Dr. Edith Chorev (Metger) - Data & Neuro - Scientist

Worked in Max-Planck institute for experimental medicine
Worked in Humboldt University of Berlin
Worked in Ligature ltd.

DSR-19 graduate

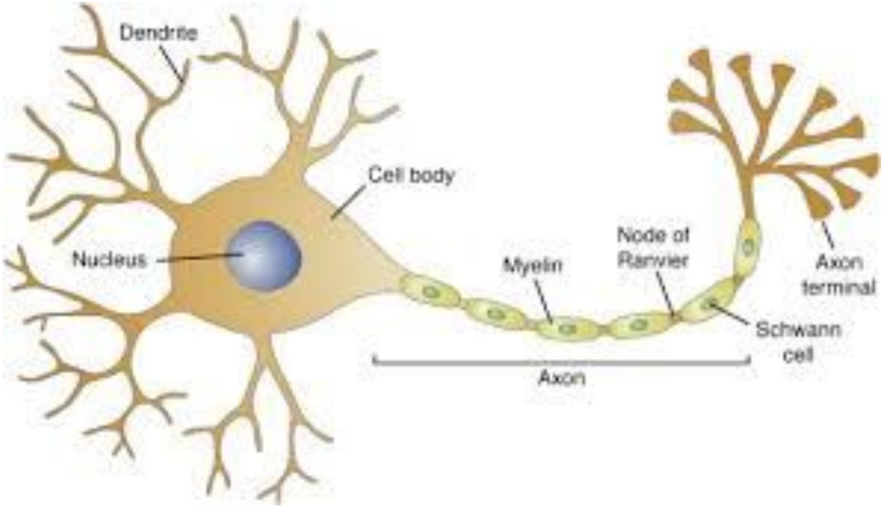
edith.chorev@gmail.com
https://www.linkedin.com/in/edith-chorev-bbb719/

Back Propagation

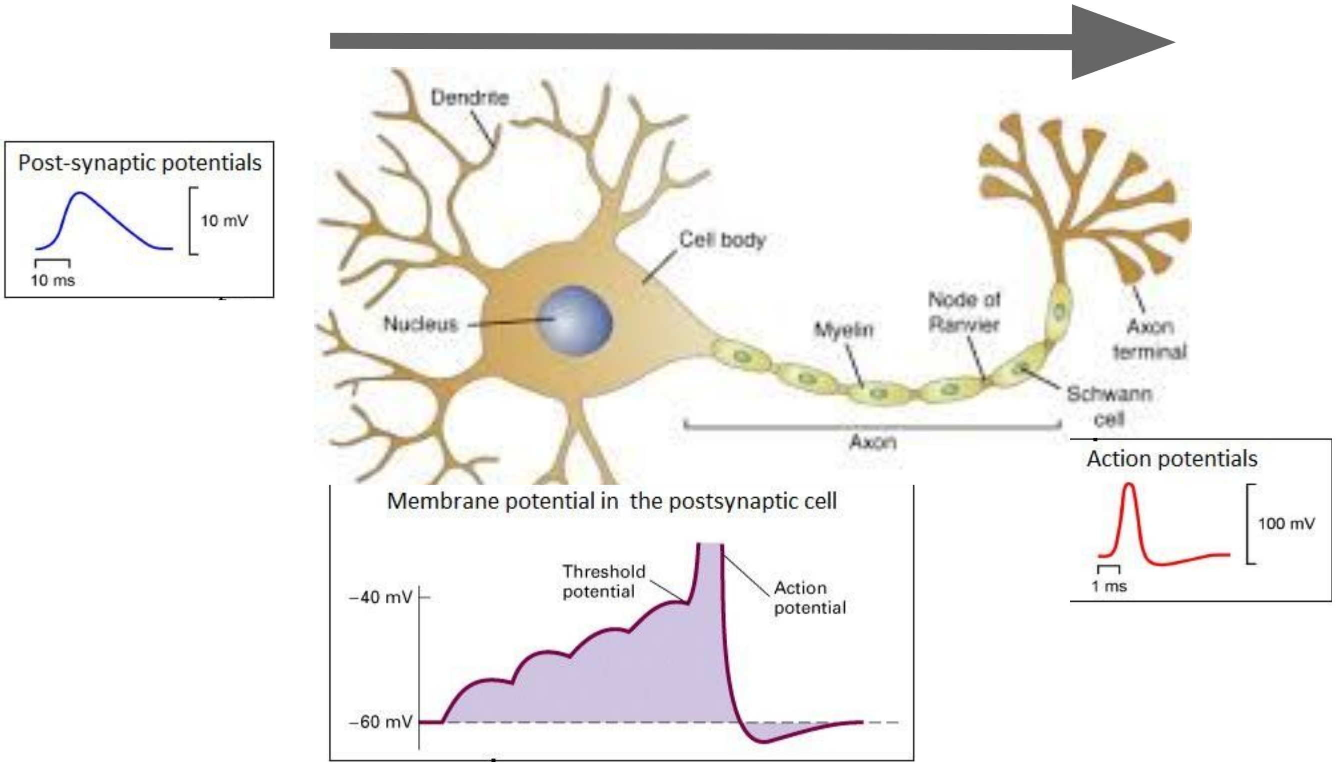
Neural Networks

In an attempt to create machines that can learn, people looked at the brain for inspiration

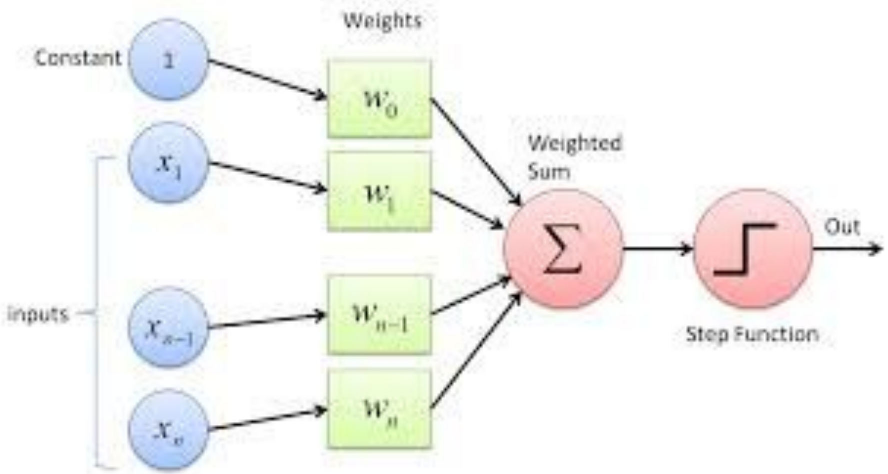
A neuron


Dendrite
Cell body
Nucleus
Myelin
Node of Ranvier
Axon terminal
Schwann cell
Axon

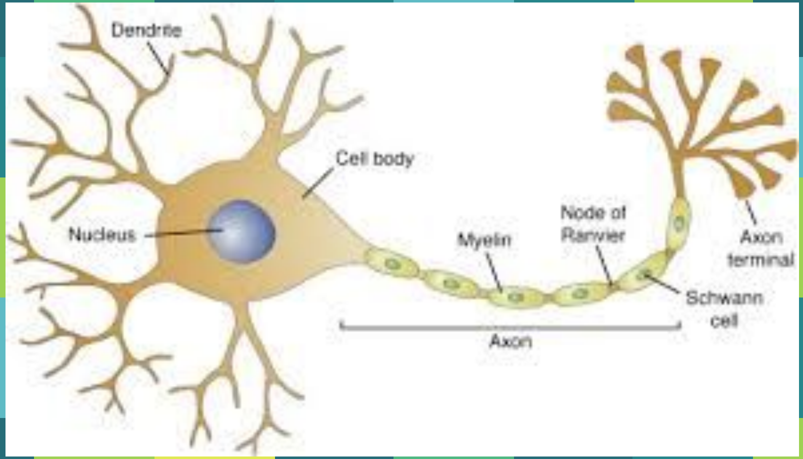
A neuron


Post-synaptic potentials
10 mV
10 ms
Dendrite
Cell body
Nucleus
Myelin
Node of Ranvier
Axon terminal
Schwann cell
Axon
Action potentials
100 mV
1 ms
Membrane potential in the postsynaptic cell
Threshold potential
Action potential
−40 mV
−60 mV

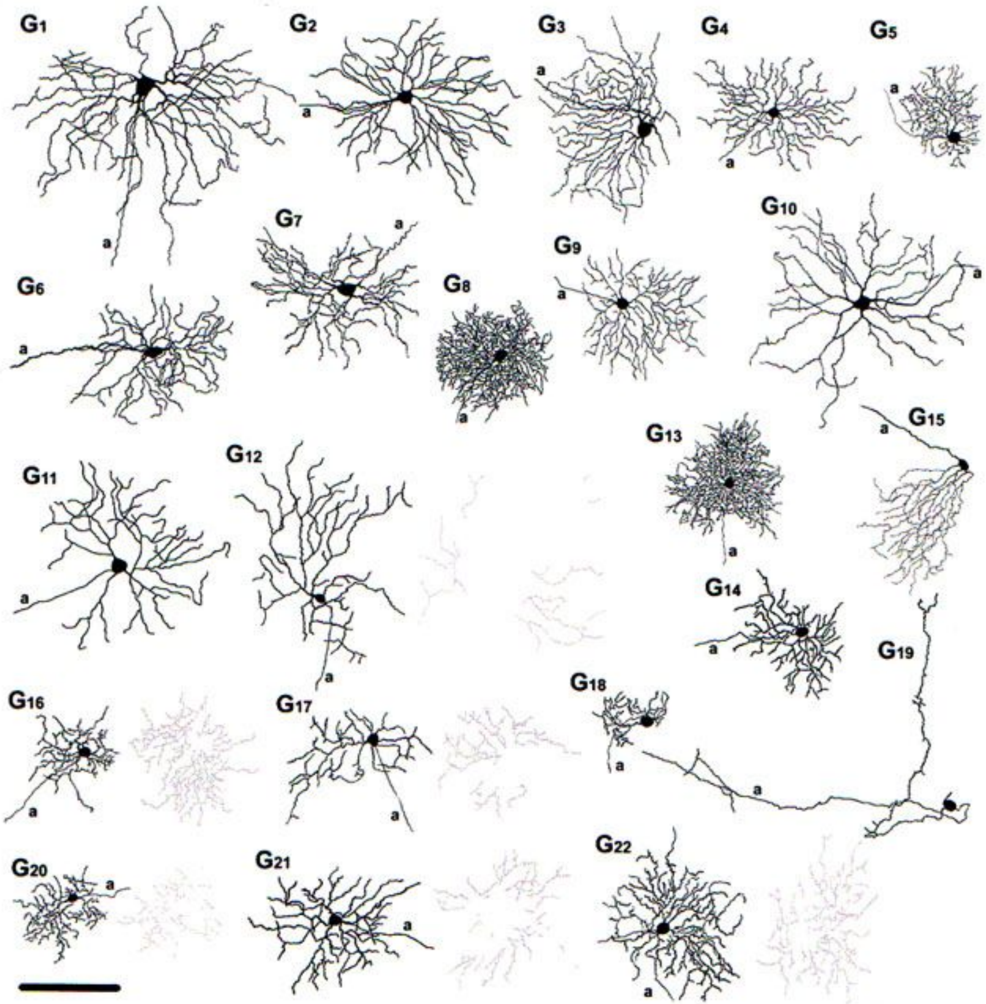
An artificial neuron



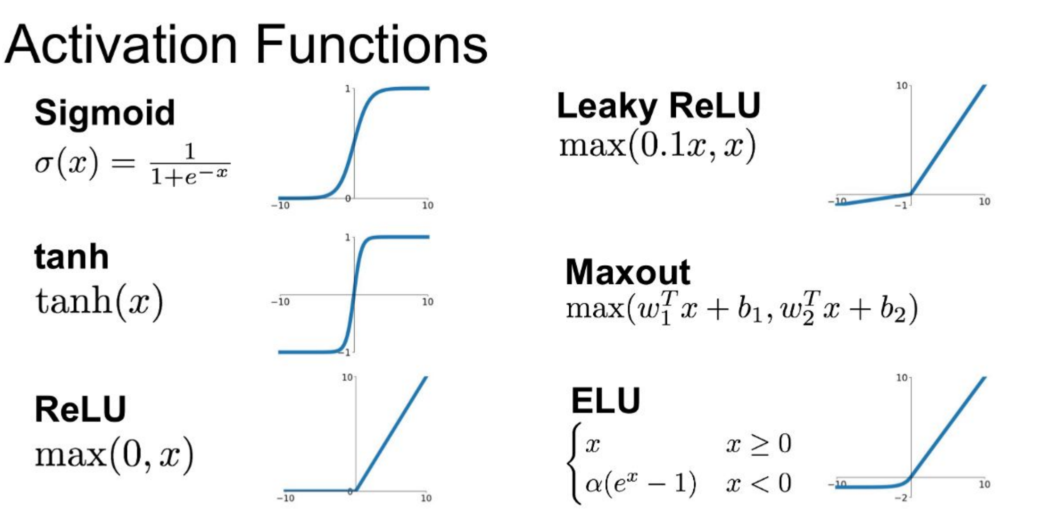
A Biological Neurons



The Neuronal Jungle



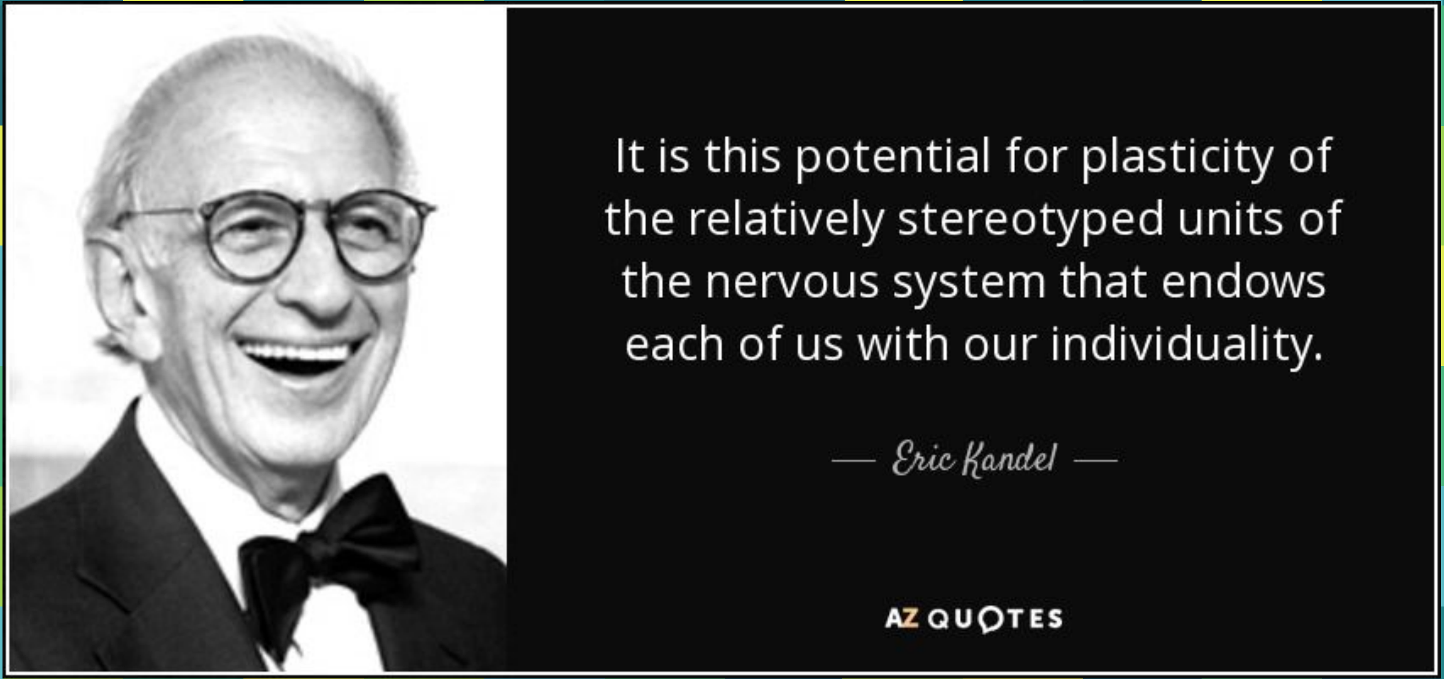
The AN Jungle



https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

A trip down memory lane...

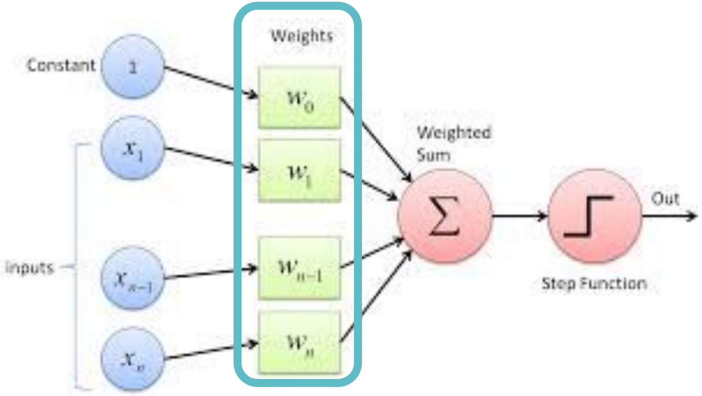
1. Perceptron
2. Optimization problem
3. Gradient descent
4. Multilayer NN
5. ANN vs. BNN


It is this potential for plasticity of the relatively stereotyped units of the nervous system that endows each of us with our individuality.
— Eric Kandel —
AZ QUOTES

The perceptron Algorithm: The Δ rule

Rosenblatt 1958 “The perceptron: a probabilistic model for information storage and organization in the Brain”.

Learning from labeled data - supervised learning.



The learning Algorithm: Δ rule

Perceptron training:

1. Initialize weights vector with small random numbers
2. Repeat until convergence:
   a. Loop over feature vector (x_j) and labels (l_i) in training set *D*.
   b. Take x and pass it through the perceptron, calculating the output values: $y_j = f(w(t) \square x_j)$
   c. *Update weights:*

      $w_i(t+1) = w_i(t) + \alpha(l_j - y_j)$

      for all $0 <= i < n$

W_i

$$w_i(t+1) = w_i(t)+\alpha(l_j-y_j) = w_i(t)+\alpha(l_j-w_i(t)\,\square\,x_j)$$

for all $0 <= i < n$

α - *learning rate*

(l_j-y_j) - the Δ between the actual label and the predicted label (also known as **error**).

3. Termination of training:
    a. When all samples are correctly labeled
    b. After a pre-set number of epochs
    c. After a pre-set number of epochs where the percentage of misclassified labels remains stable.

The learning step:

$w_i(t+1) = w_i(t) + \alpha(l_j\text{-}y_j) =$

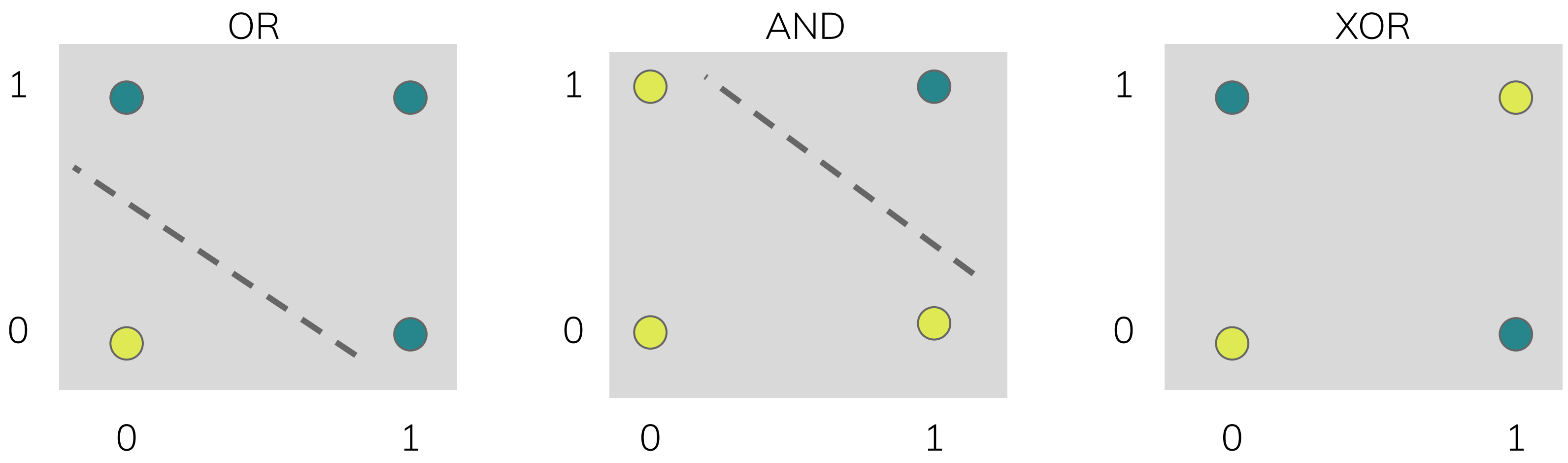
$w_i(t) + \alpha(l_j\text{-}x_{j,i}\,\square\,w(t))$

$(l_j\text{-}y_j)>0$: w_i needs to become bigger

$(l_j\text{-}y_j)<0$: w_i needs to become smaller

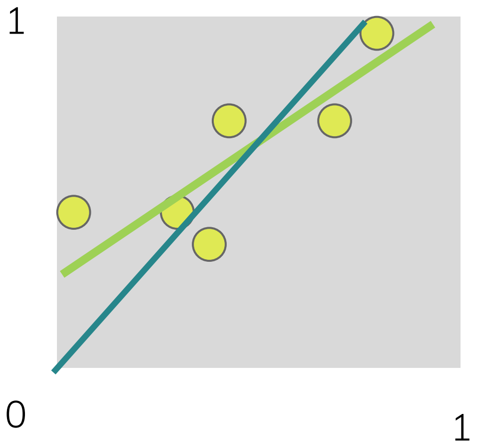
$(l_j\text{-}y_j)=0$: w_i should not change

Practical #1: the perceptron



Perceptron solves only linearly-separable problems!

$$y = wx + b$$



Bias - another variable that is not input dependent (as opposed to the weights).

Normalization of inputs and outputs

Neural Networks Winter

Winter did not last forever

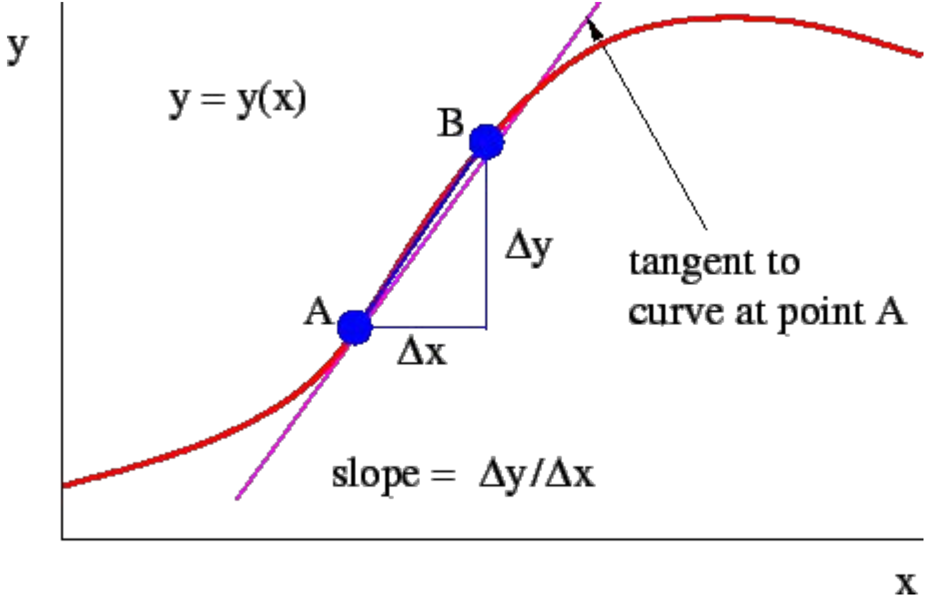
Some developments from the field of control theory emerged that will revive the NN field.

But first, we will need to do a short refresh on:

1. Derivatives
2. Partial derivatives and the chain rule

Calculus 101:

The derivative:



Definition of Derivative

Derivative of $f(x)$ at $x = a$

$$f'(a) = \lim_{x \to a} \frac{f(\mathrm{x}) - f(a)}{x - a}$$

$$\frac{d}{dx}(a) =$$

$$\frac{d}{dx}(ax) =$$

$$\frac{d}{dx}(x^n) =$$

$$\frac{d}{dx}(ax^n) =$$

Definition of Derivative

Derivative of *f(x)* at *x* = *a*

$$f'(a) = \lim_{x \to a} \frac{f(\mathrm{x}) - f(a)}{x - a}$$

$$\frac{d}{dx}(a) = 0$$

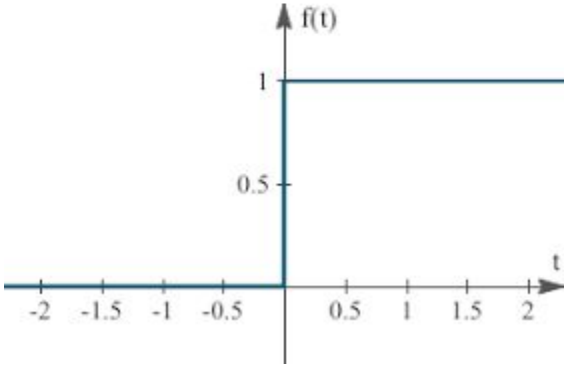
$$\frac{d}{dx}(ax) = a$$

$$\frac{d}{dx}(x^n) = n \cdot x^{n-1}$$

$$\frac{d}{dx}(ax^n) = a \cdot n \cdot x^{n-1}$$

Δ

Back to our step function - can it be differentiated?



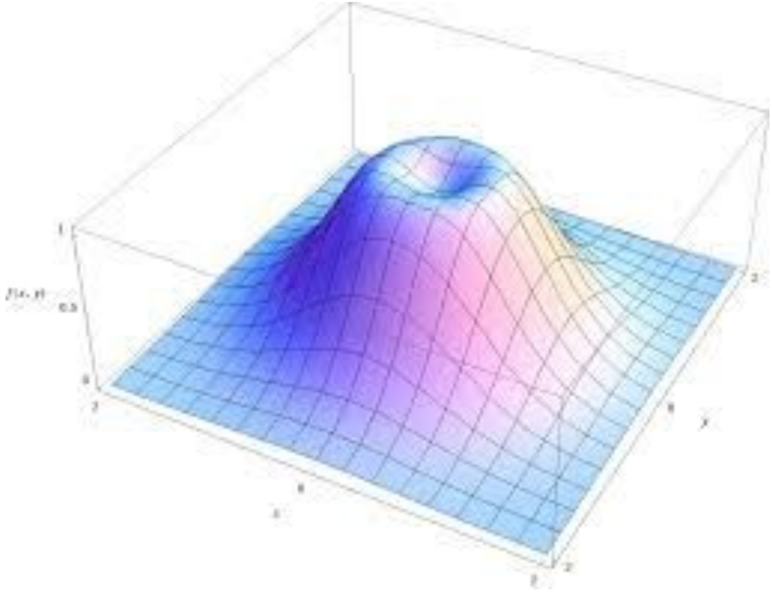
The chain rule

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

$$f(x) = (3x^3 + 2x)^4 = z^4$$

$$\frac{df}{dx} = 4z^3 \cdot \frac{dz}{dx} = 4z^3 \cdot (9x^2 + 2) = 4(3x^3 + 2x)^3 \cdot (9x^2 + 2)$$

Even the simple perceptron has more than one variable

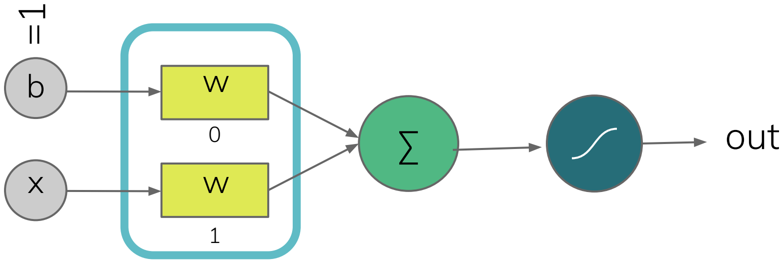


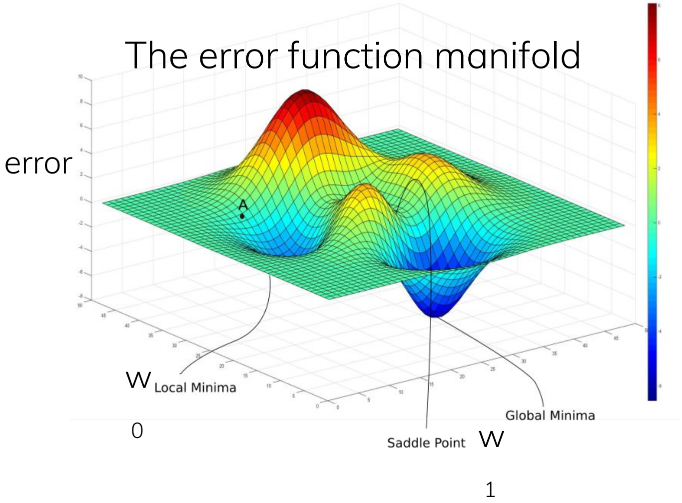
Partial Derivatives:

$$f(x, y) = f_y(x) = x^2 + xy + y^2.$$

$$\frac{\partial f}{\partial y}(x, y) = x + 2y.$$

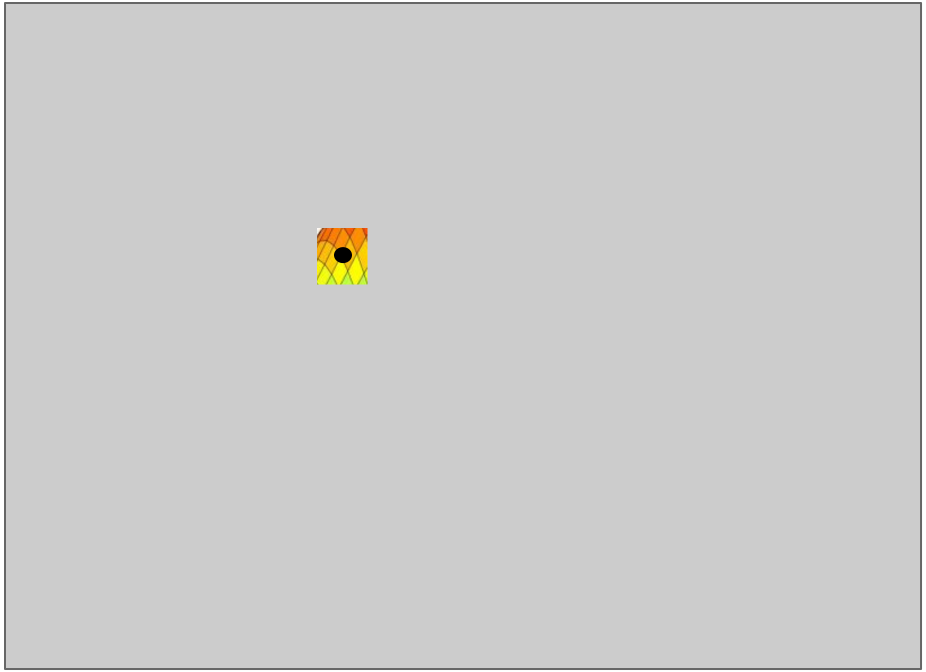
Back to the perceptron...





We want to find the minimum of this error function

We Don’t know how this manifold looks - what to do???



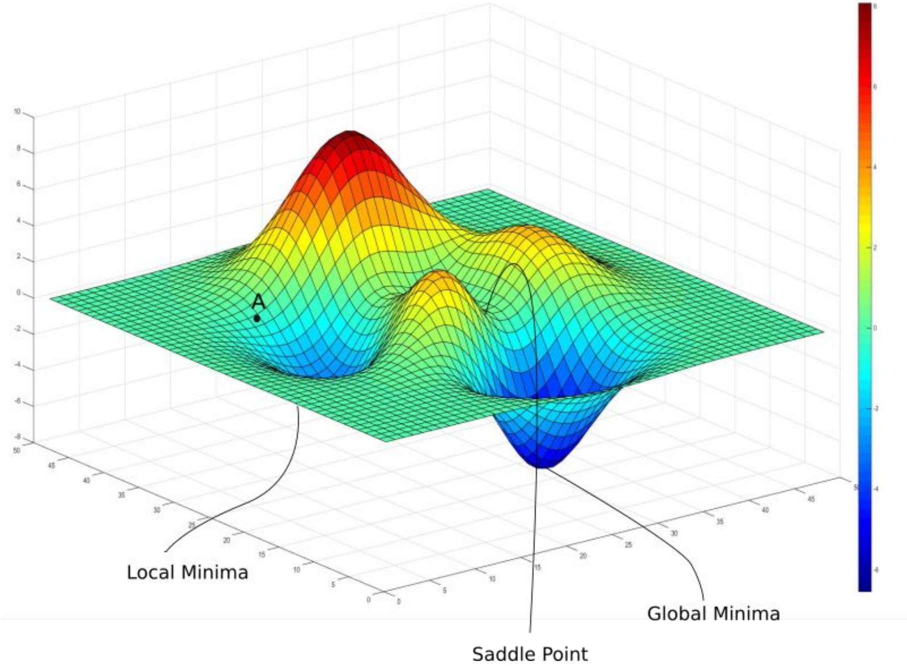
Optimization problem

Finding the minimum of an unknown error manifold

Gradient Descent

Gradient Descent:

Follow the downward slope... (i.e derivative)



The “**good enough**” principle :

“[An] optimization algorithm may not be guaranteed to arrive at even a local minimum in a reasonable amount of time, **but** it often finds a very low value of the [loss] function quickly enough to be useful.”

Goodfellow et al.

Backpropagation:

Developed by Henry J. Kelley (1960) & by Arthur E. Bryson (1961). In 1962 Stuart Dreyfus derives a simpler form using the chain rule.
In 1986 Rumelhart, Hinton and Williams showed experimentally that this method can generate useful internal representations of incoming data in hidden layers of neural networks.

Backpropagation Algorithm

For number of epochs / until loss sufficiently low / loss no longer improves:

Wgradient = Evaluate_gradient (current_loss, data, W)

W += -α * Wgradient

α - a very critical parameter. What will happen if it is too big? And what will happen if it is too small?

The loss function:

There are many different loss functions. Commonly used examples are:

1. Classification: Categorical cross entropy
2. Regression: Mean square error

In order to use GDA we need a function that is differentiable!

In the perceptron example we used a simple delta function:

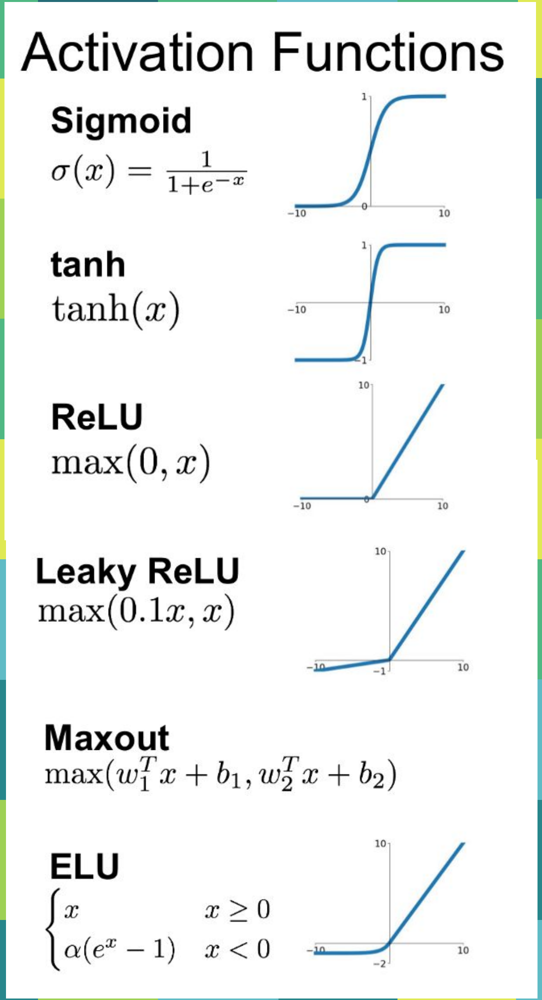
(true - predicted)

The activation function:

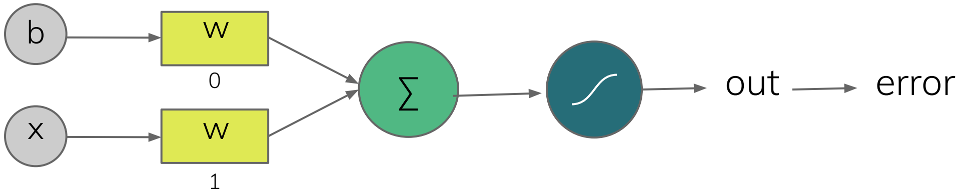
Also needs to be differentiable.

We used:

1. Classification: step function
2. Regression: ReLU



The Forward Pass



Calculate the output of the following inputs, calculate a delta error. First do a symbolic solution and then plug in the numbers.

X =0.1, b=?

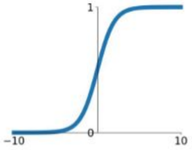
W=[0.2,0.1]

label=0.56

Sigmoid activation function:

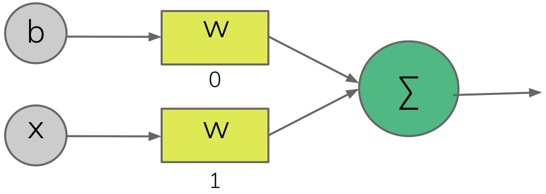
Sigmoid

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

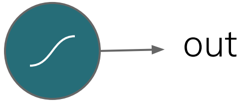


Forward Pass

$$\begin{bmatrix} w_0 & w_1 \end{bmatrix} \bullet \begin{bmatrix} b \\ x \end{bmatrix}$$



$$\text{Output} = 1/(1+e^{-W\square X})$$



$$\text{Error} = \text{Output-Label}$$

$$\text{Loss} = 0.5(\text{Error})^2$$

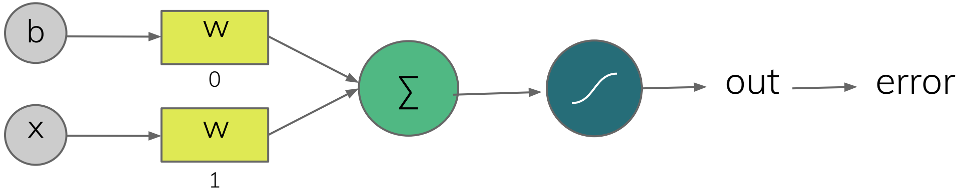
Backward pass - using gradient descent

We want to update each one of the weights such that we will move down slope on the error function. For that we use the gradient descent algorithm.

Output = $1/(1+e^{-W \cdot X})$

Error = label-Output = label - $1/(1+e^{-W \cdot X})$

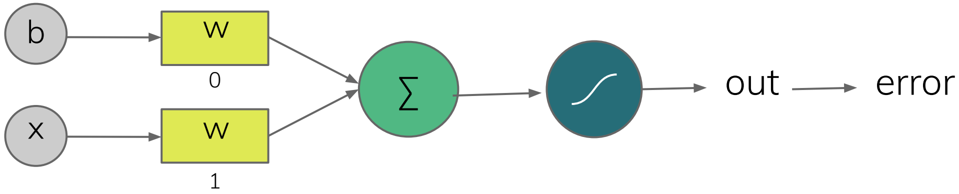
Backward pass



$$\frac{\partial loss}{\partial w_i} =$$

Remember the chain rule

Backward pass



$$\frac{\partial loss}{\partial w_i} = \frac{\partial loss}{\partial error} * \frac{\partial error}{\partial activation} * \frac{\partial activation}{\partial net} * \frac{\partial net}{\partial w_i}$$

Develop the equation for the updating of each one of the weights.

Backward pass

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Practical #2: GDA

Vanilla GD can be slow and wasteful

Stochastic Gradient Descent (SGD):

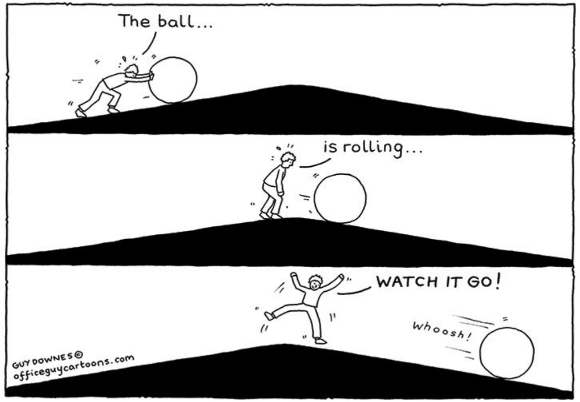
Instead of calculating the gradient on the entire training dataset, we calculate it on small **batches.**

 # implement SGD for the last regression problem.

Other modifications to GD

Momentum:

In physics it is a vector: p = mv (kg·m/s)



Other modifications to GD

Momentum:

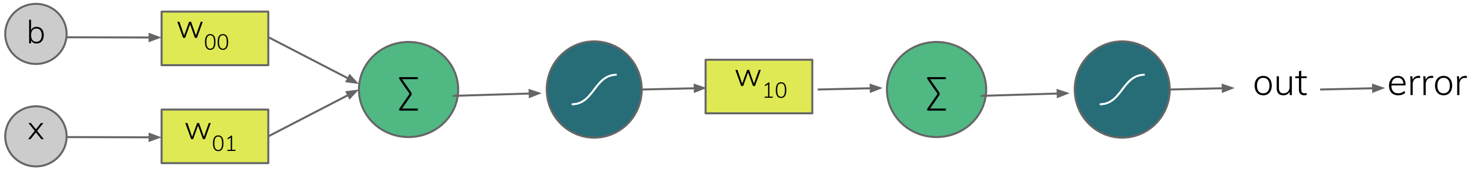
Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go. The equations of gradient descent are revised as follows:

Original: $W = W - \alpha \nabla \ W f(W)$

$V = \gamma V_{t-1} + \alpha \nabla \ W f(W)$

$W = W - V t$

Multi-layer perceptron (NN)



How will the backward pass look now?

Practical #2: Multi_layer_perceptron

There are infinite sets of weights that can yield a reasonable outcome. How do we make sure that the the set we end up with generalizes well (i.e. does well also on test set)??

Regulariztion

Regulariztion

1. Adding a regularization term to the loss function: L1/L2/Elastic Net
2. Explicit method: by adding dropout to the NN architecture
3. Implicit methods: data augmentation, early stopping of training

Regulariztion

1. Adding a regularization term to the loss function: L1/L2/Elastic Net

L2: $$R(W) = \sum_i \sum_j W_{i,j}^2$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W)$$

Neural Networks

How close are ANN to the real thing?

No Backward pass!

Our brain:

- ~ 10^{12} Neurons
- 1000-100000 inputs per neuron
- Information passes continuously, with many recurrent connections

NN:

- ~ 1000 Neurons
- 100-10000 inputs
- Information passes sequentially in a directional manner

Our brain:

- 20 watts
- 37℃

NN:

- Nvidia GeForce works in 250 watts
- Heats to 50-80℃

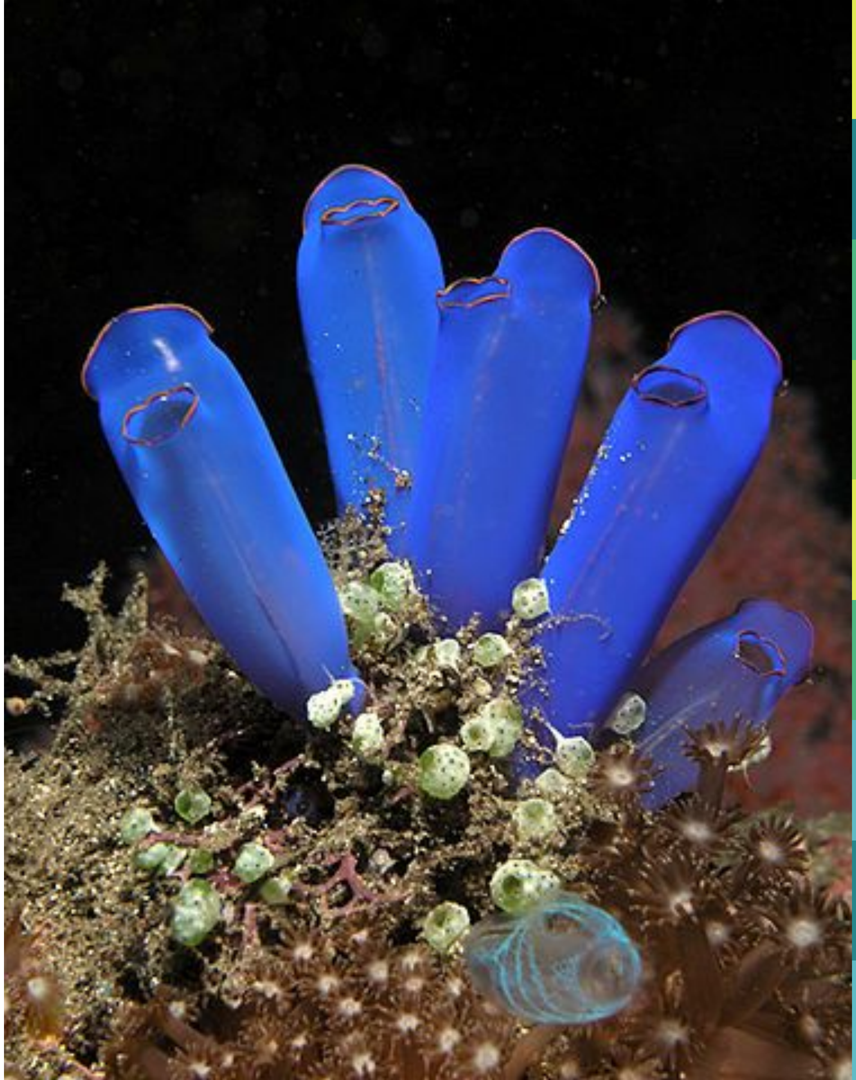
No Backward Pass!

“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.”**

HEBBIAN LEARNING

Evolution

Nervous systems developed when locomotion was developed.



Evolution

Nervous systems developed when locomotion was developed.

ANTICIPATION / PREDICTION

Reinforcement, rather than supervised:

Learning is reinforced in the brain by mechanisms that report:

- salience:
  A successful outcome (food/ goal achieved etc).
  Bad outcome (pain/threat etc).
- Surprise: an unpredicted outcome can mean novelty or change of pattern - thus learning mechanisms should be enhanced

Top Down :

https://www.youtube.com/watch?v=vJG698U2Mvo

Top Down :

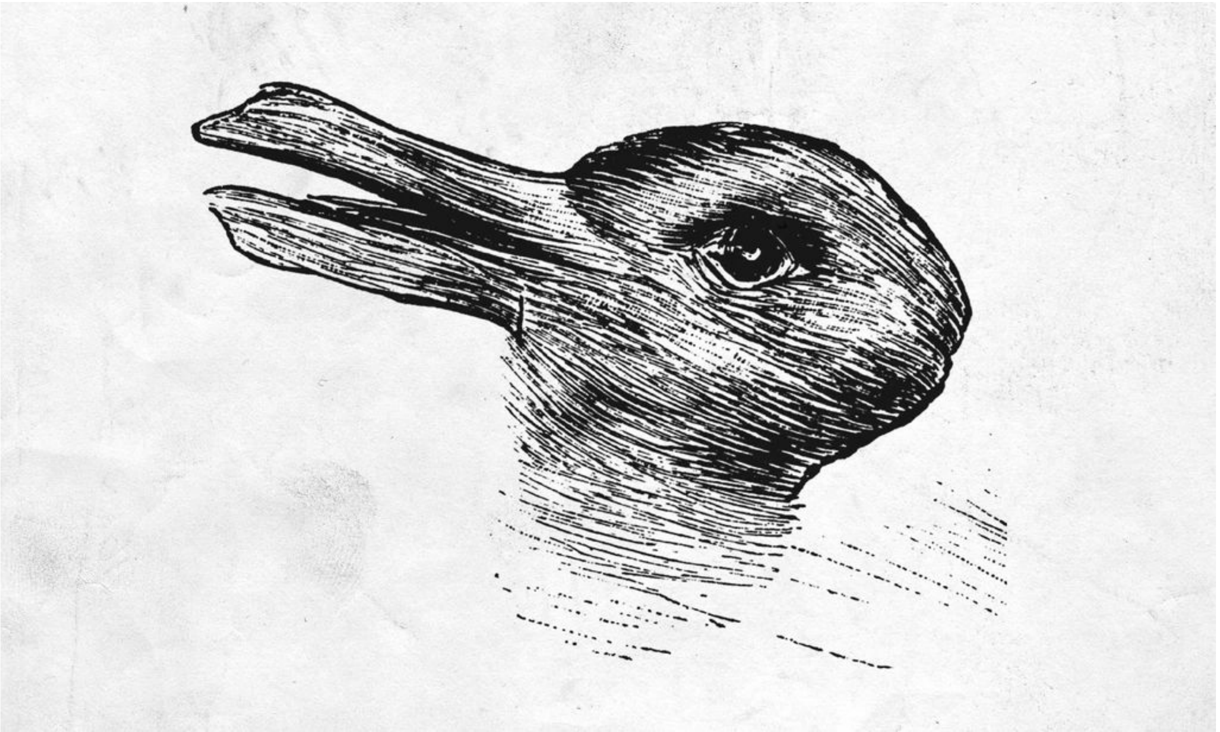
The brain is linked to behavior.

Depending on the wanted outcome the system can be brought to different states.

Brains can be conditioned:

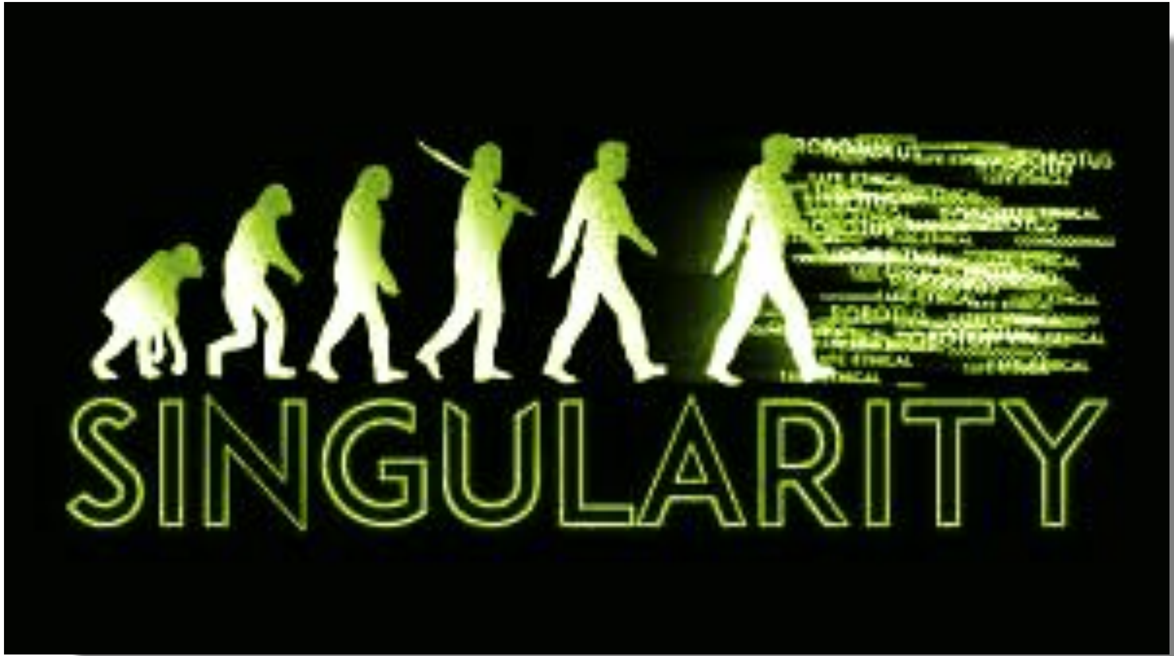
People are less likely to cheat in a test, if prior to the test they were given to read the 10 commandments (Dan Arieli “predictably irrational”)

Ambivalence - how sure are we?



Logic and reasoning

NN are not very good in learning logical rules (Fibonacci etc.)


SINGULARITY