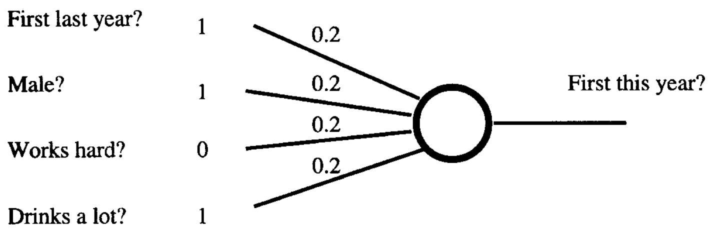
COMP 6721 Applied Artificial Intelligence (Fall 2023)

Worksheet #5: Artificial Neural Networks

Perceptron. Calculate your first neuron activation for the *Perceptron* (only 100 billion−1 more to go!):



Activation function:

$$f(\vec{x}) = \begin{cases} 1, & \text{if } \vec{x} \cdot \vec{w} \geq \text{threshold} \\ 0, & \text{otherwise} \end{cases}$$

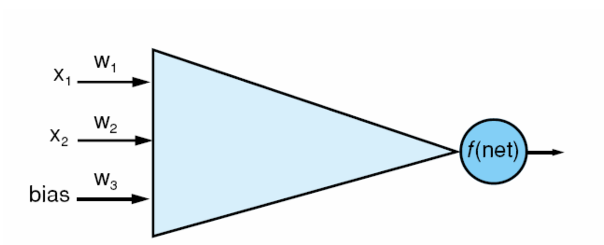
(use a threshold of 0.55):
$f(\vec{x}) =$ ....

Perceptron Learning. Ok, so for the first training example, the perceptron did not produce the right output. To learn the correct result, it has to adjust the weights: $\Delta w = \eta(T - O)$, where we set $\eta = 0.05$ (our *learning rate*). The threshold stays at 0.55. T is the expected output and O the output produced by the perceptron ($= f(\vec{x})$). (1) Start by computing the output for Richard as before. (2) Check if the computed output O is correct or not by comparing it with the expected output T. (3) If the output was not correct, compute Δw. (4) Write down the *new* weights in the next row (Alan). Remember to only update weights where the current sample (Richard) had an *active* connection (i.e., with non-zero input, here "No" = 0, "Yes"= 1). (5) Now repeat the steps, computing the output for Alan using the updated weights:

	Features (x_i)				Output
Student	'A' last year?	Male?	Works hard?	Drinks?	**'A' this year?**
Richard	Yes	Yes	No	Yes	**No**
Alan	Yes	Yes	Yes	No	**Yes**
Alison	No	No	Yes	No	**No**

Student	w_1	w_2	w_3	w_4	$f(\vec{x})$	ok?	Δw
Richard	0.2	0.2	0.2	0.2			
Alan							
Alison							

Delta Rule. In the generalized delta rule for training the perceptron, we add a *bias* input that is always one and has its own weight (here w_3). Weight changes Δw_i now take the input value x_i into account. We want the perceptron to learn the two-dimensional data shown on the right:



x_1	x_2	Output
1.0	1.0	1
9.4	6.4	−1
2.5	2.1	1
8.0	7.7	−1
0.5	2.2	1

Assume we use the sign function and set the learning rate $\eta = 0.2$. The weights are initialized randomly as shown in the table. Apply the generalized delta rule for updating the weights:

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\Delta w_i = \eta(T - O)x_i$$

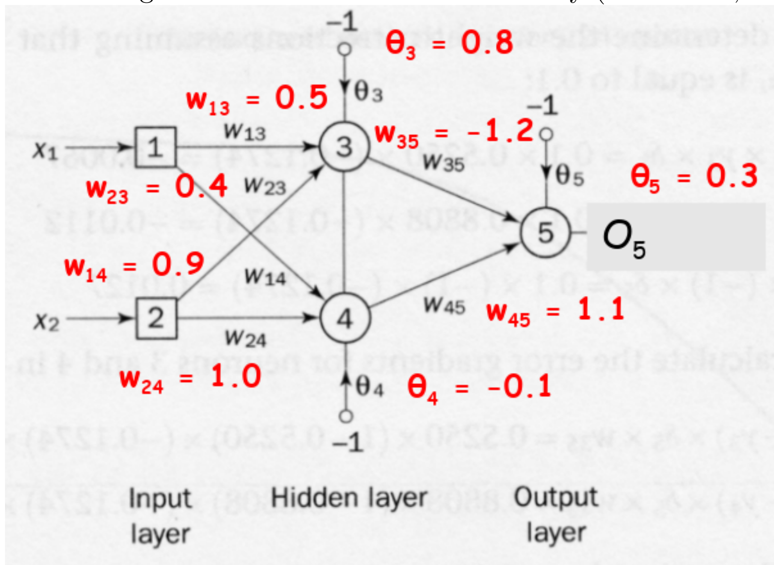
$$w' = w + \Delta w$$

Data	w_1	w_2	w_3	$f(\vec{x})$	ok?	Δw_1	Δw_2	Δw_3
#1	0.75	0.5	-0.6					
#2								
#3								
#4								

Neural Network for XOR: Backpropagation. To learn a non-linearly separable function like XOR, we'll use a neural network with a hidden layer. The weights have been initialized randomly (note: here, the bias is set to -1):

x_1	x_2	x_1 XOR x_2
1	1	0
0	0	0
1	0	1
0	1	1

$$O_i = \text{sigmoid}\left(\sum_j w_{ji}x_j\right) = \frac{1}{1+e^{-(\sum_j w_{ji}x_j)}}$$



Step 1. Compute the output for the three neurons O_3, O_4 and O_5 for the first input ($x_1 = 1, x_2 = 1$):
$O_3 =$ .... $O_4 =$ .... $O_5 =$ ....

Step 2. The next step is to calculate the error

$$\delta_o \leftarrow g'(x_o) \times \text{Err}_o = O_o(1 - O_o) \times (O_o - T_o)$$

starting from the output neuron O_5: $\delta_5 = O_5(1-O_5)\times(O_5-T_5) =$ ....

Step 3. Now we calculate the error terms for the hidden layer:

$$\delta_h \leftarrow g'(x_h) \times \text{Err}_h = O_h(1 - O_h) \times \sum_{k\in\text{outputs}} \delta_k w_{hk}$$

For the two neurons (3), (4) in the hidden layer:

- $\delta_3 = O_3(1 - O_3)\delta_5 w_{35} =$ ....
- $\delta_4 = O_4(1 - O_4)\delta_5 w_{45} =$ ....

Step 4. Now we compute our weight changes, using a constant learning rate $\eta = 0.1$:

$$\Delta w_{ij} = -\eta\delta_j x_i$$

- $\Delta w_{14} =$ ....
- $\Delta w_{24} =$ ....
- $\Delta w_{45} =$ ....
- $\Delta\Theta_5 =$ ....

Step 5. And finally, we update the weights ($w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$):

- $w_{14} = w_{14} + \Delta w_{14} =$ ....
- $w_{24} = w_{24} + \Delta w_{24} =$ ....
- $w_{45} = w_{45} + \Delta w_{45} =$ ....
- $\Theta_5 = \Theta_5 + \Delta\Theta_5 =$ ....