Neural Network I

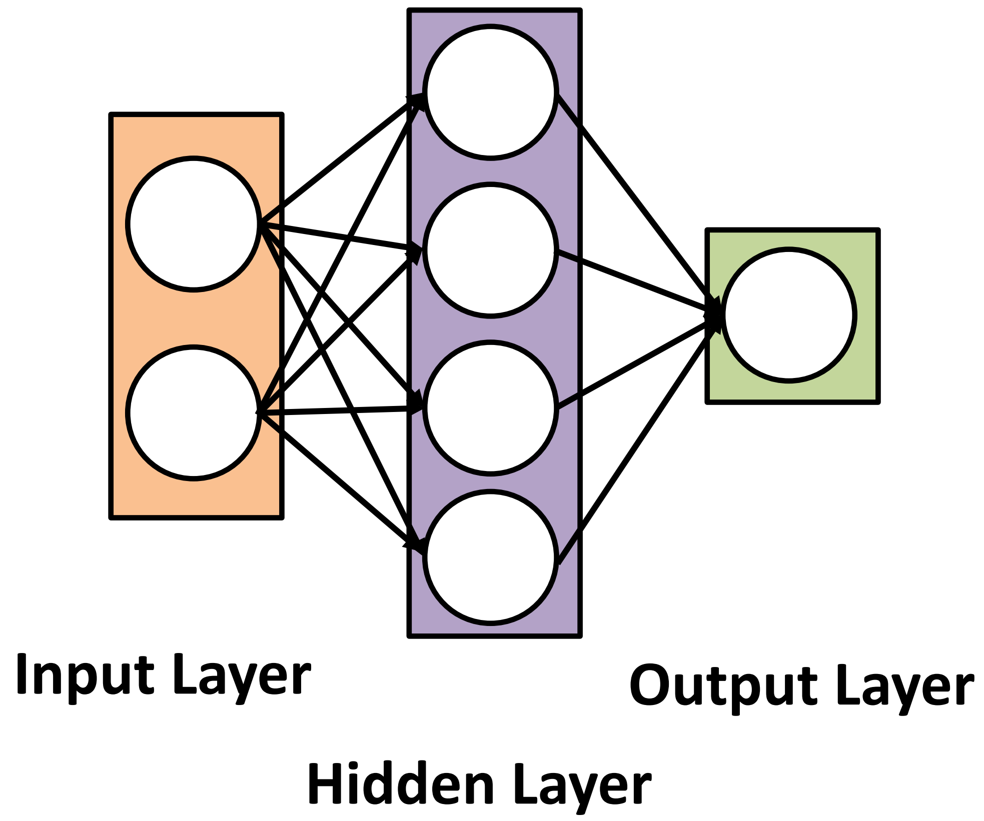
Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

CIVE 497 – CIVE 700: Smart Structure Technology

Last updated: 2021-04-05

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

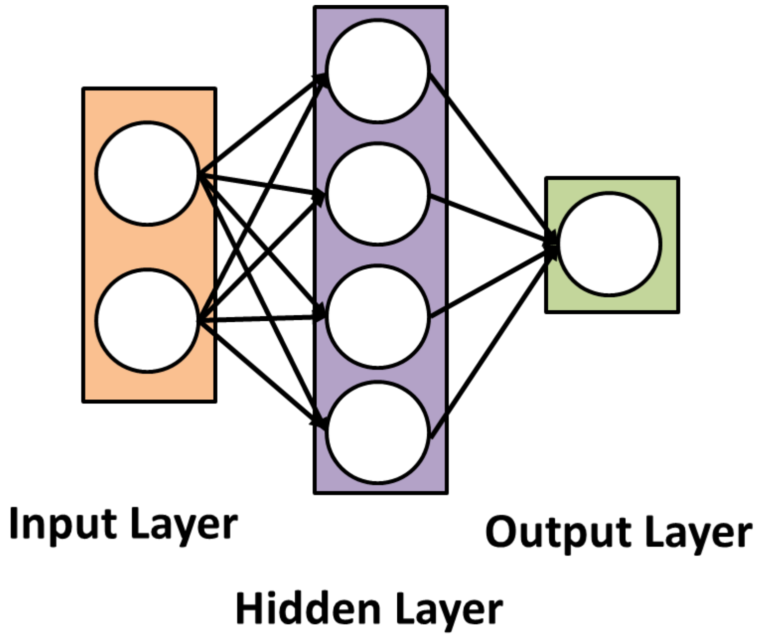
What's a Neural Network



- An input layer $\boldsymbol{x}$
- An arbitrary amount of hidden layer**<u>(s)</u>**
- An output layer, $\hat{\boldsymbol{y}}$
- A set of weights and biases between each layers, $\boldsymbol{W}$ and $\boldsymbol{b}$
- A choice of activation function for nodes in hidden layers

<u>Goal: find the best set of weights and biases that minimizes the loss (cost) function.</u>

What’s a Neural Network (Continue)



Cost function:

$$J(\theta) = J(\theta^1, \theta^2) = \sum_{i=1}^{n} (y_i - \theta^1 x_i - \theta^2)^2$$

Data (measurement): $(x_1, y_1), \ldots, (x_n, y_n)$

Model: Line ($y_i = m x_i + b$)

Task: Find (m, b)

Minimize $\boxed{E = J(m,b) = \sum_{i=1}^{n} (y_i - m x_i - b)^2}$

Derivatives:

$$\frac{\partial J(\theta^1, \theta^2)}{\partial \theta^1} = -2 \sum_{i=1}^{n} [y_i - \theta^1 x_i - \theta^2] x_i$$

$$\frac{\partial J(\theta^1, \theta^2)}{\partial \theta^2} = -2 \sum_{i=1}^{n} [y_i - \theta^1 x_i - \theta^2]$$

Updated rules:

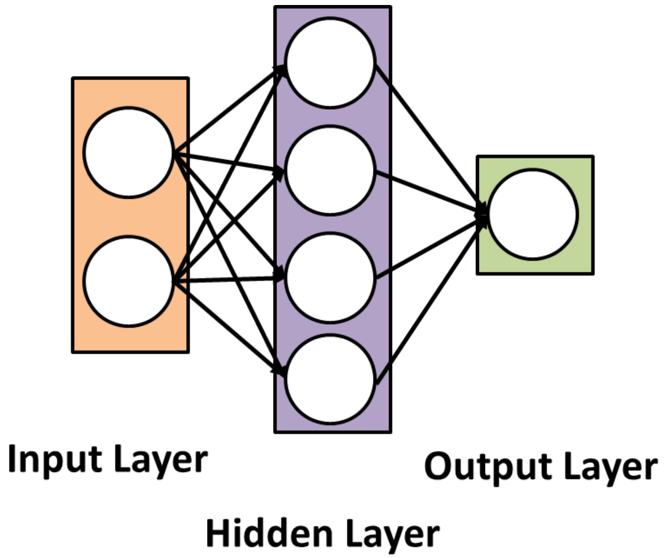
$$\theta^1_{j+1} \leftarrow \theta^1_j - \alpha \frac{\partial}{\partial \theta^1_j} J(\theta)$$

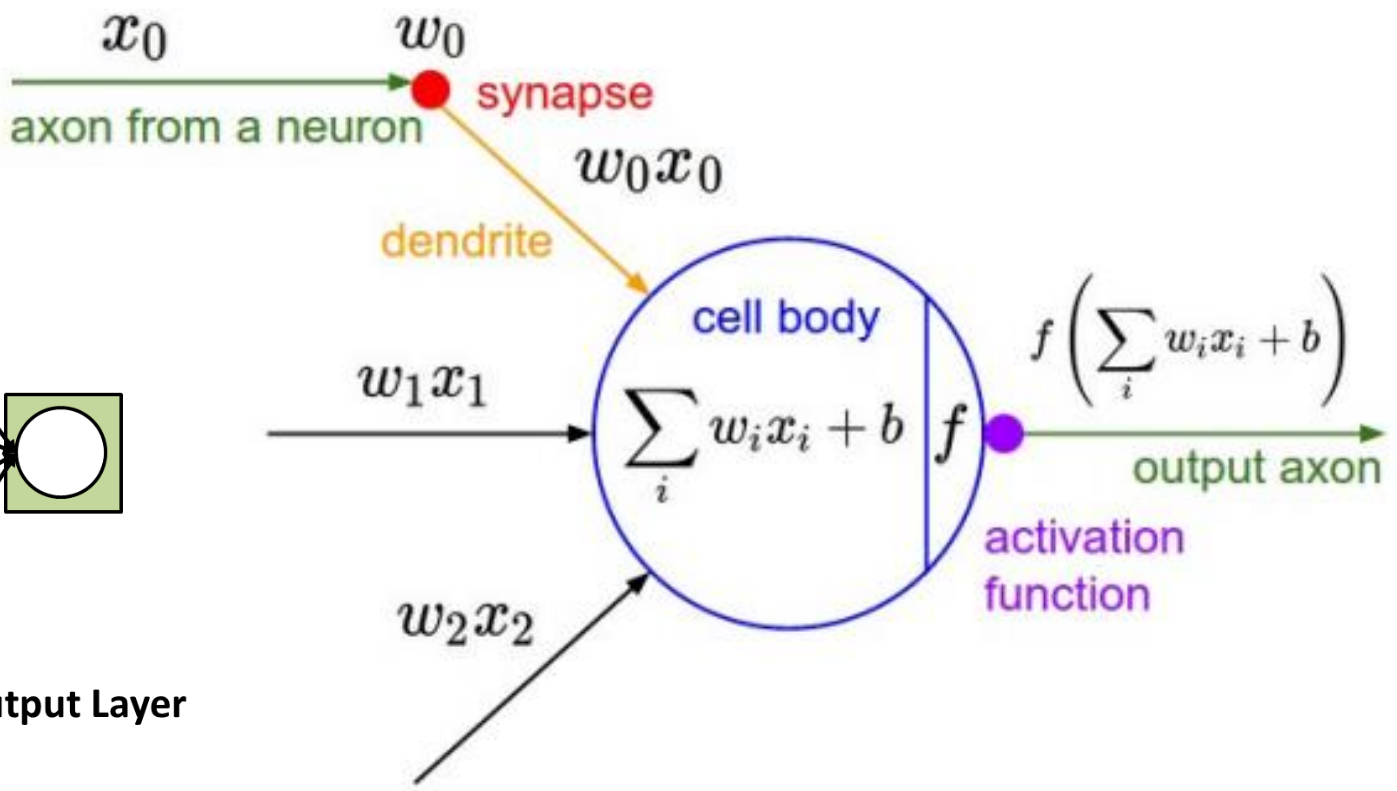
$$\theta^2_{j+1} \leftarrow \theta^2_j - \alpha \frac{\partial}{\partial \theta^2_j} J(\theta)$$

- Calculating the predicted output y, known as **feedforward**
- Updating the weights and biases, known as **backpropagation**

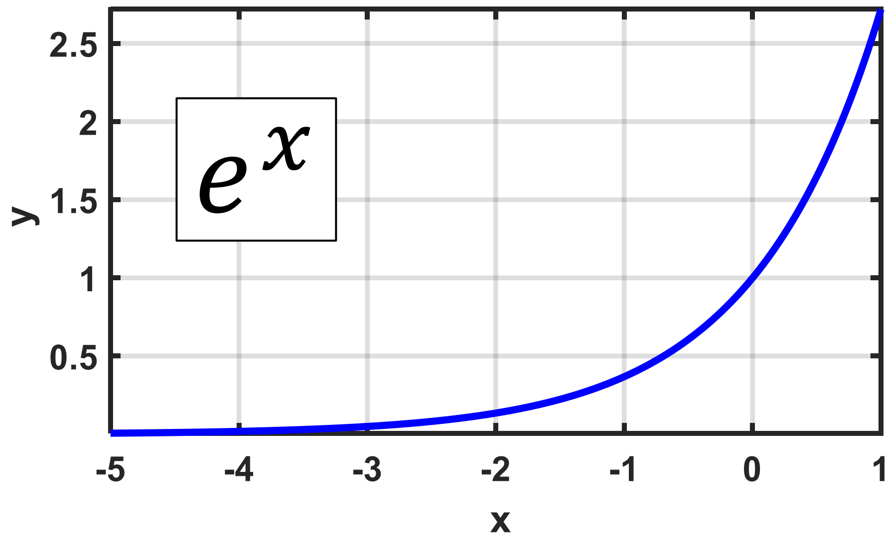
Mathematical Model of a Neuron

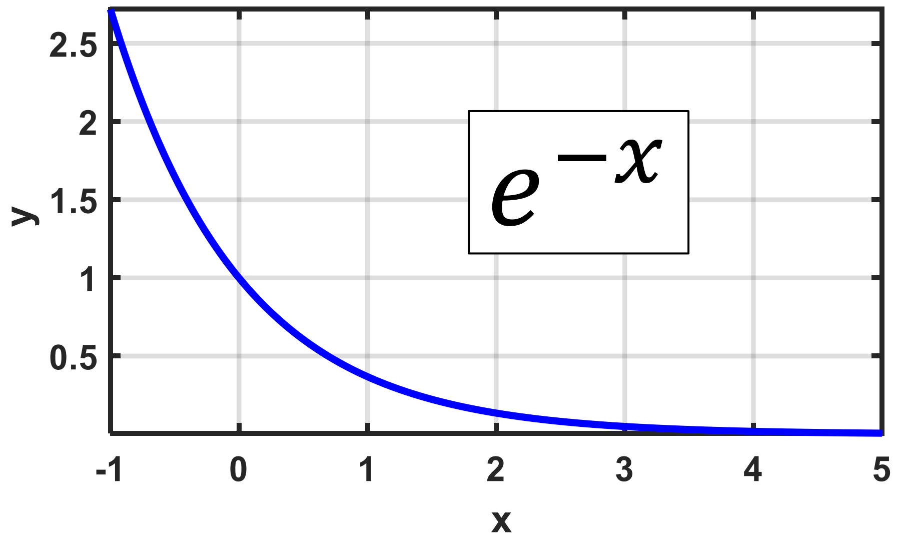
- Neural networks define functions of the inputs (hidden features), computed by neurons
- Artificial neurons are called units

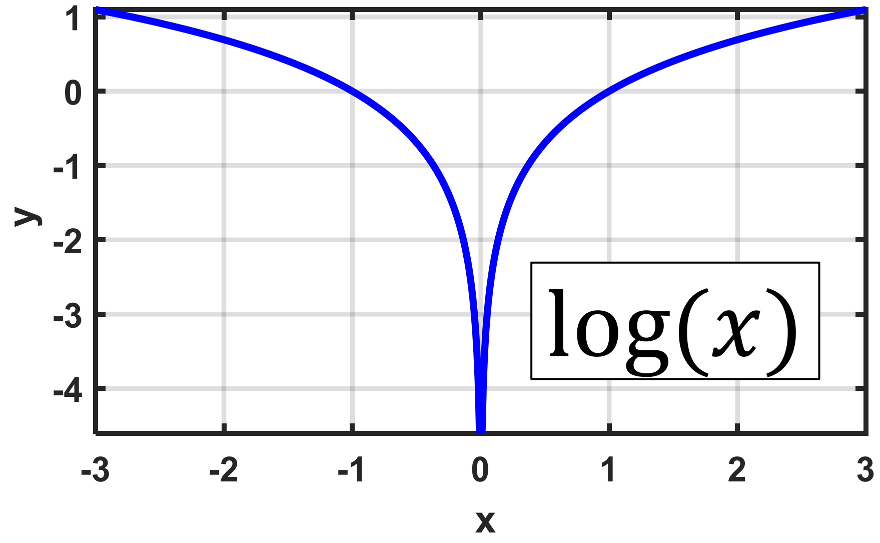


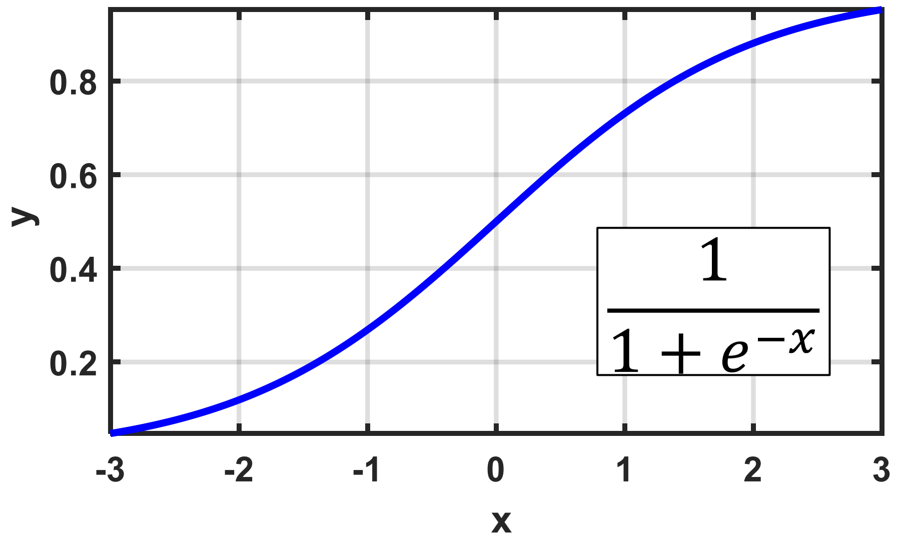


Some Functions



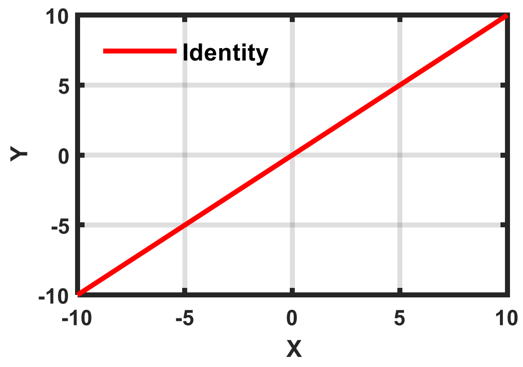


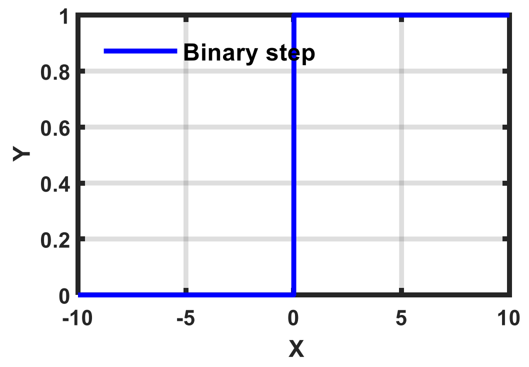


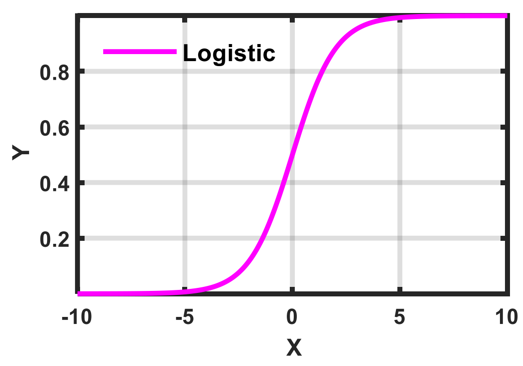


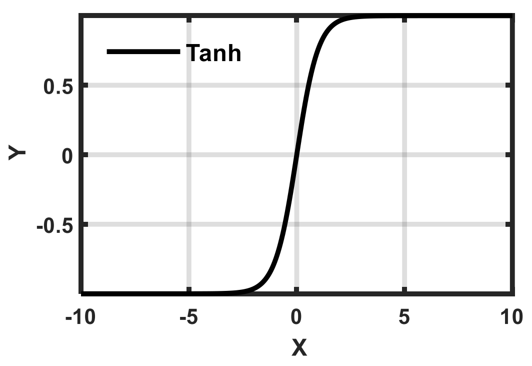
Activation Functions

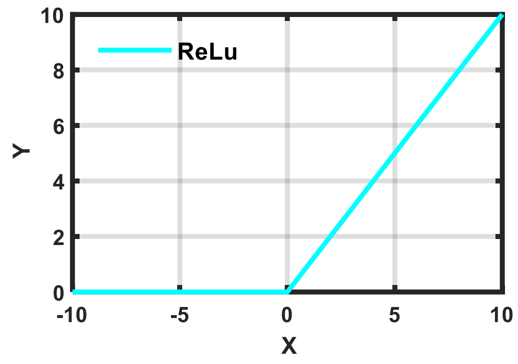
Name	Equation	Derivative
Identity	$f(x) = x$	$f'(x) = 1$
Binary step	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f(x) = \begin{cases} 0 & \text{for } x = 0 \\ \infty & \text{for } x \neq 0 \end{cases}$
Logistic (sigmoid)	$f(x) = \dfrac{1}{1 + e^{-x}}$	$f(x) = f(x)(1 - f(x))$
Tanh	$f(x) = \dfrac{2}{1 + e^{-2x}} - 1$	$f(x) = 1 - f(x)^2$
ReLu	$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
PReLu	$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

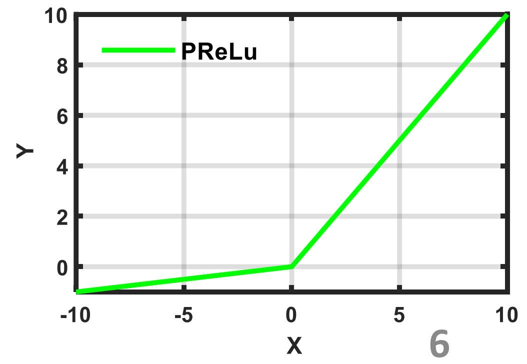




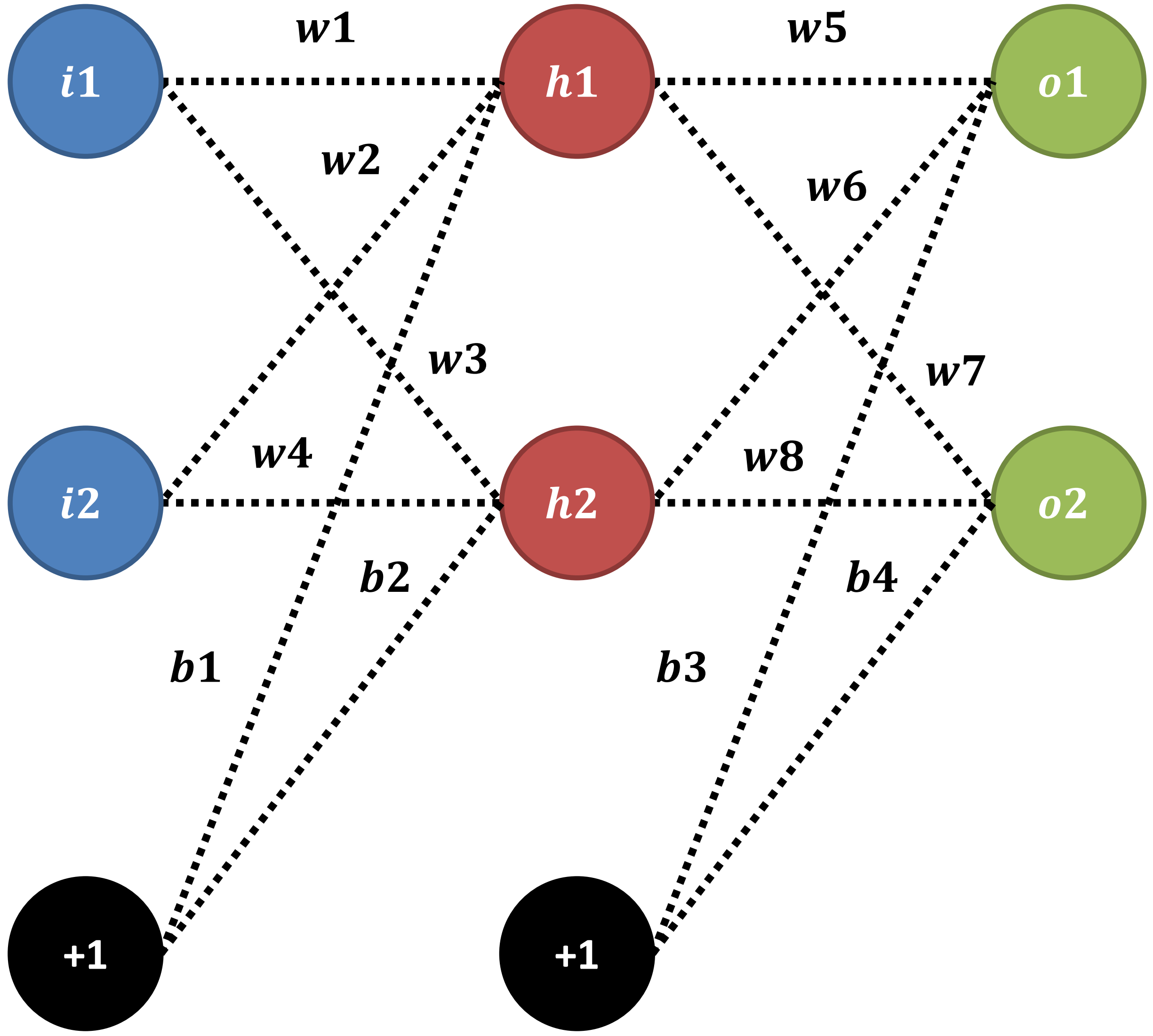








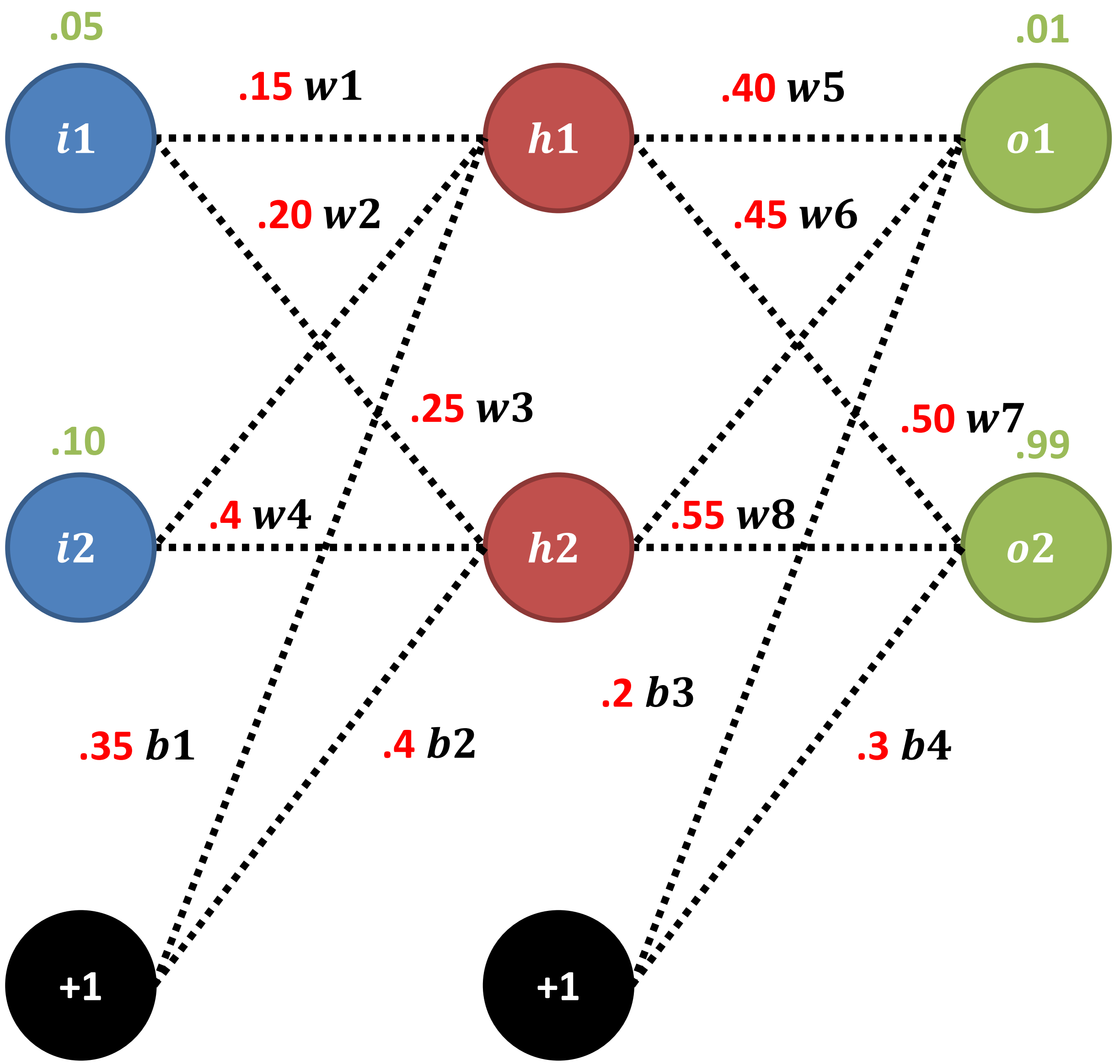
A Step-by-Step Forward and Backward Propagation



- One hidden layer (two hidden neurons)
- Two input neuron
- Two output neuron
- Activation function: Logistic function
- Developing $f: \mathbb{R}^2 \to \mathbb{R}^2$

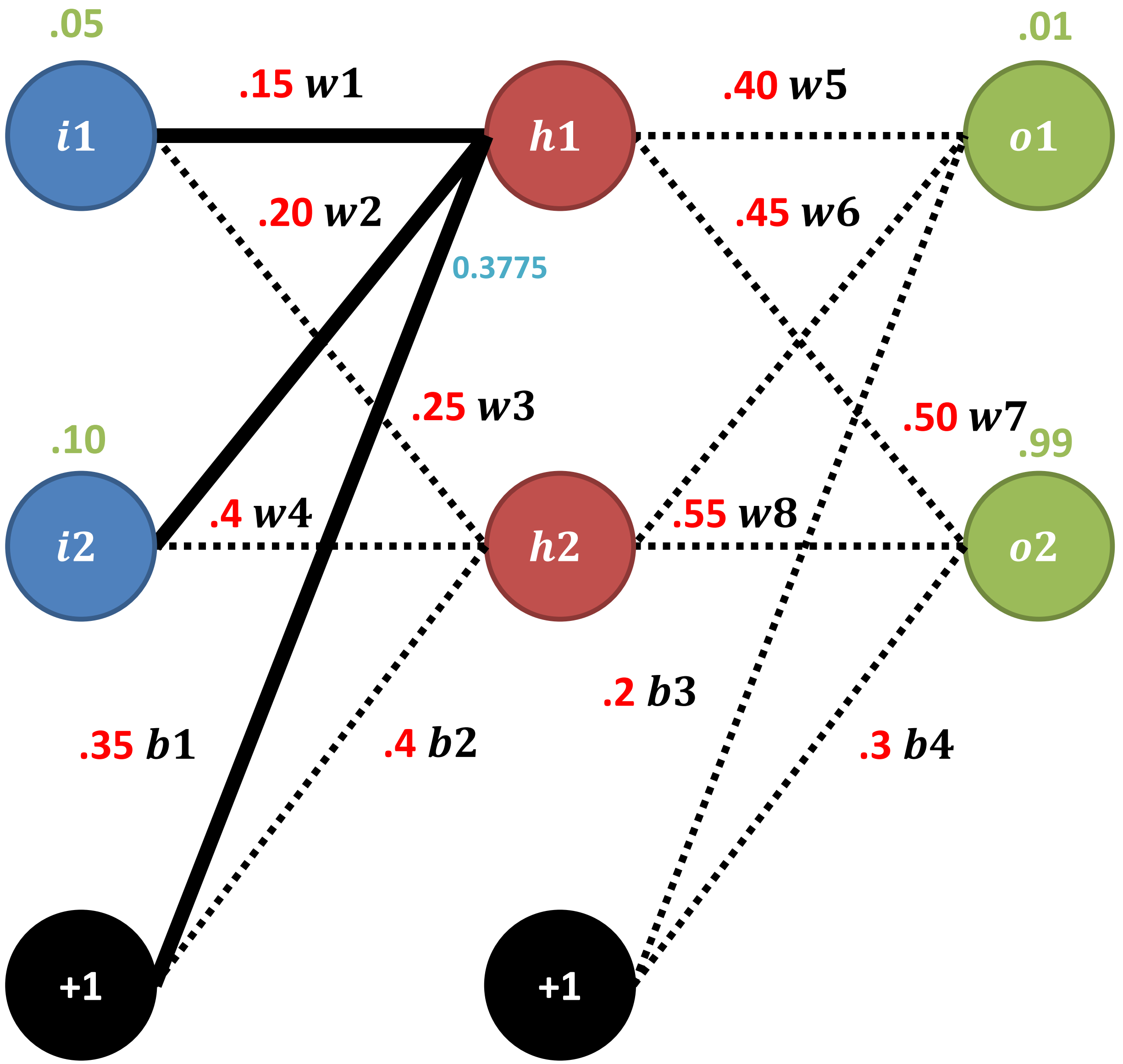
$$i_1, i_2 \to (o_1, o_2)$$

A Step-by-Step Forward and Backward Propagation (Continue)



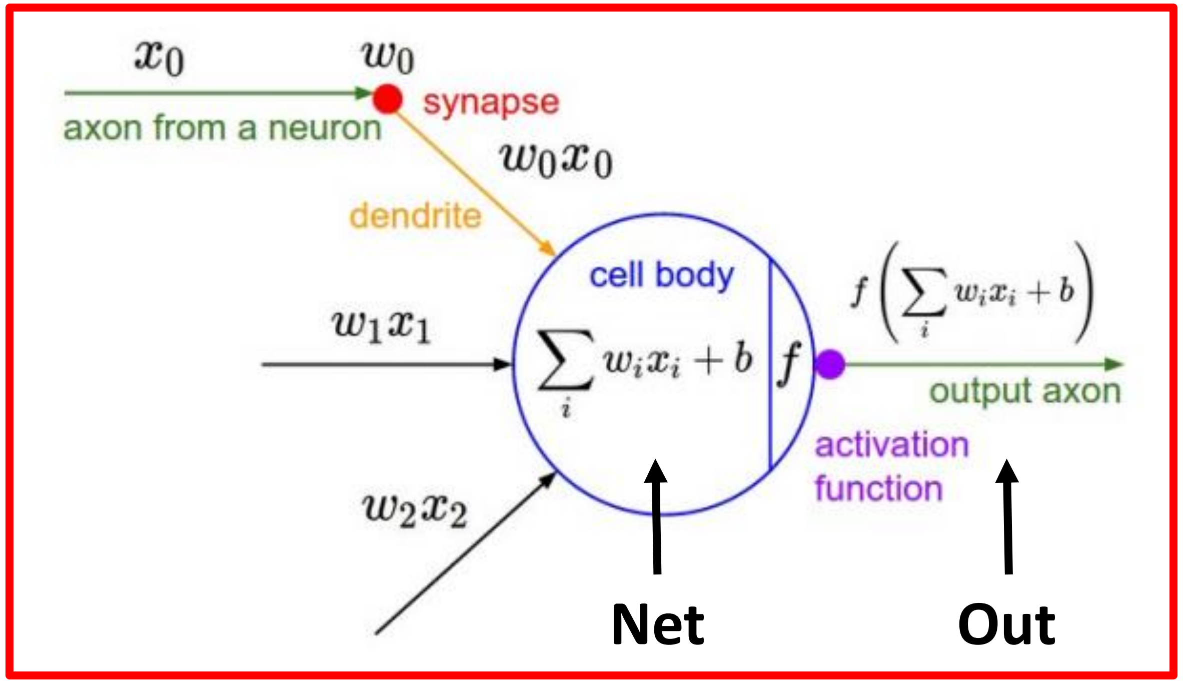
- You only have one training data, which is ((0.5,0.1), (0.1, 0.99))
- You are going to update bi and wi in the neural network
- The weight and biases are randomly initialized in the beginning.
- A logistic activation function is used in each neuron at the hidden and output layers

Forward Pass (h_1)

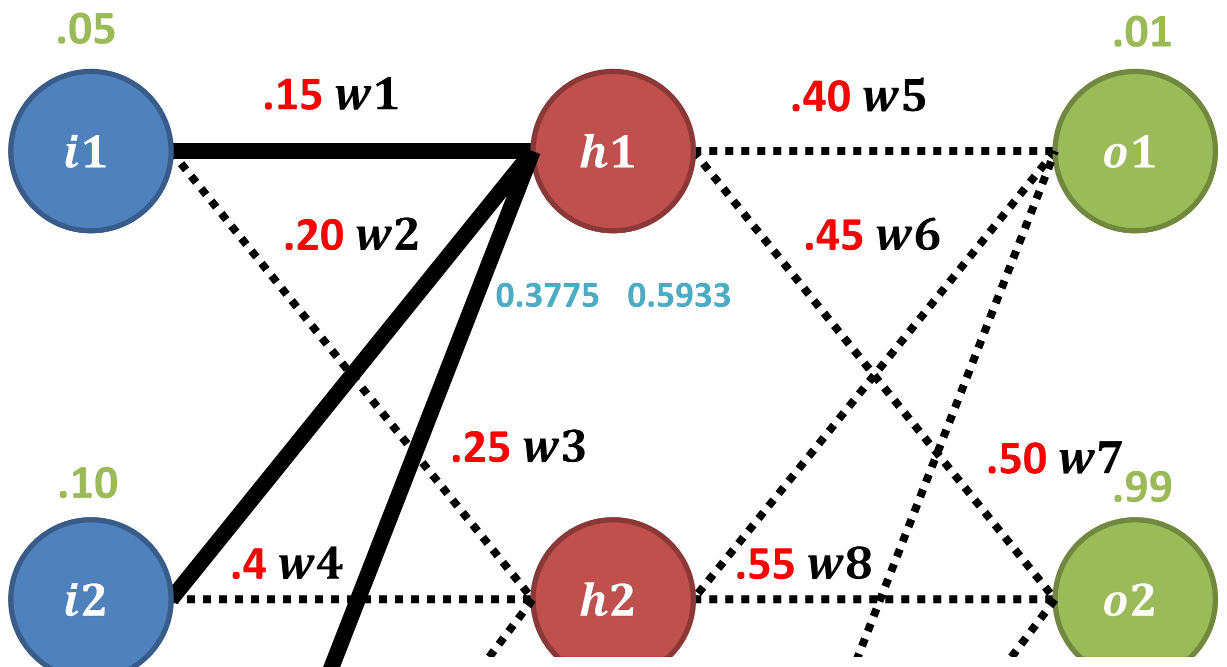


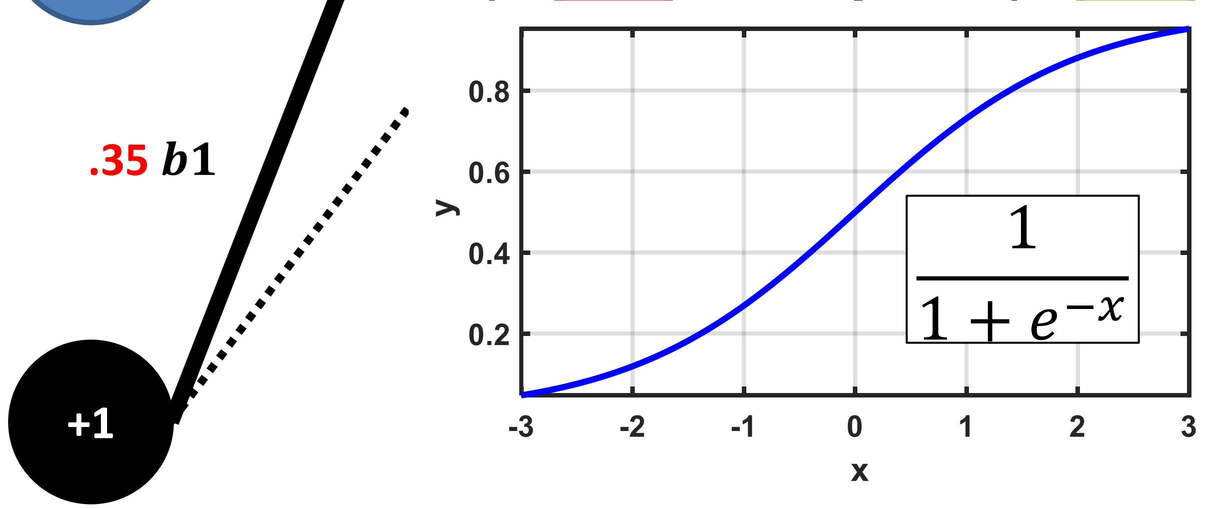
$$net_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 = 0.3775$$



Forward Pass (h_1)



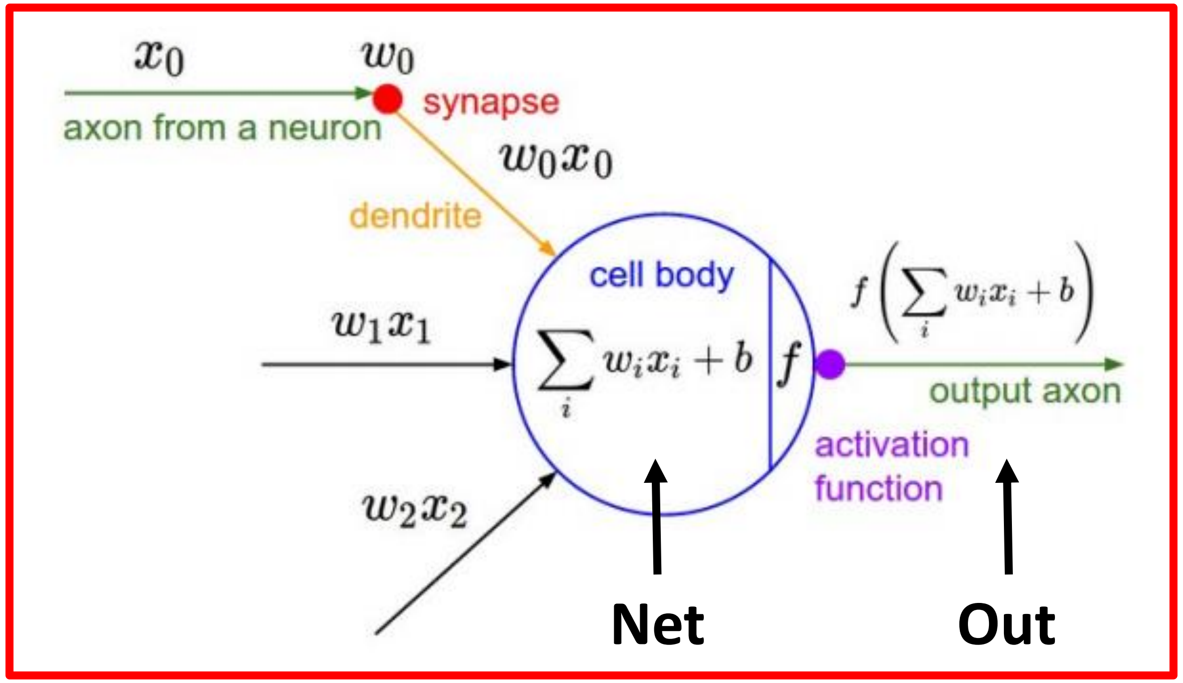


$$net_{h1} = w_1 i_1 + w_2 i_2 + b_1$$

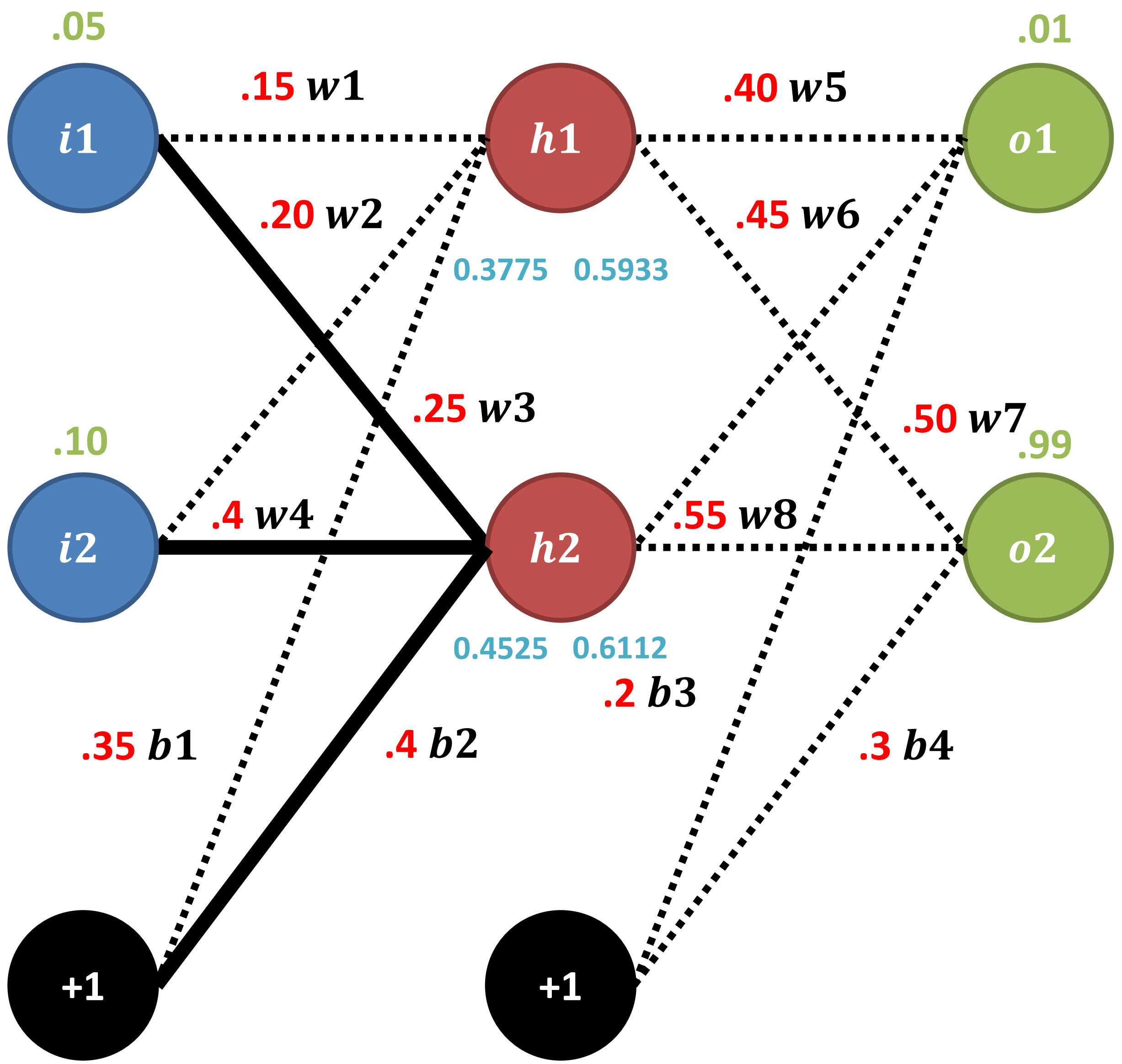
$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 = 0.3775$$

$$out_{h1} = f(net_{h1}) = \frac{1}{1+e^{-net_{h1}}}$$

$$out_{h1} = \frac{1}{1+e^{-0.3775}} = 0.5933$$



Forward Pass (h_2)

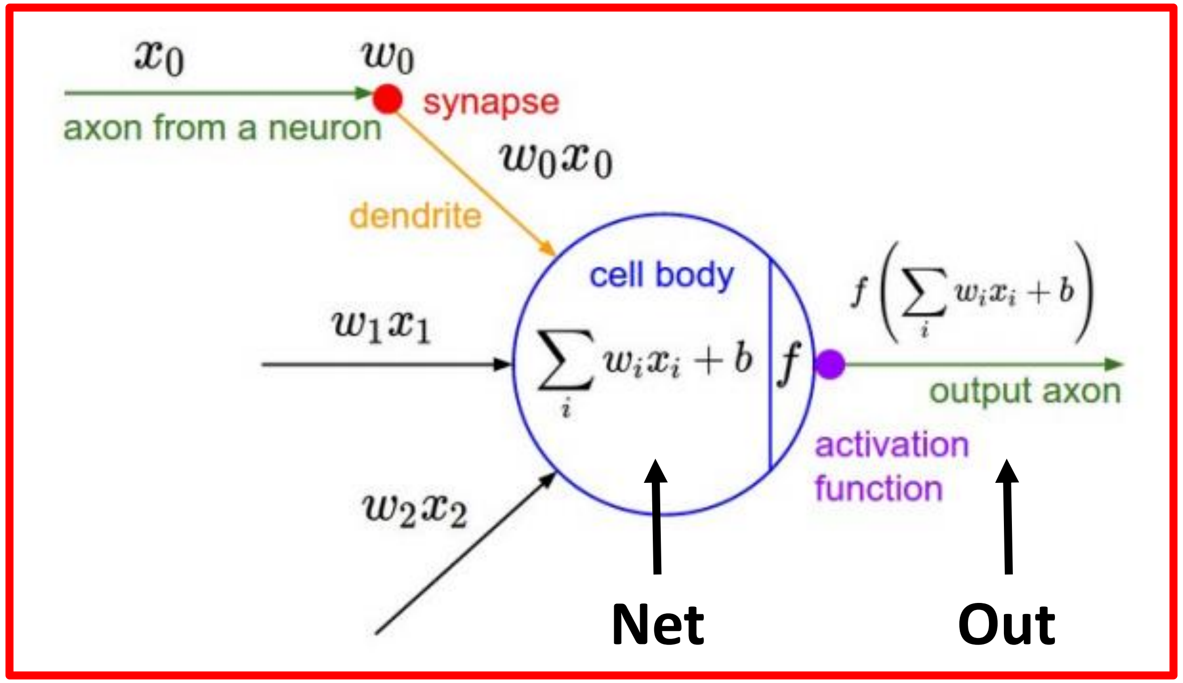


$$net_{h2} = w_3 i_1 + w_4 i_2 + b_2$$

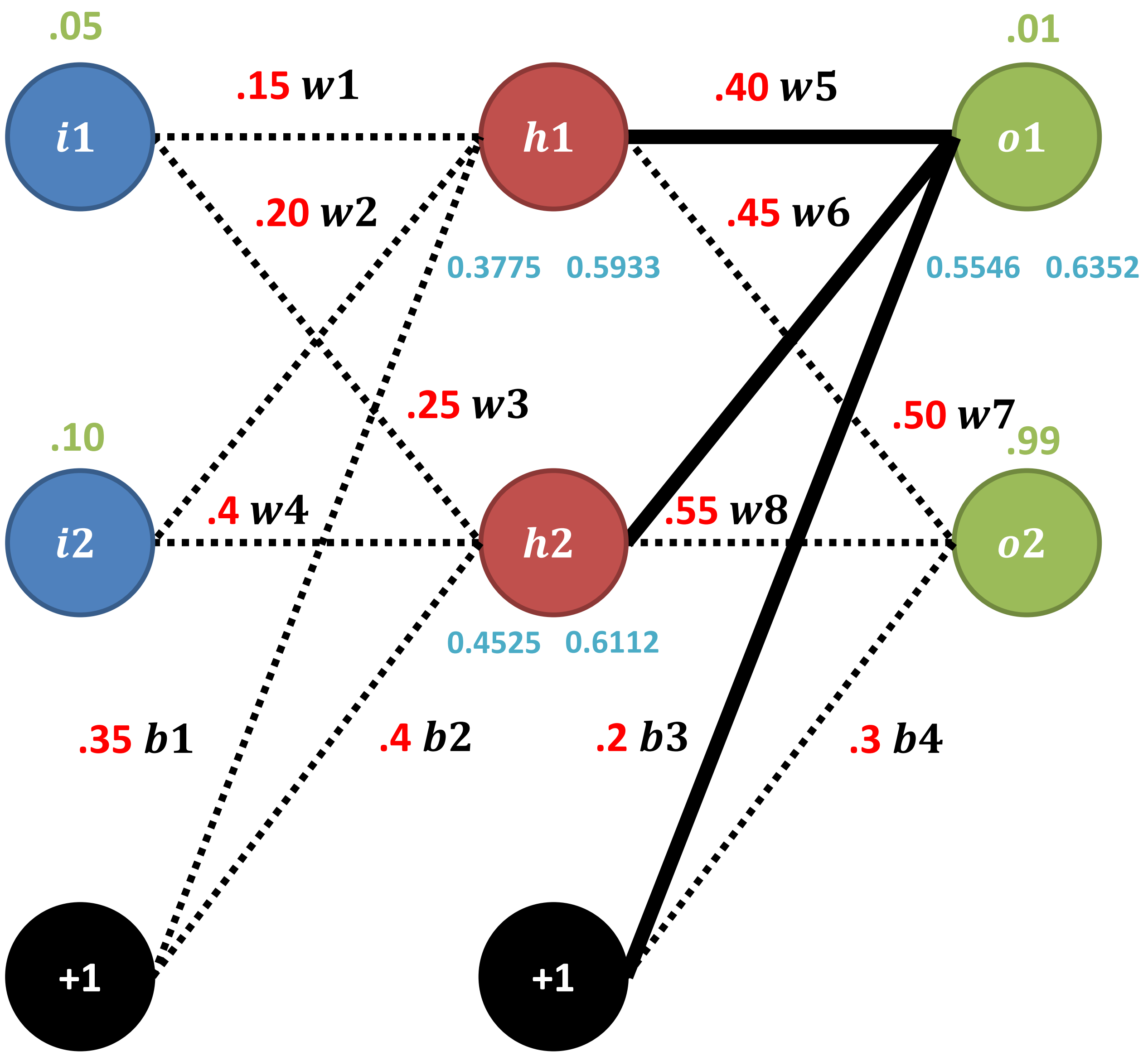
$$net_{h2} = 0.25 * 0.05 + 0.4 * 0.1 + 0.4 = 0.4525$$

$$out_{h2} = f(net_{h2}) = \frac{1}{1 + e^{-net_{h2}}}$$

$$out_{h2} = \frac{1}{1 + e^{-0.4525}} = 0.6112$$



Forward Pass (o_1)

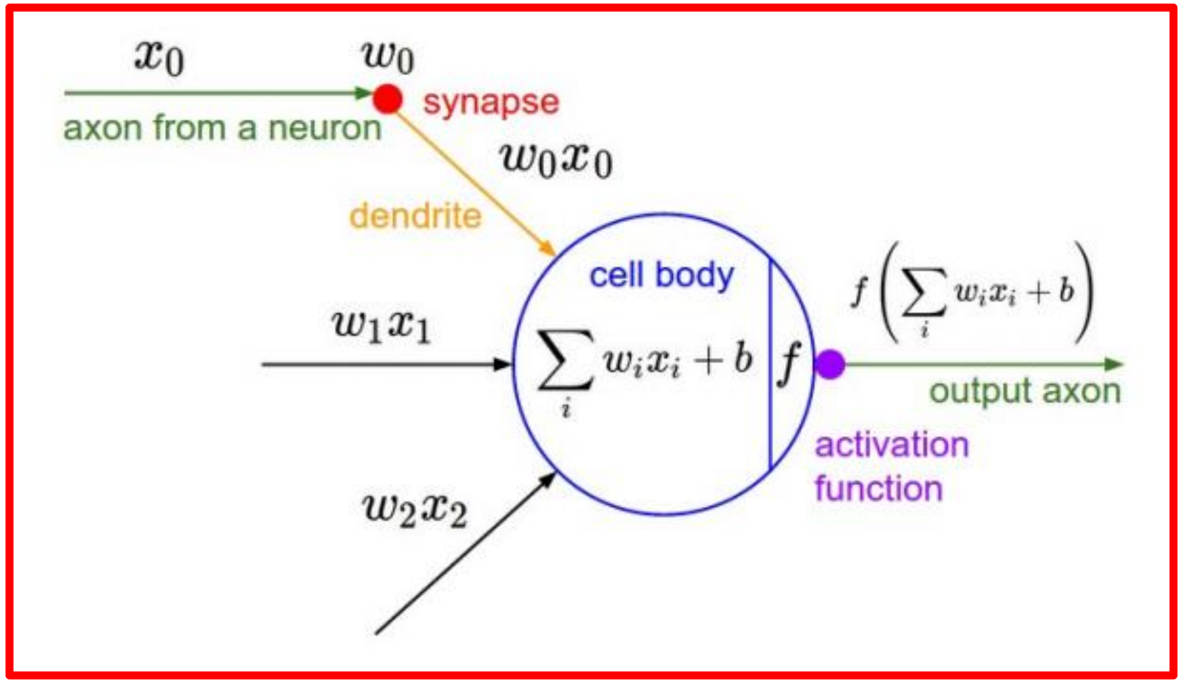


$$net_{o1} = w_5 out_{h1} + w_6 out_{h2} + b_3$$

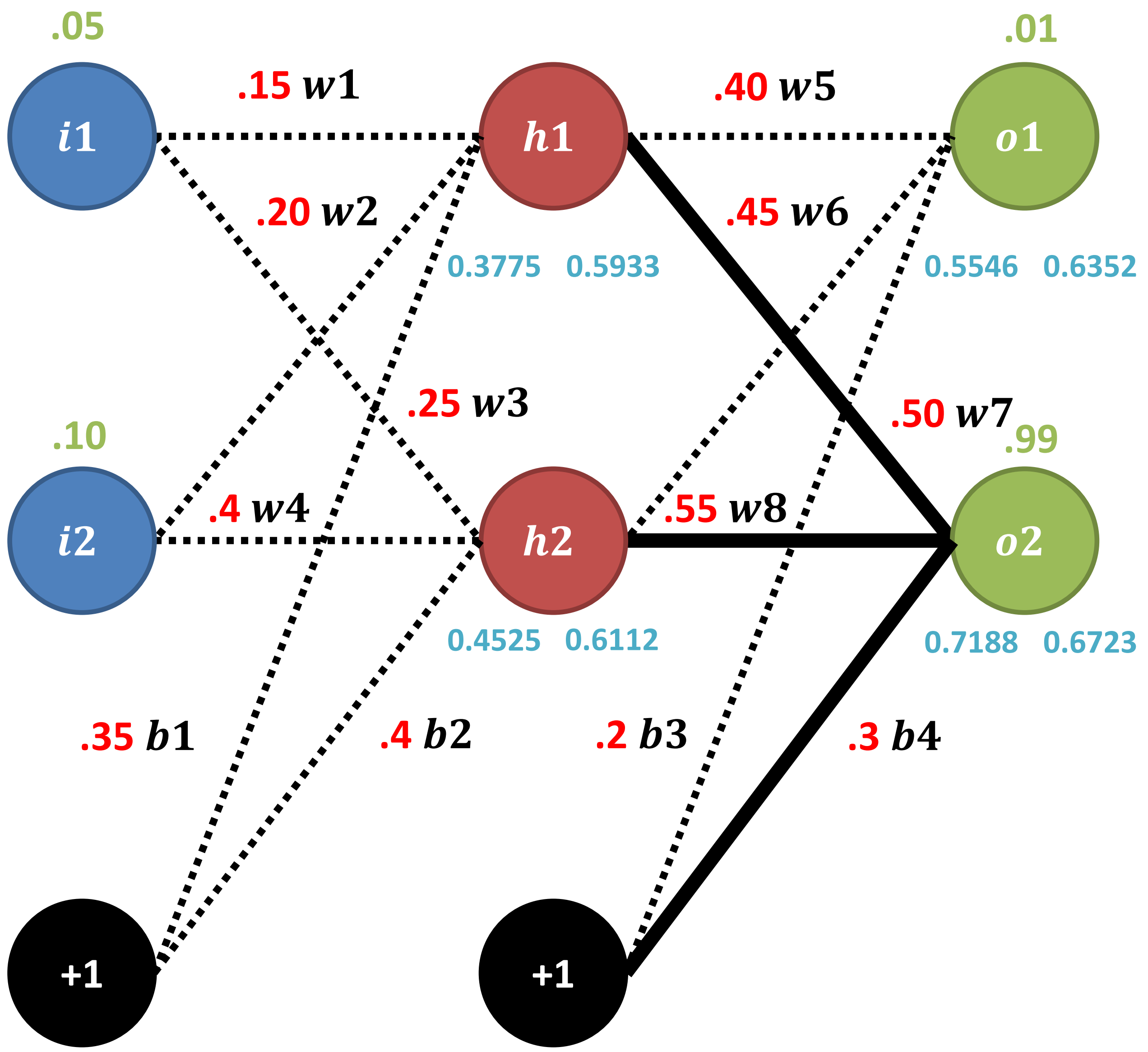
$$net_{o1} = 0.4 * 0.5933 + 0.45 * 0.6112 = 0.5546$$

$$out_{o1} = f(net_{o1}) = \frac{1}{1 + e^{-net_{o1}}}$$

$$out_{o1} = \frac{1}{1 + e^{-0.5546}} = 0.6352$$



Forward Pass (o_2)

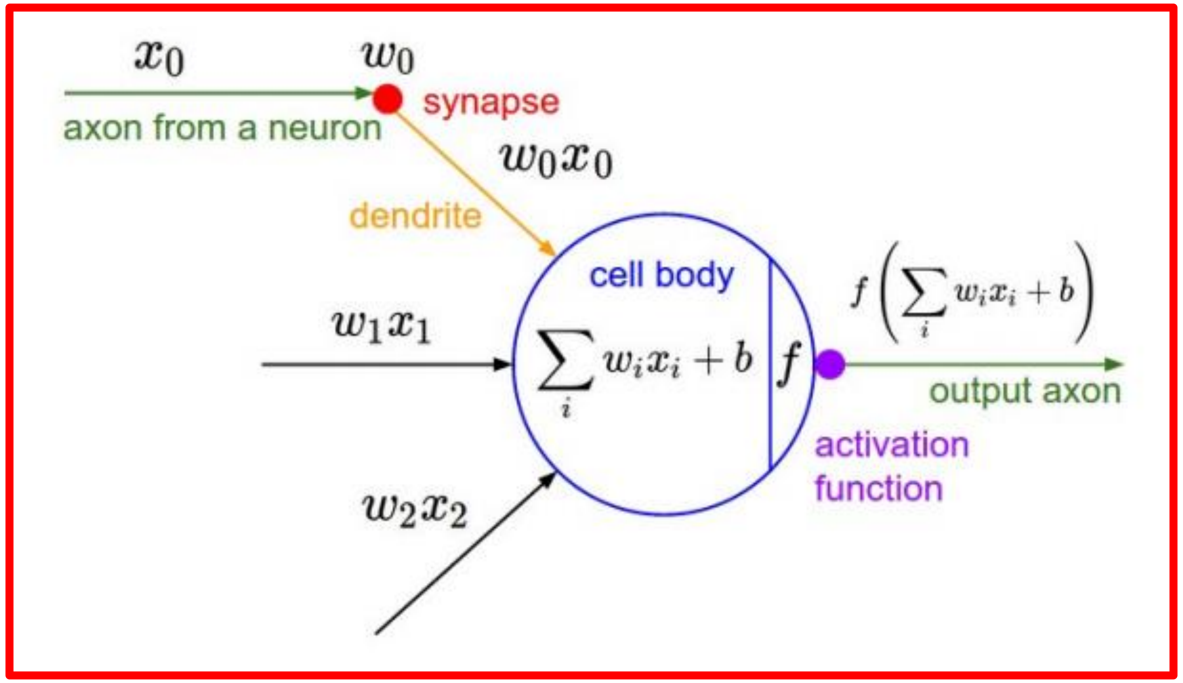


$$net_{o2} = w_7 out_{h1} + w_8 out_{h2} + b_4$$

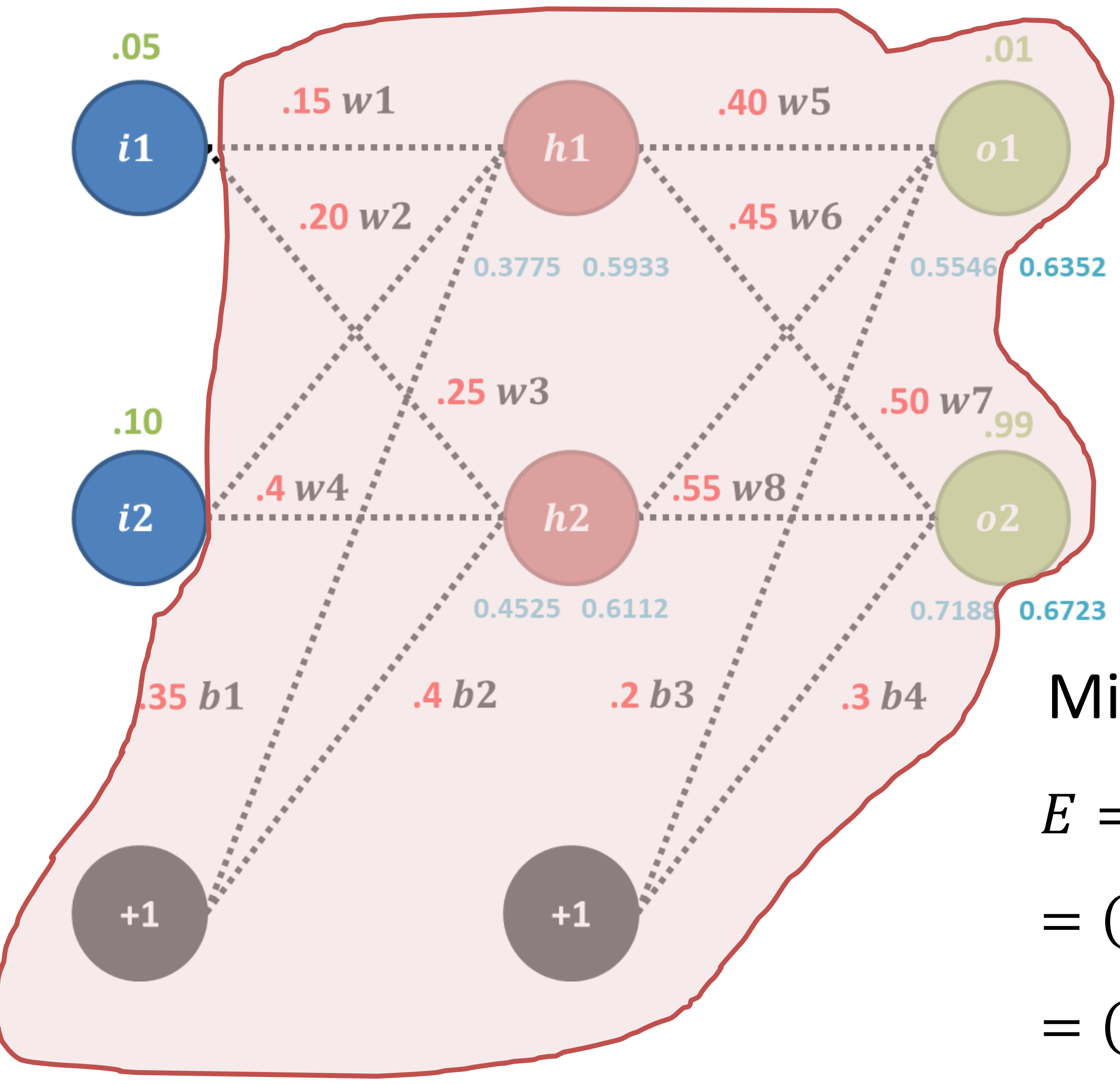
$$net_{o2} = 0.45 * 0.5933 + 0.55 * 0.6112 = 0.7188$$

$$out_{o2} = f(net_{o2}) = \frac{1}{1 + e^{-net_{o2}}}$$

$$out_{o2} = \frac{1}{1 + e^{-0.7188}} = 0.6723$$



What Do We Do in the Forward Pass?

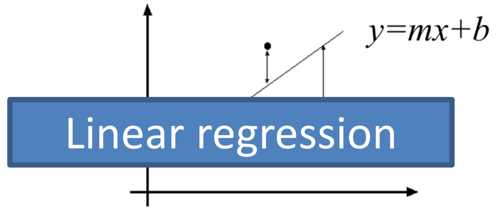


Data (measurement): $(x_1, y_1), \dots, (x_n, y_n)$

Model: Line $f(x_i, m, b) = mx_i + b$

Task: Find (m, b)

Minimize $E = J(m,b) = \sum_{i=1}^{n} (y_i - f(x_i, m, b))^2 = \sum_{i=1}^{n} (y_i - mx_i - b)^2$



$$\longrightarrow \quad (out_1, out_2) = f(i_1, i_2, \mathbf{w}, \mathbf{b})$$

Model: Neural network!

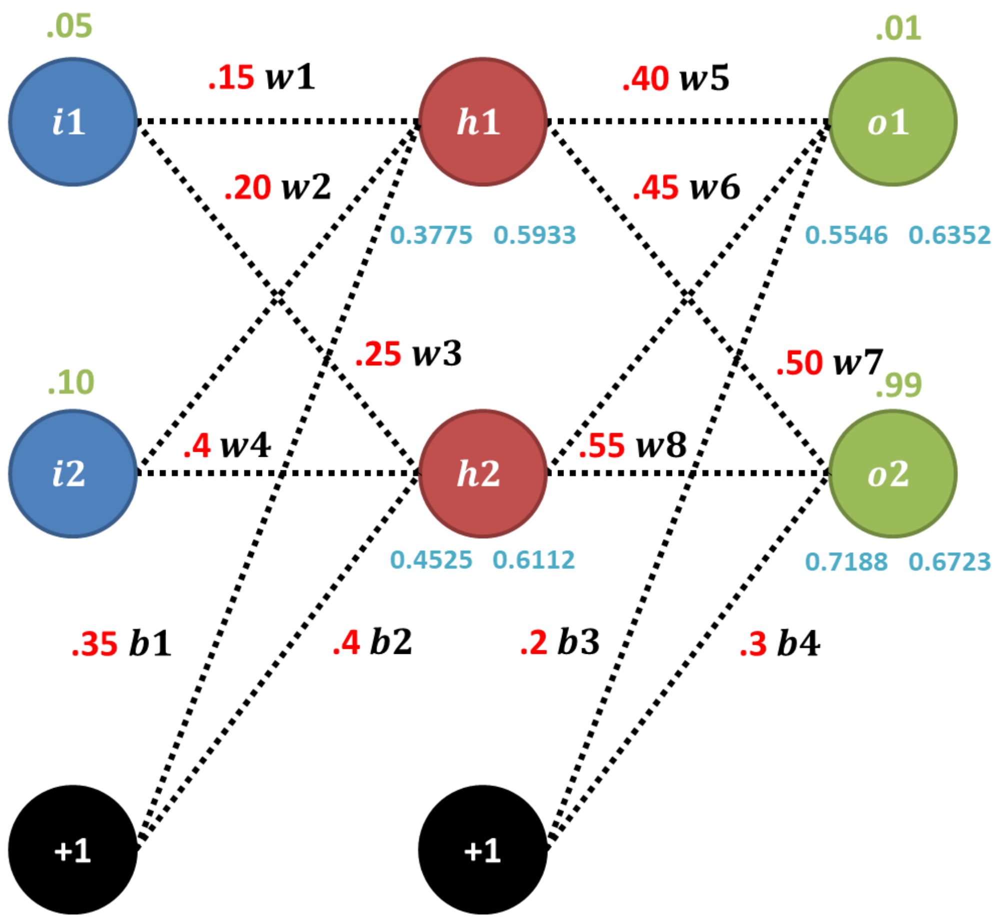
Minimize

$$E = J(\mathbf{w}, \mathbf{b})$$

$$= (o_1 - f(i_1, i_2, \mathbf{w}, \mathbf{b})_1)^2 + (o_2 - f(i_1, i_2, \mathbf{w}, \mathbf{b})_2)^2$$

$$= (o_1 - out_1)^2 + (o_2 - out_2)^2$$

Calculating the Total Error

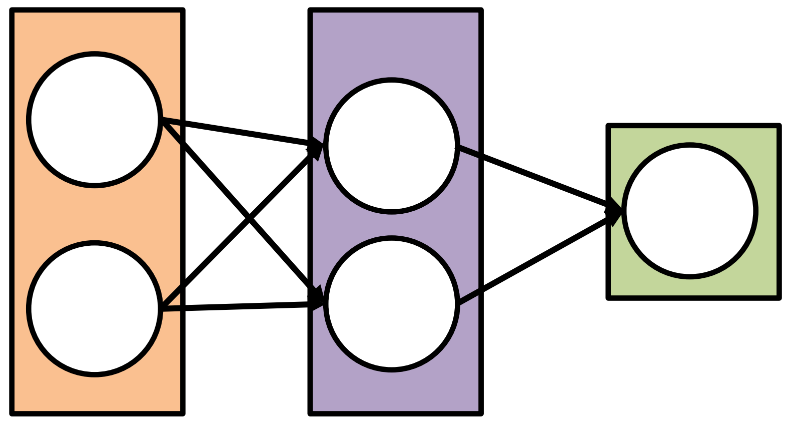


$$E = J(\mathbf{w}, \mathbf{b})$$
$$= (o_1 - f(i_1, i_2, \mathbf{w}, \mathbf{b})_1)^2 + (o_2 - f(i_1, i_2, \mathbf{w}, \mathbf{b})_2)^2$$
$$= (o_1 - out_{o1})^2 + (o_2 - out_{o2})^2$$
$$= (0.01 - 0.6352)^2 + (0.99 - 0.6723)^2 = 0.7013$$

We do not know the analytic form of the cost function with respect to estimating parameters $(\mathbf{w}, \mathbf{b})$. However, we just calculate its values at $(\mathbf{w}_0, \mathbf{b}_0)$, which correspond to the values in red.

End of a forward pass

(Optional) Activation Function

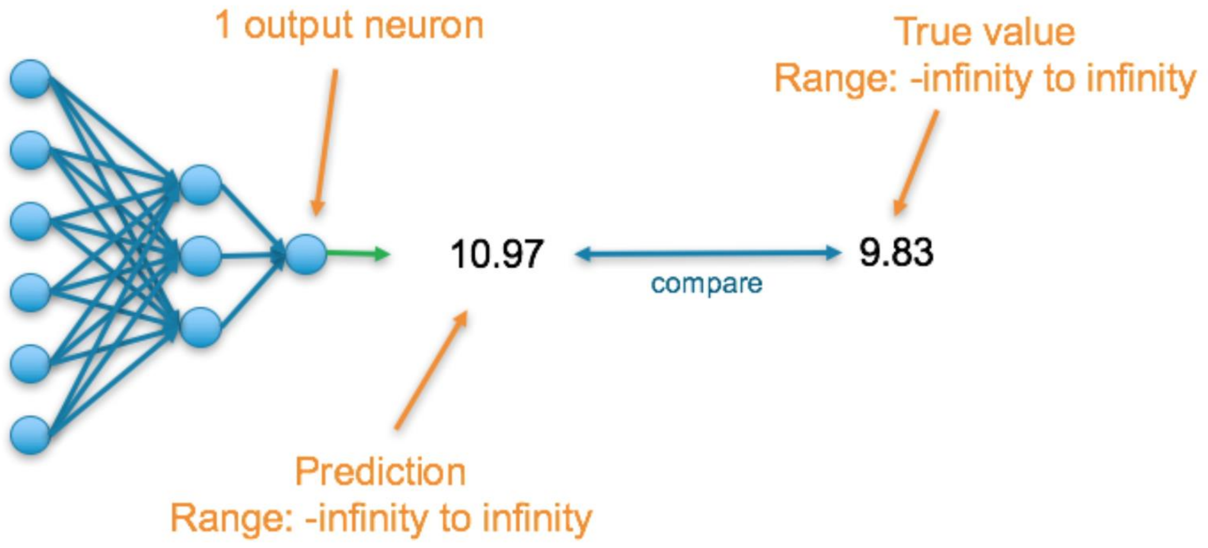


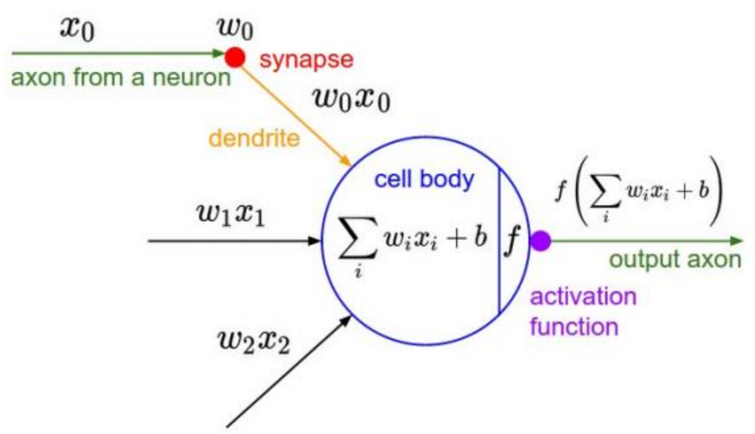
Input Layer

Output Layer

Hidden Layer

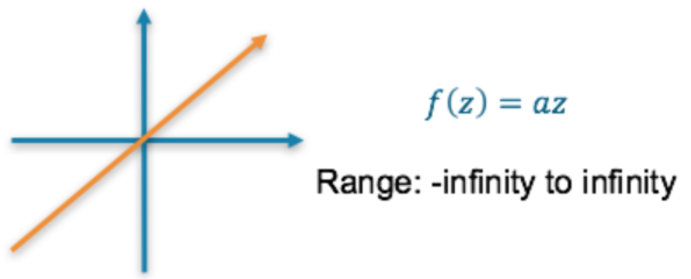
Final (Output) Neuron (Regression)





Final Activation Function

Linear—This results in a numerical value which we require



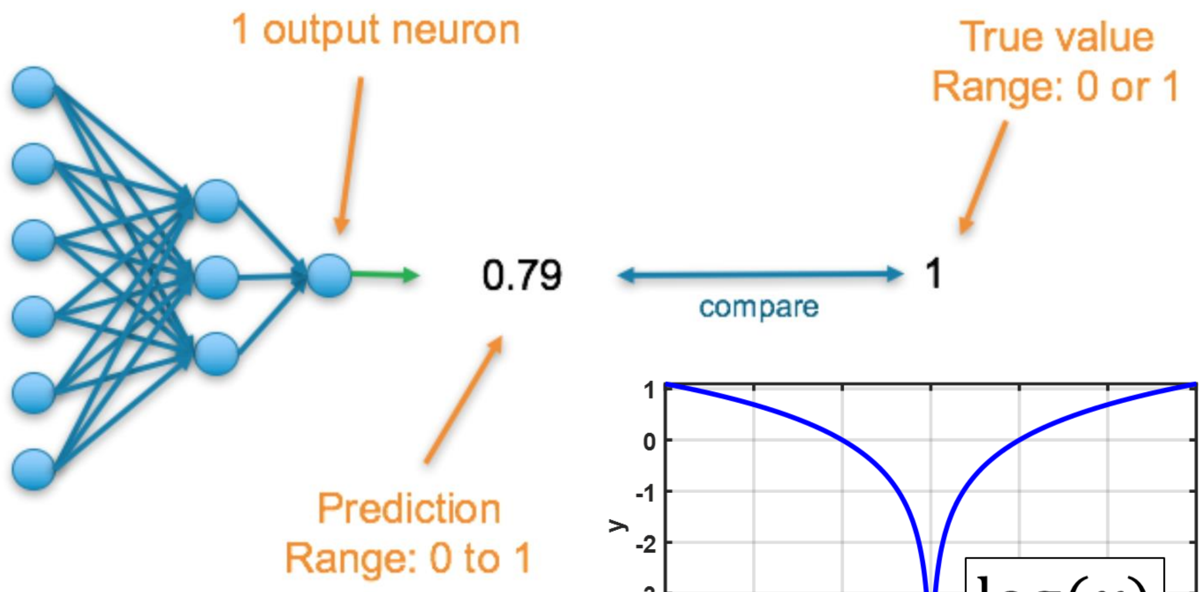
Loss Function

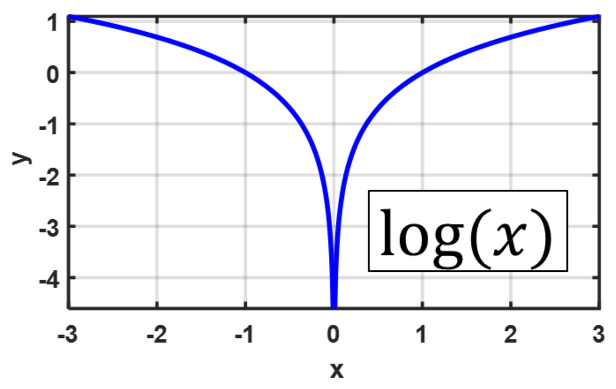
Mean squared error (MSE)—This finds the average squared difference between the predicted value and the true value

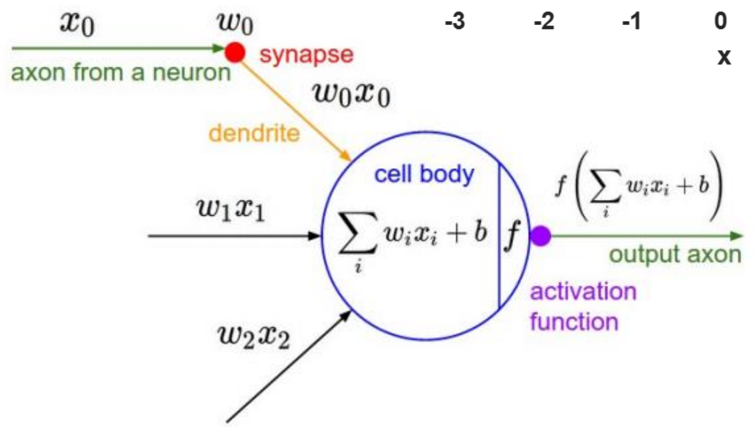
$$\text{MSE} = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where $\hat{y}$ is the predicted value and y is the true value

Final (Output) Neuron (Binary Classification)

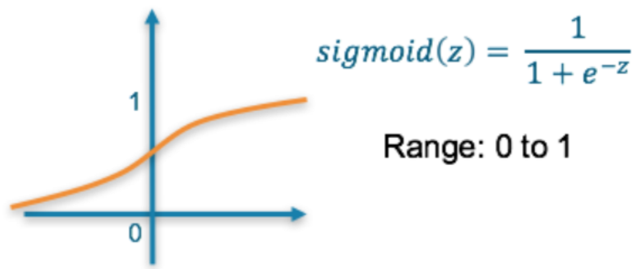






Final Activation Function

Sigmoid—This results in a value between 0 and 1 which we can infer to be how confident the model is of the example being in the class

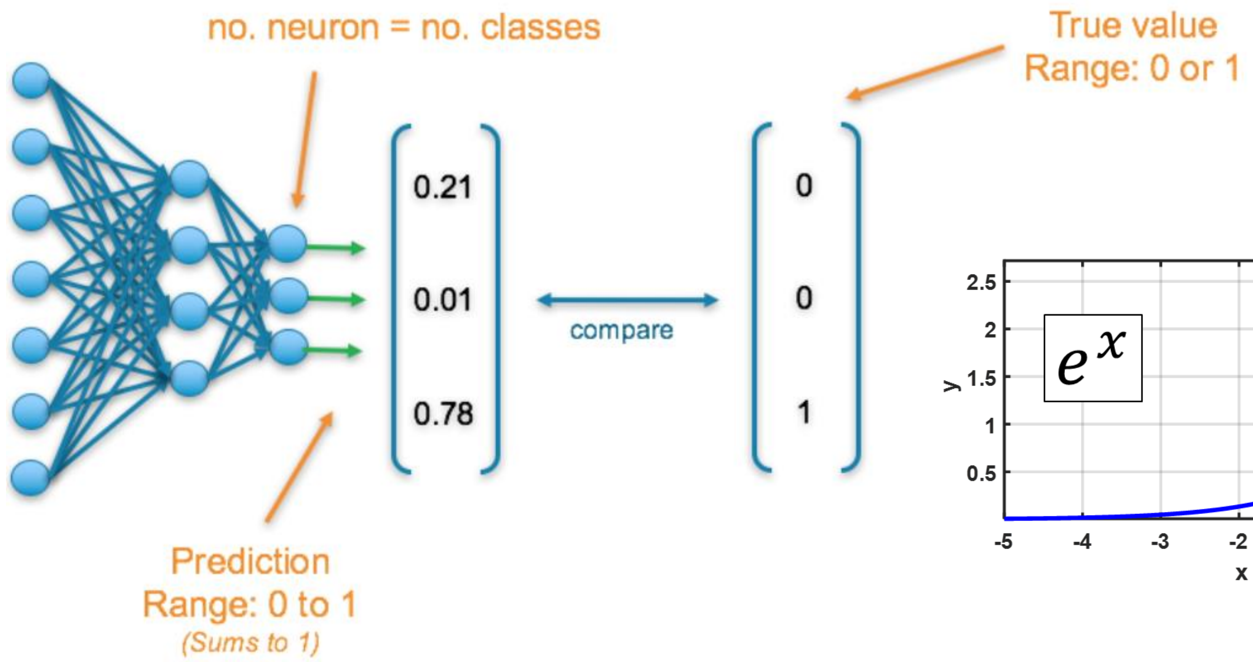


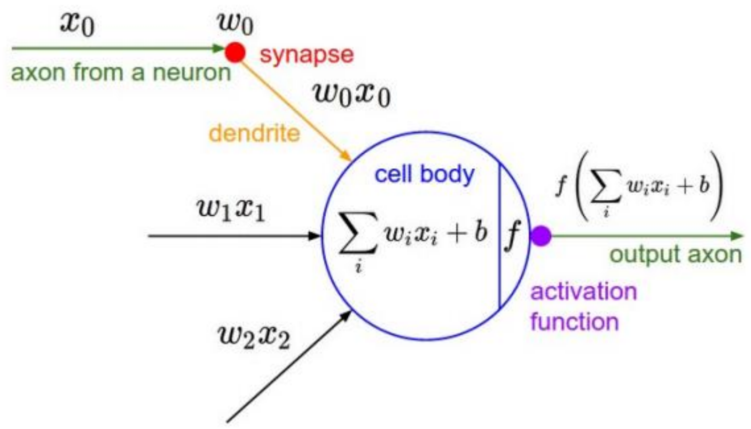
Loss Function

Binary Cross Entropy—Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p, 1-p} as we have a binary distribution. We use binary cross-entropy to compare this with the true distribution {y, 1-y}

Binary cross entropy = $-(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$
Where $\hat{y}$ is the predicted value and y is the true value

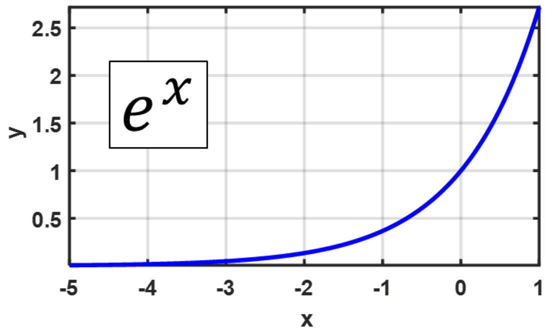
Final (Output) Neuron (Multi-Classification)

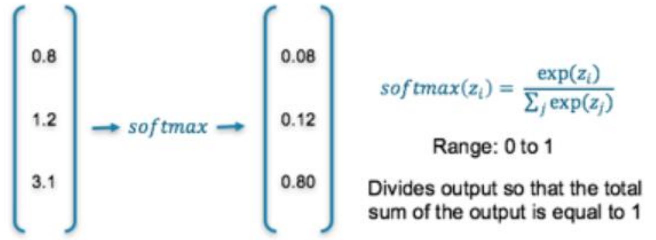




Final Activation Function

Softmax—This results in values between 0 and 1 for each of the outputs which all sum up to 1. Consequently, this can be inferred as a probability distribution





0.8
1.2
3.1
softmax
0.08
0.12
0.80
$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$
Range: 0 to 1
Divides output so that the total sum of the output is equal to 1

Loss Function

Cross Entropy—Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p1, p2, p3} (where p1+p2+p3 = 1). We use cross-entropy to compare this with the true distribution {y1, y2, y3}

Cross entropy = $-\sum_i^M y_i \log(\hat{y}_i)$

Where $\hat{y}$ is the predicted value, y is the true value and M is the number of classes