Departamento de Eletrónica, Telecomunicações e Informática

NEURAL NETWORKS

Author: Petia Georgieva
Edited by: Susana Brás (Susana.bras@ua.pt)

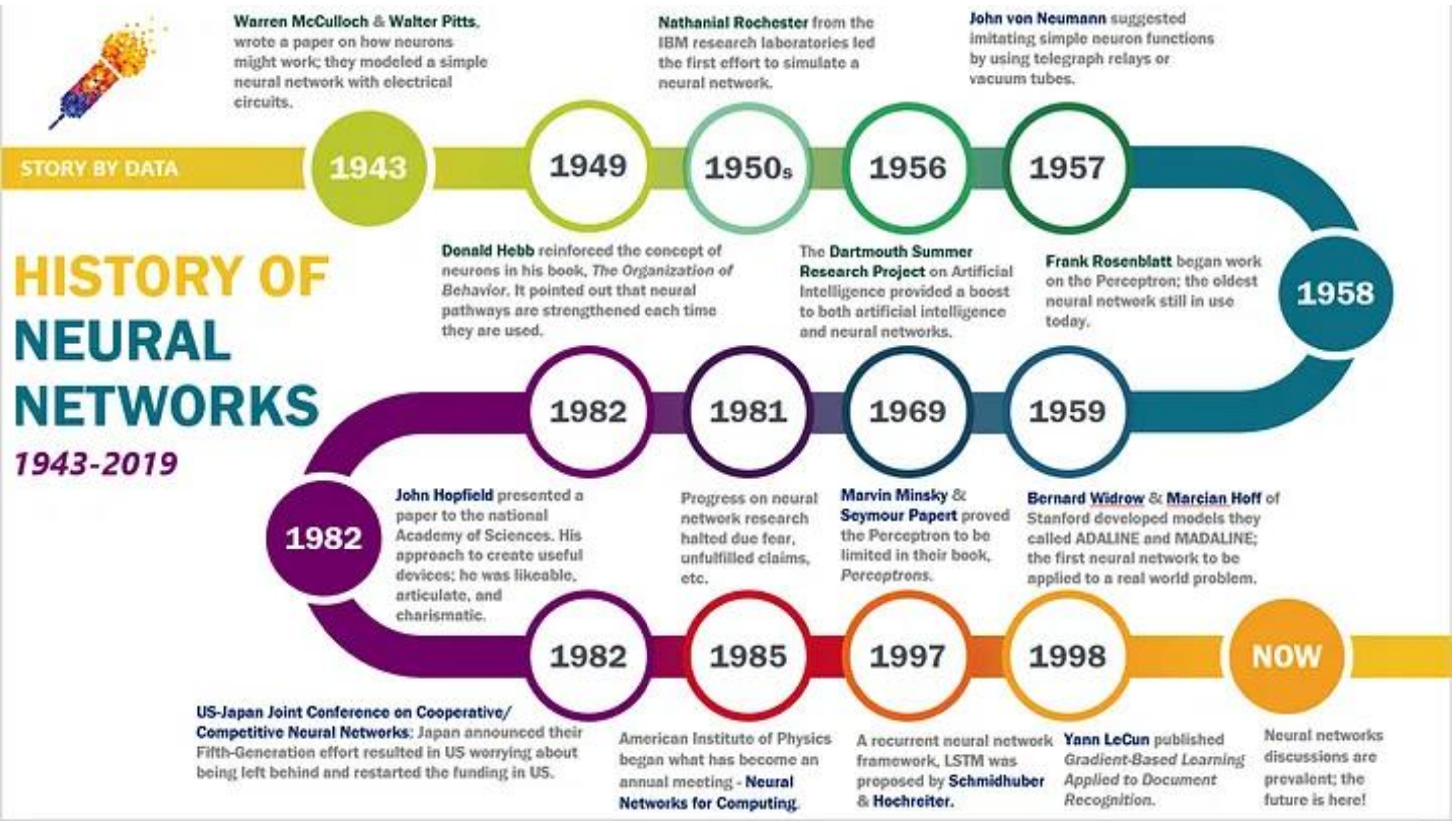
universidade de aveiro

Neural Networks- outline

1. NN - non-linear classifier
2. Neuron model: logistic unit
3. NN - binary versus multi-class classification
4. Cost function (with or without regularization)
5. NN learning - Error Backpropagation algorithm

NN - non-linear classifier

NN History



Warren McCulloch & **Walter Pitts**, wrote a paper on how neurons might work; they modeled a simple neural network with electrical circuits.

Donald Hebb reinforced the concept of neurons in his book, *The Organization of Behavior*. It pointed out that neural pathways are strengthened each time they are used.

Nathanial Rochester from the IBM research laboratories led the first effort to simulate a neural network.

The **Dartmouth Summer Research Project** on Artificial Intelligence provided a boost to both artificial intelligence and neural networks.

John von Neumann suggested imitating simple neuron functions by using telegraph relays or vacuum tubes.

Frank Rosenblatt began work on the Perceptron; the oldest neural network still in use today.

Bernard Widrow & **Marcian Hoff** of Stanford developed models they called ADALINE and MADALINE; the first neural network to be applied to a real world problem.

Marvin Minsky & **Seymour Papert** proved the Perceptron to be limited in their book, *Perceptrons*.

Progress on neural network research halted due fear, unfulfilled claims, etc.

John Hopfield presented a paper to the national Academy of Sciences. His approach to create useful devices; he was likeable, articulate, and charismatic.

US-Japan Joint Conference on Cooperative/ Competitive Neural Networks: Japan announced their Fifth-Generation effort resulted in US worrying about being left behind and restarted the funding in US.

American Institute of Physics began what has become an annual meeting - **Neural Networks for Computing**.

A recurrent neural network framework, LSTM was proposed by **Schmidhuber** & **Hochreiter**.

Yann LeCun published *Gradient-Based Learning Applied to Document Recognition*.

Neural networks discussions are prevalent; the future is here!

Only with the advent of hyper-fast processing, massive data storage capabilities, and access to computing resources were neural networks were able to advance to the point they have reached today.

Developments are still being made in this field; one of the most important types of neural networks in use today, the transformer, dates to 2017.

INTERNATIONAL NEURAL NETWORK SOCIETY
EXPAND YOUR AI NETWORK
JOIN OUR COMMUNITY
www.inns.org

Classification of non-linearly separable data

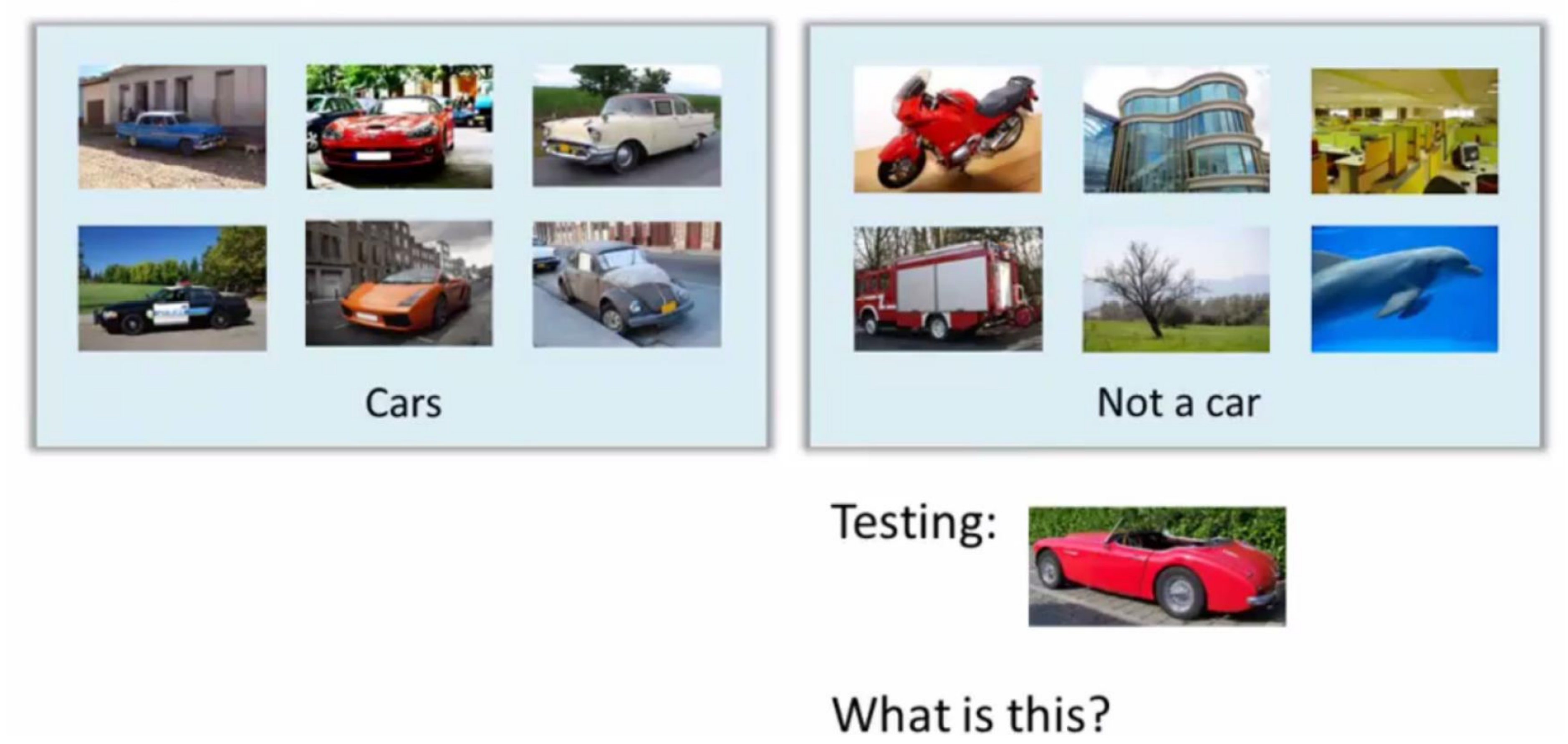
$$\begin{aligned} x_1 &= \text{size of house} \\ x_2 &= \text{no. of bedrooms} \\ x_3 &= \text{no. of floors} \\ x_4 &= \text{age of house} \\ x_5 &= \text{average income in neighborhood} \\ x_6 &= \text{kitchen size} \\ &\vdots \\ x_{100} & \end{aligned}$$

Let we have 100 original features:

If using quadratic combinations of the features to get nonlinear decision boundary, we end up with 5000 features

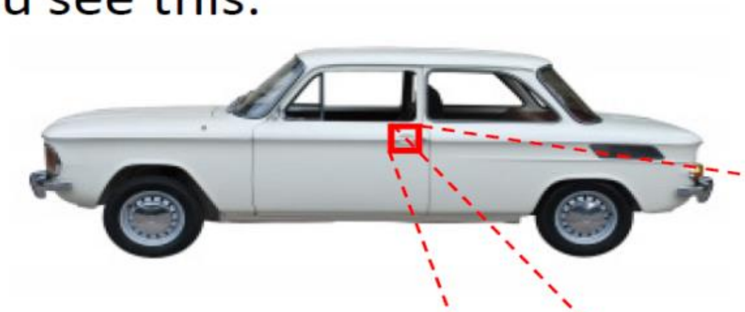
Logistic regression is not efficient for such complex nonlinear models.

Computer vision: car detection



Computer vision

You see this:

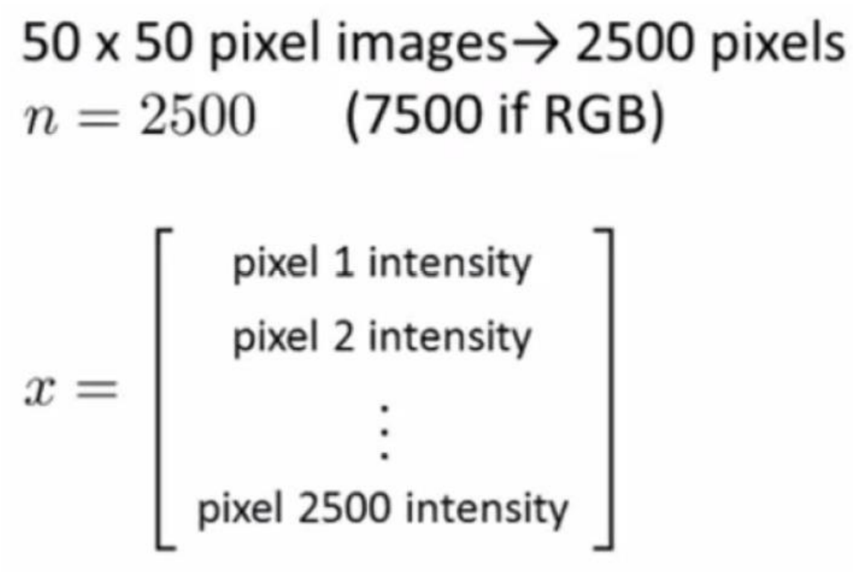


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

For a small peace of the car image we may have too many features (pixels)

Computer vision: object detection



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

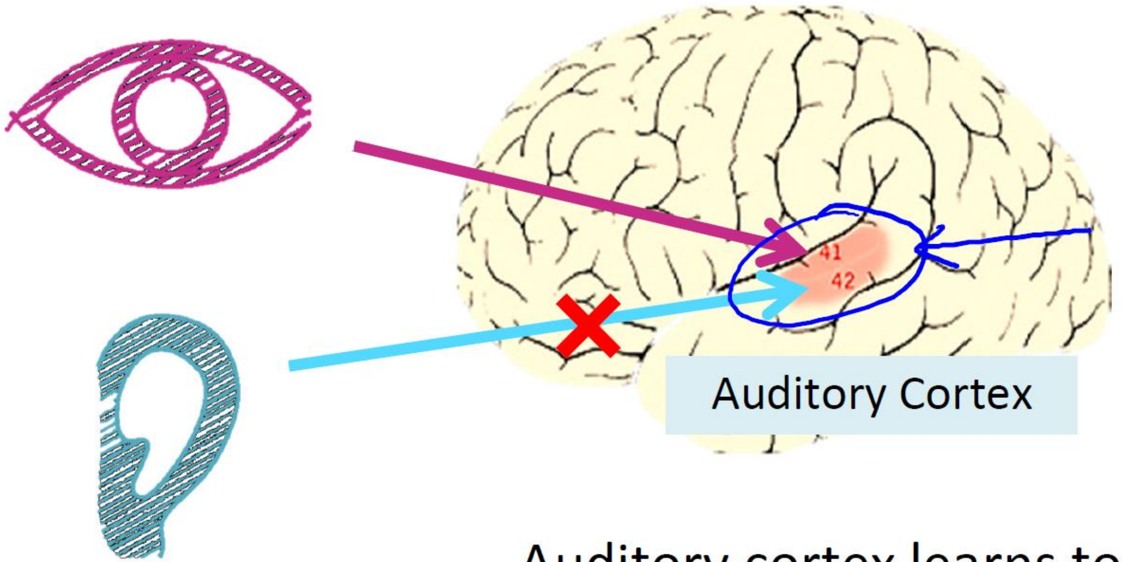
50 x 50 pixel images =>
2500 pixels (features) for a gray scale image
7500 pixels (features) for a RGB image

If using quadratic features => 3 million features

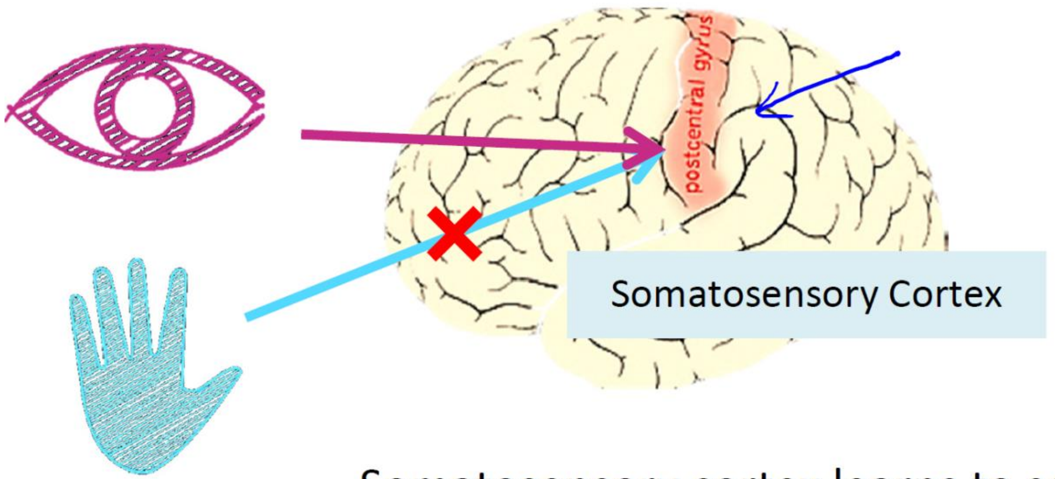
Logistic regression is not suitable for such complex nonlinear models.

Neural Networks fit better complex nonlinear models.

Brain experiments
(brain can learn from any sensor wired to it)



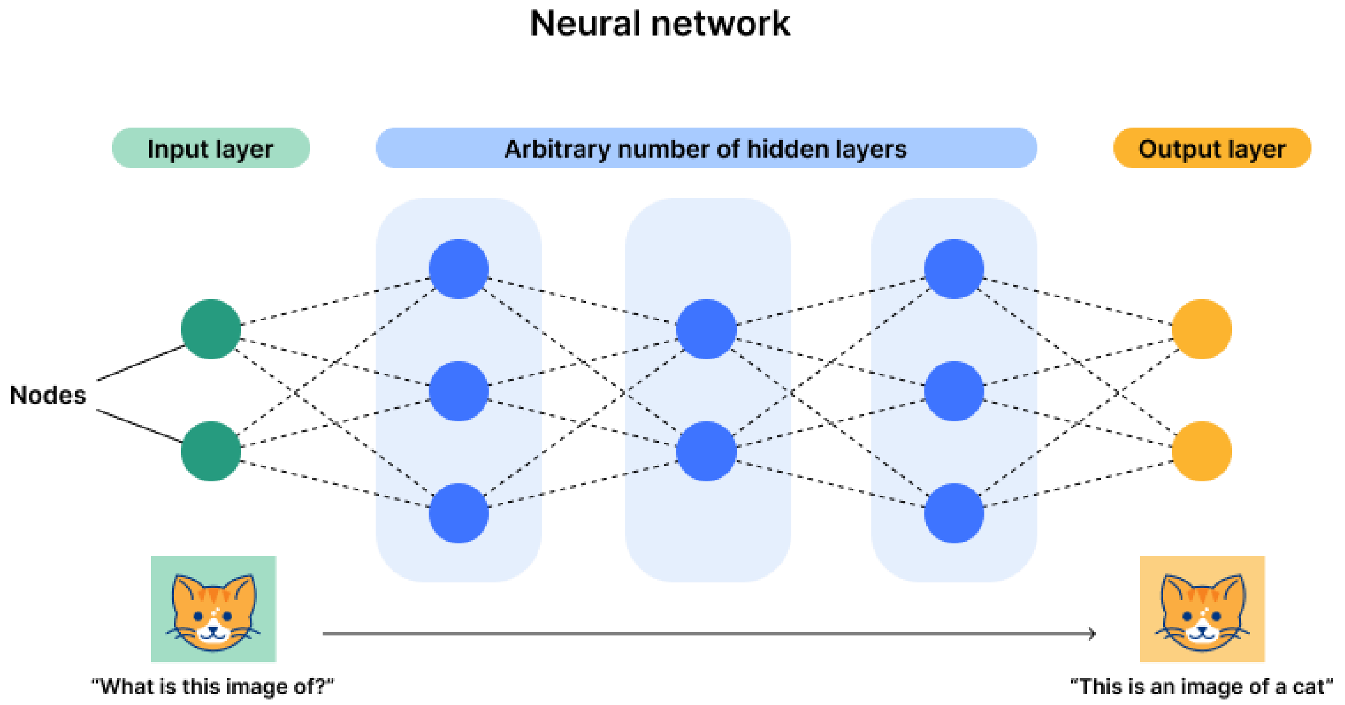
Auditory cortex learns to see



Somatosensory cortex learns to see

NN

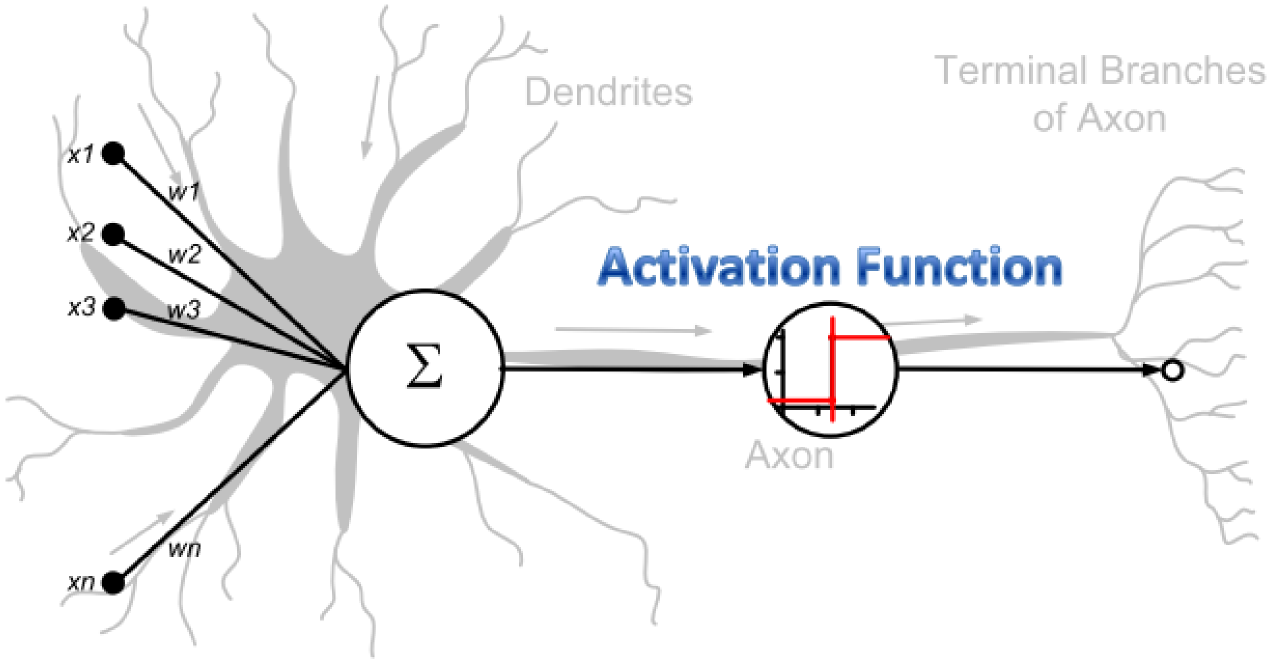
A neural network, or artificial neural network, is a computing architecture based on a model of how a human brain works — hence the name "neural.“
NN are made up of a collection of processing units called "nodes.“
These nodes pass data to each other, just like how in a brain, neurons pass electrical impulses to each other.
NN are used in machine learning, which refers to a category of computer programs that learn without definite instructions, they learn from data.



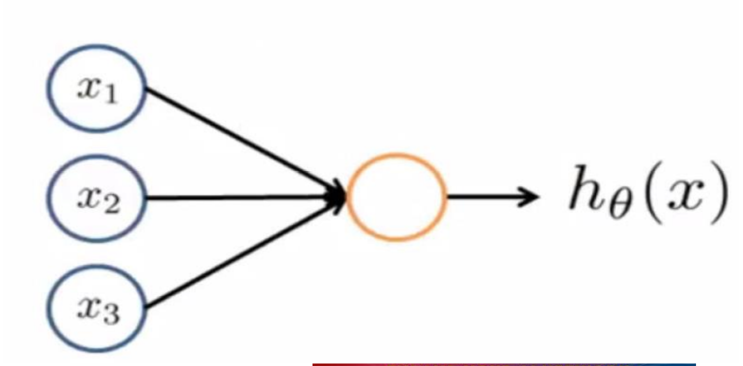
https://www.cloudflare.com/learning/ai/what-is-neural-network/

Neuron model

Origins: NN models inspired by biological neuron structures and computations.



NN are particularly useful for complex tasks where traditional machine learning algorithms fail.
The main advantage of NN is their ability to learn intricate patterns and relationships in data, even when the data is highly dimensional or unstructured.

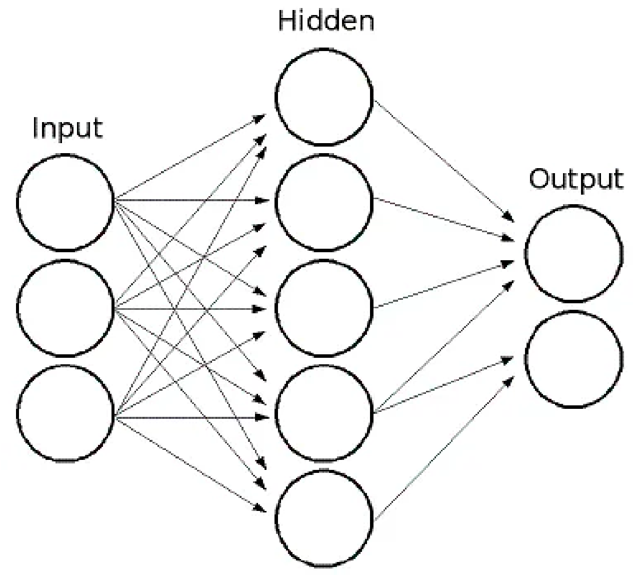


Model of artificial neuron (unit, node)

Neural Network - principle

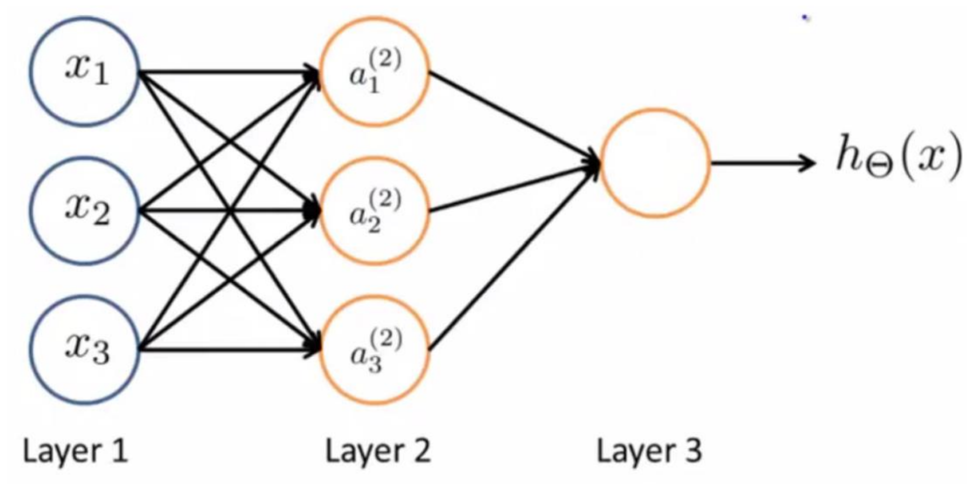
NN are composed of layers of computational units (neurons), with connections in different layers.

Goal: the network transforms the data until it is able to classify this data into an output.



How:
1. Each neuron multiplies an initial value by some weight
2. The results are summed with other values coming into the same neuron
3. The resulting number is adjusted by the neuron's bias
4. The output is normalized with an activation function.

Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

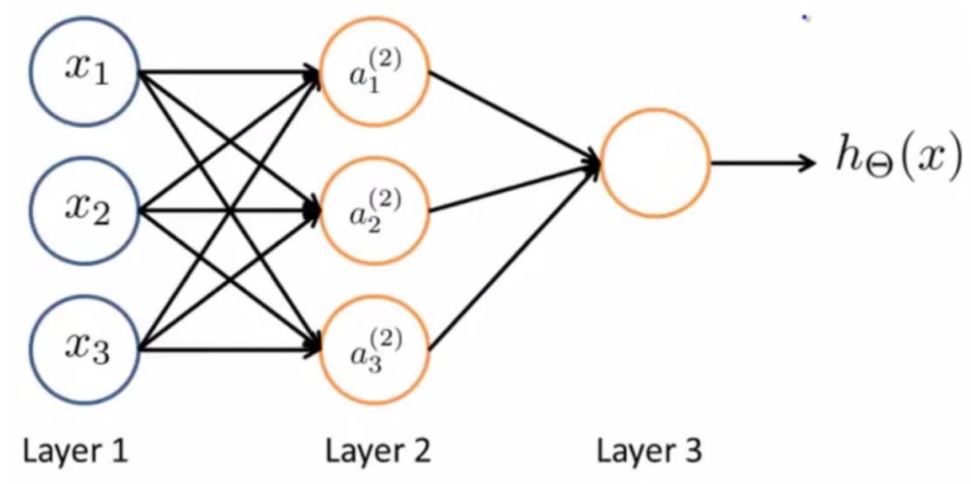
Input layer hidden layer output layer

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

universidade de aveiro

Neural Network –vectorized implementation

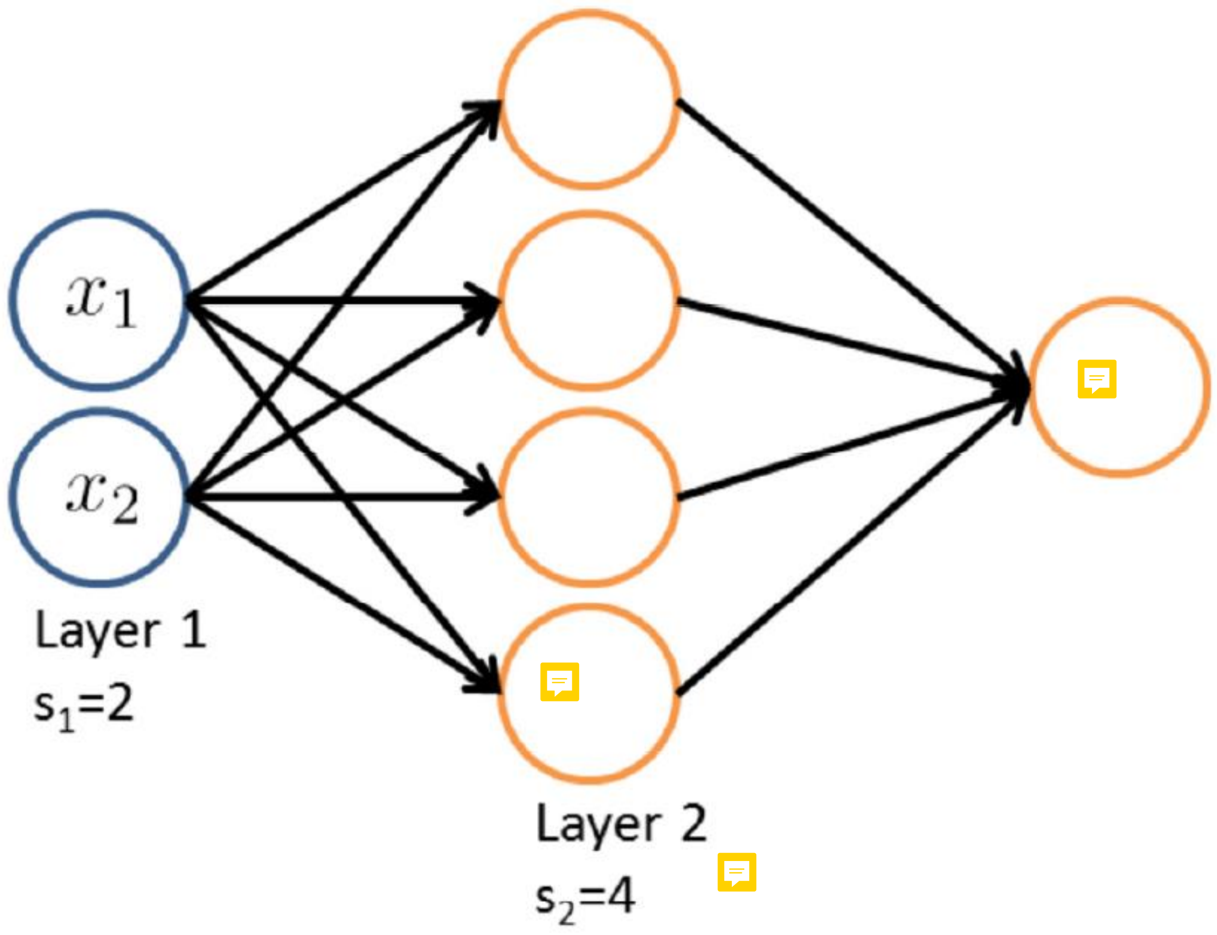


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \underline{z^{(2)}} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

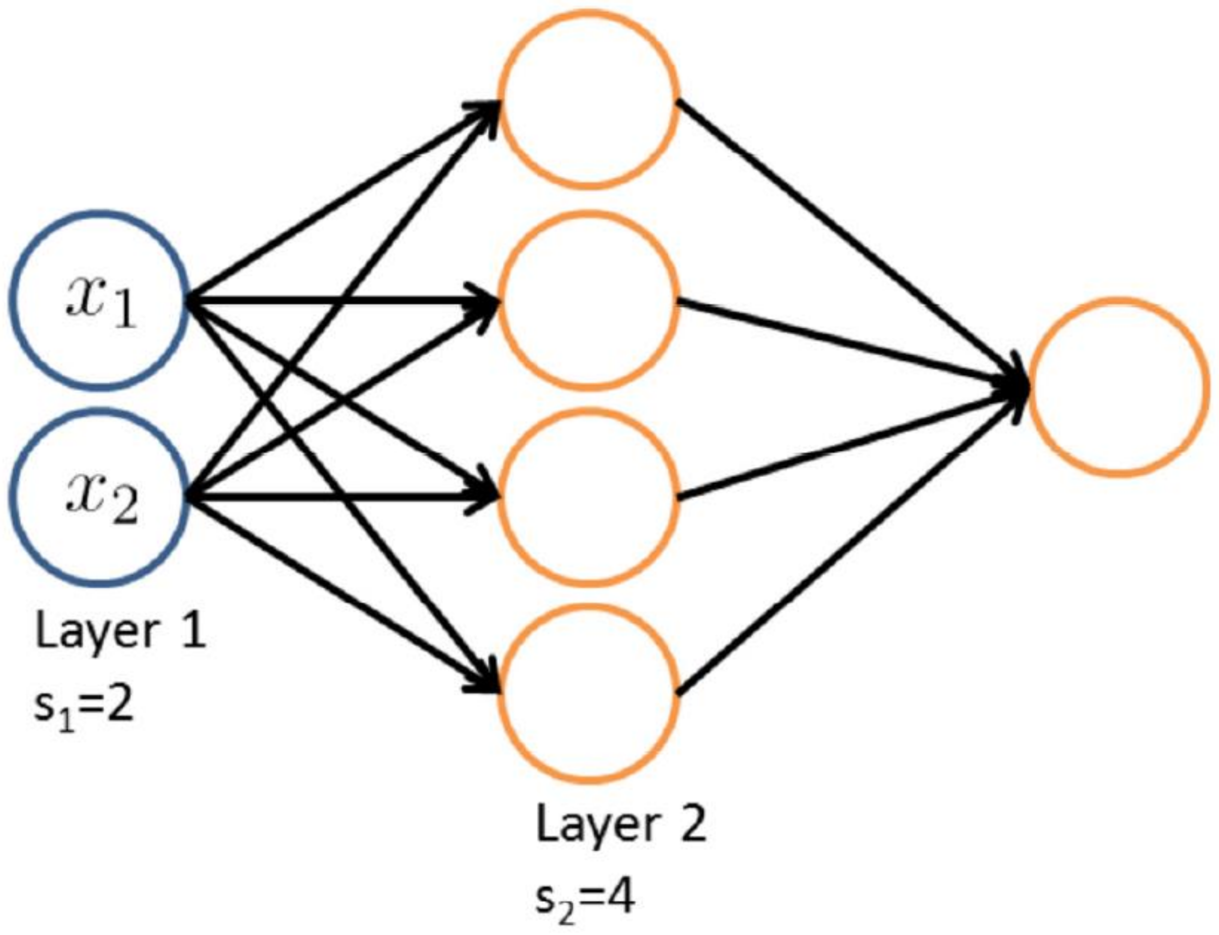
Question: how many weight matrices has the NN and what is the dymension of each matrix ?



universidade de aveiro

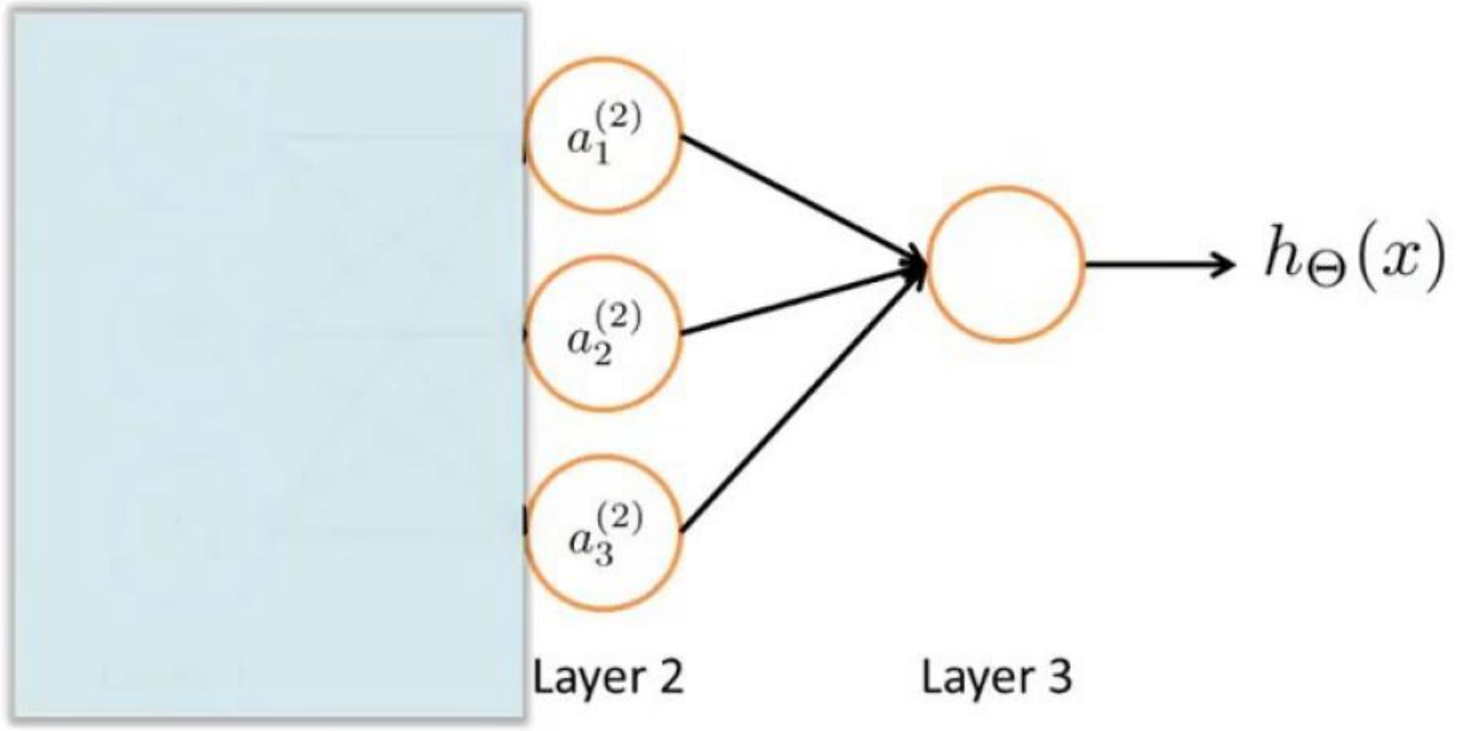
International Neural Network Society
EXPAND YOUR AI NETWORK
JOIN OUR COMMUNITY
www.inns.org

Question: how many weight matrices has the NN and what is the dymension of each matrix ?

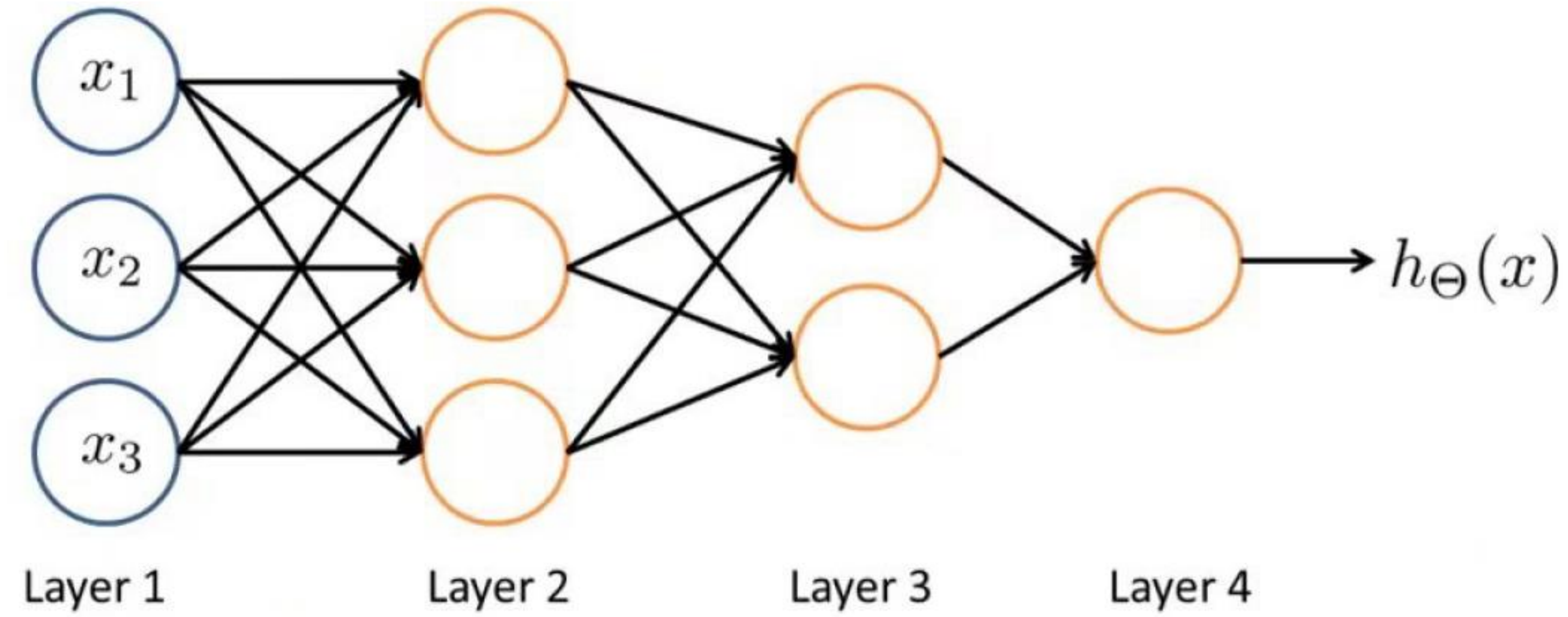


Q1=> 4x3 **Q2=> 1x5**

Neural Network is learning its own features



Other Network Architectures



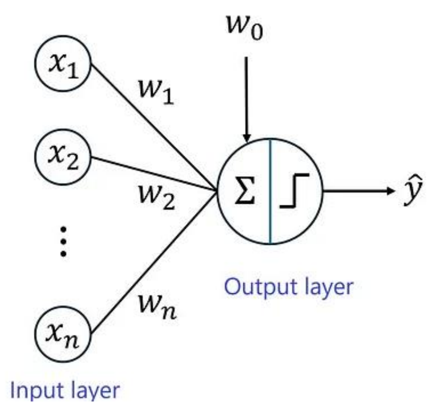
Many hidden layers can built more complex functions of the inputs (the data) => NN can learn pretty complex functions => **deep learning**

NN - base

NN are composed of a collection of nodes.
Usually, the nodes are spread out across at least three layers.

- input layer
- hidden layer
- output layer

NN can have more than one hidden layer, in addition to the input layer and output layer.



Node operation (independently of the layer)

- information processing task
- each node contains a mathematical formula, with each variable within the formula weighted differently
- (activation function) the output of applying that mathematical formula to the input:
  - If exceeds a certain threshold the node passes data to the next layer
  - if the output is below the threshold, no data is passed to the next layer.

https://www.cloudflare.com/learning/ai/what-is-neural-network/

Neuron model: logistic unit

Typical Activation functions

Properties of an activation function, that should be evaluated in its selection:

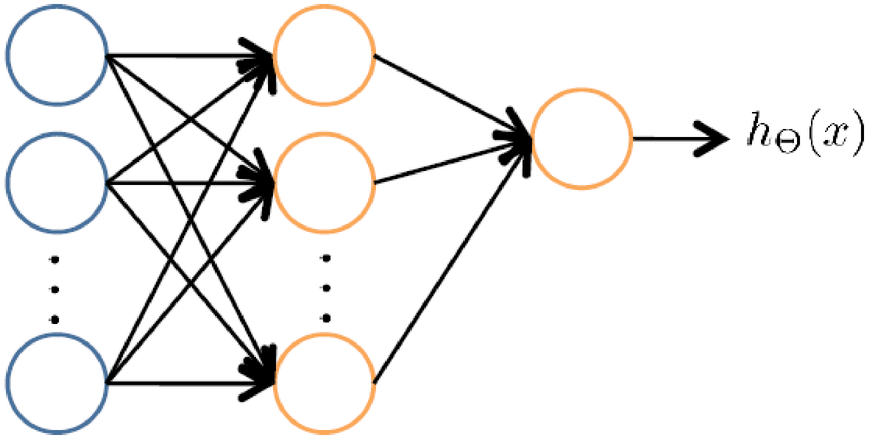
•**Nonlinearity**: The main property that comes to mind is nonlinearity. It is well-known that compared to a linear function, nonlinearity improves the training of ANN. This is mostly due to the fact that the non-linear activation function allows the ANN to separate high-dimensional nonlinear data instead of being restricted to linear space.

•**Computational cost**: The activation function is being used at every timestep during the simulations, in particular with backpropagation during training. It is thus essential to make sure that the activation function is trackable in terms of computation.

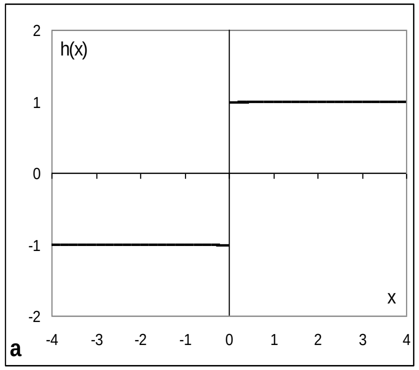
•**The gradient**: When training ANN, the gradient can be subject to vanishing or exploding gradient problems. This is due to the way activation functions are contracting the variables after every step, for example, the logistic function contracting towards [0,1]. This can lead the network to have no gradients left to propagate back after a few iterations. A solution to this is to use non-saturating activation functions.

•**Differentiability**: The training algorithms being the backpropagation algorithm, it is necessary to ensure the differentiability of the activation function to make sure the algorithm works properly.

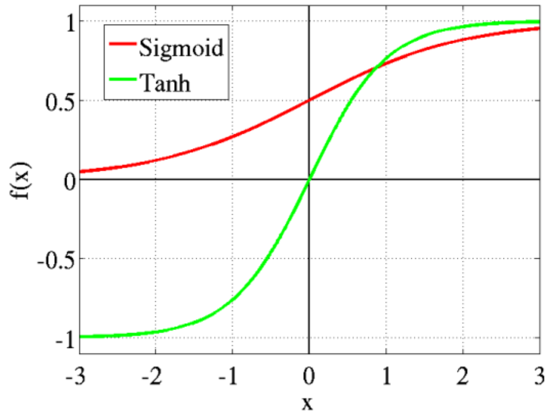
Typical Activation functions



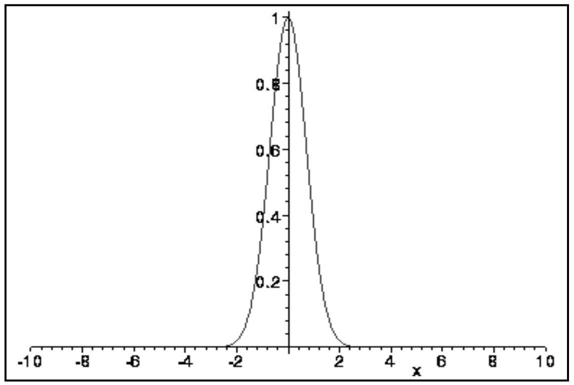
What is an Activation function in Neural networks?
Activation function **helps decide if we need to fire a neuron or not** . If we need to fire a neuron then what will be the strength of the signal.



Step (heaviside)

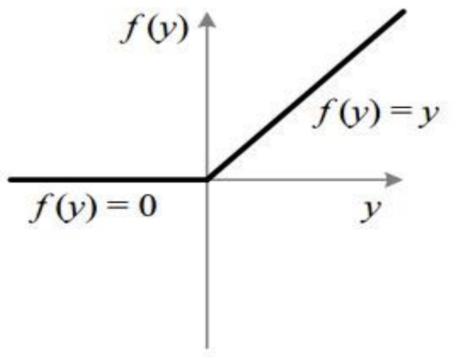


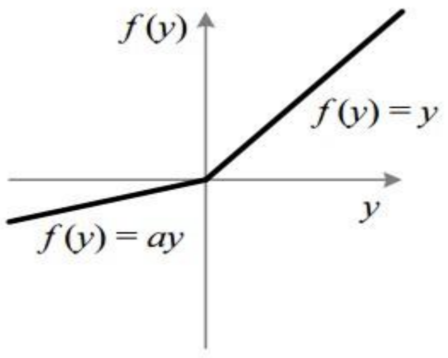
Sigmoid (logistic) vs. Hyperbolic tangent (Tanh)



Radial Basis Function (RBF)

Typical Activation functions





ReLU (Rectified Linear Unit)   vs. Leaky ReLU

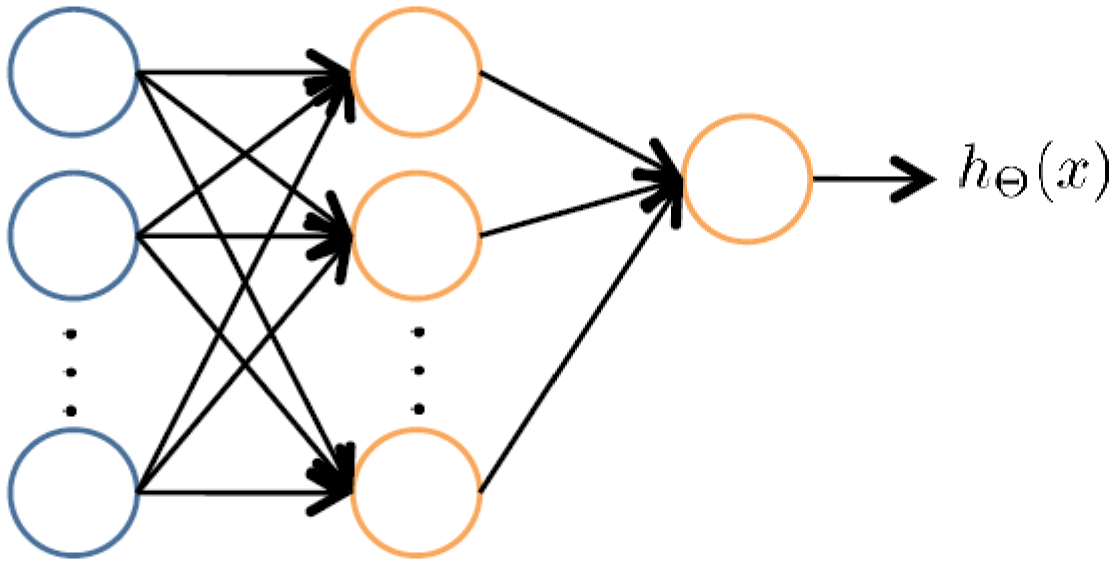
RELU:
+ Computationally efficient— the network training can converge faster
+ Non-linear (though it looks like a linear function), it is easy to compute the ReLU derivative => suitable to be used for backpropagation.
- Dying ReLU problem—when inputs approach zero, or are negative, ReLu gradient = 0, the network cannot perform backpropagation and cannot learn.

Leaky ReLU:
+ Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values.

Softmax: handles multiple classes, has as many outputs as classes. The value of each output is the probability of the class. The sum of all softmax outputs = 1.

NN - binary versus multi-class classification

NN - binary classification



Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

2 classes { 0,1 } => one output unit

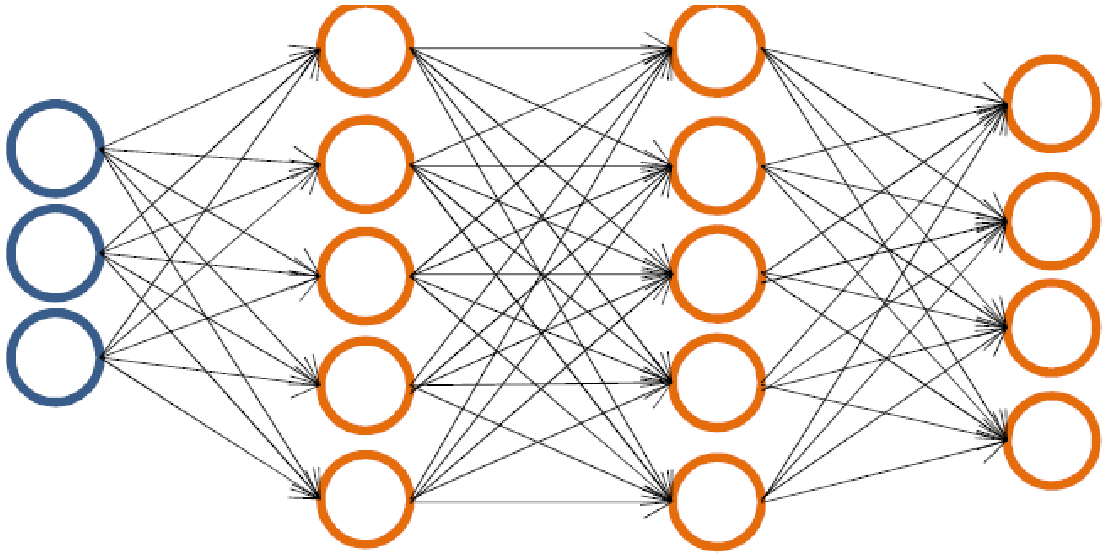
NN - multi-class classification

Pedestrian

Car

Motorcycle

Truck



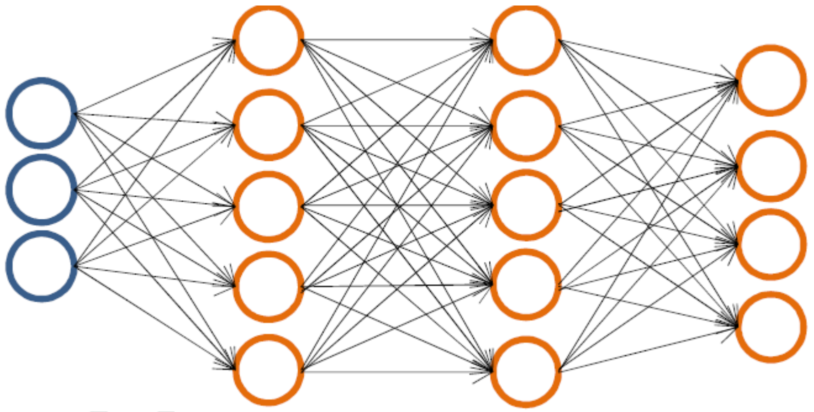
$$h_\Theta(x) \in \mathbb{R}^4$$

K classes {1,2, K} => K output units

Multiple output units: One versus all

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

universidade de aveiro

International Neural Network Society
EXPAND YOUR AI NETWORK
JOIN OUR COMMUNITY
www.inns.org

Cost function (with or without regularization)

NN Cost Functions (without regularization)

Logistic Regression (Binary cross entropy loss function) :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

NN with 1 output (logistic) unit (suitable for binary classification):
(the same as log regression)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

NN Cost Functions (without regularization)

NN with 1 output (logistic) unit (suitable for binary classification problems):

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

NN with K output (logistic) units (suitable for multiclass classif. problems):

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \boxed{\sum_{k=1}^{K}} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right]$$

NN with 1 output (not logistic) suitable for nonlinear regression problems:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

Cost Function with regularization

Regularization ter

Regularized Logistic Regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural Network with K output (logistic) units:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right]$$

Regularization term

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$L =$	total no. of layers in network
$s_l =$	no. of units (not counting bias unit) in layer l

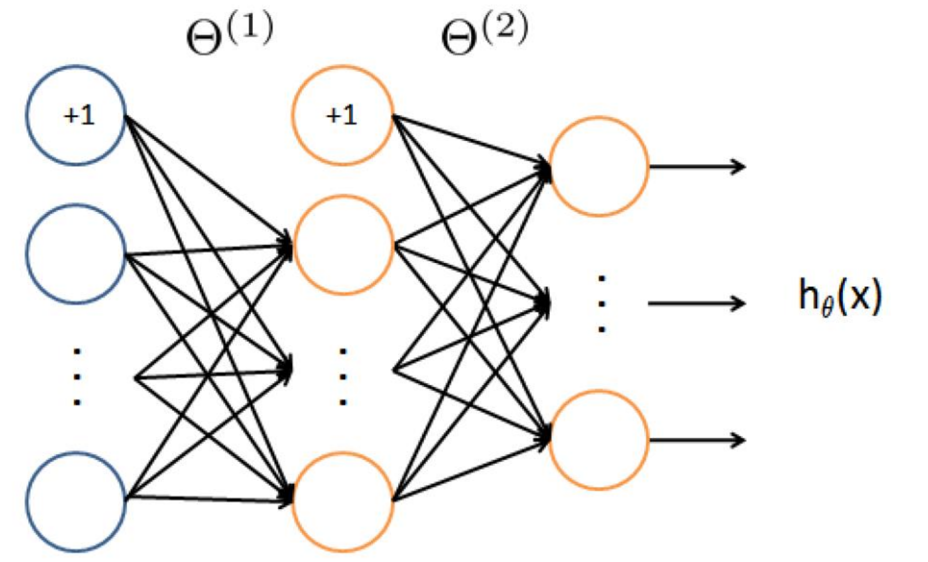
NN classification - example

MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/). 5000 training examples (20x20 pixels image, indicating the grayscale color intensity). The image is transformed into a row vector (with 400 elements). This gives 5000 x 400 data matrix X (every row is a training example).

NN model - example

input layer – 400 units = 20x20 pixels (input features) + 1 unit(=1, the bias)
hidden layer – 25 units + 1 unit(=1, the bias)
output layer - 10 output units (corresponding to 10 digit classes 0,1,2....9).

Matrix parameters: Θ_1 has size 25x401; Θ_2 has size 10x26.



$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots \text{ or } \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

$a^{(1)} = x$ (add $a_0^{(1)}$)

$z^{(2)} = \Theta^{(1)} a^{(1)}$
$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

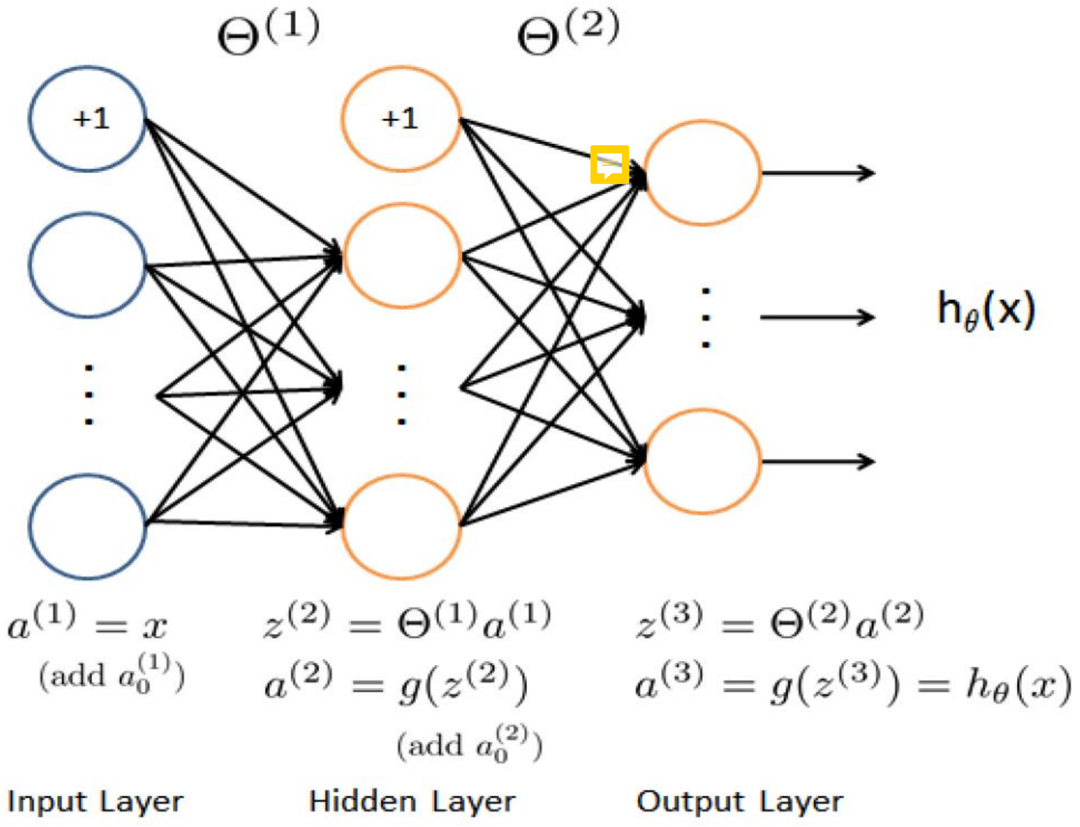
$z^{(3)} = \Theta^{(2)} a^{(2)}$
$a^{(3)} = g(z^{(3)}) = h_\theta(x)$

Input Layer Hidden Layer Output Layer

NN model learning – forward pass

- Randomly initialize the NN parameters (matrices Q_1 and Q_2).
- Provide features as inputs to the NN, make a forward pass to compute all activations through the NN and the NN outputs.
- Repeat for all examples (batch training)

Feedforward Neural Networks
Feedforward neural networks are the simplest type of neural network. Information flows in one direction, from the input layer to the output layer, without any loops or feedback connections. These networks are commonly used for classification and regression tasks.



$$a^{(1)} = x \quad (\text{add } a_0^{(1)})$$

$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$

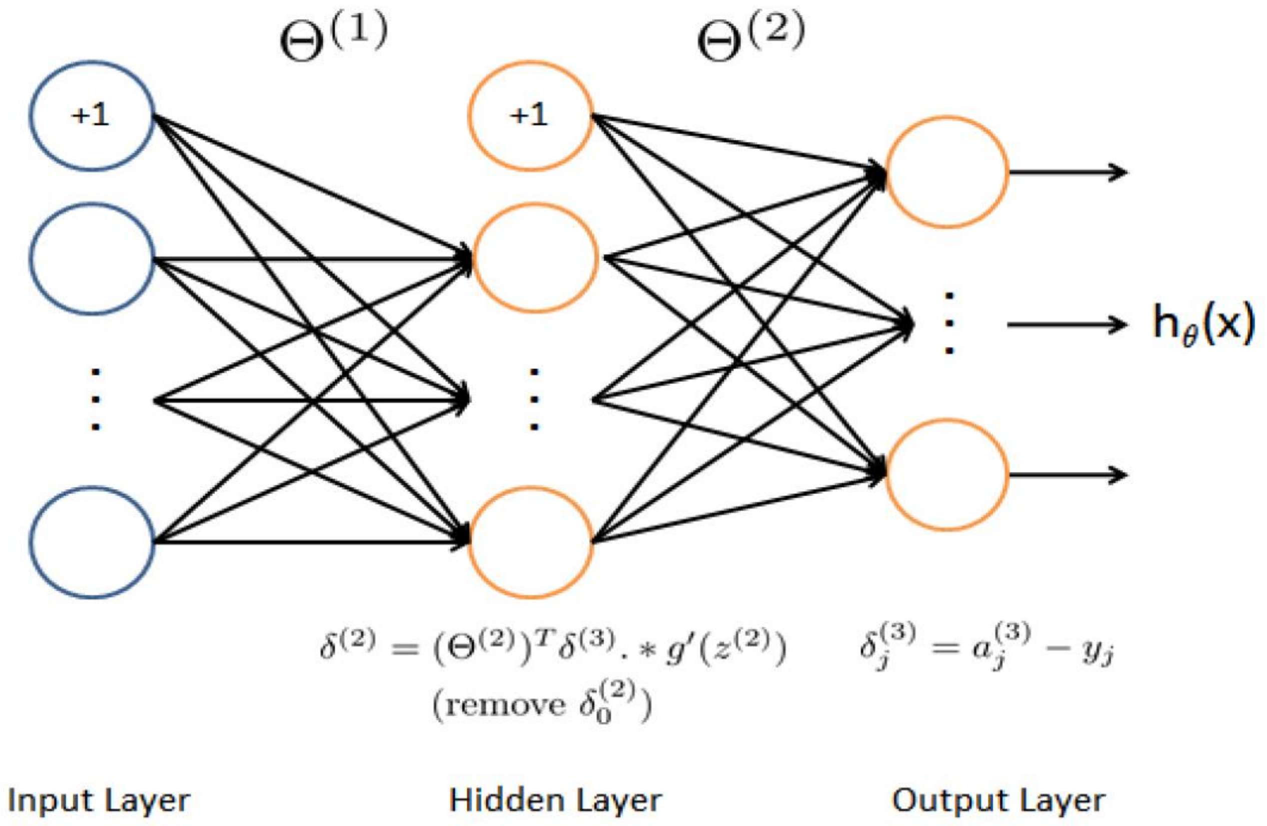
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) = h_\theta(x)$$

NN learning - Error Backpropagation algorithm

NN model learning -Error Backpropagation

- Compute the output error (the difference between the NN output value and the true target value).
- For all hidden layer nodes compute an “error term" that measures how much that node was “responsible" for the NN output error.
- Compute the gradient as sum of the accumulated errors for all examples.
- Update the weights.

After each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases).



$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)}) \quad \text{(remove } \delta_0^{(2)}\text{)}$$

$$\delta_j^{(3)} = a_j^{(3)} - y_j$$

universidade de aveiro

EXPAND YOUR AI NETWORK JOIN OUR COMMUNITY www.inns.org

Error Backpropagation algorithm

0) Randomly initialize the parameters (matrices Θ_1 and Θ_2)
1) For ii =1:number of examples (m)
2) Provide training example *ii* at the NN input.
3) Perform a feedforward pass to compute z2, a2 (for the hidden layer) and z3, a3 (for the output layer)

4) For each unit k in the output layer compute: $\delta_k^{(3)} = (a_k^{(3)} - y_k)$

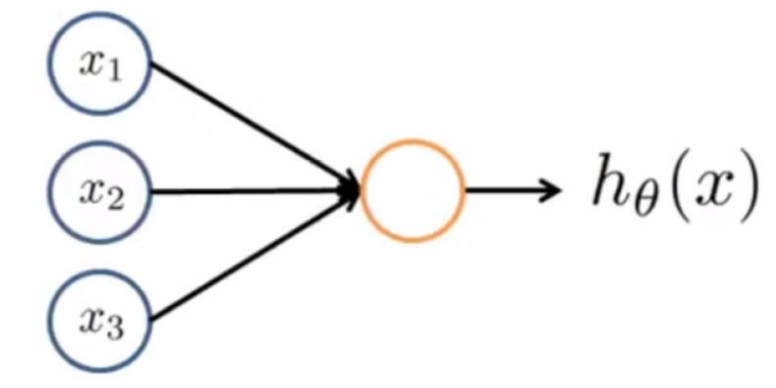
5) For the hidden layer, compute:
(***error backpropagation***) $\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$

6) Accumulate the gradient from this example: $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

7) NN gradient (no regularization) $$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \frac{1}{m}\Delta_{ij}^{(l)}$$

8) Update NN parameters: $$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$$

Sigmoid gradient



$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

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

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

universidade de aveiro

International Neural Network Society EXPAND YOUR AI NETWORK JOIN OUR COMMUNITY www.inns.org

Regularized Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] +$$

Regularization term

$$\frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right]$$

After computing the gradient by backpropagation, add the regularization term

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

universidade de aveiro

Adaptive learning rate

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

α -**Learning rate**

- **Fixed or**
- **Adaptive:**

$$\alpha^{(r+1)} = \begin{cases} b\alpha^{(r)} & if \quad J^{(r+1)} \leq J^{(r)}, \quad b \geq 1\,(\text{ex.}\,b = 1.2) \\ b\alpha^{(r)} & if \quad J^{(r+1)} > J^{(r)}, \quad b < 1\,(\text{ex.}\,b = 0.2) \end{cases} \qquad \alpha^{(0)} = 0.01$$

Gradient Descent with momentum (extra term - momentum)

$$\theta_j^{(r)} = \theta_j^{(r-1)} - \alpha \frac{\partial J}{\partial \theta_j} + \boxed{\beta\left(\theta_j^{(r-1)} - \theta_j^{(r-2)}\right)}$$

β **<u>- coefficient of momentum</u>**

•Increase convergence rate far from minima

•Slow down near minima

Gradient Descent with momentum is analogous to a ball moving on a surface with multiple valleys, accelerating on steep slides and decelerating when it reaches a valley.
The intuition behind is to add inertia to the gradient descent so that it smooth's the overall trajectory, in order to find better convergence points.

NN Parameters (weights) Initialization

- Setting the weights to zero (Simplest approach)
However, by initializing every weight to zero, every neuron will have the same activations, all the calculated gradients will be the same, and consequently, each parameter will suffer the same update. Therefore, it is crucial that the initialization of the weights breaks the symmetry between different units.

- **Drawn from random Gaussian distribution with mean 0 & deviation 1** may lead to vanishing gradients

Empirical initializations:
-**Xavier/ Glorot's initialization:** drawn from uniform distribution near zero.

$$\sim U\left(-\frac{\sqrt{6}}{\sqrt{m}}, \frac{\sqrt{6}}{\sqrt{m}}\right)$$

- **LeCun initialization:**

$$\sim U\left(-\frac{\sqrt{3}}{\sqrt{m}}, \frac{\sqrt{3}}{\sqrt{m}}\right)$$

Advantages

•Ability to Learn Complex Patterns: Neural networks can discover intricate patterns and relationships within data, even when they are not explicitly defined.

•Adaptability: They can adapt to changing conditions by adjusting their internal parameters, making them flexible and robust.

•Parallel Processing: Neural networks can process multiple inputs simultaneously, enabling fast and efficient computations.

•Non-linear Transformations: Activation functions introduce non-linearities, allowing neural networks to model complex non-linear relationships.

Limitations

•Computational Requirements: Training and evaluating large neural networks can be computationally intensive, requiring substantial computing resources.

•Data Dependency: Neural networks heavily rely on large, high-quality datasets for effective training and generalization. Insufficient or biased data can impact their performance.

•Lack of Explainability: Neural networks are often referred to as “black boxes” since it can be challenging to understand their decision-making process. Interpreting their innerworkings and providing explanations for their predictions can be difficult.

Examples of NN algorithms

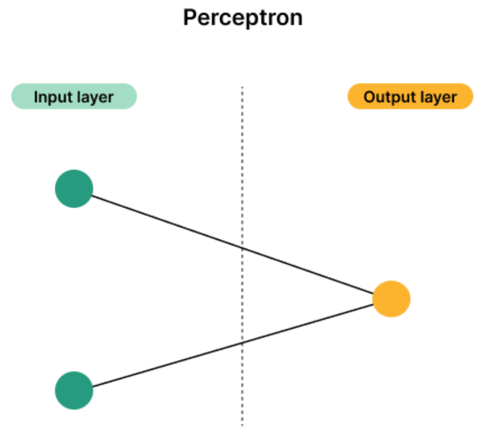
There is no limit on how many nodes and layers a neural network can have, and these nodes can interact in almost any way. Because of this, the list of types of neural networks is ever-expanding.

But, they can roughly be sorted into these categories:

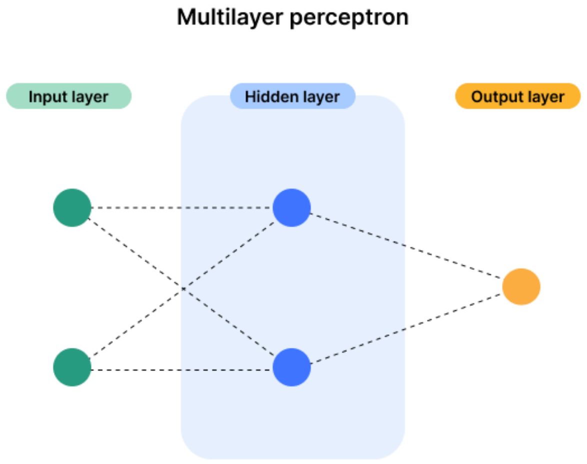
- **Shallow neural networks** usually have only one hidden layer
- **Deep neural networks** have multiple hidden layers

Shallow neural networks are fast and require less processing power than deep neural networks, but they cannot perform as many complex tasks as deep neural networks.

Examples of NN algorithms

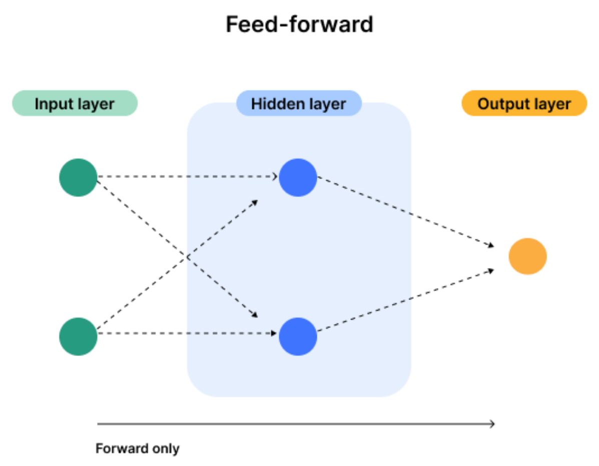


Perceptron neural networks are simple, shallow networks with an input layer and an output layer.

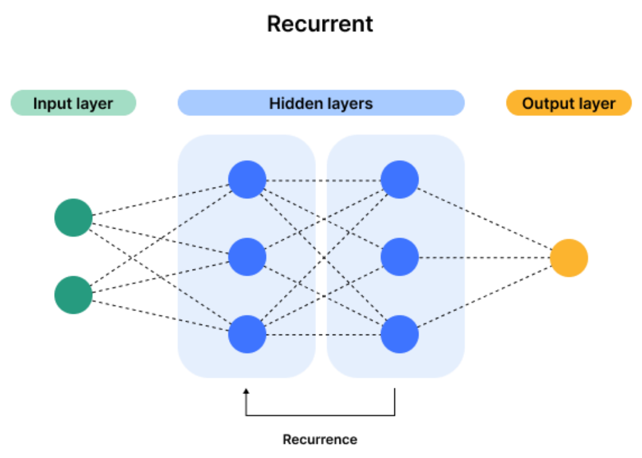


Multilayer perceptron neural networks add complexity to perceptron networks, and include a hidden layer.

Examples of NN algorithms

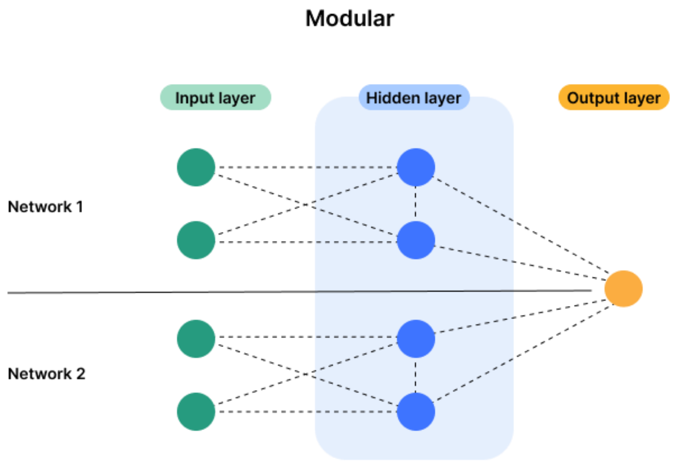


Feed-forward neural networks only allow their nodes to pass information to a forward node.

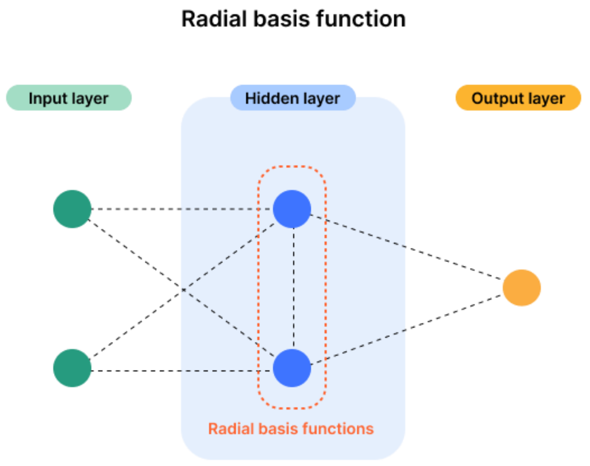


Recurrent neural networks (RNN) can go backwards, allowing the output from some nodes to impact the input of preceding nodes. These learning algorithms are primarily leveraged when using time-series data to make predictions about future outcomes, such as stock market predictions or sales forecasting.

Examples of NN algorithms

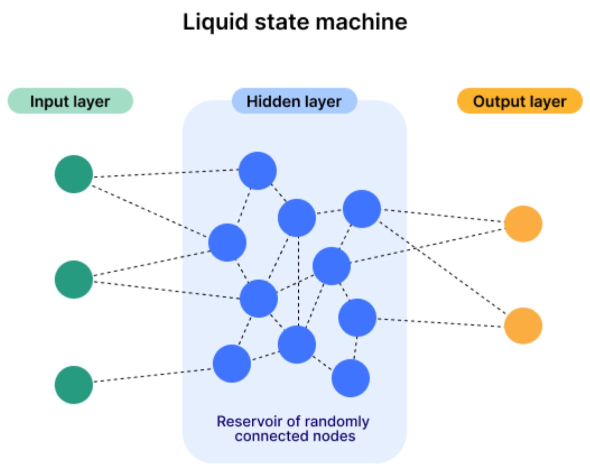


Modular neural networks combine two or more neural networks in order to arrive at the output.

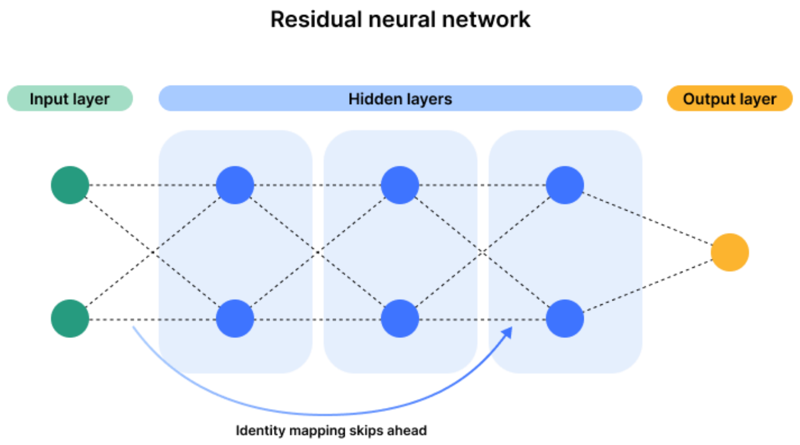


Radial basis function neural network nodes use a specific kind of mathematical function called a radial basis function.

Examples of NN algorithms

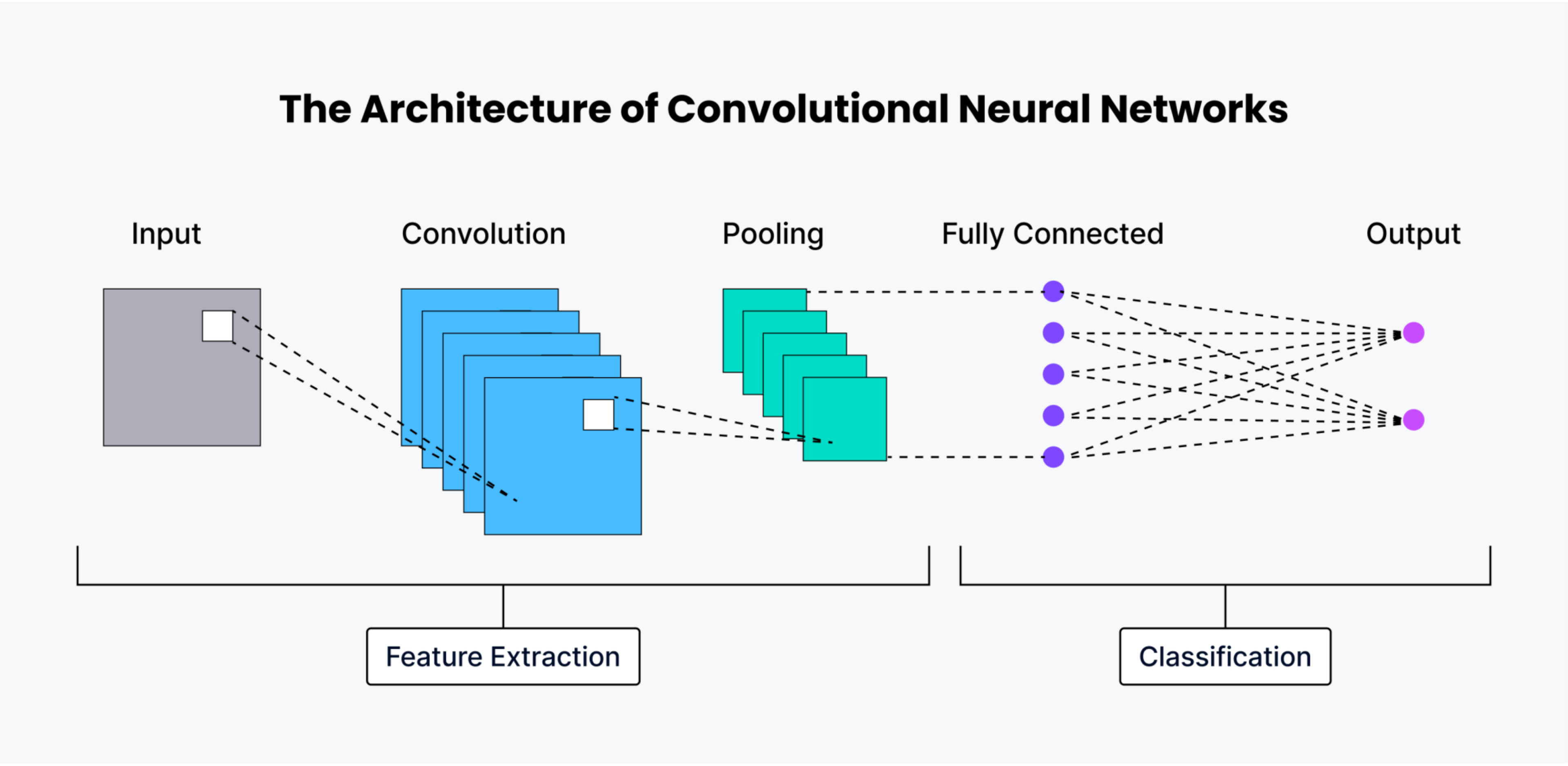


Liquid state machine neural networks feature nodes that are randomly connected to each other.



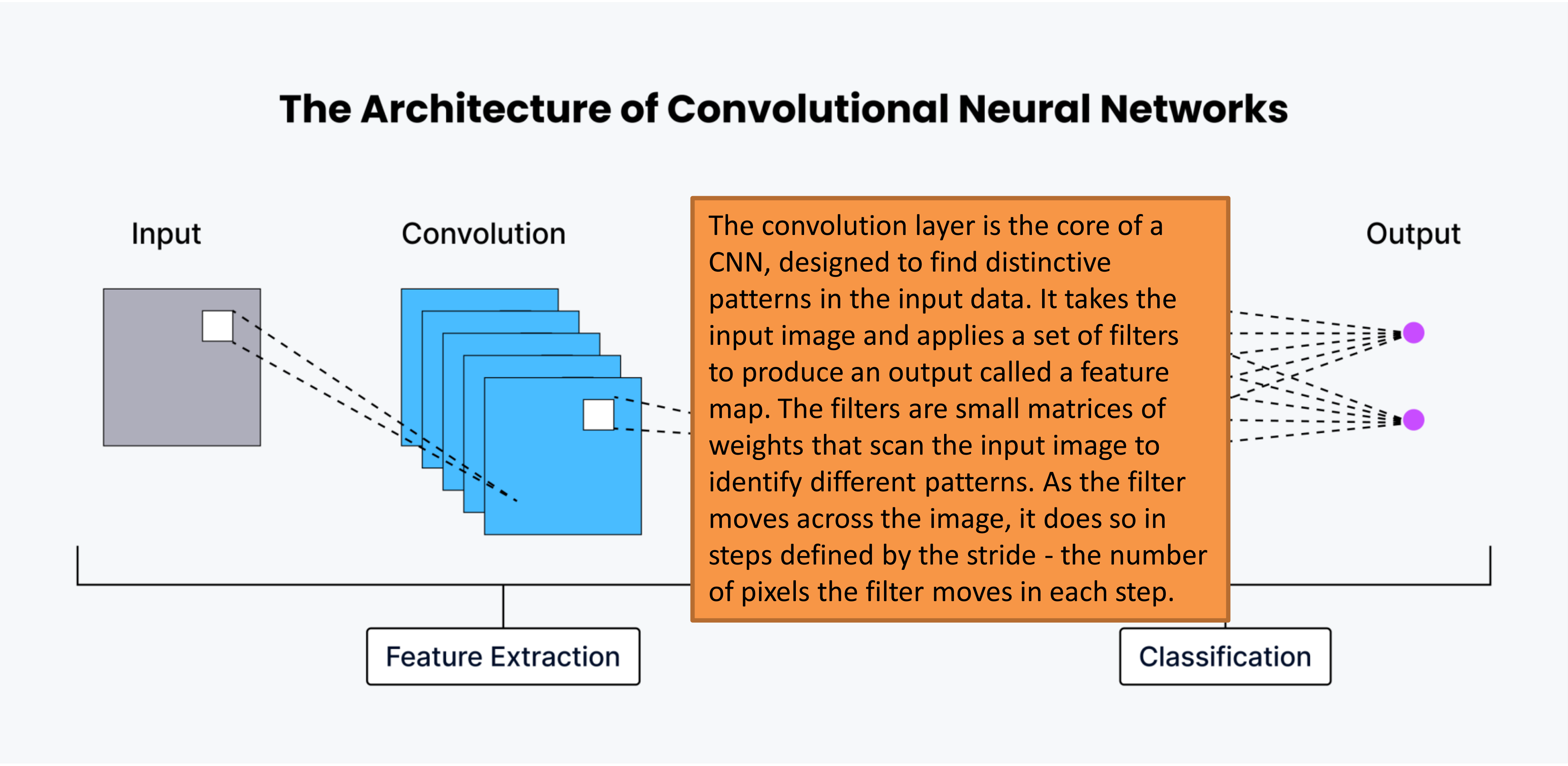
Residual neural networks allow data to skip ahead via a process called identity mapping, combining the output from early layers with the output of later layers.

Examples of NN algorithms



Convolutional neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.
They preserve image structure, such as local connectivity and content of the pixels of the image data, making them efficient at pattern recognition.

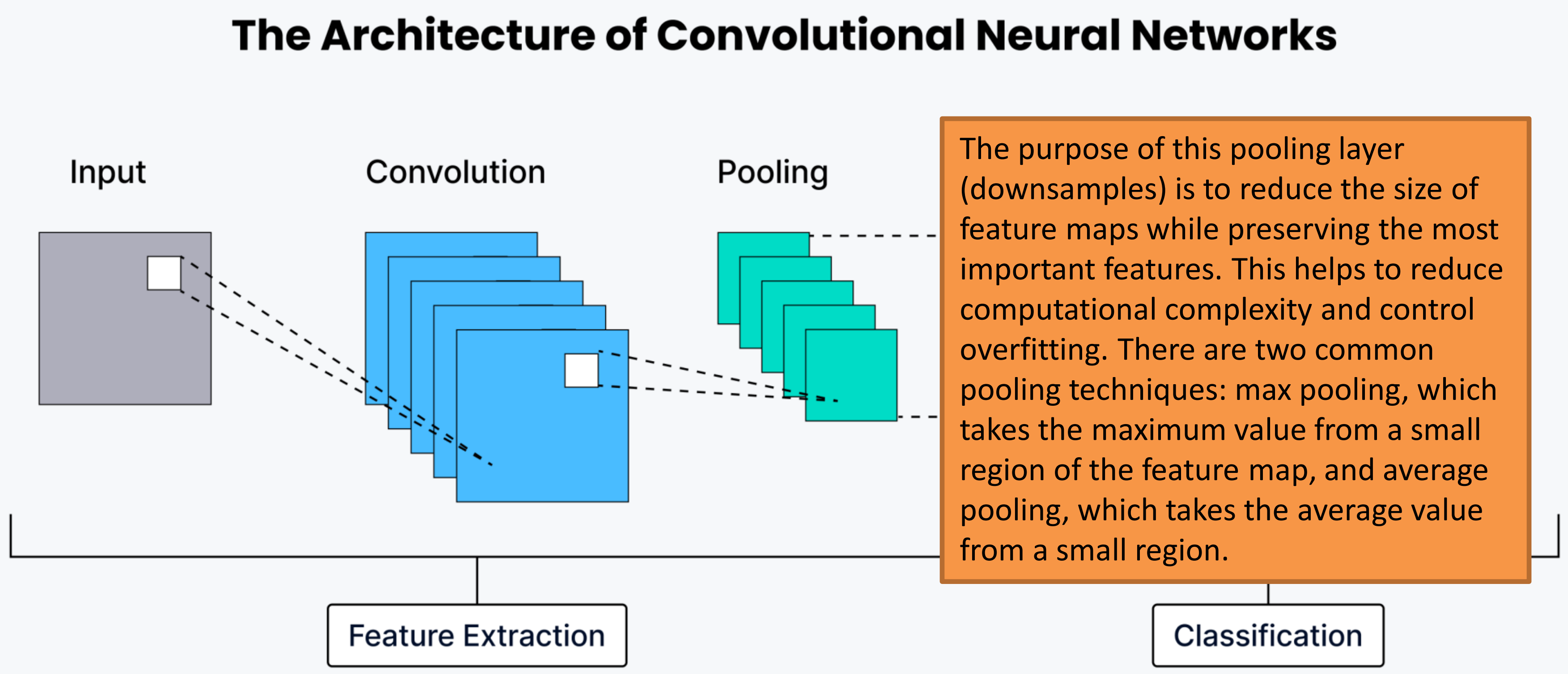
Examples of NN algorithms



The convolution layer is the core of a CNN, designed to find distinctive patterns in the input data. It takes the input image and applies a set of filters to produce an output called a feature map. The filters are small matrices of weights that scan the input image to identify different patterns. As the filter moves across the image, it does so in steps defined by the stride - the number of pixels the filter moves in each step.

Convolutional neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.
They preserve image structure, such as local connectivity and content of the pixels of the image data, making them efficient at pattern recognition.

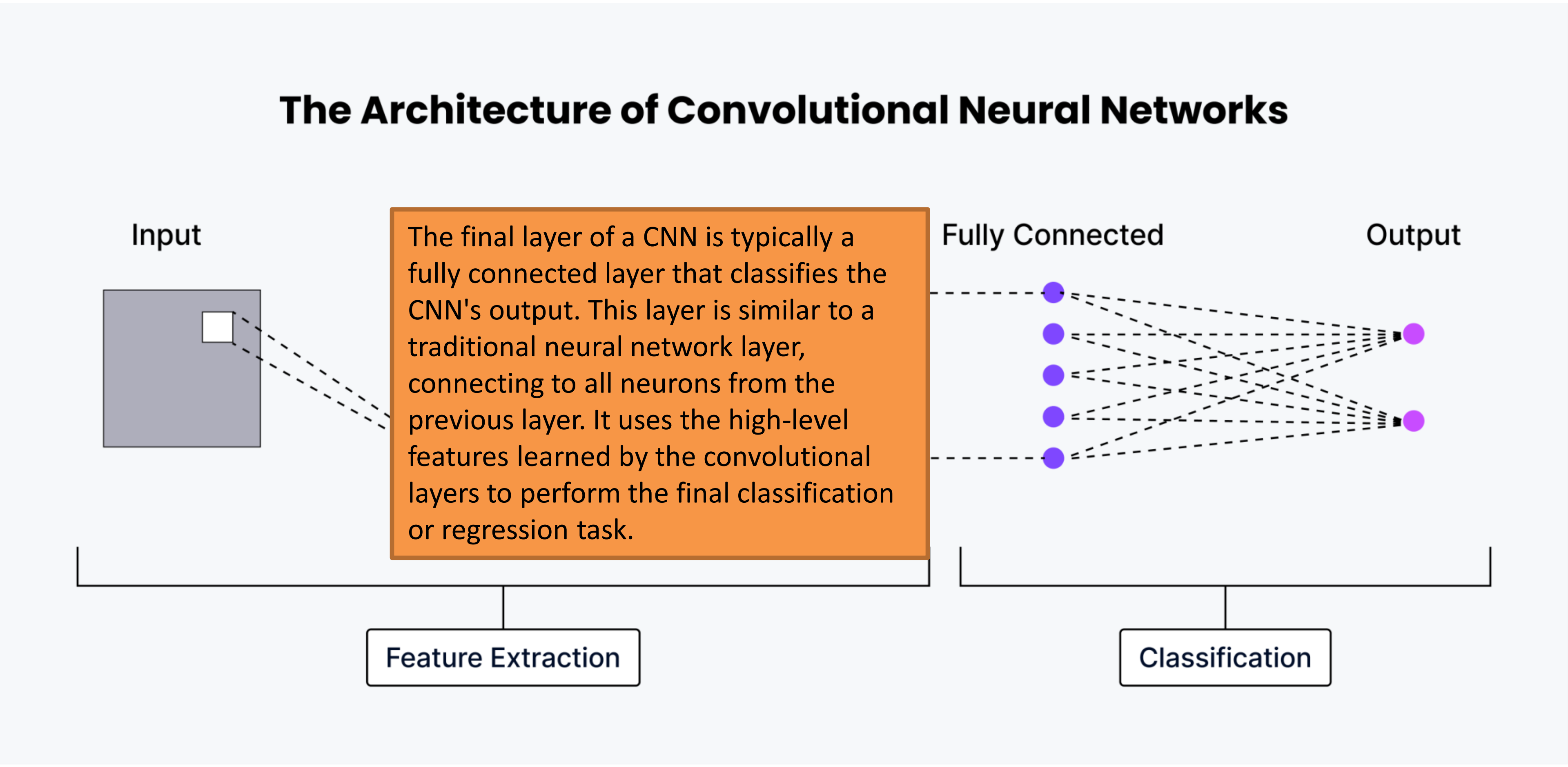
Examples of NN algorithms



The purpose of this pooling layer (downsamples) is to reduce the size of feature maps while preserving the most important features. This helps to reduce computational complexity and control overfitting. There are two common pooling techniques: max pooling, which takes the maximum value from a small region of the feature map, and average pooling, which takes the average value from a small region.

Convolutional neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.
They preserve image structure, such as local connectivity and content of the pixels of the image data, making them efficient at pattern recognition.

Examples of NN algorithms



The final layer of a CNN is typically a fully connected layer that classifies the CNN's output. This layer is similar to a traditional neural network layer, connecting to all neurons from the previous layer. It uses the high-level features learned by the convolutional layers to perform the final classification or regression task.

Convolutional neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.
They preserve image structure, such as local connectivity and content of the pixels of the image data, making them efficient at pattern recognition.

Deep Learning

Specifically, neural networks are used in deep learning — an advanced type of machine learning that can draw conclusions from unlabeled data without human intervention. For instance, a deep learning model built on a neural network and fed sufficient training data could be able to identify items in a photo it has never seen before.

Neural networks make many types of artificial intelligence (AI) possible. Large language models (LLMs) such as ChatGPT, AI image generators like DALL-E, and predictive AI models all rely to some extent on neural networks.

Transformer neural networks assumed a place of outsized importance in the AI models in widespread use today. First proposed in 2017, transformer models are neural networks that use a technique called "self-attention" to take into account the context of elements in a sequence, not just the elements themselves. Via self-attention, they can detect even subtle ways that parts of a data set relate to each other.

Transformer models are an integral component of generative AI, in particular LLMs that can produce text in response to arbitrary human prompts.