CSE-E4810 Machine Learning and Neural Networks (5 cr)

Lecture 11: Processing of Temporal Information

Prof. Juha Karhunen

https://mycourses.aalto.fi/course/view.php?id=13086

Introduction

- This lecture deals with the use of temporal (time) information in neural networks.
- Thus far we have discussed methods where the data (input) vectors can have an arbitrary order.
- However, their **time order** is often an important factor in the data.
- Containing additional information which should be utilized for getting optimal results.
- Examples are speech and video sequences, as well as time series.
- Time-varying data is needed in adaptive control and identification of nonlinear dynamic systems.
- Analysis of time series is an important own branch of science.
- Some examples of time series are:

  - Daily weather measurements at a selected observation station.
  - Exchange rates of currencies, for example euro versus US dollar.
  - Hourly consumption of electricity in Finland.
- Neural networks can predict nonlinearly behaving time series, too.
- Standard linear time series prediction methods are not able to characterize them adequately.
- Prediction of time series for example in the above mentioned examples has a great economic significance.
- Processing of temporal information is considered briefly in Du's and Swamy's book in Chapter 11: Recurrent Neural Networks.
- That chapter deals with recurrent neural networks rather superficially, with no equations.
- Feedforward neural networks for temporal processing are discussed

  there only in Figure 11.2 and its explanation.
- We discuss this topic much more thoroughly based on Chapter 13: "Temporal Processing Using Feedforward Networks".
- In the book S. Haykin, "Neural Networks - A Comprehensive Foundation", Prentice-Hall, 1998.
- A .pdf file of a copy of this chapter is available for your reading and printing on the home page of our course.
- In the later part of this lecture, we discuss briefly recurrent networks.
- Our discussion is based on Chapter 15: "Dynamically Driven Recurrent Networks" of the Haykin's book mentioned above.
- In the new edition of this book, S. Haykin, "Neural Networks and Learning Machines, 3rd ed.", Pearson 2009, this chapter has not changed much.

Goals

- We assume now that the input signals or vectors are **time-varying.**
- That is, the **order of presentation** of the inputs may contain additional information.
- The time t is quantized to discrete time steps t_i which are usually equispaced.
- The time steps t_i are scaled so that they correspond to the sample vectors $\mathbf{x}(i)$.
- The sample vectors $\mathbf{x}(i)$ are often measurements from a continuous (vector-valued) time series $\mathbf{x}(t)$ at time instants t_i.
- At least **three different goals** are possible when incorporating time in the models:
  1. Utilize the short-time sequence as extra **contextual information.**

  2. Cope with **non-stationary data.** Non-stationarity means that the statistics of the data changes with time.
  3. Build a **dynamics** (inner state) into the models.

Types of neural models for temporal processing

- There exists plenty of models and theory for the **linear case.**
- In classical linear discrete-time signal processing, control, estimation theory, and time series modeling.
- Here the aim is to use neural networks as **nonlinear temporal models.**
- Two major types of neural networks can be used to that end:
  1. **Feedforward networks.**
     - Temporal information is incorporated by compiling the input vector from subsequent samples of a scalar input signal.

     - Or more generally from subsequent data vectors.
  2. **Recurrent networks.**
     - Internal states and dynamics are built into recurrent networks by feeding back some of the output signals to the network.
- We consider here variants of the MLP network only.
- Their learning rules are mostly variants of the backpropagation rule.
- The analysis of time-dependent models is more difficult than analysis of static (memoryless) models.
- In this lecture, we introduce and discuss briefly a few models used.
- However, their learning algorithms are discussed very briefly because of lack of time.
- This concerns also general properties of the neural networks used for temporal processing.

Feedforward networks for temporal processing

Delayed samples as a short term memory

- In digital signal processing it is customary to use data vectors

$$\mathbf{x}(n) = [x(n)\ x(n-1)\ \ldots\ x(n-d)]^T \tag{1}$$

  which have been collected from the latest sample $x(n)$ and d preceding samples.
- There the subsequent samples $x(i)$ are obtained by "tapping" the input signal x at equispaced discrete time intervals.
- As depicted in the schematic diagram of Figure 1.
- In other branches of science the vector (1) consists just of subsequent measurement values with no connection to any tapped delay line.

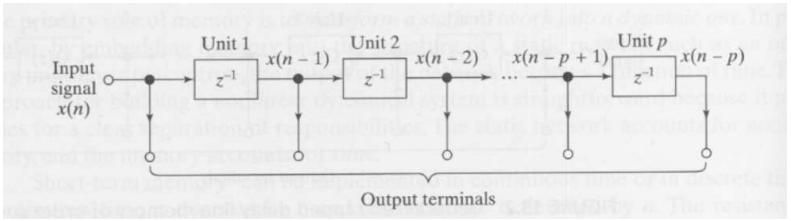

Figure 1: Ordinary tapped delay line memory of order p in digital signal processing. The operator z^{-1} corresponds there unit delay, referring to the z-transform. In a more general case, the unit delay operators can be replaced by operators $G(z)$ filtering the signals at the same time before sampling them.

- One can use a standard static network but utilize the time sequence in forming the inputs.
- Subsequent time samples are used to form the short-time memory in Figure 2.
- The static network can be for example a multilayer perceptron (MLP) network or a radial basis function (RBF) network.
- In this way, one can build the nonlinear filter depicted schematically in Figure 2.
- The simplest approach is to use the input vector (1) collected of subsequent samples of the data.
- More generally, one can use suitable filters to preprocess the inputs.
- For example wavelets or short-time Fourier transforms.

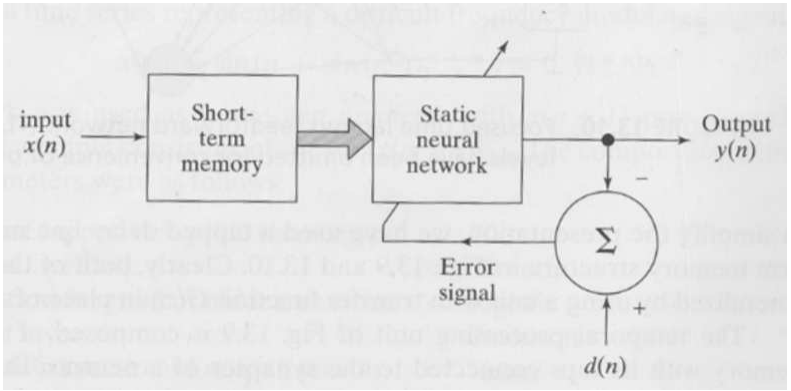

Figure 2: Nonlinear filter built on a static neural network.

Example: NETtalk

- NETtalk is an early demonstration of the general structure shown in Figure 2.
- It was devised by Sejnowski and Rosenberg in 1987.
- They used a static MLP network, taught with the standard backpropagation algorithm.
- There were 203 nodes in the input layer, a hidden layer of 80 neurons, and an output layer of 26 neurons.
- All the neurons used sigmoid (logistic) activation functions.
- The network had seven groups of nodes in the input layer.
- Each group coded one letter of the input text.
- Thus strings of seven subsequent letters were presented to the input layer at any one time.

- The NETtalk network converts English text to phonemes that can then be fed to a speech synthesizer:

  bird $\rightarrow$ /böörd/

- The delay line provides the **context.**
- The **result** was a very nice demo of human interest:
- During learning the performance improved gradually.
- At some stages the output resembled "babbling" of babies, and later NETtalk made similar mistakes as kids do.
- Its performance degraded very slowly as synaptic connections in the network were damaged.
- Relearning after damage was much faster than learning during the original training.

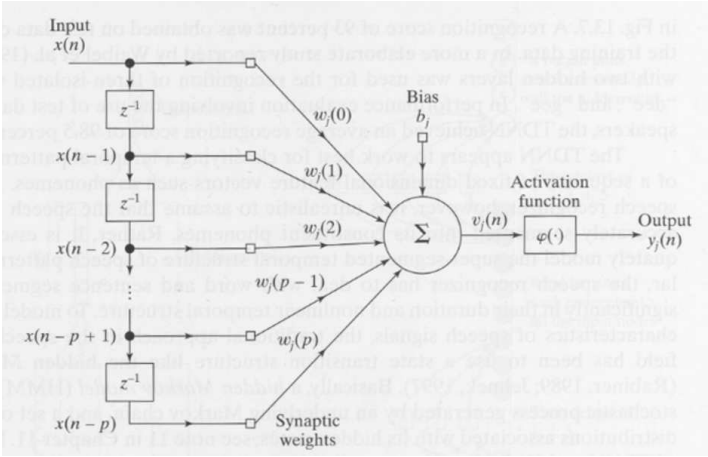

Figure 3: Focused neuronal filter.

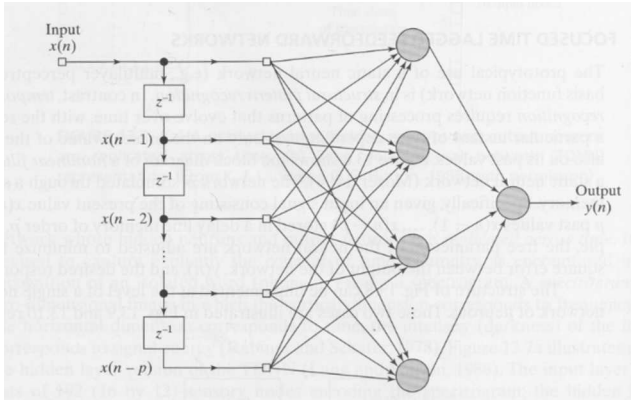

Figure 4: Focused time lagged feedforward network (TLFN). The bias terms have been omitted for convenience.

Simple feedforward structures

- The general structure of Figure 2 can be implemented at the level of:
  - Single neuron, leading to the **focused neuronal filter** shown in Figure 3.
  - A network of neurons, leading to the **focused time lagged feedforward network (TLFN)** shown in Figure 4.
- The output of the focused neuronal filter in Figure 3 is

$$y_j(n) = \varphi(\sum_{l=0}^{p} w_j(l)x(n-l) + b_j) \qquad (2)$$

- It is also the output of the j:th neuron in the hidden layer of the TLFN network depicted in Figure 4.
- This output is the **convolution** of the input signal $x(n-l)$ and the synaptic weights $w_j(l)$ of the neuron j.

- Convolutions could be analyzed conveniently using the Z-transform.
- The final output of the TLFN network is simply the linear combination of the outputs $y_j(n)$ of its m hidden layer neurons:

$$y(n) = \sum_{j=1}^{m} w_j y_j(n) \tag{3}$$

- The network of Figure 4 is still relatively simple.
- It could be generalized by having:
  - More than one neuron in the output layer;
  - A nonlinear output neuron or output layer;
  - More hidden layers.
- This type of neural networks have already fairly good approximation properties.
- They have nice properties, and it is easy to design such networks,

since:

- The delays occur only in the preprocessing stage ⇒ it is relatively easier to analyze the network.
- Time-dependent processing takes place only in the delay line or more generally in preprocessing filters ⇒ the network is stable if the filters are.
- A familiar network, for example MLP or RBF network, can be used as the static network.

Time series prediction experiment

- This computer experiment is about prediction of a "difficult" frequency modulated nonlinear signal

$$x(n) = \sin(n + \sin(n^2)) \tag{4}$$

- The next value $x(n+1)$ of this time series was predicted using 20

last samples $x(n), x(n-1) \ldots, x(n-19)$.

- The focused TLFN network of Figure 4 was used for prediction.
- There were 10 neurons in the hidden layer.
- The activation function of hidden neurons was the standard logistic (sigmoidal) nonlinearity.
- The output layer consisted of one linear neuron.
- The TLFN network was trained using the standard backpropagation algorithm using 500 randomized training vectors containing each 20 subsequent samples from the time series.
- Figure 5 shows that the prediction results are quite good, with a mean-square prediction error of 1.2×10^{-3}.
- This is because 20 known values are used for just one-step prediction ⇒ the prediction task is fairly easy.

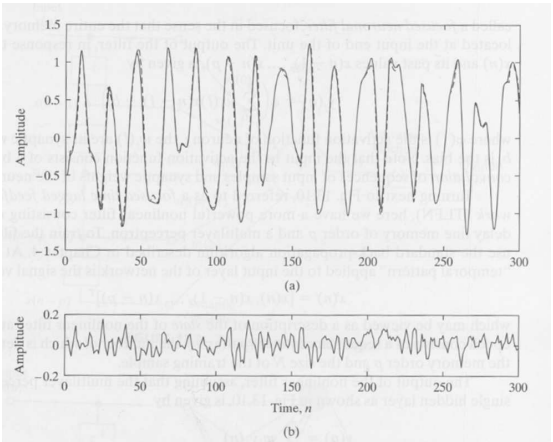

Figure 5: Results of one-step prediction experiment. (a) Actual (continuous) and predicted (dashed) signals. (b) Prediction error.

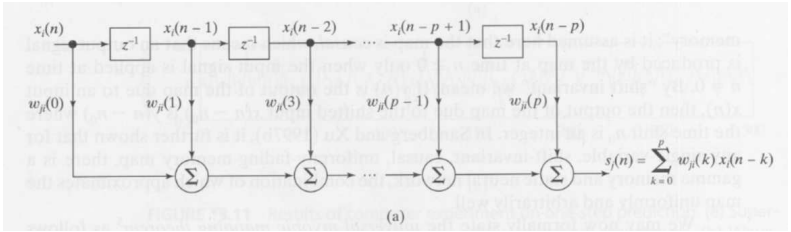

Figure 6: Linear finite-duration impulse response (FIR) filter.

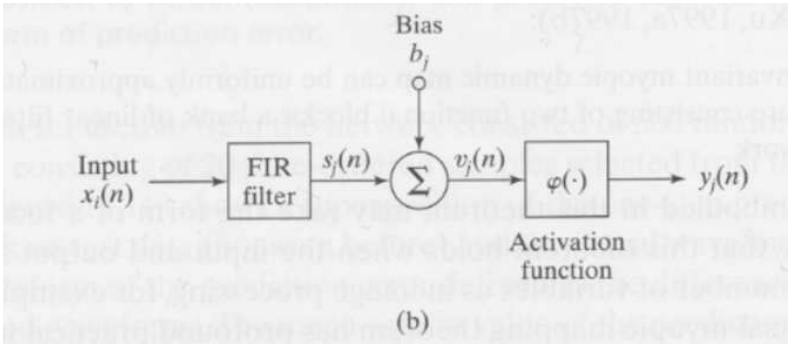

Figure 7: Interpretation of focused neuronal filter as a nonlinear FIR filter.

Spatio-temporal models of a neuron

- Figure 6 shows the schematic diagram of a standard **finite-duration impulse response (FIR) filter.**
- It is a basic tool in linear digital signal processing.
- The focused neuronal filter of Figure 3 can be interpreted as the **nonlinear FIR filter** shown in Figure 7.
- Its output is given by

$$y_j(n) = \varphi(v_j(n)) = \varphi(s_j(n) + b_j) \tag{5}$$

- Here the output $s_j(n)$ of the linear FIR filter having $p+1$ taps and input signal $x_i(n)$ is the convolution sum

$$s_j(n) = \sum_{k=0}^{p} w_{ji}(k) x_i(n-k) \tag{6}$$

- Recall that $\varphi(\cdot)$ is the nonlinearity (usually sigmoidal), b_j is the bias term of jth neuron, and $v_j(n)$ its linear response.
- The subscripts i and j could be dropped out from Figure 6 for now, but they are needed later on.
- Figure 8 shows another structure, so-called **multiple input neuronal filter**.
- There each input signal $x_i(n)$ is first passed through its own linear FIR filter, yielding respective output signal $s_{ji}(n)$.
- The outputs of these FIR filters are then combined in standard manner, providing the final output

$$y_j(n) = \varphi(\sum_{i=1}^{m_0} \sum_{l=0}^{p} w_{ji}(l) x_i(n-l) + b_j) \tag{7}$$

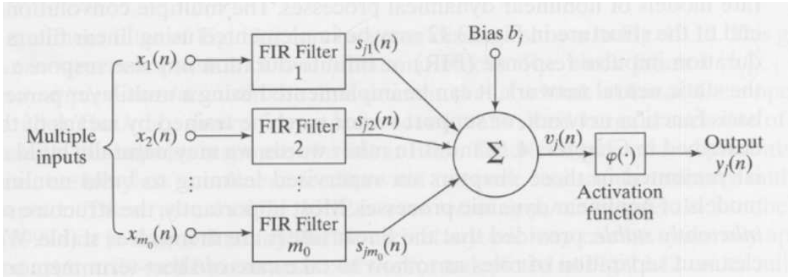

Figure 8: Multiple input neuronal filter.

- Here $w_{ji}(l)$ is the l:th weight of the ith FIR filter, corresponding to the ith input signal $x_i(n-l)$ at lag value l.
- The multiple input neuronal filter differs from the focused time lagged feedforward network (TLFN) in Figure 4.
- In Figure 8, there are m_0 **different** input signals $x_1(n), \ldots, x_{m_0}(n)$ at the same time instant n.
- In TLFN, the inputs $x(n), \ldots, x(n-p)$ of the neuron are subsequent samples from the **same** scalar time series $\{x(n)\}$.
- Multiple input neuronal filter processes input signals both **spatially and temporally.**
- The FIR filters process each input signal $x_i(n)$ temporally.
- Combining the outputs of the FIR filters then provides spatial processing of the m_0 input signals $x_1(n), \ldots, x_{m_0}(n)$.

Distributed time lagged feedforward networks

- The focused time lagged feedforward networks (TLFN) are restricted in use.
- They are limited to mappings which are shift invariant.
- Therefore they are not suitable for nonstationary environments.
- This limitation can be overcome by using the multiple input neuronal filter of Figure 8.
- There the implicit influence of the time is distributed throughout the network.
- Such structures are called **distributed time lagged feedforward networks** or distributed TLFN's.
- The output (7) of the multiple input neuronal filter can be expressed more conveniently in vector form.

- Denote the vector which contains subsequent (tapped) values of the ith input signal $x_i(n)$ to the neuron by

$$\mathbf{x}_i(n) = [x_i(n), x_i(n-1), \ldots, x_i(n-p)]^T \tag{8}$$

- The weight vector that implements the FIR filter used for processing the ith input signal of neuron j is denoted by

$$\mathbf{w}_{ji} = [w_{ji}(0), w_{ji}(1), \ldots, w_{ji}(p)]^T \tag{9}$$

- The FIR filtering can then be represented as the inner product $\mathbf{w}_{ji}^T \mathbf{x}_i$.
- The output of the neuron j in (7) becomes

$$y_j(n) = \varphi\left(\sum_{i=1}^{m_0} \mathbf{w}_{ji}^T \mathbf{x}_i(n) + b_j\right) \tag{10}$$

- Note that this resembles closely the neuronal model used in

standard multilayer perceptron (MLP) networks.

- Difference: $\mathbf{w}_{ji}$ and $\mathbf{x}_i(n)$ are now **vectors,** while in standard MLP they are **scalars.**

Temporal back-propagation algorithm

- Consider now training of the distributed TLFN network.
- We consider a network that is a generalization of the multiple input neuronal filter of Figure 8.
- Because now there are several neurons in the output layer.
- The goal is to derive a supervised backpropagation (BP) type learning algorithm for such a network.
- The performance criterion is the mean-square error of the outputs over time, which should be minimized.

- The mean-square output error at time step n is

$$\mathcal{E}(n) = \sum_j e_j^2(n) = \sum_j [d_j(n) - y_j(n)]^2 \tag{11}$$

- There y_j is the output of neuron j in the output layer and d_j is the corresponding desired response (output).
- And $e_j(n)$ is the error of neuron j in the output layer at time step n.
- The total **cost function** is

$$\mathcal{E}_{\text{total}} = \sum_n \mathcal{E}(n) \tag{12}$$

- The cost functions (11) and (12) are similar as for the standard MLP network.
- The standard BP algorithm is derived by differentiating the cost function (12) with respect to the weights w_{ij} of the MLP network.

- Now one can proceed in a similar manner, but by differentiating (12) with respect to the weight vectors $\mathbf{w}_{ji}$ in (10).
- It is possible to derive a relatively simple expression for the gradient

$$\frac{\partial \mathcal{E}_{\text{total}}}{\partial \mathbf{w}_{ji}} \tag{13}$$

of the cost function for each weight vector $\mathbf{w}_{ji}$.

- The algorithm is then simply a gradient descent algorithm based on these gradients.
- The derivation is presented in Section 13.9 of the book S. Haykin, "Neural Networks: A Comprehensive Foundation", 2nd ed., Prentice-Hall 1998.
- It is lengthy and somewhat involved, but proceeds fairly similarly as for the standard BP algorithm.

- The core of this **temporal backpropagation algorithm** are the following **update formulas:**
  - The weight vectors are updated by

$$\mathbf{w}_{ji}(n+1) = \mathbf{w}_{ji}(n) + \eta \delta_j(n) \mathbf{x}_i(n) \tag{14}$$

  - The local gradient $\delta_j(n)$ at time step n is defined by

$$\delta_j(n) = e_j(n) \varphi'(v_j(n)) \tag{15}$$

for neurons j in the output layer, and by

$$\delta_j(n) = \varphi'(v_j(n)) \sum_r \mathbf{\Delta}_r^T(n) \mathbf{w}_{rj} \tag{16}$$

if the neuron j is in the hidden layer.

- In the formulas (14), (15), and (16):
  - η is the learning parameter (depends generally on time n)
  - $\varphi'(\cdot)$ is the derivative of the common nonlinear activation function $\varphi(\cdot)$ of the neurons;
  - $v_j(n)$ is the linear response of neuron j at time step n:

$$v_j(n) = \sum_{i=1}^{m_0} \mathbf{w}_{ji}^T \mathbf{x}_i(n) + b_j$$

  - And $\mathbf{\Delta}_r(n)$ is a vector of local gradients:

$$\mathbf{\Delta}_r(n) = [\delta_r(n), \delta_r(n+1), \ldots, \delta_r(n+p)]^T$$

- Positive aspects:
  - The algorithm consists of a simple forward phase and an error backpropagation phase like the normal BP algorithm.
  - In fact, the algorithm is a generalization of the standard BP algorithm to vector-valued data.
  - Computationally it is more demanding than the learning algorithm for the simpler "focused" time-lagged network.
  - The algorithm is very accurate: it has won a contest on predicting a nonlinear nonstationary time series.
- The algorithm is **non-causal** but it can be made causal with certain modifications if necessary.
- See again Section 13.9 in Haykin's book for details.
- Non-causality means that the algorithm depends on future inputs.

Recurrent networks

- Recurrent networks are neural networks involving **feedback**.
- That is, the outputs of some neurons or usually a layer of neurons are connected to their inputs.
- One can distinguish between
  - **local** feedback: feedback within single neurons.
  - **global** feedback: feedback encompassing the whole network.
- We consider only global feedback, either from the output layer or the hidden layer to the inputs.
- As we saw, temporal feedforward networks are generalization of the FIR (finite impulse response) filters in signal processing.
- Recurrent neural networks are similarly generalization of the IIR (infinite impulse response) filters in signal processing.

- In this course, we consider very briefly recurrent neural networks, skipping their theory and learning algorithms.
- These matters are discussed in chapters 15: "Dynamically Driven Recurrent Networks", in S. Haykin's books mentioned on Slide 4 of this lecture.
- Recurrent neural networks can be used for

1. Constructing an **associative memory.**
   - An example is the Hopfield network discussed in Chapter 6 in Du's and Swamy's book.
   - We skip in this course Hopfield network and associative memories (Chapter 7 in Du's and Swamy's book) because they are now largely out-of-date topics.
2. Constructing **nonlinear input-output mappings.**

Pros and cons of recurrent networks

- Consider the benefits and drawbacks of recurrent nonlinear networks compared with time-lagged feedforward networks on a general level:

+ The delay lines needed in temporal feedforward networks to represent adequately the dependence between the input signal $x(n)$ and the output $y(n)$ can be extremely long.

+ The recurrent networks may **potentially** save memory.

– The learning algorithms of recurrent neural networks are computationally heavy and converge slowly.

– Time-lagged feedforward networks are simpler.

– When applied improperly, feedback can produce harmful effects. In particular, a system that is originally stable can become unstable with application of feedback.

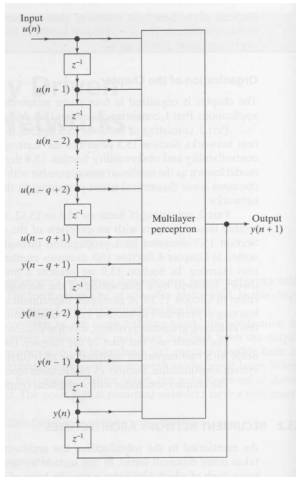

Figure 9: Neural network implementation of the NARX model.

The NARX model and network structure

- Figure 9 shows the architecture of the **NARX network.**
- The abbreviation NARX stands for a **nonlinear autoregressive model with exogenous inputs.**
- Exogenous inputs mean inputs originating outside the network.
- The input signal $u(n)$ comes in through a tapped delay line.
- The (scalar) output $y(n)$ of the NARX network is fed back to the input through another tapped delay line.
- The actual network is the familiar static multilayer perceptron.
- Denote by $F(\cdot)$ the nonlinear function formed by the multilayer perceptron network.

- The output (prediction) of the NARX network in Figure 9 at time $n+1$ is

$$y(n+1) = F(y(n), \ldots, y(n-q+1), u(n), \ldots, u(n-q+1)) \quad (17)$$

Relation of the NARX model to linear time series models

- Consider the relationship of the NARX model to standard linear models used in digital signal processing and time series analysis.
- A popular model is the linear **autoregressive (AR) model** of order M:

$$y(n+1) = -a_0 y(n) - a_1 y(n-1) \ldots - a_{M-1} y(n-M+1) + u(n) \quad (18)$$

- There $y(n)$ is the output and the input signal $u(n)$ is white noise, making the process stochastic.
- The coefficients $a_0, a_1, \ldots, a_{M-1}$ are constants.

- The AR model is used widely because its coefficients can be estimated from the available data by solving linear equations.
- The AR model describes well for example sinusoidal type signals.
- Another basic linear model is the **moving average (MA) model** of order N:

$$y(n+1) = u(n) + b_1 u(n-1) + b_2 u(n-2) \dots b_{n-N} u(n-N) \quad (19)$$

- The coefficient of the first term $u(n)$ can be taken $b_0 = 1$ without a loss of generality.
- The terms $u(n), \dots, u(n-N)$ are all white noise.
- Estimating the coefficients of the MA model requires nonlinear methods.
- Therefore the MA model is not so popular as the AR model.

- The AR and MA models can be combined to linear **autoregressive moving average (ARMA) model:**

$$y(n+1) + \sum_{i=0}^{M-1} a_i y(n-i) = u(n) + \sum_{j=1}^{N} b_i u(n-i) \quad (20)$$

- Sometimes a constant term c is added to the AR, MA, and ARMA models, but it is often assumed or has preprocessed to zero.
- The ARMA model is the most general model used in linear discete-time digital signal processing.
- We see that the NARX model is actually a nonlinear generalization of the linear ARMA model.
- With equal number q of lagged input signals u and output signals y.

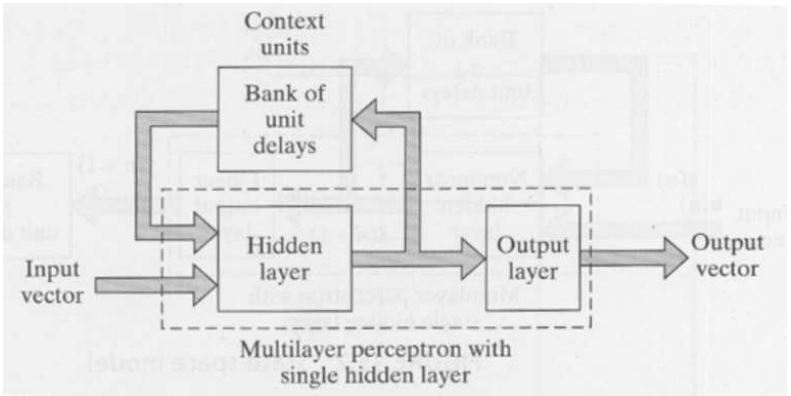

Figure 10: Block diagram of simple recurrent network.

Simple recurrent network and its state-space model

- Consider briefly the simple recurrent network (SRN) proposed by Elman (1990).
- Its schematic block diagram is shown in Figure 10.
- Simple recurrent network can be described mathematically using a **state-space model.**
- The hidden neurons are considered to represent the **state** of the network, and their activities are fed back to the input.
- The inner state $\mathbf{x}$ of the network at time $n+1$ is therefore a nonlinear function $\boldsymbol{\varphi}(\cdot)$ of the external inputs $\mathbf{u}$ and the inner state at time n:

$$\mathbf{x}(n+1) = \boldsymbol{\varphi}(\mathbf{x}(n), \mathbf{u}(n)) \quad (21)$$

- The output of the whole network $\mathbf{y}$ is a function of the inner state, which generally depends nonlinearly on the state vector $\mathbf{x}$.
- We consider for simplicity a linear output vector

$$\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n) \tag{22}$$

- The hidden state functions as a "dynamic memory" that is modified in a nonlinear manner.
- Figure 11 presents the state-space model of this network.
- The equation (21) describing the evolution of the inner state $\mathbf{x}(n)$ of the simple recurrent network can be presented in more detail.
- Assume that the weight matrix connecting the inputs to the hidden units is $\mathbf{W}_b$.

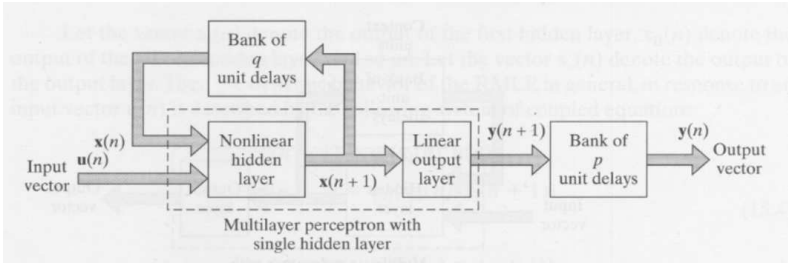

Figure 11: State-space model of simple recurrent network having a linear output layer.

- And the weight matrix connecting the feedback to the hidden units is $\mathbf{W}_a$.
- The "state-space" network is then described by the equations

$$\mathbf{x}(n+1) = \boldsymbol{\varphi}(\mathbf{W}_a\mathbf{x}(n) + \mathbf{W}_b\mathbf{u}(n)) \tag{23}$$

$$\mathbf{y}(n) = \mathbf{C}\mathbf{x}(n) \tag{24}$$

- Generally, the dimensions of the vectors $\mathbf{u}(n)$, $\mathbf{x}(n)$, and $\mathbf{y}(n)$ are different: q, m, and p, respectively.
- Of course, the weight matrices $\mathbf{W}_a$, $\mathbf{W}_b$, and $\mathbf{C}$ must have compatible dimensions.
- The matrix $\mathbf{W}_b$ contains bias terms, too.
- The componentwise nonlinearities φ in the vector $\boldsymbol{\varphi}$ are typically sigmoids, either $\tanh(\alpha t)$ or the logistic sigmoidal function.

Learning algorithms of recurrent networks

- Several learning algorithms are available for recurrent networks.
- We just briefly list them; their details, derivations and theory can be found for example in Chapter 15 of Haykin's book.
- In the **back-propagation through time** learning algorithm, the temporal operation of the recurrent network is unfolded.
- Into a layered feedforward network which grows at each step by one layer.
- In **real-time recurrent learning** , the state-space equation (23) is differentiated directly for developing a learning algorithm.
- Finally, **decoupled extended Kalman filter** learning algorithm is developed by linearizing the nonlinear state-space model around the current operation point.

Applications of temporal processing

- Major applications of temporal feedforward and recurrent networks are:
- Prediction of nonlinear time series.
- Identification of nonlinear dynamic systems.
- Adaptive control of nonlinear systems.

Concluding remarks

- Generally, recurrent networks can yield in time series prediction more accurate results than feedforward networks.
- But they are in many ways more difficult to use.

- Instead of MLP networks, extreme learning machines, radial basis function networks, self-organizing maps, and support vectors machines have been used in time series prediction.
- The AML (applications of machine learning) research group in our department has studied time series prediction.
- See http://research.ics.aalto.fi/eiml/publications.shtml for its publications.
- The AML group has used extreme learning machine and Gaussian processes in time series prediction and some of its applications.