Week 2

Introduction to neural networks and deep learning

greatlearning
Learning for Life

Topics you have covered in week 2 videos

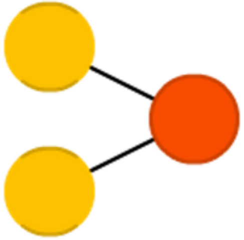
- Feed forward
- Back propagation
- Fully connected layer - forward pass
- Fully connected layer - backward pass
- Activation functions
- Activation functions in practice
- Softmax
- Cross entropy loss

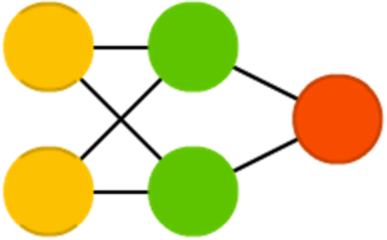
Session agenda

- Building blocks of neural network
- Case study
- Questions

Feed forward

Feed forward neural network





Feed forward neural networks (FF or FFNN) and perceptrons (P) are very straight forward, they feed information from the front to the back (input and output, respectively). Neural networks are often described as having layers, where each layer consists of either input, hidden or output cells in parallel. A layer alone never has connections and in general two adjacent layers are fully connected (every neuron form one layer to every neuron to another layer). The simplest somewhat practical network has two input cells and one output cell, which can be used to model logic gates.

Image and text source: asimovinstitute

Feed forward network characteristics

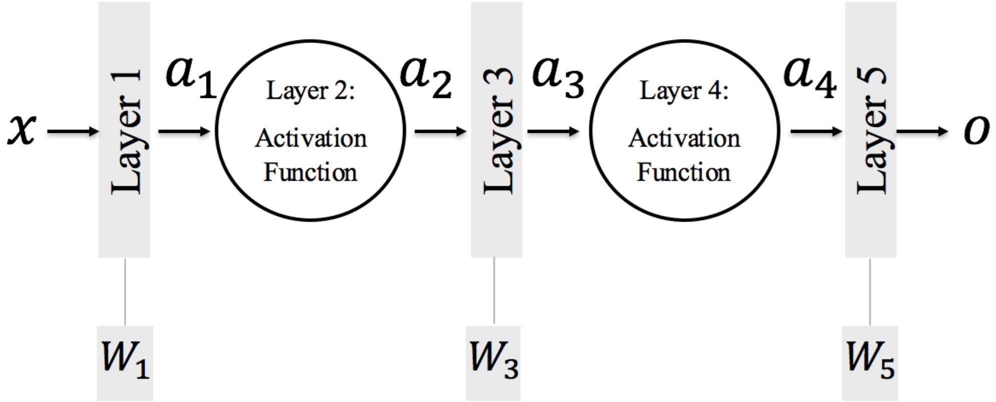
1. Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.

1. Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next., and this explains why these networks are called feed-forward networks.

1. There is no connection among perceptrons in the same layer.

Text source: [cs.stanford.edu]()

Feed forward - composition of functions



$a_1 = F(x, W_1),\ x \in \mathbb{R}^n$

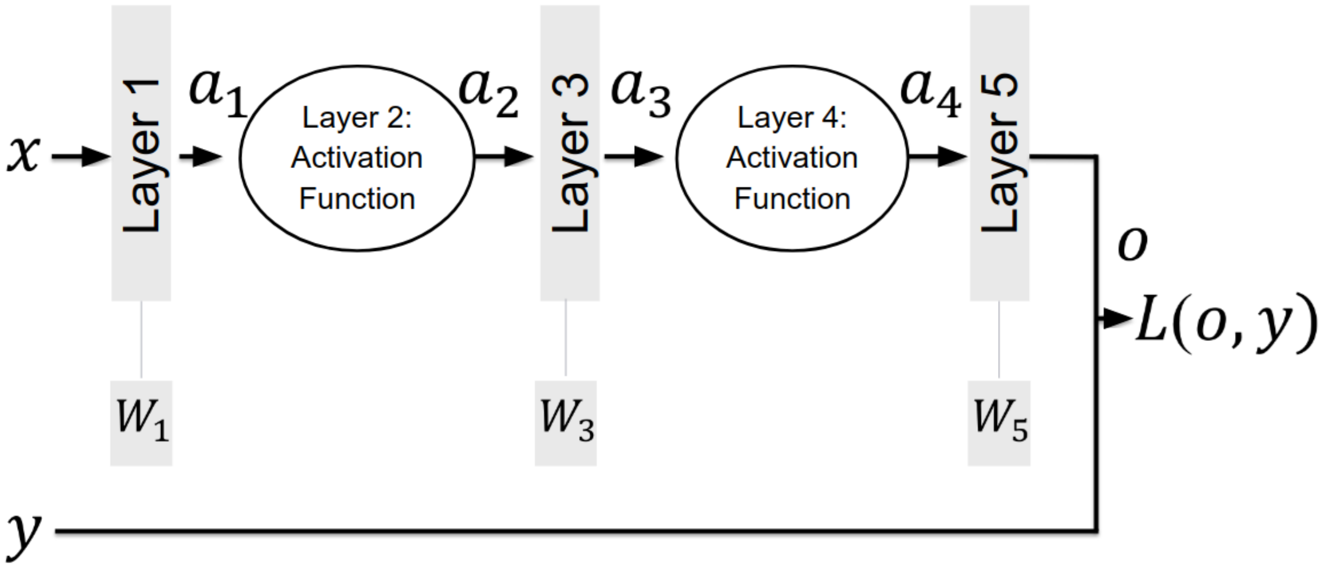
$a_2 = G(a_1)$

$a_3 = H(a_2, W_3),$

$a_4 = J(a_3)$

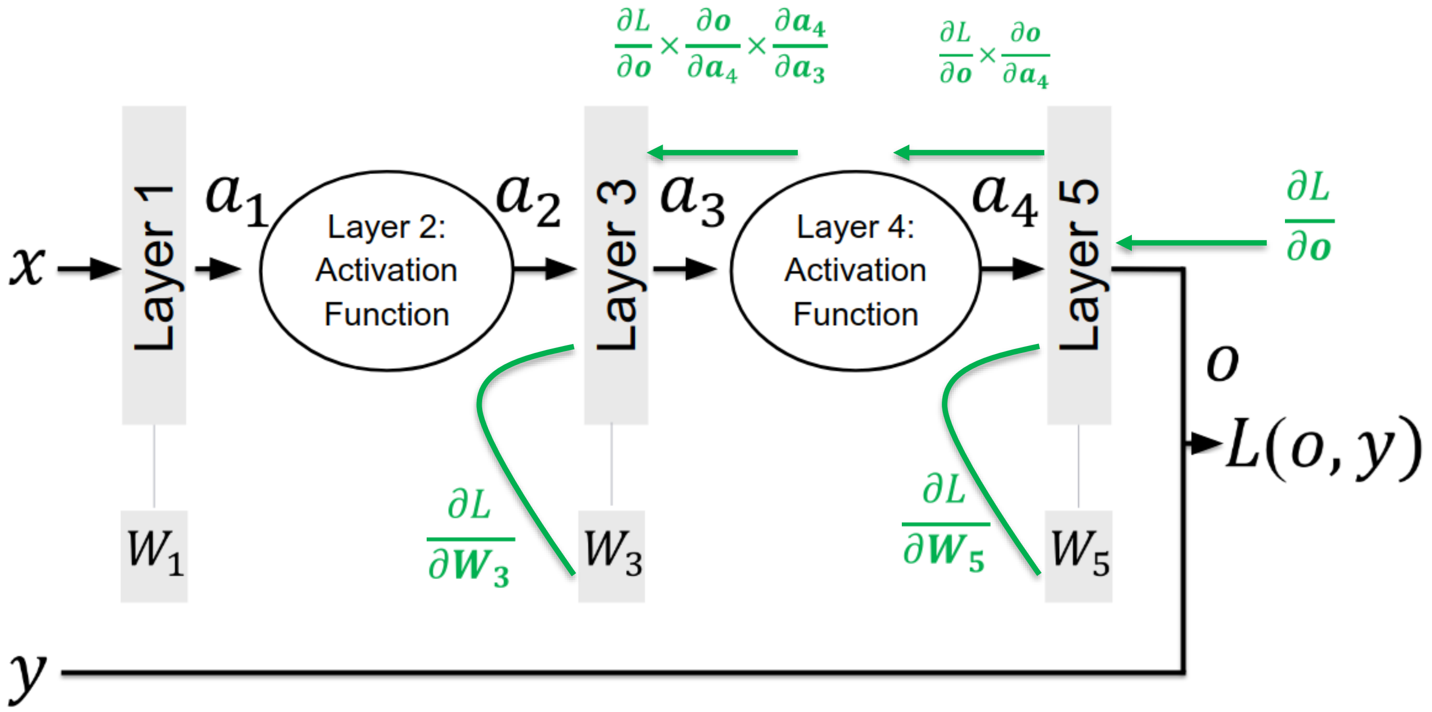
$o = K(a_4, W_5) = K(J(H(G(F(x, W_1)), W_3)), W_5) \in \mathbb{R}^m$

Feed forward - loss



Backward propagation

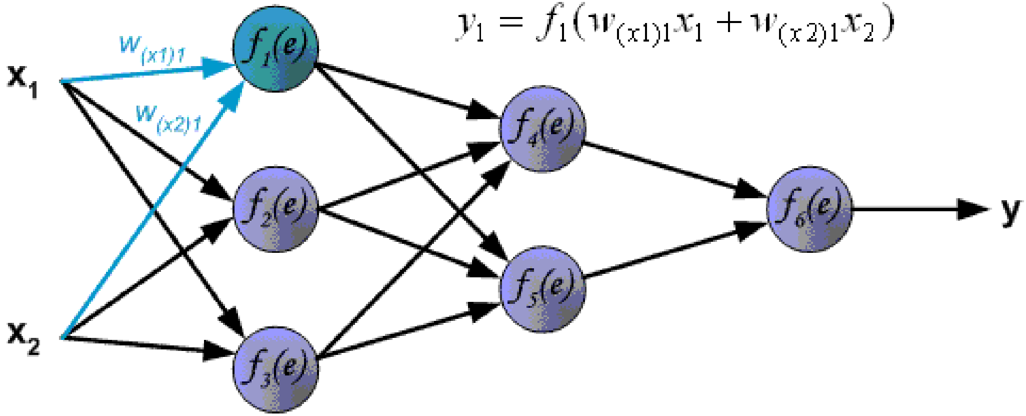
Backprop



$\frac{\partial L}{\partial \boldsymbol{o}} \times \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{a_4}} \times \frac{\partial \boldsymbol{a_4}}{\partial \boldsymbol{a_3}}$
$\frac{\partial L}{\partial \boldsymbol{o}} \times \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{a_4}}$
x
Layer 1
a_1
Layer 2: Activation Function
a_2
Layer 3
a_3
Layer 4: Activation Function
a_4
Layer 5
$\frac{\partial L}{\partial \boldsymbol{o}}$
o
$L(o, y)$
W_1
$\frac{\partial L}{\partial \boldsymbol{W_3}}$
W_3
$\frac{\partial L}{\partial \boldsymbol{W_5}}$
W_5
y

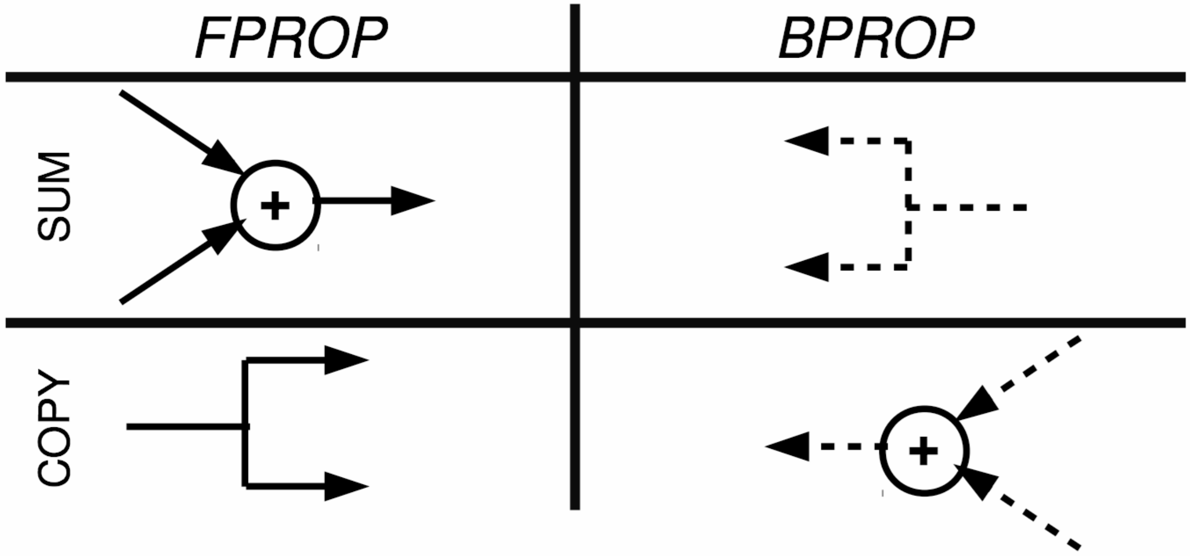
Backprop

FP



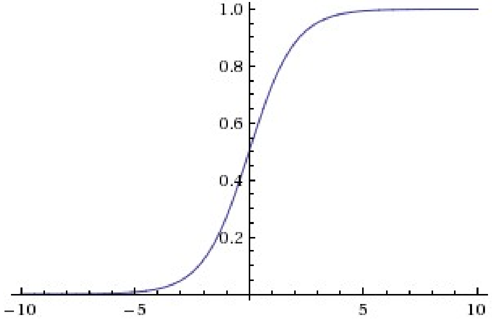
$$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$

Forward prop and backward prop are duals

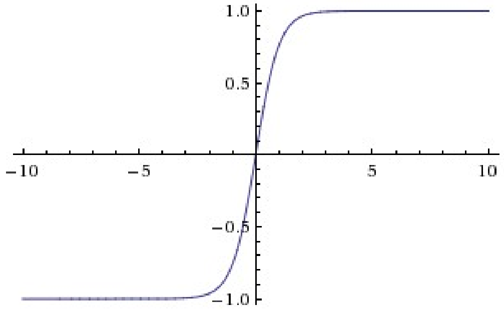


Activation functions

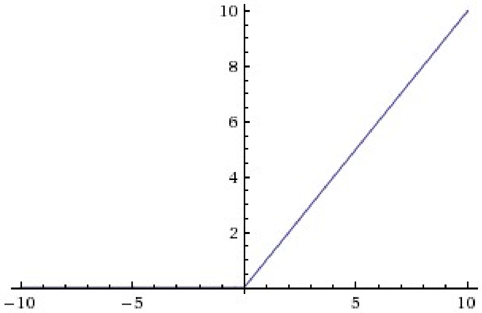
Types of activation function



Sigmoid



Tanh



ReLU

Every activation function (or non-linearity) takes a single number and performs certain fixed mathematical operation on it.

Images and text source: [cs231n stanford]

Sigmoid function

- Activation function of form **f(x) = 1 / 1 + exp(-x)**
- Ranges from 0-1
- S-shaped curve
- Historically popular
    - Interpretation as a saturating “firing rate” of a neuron

Drawbacks

- Its output is not zero centered. Hence, make the gradient go too far in different directions
- Vanishing Gradient Problem
- Slow convergence

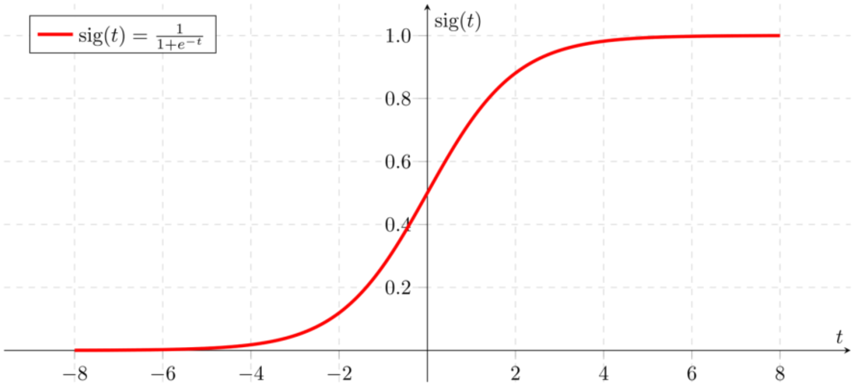


Image source: [wikimedia]

Tanh function

- Ranges between -1 to +1
- Output is zero centered
- Generally preferred over Sigmoid function

Drawbacks

- Though optimisation is easier, it still suffers from the Vanishing Gradient Problem

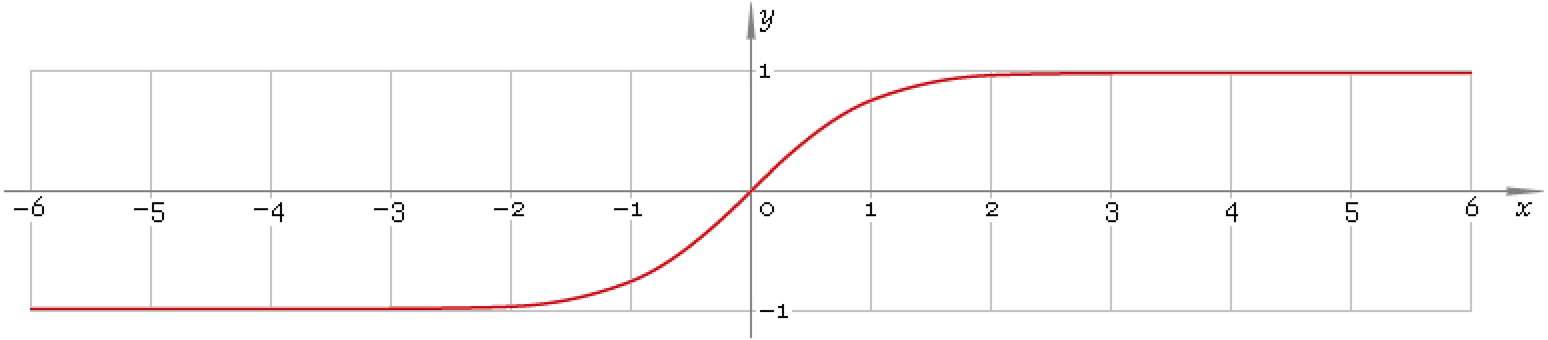


Image source: librow

ReLU function

- Very simple and efficient
- Have 6x times better convergence than tanh and sigmoid function.
- Very efficient in computation

Drawbacks

- Output is not zero centered.
- Should only be used within hidden layers of a NN model

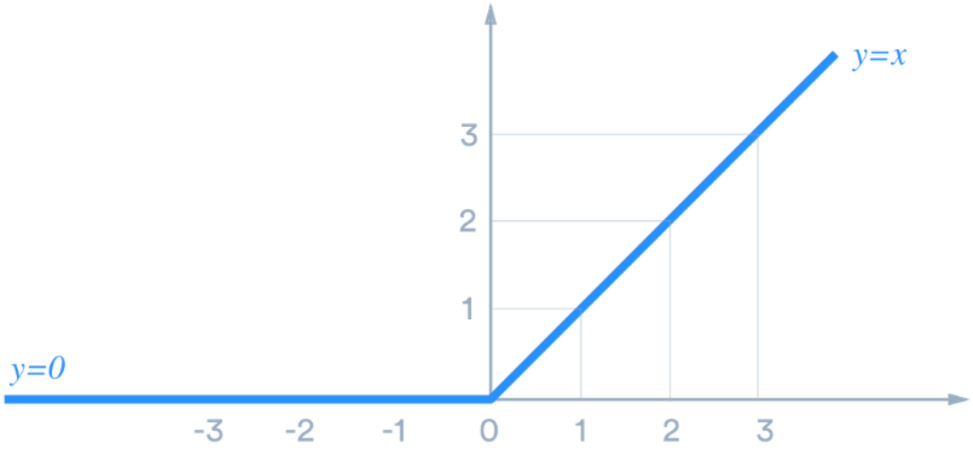


Image source: medium

Leaky ReLU function

- Leaky ReLU was introduced to overcome the problem of dying neurons.
- Leaky ReLU introduces a small slope to keep the neurons alive
- Does not saturate (in +region)

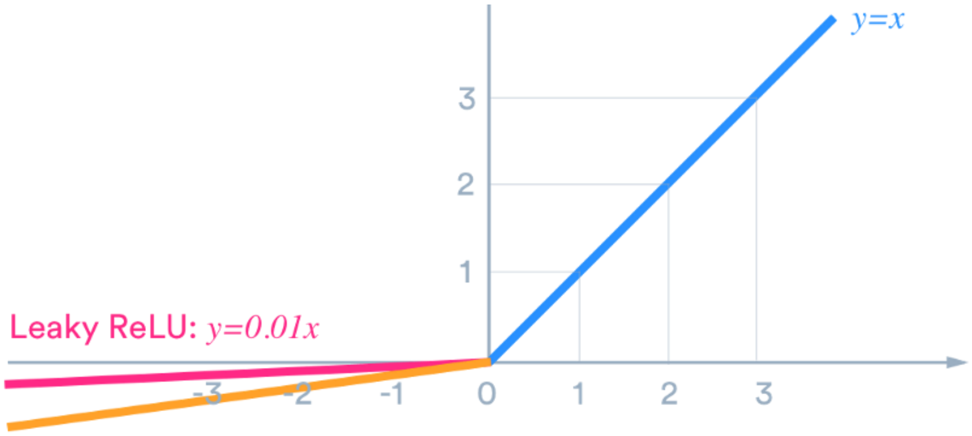


Image source: [medium]()

ELU function

- ELU function tend to converge cost to zero faster and produce more accurate results
- Closer to zero mean outputs
- Has a extra alpha constant which should be positive number
- ELU is very similar to RELU except negative inputs
- Have all advantages of ReLU

Drawbacks

- Computation requires exp()

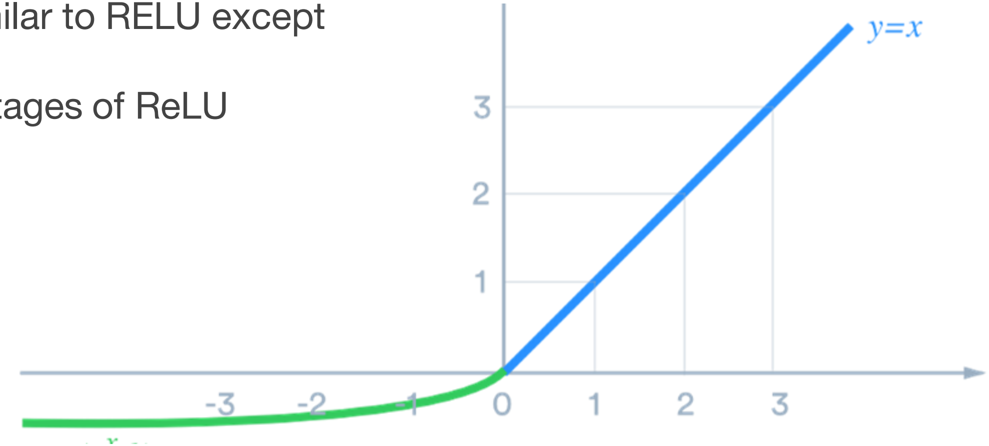


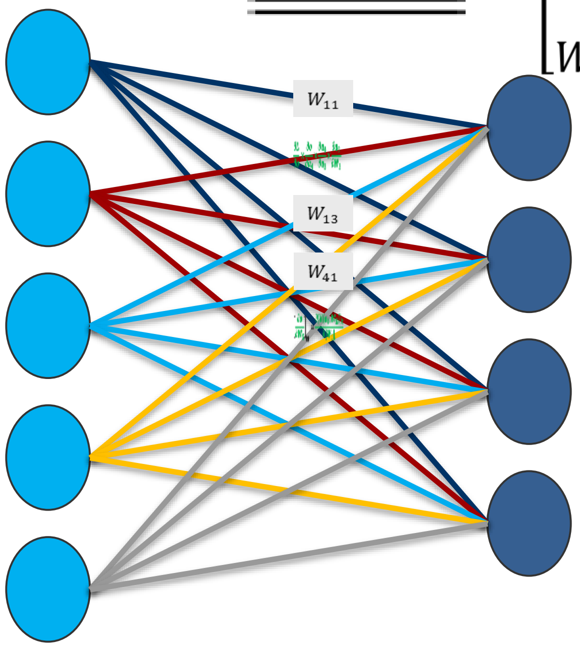
Image source: [medium]

Fully-connected layer

Fully connected layer

$$\begin{bmatrix} W_{11} & \cdots & W_{15} \\ \vdots & \ddots & \vdots \\ W_{41} & \cdots & W_{45} \end{bmatrix}$$

$\boldsymbol{n} \in \mathbb{R}^5$



$\boldsymbol{m} \in \mathbb{R}^4$

$m_1 = n_1 \times W_{11} + n_2 \times W_{12} + .. + n_5 \times W_{15}$

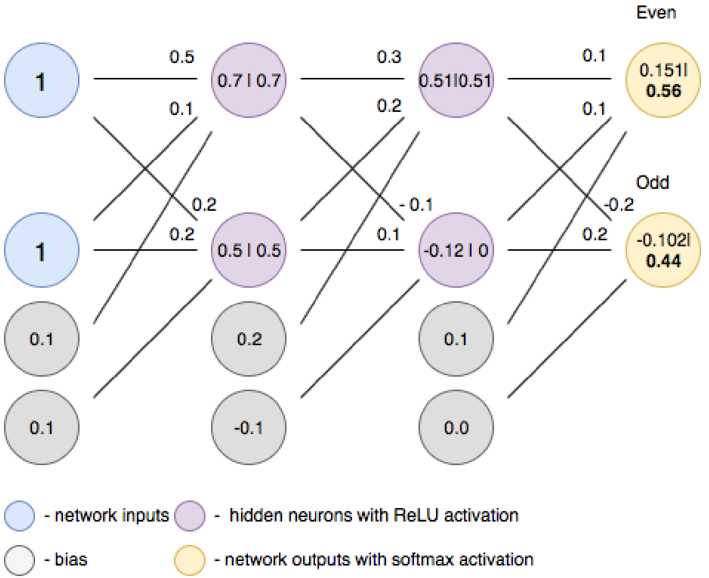
$m_2 = n_1 \times W_{21} + n_2 \times W_{22} + .. + n_5 \times W_{25}$

$m_3 = n_1 \times W_{31} + n_2 \times W_{32} + .. + n_5 \times W_{35}$

$m_4 = n_1 \times W_{41} + n_2 \times W_{24} + .. + n_5 \times W_{45}$

FC layer - forward pass

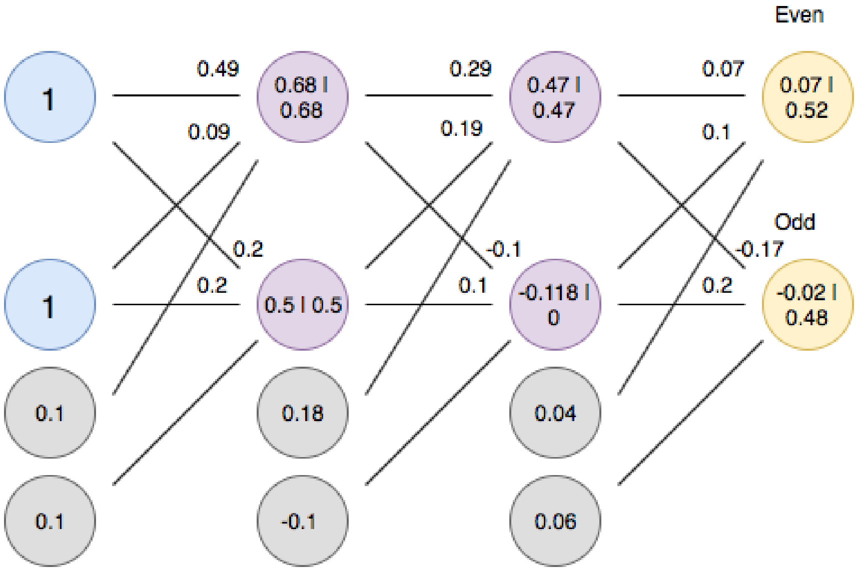
Forward pass is basically a set of operations which transform network input into the output space. During the inference stage neural network relies solely on the forward pass.



Images and text source: [towardsdatascience]()

FC layer - backward pass

Backpropagation is an algorithm which calculates error gradients with respect to each network variable (neuron weights and biases). Those gradients are later used in optimization algorithms, such as Gradient Descent, which updates them correspondingly. The process of weights and biases update is called Backward Pass.



Images and text source: [towardsdatascience]()

Softmax function

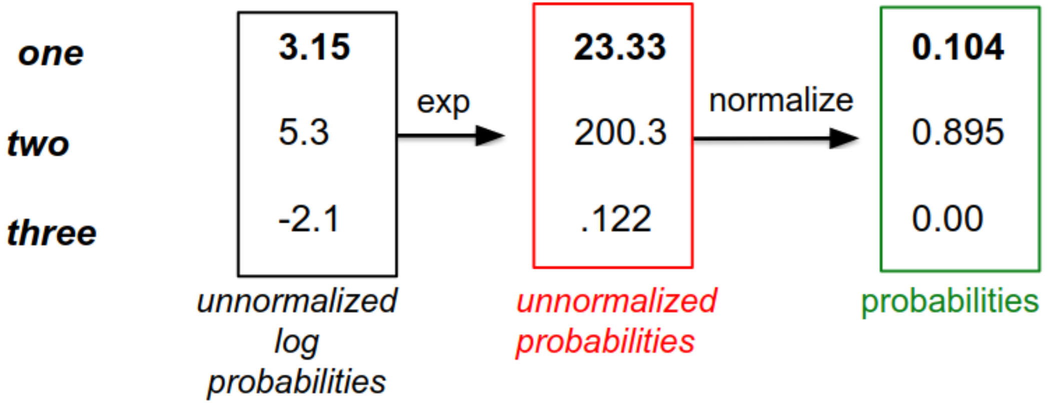
Softmax

- Softmax function is a multinomial logistic classifier, i.e. it can handle multiple classes
- Softmax typically the last layer of a neural network based classifier
- Softmax function is itself an activation function, so doesn't need to be combined with an activation function

greatlearning
Learning for Life

Softmax

$$S \in \mathbb{R}^d \longrightarrow \boxed{\text{SoftMax}} \longrightarrow p \in \mathbb{R}^d \quad p_i = \frac{e^{s_i}}{\sum_{j=1}^{d} e^{s_j}}$$



Cross-entropy loss

Cross-entropy loss

- Cross-entropy loss (often called Log loss) quantifies our unhappiness for the predicted output based on its deviation from the desired output
- Perfect prediction would have a loss of 0
- With gradient descent, we try to reduce this (cross-entropy) loss for a classification problem

Cross-entropy loss

$S \in \mathbb{R}^d$ → Cross Entropy → $\boldsymbol{cost} \in \mathbb{R}^1$

y_i is 1 (and 0 otherwise) if and only if sample belongs to class i

$$s = f(x_i; W)$$

$$L_i = -y_i \cdot log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$$

$$L = \sum_i L_i$$

Summary

In summary - building blocks of deep NNs

- Forward Propagation
- Backward Propagation
- Activation Layers (ReLU, Sigmoid, tanh…)
- Fully Connected Layer
- Convolution Layer
- Max Pooling Layer
- … and so on

Thank you! :)

Questions are always welcome