

Neural Networks and Backpropagation Algorithm

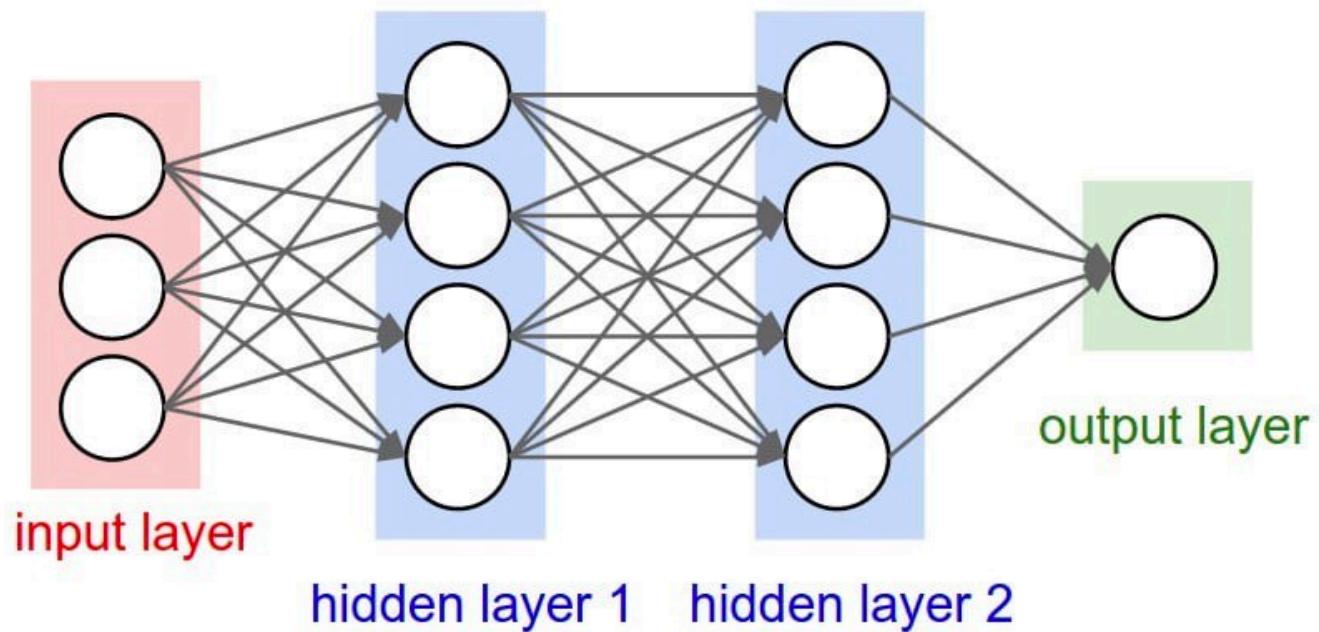
Lê Anh Cường

TDTU

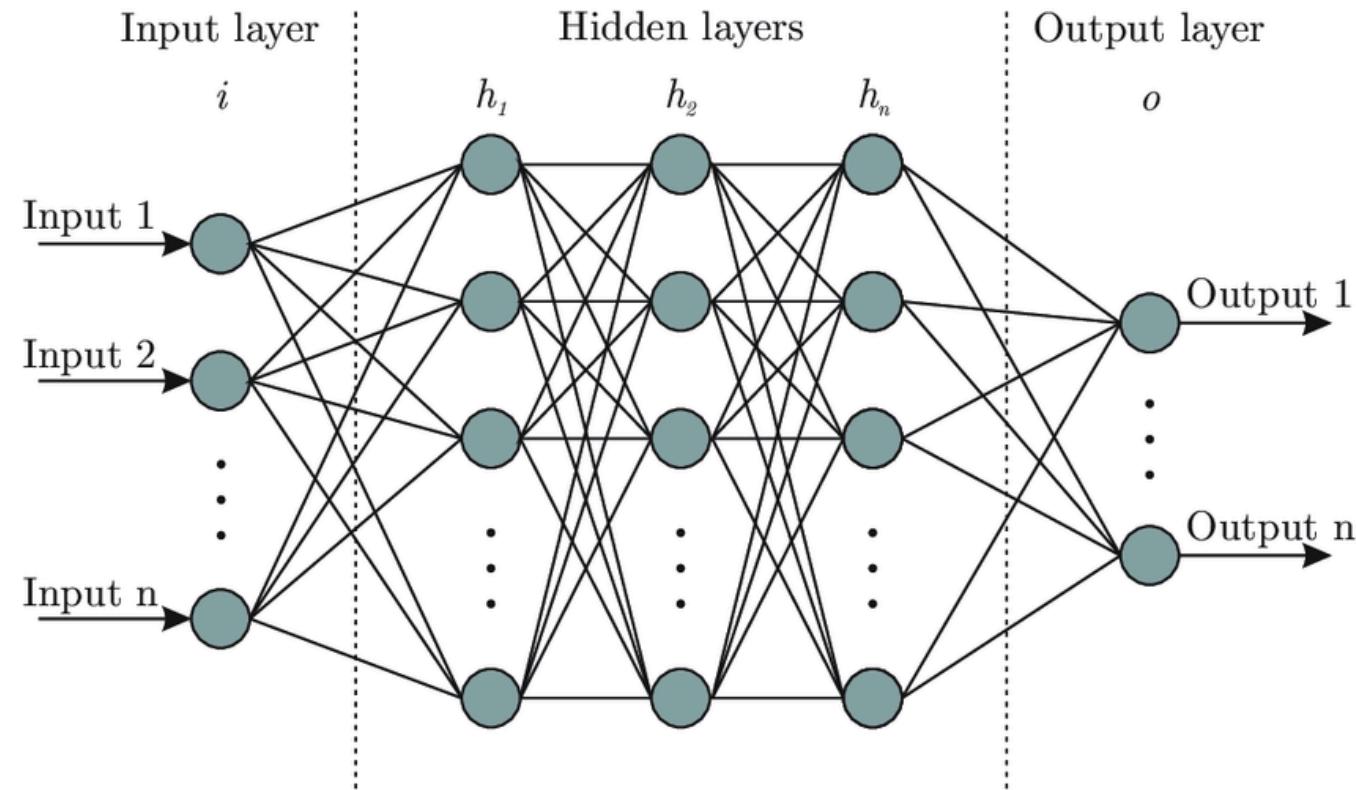
Outline

- General architecture of Neural Networks (NN)
- Forward computation for Feed-Forward Neural Networks (FFNN)
- Backpropagation for learning NN's parameters
- Implementation

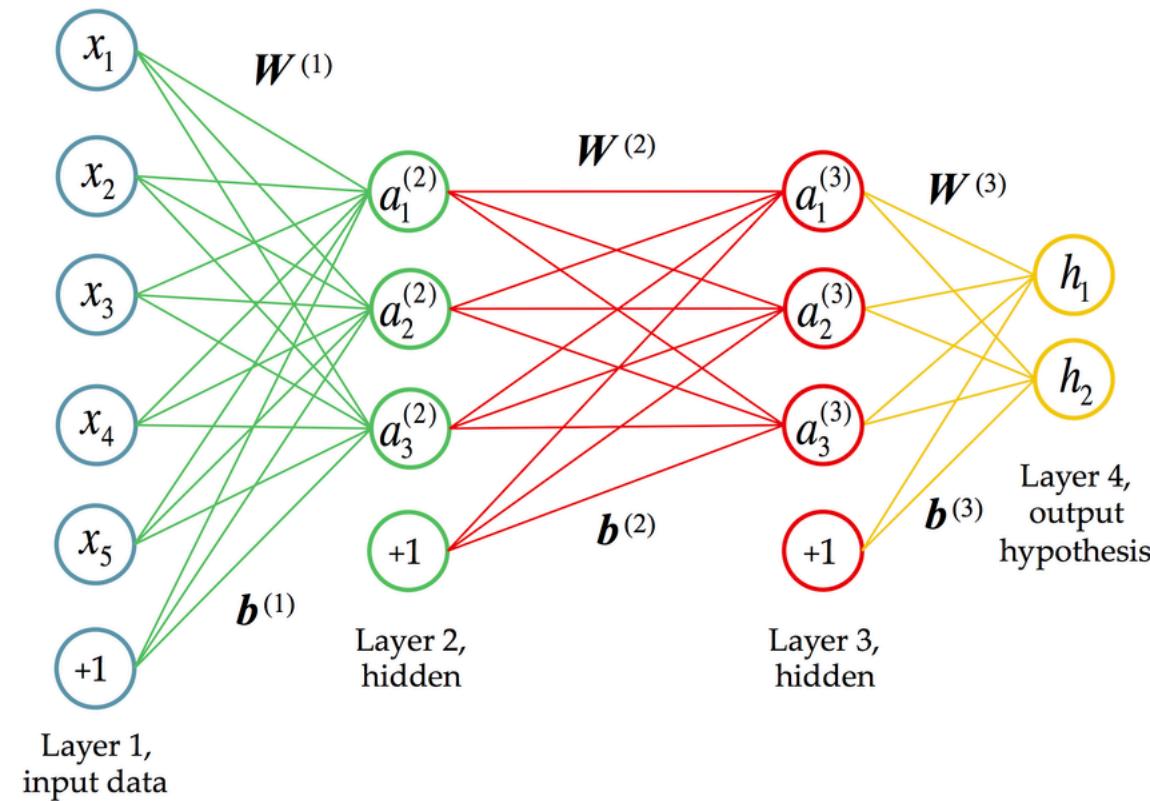
Neural Network



Neural Network

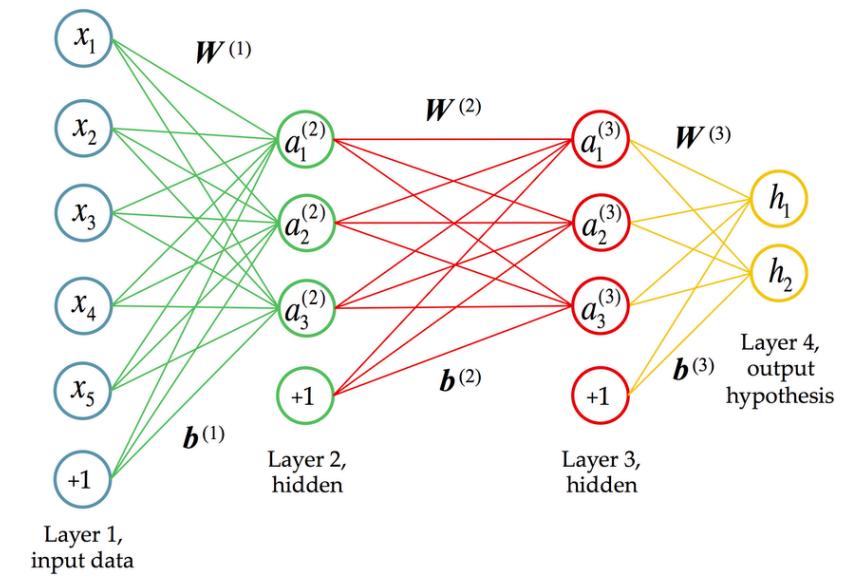


Neural Network

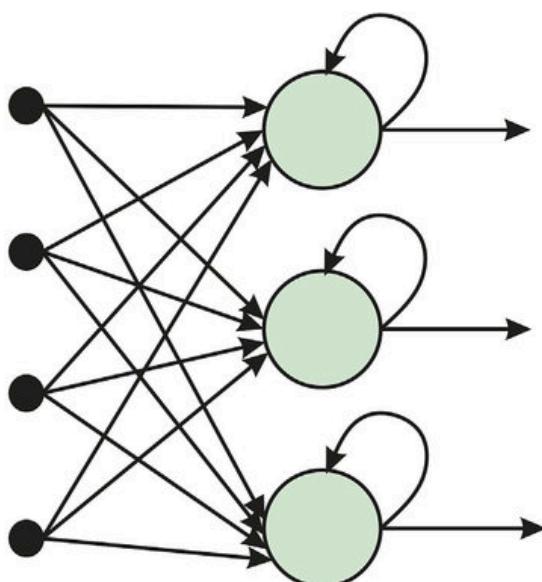


Neural Network: Definition

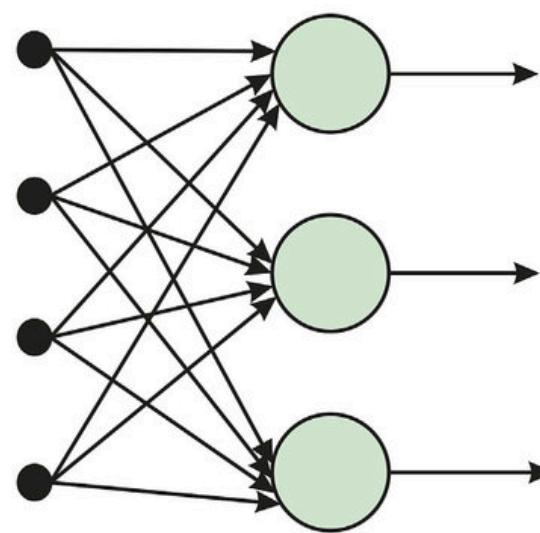
- **Artificial neural networks (ANNs)**, usually simply called **neural networks (NNs)**.
- An ANN is based on a collection of connected units or nodes called artificial neurons.
- Each connection can transmit a signal to other neurons.
- The connections are called edges which typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection.
- Different layers may perform different transformations on their inputs.



NN Types

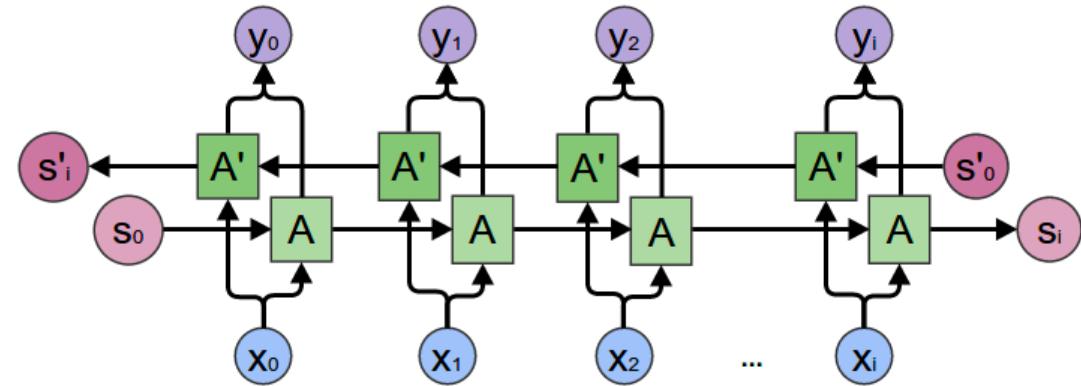
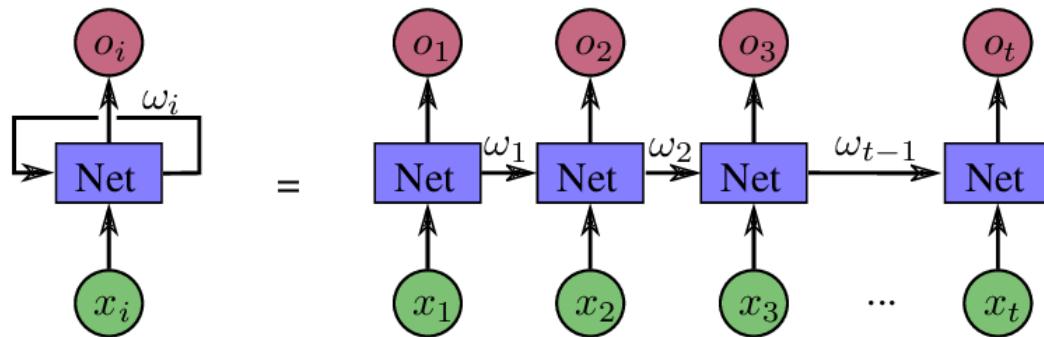


(a) Recurrent Neural Network



(b) Feed-Forward Neural Network

Recurrent Neural Network



Multiple Perceptron vs Neural Networks

A **multilayer perceptron** (MLP) is a class of [feedforward artificial neural network](#) (ANN). The term MLP is used ambiguously, sometimes loosely to *any* feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of [perceptrons](#) (with threshold activation); see § [Terminology](#). Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer.^[1]

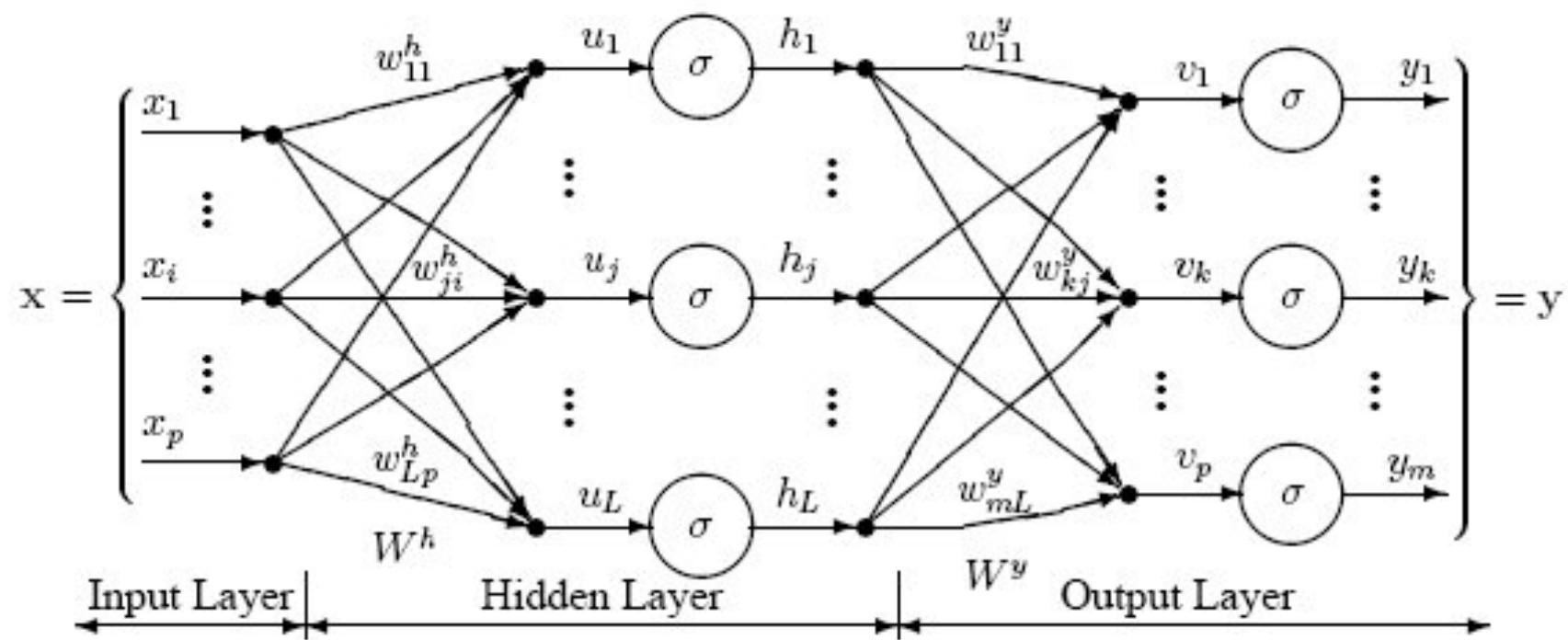
An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear [activation function](#). MLP utilizes a [supervised learning](#) technique called [backpropagation](#) for training.^{[2][3]} Its multiple layers and non-linear activation distinguish MLP from a linear [perceptron](#). It can distinguish data that is not linearly separable.^[4]

https://en.wikipedia.org/wiki/Multilayer_perceptron

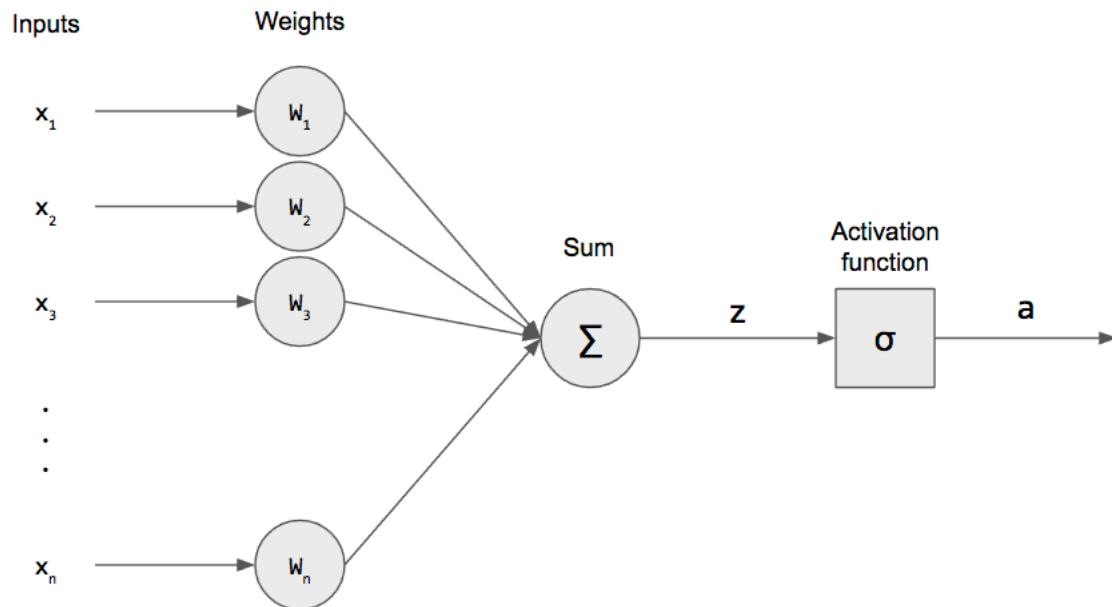
Outline

- General architecture of Neural Networks (NN)
- Forward computation for Feed-Forward Neural Networks (FFNN)
- Backpropagation for learning NN's parameters
- Implementation

FFNN and forward Computation

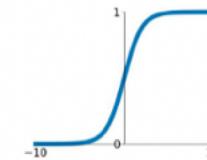


A Neural



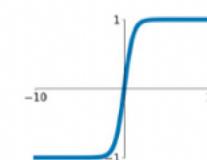
Sigmoid

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



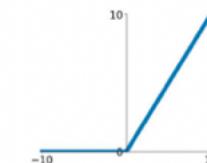
tanh

$$\tanh(x)$$

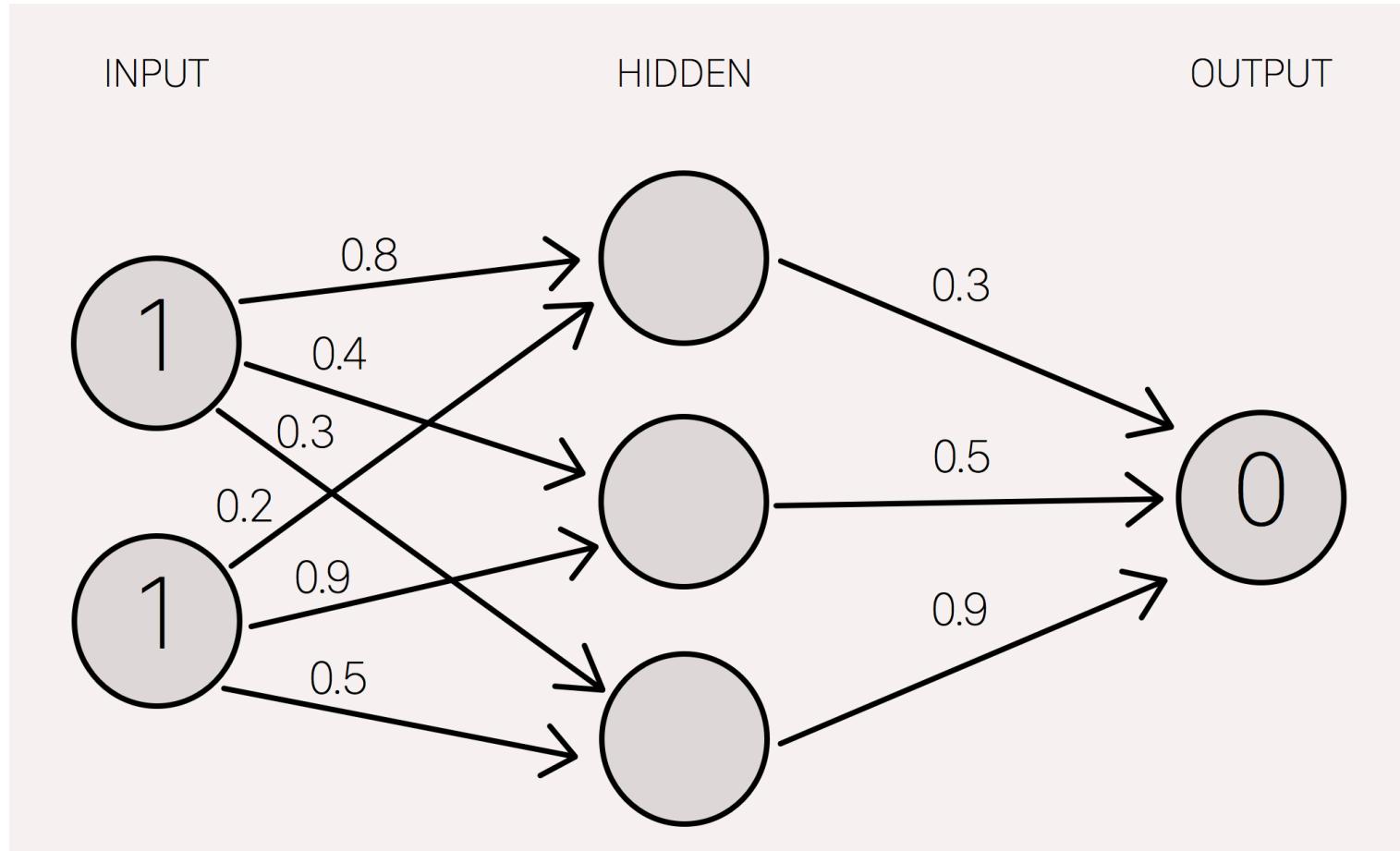


ReLU

$$\max(0, x)$$

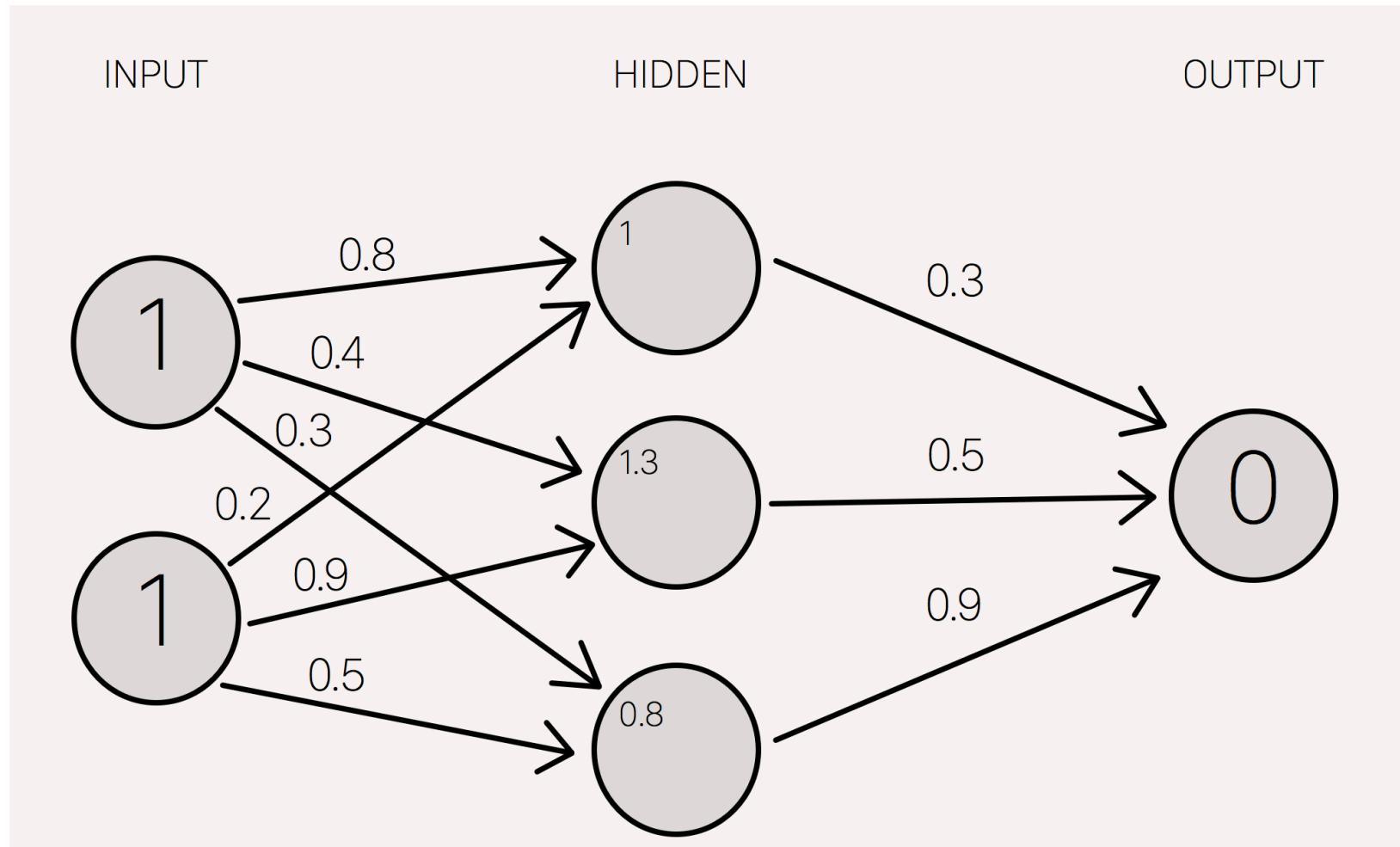


Example

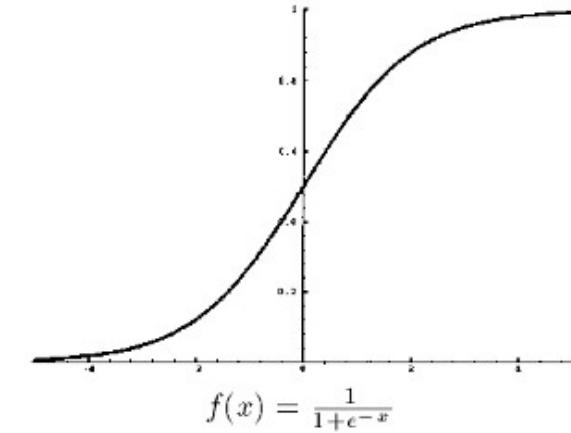
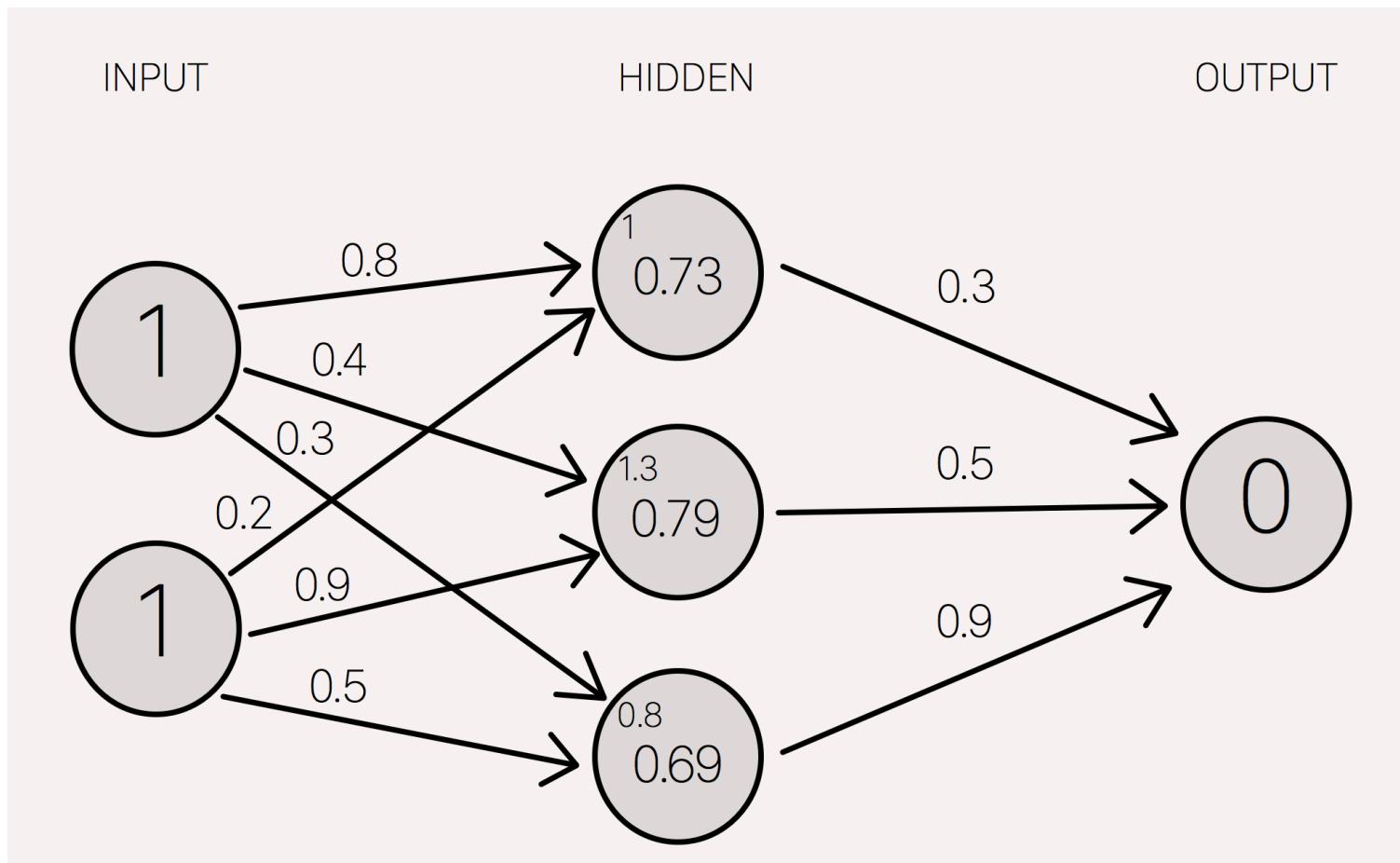


<https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>

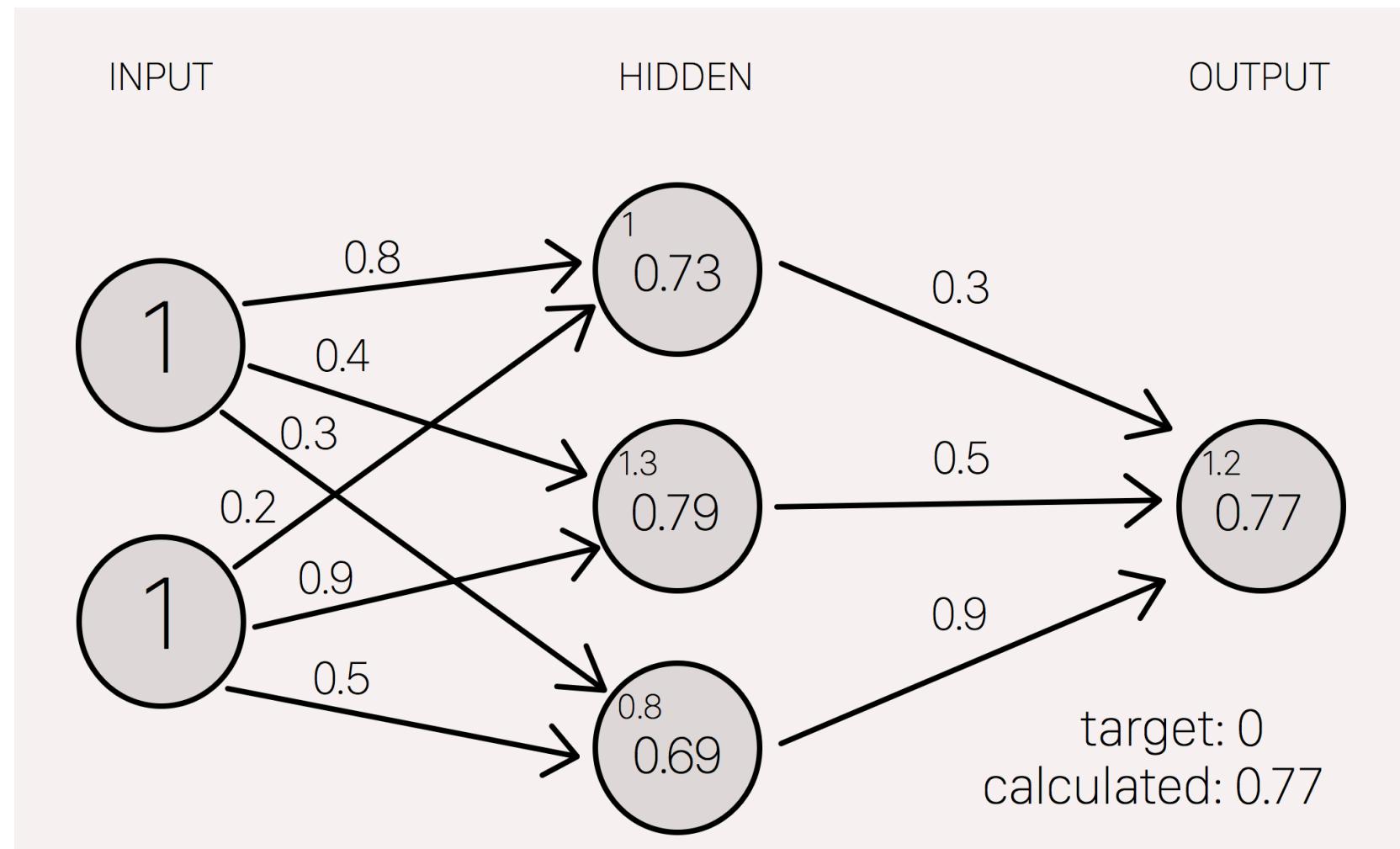
Example



Example



Example



Outline

- General architecture of Neural Networks (NN)
- Forward computation for Feed-Forward Neural Networks (FFNN)
- Backpropagation for learning FFNN's parameters
- Implementation

Backpropagation for NN

- The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams.
- The paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural nets to solve problems which had previously been insoluble.
- Today, the backpropagation algorithm is the workhorse of learning in neural networks.

Loss/Cost function

$$C = \frac{1}{n} \sum_x C_x$$

the cost for a single training example is $C_x = \frac{1}{2} \|y - a^L\|^2$.

$$y = y(x)$$

$$a^L = a^L(x)$$

Loss/Cost function

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2,$$

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2,$$

Learning Weights by Gradient Descent

1. Initialize random weights W

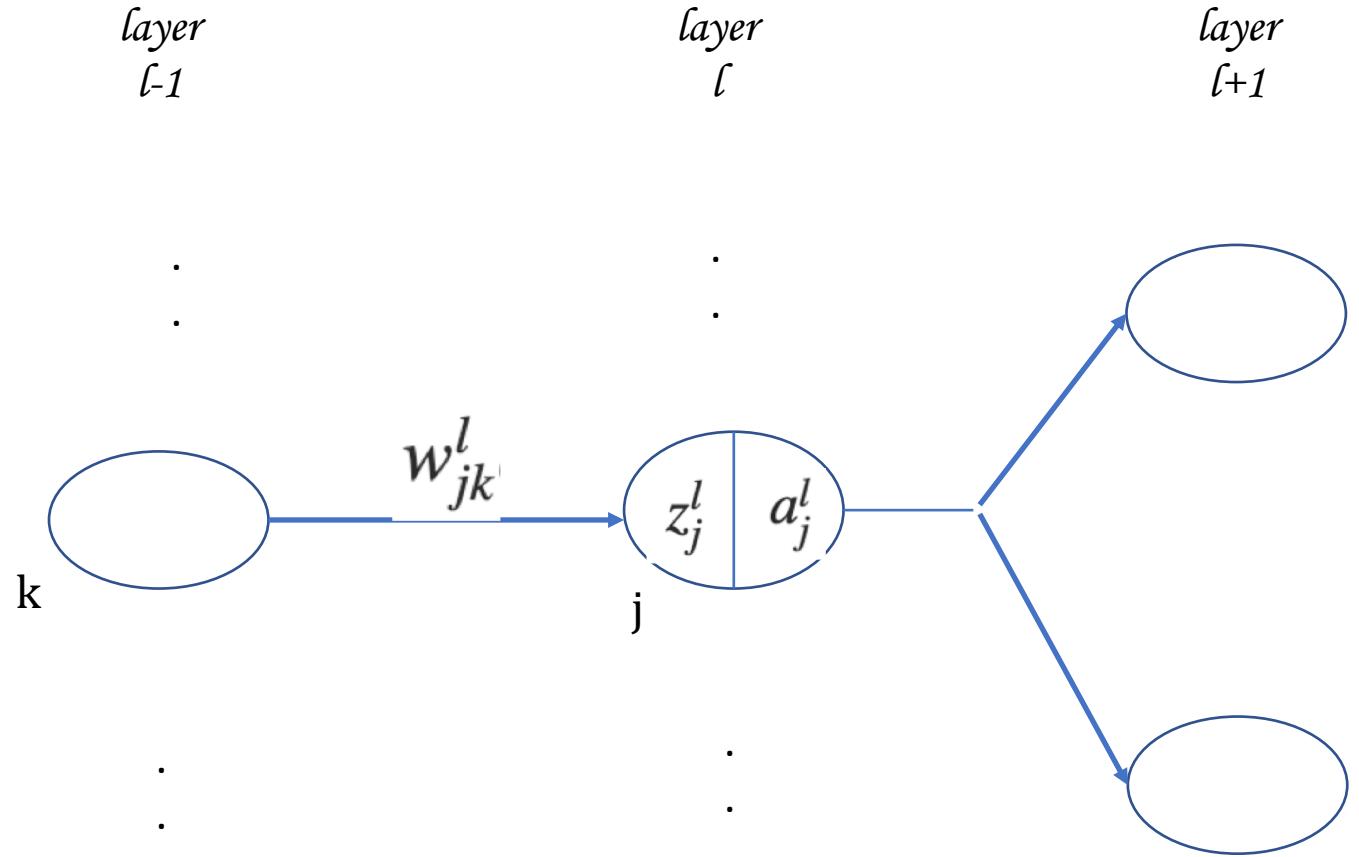
2. Repeat until convergence

3. Calculate gradient $\frac{\partial J(W)}{\partial W}$

4. Update weights $W \leftarrow W - \alpha \frac{\partial J(W)}{\partial W}$

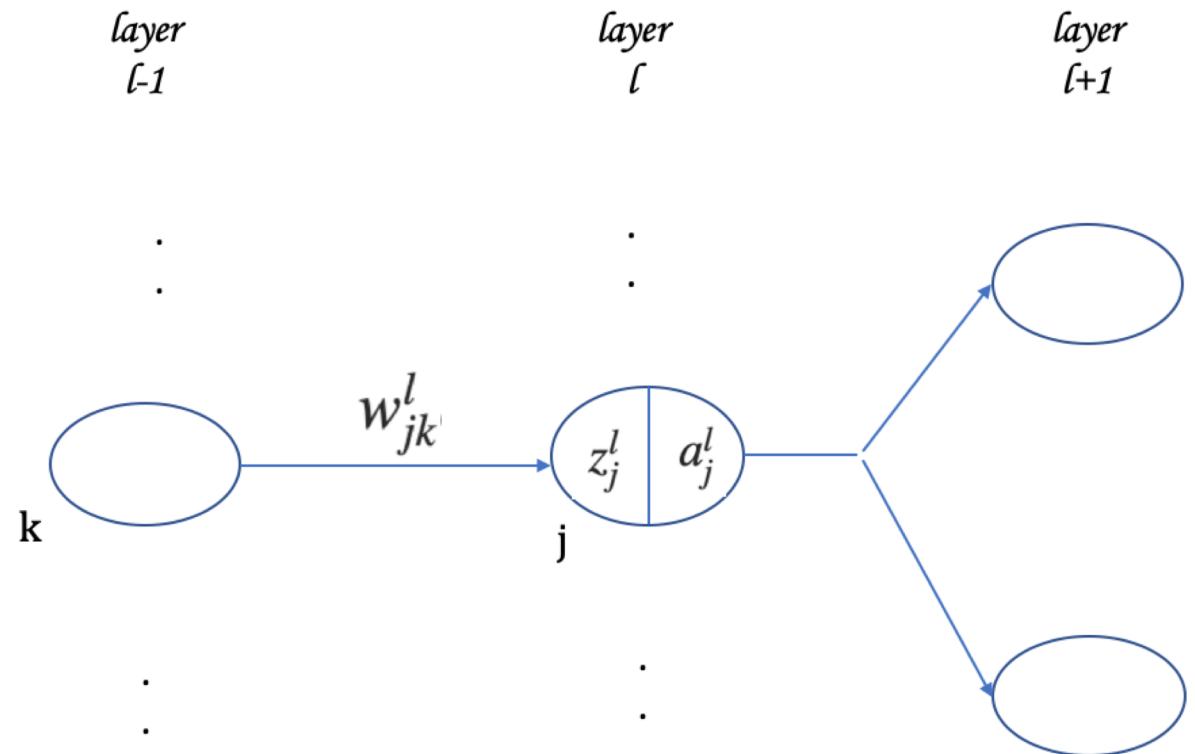
5. Return weights W

$$w_i = w_i - \mu \frac{\partial C}{\partial w_i}$$



$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2,$$

w_{jk}^l is the weight from the k^{th} neuron
in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron
in the l^{th} layer

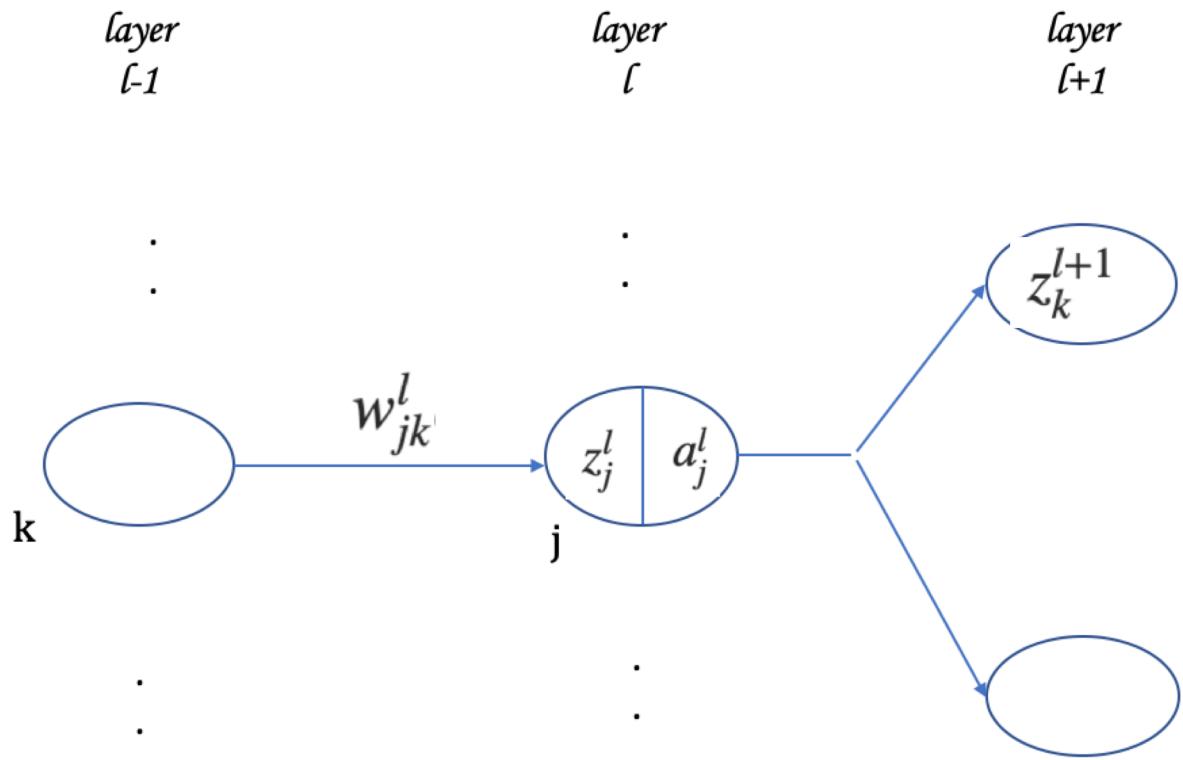


$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$



$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1},\end{aligned}$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\rightarrow \frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l).$$

$$\rightarrow \delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l).$$

Gradient Calculation

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2,$$

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right),$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad \delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

Calculate for Gradient at the last layer L

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right),$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2,$$

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$



$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}.$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

Gradient Calculation

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right),$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2,$$

$$\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$$



$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$

Gradient Calculation

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right),$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_i (y_j - a_j^L)^2,$$

$$\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L).$$



$$\begin{aligned} \delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}, \end{aligned}$$

Gradient Calculation

$$\delta^L = (a^L - y) \odot \sigma'(z^L). \quad (\text{BP1})$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \quad (\text{BP2})$$

By combining (BP2) with (BP1) we can compute the error δ^l for any layer in the network. We start by using (BP1) to compute δ^L , then apply Equation (BP2) to compute δ^{L-1} , then Equation (BP2) again to compute δ^{L-2} , and so on, all the way back through the network.

Gradient Calculation

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L). \quad (30)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \quad (\text{BP2})$$

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad (\text{BP1})$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad (\text{BP2})$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (\text{BP3})$$

$$\frac{\partial C}{\partial w_{j,k}^l} = a_k^{l-1} \delta_j^l \quad (\text{BP4})$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L).$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

The backpropagation algorithm

1. **Input x :** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \dots, L$ compute
$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l).$$
3. **Output error δ^L :** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L).$
4. **Backpropagate the error:** For each $l = L - 1, L - 2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l).$
5. **Output:** The gradient of the cost function is given by
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \text{ and } \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

Outline

- General architecture of Neural Networks (NN)
- Forward computation for Feed-Forward Neural Networks (FFNN)
- Backpropagation for learning FFNN's parameters
- Implementation

Reference

- <http://neuralnetworksanddeeplearning.com/chap2.html>