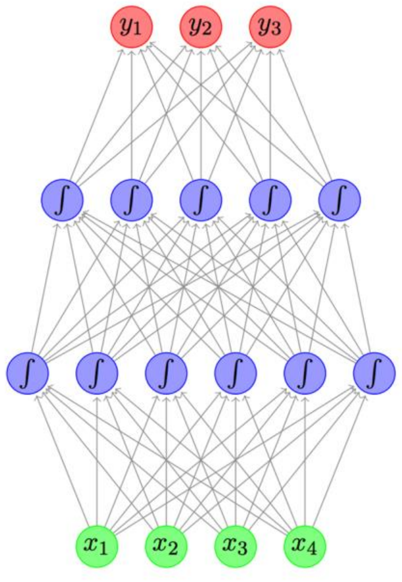
Building blocks of Artificial Neural Networks

Topics covered this week

- Introduction to artificial neural networks
- Feed forward neural networks
    - Basic structure of neural network
    - Function of a neural network
    - Training a neural network
    - Loss function
- Activation functions
    - Types of activation functions
- Learning rate
- Backpropagation and Optimization techniques: gradient descent
- Case Study

Structure of an ANN

Layer details



Output layer

- Represents the output of the neural network
- Most commonly it doesn't have any activation function

Hidden layer(s)

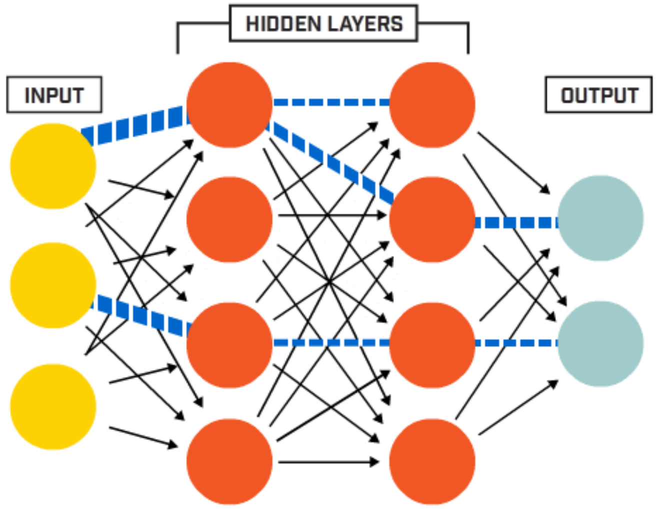
- Represents the intermediary nodes that divide the input space into regions with (soft) boundaries
- Given enough hidden nodes, we can model an arbitrary input-output relation
- It takes in a set of weighted input and produces output through an activation function

Input layer

- Represents dimensions of the input vector (one node for each dimension)

Image source: [A Primer on Neural Network Models for Natural Language Processing]

Train a neural network

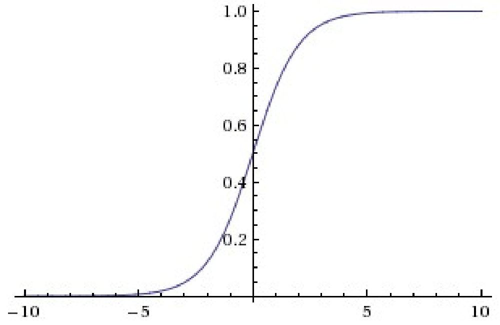


- Choose hyper parameters
- Choose network design
- Form a neural network
- Compute an estimate value for all samples
- Compute loss
- Reduce loss
- Repeat last three steps

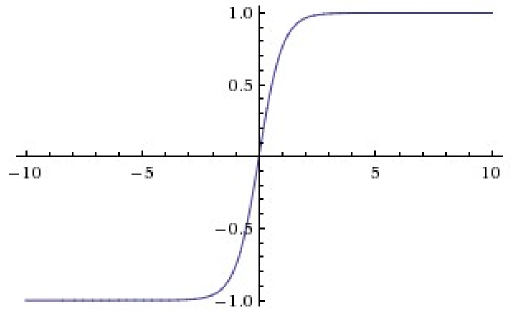
Image source: ptgrey.com

Activation functions

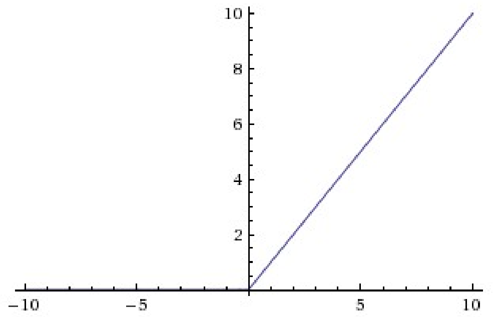
Types of activation function



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

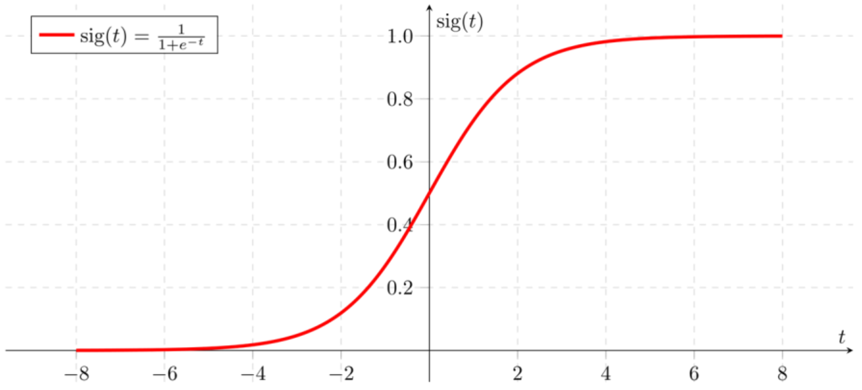


Image source: wikimedia

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

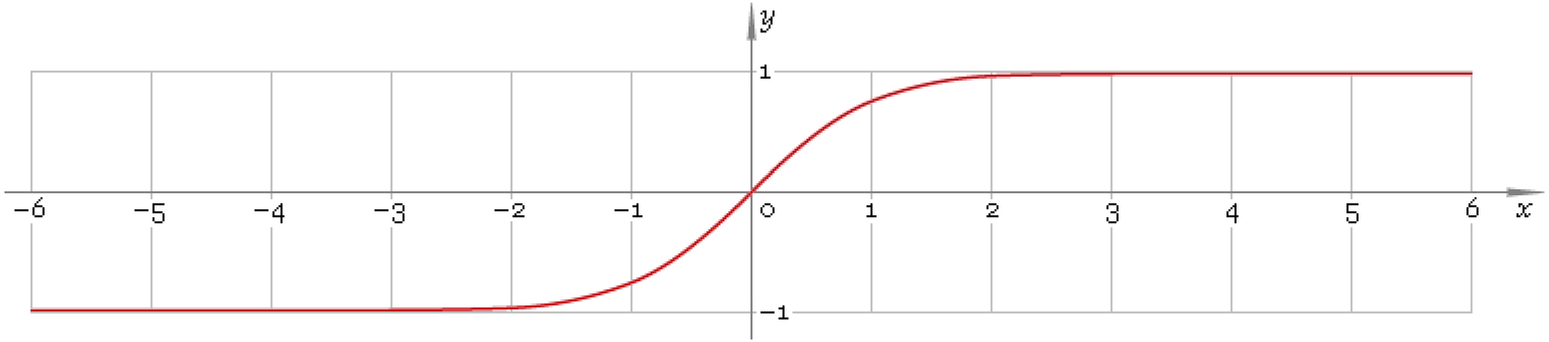


Image source: librow

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

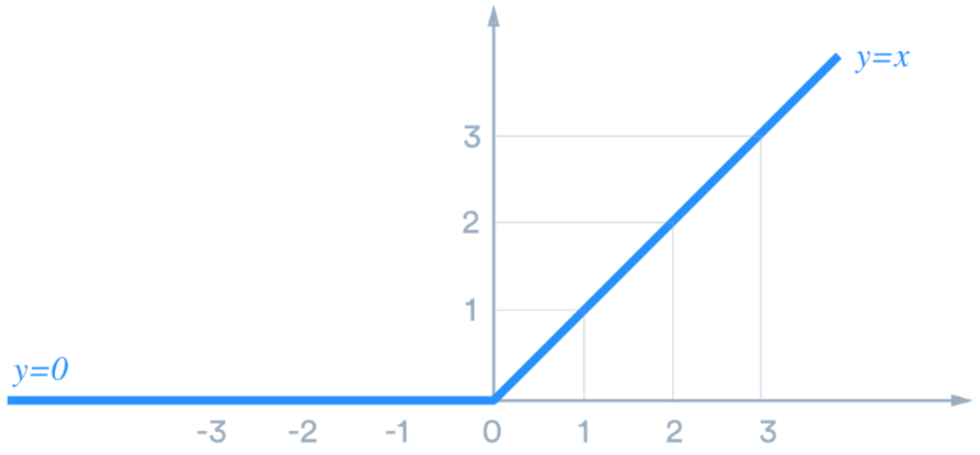


Image source: medium

greatlearning
Learning for Life

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)

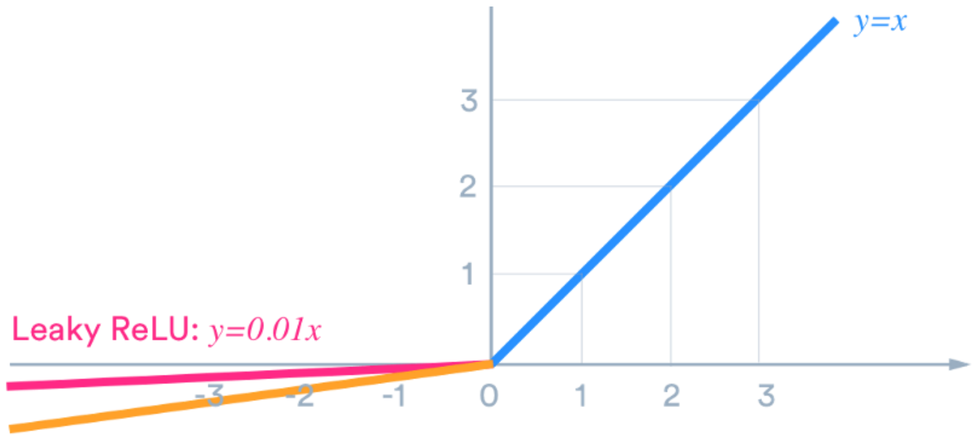


Image source: medium

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()

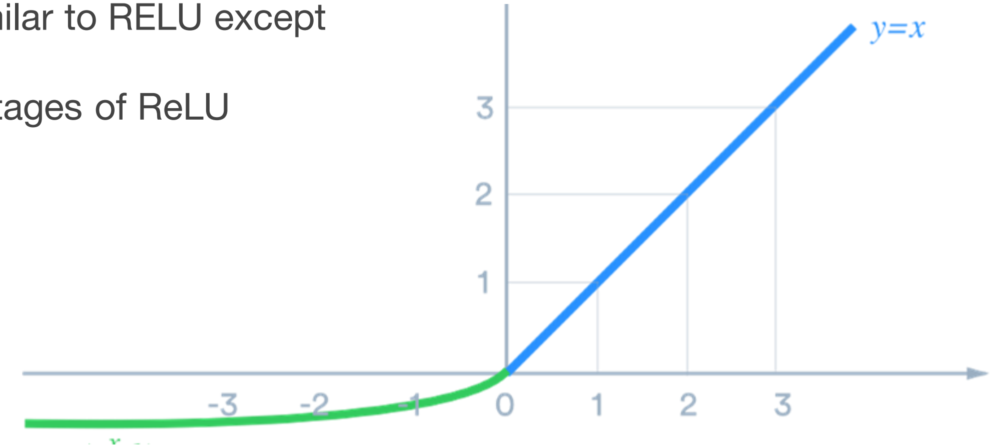


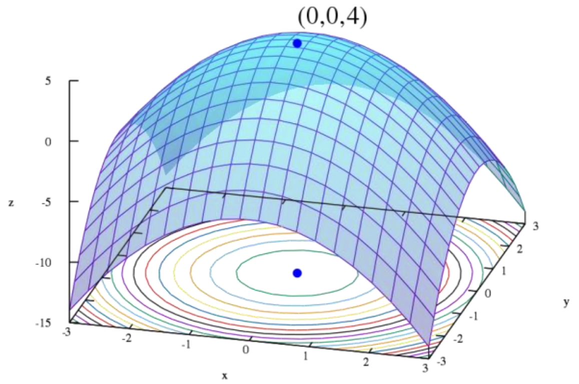
Image source: medium

Error and Loss function

- In most learning networks, error is calculated as the difference between the actual output and the predicted output.
- The function that is used to compute this error is known as Loss Function.
- Different loss functions will give different errors for the same prediction, and thus have a considerable effect on the performance of the model.
- One of the most widely used loss function is mean square error, which calculates the square of difference between actual value and predicted value.
- Different loss functions are used to deal with different type of tasks, i.e. regression and classification.

Text source: xpertup

Loss functions and Optimization

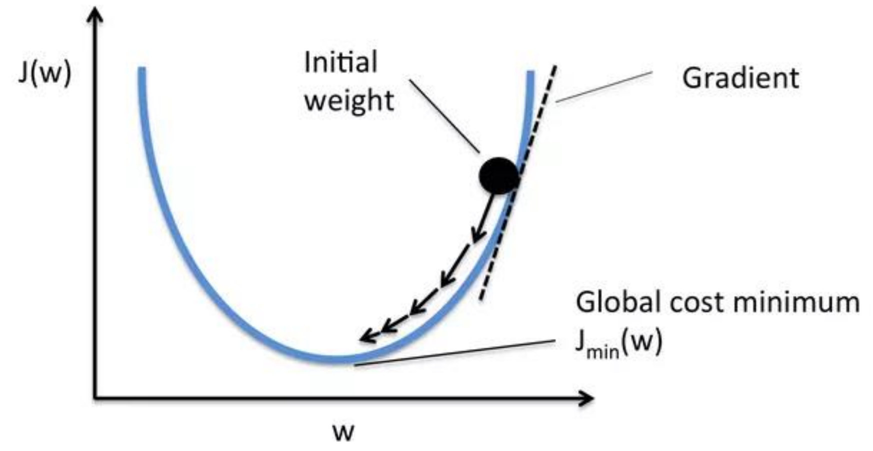


Optimization

The goal of optimization is to find a set of weights that minimizes the loss function

Image source: wikipedia

Gradient Descent

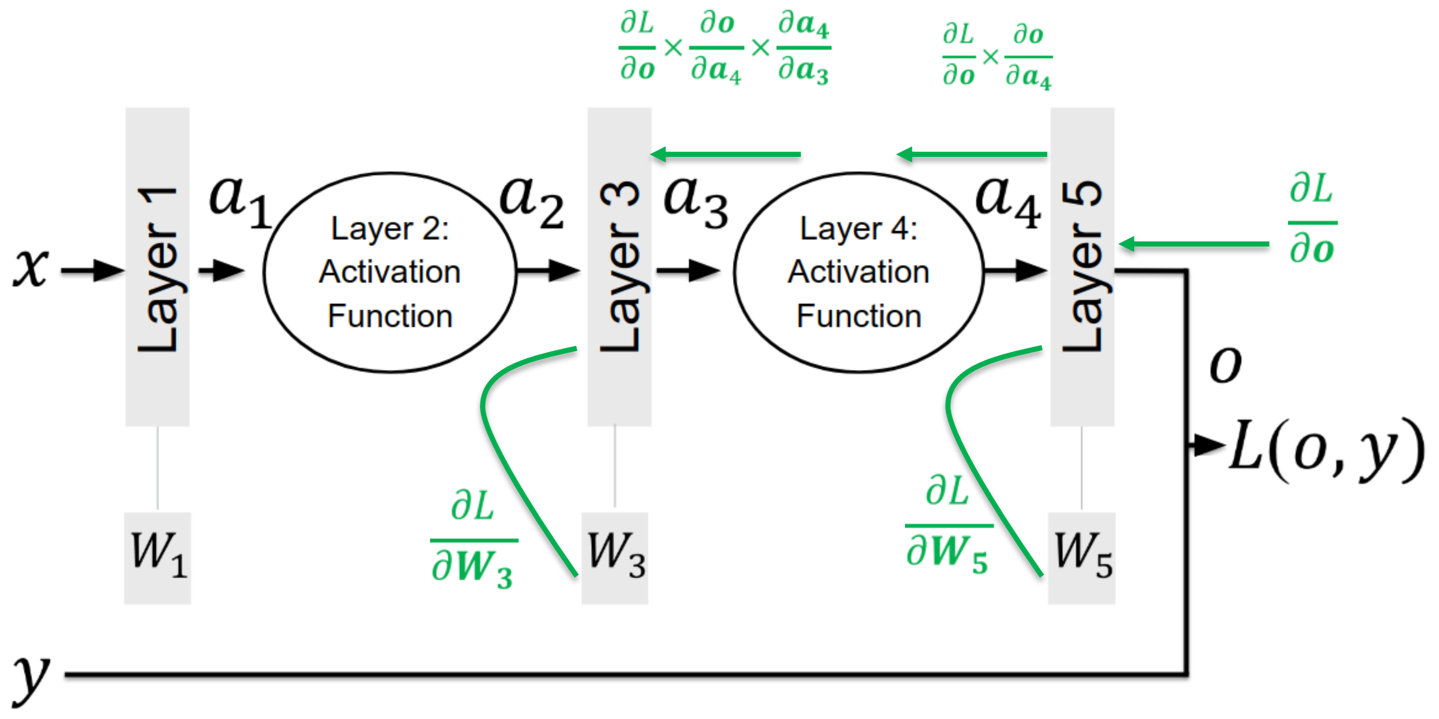


Optimisation functions usually calculate the **gradient** i.e. the partial derivative of loss function with respect to weights, and the weights are modified in the opposite direction of the calculated gradient. This cycle is repeated until we reach the minima of loss function.

Image and text source: xpertup

Backward propagation

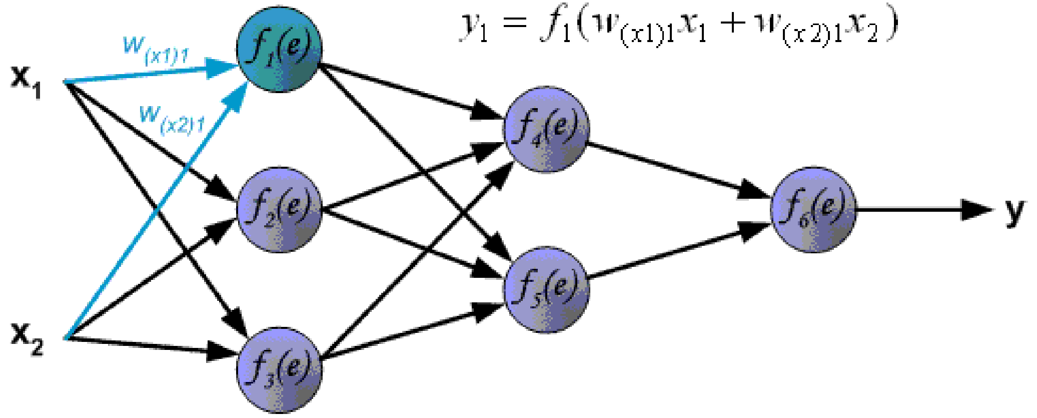
Backprop



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

Backprop

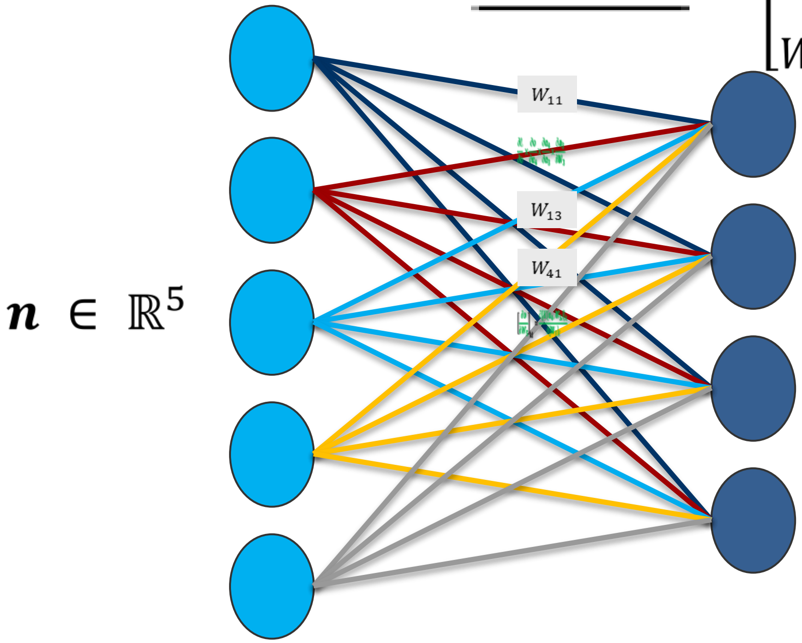
FP



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

Fully-connected layer

Fully connected layer



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

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

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

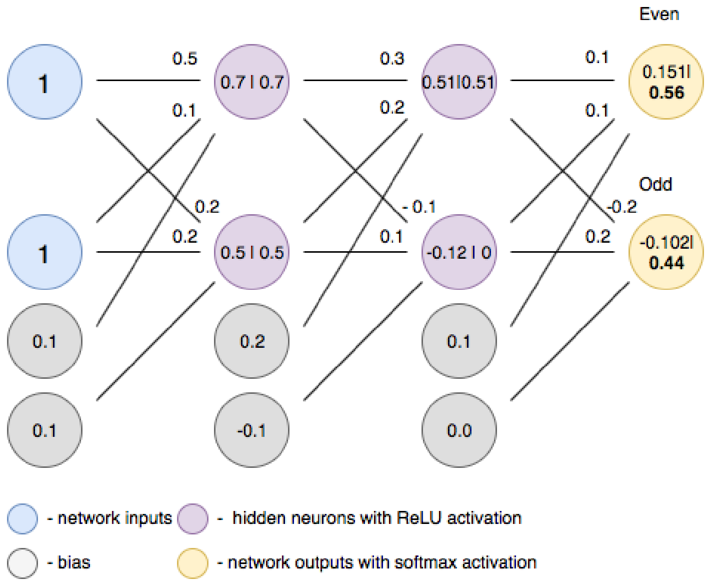
$$\boldsymbol{m} \in \mathbb{R}^4$$

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

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

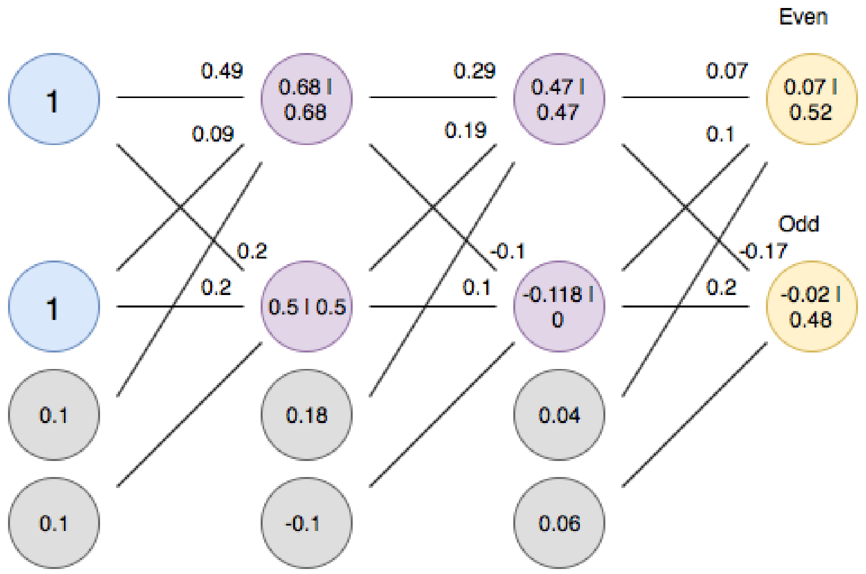
FC layer - forward pass

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.



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

Case Study – Credit Card Fraud Detection

Context

- It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

- In this case study we will construct a neural network based model using tf.keras sequential model.

Case Study – Data Description

Dataset

- The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

- It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Thank you! :)

Questions are always welcome