UNIT 7 APPLICATIONS OF MLP

7.1 INTRODUCTION

In the previous units of neural networks, we discussed the general introduction to neural networks. This involved the model of neuron, the McCulloch-Pits neuron, network topologies, learning methods, as well as properties and applications of neural networks. In this unit, we shall start with function approximation in Sec. 7.2. Also, we shall discuss learning and generalization in Sec. 7.3 and selection of architecture in Sec. 7.4.

Objectives

After studying this unit you should be able to:

- define the function approximation in neural networks;
- apply generalization in the context of neural networks;
- find the selection of architecture for neural networks.

7.2 FUNCTION APPROXIMATION

Most applications of neural networks utilize the function approximation capability. These include associative memory, modeling, signal processing, pattern recognition, regression and prediction and system identification. Image restoration is also an important function approximation problem. For this, the supervised learning is used in the neural networks. The input and output layers needed in the neural networks are defined on the basis of the dimension of the input and output layer patterns. And the networks parameters are adjusted accordingly. A trained network represents the learned non-linear relation between the input-output pairs, and can generalize where an unknown input pattern is presented to the network.

On one hand, the capability of universal function approximation lays the theoretical foundation for the wide application of forward neural networks, on the other hand, recurrent neural networks using the sigmoid activation function are Turing equivalent and can compute whatever function any digital computer.

Some aspects of the representation capability of forward neural networks are described below.

1. Boolean Function Approximation

To represent logic or Boolean functions, the forward neural networks (FNNs) with binary neurons are used binary neural networks, the input and output values for each neuron are represented either by Boolean variables (0 or 1) or by bipolar (– 1 or +1). For N independent Boolean variables, there are 2^N combinations of these variables.

This leads to a total of 2^{2^N} different Boolean functions of N variables. Two class can always be discriminated by a **linear threshold gate (LTG)**. Now using a network of LTG, any Boolean function becomes realizable.

The number of linearly separable dichotomies of k points in general position in R^n is found using the function counting theorem which essentially estimates the separating capability of an LTG. The function counting theorem is given below.

Theorem 1 (Function counting theorem): The number of linearly separable dichotomies of m points in general position in R^n is

$$C(k,n) = \begin{cases} 2\sum_{i=0}^{n} C(k-1,i), & k > n+1 \\ 2^k, & k \le n+1 \end{cases} \tag{1}$$

A set of k points in R^n is said to be in a general position if every subset of n or fewer points is linearly independent.

The total number of possible dichotomies of k points is 2^k. Under the assumption of 2^k equiprobable dichotomies, the probability of a single LTG with n inputs to separate k points in general position is given by

$$P(k,n) = \frac{C(k,n)}{2^k} = \begin{cases} 2^{1-k}\sum_{i=0}^{n} C(k-1,i), & k > n+1 \\ 1, & k \le n+1 \end{cases} \tag{2}$$

The fraction P(k,n) is said to be the probability of linear dichotomy. This relation is visualizes in Fig 1.

Here using Eqn. (2), we get

$$\begin{aligned} P &= 1 && \text{if } \frac{k}{n+1} \le 1 \\ P &\to 1, n \to \infty && \text{if } 1 < \frac{k}{n+1} < 2 \\ P &= 1/2 && \text{if } \frac{k}{n+1} = 2 \end{aligned}$$

Usually, $\frac{k}{n+1} = 2$ is used to characterize the statistical capability of a single LTG.

A three-layer $(N - 2^N - 1)$ feed forward LTG network can represent any Boolean function with N arguments. To realize an arbitrary function $f : R^N \to \{0,1\}$ defined on M arbitrary points in R^N, the lower bound for the number of hidden nodes is derived as $O\left(\frac{M}{N\log_2 \frac{M}{N}}\right)$ for $M \ge 3N$ and $N \to \infty$; for M points in general position, the lower bound is $\frac{M}{2N}$ when $N \to \infty$. Networks with two or more hidden layers are found to be potentially more size efficient than networks with a single hidden layer.

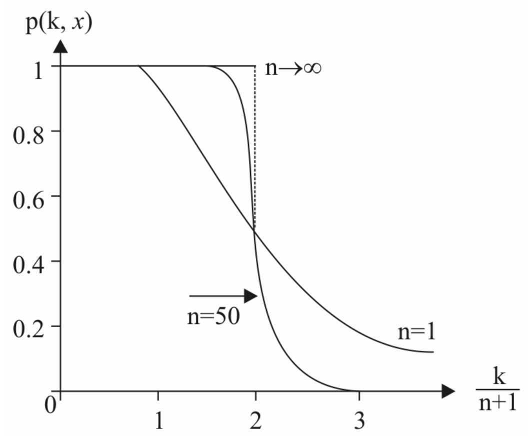

Fig. 1: The probability of linear dichotomy of k points in n dimensions

2. Linear Separability and Nonlinear Separability

Consider a set **X** and M pattern x_i of N dimensions, such that each belonging to one of two classes C_1 and C_2, then such a classification problem is said to be linearly separable. A single LTG can realize linearly separable dichotomy function, characterized by a linear separating surface (hyper plane)

$$\mathbf{w}^T\mathbf{x} + w_0 = 0 \tag{3}$$

where,

w is an N-dimensional vector, and w_0 is a bias toward the origin.

For a pattern,

if $\mathbf{w}^T\mathbf{x} + w_0 > 0$, the pattern belongs to the class C_1

and if $\mathbf{w}^T\mathbf{x} + w_0 < 0$, the pattern belongs to the class C_2.

Some examples of linearly separable classes and linearly inseparable classes in two-dimensional space are shown in Fig. 2, where dots and circles denote patterns of different classes. Here, it can also be noted that these two classes are linearly separable with $x_1 - x_2 = 0$ as the delimiter.

A linearly inseparable dichotomy can become nonlinearly separable. A dichotomy $\{C_1, C_2\}$ of set X is said to be f-separable if there exists a mapping $f : R^N \to R^{N'}$ that satisfies a separating surface

$$\mathbf{w}^T f(\mathbf{x}) = 0 \tag{4}$$

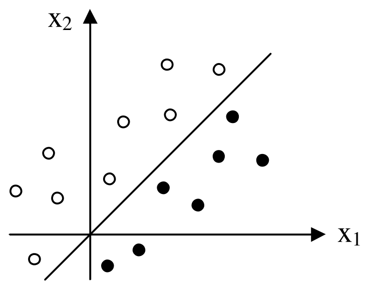

Fig. 2: Linearly separable classification in 2-D

Such that $\mathbf{w}^T f(\mathbf{x}) > 0$ if $\mathbf{x} \in C_1$ and $\mathbf{w}^T f(\mathbf{x}) < 0$ if $\mathbf{x} \in C_2$. Here **w** is a N'-dimensional vector. As shown in Fig. 3, the two linearly inseparable dichotomies become f-separable. Also, Fig. 3 (a) is the XOR problem. It can be noted here that linearly inseparable classification in cases (a) and (b) become non-linearly separable where the separation surfaces are ellipses.

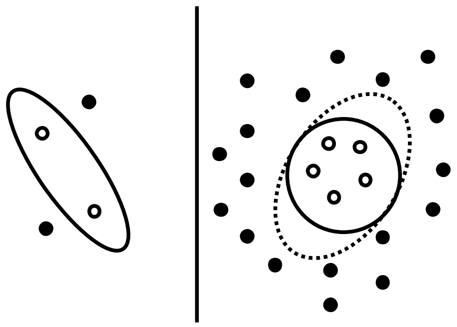
Fig. 3 (a and b): Linear inseparable classification in 2D

The nonlinearly separable problem can be realized by using a polynomial threshold gate (PTG), which changes the linear term in the LTG into high-order polynomials. The function counting theorem is also applicable to PTGs.

The function counting theorem still holds true if the set of k points is in general position in f-space, that is, the set of k points is in f-general position.

3. Continuous Function Approximation

A three-layer forward neural network with a sufficient number of hidden units can approximate any continuous function to any degree of accuracy. This is guaranteed by Kolmogorov’s theorem, which is stated below:

Theorem 2 (Kolmogorov): Any continuous real-valued function $F(x_1,\cdots,x_n)$ defined on $[0,1]^n$, $n \geq 2$, can be represented in the form

$$F(x_1,\cdots,x_n)=\sum_{j=1}^{2n+1} h_j\left(\sum_{i=1}^{n}\phi_{ij}(x_i)\right) \quad (5)$$

where h_j and ϕ_{ij} are continuous functions of one variable, and ϕ_{ij} are monotonically increasing functions independent of F.

According to Kolmogorov’s theorem, a continuous multivariate function on a compact set can be expressed using superpositions and compositions of a finite number of single-variable functions.

Based on Kolmogorov’s theorem, Hecht-Nielsen provided a theorem that is directly related to neural networks, and is stated below.

Theorem 3 (Hecht-Nielsen): Any continuous real-valued mapping $F:[0,1]^n \rightarrow R^m$ can be approximated to any degree of accuracy by an FNN with n input nodes, 2n+1 hidden units, and k output units.

In continuation to this, the Weierstrass theorem asserts that any continuous real-valued multivariate function can be approximated to any accuracy using a polynomial. The Stone-Weierstrass theorem is a generalization of the Weierstrass theorem, and is usually used for verifying a model’s approximation capability to dynamic systems, which is stated as below.

Theorem 4 (Stone-Weierstrass): Let F be a set of real continuous functions on a compact domain u of n dimensions. Let F satisfy the following criteria

1. **Algebraic closure:** F is closed under addition, multiplication, and scalar multiplication. That is, for any two $f_1, f_2 \in F,$ we have $f_1 f_2 \in F$ and $a_1 f_1 + a_2 f_2 \in F,$ where a_1 and a_2 are any real numbers.
2. **Separability on u:** for any two different points $x_1, x_2 \in u, x_1 \neq x_2,$ there exists $f \in F$ such that $f(x_1) \neq f(x_2).$
3. **Not constantly zero on u:** for each $x \in u,$ there exists $f \in F$ such that $f(x) \neq 0.$

Then F is a dense subset of c(u), the set of all continuous real-valued functions on u. In other words, for any $\varepsilon > 0$ any function $g \in C(u),$ there exists $f \in F$ such that $|g(x) - f(x)| < \varepsilon$ for any $x \in u.$

4. Universal approximation

Universal approximation to a given nonlinear functional under certain conditions can be realized by using the classical Volterra series or the Wiener series. The MLP is also a universal approximator and has a strong classification capability. The universal approximation capability of the MLP stems from the nonlinearities used in the nodes. A four-layer MLP can approximate any non-linear relation to any accuracy as long as the number of nodes in the hidden layers are sufficiently large. According to Kolmogorov's theorem, a four-layer MLP with N(2N + 1) nodes using continuously increasing non-linearities can compute any continuous function of N variables. In, a four-layer MLP with $\frac{n}{2}+3$ hidden neurons has been designed and it can represent M distinct examples, with negligibly small error. A four-layer MLP with $2\sqrt{(k+2)M}$ hidden neurons can learn any distinct examples with any arbitrarily small error, where k is the required number of output neurons.

The MLP is very efficient for function approximation in high-dimensional spaces. The error convergence rate of the MLP is independent of the input dimensionality, while conventional linear regression methods suffer from the curse of dimensionality, which results in a decrease of the convergence rate with an increase of the input dimensionality. The necessary number of MLP neurons for approximating a target function depends only upon the basic geometrical shape of the target function, and not on the dimensionality of the input space. Based on geometrical interpretation of the MLP, the minimal number of line segments or hyper planes that can construct the basic geometrical shape of the target function is suggested as the first trial for the number of extrema and the number of hidden nodes should be selected as the number of extrema.

Since both the three-layer and four-layer MLPs can be used as universal approximators, one may wonder as to which one is more effective. Usually, a four-layer network can approximate the target with fewer connection weights, but this may, however, introduce extra local minima. The geometrical interpretation of the MLP on the basis of the special geometrical shape of the activation function can also be found. For the target function with a flat surface located in the domain, a small four-layer MLP can generate better results.

5. Sigma-Pi Networks

The Sigma-Pi network is a generalization of the MLP. Unlike the MLP, the Sigma-Pi network uses product units as well as summation units to build higher-order terms. The BP learning rule can be applied to the learning of the network. The Sigma-Pi network is known to provide inherently more powerful mapping capabilities than first-order models such as the MLP. It is a universal approximator. However, the Sigma-Pi network has a combinatorial increase in the number of product terms and weights. This limits its applications.

Example 1: Solve the network to approximate the function $f(x) = 1 + \sin \pi x$ for $-1 \le x \le 1$, choosing initial weights and bias as the random numbers, using back propagation algorithm.

Solutions: Choosing for three layer, 1-2-1 network is one neuron in input layer, two neurons in hidden layer and one neuron in output layer, with the weighted structure given as

$$[W]^{\circ} = \begin{bmatrix} -0.25 \\ -0.40 \end{bmatrix}, \text{ bias } \phi^{(1)}(0) = \begin{bmatrix} -0.50 \\ -0.1 \end{bmatrix}$$

and $[V]^{\circ} = [0.1, \quad -0.2]$, bias $\phi^{(2)}(0) = \{0.5\}$

the architecture of this model is given in Fig. 4.

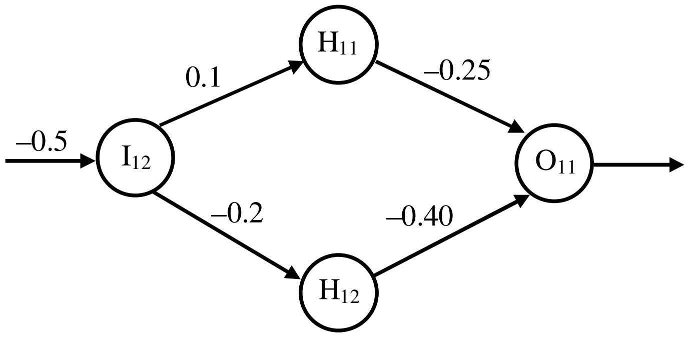

Fig. 4

Let the initial input be 1, then the output of the first hidden layer is given by

$$[O_1]_H = f^{(1)}\left([W]^{\circ} \cdot [O]^{(0)} + \phi^{(1)}\right)$$
$$= f^{(1)}\left(\begin{bmatrix} -0.25 \\ -0.40 \end{bmatrix}[1] + \begin{bmatrix} -0.5 \\ -0.1 \end{bmatrix}\right)$$
$$= f^{(1)}\begin{pmatrix} -0.75 \\ -0.5 \end{pmatrix}$$
$$= \begin{matrix} \dfrac{1}{1+e^{0.75}} \\ \dfrac{1}{1+e^{0.5}} \end{matrix}$$

$$[O_2]_H = f^{(2)}\left([V]^{\circ}[O_1]_H + \phi^{(2)}\right)$$
$$= f^{(2)}\left([0.1 \quad -0.2]\begin{bmatrix} 0.321 \\ 0.358 \end{bmatrix} + [0.5]\right)$$
$$= f^{(2)}(0.115) = 0.115 \quad (\text{selecting f as linear function})$$

$$\text{The error} = O_{TO} - [O_2]_H$$
$$= 1 + \sin \pi x - 0.115$$
$$= 0.835$$

Further $[\Delta W]$ and $[\Delta V]'$ can be found and accordingly modified weights can be calculated. The Bias is calculated as given below

$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta\, s^{(1)}$$

$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta\, s^{(2)}$$

where $s^{(1)} = F^{(1)} V^t s^{(2)}$ $\left[\text{Here } F'^{(1)} \text{ is a matrix of } f'^{(1)} \text{ for two rows of vector}\right]$

$s^{(2)} = -f'^{(2)} [O_2]_H$ and η is the learning rate.

The first iteration of backpropagation algorithm is complete here. The another iteration can be perform by choosing another input. The process can be continued to iterate until the difference between the target function and the network response reaches at some acceptable level.

Now try the following exercise.

E1) Solve the network to approximate the function $g(x) = 1 + \sin(\pi x/4)$ for $-2 \le x \le 2$, choosing initial weights and bias as the random numbers.

E2) In a 3-layer backpropagation network, if initial weights and biases are choosen as: $W = -1, \phi^{(1)} = 1, V = -2$, $\phi^{(2)} = 1$ and an input/target pair is given to be $(I = -1, t = 1)$, perform one iteration of backpropagation with $\eta = 1$.

E3) Show that a multi-layer network with linear transfer functions is equivalent to a single-layer linear network.

E4) Show that backpropagation reduces to the LMS algorithm for a single-layer linear network (ADALINE).

E5) Train a BP network to approximate the plot of the function $f(x) = \sin 2\pi x$ over the interval $0 \le x \le 1$.

E6) Train BP network so that it is capable of approximating the plot of the function $f = x^2 + y^2$ over the interval $x \in (0,1)$ and $y \in (0,1)$.

In the following section, we shall discuss generalization.

7.3 GENERALIZATION

According to the approximation of function, learning can be defined as a hyper surface reconstruction based on existing examples, while generalization is estimating the value on the hyper surface where there is no example. In approximation, an example is always required but in generalization examples is not needed. Mathematically, the learning process is a nonlinear curve-fitting process, while generalization is the interpolation and extrapolation of the input data. Neural network have the capabilities of learning and generalization.

If an input always generates a unique output, and the mapping is continuous, the problem of reconstructing the mapping is said to be well-posed. Learning is an ill-posed inverse problem. An approximate solution is required to be found for the mapping in the given example of an input-output mapping. The input data may be noisy or imprecise, and also may be insufficient to uniquely construct the mapping. Here an ill-posed problem is transformed into a well-posed one so as to stabilize the solution by adding some auxiliary non-negative functional that embeds prior information such as the smoothness constrained for approximation.

When a network is over trained with too many examples, parameters or epochs, it may produce good results for the training data, but has a poor generalization capability. This is the over fitting phenomenon, and is illustrated in Fig. 5. In statistics, over fitting applies to the situation wherein a model possesses too many parameters, and fits the noise in the data rather than the underlying function. A simple network with smooth input-output mapping usually has a better generalization capability. Generally, the generalization capability of a network is jointly determined by the size of the training pattern set, the complexity of the problem, and the architecture of the network.

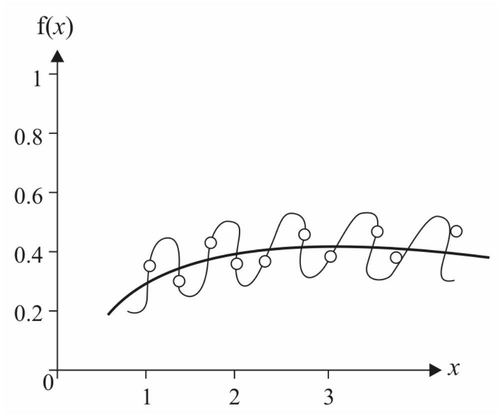

Fig. 5: Proper fitting and over fitting

In Fig. 5, the solid line corresponds to proper fitting, and dashed line corresponds to over fitting.

Size of Training Set

For a given network topology, we can estimate the minimal size of the training set for successfully training the network. For conventional curve-fitting technique, the required number of examples usually grows with the dimensionality of the input space, namely, the curse of dimensionality. Feature extraction can reduce input dimensionality and thus improve the generalization capability of the network.

The training set should be sufficiently large and diverse so that it could represent the problem well. For good generalization, the size of the training set should be at least several times larger than the network's capacity, i.e. $N \gg \frac{N_w}{N_y}$, where N is the size of training set, N_w the total number of weights or free parameters, and N_y the number of output components.

Generalization Error

The generalization error of a trained network can be computed by errors, one is approximation error (AE) and the other is estimation error (EE) two. An approximation error that is due to the finite number of parameters of the approximation scheme used, and an estimation error that is due to the finite number of data available.

For an forward neural network with I input nodes and a single output node, a bound for the generalization error is (GE) given by

$$GE = AE + EE$$

$$= O\left(\frac{1}{P}\right) + O\left(\left[\frac{PI\ln(PN) - \ln\delta}{N}\right]^{1/2}\right) \tag{6}$$

With a probability greater than $1-\delta$, where N is the number of examples, $\delta \in (0,1)$ is the confidence parameter, and P is proportional to the number of parameters, such as P sigmoid hidden units or P threshold units in an MLP.

A bound on the total generalization error is a function of the order of the number of hypothesis parameters P and the number of examples N. The approximation error decreases as P, the order of the model, increases. Since we are using a larger model; however, the estimation error increases due to over fitting (or alternatively, more data). Thus, one cannot reduce the upper bounds on both the error components simultaneously. When P is selected as

$$P \,\alpha\, N^{\frac{1}{3}} \tag{7}$$

then the tradeoff between the approximation and estimation errors is best maintained, and this is the optimal size of the model for the amount of data available. After suitable selecting P and N, the generalization error for FNNs should be $O\left(\frac{1}{P}\right)$. This result is similar to that for an MLP with sigmoid functions.

The objective of learning is to achieve good generalization to new cases, otherwise just use a look-up table. Generalization can be defined as a mathematical interpolation or regression over a set of training points:

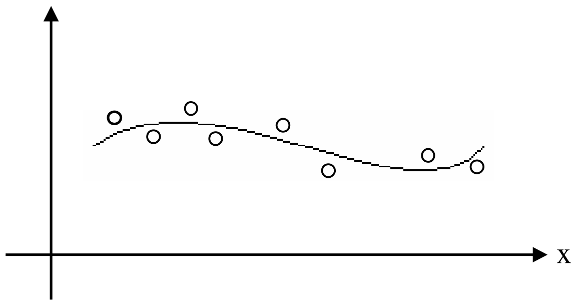

Fig. 6: A set of training points

Example 2 (Computing Parity): **T**he network given in Fig. 7 can learn from m examples to generalize to all 2^n possibilities?

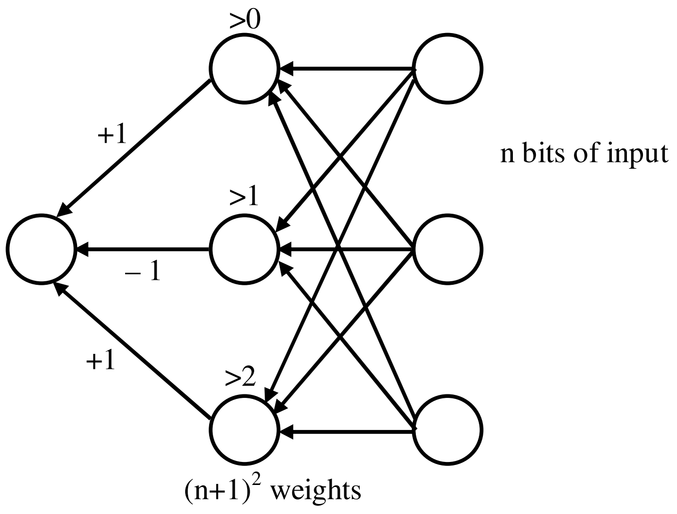

Fig. 7: 2^n possible examples

If we look at many of the problems that have been considered in the field of machine learning, we notice that in all the cases an instance of the problem is given by a set D of input-output pairs, $D \equiv \{(x_i, y_i) \in X \times Y\}_{i=1}^{N}$, belonging to some input and output space.

We assume that there is some relationship between the input and the output, and consider the pairs (x_i, y_i) as examples of it. Therefore we assume that there exists a map

$$f : X \to Y$$

with the property:

$$f(x_i) = y_i \quad i = 1, \cdots, N.$$

Learning, or generalizing, means being able to estimate the function at points of its domain X where no data are available. From a mathematical point of view this means estimating the function f from the knowledge of a subset D of its graph, which is from a set of sparse data. Therefore from this point of view the problem of learning is equivalent to the problem of surface reconstruction. Of course learning and generalization are possible only under the assumption that the world in which we live is at the appropriate level of description redundant. In terms of surface approximation it means that the surface to be reconstructed have to be smooth: small changes in some input determine a correspondingly small change in the output (it may be necessary in some cases to accept piecewise smoothness). Generalization is not possible if the mapping is completely random. For instance, any number of examples for the mapping represented by a telephone directory (people's names into telephone numbers) do not help in estimating the telephone number corresponding to a new name.

Some examples are in order at this point.

Example 3: Learning to pronounce English words from their spelling.

Solution: A well known neural network implementation has been claimed to solve the problem of converting strings of letters in strings of phonemes (Sejnowski and Rosenberg, 1987). The mapping to be learned was therefore $X \equiv \{\text{English words}\} \to Y \equiv \{\text{phonemes}\}$. The input string consisted of 7 letters, and the output was the phoneme corresponding to the letter in the middle of string. Feeding English text in a window of 7 letters, a string of phonemes was produced and then processed by a digital speech synthesizer, therefore enabling the machine to read the text aloud. Because of the particular way of encoding letters and phonemes, the input consisted of a binary vector of 203 elements, and the output of a vector of 26 elements. Clearly in this case the smoothness assumption is often violated, since similar words do not always have similar pronunciations, and this is reflected in the fact that the percentage error on a test set of data was 78%, for continuous informal speech.

Example 4: Learning to recognize a 3D objects from its 2D image.

Solution: The input data set consists of 2D views of different objects in different poses. The output corresponding to a given input is 1 if the input is a 2D view of the object that has to be recognized, and 0 otherwise. A 2D view is a set of features of the 2D images of the object. In the simplest case a feature is the location, on the image plane, of some reference point, but may be any parameter associated to the image. For example, if we are interested in recognizing faces, common features are nose and mouse width, eyebrows thickness, pupil to eyebrows separation, pupil to nose vertical distance, mouth height. In general the mapping to be learned is:

$$X \equiv \{\text{2D view}\} \to Y \equiv \{0,\ 1\}.$$

Example 5: Learning Navigation tasks.

Solution: An indoor robot is manually driven through a corridor, while its frontal camera records a sequence of frontal images. Can this sequence be used to train the robot to drive by itself without crashing into the walls by using the visual input and to generalize to slightly different corridors and different positions within them? The map to be approximated is

$$X \equiv \{\text{images}\} \to Y \equiv \{\text{steering} \quad \text{command}\}.$$

Example 6: Learning Motor Control.

Solution: Consider a multiple-joint robot arm that has to be controlled. One needs to solve the inverse dynamics problem: Compute from position, velocities and acceleration of the joint the corresponding torques at the joints. The map to be learned is, therefore, the "inverse dynamics" map

$$X \equiv \{\text{state space}\} \to Y \equiv \{\text{torques}\}.$$

where the state space is the set of all admissible position, velocities and accelerations of the robot. For a two-joint arm the state space is six dimensional and there are two torques to be learned, one for each joint.

Example 7: Learning to predict time series.

Solution: Suppose we observe and collect data about the temporal evolution of a multi-dimensional dynamical system from some time in the past up to now. Given the state of the system at some time l in the future we would like to be able to predict its state at a successive time l + T. In practice we usually observe the temporal evolution of one of the variables of the system, say $x(l)$, and represent the state of the system as a vector of "delay variables" (Farmer and Sidorowich, 1987):

$$x(l) \equiv \{x(l),\ x(l-\tau), \ldots, x\,(l-(d+1)\tau)\}$$

where τ is an appropriate delay time, and d is larger than the dimension of the attractor of the dynamical system. The map that has to be learned is therefore

$$f\,T : R^d \to R,$$
$$f\,T(x(l)) \to x(l+T).$$

Now, let us discuss the selection of architecture.

7.4 SELECTION OF ARCHITECTURE

The objective of model selection is to find a model that is as simple as possible that fits a given data set with sufficient accuracy, and has a good generalization capability to unseen data. The generalization performance of a network gives a measure of the quality of the chosen model. Existing model-selection approaches can be generally grouped into four categories: crossvalidation, complexity criteria, regularization, and network pruning/growing. Some existing model-selection methods have been reviewed in.

In crossvalidation methods, many networks of different complexity are trained and then tested on an independent validation set. The procedure is computationally demanding and/or requires additional data withheld from the total pattern set. In complexity criterion-based methods, training of many networks is required and hence, computationally demanding, though a validation set is not required. Regularization methods are described in the preceding section; they are more efficient than crossvalidation techniques, but the results may be suboptimal since the penalty terms damage the representation capability of the network. Pruning/growing methods can be under the framework or regularization, which often makes restrictive assumptions, resulting in networks that are suboptimal. In addition, the evolutionary approach is also used to select an optimum architecture of neural networks.

Crossvalidation

Crossvalidation, a standard tool in statistics, is a classical model-selection method. The total pattern set is randomly partitioned into a training set and a validation (test) set. The major part of the total pattern set is included in the training set, which is used to train the network. The remaining, typically, 10 to 20 percent, is included in the validation set and is used for validation. When only one sample is used for validation, the method is called leave-one-out crossvalidation.

Complexity Criteria

An efficient approach for improving the generalization performance is to construct a small network using a parsimonious principle. Statistical model selection with information criteria such as Akaike's final prediction error (FPE) criterion, Akaike information criterion(AIC), Schwartz's Bayesian information criterion (BIC), and Rissanen's minimum description length (MDL) principle are popular and have been widely used for model selection of neural networks.

The selection of architecture is done with the help of following aspects.

Design:

- Architecture of network
- Structure of artificial neurons
- Learning rules

Training:

- Ensuring optimum training
- Learning parameters
- Data preparation
- and more…

Now let us discuss every aspect in detail.

Architecture of the network: First of all we determine the number of network weights, the number of layers, and the number of nodes per layer.

The automated methods like augmentation (cascade correlation) and weight pruning and elimination are applied to find the architecture of the network.

Now for the connectivity of the Architecture of the network the model or hypothesis space are conceptualized. The number of hypotheses are constrained by selective connectivity, shared weights and recursive connections.

Once architecture is finalized, then the structure of artificial neuron nodes is processed by the choice of input integration and choice of activation (transfer) function.

Then a learning rule is selected from the following list:

1. Generalized delta rule (steepest descent)
2. Momentum descent
3. Advanced weight space search techniques
4. Global Error function can also vary

   - normal -quadratic - cubic

With this the designing part of the network is done. Now the training aspect of a neural network for training the following steps are followed:

1. Establish a maximum acceptable error rate.
2. Train the network using a validation test set to tune it.
3. Validate the trained network against a separate test set which is usually referred to as a production test set.

Now try the following exercise.

E7) Discuss the architecture of a neural network.

Now, let us summarize the unit.

7.5 SUMMARY

In this unit, we have covered the following points.

1. The function approximation lays the theoretical foundation for the wide application of forward neural networks.
2. A trained network represents the learned non-linear relation between the input-output pairs, and can generalize where an unknown input pattern is presented to the network.
3. The selection of architecture is done to find a model that is as simple as possible that fits a given data set with sufficient accuracy, and has a good generalization capability to unseen data.

7.6 SOLUTIONS/ANSWERS

E1) Choosing for a 1-2-1 network, a weight structure of

$$W(0)=\begin{bmatrix}-0.27\\-0.41\end{bmatrix},\ \phi^{(1)}(0)=\begin{bmatrix}-0.48\\-0.13\end{bmatrix}$$

and $V(0)=[0.09-0.17],\ \phi^{(2)}(0)=\{0.48\}$

Let initial input be $a^{(0)}=1,$ the output of the first-layer (hidden) is then:

$$a^{(1)}=f^{(1)}\left(W\cdot a^{(0)}+\phi^{(1)}\right)$$

$$=f^{(1)}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1]+\begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right)$$

$$=f^{(1)}\begin{pmatrix}-0.75\\-0.54\end{pmatrix}=\begin{bmatrix}1/1+e^{0.75}\\1/1+e^{0.54}\end{bmatrix}=\begin{bmatrix}0.321\\0.368\end{bmatrix}$$

The second-layer output is:

$$a^{(2)} = f^{(2)}\left(V \cdot a^{(1)} + \phi^{(2)}\right)$$

$$= f^{(2)}\left(\left[0.09 - 0.17\begin{bmatrix}0.321\\0.368\end{bmatrix} + 0.48\right]\right)$$

$$= f^{(2)}(0.446) = 0.446 \quad \left(\text{by choosing } f^{(2)} \text{ as linear function}\right)$$

The error would then be

$$e = t - a^{(2)} = \left[1 + \sin\frac{\pi}{4}x\right] - a^{(2)} = \left[1 + \sin\frac{\pi}{4}\right] - 0.446 = 1.261$$

Next step of algorithm is to backpropagate the sensitivities. Before, beginning backpropagation: recalling that the derivatives of transfer function are needed.

We have for the first-layer

$$f'^{(1)}(x) = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] = \left(1 - a^{(1)}\right) \cdot a^{(1)}$$

For the second layer: $f'^{(2)}(x) = \frac{d}{dx}x = 1.$

Starting point is found at the second layer:

$$V(1) = V(0) - \eta \cdot s^{(2)} \cdot \left(a^{(1)}\right)^T$$

$$= V(0) - 0.2 \cdot \left[-f'^{(2)}\left(a^{(2)}\right)\right] \cdot (t - a) \cdot \left(a^{(1)}\right)^T$$

$$= [0.09 \quad -0.17] + 0.2 \cdot [1] \cdot (1.261) \cdot [0.321 \quad 0.368]$$

$$= [0.171 \quad -0.07721]$$

and $W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T$

where $s^{(1)} = F'^{(1)} \cdot V^T \cdot s^{(2)}$, where $F'^{(1)}$ is a matrix of $f'^{(1)}$ for two rows of vector.

$$= \begin{bmatrix}\left(1 - a_1^{(1)}\right) \cdot a_1^{(1)} & 0 \\ 0 & \left(1 - a_2^{(1)}\right) \cdot a_2^{(1)}\end{bmatrix} \cdot \begin{bmatrix}0.09\\-0.17\end{bmatrix} \cdot [-0.2 \text{x} 1.261]$$

$$= \begin{bmatrix}-0.0495\\0.0997\end{bmatrix}$$

$$\therefore\ W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T = \begin{bmatrix}-0.27\\-0.41\end{bmatrix} - 0.2\begin{bmatrix}-0.0495\\0.0997\end{bmatrix} = \begin{bmatrix}-0.260\\-0.429\end{bmatrix}$$

Bias is likewise calculated:

$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta s^{(2)} = 0.48 - 0.2 \times [-2.522] = 0.9844$$

$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta s^{(1)} = \begin{bmatrix}-0.48\\-0.13\end{bmatrix} - 0.2\begin{bmatrix}-0.0495\\0.0997\end{bmatrix} = \begin{bmatrix}-0.4701\\-0.1499\end{bmatrix}$$

This completes one iteration of backpropagation algorithm. Next proceed to choose another input and perform another iteration. Continue to iterate until the difference between network response and the target function reaches some acceptable level.

E2) First step is to propagate the input through network:

$$n^{(1)} = W \cdot I - + \phi^{(1)} = (-1)\cdot(-1)+1 = 2$$

Hence the activation $a^{(1)} = \tanh(2) = 0.964$

Now $\quad n^{(2)} = V \cdot a(1) + \phi^{(2)} = -2(0.964)+1 = -0.928$

Hence its activation $a^{(2)} = \tanh\left(n^{(2)}\right) = \tanh(-0.928) = -0.7297$

Therefore error

Now, backpropagation sensitivities:

$$s^{(2)} = -1 \cdot f'^{(2)}\left(n^{(2)}\right)\cdot\left(t - a^{(2)}\right)$$
$$= -1 \cdot \left[1 - \left(a^{(2)}\right)^2\right] \times 1.7297$$
$$= -\left[1 - 0.7297^2\right] \times 1.7297 = -0.8087$$

and $\quad s(1) = f'^{(1)}\left(n^{(1)}\right)V^T \cdot s^{(2)},$

$$= \left[1 - \left(a^{(1)}\right)^2\right] \times V^T \cdot s^{(2)} = \left[1 - 0.964^2\right]\cdot(-2)\cdot(-0.8087)$$
$$= 0.1142$$

Finally, the weights and biases are updated using the equations:

$$V(1) = V(0) - \eta \cdot s^{(2)} \cdot \left(a^{(1)}\right)^T = -2 - 1\times(-0.8087)\cdot(0.964) = -1.2204$$
$$W(1) = W(0) - \eta \cdot s^{(1)} \cdot \left(a^{(0)}\right)^T = -1 - 1\times(0.1142)(-1) = -0.8858.$$
$$\phi^{(1)}(1) = \phi^{(1)}(0) - \eta s^{(1)} = 1 - 1\times 0.1142 = 0.8858$$
$$\phi^{(2)}(1) = \phi^{(2)}(0) - \eta s^{(2)} - 1 - 1\times(-0.8087) = 1.8087.$$

E3) For a multi-layer network, the forward equations are:

$$a^{(1)} = W^{(1)} \cdot I + \phi^{(1)}$$
$$a^{(2)} = W^{(2)} \cdot a^{(1)} + \phi^{(2)} = W^{(2)} \cdot W^{(1)} \cdot I + W^{(2)} \cdot \phi^{(1)} + \phi^{(2)}$$
$$a^{(3)} = W^{(3)} \cdot a^{(2)} + \phi^{(3)}$$
$$= \left[W^{(3)} \cdot W^{(2)} \cdot W^{(1)} \cdot I + W^{(3)} \cdot W^{(2)} \cdot \phi^{(1)} + W^{(3)} \cdot \phi^{(2)} + \phi^{(3)}\right]$$

For an M-layer linear network, the equivalent single-layer linear network has weight matrix and bias vector as:

$$W = W^{(M)} \cdot W^{(M-1)} \ldots W^{(1)},$$

and $\phi = \left[W^{(M)} \cdot W^{(M-1)} \cdots W^{(2)}\right]\cdot \phi^{(1)}\Big] + W^{M)} \cdot W^{M-1)} \cdots W^{(3)}\Big]\cdot \phi^{(2)} + \ldots + \phi^{(M)}.$

E4) Sensitivity to be backpropagated for single-layer linear network is:

$$s^{(1)} = -f'^{(1)}\left(n^{(1)}\right)\cdot(t - a) = -e(\text{since } f'(x) = 1)$$

The weight update is given by

$$W(k+1) = W(k) - \eta \cdot s^{(1)} \cdot a\left(a^{(0)}\right)^T$$
$$= W(k) + \eta . e . (I)^T$$

and bias is $\phi(k+1) = \phi(k) - \eta s^{(1)} = \phi(k) + \eta e$

This is identical to LMS algorithm (ADALINE).