Artificial Neural Networks

Fátima Rodrigues
mfc@isep.ipp.pt
Departamento de Engenharia Informática (DEI/ISEP)

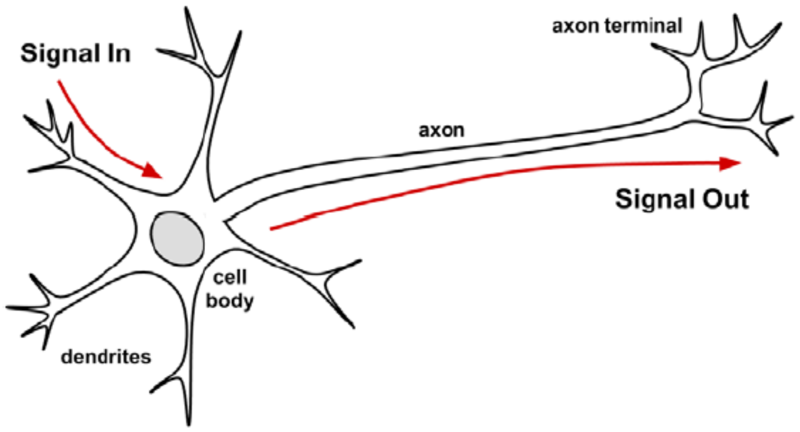
Symbolic Learning vs. Neuronal Learning

- Symbolic Learning
  - induction of Rules and Decision Trees
  - works with discrete combinations of attribute values
  - uses logical / relational operators (=,>, <)

- Neuronal Learning
  - works by adjusting non-linear and continuous weights of its inputs
  - uses numeric operators (×, +)
  - makes a search in a finer space of granularity than the algorithms of induction of rules

Artificial Neural Network

- Inspired in the human brain consist of a huge number of neurons with extremely high inter-connectivity



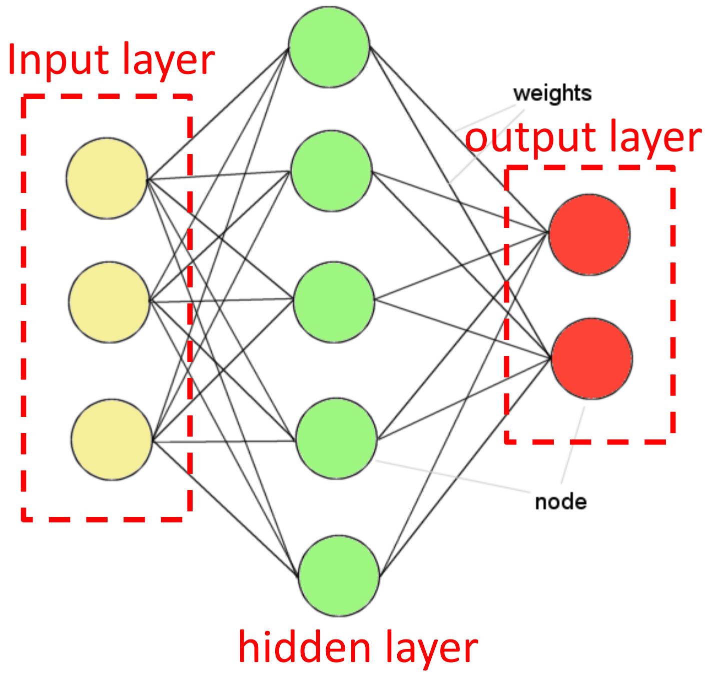
- ANNs incorporate the two fundamental components of biological neural nets:
  1. Neurones (nodes)
  2. Synapses (weights)

Neural Network – The basics

A Neural Network consists of:

– Neurons (processing elements)

– Inter-connections between neurons with numerical weights

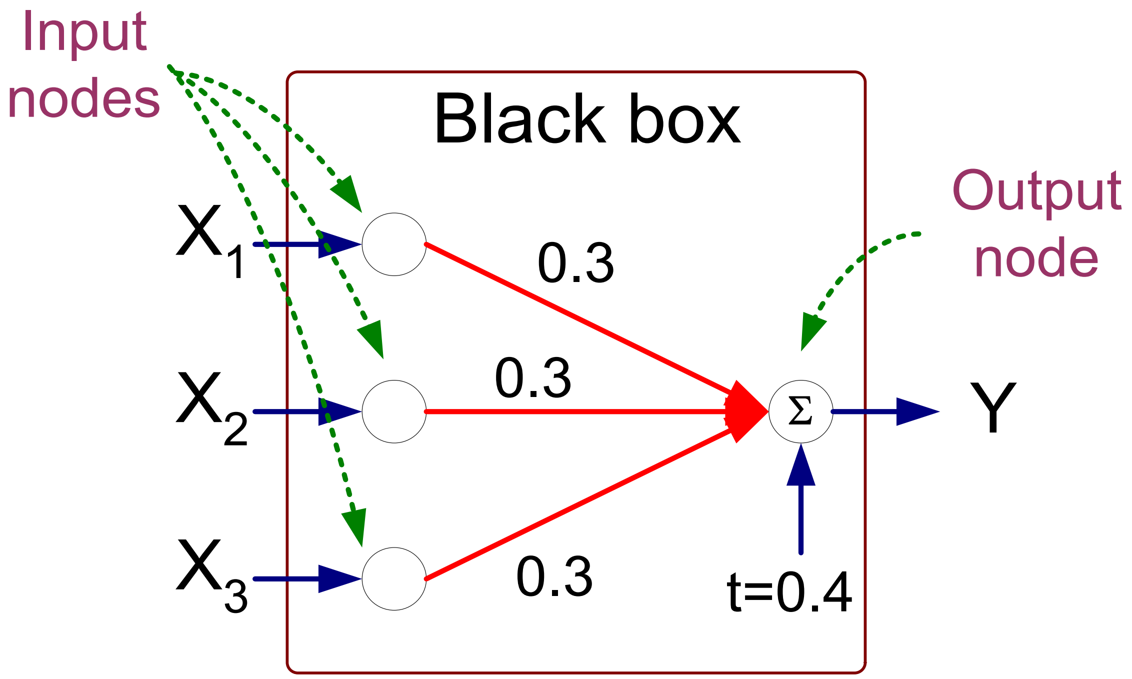


Learning process

– Consists in adjusting the intensity of the connections between the neurons (synapses) represented by weights, during the training process of the network

Neural Networks – The basics

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0

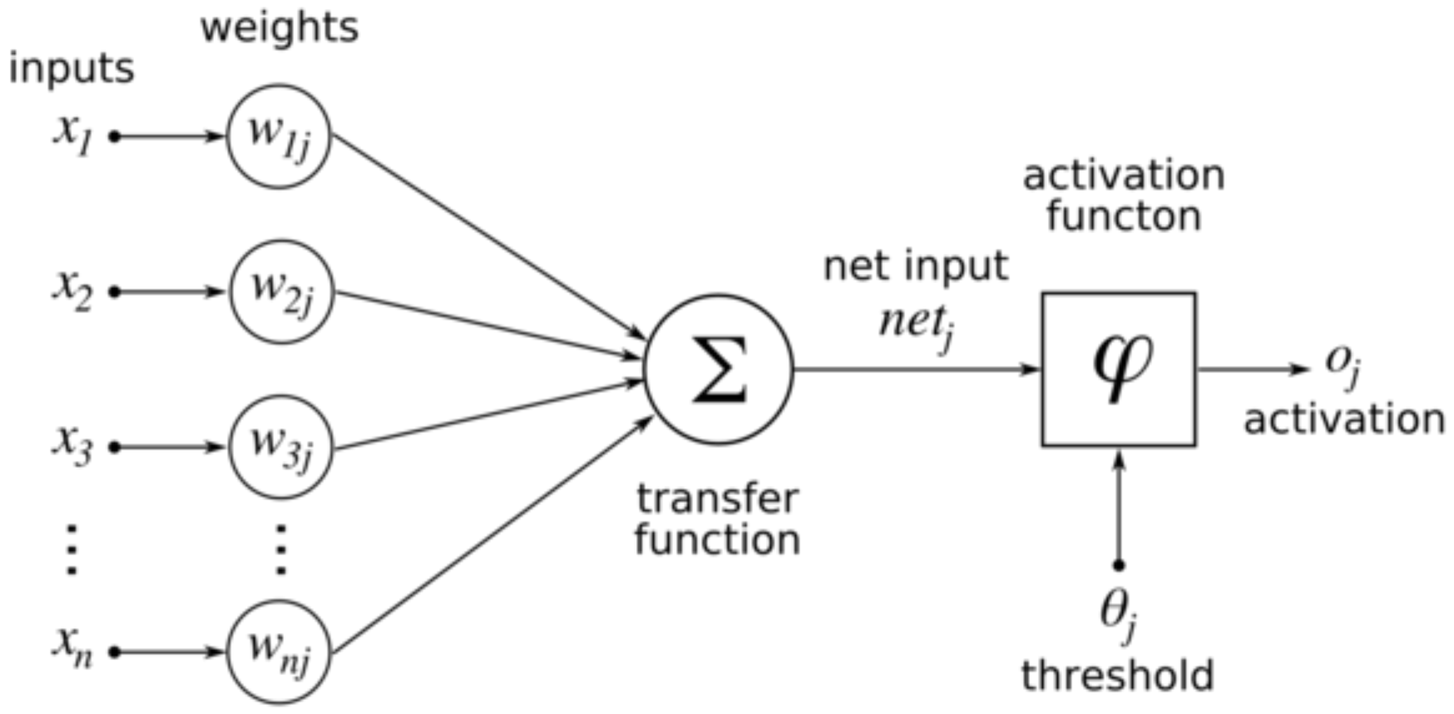


A neural network maps the input variables to an output variable y

- The input variables corresponds to the attributes or characteristics of our problem
- The output variable can be discrete (classification) or continuous (regression)

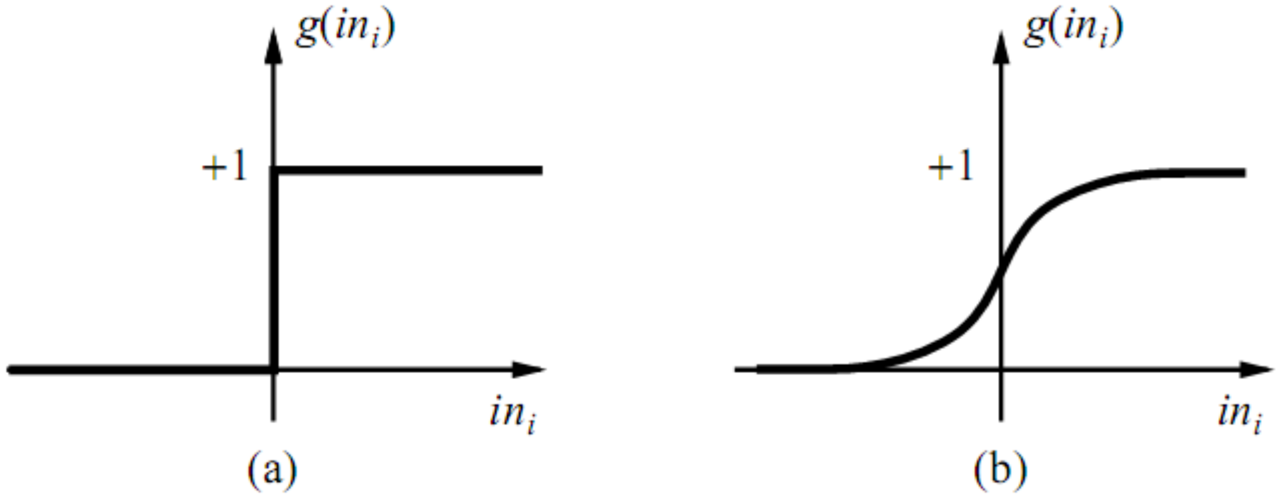
Mathematical Model of Artificial Neuron

Each node in ANN do the sum of products of its inputs(**X**) and their corresponding weights(**W**), adds a bias and apply an Activation function g(x) to get its output and feed it as an input to the next neuron



Activation function main purpose is to convert an input signal of a node to an output signal

Most popular types of Activation Functions



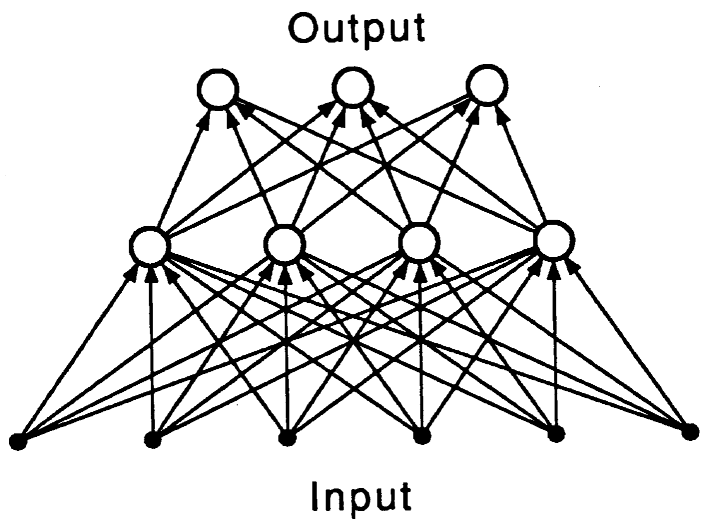
- (a) is a step function or threshold function, sign function
  (b) is a sigmoid function $1/(1+e^{-x})$

- Changing the bias weight $W_{0,i}$ moves the threshold location
- Different functions give different models
- Using a nonlinear function which approximates a linear threshold allows a network to approximate nonlinear functions

Topologies of Neural Networks

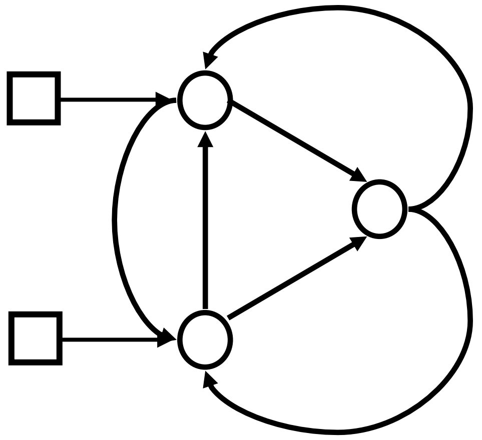
Feed-forward neural networks are the most common models

- These are directed acyclic graphs
  - Single-Layer Feed-Forward (Perceptron)
  - Multi-Layer Feed-Forward



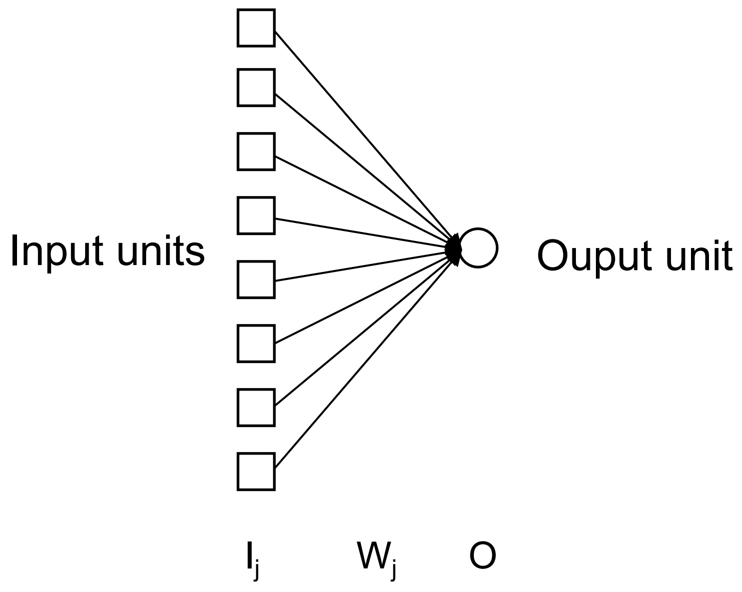
Recurrent networks have at least one feedback connection

- They have directed cycles
- The response to an input depends on the initial state which may depend on previous inputs

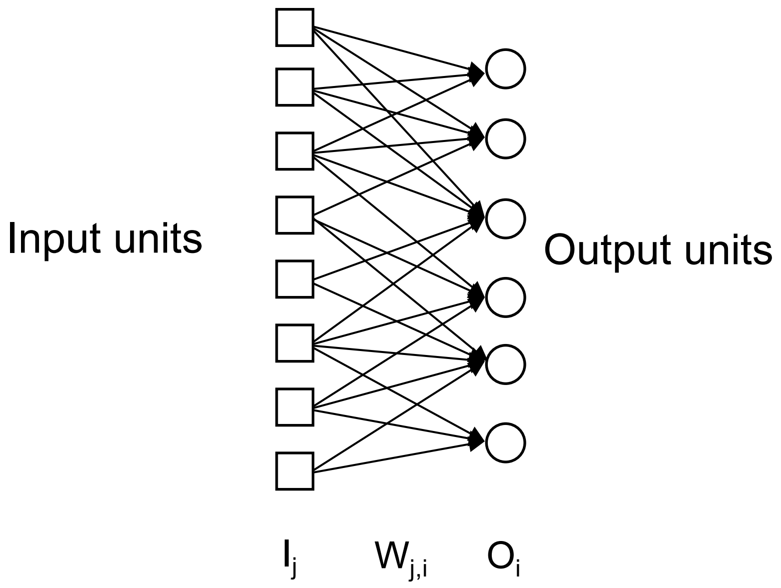


Perceptron

- Perceptron is a network of one input layer of neurons that feed forward to one output layer of neurons - there is no intermediate level, only the level of entry and exit



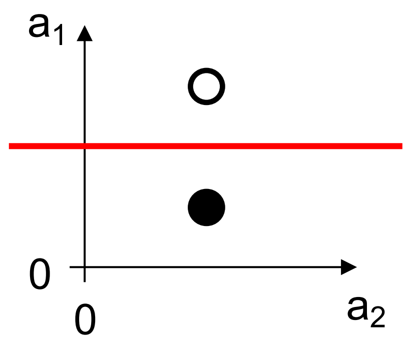
Single Perceptron



Perceptron Network

Expressiveness of Perceptron

- Perceptron can represent only linearly separable functions (i.e. functions for which such a separation hyperplane exists)

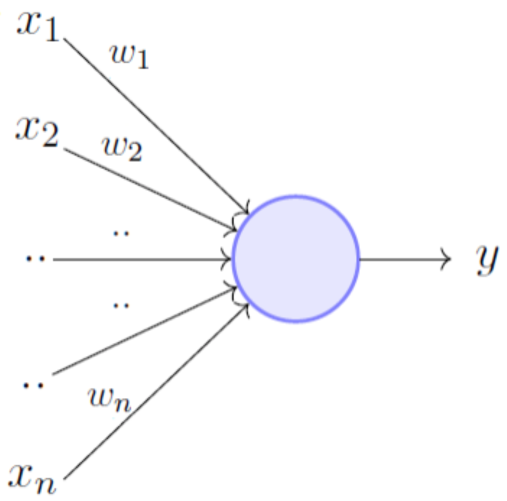


- Such perceptron have limited expressivity
- There exists an algorithm that can fit a threshold perceptron to any linearly separable training set

Perceptron Learning Algorithm

- Initialize the weights and threshold to small random numbers
- Present a vector **x** to the neuron inputs and calculate the output
- Update the weights according to the error

- Applied learning function: $W_j(t+1) = W_j(t) + \alpha \times (y - g_W(x)) \times x_j$



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Perceptron Learning

- The squared error for an example with input x and desired output y is:

$$E = \frac{1}{2} Err^2 = \frac{1}{2}(y - g_W(x))^2$$

- Perform optimization search by **gradient descent**:

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j}(y - g(\Sigma_{j=0}^{n} W_j x_j)) = -Err \times g'(in) \times x_j$$

- Simple weight update rule: $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$

- Positive error $\Rightarrow$ increase network output
  - increase weights on positive inputs,
  - decrease on negative inputs

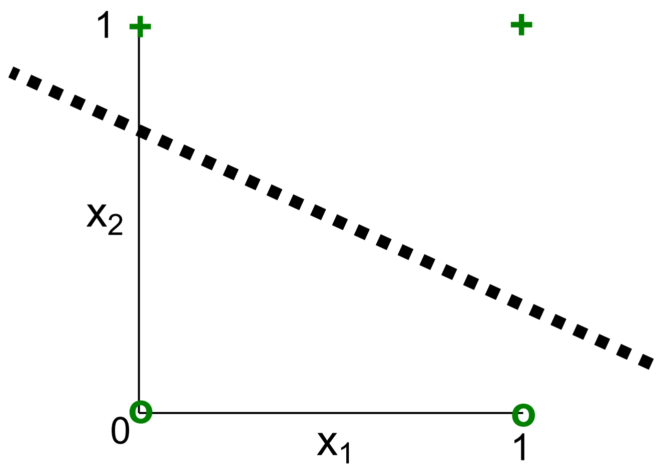
Perceptron Learning

- The weight updates need to be applied repeatedly for each weight W_j in the network, and for each training suite in the training set
- One such cycle through all weighty is called an **epoch** of training
- Eventually, mostly after **many epochs**, the weight changes converge towards zero and the training process terminates

- The **perceptron learning process always finds** a set of weights for a perceptron that solves a problem correctly with **a finite number of epochs, if such a set of weights exists**

Perceptron Learning Example

- Data: (0,0)→0, (1,0)→0, (0,1)→1, (1,1)→1



- Initialization:
  - $W_1(0) = 0.92$,
  - $W_2(0) = 0.62$,
  - $W_0(0) = 0.22$,
  - $\alpha = 0.1$

- Training – epoch 1:

out1 = sign(0.92*0 + 0.62*0 – 0.22) = sign(-0.22) = 0 ✓
out2 = sign(0.92*1 + 0.62*0 – 0.22) = sign(0.7) = 1 X

$W_1(1) = 0.92 + 0.1 * (0 – 1) * 1 = 0.82$
$W_2(1) = 0.62 + 0.1 * (0 – 1) * 0 = 0.62$
$W_0(1) = 0.22 + 0.1 * (0 – 1) * (-1) = 0.32$

out3 = sign(0.82*0 + 0.62*1 – 0.32) = sign(0.5) = 1 ✓
out4 = sign(0.82*1 + 0.62*1 – 0.32) = 1 ✓

Perceptron Learning Example

- Training – epoch 2:

  out1 = sign(0.82*0 + 0.62*0 – 0.32) = sign(0.32) = 0 ✓
  out2 = sign(0.82*1 + 0.62*0 – 0.32) = sign(0.5) = 1 X

  $W_1(2) = 0.82 + 0.1 * (0 - 1) * 1 = 0.72$
  $W_2(2) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
  $W_0(2) = 0.32 + 0.1 * (0 - 1) * (-1) = 0.42$

  out3 = sign(0.72*0 + 0.62*1 – 0.42) = sign(0.3) = 1 ✓
  out4 = sign(0.72*1 + 0.62*1 – 0.42) = 1 ✓

- Training – epoch 3:

  out1 = sign(0.72*0 + 0.62*0 – 0.42) = 0 ✓
  out2 = sign(0.72*1 + 0.62*0 – 0.42) = 1 X

  $W_1(3) = 0.72 + 0.1 * (0 - 1) * 1 = 0.62$
  $W_2(3) = 0.62 + 0.1 * (0 - 1) * 0 = 0.62$
  $W_0(3) = 0.42 + 0.1 * (0 - 1) * (-1) = 0.52$

  out3 = sign(0.62*0 + 0.62*1 – 0.52) = 1 ✓
  out4 = sign(0.62*1 + 0.62*1 – 0.52) = 1 ✓

Perceptron Learning Example

- Training – epoch 4:

  out1 = sign(0.62*0 + 0.62*0 – 0.52) = 0 ✓

  out2 = sign(0.62*1 + 0.62*0 – 0.52) = 1 X

  $W_1(4) = 0.62 + 0.1 * (0 – 1) * 1 = 0.52$
  $W_2(4) = 0.62 + 0.1 * (0 – 1) * 0 = 0.62$
  $W_0(4) = 0.52 + 0.1 * (0 – 1) * (-1) = 0.62$

  out3 = sign(0.52*0 + 0.62*1 – 0.62) = 0 X

  $W_1(4) = 0.52 + 0.1 * (1 – 0) * 0 = 0.52$
  $W_2(4) = 0.62 + 0.1 * (1 – 0) * 1 = 0.72$
  $W_0(4) = 0.62 + 0.1 * (1 – 0) * (-1) = 0.52$

  out4 = sign(0.52*1 + 0.72*1 – 0.52) = 1 ✓

  .....

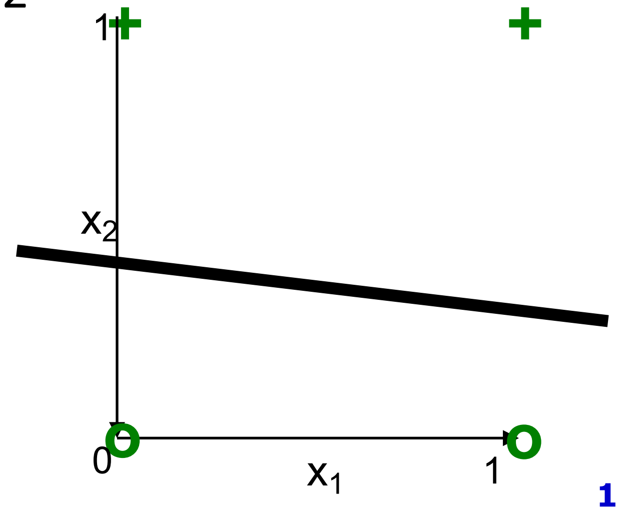
- Finally:

  out1 = sign(0.12*0 + 0.82*0 – 0.42) = 0 ✓

  out2 = sign(0.12*1 + 0.82*0 – 0.42) = 0 ✓

  out3 = sign(0.12*0 + 0.82*1 – 0.42) = 1 ✓

  out4 = sign(0.12*1 + 0.82*1 – 0.42) = 1 ✓



Example: Finding Weights for AND Operation

There are two input weights W_1 and W_2 and a threshold W_0. For each training pattern the perceptron needs to satisfy the following equation:

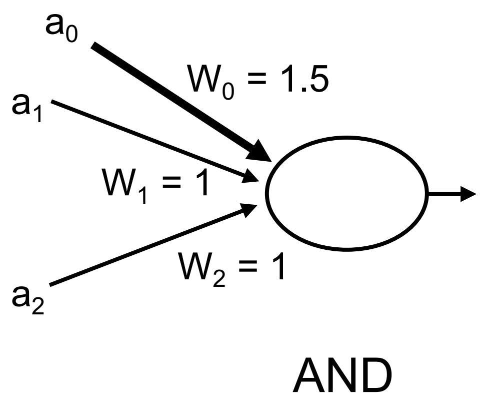
$$out = g(W_1 * a_1 + W_2 * a_2 - W_0) = sign(W_1 * a_1 + W_2 * a_2 - W_0)$$

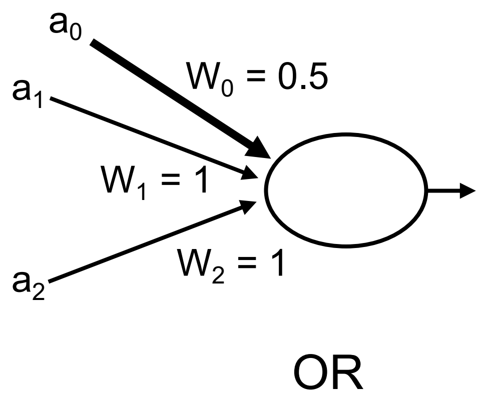
For a binary AND there are four training data items available that lead to four inequalities:

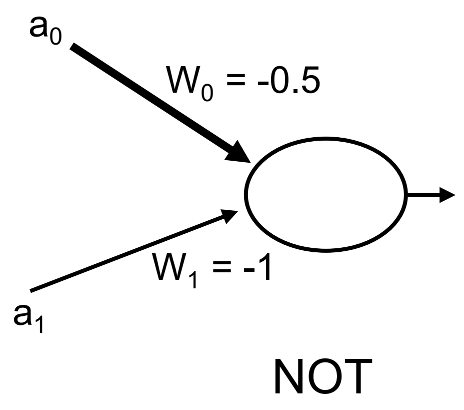
- $W_1 * 0 + W_2 * 0 - W_0 < 0 \quad \Rightarrow W_0 > 0$
- $W_1 * 0 + W_2 * 1 - W_0 < 0 \quad \Rightarrow W_2 < W_0$
- $W_1 * 1 + W_2 * 0 - W_0 < 0 \quad \Rightarrow W_1 < W_0$
- $W_1 * 1 + W_2 * 1 - W_0 \geq 0 \quad \Rightarrow W_1 + W_2 \geq W_0$

There is obvious an **infinite number of solutions** that realize a **logical AND**; e.g. $W_1 = 1$, $W_2 = 1$ and $W_0 = 1.5$

Logical Functions







- These Boolean functions that can be implemented with an artificial neuron

Limitations of Simple Perceptrons

XOR

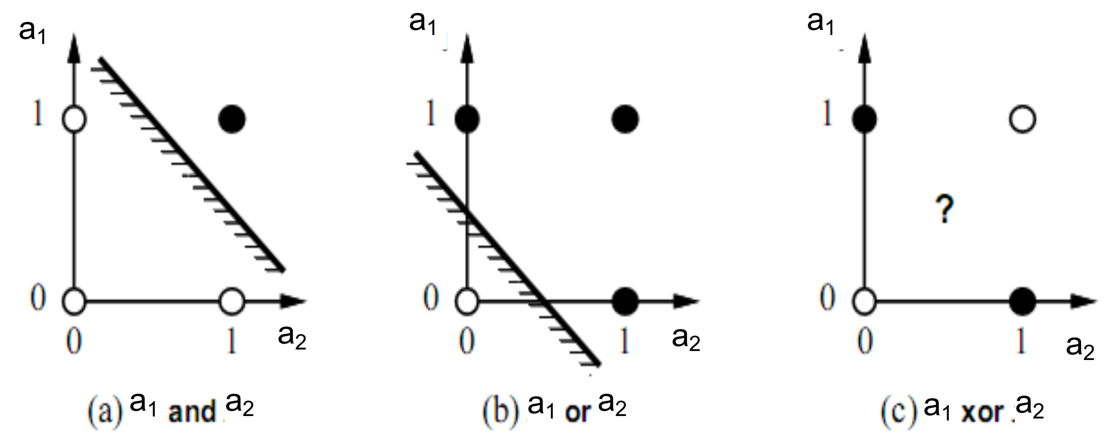
- $W_1*0 + W_2*0 - W_0 < 0$ $\Rightarrow W_0 > 0$
- $W_1*0 + W_2*1 - W_0 \geq 0$ $\Rightarrow W_2 \geq W_0$
- $W_1*1 + W_2*0 - W_0 \geq 0$ $\Rightarrow W_1 \geq W_0$
- $W_1*1 + W_2*1 - W_0 < 0$ $\Rightarrow W_1 + W_2 < W_0$

- The 2nd and 3rd inequalities are not compatible with inequality 4, and there is no solution to the XOR problem
- XOR requires two separation hyperplanes!
- There is thus a need for more complex networks that combine simple perceptrons to address more sophisticated classification tasks

Expressiveness of Perceptrons

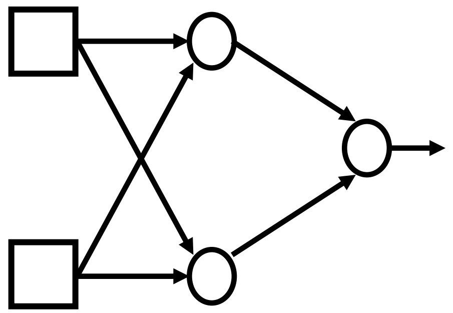
- A perceptron with g = step function can model Boolean functions and linear classification:
  - a perceptron can represent AND, OR, NOT, but **not XOR**
- A perceptron represents a linear separator for the input space

$\sum_j W_j a_j > 0$



Multi-Layer Feed-Forward

- Multi-Layer Feed-Forward Structures have:
  - one input layer
  - one output layer
  - one or many hidden layers of processing units

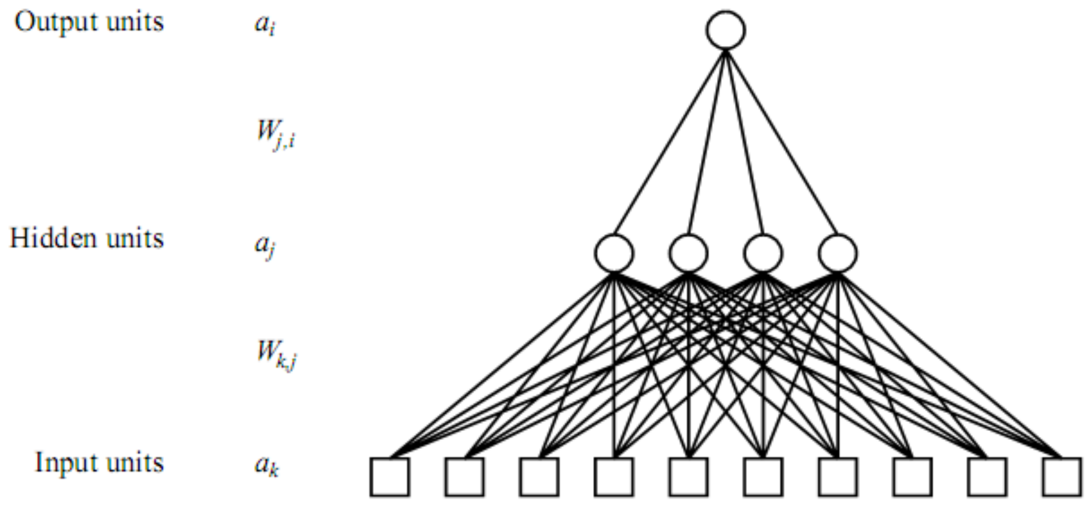


- The hidden layers are between the input and the output layer

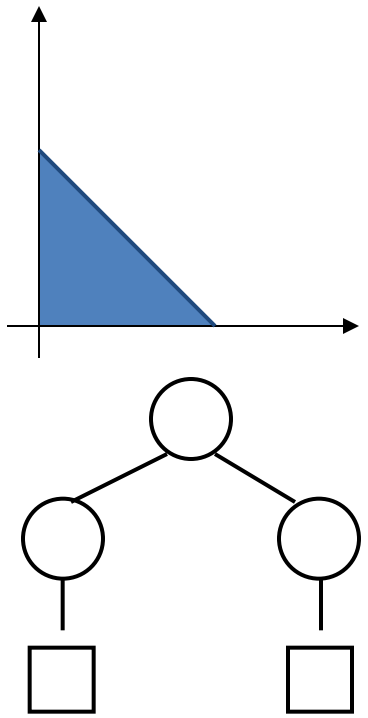
Multi-Layer Feed-Forward

- Multi-Layer Perceptrons (MLP) have fully connected layers
- Hidden layers enlarge the space of hypotheses that the network can represent
- Learning is done by **back-propagation algorithm**

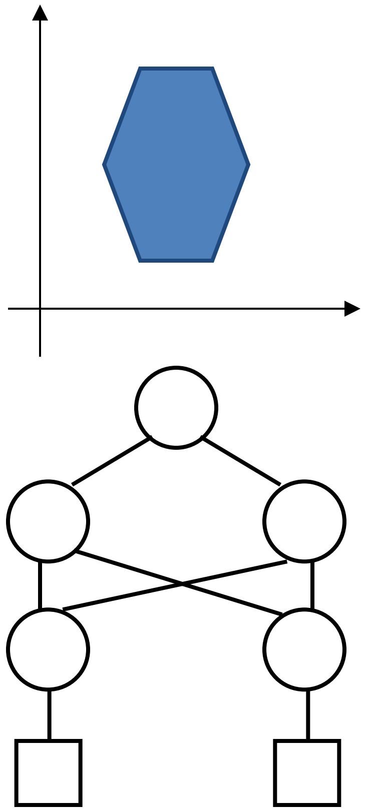
  ➥ errors are back-propagated from the output layer to the hidden layers



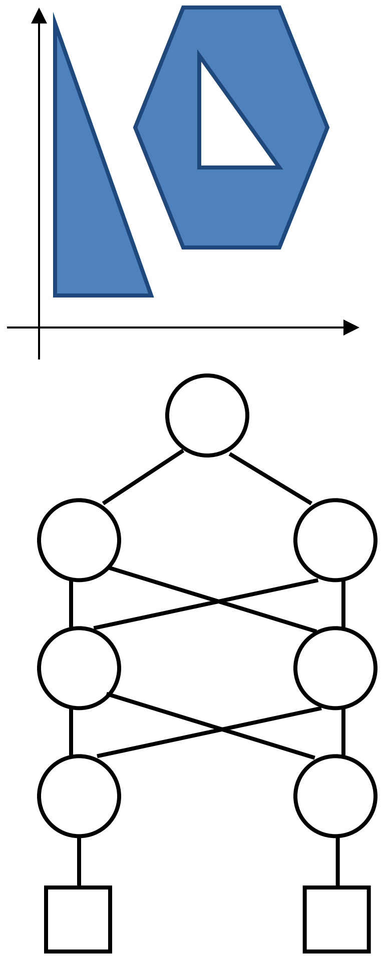
Number of Hidden Layers vs. Expressiveness



One layer draws linear boundaries



Two layer combines the boundaries.



Three or more layers can generate arbitrarily boundaries

Neural Networks

Advantages

- Accuracy of classification is usually high, even for complex problems
- Distributed processing, the knowledge is distributed through the weights of the links
- Robust in handling examples even if they contain errors
- They handle well redundant attributes, since the weights associated with them are usually very small
- Results can be discrete, real values, or a vector of values (discrete or real)

Disadvantages

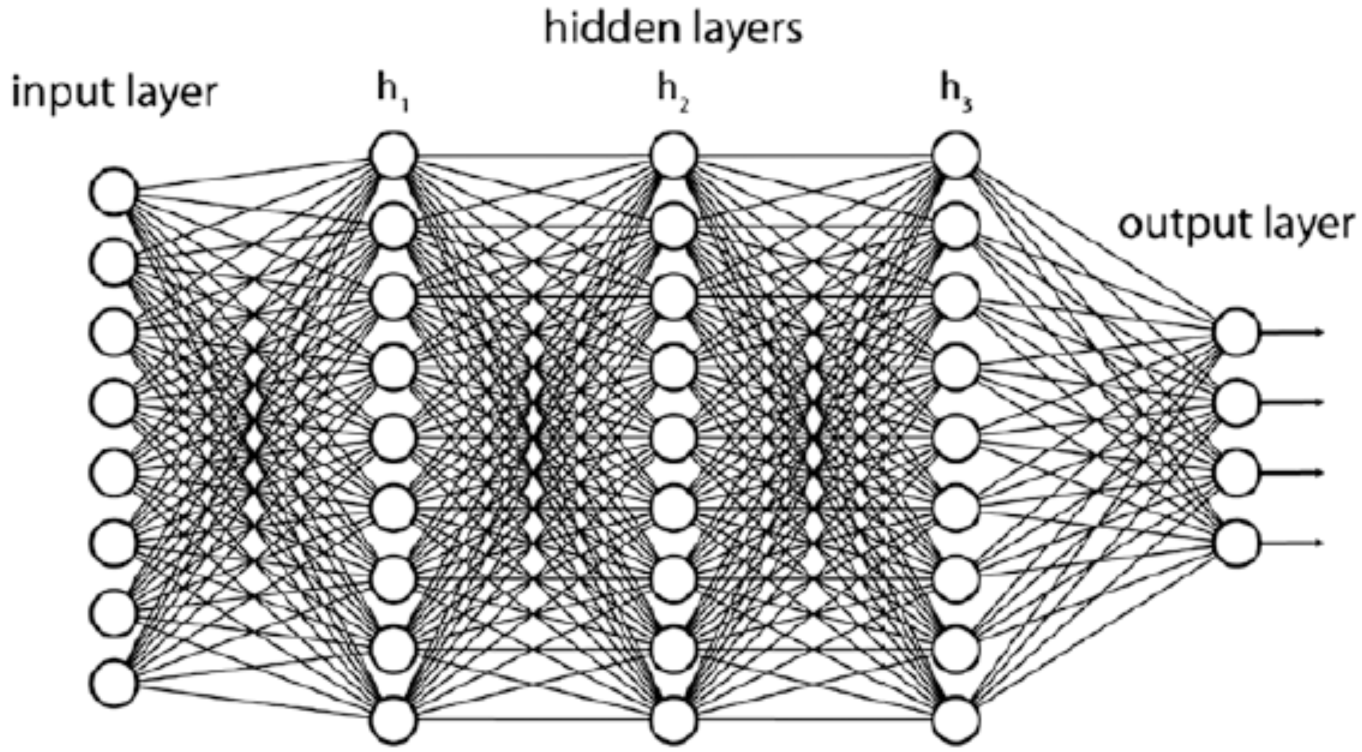
- Difficult to determine optimal network topology for a problem
- Difficult to use - have many parameters to define, require long training time
- Requires specific pre-processing of data
- Difficult to understand the learning function (weights)
- Do not provide a model or explanations of results
- It is not easy to incorporate domain knowledge

Deep Learning

Deep Learning

Deep neural networks are distinguished from ANNs by having many hidden layers

Deep means **many layers**



Deep Learning – APIs

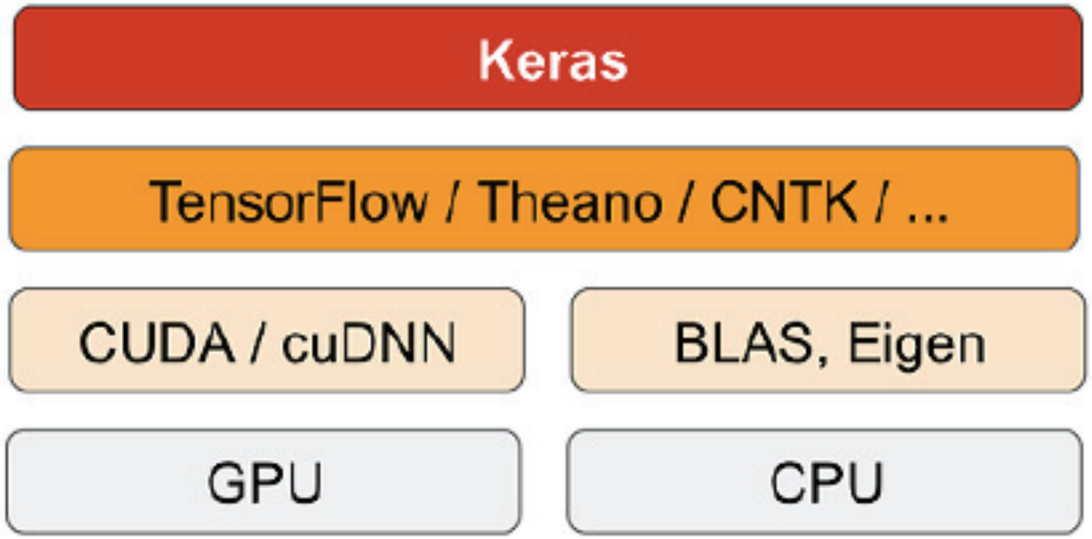
Keras is a high-level **deep learning API** written in Python that can run on top of any of these **three deep learning frameworks**:

- TensorFlow (from Google)
- CNTK (from Microsoft)
- Theano (from the Montreal Institute for Learning Algorithms, Université Montréal, Canada)

Keras facilitates the following key aspects:

- Built-in CNN, RNN, and autoencoder models as well as support classes and methods (metrics, optimizers, regularizers, visualization, and so on) - enables **easy and fast prototyping**
- Excellent modularity and extensibility
- Allow the same code to run seamlessly on CPU and GPU

Keras model-level library



- Any code written with Keras can be run with any of these backends without having to change anything in the code
- it is possible to switch between any backend engine during development — if one of these backends proves to be faster for a specific task
- It is **recommended to use the TensorFlow** backend as the default, because it's the most widely adopted, most scalable, and most, production ready

Data pre-processing

VECTORIZATION

All inputs and targets in a deep network must be **tensors of floating-point data** (or, in specific cases, tensors of integers)

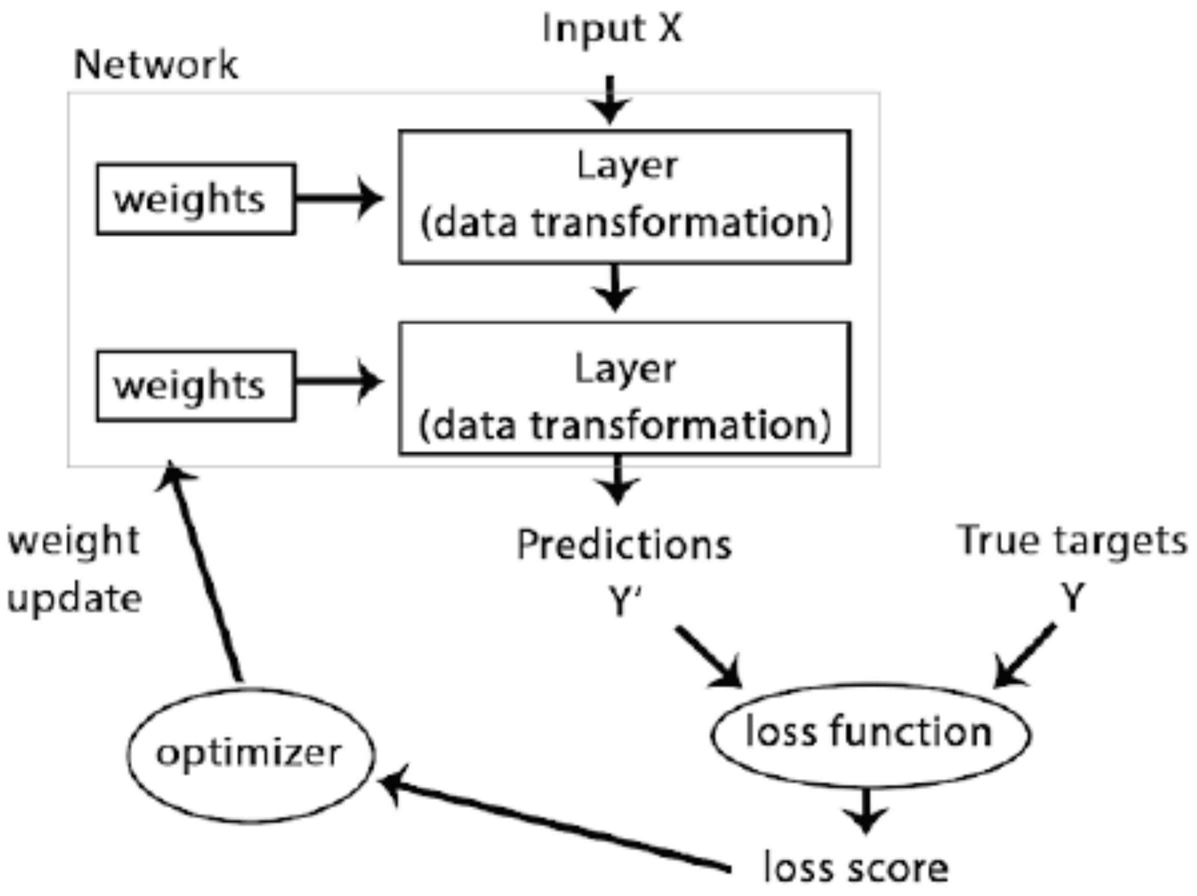
All data need to process: sound, images, text must first turn into tensors — a step called **data vectorization**

SCALING

- It isn't safe to feed into a neural network data that takes relatively large values
- Or data that is heterogeneous, where the ranges vary greatly — Doing so can trigger large gradient updates that will prevent the network from converging

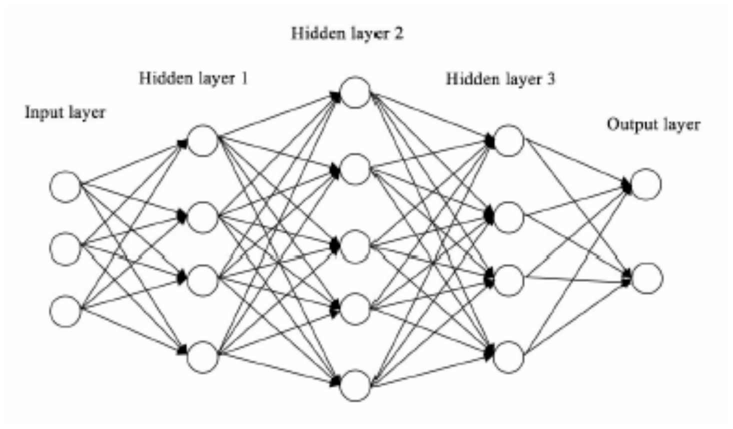
  The data should have the following characteristics:
  - **Take small values**—Typically, most values should be in the 0–1 range
  - **Be homogenous**—, all features should take values in roughly the same range

Training a Neural Network

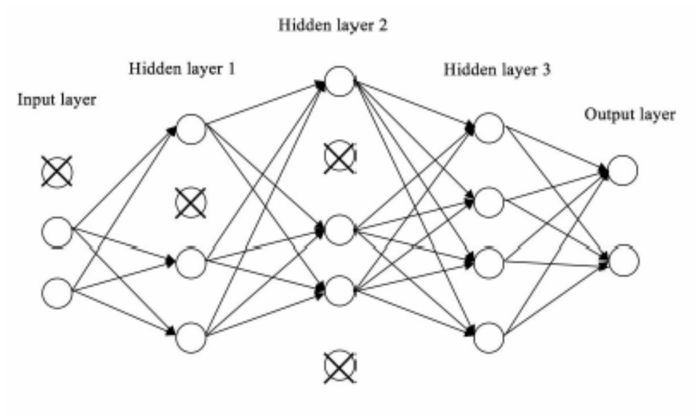


Prevent overfitting: Dropout

Dropout consists of **randomly** dropping out (**setting to zero**) a number of output features of the internal layers **during training**



Standard NN



NN with dropout

- **Dropout rate:** is the fraction of the features that are zeroed out; it's usually set between 0.2 and 0.5
- At **test time**, no units are dropped out

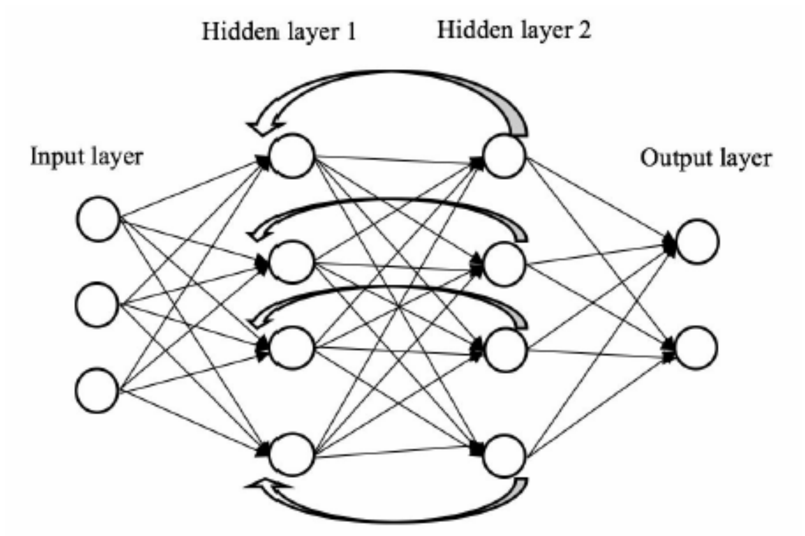
Keras workflow

1. Define the training data: input tensors and target tensors
2. Define a network of layers (or model) that maps the inputs to the targets
3. Configure the learning process by choosing:
   - a loss function,
   - an optimizer,
   - some metrics to monitor
4. Iterate on the training data by calling the fit() method with the model developed

Deep Learning Types

Recurrent Neural Networks (RNNs)

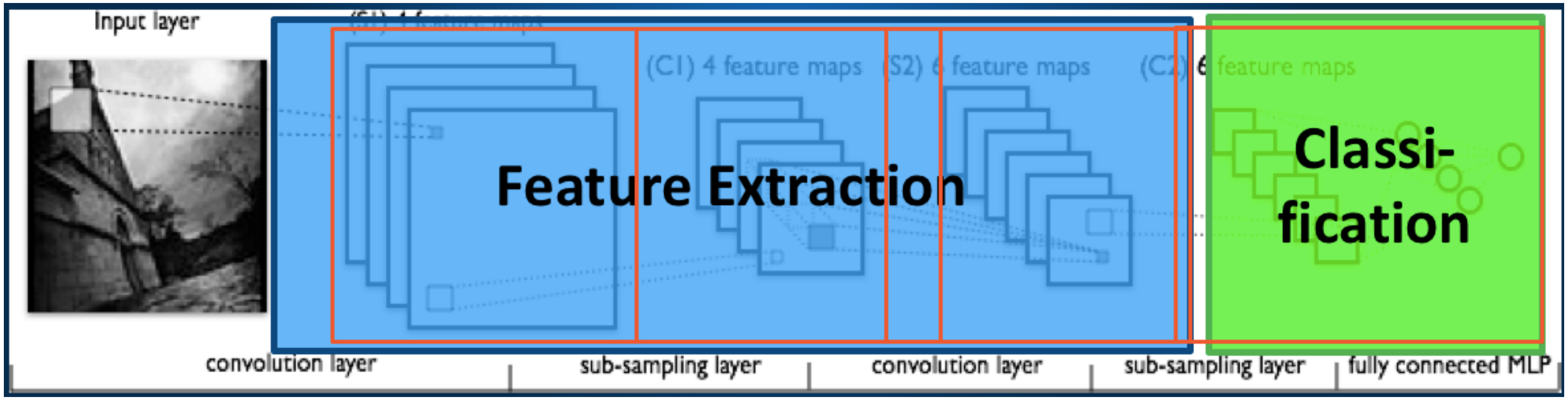
- processes information incrementally while maintaining an internal model of what it's processing, built from past information and constantly updated as new information comes in — **keeps memories of what came before**
- are designed to work with sequence prediction problems and sequence data in general such as, **speech recognition, natural language processing, timeseries.**



Deep Learning Types

Convolutional Neural Networks (CNNs / ConvNets)

- Is a type of artificial neural network used in **image processing and computer vision** that is specifically designed to process **pixel data**


Input layer
(C1) 4 feature maps
(S2) 6 feature maps
(C2) 6 feature maps
Feature Extraction
Classi-
fication
convolution layer
sub-sampling layer
convolution layer
sub-sampling layer
fully connected MLP