta_i = \eta(y - \hat{y}) p_i$
①	$\langle -1, 0, 1 \rangle$	0 update θ: $\theta \leftarrow \theta + \Delta\theta$ $\langle 0,0,0 \rangle + \langle -0.1, 0, 0.1 \rangle$ $= \langle -0.1, 0, 0.1 \rangle$	0	1	$0.1(1-0) \times \langle -1, 0, 1 \rangle$ $= \langle -0.1, 0, 0.1 \rangle$
②	$\langle -1, 0, 1 \rangle$	$-1(-0.1) + 0.1 = 0.2$	1	1	$\times$ (no update)
③	$\langle -1, 1, 1 \rangle$	$-0.1(-1) + 0.1 \times 1 = 0.2$ update θ: $\langle -0.1, 0, 0.1 \rangle + \langle 0.1, -0.1, -0.1 \rangle$ $= \langle 0, -0.1, 0 \rangle$	1	0	$0.1(0-1) \times \langle -1, 1, 1 \rangle$ $= \langle 0.1, -0.1, -0.1 \rangle$
④	$\langle -1, 0, 0 \rangle$	$-0.1 \times 0 = 0$	0	0	$\times$ (no update)
...	...	...	...	...	...

No changes after 4 epochs $\longrightarrow$ Converge

Final $\theta = \langle 0, -0.2, 0.1 \rangle$

2. Consider the following multilayer perceptron.



The network should implement the XOR function. Perform one epoch of *backpropagation* as introduced in the lecture on multilayer perceptrons.

Notes:

- The activation function f for a perceptron is the *sigmoid function*:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- The thresholds (bias) are not shown in the network. The threshold (bias) nodes are set to -1.
- Use the following initial parameter values:

$$\theta_{01}^{(1)} = 2 \qquad \theta_{02}^{(1)} = -1 \qquad \theta_{01}^{(2)} = -2$$

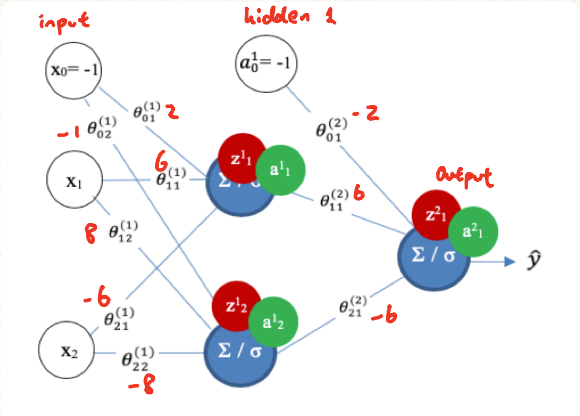
$$\theta_{11}^{(1)} = 6 \qquad \theta_{12}^{(1)} = 8 \qquad \theta_{11}^{(2)} = 6$$
$$\theta_{21}^{(1)} = -6 \qquad \theta_{22}^{(1)} = -8 \qquad \theta_{21}^{(2)} = -6$$

- The learning rate is set to $\eta = 0.7$

(i). Compute the activations of the hidden and output neurons; $a_1^{(1)}, a_2^{(1)}, a_1^{(2)}$ (level 1, level 2)

(ii). Compute the error of the network;

(iii). Backpropagate the error to determine $\Delta\theta_{ij}$ for all weights θ_{ij} and updates the weight θ_{ij}.



(i) For level 1:

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \sigma(2(-1) + 6x_1 - 6x_2) = \sigma(-2 + 6x_1 - 6x_2)$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \sigma(-1(-1) + 8x_1 - 8x_2) = \sigma(1 + 8x_1 - 8x_2)$$

For level 2:

(Output $\hat{y}$ =) $a_1^{(2)} = \sigma(z_1^{(2)}) = \sigma(-2(-1) + 6a_1^{(1)} - 6a_2^{(1)}) = \sigma(2 + 6a_1^{(1)} - 6a_2^{(1)})$

(ii) Error:

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - a_1^{(2)})^2$$

(iii) Backpropagation (efficient way of computing partial derivatives of the error of MLP wrt. each weight $\theta_{jk}^{(l)}$. $\frac{\partial E}{\partial \theta_{jk}^{(l)}}$

1. Foward pass 2. Compute error 3. Backward pass (propagate error terms)

4. Update all θ (at once) $\Delta\theta_i$

GD: $\theta_i \leftarrow \theta_i - \eta \frac{\partial E}{\partial \theta_i}$ ← By back propagation

In MLP:

$$\theta_{jk}^{(l)} \leftarrow \theta_{jk}^{(l)} + \Delta\theta_{jk}^{(l)}$$

$$\Delta\theta_{jk}^{(l)} = -\eta \frac{\partial E}{\partial \theta_{jk}^{(l)}} = \eta \delta_k^{(l)} a_j^{(l-1)}$$ ← prev layer

error term: $\delta_k^{(l)} = (y - a_k^{(l)})\, \sigma'(z_k^{(l)})$ (lecture)

(last layer) $= (y - a_k^{(l)})\, \sigma(z_k^{(l)})(1 - \sigma(z_k^{(l)}))$

Chain Rule:

$$E = \frac{1}{2}(y - a_k^{(l)})^2$$

$$\frac{\partial E}{\partial \theta_{jk}^{(l)}} = \frac{\partial E}{\partial a_k^{(l)}} \cdot \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}} \cdot \frac{\partial z_k^{(l)}}{\partial \theta_{jk}^{(l)}}$$

$$= -(y - a_k^{(l)}) \cdot \sigma'(z_k^{(l)}) \cdot a_j^{(l)}$$

$$= -\delta_k^{(l)} a_j^{(l-1)}$$

prev layer: $(l-1)$

✡ If $\frac{\partial E}{\partial \theta_{jk}^{(l-1)}} = \underbrace{\frac{\partial E}{\partial a_k^{(l)}} \cdot \frac{\partial a_k^{(l)}}{\partial z_k^{(l)}}}_{\delta_k^{(l)}} \cdot \underbrace{\frac{\partial z_k^{(l)}}{\partial a_k^{(l-1)}}}_{\theta_{jk}^{(l)}} \cdot \underbrace{\frac{\partial a_k^{(l-1)}}{\partial z_k^{(l-1)}}}_{\sigma'(z_k^{(l-1)})} \cdot \underbrace{\frac{\partial z_k^{(l-1)}}{\partial \theta_{jk}^{(l-1)}}}_{a_{jk}^{(l-2)}}$

$\underbrace{\qquad\qquad}_{\text{new } \delta_k^{(l-1)} \text{ at prev layer}}$

For training instance $\langle 1, 0 \rangle$: $y = 1$

$$a_1^{(1)} = \sigma(-2 + 6x_1 - 6x_2) = \sigma(4) = 0.982$$

$$a_2^{(1)} = \sigma(1 + 8x_1 - 8x_2) = \sigma(9) = 0.999$$

Output : $a_1^{(2)} = \sigma(2 + 6a_1^{(1)} - 6a_2^{(1)}) = \sigma(2 + 6 \times 0.982 - 6 \times 0.999) = \sigma(1.898) = 0.8691$

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(1 - 0.8691) = 0.0086$$

Back propagation : (right to left)

$$\delta_1^{(2)} = (y - a_1^{(2)})\, \sigma(z_1^{(2)})\, (1 - \sigma(z_1^{(2)}))$$

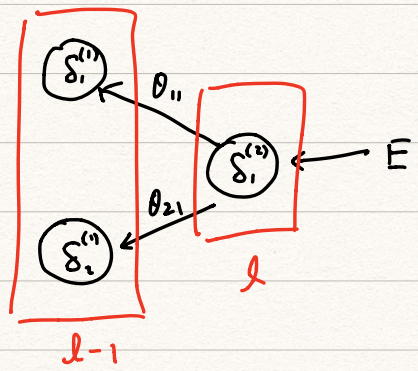
$$= (1 - 0.8691)\, \sigma(1.898)\, [1 - \sigma(1.898)]$$

$$= 0.0149$$

$$\delta_1^{(1)} = \theta_{11}^{(2)}\, \delta_1^{(2)}\, \sigma(z_1^{(1)})\, [1 - \sigma(z_1^{(1)})]$$

$$= 6 \times 0.0149 \times \sigma(4) \times [1 - \sigma(4)]$$

$$= 0.0016$$



$$\delta_2^{(1)} = \theta_{12}^{(2)}\, \delta_1^{(2)}\, \sigma(z_2^{(1)})\, [1 - \sigma(z_2^{(1)})]$$

$$= -6 \times 0.0149 \times \sigma(9) \times [1 - \sigma(9)]$$

$$= -0.00000103$$

$\eta = 0.7$, calculate $\Delta\theta$: bias (at once)

$$\Delta\theta_{01}^{(2)} = \eta\, \delta_1^{(2)}\, a_0^{(1)} = 0.7 \times 0.0149 \times (-1) = -0.0104$$

$$\Delta\theta_{11}^{(2)} = \eta\, \delta_1^{(2)}\, a_1^{(1)} = 0.7 \times 0.0149 \times 0.982 = 0.0102$$

...

...

Update Θ:

$$\Theta_{01}^{(2)} \leftarrow \Theta_{01}^{(2)} + \Delta\Theta_{01}^{(2)}$$

$$= -2 + (-0.0104) = -2.0104$$

$$\Theta_{11}^{(2)} \leftarrow \Theta_{11}^{(2)} + \Delta\Theta_{11}^{(2)}$$

$$= 6 + 0.0102 = 6.0102$$

...

This is only one training instance $\Rightarrow$ do this for all training instances to finish one epoch.