e-PGPathshala

Subject : Computer Science

Paper: Machine Learning

Module: Neural Networks – II

Module No: CS/ML/20

Quadrant I – e-text

Welcome to the e-PG Pathshala Lecture Series on Machine Learning. In the previous module we had discussed the basics of neural networks. In this module the use of neural networks for classification is discussed. We also outline the other types of neural networks.

Learning Objectives:

The learning objectives of this module are as follows:

- To understand the Back propagation method associated with neural networks
- To discuss the concept of Regularization
- To outline some applications of Neural Networks

20.1 Introduction

In the basic neuron model of the feedforward neural network, the inputs arrive through pre-synaptic connections and the synaptic influence is modeled using weights. The output of the neuron is a non-linear function of the weighted sum of inputs. The Back propagation algorithm learns in the same way as single perceptron which we discussed in the previous module. However it searches for weight values that minimize the total error of the network over the set of training examples (training set). Back-propagation consists of the repeated application of the following two passes namely the forward pass in which the network is activated on one example and the error of (each neuron of) the output layer is computed. This is followed by the backward pass where the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

20.1.1 Error Back-Propagation

Most of the training algorithms use iterative procedure for error function minimization. Each iteration involves two steps namely evaluation of derivatives of the error function with respect to the weights and the propagation of the error backwards in order to evaluate derivatives. It then uses the derivatives to compute the adjustments to be made to the weights.

20.2 Back-propagation Terminology

The goal of the back-propagation network is to find an efficient technique for evaluating gradient ∇ of an error function *E(w) for a* feed-forward neural network. Back-propagation provides a computationally efficient method for evaluating such derivatives. In subsequent stages the weights are adjusted using the derivatives. This adjustment is achieved using a local message passing scheme where information is passed forwards and backwards alternately. The basic neural network structure is given in Figure 20.1

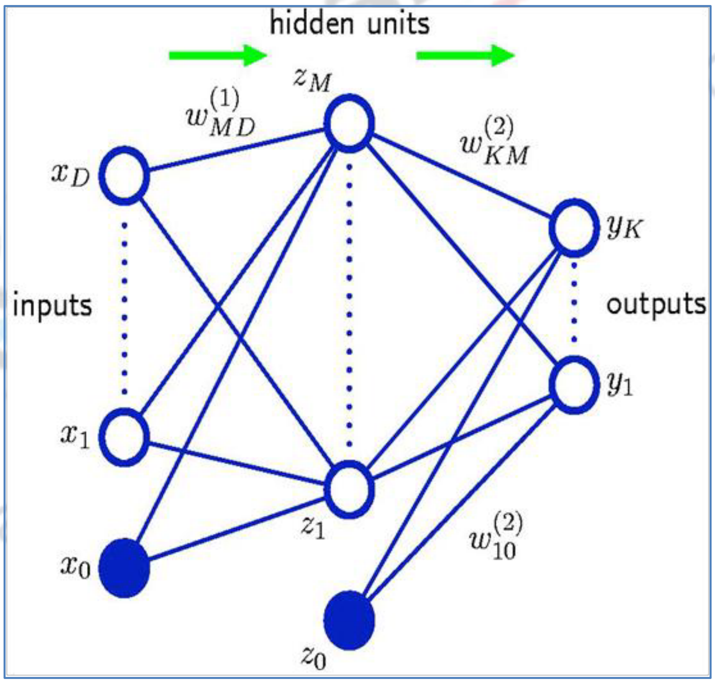


Figure 20.1 Basic Structure of the Neural Network

20.2.1 Overview of Back-prop algorithm

We start the process by randomly choosing initial *weights for the network. Then we feed in a* sample and get the result. This process is similar to the Feed-Forward network. Now we calculate the error for each node starting from the last stage to the input node that is we propagate the error backwards and hence the name Back-Propagation. Now we update the weights accordingly so as to reduce the error. We repeat the complete process *with other samples until* *the* target output converges. The tackling of division of errors requires some calculus.

20.2.2 Evaluation of Derivative Sum of Squares Error E_n with respect to a weight w_{ji}

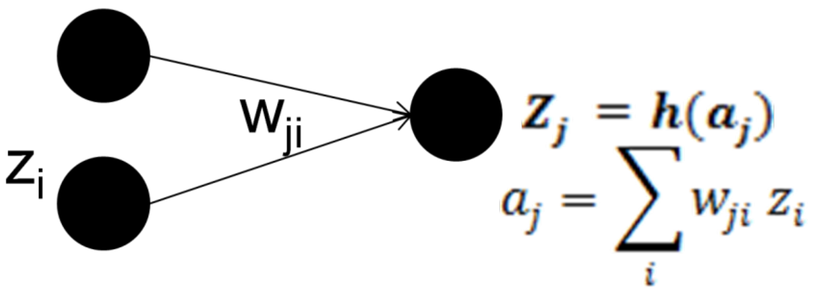


Figure 20.2 The weight factor

Figure 20.2 shows sum of the ith inputs affecting the jth node, a_j the activation function outputting the function Z_j. Here 'i' is for input unit and 'j' is for hidden unit. By chain rule for partial derivatives we get the following derivative

$$\boxed{\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}}$$

where we define δ_j as the derivative of an arbitrary differentiable Error function E_n with respect to the function a_j.

$$\boxed{\delta_j \equiv \frac{\partial E_n}{\partial a_j}} \qquad \boxed{a_j = \sum_i w_{ji} z_i}$$

Now from Figure 20.2 we can derive the following

$$\boxed{\frac{\partial a_j}{\partial w_{ji}} = z_i}$$

And by substituting the above in error derivative we get

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

Thus the required derivative

$$\frac{\partial E_n}{\partial w_{ji}}$$

is obtained by multiplying value of δ for the unit at output end of weight by value of z for unit at input end of weight. However we need to determine how to calculate

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

20.3 Error Back-propagation Algorithm

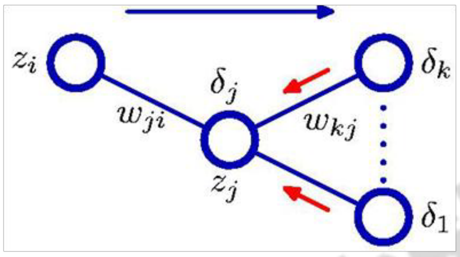


Figure 20.3 The Neuron Structure

Back-propagation Formula

$$\delta_j = h'(a_j)\sum_k w_{kj}\,\delta_k$$

The value of δ for a particular hidden unit can be obtained by propagating the δ's backward from units higher-up in the network. The steps of the algorithm are given below

1. Apply input vector x_n to network and forward propagate through network using

   $$a_j = \sum_i w_{ji}\,z_i$$

   and $z_j = h(a_j)$. Note that one or more of the variables z_i in the sum could be an input, and similarly, the unit j in Z_j could be an output.

2. Evaluate δ_k for all output units using $\delta_k = y_k - t_k$

3. Back-propagate the δ's using to obtain δ_j for each hidden unit

   $$\delta_j = h'(a_j)\sum_k w_{kj}\,\delta_j$$

We use the following to evaluate required derivatives

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

4. The weights of hidden layer to output layer are modified using the formula $\Delta w_{kj}^{n} = l_r\ \delta_k z_k + \Delta w_{kj}^{(n-1)}\mu$ and input layer to hidden layer as $\Delta w_{ji}^{n} = l_r\ \delta_j z_j + \Delta w_j^{(n-1)}\mu$ where l_r is the learning rate and μ is the momentum factor.

   Backpropagation in multi-layer neural networks is vulnerable to overfitting to the training set though this essentially means that the erroe with respect to the training set gradually decreases, the error with respect to the test set begins to increase. An epoch is training your network on each item of the set once. The number of epochs depends on the learning rate and the momentum factor. Learning rate decides whether the network learns the function slower or faster. Momentum factor helps in accelerating the learning process and also prohibits the network to settle sooner than desirable. With proper learning rate and momentum factor the learning can be made faster but at the same time accurate.

20.4 Simple Example: Forward and Backward Propagation

Consider a two-layer with a sum-of-squares error, where the output units have linear activation functions, and *the hidden units have* logistic sigmoid activation functions and t_k is the expected output of the K^th neuron. This model is taken for simplicity and practical importance as most network uses this type of network.

- For each input in training set, the forward propagation is given as:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \qquad y_k = \sum_{j=0}^{M} w_{kj}^{(2)} z_j$$

$$z_j = \tanh(a_j)$$

Here the output differences can be specified as $\delta_k = y_k - t_k$

Now the backward propagation (δs for the hidden units) is given as:

$$\delta_j = (1 - z_j^2) \sum_{k=1}^{K} w_{kj} \delta_k$$

$$\boxed{\delta_j = h'(a_j) \sum_k w_{kj}\, \delta_k}$$

h'(a) = 1-h(a)^2

Derivatives w.r.t first layer and second layer weights is given as

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \qquad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

Now we update the weights after each iteration using

$w^{(t+1)} = w^{(t)} + \eta \Delta E_n(w^{(t)})$

20.5 Strengths and drawbacks of Back-Propagation

The main strength of the back-propagation method is the great representation power where it is possible to learn many functions using back propagation learning. Back propagation learning has wide applicability since it requires only a good set of training samples and no prior knowledge or deep understanding of the domain is needed. Therefore this method can be used for ill structured problems. The method exhibits graceful degradation in the presence of missing data and is tolerant to noise. It is relatively easy to implement the central component of the learning algorithm. It exhibits good generalization power and gives fairly accurate results for hitherto unseen data.

In spite of many strengths, the back propagation method has some drawbacks. The main weakness is that it cannot avoid the local minimum error state. Not every function that can be represented can be learnt. Other disadvantages include non-termination or extremely slow convergence, possibility of overfitting, difficulty in proper selection of initial weights etc..

20.6 Network parameters selection

A good and effective neural network model is not only dependent the actual learning algorithms. The selection of the appropriate network parameters is an important aspect of the neural network approach. The learning accuracy, speed and effective generalization of the neural network are highly dependent of the set of learning parameters. The following are some of the aspects that affect the performance of the model:

- The Initial weights are generally chosen at random but for hidden layer we need to normalize the weights.

- An important aspect is the choice of training samples. They must be good representatives of problem space.
- Again the number of training samples is an important consideration especially to achieve good generalization. A general rule of thumb is for ‘w’ weights and ‘e’ desired error rate we need at least w/e samples.
- The number of hidden layers that have to be included is very much dependent on the nature of the problem.
- The number of hidden units per layer usually follows the heuristic that for n input units we need 2n hidden units for binary data and >>2n for real data.

Most of the above mentioned parameters are determined empirically.

20.6 Regularization

In machine learning and statistics regularization is used for model selection. This concept introduces additional information to prevent over-fitting or in some cases to solve ill-posed problems. Regularization penalizes models with extreme parameters, e.g., places restrictions for smoothness and bounds on the vector space form. Theoretically attempts to impose Occam’s razor on the solution by simplifying the model wherever feasible. From a Bayesian point of view, regularization corresponds to imposition of prior distributions on model parameters. The simplest form of regularization is the least-squares method. Now let us discuss different techniques used for regularization.

20.6.1 Regularization by determining number of hidden units

In general the number of input and output units in a neural network is determined by the dimensionality of the data set. However the number of hidden units M is a free parameter and controls the number of parameters that is the weights and biases in the network. Regularization is possible where the number of hidden units M can be adjusted to achieve the best predictive performance. One possible approach to adjust this parameter is to get the maximum likelihood estimate of M that gives the best generalization performance.

20.6.2 Regularization using Simple Weight Decay

The generalization error is not a simple function of M due to presence of local minima. There is a need to control the network complexity so as to avoid over-fitting and achieve better generalization. One method is to have a relatively large number of hidden units (M) and restrict the complexity by the addition of a regularization term. The simplest regularizer can be achieved by using the concept of weight decay $\tilde{E}(w) = E(w) + \lambda/2\ w^T w$. The model complexity is determined by the choice of the regularization coefficient λ .

20.6.3 Regularization using Early Stopping

The simple weight decay affects the scaling nature of the network. Another alternate method is to stop the process early when the error reaches an acceptable value determined by a validation set. In general the error decreases at first, and then increases as the network starts to over-fit. The training process can be stopped at the point of smallest error with validation data.

20.7 Desirable Invariance Property of Regularizer

In an ideal learning scenario, the predictions learnt should be unchanged, or invariant, even when the input variables undergo transformation. For example, in the classification of handwritten digits, an object should be classified in the same way irrespective of its position within the image (translation invariance) or of its size (scale invariance). There are basically four approaches for learning invariance, by artificially augmenting the training set using duplicates of the training patterns, by adding a regularization term to the error function, by extracting features from the input that are invariant during pre-processing and by integrating the invariance properties into the structure of the neural network.

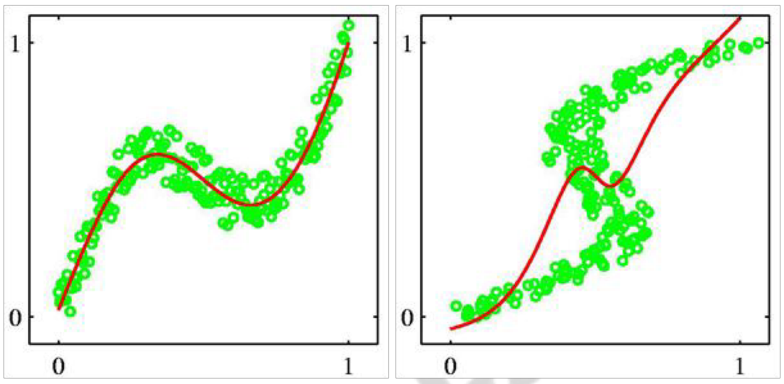
20.8 Mixture Density Networks

In general using the Bayesian approach the posterior conditional probability of the output given input is determined given the prior probability of the output and the likelihood of the input given the output. Mixture density network is a general framework for modelling the posterior conditional probability distributions. In general the goal of supervised learning is to model the conditional distribution p(t/x), that is finding the probability of t given the input x. In reality, the distribution of the problem dataset can be multimodal particularly in inverse problems. In such cases the use of the Gaussian assumption that is generally used for conditional probability distribution will not give good results. However in the case of regression the conditional probability p(t/x) is typically assumed to be Gaussian that is i.e., $p(t/x) = N(t/y(x,w),b^{-1})$

20.8.1 Data Set for Forward and Inverse Problems

In general least squares corresponds to Maximum likelihood under a Gaussian assumption and leads to poor results for highly non-Gaussian inverse problem. This is because the average of the correct output values is not necessarily the correct solution. Therefore we need to model conditional probability distributions by using a mixture model for p(t|x) that is using a linear combination of a kernel function which we will discuss later.

Forward problem data set is given in Figure 20.4 (a). In this case x is sampled uniformly over (0,1) to give values $\{x_n\}$. The target t_n is obtained by the function $x_n+0.3\sin(2\pi x_n)$. We then add noise over (-0.1,0.1). The red curve shown in Figure 20.4 (a) is the result of fitting a two-layer neural network by minimizing SSE (Sum of Squared Error). The inverse problem obtained by reversing x and t is shown in Figure 20.4 (b) and we can see that it is a very poor fit to the data.



(a) (b)

Figure 20.4 Forward and Inverse Data

20.8.2 A Mixture Density

Let us discuss a generic mixture model which is a linear combination of the kernel function with K components given below:

$$p(t|x) = \sum_{k=1}^{K} \pi(x) N(t|\mu_k(x), \sigma_k^2(x))$$

Here a Gaussian mixture is used but any distribution could have been used. The continuous variables use Gaussian distribution while the binary variables use Bernoulli distribution. The parameters that are to be estimated are π_k (x), μ_k (x) and $\sigma_k^2(x)$*. These are governed* by the outputs of a neural network with x as input. Note that mixing coefficient π is dependent on x and sums to 1 for each x. In addition the mean and variance are also dependent on x. The weights are determined by minimizing the negative logarithm of the likelihood, where we try to minimize the error and consequently maximize the likelihood. A single network predicts the parameters of all the component densities. If there are L components in mixture model and t has K components then there are L output units and there are (K+2)L outputs instead of usual K.

20.8.3 Structure of the Mixture Density Network

The network represents the general conditional probability densities p(t|x) by considering a parametric mixture model. As already explained it takes x as input and provides the parameters of the distribution as output. In other words it determined the conditional distribution of p(t|x) given x as the input vector (Figure 20.5).

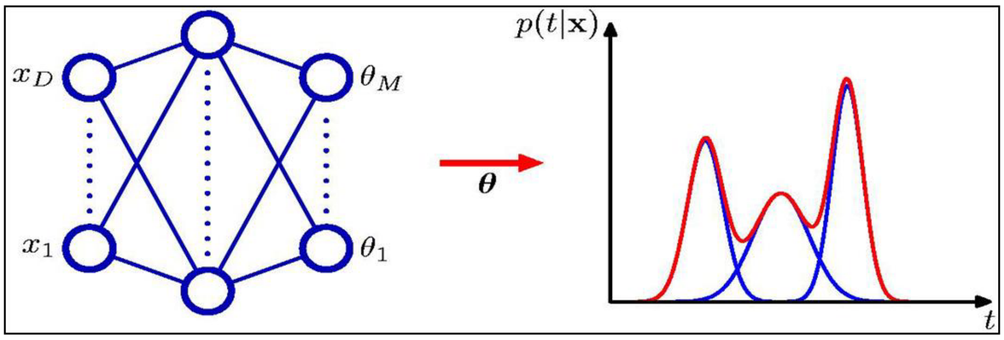


Figure 20.5 Mixture Density Network

20.9 Why Bayesian?

When we construct complex models, they fit the data better but does not generalize well. Therefore what model should we use? Should we use linear with two free parameters, quadratic with three, cubic with four?. As already discussed Occam's razor says that unnecessarily complex models should not be preferred to simpler ones when simpler ones fit the data fairly well. Neural networks are popular but do not have an objective grounding. Therefore the Bayesian approach can be used to compare different models of the network with different number of hidden units for example.

20.9.1 Classical and Bayesian Neural Networks

Conventional training methods for multilayer perceptron such as back propagation can be interpreted statistically as variation on maximum likelihood estimation. As we know neural network training allows the prediction of future unseen cases where we are given only the input. In traditional neural network training we normally come up with a single set of weights that can be used to make predictions. However when we use Bayesian methods we get a posterior distribution over the network weights. In some cases classical neural networks use maximum likelihood for estimating the network parameters such as weights and biases. The regularized maximum likelihood is MAP the maximum a posteriori method where the regularizer is the logarithm of prior parameter distribution. The Bayesian treatment marginalizes over the distribution of parameters in order to make prediction.

20.9.2 Need for Approximation in Bayesian treatment

Generally in the case of simple linear regression problem, under assumption of Gaussian noise, we assume that the posterior distribution is Gaussian and can be evaluated exactly. In this case the predictive distribution is found in closed form. However in the multilayered network, the network function has a highly nonlinear dependence on the network parameter values and therefore determination of exact distribution using Bayesian method is not possible. The log of the posterior distribution is now non-convex implying multiple local minima of error function. Therefore approximate methods are necessary.

20.9.2 Two Approaches To Bayesian Treatment

There are two approaches to determining the posterior distribution. One method is the use of variational inference using a factorized Gaussian approximation to the posterior distribution and the other method is the use of a full covariance Gaussian.

Here we discuss Laplace approximation which is the most complete treatment. This involves two approximations, where we replace the posterior by a Gaussian centred at a mode of true posterior. The covariance of Gaussian is small so that network function is approximately linear with respect to the parameters over region of parameter space for which the posterior probability is significantly nonzero. Using the above framework we can compare alternative model with different number of hidden units.

20.10 ANN applications

Some of the typical types of applications and associated examples that use ANNs are listed below:

Pattern Recognition and Forecasting: An important application of NNs is pattern recognition. A simple implementation of pattern recognition is the use of feed forward network that has been appropriately trained. The neural network is able to recognize unknown patterns by giving as output that is least different from the unknown pattern. Neural networks are well suited for identifying patterns or trends in data and hence can be used for prediction or forecasting needs. Examples include simple pattern recognition such as reading zip codes, industrial process control, data validation and forecasting such as sales forecasting, risk management, crop yield forecasting and prediction of customer behavior. Successful use of ANNs include recognition of speakers in communication, and image processing applications including diagnosis of hepatitis, undersea mine detection, texture analysis, three dimensional object recognition, hand written character recognition, face recognition etc..

ANNs are also used in other types of applications such as signal filtering including reduction of radio noise, data segmentation including detection of seismic onsets, data compression including TV image transmission, database mining for marketing and finance analysis and adaptive control such as vehicle guidance.

20.10.1 ANN vehicle steering application

We will discuss in detail the autonomous vehicle as an example of a system that uses an ANN to steer a vehicle (that is vehicle guidance) on a public high way. One implementation uses as input a 30X32 pixel intensities (has 960 input units) of the sensor input retina (Figure 20.6). The network model uses 4 hidden units and has 30 output units. The output is the vehicle steering direction where different output units are for different directions of movement.

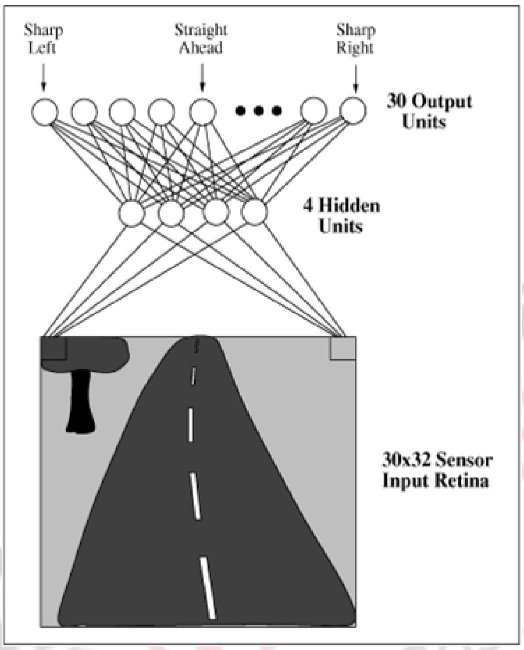


Figure 20.6 ANN for Vehicle Steering Application

20.10.2 ANN Character recognition

Optical character recognition systems were among the first commercial applications of neural networks. We discuss the use of a multilayer feed-forward network for printed character recognition. The task is to recognize digits from 0 to 9. Each digit is represented in our example by a 5X9 bit map, however commercial applications normally use 16X16 bit map (Figure 20.7).

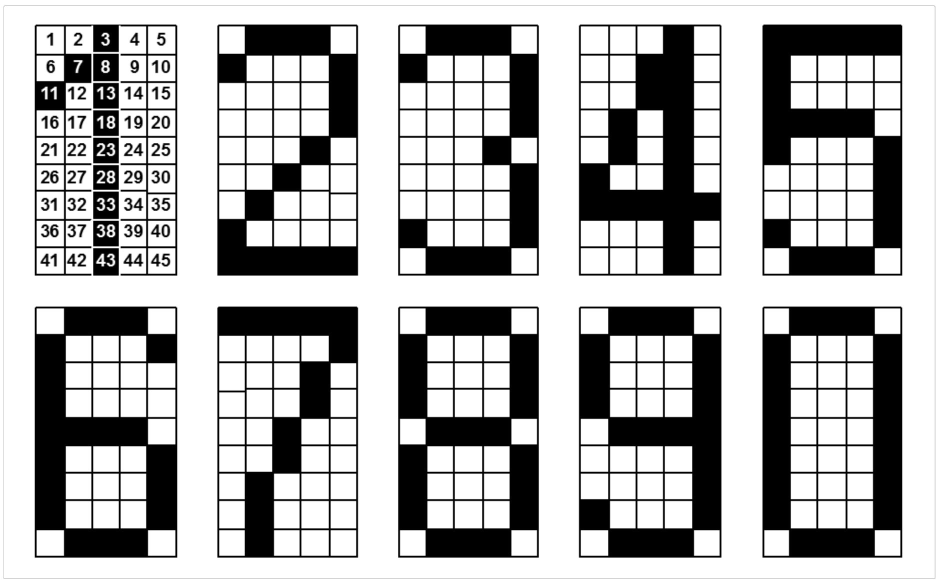


Figure 20.7 Bit maps for digit recognition

The architecture of the neural network is discussed below. The number of neurons in the input layer is equal to the number of pixels in the bit map. Hence in our example there are 45 input neurons. The output layer has 10 neurons that is one neuron for each digit to be recognized. The number of hidden layer neurons is varied and decided through experiments (Figure 20.8).

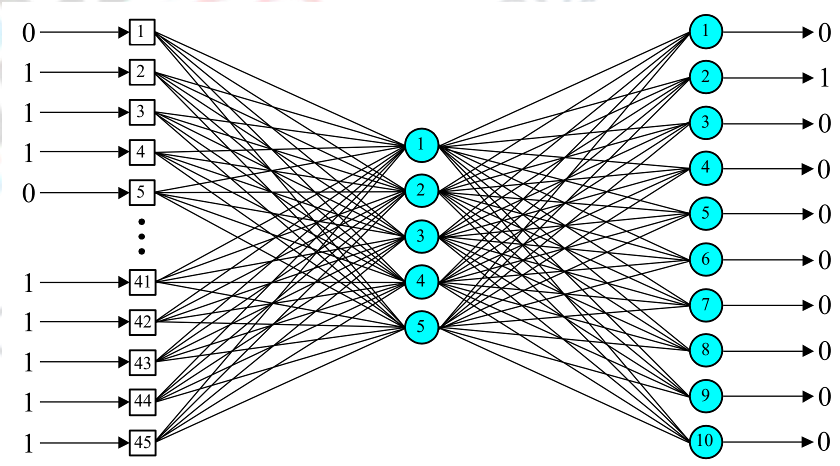


Figure 20.8 Neural Network for Printed Digit Recognition

The performance of the neural network depends on the goodness of the examples used to train it. In the case digit recognition, learning can be improved by feeding the network with "noisy" examples of digits from 0 to 9. A test set has to be strictly independent from the training examples. To test the character recognition network, examples that include "noise" that is with distortion of the input patterns are used as training samples. The performance of the printed digit recognition network was evaluated with 1000 test examples (100 for each digit to be recognized). The learning curves for a three layer neural network with varying number of hidden layer neurons are shown in Figure 20.9. As you can see the error decreases with lesser number of epochs for neural networks with twenty hidden neurons. The performance evaluation of the neural network with “noisy” samples are shown in Figure 20.10 where we see that the recognition error is least when there are twenty hidden neutrons and the noise level of the training samples is higher.

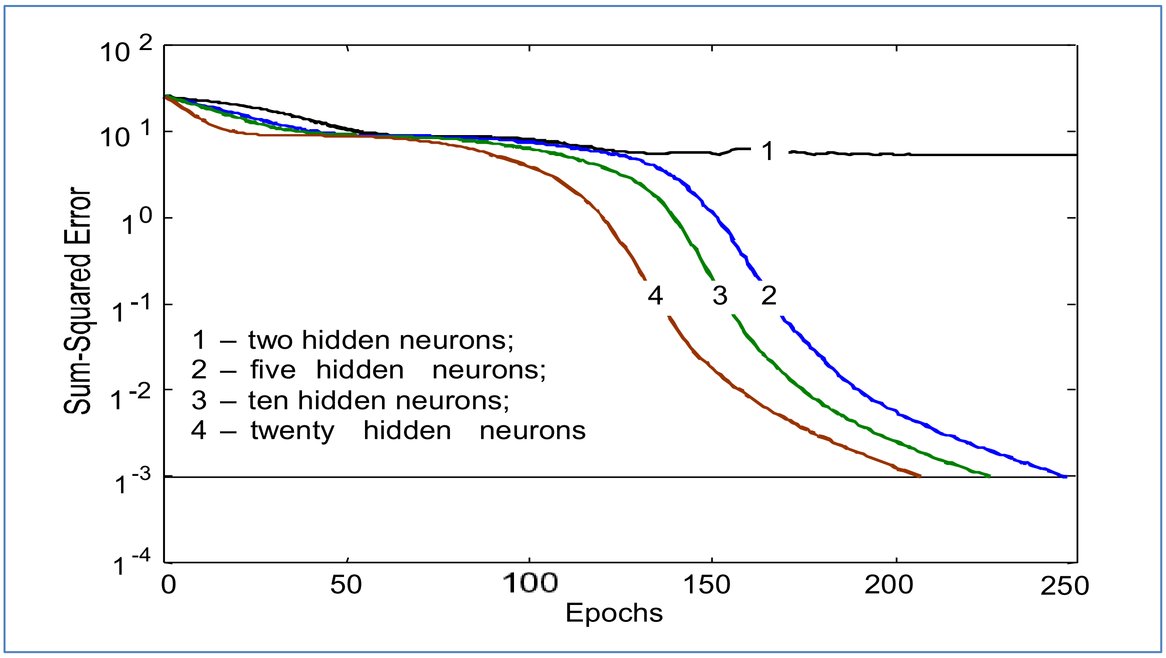


Figure 20.9 Learning Curves of Digit Recognition Three-layer Neural Networks

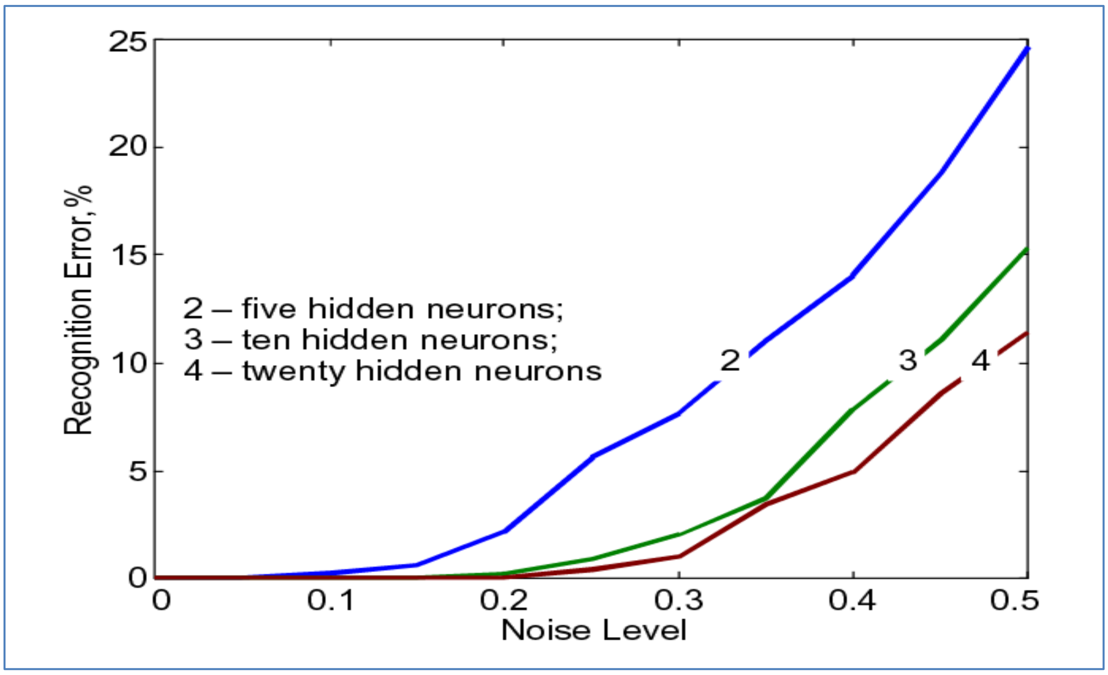


Figure 20.10 Performance Evaluation of ANN with noisy samples

Source of the slides: www2.cs.siu.edu/~rahimi/cs437/slides/lec14.ppt

20.11 Recent Advances in Neural Network

Some of the recent advances of neural network include the integration of fuzzy logic into neural networks. Then there is the pulsed neural networks where the communication in the network is through pulses, and the timing of the pulses is used to transmit information and perform computation. There is also a trend to design hardware specialized for neural networks where the neural networks are directly encoded onto chips or specialized analog devices to achieve increase in speed

Summary

- Explained the Back-propagation learning method for a Feed-Forward network.
- Discussed about reducing over-fitting of neural network and Bayesian neural networks.
- Discussed a real time application and a common application of neural network.