Agenda:

- Session 1: Deep Learning fundamentals
- Session 2: CNN fundamentals
- Session 3: Practical considerations about CNN
- Session 4: Evaluation

2025 SCHOOL AT THE IAA-CSIC

EDUARDO SÁNCHEZ KARHUNEN (fesanchez@us.es)

DEPT. ARTIFICIAL INTELLIGENCE. UNIV. SEVILLE. SPAIN

BASICS OF NEURAL NETWORKS

Session: DL Fundamentals

2025 SCHOOL AT THE IAA-CSIC

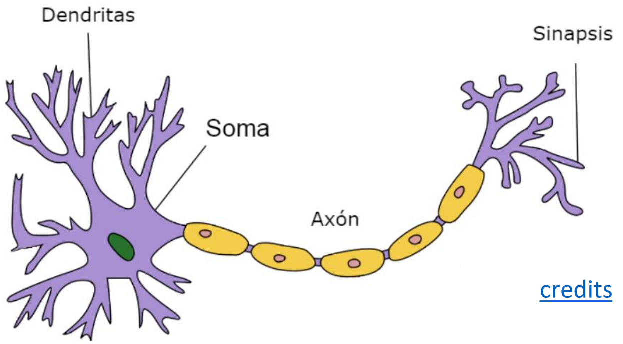
EDUARDO SÁNCHEZ KARHUNEN

DEPT. ARTIFICIAL INTELLIGENCE. UNIV. SEVILLE. SPAIN

Neuroscience – The Origin

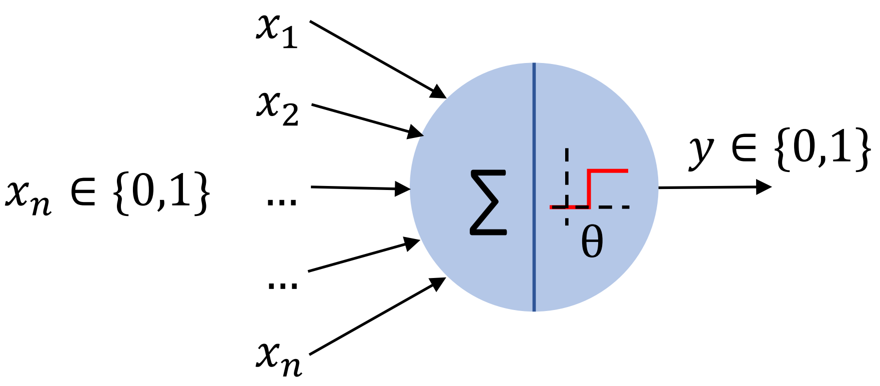
McCulloch-Pitts model (Chicago,1943):

- First computational model of a neuron.
- Inspiration: brain operation = composition of logical functions.



How it works:

- Inputs: boolean.
- Aggregation: all inputs summed together.
- Neuron "fires or triggers" an output: takes a decision.
  - Output: 1/0. If sum reaches a certain threshold θ.



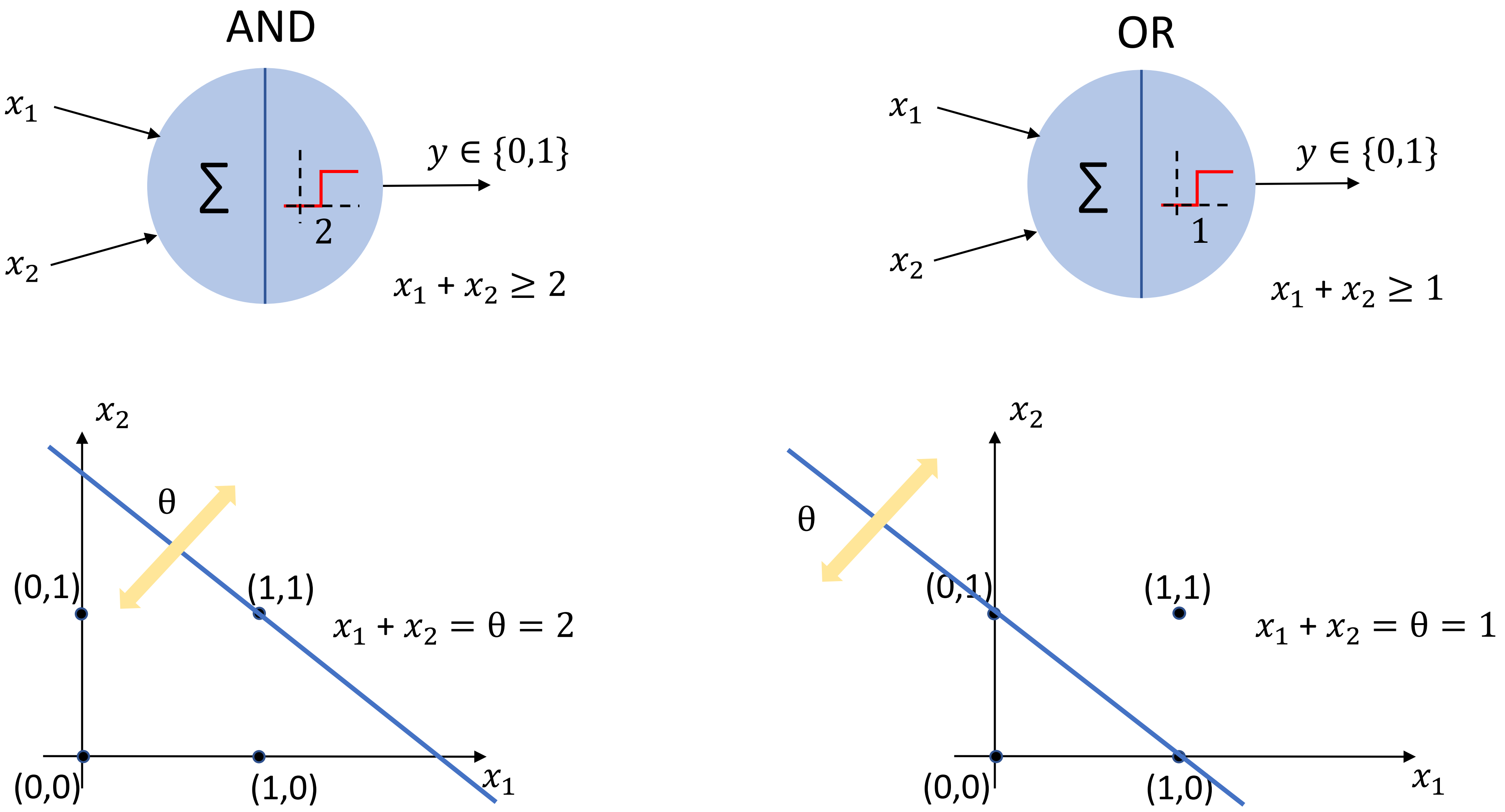
Model parameters:

- Threshold θ (manually set).

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

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

Functional & Geometric Interpretation



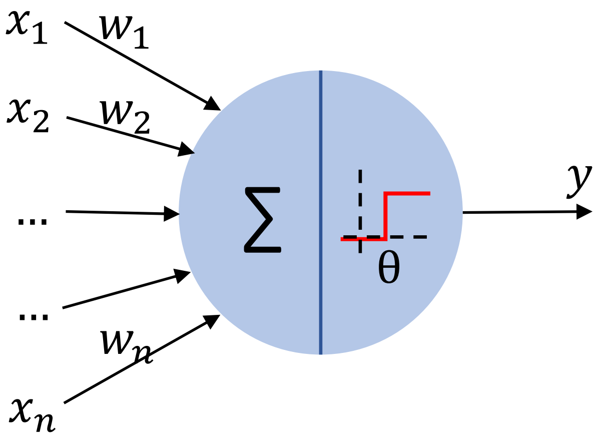
#params = 1. θ only shifts the decision boundary

The First Trainable Neuron (or The Fly's Eye)

- Perceptron - Rosenblatt (NY, 1958):

Generalization of the McCulloch-Pitts' model:

- More realistic: inputs $\in \mathbb{R}$.
- Allows assign importance (or "weights") to inputs: w_i.
- Proposed neuron = almost "exactly" nowadays neuron.



Generalization cost:

- Number of parameters increases: (θ) => (θ, w_i).
- (θ) set manually => (θ, w_i) needed a learning algorithm.

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

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

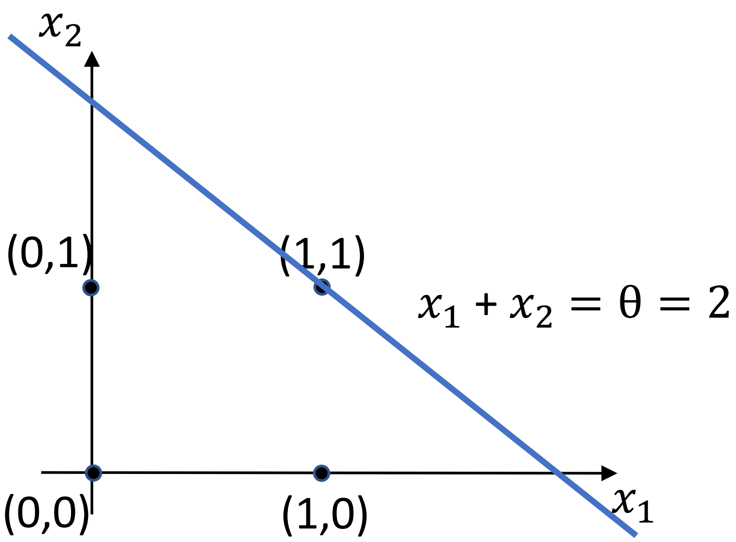
Rosenblatt, R. *The perceptron: A probabilistic model for information storage and organization in the brain*

Little Changes – Huge Geometric Impact

McCulloch-Pitch (MP)

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

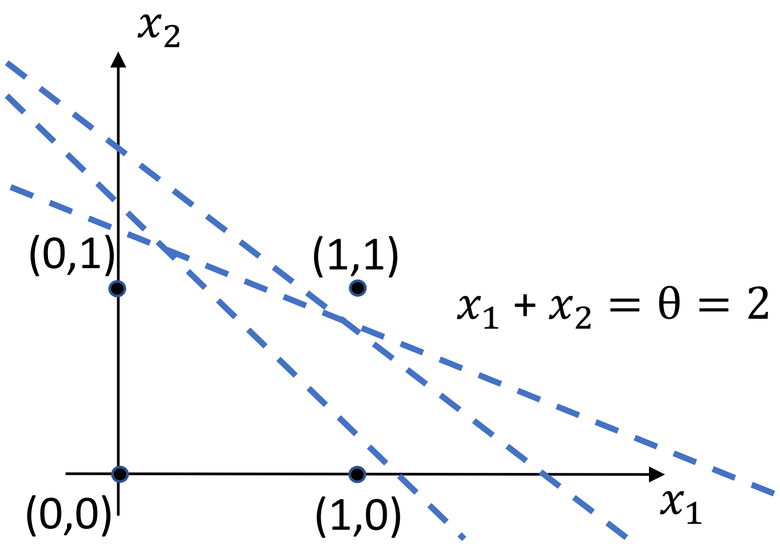
$$y = 0 \ \text{ if } \sum_{i=1}^{n} x_i < \theta$$



Perceptron

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

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



#params = 1 + $\#w_i$. Can rotate the decision boundary

Luckily There was a Learning Algorithm

- 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.
  - How: iterative adjustment of weight vector.

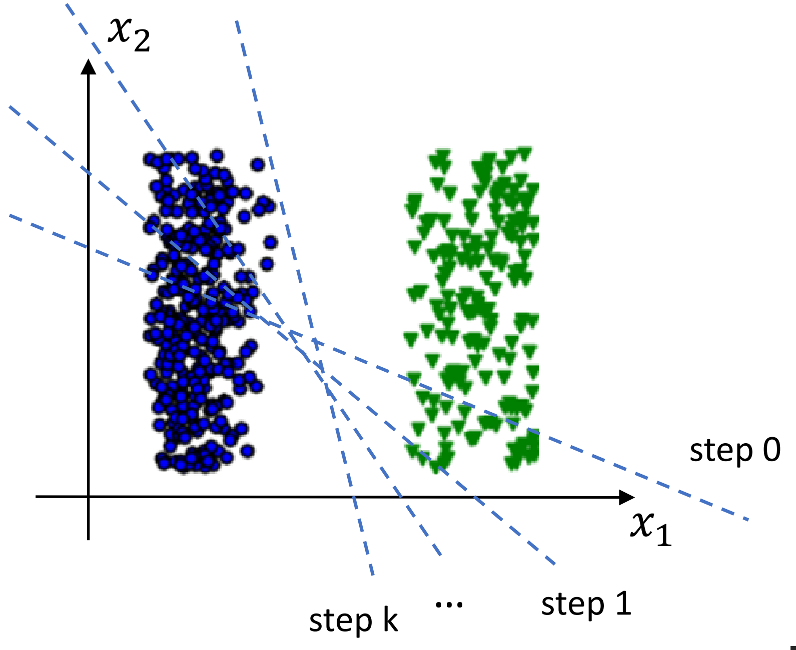
$$\sum_{i=1}^{n} w_i x_i > \theta \ \Rightarrow \sum_{i=1}^{n} w_i x_i - \theta > 0$$

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

con x_0=1, w_0= θ

Perceptron Convergence Theorem:

- θ is considered as a weight $\mathbf{w_0}$.
- Based on same distance between prediction and true value.
- If dataset is linearly separable:
  - Guaranteed to find a solution in a finite number of steps.



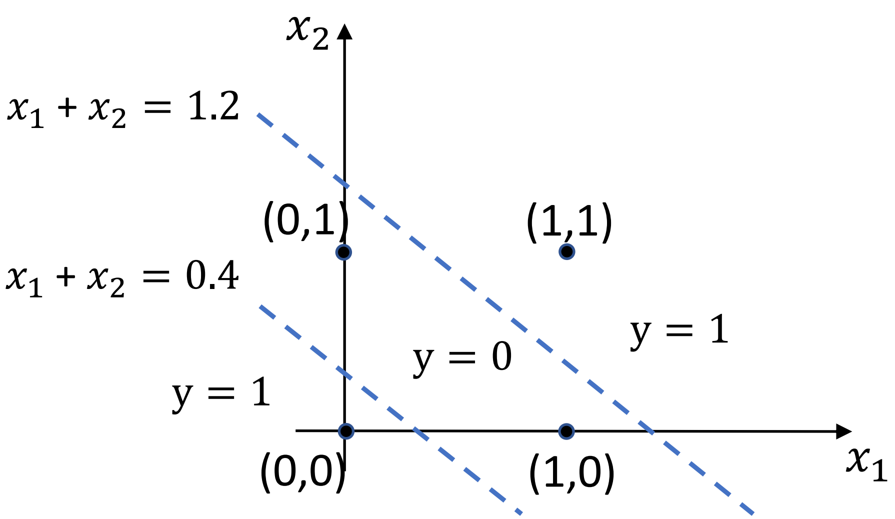
The XOR Problem (Cards on the Table)

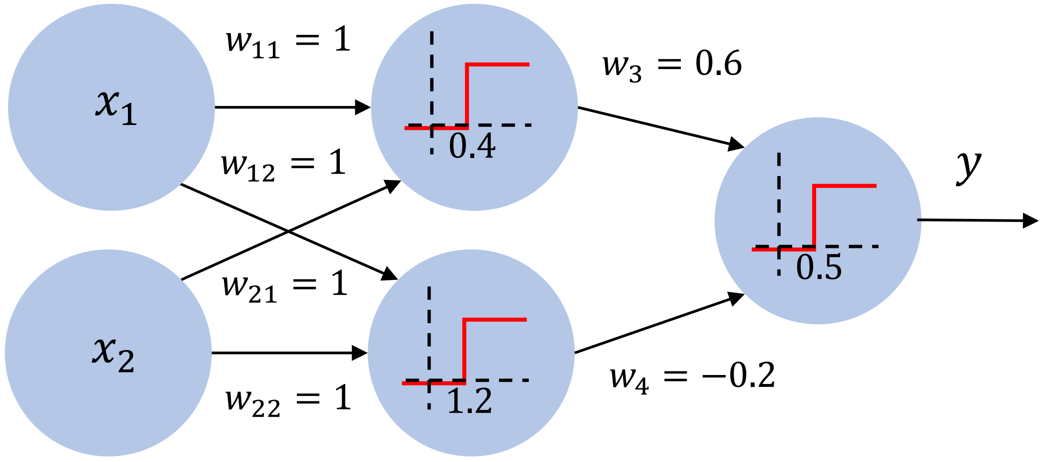
- Minsky & Papert (MIT, 1969):

Explained perceptron's critical limitation:

- Not able to solve the simple XOR function.
- Solution: stacking layers of neurons.
- Problem: no known algorithm for training multi-layer networks.

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

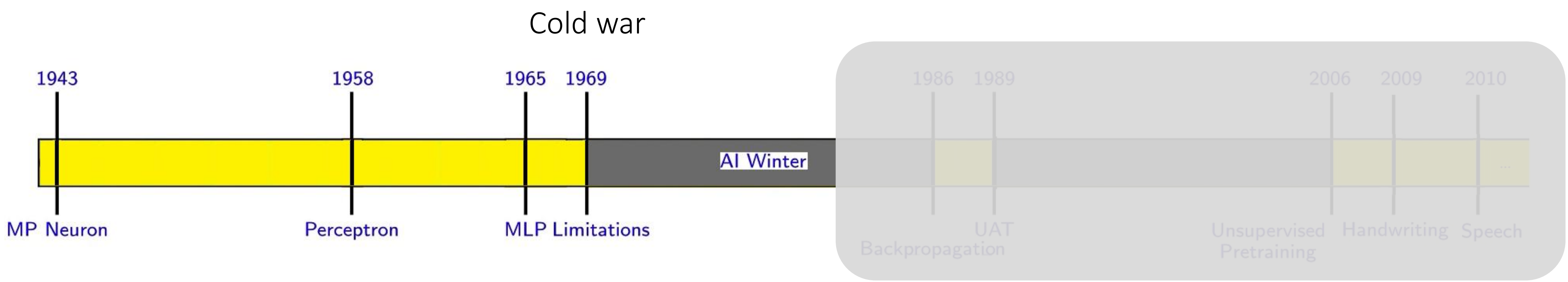




Minsky, R. *Perceptrons: An introduction to computational geometry*

Simplifying notation: remove sum operator

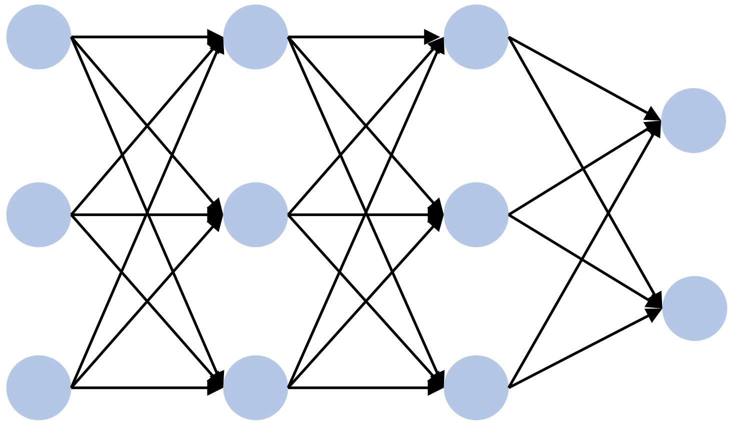
Not Everything are Peaches & Cream



- **Obstacle race:**
  - This criticism led to widespread pessimism about future of Neural Networks.
  - Funding dried up. Scientific community abandoned this line of research (1969 - …)
  - Alternation of hypes and winters:
    - hypes: periods of great optimism, huge expectations and advances.
    - winters: expectations are not met, starts a period of pessimism. Reduction of investments and leaves this line of research.

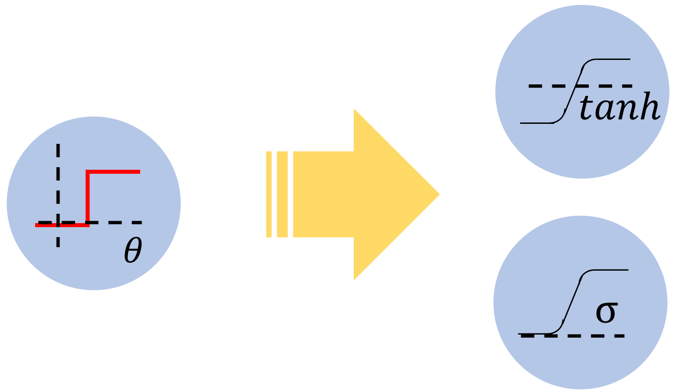
AI Winter: 20 Years Searching the Holy Grail

- 1^{st} 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 & Hinton (1986):
    - Backpropagation + Gradient descent.
  - Still the main training technique nowadays.

Key requirement: all components must be differentiable: new smooth activation functions.

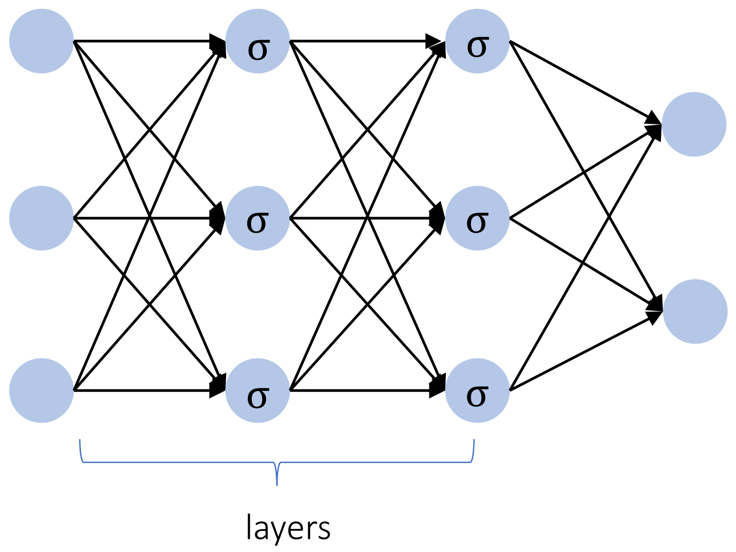


Rumelhart, R., Hinton, G. & Williams. (1986). *Learning representations by back-propagating errors*

Multi-Layer Perceptron (MLP)

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

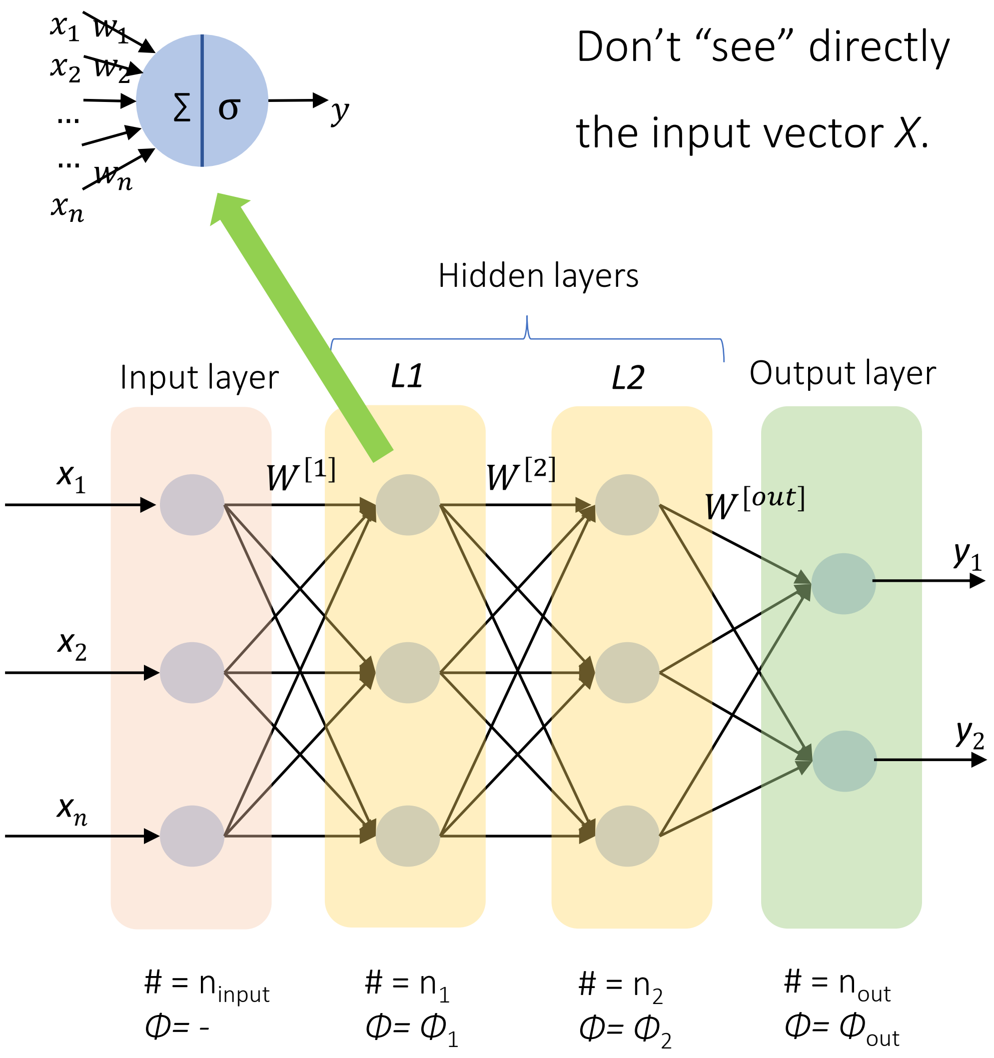
Multilayer perceptron



- MLP is a Feedforward NN:
  - Groups of perceptrons: arranged in layers.
  - Signal flows in only one direction: “no cycles”.
  - Dense layers: every neuron is connected to every neuron in the next layer.

Ingredients of MLPs

- Layers and parameters:
  - Input layer: Network's entry point.
    - Purpose: Receive the raw data vector ***X***.
    - Structure: # neurons = # input features in *X*.
  - Hidden layers: Workhorses.
    - Purpose: if multiple hidden layers is called "Deep".
    - Structure: Dense layers
      - $W^{[i]}$: weight matrix and b_i: bias vector.
      - Φ_i : a common activation function.
  - Output layer: Generates the prediction.
    - Structure: # neurons and activation function Φ_i depend on the specific task to solve (e.g. classification, regression).



The Mathematics of an MLP

Mathematical standpoint:

- Is a sequence of matrix operations + 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

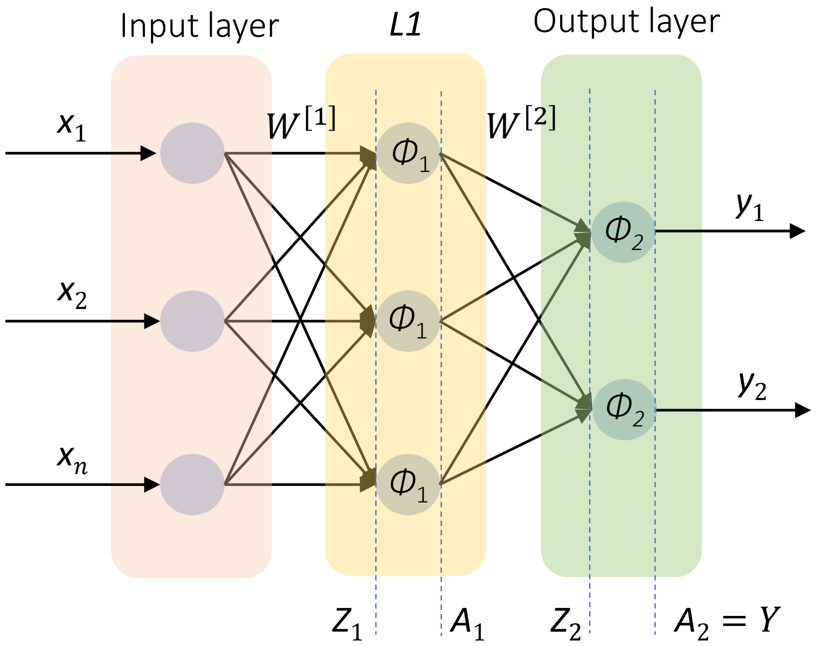
Find values of matrices: $W^{[1]}$ and $W^{[2]}$

Crucial role of non-linearity:

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

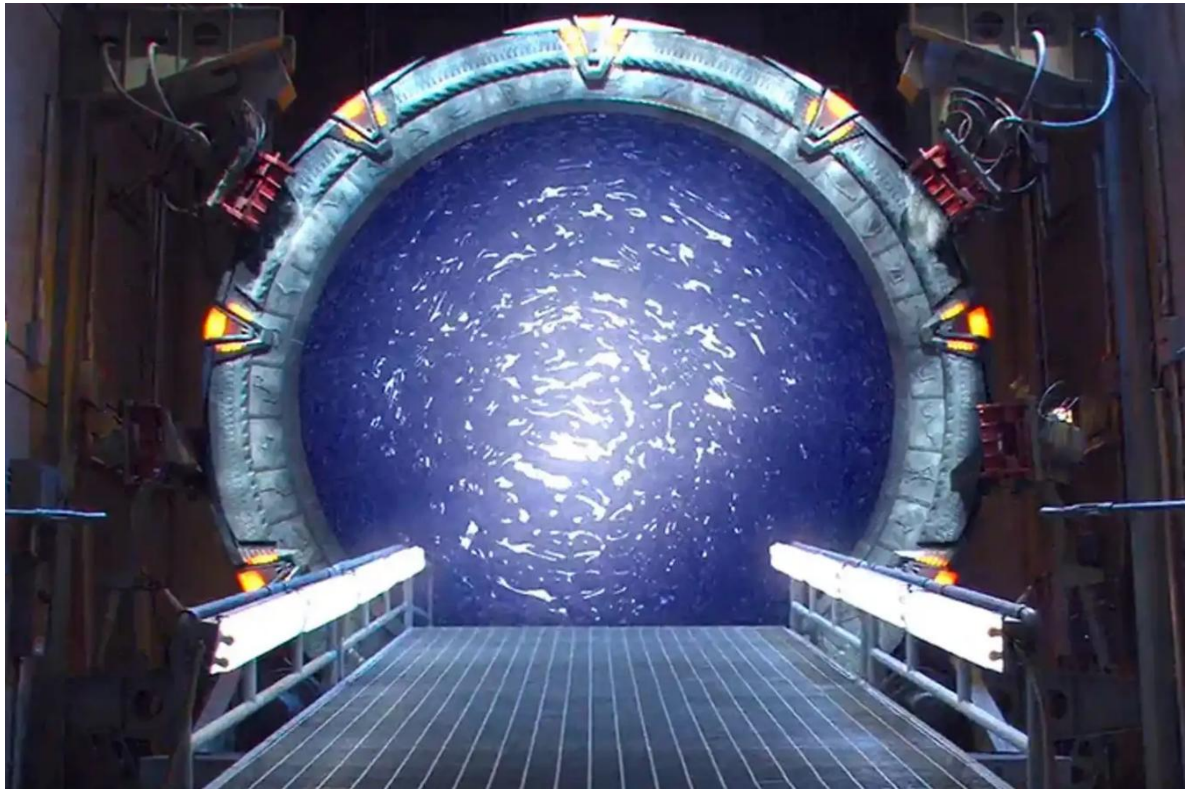
$$Y = W^{[2]}A_1 = W^{[2]}(W^{[1]}X) = (W^{[2]}W^{[1]})X$$

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



Layers as Representation Transformers

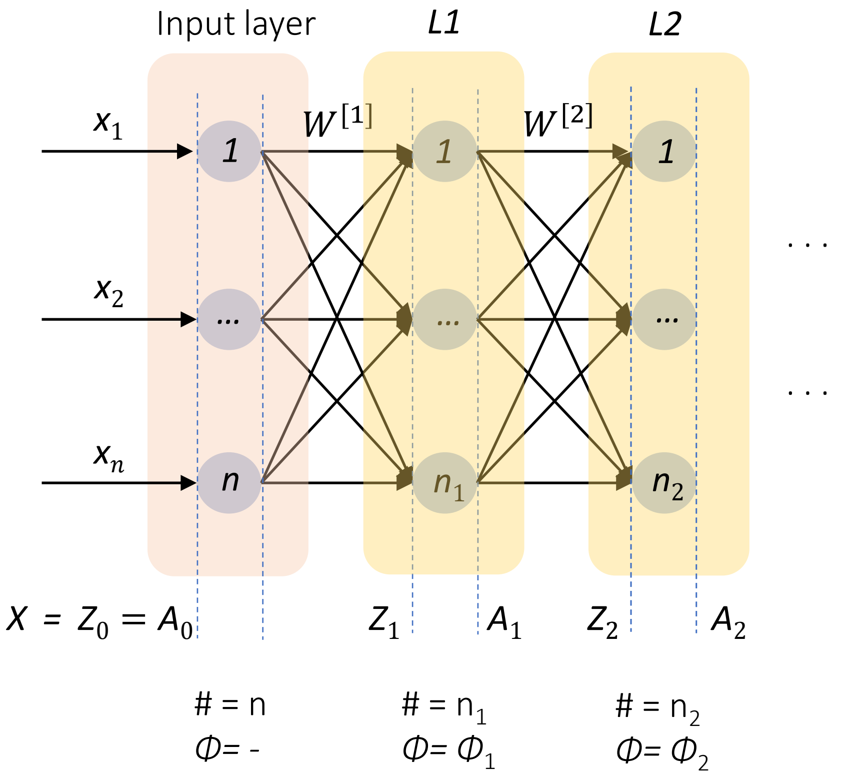
- The real role of a layer = interdimensional portal



credits

Matrix Operations are not Easy to Interpret

- Dimensional jump in action:
  - Entering the portal (entering a layer):
    - Layer receives a vector of n_1 components.
    - A point in a n_1-dim space.
  - Exiting the portal (output a layer):
    - The layer "performs" its operations: sums and non-linearities.
    - Outputs a vector of n_2 = # neurons of layer.
    - A point in a new n_2-dim space.

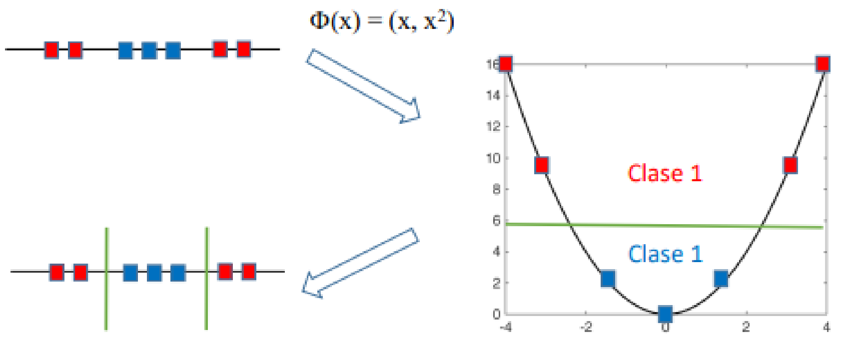


Why Change Problem Dimension?

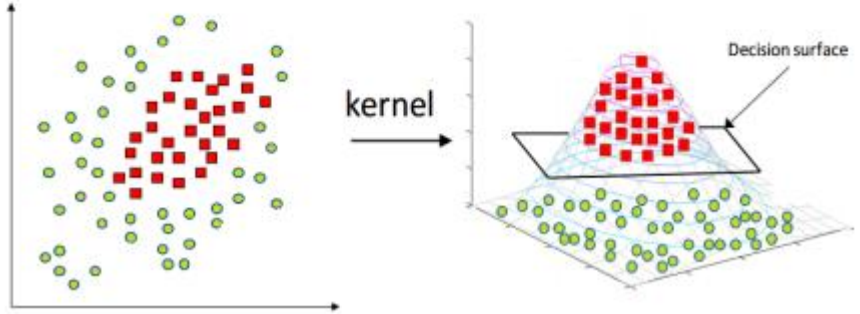
- In SVM (Support Vector Machines):
  - Facing a non-linearly separable dataset:

    Dimensionality expansion

  - Choose kernel from predetermined "Catalogue" of kernels.
  - By trial & error: Projects data into a higher-dim space.
  - Where a linear separator can be found.



- In Neural Networks:
  - Idea is generalized and automated.
  - Network learns the best "kernel".
  - The one that better separate the classes in the concrete task.

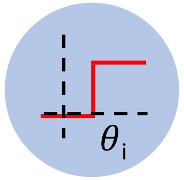


credits

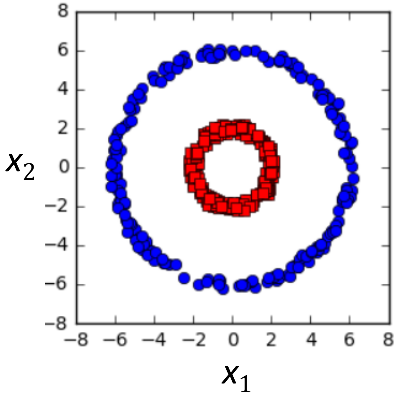
Visualizing the Transformation

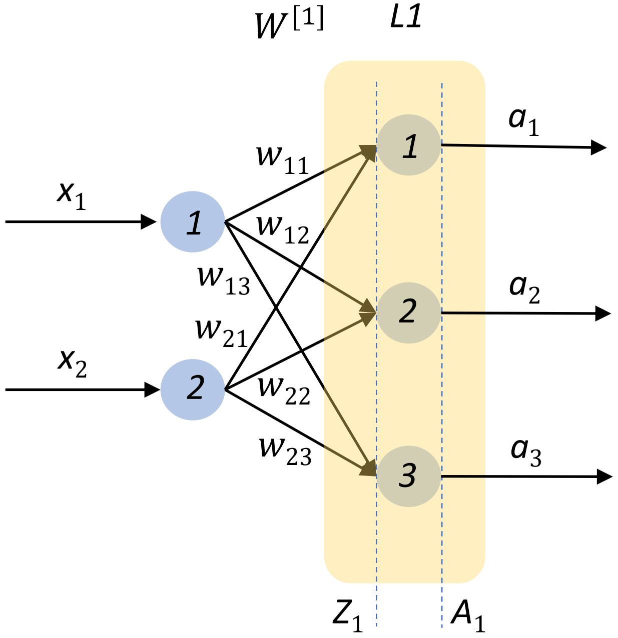
Non-linearly separable in 2D

ϕ_i:



2D-space



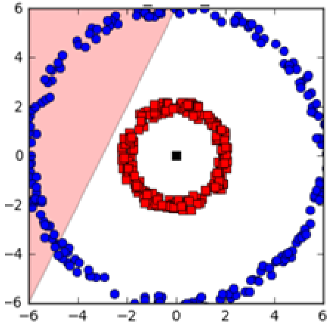


= n_{input}
Φ= -

= n_1
Φ= Φ_1

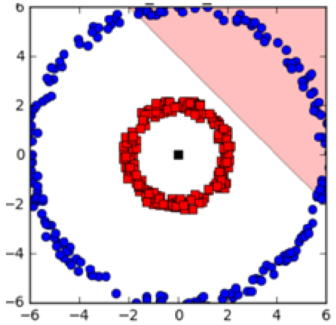
neuron 1

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



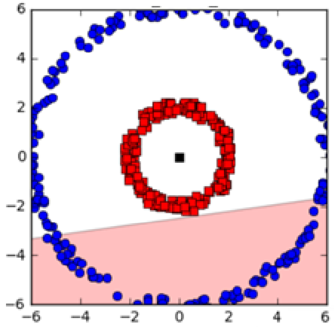
neuron 2

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



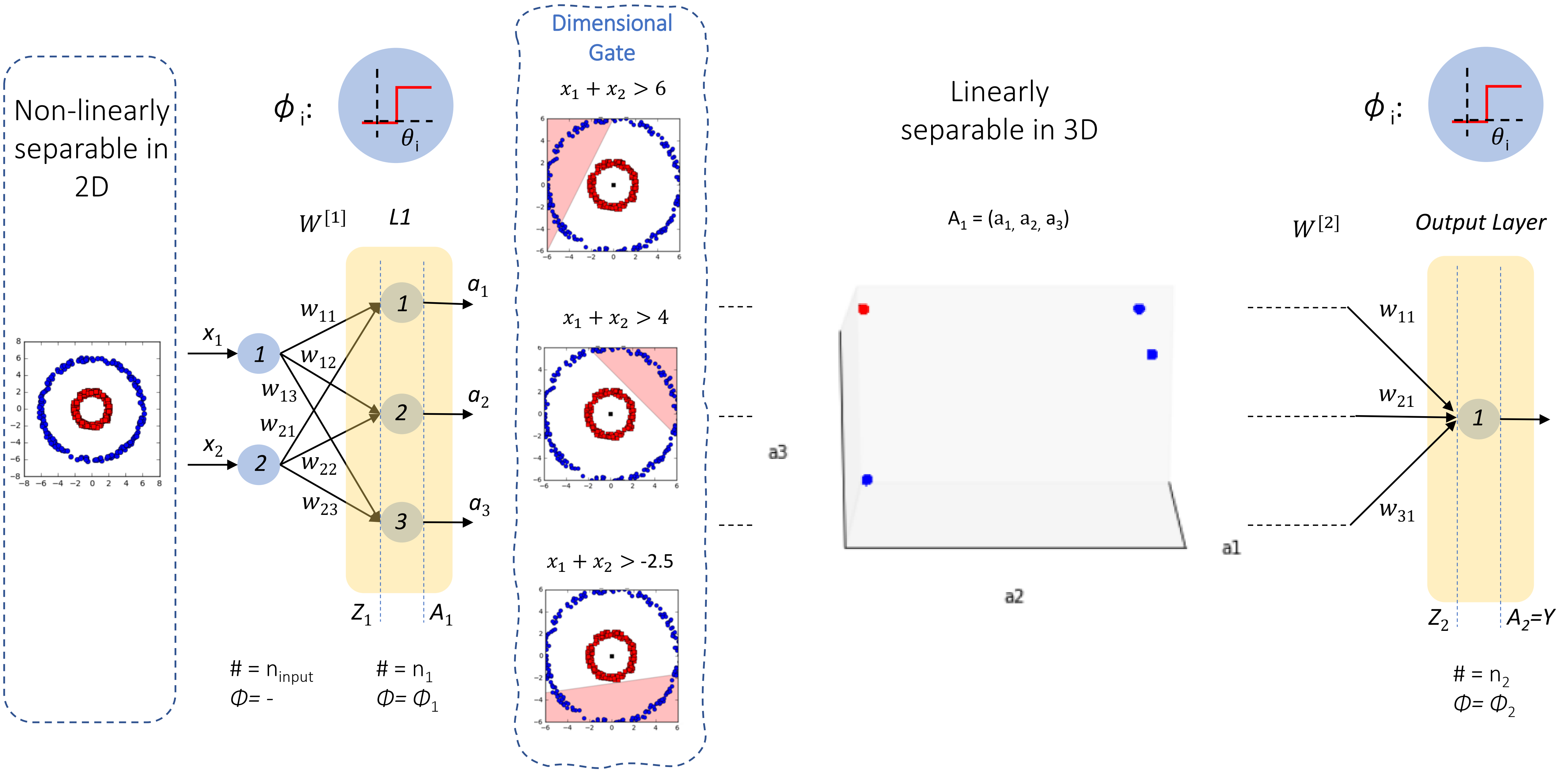
neuron 3

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



credits

Now Decision is Trivial



What the Network Actually Learns

Hierarchical learning:

- Each layer builds a new "vision of the world" based on the features of the previous layer vision.
  - Each neuron is asking a simple yes/no question.
  - The # neurons in a layer = # questions to ask.
  - Dimension of the new representation = # neurons of the layer.
- NN has learnt a hierarchical representation of the dataset useful to solve a specific task.

Knowledge is stored in:

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

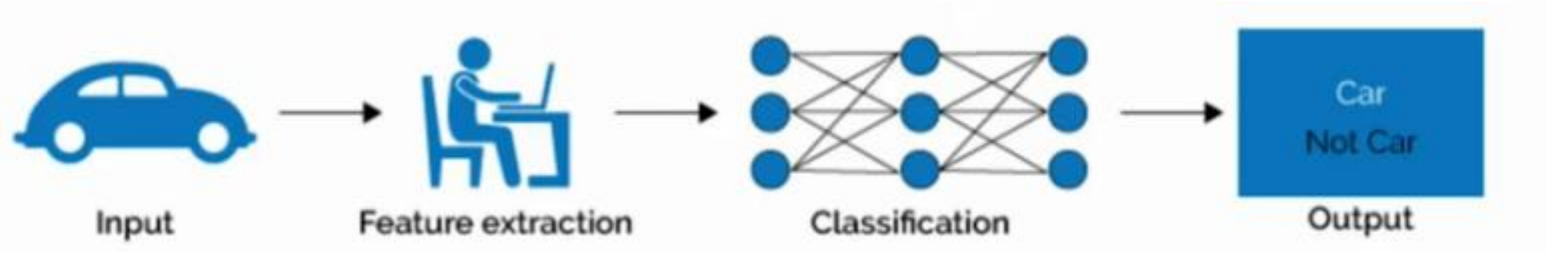
Internalized Feature Engineering

- Workflows have changes:

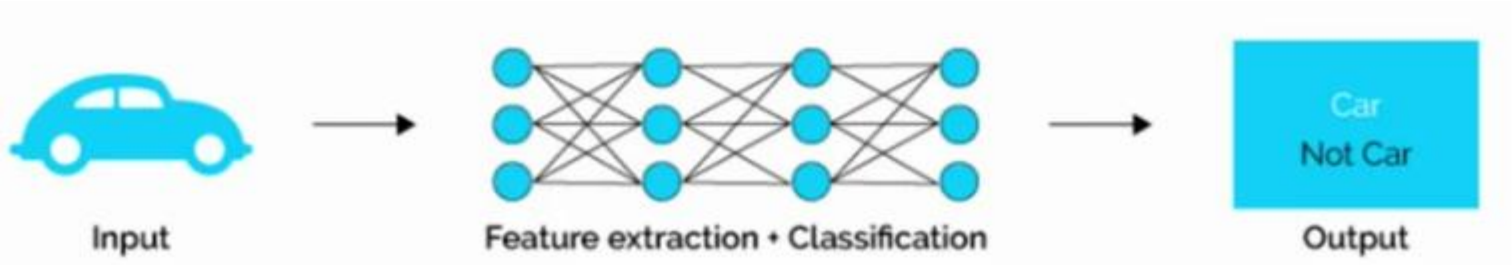
Domain expert

Result highly dependent on quality of handcrafted features

TRADITIONAL ML



NEURAL NETWORKS

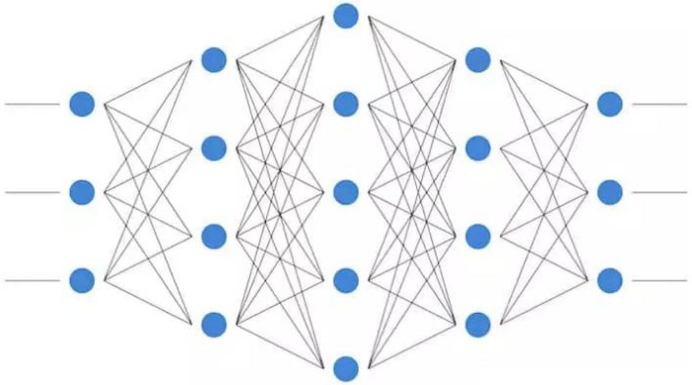


credits

NN learns relevant features automatically from the raw data

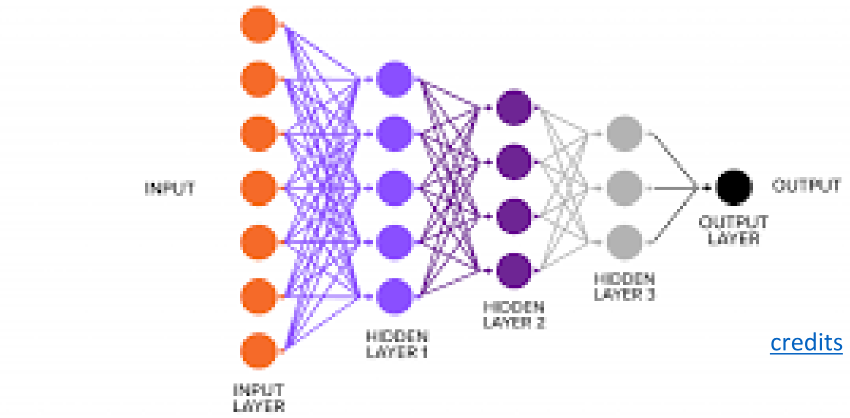
Fortunately: It is Still an Art

- The art of architecture design:
  - For low-dimensional datasets:
    - To expand dimensionality = increasing #neurons in the following layers.
    - Crate a reach representation space.
    - Reduce dimensionality to “distill” key features.
    - Typically, dimension is reduced gradually.



credits

  - For high-dimensional dataset: (images)
    - There is an excess of information.
    - Progressively reduce dimensionality step by step.
    - Result: create more meaningful and compact representations.

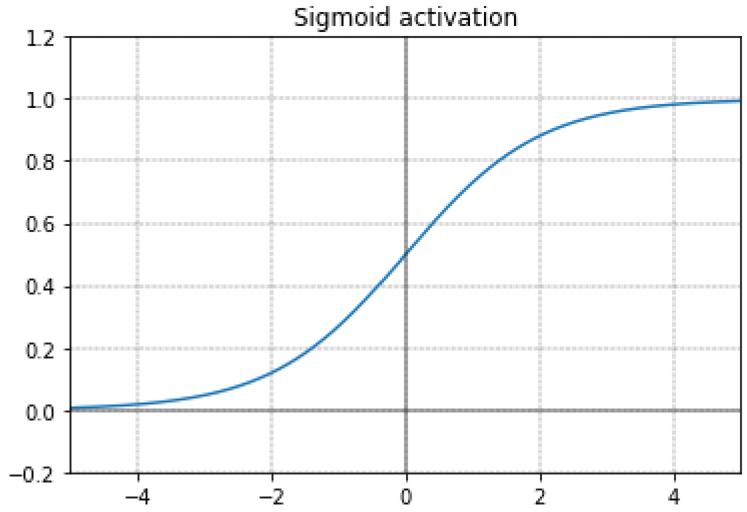


credits

Output Layer = f(stack)

Binary classification:

- Classical example: dogs vs. cats
- # 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: dogs vs. cats vs. horses
- # 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. $\phi_{out} = softmax(z)$

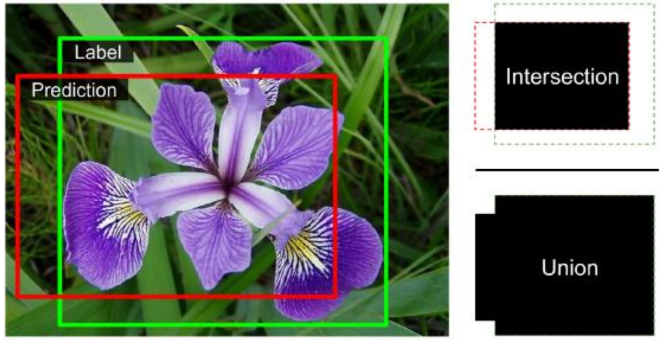
$$softmax(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$$

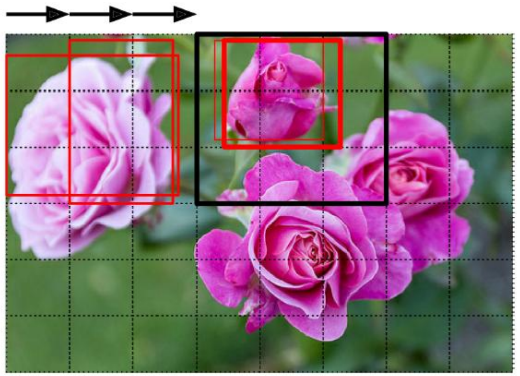
a) If $z_i < 0$, then $e^{z_i} > 0$

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

Measuring Error: Loss Functions

- Specific loss for tasks:
  - 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



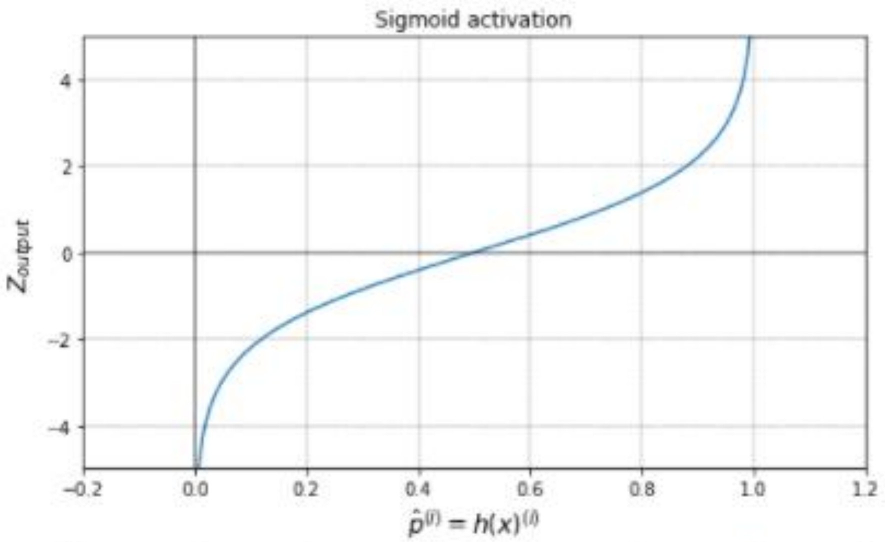


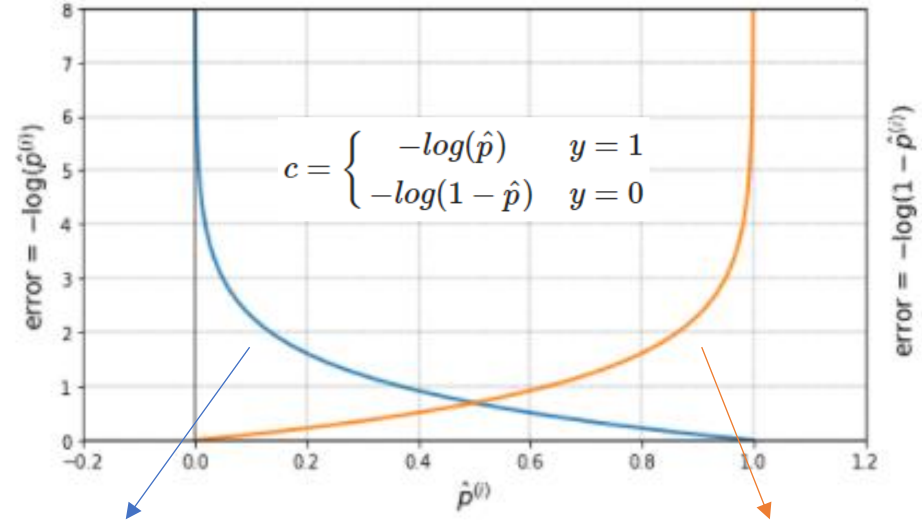
Classification Problems: Crossentropy

- Why such a strange expression?
  - K: network outputs are a probability.
  - Heavily penalize **(log near zero)** confident wrong answers
    - If true value = 1. If $P(Y = 1) \approx 0$
    - If true value = 0. If $P(Y = 1) \approx 1$
  - Combine both branches:
    - Each branch is weighted using $y^{(i)}$ and $1 - y^{(i)}$.
  - Generalized for > 2 classes.

- Total loss function:

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





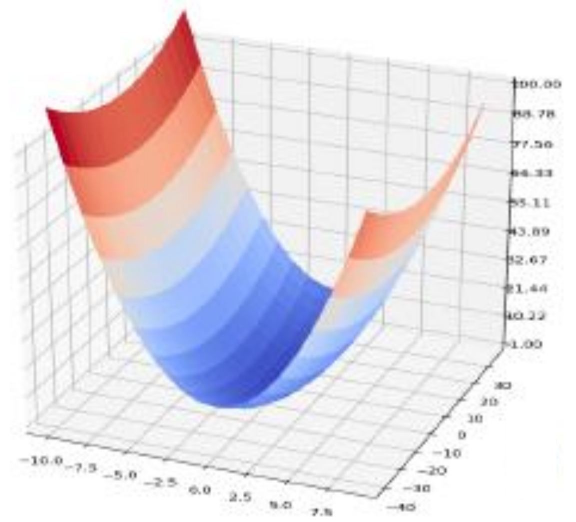
use if true value = 1

use if true value = 0

Giving Sense to Network Training

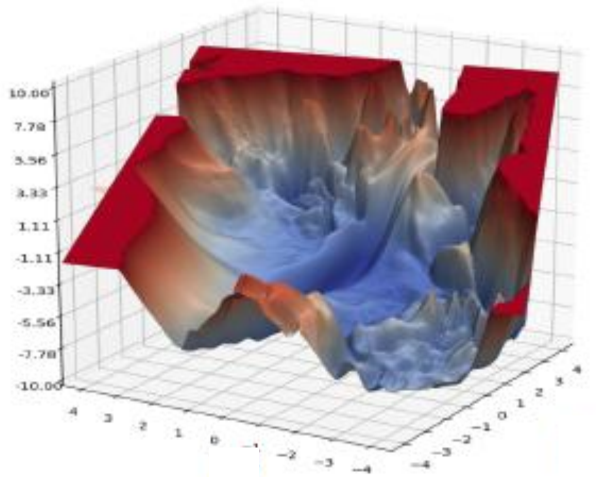
Meaning of network training?

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



Challenges:

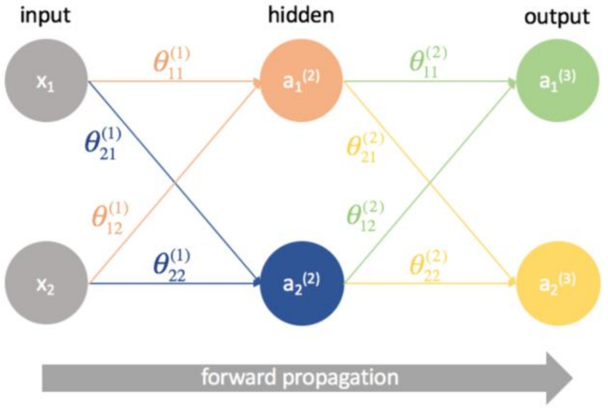
- Loss landscape is non-convex: due to the non-linear activation functions.
  - Probably only local minima will be found.
- Parameter space of dimension 10^4-10^7.

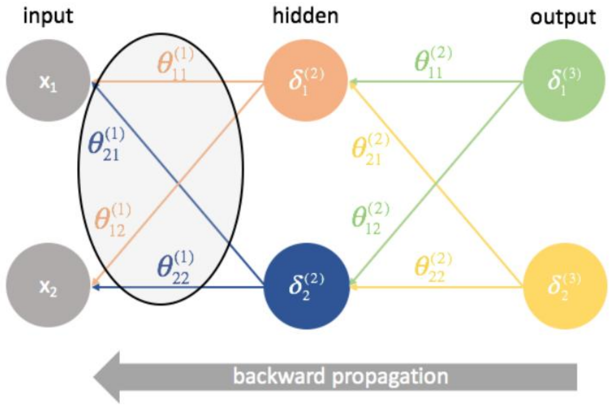


Network Training: Backpropagation + GD

- Backpropagation (Rumelhart, 1986)
  - Forward pass (or forward propagation):
    - Fed input, network make a prediction..
    - Obtain the error between prediction and true label.
  - Reverse pass (backward propagation):
    - Obtain error gradient w. r. t. all weights: $\nabla J = \frac{\partial J}{\partial W_i}$
    - Making use chain-rule: backpropagated errors traversing the NN.
  - Gradient descent:
    - Weights adjusted in the direction that reduces error.
    - Size of this step controlled by **Learning rate**.

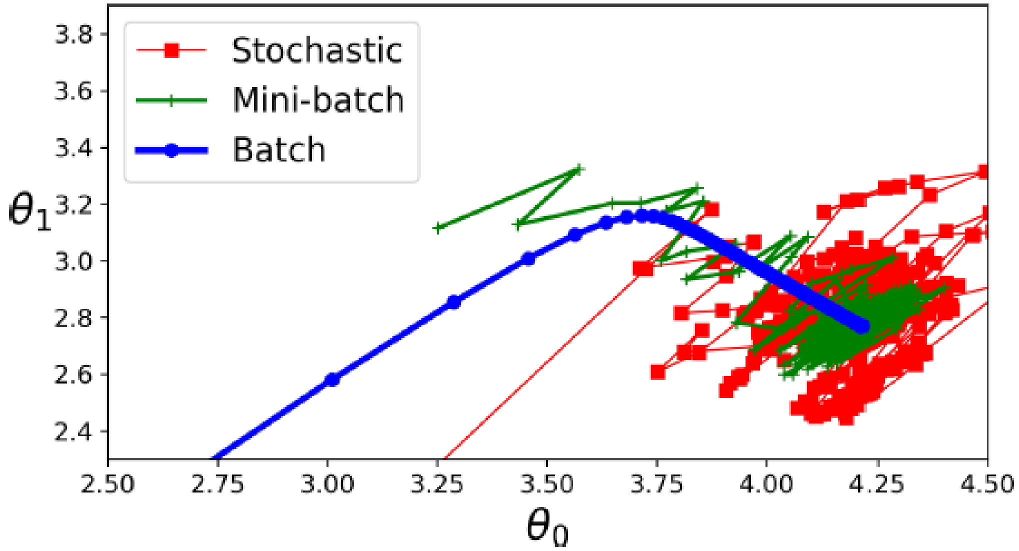
$$\Delta W = -\eta \nabla J$$





Backpropagation Variants: Batchsize

- Key point:
  - ¿How much input samples are considered to obtain the error?
  - All samples are considered (Batch GD):
    - Update has access to the whole info.
    - Very slow, more stable towards the local minima.
  - One sample at a time (Stochastic GD, SGD):
    - Strongly biased update.
    - High variance in the obtained gradients.
  - Small random subsets (Mini-Batch GD):
    - Reduces variance, with a more stable convergence.
    - Standard nowadays. New hyperparameter: Batchsize (32, 64, 128, …).



2nd AI Winter: What is Happening Here?

- 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

- Gradient Descent = slow!!

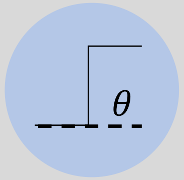
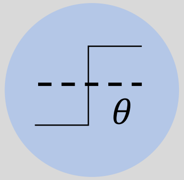
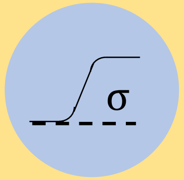
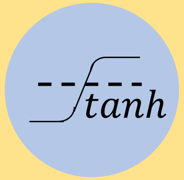
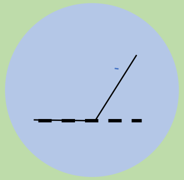
$$\Delta W = -\eta \nabla J$$

∇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)	$= \begin{cases} 0, z < \theta \\ 1, z \geq \theta \end{cases}$	Non-differentiable	Theoret. Use
		Sign function	$= \begin{cases} -1, z < \theta \\ +1, z \geq \theta \end{cases}$	Non-differentiable	Theoret. Use
Backpropagation (90s)		Sigmoid/logistic	$= \dfrac{1}{1 + e^{-z}}$	Slow train	Last layer
		Tanh	$= 2\sigma(2z) - 1$	Slow train	Last layer
Nowadays (<2015)		ReLU (Rectified Linear Unit)	$= \max(0, z)$	Fast train	Hidden layer

29