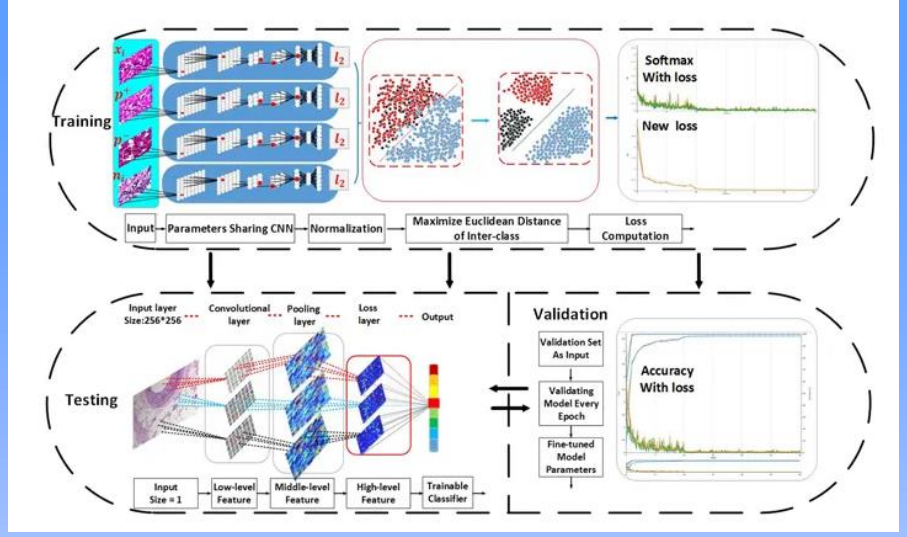
Artificial Intelligence

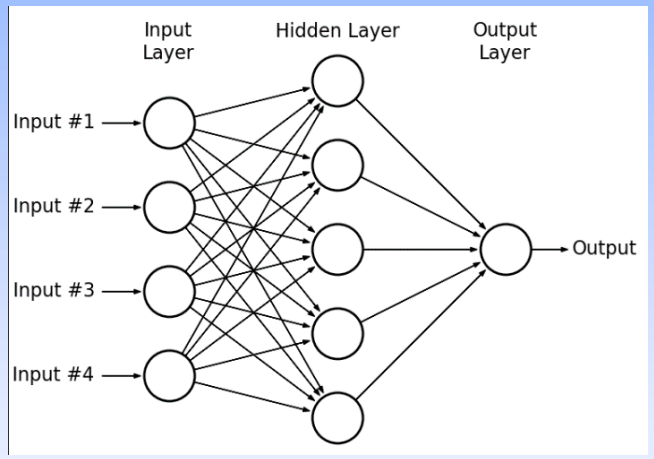
Lecturer: Professor Pekka Toivanen

E-mail: Pekka.Toivanen@uef.fi

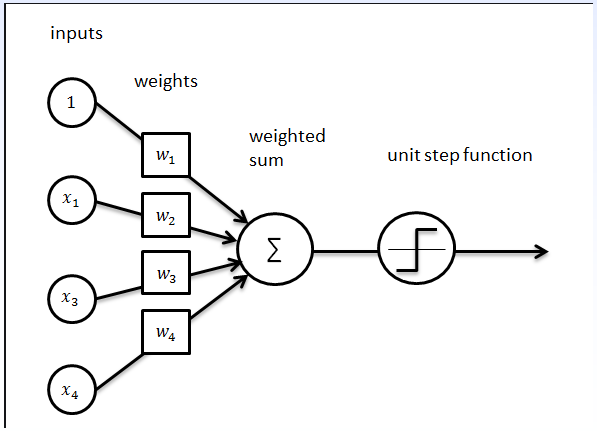
Deep learning neural networks



Multi-Layer Perceptron

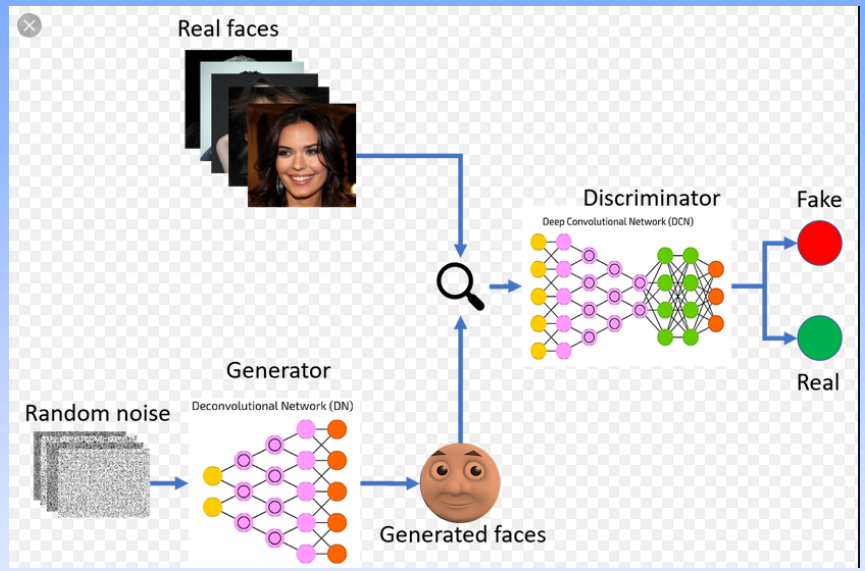


Perceptron

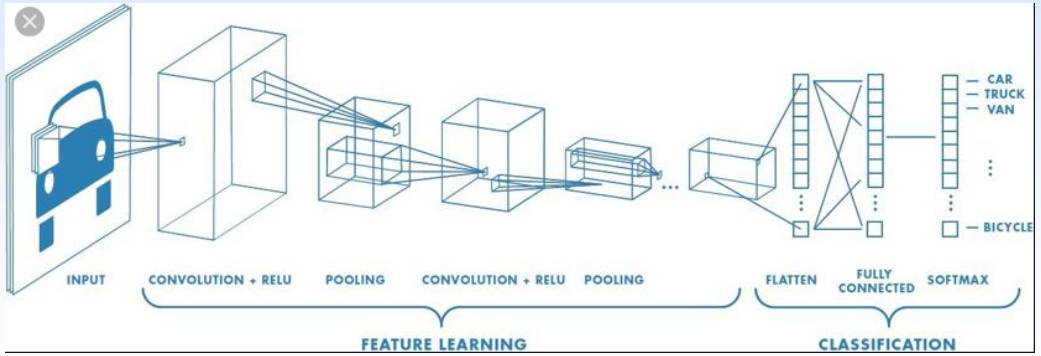


...

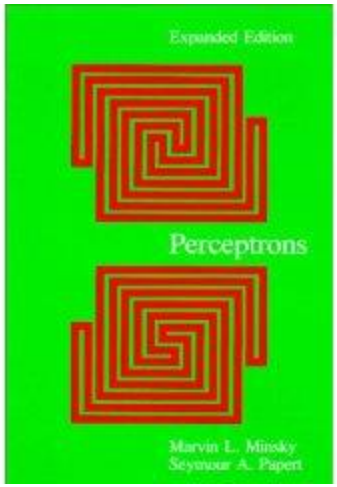
Generative Adversarial Networks (GAN)

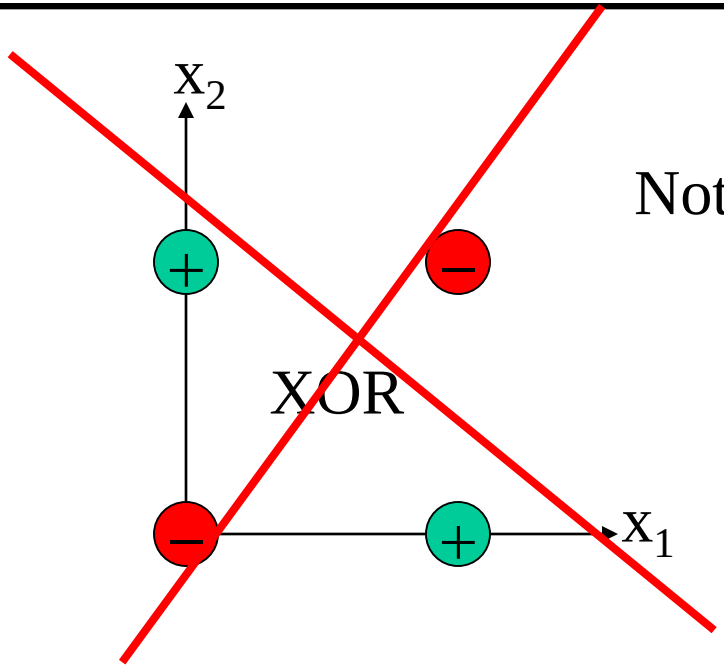


Convolutional Neural Networks



Perceptron: Linear Separable Functions

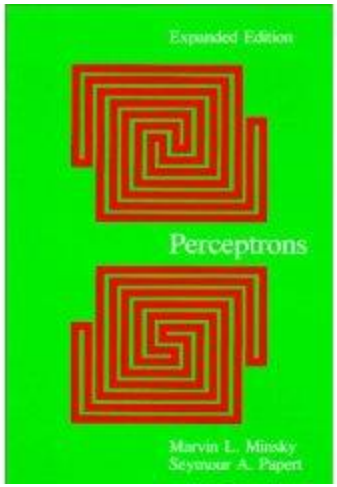




Not linearly separable

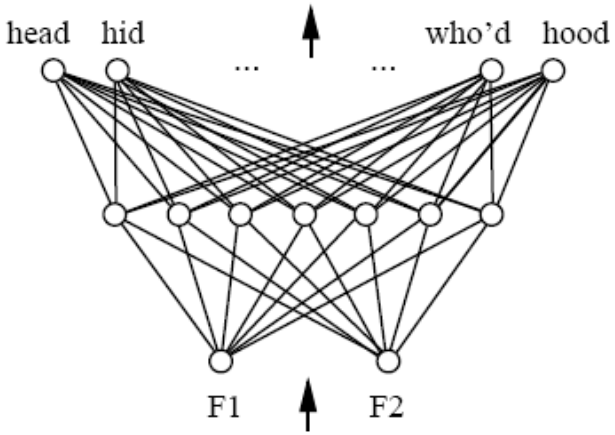
Minsky & Papert (1969)

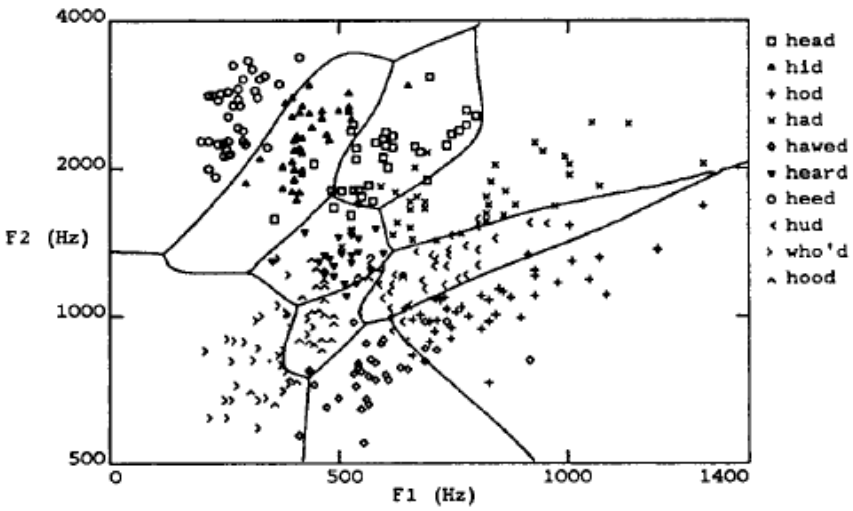
Perceptrons can only represent linearly separable functions.



Good news: **Adding hidden layer allows more target functions to be represented!!!!!**

Minsky & Papert (1969)





Hidden Units

Hidden units are nodes that are situated between the input nodes and the output nodes.

Hidden units allow a network to learn non-linear functions.

Hidden units allow the network to represent combinations of the input features.

Boolean functions

Perceptron can be used to represent the following Boolean functions

- AND
- OR
- Any m-of-n function
- NOT
- NAND (NOT AND)
- NOR (NOT OR)

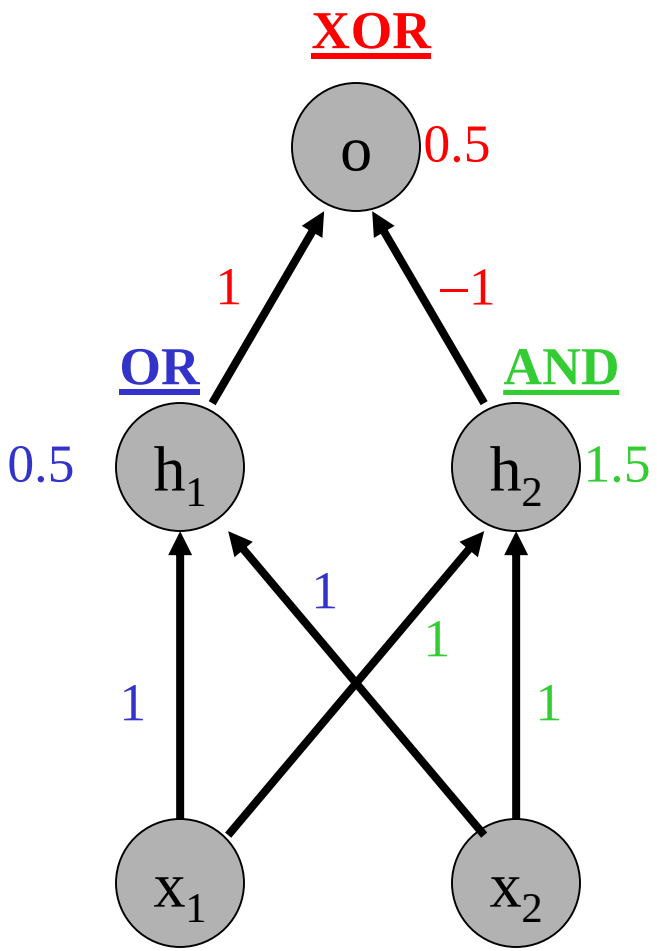
Every Boolean function can be represented by a network of interconnected units based on these primitives

Two levels (i.e., one hidden layer) is enough!!!

Boolean XOR

$X1 \oplus X2 \Leftrightarrow (X1 \vee X2) \wedge \neg(X1 \wedge X2)$

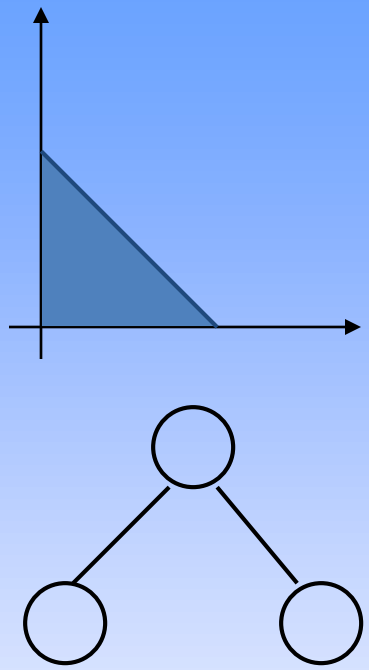
Not Linear separable →
Cannot be represented by a single-layer perceptron



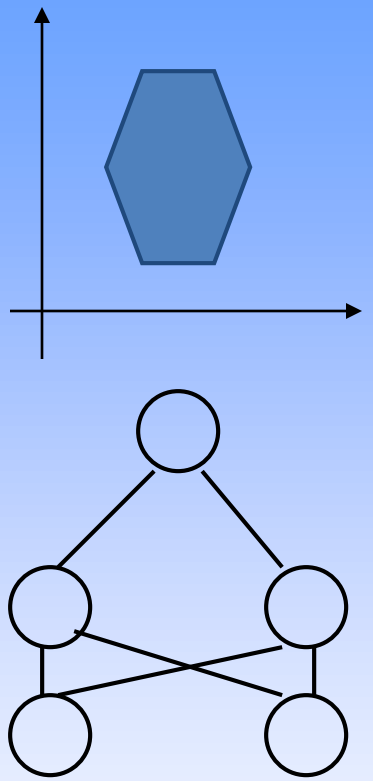
Let's consider a single hidden layer network, using as building blocks threshold units.

$w_1 x_1 + w_2 x_2 - w_0 > 0$

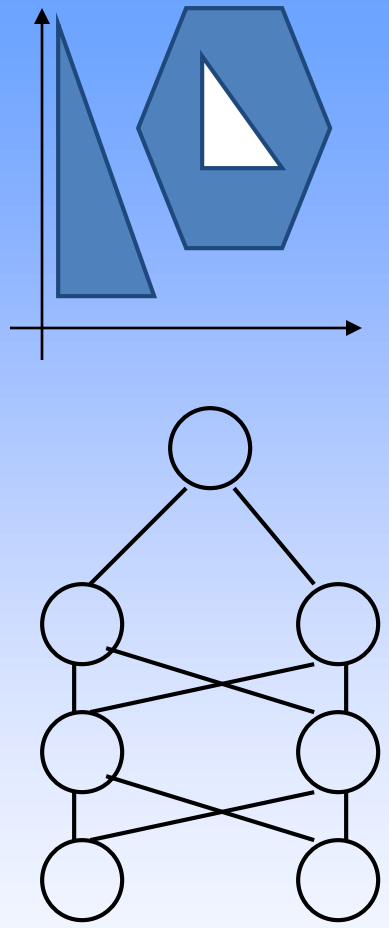
What do each of the layers do?



1st layer draws linear boundaries



2nd layer combines the boundaries



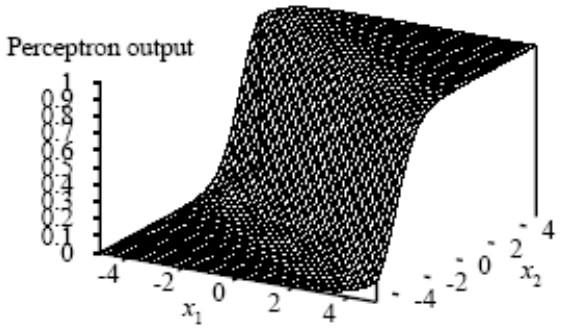
3rd layer can generate arbitrarily complex boundaries

Expressiveness of MLP: Soft Threshold

Advantage of adding hidden layers

→ it enlarges the space of hypotheses that the network can represent.

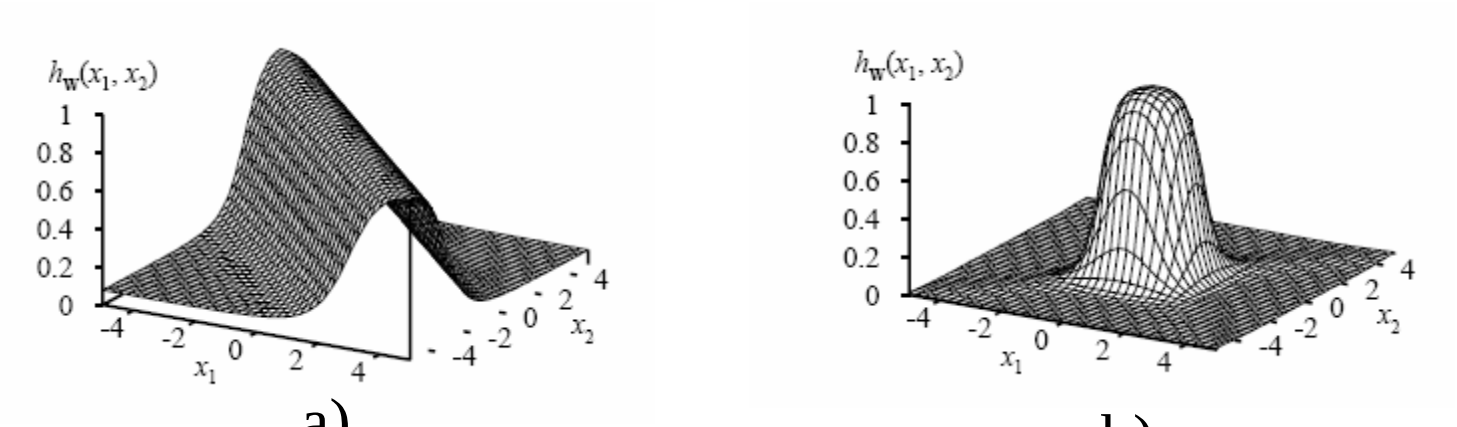
Example: we can think of each hidden unit as a perceptron that represents a soft threshold function in the input space, and an output unit as as a soft-thresholded linear combination of several such functions.



(b)

Soft threshold function

Expressiveness of MLP



(a) The result of combining two opposite-facing soft threshold functions to produce a ridge.

(b) The result of combining two ridges to produce a bump.

Add bumps of various sizes and locations to any surface

All continuous functions w/ 2 layers, all functions w/ 3 layers

Expressiveness of MLP

With a single, sufficiently large hidden layer, it is possible to represent **any continuous function** of the inputs with arbitrary accuracy;

With two layers, even discontinuous functions can be represented.

The proof is complex → main point, required number of hidden units grows exponentially with the number of inputs.

For example, $2^n/n$ hidden units are needed to encode all Boolean functions of n inputs.

Issue: For any *particular* network structure, it is harder to characterize exactly which functions can be represented and which ones cannot.

Multi-Layer Feedforward Networks

Boolean functions:

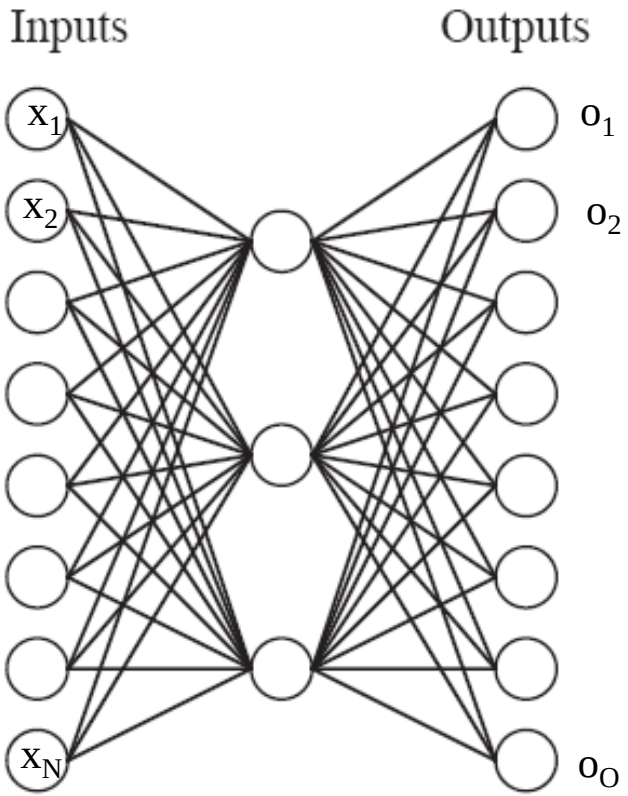
Every boolean function can be represented by a network with single hidden layer

But might require exponential (in number of inputs) hidden units

Continuous functions:

Every bounded continuous function can be approximated with arbitrarily small error, by network with single hidden layer [Cybenko 1989; Hornik et al. 1989]

Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].



$$o_i = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_j\right)\right)$$

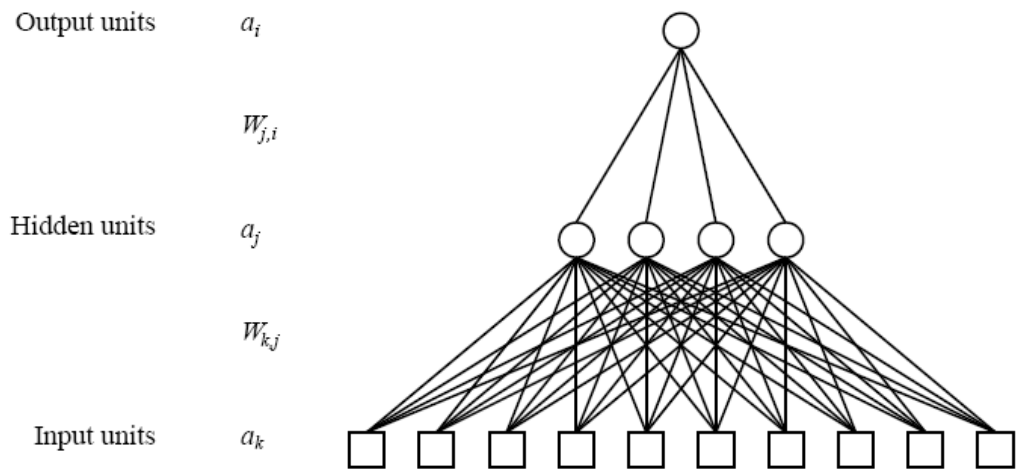
The Restaurant Example

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Goal predicate: WillWait?

Multi-layer Feedforward Neural Networks or Multi-Layer Perceptrons (MLP)



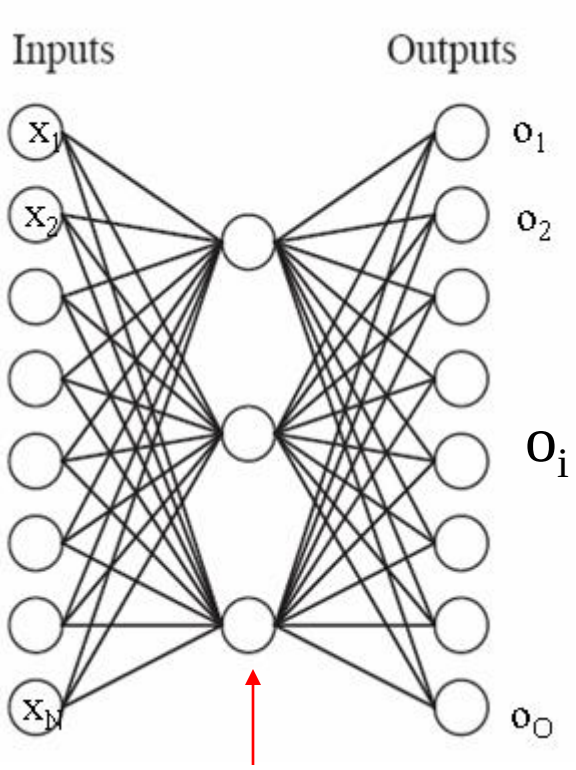
A multilayer neural network with one hidden layer and 10 inputs, suitable for the restaurant problem.

Layers are often fully connected (but not always).

Number of hidden units typically chosen by hand.

Learning Algorithm for MLP

Learning Algorithms for MLP



How to compute the errors for the hidden units?

Goal: minimize sum squared errors

$Err_1=y_1-o_1$

$Err_2=y_2-o_2$

$$E = \frac{1}{2}\sum_i (y_i - o_i)^2$$

$Err_i=y_i-o_i$

$$o_i = g\left(\sum_h w_{h,i}\, g\left(\sum_j w_{j,h}\, x_j\right)\right)$$

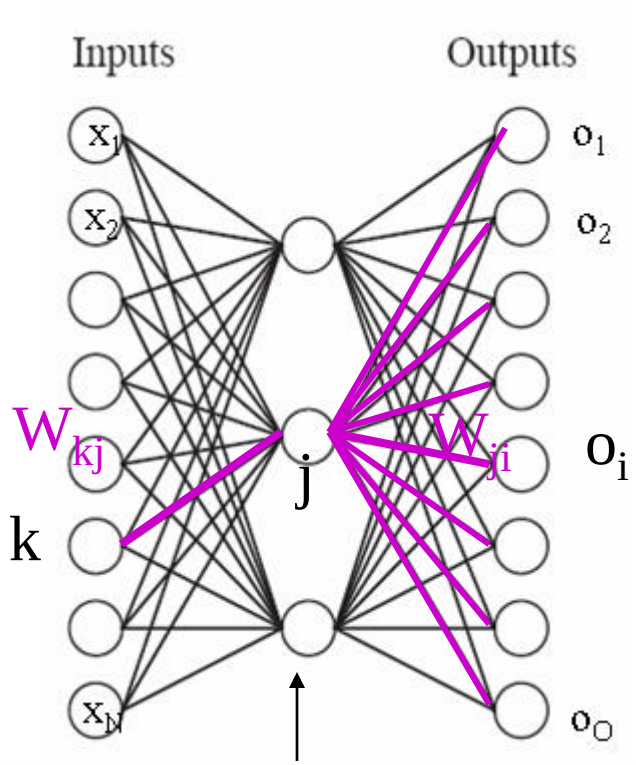
parameterized function of inputs: weights are the parameters of the function.

$Err_o=y_o-o_o$

Clear error at the output layer

We can **back-propagate** the error from the output layer to the hidden layers.
The back-propagation process emerges directly from a derivation of the overall error gradient.

Backpropagation Learning Algorithm for MLP



Perceptron update:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

$Err_i = y_i - o_i$

Output layer weight update (similar to perceptron)

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

$$\Delta_i = Err_i \times g'(in_i)$$

Hidden layer: back-propagate the error from the output layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

$$\Delta_j = g'(in_j) \underbrace{\sum_i W_{j,i}\Delta_i}_{} \ .$$

Err_j → "Error" for hidden node j

Hidden node j is "responsible" for some fraction of the error i in each of the output nodes to which it connects

→ depending on the strength of the connection between the hidden node and the output node i.

Backpropagation Training (Overview)

Optimization Problem

- Obj.: minimize E

$$E = \frac{1}{2}\sum_i (y_i - a_i)^2 \ ,$$

Choice of learning rate α

How many restarts (local optima) of search to find good optimum of objective function?

Variables: network weights w_{ij}

Algorithm: local search via gradient descent.

Randomly initialize weights.

Until performance is satisfactory, cycle through examples (epochs):

- Update each weight:

Output node:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

$$\Delta_i = Err_i \times g'(in_i)$$

Hidden node:

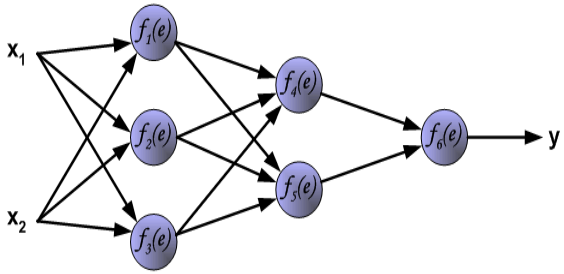
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

$$\Delta_j = g'(in_j)\sum_i W_{j,i}\Delta_i \ .$$

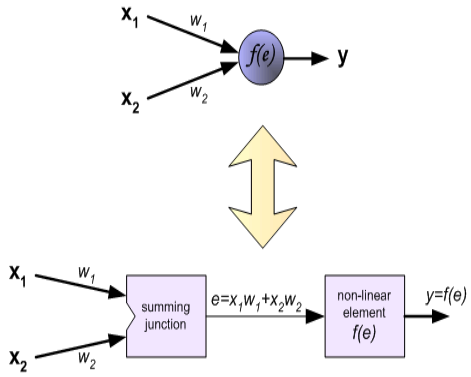
See derivation details in the next hidden slides (pages 745-747 R&N)

Principles of training multi-layer neural network using backpropagation

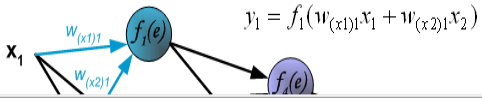
The project describes teaching process of multi-layer neural network employing *backpropagation* algorithm. To illustrate this process the three layer neural network with two inputs and one output,which is shown in the picture below, is used:



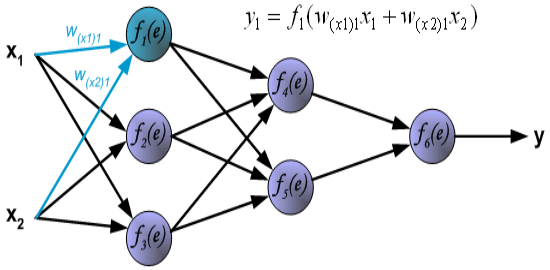
Each neuron is composed of two units. First unit adds products of weights coefficients and input signals. The second unit realise nonlinear function, called neuron activation function. Signal e is adder output signal, and $y = f(e)$ is output signal of nonlinear element. Signal y is also output signal of neuron.

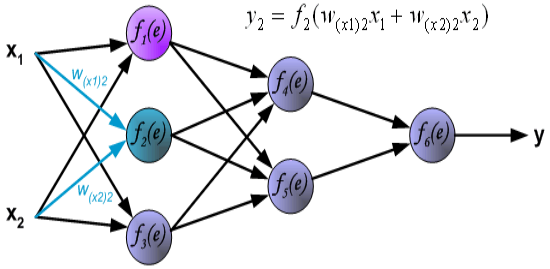


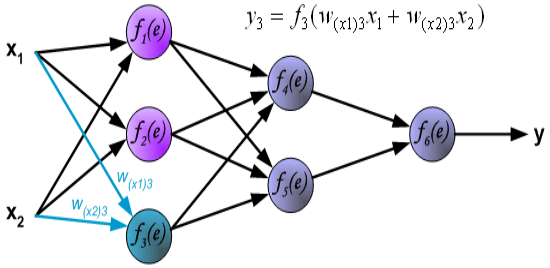
To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) z. The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below: Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer. Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.



To teach the neural network we need training data set. The training data set consists of input signals (x_1 and x_2) assigned with corresponding target (desired output) z. The network training is an iterative process. In each iteration weights coefficients of nodes are modified using new data from training data set. Modification is calculated using algorithm described below: Each teaching step starts with forcing both input signals from training set. After this stage we can determine output signals values for each neuron in each network layer. Pictures below illustrate how signal is propagating through the network, Symbols $w_{(xm)n}$ represent weights of connections between network input x_m and neuron n in input layer. Symbols y_n represents output signal of neuron n.

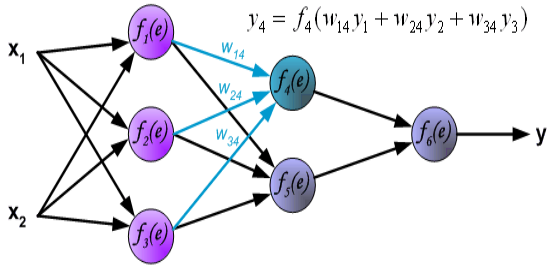


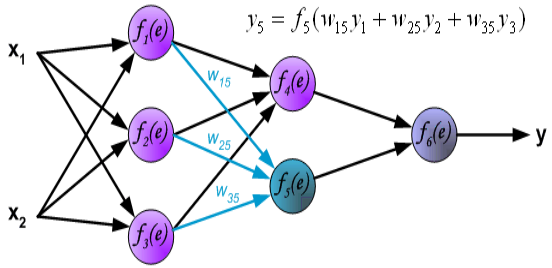




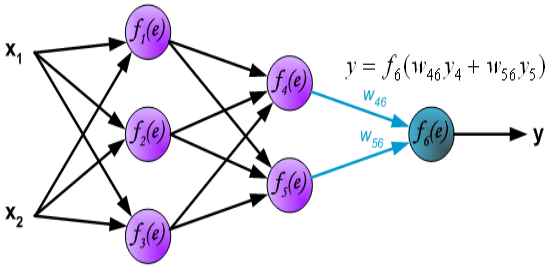
Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.

Propagation of signals through the hidden layer. Symbols w_{mn} represent weights of connections between output of neuron m and input of neuron n in the next layer.



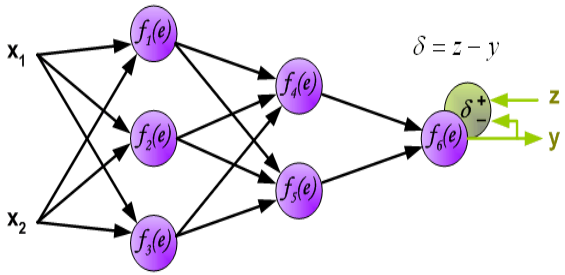


Propagation of signals through the output layer.

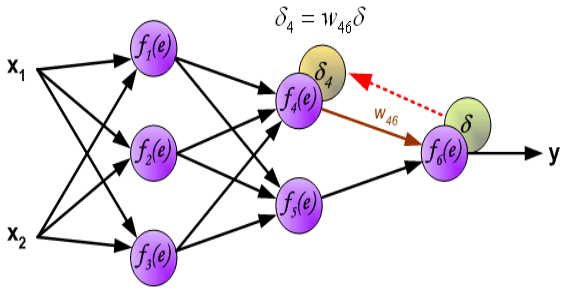


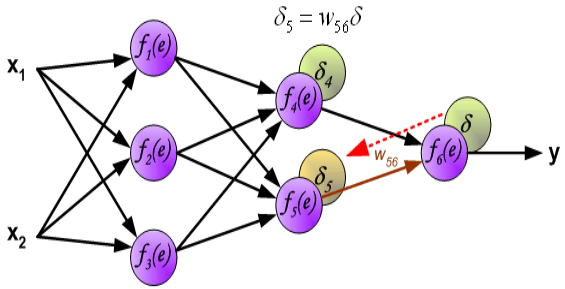
In the next algorithm step the output signal of the network y is compared with the desired output value (the target), which is found in training data set. The difference is called error signal δ of output layer neuron.

In the next algorithm step the output signal of the network y is compared with the desired output value (the target), which is found in training data set. The difference is called error signal δ of output layer neuron.



It is impossible to compute error signal for internal neurons directly, because output values of these neurons are unknown. For many years the effective method for training multiplayer networks has been unknown. Only in the middle eighties the backpropagation algorithm has been worked out. The idea is to propagate error signal δ (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.

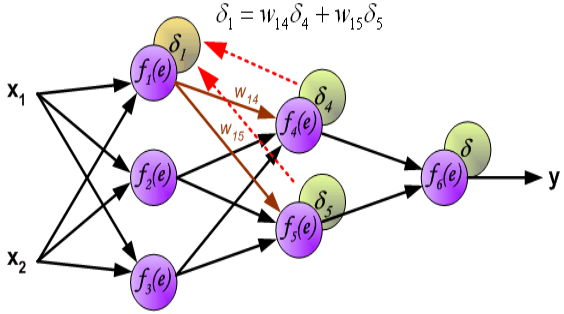




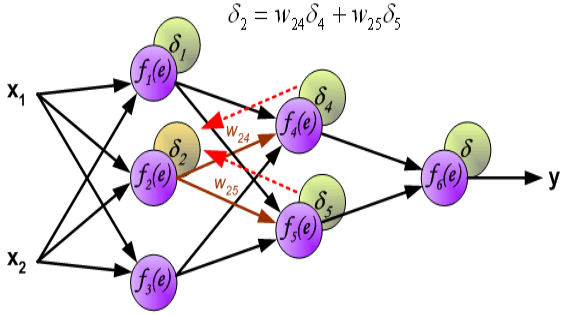
The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers.

The weights' coefficients w_{mn} used to propagate errors back are equal to this used during computing output value. Only the direction of data flow is changed (signals are propagated from output to inputs one after the other). This technique is used for all network layers. If propagated errors came from few neurons they are added. The illustration is below:

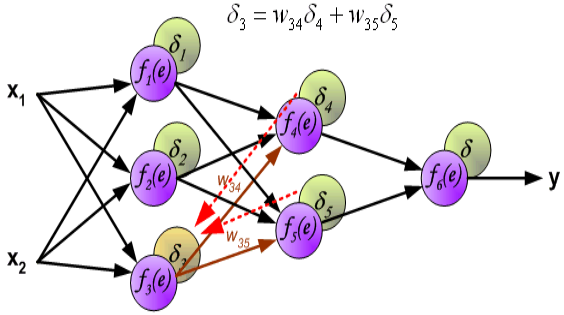
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



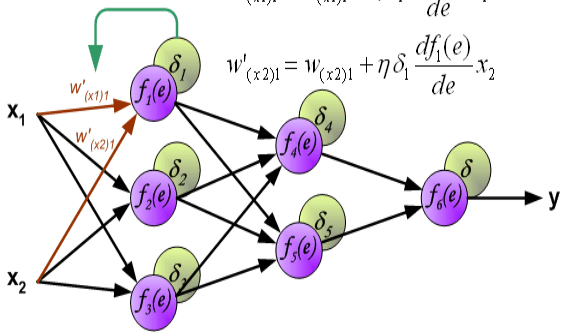
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified. In formulas below *df(e)/de* represents derivative of neuron activation function (which weights are modified).

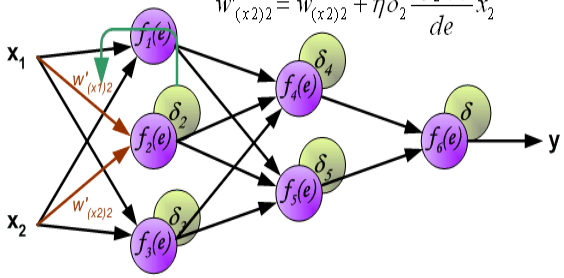
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



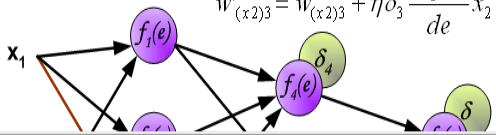
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



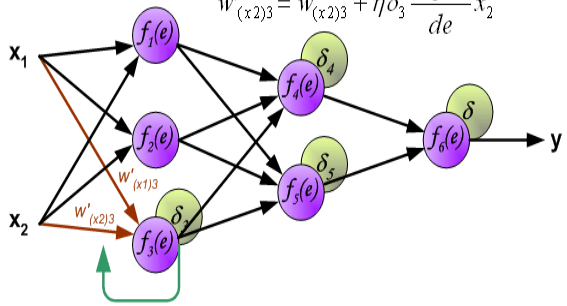
$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

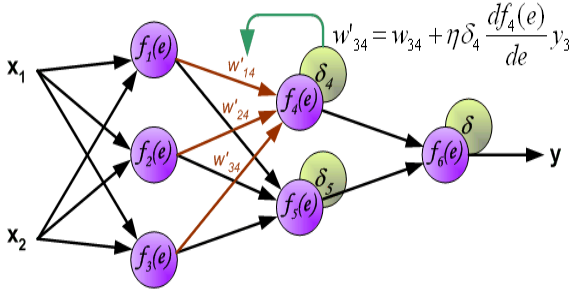
$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

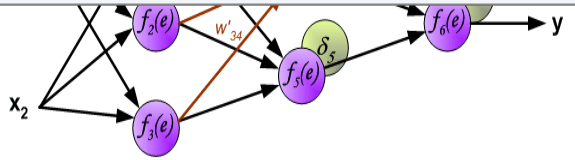
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

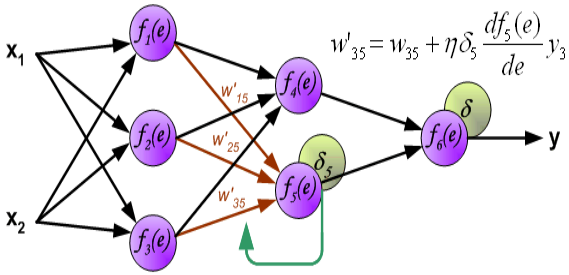
$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

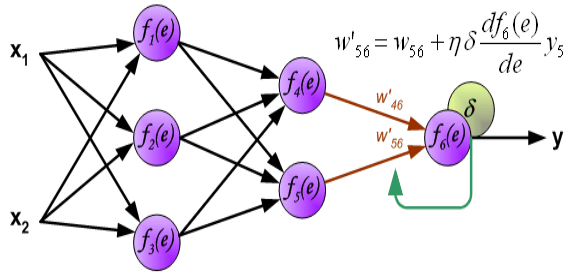
$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

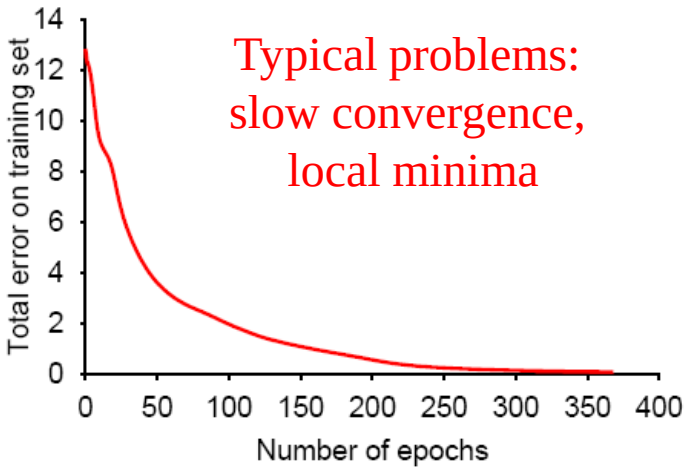


$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$

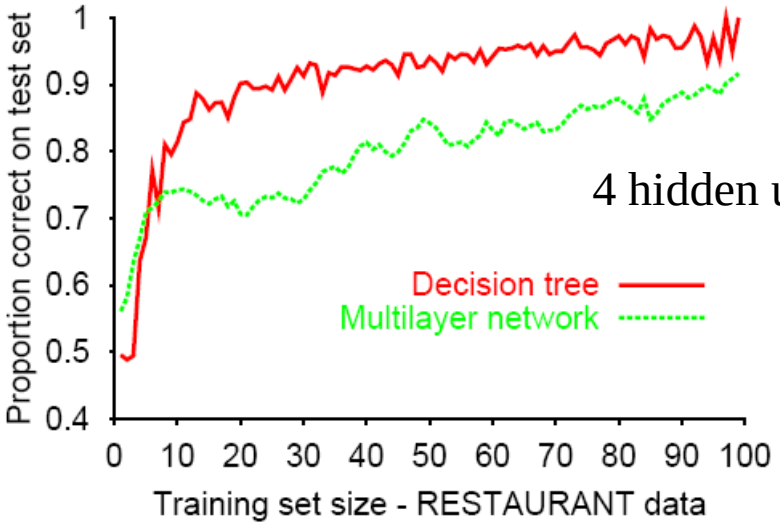


Coefficient η affects network teaching speed. There are a few techniques to select this parameter. The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually. The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.



Typical problems:
slow convergence,
local minima

(a)



4 hidden units:

(b)

(a) Training curve showing the gradual reduction in error as weights are modified over several epochs, for a given set of examples in the restaurant domain.

(b) Comparative learning curves showing that decision-tree learning does slightly better than back-propagation in a multilayer network.

Design Decisions

Network architecture

How many hidden layers? How many hidden units per layer?

Given too many hidden units, a neural net will simply memorize the input patterns (overfitting).

Given too few hidden units, the network may not be able to represent all of the necessary generalizations (underfitting).

How should the units be connected? (Fully? Partial? Use domain knowledge?)

Learning Neural Networks Structures

Fully connected networks

How many layers? How many hidden units?

Cross-validation to choose the one with the highest prediction accuracy on the validation sets.

Not fully connected networks – search for right topology (large space)

Optimal- Brain damage – start with a fully connected network; Try removing connections from it.

Tiling – algorithm for growing a network starting with a single unit

How long should you train the net?

The goal is to achieve a balance between correct responses for the training patterns and correct responses for new patterns. (That is, a balance between memorization and generalization).

If you train the net for too long, then you run the risk of overfitting.

Select number of training iterations via cross-validation on a holdout set.

Multi Layer Networks: expressiveness vs. computational complexity

Multi- Layer networks → very expressive!

They can represent general non-linear functions!!!!

But…

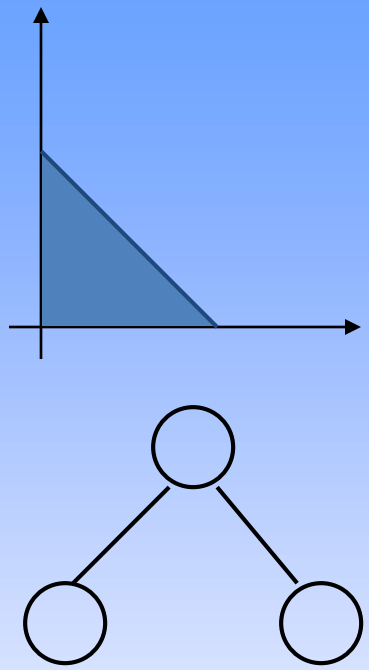
In general they are hard to train due to the abundance of local minima and high dimensionality of search space

Also resulting hypotheses cannot be understood easily

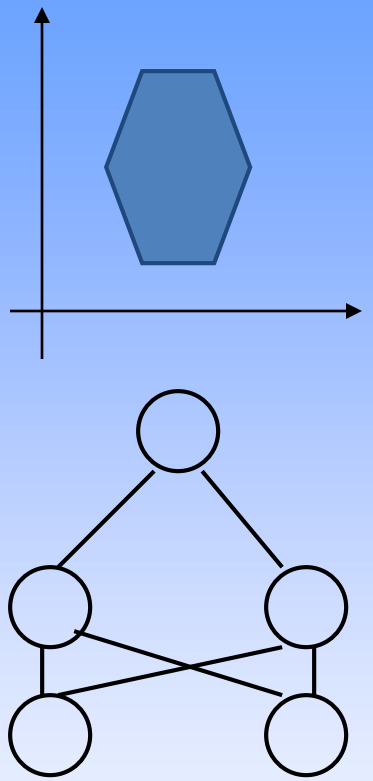
Summary

- Perceptrons (one-layer networks) limited expressive power—they can learn only linear decision boundaries in the input space.
- Single-layer networks have a simple and efficient learning algorithm;
- Multi-layer networks are sufficiently expressive —they can represent general nonlinear function ; can be trained by gradient descent, i.e., error back-propagation.
- Multi-layer perceptron harder to train because of the abundance of local minima and the high dimensionality of the weight space
- Many applications: speech, driving, handwriting, fraud detection, etc.

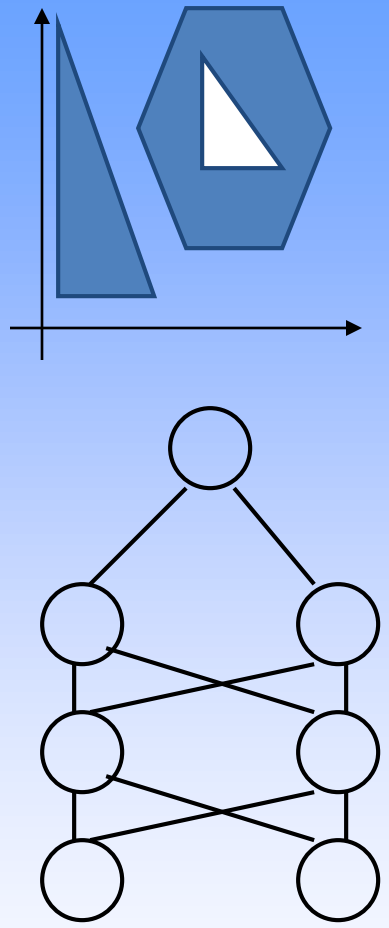
What do each of the layers do?



1st layer draws linear boundaries



2nd layer combines the boundaries



3rd layer can generate arbitrarily complex boundaries

Real-world examples: sales rate prediction

- Bild-Zeitung is the most frequently sold newspaper in Germany, approx. 4.2 million copies per day
- it is sold in 110 000 sales outlets in Germany, differing in a lot of facets
- problem: how many copies are sold in which sales outlet?

Real-world examples: sales rate prediction

- Bild-Zeitung is the most frequently sold newspaper in Germany, approx. 4.2 million copies per day
- it is sold in 110 000 sales outlets in Germany, differing in a lot of facets
- problem: how many copies are sold in which sales outlet?
- neural approach: train a neural network for each sales outlet, neural network predicts next week's sales rates
- system in use since mid of 1990s

International Journal of Computer Applications (0975 – 8887)
International Conference on Advances in Science and Technology (ICAST-2014)

Design of Optimal MLP NN for Speaker Dependent Spoken Words Recognition Application

Sneha B. Lonkar
PG Student M.E. (Electronics and Telecom)
Vivekanand Education Society's Institute of Technology, Mumbai

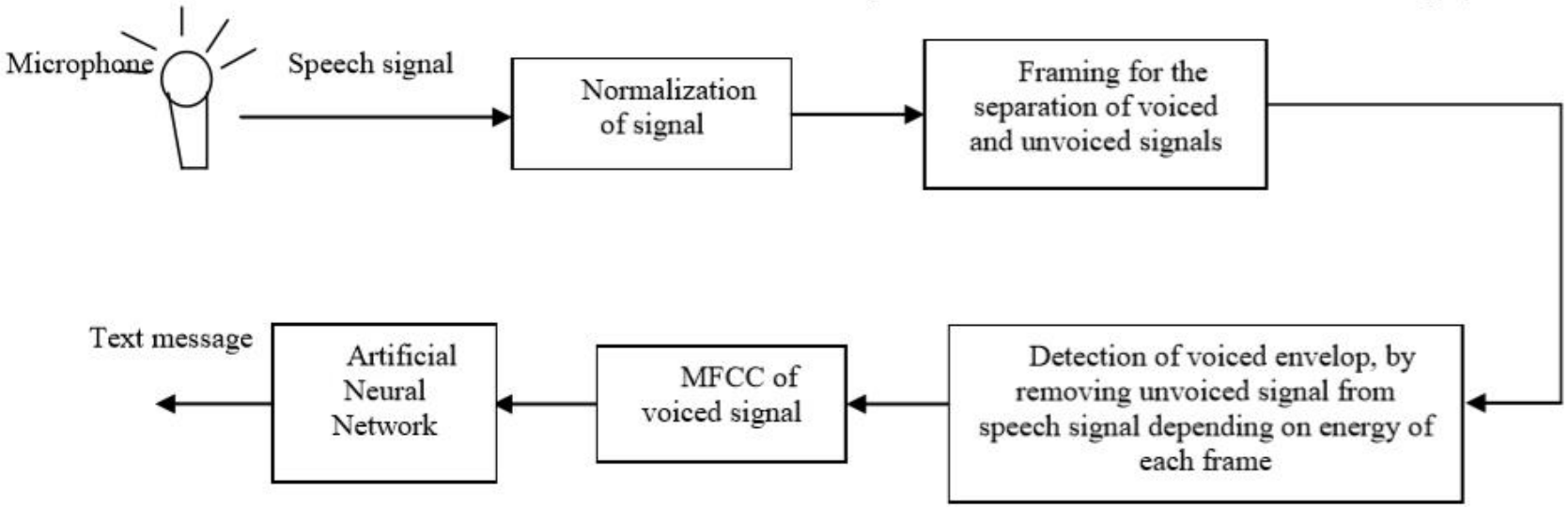
Nadir N. Charniya
Professor (Electronics and Telecom)
Vivekanand Education Society's Institute of Technology, Mumbai

ABSTRACT

Spoken words recognition provides applications like spoken commands recognitions in robotics command, speech based number dialing for phones and mobiles, etc. It also provides applications in railway and banking areas. This work aims at designing of optimal Multilayer Perceptron Neural Network (MLP NN) based classifiers for speaker dependent spoken digits recognition. The classifier attempted as optimal leading to less number of computations and few components requirement for its future implementation in hardware leading to a low cost speech recognition system. Isolated spoken digits were used as an input data to the neural networks based classifiers. Each spoken word was analyzed for the feature like Mel Frequency Cepstral Coefficients (MFCC). The MLP NN based classifier was designed meticulously with the condition of minimum components and attempting maximum classification accuracy.

hidden layer and linear transfer function in the output layer can virtually approximate any nonlinear function to any degree of accuracy provided sufficient number of neurons in the hidden layer is available [7], [8]. It is found that the choice of learning algorithm and activation function is important to improve network performance [9]. The point where the validation set starts to decrease in performance, while the training set continues to increase is the point that gives the maximum training and validation performance [10]. Methodology for design of near-optimal MLP NN based classifier based stopping point from learning curve for training and validation sets is presented by [11]. S V Dudul [12] described training procedure of MLP NN. Meng Joo Er *et al.* [13] presented a general design approach using neural network based classifier for face recognition to cope with small training sets of high-dimensional problem.

Several researchers have used neural network for speech


Microphone
Speech signal
Normalization of signal
Framing for the separation of voiced and unvoiced signals
Detection of voiced envelop, by removing unvoiced signal from speech signal depending on energy of each frame
MFCC of voiced signal
Artificial Neural Network
Text message

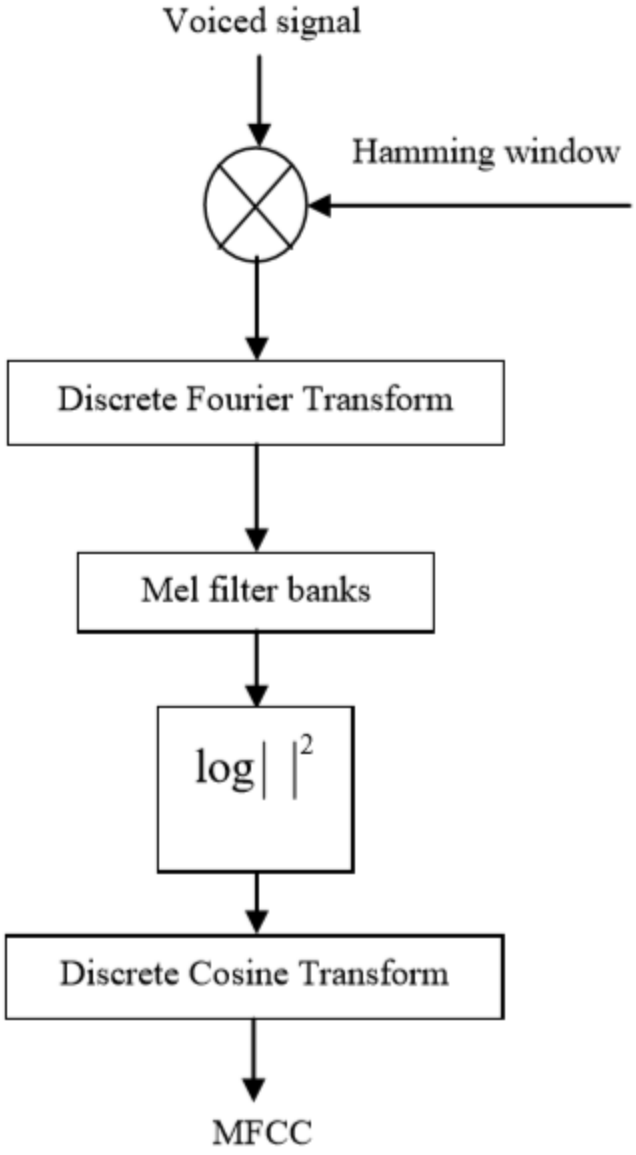


Fig 2: Block diagram of MFCC

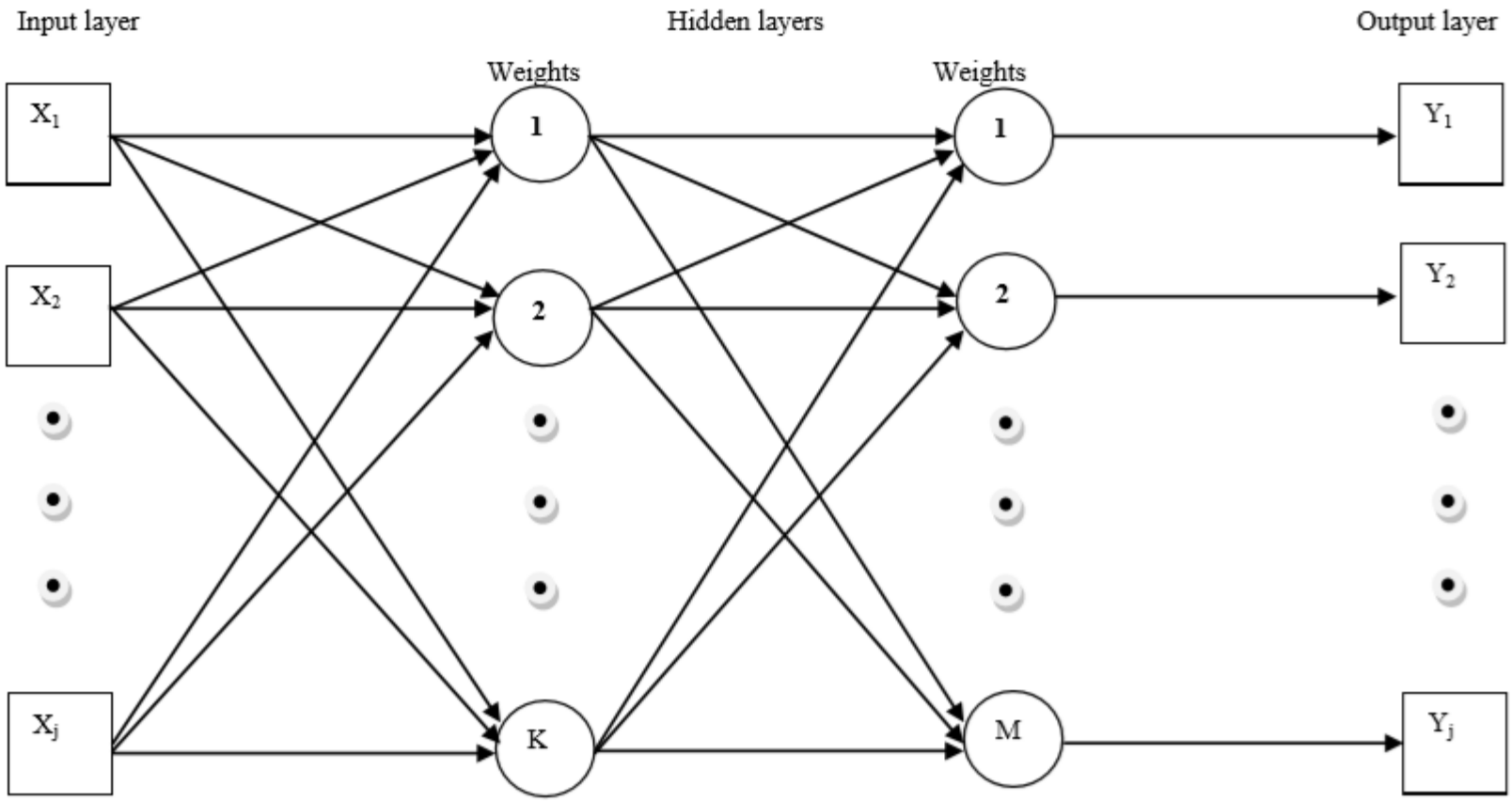


Fig 3: Multilayer neural network

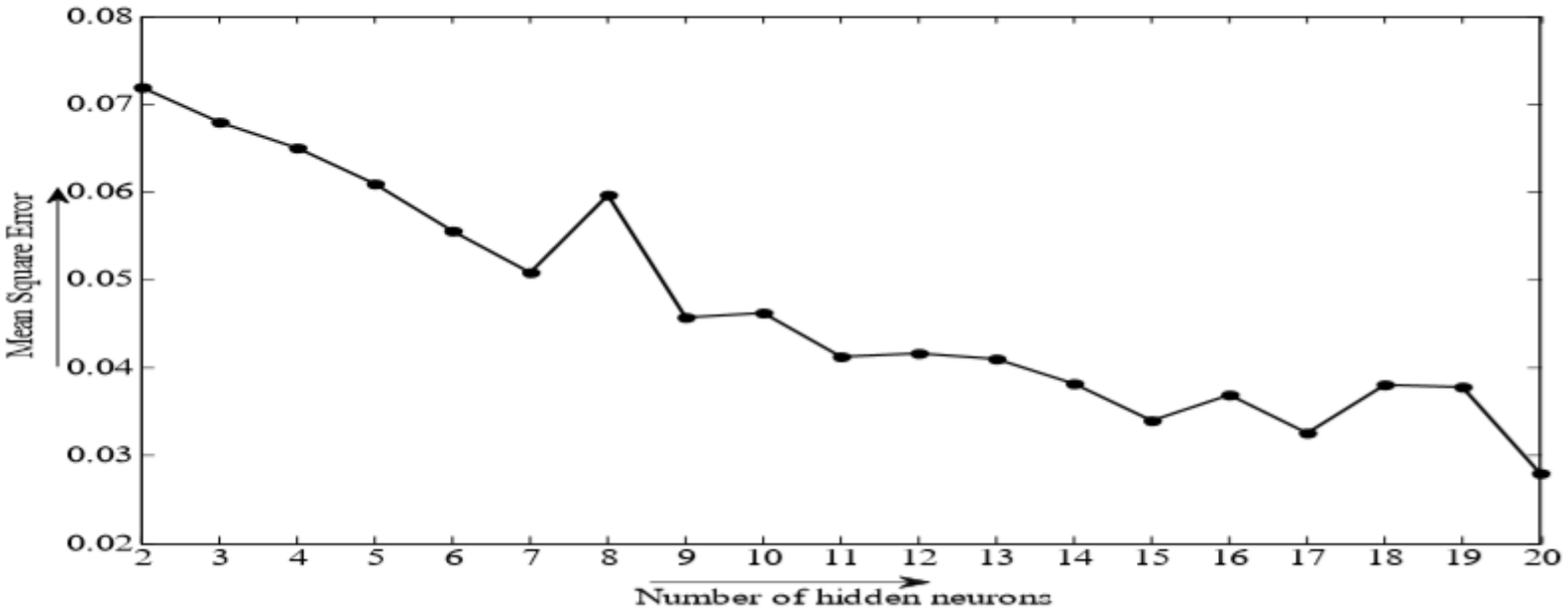


Fig 4: Plot of Mean Square Error verses Number of hidden neurons

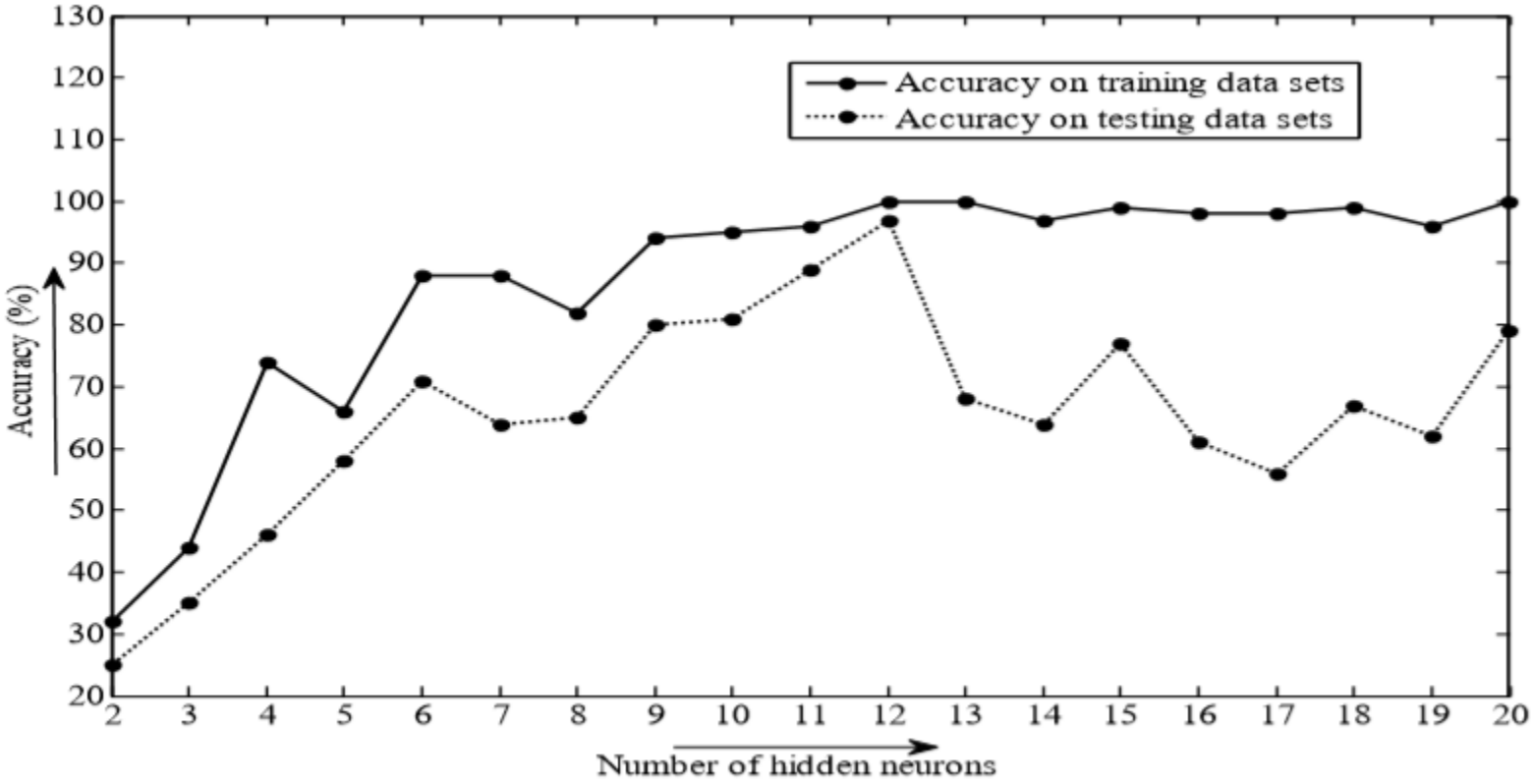


Fig 5: Plot of Accuracy verses Number of neurons on training and testing data sets

Gradient descent, how neural networks learn:

https://www.youtube.com/watch?v=IHZwWFHWa-w

Backpropagation:

https://www.youtube.com/watch?v=Ilg3gGewQ5U

Backpropagation calculus:

https://www.youtube.com/watch?v=tIeHLnjs5U8