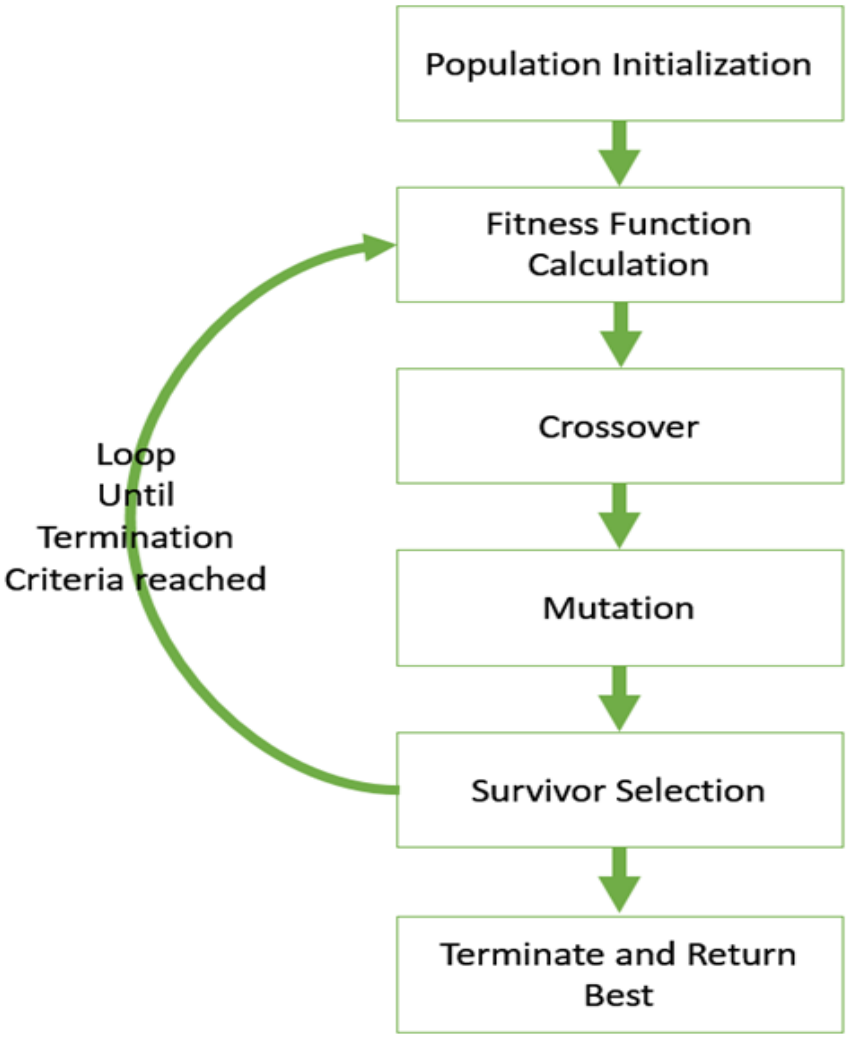
UNIT-5: Machine Learning

Genetic Algorithm in Machine Learning

Genetic algorithms are iterative optimization techniques that simulate natural selection to find optimal solutions. A genetic algorithm is a search algorithm that is inspired by "Darwin's idea of evolution in Nature." It is used in machine learning to tackle optimization problems. It is an important algorithm since it aids in the solution of difficult problems that would otherwise take a long time to solve.

The basic structure of a GA is as follows –

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent.



A generalized pseudo-code for a GA is

```
GA()
  initialize population
  find fitness of population

  while (termination criteria is reached) do
    parent selection
    crossover with probability pc
    mutation with probability pm
    decode and fitness calculation
    survivor selection
    find best
  return best
```

How genetic algorithm works?

In order to understand the simple genetic algorithm and how it works, there are some basic terminologies that will help you understand it better. We have described them below.

- **Genetic operators**: In genetic algorithms, genetic operators are used when you wish to change the genetic composition of the next generation.
- **Chromosome/Individual**: It refers to the collection of genes that can be represented with a string of each bit as a gene.
- **Population:** Each chromosome represents an individual and a collection of chromosomes/individual make up the population.
- **Fitness function:** This function in genetic algorithms produces an improved output for a specific input.

Now let’s move on to understand the working of genetic algorithms in machine learning.

There are five phases that illustrate the entire process of how this algorithm works.

1. Initialization

The working of genetic algorithms starts with the process of initialization where a set of individuals is generated that we refer to as population. It contains a set of parameters called genes that are combined into a string and generate chromosomes. These chromosomes are the solution to the problem that are derived by the technique of random binary strings.

2. Fitness assignment

The fitness function of the genetic algorithm determines an individual's ability to compete with other individuals. It provides the score to each individual that determines its probability of being selected for the process of reproduction.

Higher the fitness score, greater are the chances of an individual getting selected for reproduction.

3. Selection

In this phase, out of all the steps of genetic algorithm, individuals are selected to produce offspring by arranging them in a pair of two. Three types of selection methods are leveraged in this process that are listed below.

- Rank based selection
- Tournament selection
- Roulette wheel selection

4. Reproduction

In the reproduction step, the genetic algorithm leverages two variation operators. These are applied to the parent population. These two operators are:

- Crossover

In this process, a crossover point is selected within the genes randomly. This operator then swaps the genetic information of the two selected parents or say individuals from the current generation to produce an offspring.

There are various styles for crossover among the parents namely one-point crossover, two-point crossover, livery crossover, and inheritable algorithms crossover.

The genes of the chosen fittest parents are exchanged until the crossover point is met. When the process ends, an offspring is produced that includes genes from both the parents.

- Mutation

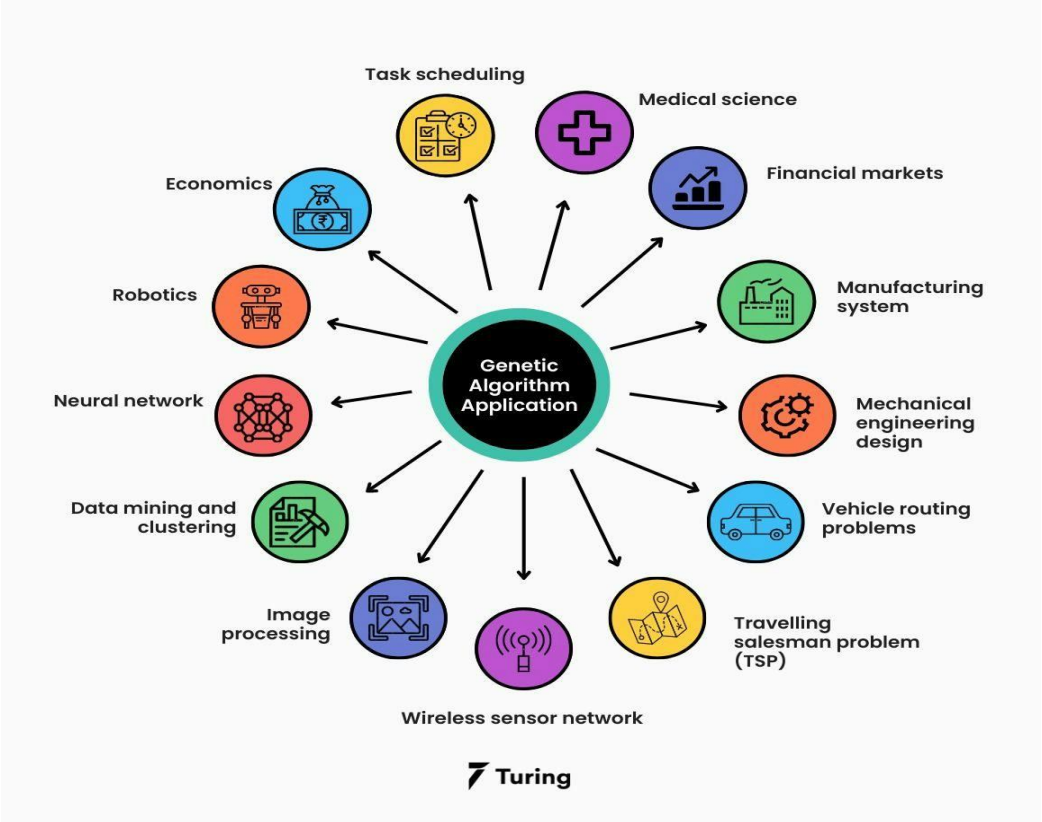
In this process, a random gene is inserted in the offspring in order to maintain the diversity in the population. There are three types of mutation styles available namely flip bit mutation, gaussian mutation, and exchange or swap mutation.

This operator helps in resolving the premature convergence issue and enhances diversification in the population.

5. Termination

Now that the offspring is produced, the reproduction face is terminated by applying a stopping criterion. The algorithm reaches the end of the process after the threshold fitness solution is reached. Further, it identifies the final yet best solution in the population.

Applications of genetic algorithm in machine learning:



Statistical-based Learning: Naïve Bayes Model

Naive Bayes algorithm is a supervised machine learning algorithm which is based on Bayes Theorem used mainly for classification problem.

Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick prediction.

It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. Some popular examples of Naive Bayes Algorithm are spam filtration, Sentimental analysis, and classifying articles.

Bayes Theorem:

Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

$$P(A|B) = (P(B|A) * P(A))/ P(B) == P(A \cap B) / P(B)$$

|||ly $P(B|A) = (P(A|B) * P(B))/ P(A) == P(B \cap A) / P(A)$

if $P(A \cap B) == P(B \cap A)$.

then $P(B|A) * P(A) = P(A|B) * P(B)$

Types Of Naive Bayes:

There are three types of Naive Bayes model under the scikit-learn library:

Gaussian: It is used in classification and it assumes that features follow a normal distribution.

Multinomial: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".

Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

Numerical Example 1:

Consider the given Dataset ,Apply Naive Baye's Algorithm and Predict that if a fruit has the following properties then which type of the fruit it is
Fruit = {Yellow , Sweet ,long}

Frequency Table:

Fruit	Yellow	Sweet	Long	Total
Mango	350	450	0	650
Banana	400	300	350	400
Others	50	100	50	150
Total	800	850	400	1200

Solution:

P(A|B) = (P(B|A) * P(A))/ P(B)

1. Mango:

P(X | Mango) = P(Ye | Yellow) * P(Sweet | Mango) * P(Long | Mango)

1. a)P(Yellow | Mango) = (P(Mango | Yellow) * P(Yellow))/ P (Mango)

= ((350/800) * (800/1200)) / (650/1200)

P(Yellow | Mango)= 0.53 →1

1.b)P(Sweet | Mango) = (P(Sweet | Mango) * P(Sweet))/ P (Mango)

= ((450/850) * (850/1200)) / (650/1200)

P(Sweet | Mango)= 0.69 → 2

1. c)P(Long | Mango) = (P(Long | Mango) * P(Long))/ P (Mango)

= ((0/650) * (400/1200)) / (800/1200)

P(Long | Mango)= 0 → 3

On multiplying eq 1,2,3 ==> P(X | Mango) = 0.53 * 0.69 * 0

P(X | Mango) = 0

2. Banana:

P(X | Banana) = P(Yellow | Banana) * P(Sweet | Banana) * P(Long | Banana)

2.a) P(Yellow | Banana) = (P(Banana | Yellow) * P(Yellow))/ P (Banana)

= ((400/800) * (800/1200)) / (400/1200)

P(Yellow | Banana) = 1 → 4

2.b) P(Sweet | Banana) = (P(Banana | Sweet) * P(Sweet))/ P (Banana)

= ((300/850) * (850/1200)) / (400/1200)

P(Sweet | Banana) = .75→ 5

2.c)P(Long | Banana) = (P(Banana | Yellow) * P(Long))/ P (Banana)

= ((350/400) * (400/1200)) / (400/1200)

P(Yellow | Banana) = 0.875 → 6

On multiplying eq 4,5,6 ==> P(X | Banana) = 1 * .75 * 0.875

P(X | Banana) = 0.6562

3. Others:

P(X | Others) = P(Yellow | Others) * P(Sweet | Others) * P(Long | Others)

3.a) P(Yellow | Others) = (P(Others| Yellow) * P(Yellow))/ P (Others)

= ((50/800) * (800/1200)) / (150/1200)

P(Yellow | Others) = 0.34→ 7

3.b) P(Sweet | Others) = (P(Others| Sweet) * P(Sweet))/ P (Others)

= ((100/850) * (850/1200)) / (150/1200)

P(Sweet | Others) = 0.67 → 8

3.c) P(Long | Others) = (P(Others| Long) * P(Long))/ P (Others)

= ((50/400) * (400/1200)) / (150/1200)

P(Long | Others) = 0.34 → 9

On multiplying eq 7,8,9 ==> P(X | Others) = 0.34 * 0.67* 0.34

P(X | Others) = 0.07742

So finally from P(X | Mango) == 0 , P(X | Banana) == 0.65 and P(X| Others) == 0.07742.

We can conclude **Fruit{Yellow,Sweet,Long} is Banana.**

Numerical Example 2: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No

12	Overcast	Yes
13	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Working of Naïve Bayes' Classifier:

Suppose we have a dataset of weather conditions and corresponding target variable "Play". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Neural Networks:

A neuron is a cell in brain whose principle function is the collection, Processing, and dissemination of electrical signals. Brains Information processing capacity comes from networks of such neurons. Due to this reason some earliest AI work aimed to create such artificial networks. (Other Names are Connectionism; Parallel distributed processing and neural computing).

What is a Neural Network?

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Neural networks versus conventional computers

Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurones) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Units of Neural Network:

Nodes(units):

Nodes represent a cell of neural network.

Links:

Links are directed arrows that show propagation of information from one node to another node.

Activation:

Activations are inputs to or outputs from a unit.

Weight:

Each link has weight associated with it which determines strength and sign of the connection.

Activation function:

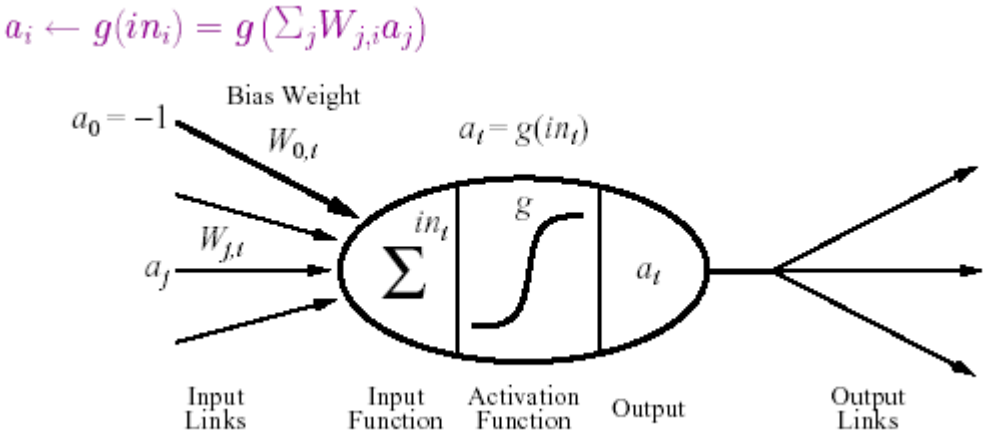
A function which is used to derive output activation from the input activations to a given node is called activation function.

Bias Weight:

Bias weight is used to set the threshold for a unit. Unit is activated when the weighted sum of real inputs exceeds the bias weight.

Simple Model of Neural Network

A simple mathematical model of neuron is devised by McCulloch and Pit is given in the figure given below:



It fires when a linear combination of its inputs exceeds some threshold.

A neural network is composed of nodes (units) connected by directed links A link from unit j to i serve to propagate the activation a_j from j to i. Each link has some numeric weight $W_{j,i}$ associated with it, which determines strength and sign of connection.

Each unit first computes a weighted sum of it's inputs:

$$in_i = \sum_{J=0}^{n} W_{j,i}\ a_j$$

Then it applies activation function g to this sum to derive the output:

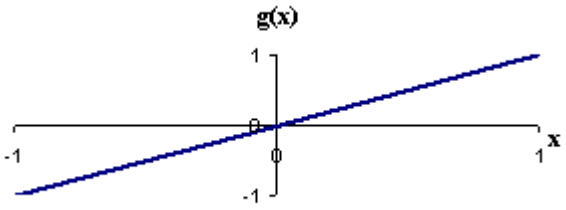
$$a_i = g(in_i) = g\left(\sum_{J=0}^{n} W_{j,i}\ a_j\right)$$

Here, a_j output activation from unit j and $W_{j,i}$ is the weight on the link j to this node. Activation function typically falls into one of three categories:

- Linear
- Threshold (*Heaviside function)*
- Sigmoid
- Sign

For **linear activation functions**, the output activity is proportional to the total weighted output.

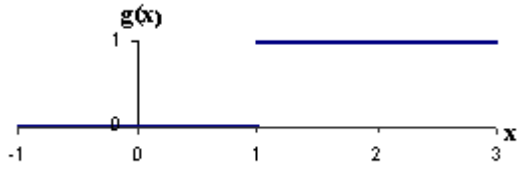
g(x) = k x + c, where k and x are constant



For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

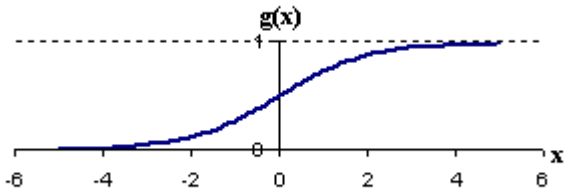
g(x) = 1 if x>= k

= 0 if x < k

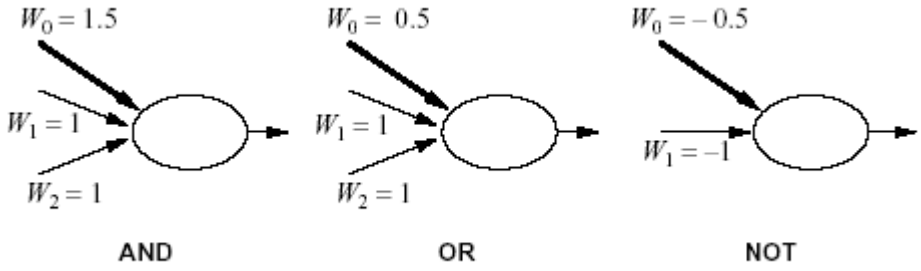


For **sigmoid activation functions**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units. It has the advantage of differentiable.

$g(x) = 1/ (1 + e^{-x})$



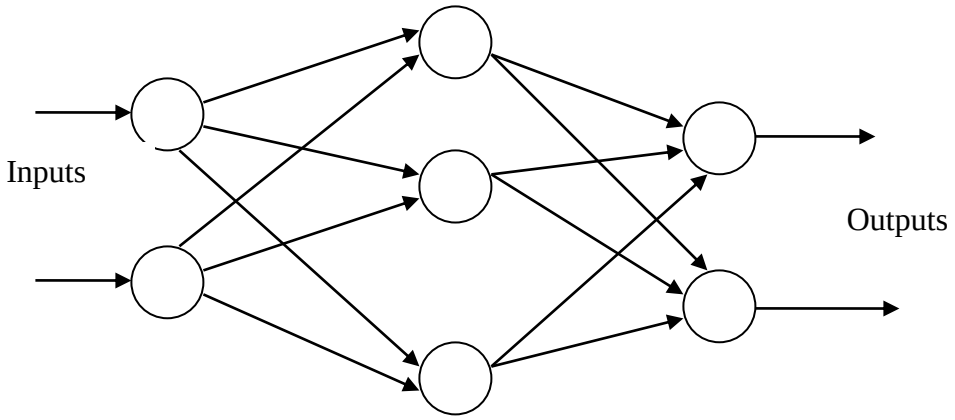
Realizing logic gates by using Neurons:



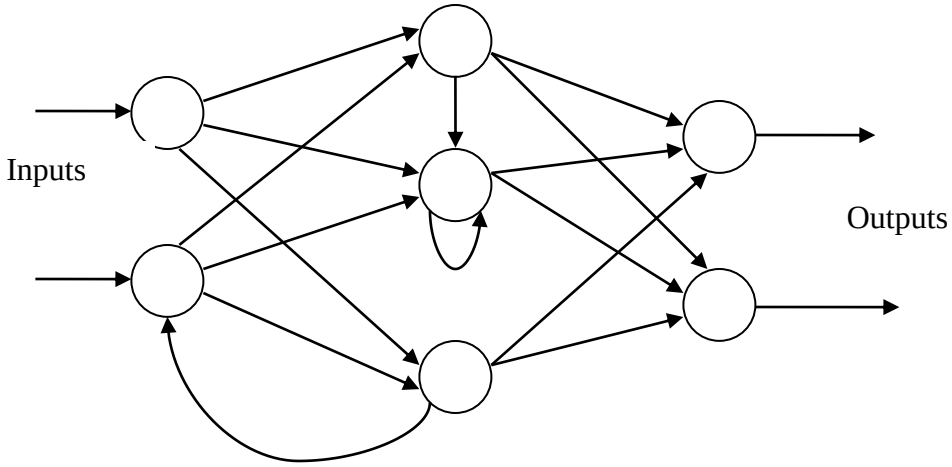
Network structures:

Feed-forward networks:

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

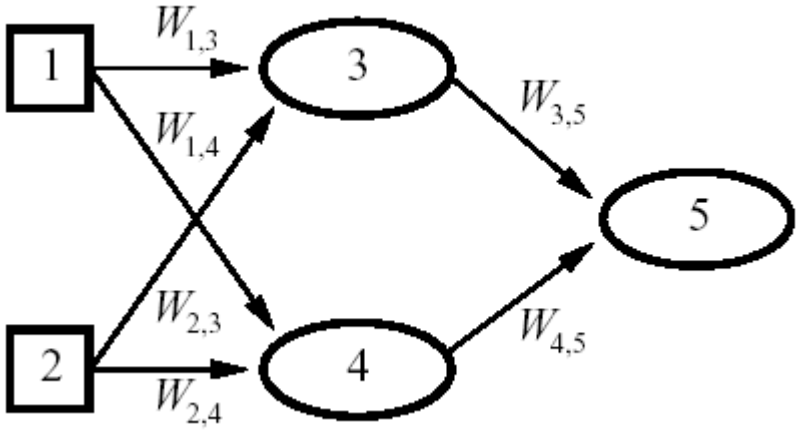


Feedback networks (Recurrent networks:)



Feedback networks (figure 1) can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent.

Feed-forward example



Here;

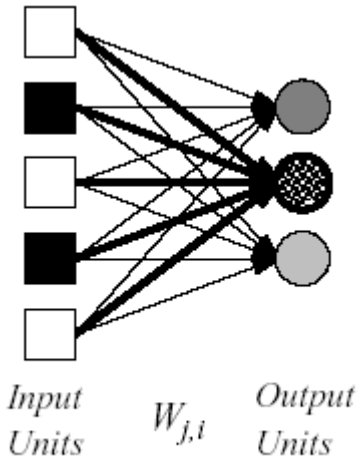
$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

$= g(W_{3;5}\ g(W_{1;3}\ a_1 + W_{2;3}\ a_2) + W4;5\ g(W_{1;4}\ a_1 + W_{2;4}\ a_2)$

Types of Feed Forward Neural Network:

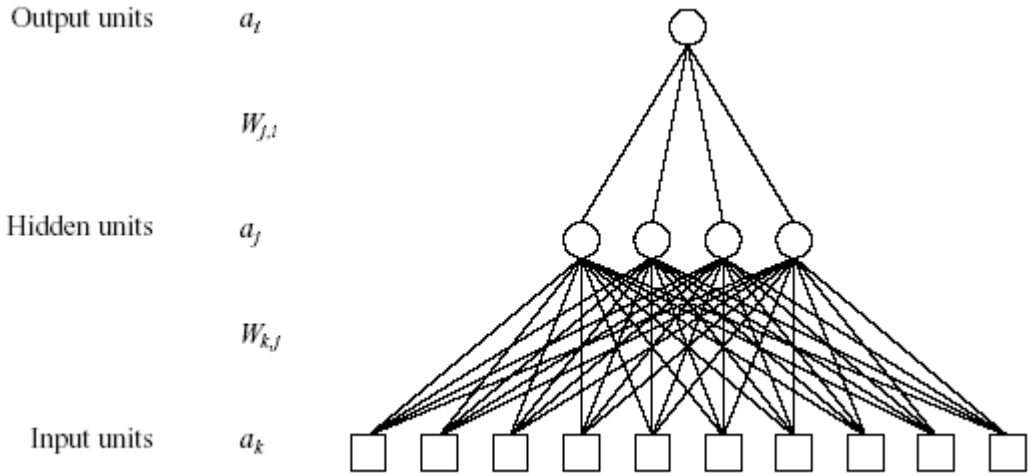
Single-layer neural networks (perceptrons)

A neural network in which all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. Since each output unit is independent of the others each weight affects only one of the outputs.



Multilayer neural networks (perceptrons)

The neural network which contains input layers, output layers and some hidden layers also is called multilayer neural network. The advantage of adding hidden layers is that it enlarges the space of hypothesis. Layers of the network are normally fully connected.



Once the number of layers, and number of units in each layer, has been selected, training is used to set the network's weights and thresholds so as to minimize the prediction error made by the network

Training is the process of adjusting weights and threshold to produce the desired result for different set of data.

Learning in Neural Networks:

Learning: One of the powerful features of neural networks is learning. **Learning in neural networks is carried out by adjusting the connection weights among neurons**. It is similar to a biological nervous system in which learning is carried out by changing synapses connection strengths, among cells.

The operation of a neural network is determined by the values of the interconnection weights. There is no algorithm that determines how the weights should be assigned in order to solve specific problems. Hence, the weights are determined by a learning process

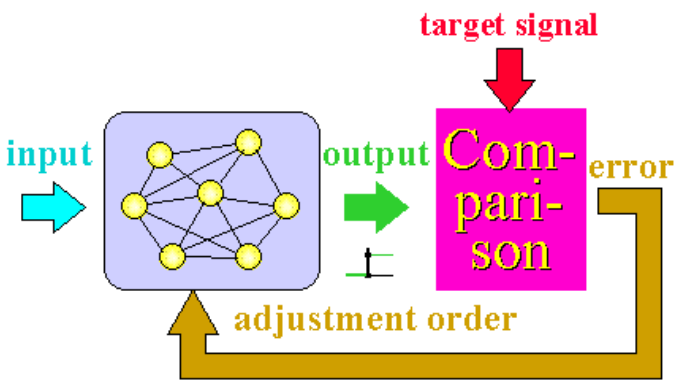
Learning may be classified into two categories:

(1) Supervised Learning
(2) Unsupervised Learning

Supervised Learning:

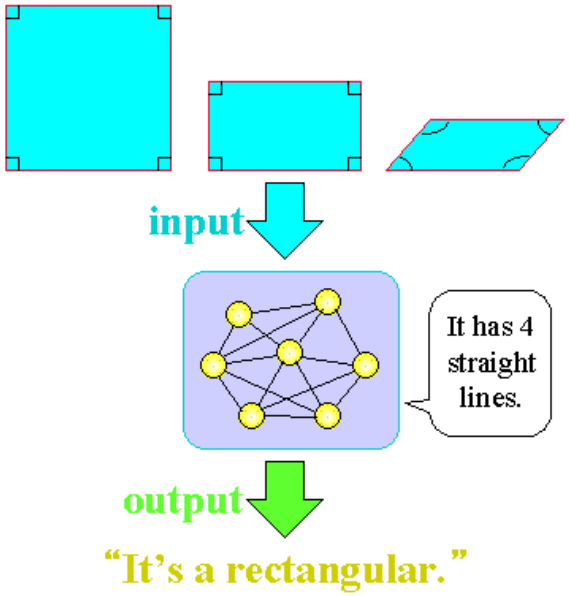
In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs. Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights. The most common supervised learning method is the "error correction method".

Error correction method is used for networks which their neurons have discrete output functions. Neural networks are trained with this method in order to reduce the error (difference between the network's output and the desired output) to zero.



Unsupervised Learning:

In unsupervised learning, there is no teacher (target signal) from outside and the network adjusts its weights in response to only the input patterns. A typical example of unsupervised learning is **Hebbian learning.**



Consider a machine (or living organism) which receives some sequence of inputs x1, x2, x3, . . ., where xt is the sensory input at time t. In supervised learning the machine is given a sequence of input & a sequence of desired outputs y1, y2, . . . , and the goal of the machine is to learn to produce the correct output given a new input. While, in unsupervised learning the machine simply receives inputs x1, x2, . . ., but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

Hebbian Learning:

The oldest and most famous of all learning rules is Hebb's postulate of learning:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased"

From the point of view of artificial neurons and artificial neural networks, Hebb's principle can be described as a method of determining how to alter the weights between model neurons. **The weight between two neurons increases if the two neurons activate simultaneously—and reduces if they activate separately.** Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights.

Hebb's Algorithm:

Step 0: initialize all weights to 0

Step 1: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$

Step 2: Set the activation of the output unit to the target value: $y = t$

Step 3: Adjust the weights: $w_i(new) = w_i(old) + x_i y$

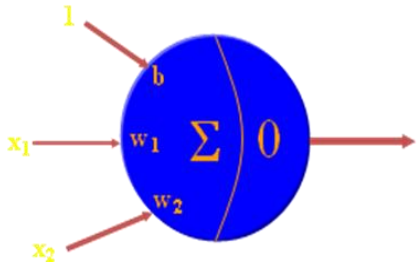
Step 4: Adjust the bias (just like the weights): $b(new) = b(old) + y$

Example:

PROBLEM: Construct a Hebb Net which performs like an AND function, that is, only when both features are "active" will the data be in the target class.

TRAINING SET (with the bias input always at 1):

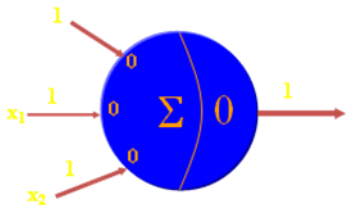
x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Training-First Input:

- Initialize the weights to 0

Present the first input:
(1 1 1) with a target of 1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

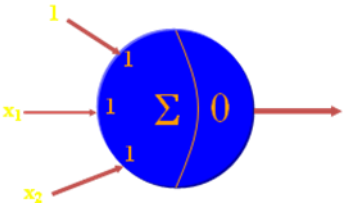
$= 0 + 1 = 1$

$w_2(new) = w_2(old) + x_2 t$

$= 0 + 1 = 1$

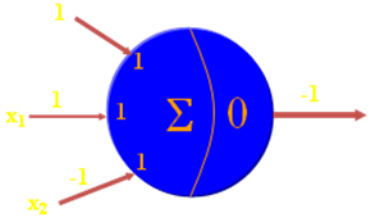
$b(new) = b(old) + t$

$= 0 + 1 = 1$



Training- Second Input:

- **Present the second input:**
  (1 -1 1) with a target of -1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

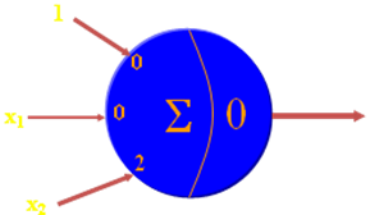
$= 1 + 1(-1) = 0$

$w_2(new) = w_2(old) + x_2 t$

$= 1 + (-1)(-1) = 2$

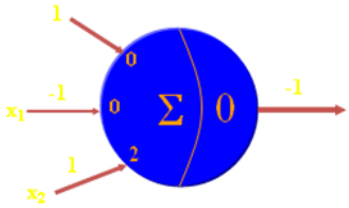
$b(new) = b(old) + t$

$= 1 + (-1) = 0$



Training- Third Input:

- **Present the third input: (-1 1 1) with a target of -1**



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

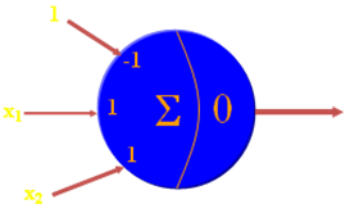
$= 0 + (-1)(-1) = 1$

$w_2(new) = w_2(old) + x_2 t$

$= 2 + 1(-1) = 1$

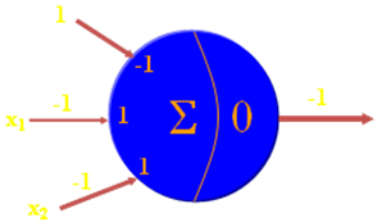
$b(new) = b(old) + t$

$= 0 + (-1) = -1$



Training- Fourth Input:

- **Present the fourth input: (-1 -1 1) with a target of -1**



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

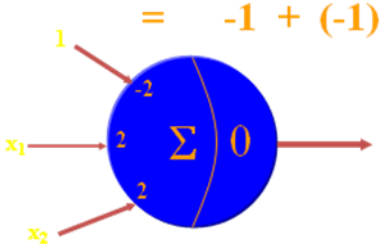
$= 1 + (-1)(-1) = 2$

$w_2(new) = w_2(old) + x_2 t$

$= 1 + (-1)(-1) = 1$

$b(new) = b(old) + t$

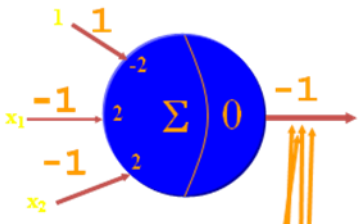
$= -1 + (-1) = -2$



Final Neuron:

- **This neuron works:**

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



$1*2 + 1*2 + 1*(-2) = 2 > 0$

$(-1)*2 + 1*2 + 1*(-2) = -2 < 0$

$1*2 + (-1)*2 + 1*(-2) = -2 < 0$

$(-1)*2 + (-1)*2 + 1*(-2) = -6 < 0$

Perceptron Learning Theory:

The term "Perceptrons" was coined by Frank RosenBlatt in 1962 and is used to describe the connection of simple neurons into networks. These networks are simplified versions of the real nervous system where some properties are exagerrated and others are ignored. For the moment we will concentrate on Single Layer Perceptrons.

So how can we achieve learning in our model neuron? We need to train them so they can do things that are useful. To do this we must allow the neuron to learn from its mistakes. There is in fact a learning paradigm that achieves this, it is known as supervised learning and works in the following manner.

i. set the weight and thresholds of the neuron to random values.
ii. present an input.
iii. caclulate the output of the neuron.
iv. alter the weights to reinforce correct decisions and discourage wrong decisions, hence reducing the error. So for the network to learn we shall increase the weights on the active inputs when we want the output to be active, and to decrease them when we want the output to be inactive.
v. Now present the next input and repeat steps iii. - v.

Perceptron Learning Algorithm:

The algorithm for Perceptron Learning is based on the supervised learning procedure discussed previously.

Algorithm:

i. Initialize weights and threshold.

   Set $w_i(t)$, $(0 <= i <= n)$, to be the weight i at time t, and $ø$ to be the threshold value in the output node. Set w_0 to be $-ø$, the bias, and x_0 to be always 1.

   Set $w_i(0)$ to small random values, thus initializing the weights and threshold.

ii. Present input and desired output

   Present input $x_0, x_1, x_2, ..., x_n$ and desired output $d(t)$

iii. Calculate the actual output

   $y(t) = g\,[w_0(t)x_0(t) + w_1(t)x_1(t) + ... + w_n(t)x_n(t)]$

iv. Adapts weights

   $w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$, where $0 <= \alpha <= 1$ (learning rate) is a positive gain function that controls the adaption rate.

Steps iii. and iv. are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

Please note that the weights only change if an error is made and hence this is only when learning shall occur.

Delta Rule:

The **delta rule** is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. It is a special case of the more general backpropagation algorithm. For a neuron j with activation function $g(x)$ the delta rule for j's ith weight $w_{ji}$is given by

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i,$$

where α is a small constant called *learning rate*, $g(x)$ is the neuron's activation function, $t_j$is the target output, h_j is the weighted sum of the neuron's inputs, y_j is the actual output, and x_i is the ith input. It holds $h_j = \sum x_i w_{ji}$and $y_j = g(h_j)$.

The delta rule is commonly stated in simplified form for a perceptron with a linear activation function as

$$\Delta w_{ji} = \alpha(t_j - y_j)x_i$$

Backpropagation

It is a supervised learning method, and is an implementation of the **Delta rule**. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Backpropagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, backpropagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple *stochastic gradient descent algorithm, is a general optimization algorithm, but is typically used to fit the parameters of a machine learning model, to find weights that minimize the error*. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network.

Summary of the backpropagation technique:

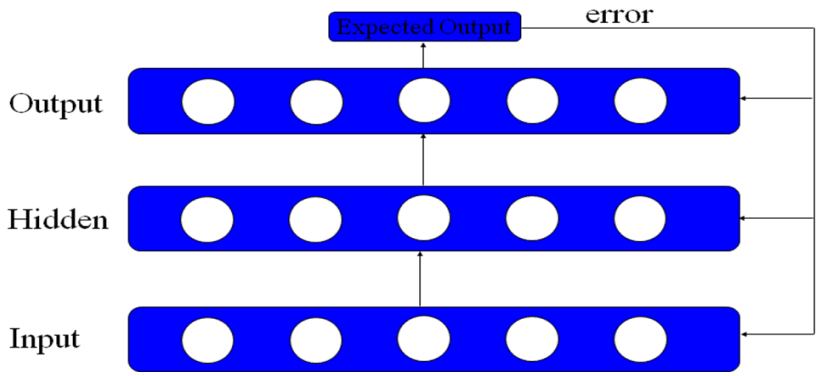
1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a *scaling factor*, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat from step 3 on the neurons at the previous level, using each one's "blame" as its error.

Characteristics:

- A multi-layered perceptron has three distinctive characteristics
  - The network contains one or more layers of hidden neurons
  - The network exhibits a high degree of connectivity
  - Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity:

$$y_j = \frac{1}{1 + e^{s_j}}$$

- The back propagation algorithm provides a computational efficient method for training multi-layer networks



Algorithm:

Step 0: Initialize the weights to small random values

Step 1: Feed the training sample through the network and determine the final output

Step 2: Compute the error for each output unit, for unit k it is:

$$\delta_k = (t_k - y_k) f'(y\_in_k)$$

Actual output

Required output

Derivative of f

Step 3: Calculate the weight correction term for each output unit, for unit k it is:

$$\Delta w_{jk} = \alpha \delta_k z_j$$

Hidden layer signal

A small constant

Step 4: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the j^{th} hidden unit is:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

The delta term for j^{th} hidden unit is: $\delta_j = \delta\_in_j f'(z\_in_j)$

Step 5: Calculate the weight correction term for the hidden units: $\Delta w_{ij} = \alpha \delta_j x_i$

Step 6: Update the weights: $w_{ik}(new) = w_{ik}(old) + \Delta w_{ik}$

Step 7: Test for stopping (maximum cylces, small changes, etc)

Note: There are a number of options in the design of a backprop system;

- Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say –0.5 to 0.5)
- Number of cycles – tend to be quite large for backprop systems
- Number of neurons in the hidden layer – as few as possible