

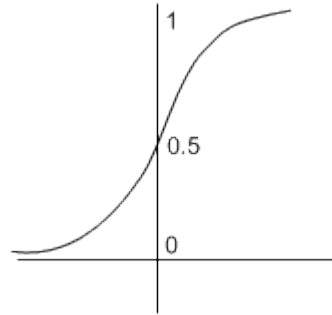
Neural Networks

Summarization

Le, My Ha, Ph.D

Example Multi-layer ANN with Sigmoid Units

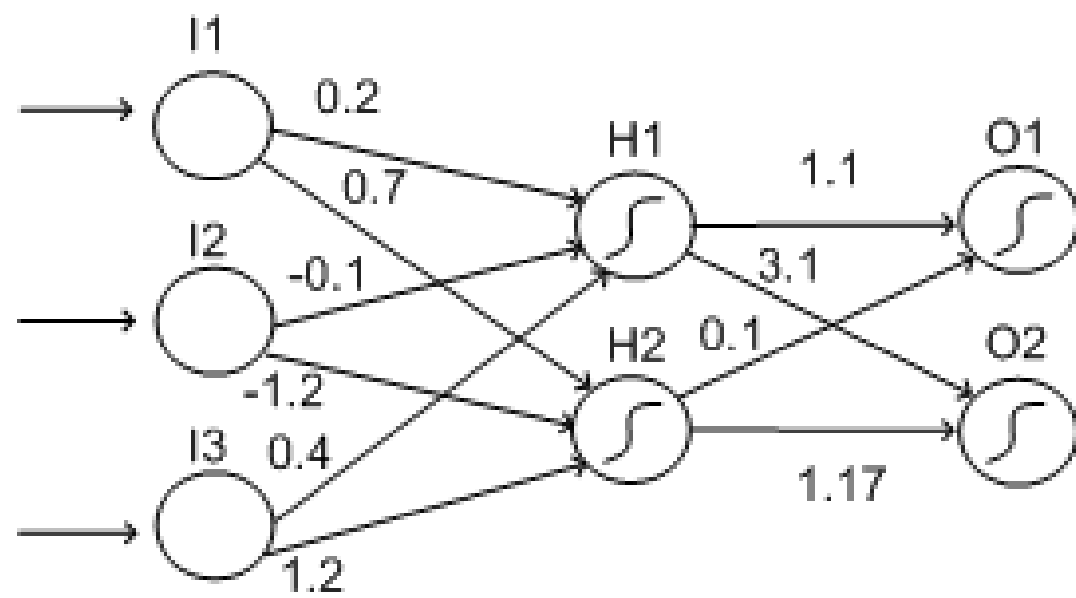
$$\sigma(S) = \frac{1}{1 + e^{-S}}$$



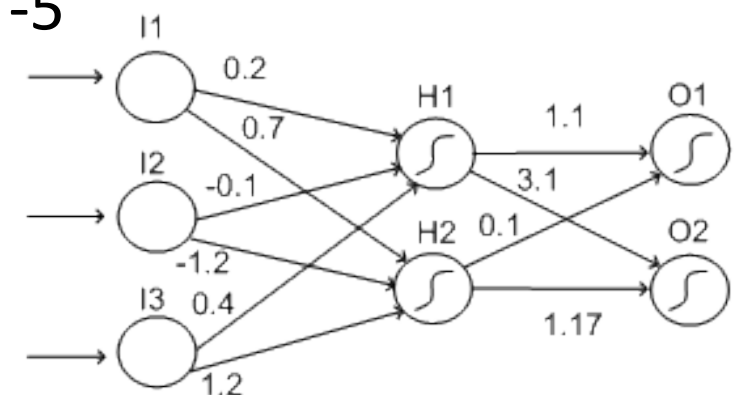
$$\frac{d\sigma(S)}{dS} = \sigma(S)(1 - \sigma(S))$$

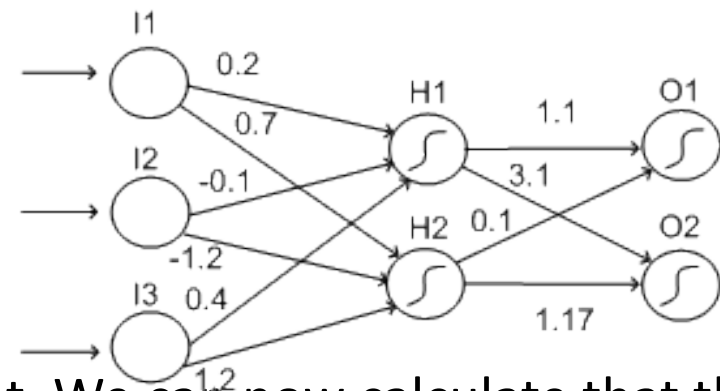
Bài tập

- Cho mạng nơ ron có 3 lớp: lớp vào 3 nơ ron, lớp ẩn 2 nơ ron, lớp ngõ ra 2 nơ ron. Hàm tác động là hàm sigmod. Ma trận trọng số $W1 = \begin{bmatrix} 0.2 & 0.7 \\ -0.1 & -1.2 \\ 0.4 & 1.2 \end{bmatrix}$; $w2 = \begin{bmatrix} 1.1 & 3.1 \\ 0.1 & 1.17 \end{bmatrix}$; tốc độ huấn luyện là 0.1.
- A. Vẽ mô hình mạng nơ ron
- B. Tính giá trị ngõ ra (lan truyền thuận) khi ngõ vào là $[10 \ 30 \ 20]$;
- C. Thực hiện lan truyền ngược và tính bộ trọng số hiệu chỉnh mới khi ngõ vào là $[10 \ 30 \ 20]$; ngõ ra là $[1 \ 0]$. Giả sử ma trận trọng số ban đầu là $W1 = \begin{bmatrix} 0.2 & 0.7 \\ -0.1 & -1.2 \\ 0.4 & 1.2 \end{bmatrix}$; $w2 = \begin{bmatrix} 1.1 & 3.1 \\ 0.1 & 1.17 \end{bmatrix}$;



- Suppose we input the values 10, 30, 20 into the three input units, from top to bottom. Then the weighted sum coming into H1 will be:
- $S_{H1} = (0.2 * 10) + (-0.1 * 30) + (0.4 * 20) = 2 - 3 + 8 = 7$.
- Then the σ function is applied to S_{H1} to give:
- $\sigma(S_{H1}) = 1/(1+e^{-7}) = 1/(1+0.000912) = 0.999$
- [Don't forget to negate S]. Similarly, the weighted sum coming into H2 will be:
- $S_{H2} = (0.7 * 10) + (-1.2 * 30) + (1.2 * 20) = 7 - 36 + 24 = -5$
- and σ applied to S_{H2} gives:
- $\sigma(S_{H2}) = 1/(1+e^5) = 1/(1+148.4) = 0.0067$





- From this, we can see that H1 has fired, but H2 has not. We can now calculate that the weighted sum going in to output unit O1 will be:
- $S_{O1} = (1.1 * 0.999) + (0.1 * 0.0067) = 1.0996$
- and the weighted sum going in to output unit O2 will be:
- $S_{O2} = (3.1 * 0.999) + (1.17 * 0.0067) = 3.1047$
- The output sigmoid unit in O1 will now calculate the output values from the network for O1:
- $\sigma(S_{O1}) = 1/(1+e^{-1.0996}) = 1/(1+0.333) = 0.750$
- and the output from the network for O2:
- $\sigma(S_{O2}) = 1/(1+e^{-3.1047}) = 1/(1+0.045) = 0.957$
- Therefore, if this network represented the learned rules for a categorisation problem, the input triple (10,30,20) would be categorised into the category associated with O2, because this has the larger output.

The Backpropagation Learning Routine

- In outline, the backpropagation method is the same as for perceptrons:
- We choose and fix our architecture for the network, which will contain input, hidden and output units, all of which will contain sigmoid functions.
- We randomly assign the weights between all the nodes. The assignments should be to small numbers, usually between -0.5 and 0.5.
- Each training example is used, one after another, to re-train the weights in the network. The way this is done is given in detail below.
- After each epoch (run through all the training examples), a termination condition is checked (also detailed below). Note that, for this method, we are not guaranteed to find weights which give the network the global minimum error, i.e., perfectly correct categorisation of the training examples. Hence the termination condition may have to be in terms of a (possibly small) number of mis-categorisations. We see later that this might not be such a good idea, though.

Equations

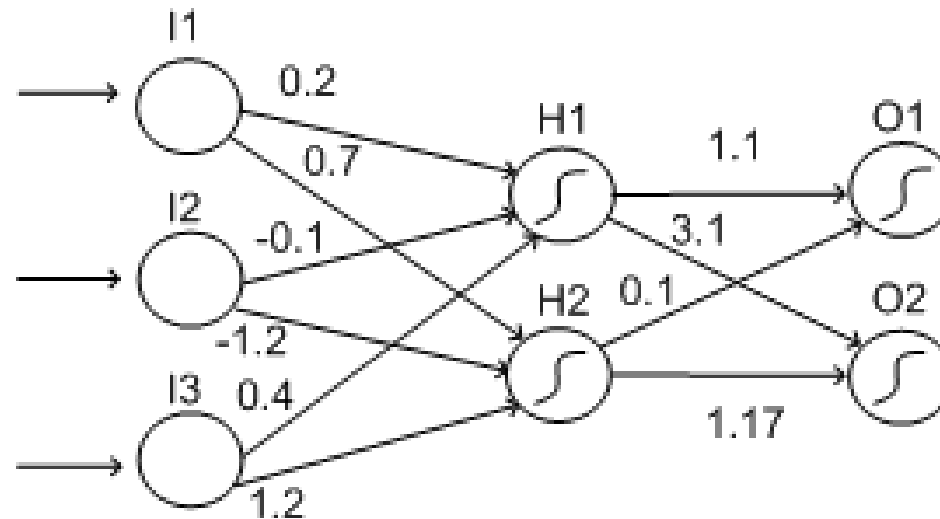
$$\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$$

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in \text{outputs}} w_{ki} \delta_{O_i}$$

$$\Delta_{ij} = \eta \delta_{H_j} x_i$$

$$\Delta_{ij} = \eta \delta_{O_j} h_i(E)$$

- We will re-use the example from section 13.1, where our network originally looked like this:



- we propagated the values (10,30,20) through the network. When we did so, we observed the following values:

Input units		Hidden units			Output units		
Unit	Output	Unit	Weighted Sum Input	Output	Unit	Weighted Sum Input	Output
I1	10	H1	7	0.999	O1	1.0996	0.750
I2	30	H2	-5	0.0067	O2	3.1047	0.957
I3	20						

$$\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$$

- Suppose now that the target categorisation for the example was the one associated with O1. This means that the network mis-categorised the example and gives us an opportunity to demonstrate the backpropagation algorithm: we will update the weights in the network according to the weight training calculations provided above, using a learning rate of $\eta = 0.1$.
- If the target categorisation was associated with O1, this means that the target output for O1 was 1, and the target output for O2 was 0. Hence, using the above notation,
- $t_1(E) = 1; \quad t_2(E) = 0; \quad o_1(E) = 0.750; \quad o_2(E) = 0.957$
- That means we can calculate the error values for the output units O1 and O2 as follows:
- $\delta_{O1} = o_1(E)(1 - o_1(E))(t_1(E) - o_1(E)) = 0.750(1-0.750)(1-0.750) = 0.0469$
- $\delta_{O2} = o_2(E)(1 - o_2(E))(t_2(E) - o_2(E)) = 0.957(1-0.957)(0-0.957) = -0.0394$

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in \text{outputs}} w_{ki} \delta_{O_i}$$

- We can now propagate this information backwards to calculate the error terms for the hidden nodes H1 and H2. To do this for H1, we multiply the error term for O1 by the weight from H1 to O1, then add this to the multiplication of the error term for O2 and the weight between H1 and O2. This gives us: $(1.1 * 0.0469) + (3.1 * -0.0394) = -0.0706$. To turn this into the error value for H1, we multiply by $h_1(E) * (1 - h_1(E))$, where $h_1(E)$ is the output from H1 for example E, as recorded in the table above. This gives us:
- $\delta_{H1} = -0.0706 * (0.999 * (1 - 0.999)) = -0.0000705$
- A similar calculation for H2 gives the first part to be: $(0.1 * 0.0469) + (1.17 * -0.0394) = -0.0414$, and the overall error value to be:
- $\delta_{H2} = -0.0414 * (0.067 * (1 - 0.067)) = -0.00259$

- We now have all the information required to calculate the weight changes for the network. We will deal with the 6 weights between the input units and the hidden units first:

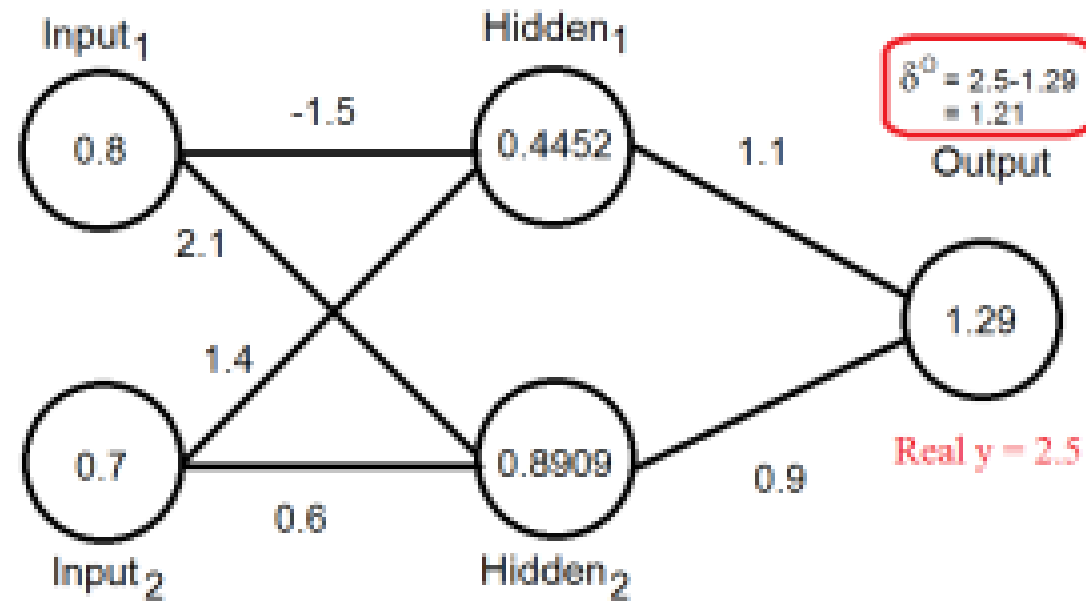
Input unit	Hidden unit	η	δ_H	x_i	$\Delta = \eta * \delta_H * x_i$	Old weight	New weight
I1	H1	0.1	-0.0000705	10	-0.0000705	0.2	0.1999295
I1	H2	0.1	-0.00259	10	-0.00259	0.7	0.69741
I2	H1	0.1	-0.0000705	30	-0.0002115	-0.1	-0.1002115
I2	H2	0.1	-0.00259	30	-0.00777	-1.2	-1.20777
I3	H1	0.1	-0.0000705	20	-0.000141	0.4	0.39999
I3	H2	0.1	-0.00259	20	-0.00518	1.2	1.1948

- We now turn to the problem of altering the weights between the hidden layer and the output layer. The calculations are similar, but instead of relying on the input values from E, they use the values calculated by the sigmoid functions in the hidden nodes: $h_i(E)$. The following table calculates the relevant values:

Hidden unit	Output unit	η	δ_o	$h_i(E)$	$\Delta = \eta * \delta_o * h_i(E)$	Old weight	New weight
H1	O1	0.1	0.0469	0.999	0.000469	1.1	1.100469
H1	O2	0.1	-0.0394	0.999	-0.00394	3.1	3.0961
H2	O1	0.1	0.0469	0.0067	0.00314	0.1	0.10314
H2	O2	0.1	-0.0394	0.0067	-0.0000264	1.17	1.16998

Learning Rate = 0.1

$$\delta_1^H = 0.4452 * (1 - 0.4452) * 1.1 * 1.21 \\ = 0.3288$$

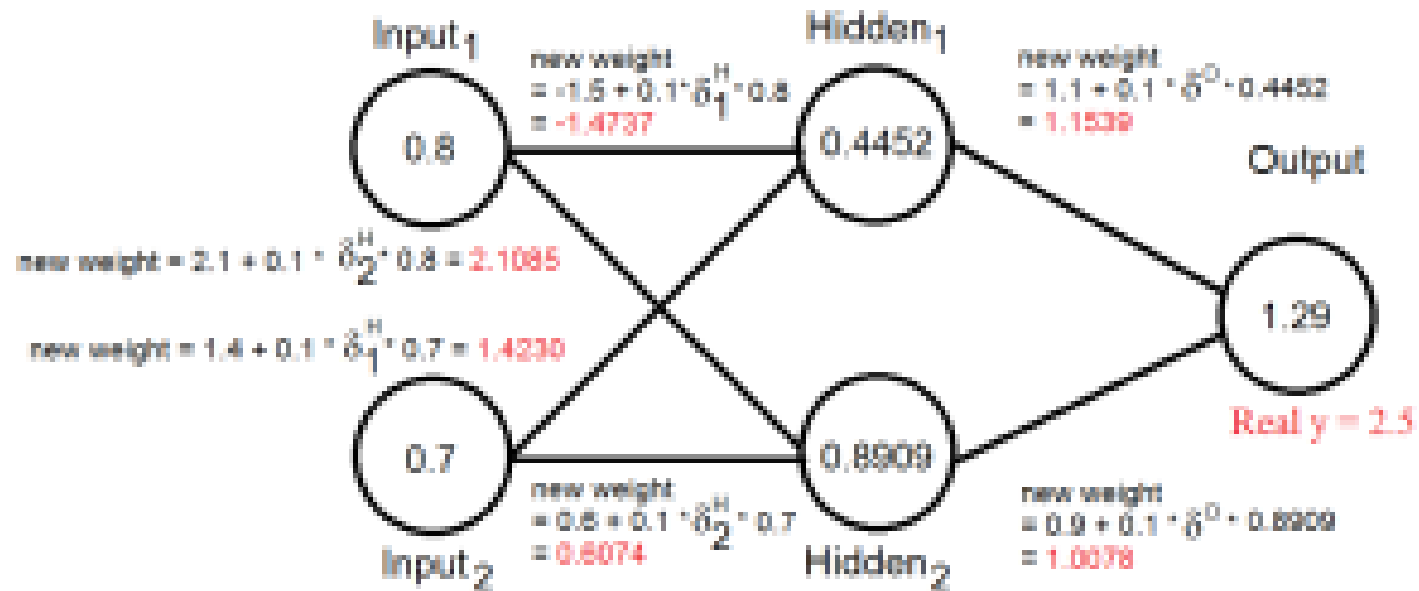


$$\delta^O = 2.5 - 1.29 \\ = 1.21$$

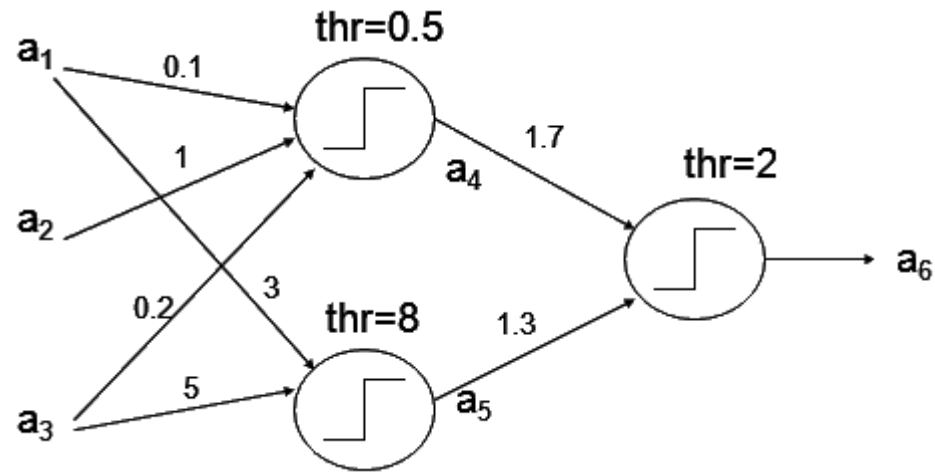
$$\delta_2^H = 0.8909 * (1 - 0.8909) * 0.9 * 1.21 \\ = 0.1058$$

Exercise 2

Learning Rate = 0.1



Exercise 3



Given: $a_1=2$, $a_2=0.05$, $a_3=3$.

First sum: $0.1 \cdot 2 + 0.05 \cdot 1 + 0.2 \cdot 3 = 0.85 \Rightarrow a_4=1$

Second: $3 \cdot 2 + 5 \cdot 3 = 21 \Rightarrow a_5 = 1$

Third sum: $1.7 \cdot 1 + 1.3 \cdot 1 = 3 \Rightarrow a_6 = 1$ (output)

- **Delta Learning for a Neuron**

- Suppose that we have some examples of input sets ($x = \langle x_1, x_2, \dots, x_n \rangle$) with known output (y).
- Error: the difference between the neuron's output and the desired output. Let h be the output of the neuron. We denote by
- $Err = (y - h(x))$
- Then we try to minimize the function
- $E = 1/2 (y - h(x))^2 = 1/2 (y - g(\sum_i w_i x_i))^2$.
- Computing the gradient of this and following the direction in which it decreases, we obtain
- $w_i = w_i + a * Err * g'(x) * x_i$,
- where a is the learning rate.

Exercise

- The formula of sigmoid activation is:
 - $$f(x) = \frac{1}{1 + e^{-\text{input}}}$$
- The algorithm works as follows:
- Perform the forwardpropagation phase for an input pattern and calculate the output error
- Change all weight values of each weight matrix using the formula $\text{weight}(\text{old}) + \text{learning rate} * \text{output error} * \text{output}(\text{neurons } i) * \text{output}(\text{neurons } i+1) * (1 - \text{output}(\text{neurons } i+1))$
- Go to step 1
- The algorithm ends, if all output patterns match their target patterns

Bài tập

- Cho mạng nơ ron có 3 lớp: lớp vào 2 nơ ron, lớp ẩn 2 nơ ron, lớp ngõ ra 1 nơ ron. Hàm tác động là hàm sigmod $f(x) = \frac{1}{1+e^{-x}}$; Ma trận trọng số $W1 = [0.42 \quad 0.62; 0.55 \quad -0.17]$; $w2 = [0.35 \quad 0.81]$; tốc độ huấn luyện là 0.25.
- A. Vẽ mô hình mạng nơ ron
- B. Tính giá trị ngõ ra (lan truyền thuận) khi ngõ vào là $[0 \ 1]$ và $[1 \ 1]$;
- C. Thực hiện lan truyền ngược và tính bộ trọng số hiệu chỉnh mới khi ngõ vào là $[0 \ 1]$ và $[1 \ 1]$; ngõ ra là $[0 \ 1]$. Giả sử ma trận trọng số ban đầu là $W1 = [0.42 \quad 0.62; 0.55 \quad -0.17]$; $w2 = [0.35 \quad 0.81]$;

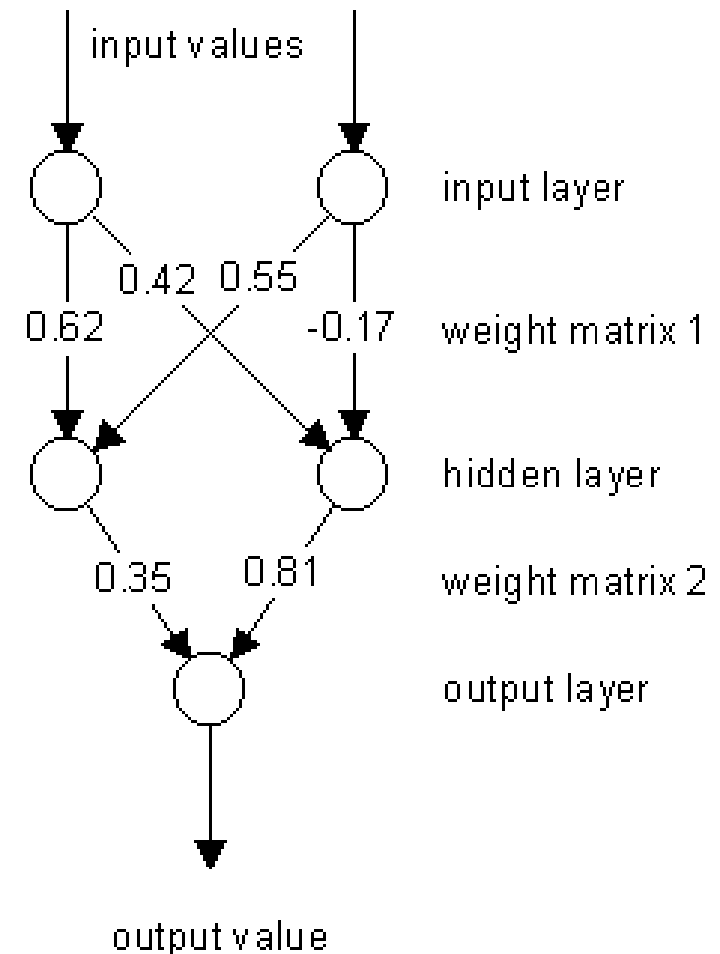
- Suppose you have the following 3-layered Multi-Layer-Perceptron:

- Patterns to be learned:

- input target

- 0 1 0

- 1 1 1



- First, the weight values are set to random values: 0.62, 0.42, 0.55, - 0.17 for weight matrix 1 and 0.35, 0.81 for weight matrix 2.
- The learning rate of the net is set to 0.25.
- Next, the values of the first input pattern (0 1) are set to the neurons of the input layer (the output of the input layer is the same as its input).

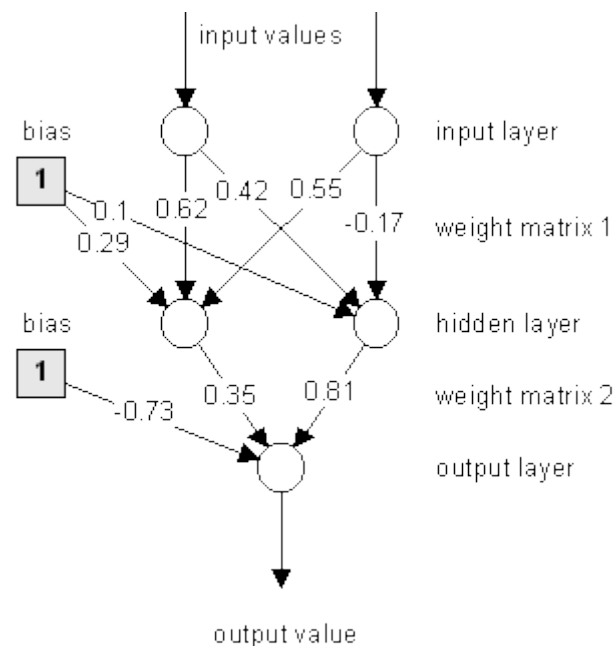
- The neurons in the hidden layer are activated:
- Ngõ vào của nơ ron ẩn thứ 1, H1: $0 * 0.62 + 1 * 0.55 = 0.55$
- Ngõ vào của nơ ron ẩn thứ 2, H2 : $0 * 0.42 + 1 * (-0.17) = -0.17$
- Ngõ ra của nơ ron ẩn thứ 1, H1 : $1 / (1 + \exp(-0.55)) = 0.634135591$
- Ngõ ra của nơ ron ẩn thứ 2, H2 : $1 / (1 + \exp(+0.17)) = 0.457602059$
- The neurons in the output layer are activated:
- Ngõ vào của nơ ron ngõ ra: $0.634135591 * 0.35 + 0.457602059 * 0.81 = 0.592605124$
- Ngõ ra của nơ ron ngõ ra : $1 / (1 + \exp(-0.592605124)) = 0.643962658$
- Compute an error value by subtracting output from target: $0 - 0.643962658 = -0.643962658$

- Now that we got the output error, let's do the backpropagation.
- Thực hiện thay đổi giá trị trọng số trong ma trận trọng số W2:
- Giá trị thay đổi cho trọng số thứ 1: $0.25 * (-0.643962658) * 0.634135591 * 0.643962658 * (1 - 0.643962658) = -0.023406638$
- Giá trị thay đổi cho trọng số thứ 2: $0.25 * (-0.643962658) * 0.457602059 * 0.643962658 * (1 - 0.643962658) = -0.016890593$
- Change weight 1: $0.35 + (-0.023406638) = 0.326593362$
- Change weight 2: $0.81 + (-0.016890593) = 0.793109407$

- Now we will change the weights in weight matrix 1:
- Value for changing weight 1: $0.25 * (-0.643962658) * 0 * 0.634135591 * (1 - 0.634135591) = 0$
- Value for changing weight 2: $0.25 * (-0.643962658) * 0 * 0.457602059 * (1 - 0.457602059) = 0$
- Value for changing weight 3: $0.25 * (-0.643962658) * 1 * 0.634135591 * (1 - 0.634135591) = -0.037351064$
- Value for changing weight 4: $0.25 * (-0.643962658) * 1$
- $* 0.457602059 * (1 - 0.457602059) = -0.039958271$
- Change weight 1: $0.62 + 0 = 0.62$ (not changed)
- Change weight 2: $0.42 + 0 = 0.42$ (not changed)
- Change weight 3: $0.55 + (-0.037351064) = 0.512648936$
- Change weight 4: $-0.17 + (-0.039958271) = -0.209958271$

"What happens, if all values of an input pattern are zero?"

- If all values of an input pattern are zero, the weights in weight matrix 1 would never be changed for this pattern and the net could not learn it. Due to that fact, a "pseudo input" is created, called Bias that has a constant output value of 1.



Bài tập

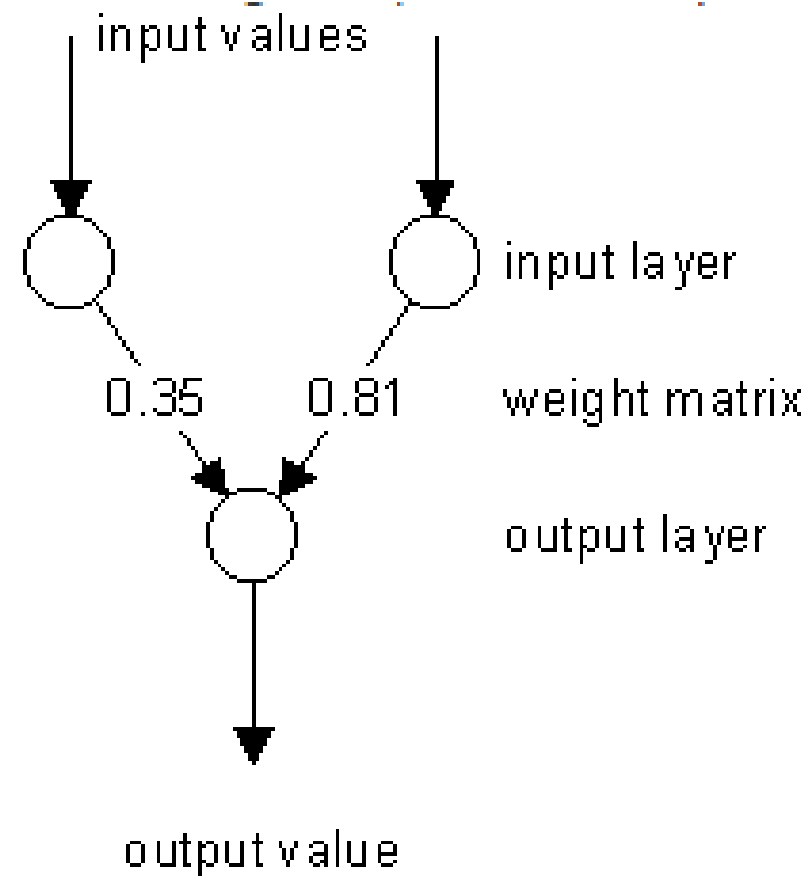
$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

$$\Delta w_{ij}(k) = \eta[d_i(k) - y_i(k)]x_j(k)$$

- Cho mạng nơ ron có 2 lớp: lớp vào 2 nơ ron, lớp ngõ ra 1 nơ ron. Hàm ngõ ra là hàm cộng. Ma trận trọng số $w = [0.35 \ 0.81]$; tốc độ huấn luyện là 0.25.
- A. Vẽ mô hình mạng nơ ron
- B. Tính giá trị ngõ ra (lan truyền thuận) khi ngõ vào là $[0 \ 1]$ và $[1 \ 1]$;
- C. Thực hiện lan truyền ngược và tính bộ trọng số hiệu chỉnh mới khi ngõ vào là $[0 \ 1]$ và $[1 \ 1]$; ngõ ra là $[0 \ 1]$. Giả sử ma trận trọng số ban đầu là $w = [0.35 \ 0.81]$.

Suppose you have the following 2-layered Perceptron:

- Huấn luyện 1 chu kỳ mạng sau:



- Patterns to be learned:

- input target

- 0 1 0

- 1 1 1

- First, the weight values are set to random values (0.35 and 0.81).
- The learning rate of the net is set to 0.25.
- Next, the values of the first input pattern (0 1) are set to the neurons of the input layer (the output of the input layer is the same as its input).

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$

$$\Delta w_{ij}(k) = \eta[d_i(k) - y_i(k)]x_j(k)$$

- The neurons in the following layer (only one neuron in the output layer) are activated:
- Ngõ vào 1 của nơ ron ngõ ra: $0 * 0.35 = 0$
- Ngõ vào 2 của nơ ron ngõ ra : $1 * 0.81 = 0.81$
- Giá trị ngõ ra: $0 + 0.81 = 0.81$ (= ngõ ra)
- Tính sai số ngõ ra bằng cách trừ ngõ ra thực với ngõ ra mong muốn: $0 - 0.81 = -0.81$
- Giá trị thay đổi trọng số thứ 1: $0.25 * 0 * (-0.81) = 0$ (0.25 = tốc độ học)
- Giá trị thay đổi trọng số thứ 2: $0.25 * 1 * (-0.81) = -0.2025$
- Giá trị trọng số 1: $0.35 + 0 = 0.35$ (not changed)
- Giá trị trọng số 2: $0.81 + (-0.2025) = 0.6075$

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k)$$
$$\Delta w_{ij}(k) = \eta[d_i(k) - y_i(k)]x_j(k)$$

- Now that the weights are changed, the second input pattern (1 1) is set to the input layer's neurons and the activation of the output neuron is performed again, now with **the new weight values**:
- Ngõ vào 1 của nơ ron ngõ ra: $1 * 0.35 = 0.35$
- Ngõ vào 2 của nơ ron ngõ ra : $1 * 0.6075 = 0.6075$
- Giá trị ngõ ra : $0.35 + 0.6075 = 0.9575$ (= output)
- Compute an error value by subtracting output from target: $1 - 0.9575 = 0.0425$
- Value for changing weight 1: $0.25 * 1 * 0.0425 = 0.010625$
- Value for changing weight 2: $0.25 * 1 * 0.0425 = 0.010625$
- Change weight 1: $0.35 + 0.010625 = 0.360625$
- Change weight 2: $0.6075 + 0.010625 = 0.618125$

- That was one learning step. Each input pattern had been propagated through the net and the weight values were changed.
- The error of the net can now be calculated by adding up the squared values of the output errors of each pattern:
- Compute the net error: $(-0.81)^2 + (0.0425)^2 = 0.65790625$
- By performing this procedure repeatedly, this error value gets smaller and smaller.
- The algorithm is successfully finished, if the net error is zero (perfect) or approximately zero.

Exercise xx

- Given a neural network have 2 input, 3 neural in hidden, 1 output.
The initial weight matrix $w1 = [0.8 \ 0.4 \ 0.3; 0.2 \ 0.9 \ 0.5]$, $w2 = [0.3 \ 0.5 \ 0.9]$. The activation function is sigmoid
- a. Calculate the output of network when input is $[1 \ 1]$
- b. Training the weight of networks using the following data set.
Learning speed = 1

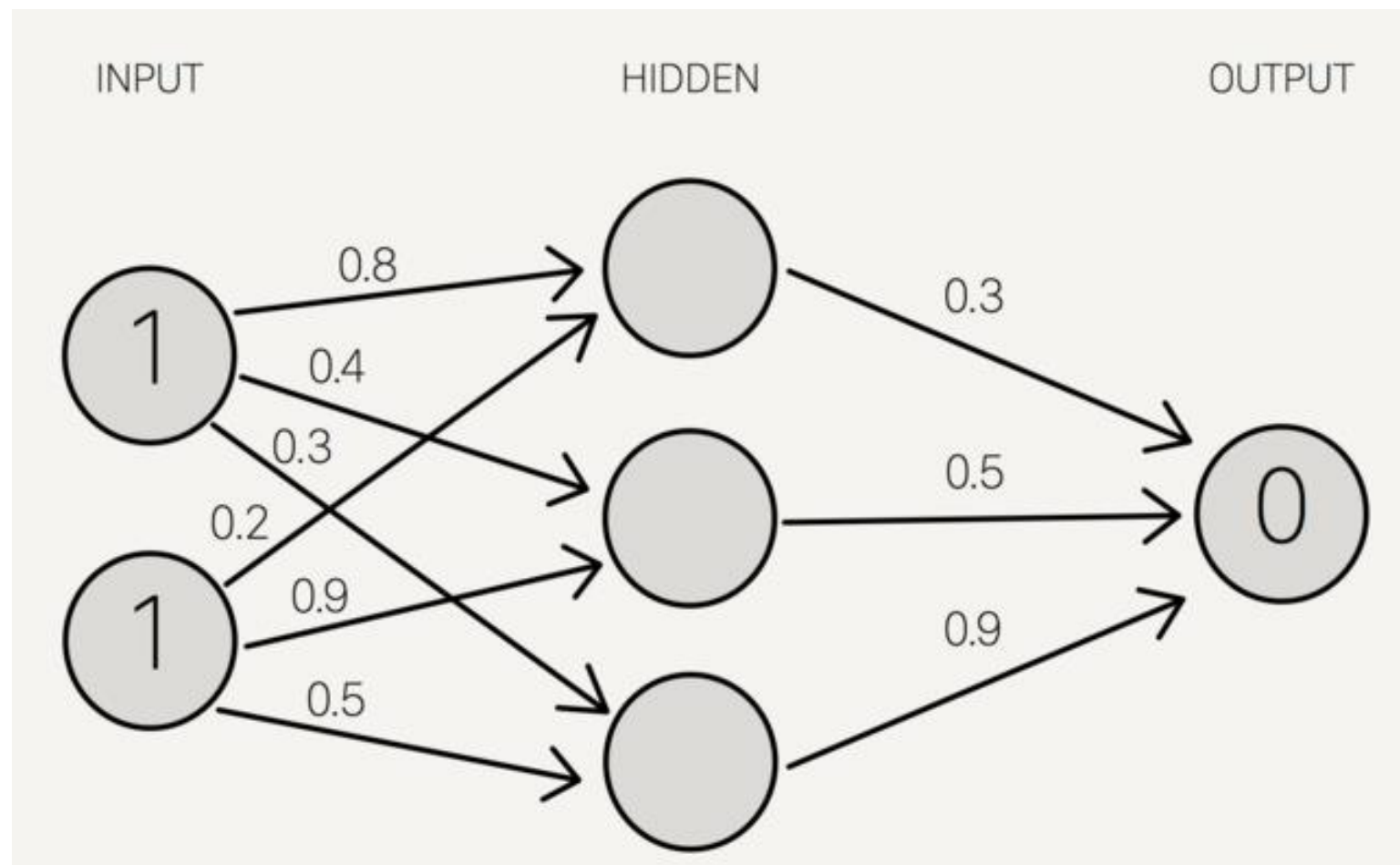
Input | output

0, 0 | 0

0, 1 | 1

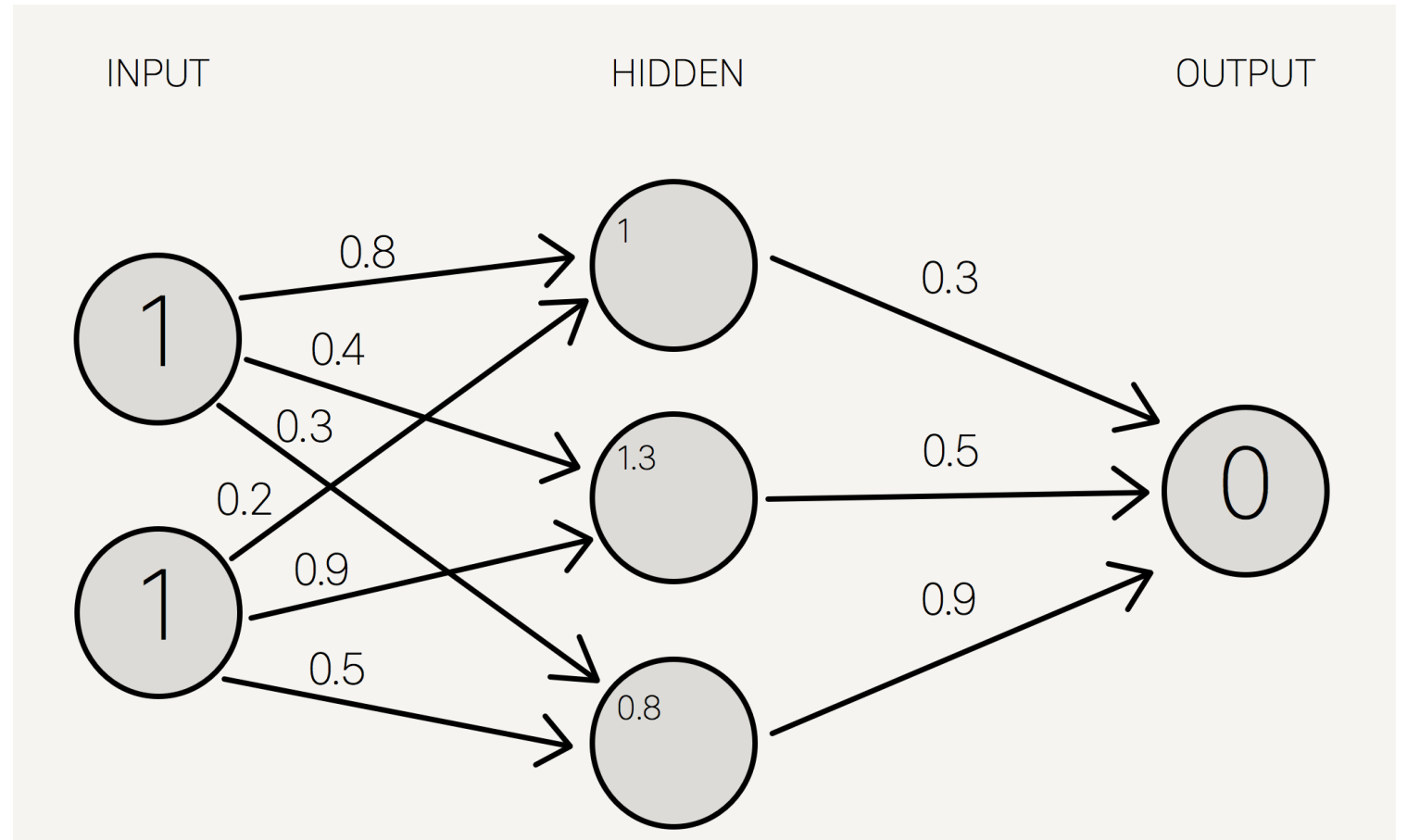
1, 0 | 1

1, 1 | 0

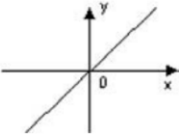
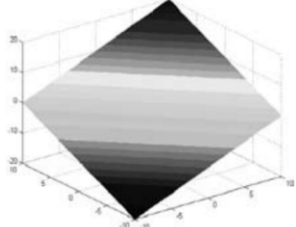
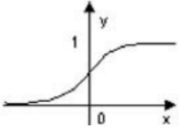
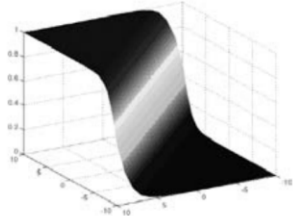
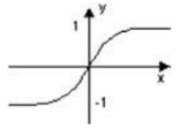
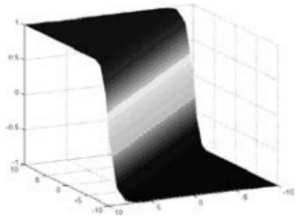


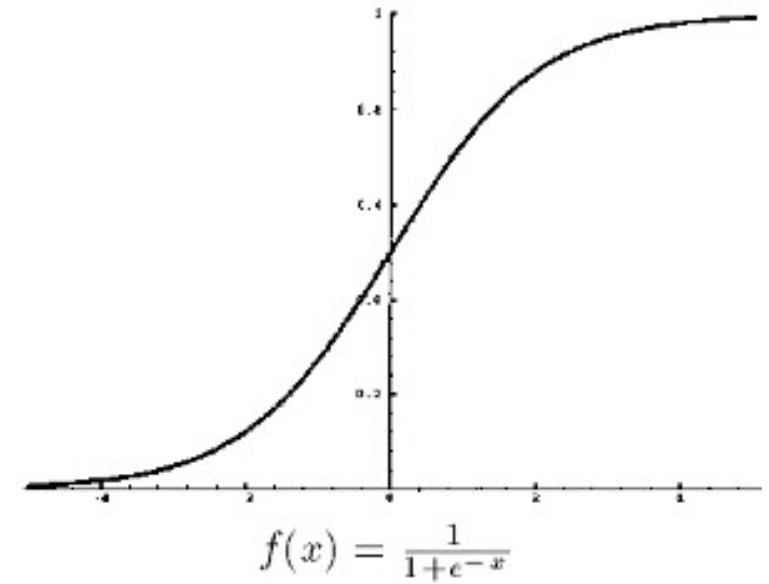
Input of hidden layer

- $1 * 0.8 + 1 * 0.2 = 1$
- $1 * 0.4 + 1 * 0.9 = 1.3$
- $1 * 0.3 + 1 * 0.5 = 0.8$



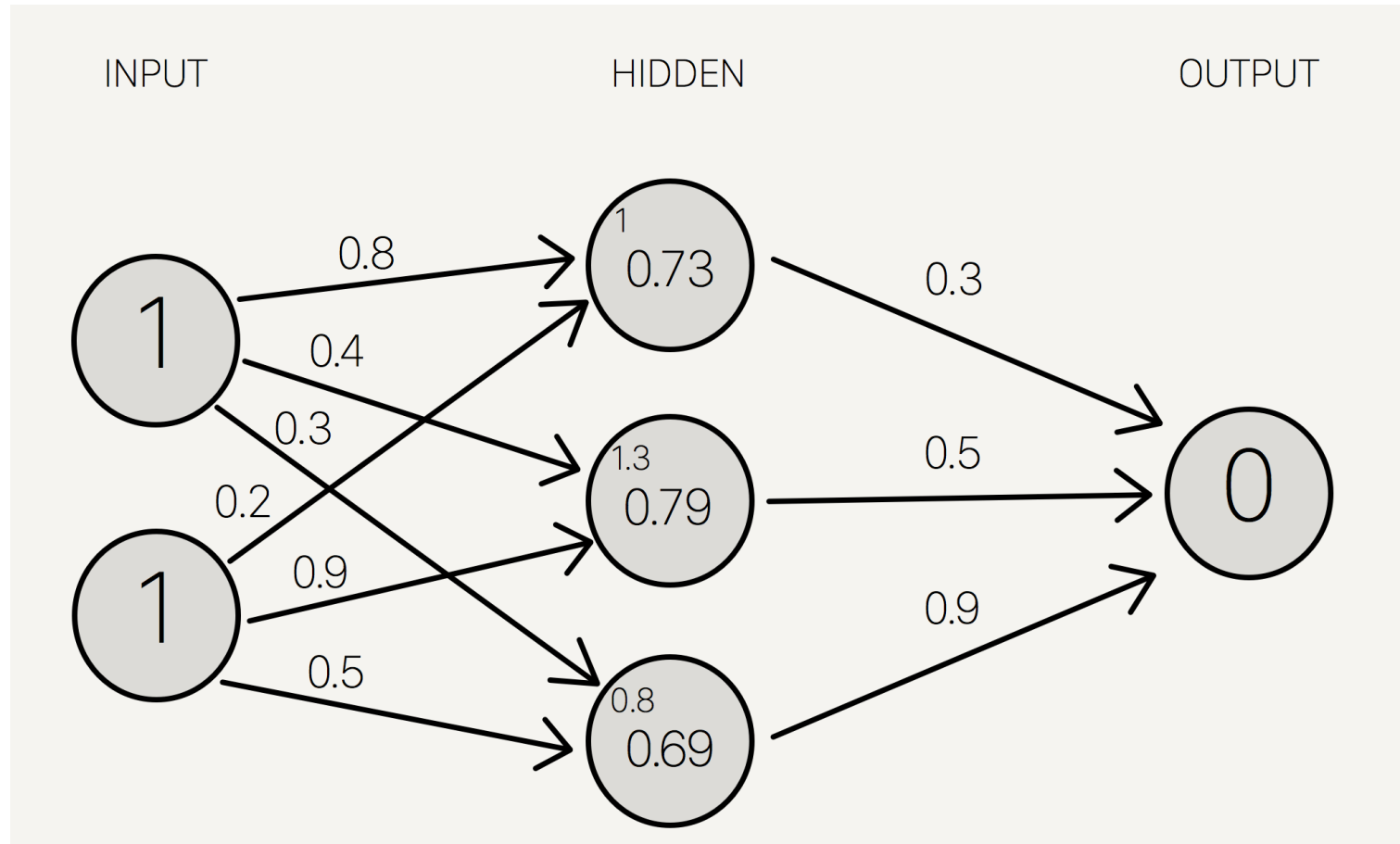
Activation functions

Activation Function	Mathematical Equation	2D Graphical Representation	3D Graphical Representation
Linear	$y = x$		
Sigmoid (logistic)	$y = \frac{1}{1 + e^{-x}}$		
Hyperbolic tangent	$y = \frac{1 - e^{-2x}}{1 + e^{2x}}$		



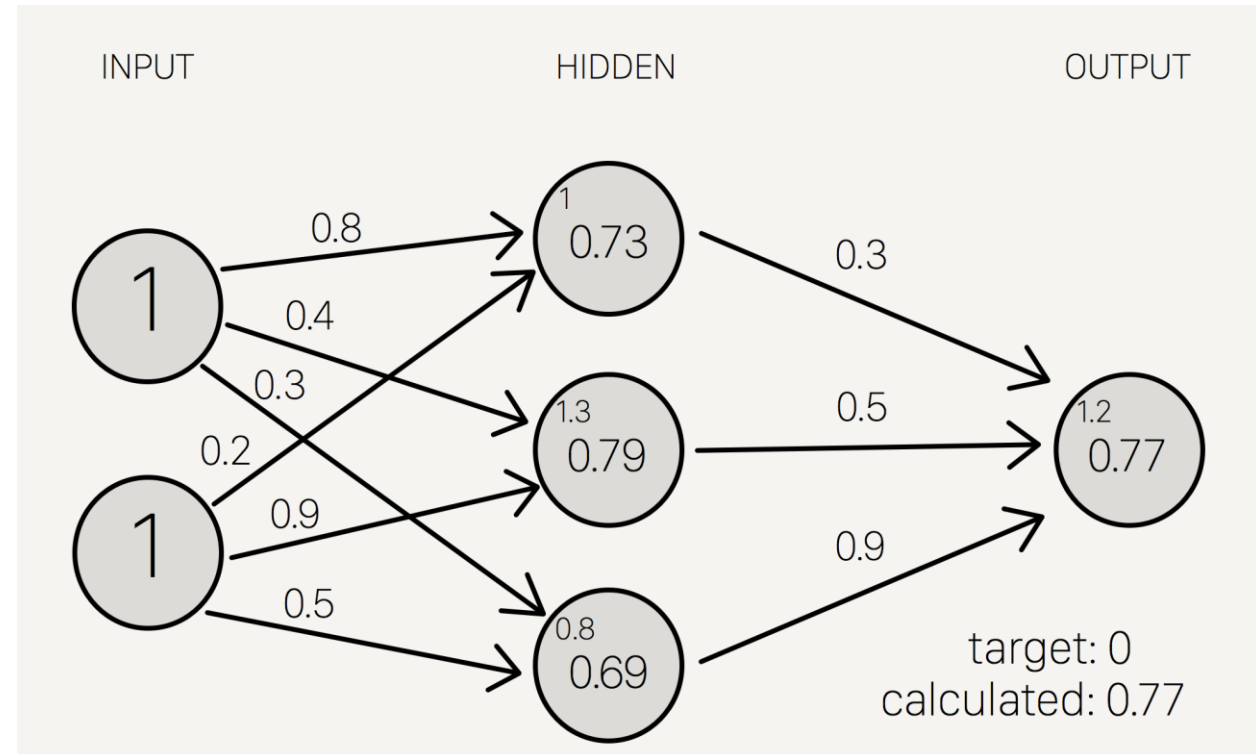
Output of hidden layer

- $S(1.0) = 0.73105857863$
- $S(1.3) = 0.78583498304$
- $S(0.8) = 0.68997448112$

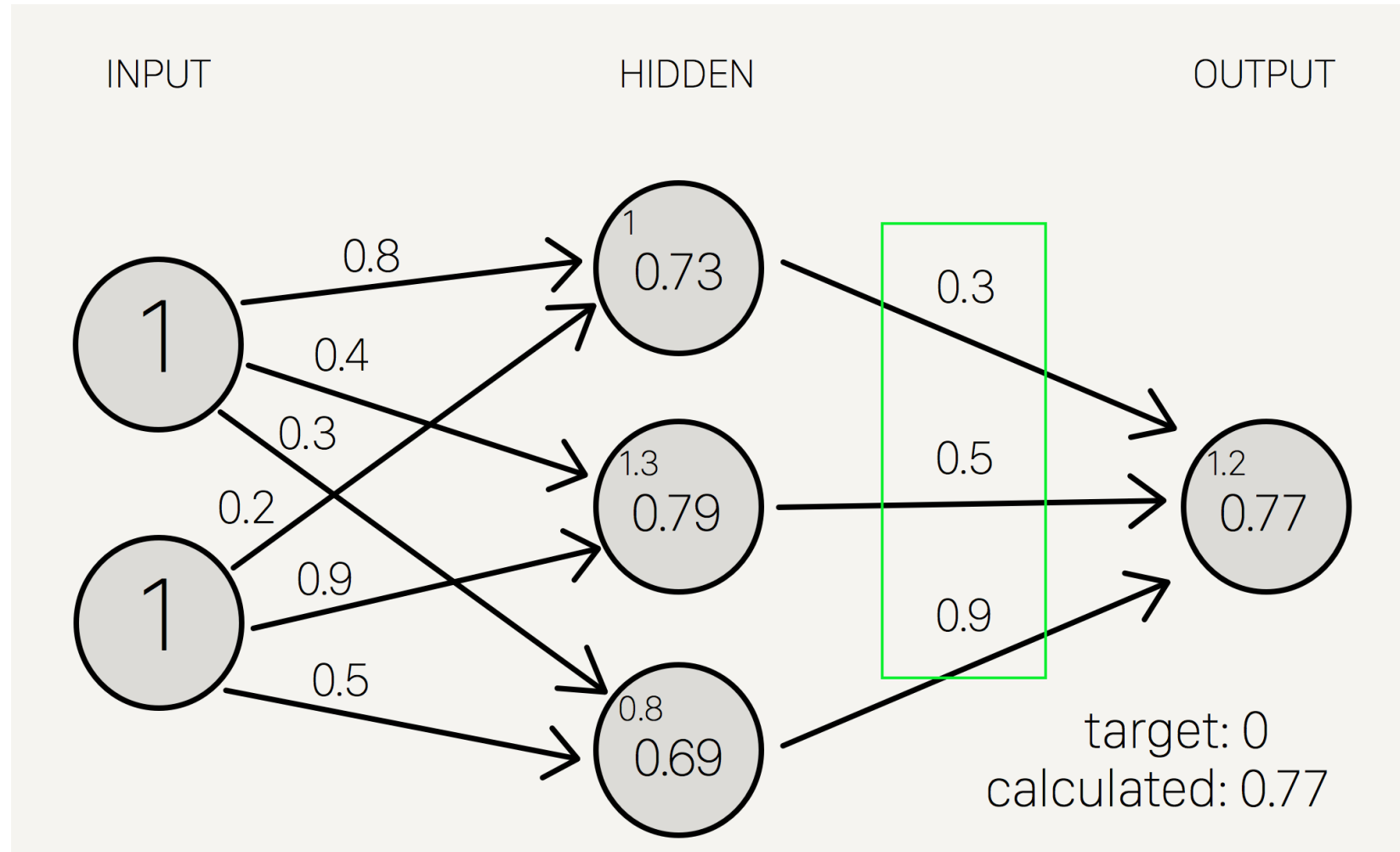


Input of output neural

- $0.73 * 0.3 + 0.79 * 0.5 + 0.69 * 0.9 = 1.235$
- Output of output neural
- $S(1.235) = 0.7746924929149283$



Back propagation training



Back propagation training

- Calculating the incremental change to these weights happens in two steps: 1) we find the margin of error of the output result (what we get after applying the activation function) to back out the necessary change in the output sum (we call this delta output sum) and
- 2) we extract the change in weights by multiplying delta output sum by the hidden layer results.
- The output sum margin of error is the target output result minus the calculated output result:

$$\text{Output sum margin of error} = \textit{target} - \textit{calculated}$$

Back propagation training

- Target = 0
- Calculated = 0.77
- Target - calculated = -0.77

2. Tế bào thần kinh và mạng thần kinh nhân tạo

5. Mạng truyền thẳng nhiều lớp.

5.3. Thuật toán lan truyền ngược :

Bước 4: (Lan truyền ngược sai số) Cập nhật trọng số của mạng:

Lớp ra:

$$\delta_{oi}(k) = [(d_i(k) - y_i(k))][a'_o(net_i(k))] \quad (i = 1, n)$$

$$w_{iq}(k+1) = w_{iq}(k) + \eta \delta_{oi}(k) z_q(k) \quad (i = 1, n; q = 1, l)$$

Lớp ẩn:

$$\delta_{hq}(k) = \left[\sum_{i=1}^n \delta_{oi}(k) w_{iq}(k) \right] [a'_h(net_q(k))] \quad (q = 1, l)$$

$$v_{qj}(k+1) = v_{qj}(k) + \eta \delta_{hq}(k) x_j(k) \quad (j = 1, m; q = 1, l)$$

Bước 5: Tính sai số tích lũy:

$$E = \frac{1}{2} \sum_{i=1}^n (d_i(k) - y_i(k))^2 + E$$

Equations

$$\delta_{O_k} = o_k(E)(1 - o_k(E))(t_k(E) - o_k(E))$$

$$\delta_{H_k} = h_k(E)(1 - h_k(E)) \sum_{i \in \text{outputs}} w_{ki} \delta_{O_i}$$

$$\Delta_{ij} = \eta \delta_{H_j} x_i$$

$$\Delta_{ij} = \eta \delta_{O_j} h_i(E)$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

- To calculate the necessary change in the output sum, or delta output sum, we take the derivative of the activation function and apply it to the output sum. In our example, the activation function is the sigmoid function.
- To refresh your memory, the activation function, sigmoid, takes the sum and returns the result:

$$S(\textit{sum}) = \textit{result}$$

- So the derivative of sigmoid, also known as sigmoid prime, will give us the rate of change (or “slope”) of the activation function at the output sum:

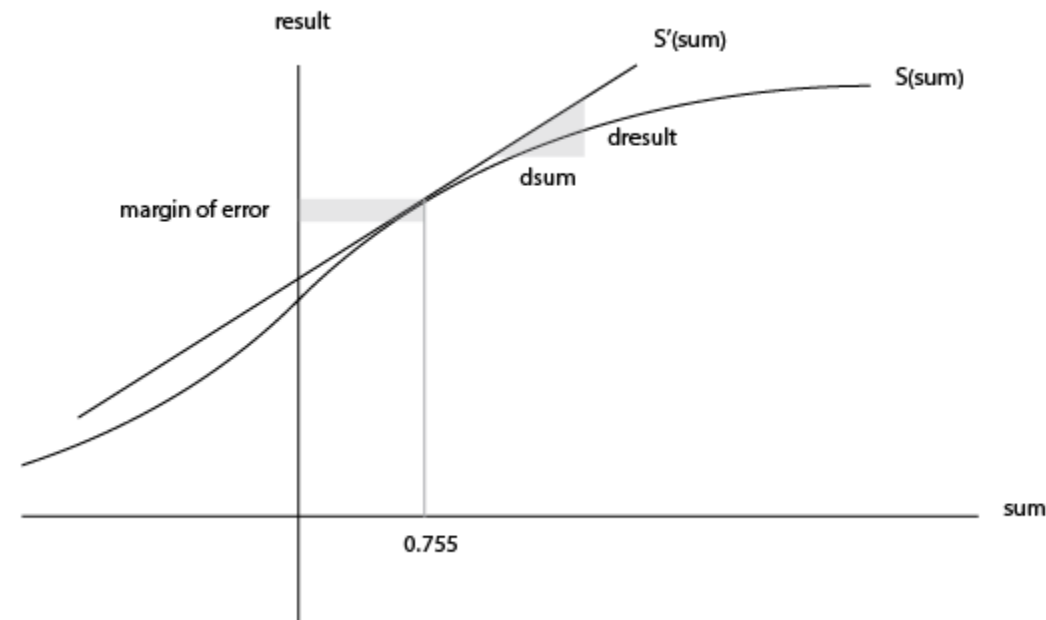
$$S'(sum) = \frac{dsum}{dresult}$$

- Since the output sum margin of error is the difference in the result, we can simply multiply that with the rate of change to give us the delta output sum:

$$\frac{dsum}{dresult} \times (\text{target result} - \text{calculated result}) = \Delta \text{sum}$$

- Conceptually, this means that the change in the output sum is the same as the sigmoid prime of the output result. Doing the actual math, we get:
- Delta output sum = $S'(\text{sum}) * (\text{output sum margin of error})$
- Delta output sum = $S'(1.235) * (-0.77)$
- Delta output sum = -0.13439890643886018

- Here is a graph of the Sigmoid function to give you an idea of how we are using the derivative to move the input towards the right direction. Note that this graph is not to scale.



- Now that we have the proposed change in the output layer sum (-0.13), let's use this in the derivative of the output sum function to determine the new change in weights.
- As a reminder, the mathematical definition of the output sum is the product of the hidden layer result and the weights between the hidden and output layer:

$$H_{result} \times w_{h \rightarrow o} = O_{sum}$$

- The derivative of the output sum is:

$$\frac{dO_{sum}}{dw_{h \rightarrow o}} = H_{results}$$

- ..which can also be represented as:

$$dw_{h \rightarrow o} = \frac{dO_{sum}}{H_{results}}$$

- This relationship suggests that a greater change in output sum yields a greater change in the weights; input neurons with the biggest contribution (higher weight to output neuron) should experience more change in the connecting synapse.
- Let's do the math:

- hidden result 1 = 0.73105857863
 - hidden result 2 = 0.78583498304
 - hidden result 3 = 0.68997448112
-
- Delta weights = delta output sum / hidden layer results
 - Delta weights = -0.1344 / [0.73105, 0.78583, 0.69997]
 - Delta weights = [-0.1838, -0.1710, -0.1920]
-
- old w7 = 0.3
 - old w8 = 0.5
 - old w9 = 0.9
-
- new w7 = 0.1162
 - new w8 = 0.329
 - new w9 = 0.708

- To determine the change in the weights between the input and hidden layers, we perform the similar, but notably different, set of calculations. Note that in the following calculations, we use the initial weights instead of the recently adjusted weights from the first part of the backward propagation.
- Remember that the relationship between the hidden result, the weights between the hidden and output layer, and the output sum is:

$$H_{result} \times w_{h \rightarrow o} = O_{sum}$$

- Instead of deriving for output sum, let's derive for hidden result as a function of output sum to ultimately find out delta hidden sum:

$$\frac{dH_{result}}{dO_{sum}} = \frac{1}{w_{h \rightarrow o}}$$

$$dH_{result} = \frac{dO_{sum}}{w_{h \rightarrow o}}$$

- Also, remember that the change in the hidden result can also be defined as:

$$S'(H_{sum}) = \frac{dH_{sum}}{dH_{result}}$$

- Let's multiply both sides by sigmoid prime of the hidden sum:

$$dH_{result} \times \frac{dH_{sum}}{dH_{result}} = \frac{dO_{sum}}{w_{h \rightarrow o}} \times \frac{dH_{sum}}{dH_{result}}$$

$$dH_{sum} = \frac{dO_{sum}}{w_{h \rightarrow o}} \times S'(H_{sum})$$

- All of the pieces in the above equation can be calculated, so we can determine the delta hidden sum:
- Delta hidden sum = delta output sum / hidden-to-outer weights * $S'(\text{hidden sum})$
- Delta hidden sum = $-0.1344 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])$
- Delta hidden sum = $[-0.448, -0.2688, -0.1493] * [0.1966, 0.1683, 0.2139]$
- Delta hidden sum = $[-0.088, -0.0452, -0.0319]$

- Once we get the delta hidden sum, we calculate the change in weights between the input and hidden layer by dividing it with the input data, (1, 1). The input data here is equivalent to the hidden results in the earlier back propagation process to determine the change in the hidden-to-output weights. Here is the derivation of that relationship, similar to the one before:

$$I \times w_{i \rightarrow h} = H_{sum}$$

$$\frac{dH_{sum}}{dw_{i \rightarrow h}} = I$$

$$dw_{i \rightarrow h} = \frac{dH_{sum}}{I}$$

- Let's do the math:
- input 1 = 1
- input 2 = 1

- Delta weights = delta hidden sum / input data
- Delta weights = $[-0.088, -0.0452, -0.0319] / [1, 1]$
- Delta weights = $[-0.088, -0.0452, -0.0319, -0.088, -0.0452, -0.0319]$

- old $w_1 = 0.8$
- old $w_2 = 0.4$
- old $w_3 = 0.3$
- old $w_4 = 0.2$
- old $w_5 = 0.9$
- old $w_6 = 0.5$

- new $w_1 = 0.712$
- new $w_2 = 0.3548$
- new $w_3 = 0.2681$
- new $w_4 = 0.112$
- new $w_5 = 0.8548$
- new $w_6 = 0.4681$

- Here are the new weights, right next to the initial random starting weights as comparison:

- old new

- -----

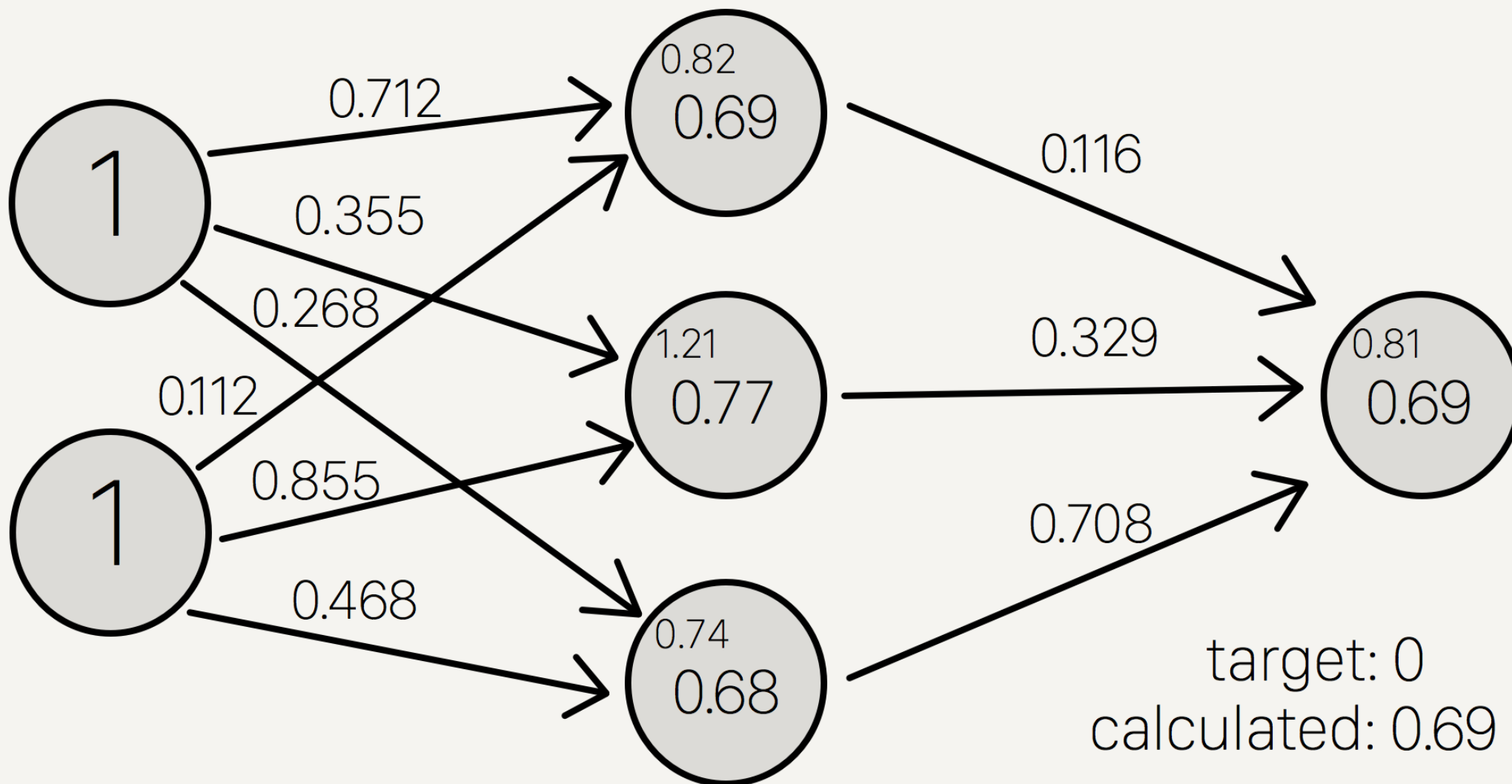
- w1: 0.8 w1: 0.712
- w2: 0.4 w2: 0.3548
- w3: 0.3 w3: 0.2681
- w4: 0.2 w4: 0.112
- w5: 0.9 w5: 0.8548
- w6: 0.5 w6: 0.4681
- w7: 0.3 w7: 0.1162
- w8: 0.5 w8: 0.329
- w9: 0.9 w9: 0.708

- Once we arrive at the adjusted weights, we start again with forward propagation. When training a neural network, it is common to repeat both these processes thousands of times (by default, Mind iterates 10,000 times).
- And doing a quick forward propagation, we can see that the final output here is a little closer to the expected output:

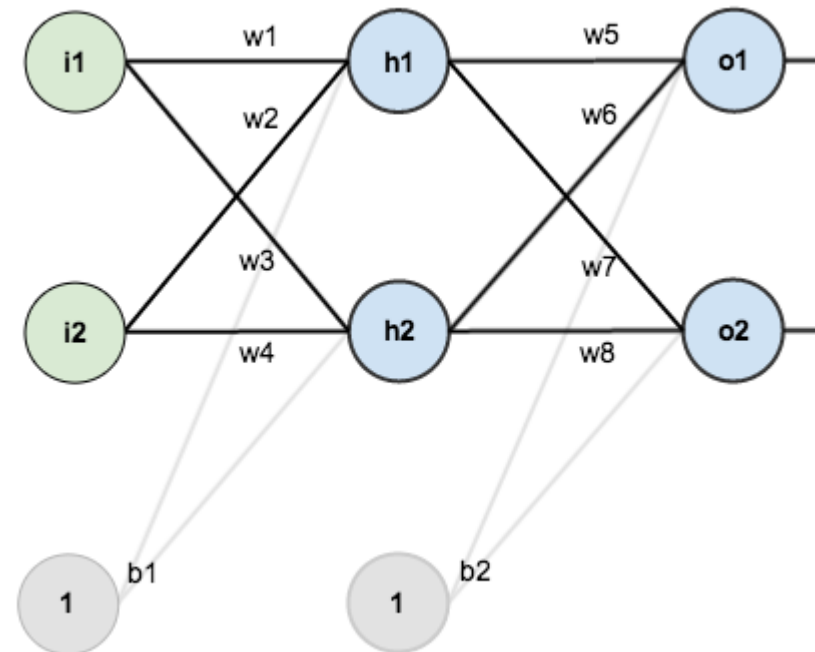
INPUT

HIDDEN

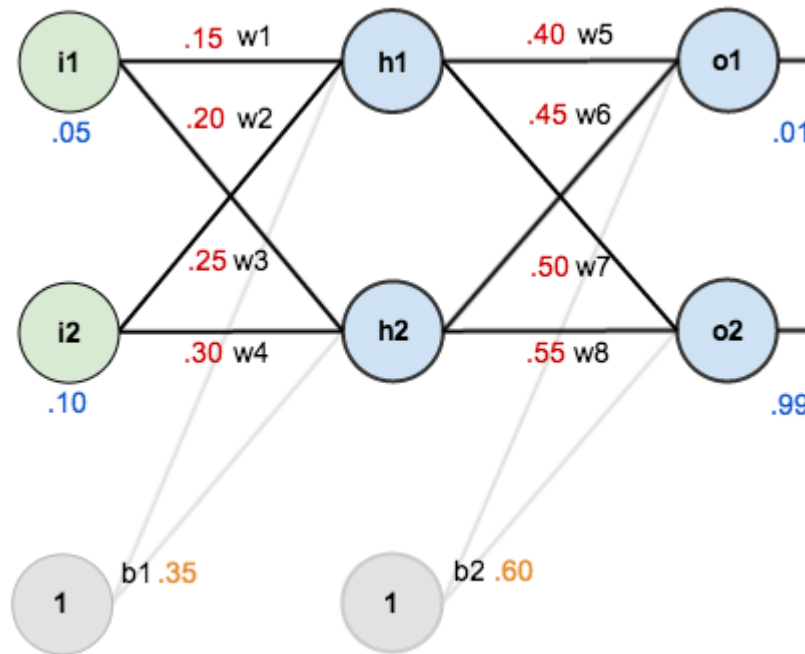
OUTPUT



- For this tutorial, we're going to use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.
- Here's the basic structure:



- In order to have some numbers to work with, here's are the initial weights, the biases, and training inputs/outputs:



- The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.
- For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

- **The Forward Pass**

- To begin, let's see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward through the network.
- We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.