Deep learning

Zoltan Miklos

University of Rennes 1

2022

Plan

What is deep learning ?

- layered representation learning
- recently very successfully applied for classification, regression tasks, as well as data generation
- a learning method that can approximate complicated decision functions

Chair ?

http://www.dekoratifblog.com/beautify-your-chair-room-with-a-variety-of-chairs/

Feature engineering

- A critical step in the machine learning workflow : which features of the data shall we use ?
- Deep learning completely removes this step
- However we lose on interpretability : a network can predict with high accuracy, for example, whether an image depicts a chair, but we have no information how he could achieve this. The large number of learned weights do not give any insights. (while in case of a decision tree classification, humans can interpret the learned structure)

Plan

Neural networks : roadmap for the topics to come

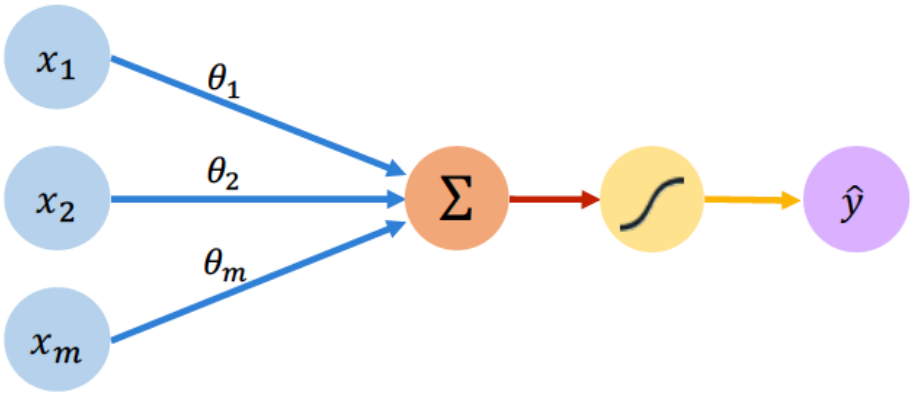
- Perceptron : basic neural network
- Multi-layer perceptron, feed-forward networks
- Training neural networks, backpropagation
- How to influence the training process ? (early stopping, regularization, dropout, etc.)
- Convolutional neural networks, basic image processing
- Recurrent neural networks, LSTM
- Word-embeddings, processing text or other sparse data
- Machine learning strategy
- Other topics

Perceptron

- Basic building block of artificial neural networks
- Forward propagation : the perceptron takes a weighted sum of the inputs and produces an output using a non-linear activation function
- $$\hat{y} = g(\Sigma_{i=1}^{n} x_i \theta_i)$$
- in vector notation $\hat{y} = g(\Theta^\top X)$

Perceptron : forward propagation

The Perceptron: Forward Propagation



Inputs Weights Sum Non-Linearity Output

Output
Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i\ \theta_i\right)$$

Non-linear activation function

http://introductiontodeeplearning.com

Bias term

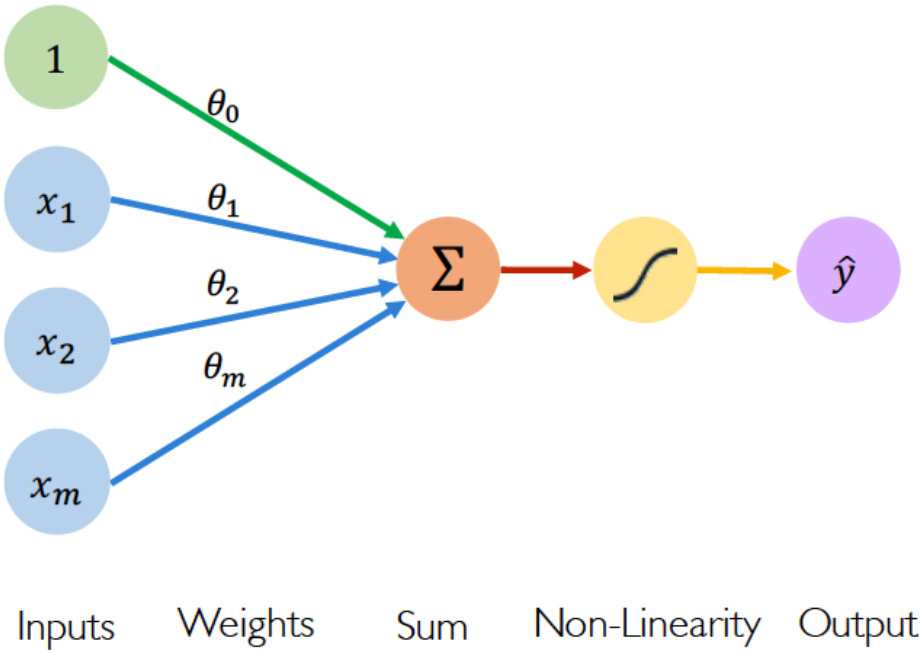
- Often we would like to add a bias term as well

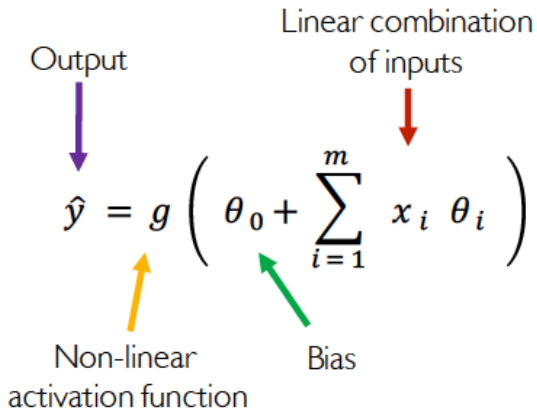
$$\hat{y} = g(\theta_0 + \Sigma_{i=1}^{n} x_i \theta_i)$$

- we can consider that we have a constant 1 input (the 0th input)
- in vector notation $\hat{y} = g(\Sigma_{i=1}^{n} \Theta^\top X)$,
  where $X = (1, x_1, \ldots, x_n)^\mathsf{T}$ and $\Theta = (\theta_0, \theta_1, \ldots, \theta_n)^\mathsf{T}$

Perceptron : the bias term

The Perceptron: Forward Propagation





http://introductiontodeeplearning.com

Activation functions

Non-linear functions :

- Sigmoid

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

- reLU (Rectified Linear Unit)

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

- tanh

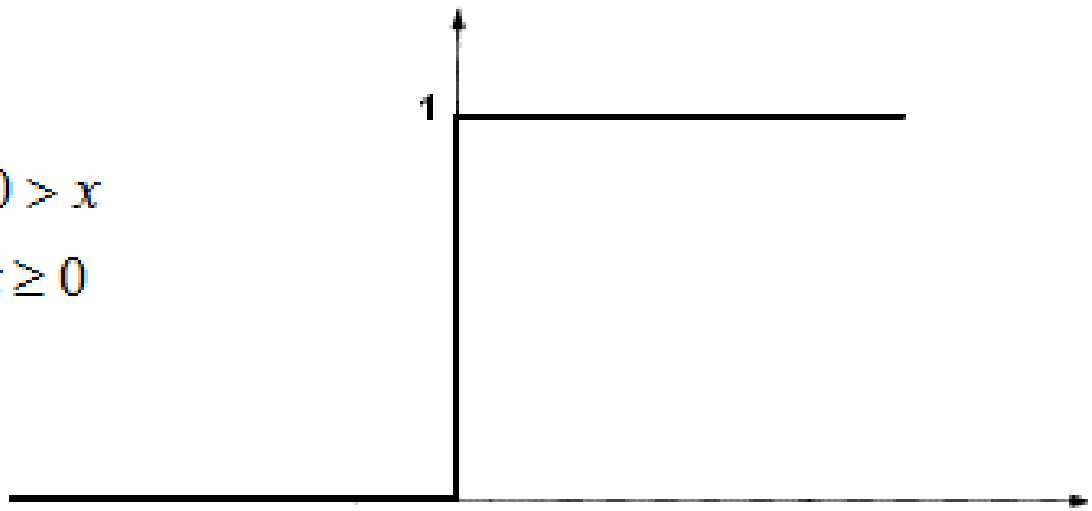
$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Activation functions : Unit-step threshold

Not used in practice (as it is not it is not differentiable), but helps to demonstrate the concepts in small toy examples

Unit step (threshold)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



https://www.saedsayad.com/artificial_neural_network.htm

Logical gates

Exercise : Create a (single layer) perceptron that works as a logical gate and realizes the

- NOT gate
- AND gate
- OR gate

XOR : is it possible to realise with a single layer perceptron ?

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Table – XOR

Decision surface and the XOR gate

Decision boundary : a hypersurface that partitions the underlying vector space into two sets, for each class (0/1).
On can identify a separating hyperplane in the case of AND and OR functions, but there is no separating hyperplane for XOR :

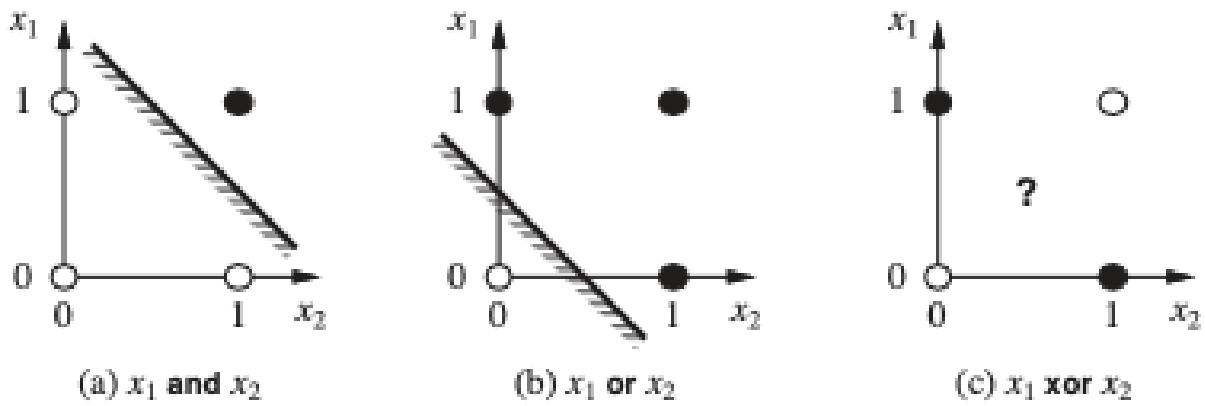


Figure 18.21 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c), no such line exists that correctly classifies the inputs.

Russell, Norvig. Artificial intelligence.

XOR : a historical side-note

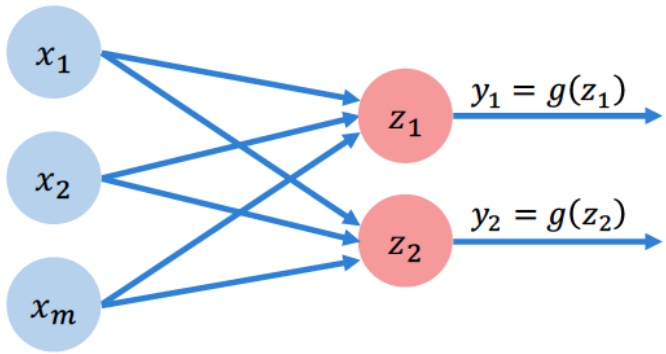
- Frank Rosenblatt (1957)
- Marvin Minsky and Seymour Papert's book on Perceptrons (1969) showed that they (single-layer version) cannot compute XOR
- While Minsky and Papert understood that multi-layer version can do it, but incorrect citations contributed to a decline of neural network research, until the 1980's

Deep neural network

- Use multiple hidden layers
- use non-linear activation functions
- IMPORTANT : one can consider the deep neural network as an ensemble of perceptions (where we apply logistic regression several times). However the high number of neurons (i.e. weights) changes the behaviour of this learning system qualitatively. This phenomenon is poorly understood theoretically.

Multi Output Perceptron

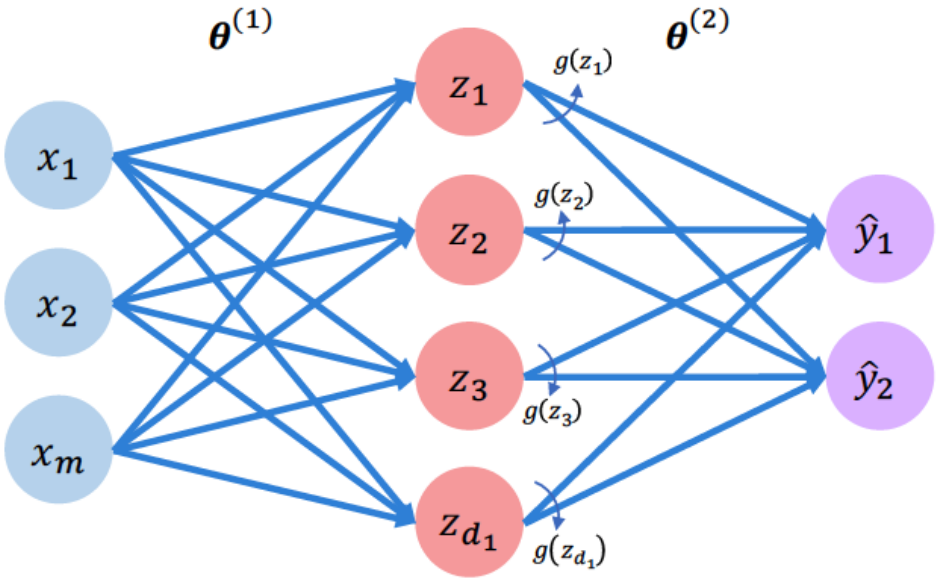
Multi Output Perceptron



$$z_i = \theta_{0,i} + \sum_{j=1}^{m} x_j \, \theta_{j,i}$$

`http://introductiontodeeplearning.com` $\theta_{j,i}$L the weight of the edge from input x_j to node z_i

Neural Network with a hidden layer



$$z_i = \theta_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, \theta_{j,i}^{(1)} \qquad \hat{y}_i = \theta_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, \theta_{j,i}^{(2)}$$

http://introductiontodeeplearning.com

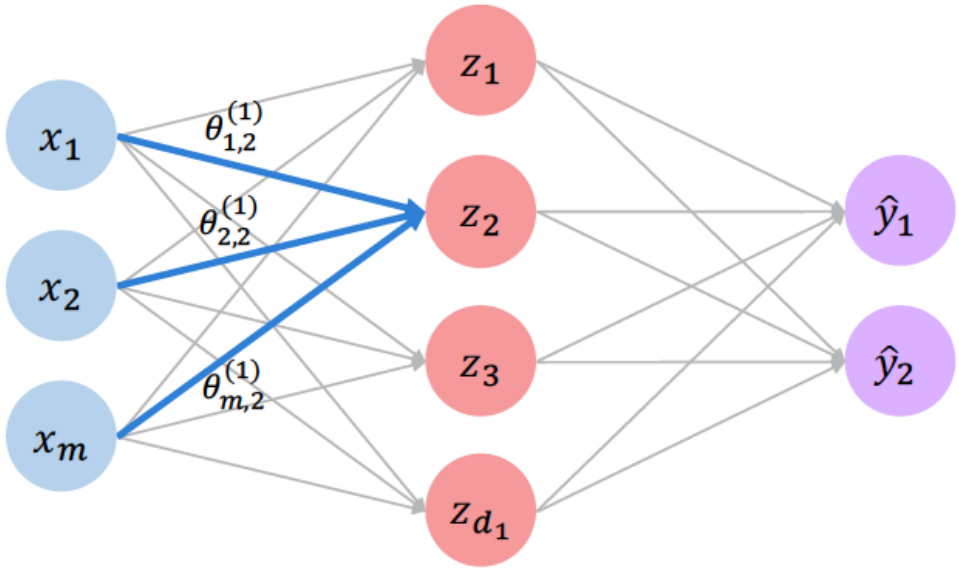
Forward calculations in the neural network

- Input layer, Hidden layer, Output layer
-

$$z_i = \theta_{0,i}^{(1)} + \Sigma_{j=1}^{n} x_j \theta_{j,i}^{(1)}$$

$$\hat{y}_i = \theta_{0,i}^{(2)} + \Sigma_{j=1}^{d_1} z_j \theta_{j,i}^{(2)}$$

A visual help for understanding the formulas



$$z_2 = \theta^{(1)}_{0,2} + \sum_{j=1}^{m} x_j \, \theta^{(1)}_{j,2}$$

$$= \theta^{(1)}_{0,2} + x_1 \, \theta^{(1)}_{1,2} + x_2 \, \theta^{(1)}_{2,2} + x_m \, \theta^{(1)}_{m,2}$$

http://introductiontodeeplearning.com

Plan

Training neural networks

We would like to use the neural networks for classification/prediction

- Supervised learning : we use a (large) set of labelled objects
- Training data : $(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_n}, y_n)$
- Training data : set of data items $\mathbf{x_i}$, for which the correct label is y_i (ideally hundreds of millions of data items)
- We define the topology of the network
- The model we would like to learn is the set of weights $\Theta_{i,j}^{(k)}$ (k-th layer, weight between the input i and output j)

Loss function

- Characterize the differences between the predicted and real output
- Cross entropy loss (for outputs between 0 and 1) (discrete case) $-\Sigma_i y_i log_2(h_w(x_i)) + (1 - y_i) log_2(h_w(x_i))$
- Mean-squared error loss
- (Besides the widely used loss functions you can also a specific one, more adapted to your particular problem.)

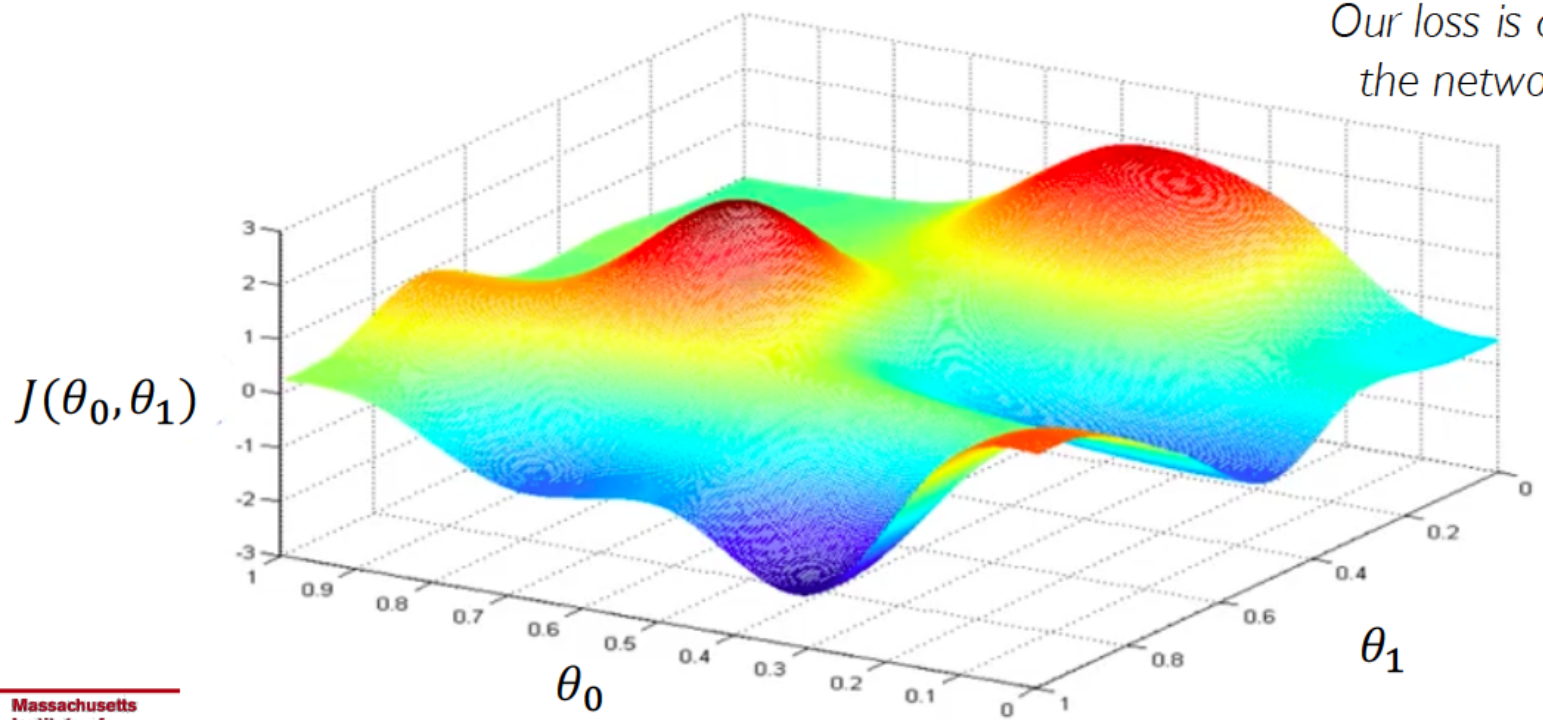
Training neural networks

- We would like to find the weights that result the lowest loss
- We can consider this as an optimization : minimize the loss function (that is a function of all weights and biases)
- Optimization : use the partial derivatives to find the optimum
- Analytical solution : calculate all the partial derivatives, set them equal to 0 ans solve the equations (however, this is only possible for very small networks, already obtaining the derivatives is too costly)

Loss Optimization

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$$

Remember:
Our loss is a function o
the network weights!



Massachusetts Institute of

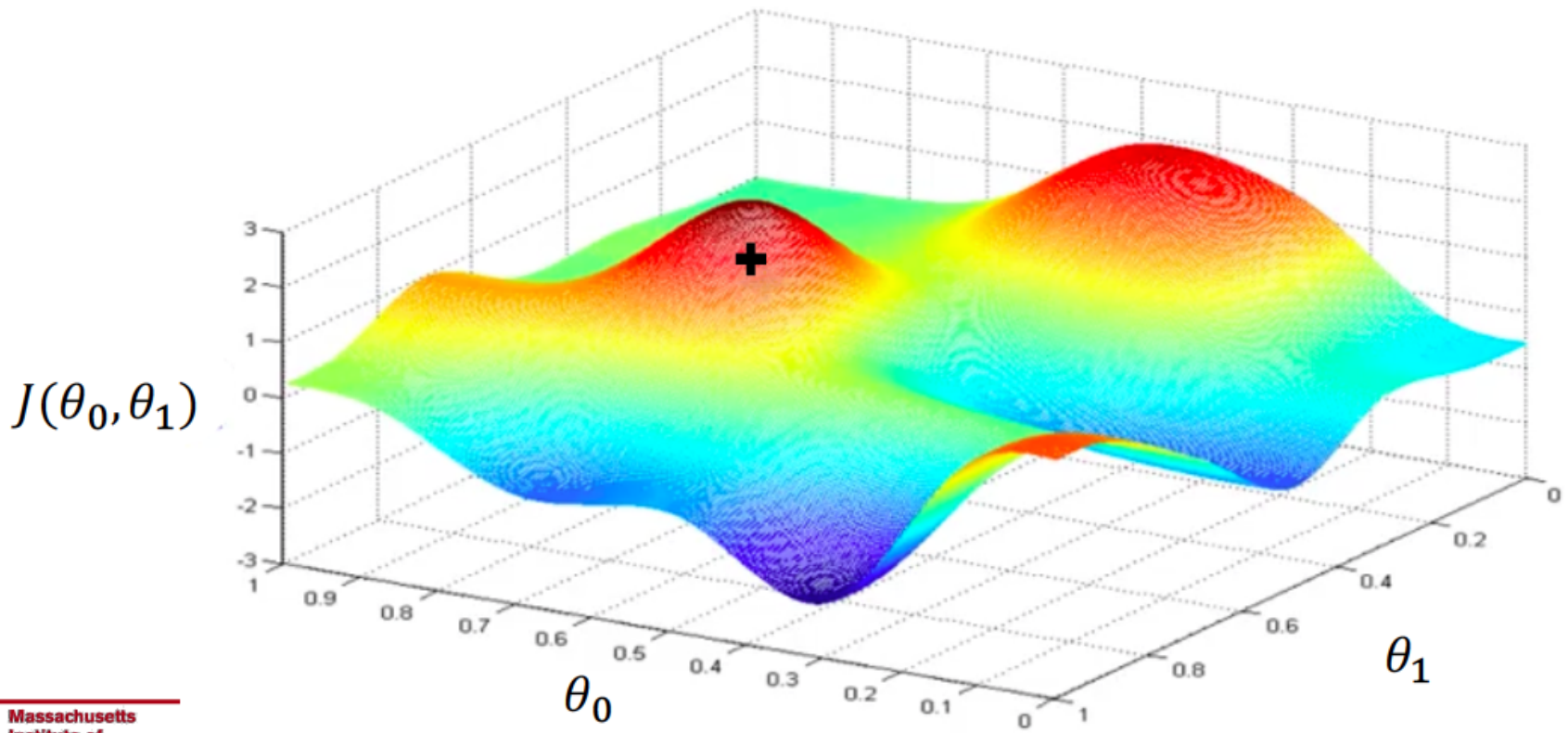
http://introductiontodeeplearning.com

Gradient descent

- Initialize the weights randomly
- Loop until convergence
  - Compute the components of the gradient $\nabla J = (\frac{\partial J}{\partial \theta_1}, \ldots, \frac{\partial J}{\partial \theta_n})$
  - Update the weights $\theta^{(i+1)} \leftarrow \theta^{(i)} - \gamma \nabla J$ where $\gamma \in [0, 1]$ is the learning rate (parameter).
  - Attention : θ here is a vector (of n dimension) that contains all network weights (including the bias terms)
- Return the weights

Gradient Descent

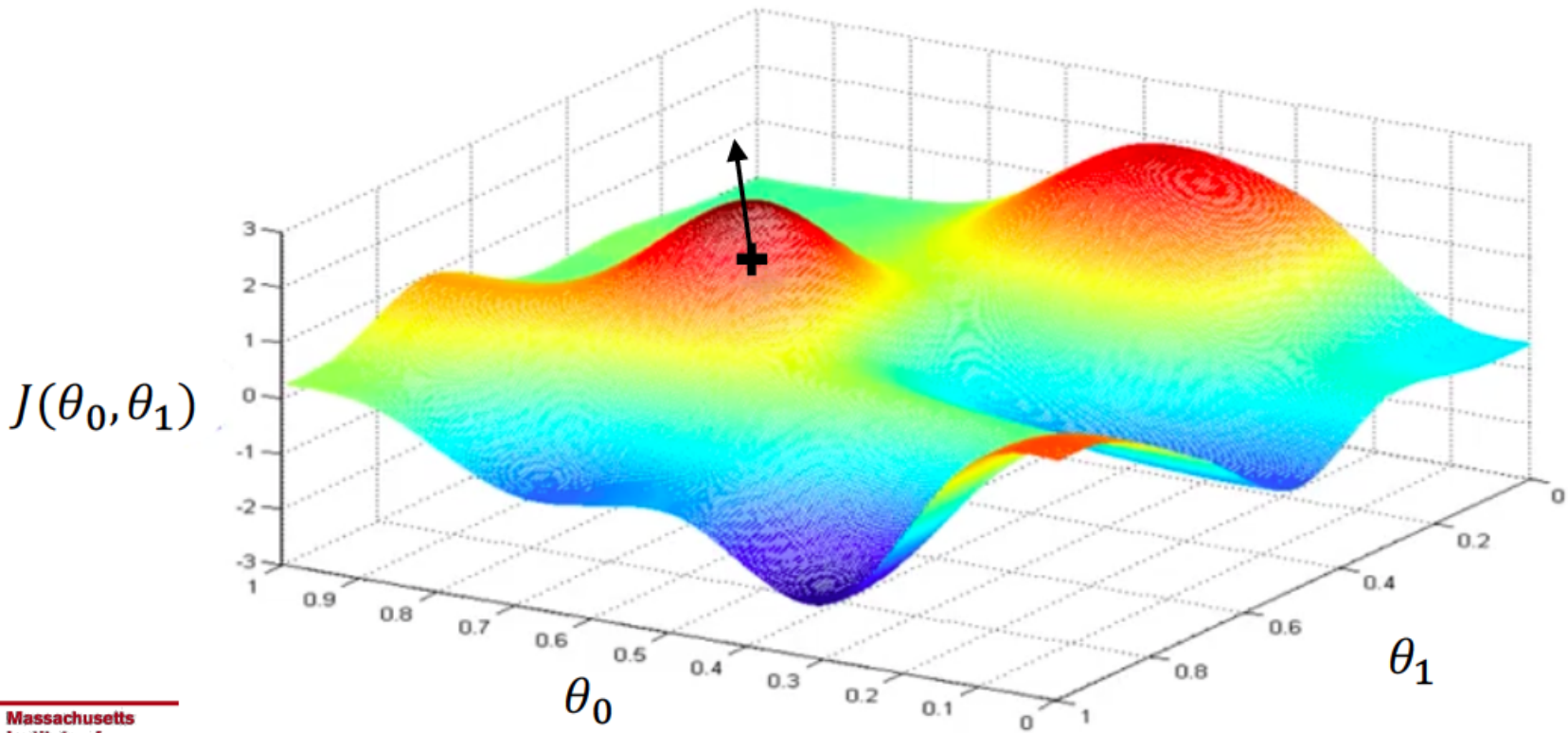
Randomly pick an initial (θ_0, θ_1)



Massachusetts Institute of

Gradient Descent

Compute gradient, $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$

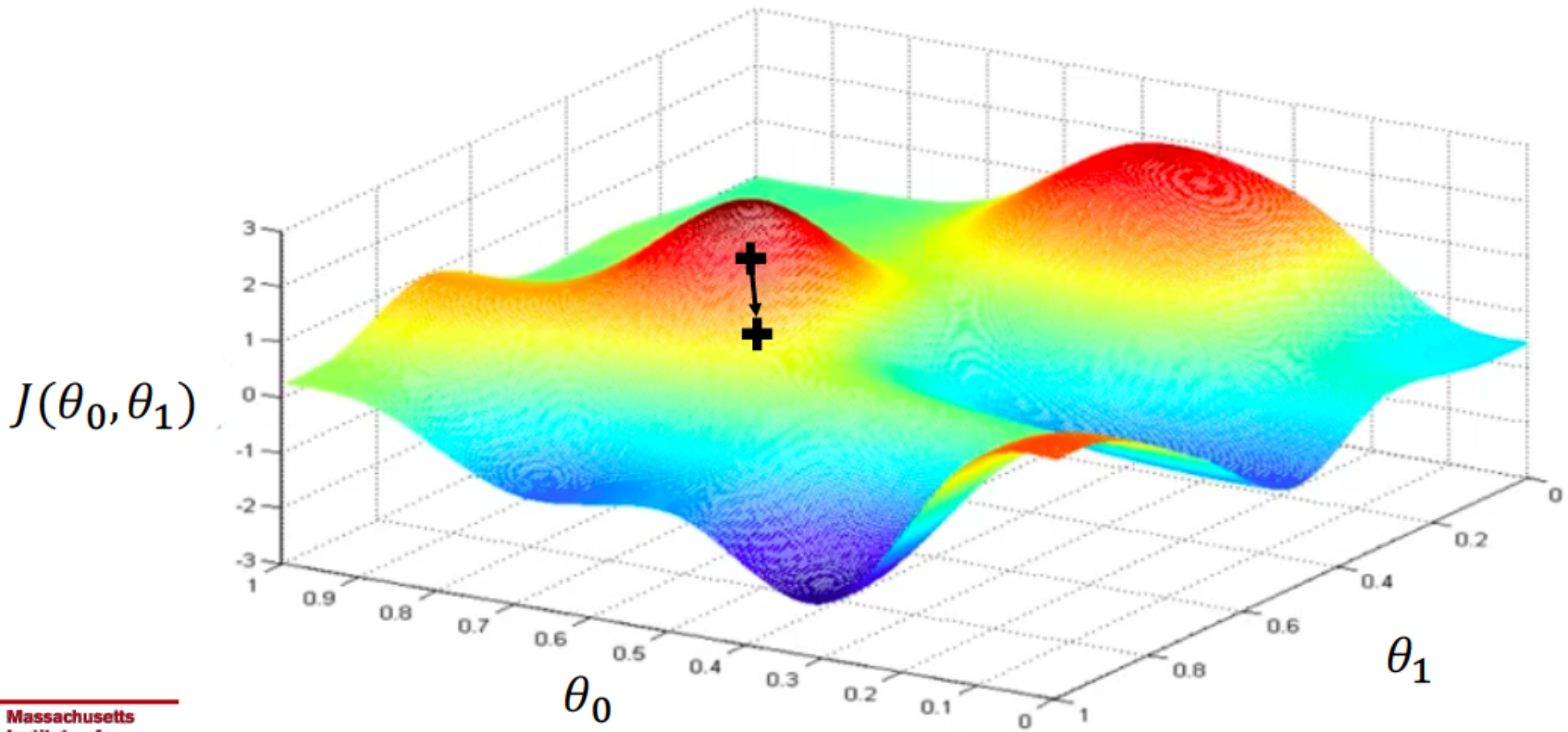


Massachusetts Institute of Technology

http://introductiontodeeplearning.com

Gradient Descent

Take small step in opposite direction of gradient

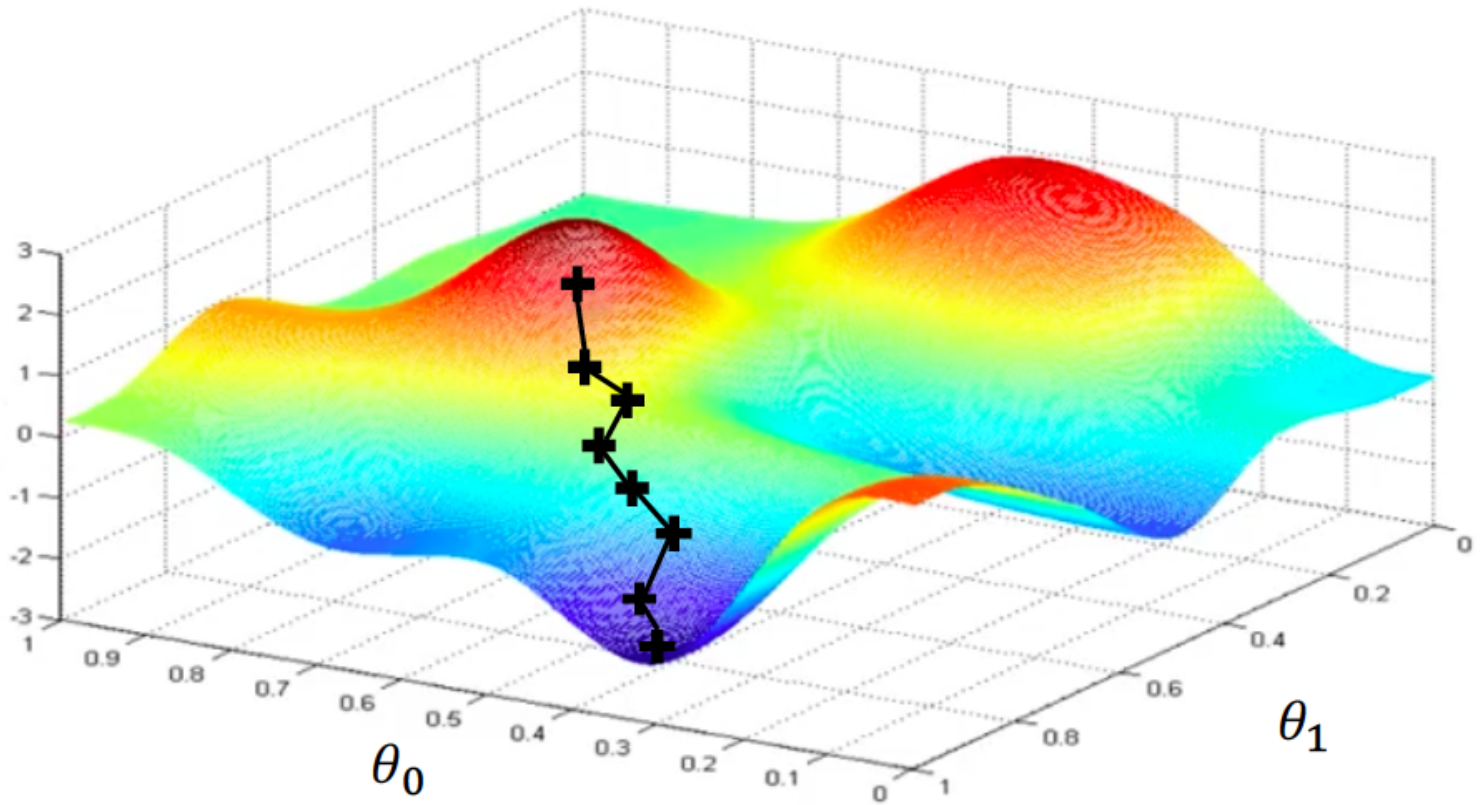


Massachusetts Institute of Technology

Gradient Descent

Repeat until convergence

$J(\theta_0, \theta_1)$

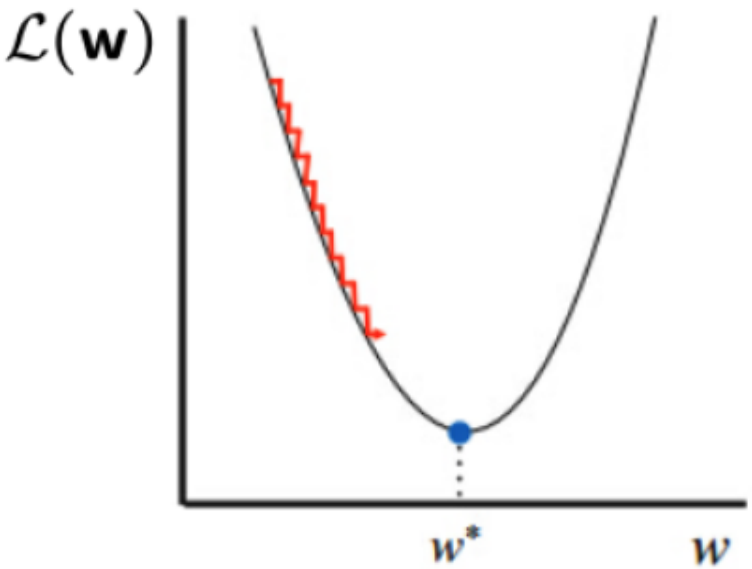


Massachusetts Institute of

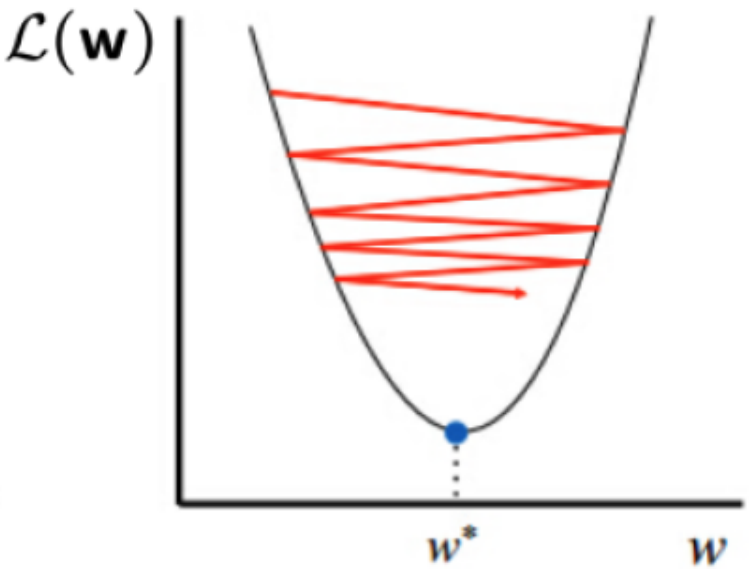
Learning rate

- Setting the learning rate is not easy
  - Small learning rate can lead to slow convergence and the algorithm could get stuck in a local minimum
  - Large learning rate : the algorithm might even diverge and we could miss the minimum
- It is important to monitor the learning process (e.g. by measuring accuracy of the trained models)

Learning rate



Too small: converge very slowly



Too big: overshoot and even diverge

http://webia.lip6.fr/~cord/pdfs/courses/multimedia/

Backpropagation

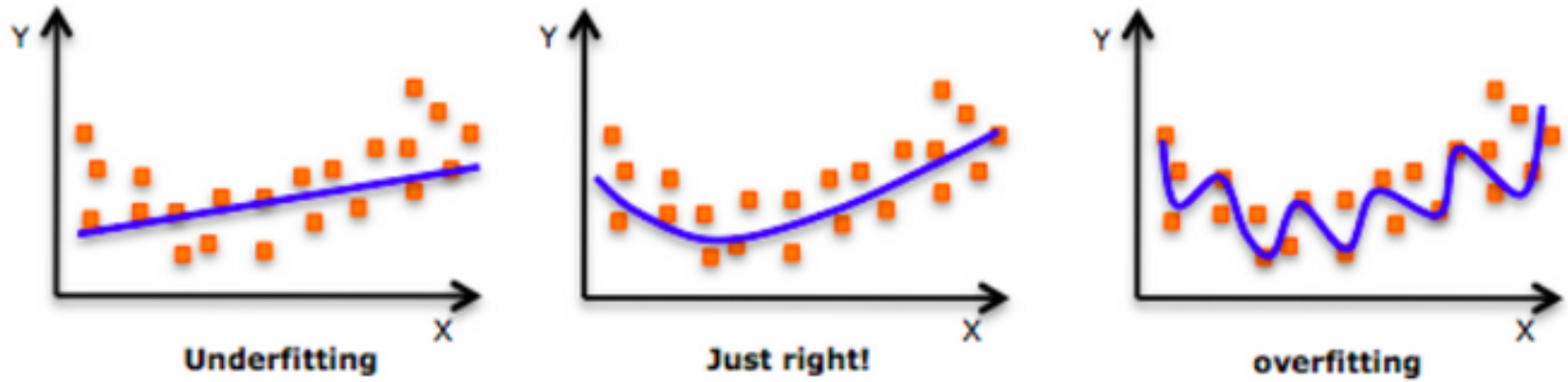
Learning algorithm of the neural networks

- Rumelhart, Hinton, Williams in 1986
- Use forward propagation to compute output, evaluate the loss function and then
- propagate back, that is update the weights (from the output layer back towards the input layer)
- updates correspond to the chain rule (of derivation)
- In this way the network computes implicitly the derivatives of the cost function
- epoch : complete set of computations where you see all your training data

Stochastic gradient descent (SGD)

- Gradient descent can be slow if the training set is very large
- Gradient descent : $\theta_i \leftarrow \theta_i - \eta \frac{\partial J}{\partial \theta_i}$
- Stochastic gradient descent :
  - Randomly shuffle the training data
  - The gradient is estimated using a small set of samples (instead of the entire training set)
- Other optimization techniques (available in tensorflow or elsewhere)
  - Momentum
  - Adagrad
  - Adam
  - RMSprop

Overfitting



https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Regularization

- Techniques to constraint the optimization problem (to avoid too complex models)
- Can help to avoid overfitting
- $L1$ (Lasso) :

$$Cost = Loss + \frac{\lambda}{2m}\Sigma|\theta|$$

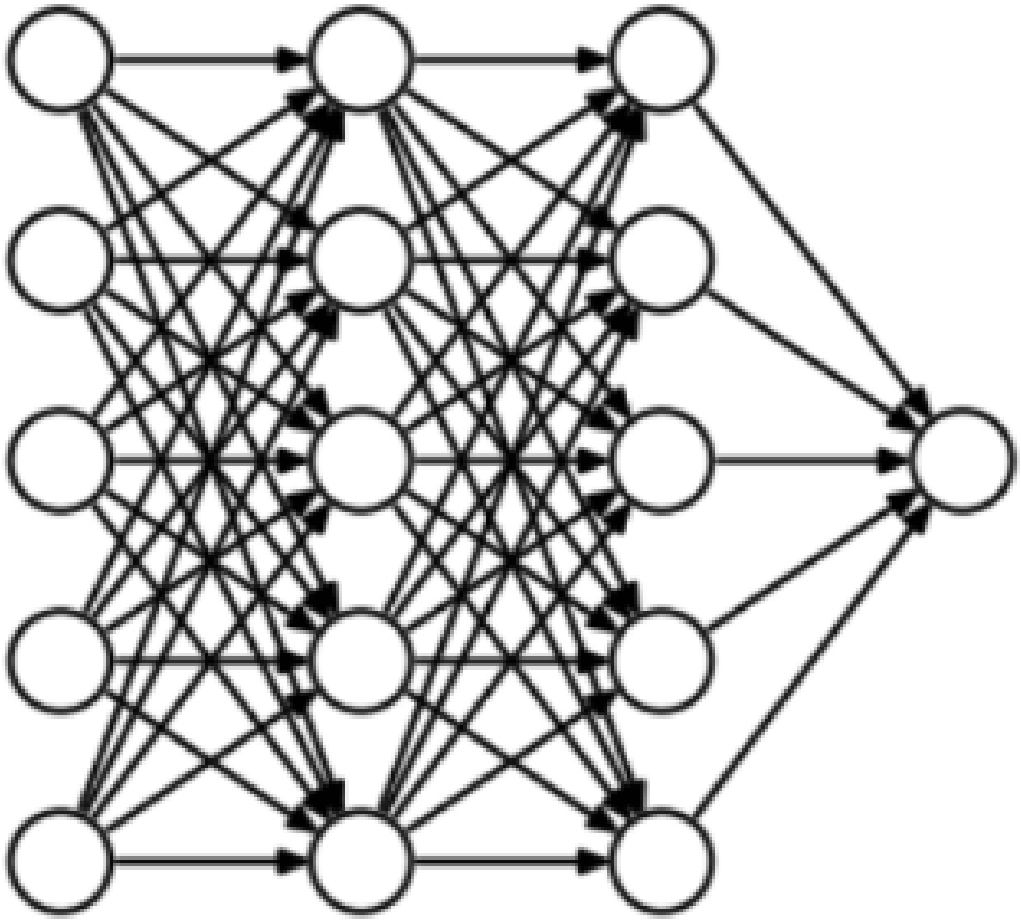
- $L2$ (Ridge) :

$$Cost = Loss + \frac{\lambda}{2m}\Sigma|\theta|^2$$

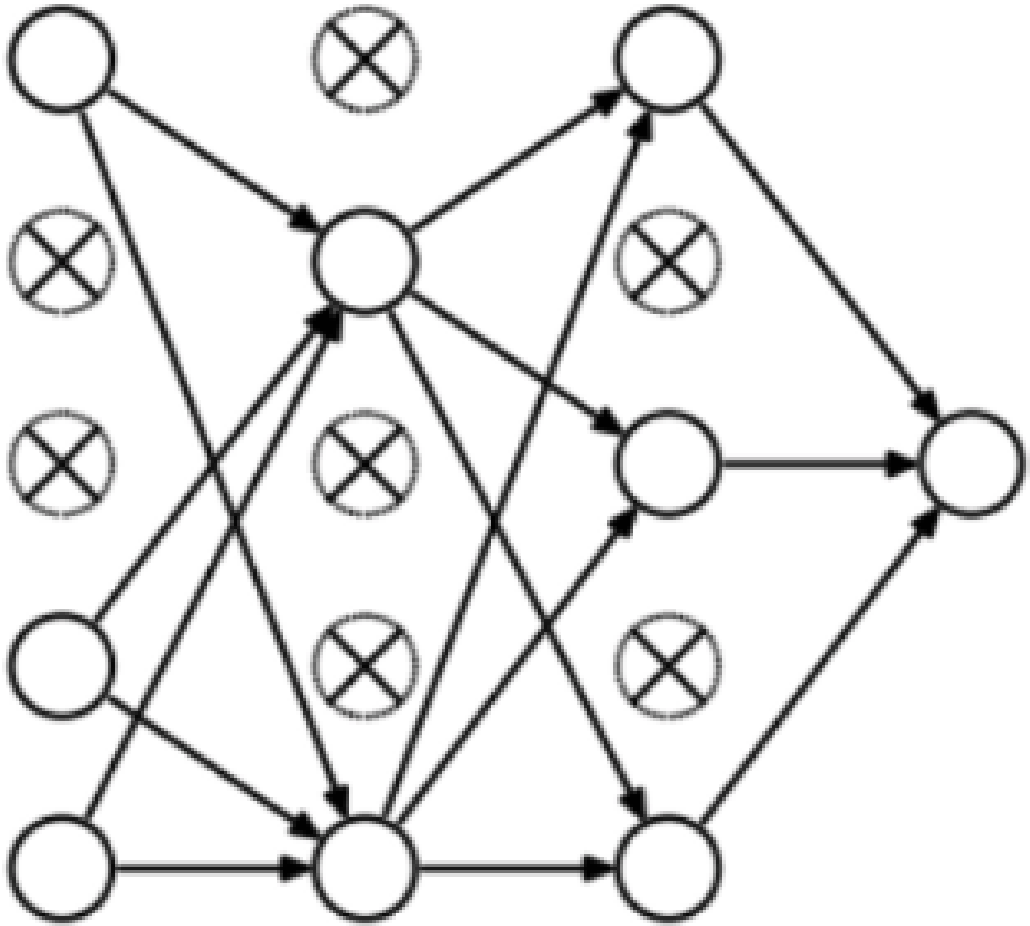
Dropout

- Set some activations to 0 (randomly, with some probability that is a parameter)
- Avoids that a network relies on some particular nodes

Dropout : example (compete network)



Dropout : example (connections after the dropout operation)

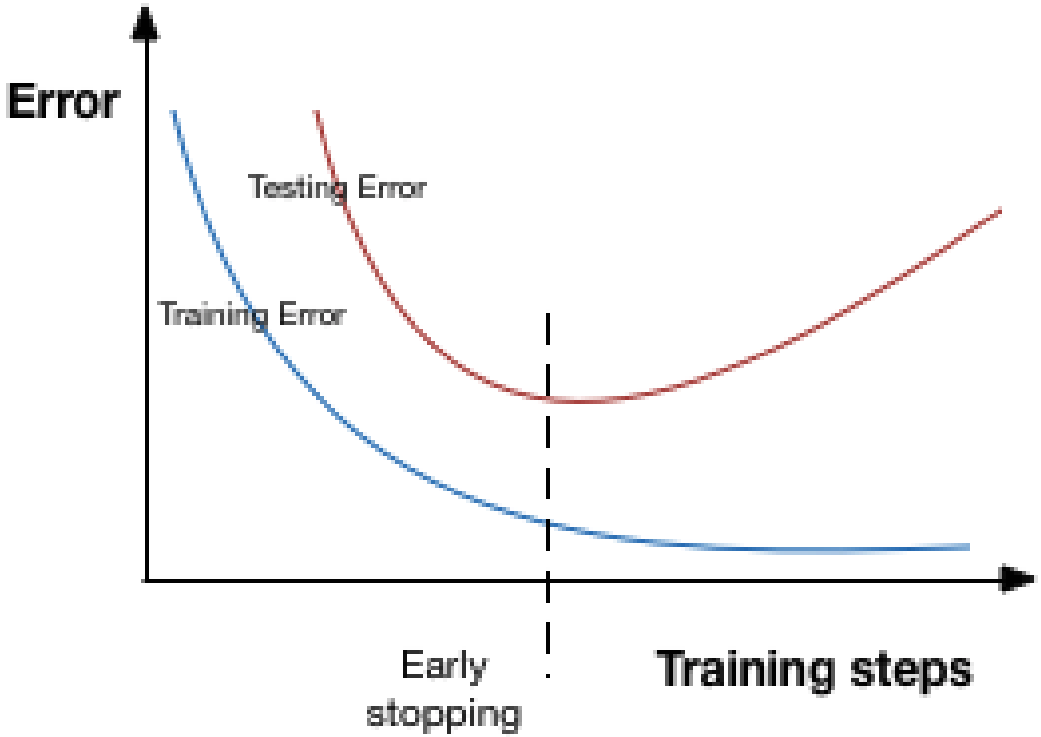


Early stopping

Early stopping is a technique to prevent overfitting

- monitor the error of the trained models
- stop the training of the network early (before convergence)
- In this way, we will not reach the maximum possible accuracy on the training set, but this could lead to better performance on the test set (if we manage to avoid overfitting)

Early stopping



https://www.analyticsvidhya.com/blog/2018/04/