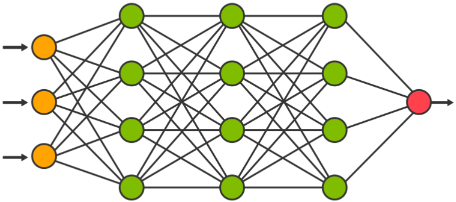
Introduction to Neural Networks

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology



Biological Analogy

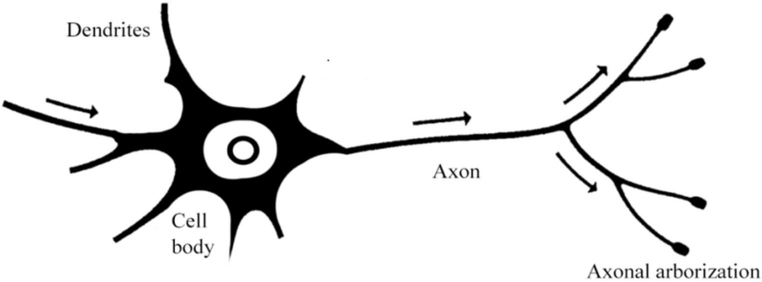


Figure: Anatomy of a biological neuron [1].

Activation Functions

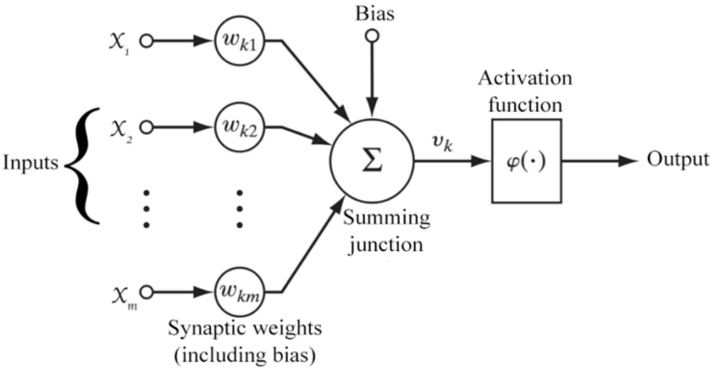


Figure: Neural network neuron [1].

Activation Functions

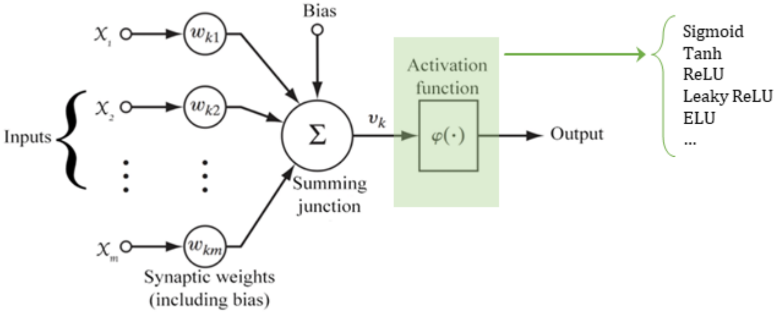
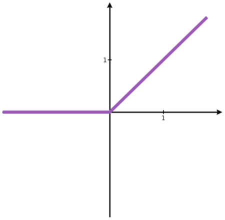
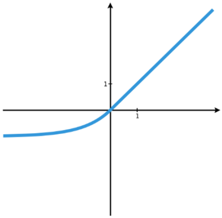
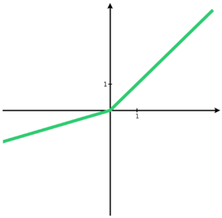


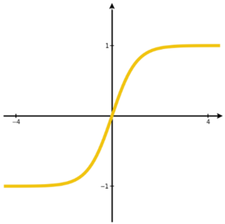
Figure: Activation function

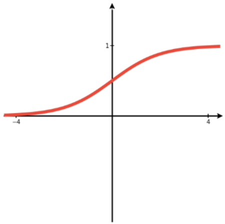
Activation Functions

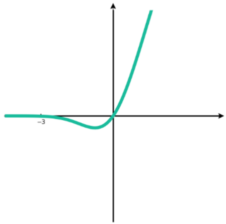
ReLU	ELU	Leaky ReLU
$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$	$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^{x} - 1) & x \leq 0 \end{cases}$	$f(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$
		

Activation Functions

Tanh	Sigmoid	GELU
$f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	$\sigma(x) = \dfrac{1}{1 + e^{-x}}$	$f(x) = \dfrac{1}{2}x\left(1 + \text{erf}\left(\dfrac{x}{\sqrt{2}}\right)\right)$







<u>Softmax</u>

$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad i = 1, \cdots, J$$

Gradient Descent

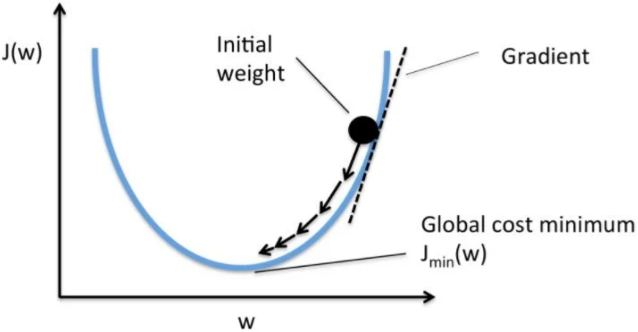


Figure: Gradient descent [3].

Gradient Descent

- Let's define our problem:
  - We have dataset $\mathcal{D} = \{x^i, y^{(i)}\}_{i=1}^n$.
  - f is a single layer perceptron.
  - Define $\hat{y}^{(i)} = f(x^{(i)})$.
- We want to minimize following cost function:

$$\mathcal{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

- We are going to use gradient descent algorithm. $\boldsymbol{w}$ will be updated as follows:

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta\nabla_{\boldsymbol{w}}\mathcal{J}$$

Gradient Descent

- Let's find $\nabla_{\boldsymbol{w}} \mathcal{J}$:

$$
\begin{aligned}
\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y^{(i)} - \hat{y}^{(i)})^2 \\
&= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (y^{(i)} - \hat{y}^{(i)})^2 \\
&= \frac{1}{2} \sum_i 2(y^{(i)} - \hat{y}^{(i)}) \frac{\partial}{\partial w_j} (y^{(i)} - \hat{y}^{(i)}) \\
&= \sum_i (y^{(i)} - \hat{y}^{(i)}) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_j w_j x_j^{(i)} \right) \\
&= \sum_i (y^{(i)} - \hat{y}^{(i)}) (-x_j^{(i)})
\end{aligned}
$$

$$
\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = -\eta \sum_i (y^{(i)} - \hat{y}^{(i)}) (-x_j^{(i)}) = \eta \sum_i (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}
$$

$$
\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}
$$

Limitations of SLP

- What are the limitations of SLP?
- Can we learn all functions with SLP?

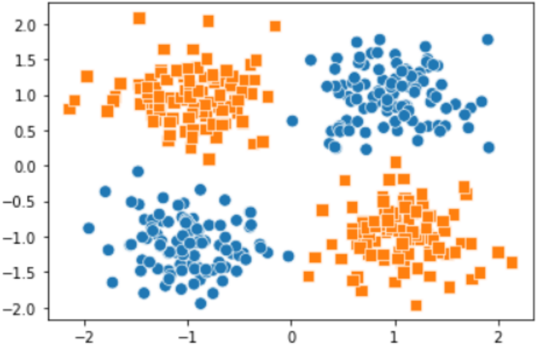


Figure: The XOR function is not linear separable.

Limitations of SLP

- As we saw in the XOR case, nonlinear separable functions can not be learned by SPLs.

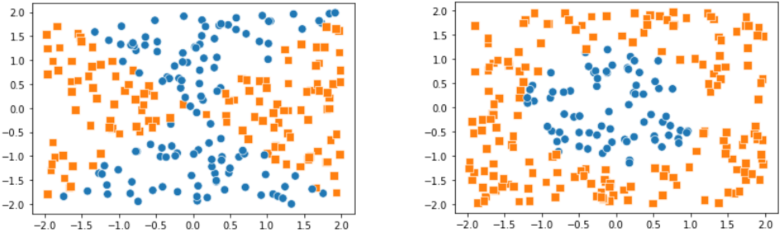


Figure: Examples of nonlinear separable functions.

- How to solve this?

Limitations of SLP

- What if we knew some feature space which our data is linear separable in?!

(x, y) $\Longrightarrow$ (r, θ)

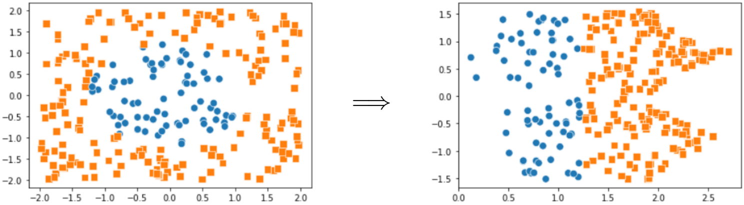


Figure: Data is linear separable after transformation.

Multi-Layer Perceptron

- So if we know some $f_1, \cdots, f_4$ we can use SLP to solve problem.

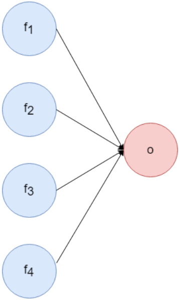


Figure: Using feature space f to solve problem.

- How to learn this f_is? Use SLP!

Multi-Layer Perceptron

- We can use inputs (x_i) to learn features (f_i)

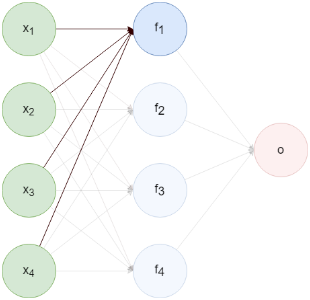


Figure: Using inputs to learn features.

- What if f_is are not sufficient? We can add more layers!

Multi-Layer Perceptron

- Adding more layers we will have a bigger network.

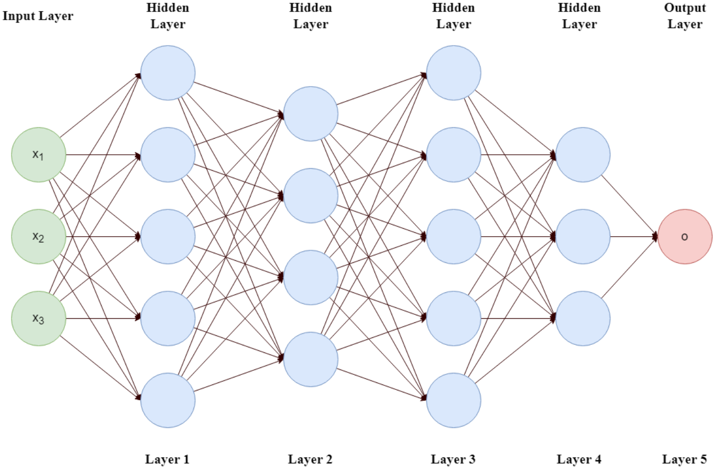


Figure: A 5 layer MLP.

Architecture of MLPs

- Important questions:
  - How many hidden layer should we have?
  - In each hidden layer, how many neuron should we have?

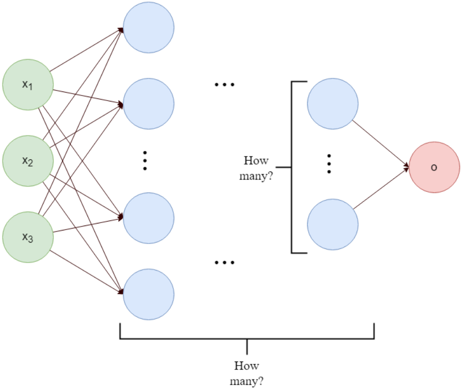


Figure: How many layers and neurons should we have?

Architecture of MLPs

- In practice:
  - You have limited resources
  - There is no universal rule to choose this hyperparameters
  - Need to experiment for different number of layers and neurons in each layer

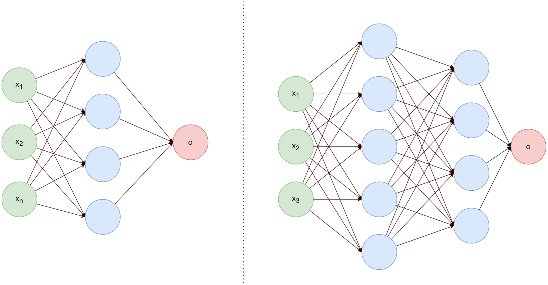


Figure: Experiment for different architecture and choose the best model.

Activation Function of Hidden Layers

- One can use any activation function for each hidden units
- Usually people use the same activation function for all neurons in one layer
- The important point is to use nonlinear activation functions

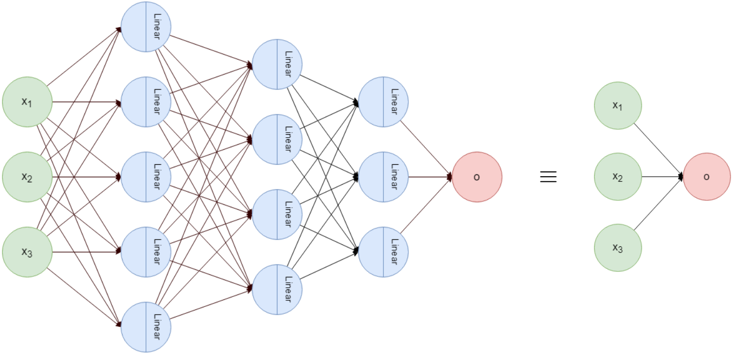


Figure: MLP with linear activation functions is equivalent to simple SLP.

XOR problem

- Now let's solve XOR problem with MLPs.
- We have two binary inputs, build an MLP to calculate their **XOR**.

- First let's build logical **AND** and **OR** functions.

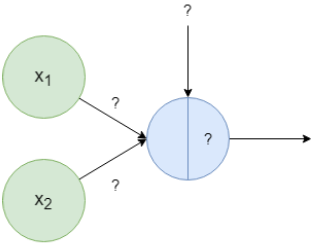
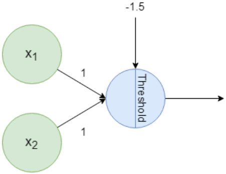
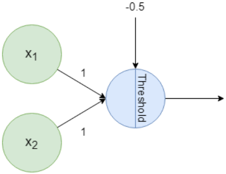


Figure: We need to find weights, biases and activation function.

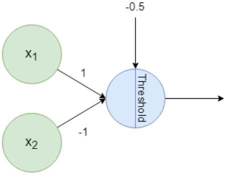
XOR problem



(a) $x_1 \wedge x_2$

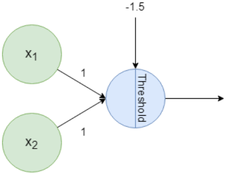


(b) $x_1 \vee x_2$

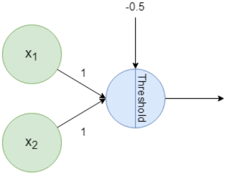


(c) $x_1 \wedge \overline{x_2}$

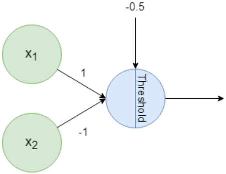
XOR problem



(a) $x_1 \land x_2$



(b) $x_1 \lor x_2$



(c) $x_1 \land \overline{x_2}$

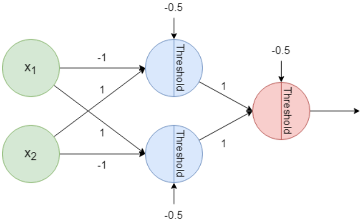
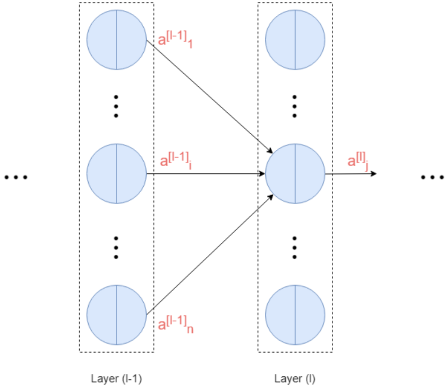


Figure: MLP for XOR function.

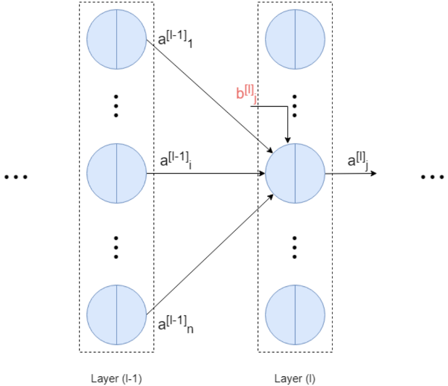
MLP notation

- $a_i^{[l]}$: i-th neuron outpu in layer l
- $\boldsymbol{a}^{[l]}$: layer l output in vector form



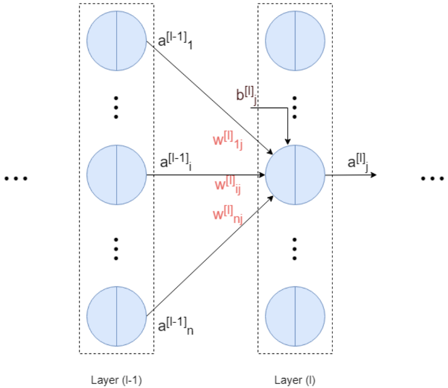
MLP notation

- $b_i^{[l]}$: i-th neuron biase in layer l
- $\boldsymbol{b}^{[l]}$: layer l biases in vector form



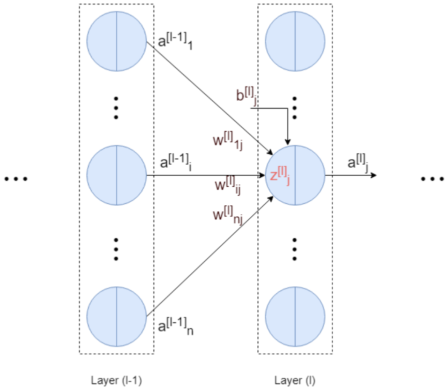
MLP notation

- $W_{ij}^{[l]}$: weight of the edge between i-th nuron in layer $l-1$ and j-th neuron in layer l



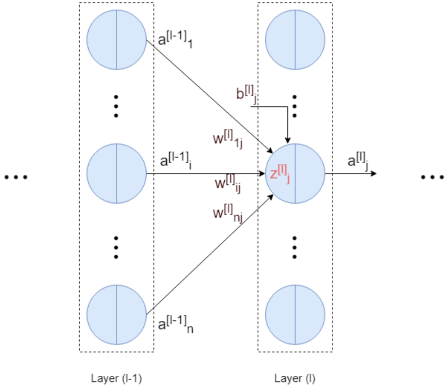
MLP notation

- $z_j^{[l]}$: j-th neuron input in layer l
- $z_j^{[l]} = b_j^{[l]} + \sum_{i=1}^{n} W_{ij}^{[l]} a_i^{[l-1]}$



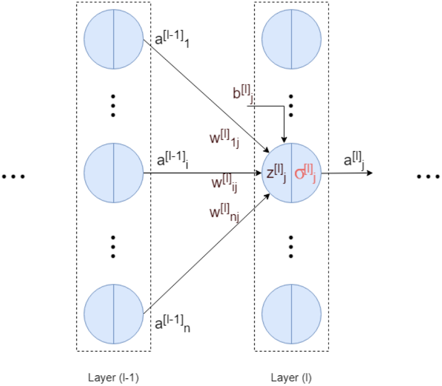
MLP notation

- $\boldsymbol{z}^{[l]}$: input of layer l in vector form
- $\boldsymbol{z}^{[l]} = \boldsymbol{b}^{[l]} + (W^{[l]})^T \boldsymbol{a}^{[l-1]}$



MLP notation

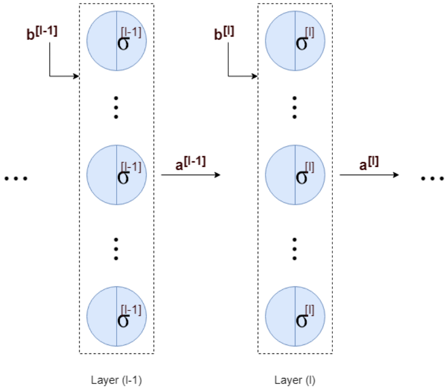
- $\sigma_j^{[l]}$: j-th neuron activation function in layer l



MLP notation

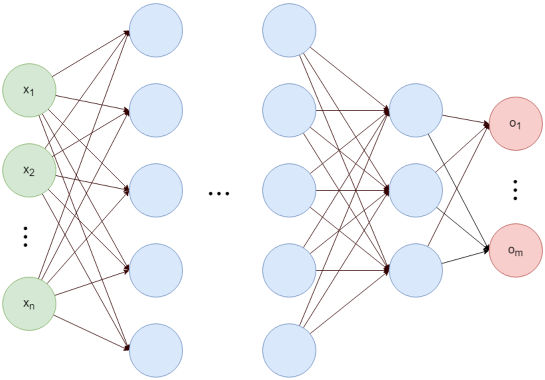
- If we assume all neurons in one layer have the same activation function then:

$$\boldsymbol{a}^{[l]} = \sigma^{[l]}\left(\boldsymbol{b}^{[l]} + (W^{[l]})^T\boldsymbol{a}^{[l-1]}\right)$$



MLP notation

- So for a network with L layer, and $\boldsymbol{x}$ as its input we will have:

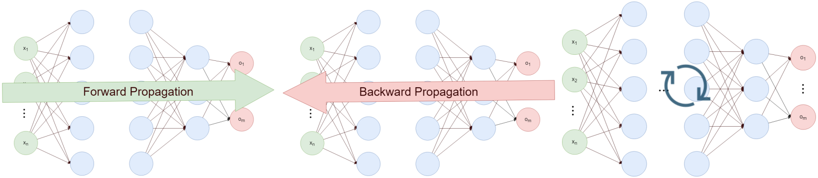


$$\boldsymbol{o} = \boldsymbol{a}^{[L]} = \sigma^{[L]}\left(\boldsymbol{b}^{[L]} + (W^{[L]})^T \sigma^{[L-1]}\left(\cdots \sigma^{[1]}\left(\boldsymbol{b}^{[1]} + (W^{[1]})^T \boldsymbol{x}\right)\cdots\right)\right)$$

Learning MLPs

- Till here we have used networks with predefined weights and biases.
- How to learn these parameters?

- The idea is to use gradient descent



(a) Forward pass (b) Backward pass (c) Update parameters

Thank You!

Any Question?

References

M. D. Ergün Akgün, "Biological and neural network neuron," 2018.
https://www.researchgate.net/publication/326417061_Modeling_Course_Achievements_of_Elementary_Education_Teacher_Candidates_with_Artificial_Neural_Networks.

L. Nalborczyk, "A gentle introduction to deep learning in r using keras," 2021.
https://www.barelysignificant.com/slides/vendredi_quanti_2021/vendredi_quantis#1.

"Gradient descent."
https://subscription.packtpub.com/book/big-data-&-business-intelligence/9781788397872/1/ch01lvl1sec22/gradient-descent.

F.-F. L. . J. J. . S. Yeung, "Training neural networks," 2018.
http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf.

I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*.
MIT Press, 2016.
http://www.deeplearningbook.org.

K. Katanforoosh and D. Kunin, "Initializing neural networks," 2019.
https://www.deeplearning.ai/ai-notes/initialization/.