BASICS OF NEURAL NETWORKS

Session: DL Fundamentals

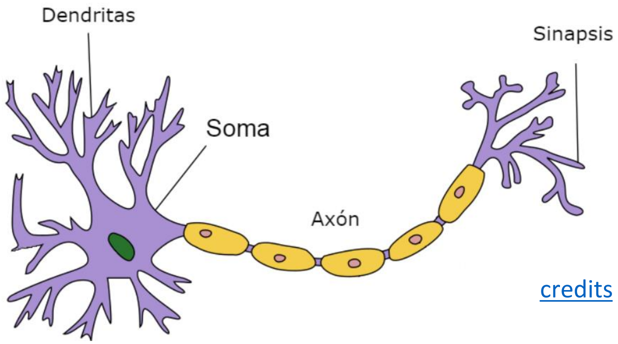
2023 SCHOOL AT THE IAA-CSIC

EDUARDO SÁNCHEZ KARHUNEN

DEPT. ARTIFICIAL INTELLIGENCE. UNIV. SEVILLE. SPAIN

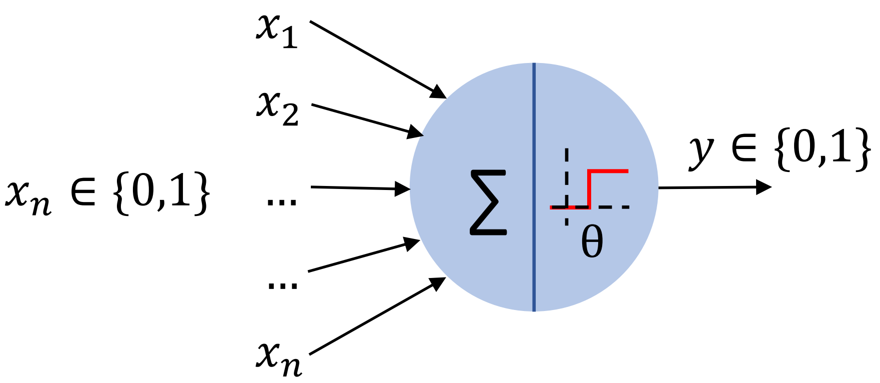
McCulloch-Pitts model (1943):

- First computational model of neurons.
- Idea: brain operation = logical functions composition.
- Model parameters: θ (hand-setting)



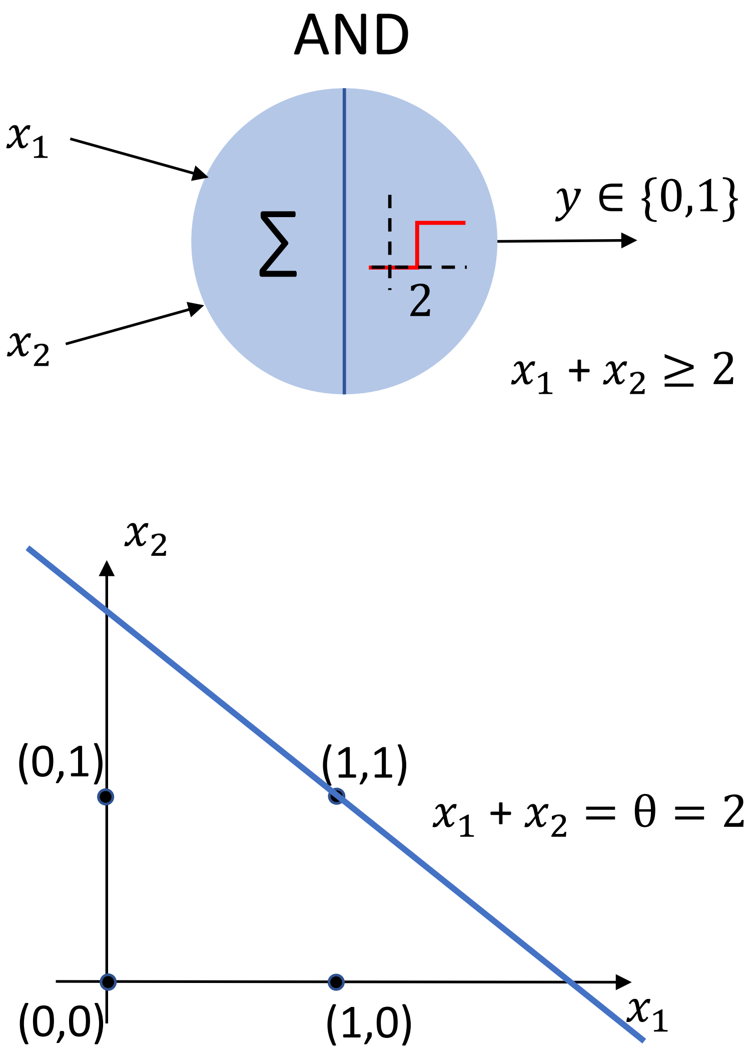
Basic unit reminds of the current ones:

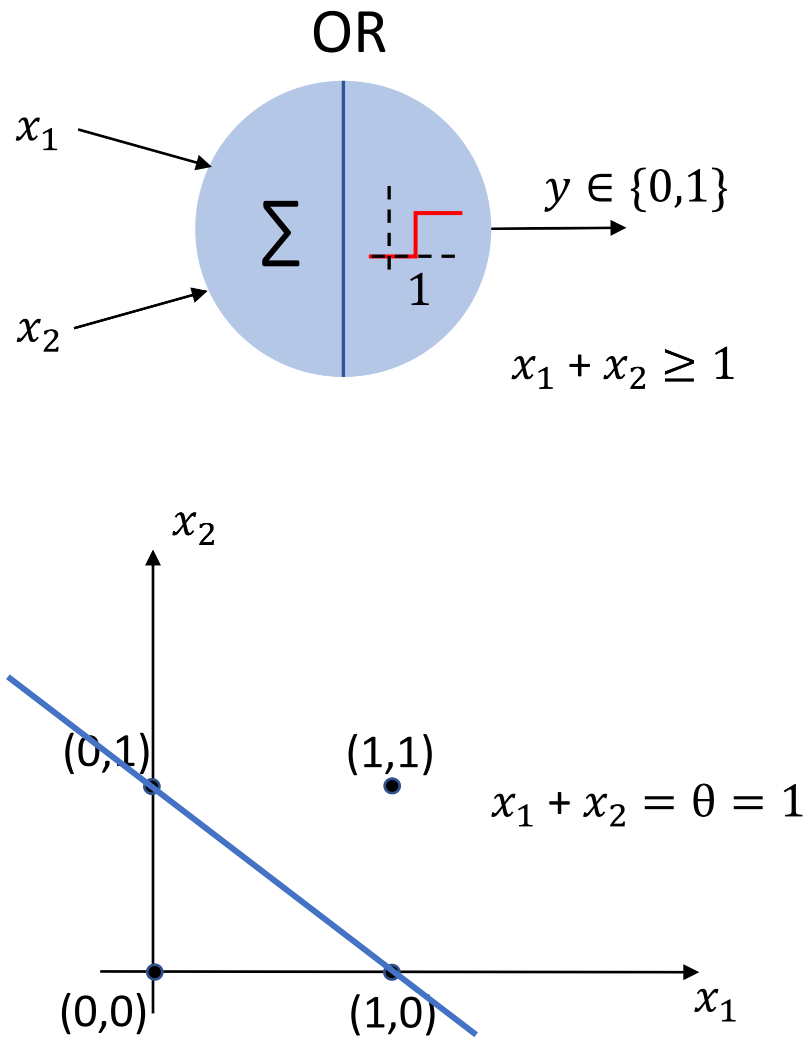
- Neuron inputs are boolean $x_i \in \{0,1\}$.
- Inputs: excitatory / inhibitory.
- Inputs are all aggregated.
- A zero-one decision is taken. Threshold θ.
- Trigger an output $y \in \{0,1\}$.



$$y = 1 \ \text{ if } \sum_{i=1}^{n} x_i \geq \theta \qquad y = 0 \ \text{ if } \sum_{i=1}^{n} x_i < \theta$$

McCulloch, W. S. y Pitts, W. H. (1943). *A logical calculus of the immanent in nervous activity*



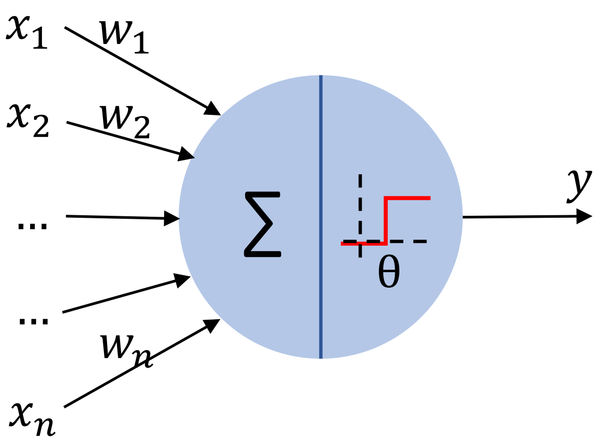


#parameters = 1. θ effect is a displacement in the plane

Perceptron - Rosenblatt (1958):

McCulloch-Pitts' model generalization:

- More realistic: inputs ϵ ℝ.
- Relative importance between inputs: w_i.
- The more inputs, the more complex to fix θ manually.
- Proposed neuron = almost "exactly" nowadays neuron.



$$y = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq \theta$$

$$y = 0 \text{ if } \sum_{i=1}^{n} w_i x_i < \theta$$

Generalization cost:

- Number of parameters increases: $M(\theta)$ => $M(\theta, w_i)$.
- (θ) fixed manually => (θ, w_i) needed a learning algorithm.

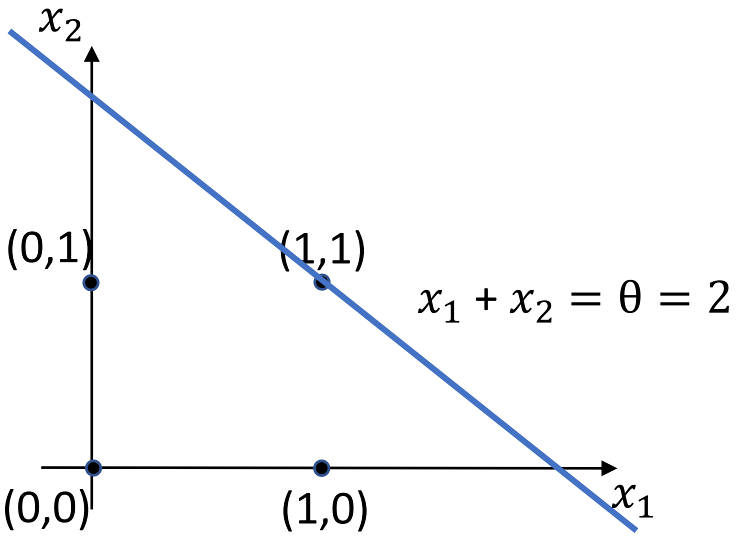
Rosenblatt, R. *The perceptron: A probabilistic model for information storage and organization in the brain*

LITTLE CHANGES – HUGE GEOMETRIC IMPACT

McCulloch-Pitch

$$y = 1 \ \text{ if } \sum_{i=1}^{n} x_i \geq \theta$$

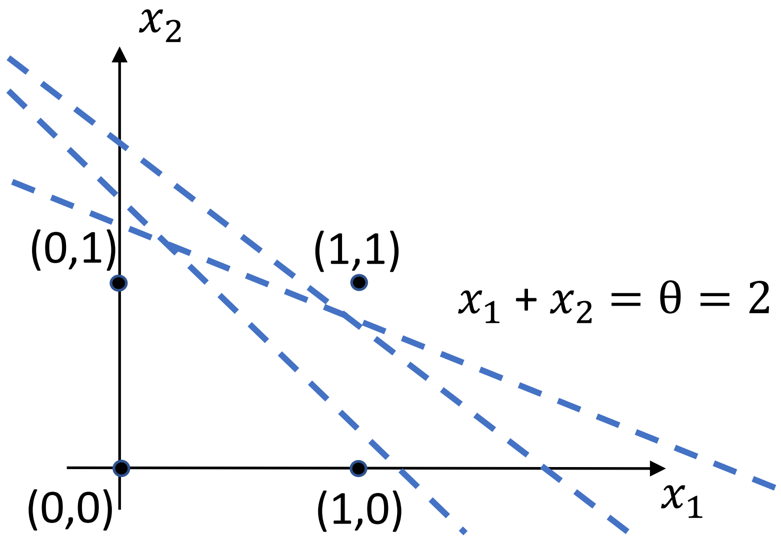
$$y = 0 \ \text{ if } \sum_{i=1}^{n} x_i < \theta$$



Perceptron

$$y = 1 \ \text{ if } \sum_{i=1}^{n} w_i x_i \geq \theta$$

$$y = 0 \ \text{ if } \sum_{i=1}^{n} w_i x_i < \theta$$



Meaning of learning:

- Dataset: tuples $\{(x_i, y_i),\ i = 1 \dots N\}$.
- Goal: obtain model parameters (θ, w_i).
- s.t: classify properly the whole input dataset.

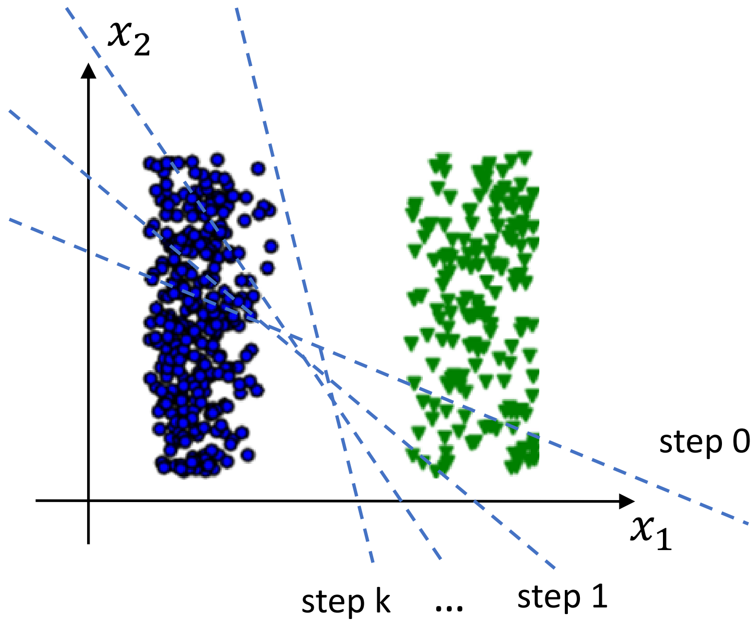
$$\sum_{i=1}^{n} w_i x_i > \theta \Rightarrow \sum_{i=1}^{n} w_i x_i - \theta > 0$$

$$\sum_{i=1}^{n} w_i x_i - \theta * 1 > 0 \Rightarrow \sum_{i=0}^{n} w_i x_i > 0$$

con x_0=1, w_0= θ

Perceptron algorithm convergence theorem:

- θ is considered as a weight $\boldsymbol{w_0}$.
- Iterative adjustment of weight vector.
- Based on same distance between prediction and true value.
- If dataset is linearly separable: converges in a finite number of steps.

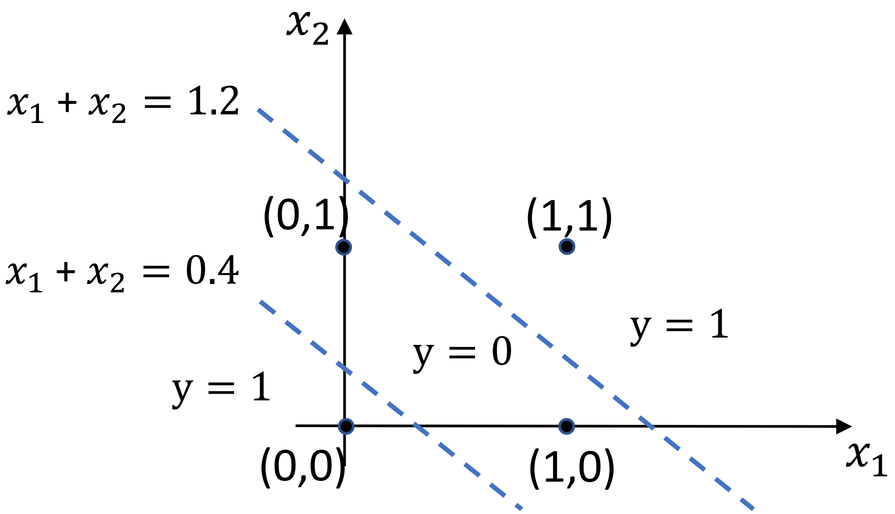


Minsky & Papert (1969):

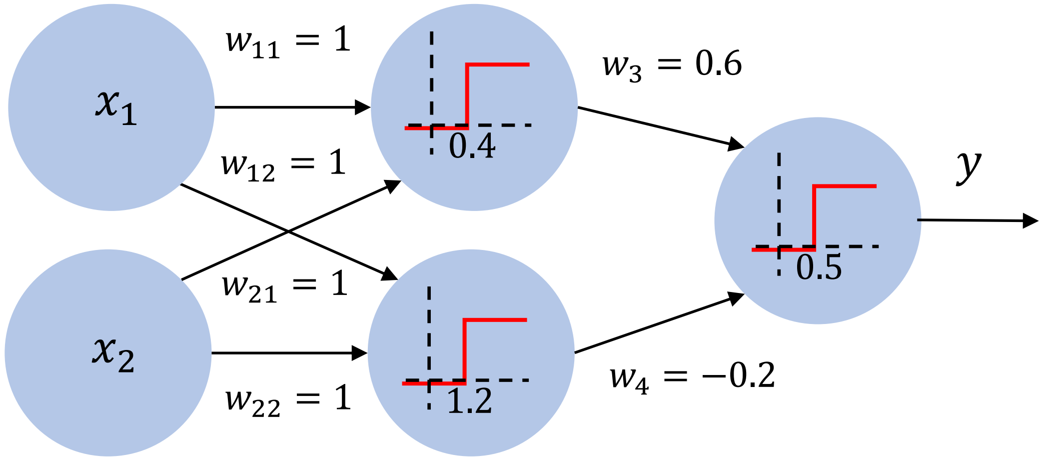
Describe perceptron limitations:

- Not able to capture a simple XOR logical function.
- Solution: add more layers (stacking).
- Problem: there was no training techniques for multilayer networks.

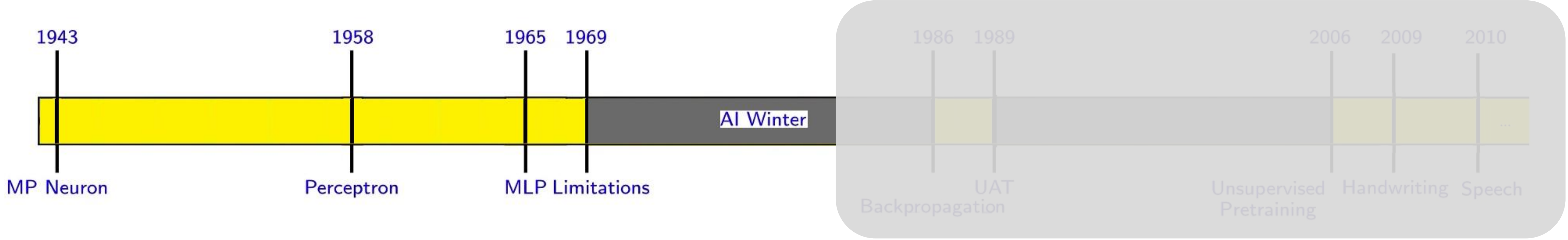
x_1	x_2	XOR
0	0	0
1	0	1
0	1	1
1	1	0



Minsky, R. *Perceptrons: An introduction to computational geometry*



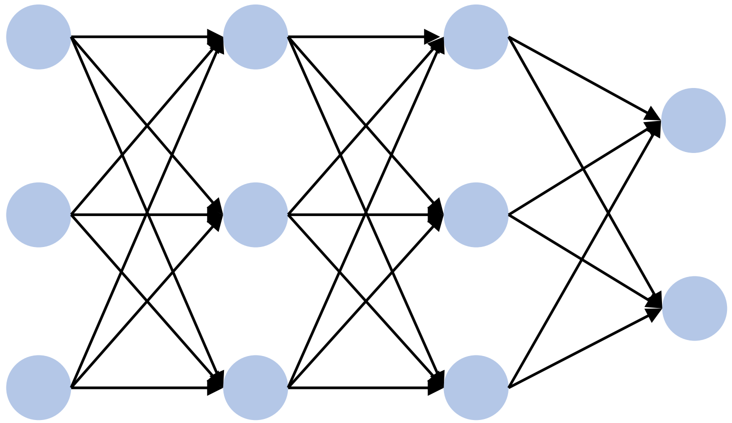
Simplifying notation: remove sum operator



- Obstacle race:
  - DL has not only been an accumulation of advances.
  - Alternation of winters and hypes:
    - hypes: periods of great optimism, huge expectations and advances.
    - winters: expectations are not met, starting periods of pessimism. Reduction of investments and scientific community leaves this line of research.

AI Winter:

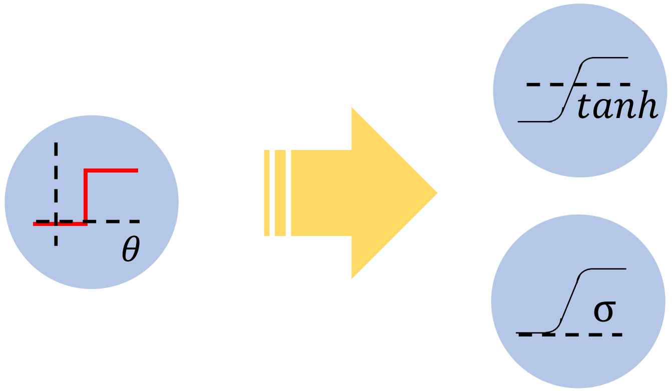
- Minsky & Papert criticism.
- Worldwide pessimism: ANN have no future.
- ANN are abandoned for years (1969-1986).



Grail = train multilayer networks

- Proposed by Werbos (1982) in his PhD.
- Popularized by Rumelhart(1986):
  - Backpropagation + Gradient descent.
- Training technique used nowadays in 99.9% of cases.

Collateral impact: all elements must be derivable: new activation functions.

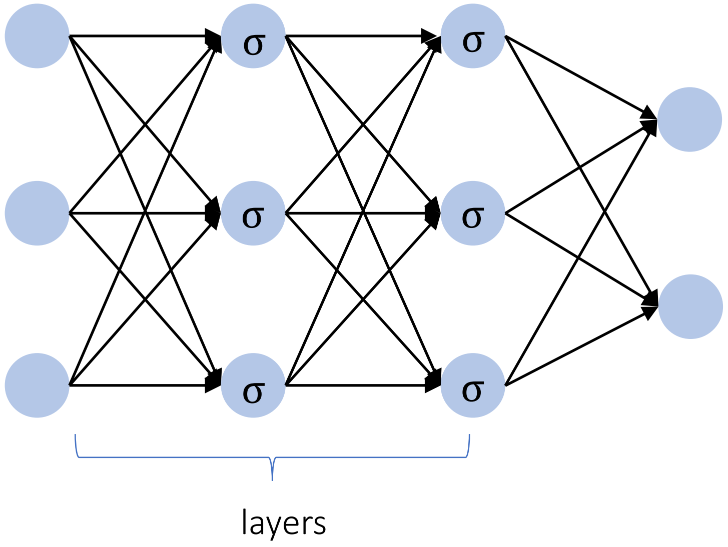


Rumelhart, R., Hinton, G. & Williams. (1986). *Learning representations by back-propagating errors*

MLP (Multi-Layer Perceptron)

- We had all ingredients:
  - Multilayer (layer stacking) is needed (1969)
  - Training algorithm was proposed (1986):
    - Backpropagation + Gradient descent
  - A new hype begins

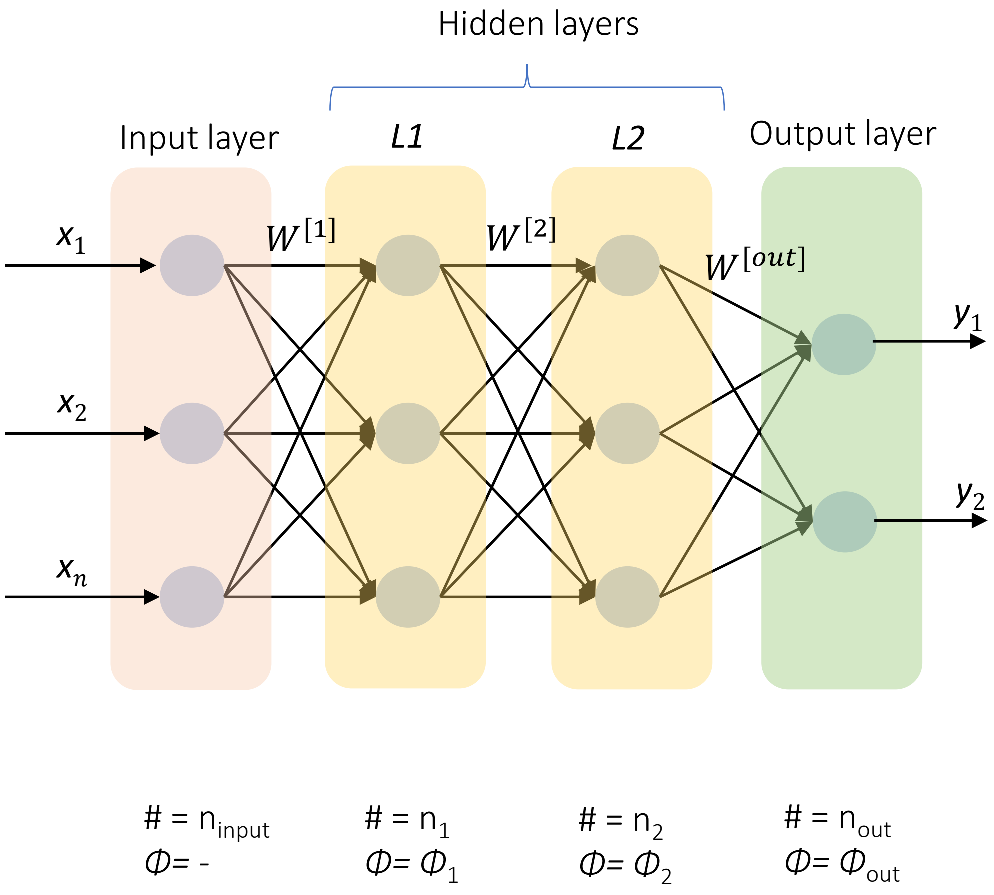
Multilayer perceptron



- First NN = Feedforward NN:
  - Groups of perceptrons (Rosenblatt neurons): arranged in layers.
  - Signal flows only in one direction: “no cycles”.
  - All neurons of a layer are interconnected with all neurons of the next layer.
  - Data is injected in the network as vectors.

Layers and its parameters:

- Input layer:
  - Data injected to the network as vectors.
  - # neurons = # components input vector X.
- Dense layers:
  - $W^{[i]}$: weight matrix of layer i.
  - b_i: bias vector (a weight matrix column).
  - Φ_i : activation function (typically common to the layer).
- Hidden layers:
  - Do not "see" directly the input vector X.
  - A network with multiple hidden layers is called Deep.
- Output layer:
  - # neurons depends on each problem to solve.



- Math implications:
  - Matrix multiplication + activation function application

$$Y = \phi_2(W^{[2]}A_1) = \phi_2(W^{[2]}\phi_1(W^{[1]}X)) = (\phi_2 \circ W^{[2]} \circ \phi_1 \circ W^{[1]})\,X$$

NN training interpretation

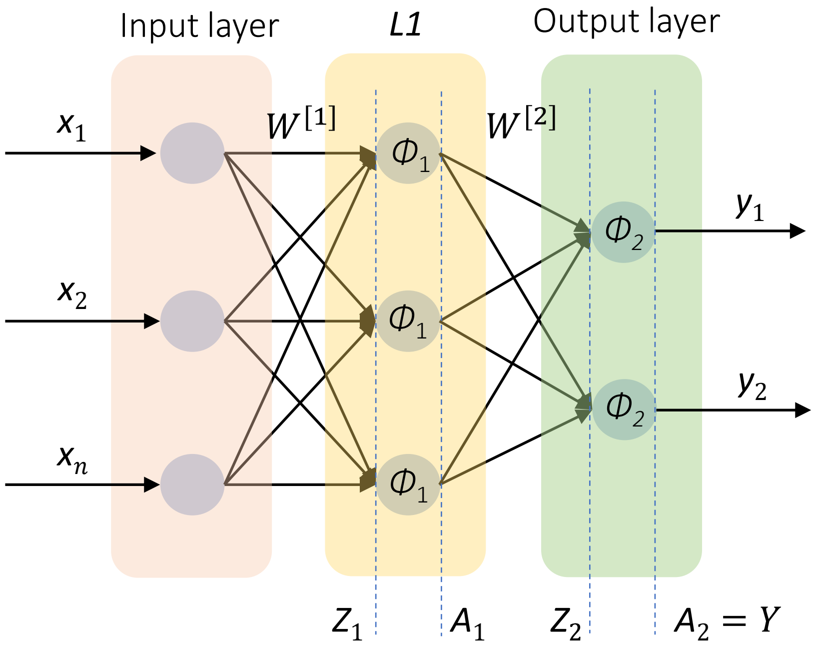
ϕ_1 and ϕ_2 nothing to learn: are prefixed

Find values of matrices: $W^{[1]}$ and $W^{[2]}$

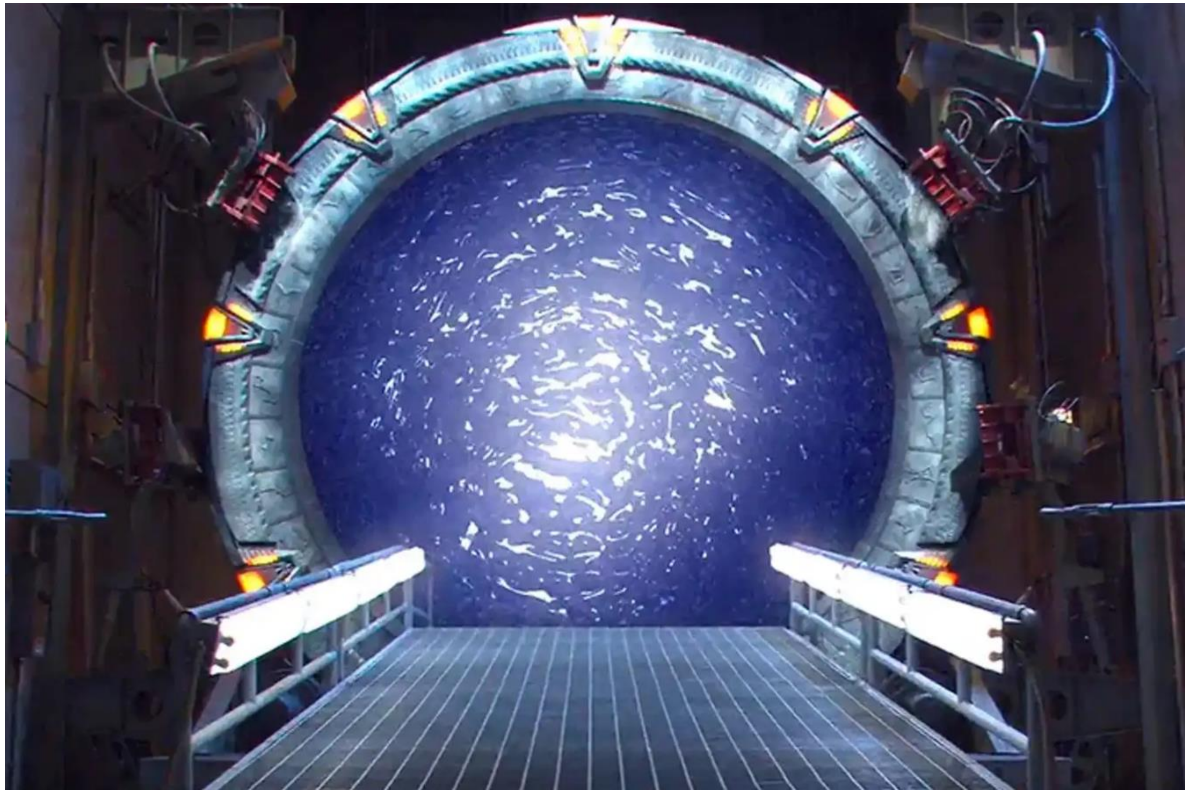
- Activation function key role:
  - Composition of linear functions = linear function
  - Can only learn linear "things":

$$Y = W^{[2]}A_1 = W^{[2]}(W^{[1]}X) = (W^{[2]}W^{[1]})\,X$$

  - To learn non-linear problems = break linearity
  - Activation function = Non-linearity



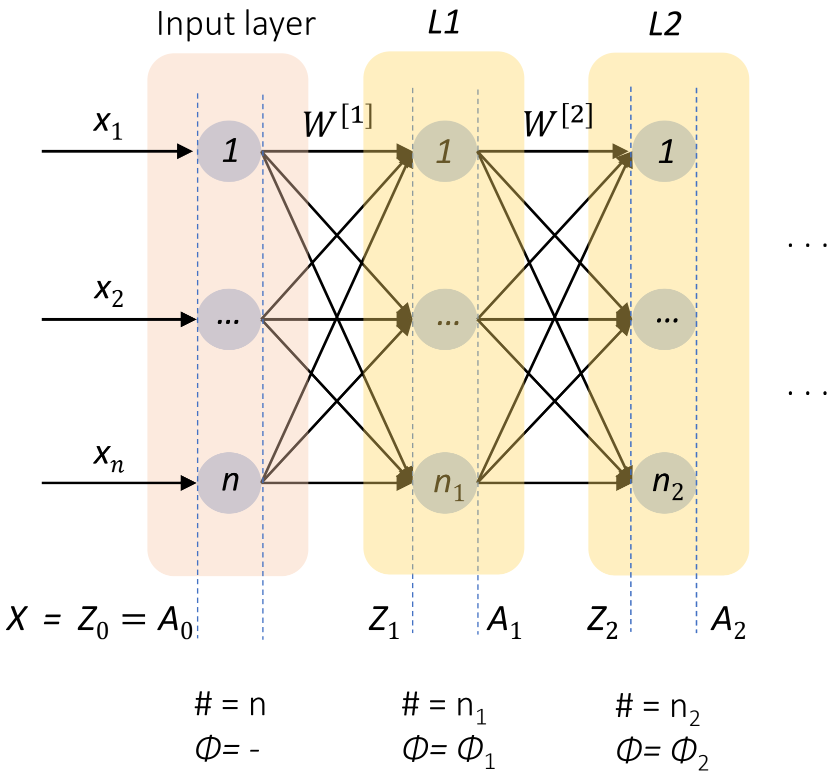
- The real role of a layer = interdimensional portal



credits

MATRIX OPERATIONS ARE NOT EASY TO INTERPRET

- Think of dimensional jumps:
  - One side of the dimensional gate (Z1):
    - Layer receives a vector of n_1 components from the previous layer.
    - A representation of the problem in a n_1-dim space.
  - The other side of the dimensional gate (A1):
    - The layer "performs" its operations: sums and non-linearities.
    - Outputs a vector with n_2 components (# neurons of the layer)
    - A new representation the problem in a n_2-dim space.

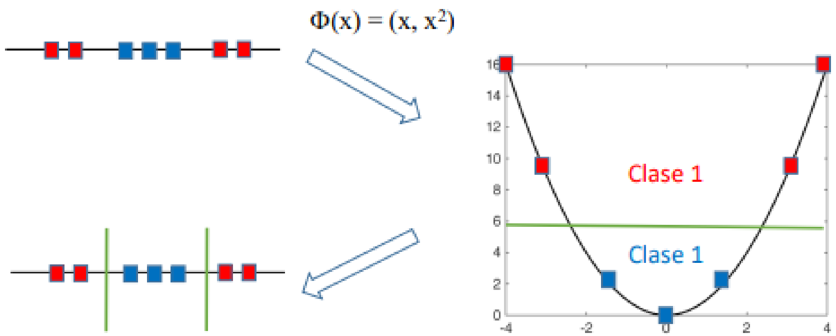


WHY CHANGE PROBLEM DIMENSION?

In SVM (Support Vector Machines):

- Strategy facing on non-linearly separable dataset:

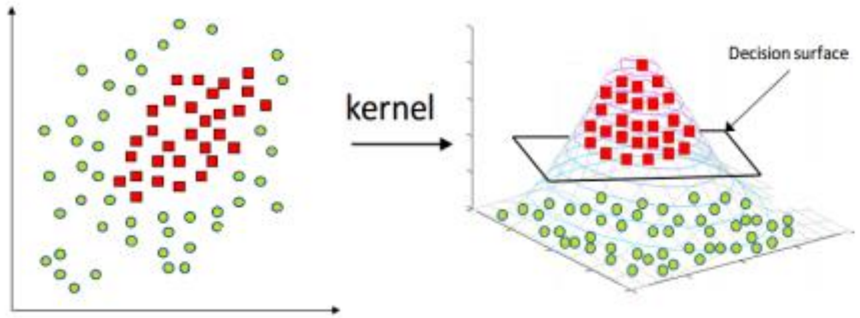
Dimensionality expansion



- A "Catalogue" of predetermined kernel functions (Gaussian, …)
- By Trial and error, the best kernel is selected.

In Neural Networks:

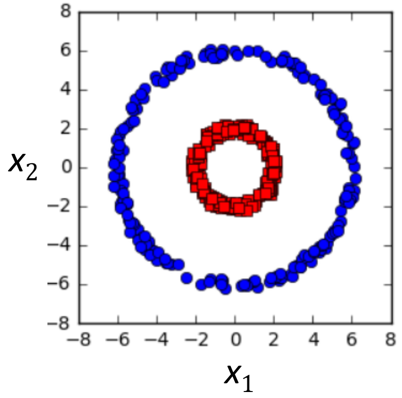
- A generalization of this idea
- Each layer learns the necessary type of expansion.
- Even more, the one that better fits to each concrete problem

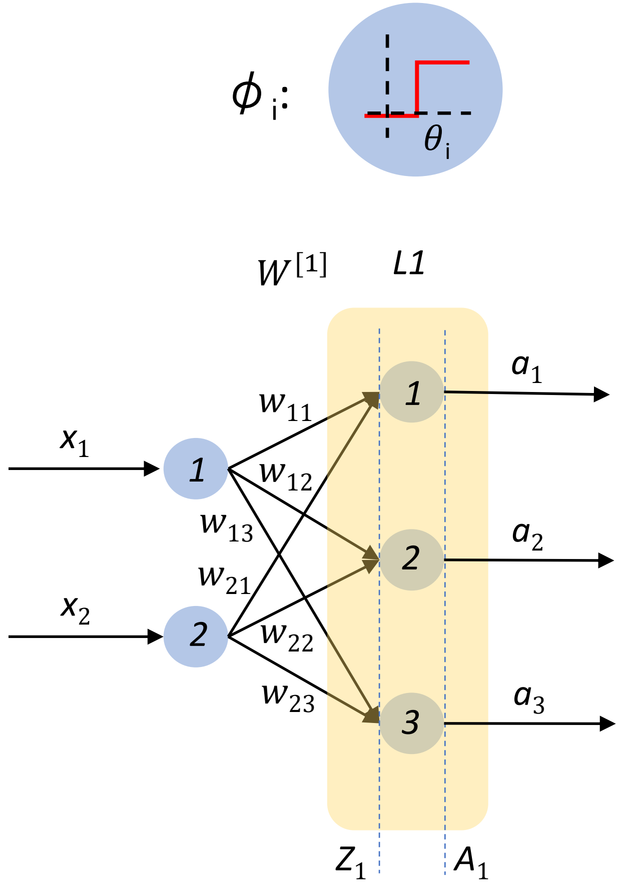


credits

LAYER AS A TRANSFORMATION

2D-space



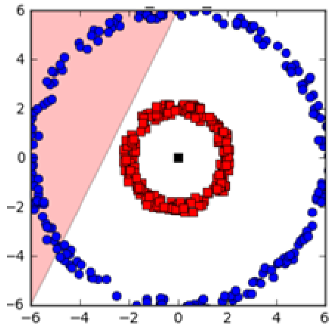


\# = n_{input}
Φ= -

\# = n_1
$\Phi = \Phi_1$

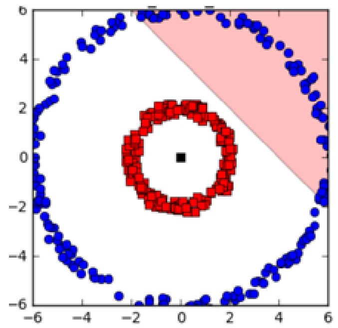
neuron 1

$$\left.\begin{aligned} w_{11} &= 1 \\ w_{21} &= 1 \\ b_1 &= 6 \end{aligned}\right\} x_1 + x_2 > 6$$



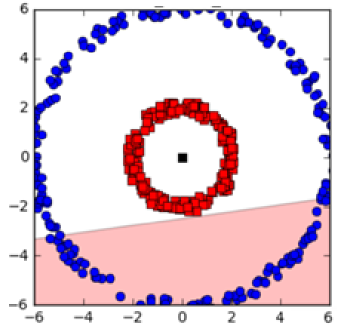
neuron 2

$$\left.\begin{aligned} w_{12} &= 1 \\ w_{22} &= 1 \\ b_2 &= 4 \end{aligned}\right\} x_1 + x_2 > 4$$



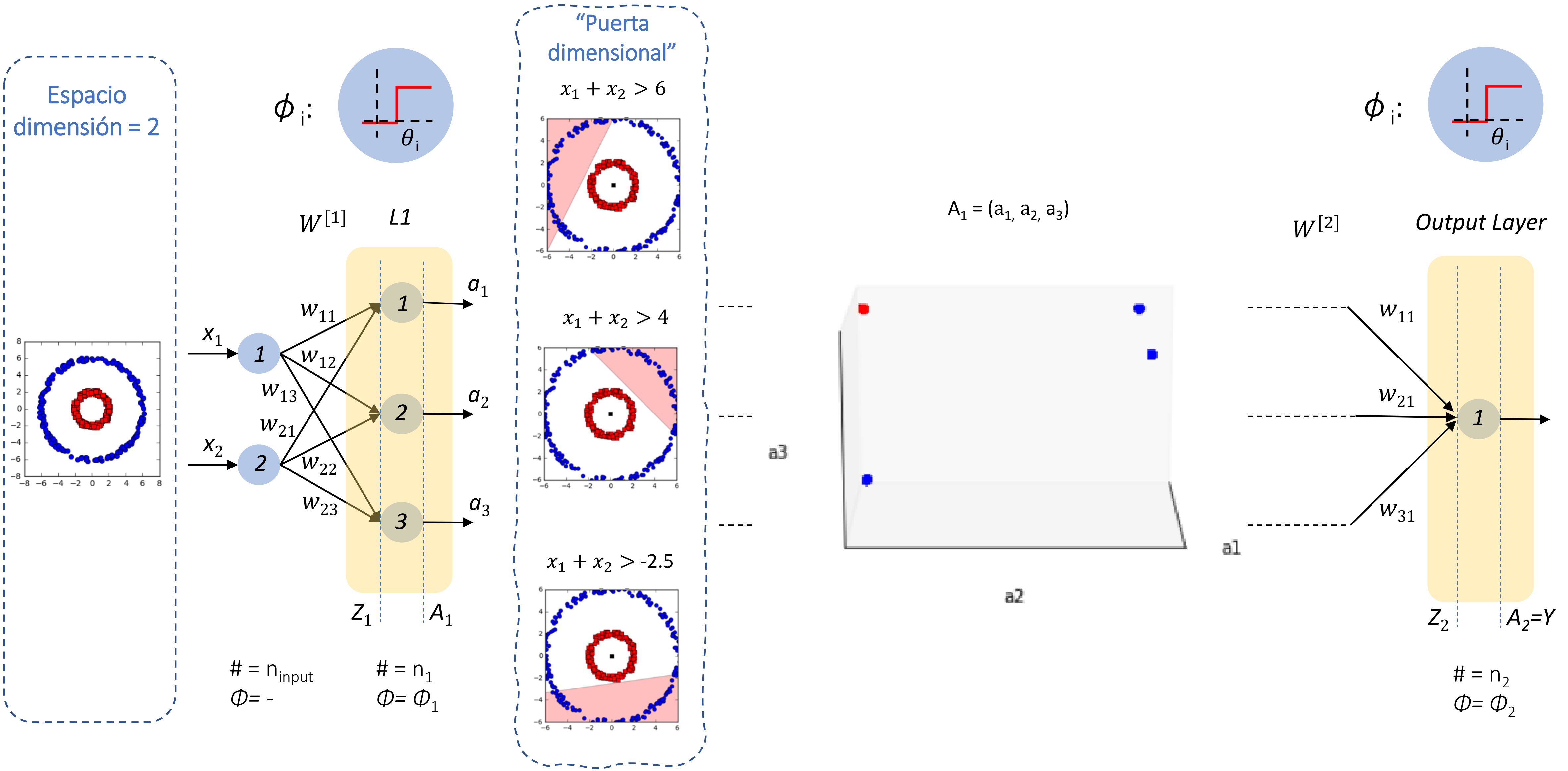
neuron 3

$$\left.\begin{aligned} w_{13} &= 1 \\ w_{23} &= 1 \\ b_3 &= 2.5 \end{aligned}\right\} x_1 + x_2 > \text{-}2.5$$



credits

NOW DECISION IS EASY



Hierarchical learning:

- Each layer builds a new “vision of the world” based on the previous layer vision.
  - Each neuron makes a question yes/no based on the representation received
  - The number of neurons in a layer (n_i) = the number of yes/no question
  - Dimension of the new representation of the input dataset (n_i –dimension), n of neurons
- NN has learnt a hierarchical representation of the dataset useful to solve a specific task.

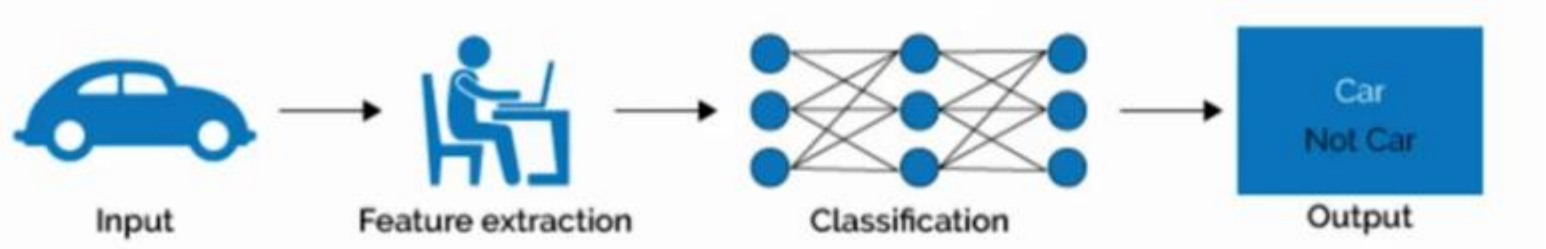
Knowledge stored:

  - Network architecture (#layers, #neurons per layer)
  - Model params $\{W^{[i]}\}$
- Transfer learning: knowledge obtained solving a task could be reused to solve a different but related task

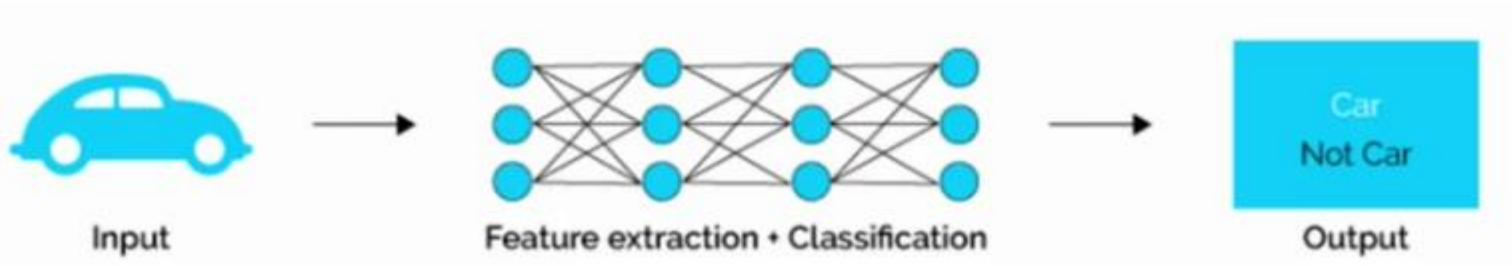
INTERNALIZED FEATURE ENGINEERING

- Pipelines comparison: (e. g. image classification)

TRADITIONAL ML



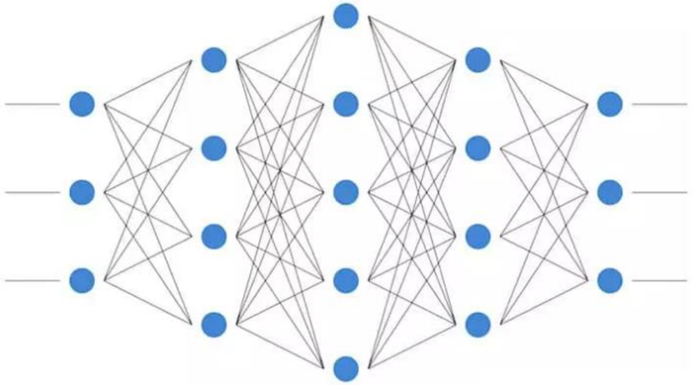
NEURAL NETWORKS



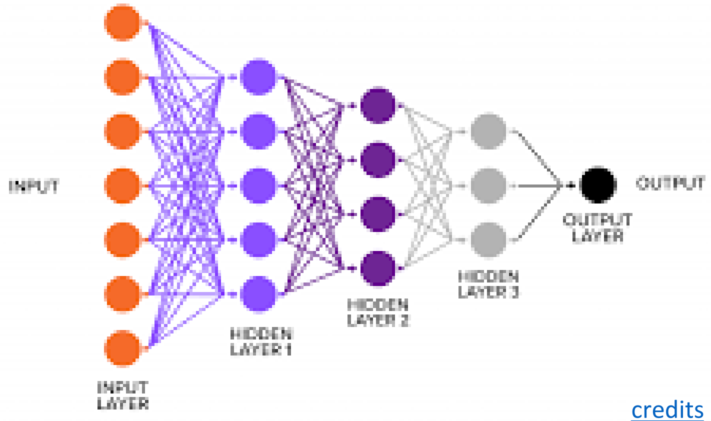
[credits]

FORTUNATELY: IT IS STILL AN ART

- Heuristics:
  - Starting from a low-dimensional dataset:
    1. To expand dimensionality = increasing #neurons in the following layers.
    2. Once reached a high enough dimensional representation.
    3. Reduce dimensionality force “learning” key features.
    4. Typically, dimension is reduced gradually.
  - Starting from a high-dimensional dataset:
    1. There is an excess of information.
    2. From the beginning, reduce dimensionality step by step.
    3. Typically: in image problems.



credits

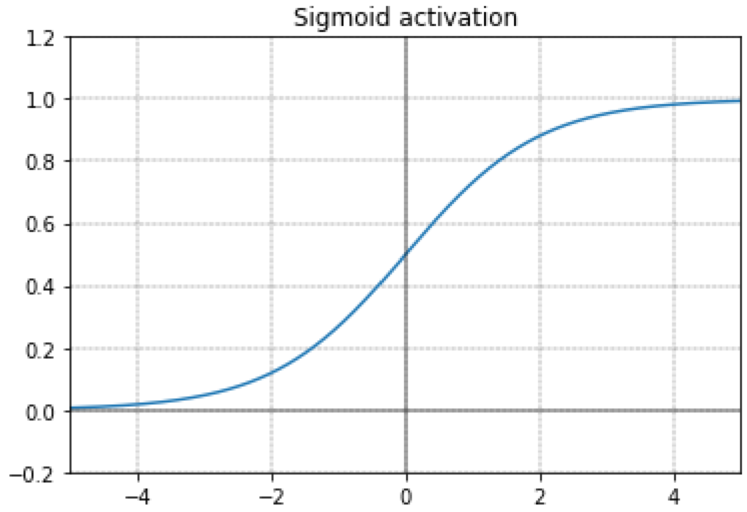


credits

OUTPUT LAYER = f(problem to tackle)

Binary classification:

- Classical example: dog / cat
- # neurons = 1.
- Desired output interpretation: $\mathrm{P}(\mathrm{A}_{\mathrm{out}} = \mathrm{Y} = 1)$
- Activation function: sigmoid. $\phi_{out} = \sigma(z)$



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

Multiclass classification:

- Classical example: dog / cat / horse
- # neurons = # classes
- Desired neuron *i* output interpretation: $\mathrm{P}(\mathrm{A}_{\mathrm{out}} = \mathrm{Y} = i)$
- To give a probability interpretation: $\sum_{i=1}^{n} P(\mathrm{A}_{\mathrm{out}} = Y = i) = 1$
- Activation function: softmax. $\phi_{out} = softmax(z)$

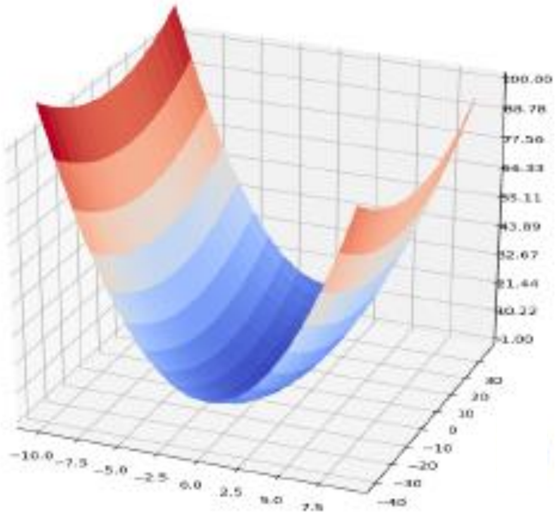
$$softmax(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

a) If $z_i < 0$, then $e^{z_i} > 0$

b) Values normalization $\sum_{i=1}^{n} e^{z_i}$

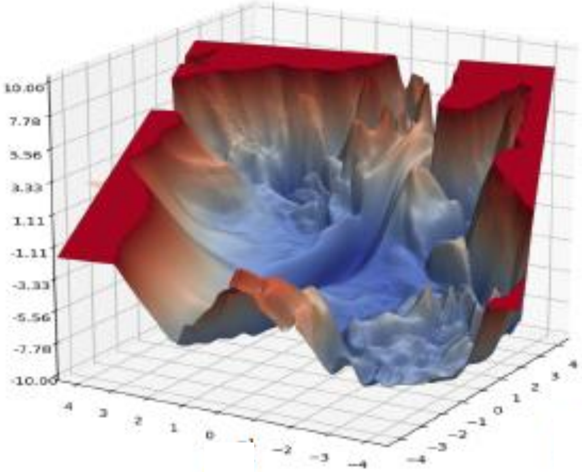
Meaning of network training?

- Once fixed the network architecture:
  - # layers & # neurons per layer.
  - Activation function of each layer.
- Given a problem: a dataset $\{(x^{(i)}, y^{(i)}),\ \ i = 1, \dots m\}$
- Given an error measure: loss function $J(\boldsymbol{x};\theta) = J(\boldsymbol{x}; W^{[i]})$
- Goal: find values of parameters = weights $\theta = W^{[i]}, i = 1, .. n$

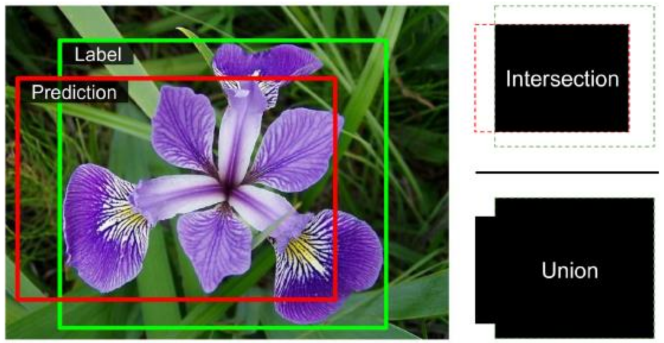


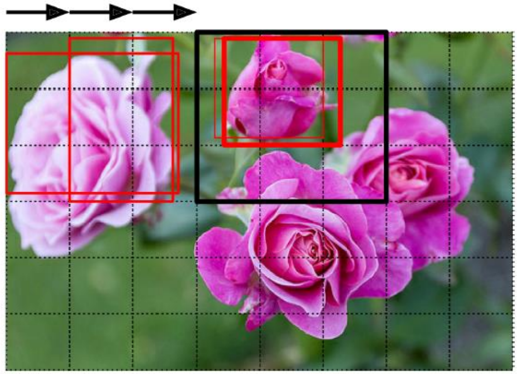
Drawbacks:

- J is non-convex: due to the non-linearities (activation functions).
  - Probably only local minima will be found.
- Search in a parameter space of dimension 10^4-10^7.



- Depends on the problem:
  - Task 1: Image location (Bounding box to locate object).
    - Problem: multi-regression, predict rectangle corners.
    - Loss function: Intersection-over-Union (IoU)
  - Task 2: Object detection.
    - Problem: detect if object appears in an image or not.
    - Loss function: Mean Average Precision (mAP)
  - Task 3: Image segmentation
    - Problem: associate to each pixel a class label.
    - Loss function: Pixel-wise cross entropy
  - Task 4: Image classification
    - Problem: associate a class to each image
    - Loss function : CrossEntropy



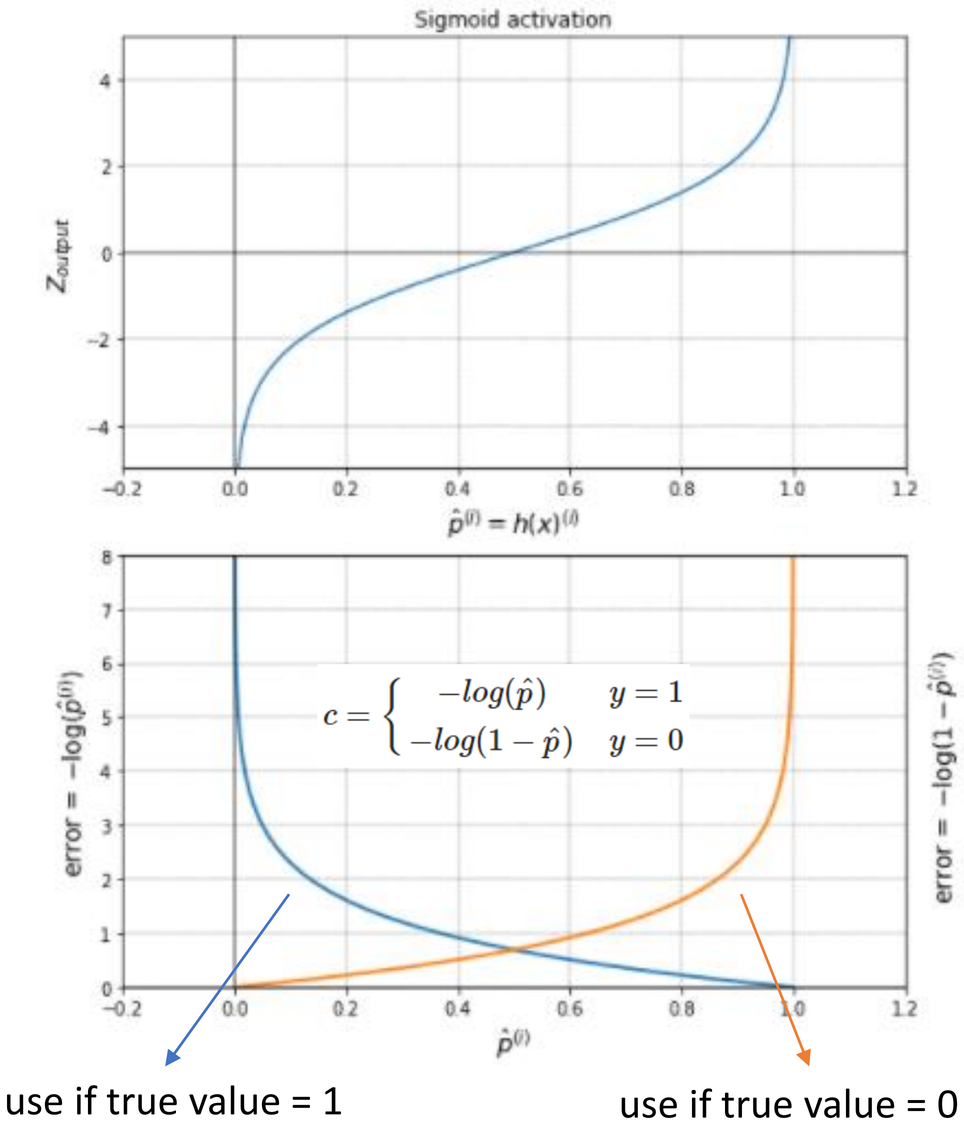


CLASSIFICATION PROBLEMS: CROSSENTROPY

- Why such a strange expression?
  - Key: network outputs are a probability.
  - How measure errors?
    - If true value = 1. If $P(Y = 1) \approx 0$, must penalize.
    - If true value = 0. If $P(Y = 1) \approx 1$, must penalize.
  - Combine both branches:
    - Each branch is weighted using $y^{(i)}$ and $1 - y^{(i)}$.
  - Expression can be generalized easily for > 2 classes.

- Total loss function:

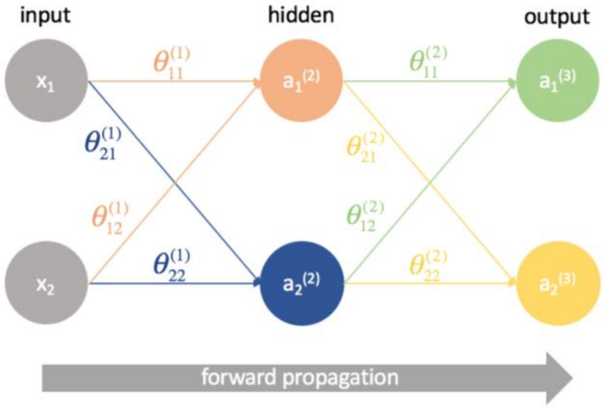
$$J = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \cdot log(\hat{y}^{(i)}) + (1 - y^{(i)}) \cdot log(1 - \hat{y}^{(i)})]$$



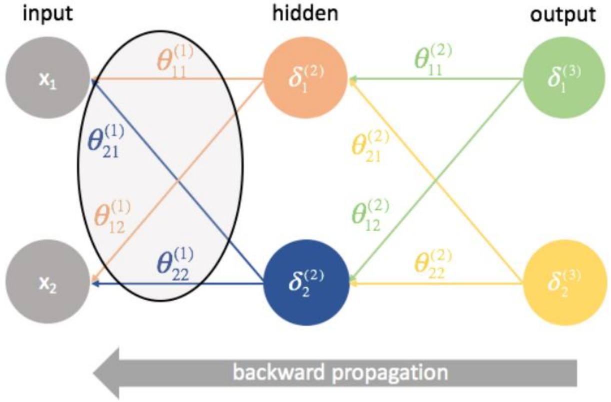
NETWORK TRAINING: BACKPROPAGATION + GD

Backpropagation (Rumelhart, 1986)

- Forward pass (or forward propagation):
  - For each input data, network predicts a probability.
  - An error is calculated between prediction and true label.
  - If more than an input data, average error for all inputs



- Reverse pass (backward propagation):
  - Obtain error gradient w. r. t. all weights: $\nabla J = \frac{\partial J}{\partial W_i}$
  - Making use of the derivative chain-rule: backpropagated errors traversing the NN.

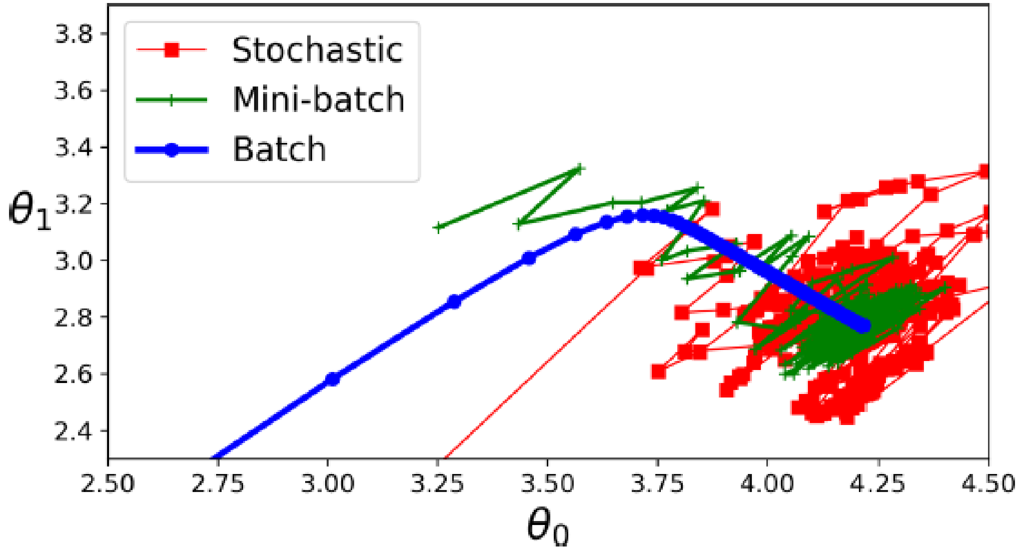


- Gradient descent: $\Delta W = -\eta \nabla J$
  - Adjust weights in the direction that maximizes the reduction of loss
  - Learning rate modulates the weights adjustment speed.

- Key point:
  - Weights are updated considering gradient of error
  - To obtain this mean error in the forward pass. ¿How much input samples are considered?

Error calculus approaches:

- Batch: all samples are considered.
  - Weight update algorithm has access to the complete info.
  - Very slow, more stable towards the local minima.
- Stochastic GD (SGD): a unique data instance is considered.
  - Weight update algorithm has access to a strongly biased info.
  - High variance in the obtained gradients.
- Mini-Batch: a random sample is considered.
  - Reduces variance, with a more stable convergence.
  - New hyperparameter: Batchsize (32, 64, 128, …).



WHAT IS HAPPENING HERE? IT DOES NOT WORK

- Against all odds:
  - During training: unexpected problems appeared.
  - Worst of all: unknown problems source.

2nd AI Winter starts
1986 - 2010

Problem	Solution
Lack of training data	Wait until digital revolution
Lack of computing power	Development of GPU, TPU, …
Strong dependence on the **value of η** (learning rate)	Learning rate schedules
Gradient descent is slow	Faster optimizers
Gradient instabilities	Novel activation functions
	Weight initialization techniques
	Batch normalization
	Gradient clipping

FASTER OPTIMIZERS

- Gradient Descent = slow!!

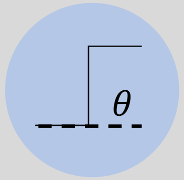
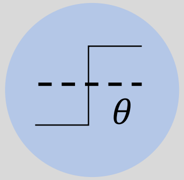
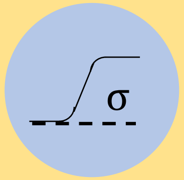
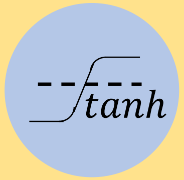
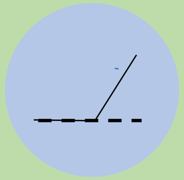
$$\Delta W = -\eta \nabla J$$

∇J

Class	Convergence speed	Convergence quality
SGD	*	***
SGD(momentum=...)	**	***
SGD(momentum=..., nesterov=True)	**	***
Adagrad	***	* (stops too early)
RMSprop	***	** or ***
Adam	***	** or ***
Nadam	***	** or ***
AdaMax	***	** or ***

η

ACTIVATION FUNCTION EVOLUTION

Period	Visualización	Name	$\phi(z)$	Características	Current use
McColluch-Pitts (50s)		Step function (Heaviside)	$= \begin{cases} 0, z < \theta \\ 1, z \geq \theta \end{cases}$	No derivable	Uso teórico
		Sign function	$= \begin{cases} -1, z < \theta \\ +1, z \geq \theta \end{cases}$	No derivable	Uso teórico
Backpropagation (90s)		Sigmoid/logistic	$= \dfrac{1}{1 + e^{-z}}$	Cálculo lento	Last layer
		Tanh	$= 2\sigma(2z) - 1$	Cálculo lento	Last layer
Actualidad (<2015)		ReLU (Rectified Linear Unit)	$= \max(0, z)$	Fast train	Hidden layer

29