Neural Network

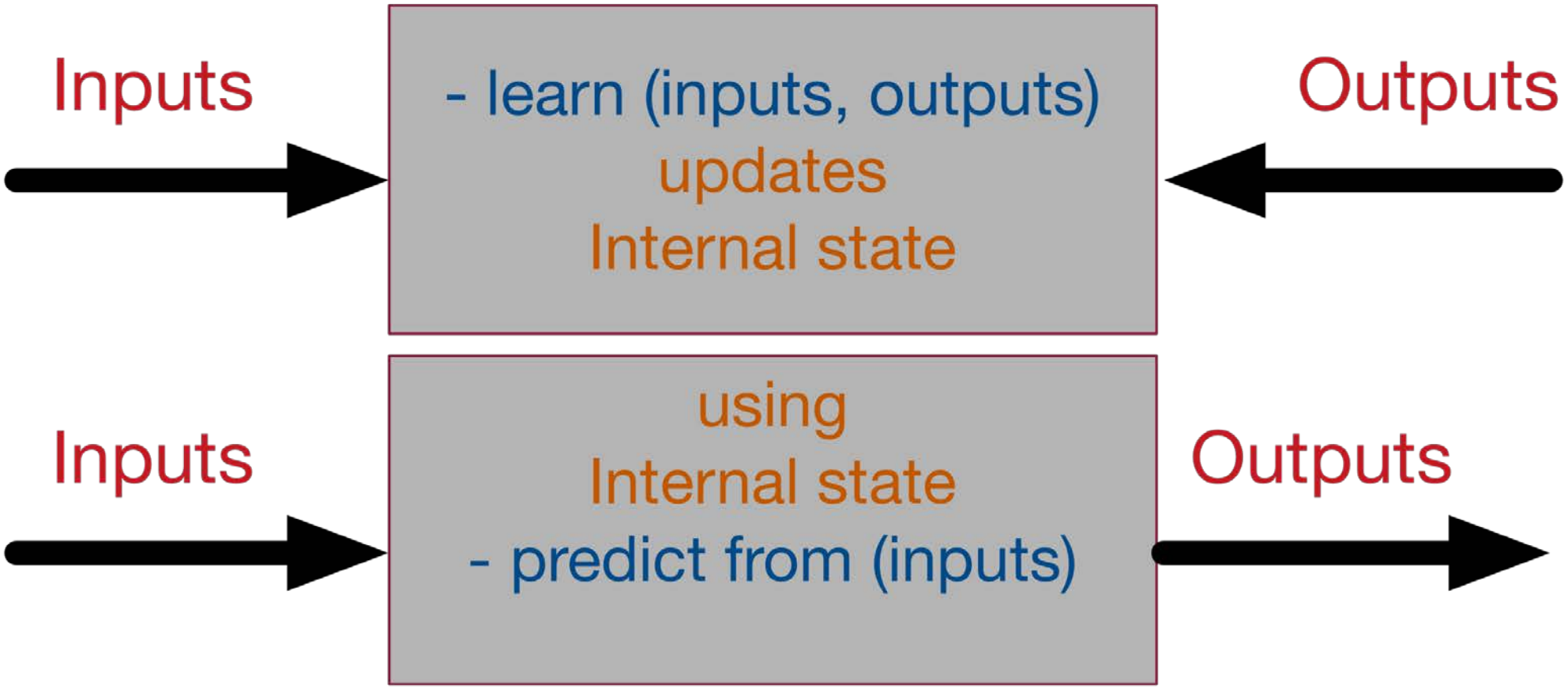
Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

CIVE 497 – CIVE 700: Smart Structure Technology

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-03-27

Supervised Neural Networks



https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e

Gradient Descent

Gradient descent is an iterative machine learning optimization algorithm to **reduce the cost function** so that we have models that makes accurate predictions.

Cost function(J) or Loss function measures **the difference between the actual output and predicted output** from the model.

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

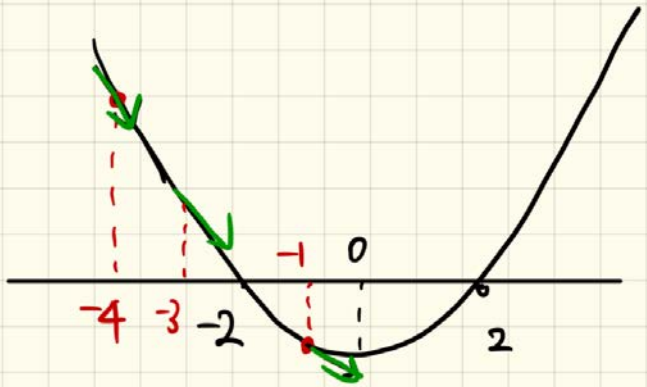
https://medium.com/@montjoile/an-introduction-to-gradient-descent-algorithm-34cf3cee752b

Example: Quadratic Function

$y = (x-2)(x+2) = x^2 - 4$

$J(x)$ cost function.

$y' = 2x$



$y'(-4) = -8$

$y'(-3) = -6$

$y'(-1) = -2$

Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\theta_0 = -4$, $\alpha = 0.1$

$y'(\theta_0) = y'(-4)$

$\theta_1 \leftarrow \theta_0 - 0.1 \cdot -8 = -3.2$

$y'(\theta_1) = y'(-3.2)$

$\theta_2 \leftarrow \theta_1 - 0.1 \cdot -6.4 = -2.56$

$\theta_3 \leftarrow \theta_2 - 0.1 \cdot -5.12 = -3.072$

$y'(\theta_2) = y'(-2.56)$

Example: Quadratic Function (Continue)

Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

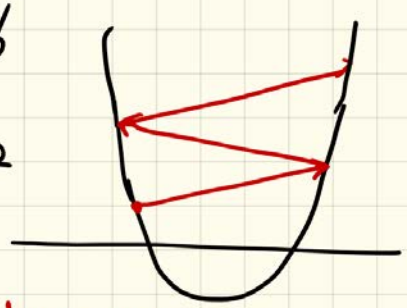
$\alpha = 1.5$

$\theta_0 = -4$, $\alpha = 1.5$

$\theta_1 \leftarrow \theta_0 - 1.5 \cdot -8 = 8$

$\theta_2 \leftarrow \theta_1 - 1.5 \cdot 16 = -16$

$\theta_3 \leftarrow \theta_2 - 1.5 \cdot -32 = 32$



Large learning rate !!

$\alpha = 0.1$

Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\theta_0 = -4$, $\alpha = 0.1$

$\theta_1 \leftarrow \theta_0 - 0.1 \cdot -8 = -3.2$ ($y'(\theta_0) = y'(-4)$)

$\theta_2 \leftarrow \theta_1 - 0.1 \cdot -6.4 = -2.56$ ($y'(\theta_1) = y'(-3.2)$)

$\theta_3 \leftarrow \theta_2 - 0.1 \cdot -5.12 = -3.072$ ($y'(\theta_2) = y'(-2.56)$)

Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

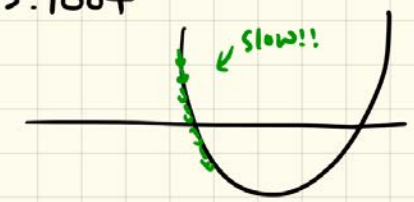
$\alpha = 0.01$

$\theta_0 = -4$, $\alpha = 0.01$

$\theta_1 \leftarrow \theta_0 - 0.01 - 8 = -3.98$

$\theta_2 \leftarrow \theta_1 - 0.01 \cdot -7.96 = -3.9004$

$\vdots$



Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\alpha = 0.1$

Gradient descent.

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$\theta_0 = -4$, $\alpha = 0.1$

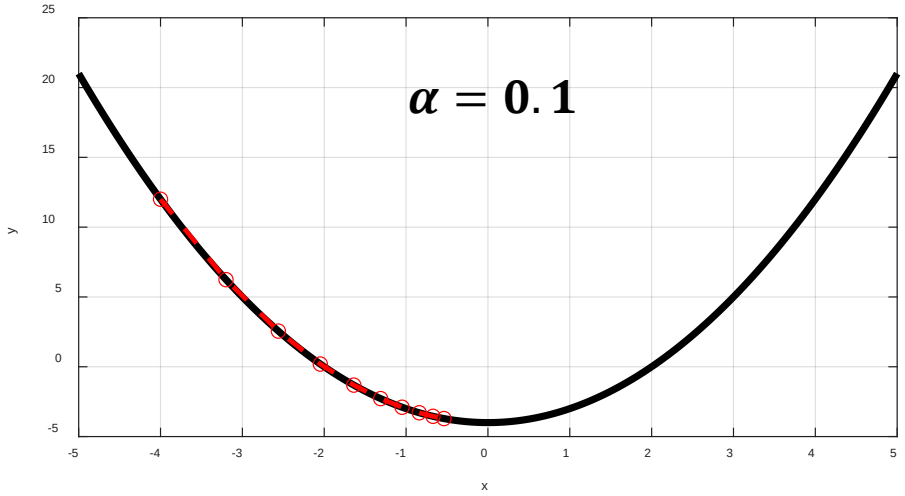
$\theta_1 \leftarrow \theta_0 - 0.1 \cdot -8 = -3.2$ ($y'(\theta_0) = y'(-4)$)

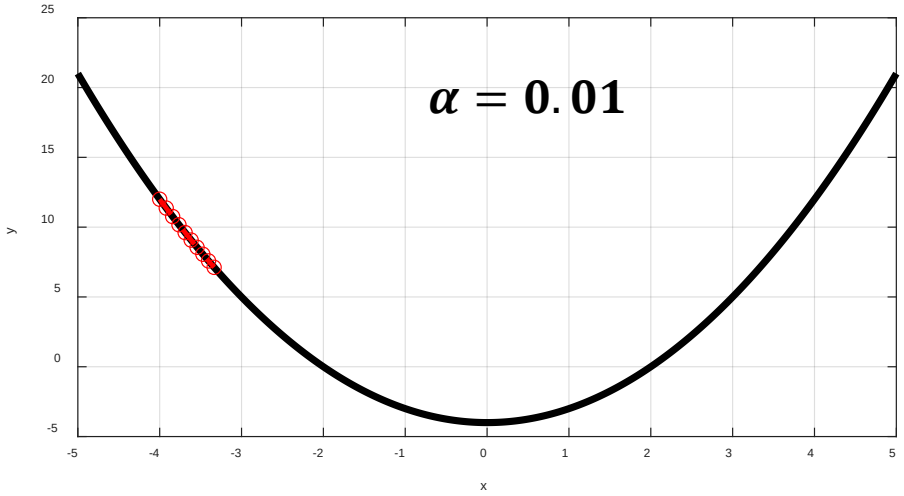
$\theta_2 \leftarrow \theta_1 - 0.1 \cdot -6.4 = -2.56$ ($y'(\theta_1) = y'(-3.2)$)

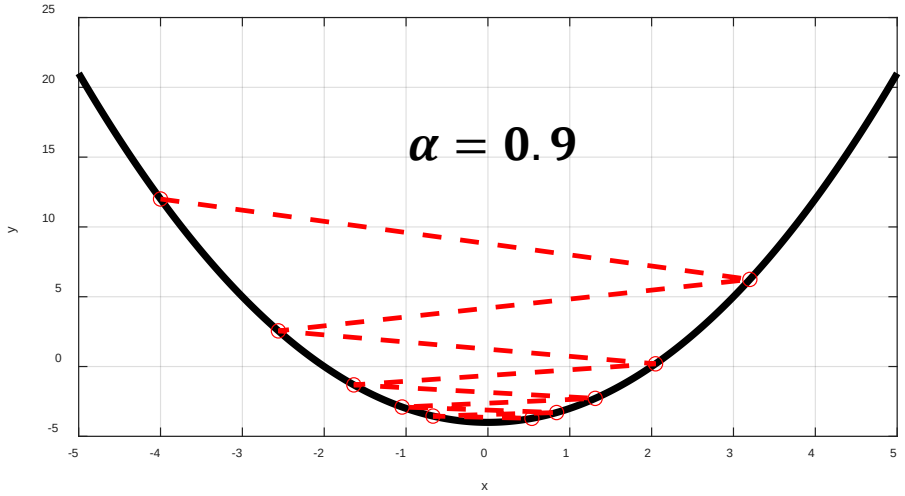
$\theta_3 \leftarrow \theta_2 - 0.1 \cdot -5.12 = -3.072$ ($y'(\theta_2) = y'(-2.56)$)

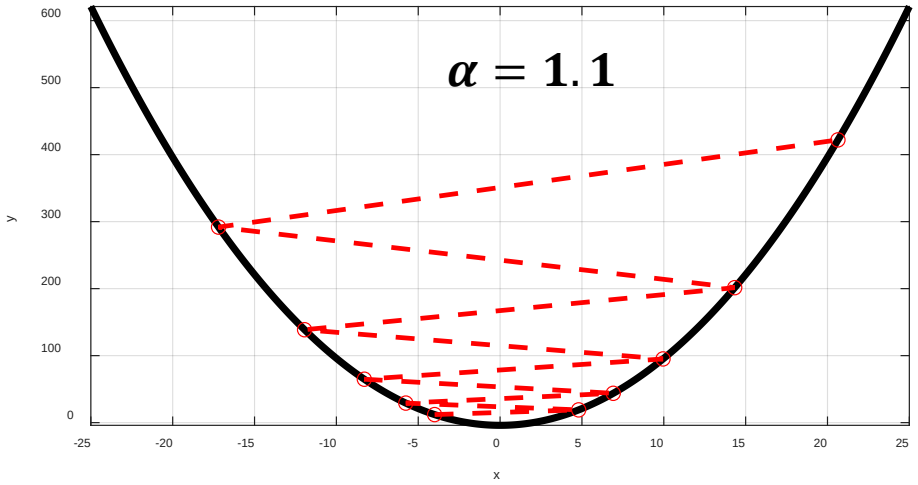
Example: Quadratic Function (Simulation)

10 iterations



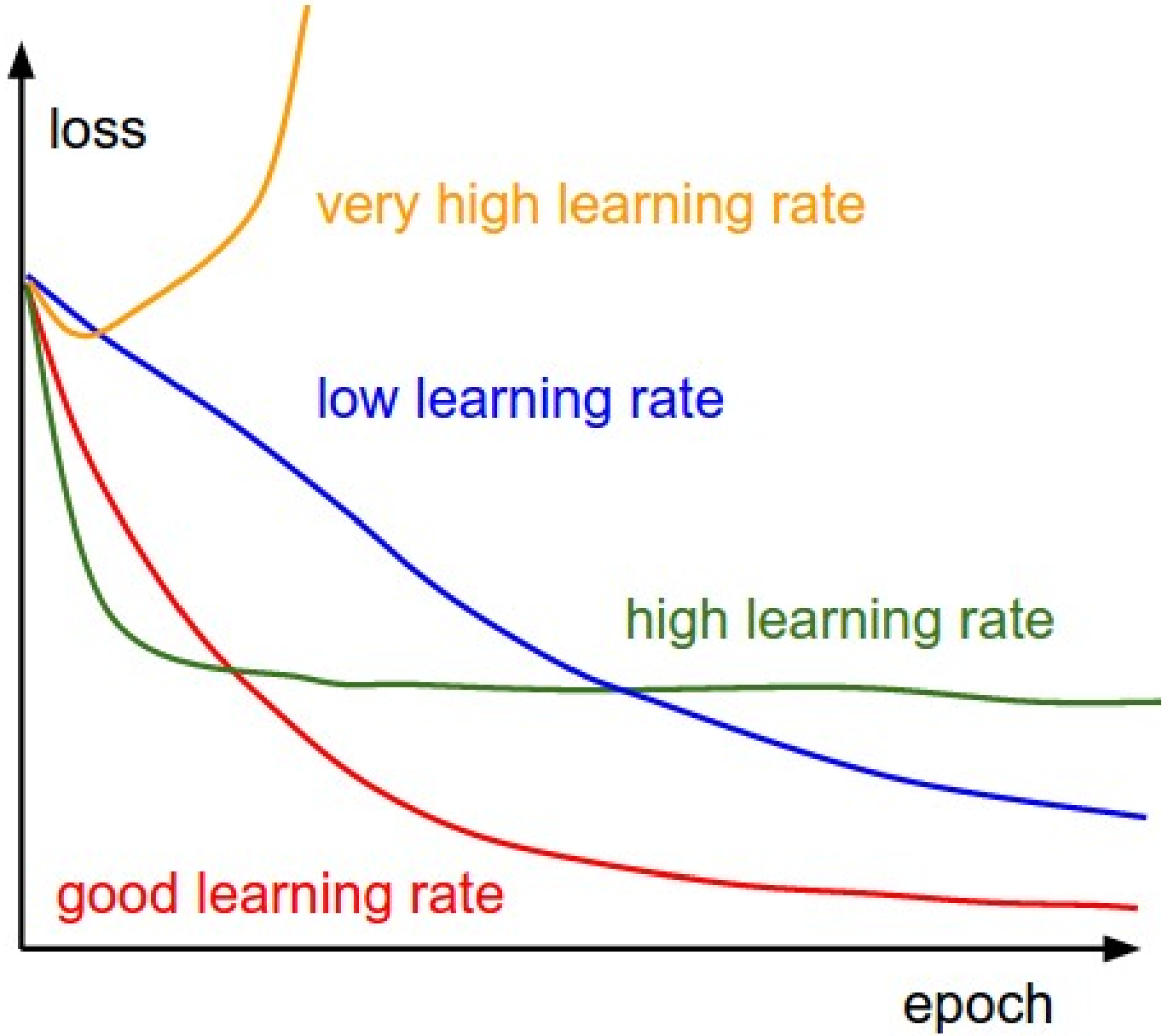




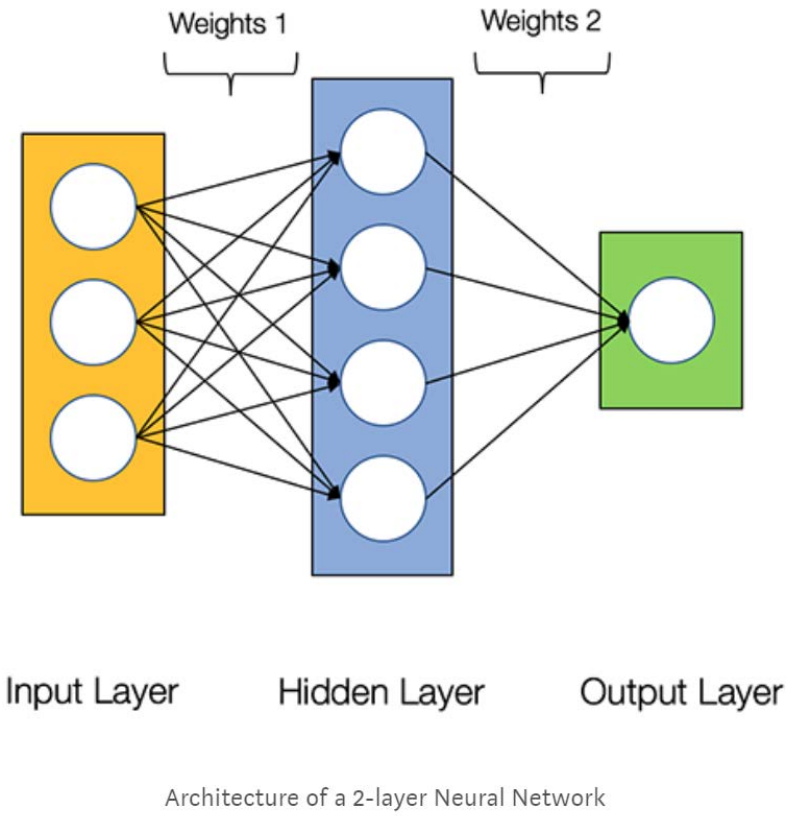


See tutorials

Effect of Learning Rates



What's a Neural Network



Architecture of a 2-layer Neural Network

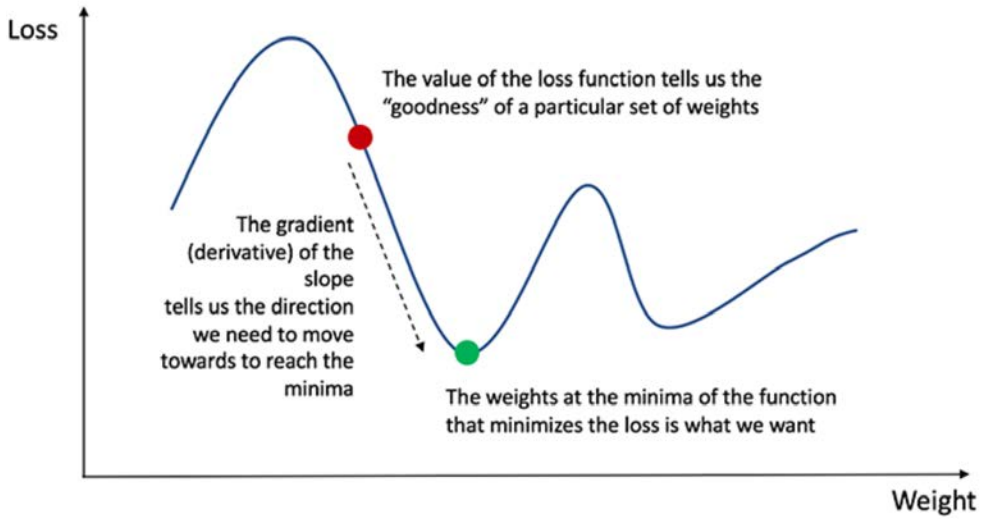
- An **input layer**, x
- An arbitrary amount of **hidden layers**
- An **output layer**, $\hat{y}$
- A set of **weights** and **biases** between each layer, W *and* b
- A choice of **activation function** for each hidden layer, σ. In this tutorial, we'll use a Sigmoid activation function.

https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6

Training the Neural Network

- Naturally, the right values for the weights and biases determines the strength of the predictions. The process of fine-tuning the weights and biases from the input data is known as training the Neural Network.
- Our goal in training is to find the best set of weights and biases that minimizes the loss function.

- Calculating the predicted output $\hat{y}$, known as **feedforward**
- Updating the weights and biases, known as **backpropagation**



Gradient descent algorithm

https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6

Mathematical Model of a Neuron

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

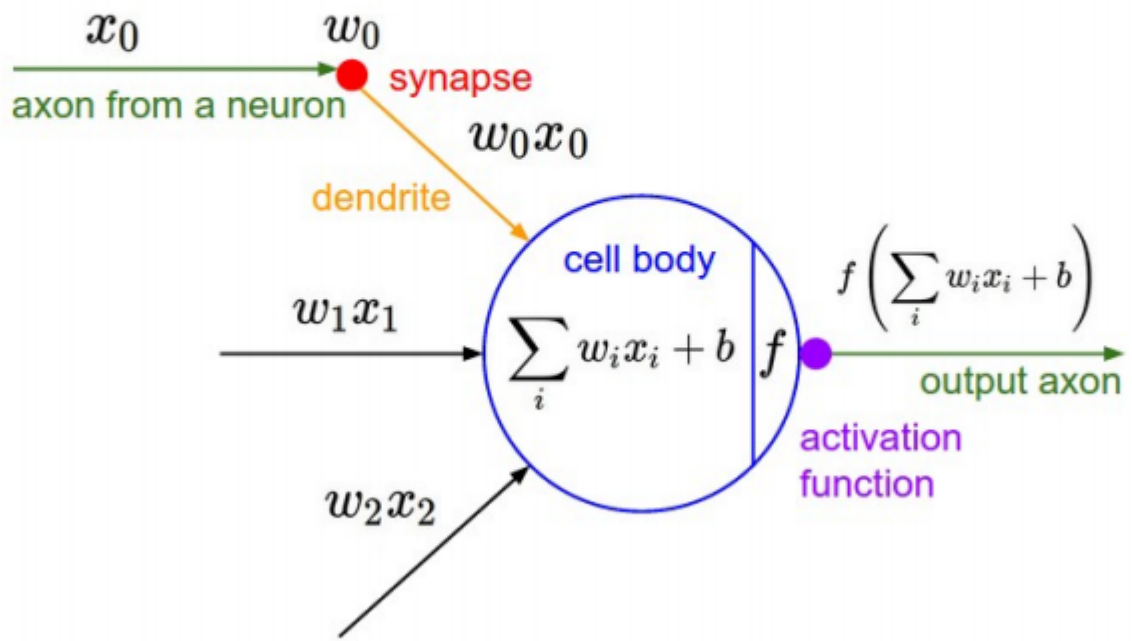
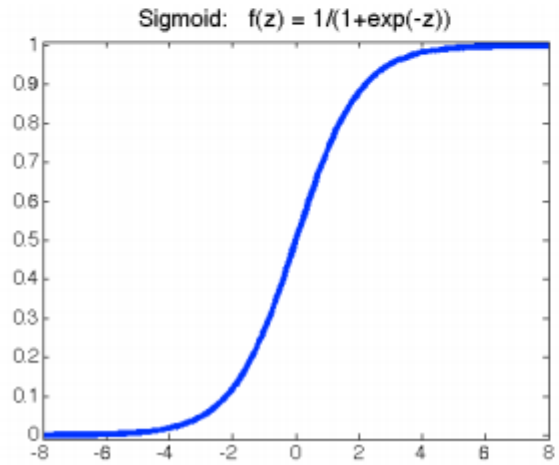


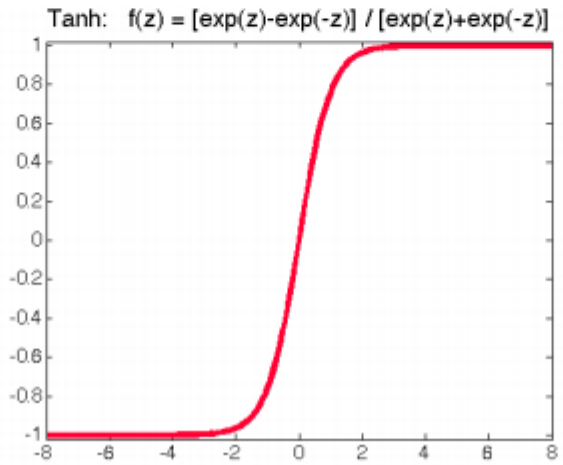
Figure : A mathematical model of the neuron in a neural network

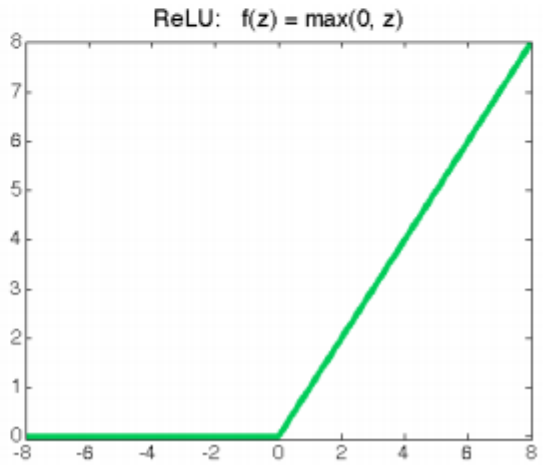
Activation Functions

Most commonly used activation functions:

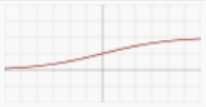
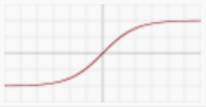
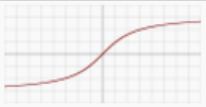
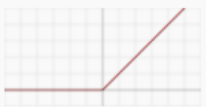
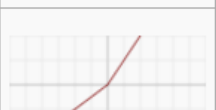
- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$







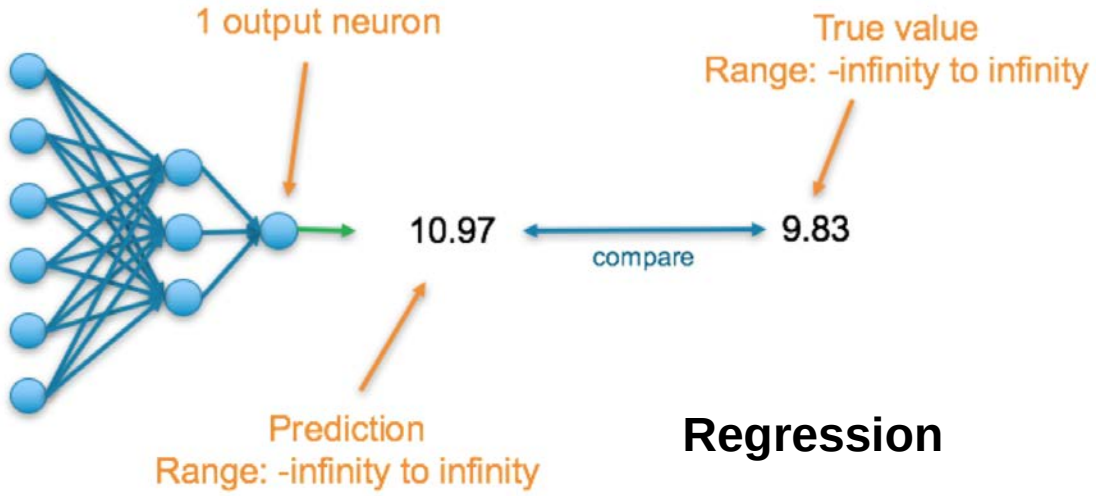
Activation Functions and Their Derivatives

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Loss Function

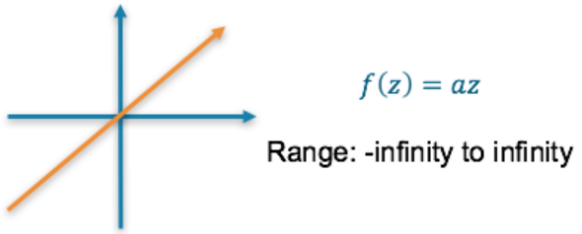
There are many available loss functions, and the nature of our problem should dictate our choice of loss function.



Regression

Final Activation Function

Linear—This results in a numerical value which we require



Loss Function

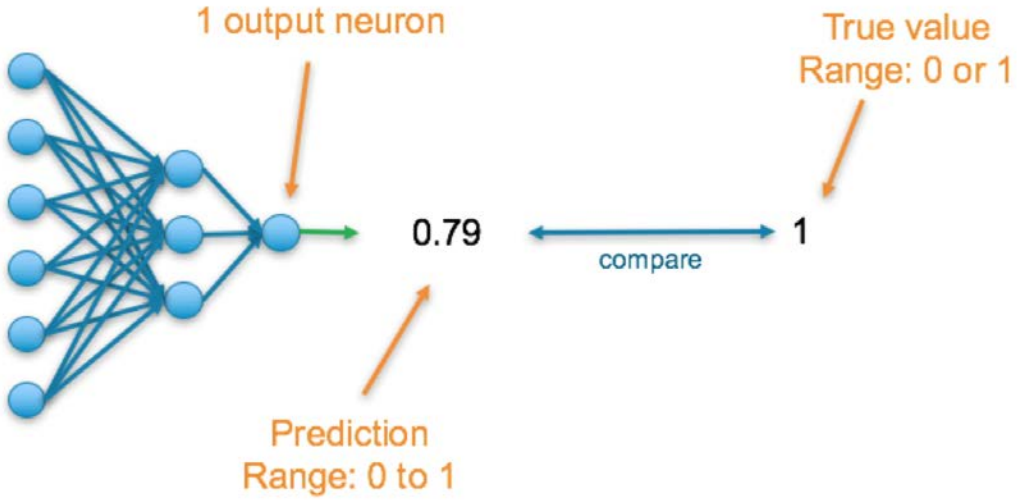
Mean squared error (MSE)—This finds the average squared difference between the predicted value and the true value

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

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

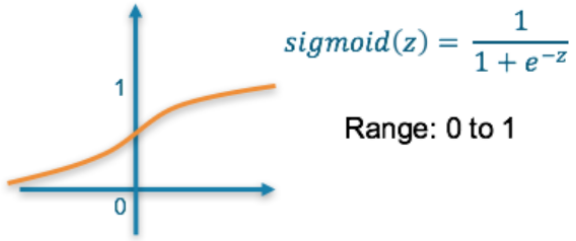
https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8
https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Loss Function (Continue)



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}

$$\text{Binary cross entropy} = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

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

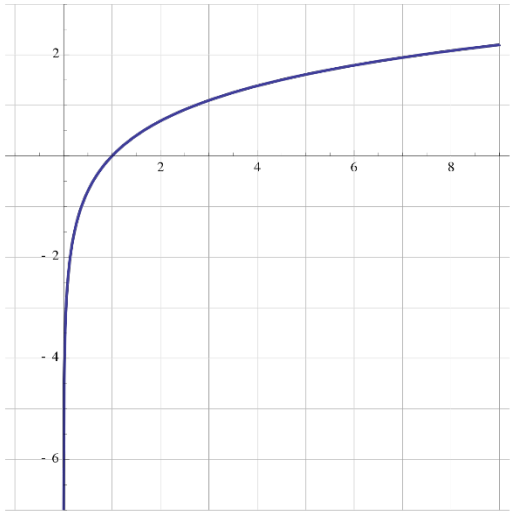
https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

Cross Entropy

Cross-Entropy:

$$crossentropy = -(1/n)(\sum_{i=1}^{3}(y_i \times \log(O_{outi})) + ((1 - y_i) \times \log((1 - O_{outi}))))$$

Cross-Entropy Formula



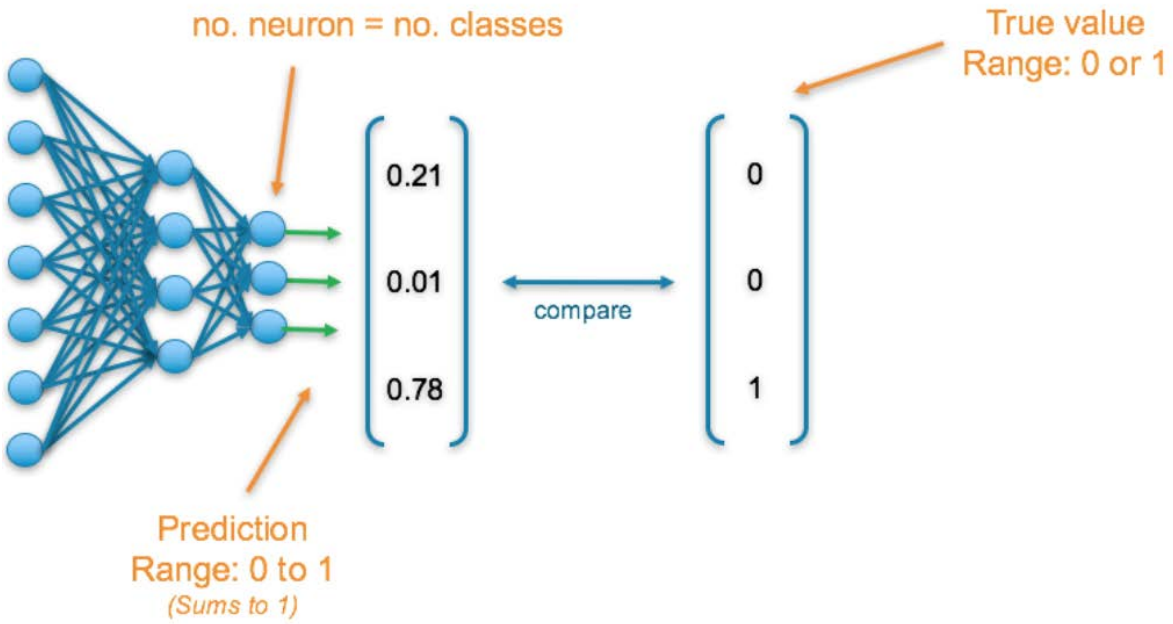
Example:

$$Error = -((1 * \log(0.2698) + 0 + 0 * \log(0.3223) + 1 * \log(1 - 0.3223) + 0 * \log(0.4078) + 1 * \log(1 - 0.4078))$$
$$Error = -\log(0.2698) - \log(0.6777) - \log(0.5922)$$
$$Error = +0.569858 + 0.16886 + 0.22753 = 0.985$$

Cross-Entropy calculation

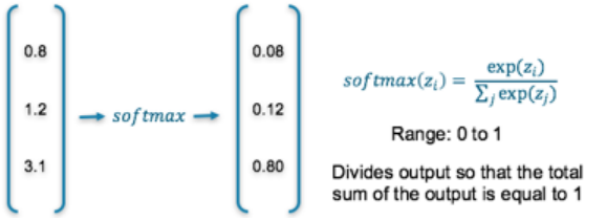
https://stackoverflow.com/questions/36515202/why-is-the-cross-entropy-method-preferred-over-mean-squared-error-in-what-cases

Loss Function (Continue)



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



Loss Function

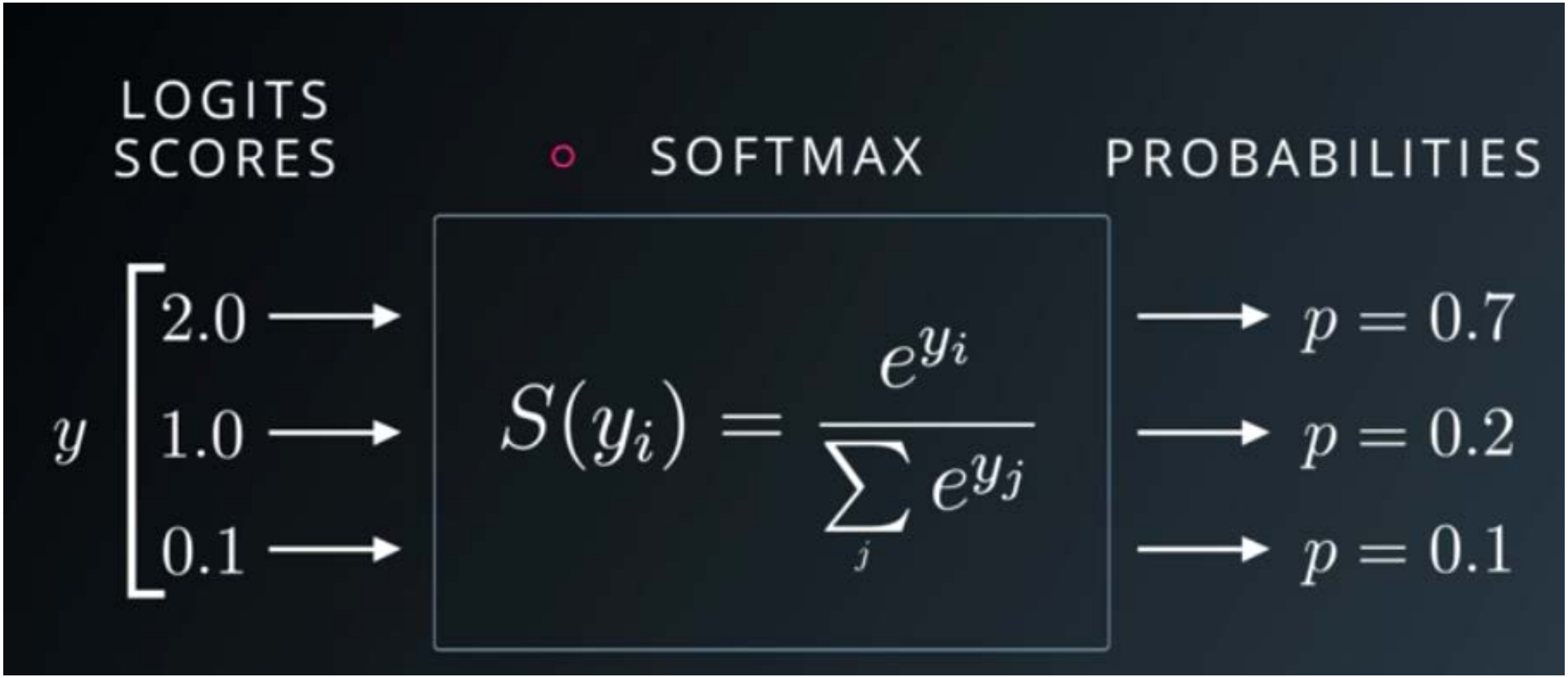
Cross Entropy—Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p1, p2, p3} (where p1+p2+p3 = 1). We use cross-entropy to compare this with the true distribution {y1, y2, y3}

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

https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

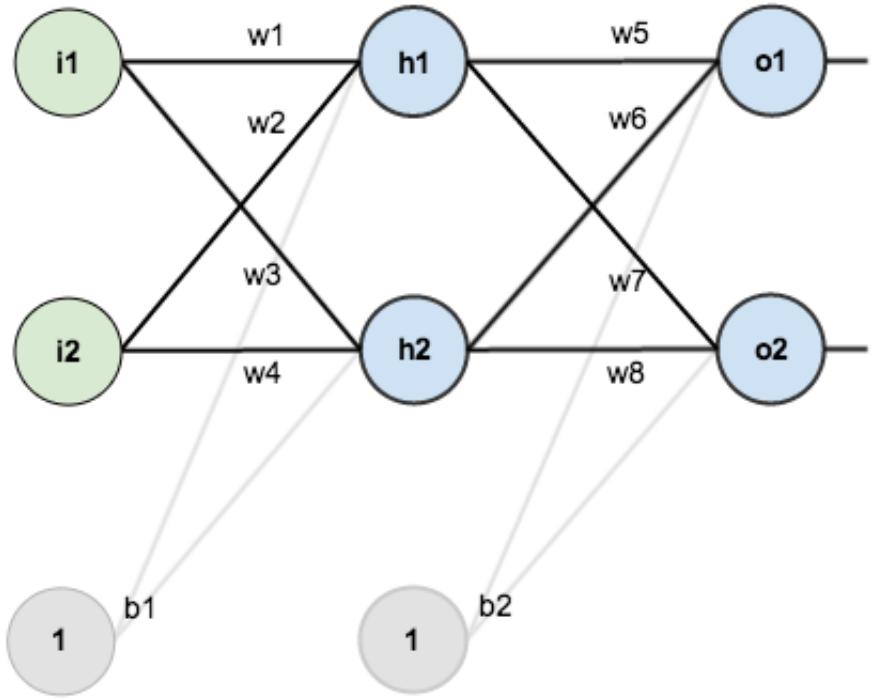
Softmax Function



https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

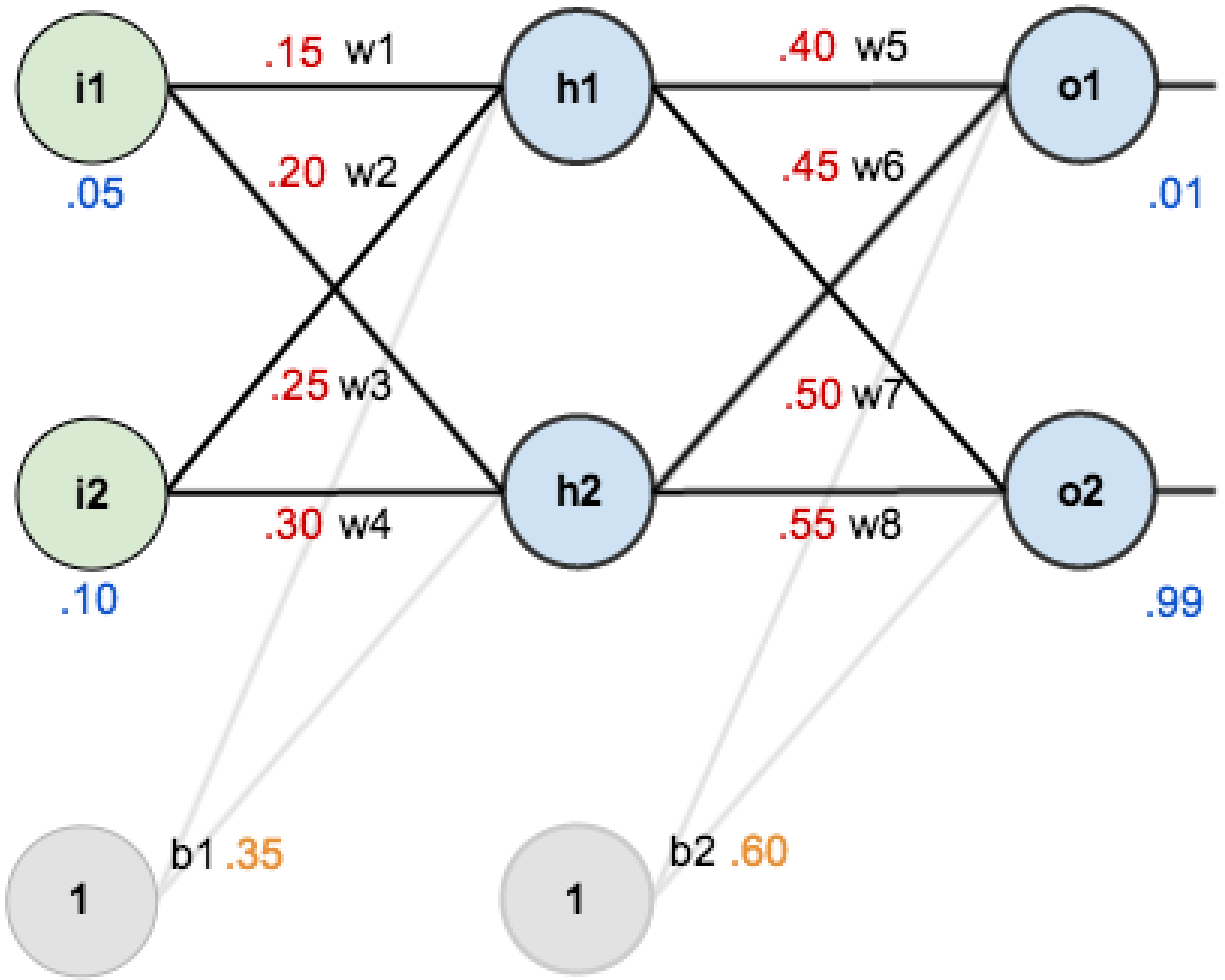
A Step-by-step Forward and Backward Propagation

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

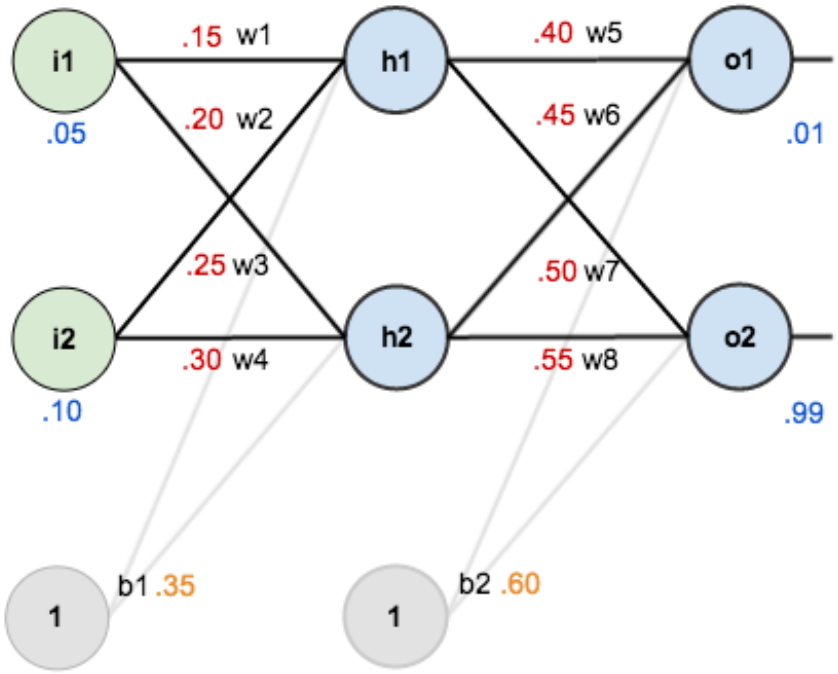


- One hidden layer (two hidden neurons)
- Two output neuron
- Two input neuron
- Activation function: Logistic function
- two input → two output

Initial Weights, Biases, and Training Inputs/Outputs



Forward Pass



$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

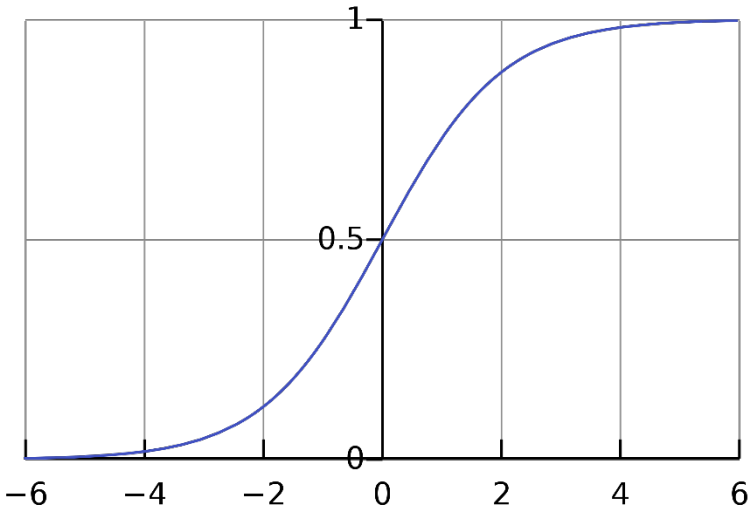
$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

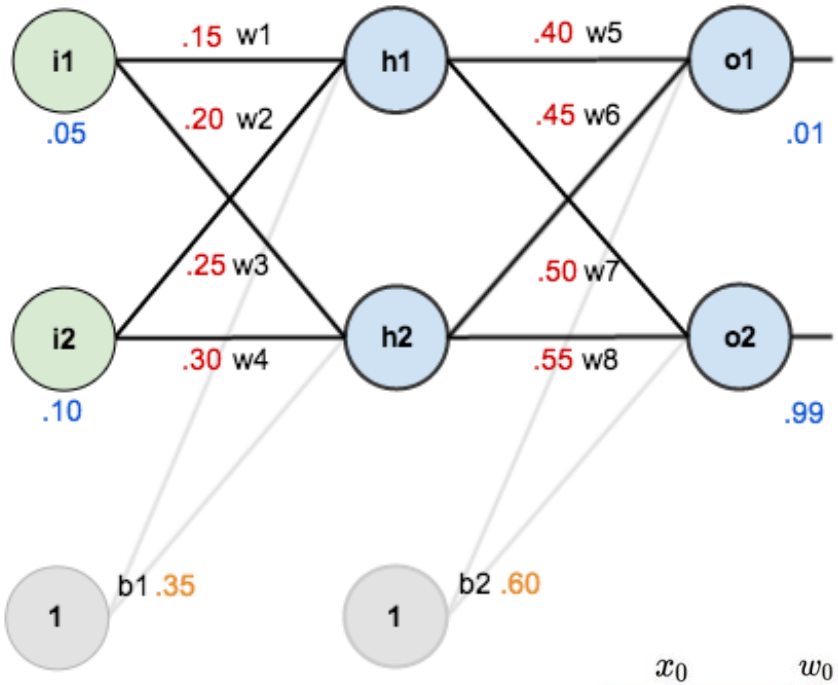
Logistic function

(sigmoid curve)

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



Forward Pass (Continue)

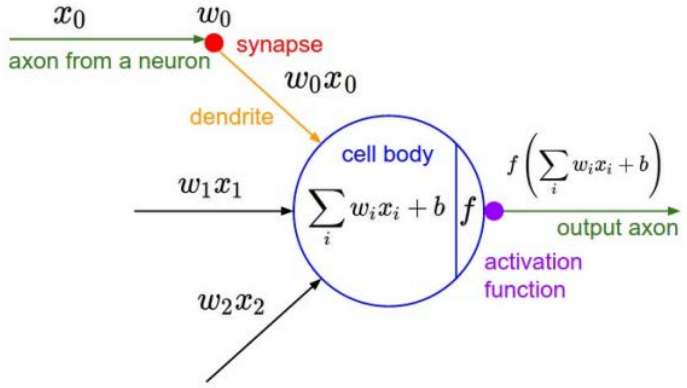


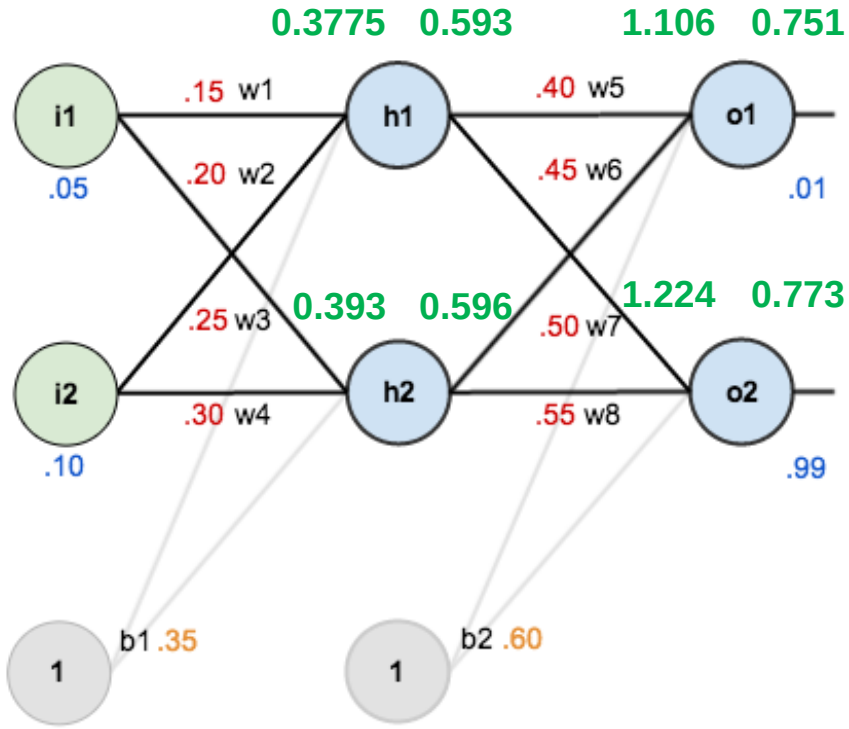
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

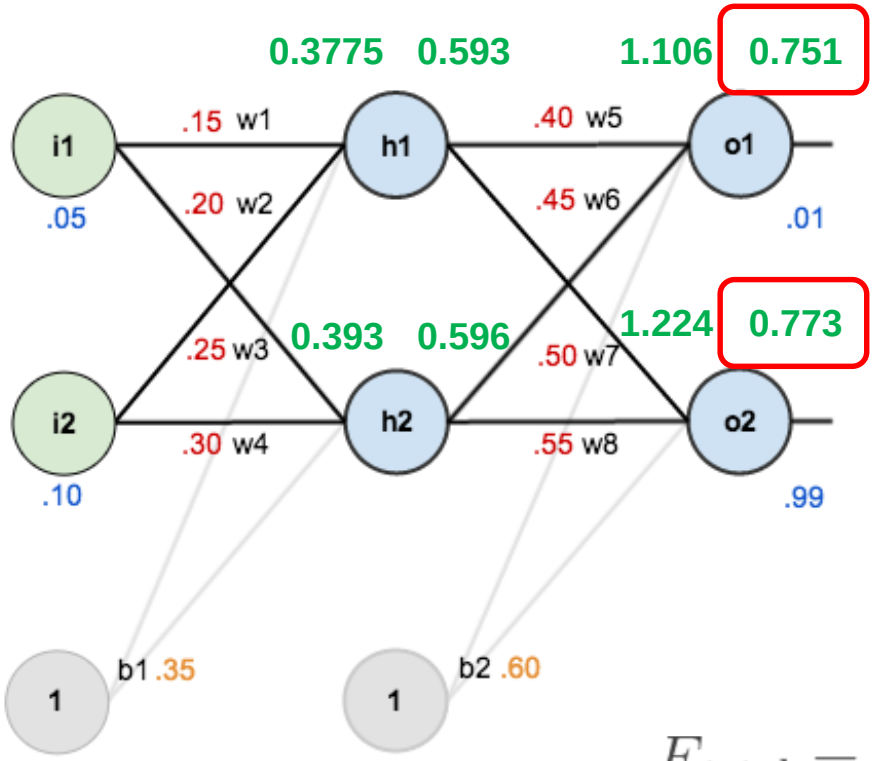
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$





Calculating the Total Error



$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

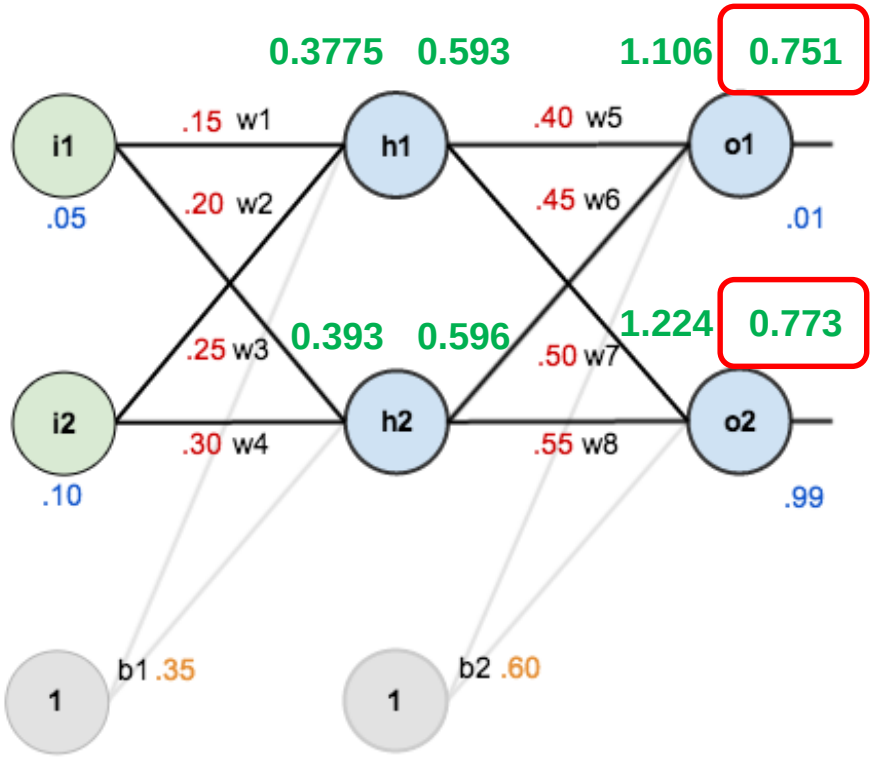
$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

End of a forward pass

Backward Pass



Our goal with backpropagation is to update each of the weights in the network so that **they cause the actual output to be closer the target output**, thereby minimizing the error for each output neuron and the network as a whole. Our goal is to make the values in the red boxes 0.01 and 0.99, respectively by changing (updating) the weights.

Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.

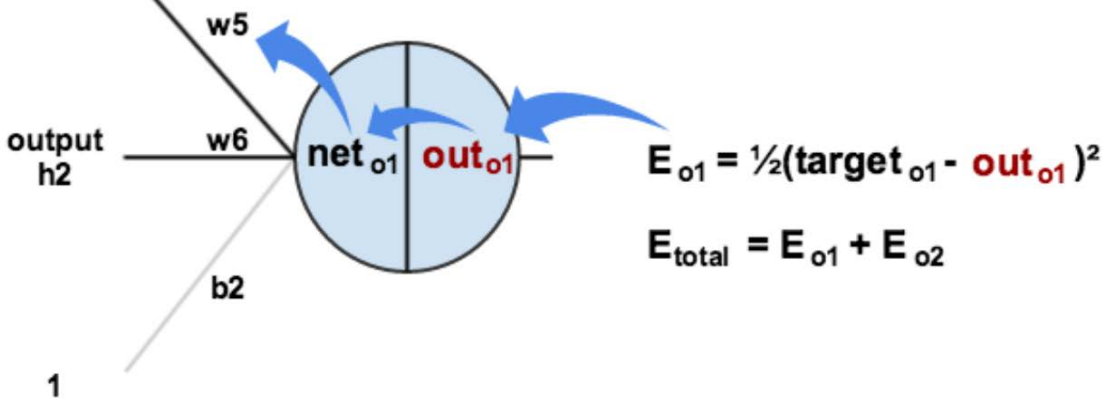
$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

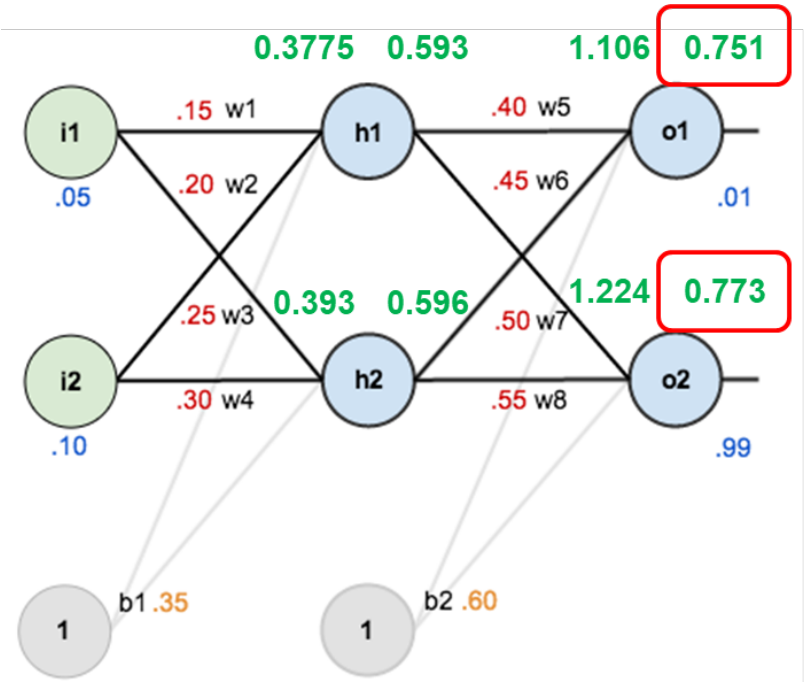
$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Chain rule

Output Layer (Continue)

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$





$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

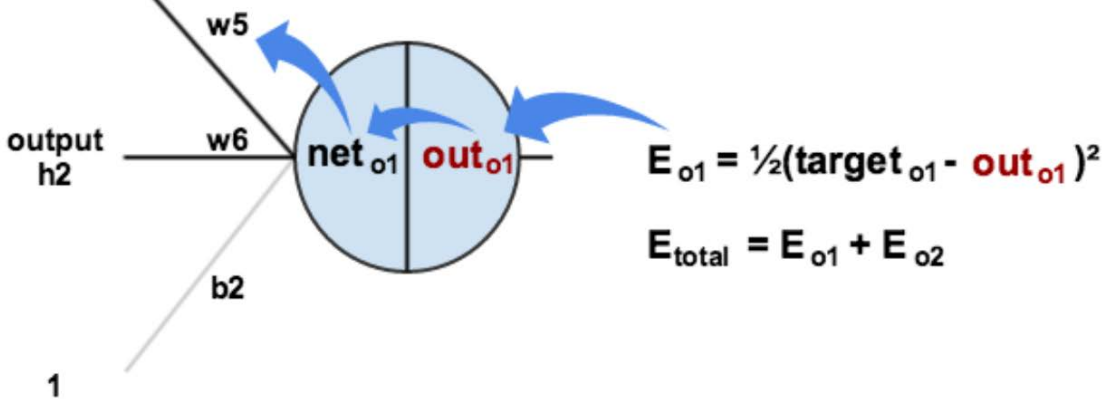
$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1}) \quad * -1 + 0$$

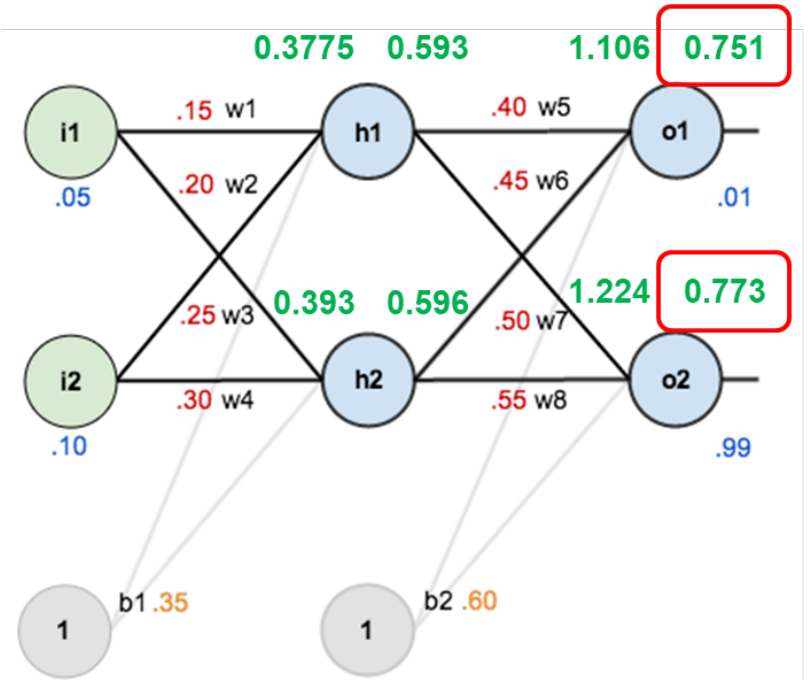
$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

When we take the partial derivative of the total error with respect to out_{o1}, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative of a constant which is zero.

Output Layer (Continue)

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$





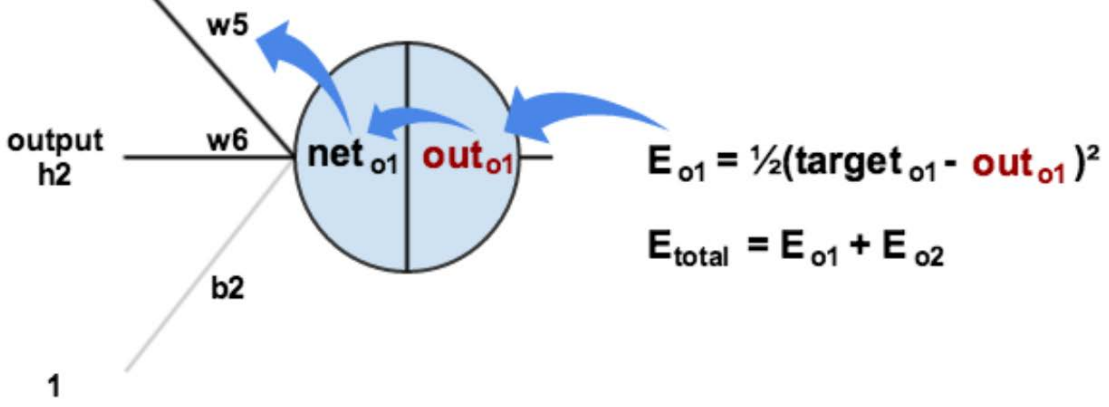
$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

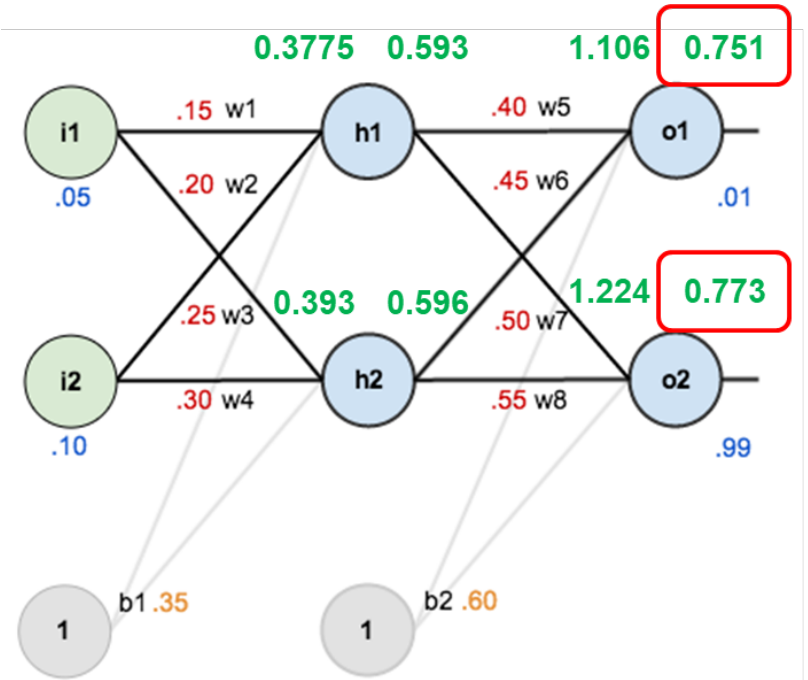
$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$

Output Layer (Continue)

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$





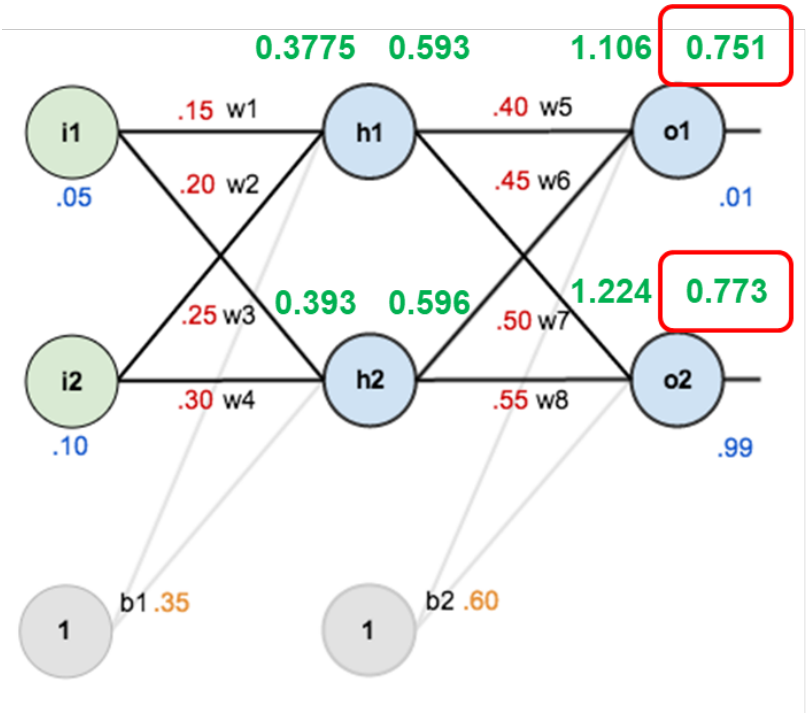
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

Output Layer (Continue)



$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

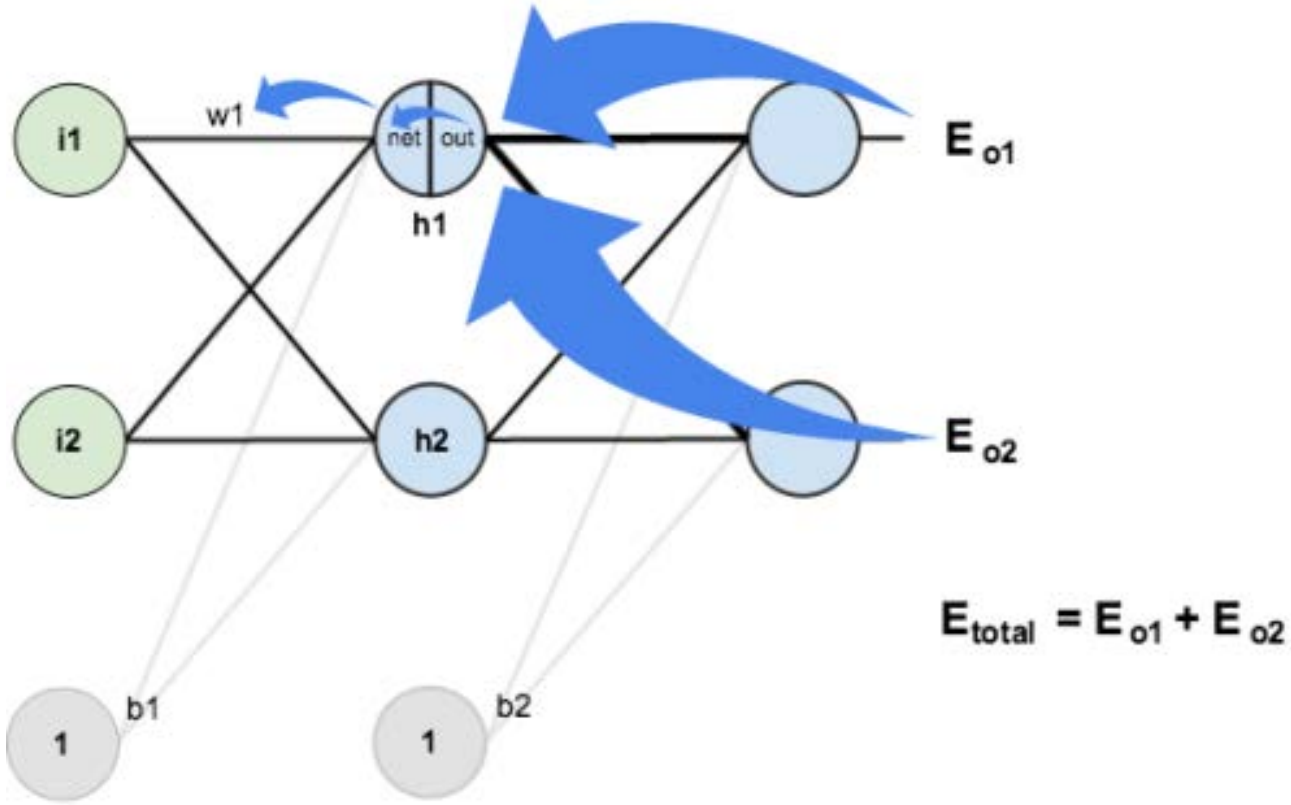
$$w_8^+ = 0.561370121$$

Some sources use α (alpha) to represent the learning rate, others use η (eta), and others even use ϵ (epsilon).

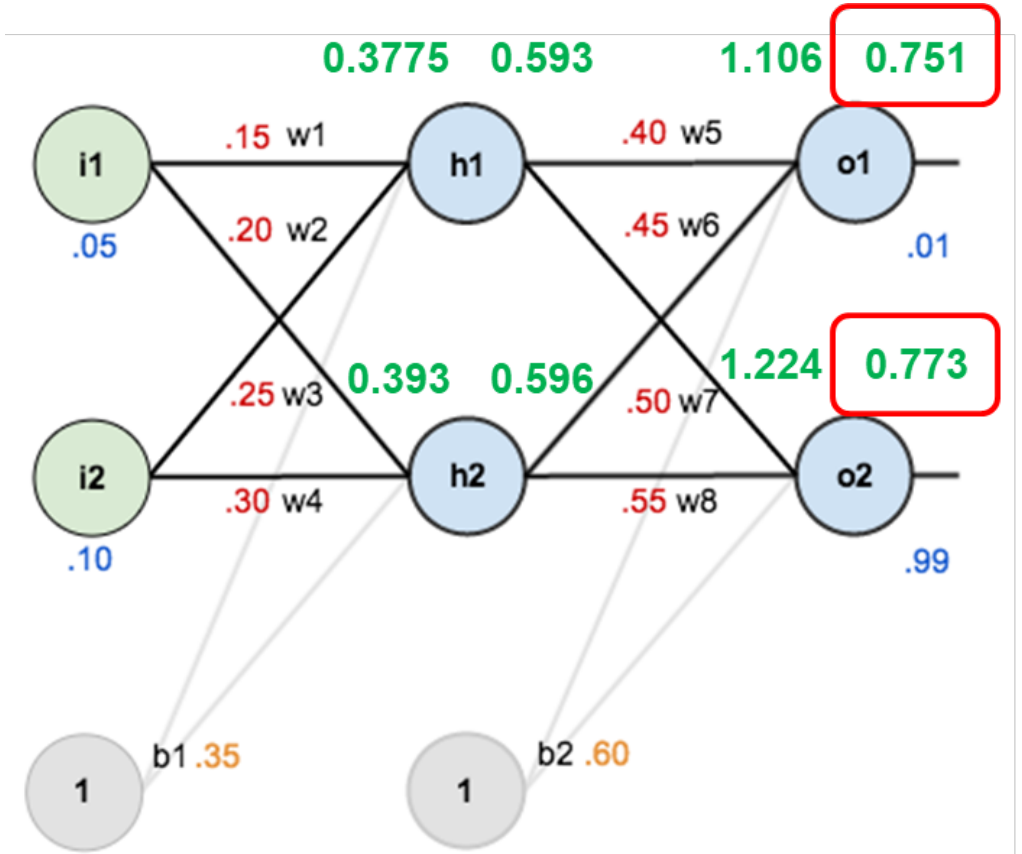
Hidden Layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$E_{total} = E_{o1} + E_{o2}$

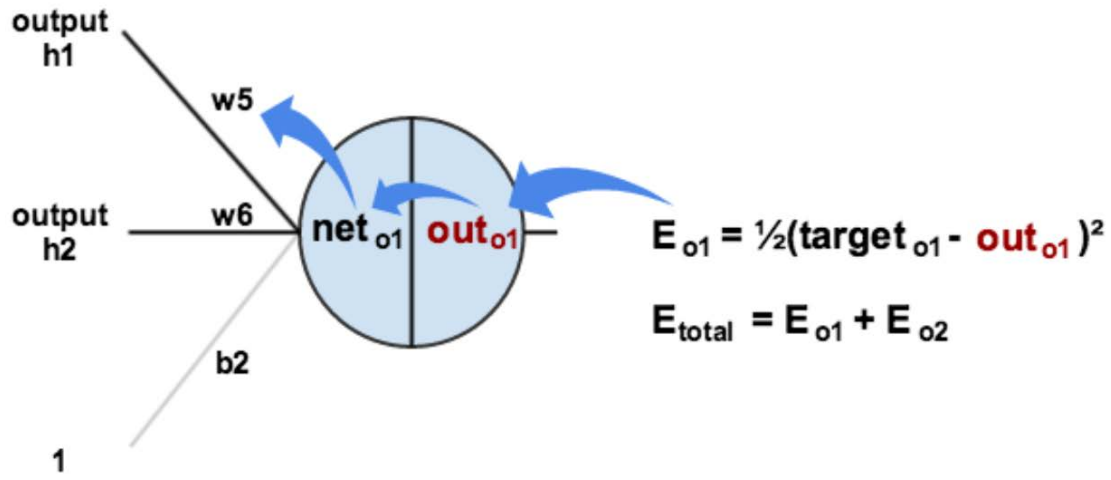


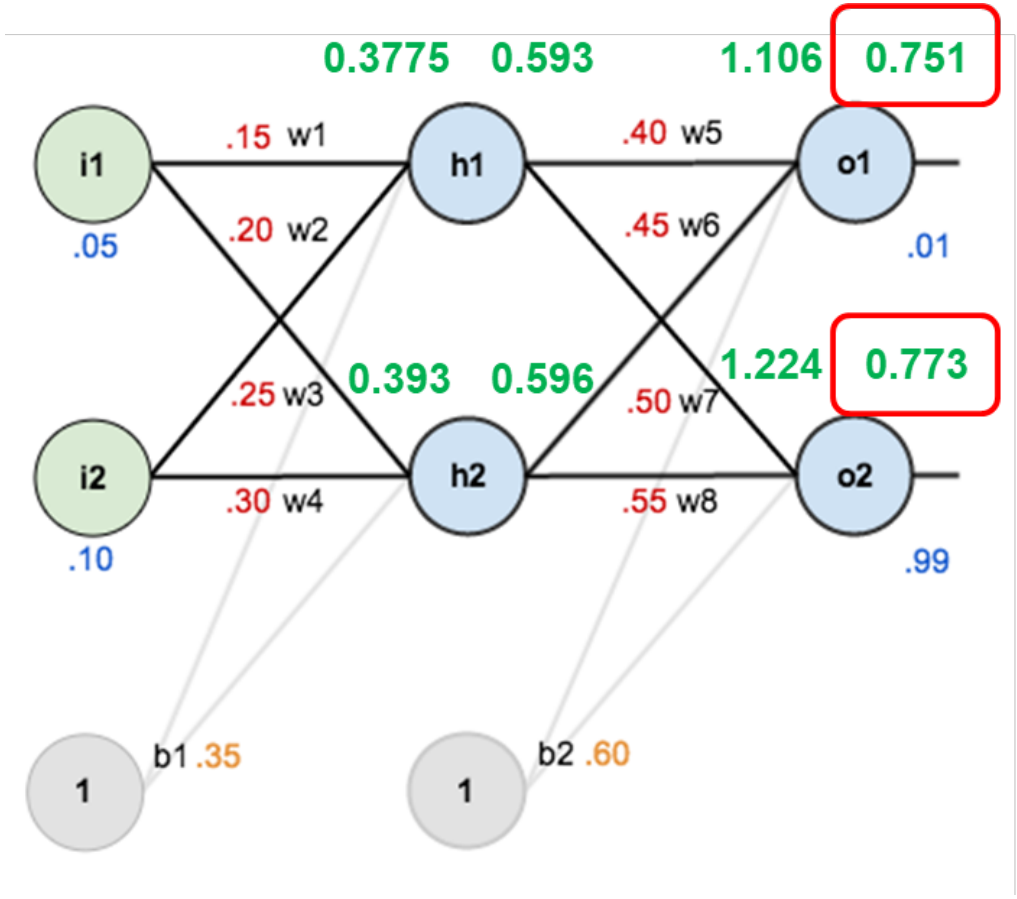
W1, W2, W3, and W4 contribute the E_{o1} and E_{o2}

What Did We Compute?

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial w_7}$$



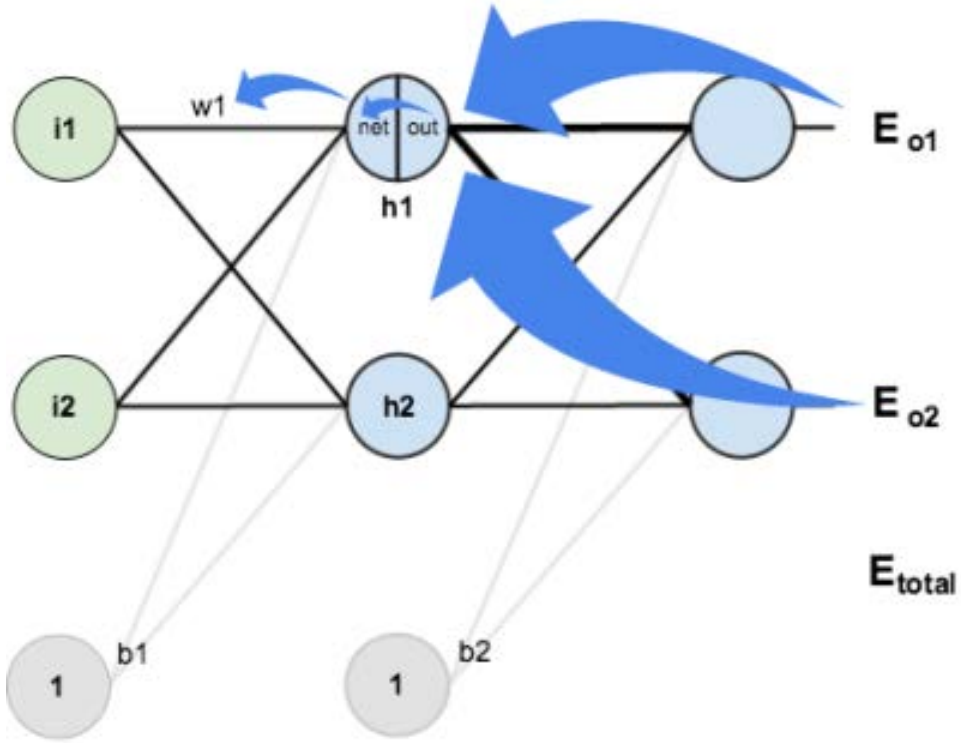


Hidden Layer (Continue)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

=

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

0.74136507 0.186815602

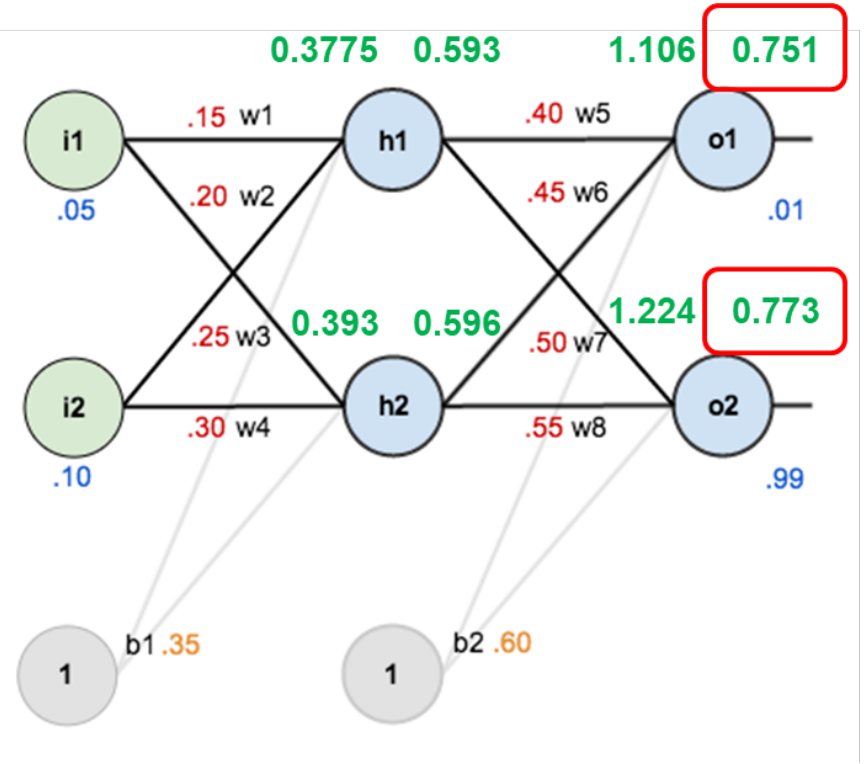
Hidden Layer (Continue)

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

II

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

0.74136507 0.186815602



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{E_{o1}}{\imath et_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Hidden Layer (Continue)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

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

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Hidden Layer (Continue)

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Complete One Iteration (Forward and Back Propagation)

We can now update w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

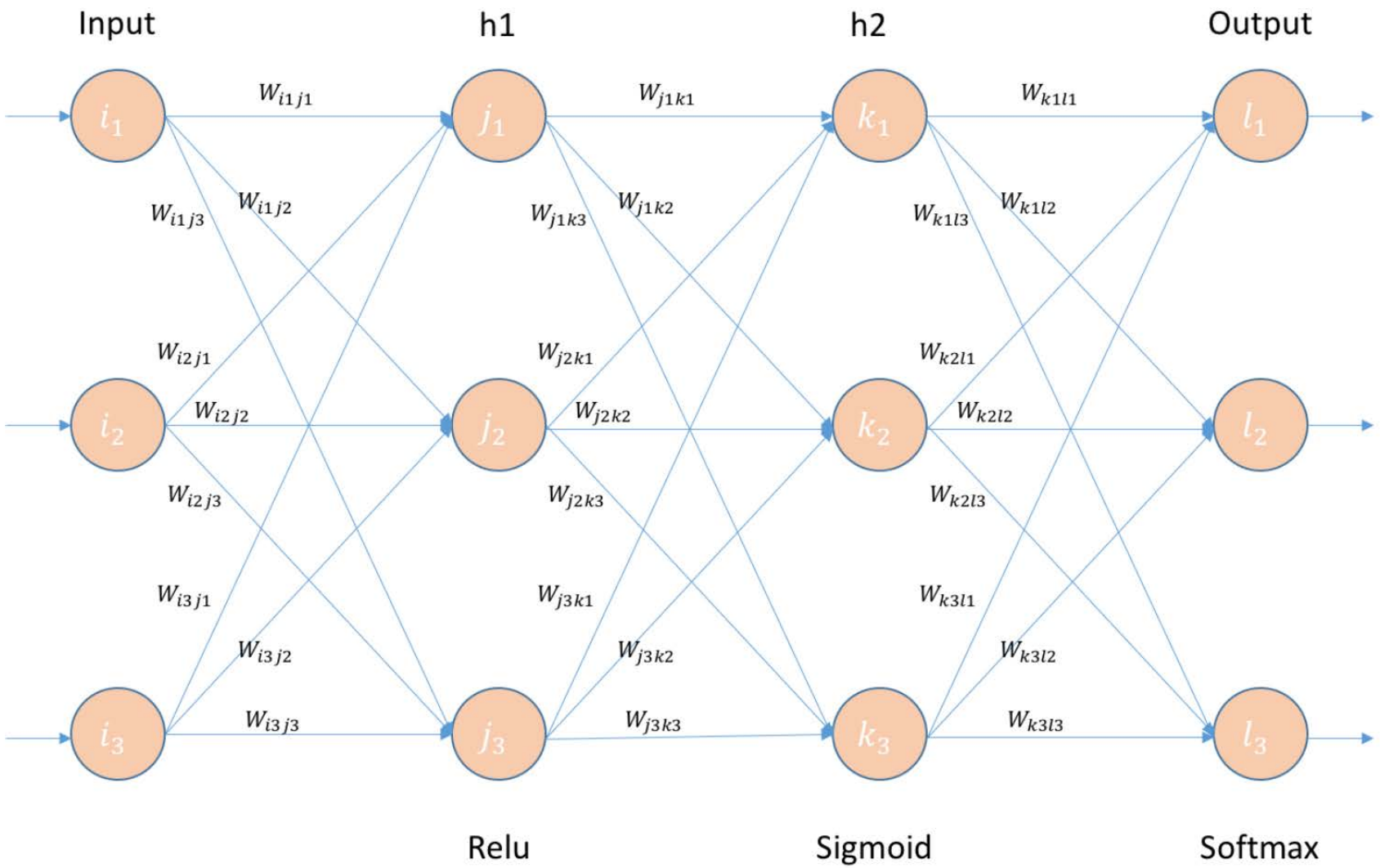
$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109. After this first round of backpropagation, the total error is now down to 0.291027924. It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

Some sources use α (alpha) to represent the learning rate, others use η (eta), and others even use ϵ (epsilon).

Another Example: A Step-by-step Forward and Backward Propagation



- Two hidden layer
- Three hidden neuron in each layer
- Three output neuron
- Three input neuron
- Different activation functions
- Three input → Three output

https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

Derivatives of Activation Functions

Sigmoid:

$$Sigmoid = 1/(1 + e^{-x})$$

$$\frac{\partial(1/(1 + e^{-x}))}{\partial x} = 1/(1 + e^{-x}) \times (1 - 1/(1 + e^{-x}))$$

$$\frac{\partial Sigmoid}{\partial x} = Sigmoid \times (1 - Sigmoid)$$

Derivative of Sigmoid

Relu:

$$relu = max(0, x)$$

$$if x > 0, \frac{\partial(relu)}{\partial x} = 1$$

$$Otherwise, \frac{\partial(relu)}{\partial x} = 0$$

Derivative of Relu

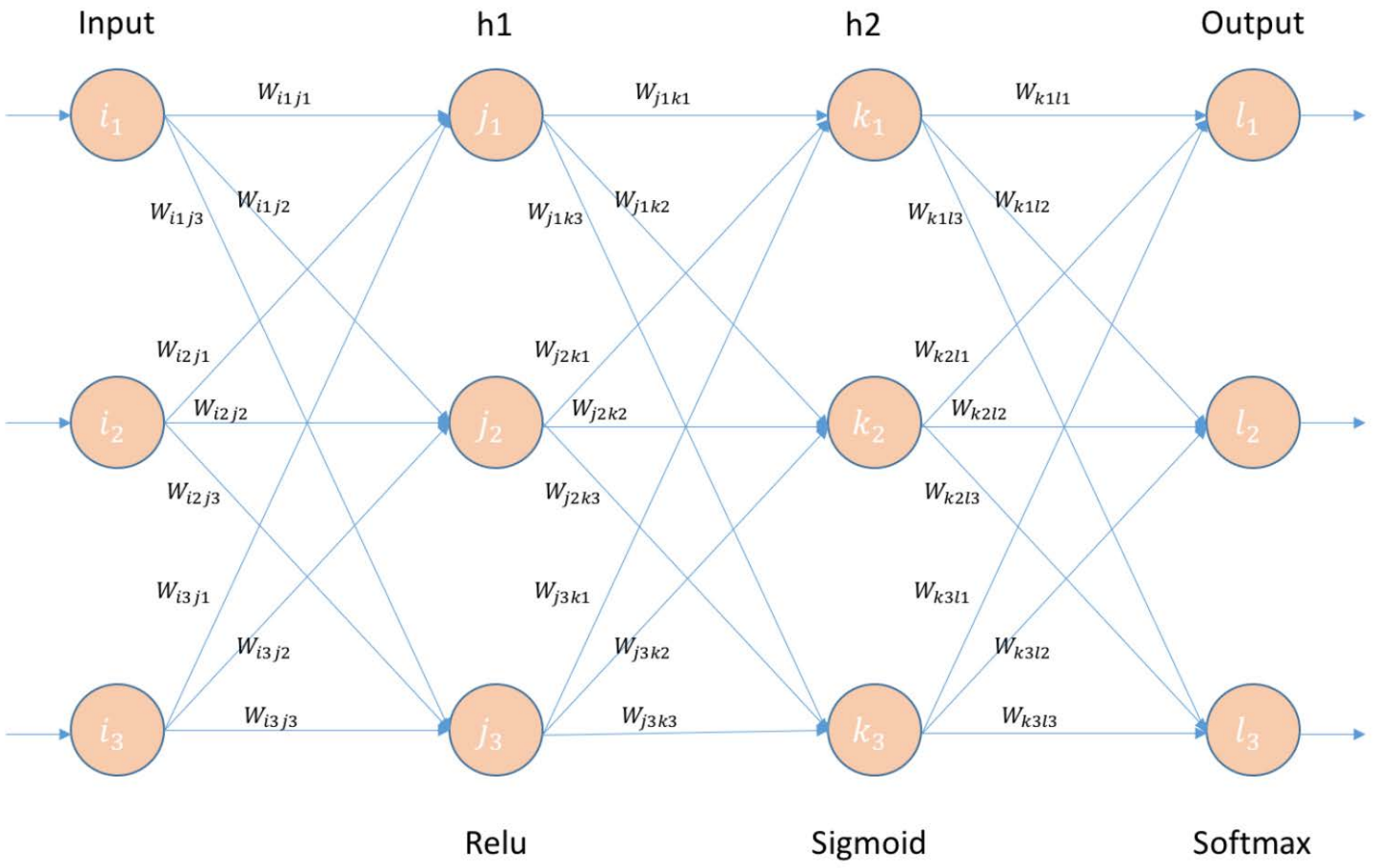
Softmax:

$$Softmax = e^{x_a}/(\sum_{a=1}^{n} e^{x_a}) = e^{x_1}/(e^{x_1} + e^{x_2} + e^{x_3})$$

$$\frac{\partial(Softmax)}{\partial x_1} = (e^{x_1} \times (e^{x_2} + e^{x_3}))/(e^{x_1} + e^{x_2} + e^{x_3})^2$$

Derivative of Softmax

Initializing Network



$$Input = \begin{bmatrix} 0.1 & 0.2 & 0.7 \end{bmatrix}$$

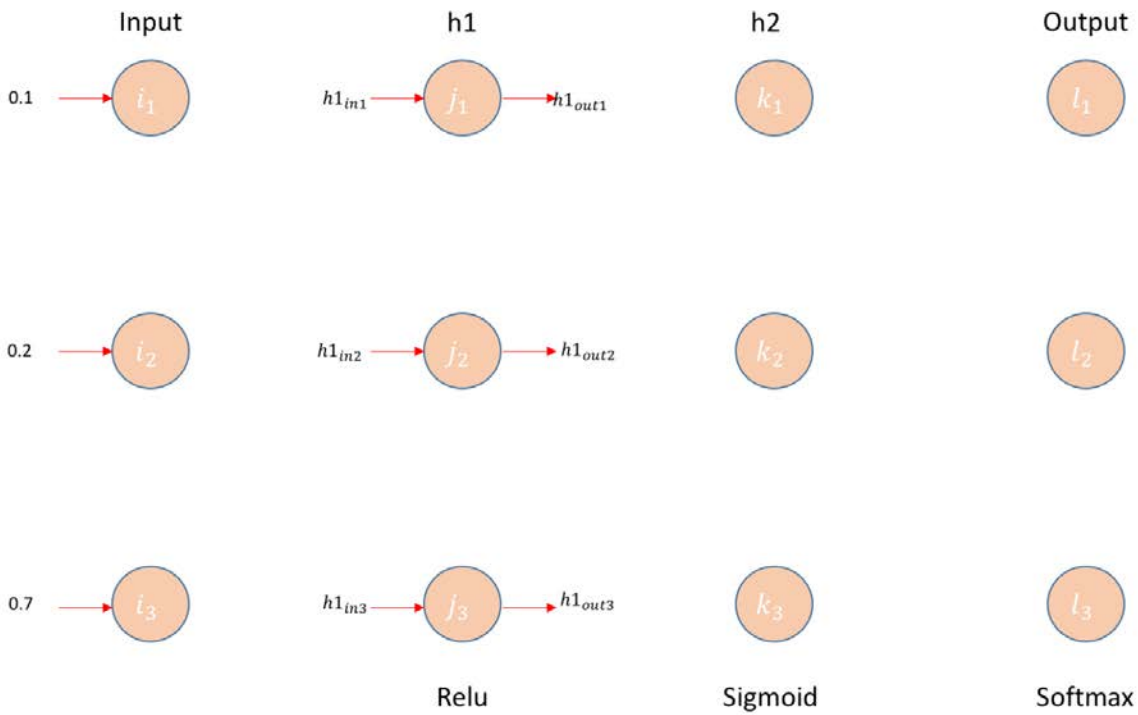
$$W_{ij} = \begin{bmatrix} W_{i1j1} & W_{i1j2} & W_{i1j3} \\ W_{i2j1} & W_{i2j2} & W_{i2j3} \\ W_{i3j1} & W_{i3j2} & W_{i3j3} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.3 & 0.2 & 0.7 \\ 0.4 & 0.3 & 0.9 \end{bmatrix}$$

$$W_{jk} = \begin{bmatrix} W_{j1k1} & W_{j1k2} & W_{j1k3} \\ W_{j2k1} & W_{j2k2} & W_{j2k3} \\ W_{j3k1} & W_{j3k2} & W_{j3k3} \end{bmatrix} = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.3 & 0.5 & 0.7 \\ 0.6 & 0.4 & 0.8 \end{bmatrix}$$

$$W_{kl} = \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix} = \begin{bmatrix} 0.1 & 0.4 & 0.8 \\ 0.3 & 0.7 & 0.2 \\ 0.5 & 0.2 & 0.9 \end{bmatrix}$$

$$Output = \begin{bmatrix} 1.0 & 0.0 & 0.0 \end{bmatrix}$$

Forward Propagation: Layer-1



Matrix Operation:

$$\begin{bmatrix} i_1 & i_2 & i_3 \end{bmatrix} \times \begin{bmatrix} W_{i1j1} & W_{i1j2} & W_{i1j3} \\ W_{i2j1} & W_{i2j2} & W_{i2j3} \\ W_{i3j1} & W_{i3j2} & W_{i3j3} \end{bmatrix} + \begin{bmatrix} b_{j1} & b_{j2} & b_{j3} \end{bmatrix} = \begin{bmatrix} h1_{in1} & h1_{in2} & h1_{in3} \end{bmatrix}$$

Layer-1 Matrix Operation

Relu operation:

$$relu = max(0, x)$$

$$\begin{bmatrix} h1_{out1} & h1_{out2} & h1_{out3} \end{bmatrix} = \begin{bmatrix} max(0, h1_{in1}) & max(0, h1_{in2}) & max(0, h1_{in3}) \end{bmatrix}$$

Layer-1 Relu Operation

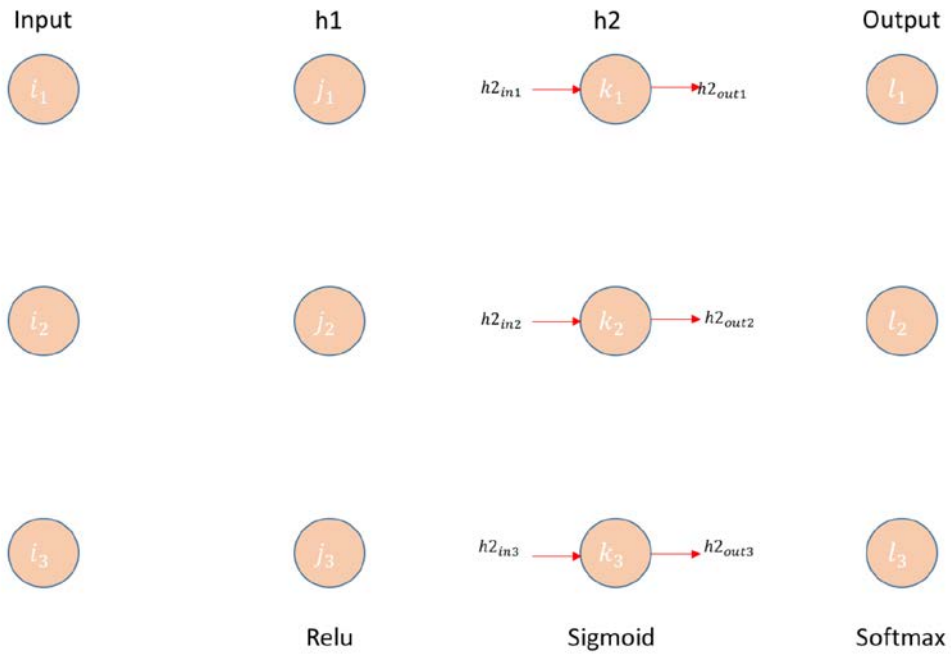
Example:

$$\begin{bmatrix} 0.1 & 0.2 & 0.7 \end{bmatrix} \times \begin{bmatrix} 0.1 & 0.4 & 0.3 \\ 0.3 & 0.7 & 0.7 \\ 0.5 & 0.2 & 0.9 \end{bmatrix} + \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} = \begin{bmatrix} 1.35 & 1.27 & 1.8 \end{bmatrix}$$

$$\begin{bmatrix} h1_{out1} & h1_{out2} & h1_{out3} \end{bmatrix} = \begin{bmatrix} 1.35 & 1.27 & 1.8 \end{bmatrix}$$

Layer-1 Example

Forward Propagation: Layer-2



Layer-2 Neural Network

Matrix operation:

$$\begin{bmatrix} h1_{out1} & h1_{out2} & h1_{out3} \end{bmatrix} \times \begin{bmatrix} W_{j1k1} & W_{j1k2} & W_{j1k3} \\ W_{j2k1} & W_{j2k2} & W_{j2k3} \\ W_{j3k1} & W_{j3k2} & W_{j3k3} \end{bmatrix} + \begin{bmatrix} b_{k1} & b_{k2} & b_{k3} \end{bmatrix} = \begin{bmatrix} h2_{in1} & h2_{in1} & h2_{in3} \end{bmatrix}$$

Layer-2 Matrix Operation

Sigmoid operation:

$$Sigmoid = 1/(1 + e^{-x})$$

$$\begin{bmatrix} h2_{out1} & h2_{out2} & h2_{out3} \end{bmatrix} = \begin{bmatrix} 1/(1 + e^{-h2_{in1}}) & 1/(1 + e^{-h2_{in2}}) & 1/(1 + e^{-h2_{in3}}) \end{bmatrix}$$

Sigmoid Operation

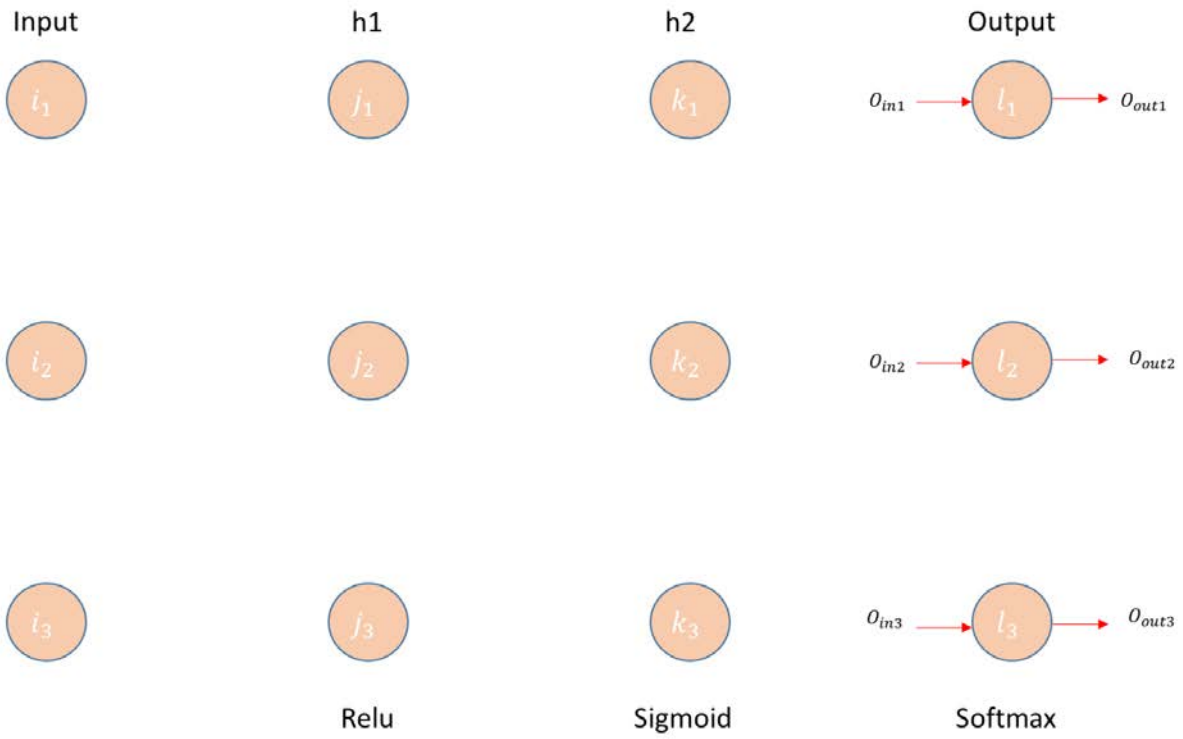
Example:

$$\begin{bmatrix} 1.35 & 1.27 & 1.8 \end{bmatrix} \times \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.3 & 0.5 & 0.7 \\ 0.6 & 0.4 & 0.8 \end{bmatrix} + \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} = \begin{bmatrix} 2.73 & 2.76 & 4.001 \end{bmatrix}$$

$$\begin{bmatrix} h2_{out1} & h2_{out2} & h2_{out3} \end{bmatrix} = \begin{bmatrix} 0.938 & 0.94 & 0.98 \end{bmatrix}$$

Layer-2 Example

Forward Propagation: Layer-3



Layer-3 Neural Network

Matrix operation:

$$\begin{bmatrix} h2_{out1} & h2_{out2} & h2_{out3} \end{bmatrix} \times \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix} + \begin{bmatrix} b_{l1} & b_{l2} & b_{l3} \end{bmatrix} = \begin{bmatrix} O_{in1} & O_{in2} & O_{in3} \end{bmatrix}$$

Layer-3 Matrix Operation

Softmax operation:

$$Softmax = e^{I_{ina}} / (\sum_{a=1}^{3} e^{O_{ina}})$$

$$\begin{bmatrix} O_{out1} & O_{out2} & O_{out3} \end{bmatrix} = \begin{bmatrix} e^{O_{in1}} / (\sum_{a=1}^{3} e^{O_{ina}}) & e^{O_{in2}} / (\sum_{a=1}^{3} e^{O_{ina}}) & e^{O_{in3}} / (\sum_{a=1}^{3} e^{O_{ina}}) \end{bmatrix}$$

Softmax formula

Example:

$$\begin{bmatrix} 0.938 & 0.94 & 0.98 \end{bmatrix} \times \begin{bmatrix} 0.1 & 0.4 & 0.8 \\ 0.3 & 0.7 & 0.2 \\ 0.5 & 0.2 & 0.9 \end{bmatrix} + \begin{bmatrix} 1.0 & 1.0 & 1.0 \end{bmatrix} = \begin{bmatrix} 1.8658 & 2.2292 & 2.8204 \end{bmatrix}$$

$$\begin{bmatrix} O_{out1} & O_{out2} & O_{out3} \end{bmatrix} = \begin{bmatrix} 0.2698 & 0.3223 & 0.4078 \end{bmatrix}$$

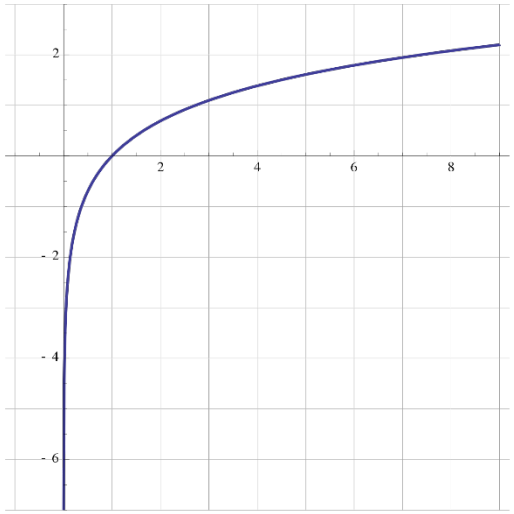
Layer-3 Output Example

Cross Entropy

Cross-Entropy:

$$crossentropy = -(1/n)(\sum_{i=1}^{3}(y_i \times \log(O_{outi})) + ((1 - y_i) \times \log((1 - O_{outi}))))$$

Cross-Entropy Formula



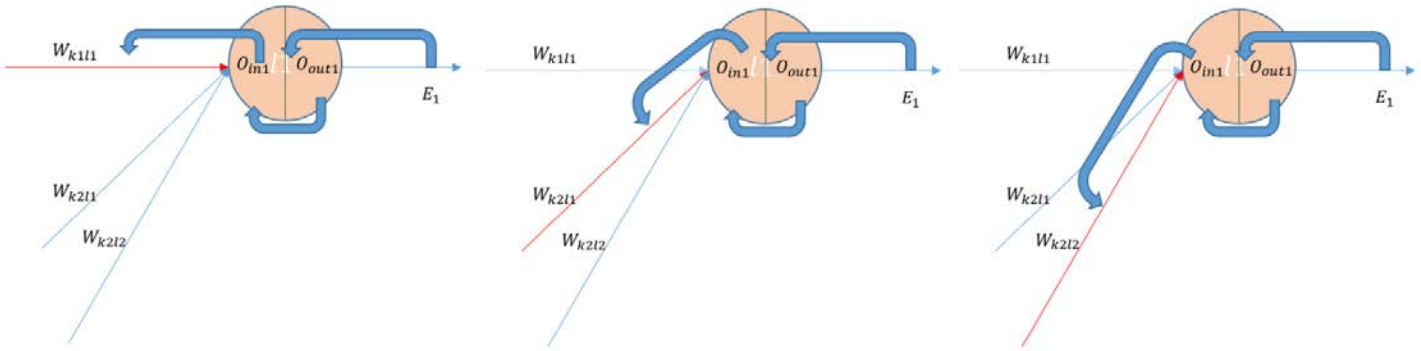
Example:

$$Error = -((1 * \log(0.2698) + 0 + 0 * \log(0.3223) + 1 * \log(1 - 0.3223) + 0 * \log(0.4078) + 1 * \log(1 - 0.4078))$$
$$Error = -\log(0.2698) - \log(0.6777) - \log(0.5922)$$
$$Error = +0.569858 + 0.16886 + 0.22753 = 0.985$$

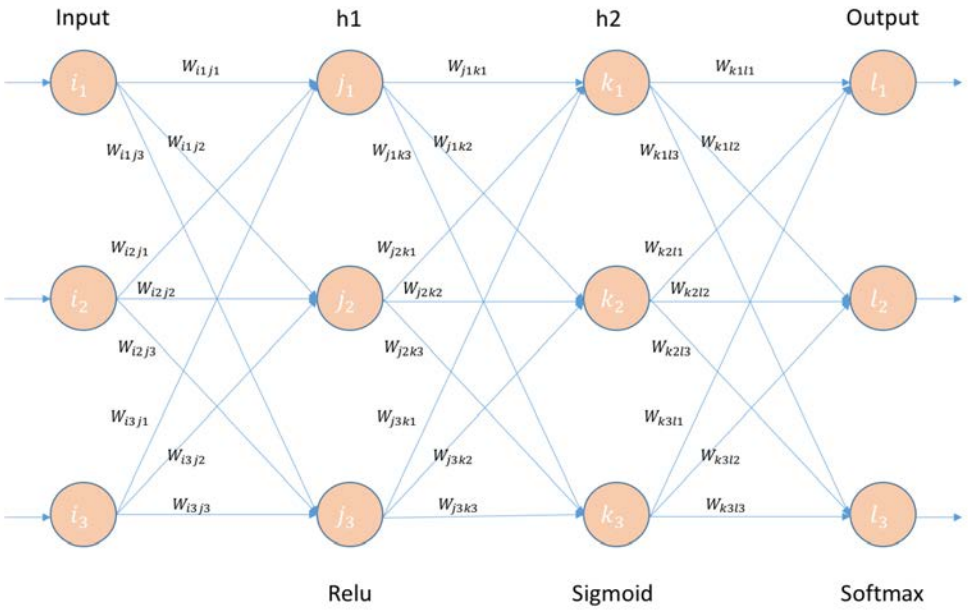
Cross-Entropy calculation

https://stackoverflow.com/questions/36515202/why-is-the-cross-entropy-method-preferred-over-mean-squared-error-in-what-cases

Back Propagation: Layer-2 – Output



Backpropagating Layer-3 weights



$$\frac{\partial E_1}{\partial O_{out1}} = \frac{\partial(-1 * ((y_1 * \log(O_{out1}) + (1 - y_1) * \log((1 - O_{out1}))}{\partial O_{out1}}$$

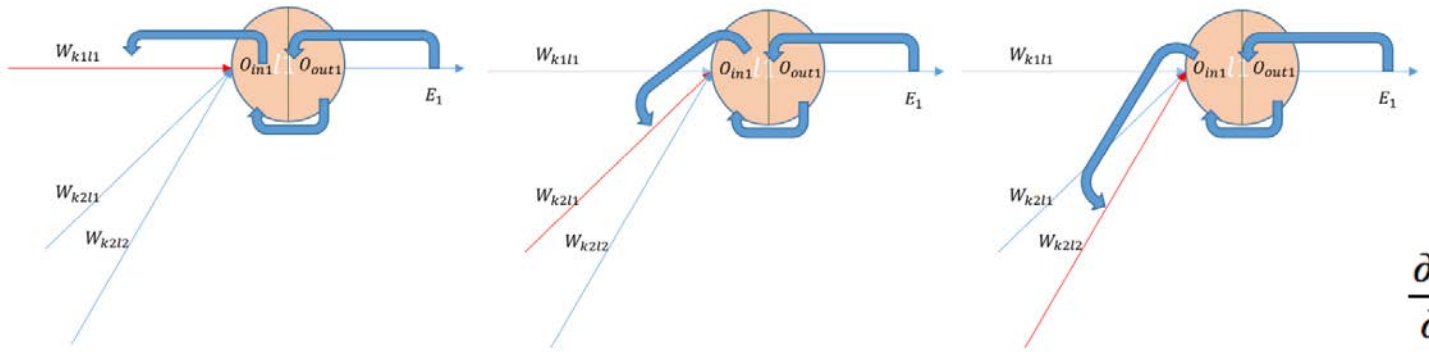
$$\frac{\partial E_1}{\partial O_{out1}} = -1 * ((y_1 * (1/O_{out1}) + (1 - y_1) * (1/(1 - O_{out1}))$$

Example: Derivative of Cross-Entropy

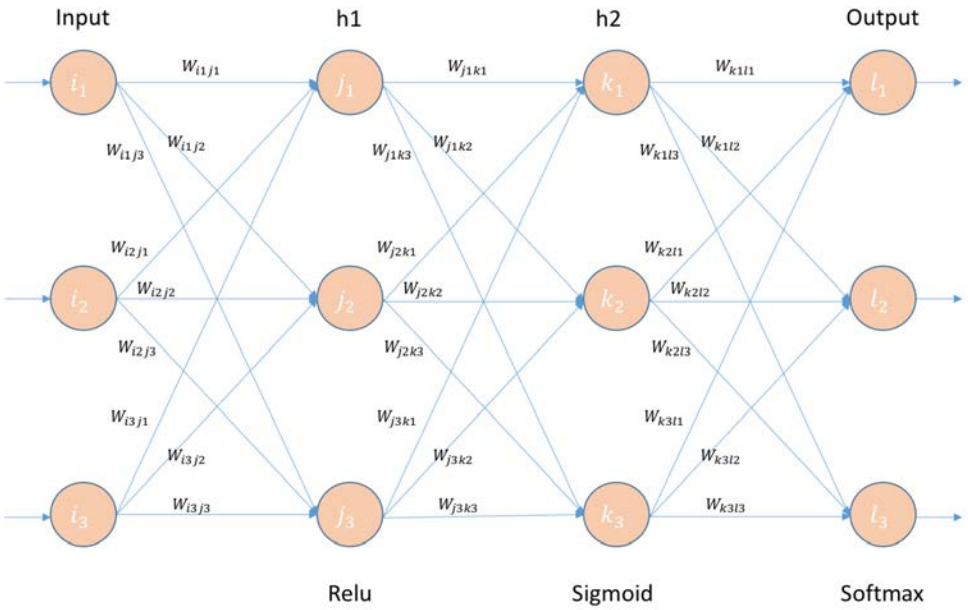
By symmetry we can calculate other derivatives also

$$\begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} \\ \frac{\partial E_2}{\partial O_{out2}} \\ \frac{\partial E_3}{\partial O_{out3}} \end{bmatrix} = \begin{bmatrix} -1 * ((y_1 * (1/O_{out1}) + (1 - y_1) * (1/(1 - O_{out1})) \\ -1 * ((y_2 * (1/O_{out2}) + (1 - y_2) * (1/(1 - O_{out2})) \\ -1 * ((y_3 * (1/O_{out3}) + (1 - y_3) * (1/(1 - O_{out3})) \end{bmatrix}$$

Back Propagation: Layer-2 – Output (Continue)



Backpropagating Layer-3 weights



$$\frac{\partial O_{out1}}{\partial O_{in1}} = \frac{\partial(e^{O_{in1}}/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}}))}{\partial O_{in1}}$$

$$\frac{\partial O_{out1}}{\partial O_{in1}} = (e^{O_{in1}} \times (e^{O_{in2}} + e^{O_{in3}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2$$

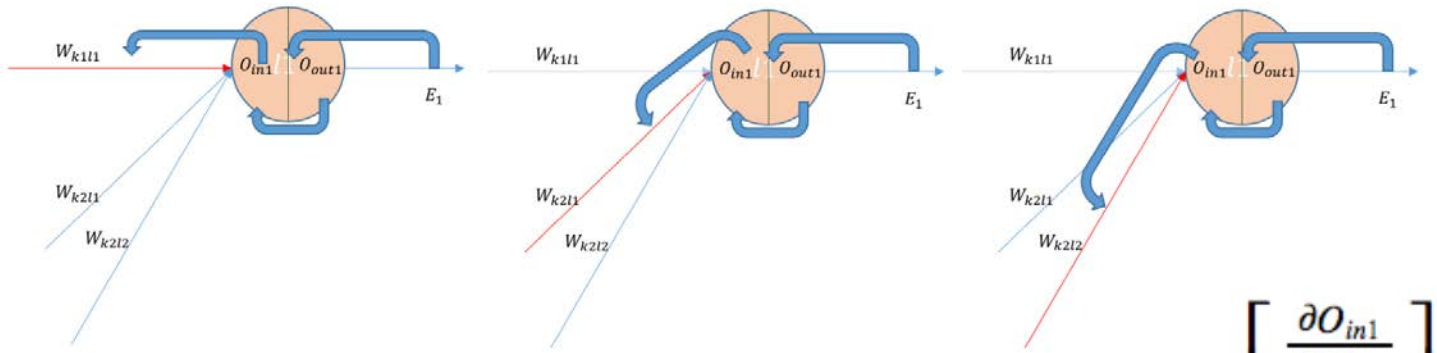
Example: Derivative of softmax wrt output layer input

By symmetry we can calculate other derivatives also

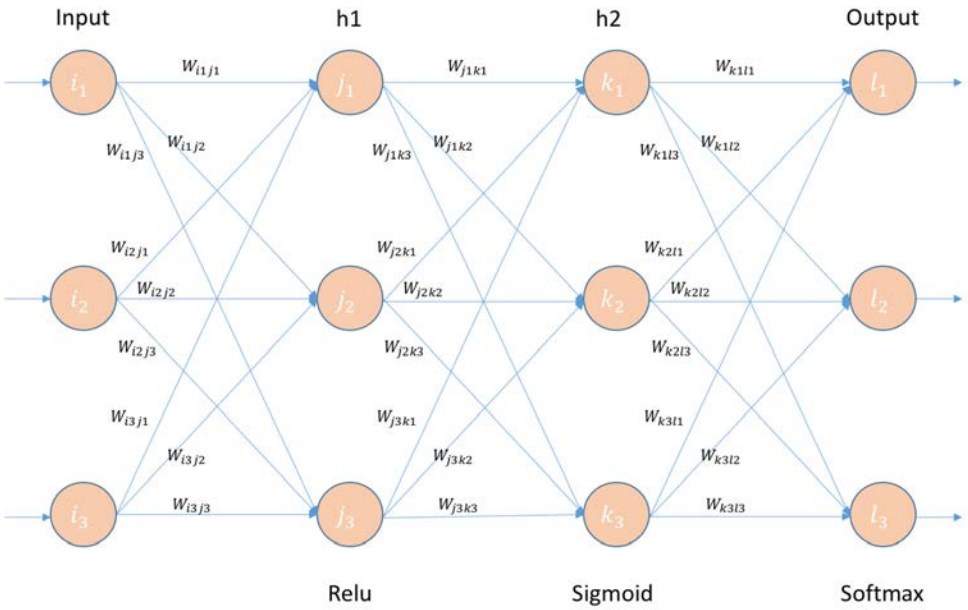
$$\begin{bmatrix} \frac{\partial O_{out1}}{\partial O_{in1}} \\ \frac{\partial O_{out2}}{\partial O_{in2}} \\ \frac{\partial O_{out3}}{\partial O_{in3}} \end{bmatrix} = \begin{bmatrix} (e^{O_{in1}} \times (e^{O_{in2}} + e^{O_{in3}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \\ (e^{O_{in2}} \times (e^{O_{in1}} + e^{O_{in3}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \\ (e^{O_{in3}} \times (e^{O_{in1}} + e^{O_{in2}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \end{bmatrix}$$

Matrix of Derivative of softmax wrt output layer input.

Back Propagation: Layer-2 – Output (Continue)



Backpropagating Layer-3 weights



$$\begin{bmatrix} \frac{\partial O_{in2}}{\partial W_{k1l2}} \\ \frac{\partial O_{in2}}{\partial W_{k2l2}} \\ \frac{\partial O_{in2}}{\partial W_{k3l2}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial O_{in1}}{\partial W_{k1l1}} \\ \frac{\partial O_{in1}}{\partial W_{k2l1}} \\ \frac{\partial O_{in1}}{\partial W_{k3l1}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix}$$

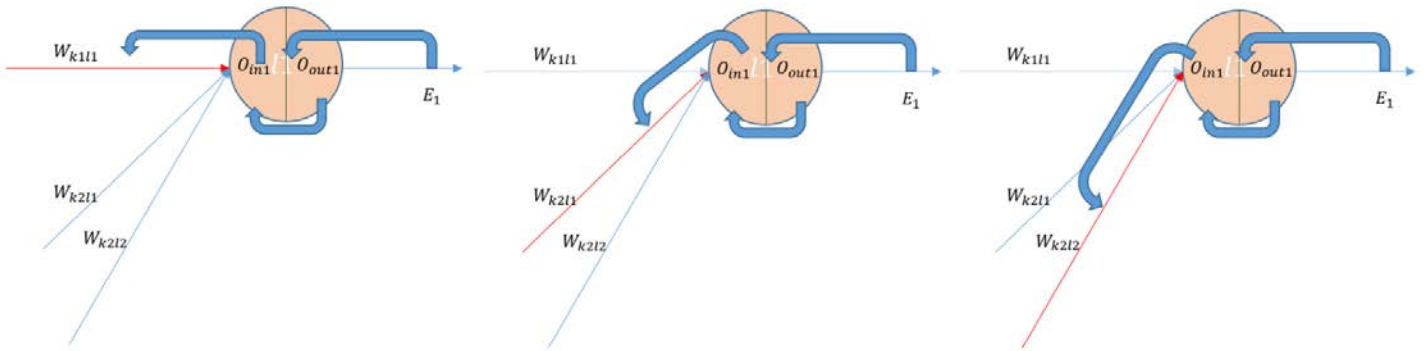
$$\begin{bmatrix} \frac{\partial O_{in3}}{\partial W_{k1l3}} \\ \frac{\partial O_{in3}}{\partial W_{k2l3}} \\ \frac{\partial O_{in3}}{\partial W_{k3l3}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix}$$

$$\frac{\partial O_{in1}}{\partial W_{k1l1}} = \frac{\partial((h2_{out1} * W_{j1k1}) + (h2_{out2} * Wj2k1) + (h2_{out3} * Wj3k1) + b_{l1})}{\partial W_{k1l1}}$$

$$\frac{\partial O_{in1}}{\partial W_{k1l1}} = h2_{out1}$$

Example: Derivative of input to output layer wrt weight

Back Propagation: Layer-2 – Output (Continue)



Backpropagating Layer-3 weights

$$\frac{\partial E_1}{\partial W_{k1l1}} = \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial W_{k1l1}}$$

$$\begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} \\ \frac{\partial E_2}{\partial O_{out2}} \\ \frac{\partial E_3}{\partial O_{out3}} \end{bmatrix} = \begin{bmatrix} -1 * ((y_1 * (1/O_{out1}) + (1 - y_1) * (1/(1 - O_{out1})) \\ -1 * ((y_2 * (1/O_{out2}) + (1 - y_2) * (1/(1 - O_{out2})) \\ -1 * ((y_3 * (1/O_{out3}) + (1 - y_3) * (1/(1 - O_{out3})) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial O_{out1}}{\partial O_{in1}} \\ \frac{\partial O_{out2}}{\partial O_{in2}} \\ \frac{\partial O_{out3}}{\partial O_{in3}} \end{bmatrix} = \begin{bmatrix} (e^{O_{in1}} \times (e^{O_{in2}} + e^{O_{in3}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \\ (e^{O_{in2}} \times (e^{O_{in1}} + e^{O_{in3}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \\ (e^{O_{in3}} \times (e^{O_{in1}} + e^{O_{in2}}))/(e^{O_{in1}} + e^{O_{in2}} + e^{O_{in3}})^2 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial O_{in1}}{\partial W_{k1l1}} \\ \frac{\partial O_{in1}}{\partial W_{k2l1}} \\ \frac{\partial O_{in1}}{\partial W_{k3l1}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{in2}}{\partial W_{k1l2}} \\ \frac{\partial O_{in2}}{\partial W_{k2l2}} \\ \frac{\partial O_{in2}}{\partial W_{k3l2}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{in3}}{\partial W_{k1l3}} \\ \frac{\partial O_{in3}}{\partial W_{k2l3}} \\ \frac{\partial O_{in3}}{\partial W_{k3l3}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix}$$

Back Propagation: Layer-2 – Output (Continue)

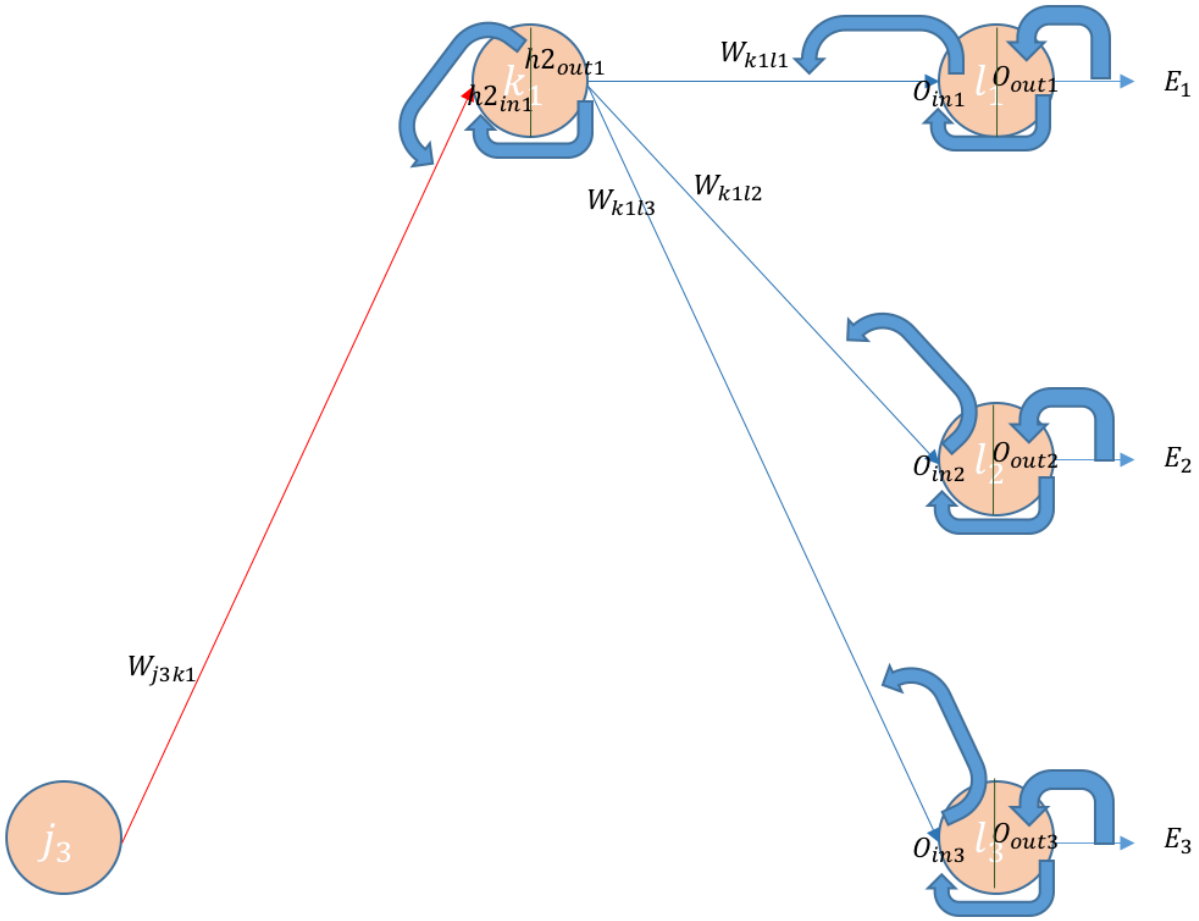
$$\frac{\partial E_1}{\partial W_{k1l1}} = \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial W_{k1l1}}$$

$$\delta W_{kl} = \begin{bmatrix} \frac{\partial E_1}{\partial W_{k1l1}} & \frac{\partial E_2}{\partial W_{k1l2}} & \frac{\partial E_3}{\partial W_{k1l3}} \\ \frac{\partial E_1}{\partial W_{k2l1}} & \frac{\partial E_2}{\partial W_{k2l2}} & \frac{\partial E_3}{\partial W_{k2l3}} \\ \frac{\partial E_1}{\partial W_{k3l1}} & \frac{\partial E_2}{\partial W_{k3l2}} & \frac{\partial E_3}{\partial W_{k3l3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial W_{k1l1}} & \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial W_{k1l2}} & \frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial W_{k1l3}} \\ \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial W_{k2l1}} & \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial W_{k2l2}} & \frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial W_{k2l3}} \\ \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial W_{k3l1}} & \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial W_{k3l2}} & \frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial W_{k3l3}} \end{bmatrix}$$

Matrix Form of all derivatives in layer-3

$$\begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} \\ \frac{\partial E_2}{\partial O_{out2}} \\ \frac{\partial E_3}{\partial O_{out3}} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{out1}}{\partial O_{in1}} \\ \frac{\partial O_{out2}}{\partial O_{in2}} \\ \frac{\partial O_{out3}}{\partial O_{in3}} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{in1}}{\partial W_{k1l1}} \\ \frac{\partial O_{in1}}{\partial W_{k2l1}} \\ \frac{\partial O_{in1}}{\partial W_{k3l1}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{in2}}{\partial W_{k1l2}} \\ \frac{\partial O_{in2}}{\partial W_{k2l2}} \\ \frac{\partial O_{in2}}{\partial W_{k3l2}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial O_{in3}}{\partial W_{k1l3}} \\ \frac{\partial O_{in3}}{\partial W_{k2l3}} \\ \frac{\partial O_{in3}}{\partial W_{k3l3}} \end{bmatrix} = \begin{bmatrix} h2_{out1} \\ h2_{out2} \\ h2_{out3} \end{bmatrix}$$

Back Propagation: Layer-1 – Layer 2



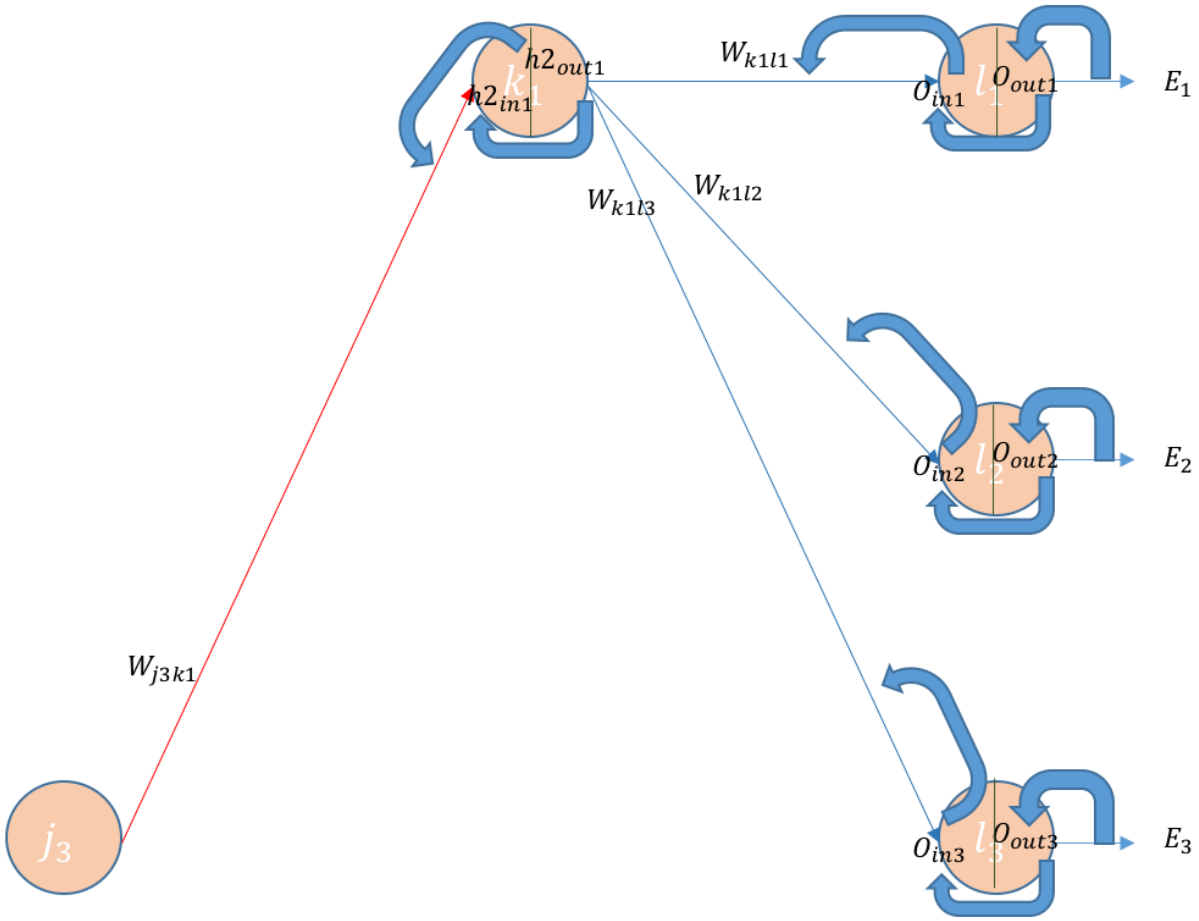
Softmax

$$\frac{\partial h2_{out1}}{\partial h2_{in1}} = \frac{\partial Sigmoid(h2_{in1})}{\partial h2_{in1}}$$

$$\frac{\partial h2_{out1}}{\partial h2_{in1}} = Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1}))$$

$$\begin{bmatrix} \frac{\partial h2_{out1}}{\partial h2_{in1}} \\ \frac{\partial h2_{out2}}{\partial h2_{in2}} \\ \frac{\partial h2_{out3}}{\partial h2_{in3}} \end{bmatrix} = \begin{bmatrix} Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1})) \\ Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1})) \\ Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1})) \end{bmatrix}$$

Back Propagation: Layer-1 – Layer 2 (Continue)



Softmax

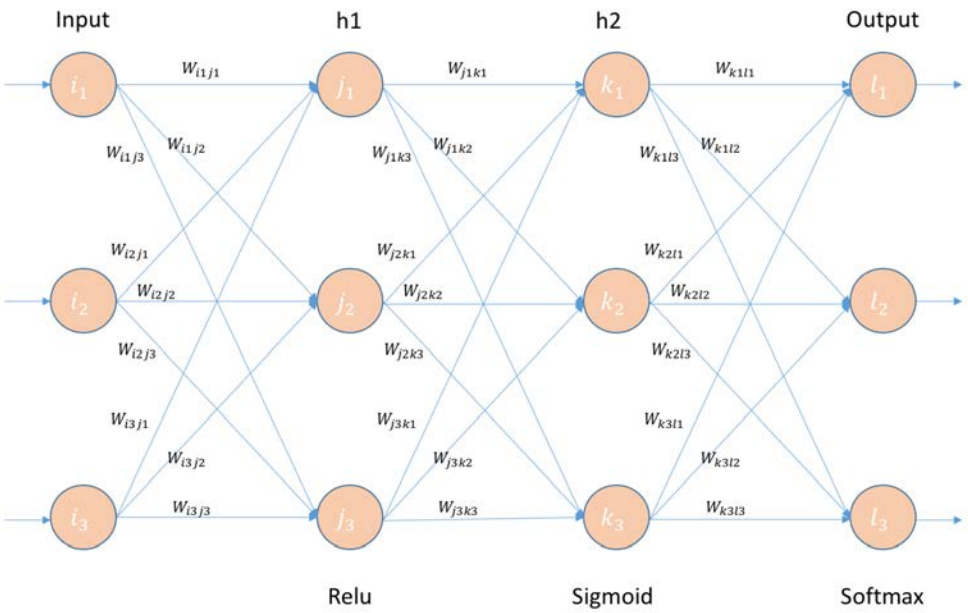
$$\frac{\partial h2_{in1}}{\partial W_{j1k1}} = \frac{\partial((h1_{out1} * W_{j1k1}) + (h1_{out2} * Wj2k1) + (h1_{out3} * Wj3k1) + b_{k1})}{\partial W_{j1k1}}$$

$$\frac{\partial h2_{in1}}{\partial W_{j1k1}} = h1_{out1}$$

$$\begin{bmatrix} \frac{\partial h2_{in1}}{\partial W_{j1k1}} \\ \frac{\partial h2_{in1}}{\partial W_{j2k1}} \\ \frac{\partial h2_{in1}}{\partial W_{j3k1}} \end{bmatrix} = \begin{bmatrix} h1_{out1} \\ h1_{out2} \\ h1_{out3} \end{bmatrix} \quad \begin{bmatrix} \frac{\partial h2_{in2}}{\partial W_{j1k2}} \\ \frac{\partial h2_{in2}}{\partial W_{j2k2}} \\ \frac{\partial h2_{in2}}{\partial W_{j3k2}} \end{bmatrix} = \begin{bmatrix} h1_{out1} \\ h1_{out2} \\ h1_{out3} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial h2_{in3}}{\partial W_{j1k3}} \\ \frac{\partial h2_{in3}}{\partial W_{j2k3}} \\ \frac{\partial h2_{in3}}{\partial W_{j3k3}} \end{bmatrix} = \begin{bmatrix} h1_{out1} \\ h1_{out2} \\ h1_{out3} \end{bmatrix}$$

Back Propagation: Layer-1 – Layer 2 (Continue)



$$\frac{\partial E_{total}}{\partial W_{j3k1}} = \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j3k1}}$$

$$\delta W_{jk} = \begin{bmatrix} \frac{\partial E_{total}}{\partial W_{j1k1}} & \frac{\partial E_{total}}{\partial W_{j1k2}} & \frac{\partial E_{total}}{\partial W_{j1k3}} \\ \frac{\partial E_{total}}{\partial W_{j2k1}} & \frac{\partial E_{total}}{\partial W_{j2k2}} & \frac{\partial E_{total}}{\partial W_{j2k3}} \\ \frac{\partial E_{total}}{\partial W_{j3k1}} & \frac{\partial E_{total}}{\partial W_{j3k2}} & \frac{\partial E_{total}}{\partial W_{j3k3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j1k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j1k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j1k3}} \\ \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j2k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j2k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j2k3}} \\ \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j3k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j3k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j3k3}} \end{bmatrix}$$

Back Propagation: Layer-1 – Layer 2 (Continue)

$$\frac{\partial E_{total}}{\partial h2_{out1}} = \frac{\partial E_1}{\partial h2_{out1}} + \frac{\partial E_2}{\partial h2_{out1}} + \frac{\partial E_3}{\partial h2_{out1}}$$

Breakdown of error.

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} \\ \frac{\partial E_{total}}{\partial h2_{out2}} \\ \frac{\partial E_{total}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial h2_{out1}}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial h2_{out1}}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial h2_{out1}}) \\ (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial h2_{out2}}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial h2_{out2}}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial h2_{out2}}) \\ (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial h2_{out3}}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial h2_{out3}}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial h2_{out3}}) \end{bmatrix}$$

Derivative of error wrt output of hidden layer 2

$$\frac{\partial E_1}{\partial h2_{out1}} = \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial h2_{out1}}$$

$$\frac{\partial E_2}{\partial h2_{out1}} = \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial h2_{out1}}$$

$$\frac{\partial E_3}{\partial h2_{out1}} = \frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * \frac{\partial O_{in3}}{\partial h2_{out1}}$$

breakdown of each error derivative

$$\begin{bmatrix} \frac{\partial O_{inl}}{\partial h2_{out1}} & \frac{\partial O_{in2}}{\partial h2_{out1}} & \frac{\partial O_{in3}}{\partial h2_{out1}} \\ \frac{\partial O_{inl}}{\partial h2_{out2}} & \frac{\partial O_{in2}}{\partial h2_{out2}} & \frac{\partial O_{in3}}{\partial h2_{out2}} \\ \frac{\partial O_{inl}}{\partial h2_{out3}} & \frac{\partial O_{in2}}{\partial h2_{out3}} & \frac{\partial O_{in3}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix}$$

Derivative of input of output layer wrt hidden layer -2

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} \\ \frac{\partial E_{total}}{\partial h2_{out2}} \\ \frac{\partial E_{total}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * W_{k1l1}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * W_{k1l2}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * W_{k1l3}) \\ (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * W_{k2l1}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * W_{k2l2}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * W_{k2l3}) \\ (\frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * W_{k3l1}) + (\frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * W_{k3l2}) + (\frac{\partial E_3}{\partial O_{out3}} * \frac{\partial O_{out3}}{\partial O_{in3}} * W_{k2l3}) \end{bmatrix}$$

Gradient Descent Variants

Batch gradient descent: Batch Gradient Descent is when we sum up over all examples on each iteration when performing the updates to the parameters. Therefore, for each update, we have to sum over all examples:

$$w = w - \alpha \nabla_w J(w)$$

Mini-batch gradient descent: Instead of going over all examples, Mini-batch Gradient Descent sums up over lower number of examples based on the batch size. Therefore, learning happens on each mini-batch of b examples:

$$w = w - \alpha \nabla_w J(x^{\{i:i+b\}}, y^{\{i:i+b\}}; w)$$

Stochastic gradient descent: Instead of going through all examples, Stochastic Gradient Descent (SGD) performs the parameters update on each example (x^i,y^i). Therefore, learning happens on every example:

$$w = w - \alpha \nabla_w J(x^i, y^i; w)$$

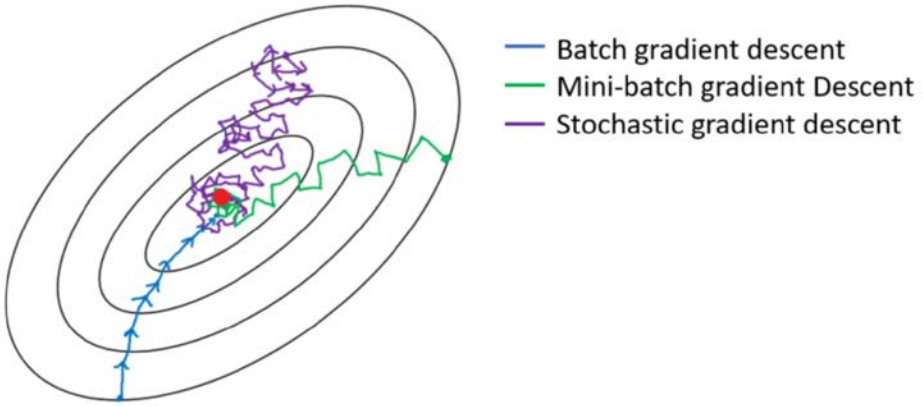


Figure 6: Gradient descent variants' trajectory towards minimum

https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

https://medium.com/@montjoile/an-introduction-to-gradient-descent-algorithm-34cf3cee752b

Epoch vs Batch Size vs Iterations

Epochs

One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Batch Size

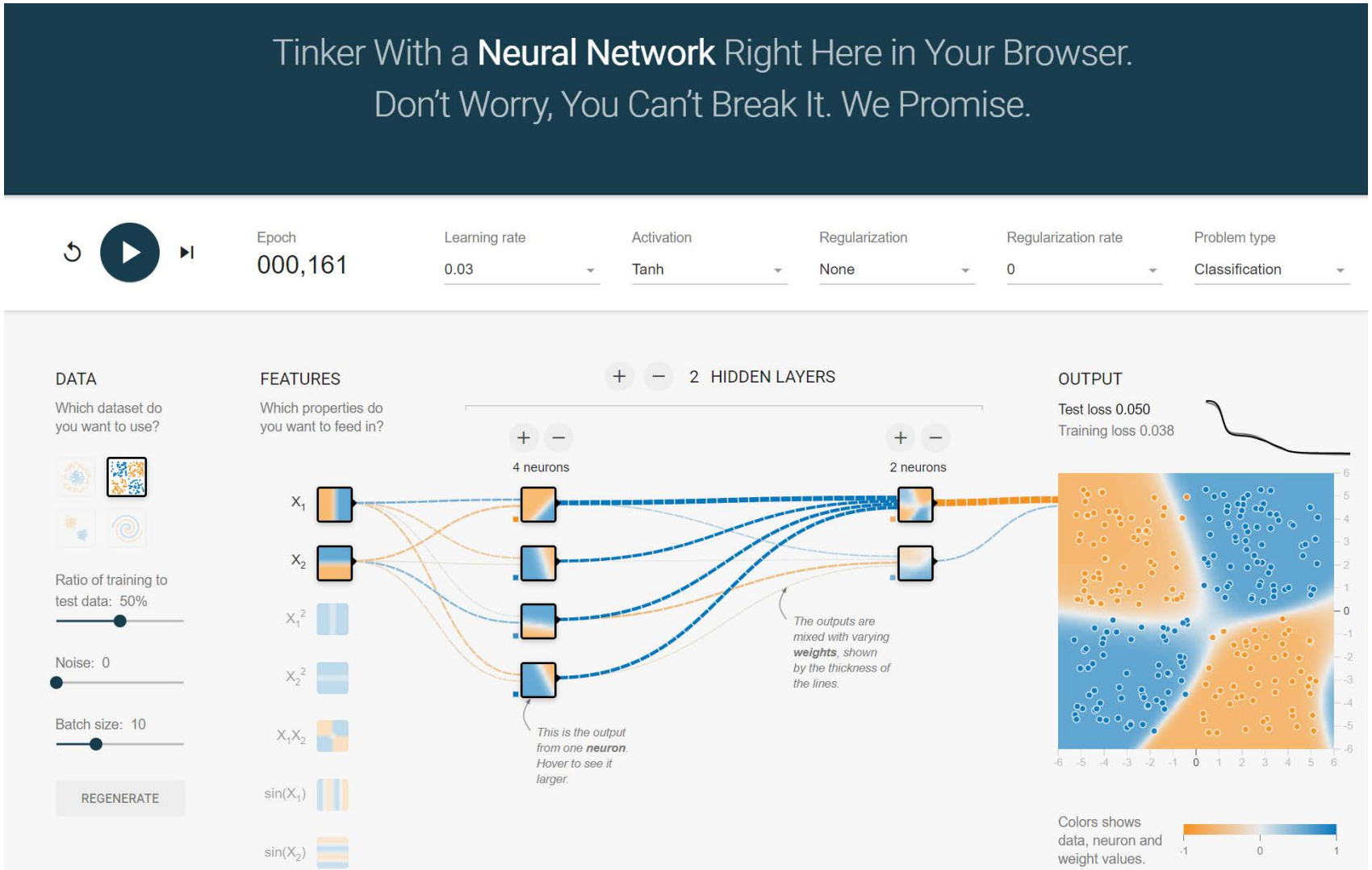
Total number of training examples present in a single batch.

Iterations

Iterations is the number of batches needed to complete one epoch.

https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

Example: Neural Network Playground



http://playground.tensorflow.org/

Tutorial: Binary Classification (Gaussian)

Tutorial: Binary Classification (Circle)