POLITECNICO
MILANO 1863

Artificial Neural Networks and Deep Learning

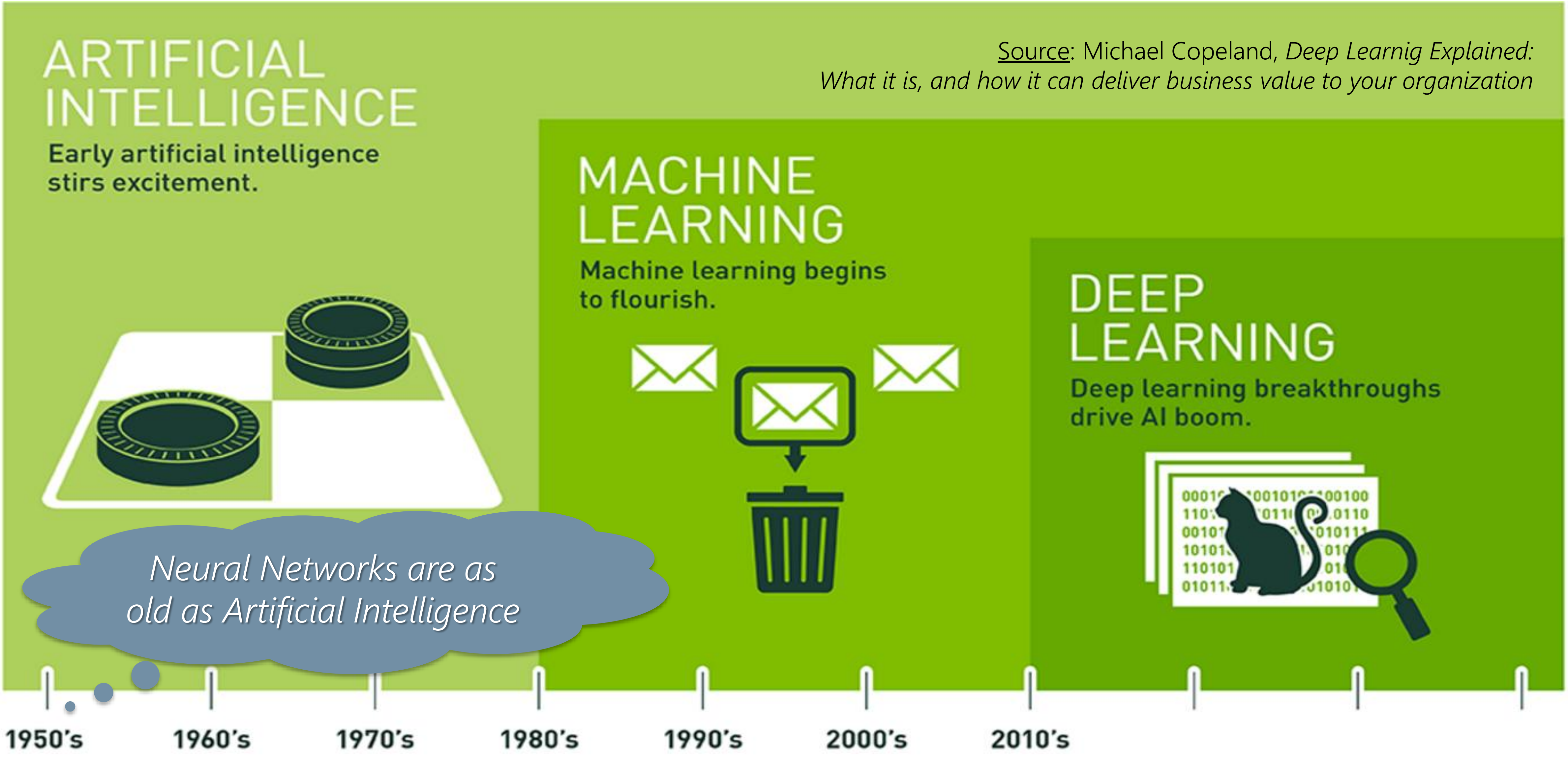
- From Perceptrons to Feed Forward Neural Networks -

Matteo Matteucci, PhD (matteo.matteucci@polimi.it)
Artificial Intelligence and Robotics Laboratory
Politecnico di Milano

AIRLAB
ARTIFICIAL INTELLIGENCE AND ROBOTICS LAB

«Deep Learning is not AI, nor Machine Learning»

<u>Source</u>: Michael Copeland, *Deep Learnig Explained: What it is, and how it can deliver business value to your organization*



Neural Networks are as old as Artificial Intelligence

The inception of AI

A PROPOSAL FOR THE

DARTMOUTH SUMMER RESEARCH PROJECT

ON ARTIFICIAL INTELLIGENCE

J. McCarthy, Dartmouth College
M. L. Minsky, Harvard University
N. Rochester, I. B. M. Corp
C. E. Shannon, Bell Telepho

August 31, 1955

A Proposal for the

DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLI

We propose that a 2 month, 10 man study of artificial intelligence carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

The following are some aspects of the artificial intelligence problem:

1) Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. The speeds and memory capacities of present computers may be insufficient to simulate many of the higher functions of the human brain, but the major obstacle is not lack of machine capacity, but our inability to write programs taking full advantage of what we have.

3. Neuron Nets

How can a set of (hypothetical) neurons be arranged so as to form concepts. Considerable theoretical and experimental work has been done on this problem by Uttley, Rashevsky and his group, Farley and Clark, Pitts and McCulloch, Minsky, Rochester and Holland, and others. Partial results have been obtained but the problem needs more theoretical work.

5) Self-Improvement

Probably a truly intelligent machine will carry out activities which may best be described as self-improvement. Some schemes for doing this have been proposed and are worth further study. It seems likely that this question can be studied abstractly as well.

6) Abstractions

A number of types of "abstraction" can be distinctly defined and several others less distinctly. A direct attempt to classify these and to describe machine methods of forming abstractions from sensory and other data would seem worthwhile.

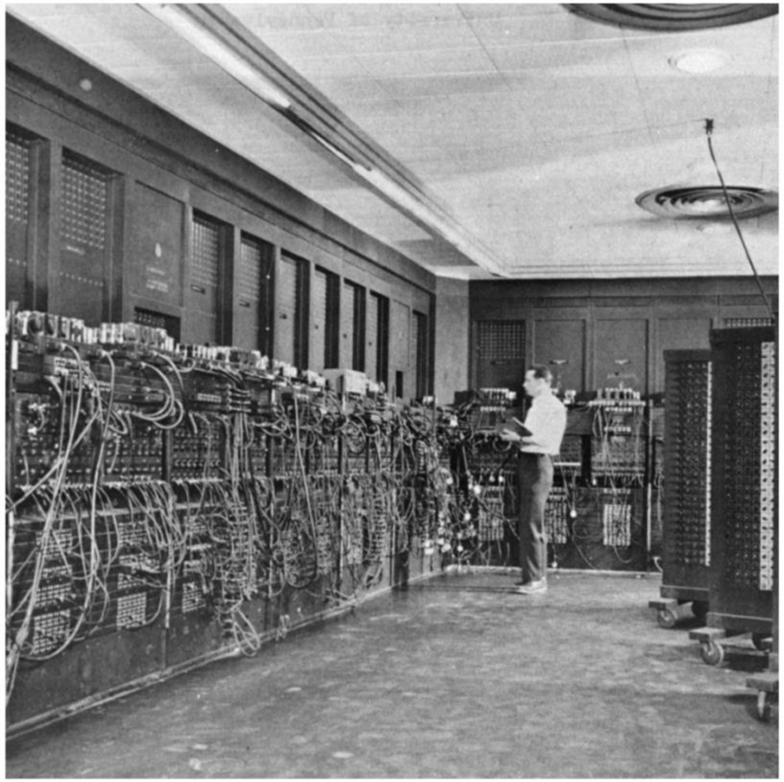
Let's go back to 1940s …

Computers were already good at

- Doing precisely what the programmer programs them to do
- Doing arithmetic very fast

However, they would have liked them to:

- Interact with noisy data or directly with the environment
- Be massively parallel and fault tolerant
- Adapt to circumstances

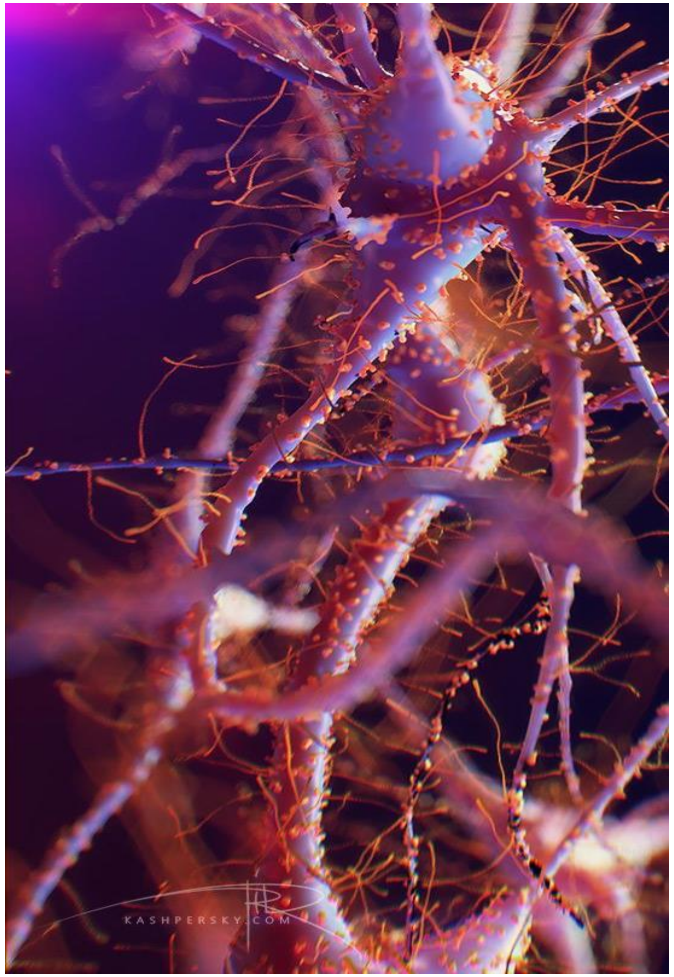


Researchers were seeking a computational model beyond the <u>*Von Neumann Machine*</u>!

The Brain Computationa Model

The human brain has a huge number of computing units:

- 10^{11} (one hundred billion) neurons
- 7,000 synaptic connections to other neurons
- In total from 10^{14} to 5×10^{14} (100 to 500 trillion) in adults to 10^{15} synapses (1 quadrillion) in a three year old child

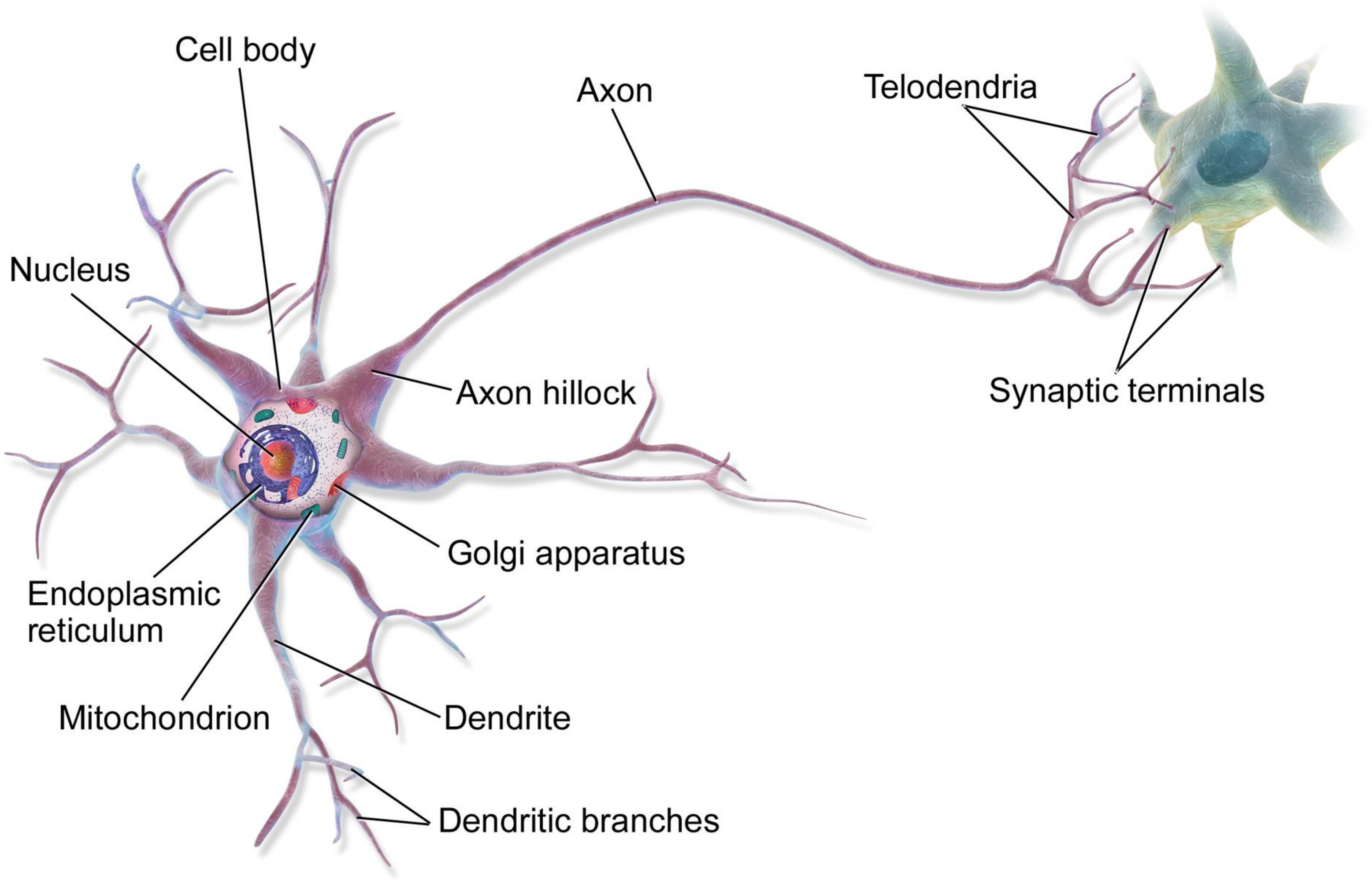


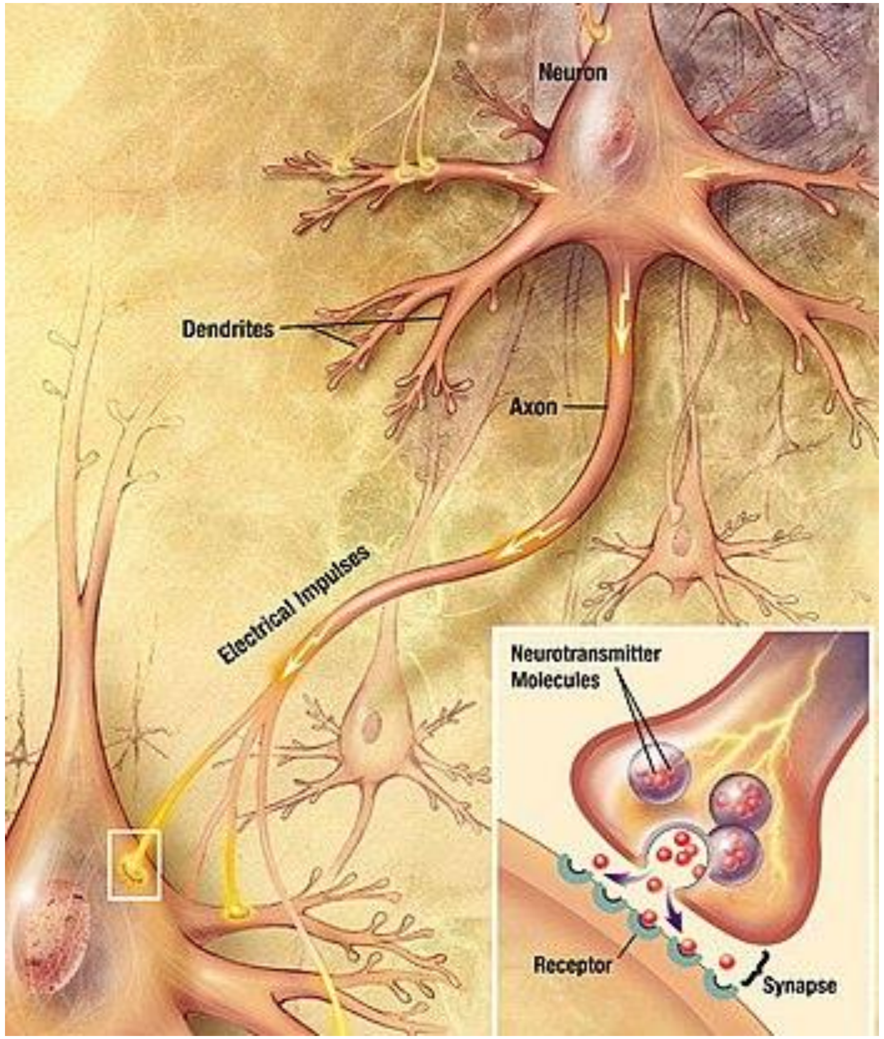
The computational model of the brain is:

- Distributed among simple non linear units
- Redundant and thus fault tolerant
- Intrinsically parallel

Perceptron: a computational model based on the brain!

Computation in Biological Neurons

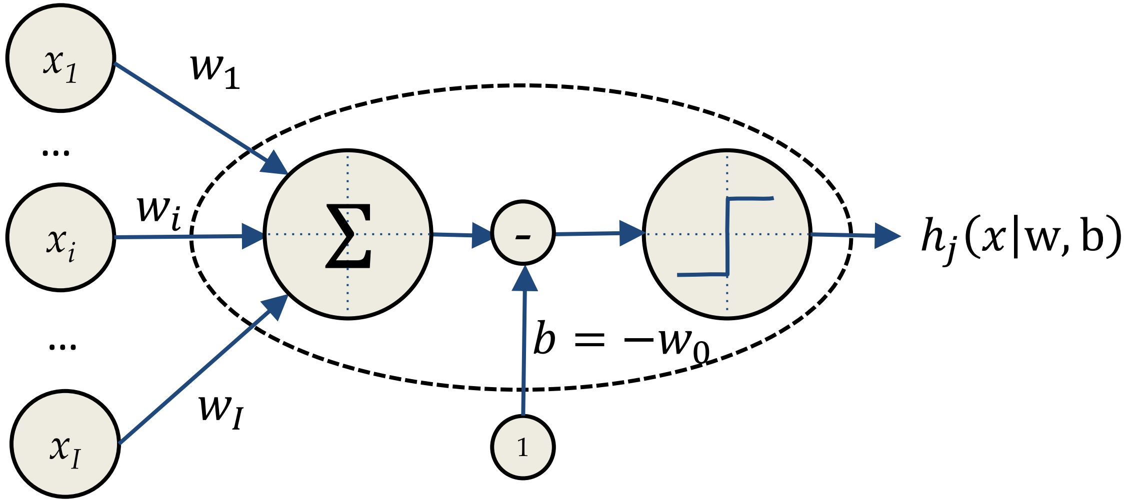




Computation in Artificial Neurons

Information is transmitted through chemical mechanisms:

- Dendrites collect charges from synapses, both Inhibitory and Excitatory
- Cumulates charge is released (neuron fires) once a Threshold is passed

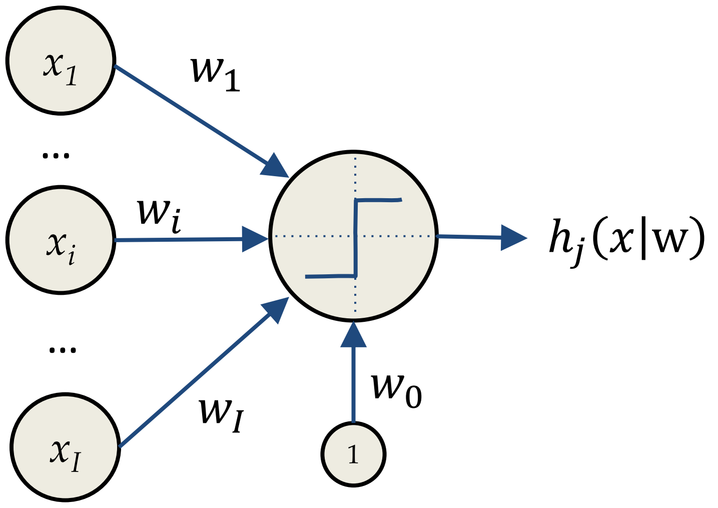


$$h_j(x|\mathrm{w},\mathrm{b}) = \mathrm{h_j}\left(\Sigma_{i=1}^{I} w_i \cdot x_i - b\right) = \mathrm{h_j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = h_j(w^T x)$$

Computation in Artificial Neurons

Information is transmitted through chemical mechanisms:

- Dendrites collect charges from synapses, both Inhibitory and Excitatory
- Cumulates charge is released (neuron fires) once a Threshold is passed

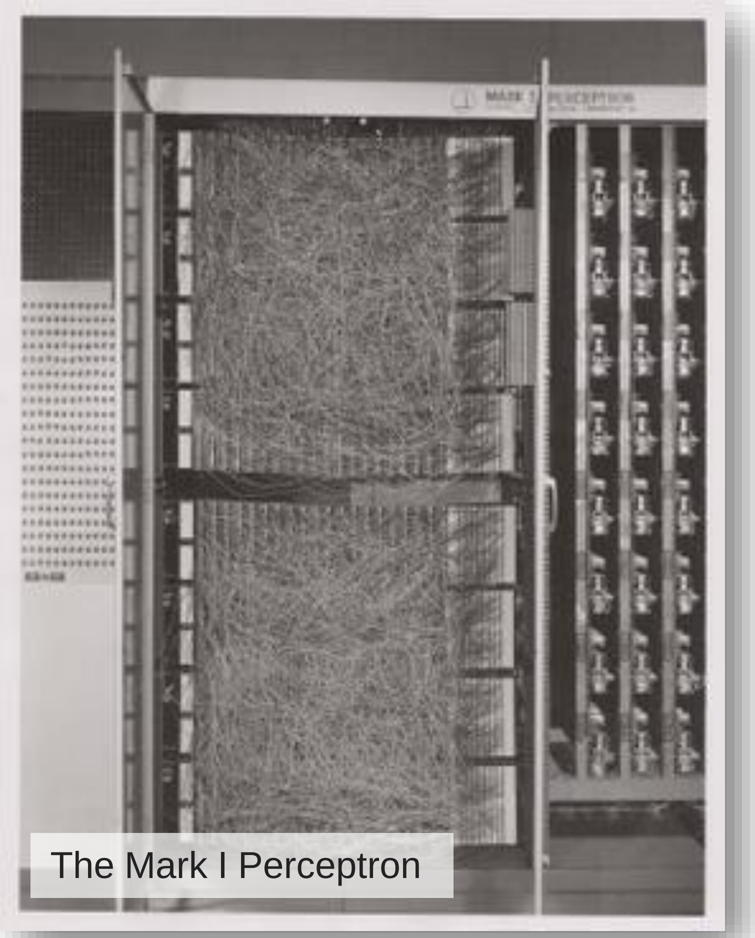


$$h_j(x|\mathrm{w},\mathrm{b}) = \mathrm{h_j}\left(\Sigma_{i=1}^{I} w_i \cdot x_i - b\right) = \mathrm{h_j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = h_j(w^T x)$$

Who did it first?

Several researchers were investigating models for the brain

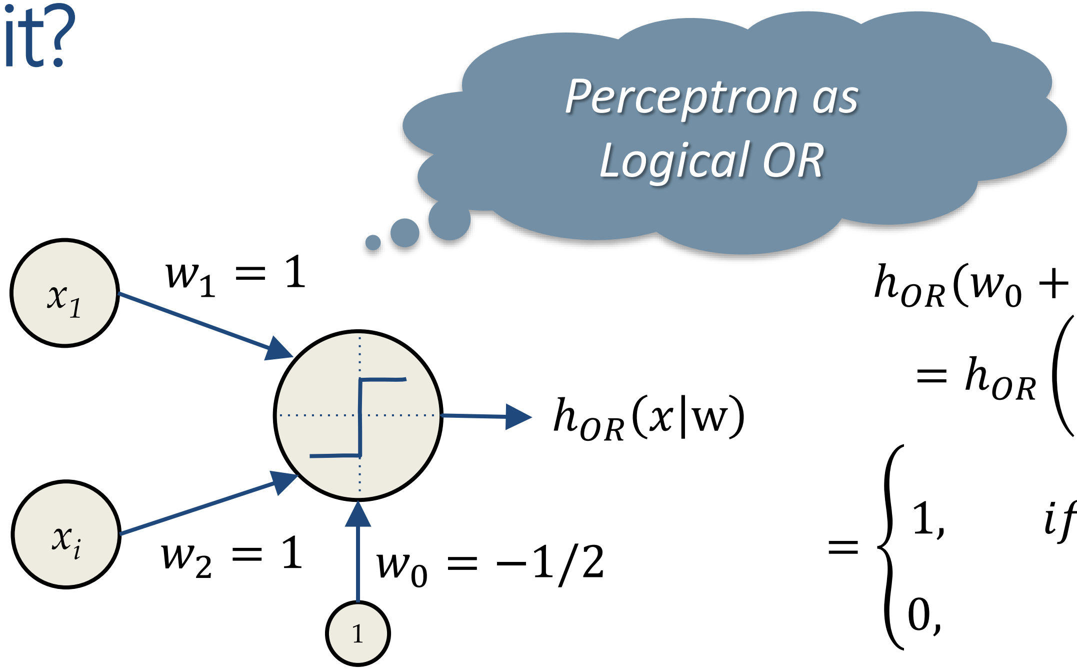
- In 1943, Warren McCullog and Walter Harry Pitts proposed the Treshold Logic Unit or Linear Unit, the activation function was a threshold unit equivalent to the Heaviside step function
- In 1957, Frank Rosemblatt developed the first Perceptron. Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors
- In 1960, Bernard Widrow introduced the idea of representing the threshold value as a bias term in the ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element)



The Mark I Perceptron

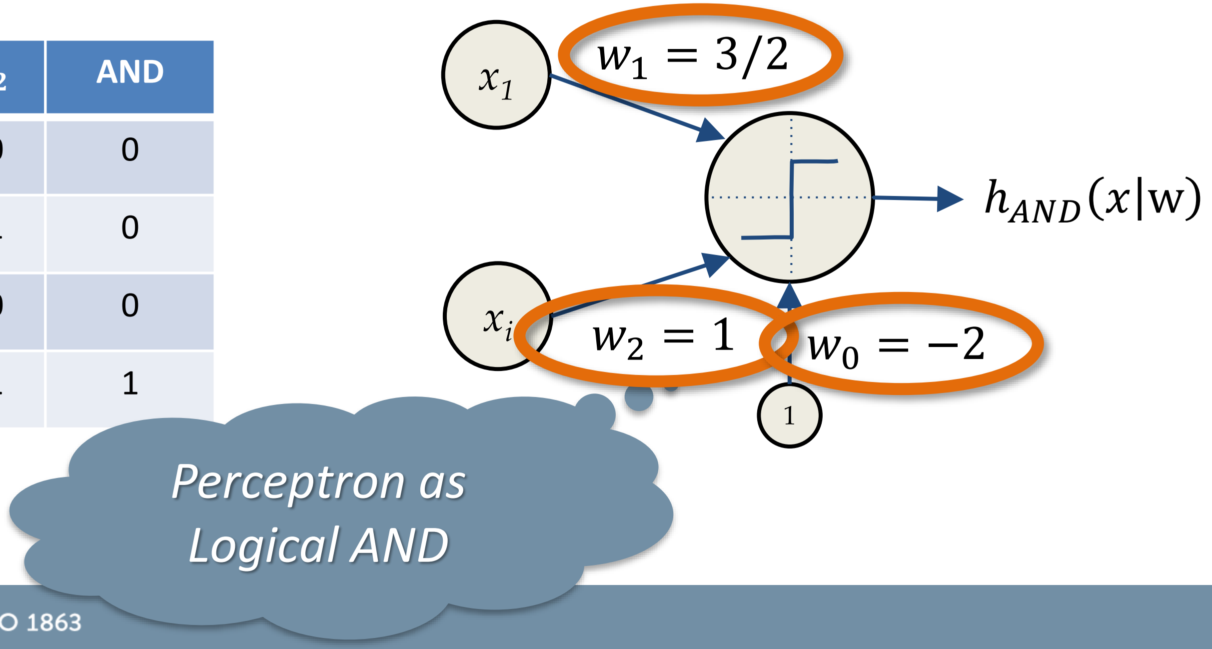
What can you do with it?

x_0	x_1	x_2	OR
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$$h_{OR}(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2) = \\ = h_{OR}\left(-\frac{1}{2} + x_1 + x_2\right) = \\ = \begin{cases} 1, & if \left(-\frac{1}{2} + x_1 + x_2\right) > 0 \\ 0, & otherwise \end{cases}$$

x_0	x_1	x_2	AND
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



$$h_{AND}(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2) = \\ = h_{AND}\left(-2 + \frac{3}{2}\,x_1 + x_2\right) = \\ = \begin{cases} 1, & if \left(-2 + \frac{3}{2}x_1 + x_2\right) > 0 \\ 0, & otherwise \end{cases}$$

Hebbian Learning

"The strength of a synapse increases according to the simultaneous activation of the relative input and the desired target"

(Donald Hebb, The Organization of Behavior, 1949)

Hebbian learning can be summarized by the follow

$$w_i^{k+1} = w_i^k + \Delta w_i^k$$

$$\Delta w_i^k = \eta \cdot x_i^k \cdot t^k$$

Start from a random initialization

*Fix the weights one sample at the time (**online**), and only if the sample is not correctly predicted*

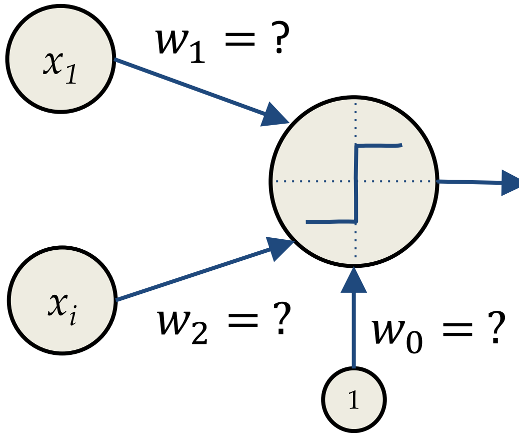
Where we have:

- η: learning rate
- x_i^k: the i^{th} perceptron input at time k
- t^k: the desired output at time k

Perceptron Example

Learn the weights to implement the OR operator

- Start from random weights, e.g.,
  $$w = [0\ \ 0\ \ 0]$$
- Chose a learning rate, e.g.,
  $$\eta = 0.5$$
- Cycle through the records by fixing those which are not correct
- End once all the records are correctly predicted



$$h(w^T x) = \begin{cases} 1\ if w^T x > 0 \\ 0\ if w^T x = 0 \\ -1\ if w^T x < 0 \end{cases}$$

x_0	x_1	x_2	OR
1	-1	-1	-1
1	-1	1	1
1	1	-1	1
1	1	1	1

Does the procedure always converge?

Does it always converge to the same sets of weights?

Perceptron Math

A perceptron computes a weighted sum, returns its Sign (Thresholding)

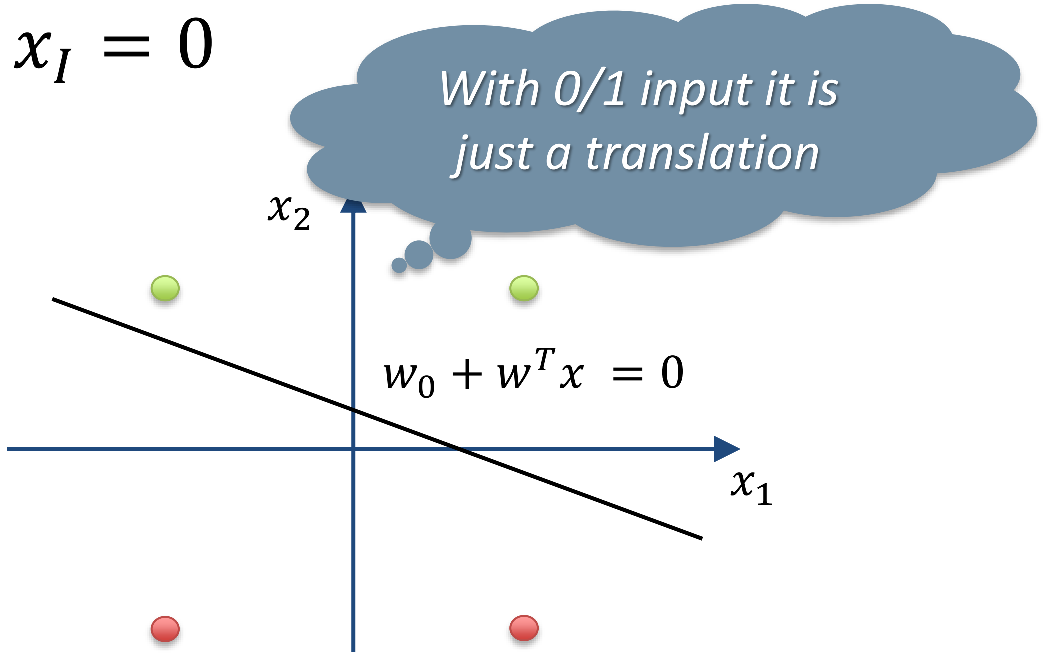
$$h_j(x|\mathrm{w}) = \mathrm{h_j}\left(\Sigma_{i=0}^{I} w_i \cdot x_i\right) = Sign(w_0 + w_1 \cdot x_1 + \cdots + w_I \cdot x_I)$$

It is a linear classifier for which the decision boundary is the hyperplane

$$w_0 + w_1 \cdot x_1 + \cdots + w_I \cdot x_I = 0$$

In 2D, this turns into

$$w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = 0$$
$$w_2 \cdot x_2 = -w_0 - w_1 \cdot x_1$$
$$x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} \cdot x_1$$



Boolean Operators are Linear Boundaries

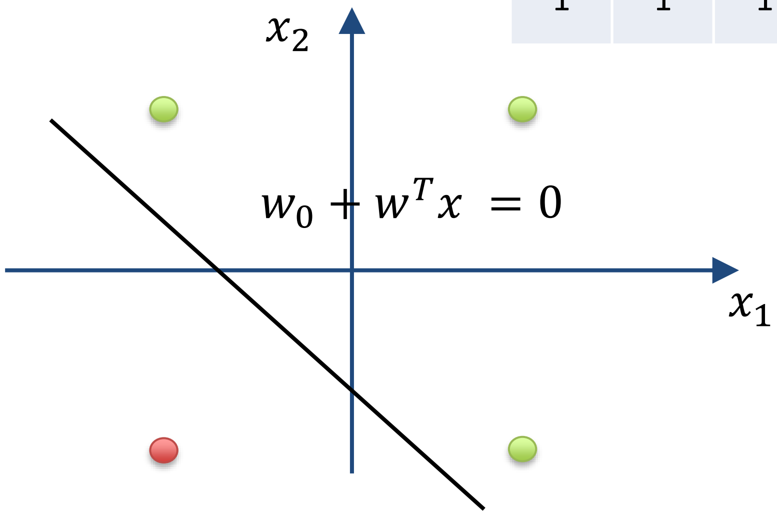
What's about it? We had already Boolean operators

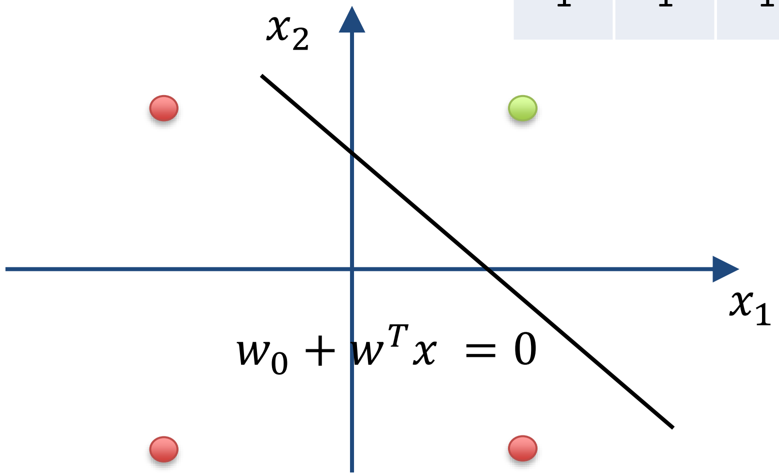
Linear boundary explains how Perceptron implements Boolean operators

x_0	x_1	x_2	OR
1	-1	-1	-1
1	-1	1	1
1	1	-1	1
1	1	1	1

Yes! But this ia a single trainable HW!

x_0	x_1	x_2	AND
1	-1	-1	-1
1	-1	1	-1
1	1	-1	-1
1	1	1	1

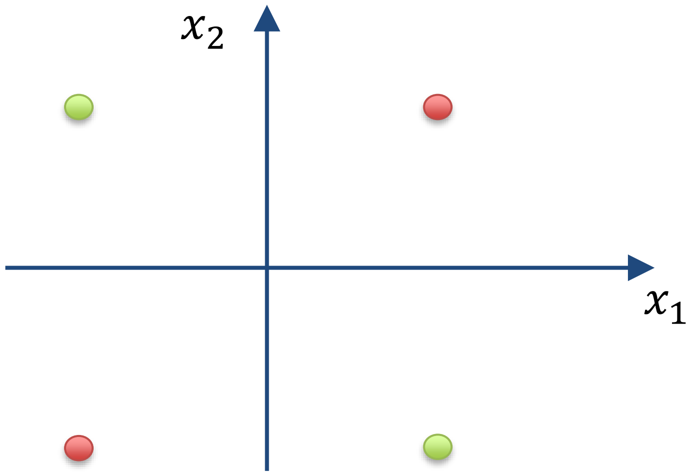




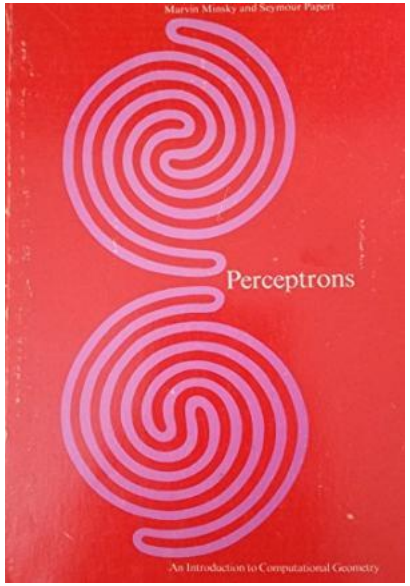
What can't you do with it?

What if the dataset we want to learn does not have a linear separation boundary?

x_0	x_1	x_2	XOR
1	-1	-1	-1
1	-1	1	1
1	1	-1	1
1	1	1	-1



Marvin Minsky, Seymour Papert
"*Perceptrons: an introduction to computational geometry*" 1969.

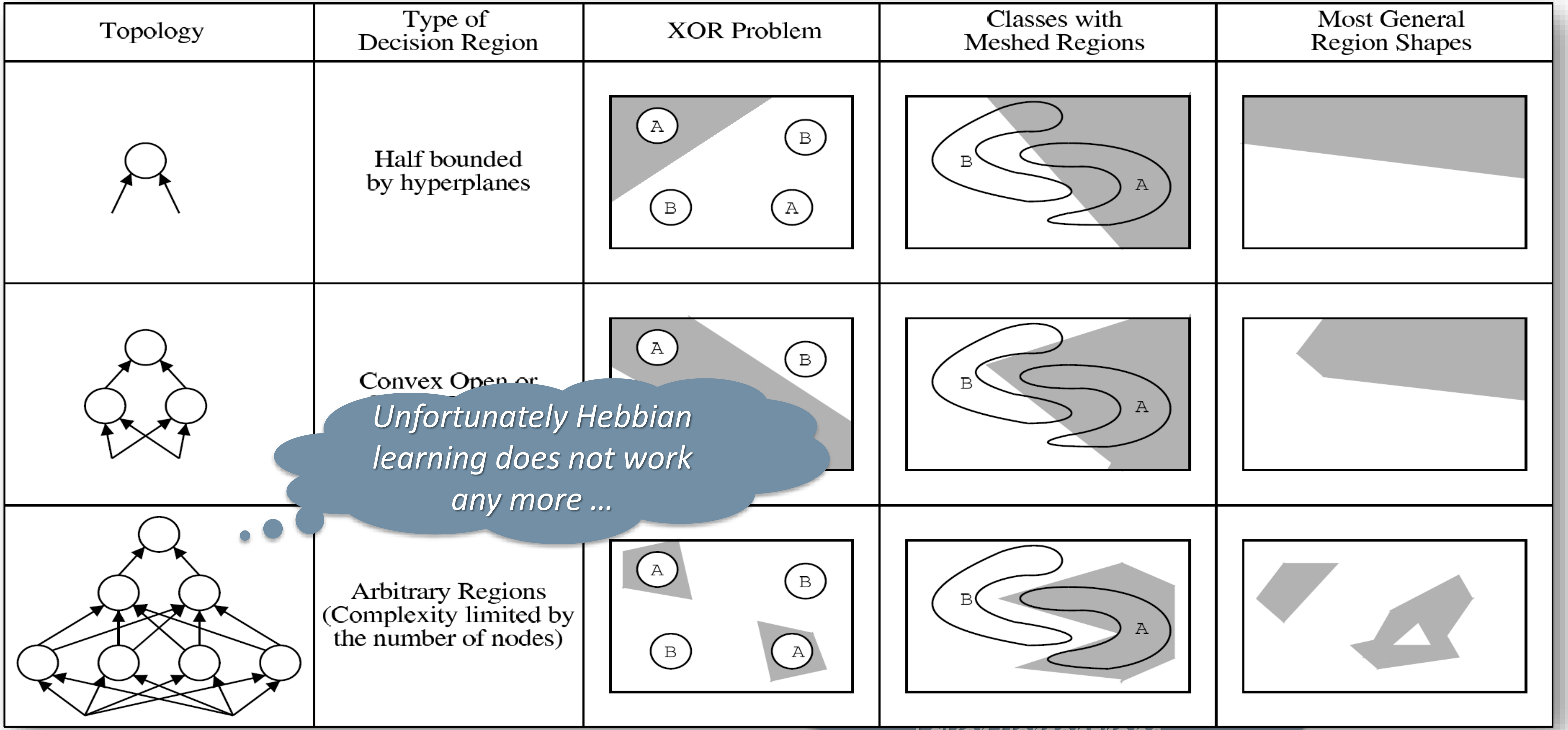


The Perceptron does not work any more and we need alternative solutions

- Non linear boundary
- Alternative input representations

The idea behind Multi Layer Perceptrons

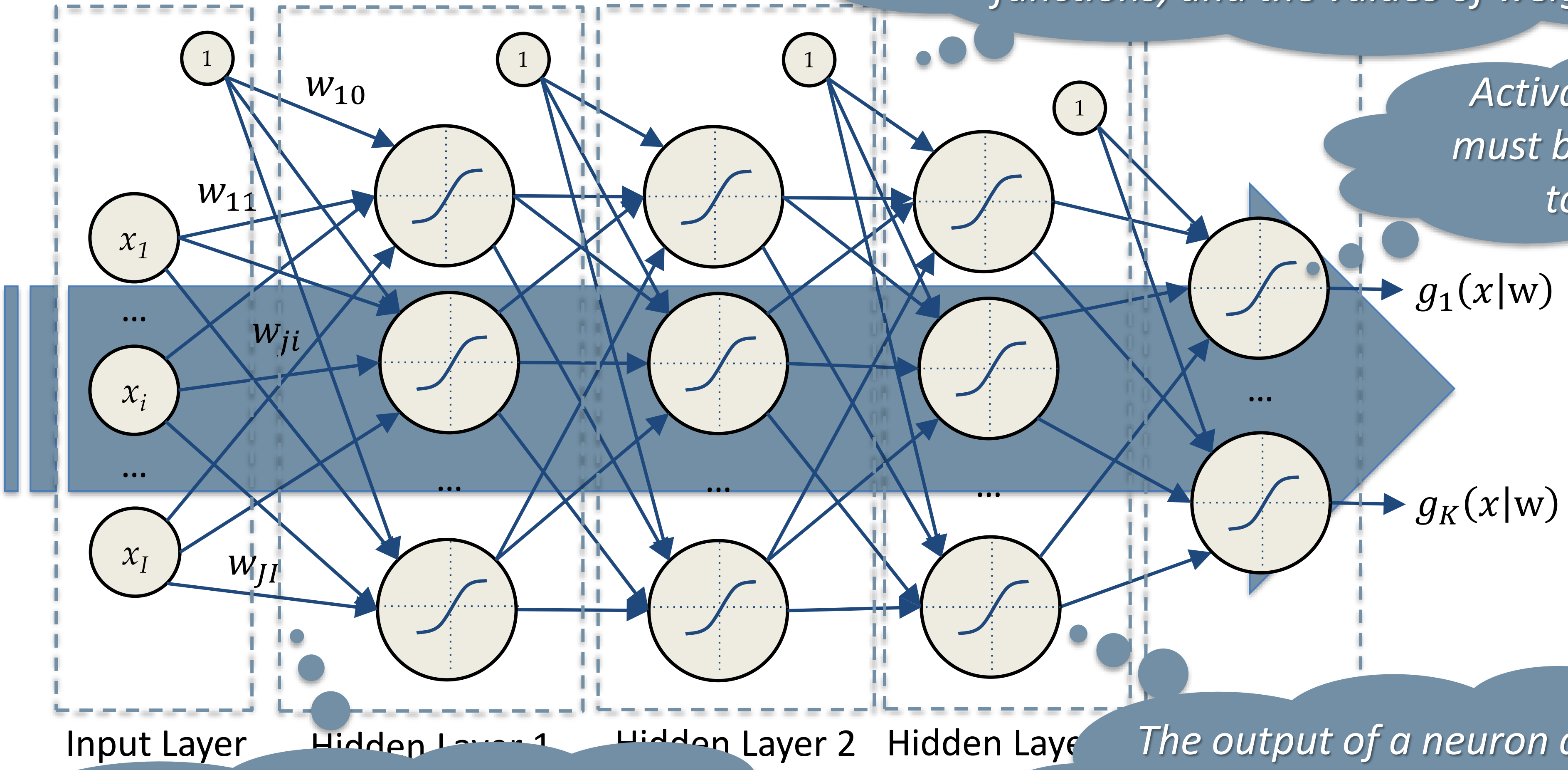
What can't you do with it?



Unfortunately Hebbian learning does not work any more ...

Layer Perceptrons

Feed Forward Neural Networks



Non-linear model characterized by the number of neurons, activation functions, and the values of weights.

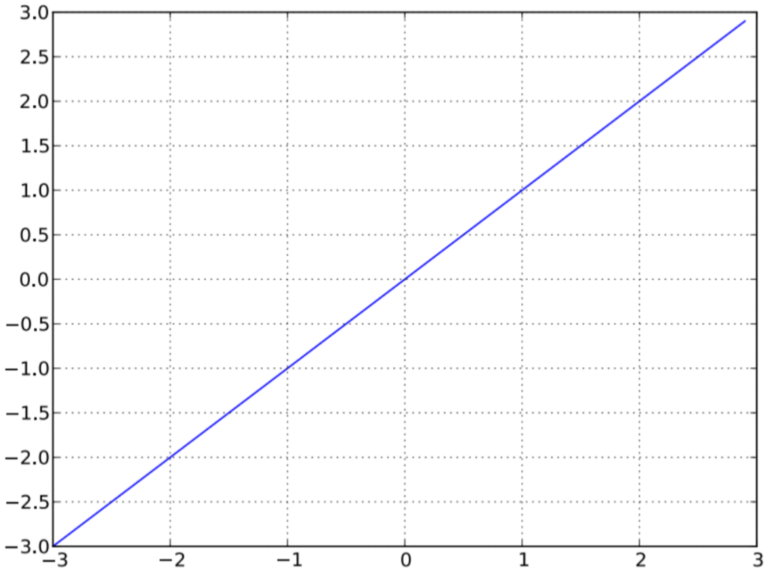
Activation functions must be differentiable to train it …

Layers are connected through weights $W^{(l)} = \left\{ w_{ji}^{(l)} \right\}$

The output of a neuron depends only on the previous layers

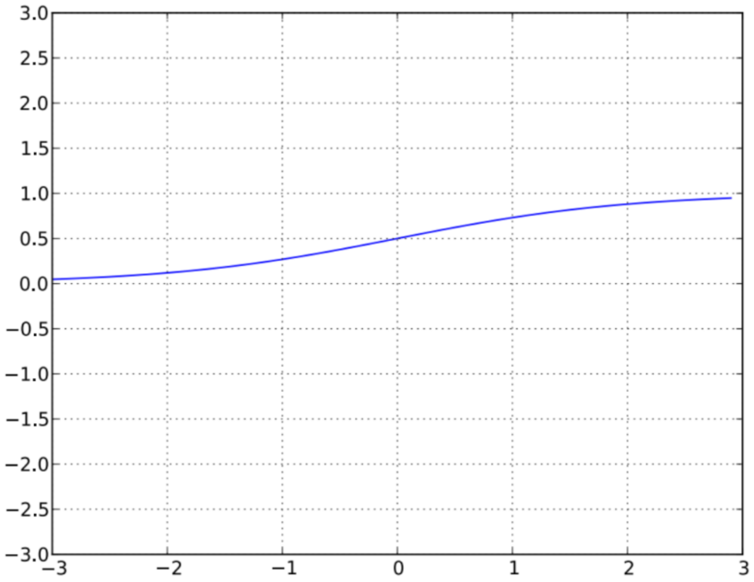
$$h^{(l)} = \left\{ h_j^{(l)}\left(h^{(l-1)}, W^{(l)}\right) \right\}$$

Which Activation Function?



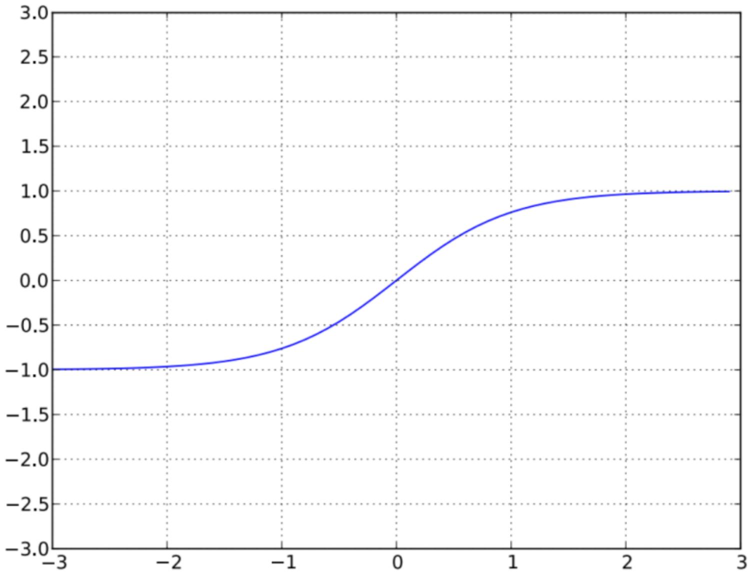
Linear activation function

$$g(a) = a$$
$$g'(a) = 1$$



Sigmoid activation function

$$g(a) = \frac{1}{1 + \exp(-a)}$$
$$g'(a) = g(a)(1 - g(a))$$



Tanh activation function

$$g(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$
$$g'(a) = 1 - g(a)^2$$

Output Layer in Regression and Classification

For all hidden neurons use sigmoid or tanh (see later)

In Regression the output spans the whole $\Re$ domain:

- Use a Linear activation function for the output neuron

In Classification with two classes, chose according to their coding:

- Two classes $\{\Omega_0 = -1, \Omega_1 = +1\}$ then use Tanh output activation
- Two classes $\{\Omega_0 = 0, \Omega_1 = 1\}$ then use Sigmoid output activation (it can be interpreted as class posterior probability)

«One hot» encoding

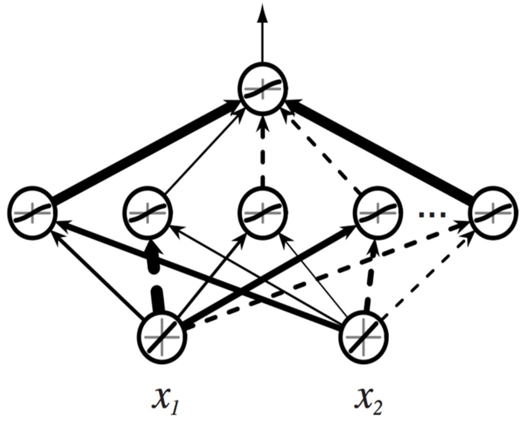
When dealing with multiple classes (K) use as many neuron as classes

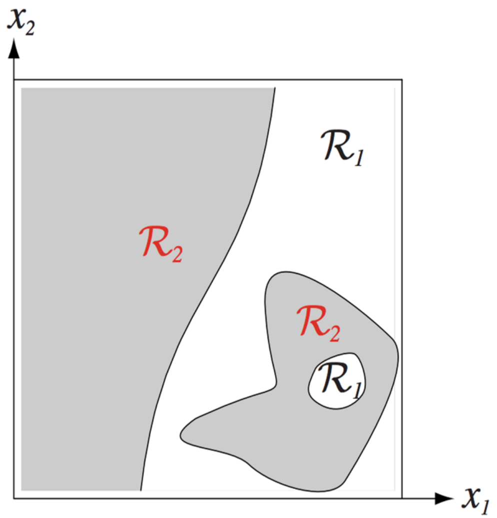
- Classes are coded as $\{\Omega_0 = [0\ 0\ 1], \Omega_1 = [0\ 1\ 0], \Omega_2 = [1\ 0\ 0]\}$
- Output neurons use a softmax unit $\mathrm{y_k} = \frac{\exp(z_k)}{\sum_k \exp(z_k)} = \frac{\exp\left(\sum_j w_{kj} h_j\left(\sum_i^I w_{ji} \cdot x_i\right)\right)}{\sum_{k=1}^K \exp\left(\sum_j w_{kj} h_j\left(\sum_i^I w_{ji} \cdot x_i\right)\right)}$

Neural Networks are Universal Approximators

"A single hidden layer feedforward neural network with S shaped activation functions can approximate any measurable function to any desired degree of accuracy on a compact set "

Universal approximation theorem
(Kurt Hornik, 1991)





Images from Hugo Larochelle's DL Summer School Tutorial

Regardless the function we are learning, a single layer can represent it:

- Doesn't mean a learning algorithm can find the necessary weights!
- In the worse case, an exponential number of hidden units may be required
- The layer may have to be unfeasibly large and may fail to learn and generalize

Classification requires just one extra layer …

Optimization and Learning (Supervised learning)

Recall learning a ***parametric model*** $y(x_n|\theta)$ in regression/classification

- Given a training set

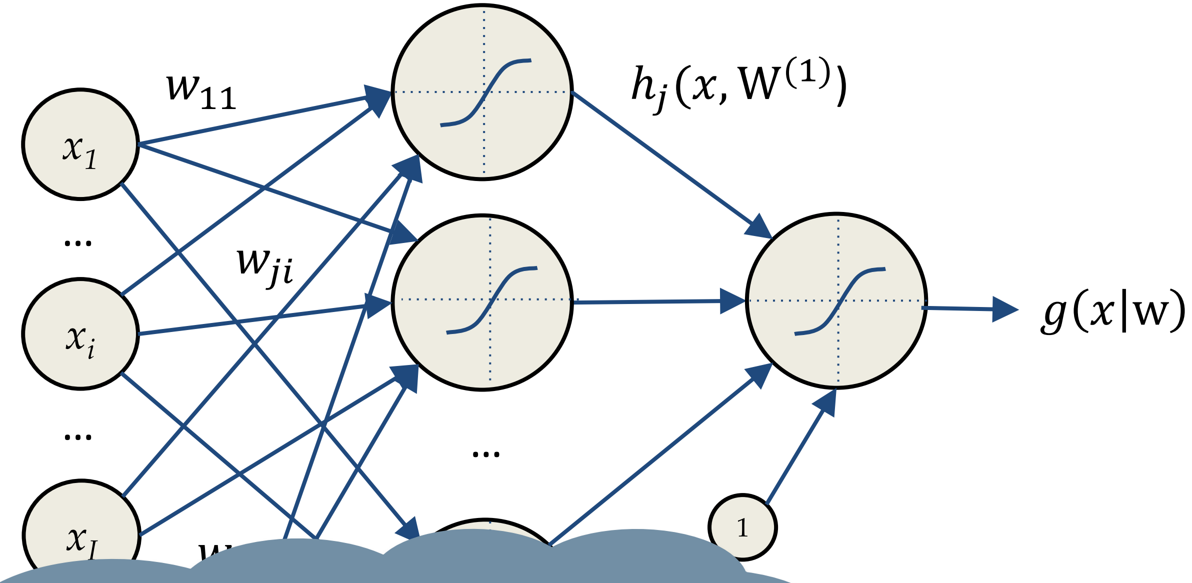
$$D =< x_1, t_1 > \cdots < x_N, t_N >$$

- We want to find model parameters such that for new data

$$y(x_n|\theta) \sim t_n$$

- In case of a Neural Network this can be rewritten as

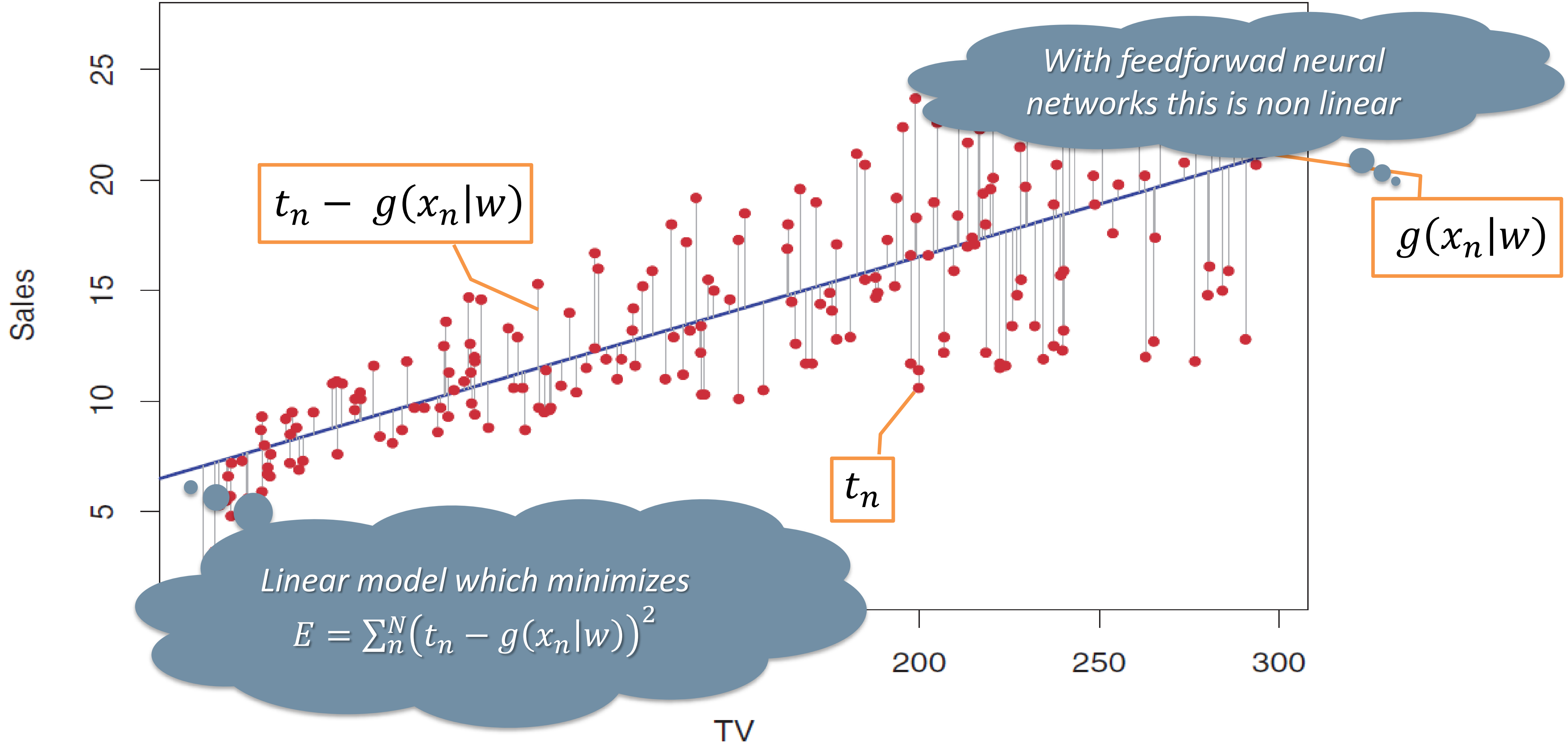
$$g(x_n|w) \sim t_n$$



For this you can minimize

$$E = \sum_n^N \left(t_n - g(x_n|w)\right)^2$$

Sum of Squared Errors



Non Linear Optimization 101

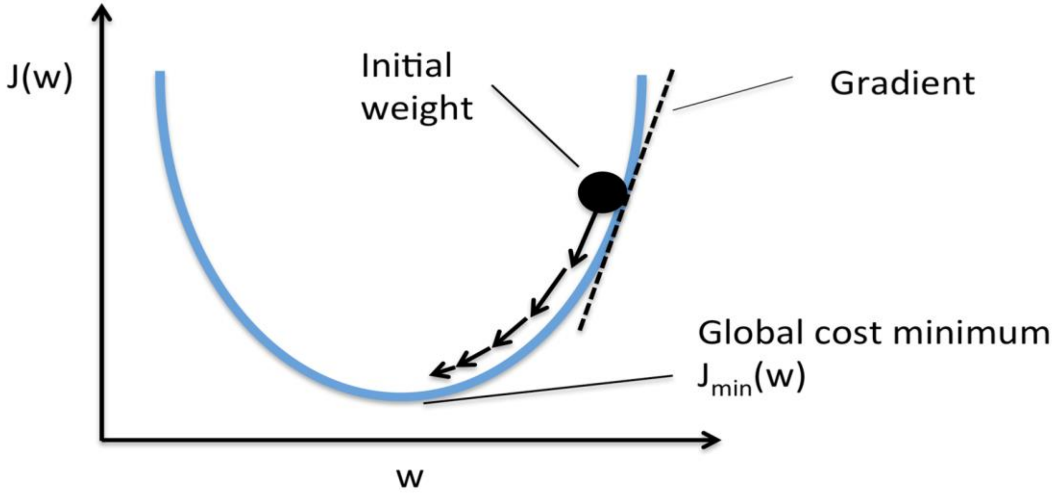
To find the minimum of a generic function, we compute the partial derivatives of the function and set them to zero

$$\frac{\partial J(w)}{\partial w} = 0$$

Closed-form solutions are practically never available so we can use iterative solutions (gradient descent):

- Initialize the weights to a random value
- Iterate until convergence

$$w^{k+1} = w^k - \eta \left. \frac{\partial J(w)}{\partial w} \right|_{w^k}$$

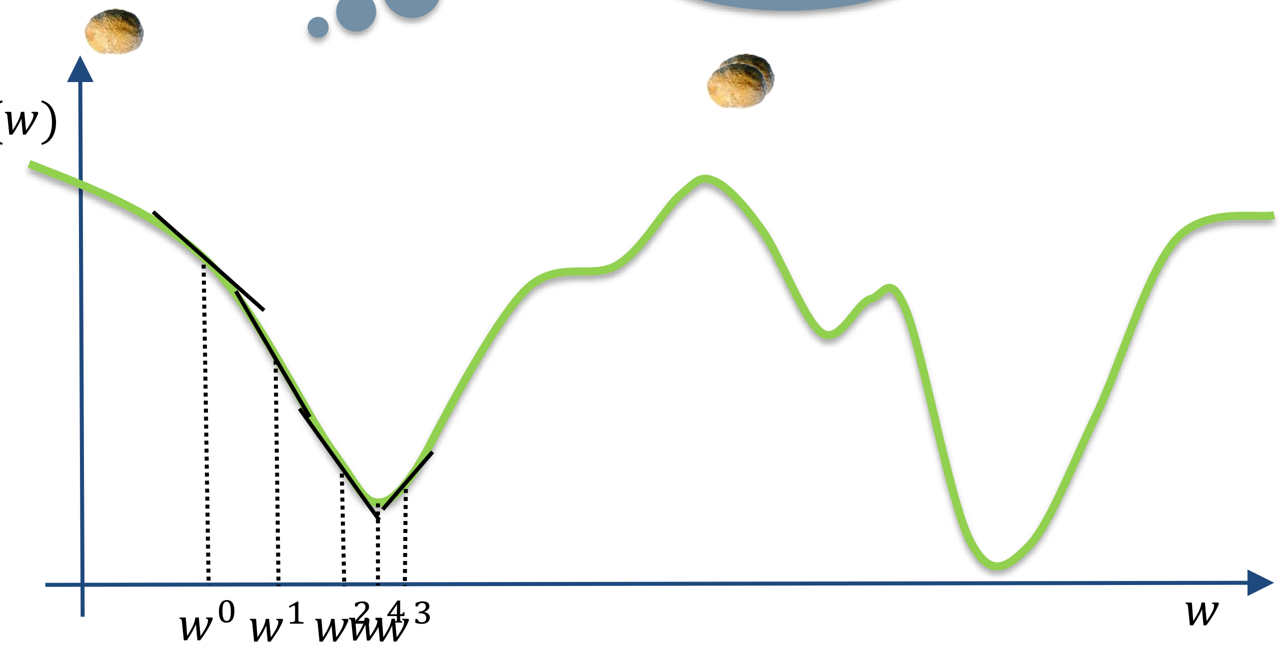


Gradient descent - Backpropagation

Use multiple restarts to seek for a proper global minimum.

Finding the weighs of a Neural Network is a non

$$argmin_w \ E(w) = \sum_{n=1}^{N} (t_n - g(x_n, w))^2$$



We iterate starting from an initial random configuration

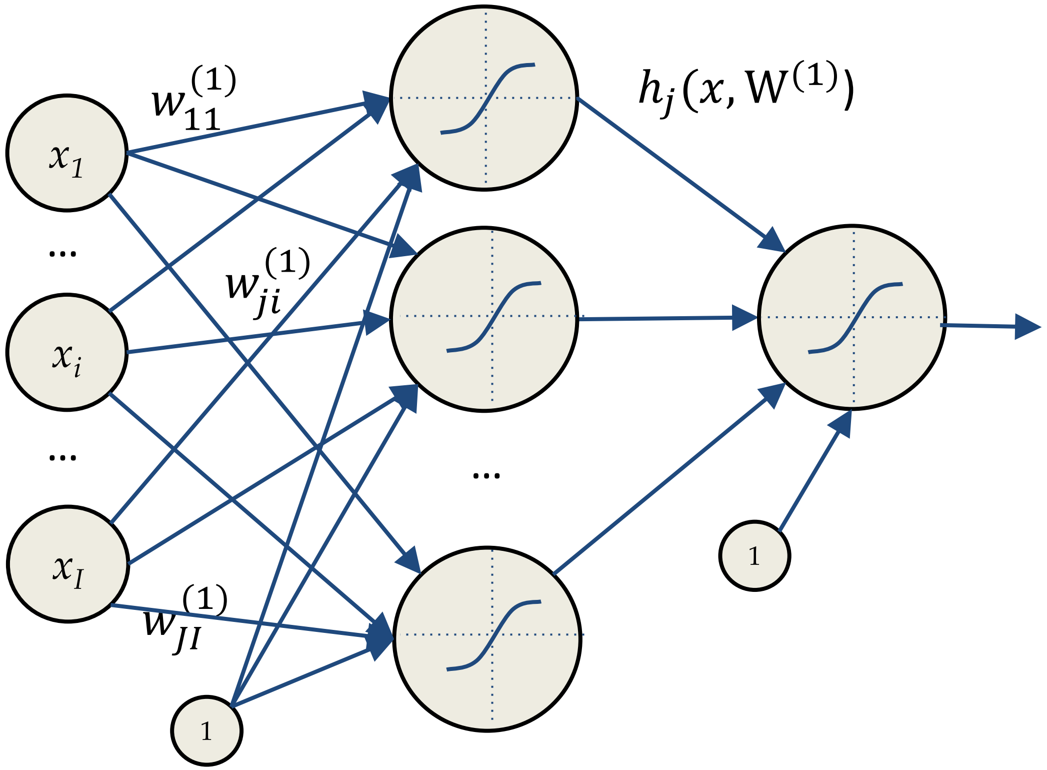
$$w^{k+1} = w^k - \eta \left. \frac{\partial E(w)}{\partial w} \right|_{w^k}$$

To avoid local minima can use momentum

It depends on where we start from

$$w^{k+1} = w^k - \eta \left. \frac{\partial E(w)}{\partial w} \right|_{w^k} - \alpha \left. \frac{\partial E(w)}{\partial w} \right|_{w^{k-1}}$$

Gradient Descent Example



$$g_1(x_n|w) = g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

$$E(w) = \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2$$

Use *j=3* and *i=5*

Compute the $w_{ji}^{(1)}$ weight update formula by gradient descent

$$g_1(x_n|w) = g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

$$E(w) = \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2$$

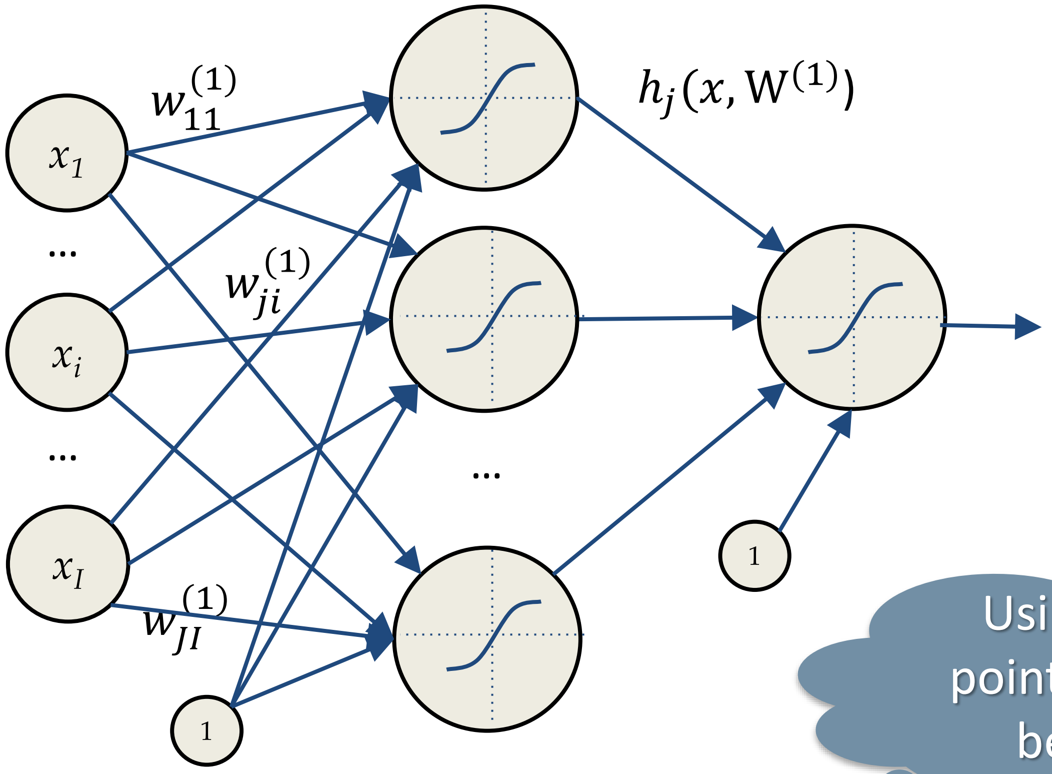
$$\frac{\partial E(w)}{\partial w_{3,5}^{(1)}} = \frac{\partial \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2}{\partial w_{3,5}^{(1)}} = \sum_{n=1}^{N} \frac{\partial (t_n - g_1(x_n, w))^2}{\partial w_{3,5}^{(1)}} = -2 \sum_{n}^{N} (t_n - g_1(x_n, w)) \frac{\partial g_1(x_n, w)}{\partial w_{3,5}^{(1)}}$$

$$\frac{\partial g_1(x_n, w)}{\partial w_{3,5}^{(1)}} = \frac{\partial g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j(.)\right)}{\partial w_{3,5}^{(1)}} = g_1'(x_n, w) \cdot \frac{\partial \sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j(.)}{\partial w_{3,5}^{(1)}} = g_1'(x_n, w) \cdot w_{1,3}^{(2)} \cdot \frac{\partial h_3\left(\sum_{i=0}^{I} w_{3i}^{(1)} \cdot x_{i,n}\right)}{\partial w_{3,5}^{(1)}}$$

$$\frac{\partial h_3\left(\sum_{i=0}^{I} w_{3i}^{(1)} \cdot x_{i,n}\right)}{\partial w_{3,5}^{(1)}} = h_3'\left(\sum_{i=0}^{I} w_{3,i}^{(1)} \cdot x_{i,n}\right) \frac{\partial \sum_{i=0}^{I} w_{3,i}^{(1)} \cdot x_{i,n}}{\partial w_{3,5}^{(1)}} = h_3'\left(\sum_{i=0}^{I} w_{3,i}^{(1)} \cdot x_{i,n}\right) x_{5,n}$$

$$\frac{\partial E(w)}{\partial w_{3,5}^{(1)}} = -2 \sum_{n}^{N} (t_n - g_1(x_n, w)) g_1'(x_n, w) w_{1,3}^{(2)} h_3'\left(\sum_{i=0}^{I} w_{3,i}^{(1)} \cdot x_{i,n}\right) x_{5,n}$$

Gradient Descent Example



$$g_1(x_n|w) = g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

$$E(w) = \sum_{n}^{N} (t_n - g_1(x_n, w))^2$$

Using all the data points (**batch**) might be unpractical

$$\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}} = -2\sum_{n}^{N}\big(t_n - g_1(x_n, w)\big) g_1'(x_n, w) w_{1j}^{(2)} h_j'\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right) x_i$$

Gradient Descent Variations

Batch gradient descent

$$\frac{\partial E(w)}{\partial w} = \frac{1}{N}\sum_{n}^{N}\frac{\partial E(x_n, w)}{\partial w}$$

Use a single sample, unbiased, but with high variance

Stochastic gradient descent (SGD)

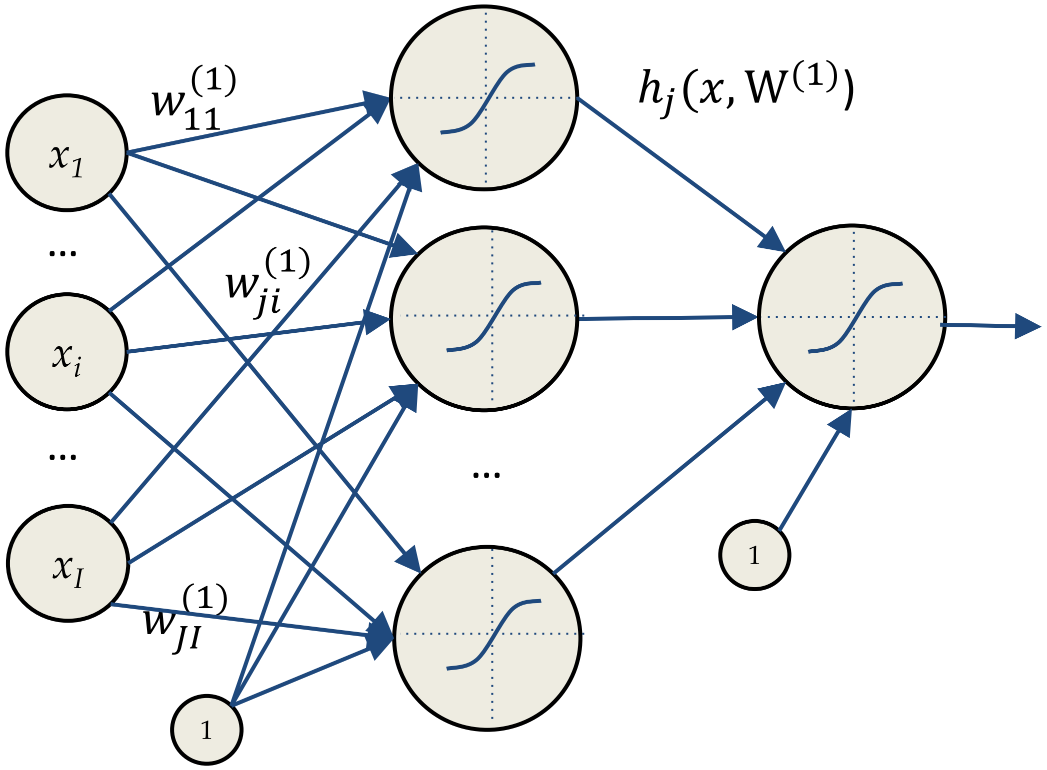
$$\frac{\partial E(w)}{\partial w} \approx \frac{\partial E_{SGD}(w)}{\partial w} = \frac{\partial E(x_n, w)}{\partial w}$$

Use a subset of samples, good trade off variance-computation

Mini-batch gradient descent

$$\frac{\partial E(w)}{\partial w} \approx \frac{\partial E_{MB}(w)}{\partial w} = \frac{1}{M}\sum_{n \in Minibatch}^{M<N}\frac{\partial E(x_n, w)}{\partial w}$$

Gradient Descent Example



$$g_1(x_n|w) = g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

$$E(w) = \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2$$

Can I make it automatic?

$$\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}} = -2\sum_{n}^{N}(t_n - g_1(x_n, w))g_1'(x_n, w)w_{1j}^{(2)}h_j'\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)x_i$$

Backpropagation and Chain Rule (1)

Weights update can be done in parallel, locally, and requires just 2 passes

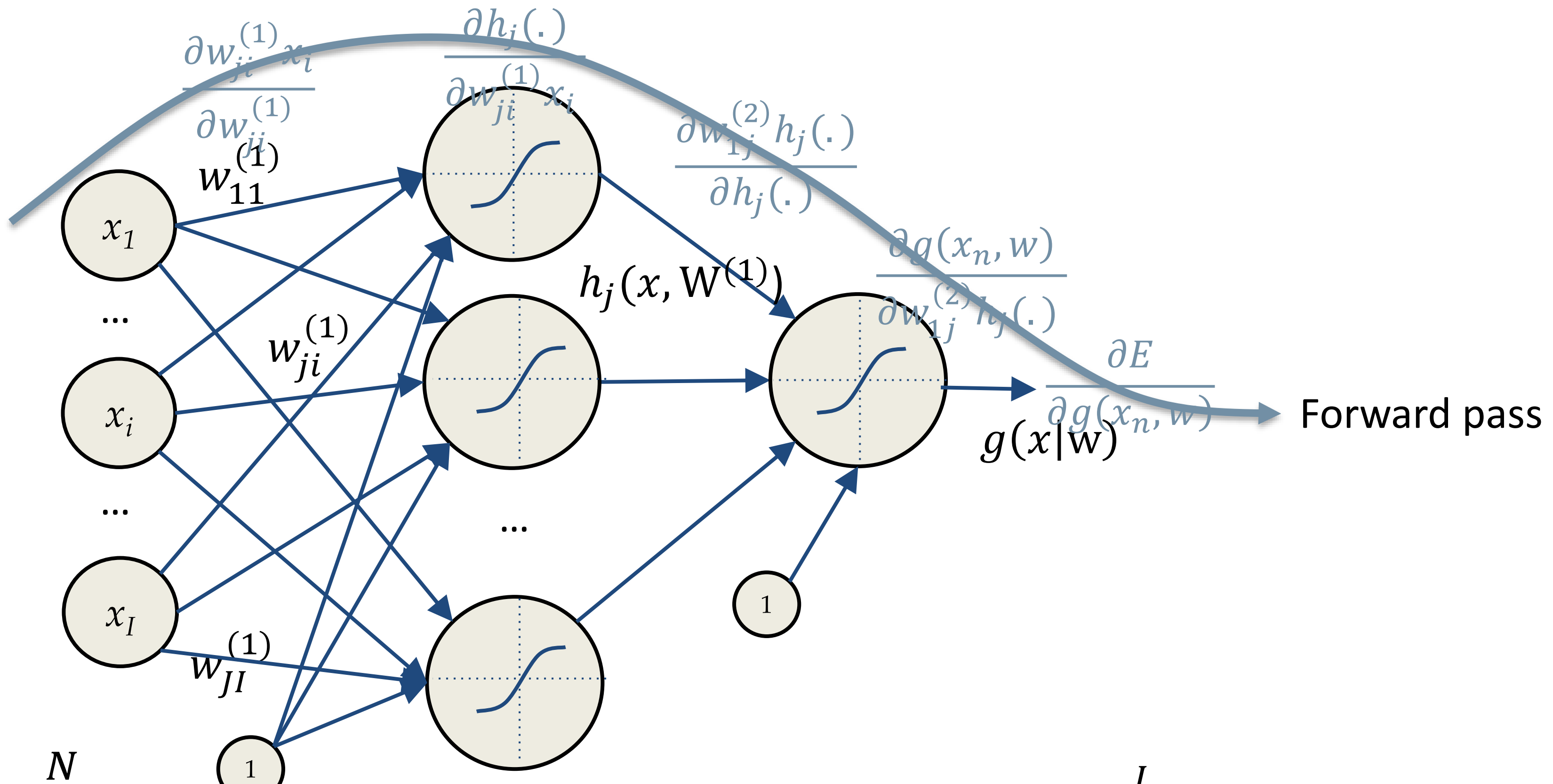
- Let x be a real number and two functions $f: \Re \rightarrow \Re$ and $g: \Re \rightarrow \Re$
- Consider the composed function $z = f\big(g(x)\big) = f(y)$ where $y = g(x)$
- The derivative of $\mathbf{z}$ w.r.t. x can be computed applying the chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = f'(y)g'(x) = f'\big(g(x)\big)g'(x)$$

The same holds for backpropagation

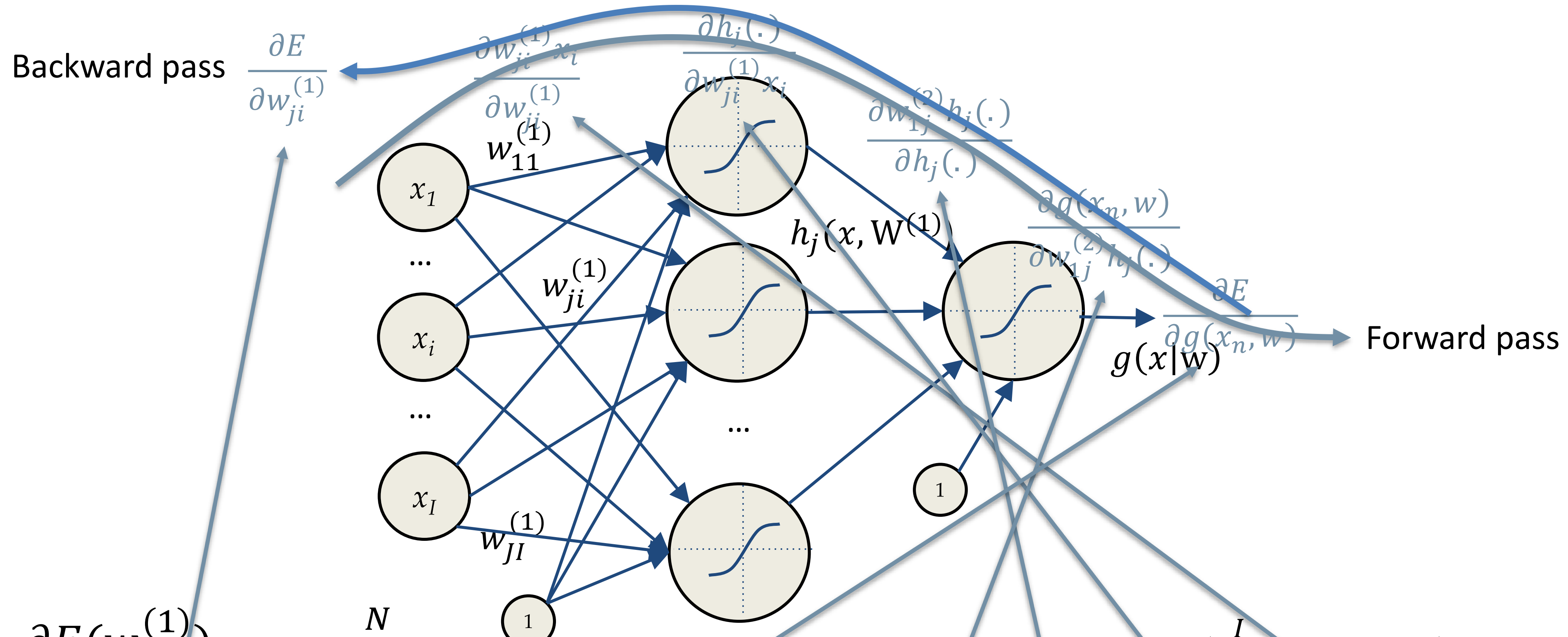
$$\underbrace{\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}}}_{\frac{\partial E}{\partial w_{ji}^{(1)}}} = \underbrace{-2\sum_{n}^{N}\big(t_n - g_1(x_n, w)\big)}_{\frac{\partial E}{\partial g(x_n, w)}} \cdot \underbrace{g_1'\,(x_n, w)}_{\frac{\partial g(x_n, w)}{\partial w_{1j}^{(2)} h_j(.)}} \cdot \underbrace{w_{1j}^{(2)}}_{\frac{\partial w_{1j}^{(2)} h_j(.)}{\partial h_j(.)}} \cdot \underbrace{h_j'\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)}_{\frac{\partial h_j(.)}{\partial w_{ji}^{(1)} x_i}} \cdot \underbrace{x_i}_{\frac{\partial w_{ji}^{(1)} x_i}{\partial w_{ji}^{(1)}}}$$

Backpropagation and Chain Rule (2)



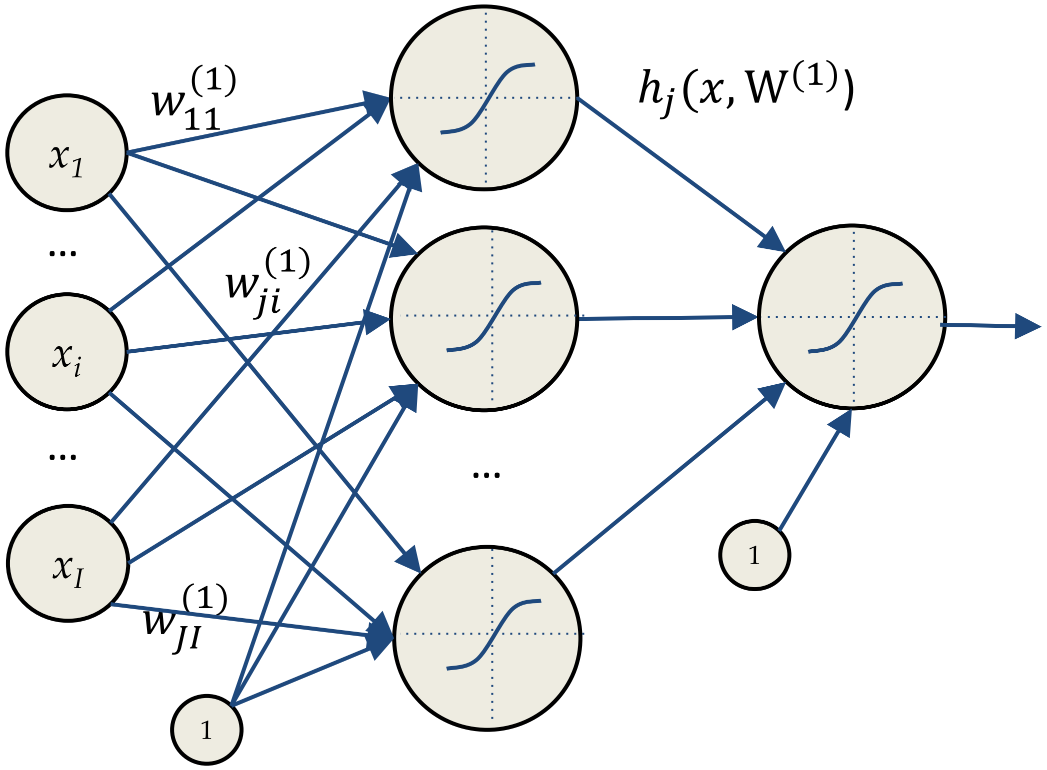
$$\underbrace{\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}}}_{\frac{\partial E}{\partial w_{ji}^{(1)}}} = \underbrace{-2\sum_{n}^{N}\left(t_n - g_1(x_n, w)\right)}_{\frac{\partial E}{\partial g(x_n,w)}} \cdot \underbrace{g_1'\,(x_n, w)}_{\frac{\partial g(x_n,w)}{\partial w_{1j}^{(2)} h_j(.)}} \cdot \underbrace{w_{1j}^{(2)}}_{\frac{\partial w_{1j}^{(2)} h_j(.)}{\partial h_j(.)}} \cdot \underbrace{h_j'\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)}_{\frac{\partial h_j(.)}{\partial w_{ji}^{(1)} x_i}} \cdot \underbrace{x_i}_{\frac{\partial w_{ji}^{(1)} x_i}{\partial w_{ji}^{(1)}}}$$

Backpropagation and Chain Rule (2)



$$\underbrace{\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}}}_{\frac{\partial E}{\partial w_{ji}^{(1)}}} = \underbrace{-2\sum_n^N \left(t_n - g_1(x_n,w)\right)}_{\frac{\partial E}{\partial g(x_n,w)}} \cdot \underbrace{g_1'(x_n,w)}_{\frac{\partial g(x_n,w)}{\partial w_{1j}^{(2)} h_j(.)}} \cdot \underbrace{w_{1j}^{(2)}}_{\frac{\partial w_{1j}^{(2)} h_j(.)}{\partial h_j(.)}} \cdot \underbrace{h_j'\left(\sum_{i=0}^I w_{ji}^{(1)} \cdot x_{i,n}\right)}_{\frac{\partial h_j(.)}{\partial w_{ji}^{(1)} x_i}} \cdot \underbrace{x_i}_{\frac{\partial w_{ji}^{(1)} x_i}{\partial w_{ji}^{(1)}}}$$

Gradient Descent Example



$$g_1(x_n|w) = g_1\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

$$E(w) = \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2$$

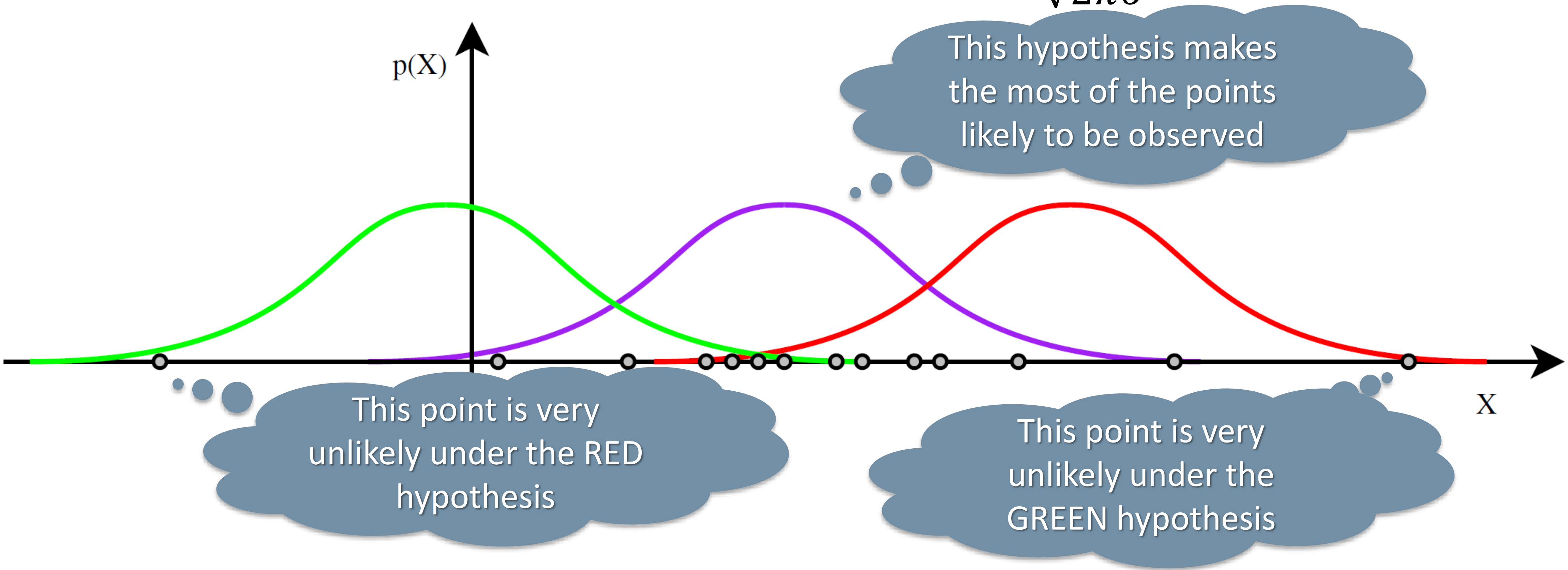
Why should I use this?

$$\frac{\partial E(w_{ji}^{(1)})}{\partial w_{ji}^{(1)}} = -2\sum_{n}^{N} (t_n - g_1(x_n, w))g_1'(x_n, w)w_{1j}^{(2)} h_j'\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)x_{i,n}$$

A Note on Maximum Likelihood Estimation

Let's observe i.i.d. samples from a Gaussian distribution with known $\boldsymbol{\sigma}^2$

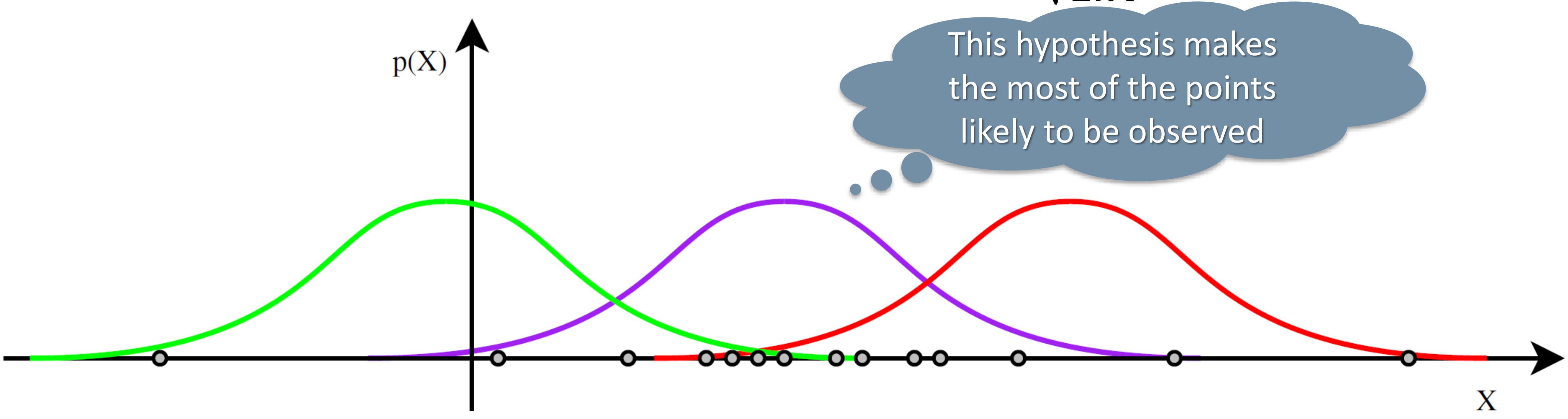
$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



A Note on Maximum Likelihood Estimation

Let's observe i.i.d. samples from a Gaussian distribution with known $\boldsymbol{\sigma}^2$

$$x_1, x_2, \ldots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



<u>Maximum Likelihood</u>: Chose parameters which maximize data probability

Maximum Likelihood Estimation: The Recipe

Let $\theta = (\theta_1, \theta_2, \dots, \theta_p)^T$ a vector of parameters, find the MLE for θ:

- Write the likelihood $L = P(Data|\theta)$ for the data
- [Take the logarithm of likelihood $\mathrm{l} = \log P(Data|\theta)$]
- Work out $\frac{\partial L}{\partial \theta}$ or $\frac{\partial l}{\partial \theta}$ using high-school calculus
- Solve the set of simultaneous equations $\frac{\partial L}{\partial \theta_i} = 0$ or $\frac{\partial l}{\partial \theta_i} = 0$
- Check that θ^{MLE} is a maximum

Optional

We know already about gradient descent, let's try with some analitical stuff ...

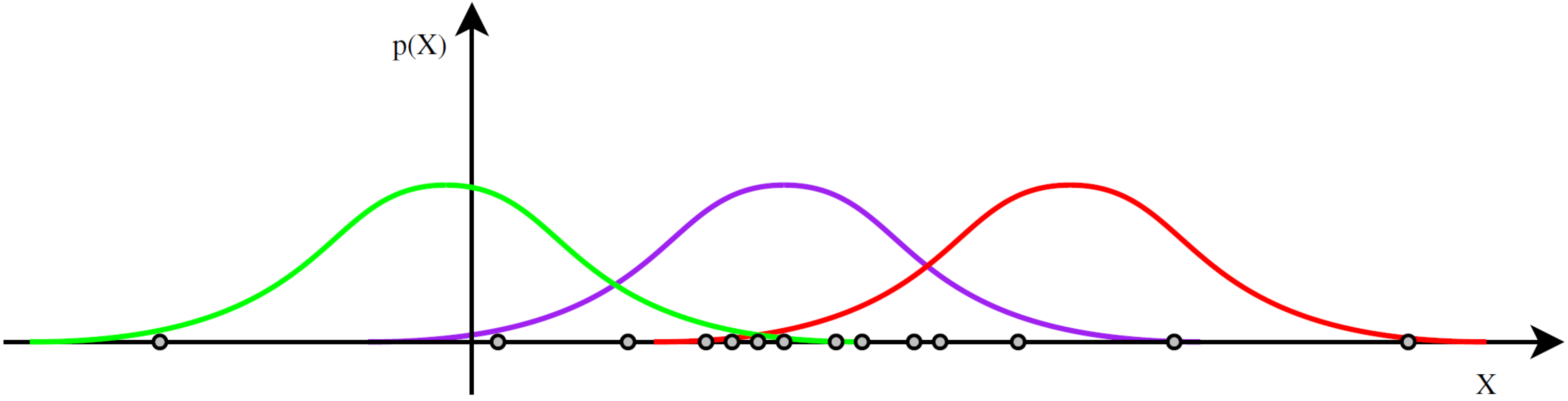
To maximize/minimize the (log)likelihood you can use:

- Analytical Techniques (i.e., solve the equations)
- Optimization Techniques (e.g., Lagrange multipliers)
- Numerical Techniques (e.g., gradient descend)

Maximum Likelihood Estimation Example

Let's observe i.i.d. samples coming from a Gaussian with known σ^2

$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Find the Maximum Likelihood Estimator for μ

Maximum Likelihood Estimation Example

Let's observe i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Write the likelihood $\boldsymbol{L = P(Data|\theta)}$ for the data

$$L(\mu) = p(x_1, x_2, \dots, x_N|\mu,\sigma^2) = \prod_{n=1}^{N} p(x_n|\mu,\sigma^2) =$$

$$= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}$$

Maximum Likelihood Estimation Example

Let's observe i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Take the logarithm $l = \log P(Data|\theta)$ of the likelihood

$$l(\mu) = \log\left(\prod_{n=1}^{N} \frac{1}{\sqrt{2\cdot\pi}\sigma} e^{-\frac{(x_n-\mu)^2}{2\cdot\sigma^2}}\right) = \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\cdot\pi}\sigma} e^{-\frac{(x_n-\mu)^2}{2\cdot\sigma^2}} =$$

$$= N \cdot \log \frac{1}{\sqrt{2\cdot\pi}\sigma} - \frac{1}{2\cdot\sigma^2} \sum_{n}^{N} (x_n - \mu)^2$$

Maximum Likelihood Estimation Example

Let's observe i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad\qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Work out $\boldsymbol{\partial l / \partial \theta}$ using high-school calculus

$$\frac{\partial l(\mu)}{\partial \mu} = \frac{\partial}{\partial \mu}\left(N \cdot \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2} \sum_n^N (x_n - \mu)^2 \right) =$$

$$= -\frac{1}{2\sigma^2} \frac{\partial}{\partial \mu} \sum_n^N (x_n - \mu)^2 = \frac{1}{2\sigma^2} \sum_n^N 2(x_n - \mu)$$

Maximum Likelihood Estimation Example

Let's observe i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$x_1, x_2, \dots, x_N \sim N(\mu, \sigma^2) \qquad p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Solve the set of simultaneous equations $\frac{\partial l}{\partial \theta_i} = 0$

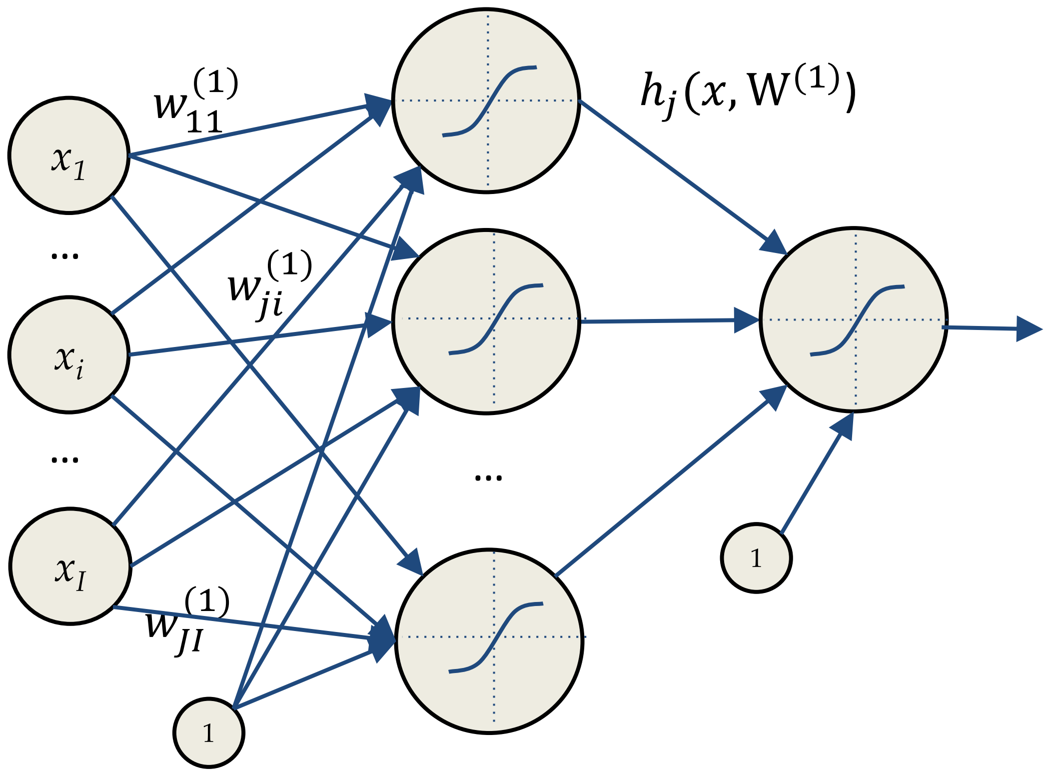
$$\frac{1}{2\sigma^2} \sum_n^N 2(x_n - \mu) = 0$$

$$\sum_n^N (x_n - \mu) = 0$$

$$\sum_n^N x_n = \sum_n^N \mu \Rightarrow \mu^{MLE} = \frac{1}{N} \sum_n^N x_n$$

Let's apply this all to Neural Networks!

Neural Networks for Regression

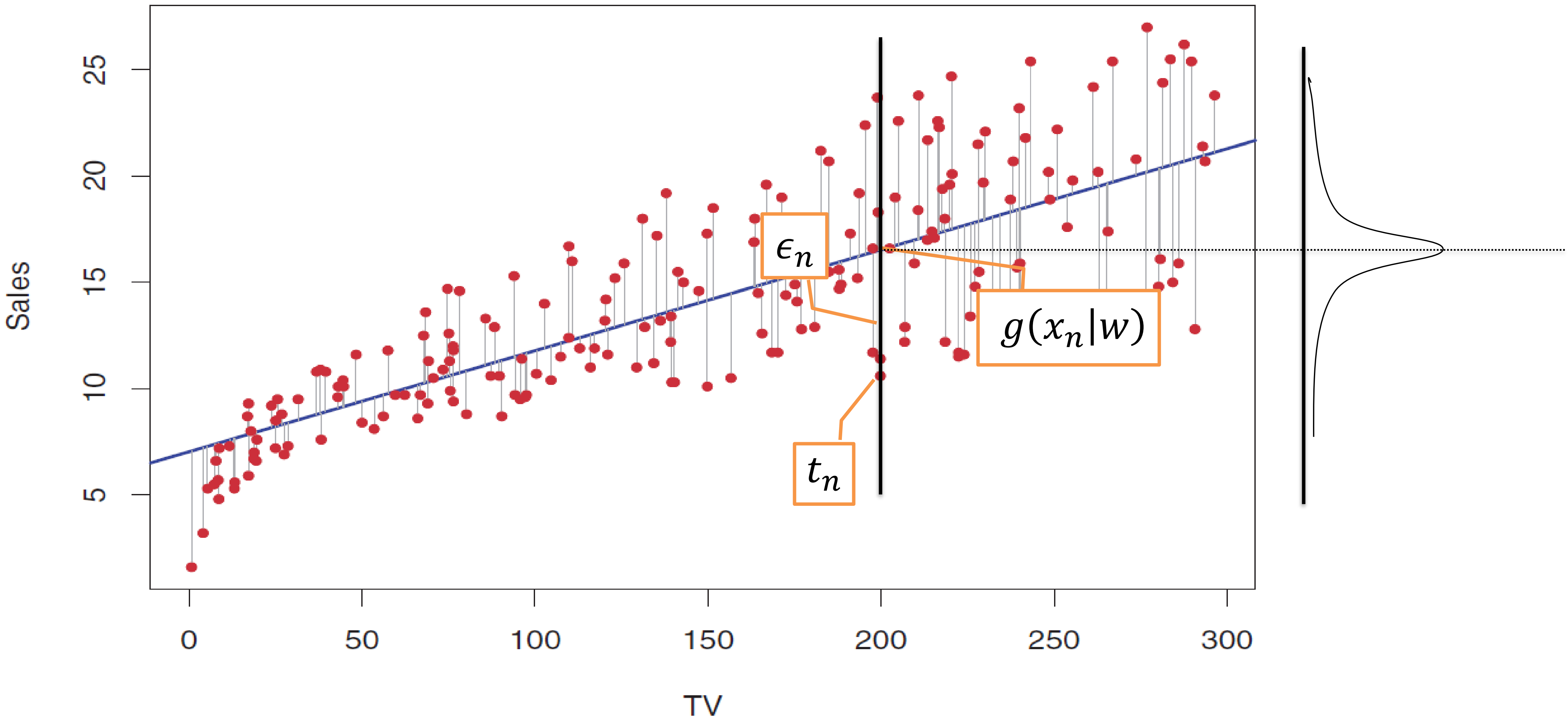


$$g(x_n|w) = g\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{j=0}^{J} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

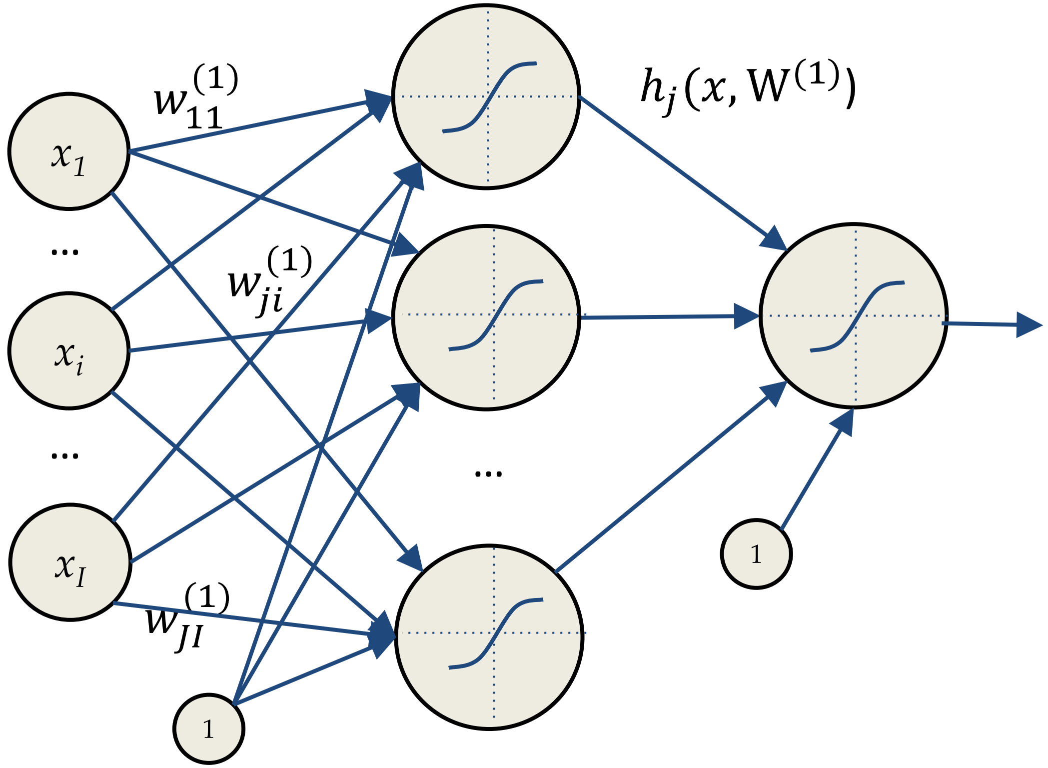
Goal: approximate a target function t having N observations

$$t_n = g(x_n|w) + \epsilon_n, \qquad \epsilon_n \sim N(0, \sigma^2)$$

Statistical Learnig Framework



Neural Networks for Regression



$$g(x_n|w) = g\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{j=0}^{J} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

Goal: approximate a target function t having N observations

$$t_n = g(x_n|w) + \epsilon_n, \qquad \epsilon_n \sim N(0, \sigma^2) \quad \Rightarrow \quad t_n \sim N(g(x_n|w), \sigma^2)$$

Maximum Likelihood Estimation for Regression

We have i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$t_n \sim N(g(x_n|w), \sigma^2) \qquad p(t|g(x|w), \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(t-g(x|w)\right)^2}{2\sigma^2}}$$

Write the likelihood $L = P(Data|\theta)$ for the data

$$L(w) = p(t_1, t_2, \dots, t_N | g(x|w), \sigma^2) = \prod_{n=1}^{N} p(t_n | g(x_n|w), \sigma^2) =$$

$$= \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(t_n-g(x_n|w)\right)^2}{2\sigma^2}}$$

Maximum Likelihood Estimation for Regression

We have i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

$$t_n \sim N(g(x_n|w), \sigma^2) \qquad p(t|g(x|w), \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-g(x|w))^2}{2\sigma^2}}$$

Write the loglikelihood $\mathbf{l = \log} \, P(Data|\theta)$ for the data

$$l(w) = log \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t_n-g(x_n|w))^2}{2\sigma^2}} = \sum_{n}^{N} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t_n-g(x_n|w))^2}{2\sigma^2}}\right)$$

$$= \sum_{n}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2}(t_n - g(x_n|w))^2$$

Maximum Likelihood Estimation for Regression

We have i.i.d. samples coming from a Gaussian with known $\boldsymbol{\sigma^2}$

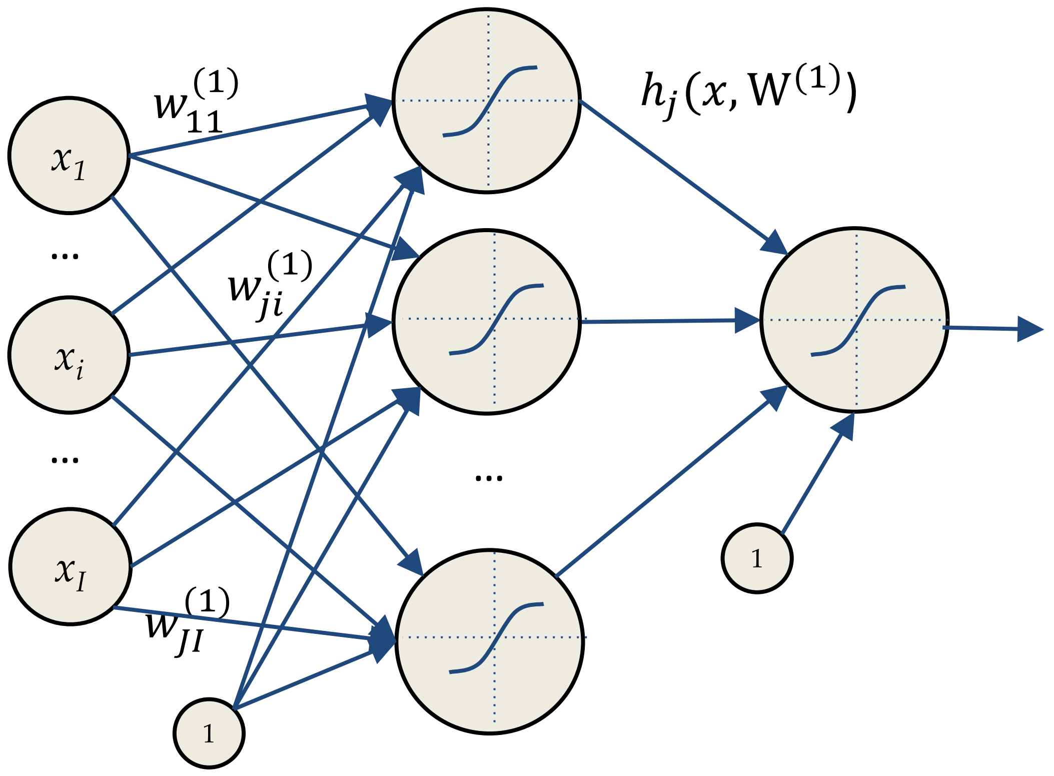
$$t_n \sim N(g(x_n|w), \sigma^2) \qquad p(t|g(x|w), \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-g(x|w))^2}{2\sigma^2}}$$

Look for the weights which maximixe the loglikelihood

$$argmax_w \, l(w) = argmax_w \sum_n^N \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2}\left(t_n - g(x_n|w)\right)^2 =$$

$$= argmin_w \sum_n^N \left(t_n - g(x_n|w)\right)^2$$

Neural Networks for Classification



$$g(x_n|w) = g\left(\sum_{j=0}^{J} w_{1j}^{(2)} \cdot h_j\left(\sum_{i=0}^{I} w_{ji}^{(1)} \cdot x_{i,n}\right)\right)$$

Goal: approximate a posterior probability t having N observations

$$g(x_n|w) = p(t_n|x_n), \quad t_n \in \{0, 1\} \quad \Rightarrow \quad t_n \sim Be\big(g(x_n|w)\big)$$

Maximum Likelihood Estimation for Classification

We have some i.i.d. samples coming from a Bernulli distribution

$$t_n \sim Be\big(g(x_n|w)\big) \qquad p\big(t|g(x|w)\big) = g(x|w)^t \cdot \big(1 - g(x|w)\big)^{1-t}$$

Write the likelihood $L = P(Data|\theta)$ for the data

$$L(w) = p\big(t_1, t_2, \dots, t_N|g(x|w)\big) = \prod_{n=1}^{N} p\big(t_n|g(x_n|w)\big) =$$

$$= \prod_{n=1}^{N} g(x_n|w)^{t_n} \cdot \big(1 - g(x_n|w)\big)^{1-t_n}$$

Maximum Likelihood Estimation for Classification

We have some i.i.d. samples coming from a Bernulli distribution

$$t_n \sim Be\big(g(x_n|w)\big) \qquad p\big(t|g(x|w)\big) = g(x|w)^t \cdot \big(1 - g(x|w)\big)^{1-t}$$

Compute the log likelihood $\boldsymbol{l = logP(Data|\theta)}$ for the data

$$l(w) = \log \prod_{n=1}^{N} g(x_n|w)^{t_n} \cdot \big(1 - g(x_n|w)\big)^{1-t_n}$$

$$= \sum_{n}^{N} t_n \log g(x_n|w) + (1 - t_n) \log(1 - g(x_n|w))$$

Maximum Likelihood Estimation for Classification

We have some i.i.d. samples coming from a Bernulli distribution

$$t_n \sim Be\big(g(x_n|w)\big) \qquad p\big(t|g(x|w)\big) = g(x|w)^t \cdot \big(1 - g(x|w)\big)^{1-t}$$

Look for the weights which maximize the loglikelihood

$$argmax_w \, l(w) = argmax_w \sum_n^N t_n \log g(x_n|w) + (1 - t_n) \log(1 - g(x_n|w))$$

$$= argmin_w - \sum_n^N t_n \log g(x_n|w) + (1 - t_n) \log(1 - g(x_n|w))$$

Crossentropy
$- \sum_n^N t_n^T \log g(x_n|w)$

What about perceptron

How to Chose the Error Function?

We have observed different error functions so far

$$E(w) = \sum_{n=1}^{N} (t_n - g_1(x_n, w))^2$$

$$E(w) = -\sum_{n}^{N} t_n \, log \, g(x_n|w) + (1 - t_n) \, log(1 - g(x_n|w))$$

Sum of Squared Errors

Binary Crossentropy

Error functions define the task to be solved, but how to design them?

- Use all your knowledge/assumptions about the data distribution
- Exploit background knowledge on the task and the model
- Use your creativity!

This requires lots of trial and errors ...

As for the Perceptron ...

Hyperplanes Linear Algebra

Let consider the hyperplane (affine set) $L \in \Re^2$

$$L: w_0 + w^T \mathrm{x} = 0$$

Any two points $\mathbf{x}_1$ and $\mathbf{x}_2$ on $L \in \Re^2$ have

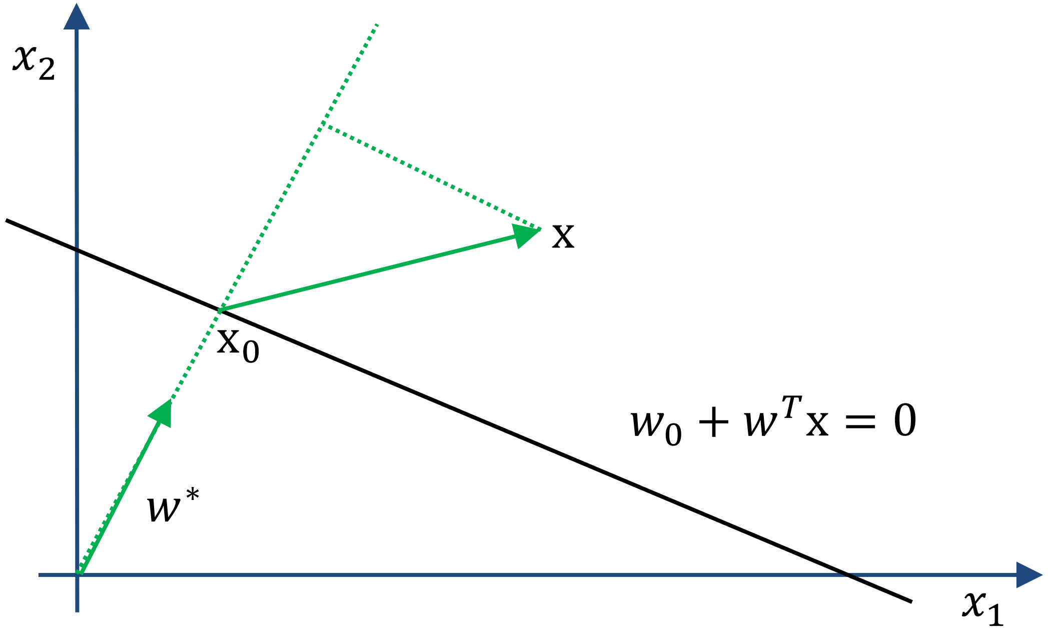
$$w^T(\mathrm{x}_1 - \mathrm{x}_2) = 0$$

The versor normal to $L \in \Re^2$ is then

$$w^* = w / \|w\|$$

For any point $\mathbf{x}_0$ in $L \in \Re^2$ we have

$$w^T \mathrm{x}_0 = -w_0$$



The signed distance of any point $\mathbf{x}$ from $L \in$

$$w^{*T}(\mathrm{x} - \mathrm{x}_0) = \frac{1}{\|w\|}(w^T \mathrm{x} + w_0)$$

$(w^T \mathrm{x} + w_0)$ is proportional to the distance of x from the plane defined by $(w^T \mathrm{x} + w_0) = 0$

Perceptron Learning Algorithm (1/2)

It can be shown, the error function the Hebbian rule is minimizing is the distance of misclassified points from the decision boundary.

Let's code the perceptron output as +1/-1

- If an output which should be +1 is misclassified then $\mathrm{w}^{\mathrm{T}}\mathrm{x} + \mathrm{w}_0 < 0$
- For an output with -1 we have the opposite

The goal becomes minimizing

Set of points misclassified

$$D(w, w_0) = -\sum_{i \in \mathrm{M}} t_i(\mathrm{w}^{\mathrm{T}}\mathrm{x}_\mathrm{i} + \mathrm{w}_0)$$

This is non negative and proportional to the distance of the misclassified points from $\mathrm{w}^{\mathrm{T}}\mathrm{x} + \mathrm{w}_0 = 0$

Perceptron Learning Algorithm (2/2)

Let's minimize by stochastic gradient descend the error function

$$D(w, w_0) = -\sum_{i \in \mathrm{M}} t_i(\mathrm{w}^{\mathrm{T}}\mathrm{x_i} + \mathrm{w_0})$$

The gradients with respect to the model parameters are

$$\frac{\partial D(w, w_0)}{\partial w} = -\sum_{i \in \mathrm{M}} t_i \cdot \mathrm{x_i} \qquad \frac{\partial D(w, w_0)}{\partial w_0} = -\sum_{i \in \mathrm{M}} t_i$$

Stochastic gradient descent applies for each misclassified point

$$\begin{pmatrix} w^{k+1} \\ w_0^{k+1} \end{pmatrix} = \begin{pmatrix} w^{k} \\ w_0^{k} \end{pmatrix} + \eta \begin{pmatrix} t_i \cdot x_i \\ t_i \end{pmatrix} = \begin{pmatrix} w^{k} \\ w_0^{k} \end{pmatrix} + \eta \begin{pmatrix} t_i \cdot x_i \\ t_i \cdot x_0 \end{pmatrix}$$

Hebbian learning implements Stocastic Gradient Descent