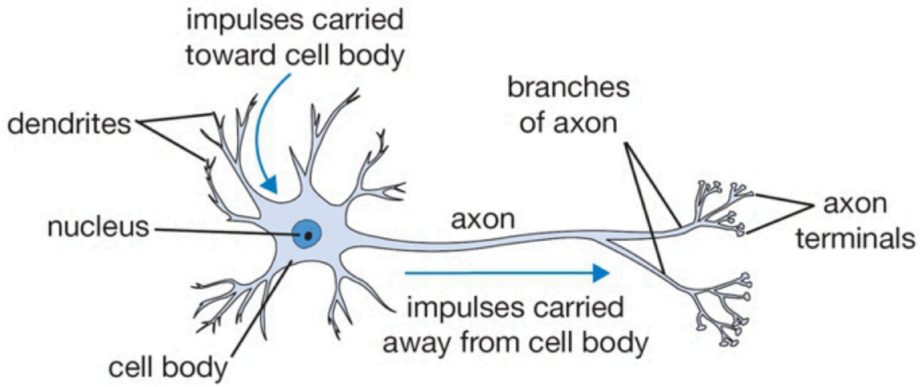
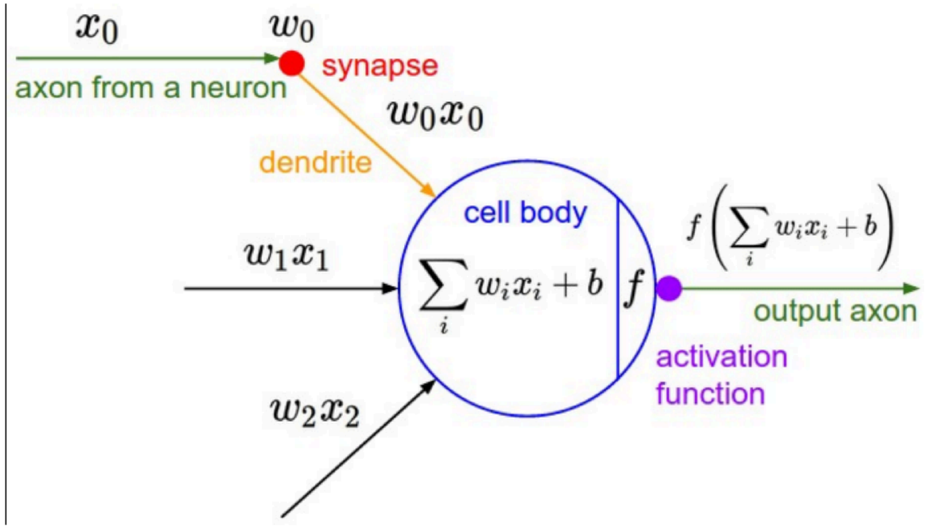
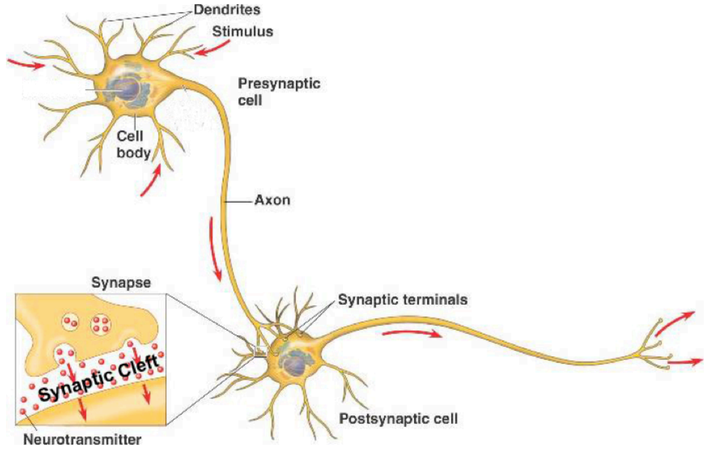
Neural Network

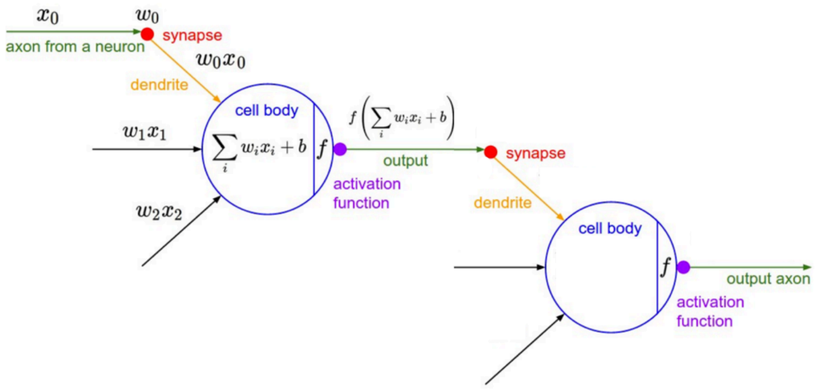
Multi-Layer Perceptron & Deep Learning

Model Representation







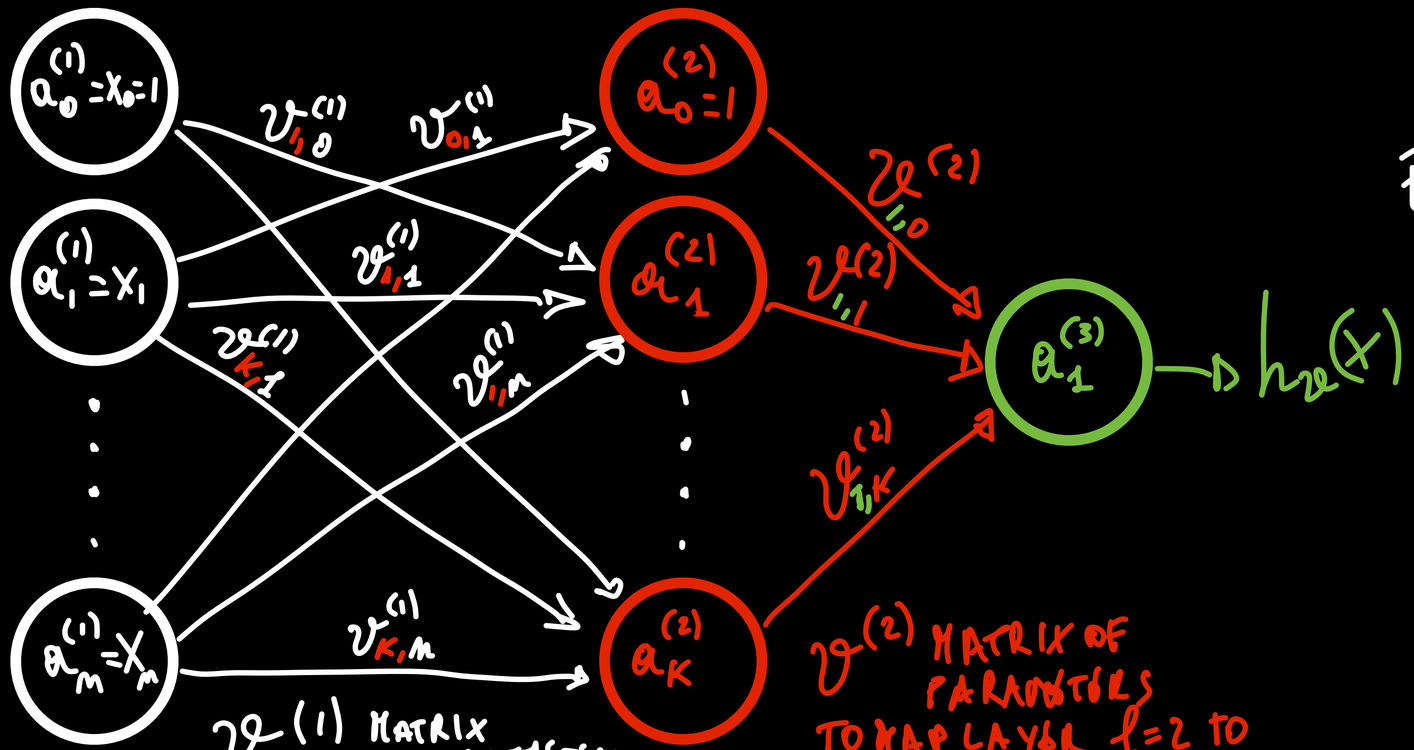


NEURAL NETWORK - MODEL REPRESENTATION

THE MULTI-LAYER PERCEPTRON (MLP)

MOTIVATIONS: 1) WE NEED TO FIT DATA WHICH LIVE IN NON-LINEAR SPACE (E.G., XOR PROBLEM)

2) GIVEN A SET OF DATA (E.G., IMAGES, TEXT DOCUMENTS) WE NEED TO LEARN A GOOD REPRESENTATION OF THEM (AND NOT AN HAND CRAFTED ONE!)



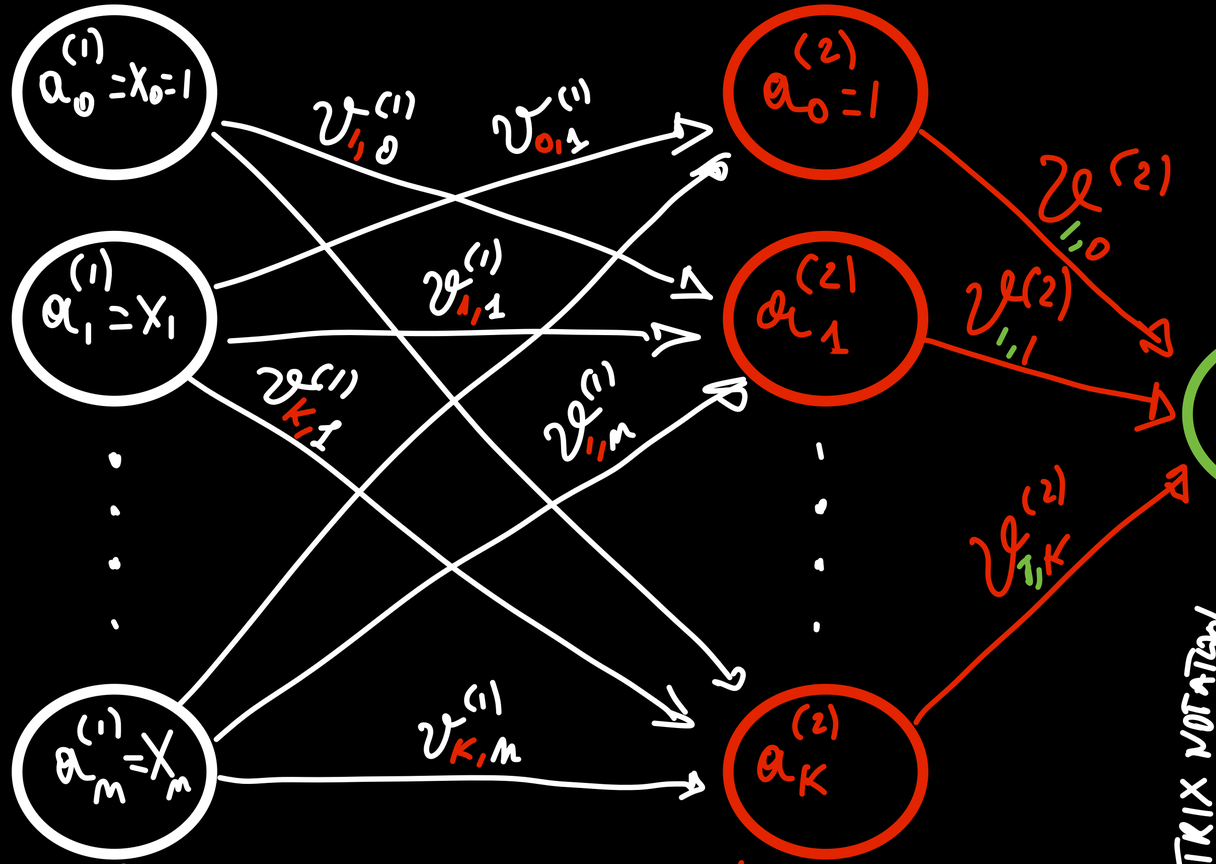
PARAMETERS: $\theta_{ji}^{(l)}$

$l \equiv$ LAYER

$i \equiv$ LAYER IN PARTENZA

$j \equiv$ LAYER IN ARRIVO

$a_i^{(l)} \equiv$ i-th ACTIVATION UNIT OF THE LAYER l



How many parameters?

TENSOR/MATRIX NOTATION

$$z^{(\ell)} = \theta^{(\ell-1)} a^{(\ell-1)}$$

$$z_j^{(\ell)} = \theta_j^{(\ell-1)} a^{(\ell-1)} = \theta_{j,0}^{(\ell-1)} a_0^{(\ell-1)} + \theta_{j,1}^{(\ell-1)} a_1^{(\ell-1)} + \dots + \theta_{j,m}^{(\ell-1)} a_m^{(\ell-1)}$$

$$a^{(\ell)} = f(z^{(\ell)})$$

$$a_j^{(\ell)} = f(z_j^{(\ell)})$$

f DERIVABLE ACTIVATION FUNCTION

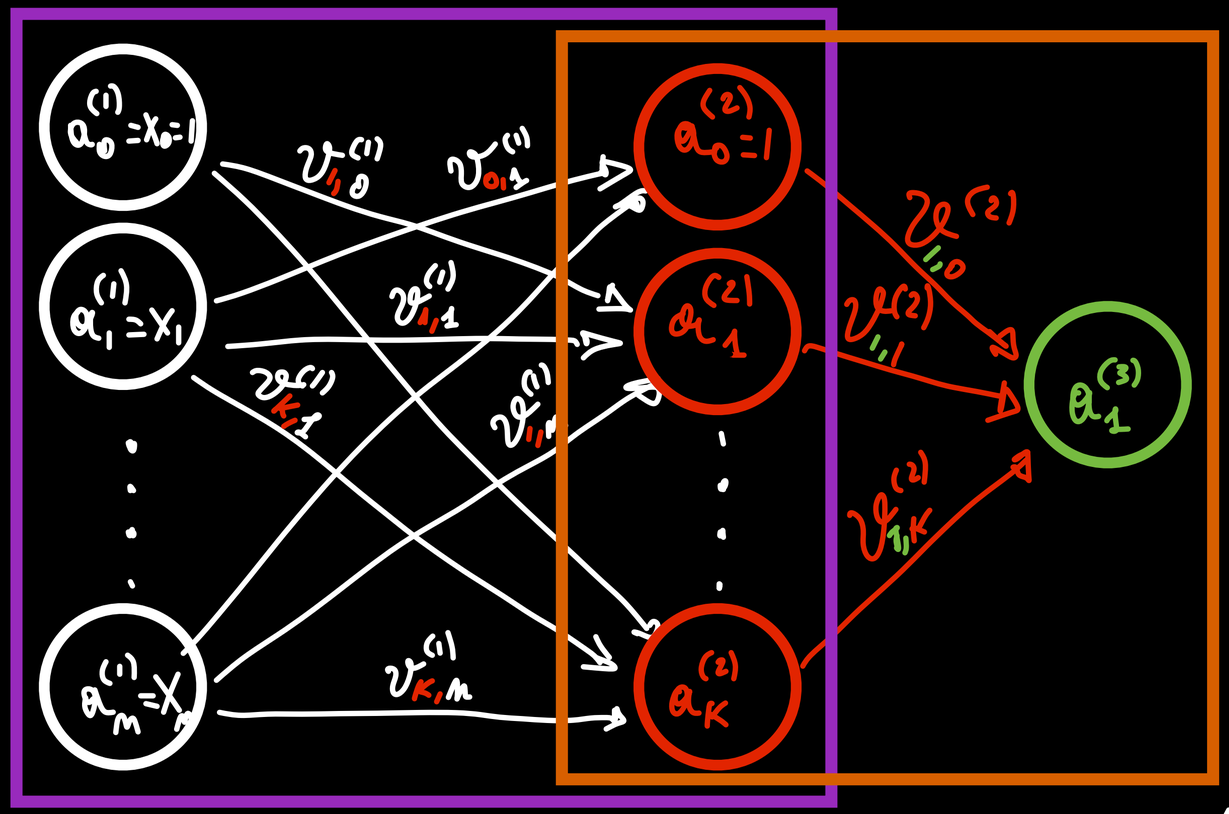
$f \simeq f$ = DERIVABLE FUNCTION (E.g. SIGMOID)

A NEURAL NET (ALSO DEEP) IS A COMPOSITION OF DERIVABLE FUNCTIONS

$$h_\theta(x) = a_1^{(3)} = f\left(\theta_{1,0}^{(2)} + \theta_{1,1}^{(2)} a_1^{(2)} + \dots + \theta_{1,K}^{(2)} a_K^{(2)}\right) =$$

$$= f\left(\theta_{1,0}^{(2)} + \theta_{1,1}^{(2)} f\left(\theta_{1,0}^{(1)} + \theta_{1,1}^{(1)} a_1^{(1)} + \dots + \theta_{1,m}^{(1)} a_m^{(1)}\right) + \dots + \theta_{1,K}^{(2)} f\left(\theta_{K,0}^{(1)} + \dots + \theta_{K,m}^{(1)} a_m^{(1)}\right)\right)$$

$z_K^{(2)}$ $z_1^{(2)}$



A NEURAL NETWORK IS COMPOSED BY TWO PARTS:

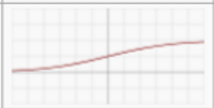
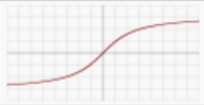
- ☐ USEFUL TO LEARN A FEATURE SPACE IN A SUPERVISED WAY DEPENDING ON THE TASK (HENCE HOW TO TRANSFORM x IN $a^{(2)}$)
- ☐ A SOFTMAX (LOGISTIC REGRESSOR IF BINARY CLASSIFIER) WHICH TAKE AS INPUT THE LEARNED FEATURE

"Universal Approximation Theorem"

A feedforward network with a single layer is sufficient to represent any continuous function, but the layer may be infeasibly large and may fail to learn and generalize correctly (Ian Goodfellow).

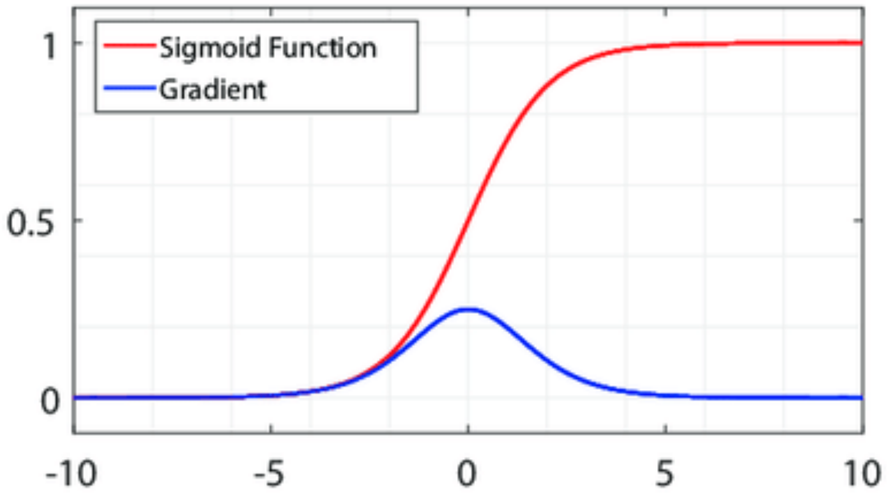
- YOU CAN ADD AS MANY AS HIDDEN LAYER YOU NEED TO MAKE THE NETWORK <u>DEEP</u> (A DEEP NETWORK!) IN ORDER TO FIT YOUR NON-LINEAR PROBLEM
- DIFFERENT HIDDEN LAYER CAN HAVE DIFFERENT NUMBER OF NEURON
- NEURONS USUALLY USE THE SAME ACTIVATION FUNCTION f

Activation Functions

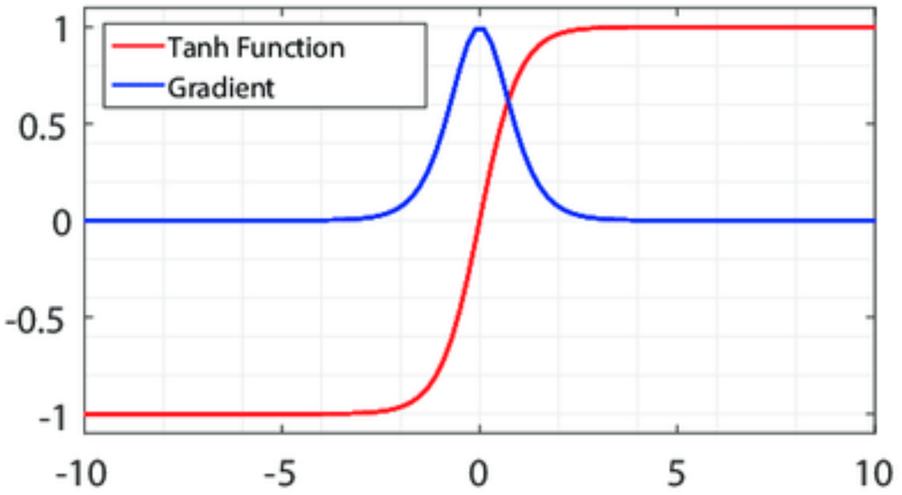
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

EVERY ACTIVATION FUCTION YOU WISH TO USE HAVE TO BE DERIVABLE
⇒ COMPOSITION (NEURAL NET) OF DERIVABLE FUNCTIONS IS DERIVABLE

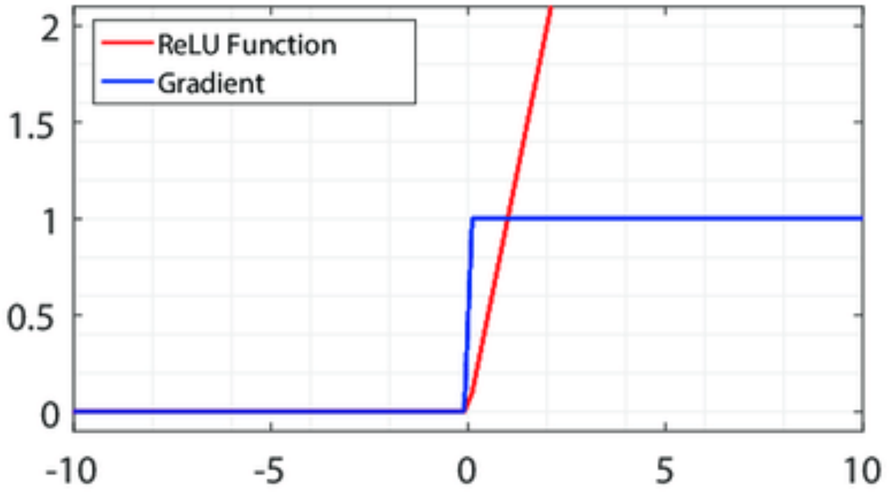
Most used Activation Functions and their Derivatives



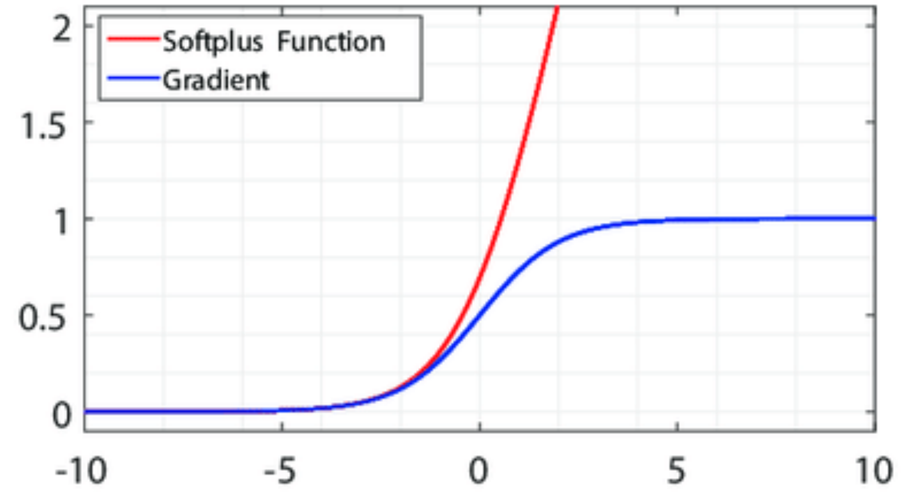
(a)



(b)

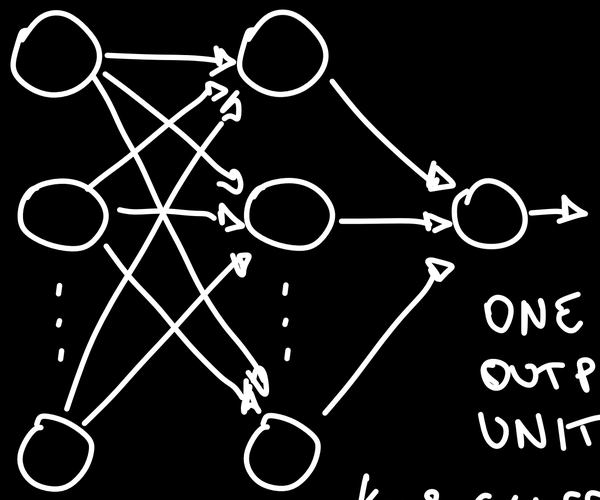


(c)



(d)

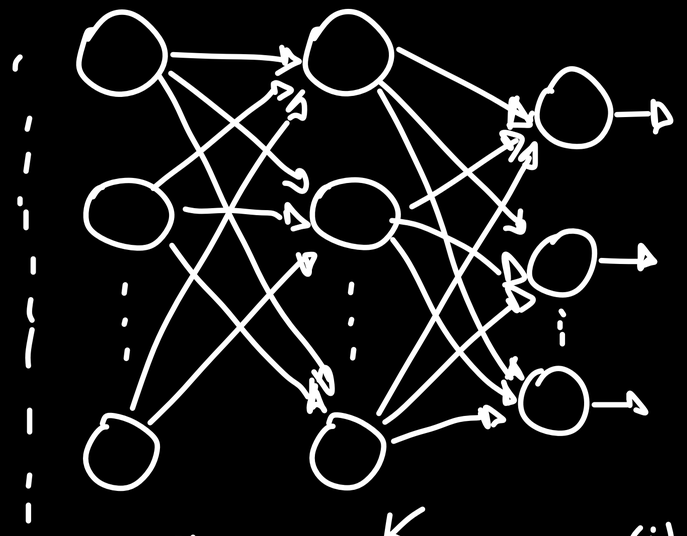
- NEURAL NETWORKS ALLOW TO TRAIN COMPLEX NON-LINEAR HYPOTHESIS ON FEATURE SPACES GIVEN A TASK
- HOW TO USE FOR REGRESSION? JUST USE LINEAR REGRESSION IN THE LAST LAYER (AND THE CORRESPONDING LOSS FUNCTION)
- HOW TO EXTEND FOR MULTI-CLASS CLASSIFICATION?



ONE OUTPUT UNIT

$k=2$ CLASSES

$y^{(i)} \in \{0,1\}$



$K \geq 3$

K OUTPUT UNITS

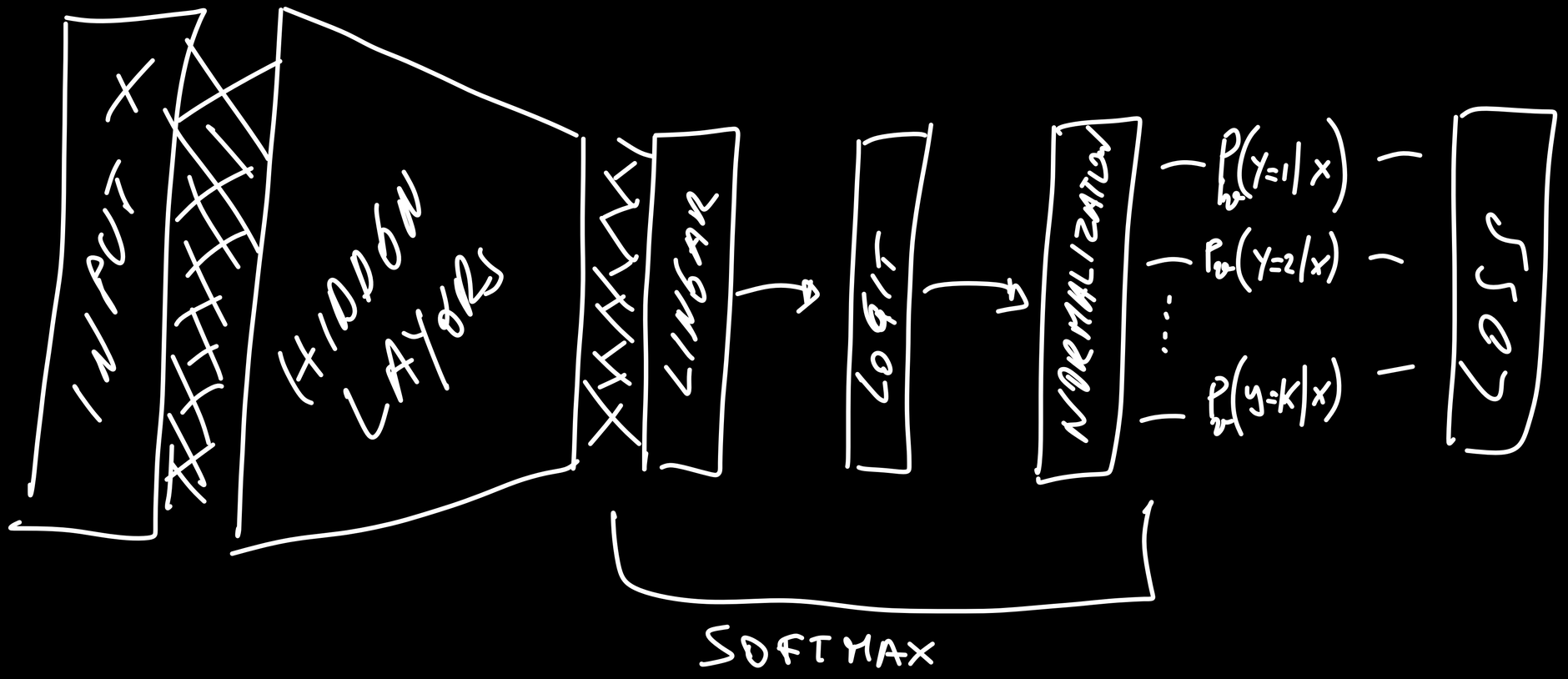
$y^{(i)} \in \{0,1\}^K$

$$y^{(i)} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

ONE-HOT ENCODING

1 ONLY IN POSITION CORRESPONDING TO THE CLASS

NEURAL NETWORK "STANDARD" STRUCTURE



IS IT POSSIBLE TO USE FOR REGRESSION (NOT CONSIDERING LOGIT AND NORMALIZATION)

NEURAL NETWORK – TRAINING & INFERENCE

- SUPPOSE WE HAVE DEFINED AN ARCHITECTURE
- DURING TRAINING PHASE OUR GOAL IS TO LEARN THE BEST PARAMETERS

  FOR OUR MODEL ⇒ WE NEED A LOSS FUNCTION + AN ALGORITHM FOR LEARNING THEM
- WHEN PARAMETERS HAVE BEEN OBTAINED WE USE THEM TO INFER ON NEW PATTERNS

COST FUNCTION (TO BE USED FOR CLASSIFICATION)

OUR GOAL IS TO FIND $\hat{\nu}$ SUCH THAT

$$\hat{\nu} = \min\left(J(\nu)\right)$$

WHERE THE COST FUNCTION IS THE FOLLOWING

$$J(\nu) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{c=1}^{K} 1\{y_c^{(i)} = c\}\log\left(\left(h_\nu^{(c)}(x^{(i)})\right)\right)$$

↑ CROSS-ENTROPY FUNCTION

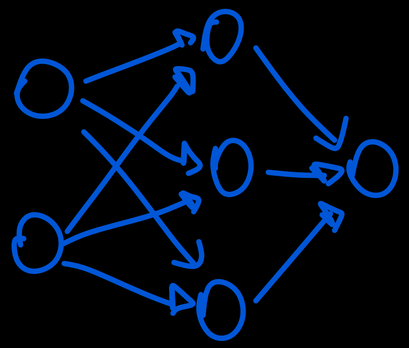
↑ Σ OVER CROSS-ENTROPY OF EVERY OUTPUT NEURON

TO REACH OUR GOAL WE SHOULD BE ABLE TO EFFICIENTLY COMPUTE $\frac{\partial J(\nu)}{\partial \nu_{j,i}^{(\ell)}}$ $\forall j,i,\ell$

Neural Network and Overfitting

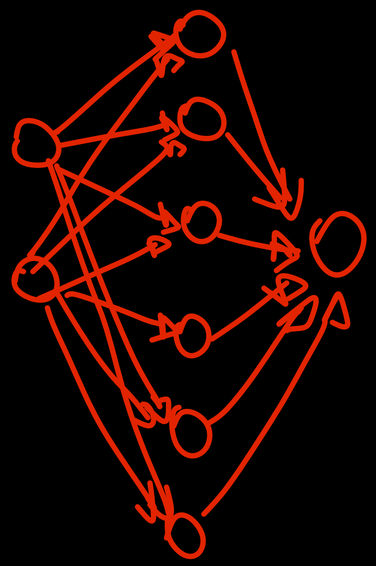
"Small" neural net

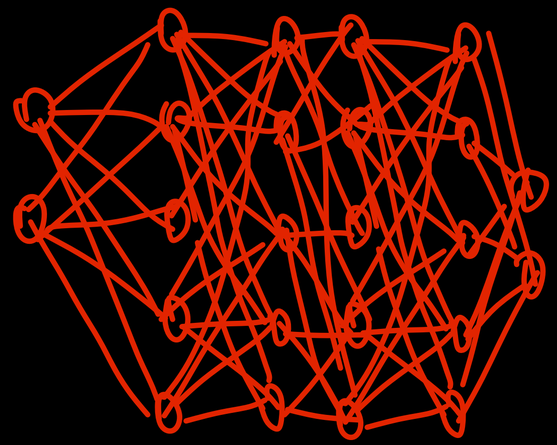
- Fewer parameters
- More prone to underfitting
- Computationally cheaper



"Large" neural net

- More parameters
- More prone to overfitting
- Computationally more expensive

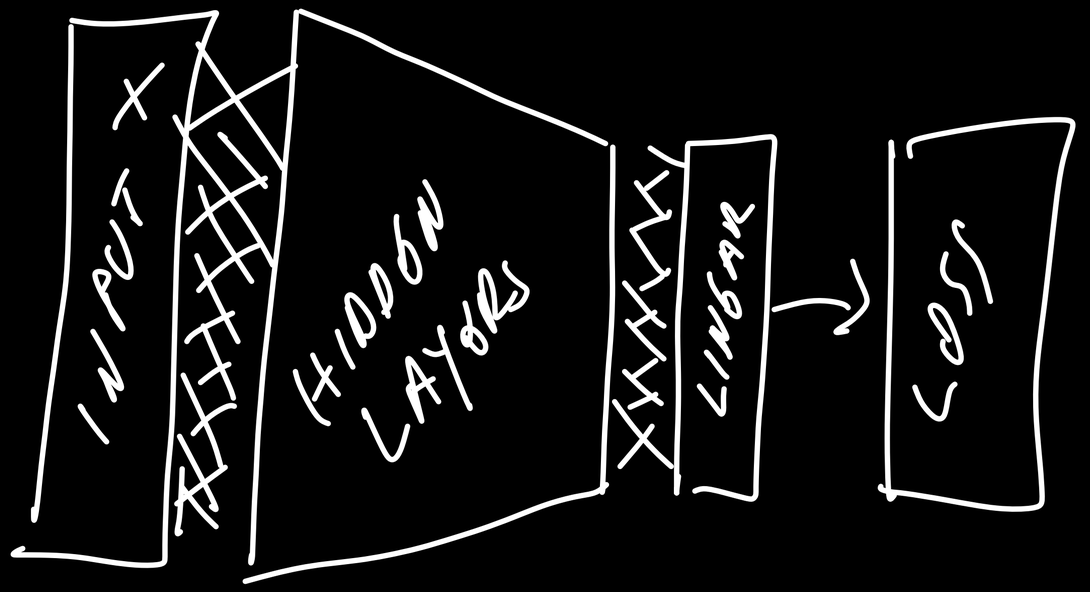




Use regularization (λ) to address overfitting on large NN

How to use for regression?

- We need to change the cost function (use MSE loss)
- Each output node is a linear regression
- Cost function should take into account the output of all regressors

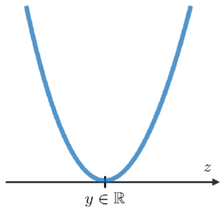
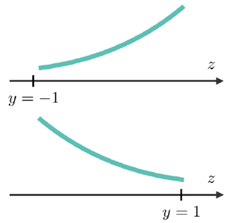
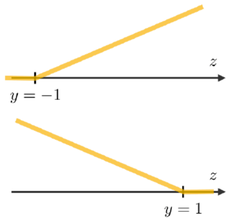
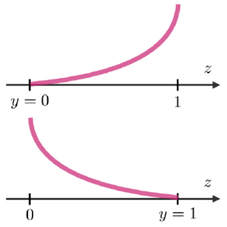


Neural Network at Glance

$h_\theta(x^{(i)})$

	Regression	Classifier
Outcome	Continuous	Class

$$L : (z,y) \in \mathbb{R} \times Y \longmapsto L(z,y) \in \mathbb{R}$$

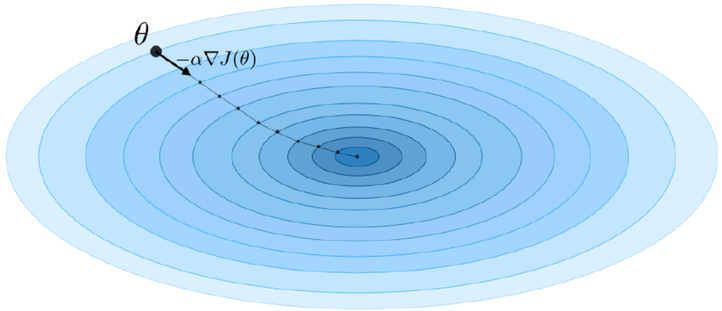
Least squared	Logistic	Hinge	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\Big[y\log(z)+(1-y)\log(1-z)\Big]$
			
Linear regression	Logistic regression	SVM	Neural Network

$$J(\theta) = \sum_{i=1}^{m} L(h_\theta(x^{(i)}), y^{(i)})$$

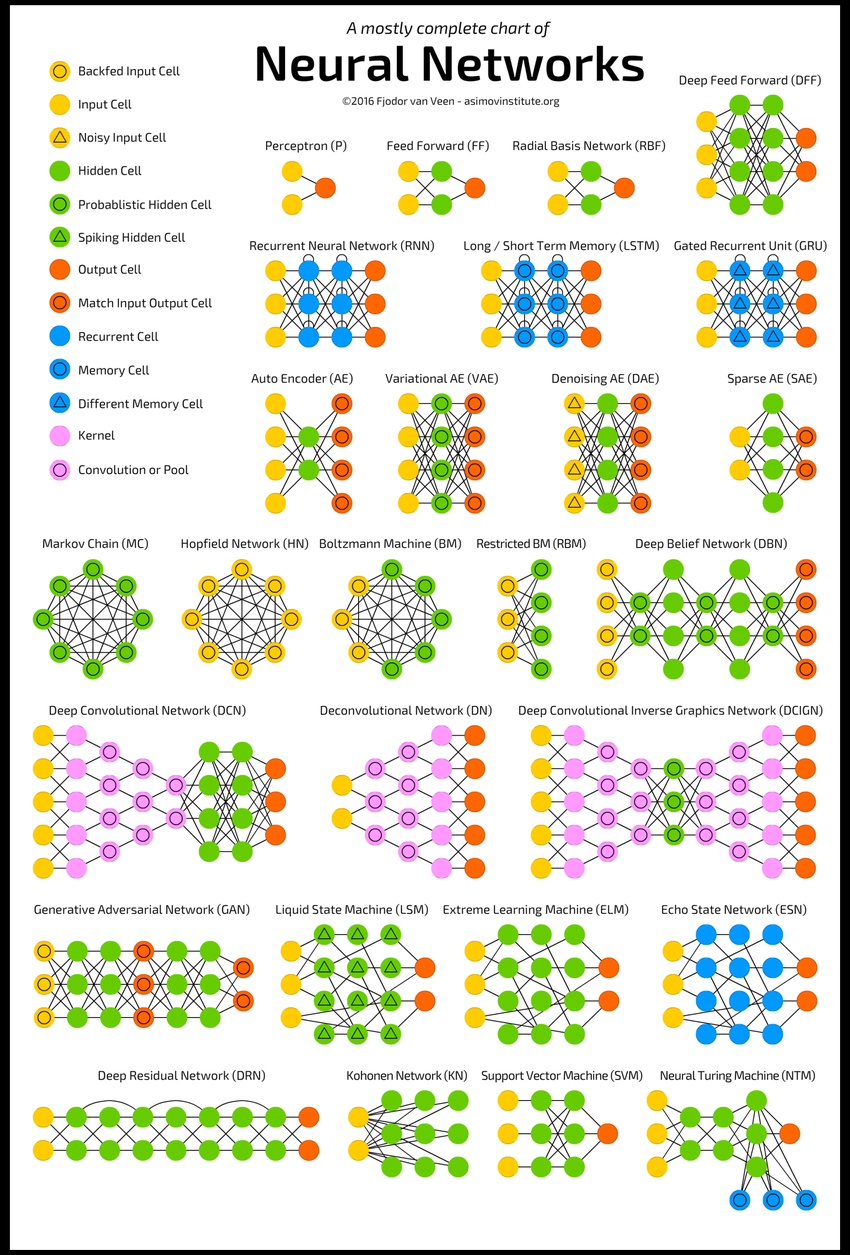
$$\theta \longleftarrow \theta - \alpha \nabla J(\theta)$$

BACKPROPAGATION

GRADIENT DESCENT



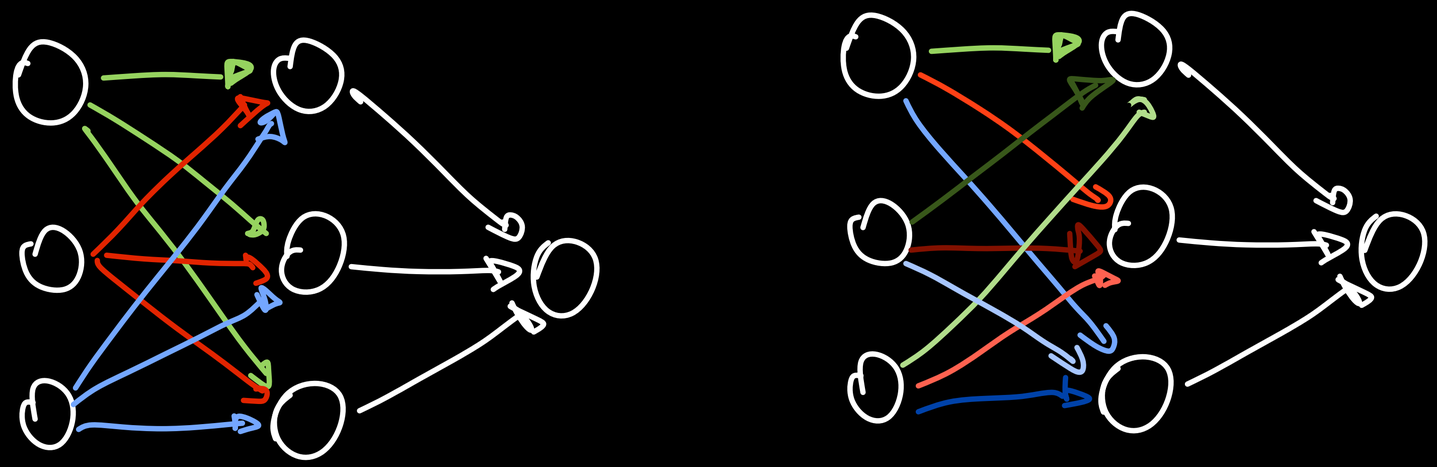
Neural Network Zoo



Parameters Initialization

How to inizialize the parameters matrix $\Theta^{(l)}$?

Symmetry breaking: the parameters depending from the same nods have to be inizialized with different values (small! $\Theta^{(l)}_{ji}$ usually $\in [-\varepsilon, \varepsilon]$)

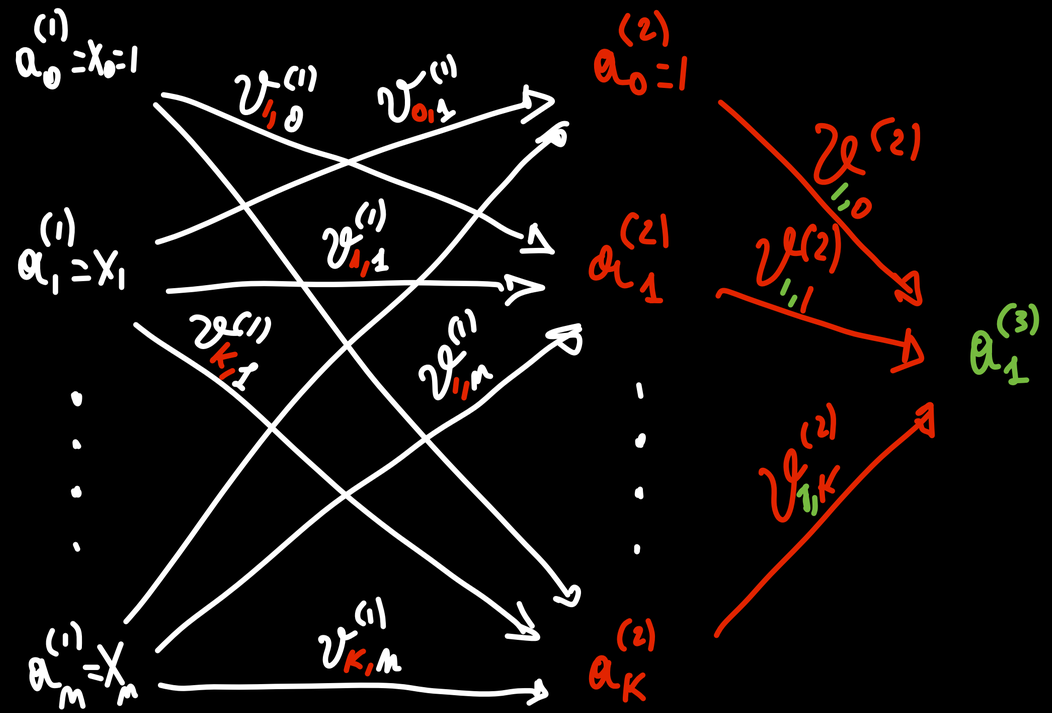


PROBLEM: HOW TO LEARN PARAMETERS IN A NEURAL NETWORK WITH A NUMBER OF HIDDEN LAYER ≥ 1?

HOW TO COMPUTE THE GRADIENT (DERIVATIVES) WITH RESPECT TO PARAMETERS!

How to learn parameters?

- one neuron ✓
  - ⇒ gradient descent (evolution of Rosenblatt learning algorithm)
- softmax ✓
  - ⇒ gradient descent
  - loss function acts on 1 layer
- NN with hidden layer?



values of hidden nodes are not known in advance ≡ are not given with training set

⇒ gradient descent should be revised ⇒ backpropagation

HENCE OUR PROBLEM IS HOW TO COMPUTE

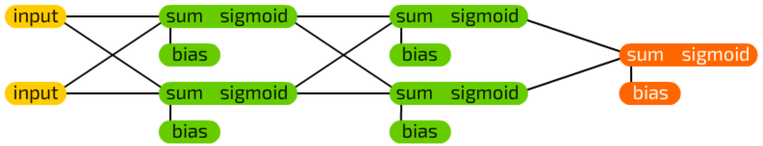
$$\frac{\partial J(\theta)}{\partial \theta_{ji}^{(l)}} \quad \forall j, i, l$$

SOLUTION IS BASED ON THE FACT THAT NEURAL NETWORKS ARE COMPUTATIONAL GRAPHS AND WE CAN USE CHAIN RULE TO DERIVE $\Rightarrow$ BACKPROPAGATION

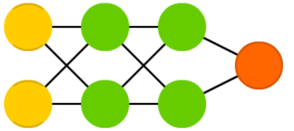
COMPUTATIONAL GRAPHS

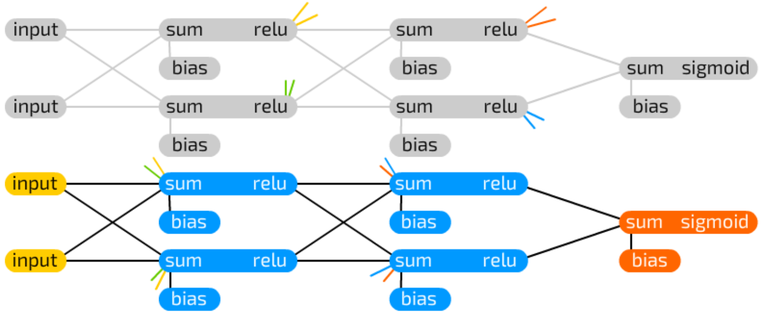
An informative chart to build

Neural Network Graphs

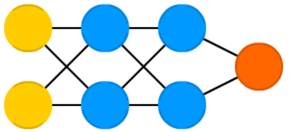


Deep Feed Forward Example

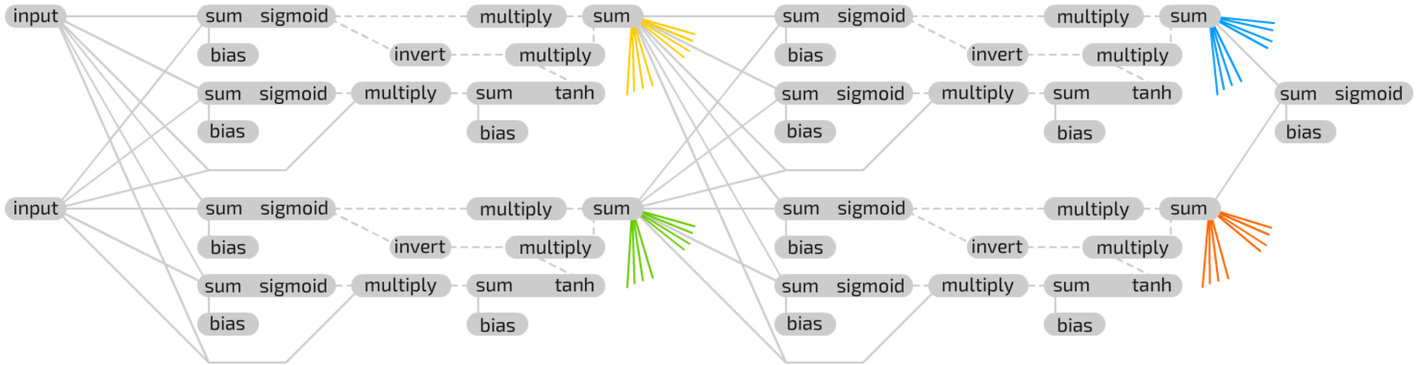




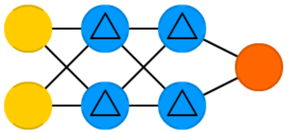
Deep Recurrent Example
(previous iteration)

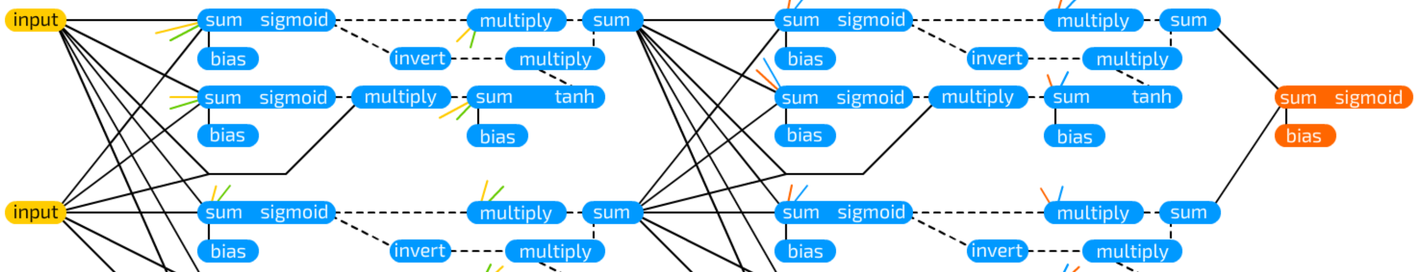


Deep Recurrent Example



Deep GRU Example
(previous iteration)





Deep GRU Example