Neural Networks for Time Series Prediction

15-486/782: Artificial Neural Networks

Fall 2006

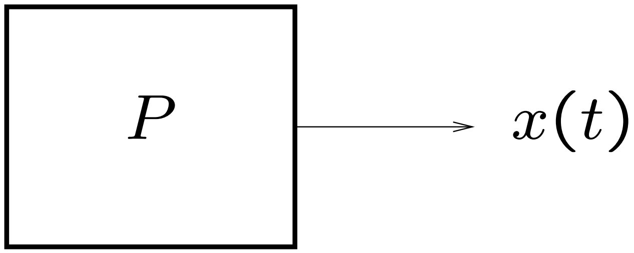
(based on earlier slides by Dave Touretzky and Kornel Laskowski)

What is a Time Series?

A sequence of vectors (or scalars) which depend on time t. In this lecture we will deal exclusively with scalars:

$$\{\ x(t_0),\ x(t_1),\ \cdots\ x(t_{i-1}),\ x(t_i),\ x(t_{i+1}),\ \cdots\ \}$$

It's the output of some process P that we are interested in:



Examples of Time Series

- Dow-Jones Industrial Average
- sunspot activity
- electricity demand for a city
- number of births in a community
- air temperature in a building

These phenomena may be *discrete* or *continuous*.

Discrete Phenomena

- Dow-Jones Industrial Average closing value each day
- sunspot activity each day

Sometimes data have to be aggregated to get meaningful values. Example:

- births per minute might not be as useful as births per month

Continuous Phenomena

t is real-valued, and $x(t)$ is a continuous signal.

To get a series $\{x[t]\}$, must *sample* the signal at discrete points.

In uniform sampling, if our sampling period is Δt, then

$$\{x[t]\} = \{x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), \cdots\} \tag{1}$$

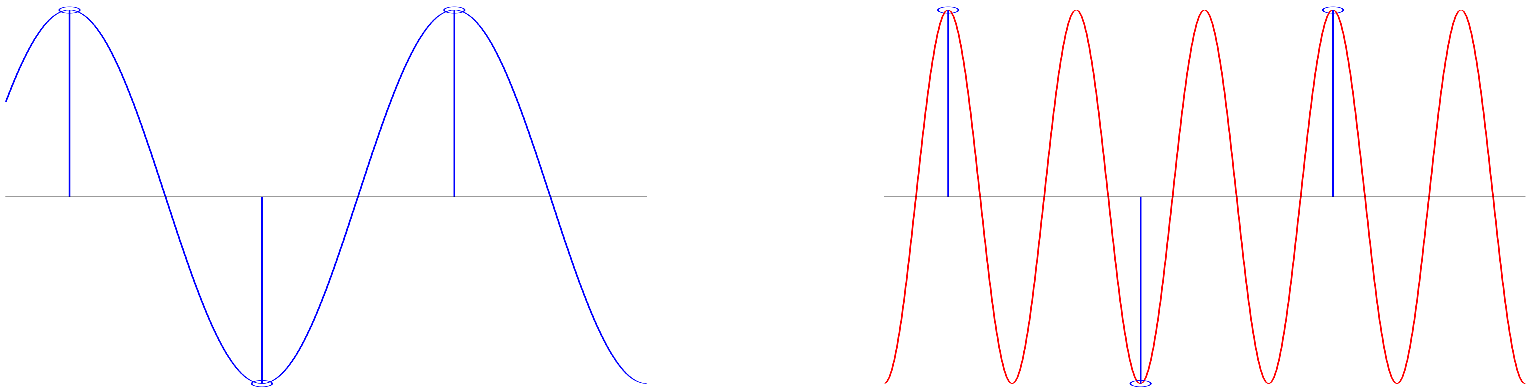
To ensure that $x(t)$ can be recovered from $x[t]$, Δt must be chosen according to the Nyquist sampling theorem.

Nyquist Sampling Theorem

If f_{max} is the highest frequency component of $x(t)$, then we must sample at a rate at least twice as high:

$$\frac{1}{\Delta t} = f_{sampling} > 2f_{max} \tag{2}$$

Why? Otherwise we will see aliasing of frequencies in the range $[f_{sampling}/2, f_{max}]$.



Studying Time Series

In addition to describing either discrete or continuous phenomena, time series can also be deterministic vs stochastic, governed by linear vs nonlinear dynamics, etc.

Time series are the focus of several overlapping disciplines:

- Information Theory deals with describing stochastic time series.

- Dynamical Systems Theory deals with describing and manipulating mostly non-linear deterministic time series.

- Digital Signal Processing deals with describing and manipulating mostly linear time series, both deterministic and stochastic.

We will use concepts from all three.

Possible Types of Processing

- **predict** future values of $x[t]$
- **classify** a series into one of a few classes

  "price will go up"

  "price will go down" — sell now

  "no change"
- **describe** a series using a few parameter values of some model
- **transform** one time series into another

  oil prices $\mapsto$ interest rates

The Problem of Predicting the Future

Extending backward from time t, we have time series $\{x[t], \ \ x[t-1], \ \ \cdots\}$. From this, we now want to estimate x at some future time

$$\widehat{x}[t+s] \ = \ f(\ x[t], \ x[t-1], \ \ \cdots \ )$$

s is called the *horizon of prediction*. We will come back to this; in the meantime, let's predict just one time sample into the future, $\Rightarrow \ \ s=1$.

This is a function approximation problem.

Here's how we'll solve it:

1. Assume a generative model.
2. For every point $x[t_i]$ in the past, train the generative model with what preceded t_i as the `Inputs` and what followed t_i as the `Desired`.
3. Now run the model to predict $\widehat{x}[t+s]$ from $\{x[t], \cdots\}$.

Embedding

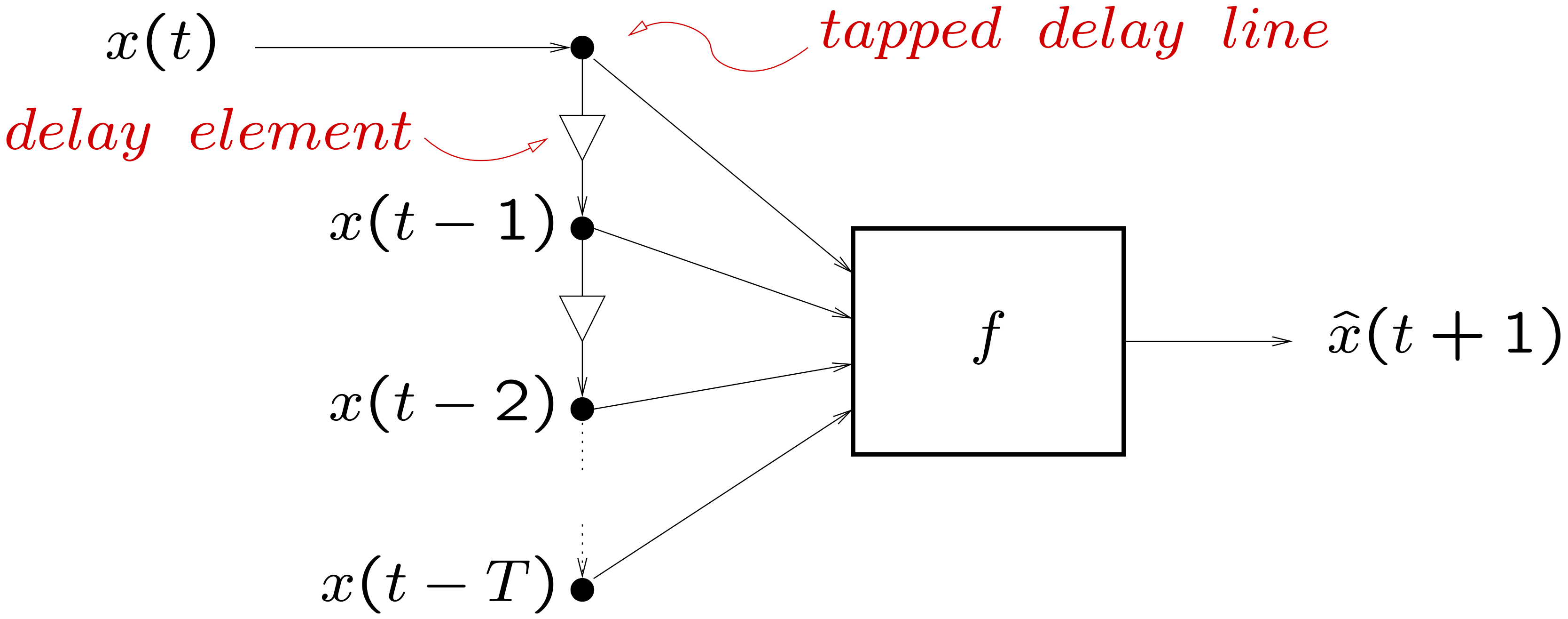
Time is constantly moving forward. Temporal data is hard to deal with...

If we set up a shift register of delays, we can retain successive values of our time series. Then we can treat each past value as an additional *spatial* dimension in the input space to our predictor.

This implicit transformation of a one-dimensional time vector into an infinite-dimensional spatial vector is called *embedding*.

The input space to our predictor must be finite. At each instant t, truncate the history to only the previous d samples. d is called the embedding dimension.

Using the Past to Predict the Future



Linear Systems

It's possible that P, the process whose output we are trying to predict, is governed by linear dynamics.

The study of linear systems is the domain of Digital Signal Processing (DSP).

DSP is concerned with linear, translation-invariant (LTI) operations on data streams. These operations are implemented by *filters*. The analysis and design of filters effectively forms the core of this field.

Filters operate on an input sequence $u[t]$, producing an output sequence $x[t]$. They are typically described in terms of their frequency response, ie. low-pass, high-pass, band-stop, etc.

There are two basic filter architectures, known as the FIR filter and the IIR filter.

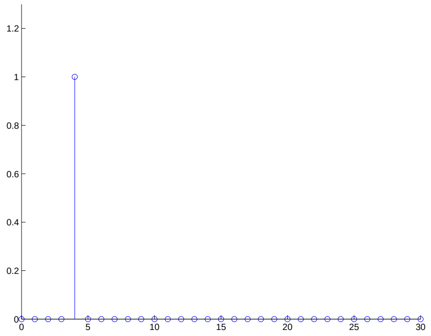
Finite Impulse Response (FIR) Filters

Characterized by $q+1$ coefficients:

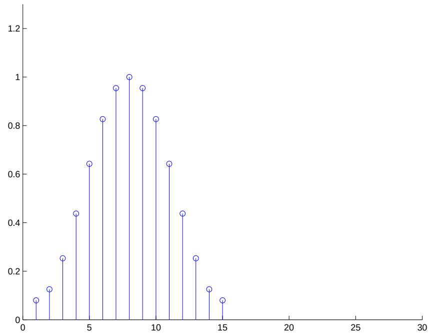
$$x[t] \;=\; \sum_{i=0}^{q} \beta_i \, u[t-i] \tag{3}$$

FIR filters implement the convolution of the input signal with a given coefficient vector $\{\beta_i\}$.

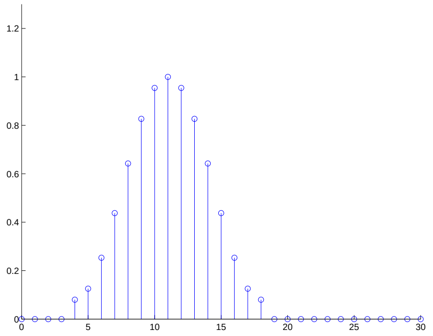
They are known as *Finite Impulse Response* because, when the input $u[t]$ is the impulse function, the output x is only as long as $q+1$, which must be finite.



IMPULSE



FILTER



RESPONSE

Infinite Impulse Response (IIR) Filters

Characterized by p coefficients:

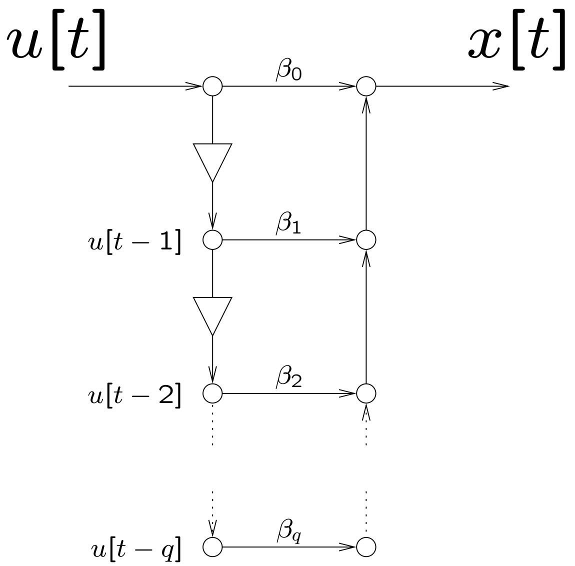
$$x[t] \;=\; \sum_{i=1}^{p} \alpha_i \, x[t-i] + u[t] \tag{4}$$

In IIR filters, the input $u[t]$ contributes directly to $x[t]$ at time t, but, crucially, $x[t]$ is otherwise a weighed sum of *its own past samples*.

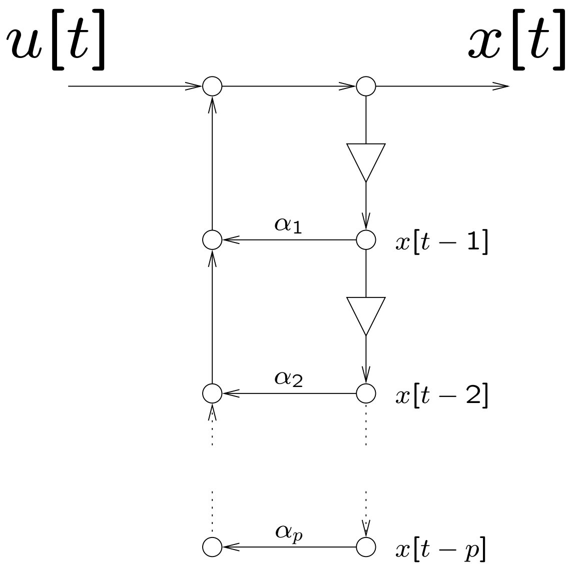
These filters are known as *Infinite Impulse Response* because, in spite of both the impulse function and the vector $\{\alpha_i\}$ being finite in duration, the response only asympotically decays to zero. Once one of the $x[t]$'s is non-zero, it will make non-zero contributions to future values of $x[t]$ ad infinitum.

FIR and IIR Differences

In DSP notation:



FIR



IIR

DSP Process Models

We're interested in modeling a particular process, for the purpose of predicting future inputs.

Digital Signal Processing (DSP) theory offers three classes of possible linear process models:

- Autoregressive (AR[p]) models
- Moving Average (MA[q]) models
- Autoregressive Moving Average (ARMA[p, q]) models

Autoregressive (AR$[p]$) Models

An AR$[p]$ assumes that at its heart is an IIR filter applied to some (unknown) internal signal, $\epsilon[t]$. p is the order of that filter.

$$x[t] \;=\; \sum_{i=1}^{p} \alpha_i \, x[t-i] \;+\; \epsilon[t] \tag{5}$$

This is simple, but adequately describes many complex phenomena (ie. speech production over short intervals).

If on average $\epsilon[t]$ is small relative to $x[t]$, then we can estimate $x[t]$ using

$$\begin{aligned} \widehat{x}[t] &\equiv x[t] \;-\; \epsilon[t] && (6) \\ &= \sum_{i=1}^{p} w_i \, x[t-i] && (7) \end{aligned}$$

This is an FIR filter! The w_i's are estimates of the α_i's.

Estimating AR$[p]$ Parameters

Batch version:

$$
\begin{aligned}
x[t] &\approx \widehat{x}[t] && (8)\\
&= \sum_{i=1}^{p} w_i \, x[t-i] && (9)
\end{aligned}
$$

$$
\begin{bmatrix} x[p+1] \\ x[p+2] \\ \vdots \end{bmatrix} = \begin{bmatrix} x[1] & x[2] & \cdots & x[p] \\ x[2] & x[3] & \cdots & x[p+1] \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix} \cdot \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}}_{\mathbf{w}} \qquad (10)
$$

Can use linear regression. Or **LMS**.

Application: speech recognition. Assume that over small windows of time, speech is governed by a static AR$[p]$ model. To learn $\mathbf{w}$ is to characterize the vocal tract during that window. This is called Linear Predictive Coding (LPC).

Estimating AR$[p]$ Parameters

Incremental version (same equation):

$$\begin{aligned} x[t] &\approx \widehat{x}[t] \\ &= \sum_{i=1}^{p} w_i \, x[t-i] \end{aligned}$$

For each sample, modify each w_i by a small Δw_i to reduce the sample squared error $(x[t] - \widehat{x}[t])^2$. One iteration of LMS.

Application: noise cancellation. Predict the next sample $\widehat{x}[t]$ and generate $-\widehat{x}[t]$ at the next time step t. Used in noise cancelling headsets for office, car, aircraft, etc.

Moving Average (MA$[q]$) Models

A MA$[q]$ assumes that at its heart is an FIR filter applied to some (unknown) internal signal, $\epsilon[t]$. $q+1$ is the order of that filter.

$$x[t] \;=\; \sum_{i=0}^{q} \beta_i \epsilon[t-i] \qquad (11)$$

Sadly, cannot assume that $\epsilon[t]$ is negligible; $x[t]$ would have to be negligible. If our goal was to describe a noisy signal $x[t]$ with specific frequency characteristics, we could set $\epsilon[t]$ to white noise and the $\{w_i\}$ would just subtract the frequency components that we do not want.

Seldom used alone in practice. By using Eq 11 to estimate $x[t]$, we are not making explicit use of past values of $x[t]$.

Autoregressive Moving Average (ARMA$[p, q]$) Models

A combination of the AR$[p]$ and MA$[q]$ models:

$$x[t] \;=\; \sum_{i=1}^{p} \alpha_i x[t-i] \;+\; \sum_{i=1}^{q} \beta_i \epsilon[t-i] \;+\; \epsilon[t] \tag{12}$$

To estimate future values of $x[t]$, assume that $\epsilon[t]$ at time t is small relative to $x[t]$. We can obtain estimates of past values of $\epsilon[t]$ at time $t-i$ from past true values of $x[t]$ and past values of $\widehat{x}[t]$:

$$\widehat{\epsilon}[t-i] \;=\; x[t-i] \;-\; \widehat{x}[t-i] \tag{13}$$

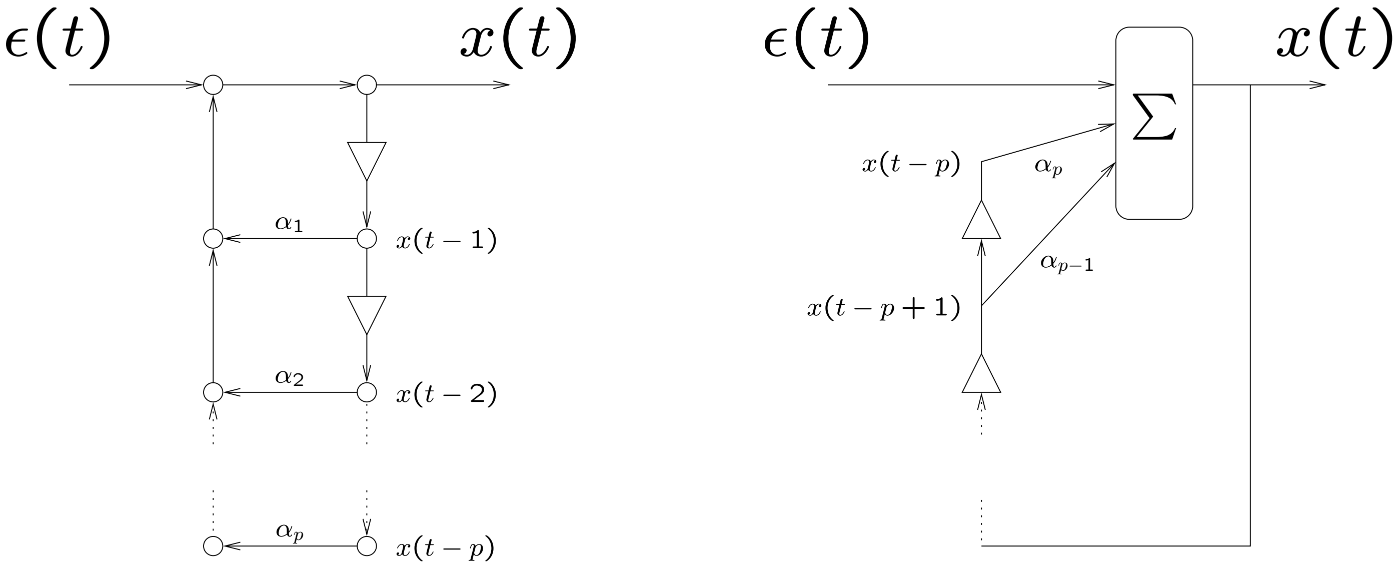
The estimate for $x[t]$ is then

$$\widehat{x}[t] \;=\; \sum_{i=1}^{p} \alpha_i x[t-i] \;+\; \sum_{i=1}^{q} \beta_i \widehat{\epsilon}[t-i] \tag{14}$$

Linear DSP Models as Linear NNs

DSP Filter	DSP Model	NN Connections
FIR	MA[q]	feedforward
IIR	AR[p]	recurrent

An AR[p] model is equivalent to:



Train using backprop as in Eq 11.

Nonlinear AR$[p]$ Models

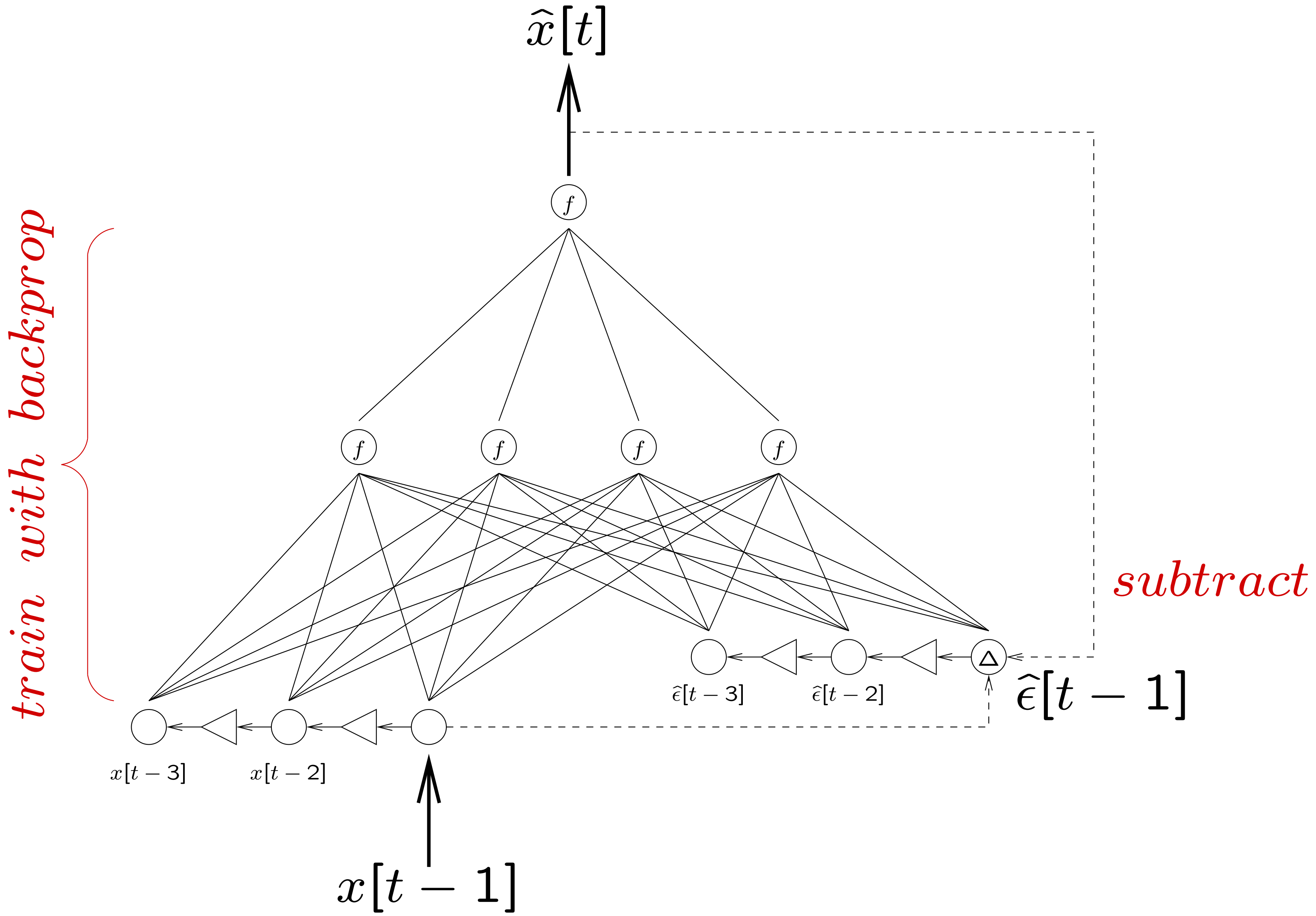
Once we've moved to NNs, there's nothing to stop us from replacing the $\sum$'s with a nonlinear activation function like $\tanh(\sum)$.

Non-linear models are more powerful, but need more training data, and are less well behaved (overfitting, local minima, etc).

TDNNs can be viewed as NAR$[p]$ models.

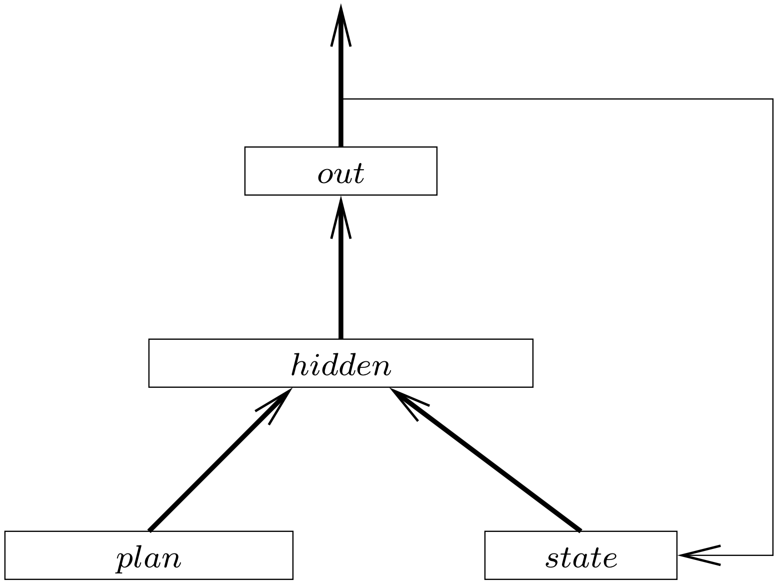
An example of nonlinear ARMA neural net ... (next slide)

Nonlinear ARMA$[p, q]$ Models



Jordan Nets

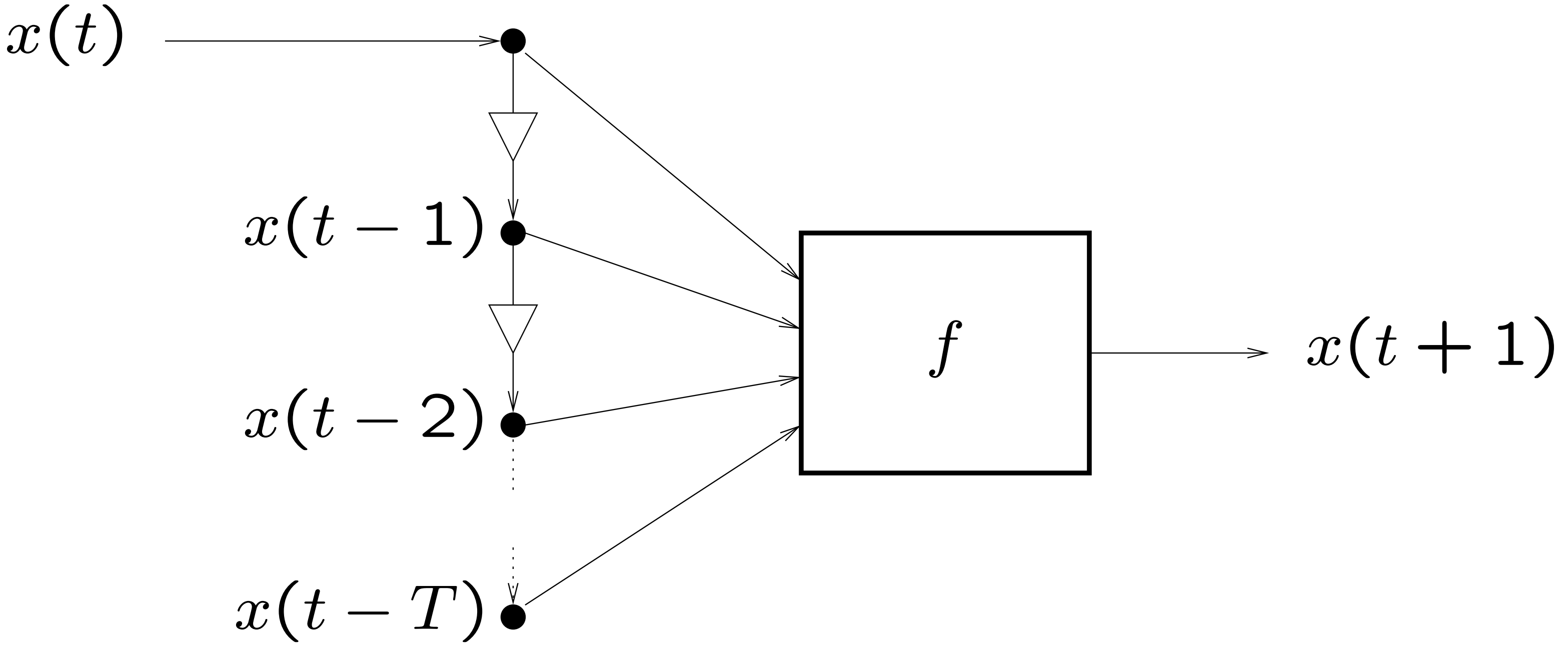
A Jordan net can be viewed as a variant of a NARMA model.



This network has no memory; it “remembers” only the output from the previous timestep.

The Case for Alternative Memory Models

Uniform sampling is simple but has limitations.



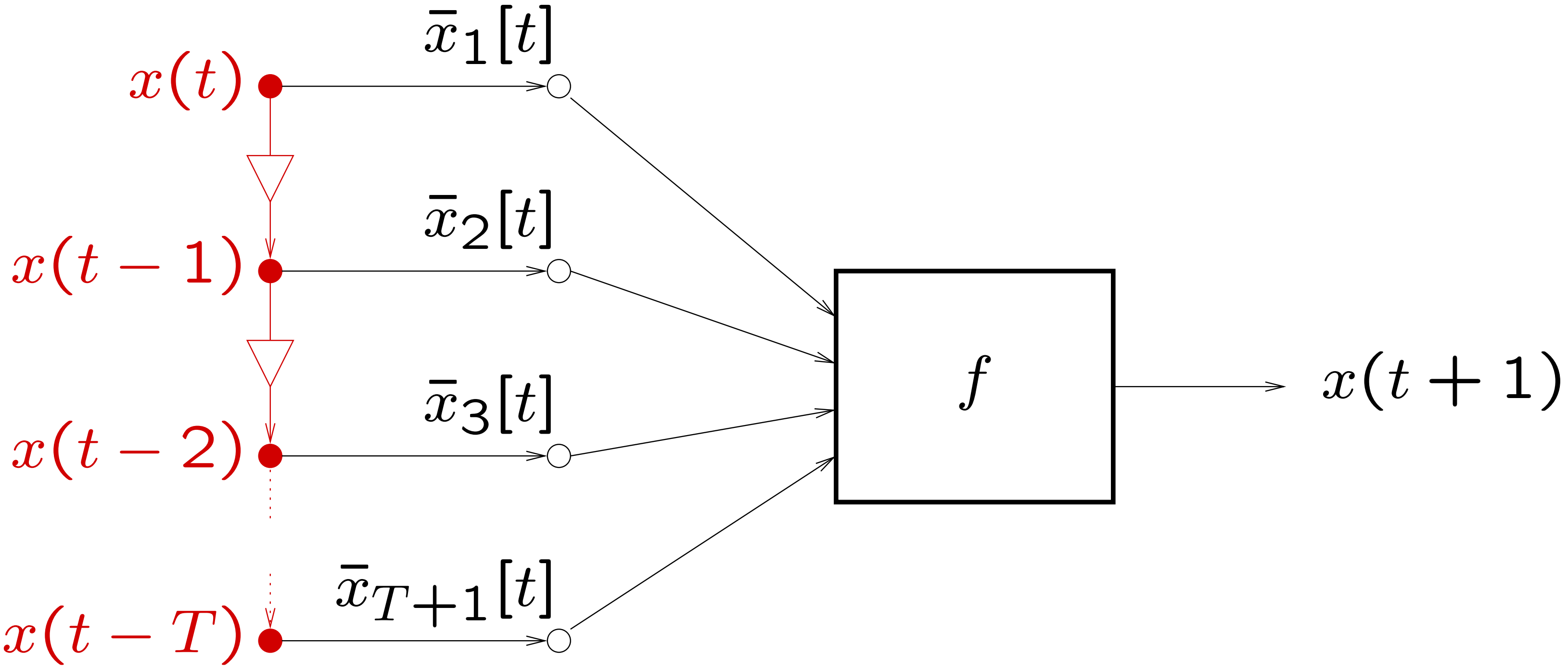
Can only look back T equispaced time steps. To look far into the past, T must be large.

Large $T \longrightarrow$ complicated model: many parameters, slow to train.

A Change of Notation

$$\bar{x}_i[t] \;=\; x[t-i+1] \tag{15}$$

This is a just a reformulation. $\bar{x}_i[t]$ is a *memory term*, allowing us to ellide the tapped delay line from our diagrams:



Propose Non-uniform Sampling

$$\bar{x}_i[t] \;=\; x[t - d_i] \;, \quad d_i \in \mathcal{N} \tag{16}$$

d_i is an integer delay; for example, for four inputs, $\mathbf{d}$ could be $\{1, 2, 4, 8\}$. This is a generalization. If $\mathbf{d}$ were $\{1, 2, 3, 4\}$, we would be back to uniform sampling.

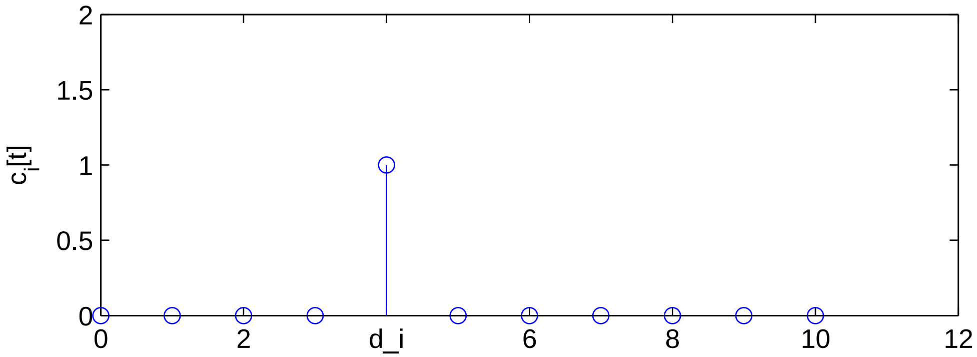
Convolutional Memory Terms

Mozer has suggested treating each memory term as a convolution of $x[t]$ with a kernel function:

$$\bar{x}_i[t] \;=\; \sum_{\tau=1}^{t} c_i[t-\tau] \;\cdot\; x[\tau] \tag{17}$$

Delay lines, non-uniformly and uniformly sampled, can be expressed using this notation, with the kernel function defined by:

$$c_i[t] \;=\; \begin{cases} 1 & \text{if } t = d_i \\ 0 & \text{otherwise} \end{cases} \tag{18}$$



Exponential Trace Memory

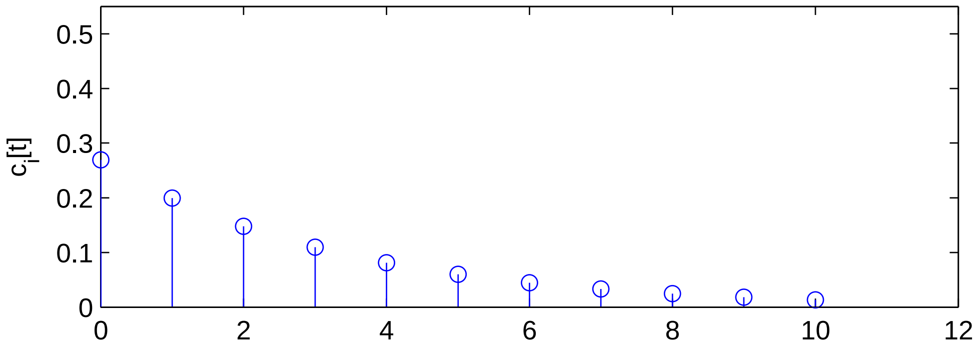
The idea: remember past values as exponentially decaying weighed average of input:

$$c_i[t] \;=\; (1-\mu_i)\cdot\mu_i^t \;, \qquad \mu \in (-1,+1) \tag{19}$$

μ_i is the *decay rate* (a discount factor), eg. 0.99.

Each $\bar{x}_i$ uses a different decay rate.

No outputs are forgotten; they just “fade away”.

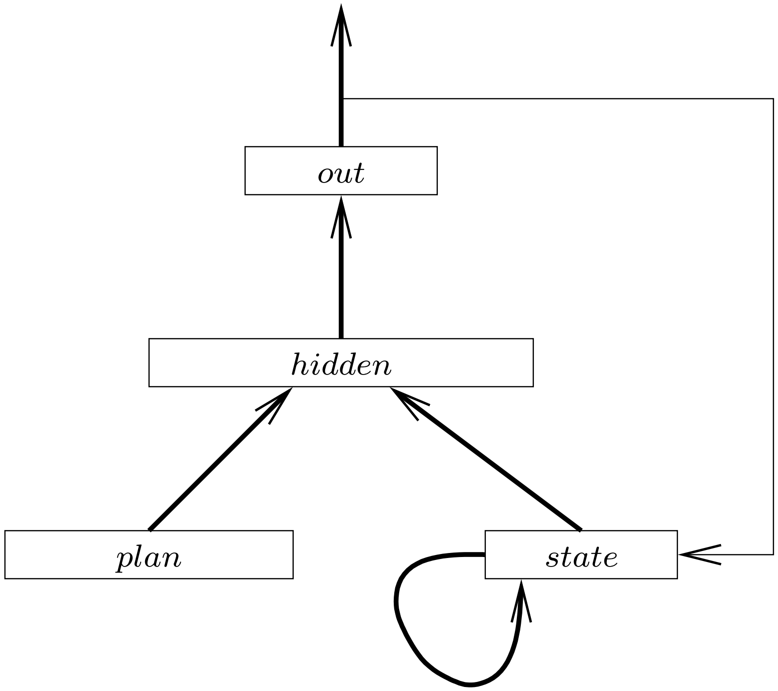


Exponential Trace Memory, cont'd

A nice feature: if all $\mu_i \equiv \mu$, don't have to do the convolution at each time step. Compute incrementally:

$$\bar{x}_i[t] = (1 - \mu)\, x[t] + \mu \bar{x}_i[t - i] \tag{20}$$

Example: a Jordan net with memory



Special Case: Binary Sequences

Let $x_i[t] \in \{0, 1\}$, with $\mu = 0.5$.

Memory $\bar{x}[t]$ is a bit string, treated as a floating point fraction.

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathrm{x[t]} & = & \{1\} & \bar{x}[t] & = & .1 \\
 & & \{1, 0\} & & & .01 \\
 & & \{1, 0, 0\} & & & .001 \\
 & & \{1, 0, 0, 1\} & & & .1001 \\
 & & \{1, 0, 0, 1, 1\} & & & .11001
\end{array}$$

Earliest bit becomes least significant bit of $\bar{x}[t]$.

Memory Depth and Resolution

Depth is how far back memory goes.

Resolution is the degree to which information about individual sequence elements is preserved.

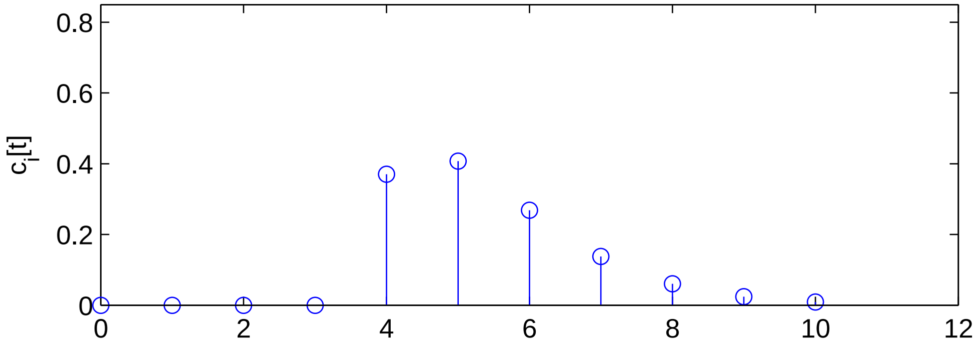
At fixed model order, we have a tradeoff.

- Tapped delay line: low depth, high resolution.
- Exponential trace: high depth, low resolution.

Gamma Memory (deVries & Principe)

$$c_i[t] \;=\; \begin{cases} \binom{t}{d_i}(1-\mu_i)^{d_i+1}\cdot\mu_i^{t-d_i} & \text{if } t \geq d_i \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

d_i is an integer; $\mu_i \in [0,1]$. Eg. for $d_i = 4$ and $\mu = 0.21$:



If $d_i = 0$, this is exponential trace memory.
As $\mu_i \to 0$, this becomes the tapped delay line.

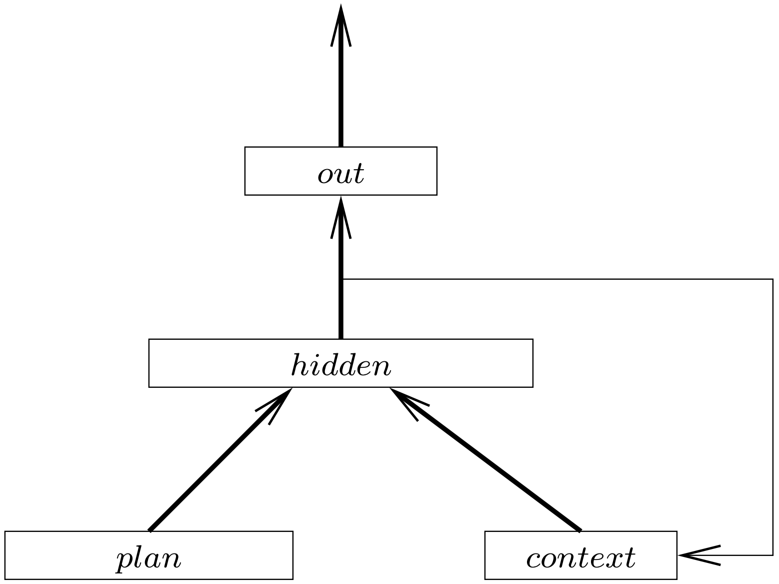
Can trade depth for resolution by adjusting d_i and μ_i.
Gamma functions form a basis for a family of kernel functions.

Memory Content

Don't have to store the raw $x[t]$.

Can store any transformation we like. For example, can store the internal state of the NN.

Example: Elman net



Think of this as a 1-tap delay line storing $f(x[t])$, the hidden layer.

Horizon of Prediction

So far covered many neural net architectures which could be used for predicting the next sample in a time series. What if we need a longer forecast, ie. not $\widehat{x}[t+1]$ but $\widehat{x}[t+s]$, with the horizon of prediction $s > 1$?

Three options:

- Train on $\{x[t], x[t-1], x[t-2], \cdots\}$ to predict $x[t+s]$.
- Train to predict all $x[t+i]$, $1 \geq i \geq s$ (good for small s).
- Train to predict $x[t+1]$ only, but iterate to get $x[t+s]$ for any s.

Predicting Sunspot Activity

Fessant, Bengio and Collobert.

Sunspots affect ionospheric propagation of radio waves.

Telecom companies want to predict sunspot activity six months in advance.

Sunspots follow an 11 year cycle, varying from 9-14 years.

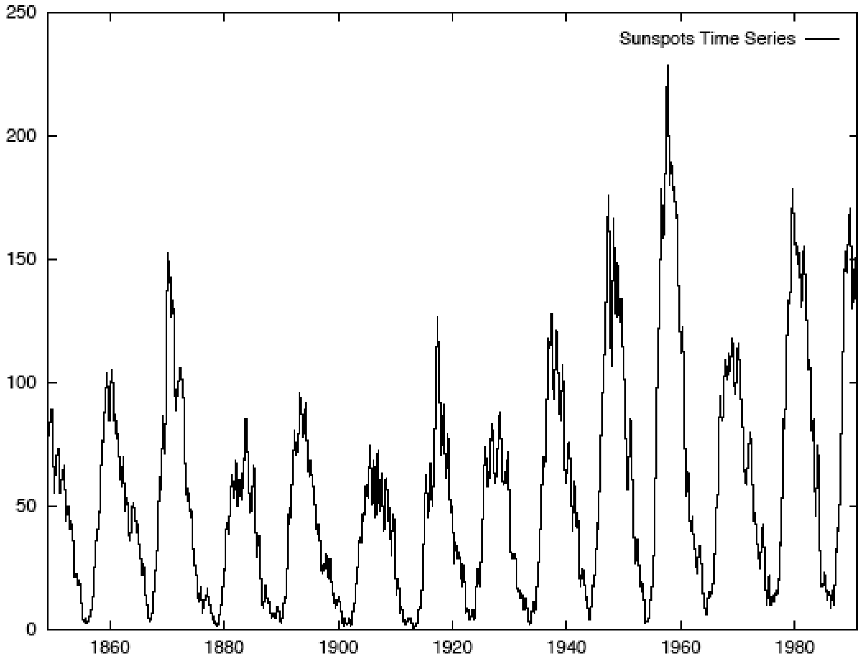
Monthly data goes back to 1849.

Authors focus on predicting $IR5$, a smoothed index of monthly solar activity.

Fessant et al: the IR5 Sunspots Series

$$IR5[t] \;=\; \frac{1}{5}\left(R[t-3]+R[t-2]+R[t-1]+R[t]+R[t+1]\right)$$

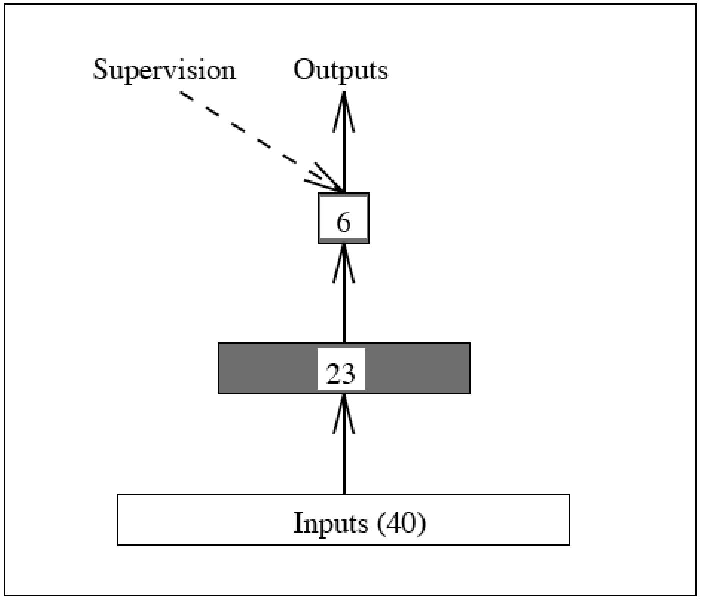
where $R[t]$ is the mean sunspot number for month t and $IR5[t]$ is the desired index.



Fessant et al: Simple Feedforward NN

Output: $\{\widehat{x}[t], \cdots, \widehat{x}[t+5]\}$

(1087 weights)

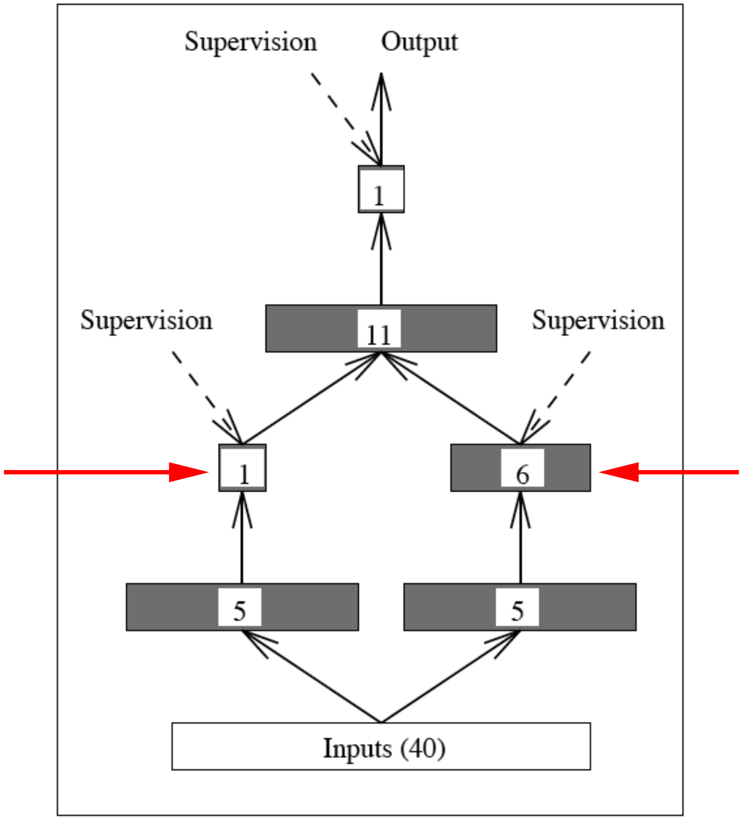


Input: $\{x[t-40], \cdots, x[t-1]\}$

Fessant et al: Modular Feedforward NN

Output: $\hat{x}[t+5]$

(552 weights)



$\hat{x}[t+5] \longrightarrow$

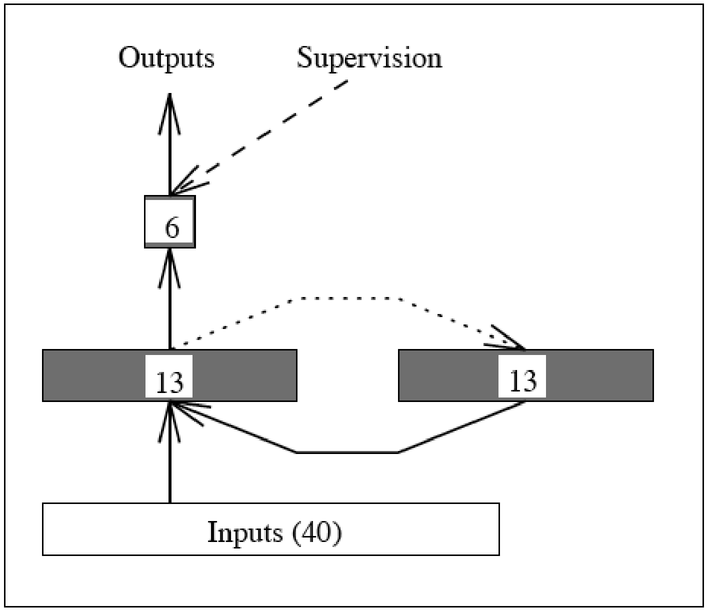
$\longleftarrow \hat{x}[t], \cdots, \hat{x}[t+5]$

Input: $\{x[t-40], \cdots, x[t-1]\}$

Fessant et al: Elman NN

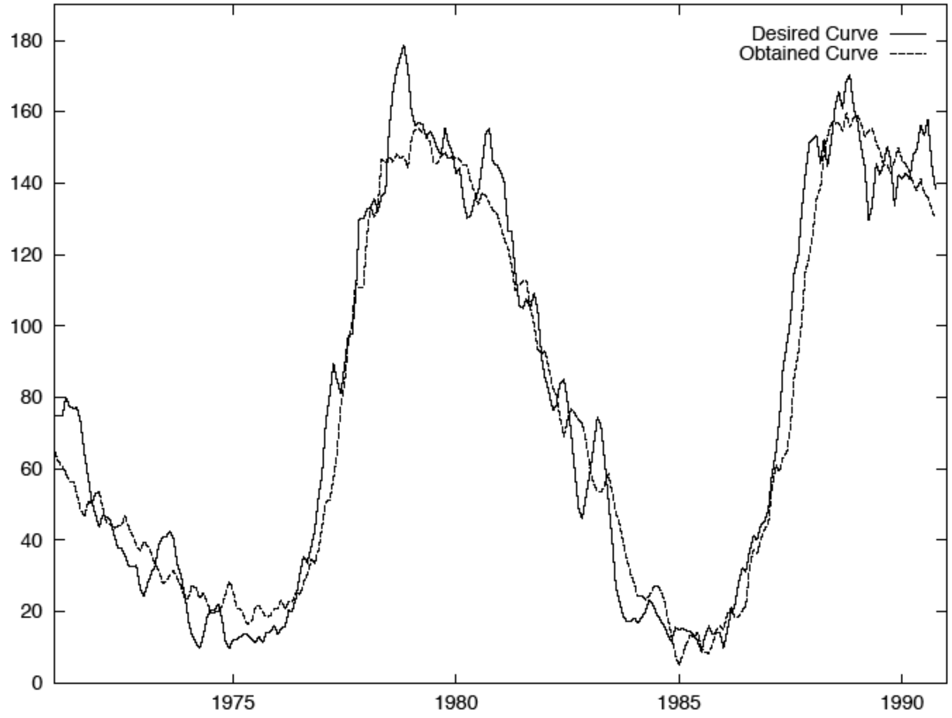
Output: $\{\widehat{x}[t], \cdots, \widehat{x}[t+5]\}$

(786 weights)



Input: $\{x[t-40], \cdots, x[t-1]\}$

Fessant et al: Results



Train on first 1428 samples Test on last 238 samples	CNET heuristic	Simple Net	Modular Net	Elman Net
Average Relative Variance	0.1130	0.0884	0.0748	0.0737
# Strong Errors	12	12	4	4