Artificial Neural Networks

Chung-Ming Kuan
Institute of Economics
Academia Sinica

Abstract

Artificial neural networks (ANNs) constitute a class of flexible nonlinear models designed to mimic biological neural systems. In this entry, we introduce ANN using familiar econometric terminology and provide an overview of ANN modeling approach and its implementation methods.

† Correspondence: Chung-Ming Kuan, Institute of Economics, Academia Sinica, 128 Academia Road, Sec. 2, Taipei 115, Taiwan; ckuan@econ.sinica.edu.tw.

†† I would like to express my sincere gratitude to the editor, Professor Steven Durlauf, for his patience and constructive comments on early drafts of this entry. I also thank Shih-Hsun Hsu and Yu-Lieh Huang for very helpful suggestions. The remaining errors are all mine.

1 Introduction

Artificial neural networks (ANNs) constitute a class of flexible nonlinear models designed to mimic biological neural systems. Typically, a biological neural system consists of several layers, each with a large number of neural units (neurons) that can process the information in a parallel manner. The models with these features are known as ANN models. Such models can be traced back to the simple input-output model of McCulloch and Pitts (1943) and the "perceptron" of Rosenblatt (1958). The early yet simple ANN models, however, did not receive much attention because of their limited applicability and also because of the limitation of computing capacity at that time. In seminal works, Rumelhart et al. (1986) and McClelland et al. (1986) presented the new developments of ANN, including more complex and flexible ANN structures and a new network learning method. Since then, ANN has become a rapidly growing research area.

As far as model specification is concerned, ANN has a multi-layer structure such that the middle layer is built upon many simple nonlinear functions that play the role of neurons in a biological system. By allowing the number of these simple functions to increase indefinitely, a multi-layered ANN is capable of approximating a large class of functions to any desired degree of accuracy, as shown in, e.g., Cybenko (1989), Funahashi (1989), Hornik, Stinchcombe and White (1989, 1990), and Hornik (1991, 1993). From an econometric perspective, ANN can be applied to approximate the unknown conditional mean (median, quantile) function of the variable of interest without suffering from the problem of model misspecification, unlike parametric models commonly used in empirical studies. Although nonparametric methods, such as series and polynomial approximators, also possess this property, they usually require a larger number of components to achieve similar approximation accuracy (Barron, 1993). ANNs are thus a parsimonious approach to nonparametric functional analysis.

ANNs have been widely applied to solve many difficult problems in different areas, including pattern recognition, signal processing, language learning, etc. Since White (1988), there have also been numerous applications of ANN in economics and finance. Unfortunately, the ANN literature is not easy to penetrate, so it is hard for applied economists to understand why ANN works and how it can be implemented properly. Fortunately, while the ANN jargon originated from cognitive science and computer science, they often have econometric interpretations. For example, a "target" is, in fact, a dependent variable of interest, an "input" is an explanatory variable, and network "learning" amounts to the estimation of unknown parameters in a network. The purpose of this entry is thus two-fold. First, this entry introduces ANN using familiar econometric terminology and hence serves to bridge the gap between the fields of ANN and economics. Second, this entry provides an overview of ANN modeling approach and its implementation methods. For an early review of ANN from an econometric perspective, we refer to Kuan and White (1994).

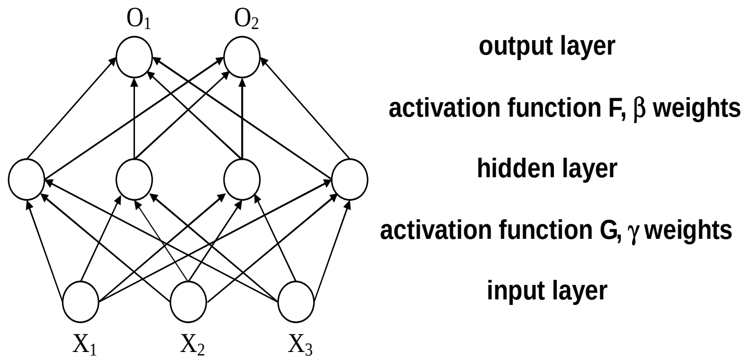
This entry proceeds as follows. We introduce various ANN model specifications and the choices of network functions in Section 2. We present the "universal approximation" property of ANN in Section 3. Model estimation and model complexity regularization are discussed in Section 4. Section 5 concludes.

2 ANN Model Specifications

Let Y denote the collection of n variables of interest with the t-th observation $\boldsymbol{y}_t$ $(n \times 1)$ and X the collection of m explanatory variables with the t-th observation $\boldsymbol{x}_t$ $(m \times 1)$. In the ANN literature, the variables in Y are known as *targets* or *target variables*, and the variables in X are *inputs* or *input variables.* There are various ways to build an ANN model that can be used to characterize the behavior of $\boldsymbol{y}_t$ using the information contained in the input variables $\boldsymbol{x}_t$. In this section, we introduce some network architectures and the functions that are commonly used to build an ANN.

2.1 Feedforward Neural Networks

We first consider a network with an *input* layer, an *output* layer, and a *hidden* layer in between. The input (output) layer contains m input units (n output units) such that each unit corresponds to a particular input (output) variable. In the hidden layer, there are q hidden units connected to all input and output units; the strengths of such connections are labeled by (unknown) parameters known as the network *connection weights.* In particular, $\boldsymbol{\gamma}_h = (\gamma_{h,1}, \dots, \gamma_{h,m})'$ denotes the vector of the connection weights between the h-th hidden unit and all m input units, and $\boldsymbol{\beta}_j = (\beta_{j,1}, \dots, \beta_{j,q})'$ denotes the vector of the connection weights between the j-th output unit and all q hidden units. An ANN in which the sample information (signals) are passed forward from the input layer to the output layer without feedback is known as a *feedforward neural network.* Figure 1 illustrates the architecture of a 3-layer feedforward network with 3 input units, 4 hidden units and 2 output units.



O1
O2
output layer
activation function F, β weights
hidden layer
activation function G, γ weights
input layer
X1
X2
X3

Figure 1: A feedforward network with 3 input units, 4 hidden units and 2 output units.

This multi-layered structure of a feedforward network is designed to function as a biological neural system. The input units are the neurons that receive the information (stimuli) from the outside environment and pass them to the neurons in a middle layer (i.e., hidden units). These neurons then transform the input signals to generate neural signals and forward them to the neurons in the output layer. The output neurons in turn generate signals that determine the action to be taken. Note that all information from the units in one layer are processed simultaneously, rather than sequentially, by the units in an "upper" layer.[1]

Formally, the input units receive the information $\boldsymbol{x}_t$ and send to all hidden units, weighted by the connection weights between the input and hidden units. This information is then transformed by the *activation function* G in each hidden unit. That is, the h-th hidden unit receives $\boldsymbol{x}_t'\boldsymbol{\gamma}_h$ and transforms it to $G(\boldsymbol{x}_t'\boldsymbol{\gamma}_h)$. The information generated by all hidden units is further passed to the output units, again weighted by the connection weights, and transformed by the activation function F in each output unit. Hence, the j-th output unit receives $\sum_{h=1}^{q}\beta_{j,h}G(\boldsymbol{x}_t'\boldsymbol{\gamma}_h)$ and transforms it into the network output:

$$o_{t,j} = F\Big(\sum_{h=1}^{q}\beta_{j,h}G(\boldsymbol{x}_t'\boldsymbol{\gamma}_h)\Big), \qquad j = 1,\ldots,n. \tag{1}$$

The output O_j is used to describe or predict the behavior of the j-th target Y_j.

In practice, it is typical to include a constant term, also known as the *bias* term, in

[1]This concept, also known as *parallel processing* or *massive parallelism*, differs from the traditional concept of sequential processing and led to a major advance in designing computer architecture.

each activation function in (1). That is,

$$o_{t,j} = F\Big(\beta_{j,0} + \sum_{h=1}^{q} \beta_{j,h} G(\gamma_{h,0} + \boldsymbol{x}_t' \gamma_h)\Big), \qquad j = 1, \ldots, n, \tag{2}$$

where $\gamma_{h,0}$ is the bias term in the h-th hidden unit and $\beta_{j,0}$ is the bias term in the j-th output unit. A constant term in each activation function adds flexibility to hidden-unit and output-unit responses (activations), in a way similar to the constant term in (non)linear regression models. Note that when there is no transformation in the output units, F is an identity function (i.e., $F(a) = a$) so that

$$o_{t,j} = \beta_{j,0} + \sum_{h=1}^{q} \beta_{j,h} G(\gamma_{h,0} + \boldsymbol{x}_t' \gamma_h), \qquad j = 1, \ldots, n, \tag{3}$$

It is also straightforward to construct networks with two or more hidden layers. For simplicity, we will focus on the 3-layer networks with only one hidden layer.

While parametric econometric models are typically formulated using a given function of the input $\boldsymbol{x}_t$, the network (2) is a class of flexible nonlinear functions of $\boldsymbol{x}_t$. The exact form of a network model depends on the activation functions (F and G) and the number of hidden units (q). In particular, the network function in (3) is an affine transformation of G and hence may be interpreted as an expansion with the "basis" function G.

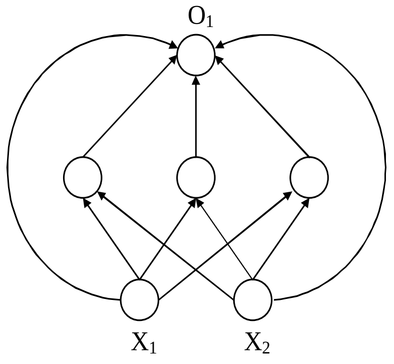
The networks (2) and (3) can be further extended. For example, one may construct a network in which the input units are connected not only to the hidden units but also directly to the output units. This leads to networks with *shortcut* connections. Corresponding to (2), the outputs of a feedforward network with shortcuts are

$$o_{t,j} = F\Big(\beta_{j,0} + \boldsymbol{x}_t' \boldsymbol{\alpha}_j + \sum_{h=1}^{q} \beta_{j,h} G(\boldsymbol{\gamma}_{h,0} + \boldsymbol{x}_t' \boldsymbol{\gamma}_h)\Big), \qquad j = 1, \ldots, n,$$

where $\boldsymbol{\alpha}_j$ is the vector of connection weights between the output and input units. and corresponding to (3), the outputs are

$$o_{t,j} = \beta_{j,0} + \boldsymbol{x}_t' \boldsymbol{\alpha}_j + \sum_{h=1}^{q} \beta_{j,h} G(\boldsymbol{\gamma}_{h,0} + \boldsymbol{x}_t' \boldsymbol{\gamma}_h), \qquad j = 1, \ldots, n.$$

Figure 2 illustrates the architecture of a feedforward network with 2 input units, 3 hidden units, 1 output unit and shortcut connections. Thus, parametric econometric models may be interpreted as feedforward networks with shortcut connections but no hidden-layer connections. The linear combination of hidden-unit activations, $\sum_{h=1}^{q} \beta_{j,h} G(\boldsymbol{\gamma}_{h,0} + \boldsymbol{x}_t' \boldsymbol{\gamma}_h)$, in effect characterizes the nonlinearity not captured by the linear function of $\boldsymbol{x}_t$.



output layer

activation function F, β weights

hidden layer

activation function G, γ weights

input layer

Figure 2: A feedforward neural network with shortcuts.

2.2 Recurrent Neural Networks

From the preceding section we can see that there is no "memory" device in feedforward networks that can store the signals generated earlier. Hence, feedforward networks treat all sample information as "new;" the signals in the past do not help to identify data features, even when sample information exhibits temporal dependence. As such, a feedforward network must be expanded to a large extent so as to represent complex dynamic patterns. This causes practical difficulty because a large network may not be easily implemented. To utilize the information from the past, it is natural to include lagged target information $\boldsymbol{y}_{t-k}$, $k = 1, \dots, s$, as input variables, similar to linear AR and ARX models in econometric studies. Yet, such networks do not have any built-in structure that can "memorize" previous neural responses (transformed sample information). The so-called *recurrent neural networks* overcome this difficulty by allowing internal feedbacks and hence are especially appropriate for dynamic problems.

Jordan (1986) first introduced a recurrent network with feedbacks from output units. That is, the output units are connected to input units but with *time delay*, so that the network outputs at time $t-1$ are also the input information at time t. Specifically, the outputs of a Jordan network are

$$o_{t,j} = F\Big(\beta_{j,0} + \sum_{h=1}^{q} \beta_{j,h} G(\boldsymbol{\gamma}_{h,0} + \boldsymbol{x}_t' \boldsymbol{\gamma}_h + \boldsymbol{o}_{t-1}' \boldsymbol{\delta}_h)\Big), \qquad j = 1, \dots, n. \tag{4}$$

where $\boldsymbol{\delta}_h$ is the vector of the connection weights between the h-th hidden unit and the input units that receive lagged outputs $\boldsymbol{o}_{t-1} = (o_{t-1,1}, \dots, o_{t-1,n})'$. The network (4) can be further extended to allow for more lagged outputs $\boldsymbol{o}_{t-2}, \boldsymbol{o}_{t-3}, \dots$.

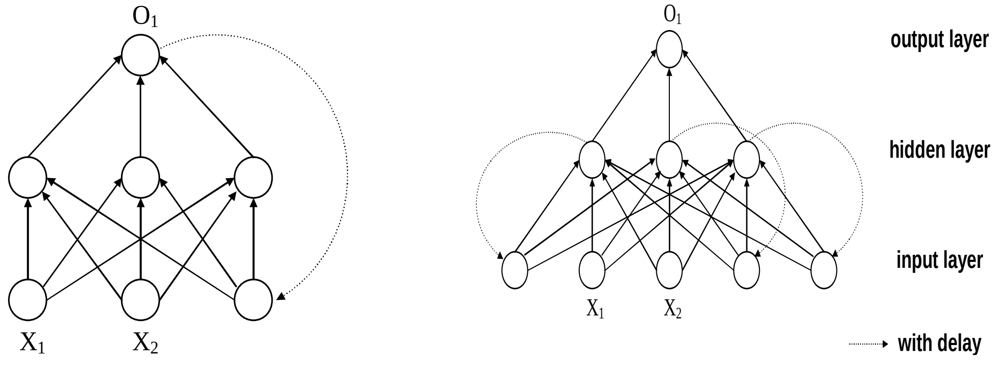


Figure 3: Recurrent neural networks: Jordan (left) and Elman (right).

Similarly, Elman (1990) considered a recurrent network in which the hidden units are connected to input units with time delay. The outputs of an Elman network are:

$$\begin{aligned} o_{t,j} &= F\Big(\beta_{j,0} + \sum_{h=1}^{q} \beta_{j,h} a_{t,h}\Big), \qquad j = 1, \ldots, n, \\ a_{t,h} &= G(\boldsymbol{\gamma}_{h,0} + \boldsymbol{x}_t' \boldsymbol{\gamma}_h + \boldsymbol{a}_{t-1}' \boldsymbol{\delta}_h), \qquad h = 1, \ldots, q, \end{aligned} \tag{5}$$

where $\boldsymbol{a}_{t-1} = (a_{t-1,1}, \ldots, a_{t-1,q})'$ is the vector of lagged hidden-unit activations, and $\boldsymbol{\delta}_h$ here is the vector of the connection weights between the h-th hidden unit and the input units that receive lagged hidden-unit activations $\boldsymbol{a}_{t-1}$. The network (5) can also be extended to allow for more lagged hidden-unit activations $\boldsymbol{a}_{t-2}, \boldsymbol{a}_{t-3}, \ldots$. Figure 3 illustrates the architectures of a Jordan network and an Elman network.

From (4) and (5) we can see that, by recursive substitution, the outputs of these recurrent networks can be expressed in terms of current and all past inputs. Such expressions are analogous to the distributed lag model or the AR representation of an ARMA model (when the inputs are lagged targets). Thus, recurrent networks incorporate the information in the past input variables without including all of them in the model. By contrast, a feedforward network requires a large number of inputs to carry such information. Note that the Jordan network and the Elman network summarize past input information in different ways and hence have their own merits. When the previous "location" of a network is crucial in determining the next move, as in the design of a robot, a Jordan network seems more appropriate. When the past internal neural responses are more important, as in language learning problems, an Elman network may be preferred.

2.3 Choices of Activation Function

As far as model specifications are concerned, the building blocks of an ANN model are the activation functions F and G. Different choices of the activation functions result in different network models. We now introduce some activation functions commonly employed in empirical studies.

Recall that the hidden units play the role of neurons in a biological system. Thus, the activation function in each hidden unit determines whether a neuron should be turned on or off. Such an on/off response can be easily represented using an indicator (threshold) function, also known as a *heaviside* function in the ANN literature, i.e.,

$$G(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h) = \begin{cases} 1, & \text{if } \gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h \geq c, \\ 0, & \text{if } \gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h < c, \end{cases}$$

where c is a pre-determined threshold value. That is, depending on the strength of connection weights and input signals, the activation function G will determine whether a particular neuron is on ($G(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h) = 1$) or off ($G(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h) = 0$).

In a complex neural system, neurons need not have only an on/off response but may be in an intermediate position. This amounts to allowing the activation function to assume any value between zero and one. In the ANN literature, it is common to choose a *sigmoid* (S-shaped) and *squashing* (bounded) function. In particular, if the input signals are "squashed" between zero and one, the activation function is understood as a smooth counterpart of the indicator function. A leading example is the logistic function:

$$G(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h) = \frac{1}{1 + \exp\big(-[\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h]\big)}.$$

which approaches one (zero) when its argument goes to infinity (negative infinity). Hence, the logistic activation function generates a partially on/off signal based on the received input signals.

Alternatively, the hyperbolic tangent (tanh) function, which is also a sigmoid and squashing function, can serve as an activation function:

$$G(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h) = \frac{\exp\big(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h\big) - \exp\big(-[\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h]\big)}{\exp\big(\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h\big) + \exp\big(-[\gamma_{h,0} + \boldsymbol{x}_t'\boldsymbol{\gamma}_h]\big)}.$$

Compared to the logistic function, this function may assume negative values and is bounded between -1 and 1. It approaches 1 (-1) when its argument goes to infinity (minus infinity). This function is more flexible because the negative values, in effect,

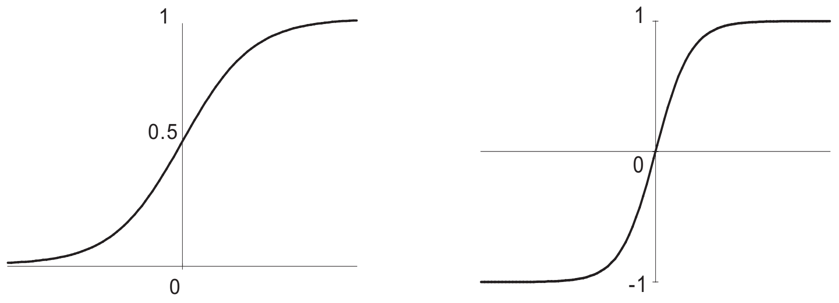


Figure 4: Activation functions: logistic (left) and tanh (right).

represent "suppressing" signals from the hidden unit. See Figure 4 for an illustration of the logistic and tanh functions. Note that for the logistic function G, a re-scaled function $\widetilde{G}$ such that $\widetilde{G}(a) = 2G(a) - 1$ also generates values between -1 and 1 and may be used in place of the tanh function.[2]

The aforementioned activation functions are chosen for convenience because they are differentiable everywhere and their derivatives are easy to compute. In particular, when G is the logistic function,

$$\frac{\mathrm{d}G(a)}{\mathrm{d}a} = G(a)[1 - G(a)];$$

when G is the tanh function,

$$\frac{\mathrm{d}G(a)}{\mathrm{d}a} = \left[\frac{2}{\exp(a) + \exp(-a)}\right]^2 = \mathrm{sech}^2(a).$$

These properties facilitate parameter estimation, as will be seen in Section 4.1. Nevertheless, these functions are not necessary for building proper ANNs. For example, smooth cumulative distribution functions, which are sigmoidal and squashing, are also legitimate candidates for activation function. In Section 3, it is shown that, as far as network approximation property is concerned, the activation function in hidden units does not even have to be sigmoidal, yet boundedness is usually required. Thus, sine and cosine functions can also serve as an activation function.

As for the activation function F in the output units, it is common to set it as the identity function so that the outputs of (3) enjoy the freedom of assuming any real

[2]A choice of the activation function in classification problems is the so-called *radial basis function*. We do not discuss this choice because its argument is not an affine transformation of inputs and hence does not fit in our framework here. Moreover, the networks with this activation function provide only *local* approximation to unknown functions, in contrast with the approximation property discussed in Section 3.

value. This choice suffices for the network approximation property discussed in Section 3. When the target is a binary variable taking the values zero and one, as in a classification problem, F may be chosen as the logistic function so that the outputs of (2) must fall between zero and one, analogous to a logit model in econometrics.

3 ANN as an Universal Approximator

What makes ANN a useful econometric tool is its *universal approximation* property which basically means that a multi-layered ANN with a large number of hidden units can well approximate a large class of functions. This approximation property is analogous to that of nonparametric approximators, such as polynomials and Fourier series, yet it is not shared by parametric econometric models.

To present the approximation property, we consider the network function element by element. Let $f_{G,q}\colon \mathbb{R}^m \times \Theta_{m,q} \to \mathbb{R}$ denote the network function with q hidden units, the output activation function F being the identity function, and the hidden-unit activation function G, i.e.,

$$f_{G,q}(\boldsymbol{x};\boldsymbol{\theta}) = \beta_0 + \sum_{h=1}^{q} \beta_h G(\gamma_{h,0} + \boldsymbol{x}'\boldsymbol{\gamma}_h),$$

as in (3), where $\Theta_{m,q}$ is the parameter space whose dimension depends on m and q, and $\boldsymbol{\theta} \in \Theta_{m,q}$ (note that the subscripts m and q for $\boldsymbol{\theta}$ are suppressed). Given the activation function G, the collection of all $f_{G,q}$ functions with different q is:

$$\mathcal{F}_G = \bigcup_{q=1}^{\infty} \left\{ f_{G,q}\colon f_{G,q}(\boldsymbol{x};\boldsymbol{\theta}) = \beta_0 + \sum_{h=1}^{q} \beta_h G(\gamma_{h,0} + \boldsymbol{x}'\boldsymbol{\gamma}_h) \right\};$$

when the union is taken up to a finite number N, the resulting collection is denoted as $\mathcal{F}_G^N$. Intuitively, $\mathcal{F}_G$ is capable of functional approximation because $f_{G,q}$ can be viewed as an expansion with the "basis" function G and hence is similar to a nonparametric approximator.

More formally, we follow Hornik (1991) and consider two measures of the closeness between functions. First define the *uniform distance* between functions f and g on the set K as

$$d_K(f,g) = \sup_{\boldsymbol{x} \in K} |f(\boldsymbol{x}) - g(\boldsymbol{x})|.$$

Let K denote a compact subset in $\mathbb{R}^m$ and $C(K)$ denote the space of all continuous functions on K. Then, *when the activation function G is continuous, bounded and nonconstant, the collection $\mathcal{F}_G$ is dense in $C(K)$ for all K in $\mathbb{R}^m$ in terms of d_K* (Theorem 2 of Hornik, 1991).[3] That is, for any function g in $C(K)$ and any $\varepsilon > 0$, there is a network function $f_{G,q}$ in $\mathcal{F}_G$ such that $d_K(f_{G,q} - g) < \varepsilon$. As $\mathcal{F}_G^N$ is not dense in $C(K)$ for any finite number N, this result shows that any continuous function can be approximated arbitrarily well on compacta by a 3-layered feedforward network $f_{G,q}$, provided that q, the number of hidden units, is sufficiently large.

Taking $\boldsymbol{x}$ as random variables, defined in the probability space with the probability measure $\mathbb{P}$, we consider the *L_r-norm* of $f(\boldsymbol{x}) - g(\boldsymbol{x})$:

$$\|f - g\|_r = \left(\int_{\mathbb{R}^m} |f(\boldsymbol{x}) - g(\boldsymbol{x})|^r \, \mathrm{d}\, \mathbb{P}(\boldsymbol{x}) \right)^{1/r},$$

$1 \leq r < \infty$. For $r = 2$ ($r = 1$), this is the well known measure of mean squared error (mean absolute error). Then, *when the activation function G is bounded and nonconstant, the collection $\mathcal{F}_G$ is dense in the L_r space* (Theorem 1 of Hornik, 1991). That is, any function g (with finite L_r-norm) can also be well approximated by a 3-layered feedforward network $f_{G,q}$ in terms of L_r-norm when q is sufficiently large.

It should be emphasized that the universal approximation property of a feedforward network hinges on the 3-layered architecture and the number of hidden units, but not on the activation function *per se*. As stated above, the activation function in the hidden unit can be a general bounded function and does not have to be sigmoidal. Hornik (1993) provides results that permit even more general activation functions. Moreover, a feedforward network with only one hidden layer suffices for such approximation property. More hidden layers may be helpful in certain applications but are not necessary for functional approximation.

Barron (1993) further derived the rate of approximation in terms of mean squared error $\|f - g\|_2^2$. It was shown that 3-layered feedforward networks $f_{G,q}$ with G a sigmoidal function can achieve the approximation rate of order $O(1/q)$, for which the number of parameters grows linearly with q (with the order $O(mq)$). This is in sharp contrast with other expansions, such as polynomial (with p the degree of the polynomial) and

[3] Hornik (1991) considered the network without the bias term in the output unit, i.e., $\beta_0 = 0$. Yet as long as G is not a constant function, all the results in Hornik (1991) carry over; see Stinchcombe and White (1998) for details.

spline (with p the number of knots per coordinate), which yield suitable approximation when the number of parameters grows exponentially (with the order $O(p^m)$). Thus, it is practically difficult for such expansions to approximate well when the dimension of the input space, m, is large.

4 Implementation of ANNs

In practice, when the activation functions in an ANN are chosen, it remains to estimate its connection weights (unknown parameters) and to determine a proper number of hidden units. Given that the connection weights of an ANN model are unknown, this network must be properly "trained" so as to "learn" the unknown weights. This is why parameter estimation is referred to as network *learning* and the sample used for parameter estimation is referred to a *training sample* in the ANN literature. As the number of hidden units q determines network complexity, finding a suitable q is known as network *complexity regularization*.

4.1 Model Estimation

The network parameters can be estimated by either *on-line* or *off-line* methods. An on-line learning algorithm is just a *recursive estimation* method which updates parameter estimates when new sample information becomes available. By contrast, off-line learning methods are based on fixed training samples; standard econometric estimation methods are typically off-line.

To ease the discussion of model estimation, we focus on the simple case that there is only one target variable y and the network function $f_{G,q}$. Generalization to the case with multiple target variables and vector-valued network functions is straightforward. Once the activation function G is chosen and the number of hidden units is given, $f_{G,q}$ is a nonlinear parametric model for the target y; the network with multiple outputs is a system of nonlinear models. Taking mean squared error as the criterion, the parameter vector of interest $\boldsymbol{\theta}^*$ thus minimizes

$$\mathbb{E}[y - f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})]^2. \tag{6}$$

It is well known that

$$\mathbb{E}\big[y - f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})\big]^2 = \mathbb{E}\big[y - E(y|\boldsymbol{x})\big]^2 + \mathbb{E}\big[\mathbb{E}(y|\boldsymbol{x}) - f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})\big]^2.$$

As $\mathbb{E}(y|\boldsymbol{x})$ is the best L_2 predictor of y, $\boldsymbol{\theta}^*$ must also minimize the mean squared approximation error: $\mathbb{E}\big[\mathbb{E}(y|\boldsymbol{x}) - f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})\big]^2$. This shows that, among all 3-layered feedforward networks with the activation function G and q hidden units, $f_{G,q}(\boldsymbol{x};\boldsymbol{\theta}^*)$ provides the best approximation to the conditional mean function.

Given a training sample of T observations, an estimator of $\boldsymbol{\theta}^*$ can be obtained by minimizing the sample counterpart of (6):

$$\frac{1}{T}\sum_{t=1}^{T}[y_t - f_{G,q}(\boldsymbol{x}_t;\boldsymbol{\theta})]^2,$$

which is just the objective function of the nonlinear least squares (NLS) method. The NLS method is an off-line estimation method because the size of the training sample is fixed. Under very general conditions on the data and nonlinear function, it is well known that the NLS estimator is strongly consistent for $\boldsymbol{\theta}^*$ and asymptotically normally distributed; see, e.g., Gallant and White (1988).

In many ANN applications (e.g., signal processing and language learning), the training sample is not fixed but constantly expands with new data. In such cases, off-line estimation may not be feasible, but on-line estimation methods, which update the parameter estimates based solely on the newly available data, are computationally more tractable. Moreover, on-line estimation methods can be interpreted as "adaptive learning" by biological neural systems. It should be emphasized that when there is only a given sample, as in most empirical studies in economics, recursive estimation is *not* to be preferred because it is, in general, statistically less efficient than the NLS method in finite samples.

Note that the parameter of interest $\boldsymbol{\theta}^*$ is the zero of the first order condition of (6):

$$\mathbb{E}\big[\nabla f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})\big(y - f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})\big)\big] = \mathbf{0},$$

where $\nabla f_{G,q}(\boldsymbol{x};\boldsymbol{\theta})$ is the (column) gradient vector of $f_{G,q}$ with respect to $\boldsymbol{\theta}$. To estimate $\boldsymbol{\theta}^*$, a recursive algorithm proposed by Rumelhart, Hinton, and Williams (1986) is

$$\hat{\boldsymbol{\theta}}_{t+1} = \hat{\boldsymbol{\theta}}_t + \eta_t \nabla f_{G,q}(\boldsymbol{x}_t;\hat{\boldsymbol{\theta}}_t)\big[y_t - f_{G,q}(\boldsymbol{x}_t;\hat{\boldsymbol{\theta}}_t)\big], \tag{7}$$

where $\eta_t > 0$ is a parameter that re-scales the adjustment term in the square bracket. It can be seen from (7) that the adjustment term is determined by the gradient descent direction and the error between the target and network output: $y_t - f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t)$, and it requires only the information at time t, i.e., y_t, $\boldsymbol{x}_t$, and the estimate $\hat{\boldsymbol{\theta}}_t$.[4]

The algorithm (7) is known as the *error back-propagation* (or simply back-propagation) algorithm in the ANN literature, because the error signal $[y_t - f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t)]$ is propagated back through the network to determine the change of each weight. The underlying idea of this algorithm can be traced back to the classical *stochastic approximation* method introduced in Robins and Monro (1951). White (1989) established consistency and asymptotic normality of $\hat{\boldsymbol{\theta}}_t$ in (7). Note that the parameter η_t in the algorithm is known as a *learning rate*. For consistency of $\hat{\boldsymbol{\theta}}_t$, it is required that η_t satisfies $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$, e.g., $\eta_t = 1/t$. The former condition ensures that the updating process may last indefinitely, whereas the latter implies $\eta_t \to 0$ so that the adjustment in the parameter estimates can be made arbitrarily small.[5]

Instead of the gradient descent direction, it is natural to construct a recursive algorithm with a Newton search direction. Kuan and White (1994) proposed the following algorithm:

$$\begin{aligned} \widehat{\boldsymbol{H}}_{t+1} &= \widehat{\boldsymbol{H}}_t + \eta_t \big[\nabla f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t) \nabla f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t)' - \widehat{\boldsymbol{H}}_t\big], \\ \hat{\boldsymbol{\theta}}_{t+1} &= \hat{\boldsymbol{\theta}}_t + \eta_t\, \widehat{\boldsymbol{H}}_{t+1}^{-1} \nabla f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t)\big[y_t - f_{G,q}(\boldsymbol{x}_t; \hat{\boldsymbol{\theta}}_t)\big], \end{aligned} \tag{8}$$

where $\widehat{\boldsymbol{H}}_{t+1}$ characterizes a Newton direction and is recursively updated via the first equation. Kuan and White (1994) showed that $\hat{\boldsymbol{\theta}}_t$ in (8) is $\sqrt{t}$-consistent, statistically more efficient than $\hat{\boldsymbol{\theta}}_t$ in (7), and asymptotically equivalent to the NLS estimator. The algorithm (8) may be implemented in different ways; for example, there is an algorithm that is algebraically equivalent to (8) but does not involve matrix inversion. See Kuan and White (1994) for more discussions on the implementation of the Newton algorithms.

On the other hand, estimating recurrent networks is more cumbersome. From (4) and (5) we can see that recurrent network functions depend on $\boldsymbol{\theta}$ directly and also indirectly through the presence of internal feedbacks (i.e., lagged output and lagged hidden-unit activations). The indirect dependence on parameters must be taken into account in

[4]The algorithm (7) is analogous to the numerical steepest-descent algorithm. However, (7) utilizes only the information at time t, whereas numerical optimization algorithms are computed using all the information in a given sample and hence are off-line methods.

[5]In many applications of ANN, the learning rate is often set to a constant η_o; the resulting estimate $\hat{\theta}_t$ loses consistency in this case. Kuan and Hornik (1991) established a convergence result based on small-η_o asymptotics.

calculating the derivatives with respect to $\boldsymbol{\theta}$. Thus, NLS optimization algorithms that require analytic derivatives are difficult to implement. Kuan, Hornik, and White (1994) proposed the *dynamic back-propagation* algorithm for recurrent networks, which is analogous to (7) but involves more updating equations. Kuan (1995) further proposed a Newton algorithm for recurrent networks, analogous to (8), and showed that it is $\sqrt{t}$-consistent and statistically more efficient than the dynamic back-propagation algorithm. We omit the details of these algorithms; see Kuan and Liu (1995) for an application of these estimation methods for both feedforward and recurrent networks.

Note that the NLS method and recursive algorithms all require computing the derivatives of the network function. Thus, a smooth and differentiable activation function, as the examples given in Section 2.3, are quite convenient for network parameter estimation. Finally, given that ANN models are highly nonlinear, it is likely that there exist multiple optima in the objective function. There is, however, no guarantee that the NLS method and the recursive estimation methods discussed above will deliver the global optimum. This is a serious problem because the dimension of the parameter space is typically large. Unfortunately, a convenient and effective method for finding the global optimum in ANN estimation is not yet available.

4.2 Model Complexity Regularization

Section 3 shows that a network model $f_{G,q}$ can approximate unknown function when the number of hidden units, q, is sufficiently large. When there is a fixed training sample, a complex network with a very large q may over fit the data. Thus, there is a trade-off between approximation capability and over-fitting in implementing ANN models.

An easy approach to regularizing the network complexity is to apply a model selection criteria,[6] such as Schwarz (Bayesian) information criterion (BIC), to the network models with various q. As is well known, BIC consists of two terms: one is based on model fitness, and the other penalizes model complexity. Hence, it is suitable for regularizing network complexity; see also Barron (1991). A different criterion introduced in Rissanen (1986, 1987) is *predictive stochastic complexity* (PSC) which is just an average

[6]Alternatively, one may consider testing whether some hidden units may be dropped from the model. This amounts to testing, say, $\beta_h = 0$ for some h. Unfortunately, the parameters in that hidden-unit activation function ($\gamma_{h,0}$ and $\boldsymbol{\gamma}_h$) are not identified under this null hypothesis. It is well known that, when there are unidentified nuisance parameters, standard econometric tests are not applicable.

of squared prediction errors:

$$\text{PSC} = \frac{1}{T-k} \sum_{t=k+1}^{T} \left[y_t - f_{G,q}(\boldsymbol{x}_t, \hat{\boldsymbol{\theta}}_t) \right]^2,$$

where $\hat{\boldsymbol{\theta}}_t$ is the predicted parameter estimate based on the sample information up to time $t-1$, and k is the total number of parameters in the network. Given the number of inputs, the network with the smallest BIC or PSC gives the desired number of hidden units q^*. Rissanen showed that both BIC and PSC can be interpreted as the criteria for "minimum description length," in the sense that they determine the shortest code length (asymptotically) that is needed to encode a sequence of numbers. In other words, these criteria lead to the least complex model that still captures the key information in data. Swanson and White (1997) showed that a network selected by BIC need not perform well in out-of-sample forecasting, however.

Clearly, PSC requires estimating the parameters at each t. It would be computationally demanding if the NLS method is to be used, even for a moderate sample. For simplicity, Kuan and Liu (1995) suggested a two-step procedure for implementing ANN models. In the first step, one estimates the network models and computes the resulting PSCs using the recursive Newton algorithm, which is asymptotically equivalent to the NLS method. When a suitable network structure is determined, the Newton parameter estimates can be used as initial values for NLS estimation in the second step. This approach thus maintains a balance between computational cost and estimator efficiency.

5 Concluding Remarks

In this entry, we introduce ANN model specifications, their approximation properties, and the methods for model implementation from an econometric perspective. It should be emphasized that ANN is neither a magical econometric tool nor a "black box" that can solve *any* difficult problems in econometrics. As discussed above, a major advantage of ANN is its universal approximation property, a property shared by other nonparametric approximators. Yet compared with parametric econometric models, a simple ANN need not perform better, and a more complex ANN (with a large number of hidden units) is more difficult to implement properly and can not be applied when there is only a small data set. Therefore, empirical applications of ANN models must be exercised with care.

References

Barron, A. R. (1991). Complexity regularization with application to artificial neural networks, in *Nonparametric Functional Estimation and Related Topics*, G. Roussas (ed.), pp. 561–576, Dordrecht, the Netherlads: Kluwer Academic Publishers.

Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function, *IEEE Transactions on Information Theory*, **IT-39**, 930–945.

Cybenko, G. (1989). Approximation by superpositions of a sigmoid function, *Mathematics of Control, Signals, and Systems*, **2**, 303–314.

Elman, J. L. (1990). Finding structure in time, *Cognitive Science*, **14**, 179-211.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks, *Neural Networks*, **2**, 183–192.

Gallant, A. R. and H. White (1988). *A Unified Theory of Estimation and Inference for Nonlinear Dynamic Models*, Oxford, UK: Basil Blackwell.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward nets, *Neural Networks*, **4**, 231–242.

Hornik, K. (1993). Some new results on neural network approximation, *Neural Networks*, **6**, 1069–1072.

Hornik, K., M. Stinchcombe, and H. White (1989). Multi-layer feedforward networks are universal approximators, *Neural Networks*, **2**, 359–366.

Hornik, K., M. Stinchcombe, and H. White (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural Networks*, **3**, 551–560.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach, Institute for Cognitive Science Report 8604, UC San Diego.

Kuan, C.-M. (1995) "A recurrent Newton algorithm and its convergence properties", *IEEE Transactions on Neural Networks*, **6**, 779–783.

Kuan, C.-M. and K. Hornik (1991). Convergence of learning algorithms with constant learning rates, *IEEE Transactions on Neural Networks*, **2**, 484–489.

Kuan, C,-M., K. Hornik, and H. White (1994). A convergence result for learning in recurrent neural networks, *Neural Computation*, **6**, 420–440.

Kuan, C.-M. and T. Liu (1995). Forecasting exchange rates using feedforward and recurrent neural networks, *Journal of Applied Econometrics*, **10**, 347–364.

Kuan, C.-M. and H. White (1994). Artificial neural networks: An econometric perspective (with discussions), *Econometric Reviews*, **13**, 1–91 and 139–143.

McClelland, J. L., D. E. Rumelhart, and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 2, Cambridge, MA: MIT Press.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**, 115–133.

Rissanen, J. (1986). Stochastic complexity and modeling, *Annals of Statistics*, **14**, 1080–1100.

Rissanen, J. (1987). Stochastic complexity (with discussions), *Journal of the Royal Statistical Society*, **B**, **49**, 223–239 and 252–265.

Robbins, H. and S. Monro (1951). A stochastic approximation method, *Annals of Mathematical Statistics*, **22**, 400–407.

Rosenblatt, F. (1958). The perception: A probabilistic model for information storage and organization in the brain, *Psychological Reviews*, **62**, 386–408.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning internal representation by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, pp. 318–362, Cambridge, MA: MIT Press.

Rumelhart, D. E., J. L. McClelland, and the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, Cambridge, MA: MIT Press.

Stinchcombe, M. B. and H. White (1998). Consistent specification testing with nuisance parameters present only under the alternative, *Econometric Theory*, **14**, 295–325.

Swanson, N. R. and H. White (1997). A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks, *Review of Economics and Statistics*, **79**, 540–550.

White, H. (1988). Economic prediction using neural networks: The case of IBM stock prices. *Proceedings of the Second Annual IEEE Conference on Neural Networks*, pp. II:451–458. New York: IEEE Press.

White, H. (1989). Some asymptotic results for learning in single hidden layer feedforward network models, *Journal of the American Statistical Association*, **84**, 1003–1013.