Artificial Intelligence

COMP3411/9814

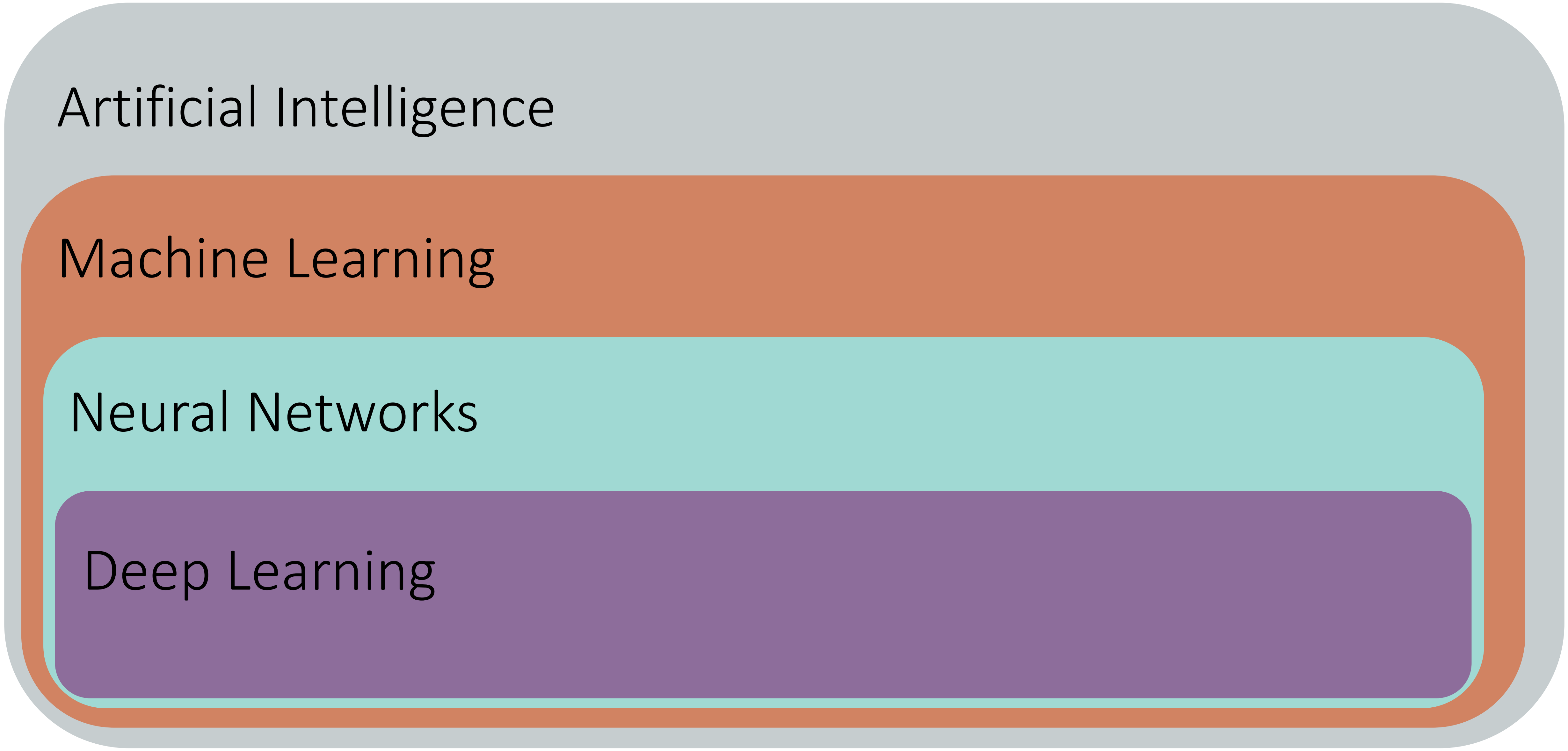
Week 3, Term 3, 2025

Maryam Hashemi (m.Hashemi@unsw.edu.au)

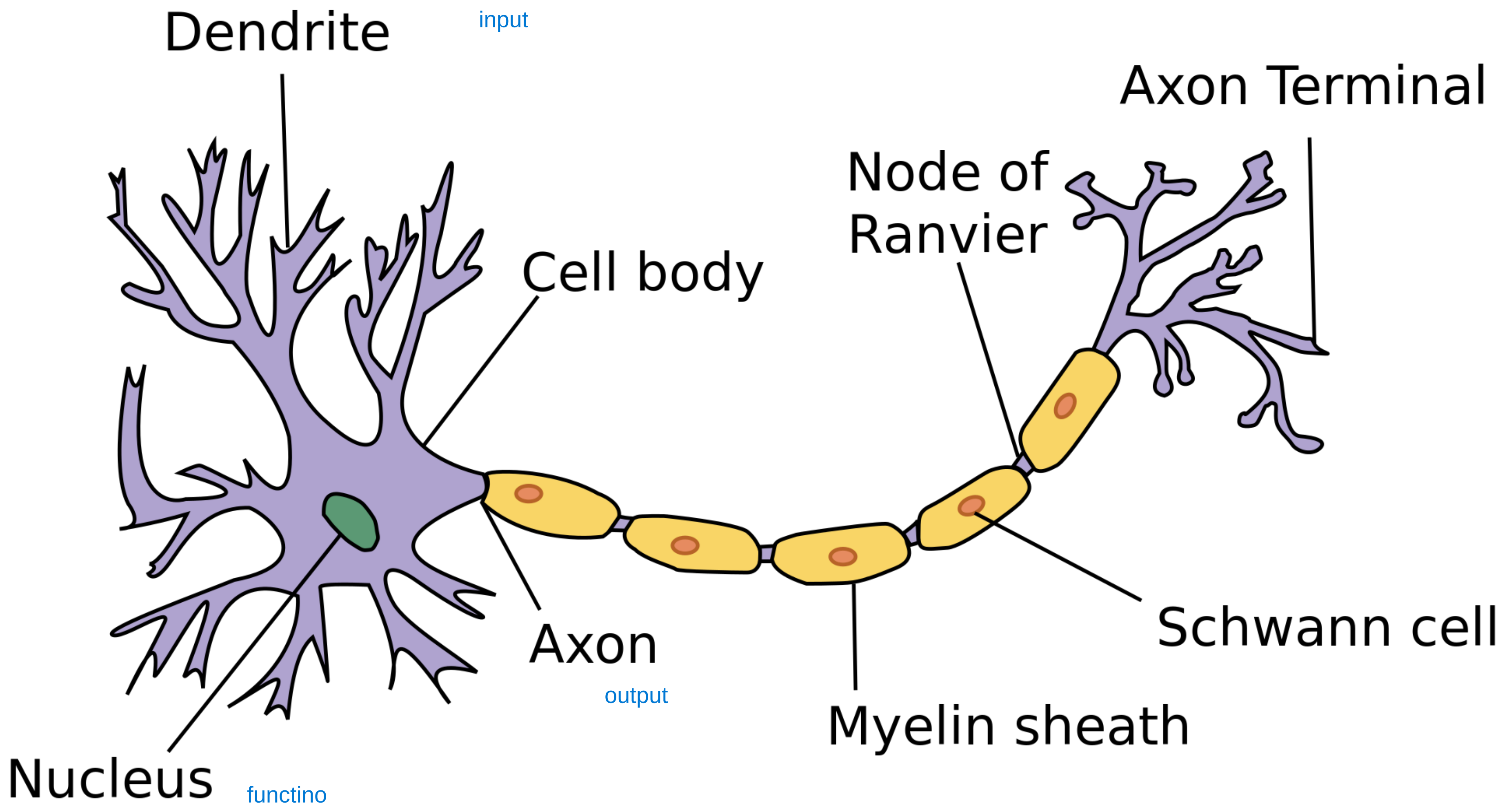
Overview

- ✓**Biological Neurons.**
- ✓Artificial Neurons.
  - ✓Perceptron.
- ✓Multi Layer Perceptron.
- ✓Learning with Gradient Descent and Backpropagation.
- ✓Neural Networks Design.
- ✓Deep Learning.

Neural Networks



Neural Networks, Biological Neurons



Neural Networks, Biological Neurons

The brain is made up of neurons (nerve cells) which have:

- a cell body (soma)
- dendrites (inputs)
- an axon (outputs)
- synapses (connections between cells)

When the inputs reach some threshold an action potential (electrical pulse) is sent along the axon to the outputs.

Neural Networks, Biological Neurons

- Human brain has 100 billion neurons (~ 10^{10} - 10^{11} neurons) with an average of 10,000 synapses each (some even with 100,000 synapses).

- Latency is about 3-6 milliseconds.

- At most a few hundred “steps” in any mental computation, but massively parallel.

- Can we imitate the behavior of biological neurons to make artificial neurons?

Neural Networks, Biological Neurons

- Ability of a neuron (or neural net) to adjust connections (weights) to obtain the intended output or that meets certain criteria is called ***learning***.

- Hebbian learning (1949): when a neuron *A* persistently activates another nearby neuron *B*, the connection between the two neurons becomes stronger. Specifically, a growth process occurs that increases how effective neuron *A* is in activating neuron *B*. As a result, the connection between those two neurons is strengthened over time.
    - "Neurons that fire together, wire together", Hebb.

Neural Networks, Biological Neurons

Artificial Neural Networks:

- An information-processing architecture loosely modelled after the brain.
- This model consists of many interconnected processing units (**neurons**) that work in parallel to accomplish a global task.
- Commonly used to model relationships between inputs and outputs or to discover patterns in data.
- Characterized by: Number of neurons, Interconnection architecture, Weight values, Activation and transfer functions.
- **Learning** occurs by adapting weights, architecture, and activation/transfer functions to improve performance.

Overview

- ✓Biological Neurons.
- ✓**Artificial Neurons.**
    - ✓**Perceptron.**
- ✓Multi Layer Perceptron.
- ✓Learning with Gradient Descent and Backpropagation.
- ✓Neural Networks Design.
- ✓Deep Learning.

Neural Networks, Artificial Neurons

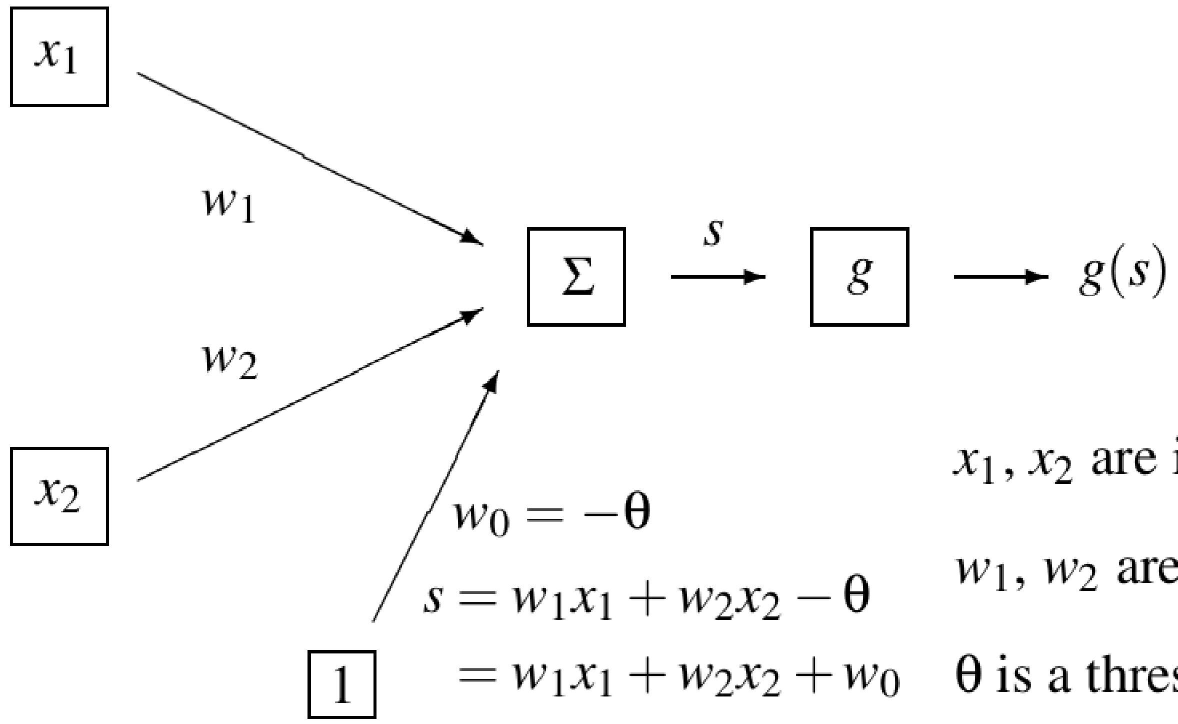
In 1943, McCulloch and Pitts published a simplified mathematical **model of a neuron**. Each input x_i is multiplied by a corresponding weight w_i and the results are added together. An extra constant w_0 called the **bias** is also added.

$$s = w_0 + \sum_i w_i x_i$$

Finally, a non-linear **transfer function** is applied to this linear combination to produce the output of the neuron.

$$z = g(s) = g(w_0 + \sum_i w_i x_i)$$

Neural Networks, Artificial Neurons



$$
\begin{aligned}
s &= w_1x_1 + w_2x_2 - \theta \\
&= w_1x_1 + w_2x_2 + w_0
\end{aligned}
$$

x_1, x_2 are inputs

w_1, w_2 are synaptic weights

θ is a threshold

w_0 is a **bias** weight

g is transfer function

Neural Networks, Artificial Neurons

(Artificial) Neural Networks are made up of nodes which have:

- ➢ Input edges, each with some weight
- ➢ Output edges (with weights)
- ➢ An activation level (a function of the inputs)

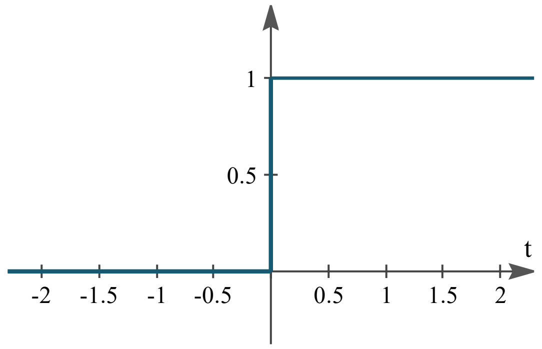
- Weights can be positive or negative and may change overtime **(learning).**
- The input function is the weighted sum of the activation levels of inputs.
- The activation level is a **non-linear transfer function (activation function),** *g,* of this input.

$$z = g(s) = g(w_0 + \sum_i w_i x_i)$$

Neural Networks, Perceptron

Originally, a (discontinuous) step function:

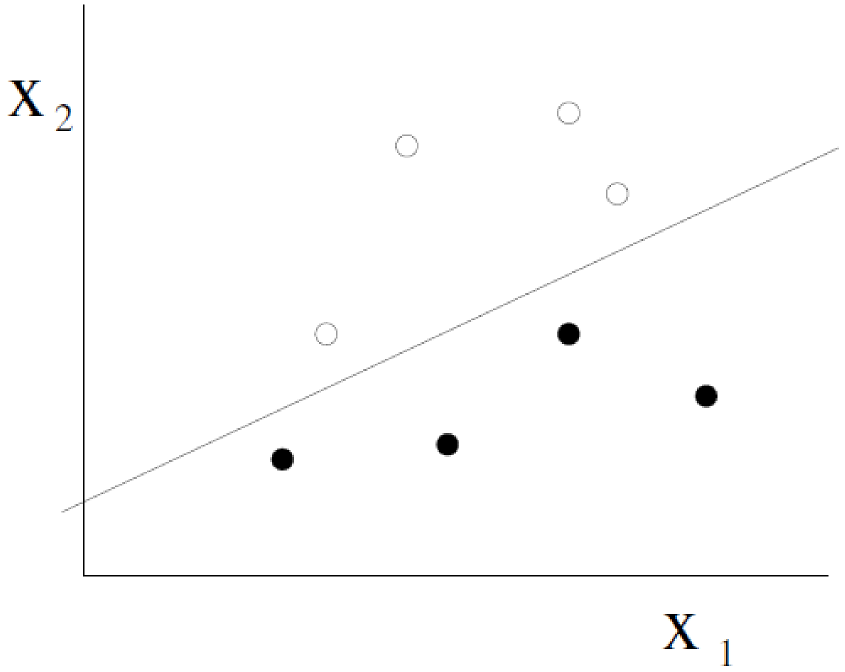
$$g(s) = \begin{cases} 1 & \text{if} \quad s \geq 0 \\ 0 & \text{if} \quad s < 0 \end{cases}$$



- An artificial neuron with **step transfer function** is called a **Perceptron**.
- The bias (w_0) was thought of as a kind of **threshold**. If the combination $-\sum_i w_i x_i$ is less than this threshold, the neuron would "fire" (its output would be 1).
- The higher the bias, the more likely the neuron is to fire.
- Later on, alternative transfer functions were introduced which are continuous and (mostly) differentiable.

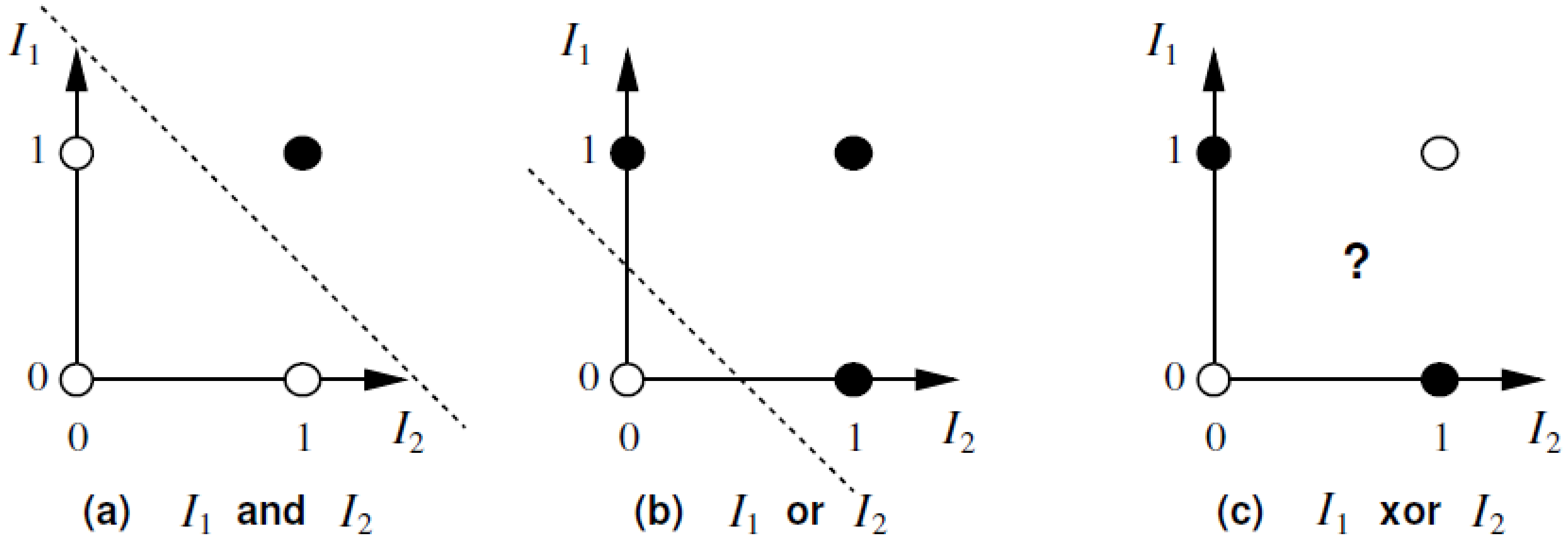
Neural Networks, Perceptron

- The **weights and bias** of a Perceptron define a **hyperplane** that divides the input space into two regions.
  - For inputs on one side of the hyperplane, the output is **0**.
  - For inputs on the other side, the output is **1**.
- Functions that can be computed in this way are called **linearly separable**. With this structure, a perceptron (artificial neuron) can learn **any linear relationship**.
- **Limitation:** What happens if the data is **not linearly separable**?



Neural Networks, Perceptron

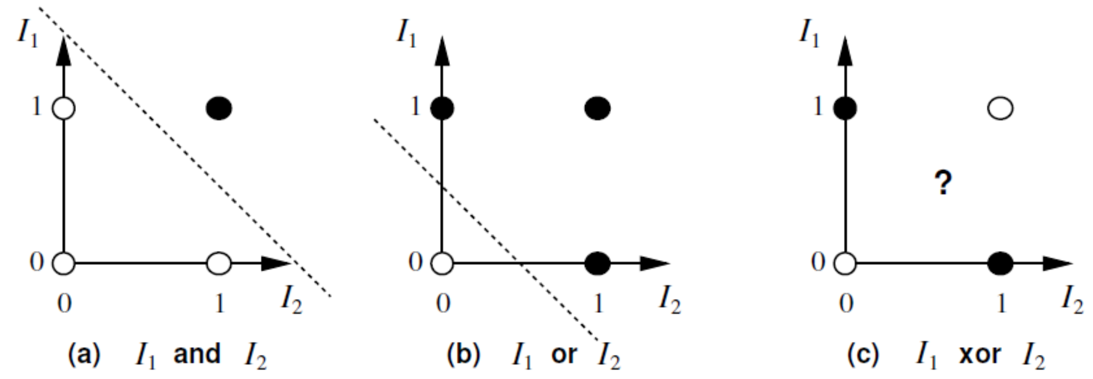
- If we adopt the convention that 0 = False and 1 = True, many simple logical functions such as AND, OR, and NOT become linearly separable.
- Using plane geometry, we can design perceptron to compute these functions.
- This diagram illustrates the AND and OR functions, which can be computed by perceptron, as well as the Exclusive OR (XOR) function, which cannot be computed by a single perceptron because it is not linearly separable.
- For the AND and OR functions, what would the weights be (w_0, w_1, w_2)?



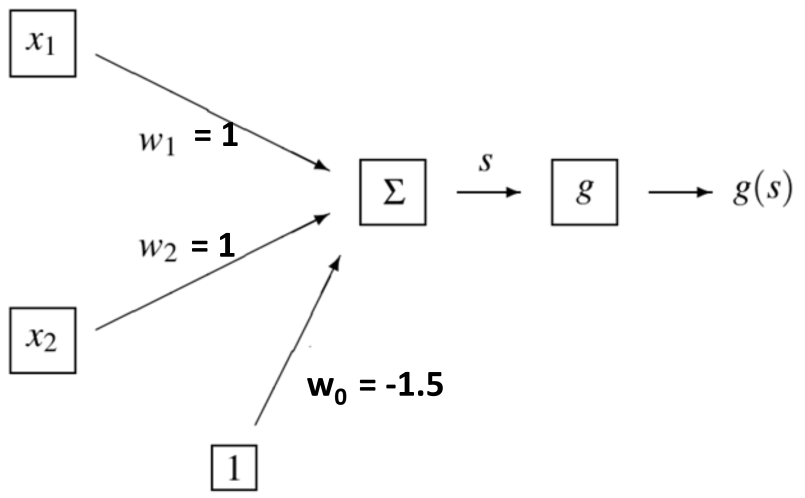
Neural Networks, Perceptron

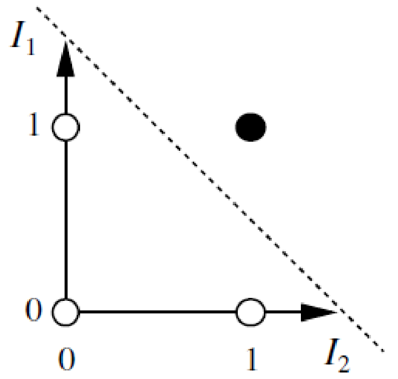
- If we adopt the convention that 0 = False and 1 = True, many simple logical functions such as AND, OR, and NOT become linearly separable.
- Using plane geometry, we can design perceptron to compute these functions.
- This diagram illustrates the AND and OR functions, which can be computed by perceptron, as well as the Exclusive OR (XOR) function, which cannot be computed by a single perceptron because it is not linearly separable.

AND	$w_0 = -1.5,\ w_1 = w_2 = 1$
OR	$w_0 = -0.5,\ w_1 = w_2 = 1$



Neural Networks, Perceptron



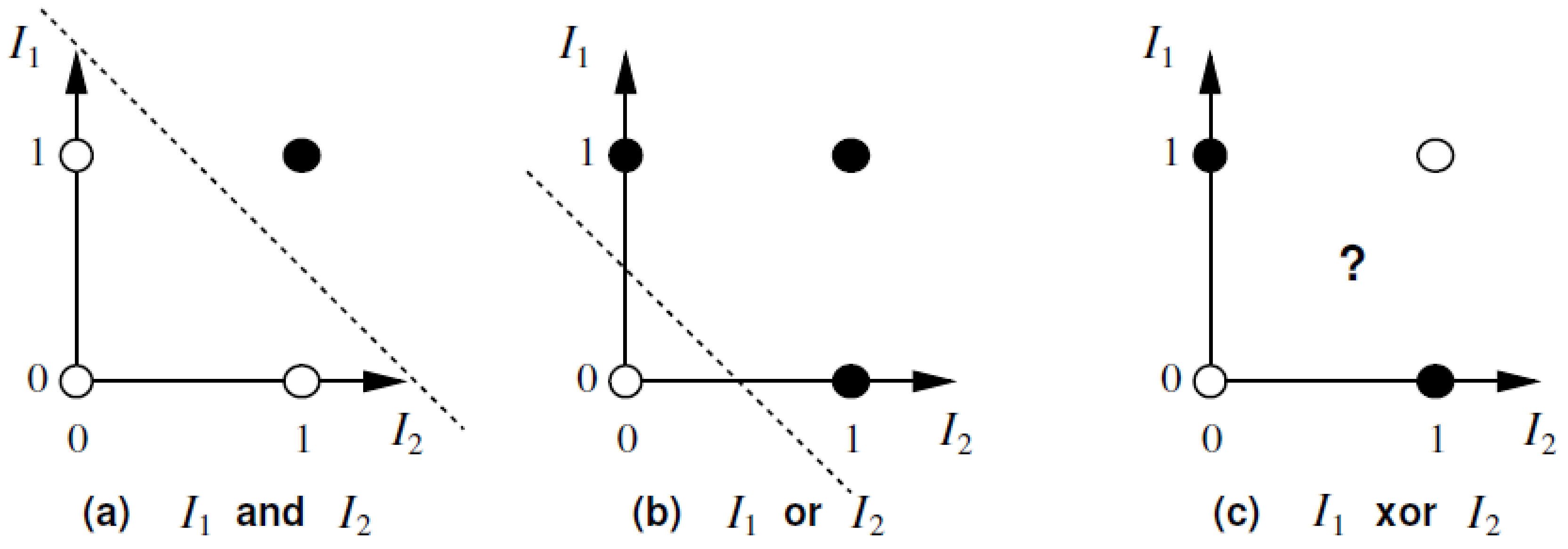


(a) I_1 and I_2

X_1	X_2	s	Output
0	0	$w_0 + \sum_i w_i x_i$ = -1.5 + 1*0 + 1*0= -1.5	g(s)= g(-1.5)= 0
1	0	$w_0 + \sum_i w_i x_i$ = -1.5 + 1*1 + 1*0= -0.5	g(s)= g(-0.5)= 0
0	1	$w_0 + \sum_i w_i x_i$ = -1.5 + 1*0 + 1*1= -0.5	g(s)= g(-0.5)= 0
1	1	$w_0 + \sum_i w_i x_i$ = -1.5 + 1*1 + 1*1= 0.5	g(s)= g(0.5)= 1

Neural Networks, Perceptron

- A Perceptron can compute simple logical functions such as AND and OR.
- But how can we compute more complex functions, such as XOR?

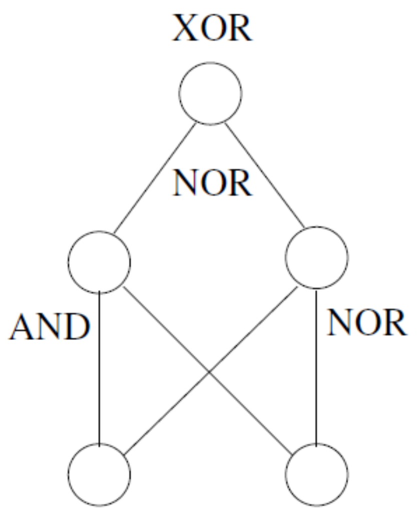


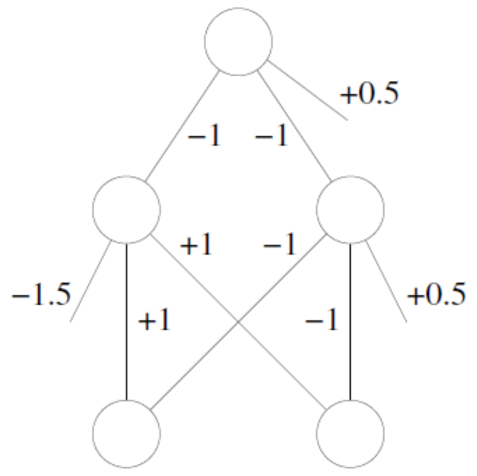
Overview

- ✓Biological Neurons.
- ✓Artificial Neurons.
    - ✓Perceptron.
- ✓**Multi Layer Perceptron.**
- ✓Learning with Gradient Descent and Backpropagation.
- ✓Neural Networks Design.
- ✓Deep Learning.

Neural Networks, Multi Layer Perceptron

- It has been known since the 1960s that any logical function (AND, OR, XOR, etc.) can be implemented using a **multi-layer perceptron** with step function activations.
- A general solution was proposed by Paul Werbos in 1974 and was rediscovered in 1986 by Rumelhart, Hinton, and Williams.
- For example, to compute XOR, one approach is to rewrite it in terms of linearly separable functions such as AND, OR, and NOR, and then arrange **several perceptron** into a network that combines these functions appropriately.



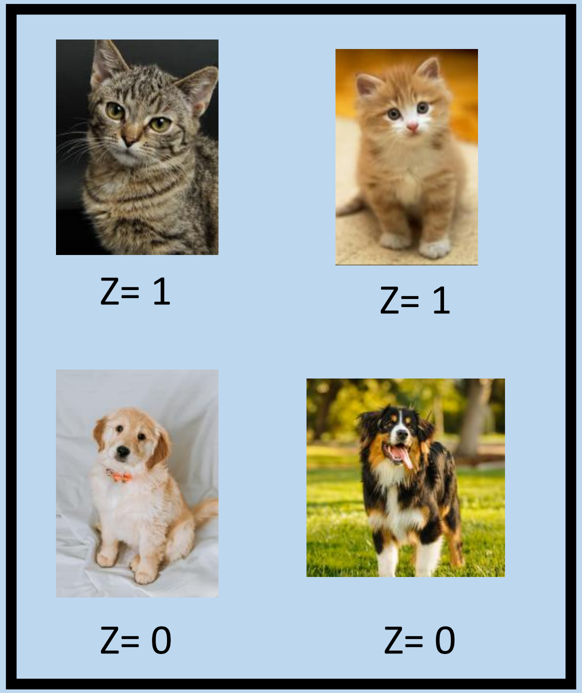


Neural Networks, Multi Layer Perceptron

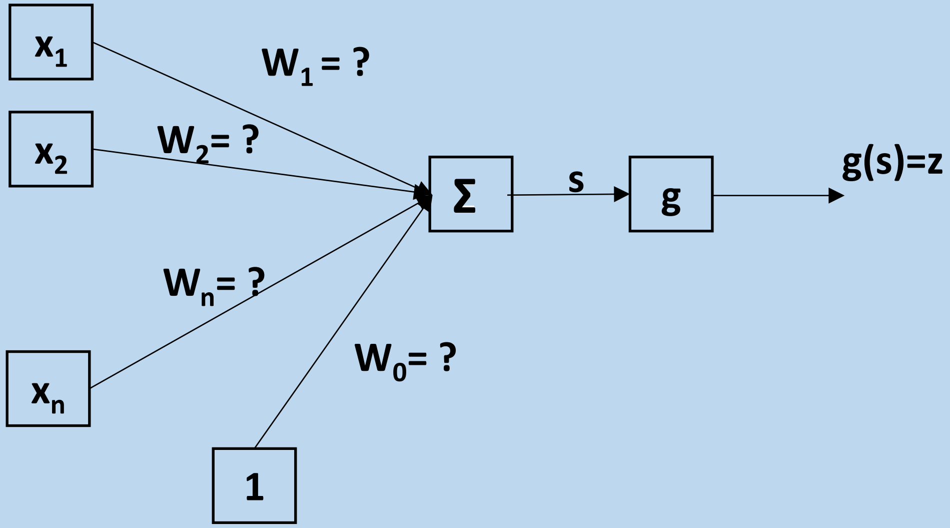
- So far, we have seen how logical functions such as AND, OR, XOR can be implemented using single-layer or multi-layer perceptron.
- However, in practice, we usually deal with **raw data** rather than explicit logical expressions.
- What we really need is a method—analogous to the perceptron learning algorithm—that can **learn the weights of a neural network** from a set of training examples.
- How does learning happens?

Neural Networks, Multi Layer Perceptron

In learning, the network finds the right w_i values in order to make the right prediction.



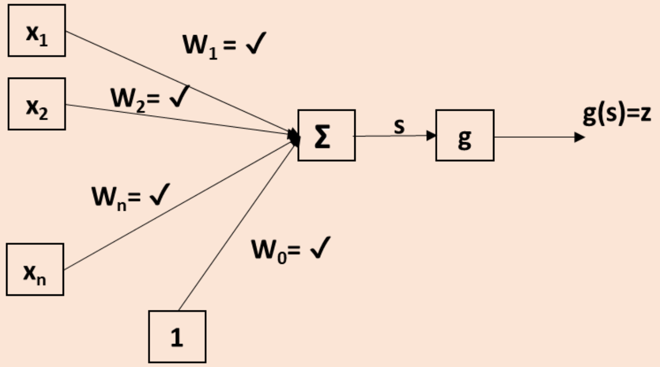
Dataset



Learning Phase

After training, based on w_i values, model can make a prediction for a new unseen data.

Z= ?



Z= 1

After Learning

Overview

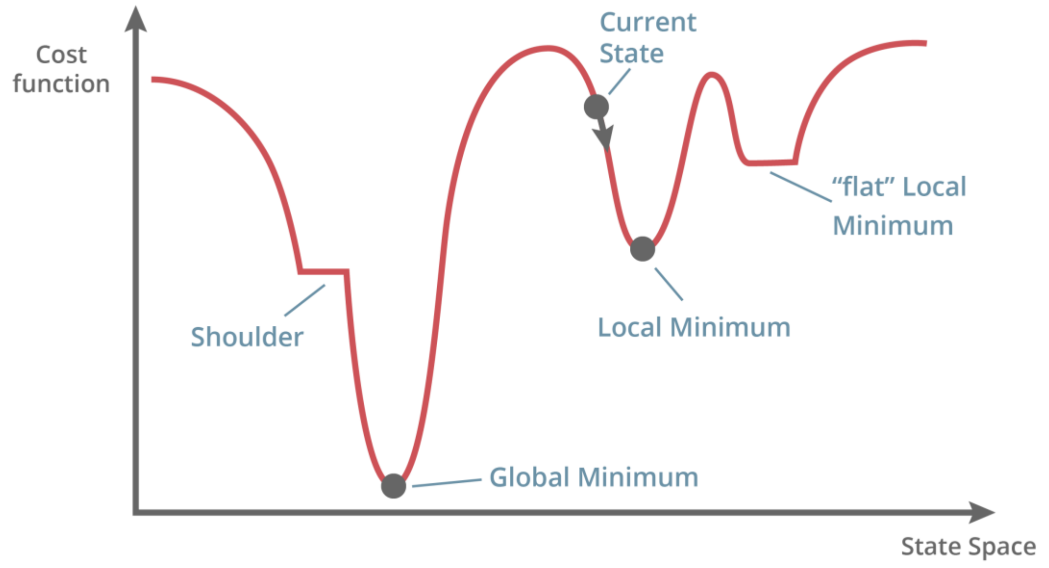
- ✓Biological Neurons.
- ✓Artificial Neurons.
    - ✓Perceptron.
- ✓Multi Layer Perceptron.
- ✓**Learning with Gradient Descent and Backpropagation.**
- ✓Neural Networks Design.
- ✓Deep Learning.

Neural Networks, Learning

- As early as the 1960s, engineers understood how to use **gradient descent** to optimize over a family of continuous and differentiable functions. The basic idea is as follows:
- We define an **error function** (also called a loss function or cost function) E as half the sum, over all input items, of the square of the difference between the actual output z_i and the target output t_i:

$$\boldsymbol{E = \frac{1}{2}\sum_i (z_i - t_i)^2}$$

The goal is to **find the minimum of the error function** E, and minimum of a function can be calculated through its **derivative**.



Neural Networks, Learning

- We can use multi-variable derivative to adjust the weights in such a way as to take us in the steepest downhill direction.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{\eta} \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}},$$ η is called the **learning rate**.

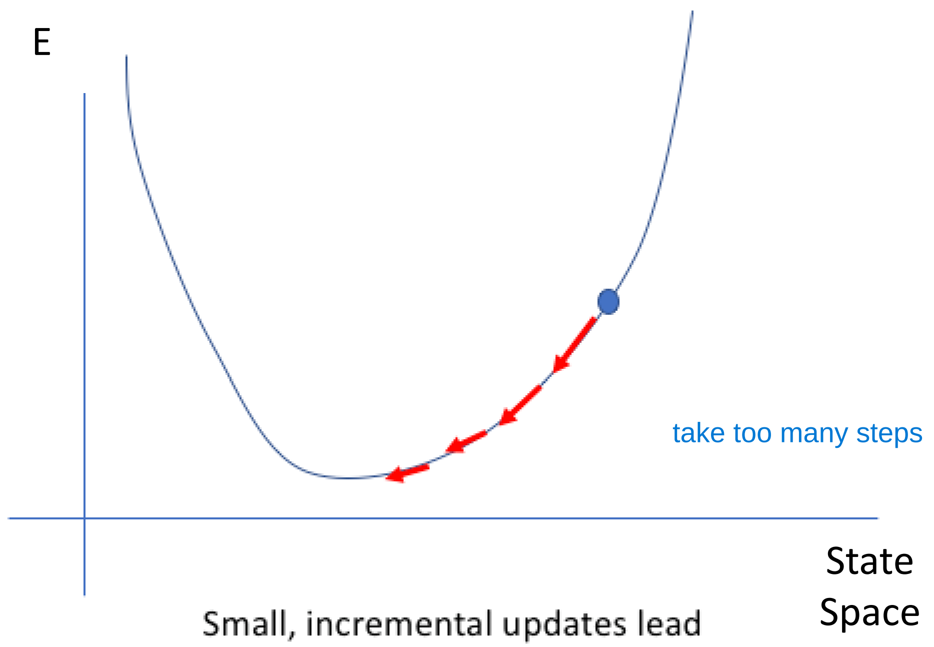
- If we use the step function as the transfer function, the error landscape is not smooth; it consists almost entirely of flat regions and "shoulders," with occasional discontinuous jumps.

- For a single-layer perceptron, this was not a problem, but for networks with two or more layers, it becomes a major obstacle.

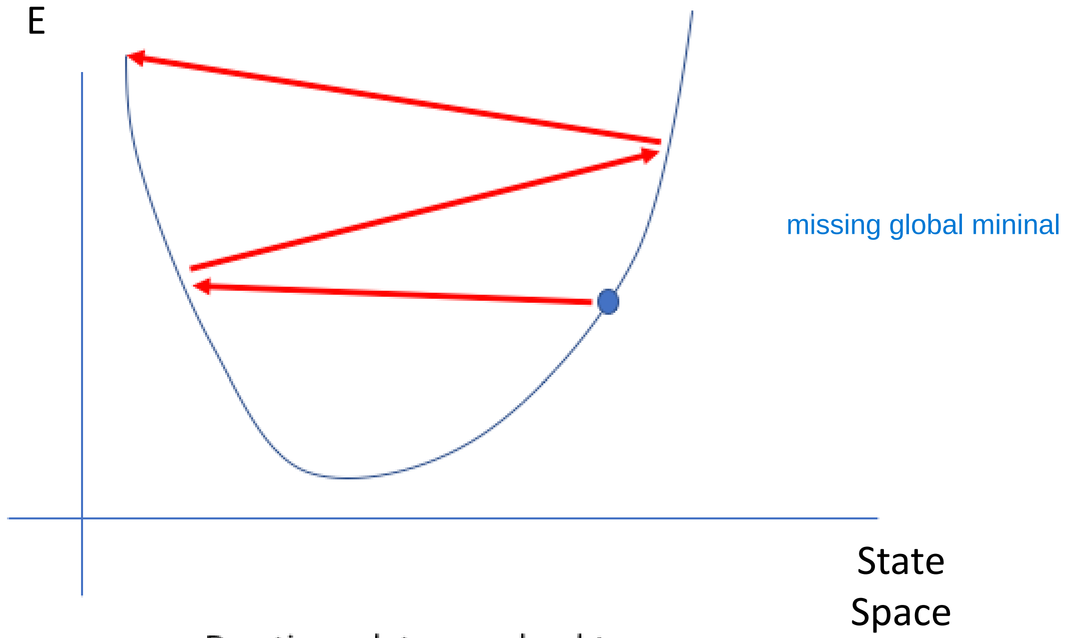
- To apply gradient descent successfully, neural networks needed to be redesigned so that the function from input to output would be **smooth and differentiable**. This was achieved by Paul Werbos in 1975 and later popularized by Rumelhart et al. in 1986.

Neural Networks, Learning

- Learning rate identify the step size.



Small, incremental updates lead us toward converging on the optimal value



Drastic updates can lead to divergent behaviors

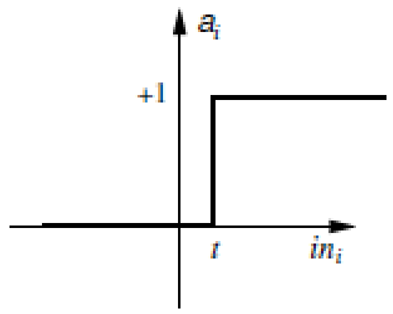
Neural Networks, Learning

- The key idea is to replace the (discontinuous) step function with a differentiable function, such as the sigmoid:

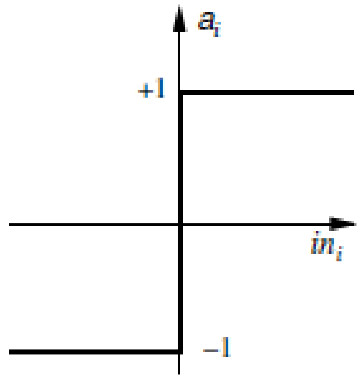
$$g(s)= \frac{1}{1+e^{-s}}$$

- or hyperbolic tangent:

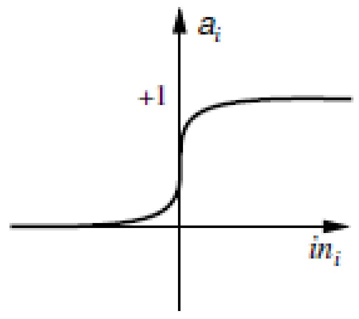
$$g(s)= tanh(s)= \frac{e^{s} - e^{-s}}{e^{s} + e^{-s}} = 2\left(\frac{1}{1+e^{-2s}}\right) - 1$$



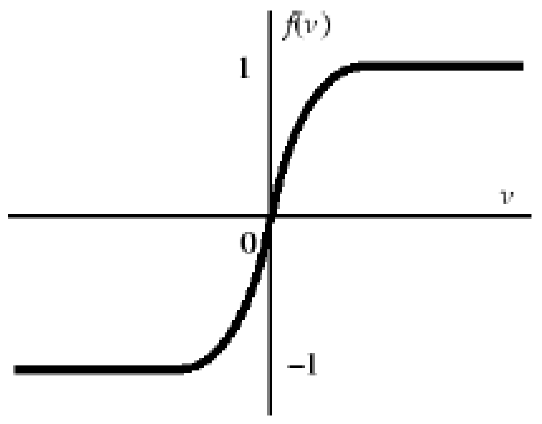
(a) Step function



(b) Sign function



(c) Sigmoid function

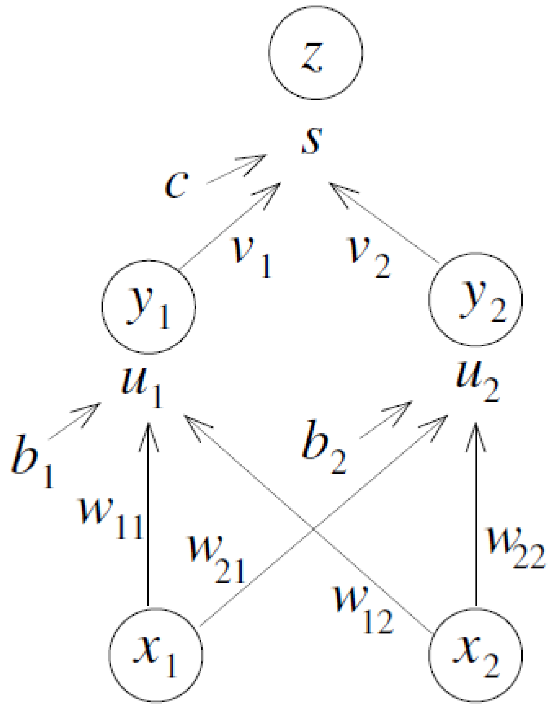


Neural Networks, Learning

- We now describe how to compute the **partial derivatives of the loss function with respect to each weight**.
- For illustration, we consider a **two-layer neural network** with **sigmoid activation** at the hidden layer, as shown in the diagram.

Here, x_i are the **inputs**, y_i are the **hidden units**, w_{ij} and v_i are the **weights**, and b_i and c are the **biases**.

$u_1 = b_1 + w_{11}x_1 + w_{12}x_2$
$y_1 = g(u_1)$
$s = c + v_1 y_1 + v_2 y_2$
$z = g(s)$



Neural Networks, Learning

Chain Rule:

- If *y= y(u,v)* where *u= u(x)* and *v= v(x)* then:

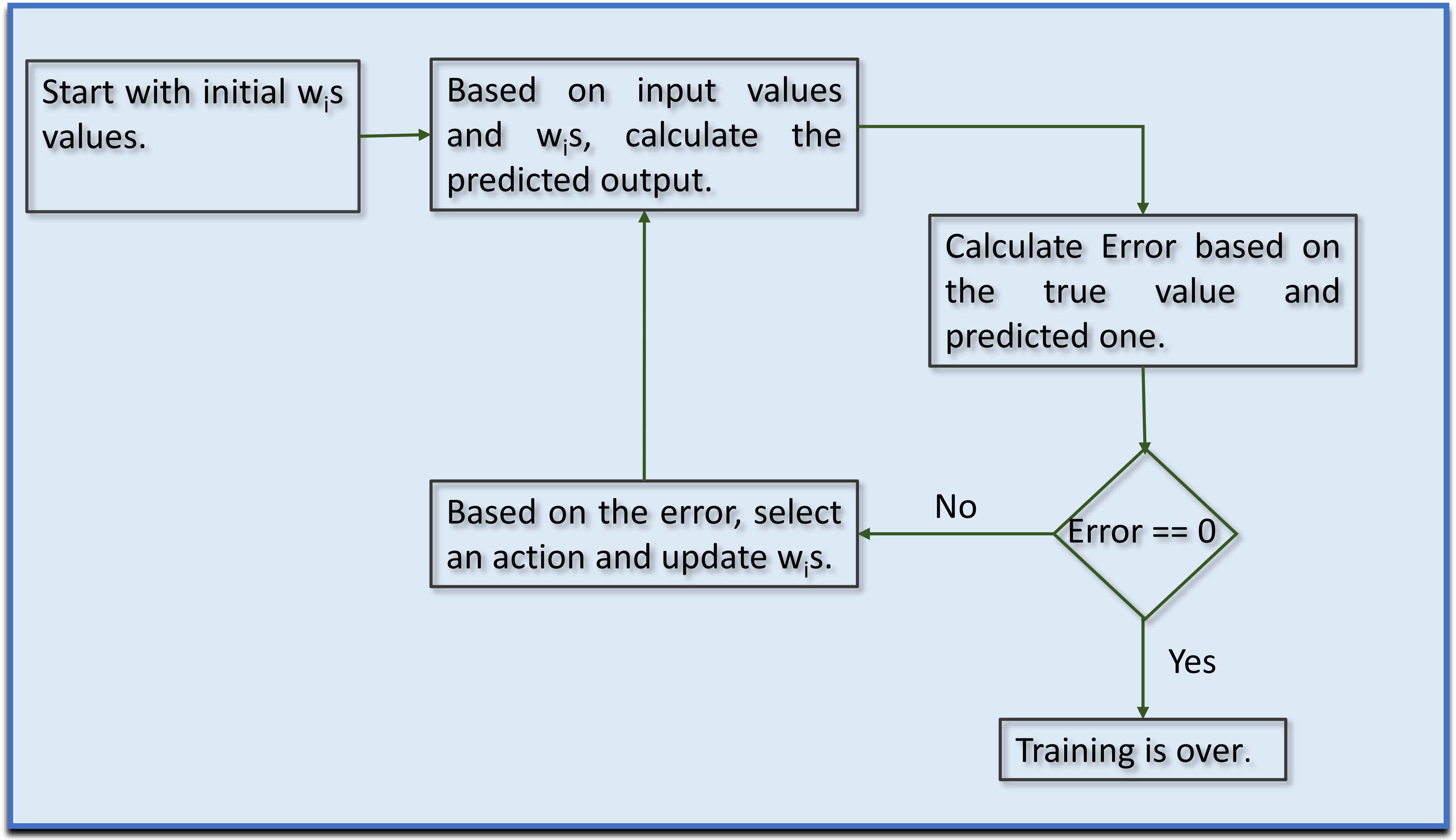
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial y}{\partial v}\frac{\partial v}{\partial x}$$

- This principle can be used to compute the partial derivatives in an efficient manner. Note that the transfer function must be differentiable (usually sigmoid, or tanh).

if $z(s)= \frac{1}{1+e^{-s}}$, z'(s)= z(1-z)

if z(s)= tanh(s), z'(s)= 1- z^2

Neural Networks, Learning


Start with initial w_is values.
Based on input values and w_is, calculate the predicted output.
Calculate Error based on the true value and predicted one.
Error == 0
No
Based on the error, select an action and update w_is.
Yes
Training is over.

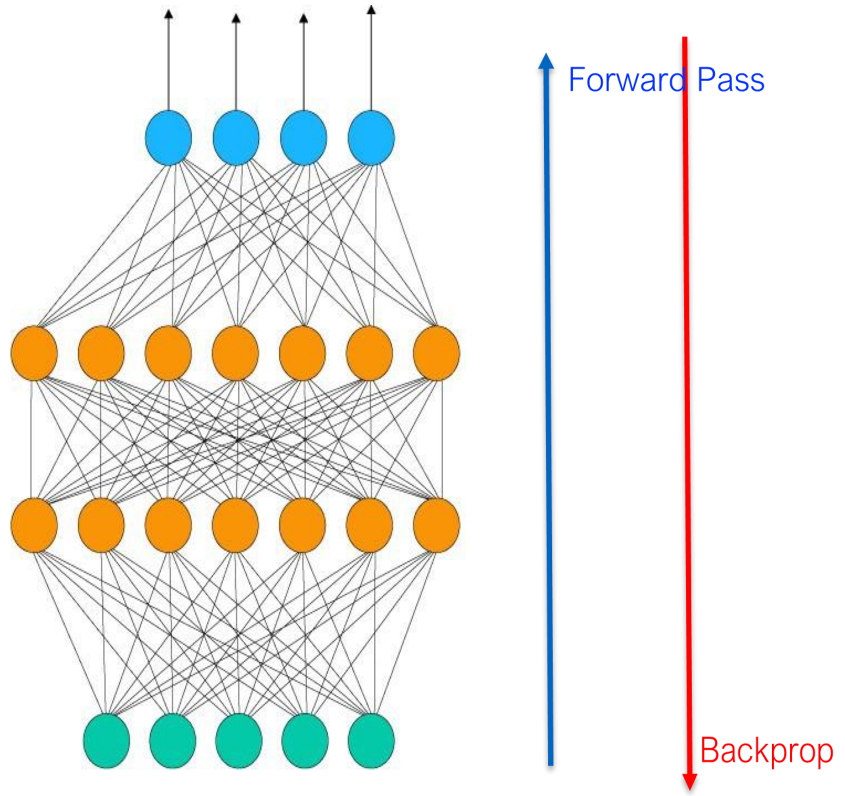
Artificial Neural Network Training Diagram

Neural Networks, Backpropagation

1. **Forward pass**: apply inputs to the "lowest layer" and feed activations forward to get output.

2. **Calculate error**: difference between desired output and actual output.

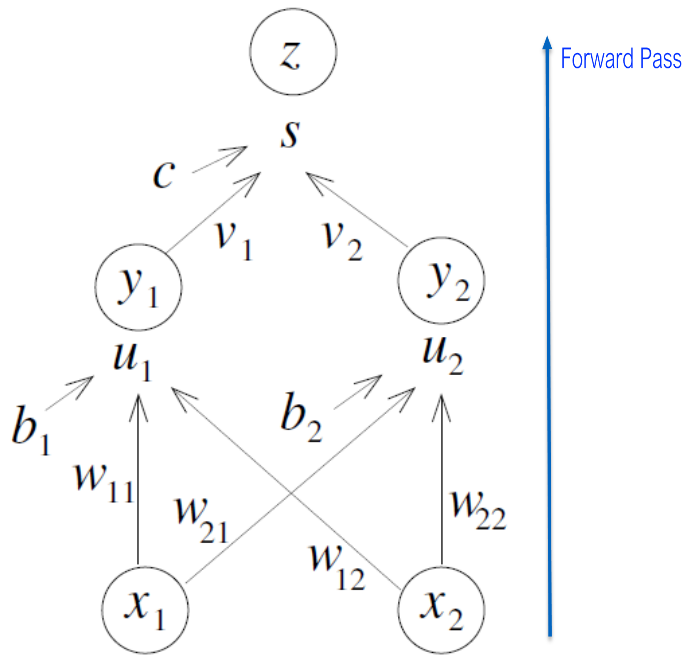
3. **Backward pass**: Propagate errors back through the network to adjust weights.



Neural Networks, Backpropagation

1. Forward pass:

$u_1 = b_1 + w_{11}x_1 + w_{12}x_2$
$y_1 = g(u_1)$
$s = c + v_1y_1 + v_2y_2$
$z = g(s)$

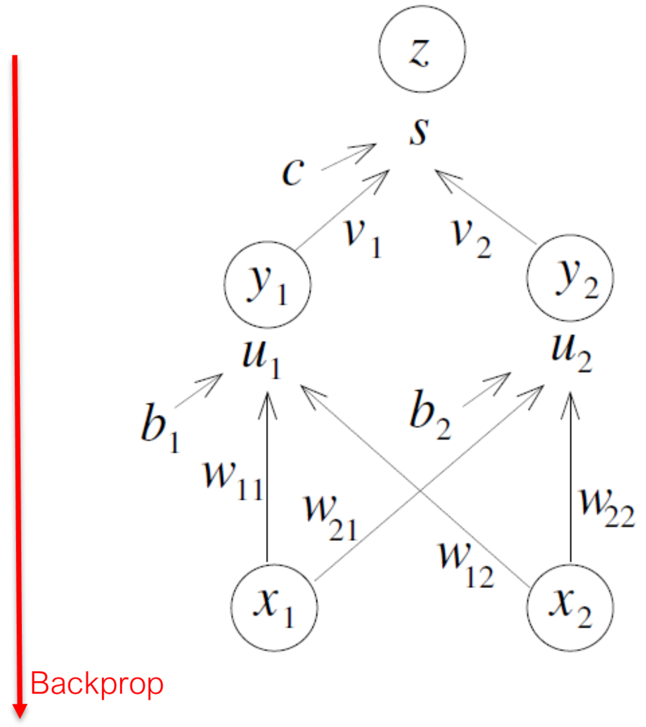


2. Calculate the error:

$$E = \frac{1}{2}\sum_i (z_i - t_i)^2$$

Neural Networks, Backpropagation

3. Backward pass: Propagate errors back through the network to adjust weights ($E = \frac{1}{2}\sum_i (z_i - t_i)^2$).



Partial derivative:

$$\frac{\partial E}{\partial z} = z - t$$

$$\frac{dz}{ds} = g'(s) = z(1 - z)$$

$$\frac{\partial s}{\partial y_1} = v_1$$

$$\frac{\partial y_1}{\partial u_1} = y_1(1 - y_1)$$

Then:

$$\frac{\partial E}{\partial s} = (z - t)z\,(1 - z)$$

$$\frac{\partial E}{\partial v_1} = (z - t)z\,(1 - z)\,y_1$$

$$\frac{\partial E}{\partial u_1} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial y_1}\frac{\partial y_1}{\partial u_1} = (z - t)z(1 - z)v_1 y_1(1 - y_1)$$

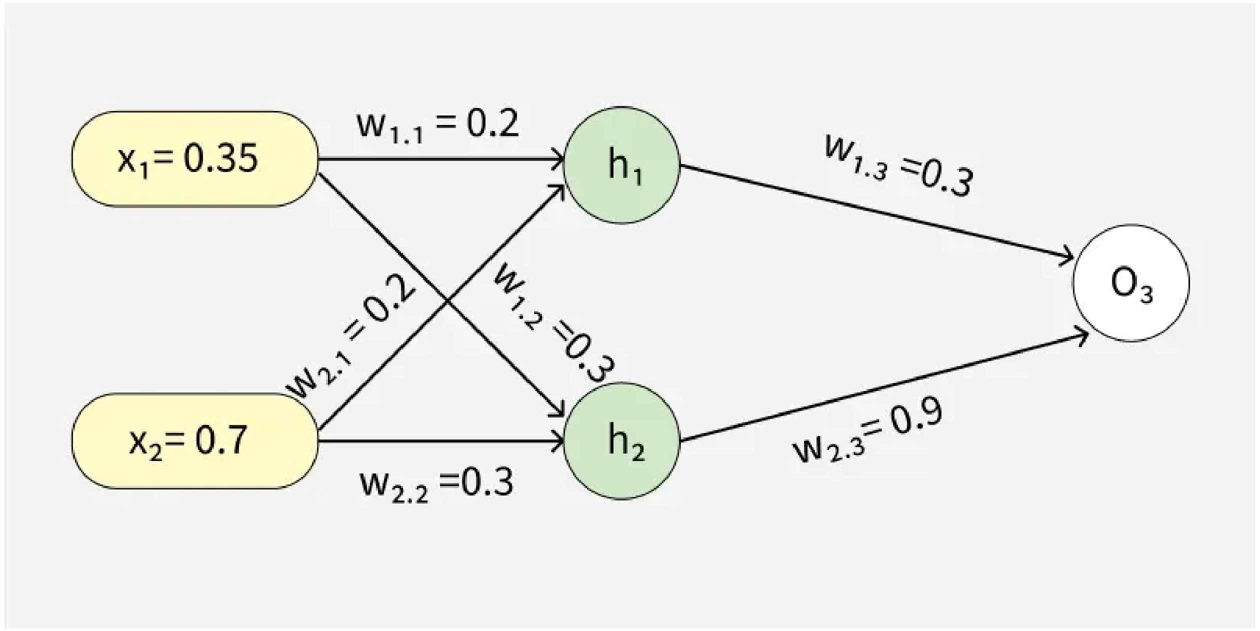
$$\frac{\partial E}{\partial w_{11}} = (z - t)z\,(1 - z)v_1\,y_1\,(1 - y_1)\,x_1$$

Neural Networks, Backpropagation

Example:
Consider the dataset $X = [0.35, 0.7]$ as the input and $y = [0.5]$ as the target output. We want to train the following neural network for 1 epoch and report the final weights. The activation function is the sigmoid function, and bias terms are set to 0. The error is defined as:

$$E = 1/2\ (y_{prediction} - y_{target})^2$$

The initial values for the weights are shown in the figure below.



https://www.geeksforgeeks.org/machine-learning/backpropagation-in-neural-network/

Neural Networks, Backpropagation

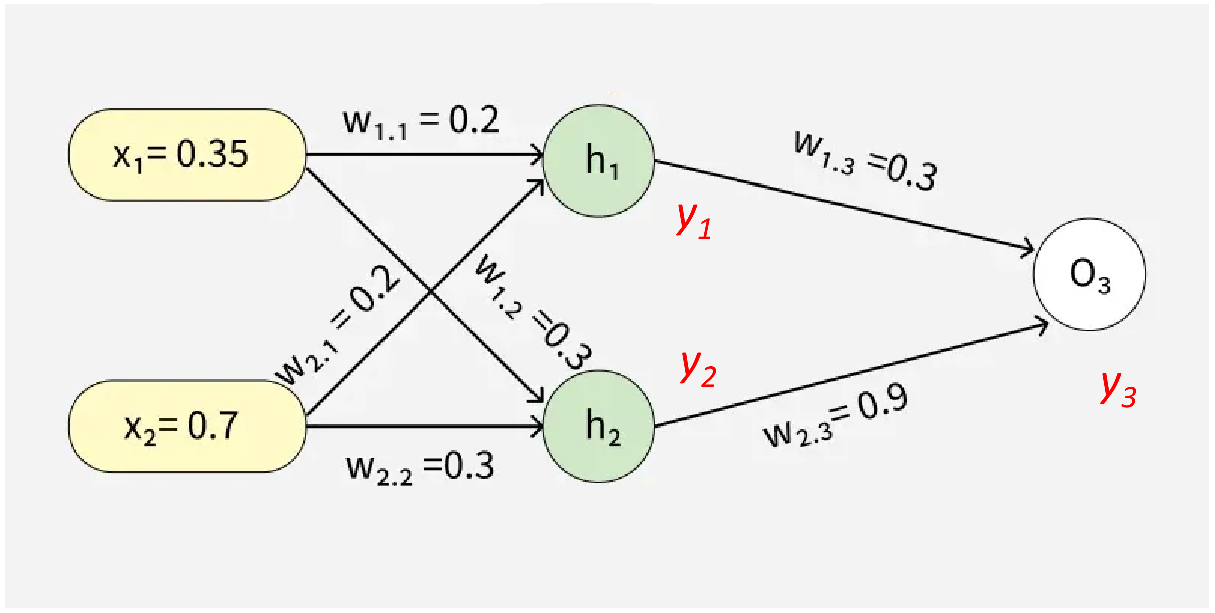
Step 1, Forward pass:

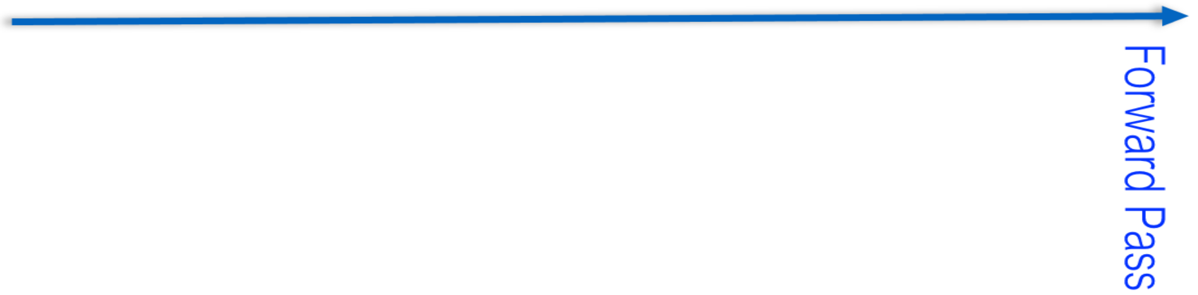
Computing the predicted output (y_3):

- $\text{Input}_{h1} = b_1 + w_{11}x_1 + w_{21}x_2 = 0 + (0.2\times0.35) + (0.2\times0.7) = 0.21$
- $y_1 = g(\text{Input}_{h1}) = \frac{1}{1+e^{-0.21}}$ **= 0.55**

- $\text{Input}_{h2} = b_2 + w_{12}x_1 + w_{22}x_2 = 0 + (0.3*0.35) + (0.3*0.7) = 0.315$
- $y_2 = g(\text{Input}_{h2}) = \frac{1}{1+e^{-0.315}}$ **= 0.58**

- $\text{Input}_{o3} = b_3 + w_{13}y_1 + w_{23}y_2 = 0 + (0.3*0.55) + (0.9*0.58) = 0.687$
- $y_3 = g(\text{Input}_{o3}) = \frac{1}{1+e^{-0.687}}$ **= 0.67**



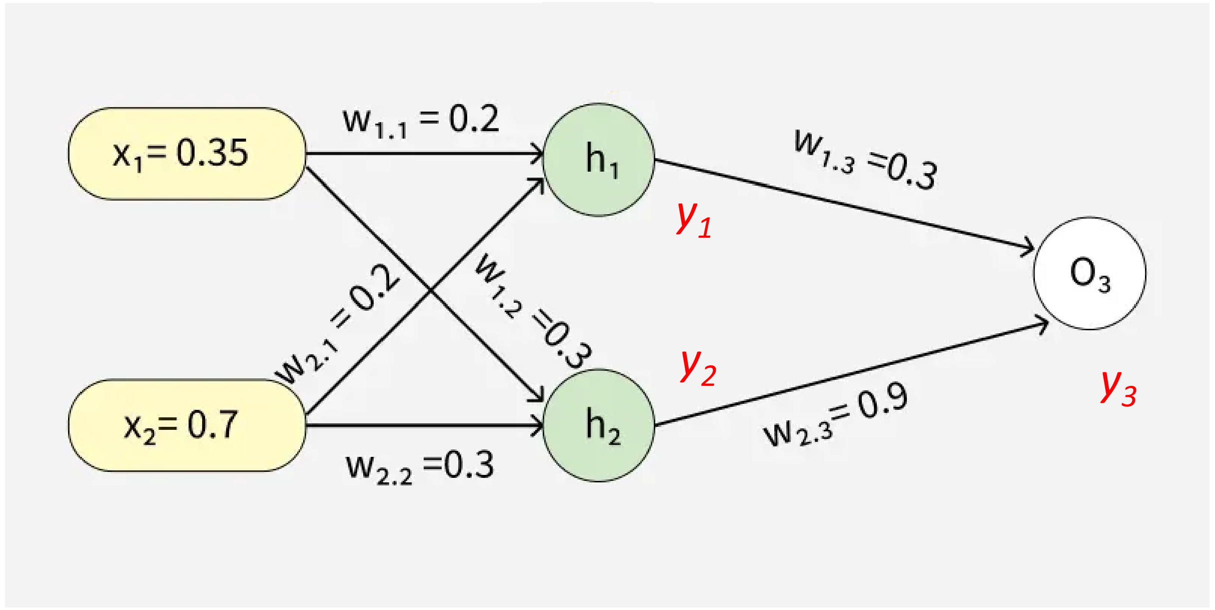


Neural Networks, Backpropagation

Step 2, Calculate the error:

$$E = 1/2\,(y_{prediction} - y_{target})^2$$

Error= ½ $(0.67-0.5)^2$ = 0.014



Neural Networks, Backpropagation

Step 3, Backpropagation:

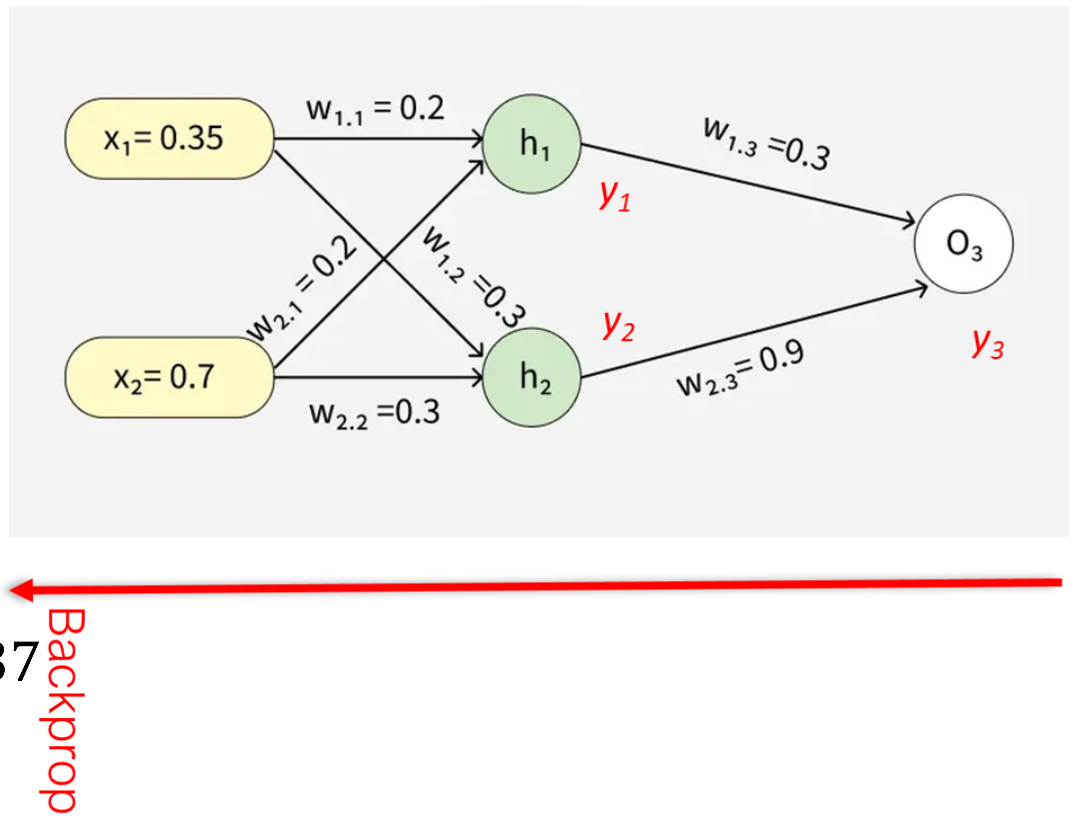
$$w_{ij}^{\text{new\_epoch}} \leftarrow w_{ij}^{\text{old\_epoch}} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial y_3} = y_3 - y_t = 0.67 - 0.5 = 0.17$$

$$\frac{\partial y_3}{\partial Input_{o3}} = y_3(1 - y_3) = 0.67(1 - 0.67) = 0.221$$

$$\frac{\partial E}{\partial Input_{o3}} = (y_3 - y_t) \cdot y_3(1 - y_3) = 0.17 \times 0.221 = 0.037$$

$$\frac{\partial E}{\partial w_{23}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial w_{23}} = \frac{\partial E}{\partial Input_{o3}} \cdot y_2 = 0.037 \times 0.58$$
$$= 0.021$$



$$\mathbf{w_{23}^{\text{new\_epoch}} = 0.9 - 1 \times (0.021) = 0.87}$$

Neural Networks, Backpropagation

Step 3, Backpropagation:

$$w_{ij}^{\text{new\_epoch}} \leftarrow w_{ij}^{\text{old\_epoch}} - \eta \frac{\partial E}{\partial w_{ij}}$$

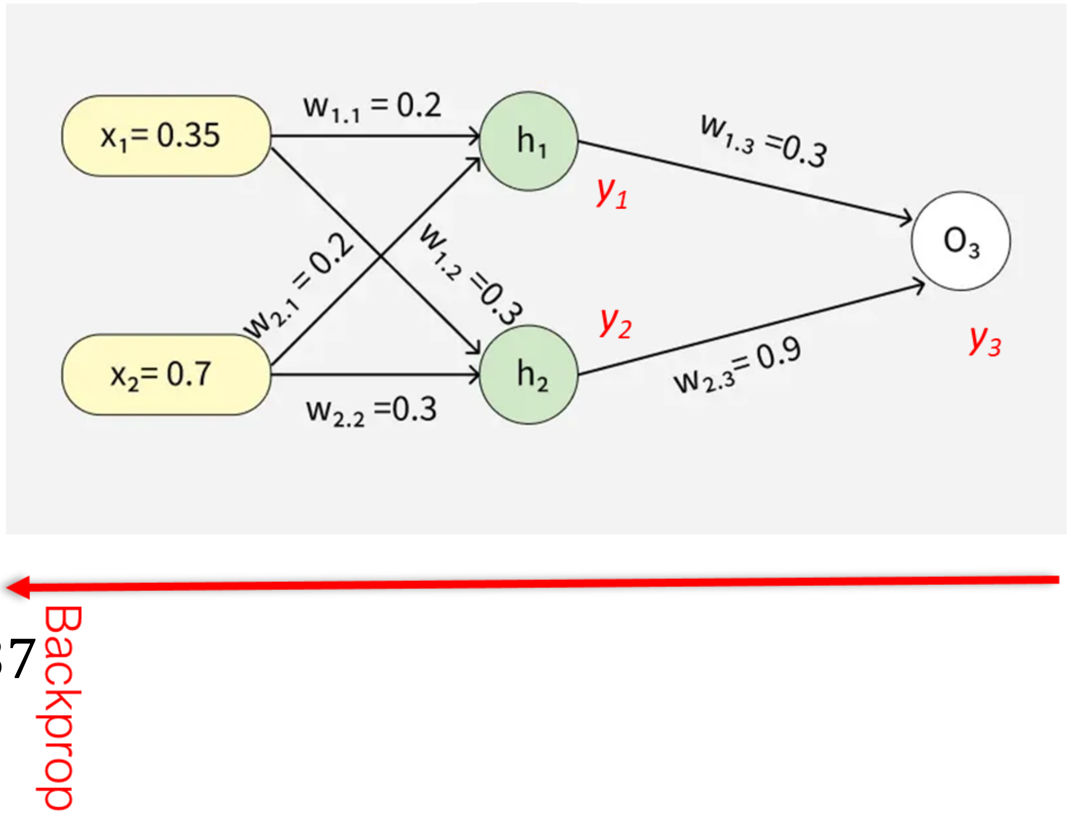
$$\frac{\partial E}{\partial y_3} = y_3 - y_t = 0.67 - 0.5 = 0.17$$

$$\frac{\partial y_3}{\partial Input_{o3}} = y_3(1 - y_3) = 0.67(1 - 0.67) = 0.221$$

$$\frac{\partial E}{\partial Input_{o3}} = (y_3 - y_t) \cdot y_3(1 - y_3) = 0.17 \times 0.221 = 0.037$$

$$\frac{\partial E}{\partial w_{13}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial w_{13}} = \frac{\partial E}{\partial Input_{o3}} \cdot y_1 = 0.037 \times 0.55$$
$$= 0.020$$

$$\mathbf{w_{13}^{\text{new\_epoch}} = 0.3 - 1 \times (0.020) = 0.28}$$



Neural Networks, Backpropagation

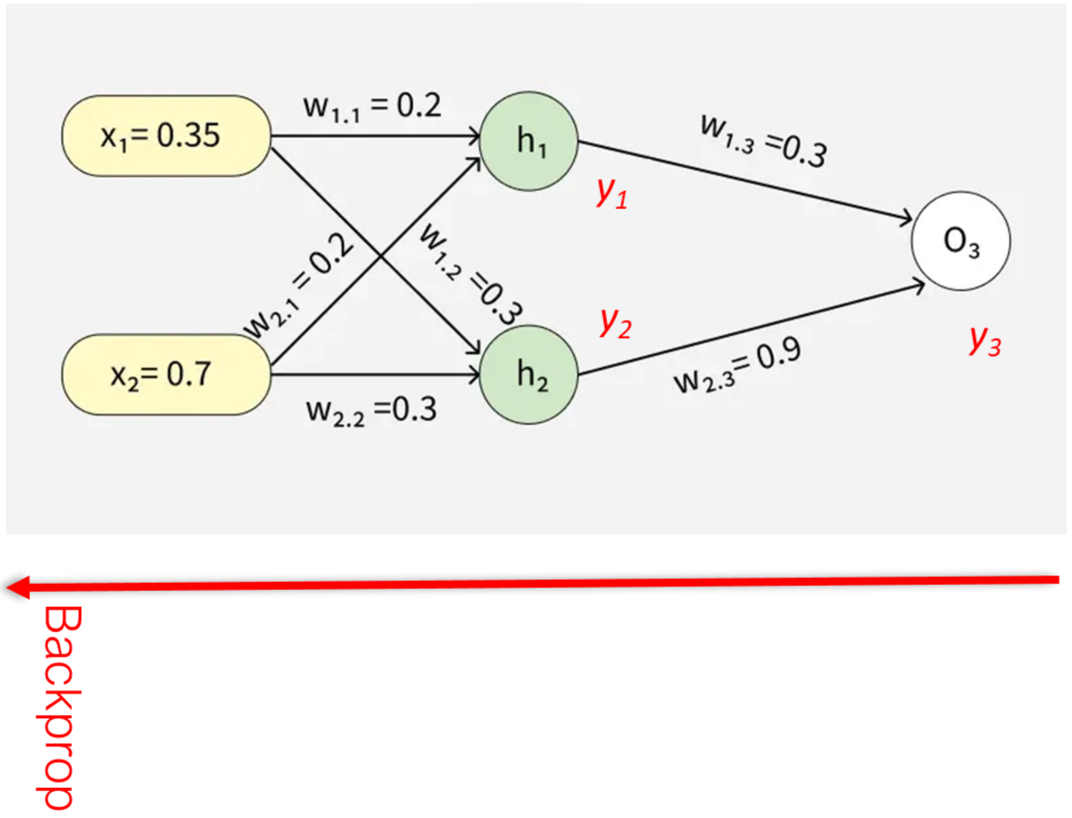
Step 3, Backpropagation:

$$\mathrm{w}_{\mathrm{ij}}^{\mathrm{new\_epoch}} \leftarrow \mathrm{w}_{\mathrm{ij}}^{\mathrm{old\_epoch}} - \eta \frac{\partial \mathrm{E}}{\partial \mathrm{w}_{\mathrm{ij}}}$$

$$\frac{\partial E}{\partial w_{11}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial \mathrm{y}_1} \frac{\partial y_1}{\partial \mathrm{Input}_{\mathrm{h1}}} \frac{\partial \mathrm{Input}_{\mathrm{h1}}}{\partial \mathrm{w}_{11}}$$

$$= (y_3 - y_t) \cdot y_3(1 - y_3) \cdot w_{13} \cdot y_1(1 - y_1) \cdot \mathrm{x}_1$$
$$= \; 0.037 \times 0.3 \times 0.55 \times (1 - 0.55) \times \; 0.35 = \; 0.0009$$

$$\mathbf{w}_{11}^{\mathbf{new\_epoch}} = \mathbf{0.2 - 1 \times (0.0009) = 0.1991}$$



Neural Networks, Backpropagation

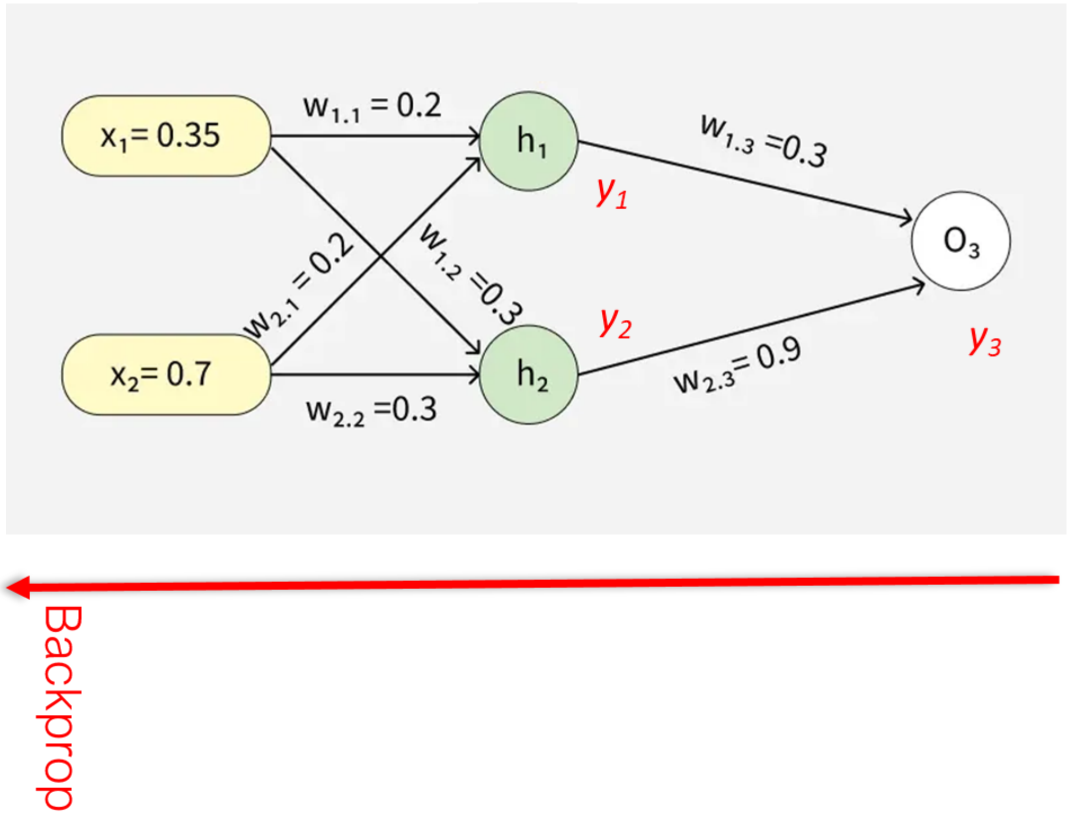
Step 3, Backpropagation:

$$w_{ij}^{\text{new\_epoch}} \leftarrow w_{ij}^{\text{old\_epoch}} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{21}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial y_1} \frac{\partial y_1}{\partial \mathrm{Input}_{h1}} \frac{\partial \mathrm{Input}_{h1}}{\partial w_{21}}$$

$$= (y_3 - y_t) \cdot y_3(1 - y_3) \cdot w_{13} \cdot y_1(1 - y_1) \cdot x_2$$
$$= 0.037 \times 0.3 \times 0.55 \times (1 - 0.55) \times 0.7 = 0.002$$

$$\mathbf{w_{21}^{new\_epoch} = 0.2 - 1 \times (0.002) = 0.198}$$



Neural Networks, Backpropagation

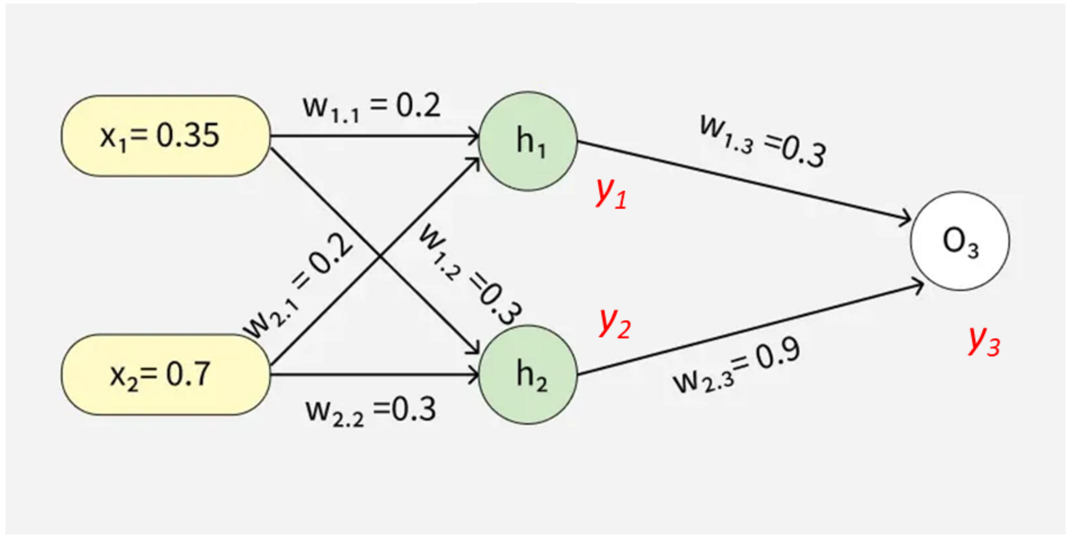
Step 3, Backpropagation:

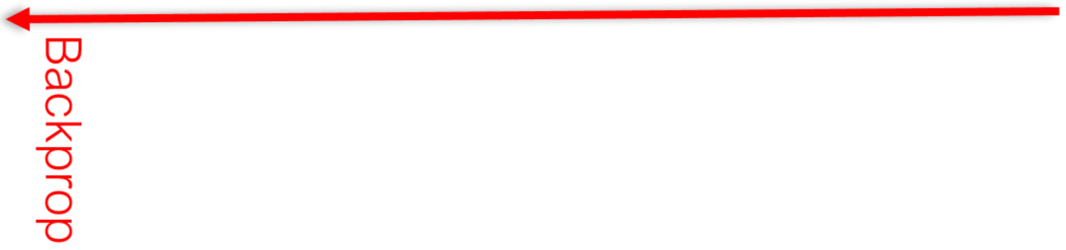
$$\mathrm{w}_{\mathrm{ij}}^{\mathrm{new\_epoch}} \leftarrow \mathrm{w}_{\mathrm{ij}}^{\mathrm{old\_epoch}} - \eta \frac{\partial \mathrm{E}}{\partial \mathrm{w}_{\mathrm{ij}}}$$

$$\frac{\partial E}{\partial w_{12}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial \mathrm{y}_2} \frac{\partial y_2}{\partial \mathrm{Input}_{\mathrm{h2}}} \frac{\partial \mathrm{Input}_{\mathrm{h2}}}{\partial \mathrm{w}_{12}}$$

$$= (y_3 - y_t) \cdot y_3(1 - y_3) \cdot w_{23} \cdot y_2(1 - y_2) \cdot \mathrm{x}_1$$
$$= \ 0.037 \times 0.9 \times 0.58 \times (1 - 0.58) \times \ 0.35 = \ 0.00355$$

$$\mathbf{w}_{12}^{\mathbf{new\_epoch}} = \mathbf{0.3 - 1 \times (0.00355) = 0.296}$$





Neural Networks, Backpropagation

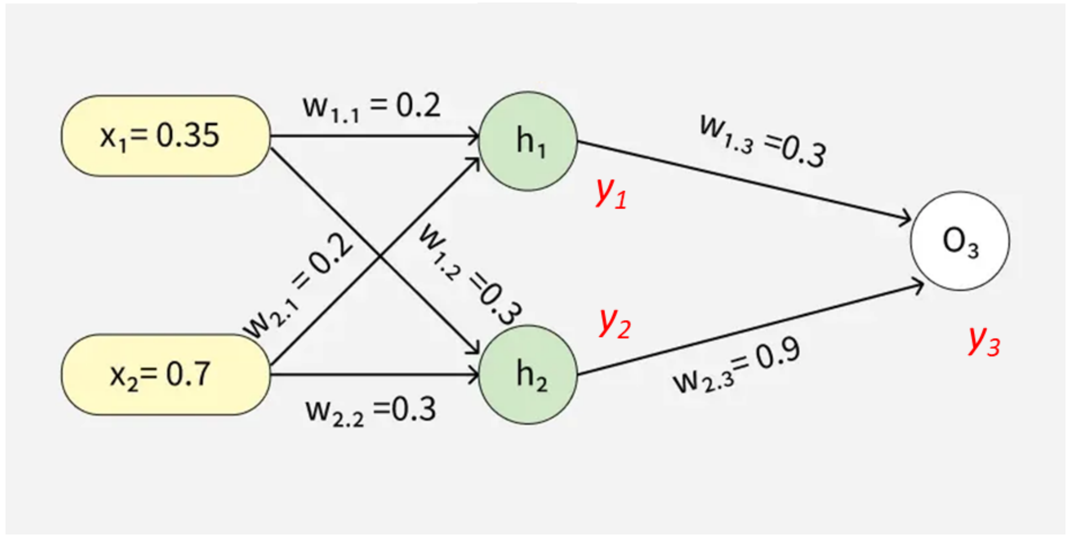
Step 3, Backpropagation:

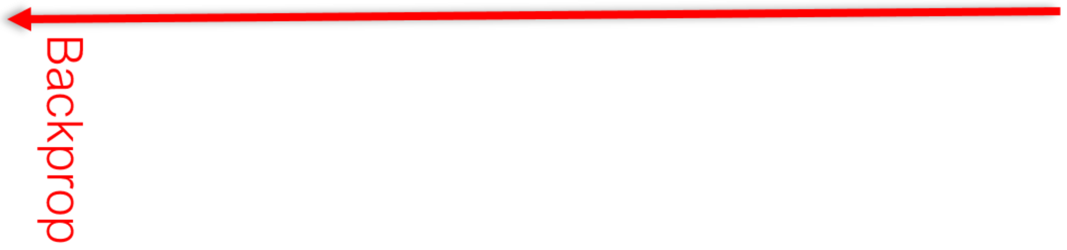
$$w_{ij}^{new\_epoch} \leftarrow w_{ij}^{old\_epoch} - \eta \frac{\partial E}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{22}} = \frac{\partial E}{\partial Input_{o3}} \cdot \frac{\partial Input_{o3}}{\partial y_2} \frac{\partial y_2}{\partial \mathrm{Input}_{h2}} \frac{\partial \mathrm{Input}_{h2}}{\partial w_{22}}$$

$$= (y_3 - y_t) \cdot y_3(1 - y_3) \cdot w_{23} \cdot y_2(1 - y_2) \cdot x_2$$
$$= 0.037 \times 0.9 \times 0.58 \times (1 - 0.58) \times 0.7 = 0.0057$$

$$\mathbf{w_{22}^{new\_epoch} = 0.3 - 1 \times (0.0057) = 0.294}$$





Neural Networks, Backpropagation

Conclusion:

$y_{target} < y_{prediction}$ (0.5 < 0.67)

$w_{11}^{old\_epoch} = 0.2 \rightarrow w_{11}^{new\_epoch} = 0.1991$
$w_{21}^{old\_epoch} = 0.2 \rightarrow w_{21}^{new\_epoch} = 0.198$
$w_{12}^{old\_epoch} = 0.3 \rightarrow w_{12}^{new\_epoch} = 0.296$
$w_{22}^{old\_epoch} = 0.3 \rightarrow w_{22}^{new\_epoch} = 0.294$
$w_{13}^{old\_epoch} = 0.3 \rightarrow w_{13}^{new\_epoch} = 0.28$
$w_{23}^{old\_epoch} = 0.9 \rightarrow w_{23}^{new\_epoch} = 0.87$

- Since our prediction is greater than the target value, we need to decrease the weight in order to reduce the prediction in the next iterations.
- As we propagate back to the weights in the earlier layers, the amount of adjustment (decrement) becomes smaller. For example, the change in w_{13} is more significant than the change in w_{11} .Why is that?

Overview

- ✓Biological Neurons.
- ✓Artificial Neurons.
  - ✓Perceptron.
- ✓Multi Layer Perceptron.
- ✓Learning with Gradient Descent and Backpropagation.
- ✓**Neural Networks Design.**
- ✓Deep Learning.

Neural Networks Design

So far, we have learned what a neural network is. But how can we actually design and implement one? What steps should we follow?

1. **Exhaustive Analysis of the System**
   Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. **Preprocessing**
   What steps should I take before feeding data into the network?

3. **Design of the Neural Network**
   What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. **Training**
   What happens during the training phase?

5. **Testing and Evaluation**
   How should I evaluate the performance of my network?

Neural Networks Design

1. **Exhaustive Analysis of the System**

Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. Preprocessing

What steps should I take before feeding data into the network?

3. Design of the Neural Network

What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. Training

What happens during the training phase?

5. Testing and Evaluation

How should I evaluate the performance of my network?

Neural Networks Design

Step 1: Exhaustive Analysis of the System

Is it really necessary to use a neural model?
Why not consider another classical model (e.g., decision trees), which are often cheaper and faster to train?

Neural Networks: The second-best option

- Neural networks are highly data-sensitive and usually require large datasets, since they need to learn a large number of parameters (w_i)
- Collecting and preparing such data can be costly and time-consuming.

If a neural network is chosen, we should ask:

- Do we have enough data that properly represents the system to be modeled?
- Is the available data sufficient in both quantity and quality?

Neural Networks Design

1. **Exhaustive Analysis of the System**

Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. Preprocessing

What steps should I take before feeding data into the network?

3. Design of the Neural Network

What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. Training

What happens during the training phase?

5. Testing and Evaluation

How should I evaluate the performance of my network?

Neural Networks Design

Step 2: Preprocessing

Data:

- A neural network is essentially a black-box model designed for interpolation (with no guarantee of good performance in extrapolation).
- Therefore, its effectiveness strongly depends on the quality and quantity of the data available.

Quality:

- This refers to how well the available data represents the underlying function being approximated.

Quantity:

- Only with a sufficiently large dataset can we expect to correctly identify the parameters (weights) of a neural model.

If the available data is insufficient, data augmentation techniques can be used to expand it.

Neural Networks Design

Step 2: **Preprocessing**

- Data cleaning:
  - Detect and, if possible, eliminate outliers, empty values, etc. It might also help to detect correlations between variables.

- Normalisation of variables:

$$X_n = (X - X_{min}) / (X_{max} - X_{min}); \quad X_n \in [0,1] \text{ or}$$

$$X_n = 2*(X - X_{min})/(X_{max} - X_{min}) - 1; \ X_n \in [-1,1]$$

- It is necessary to perform the corresponding denormalization at the output stage.

- Why data normalization?

Neural Networks Design

Step 2: Preprocessing, why data normalization?

1. **To ensure features are comparable in scale.**

- Many datasets include features with very different ranges (e.g., age in years vs. income in dollars).
- Without normalization, features with larger scales can dominate smaller ones in distance-based or gradient-based models.

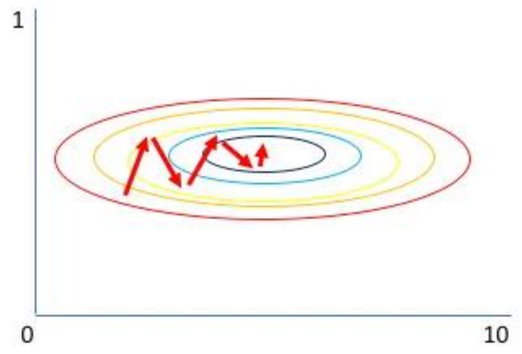
Example: If income = 60,000 and age = 30, the large numerical scale of income will overshadow the effect of age, potentially biasing the model to over rely on income.

Neural Networks Design

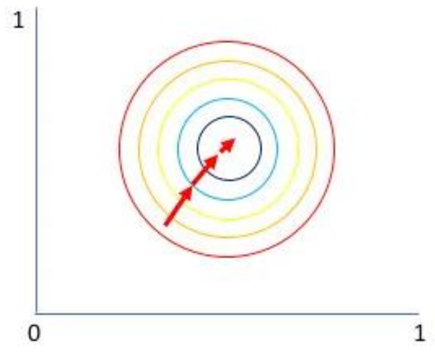
Step 2: Preprocessing, why data normalization?

2. To improve training stability and speed.

- Gradient descent converges faster when features are on a similar scale.
- If not normalized, the loss landscape can become very steep in some directions and flat in others.



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

https://www.jeremyjordan.me/batch-normalization/

Neural Networks Design

1. **Exhaustive Analysis of the System**

Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. Preprocessing

What steps should I take before feeding data into the network?

3. Design of the Neural Network

What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. Training

What happens during the training phase?

5. Testing and Evaluation

How should I evaluate the performance of my network?

Neural Networks Design

Step 3: Design of the Neural Network

Input and output neurons depend on the previous analysis of the system.
But, what about the number of neurons N_h in the hidden layer?

Rule of thumb: N_h should lead to a number of parameters (weights) N_w that:

$$N_w < (\text{Number of samples}) / 10$$

The number of weights N_w of a MLP, with N_i neurons in its input layer, a hidden layer with N_h neurons, and N_o neurons in the output layer is:

$$N_w = (N_i+1)*N_h+(N_h+1)*N_o$$

Conclusion: more parameters and deeper networks do not lead to better a performance necessarily. There should be a proportion between your dataset size, problem complexity, etc. and number of parameters.

Neural Networks Design

Step 3: Design of the Neural Network

An MLP with 3 inputs, 4 units in its hidden layer, and 2 outputs, has a number of parameters:

$$N_w = (N_i+1)*N_h+(N_h+1)*N_o$$

$$N_w = (3+1)*4+(4+1)*2 = 26$$

Then, at least 260 samples are required to train the network weights.

Neural Networks Design

Step 3: Design of the Neural Network

- In MLPs is demonstrated that using one hidden layer with a proper number of neurons is sufficient to **approximate any non-linear function with an arbitrary precision degree (Universal Approximation Theorem)**.

- Activation functions:
  - A usual criterion is to use sigmoid functions or ReLUs in the hidden layer and linear functions in the output. However, sigmoids or softmax can also be used in the output.

Neural Networks Design

1. **Exhaustive Analysis of the System**

Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. Preprocessing

What steps should I take before feeding data into the network?

3. Design of the Neural Network

What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. Training

What happens during the training phase?

5. Testing and Evaluation

How should I evaluate the performance of my network?

Neural Networks Design

Step 4: Training

- Training a neural network is a hard process due to the complexity of the error function solution space, which can have numerous local minima, minimax points, etc.

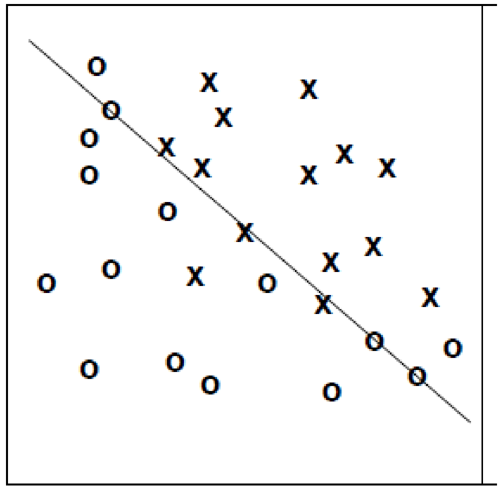
- There are three main problems that can arise during training:
    - **Bias**
    - **Overparameterization**
    - **Overfitting**
- The latter two might affect the network's ability to **generalize (high variance).**

Neural Networks Design

Step 4: **Training, Bias**

- Bias refers to the **systematic error** introduced by approximating a real-world problem (which may be very complex) with a simpler model.

- A **high bias model** makes **strong assumptions** about the data, leading to **underfitting**.
- It can’t capture the true complexity of the underlying relationship.



Neural Networks Design

Step 4: **Training, Bias**

To Decrease Bias:

- One way to reduce bias is to run multiple training processes starting from different randomly chosen initial weights (e.g., 20 or more attempts). This increases the chance of reaching a better local minimum.
- Another approach is to increase the number of neurons in the hidden layer, which allows the model to better capture the complexity of the problem.
  - However, there is a trade-off: **adding too many neurons can lead to high variance and overparameterization, causing the model to overfit.**

Neural Networks Design

Step 4: **Training, Overfitting**

There is a trade-off: **increasing the number of neurons can lead to high variance and overparameterization.**

- A model is considered overparameterized when it has more trainable parameters than the available training data can uniquely determine.

- In simpler terms, the model has *too many parameters* relative to the size or complexity of the dataset.
- As a result, it becomes powerful enough to **memorize** the training data (including noise) rather than learning the underlying general patterns.

Neural Networks Design

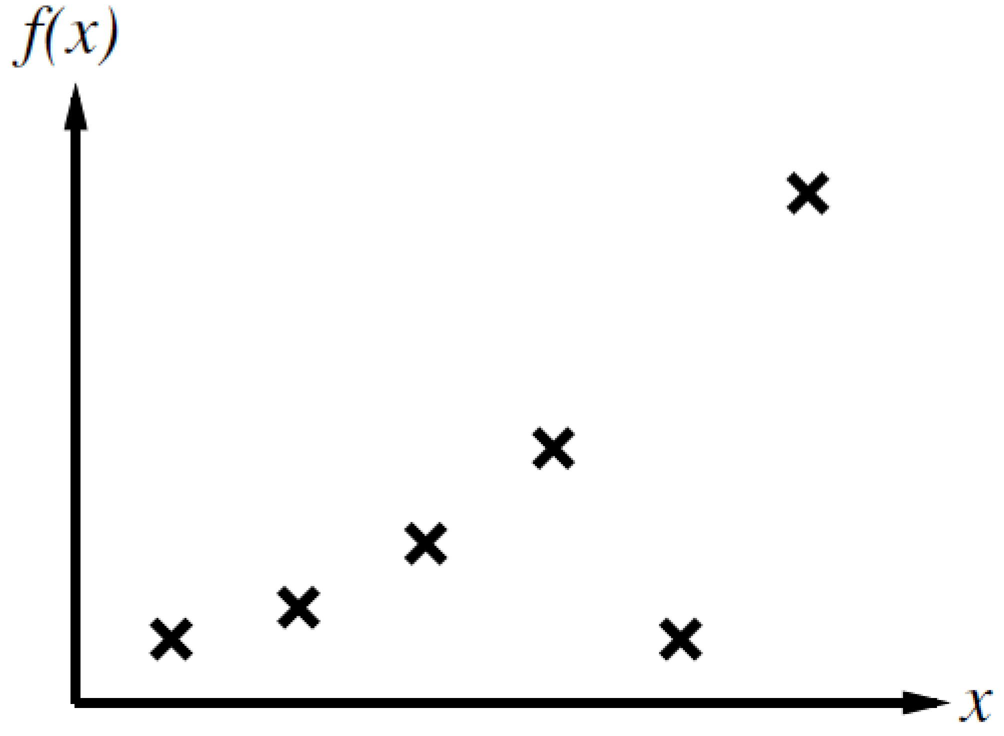
Step 4: Training, Overfitting

- The neural network is trained by presenting it with the input and target values for all items in the training set.
- After completing the learning procedure, it must then predict the outputs for items in the test set.
- The goal is to accurately predict the target values for the test set based on the input attributes.
- A common mistake to avoid is building a model that fits the training data very well but performs poorly on unseen test data—this problem is known as **overfitting**.
- In contrast, a model that achieves high accuracy on both the training and test sets is said to have good **generalization**.

Neural Networks Design

Step 4: Training, Overfitting

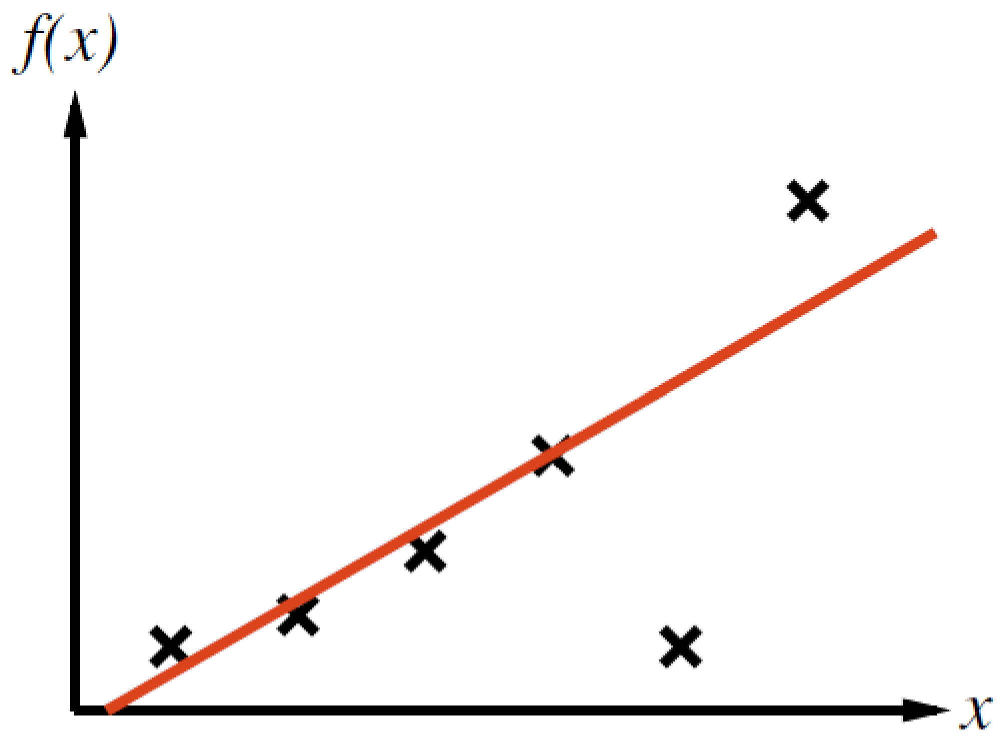
Which curve do you think gives the “best fit” to these data?



Neural Networks Design

Step 4: Training, Overfitting

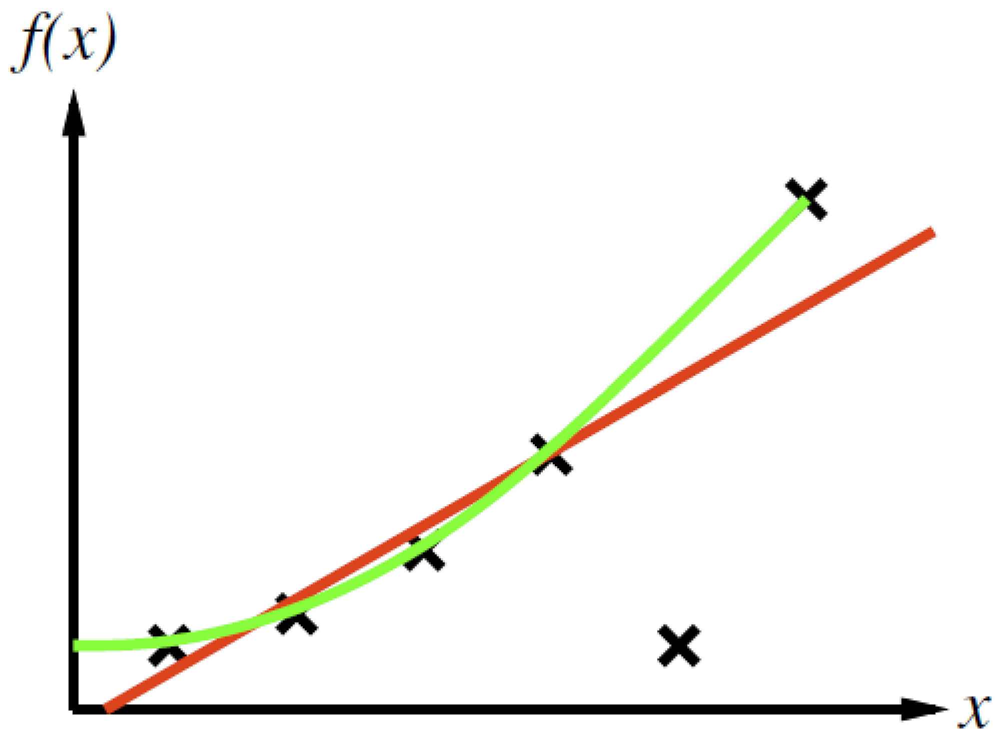
1: straight line



Neural Networks Design

Step 4: Training, Overfitting

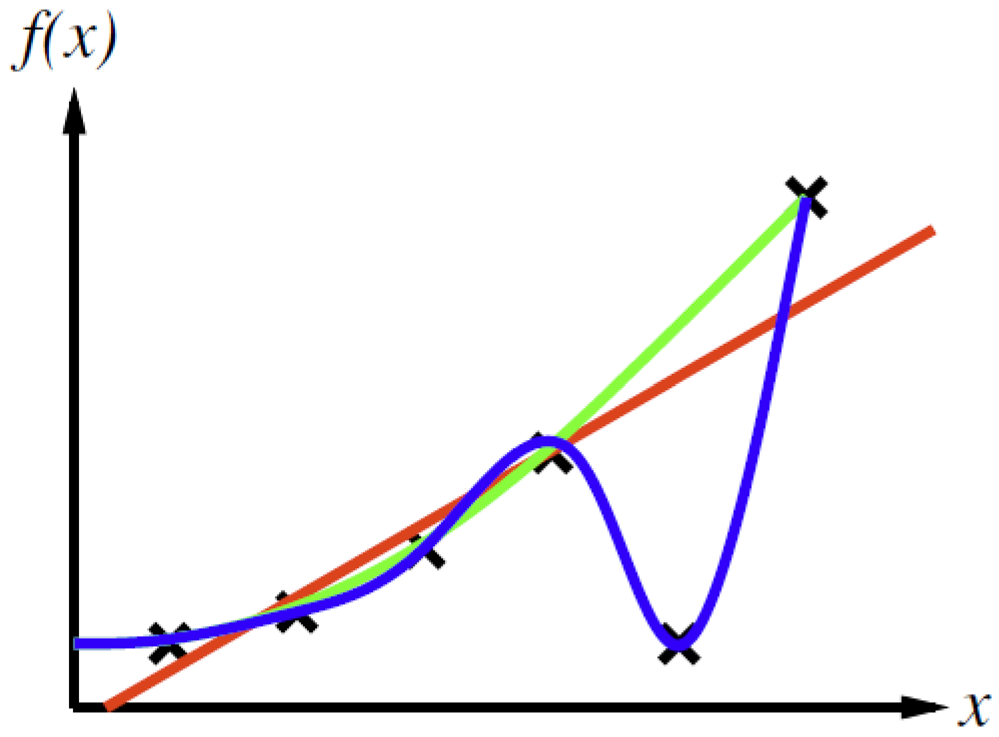
2: Parabola



Neural Networks Design

Step 4: Training, Overfitting

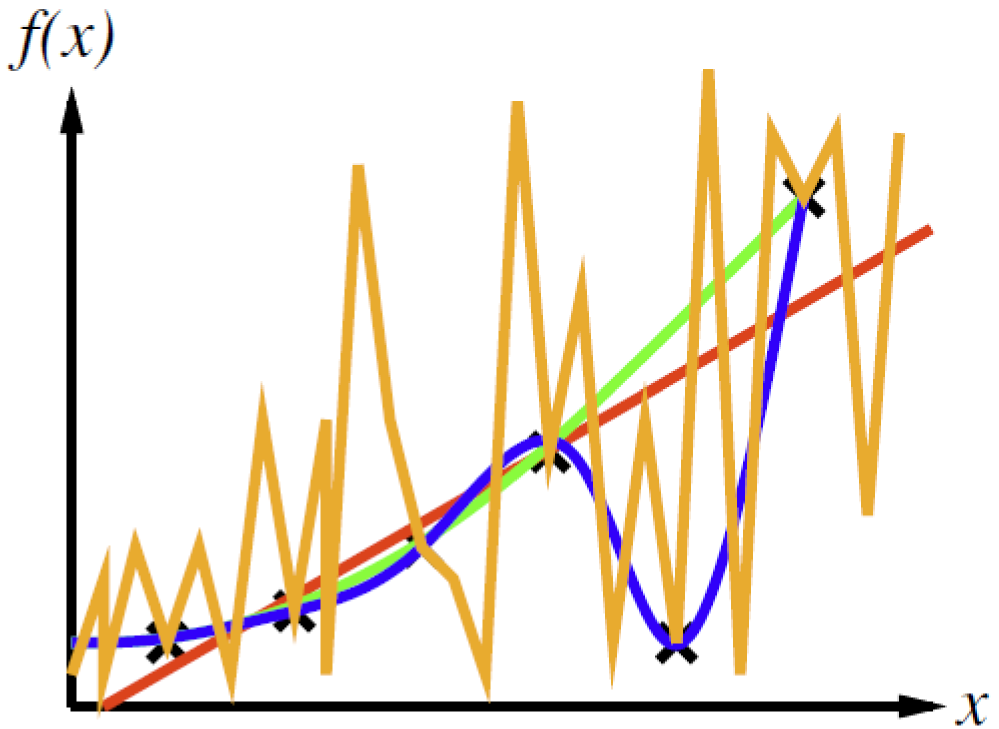
3: Fourth order polynomial



Neural Networks Design

Step 4: Training, Overfitting

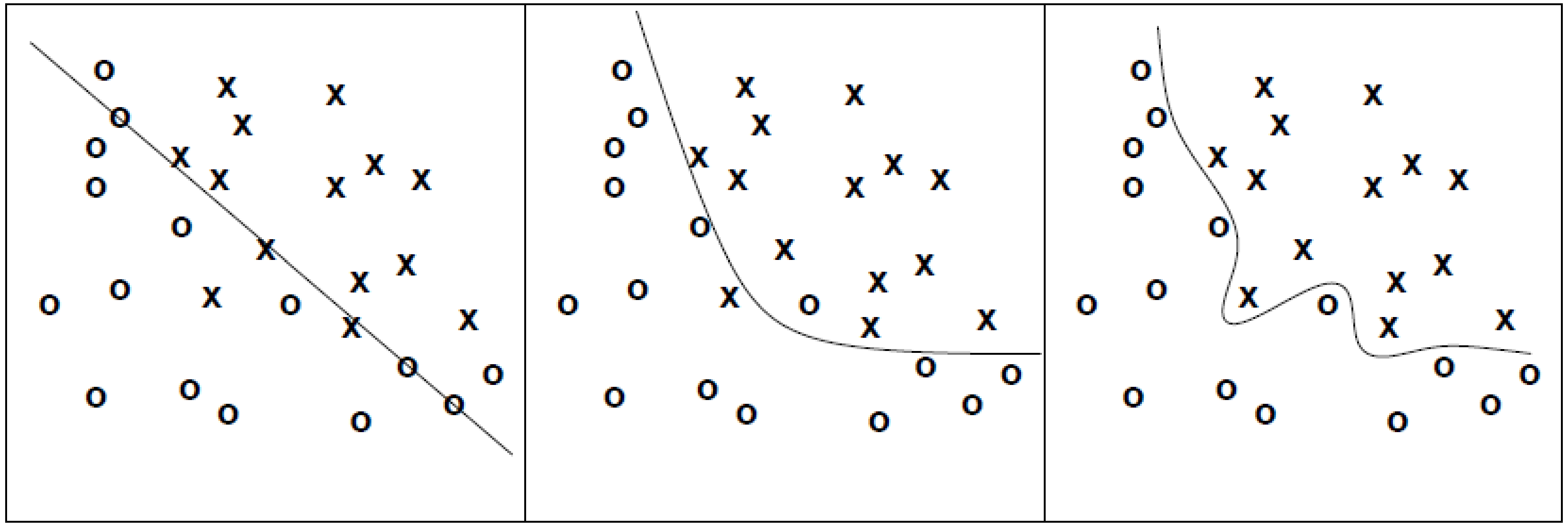
4: something else?



Neural Networks Design

Step 4: Training, Overfitting

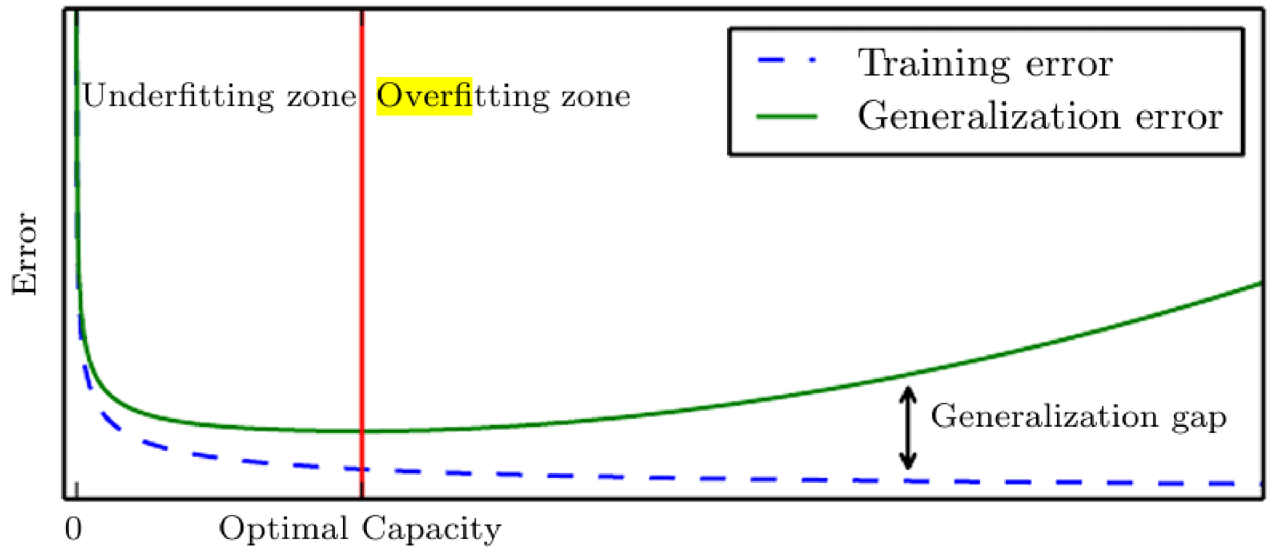
- “The most likely hypothesis is the simplest one consistent with the data.”
- This principle was popularized by William of Ockham in the thirteenth Century and came to be known as "Ockham's Razor".
- In the middle curve, two training items near the boundary are misclassified. This is acceptable since the errors may be caused by noise in the data, which often results from small unmodeled factors such as measurement errors, friction, or air resistance.



Neural Networks Design

Step 4: Training, Overfitting

- To determine the optimal model parameters, the dataset is often divided into **Training, Validation, and Test (generalization) sets**.
- Typically, as the number of hidden nodes increases, both the training and test errors initially decrease.
- However, beyond a certain point, the training error may continue to decrease while the test error plateaus or even slightly increases. The approximate **number of hidden nodes at which the test error is minimal is likely to achieve the best generalization performance**.

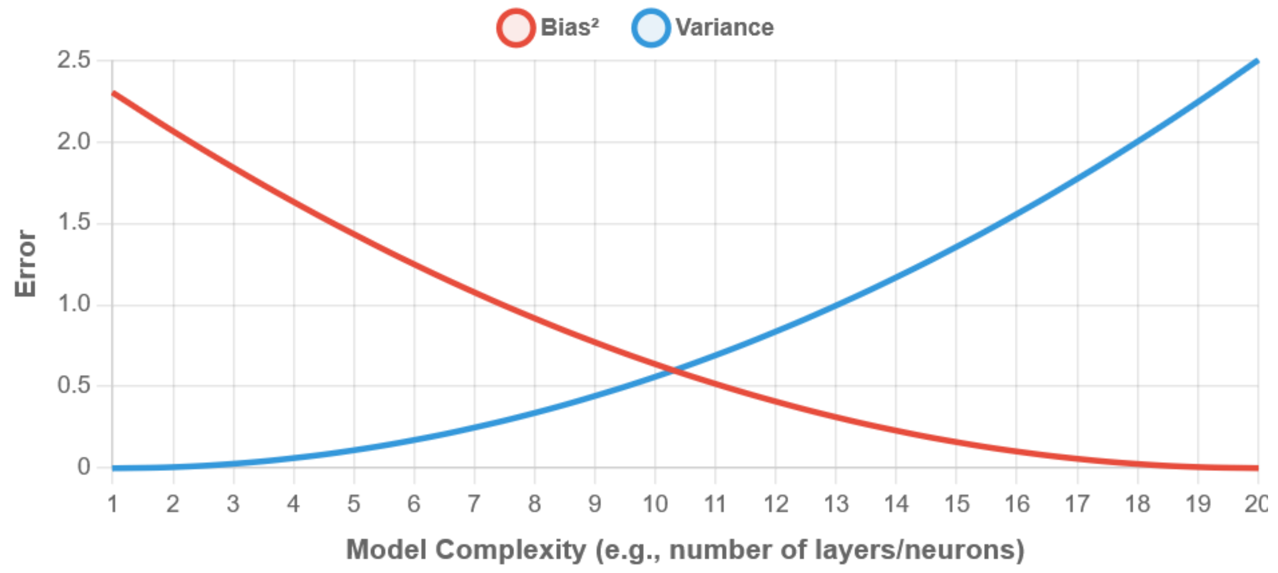


Neural Networks Design

Step 4: Training

Conclusion:

- Increasing the number of neurons can help reduce bias (underfitting), while limiting the number of neurons helps prevent overfitting (high variance). This balance reflects the well-known **bias–variance trade-off**.
- In addition to adjusting the network size, there are other techniques available to address underfitting and overfitting, such as regularization, early stopping, and data augmentation.



Neural Networks Design

1. **Exhaustive Analysis of the System**
Is a neural network really the best solution for this problem? Do I have the necessary requirements?

2. Preprocessing
What steps should I take before feeding data into the network?

3. Design of the Neural Network
What should the architecture of my network look like? (e.g., number of layers, number of neurons per layer, choice of activation functions, and other hyperparameters).

4. Training
What happens during the training phase?

5. Testing and Evaluation
How should I evaluate the performance of my network?

Neural Networks Design

Step 4: Testing

- To test the generalization capability of the network, performance is evaluated on the test set.
- Some common loss functions:

Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Cross-Entropy (Binary Cross-Entropy (for 2 classes)):

$$\text{L} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - \hat{y}_i)\log(1 - \hat{y}_i)]$$

Overview

- ✓Biological Neurons.
- ✓Artificial Neurons.
    - ✓Perceptron.
- ✓Multi Layer Perceptron.
- ✓Learning with Gradient Descent and Backpropagation.
- ✓Neural Networks Design.
- ✓**Deep Learning.**

Neural Networks and Deep Learning

- A **perceptron** is a single neuron that uses a step function as its activation function.
- Multiple perceptrons can be combined to form a **multi-layer perceptron**.
- By using other non-linear activation functions, these networks can learn complex, non-linear tasks; such networks are generally called **neural networks**.

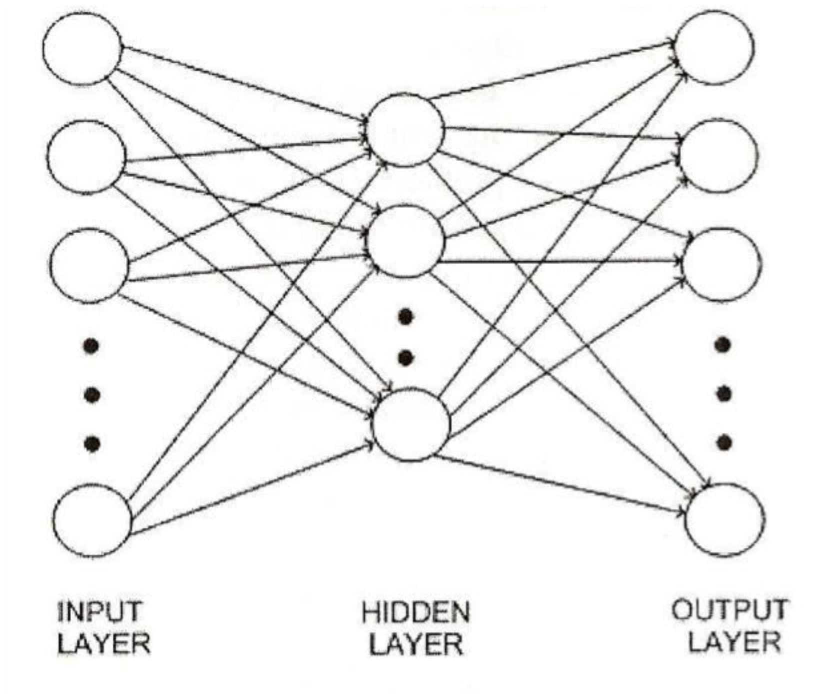
In essence, neural networks are composed of neurons arranged in layers, where each neuron applies an activation function to its input.

If we denote the first layer as $f^{(1)}$, the second layer is $f^{(2)}$, and so on. The total number of layers defines the **depth** of the model, which is why this approach is known as **deep learning**.

Neural Networks and Deep Learning

The **depth** of a model is defined as the total number of layers with learnable parameters, excluding the input layer.

- Deeper networks can typically approximate more complex functions, while shallower models have fewer layers and are limited to approximating simpler functions.



Depth= 2

Neural Networks and Deep Learning

- Deep learning is based on the philosophy of **connectionism**:
  - while an individual biological neuron is not particularly intelligent, a large population of neurons or features acting together can exhibit intelligent behavior.

- It is important to emphasize that the **number of neurons must be large**.
- One of the key factors behind the dramatic improvement in neural network accuracy and the complexity of tasks they can handle—from the 1980s to today—is the significant increase in network size.
- Interestingly, even the largest current artificial neural networks are roughly comparable in scale to the nervous systems of insects.

Neural Networks and Deep Learning

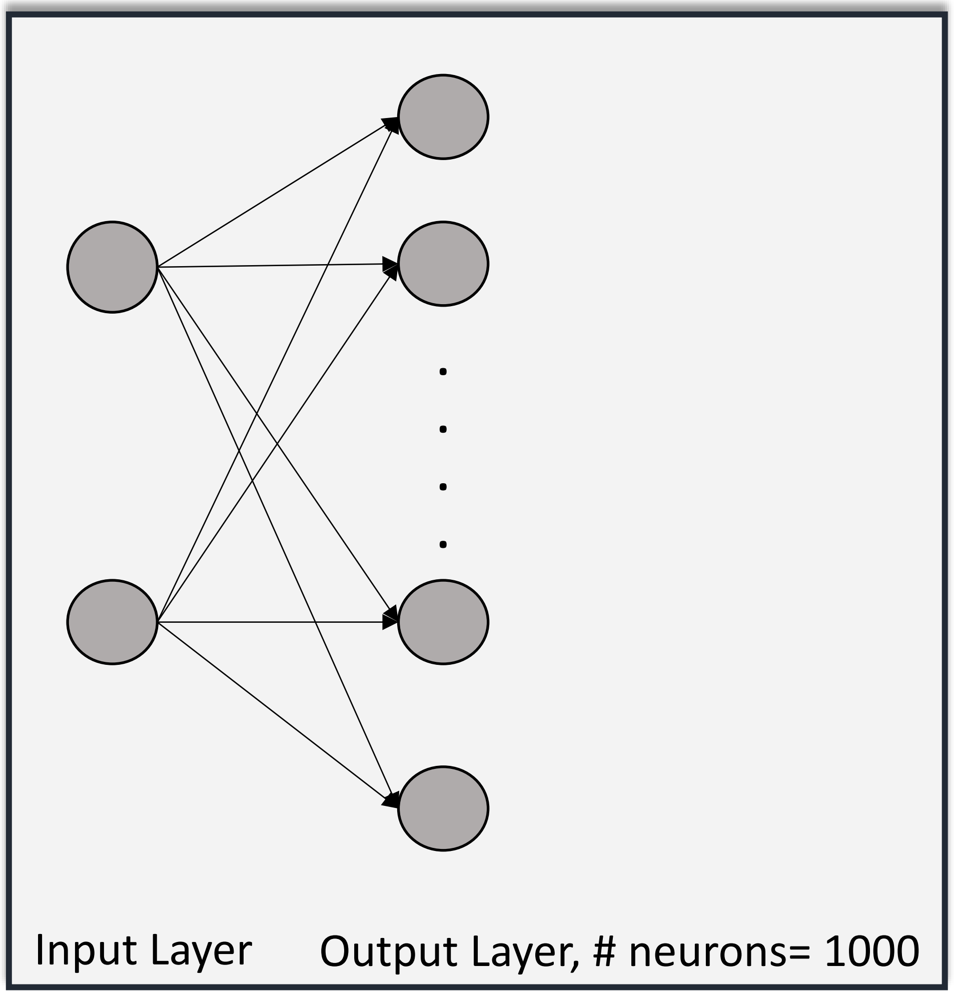
- **Deep learning** refers to artificial neural networks with multiple layers that automatically learn patterns and representations from data.

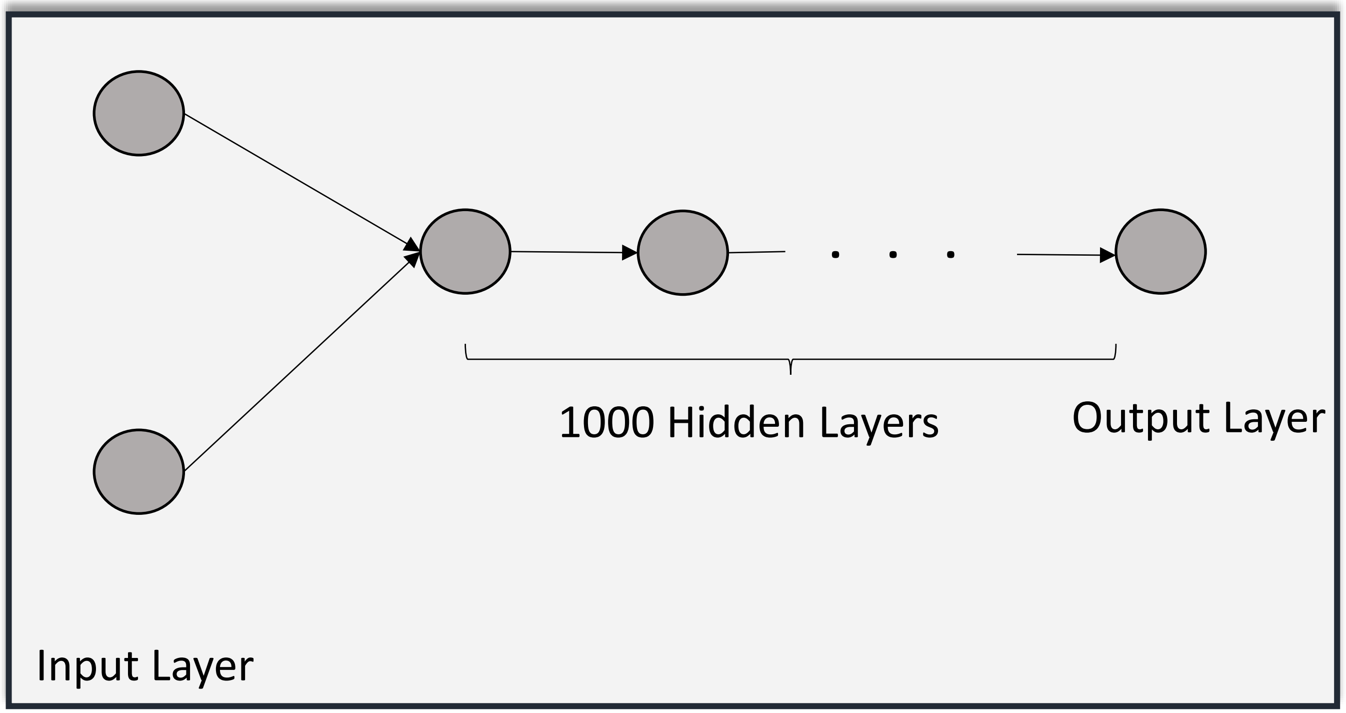
- What makes a network **"deep"** is the presence of many layers of neurons, not just one or two. A **shallow network** might have only a single hidden layer, whereas a **deep network** can have hundreds or even thousands of layers.

- Deep learning performs best when there is abundant data and access to powerful GPUs for training, as training many neurons requires substantial computational resources.
  - But when applied properly, deep networks can achieve remarkable performance.

Neural Networks and Deep Learning

Question: If we put hundreds of neurons in a single layer, or have hundreds of layers each with one neuron, which one qualifies as deep learning?





Neural Networks and Deep Learning

Case 1: One layer with many neurons:

- This network has a very wide hidden layer, but it is still **shallow** because the depth is low (only one hidden layer).
- According to the **Universal Approximation Theorem**, a single wide hidden layer can approximate any continuous function with an arbitrary precision degree.
- However, in practice, this approach is computationally inefficient and does not capture hierarchical representations effectively (low precision).

Case 2: Many layers, each with one neuron:

- This network is extremely deep but very narrow.
- It qualifies as **deep learning** because the depth (number of layers) is enormous.
- However, with only one neuron per layer, the network's expressive power is limited.

References

- Poole & Mackworth, Artificial Intelligence: Foundations of Computational Agents, chapters 1 & 2.
- Russell & Norvig, Artificial Intelligence: a Modern Approach, Chapter 2.
- Goodfellow, Bengio, Courville, and Bengio, 2016. Deep learning (Vol. 1, No. 2). Cambridge: MIT press.