Introduction to Machine Learning: Neural Networks

Stefania Sarno

January 2, 2023

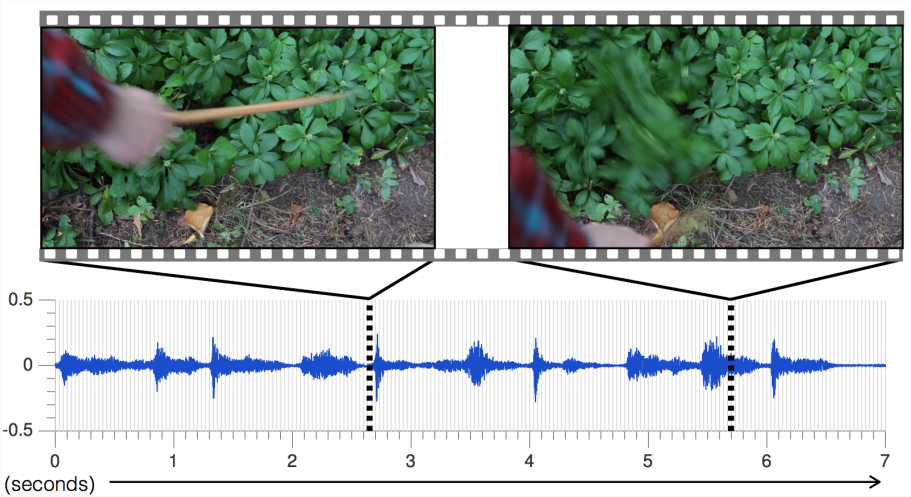
Outline

1. Recent successes of neural networks and deep learning
2. Historical development of neural networks
3. Perceptron (learning rule and limitations)
4. Multi-layer Perceptrons and the Universal Approximation Theorem
5. Backpropagation Algorithm

Automatic Translation

- Breakthrough of Google Translate (2016)
- But neural network can also automatically translate from images

Sound Generation form Silent Videos



https://news.mit.edu/2016/artificial-intelligence-produces-realistic-sounds-0613

Image segmentation

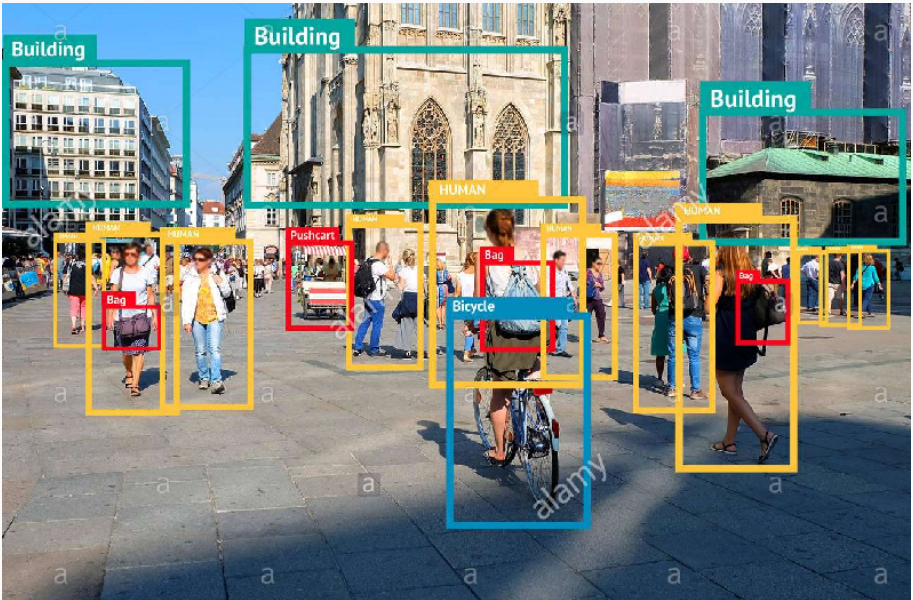
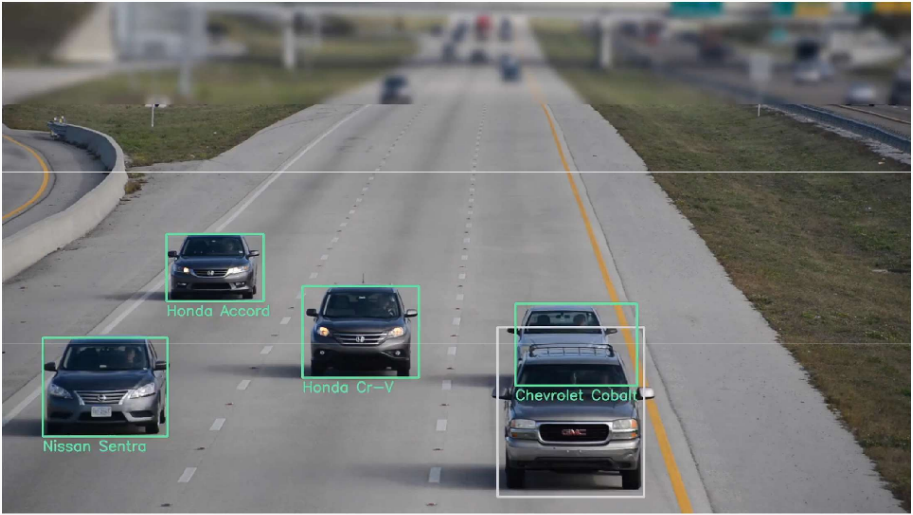


Image Recognition



https://www.sighthound.com/technology/ 9

Fake Face Generation

https://medium.com/datadriveninvestor/artificial-intelligence-gans-can-create-fake-celebrity-faces-44fe80d419f7

Image Captions Generation

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

Alpha Go

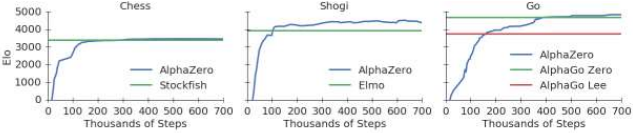
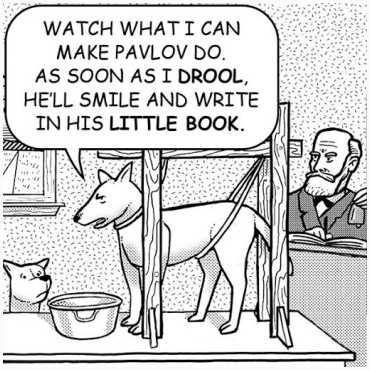


Figure 1: Training *AlphaZero* for 700,000 steps. Elo ratings were computed from evaluation games between different players when given one second per move. **a** Performance of *AlphaZero* in chess, compared to 2016 TCEC world-champion program *Stockfish*. **b** Performance of *AlphaZero* in shogi, compared to 2017 CSA world-champion program *Elmo*. **c** Performance of *AlphaZero* in Go, compared to *AlphaGo Lee* and *AlphaGo Zero* (20 block / 3 day) (*29*).

What are Neural Networks?

Everything began with the study of the brain

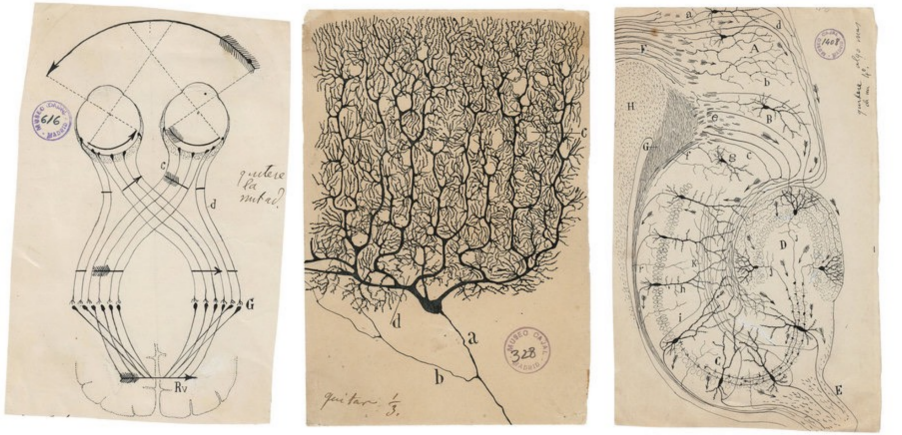
Earliest Model of Cognition: Associationism



- The ability to associate ideas is the only mental process
- We learn by cause and effect, contiguity, resemblance
- Behaviour is learned from repeated associations of actions with feedback

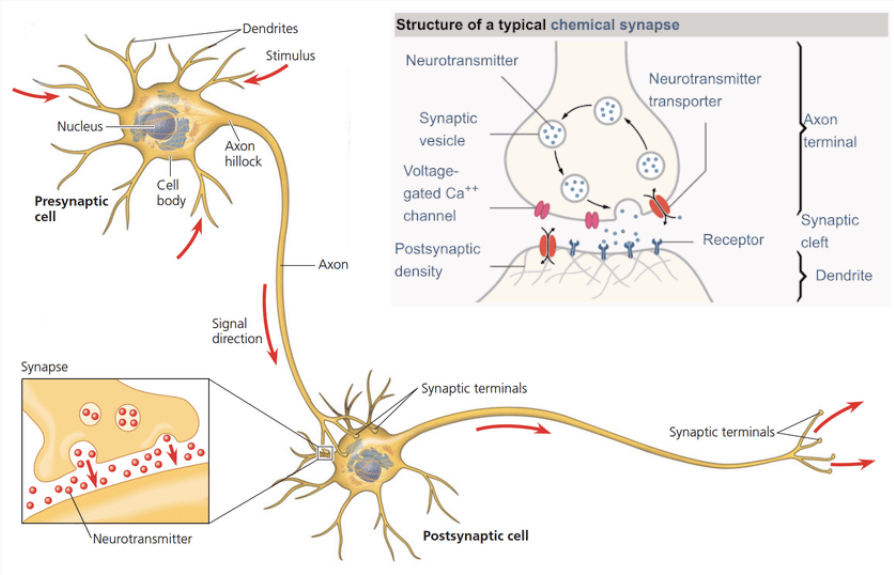
But where are associations stored?

The Discovery of the Neurons (late 1800)

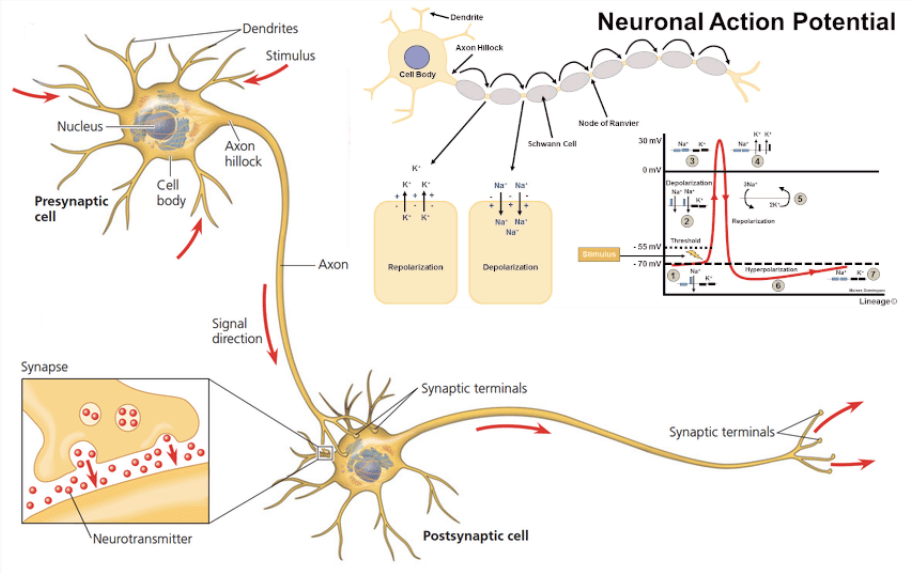


Illustrations by Santiago Ramón y Cajal, the Spanish neuroscientist, from the book "The Beautiful Brain." From left: A diagram suggesting how the eyes might transmit a unified picture of the world to the brain; a purkinje neuron from the human cerebellum; and a diagram showing the flow of information through the hippocampus in the brain.

How Real Neurons Work

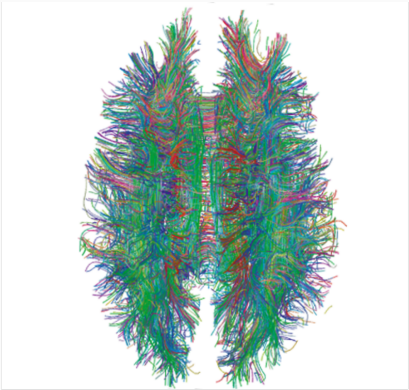


The Action Potential



The Brain as a Connectionist Machine

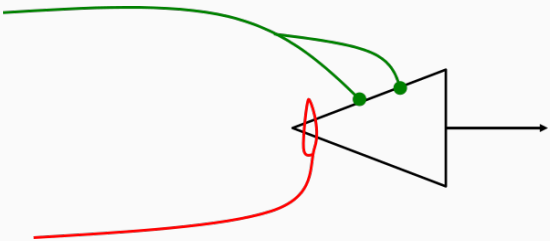
- There are ~100 billion neurons
- Each neurons has an average of 7000 synaptic connections
- 10^{15} synapses
- 1.3 – 1.4 kg



Gigandet X, Hagmann P, Kurant M, Cammoun L, Meuli R, et al. *PLoS ONE* (2008)

Connectionism: Everything is in the connections

McCulloch and Pitts model (1943)



"A Logical Calculus of the Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics

- Excitatory synapse: Transmit weighted input to the neuron
- Inhibitory synapse: If receiving signal, it prevents the neuron from firing

Hebbian Learning

Donald Hebb: *Organization of Behaviour*, 1949:

Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. ... When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

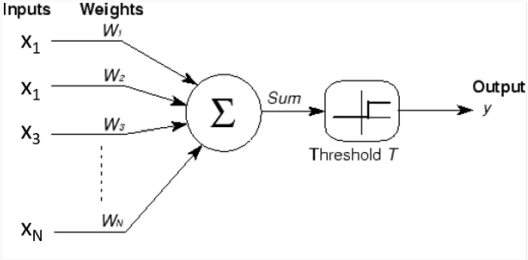
Neurons that fire together wire together

If neuron x repeatedly triggers neuron y their connection strength will change as:

$$w_{xy} = w_{xy} + \alpha xy$$

The Percepton: simplified model and leraning algorithm

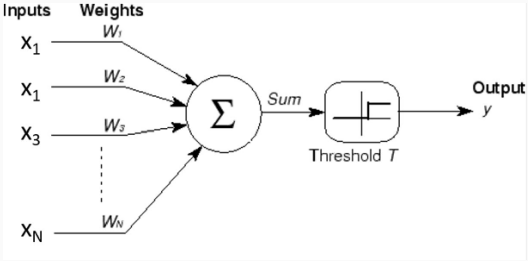
Frank Rosenblatt, 1958 (Boolean tasks and proved convergence)



$$Y = \begin{cases} 1 \;\; if \;\; \sum_i w_i x_i - T \geq 0 \\ 0 \qquad\qquad else \end{cases}$$

The Percepton: simplified model and leraning algorithm

Frank Rosenblatt, 1958 (Boolean tasks and proved convergence)

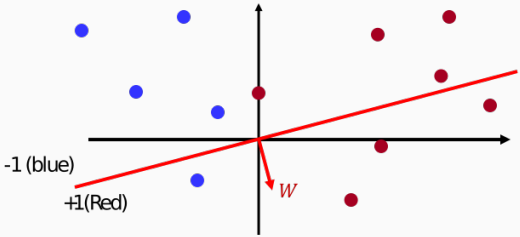


$$Y = \begin{cases} 1 \;\; if \;\; \sum_i w_i x_i - T \geq 0 \\ 0 \qquad\qquad else \end{cases}$$

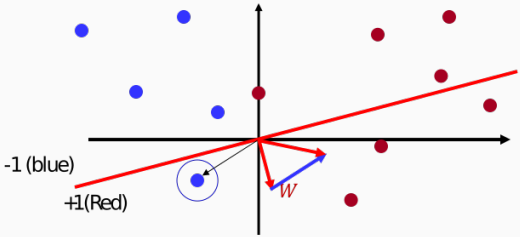
- Update the weights whenever the perceptron output is wrong as:

$$\mathbf{w} = \mathbf{w} + \alpha[d(\mathbf{x}) - y(\mathbf{x})]\mathbf{x}$$

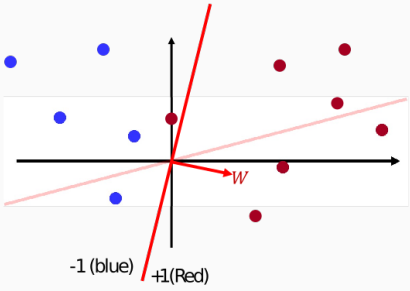
The Perceptron: the Learning Algorithm



The Perceptron: the Learning Algorithm



The Perceptron: the Learning Algorithm



The Boolean Gates

AND

INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

OR

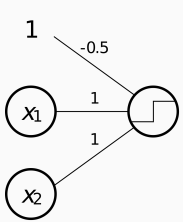
INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

XOR

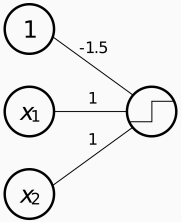
INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

The XOR Problem

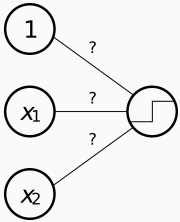
Minsky and Papert, 1968



OR



AND



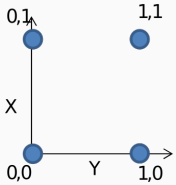
XOR

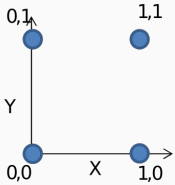
The perceptron cannot output the XOR boolean operator (correct weights do not exist)

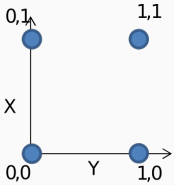
The XOR Problem

Minsky and Papert, 1968

Why do we have this?



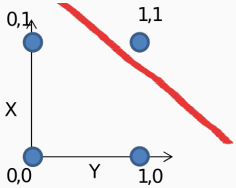


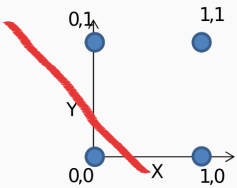


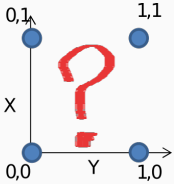
The XOR Problem

Minsky and Papert, 1968

Why do we have this?



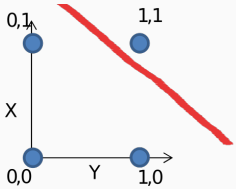


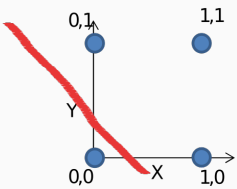


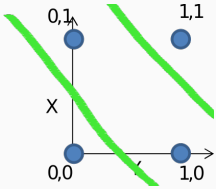
The XOR Problem

Minsky and Papert, 1968

Why do we have this?

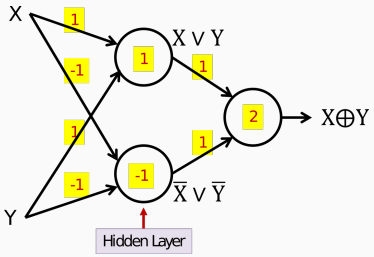






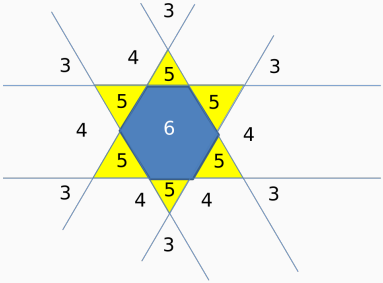
Solving the XOR Problem: the Multilayer Perceptron (MLP)

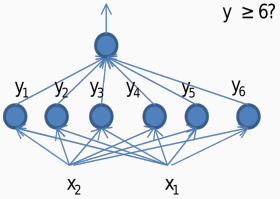
Minsky and Papert, 1968

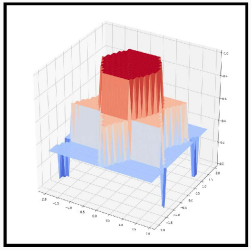


The XOR boolean operator can be obtained combining two perceptrons

MLP as Function Approximators

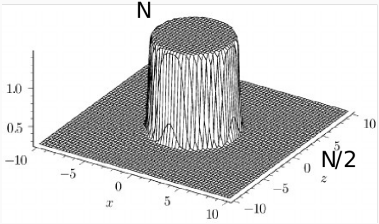


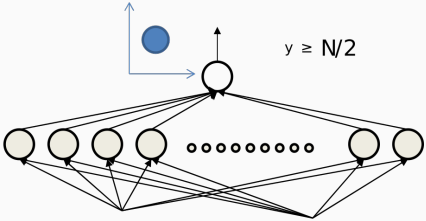




MLP as Function Approximators

For a very large number *N* of hidden units

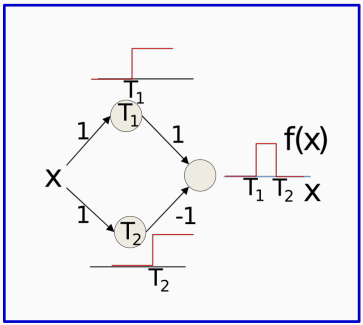


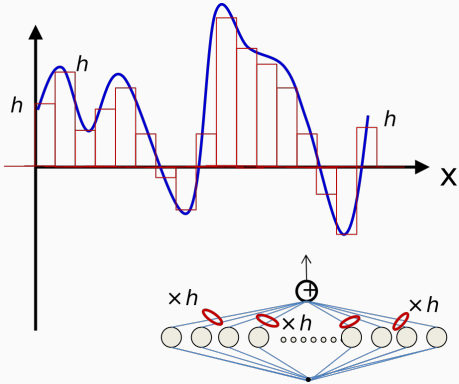


Once I have a circle I can do any weird boundary

MLP as Function Approximators

We can also approximate real function with arbitrary precision





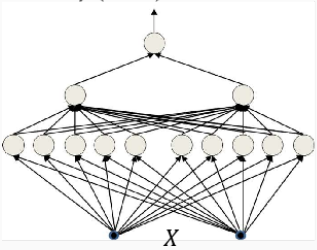
Universal Approximation Theorem

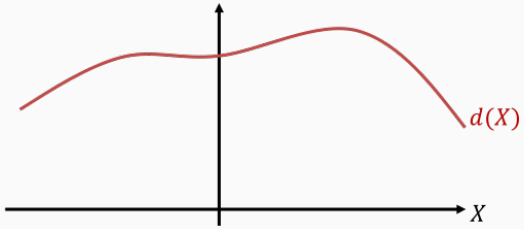
Formally we have the **universal approximation theorem**:

- A feed-forward network with a single hidden layer containing a finite number of neurons can approximate a continuous function on compact subset $\mathbb{R}^n$, under mild assumption on the activation function
- Proved by George Cybenko in 1989 for logistic activation function

Neural Networks as Universal Approximators

$Y = f(X; \boldsymbol{W})$



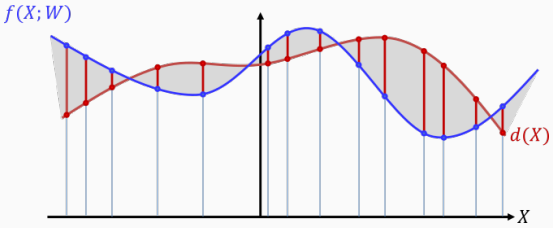


- Neural networks can approximate any function :
  here $d(X)$ is the function to approximate and $f(X, \mathbf{W})$ is the network output
- We want a neural network able to minimize the following "distance":

$$distance\big[f(X, \mathbf{W}), d(X)\big]$$

- We have to determine the weights and bias ($\mathbf{W}$) to get a good approximation

The Sampling Idea



Problem: we do not know $d(X)$ at all points X
Sampling Solution:

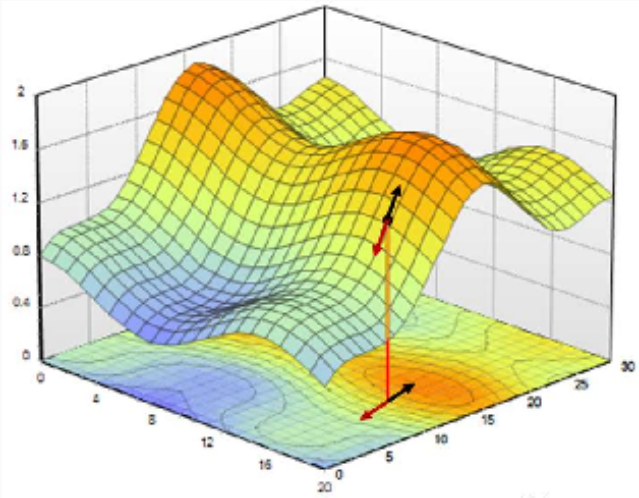
- Provided enough "training" samples we the distance can be approximated as:

$$Cost(\mathbf{W}) = \frac{1}{N}\sum_{i} distance\big[f(X_i, \mathbf{W}), d(X_i)\big]$$

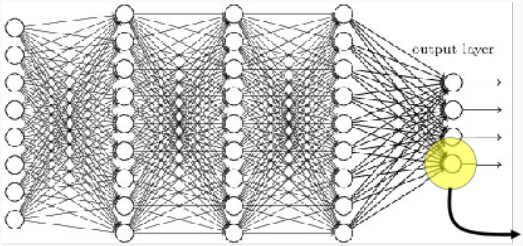
- The optimal weights $\tilde{\mathbf{W}}$ can be estimated by minimizing the cost:

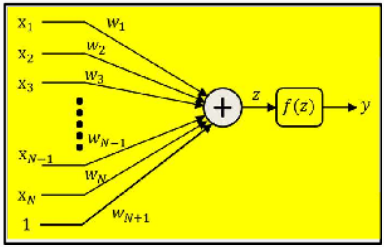
$$\tilde{\mathbf{W}} = argmin_{\mathbf{W}} Cost(\mathbf{W})$$

Finding the Minimum by Gradient Descend

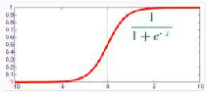


Activation Functions for Multilayer Perceptrons



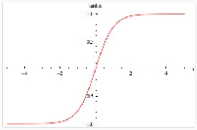


Activation Functions for Multilayer Perceptrons



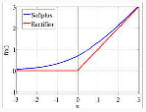
$$f(z) = \frac{1}{1 + \exp(-z)}$$

$$f'(z) = f(z)(1 - f(z))$$



$$f(z) = \tanh(z)$$

$$f'(z) = (1 - f\ \ (z))$$



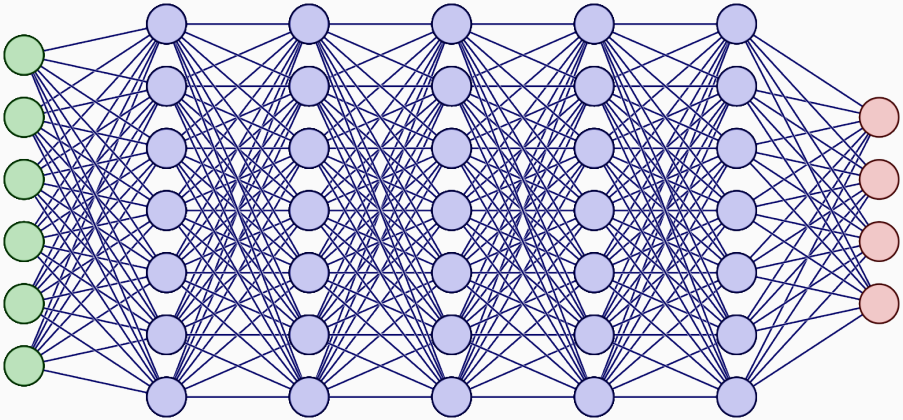
$$f(z) = \begin{cases} z, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

$$f'(z) = \begin{cases} 1, z \geq 0 \\ 0, z < 0 \end{cases}$$

$$f(z) = \log(1 + \exp(z))$$

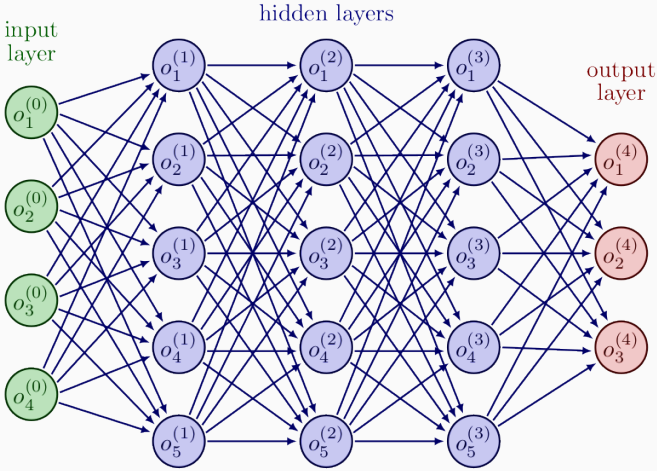
$$f'(z) = \frac{1}{1 + \exp(-z)}$$

Fully-connected Feedforward Networks

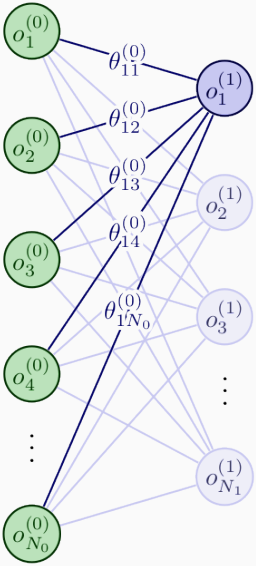


The Backpropagation Algorithm: forward pass

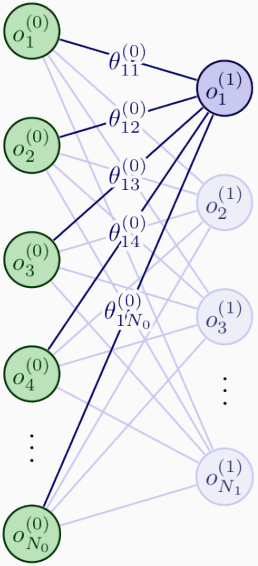
We have a fully connected network and we want to determine how the inputs influences the output



The Backpropagation Algorithm: forward pass



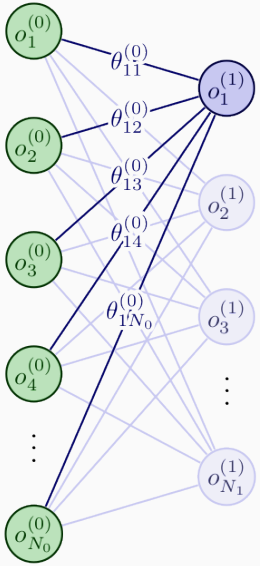
The Backpropagation Algorithm: forward pass



$$= g\left(\theta_{11}^{(0)} o_1^{(0)} + \theta_{12}^{(0)} o_2^{(0)} + \ldots + \theta_{1N_0}^{(0)} o_{N_0}^{(0)}\right)$$

$$= g\left(\sum_{i=1}^{N_0} \theta_{1i}^{(0)} o_i^{(0)}\right) = g\left(\boldsymbol{\theta}_1^{(0)} \boldsymbol{o}^{(0)}\right)$$

The Backpropagation Algorithm: forward pass



$$= g\left(\theta_{11}^{(0)} o_1^{(0)} + \theta_{12}^{(0)} o_2^{(0)} + \ldots + \theta_{1N_0}^{(0)} o_{N_0}^{(0)}\right)$$

$$= g\left(\sum_{i=1}^{N_0} \theta_{1i}^{(0)} o_i^{(0)}\right) = g\left(\boldsymbol{\theta}_1^{(0)} \boldsymbol{o}^{(0)}\right)$$

$$\begin{pmatrix} o_1^{(1)} \\ o_2^{(1)} \\ \vdots \\ o_{N_1}^{(1)} \end{pmatrix} = g\left[\begin{pmatrix} \theta_{10}^{(0)} & \theta_{11}^{(0)} & \ldots & \theta_{1N_0}^{(0)} \\ \theta_{20}^{(0)} & \theta_{21}^{(0)} & \ldots & \theta_{2N_0}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{N_1 1}^{(0)} & \theta_{N_1 2}^{(0)} & \ldots & \theta_{N_1 N_0}^{(0)} \end{pmatrix} \begin{pmatrix} o_1^{(0)} \\ o_2^{(0)} \\ \vdots \\ o_{N_0}^{(0)} \end{pmatrix}\right]$$

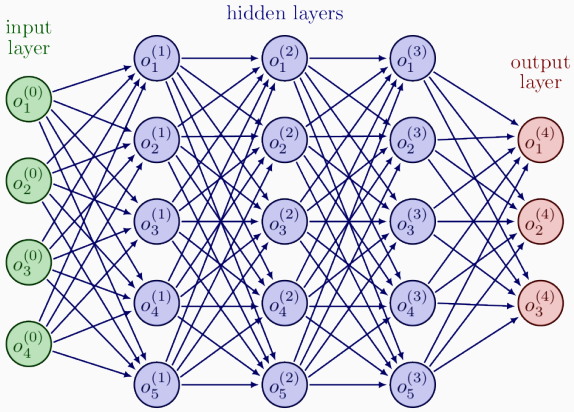
$$\boldsymbol{p}^{(1)} = \boldsymbol{\Theta}^{(0)} \boldsymbol{o}^{(0)} \quad \textit{and} \quad \boldsymbol{o}^{(1)} = g\left(\boldsymbol{p}^{(1)}\right)$$

The Backpropagation Algorithm: forward pass

- We denote by $\Theta^{(l-1)}$ the matrix of weights connecting layers $l-1$ and l
- We denote by g the nonlinear activation function
- The forward pass can be summarized as follow:

$$\boldsymbol{p}^{(l)} = \Theta^{(l-1)}\boldsymbol{o}^{(l-1)}$$

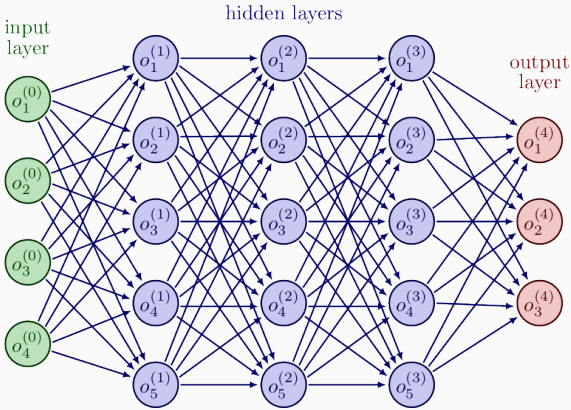
$$\boldsymbol{o}^{(i)} = g(\boldsymbol{p}^{(l)})$$



The Backpropagation Algorithm: forward pass

For a simple neural network with 3 hidden layer we have that:

$$\boldsymbol{o}^{(0)} = \boldsymbol{x}$$
$$\boldsymbol{p}^{(1)} = \Theta^{(0)}\boldsymbol{o}^{(0)}$$
$$\boldsymbol{o}^{(1)} = g(\boldsymbol{p}^{(1)})$$
$$\boldsymbol{p}^{(2)} = \Theta^{(1)}\boldsymbol{o}^{(1)}$$
$$\boldsymbol{o}^{(2)} = g(\boldsymbol{p}^{(2)})$$
$$\boldsymbol{p}^{(3)} = \Theta^{(2)}\boldsymbol{o}^{(2)}$$
$$\boldsymbol{o}^{(3)} = g(\boldsymbol{p}^{(3)})$$
$$\boldsymbol{p}^{(4)} = \Theta^{(3)}\boldsymbol{o}^{(3)}$$
$$\boldsymbol{o}^{(4)} = g(\boldsymbol{p}^{(4)}) = h_\Theta(\boldsymbol{x})$$



The Backpropagation Algorithm: backward pass

- We want to compute how each weight affect the discrepancy between the desired and the actual output of the network (respectively $\boldsymbol{y}$ and $h_\Theta(\boldsymbol{x})$)

The Backpropagation Algorithm: backward pass

- We want to compute how each weight affect the discrepancy between the desired and the actual output of the network (respectively $\boldsymbol{y}$ and $h_\Theta(\boldsymbol{x})$)
- We define a cost function:

$$C(h_\Theta(\boldsymbol{x}), \boldsymbol{y})$$

(MSE if linear regression, KL divergence if classification problem)

The Backpropagation Algorithm: backward pass

- We want to compute how each weight affect the discrepancy between the desired and the actual output of the network (respectively $\boldsymbol{y}$ and $h_{\Theta}(\boldsymbol{x})$)
- We define a cost function:

$$C(h_{\Theta}(\boldsymbol{x}), \boldsymbol{y})$$

  (MSE if linear regression, KL divergence if classification problem)
- Our goal is to find the set of weights $\boldsymbol{\Theta}^{(0)}, \cdots, \boldsymbol{\Theta}^{(L)}$ that minimizes the cost function

The Backpropagation Algorithm: backward pass

- We want to compute how each weight affect the discrepancy between the desired and the actual output of the network (respectively $\boldsymbol{y}$ and $h_{\Theta}(\boldsymbol{x})$)
- We define a cost function:

$$C(h_{\Theta}(\boldsymbol{x}), \boldsymbol{y})$$

  (MSE if linear regression, KL divergence if classification problem)
- Our goal is to find the set of weights $\boldsymbol{\Theta}^{(0)}, \cdots, \boldsymbol{\Theta}^{(L)}$ that minimizes the cost function
- Starting from the last layer we compute derivatives to "backpropagate" the error

The Backpropagation Algorithm: backward pass

We assume MSE cost function and apply derivative starting from the last layer L:

$$\frac{\partial C}{\partial o_i^{(L)}} = (h_\Theta(\boldsymbol{x}) - \boldsymbol{y})_i$$

$$\frac{\partial C}{\partial p_i^{(L)}} = g'(p_i^{(L)}) \frac{\partial C}{\partial o_i^{(L)}}$$

$$\frac{\partial C}{\partial \Theta_{ji}^{(L-1)}} = o_i^{(L-1)} \frac{\partial C}{\partial p_j^{(L)}}$$

$$\frac{\partial C}{\partial o_i^{(L-1)}} = \sum_j \Theta_{ji} \frac{\partial C}{\partial p_j^{(L)}}$$

$$\frac{\partial C}{\partial p_i^{(L-1)}} = g'(p_i^{(L-1)}) \frac{\partial C}{\partial o_i^{(L-1)}}$$

$$\frac{\partial C}{\partial \Theta_{ji}^{(L-2)}} = o_i^{(L-2)} \frac{\partial C}{\partial p_j^{(L-1)}}$$

The Backpropagation Algorithm: backward pass

For each layer l we define the vectors $\boldsymbol{\delta}^{(l)}$ as :

$$\delta_i^{(l)} = \frac{\partial C}{\partial p_i^{(l)}}$$

Assuming MSE as cost (or error) the backward pass can be summarized as follows:

- Error: $E = 1/2\Big(h_\Theta(\boldsymbol{x}) - \boldsymbol{y}\Big)^2$
- Vectors $\boldsymbol{\delta}$'s :

$$\delta^{(L)} = \Big(h_\theta - y\Big) g'(p^{(L)})$$
$$\delta^{(l)} = [\boldsymbol{\Theta}^{(l)}]^T \boldsymbol{\delta}^{(l+1)} .* g'(p^{(l)})$$

- Gradient of the error:

$$\frac{\partial E}{\partial \Theta_{ij}^{(l)}} = \delta_i^{(l+1)} o_j^{(l)}$$

About gradient descent: batch vs mini-batch vs stochastic

Methods differ in the data points used for each update of the parameters.

Example for linear regression:

- Batch gradient descent : (all data)

$$\theta_j := \theta_j - \alpha \frac{1}{N} \sum_{i=1}^{N} \left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

- Mini-batch gradient descent : (random subset of m data points)

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

- Stochastic gradient descent : (a single data point i, repeat $\forall\, N$)

$$\theta_j := \theta_j - \alpha \left(h_\theta(\boldsymbol{x}^{(i)}) - y^{(i)}\right) x_j^{(i)}$$