Perceptron Networks and Applications

M. Ali Akcayol
Gazi University
Department of Computer Engineering

Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Training

- By learning rule we mean a procedure (training algorithm) for modifying the weights and biases of a network.
- The purpose of the learning rule is to train the network to perform some task.
- There are many types of neural network learning rules.
- They fall into three broad categories:
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Training

Supervised learning

- In supervised learning, the learning rule is provided with a set of examples (the training set) of proper network behavior:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

where, p_q is an input to the network and t_q is the corresponding correct (target) output.

- As the inputs are applied to the network, the network outputs are compared to the targets.
- The learning rule is then used to adjust the weights and biases of the network in order to move the network outputs closer to the targets.
- The perceptron learning rule falls in this supervised learning category.

Training

Unsupervised learning

- In unsupervised learning, the weights and biases are modified in response to network inputs only.
- There are no target outputs available.
- At first glance this might seem to be impractical.
- How can you train a network if you don't know what it is supposed to do?
- Most of these algorithms perform some kind of clustering operation.
- They learn to categorize the input patterns into a finite number of classes.
- This is especially useful in such applications as vector quantization.

Training

Reinforcement learning

- Reinforcement learning is similar to supervised learning.
- There are no target values.
- Instead of being provided with the correct output for each network input, the algorithm is only given a grade.
- The grade (or score) is a measure of the network performance over some sequence of inputs.
- This type of learning is currently much less common than supervised learning.
- Genetic algorithms, tabu search, simulated annealing algorithms are in the reinforcement learning category.

Content

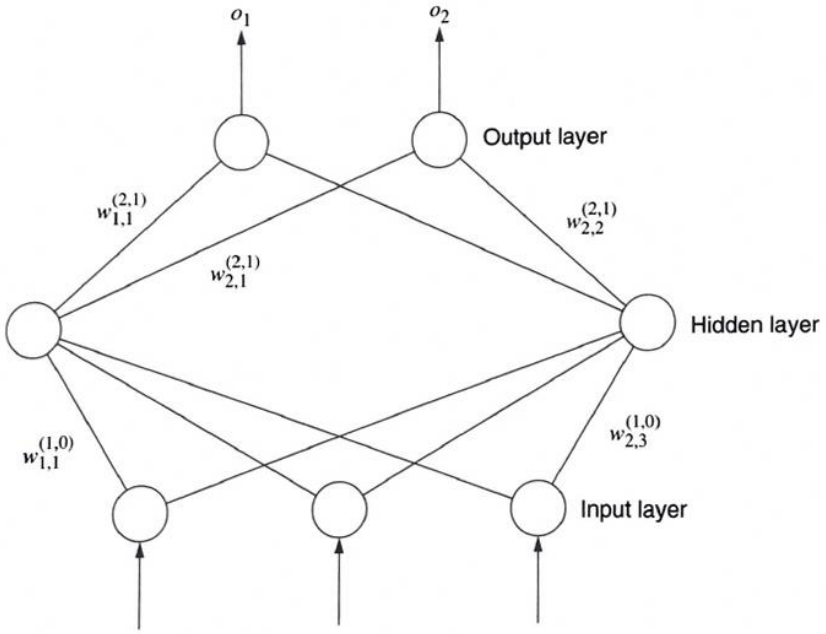
- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Backpropagation algorithm

- The backpopagation algorithm is a generalization of LMS algorithm.
- The backpropagation algorithm modifies the weights to minimize SSE or MSE.
- Backprop uses supervised learning in which the inputs and the corresponding outputs are used for training.
- Once the network is trained, the weights are frozen and the network can be used to compute output values for new input samples.
- The feedforward process involves presenting an input pattern to input layer neurons that pass the input values onto the first hidden layer.
- Each of the hidden layer nodes computes a weighted sum of its inputs, passes the sum through its activation function and presents the result to the output layer.

Backpropagation algorithm

Feedforward



- The ***i***th input node holds a value $x_{p,\,i}$ for the ***p***th pattern.
- The net input for ***j***th node in the hidden layer is (includes threshold $x_{p,\,0} = 1$),

$$net_j^{(1)} = \sum_{i=0}^{n} w_{j,i}^{(1,0)} x_{p,i}$$

- The connection from ***i***th input node to ***j***th hidden layer node,

$w_{j,i}^{(1,0)}$ where (1, 0) represents layer 1 (hidden layer), layer 0 (input layer).

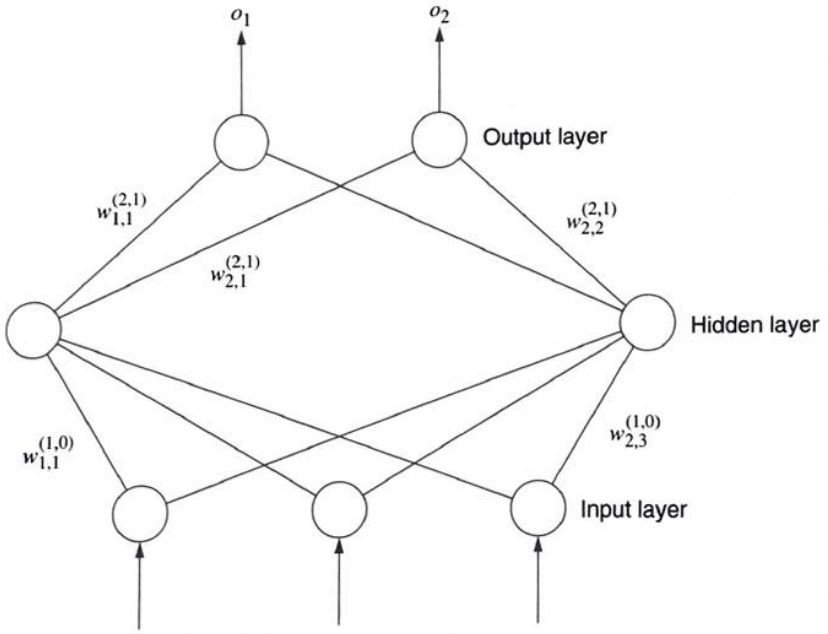
- The output of the ***j***th hidden layer node is

$$x_{p,j}^{(1)} = S\left(\sum_{i=0}^{n} w_{j,i}^{(1,0)} x_{p,i}\right)$$

where S is a sigmoid function.

Backpropagation algorithm

Feedforward



- The net input to the kth output layer node is,

$$net_k^{(2)} = \sum_j \left(w_{k,j}^{(2,1)} x_{p,j}^{(1)} \right)$$

- The connection from jth input node to kth output layer node,

$w_{k,j}^{(2,1)}$ where (2, 1) represents layer 2 (output layer), layer 1 (hidden layer).

- The output of the kth output layer node is,

$$o_{p,k} = S\left(\sum_j w_{k,j}^{(2,1)} x_{p,j}^{(1)} \right)$$

where S is a sigmoid function.

- The corresponding squared error is,

$$\ell_{p,k}^2 = |d_{p,k} - o_{p,k}|^2$$

Backpropagation algorithm

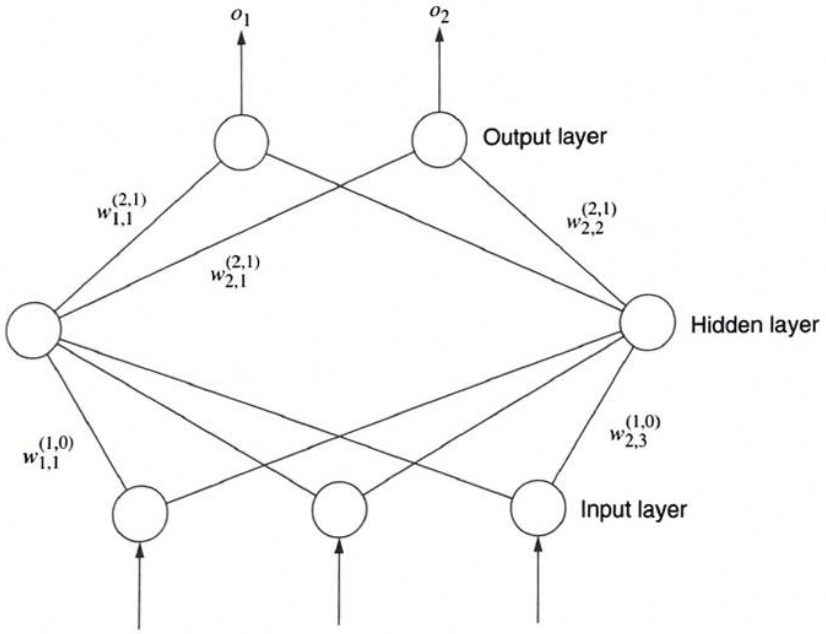
Backpropagation

- For each connection from the hidden layer to output layer, we calculate

$$\partial E / \partial w_{k,j}^{(2,1)}$$

For each connection from the input layer to hidden layer, we calculate

$$\partial E / \partial w_{j,i}^{(1,0)}$$



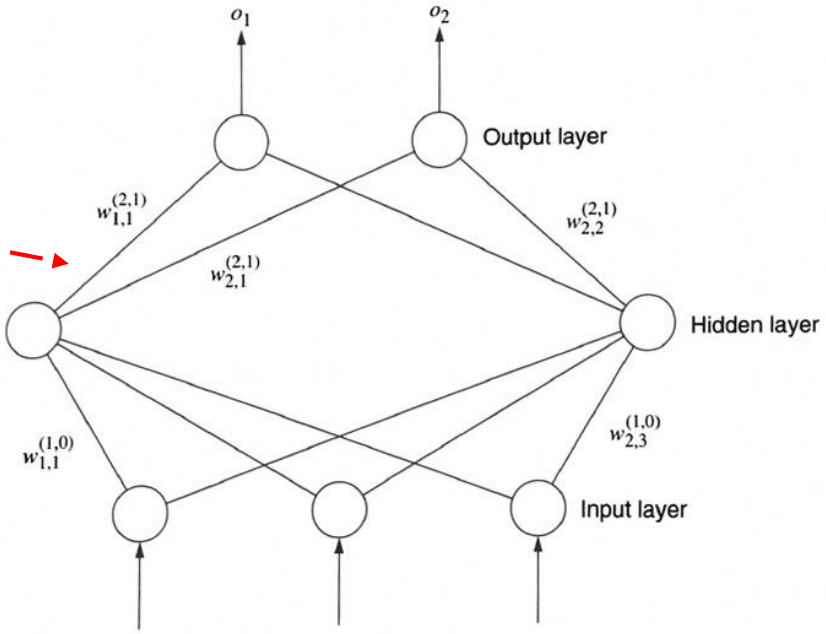
- The following two equations describe the weight changes

$$\Delta w_{k,j}^{(2,1)} = \left(\frac{-\partial E}{\partial w_{k,j}^{(2,1)}} \right) \qquad \Delta w_{j,i}^{(1,0)} = \left(\frac{-\partial E}{\partial w_{j,i}^{(1,0)}} \right)$$

Backpropagation algorithm

Backpropagation

- The chain rule is used to calculate the weight changes $\Delta w_{k,j}^{(2,1)}$



$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2,1)}}$$

- Since $E = \sum_k (d_k - o_k)^2$

$$\frac{\partial E}{\partial o_k} = -2(d_k - o_k)$$

$$\frac{\partial E}{\partial w_{k,j}^{(2,1)}} = -2(d_k - o_k)\mathcal{S}'(net_k^{(2)})x_j^{(1)}$$

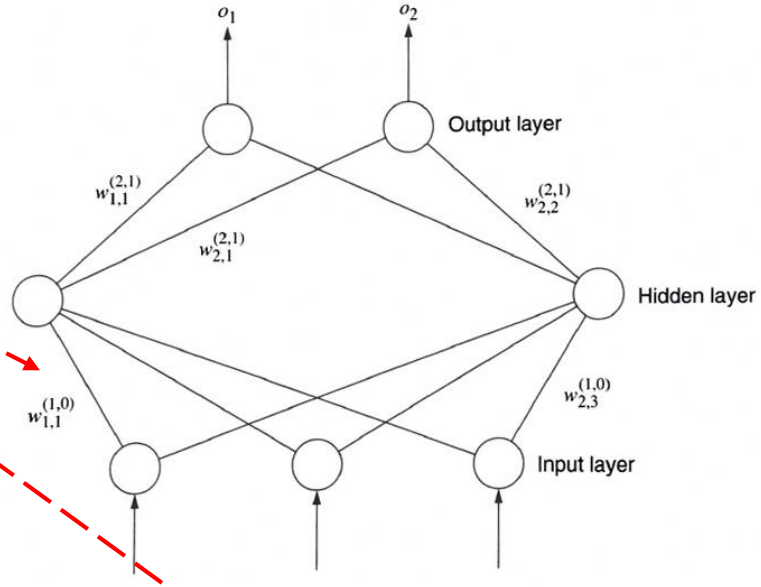
- Since $o_k = \mathcal{S}(net_k^{(2)})$, $\partial o_k / \partial net_k^{(2)} = \mathcal{S}'(net_k^{(2)})$

- Since $net_k^{(2)} = \sum_j w_{k,j}^{(2,1)} \times x_j^{(1)}$, $\partial net_k^{(2)} / \partial w_{k,j}^{(2,1)} = x_j^{(1)}$

Backpropagation algorithm

Backpropagation

- The chain rule is used to calculate the weight changes $\Delta w_{j,i}^{(1,0)}$



$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} \frac{\partial net_j^{(1)}}{\partial w_{j,i}^{(1,0)}}$$

$$\frac{\partial E}{\partial o_k} = -2(d_k - o_k)$$

$$\partial o_k / \partial net_k^{(2)} = S'(net_k^{(2)})$$

$$\frac{\partial net_k^{(2)}}{\partial x_j^{(1)}} = w_{k,j}^{(2,1)} \qquad \frac{\partial x_j^{(1)}}{\partial net_j^{(1)}} = S'\left(net_j^{(1)}\right) \qquad \frac{\partial E}{\partial w_{j,i}^{(1,0)}} = x_i$$

$$\frac{\partial E}{\partial w_{j,i}^{(1,0)}} = \sum_{k=1}^{K} \left\{ -2(d_k - o_k) S'\left(net_k^{(2)}\right) w_{k,j}^{(2,1)} S'\left(net_j^{(1)}\right) x_i \right\}$$

Backpropagation algorithm

Algorithm Backpropagation;
Start with randomly chosen weights;
while MSE is unsatisfactory
and computational bounds are not exceeded, **do**
for each input pattern x_p, $1 \le p \le P$,
Compute hidden node inputs $(net_{p,j}^{(1)})$;
Compute hidden node outputs $(x_{p,j}^{(1)})$;
Compute inputs to the output nodes $(net_{p,k}^{(2)})$;
Compute the network outputs $(o_{p,k})$;
Compute the error between $o_{p,k}$ and desired output $d_{p,k}$;
Modify the weights between hidden and output nodes:

$$\Delta w_{k,j}^{(2,1)} = \eta (d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) x_{p,j}^{(1)}$$

Modify the weights between input and hidden nodes:

$$\Delta w_{j,i}^{(1,0)} = \eta \sum_k \left((d_{p,k} - o_{p,k}) \mathcal{S}'(net_{p,k}^{(2)}) w_{k,j}^{(2,1)} \right) \mathcal{S}'(net_{p,j}^{(1)}) x_{p,i}$$

end-for
end-while.

Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Initialization of the weights

- Training is generally started with randomly chosen initial weight values.
- Typically, the weights chosen are small (between -1.0 and 1.0, -0.5 to +0.5).
- Larger weight values may drive the output nodes to saturation.
- Initialization may bias the network to give much greater importance to inputs those with higher value.
- In this case, the weights in the hidden layers can be taken the same.

Initialization of the weights

- The following equation can be used to initialize the weights between input layer and first hidden layer.

$$w_{j,i}^{(1,0)} = \pm \frac{1}{2P} \sum_{p=1}^{P} \frac{1}{|x_i|}$$

- The following equation can be used to initialize the weights between hidden layers and output layer.

$$w_{k,j}^{(2,1)} = \pm \frac{1}{2P} \sum_{p=1}^{P} \frac{1}{S(\sum w_{j,i}^{(1,0)} x_i)}$$

Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Frequency of weight updates

- There are two approaches to learning;
  - In "per-pattern" learning: weights are changed after every sample presentation.
  - In "per-epoch" (or "batch-mode") learning: weights are updated only after all samples are presented to the network.
- An *epoch* consists of such a presentation of the entire set of training samples.
- Calculated weight changes for each sample are accumulated together into a single change to occur at the end of each epoch.

$$\Delta w = \sum_{p=1}^{P} \Delta w_p$$

Frequency of weight updates

- In each case, training is continued until a reasonably low error is achieved, or until the maximum number of iterations allocated for training is exceeded.
- For some applications, the input-output patterns are presented on-line, hence batch-mode learning is not possible.
- Per-pattern training is more expensive then per-epoch training.
- For large applications, the amount of training time is large, requiring several days even on the fastest processors.

Frequency of weight updates

- The amount of training time can be reduced by exploiting parallelism in per-epoch training.
- Per-pattern training is not parallelizable in this manner.
- One problem in per-pattern learning is that the network may just learn to generate an output close to the desired output for the current pattern, without actually learning anything about the entire training set.

Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Choice of learning rate

- Weight vector changes in backpropagation are proportional to the negative gradient of the error.
- The relative changes that must occur in different weights when a training sample is presented.
- The exact magnitudes of the desired weight changes are not able to be decided.
- The magnitude change depends on the appropriate choice of the learning rate η.
- A large value of η will lead to rapid learning but the weight may then oscillate.
- Low values imply slow learning.
- This is typical of all gradient descent methods.

Choice of learning rate

- The right value of η will depend on the application.
- Values between 0.1 and 0.9 have been used in many applications.
- There have been several studies in the literature on the choice of η.
- In some formulations, each weight in the network is associated with its own learning rate.
- These weights are adapted separately from other weights.
- Each connection has its own learning rates.

Choice of learning rate

- A simple heuristic is to begin with a large value for η in the early iterations, and steadily decrease it.
- The changes of the weight vector must be small to reduce the likelihood of divergence or weight oscillations.
- This is based on the expectation that larger changes in error would occur earlier in the training.
- In this case, the error decreases more slowly in the later stages.
- Another heuristic is to increase η at every iteration that improves performance by some significant amount.
- Decrease η at every iteration that worsens performance by some significant amount.

Choice of learning rate

- The second derivative of the error measure provides information regarding the rate with which the first derivative changes.
- If the second derivative is low in magnitude, it is safe to assume a steady slope, and large steps can be taken.
- If the second derivative has high magnitude for a given choice of w, the first derivative may be changing significantly at w.
- Assumptions of steady slope are then incorrect, and a smaller choice of η may be appropriate.
- The main difficulty with this method is that a large amount of computation is required.

Content

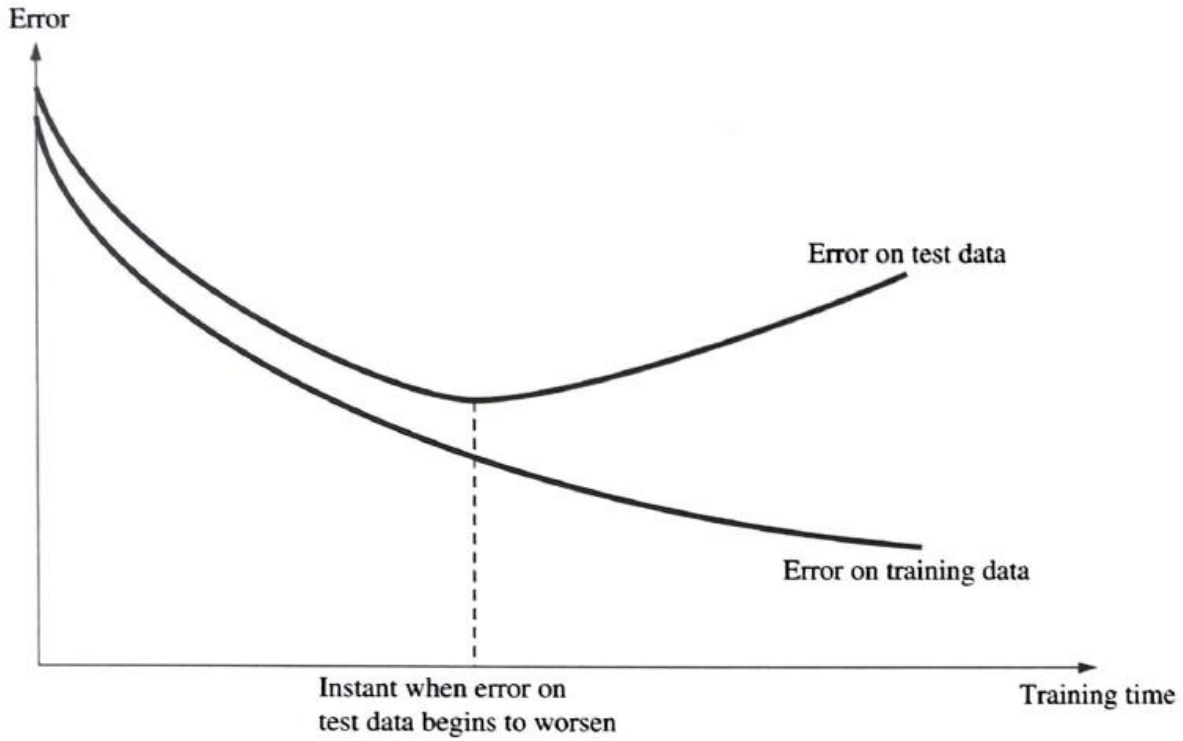
- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Generalizability

- For a large network, it is possible that repeated training iterations successively improve performance of the network on training data.
- But the resulting network may perform poorly on test data.
- This phenomenon is called overtraining.
- One solution is to constantly monitor the performance of the network on the test data.
- The weights should be adjusted only on the basis of the training set, but the error should be monitored on the test set.

Generalizability

- Training continues as long as the error on the test set continues to decrease.
- Training process is terminated if the error on the test set increases.
- Training may thus be terminated even if the network performance on the training set continues to improve.



Generalizability

- To eliminate random fluctuations, performance over the test set is monitored over several iterations.
- This method does *not* suggest using the test data for training: weight changes are computed solely on the basis of the network's performance on training data.
- With this stopping criterion, final weights do depend on the test data in an indirect manner.
- Since the weights are not obtained from the current test data, it is expected that the network will continue to perform well on future test data.

Generalizability

- A network with a large number of nodes is capable of memorizing the training set but may not generalize well.
- For this reason, networks of smaller sizes are preferred over larger networks.
- Thus, overtraining can be avoided by using networks with a small number of parameters.
- Injecting noise into the training set has been found to be a useful technique.
- This is especially the case when the size of the training set is small.
- Each training data point $(x_1, \ldots, x_n)$ is modified to a point $(x_1 \pm \alpha_1, x_2 \pm \alpha_2, \ldots, x_n \pm \alpha_n)$ where each α_i is a small randomly generated displacement.

Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Number of hidden layers and nodes

- Determining how many training samples are required for successful learning solved by trial and error.
- And, how large a neural network is required for a specific task is solved in practice by trial and error also.
- These problems are strictly dependent on the problem.
- With too few nodes, the network may not be powerful enough for a given learning task.
- With a large number of nodes, computation is too expensive.
- The network tends to perform poorly on new test samples,
- The network is not considered to have accomplished learning successfully.
- Neural learning is considered successful only if the system can perform well on test data.
- Capabilities of a neural network are emphasized to *generalize* from input training samples.

Number of hidden layers and nodes

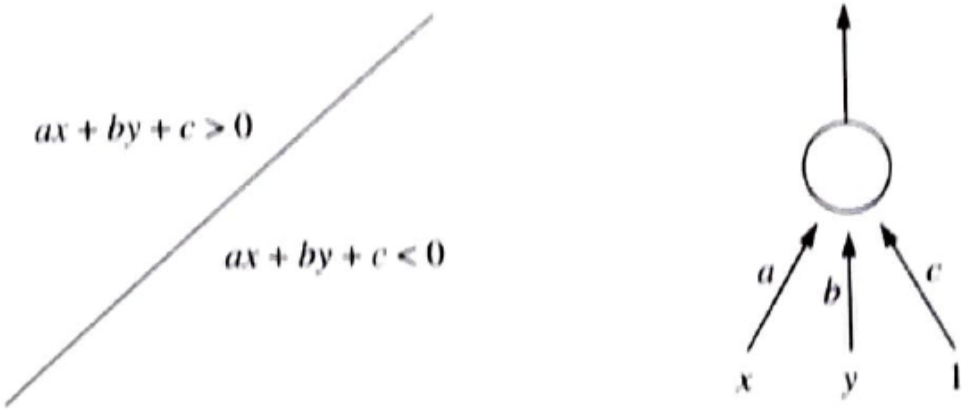
- Adaptive algorithms have been devised to obtain optimized number of neurons.
- Begin from a large network and repeatedly remove some nodes and links until network performance degrades to an unacceptable level.
- New nodes and weights can also be added, starting from a very small network and until the performance is satisfactory.
- The network is retrained at each intermediate state.

Number of hidden layers and nodes

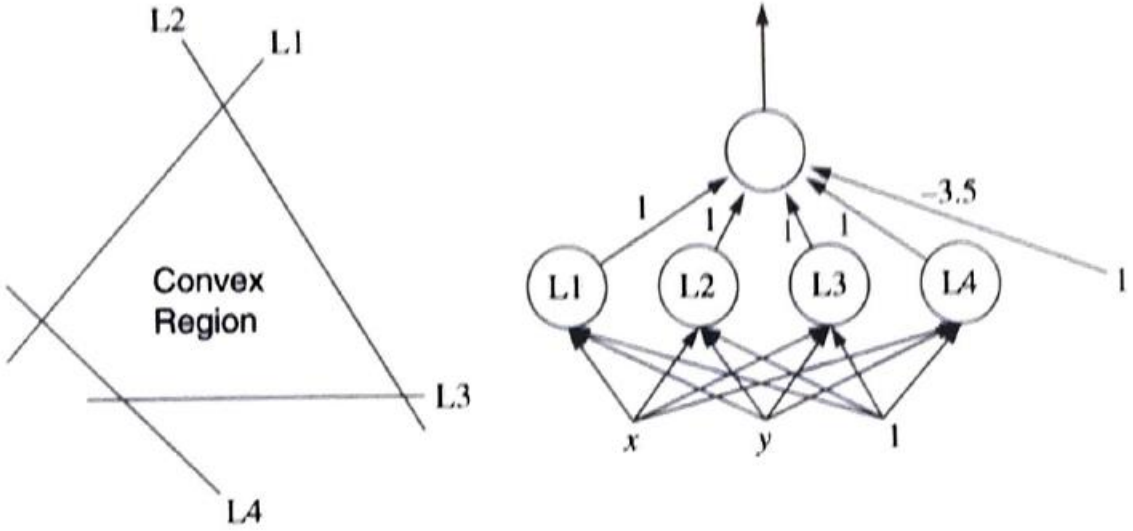
- For classification tasks with d input nodes, first hidden layer nodes often function as hyperplanes.
- That hyperplanes effectively partition d-dimensional space into various regions.
- Each node in the next layer represents a cluster of points that belong to the same class.
- All members in a set are assumed to belong to the same class, and instances of different classes are assigned to different sets.

Number of hidden layers and nodes

- Network with a single node using step function.

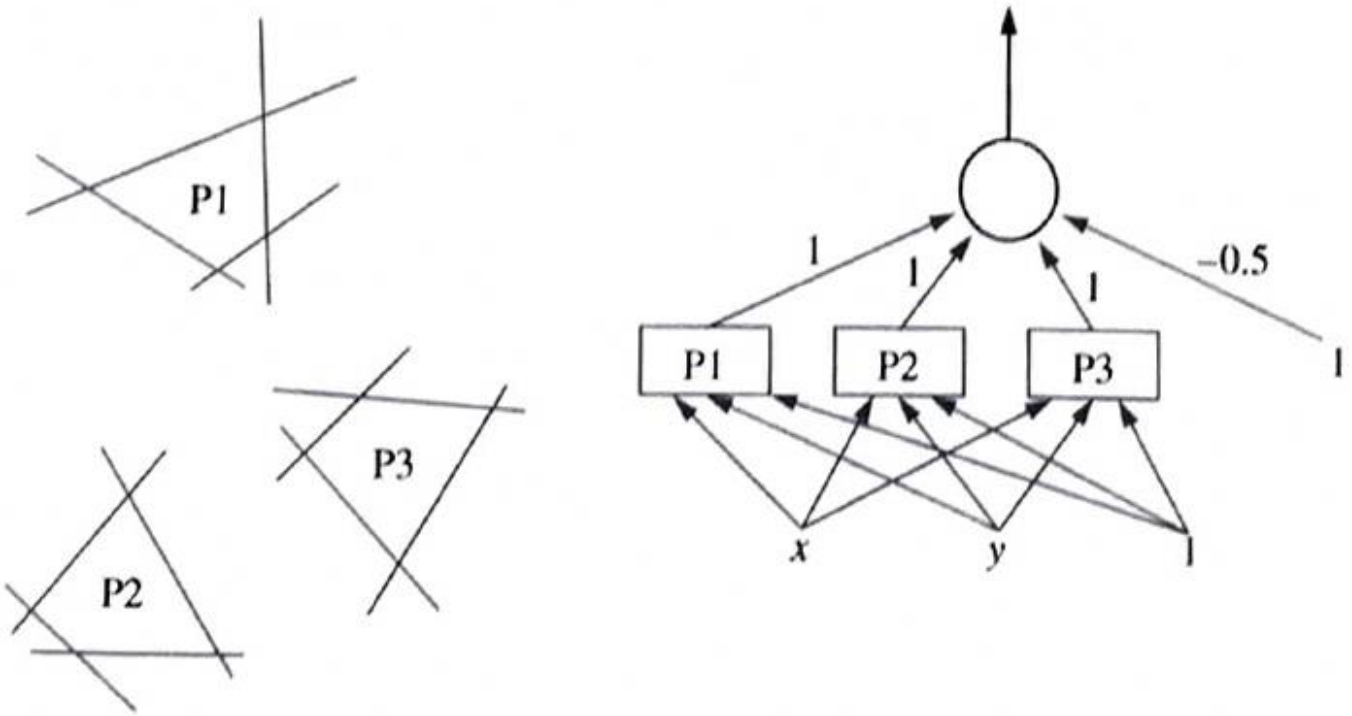


- One hidden layer network with convex region.
- Each node realizes one of the lines bounding the region.



Number of hidden layers and nodes

- Network with two hidden layers that realizes the union of three convex regions.
- Each box represents one hidden layer network.



Content

- Training
- Backpropagation algorithm
- Initialization of the weights
- Frequency of weight updates
- Choice of learning rate
- Generalizability
- Number of hidden layers and nodes
- Number of samples

Number of samples

- How many samples are needed for good training?
- At least five to ten times as many training samples as the number of weights to be trained.
- The equation is suggested on the basis of the desired accuracy on the test set:

$$P > \frac{|W|}{(1-a)}$$

P denotes the number of patterns,

$|W|$ denotes the number of weights to be trained,

a denotes the expected accuracy on the test set.

Number of samples

- Let a network contains 27 weights and the desired test set accuracy is 95% (a = 0.95).
- The analysis suggests that the size of the training set should be at least P > 27/0.05 = 540.
- The above is a necessary condition.
- A sufficient condition that ensures the desired performance is:

$$P \geq \frac{|W|}{(1-a)} \log \frac{n}{1-a}$$

where n is the number of nodes.