Time and Neural Networks

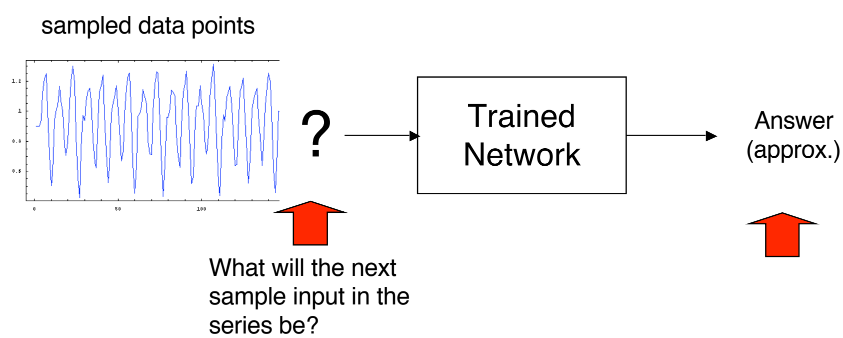
Thus far

- Networks have been "combinational"; input pattern presented at once
- Now we wish to consider cases where network inputs and learned behavior can include **functions of time**

Models to be Considered Here

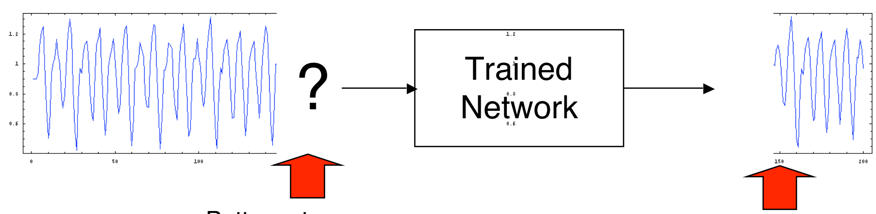
- Time-series prediction
- Adaptive (or Active) noise cancellation
- Time-Delay Neural Networks (TDNN, TLFF)
- Backpropagation through time (BPTT)
- FIR-Multi-layer networks (FIRNET)

Example: Time-Series Problems: "Predict the Future"



Time-Series Problems: "Predict the Future"

sampled data points

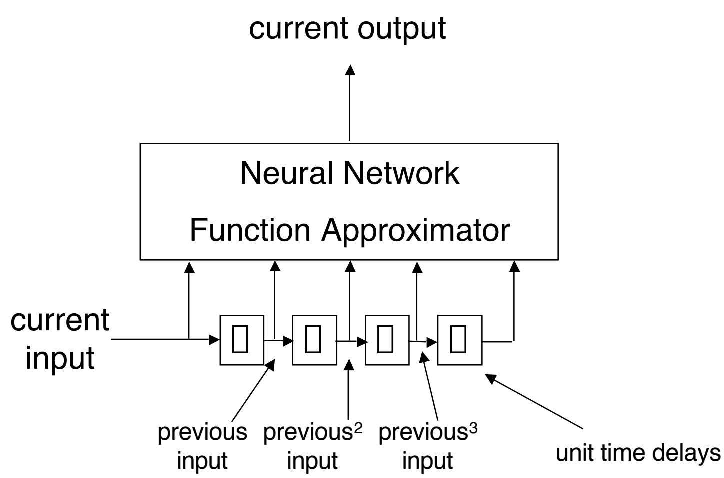


Better yet:
What will the next *n* sample inputs be, for nominal n?

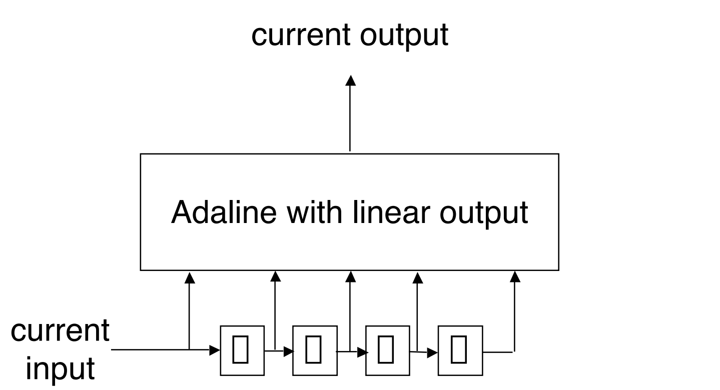
Applications

- Signal processing
- Sun-spot prediction
- Predict the degradation of the ozone layer
- Market analysis

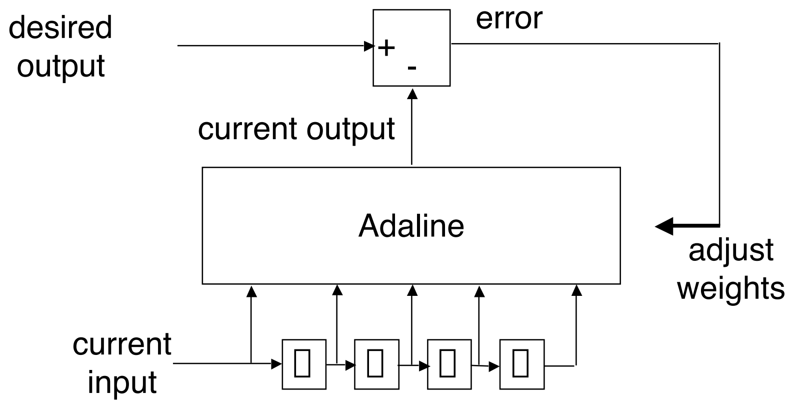
Learning to Mimic



Example: Adaline Mimic



Training the Adaline Mimic



Training the Adaline Mimic

- Recall the Adaline training rule:

$$\Delta \mathbf{W} = \eta \cdot (\text{desired - actual output}) \cdot \mathbf{input}$$

- Here input vector is the current input, along with all the delayed inputs (one per weight)

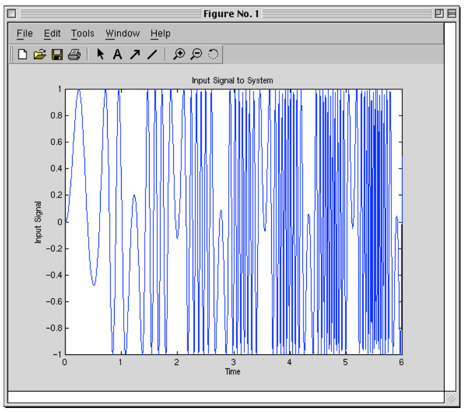
Demo applin4

- An Adaline is trained to mimic a specific input-output behavior.
- The output is an attenuated version of the input.
- When subsequently presented with the input, the output is observed and the error computed.

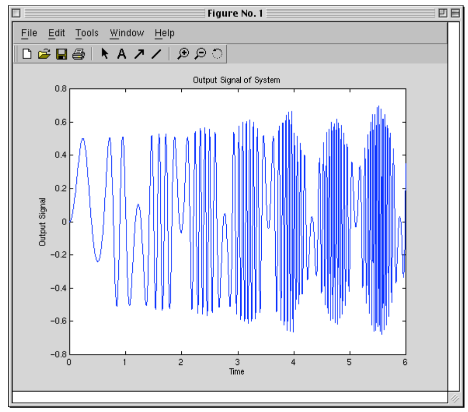
Example: applin4

```
%    NEWLIN  - Initializes a linear layer.
%    ADAPT  - Trains a linear layer with Widrow-Hoff rule.

%    ADAPTIVE LINEAR SYSTEM IDENTIFICATION:

%    Using the above functions a linear neuron is adaptively
%    trained to model a linear system.

%    The linear neuron is able to adapt to changes in the
%    model it is trying to mimic.
```

applin4: Input-Output Relation



Input



Desired Output

applin: frame 2

```
%   DEFINE THE NETWORK
%   ==================

%   NEWLIN generates a linear network.

%   We will use a learning rate of 0.5, and two
%   delays in the input.  The resulting network
%   will predict the next value of the target signal
%   using the last two values of the input.

lr = 0.5;
delays = [0 1];

net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

applin: frame 3

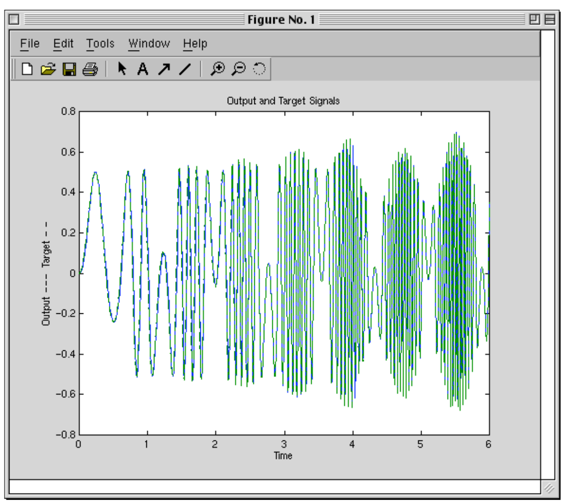
% ADAPTING THE LINEAR NEURON

% ADAPT simulates adaptive linear neurons. It takes the
% initial network, an input signal, and a target signal,
% and filters the signal adaptively. The output signal and
% the error signal are returned, along with new network.

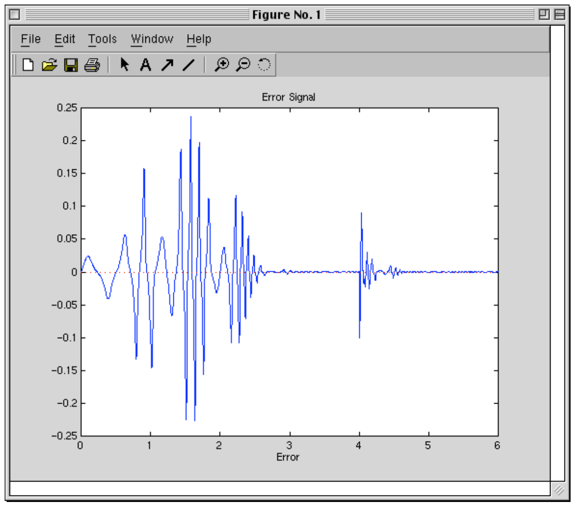
% Adapting begins...please wait...

```
[net,y,e]=adapt(net,P,T);
```

applin4: actual output vs. target



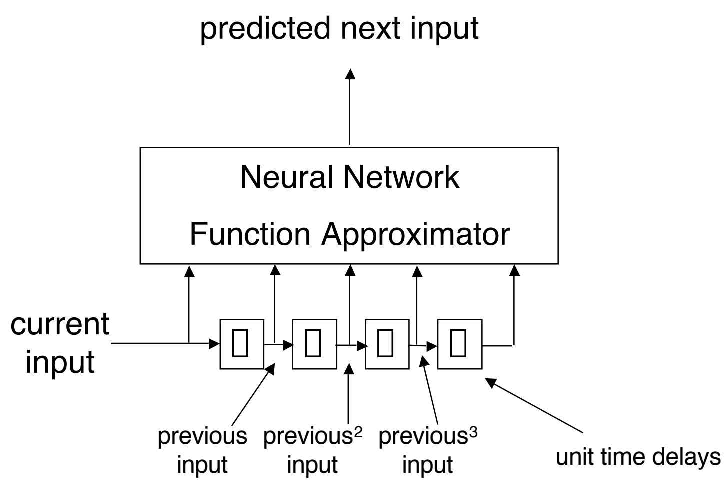
applin4: error



Interesting Point

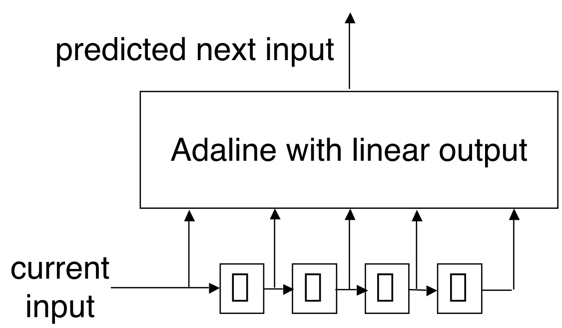
- The Adaline Predictor can be trained **during operation**.
- At each time step, one set of weight modifications can be made.
- After a transient, the network learns to mimic the desired behavior.

How to Learn to Predict?

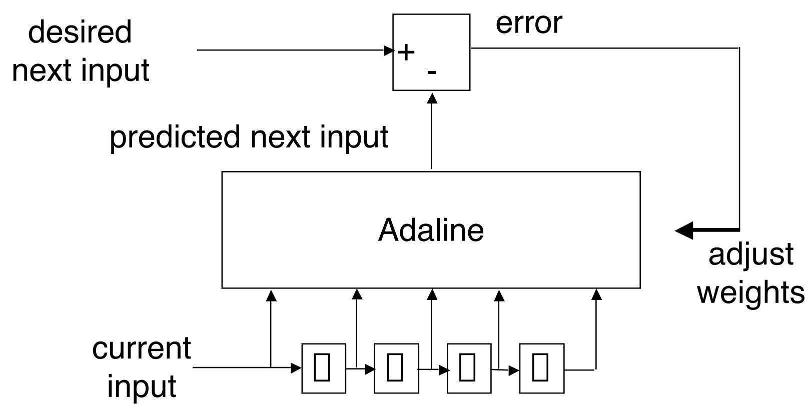


Example: Adaline Predictor

The predictor is like a mimic, where the next input is what is to be mimicked.



Training the Adaline Predictor



Training the Adaline Predictor

- Recall the Adaline training rule:

  $\Delta \mathbf{W} = \eta \cdot$ (desired - actual output) $\cdot$ **input**

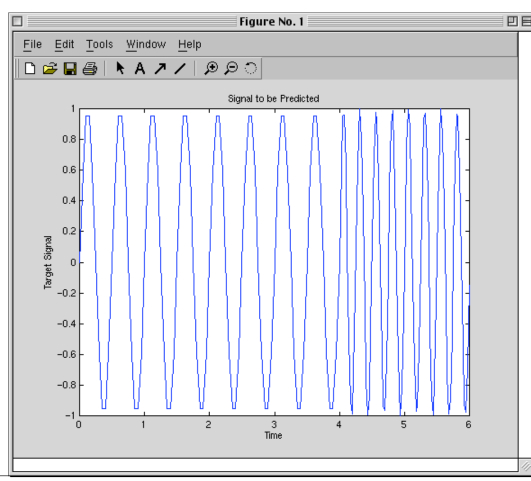
- Here “input” is the current input, along with all the delayed inputs (one per weight)

Demonstration applin2

- Predicts the next input based on 5 previous input samples.
- The input is a sine wave, but the frequency doubles after awhile.
- It is desirable for the network to adapt its behavior to the new frequency.

applin2

signal to be predicted (2 sine waves of different frequencies)



applin2: frame 1

% NEWLIN - Creates and initializes a linear layer.
% ADAPT - Trains a linear layer with Widrow-Hoff rule.

% ADAPTIVE LINEAR PREDICTION:

% Using the above functions a linear neuron is adaptively
% trained to predict the next value in a signal, given the
% last five values of the signal.

% The linear neuron is able to adapt to changes in the
% signal it is trying to predict.

applin2: frame 2

```
%    DEFINING A WAVE FORM
%    TIME1 and TIME2 define two segments of time.

time1 = 0:0.05:4;      % from 0 to 4 seconds, steps of .05
time2 = 4.05:0.024:6;  % from 4 to 6 seconds, steps of .05
%    TIME defines all the time steps of this simulation.
time = [time1 time2];  % from 0 to 6 seconds

%    T defines a signal which changes frequency once:
T = con2seq([sin(time1*4*pi) sin(time2*8*pi)]);

%    The input P to the network is the same as the target.
%    The network will use the last five
%    values of the target to predict the next value.
```

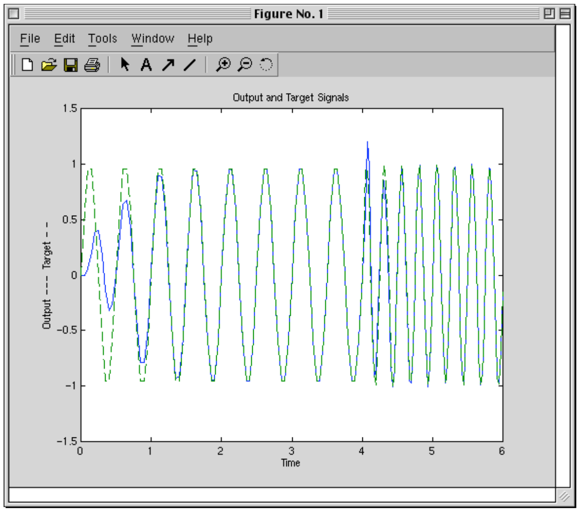
applin2: frame 3

```
%   NEWLIN generates a linear network.

%   We will use a learning rate of 0.1, and five
%   delays in the input.  The resulting network
%   will predict the next value of the target signal
%   using the last five values of the target.

lr = 0.1;
delays = [1 2 3 4 5];

net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

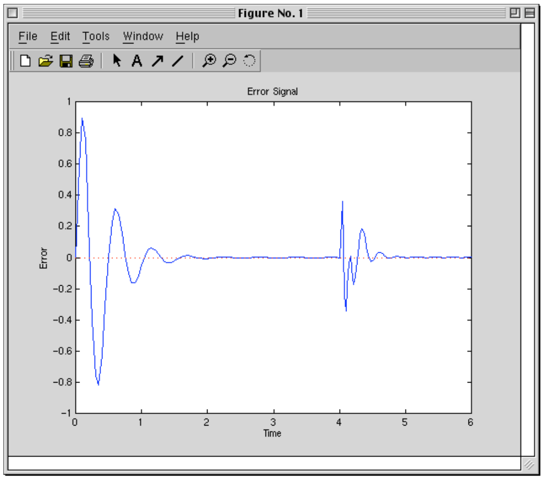
applin2: frame 4

```
%   ADAPTING THE LINEAR NEURON
%   ==========================

%   ADAPT simulates adaptive linear neurons.  It takes the initial
%   network, an input signal, and a target signal,
%   and filters the signal adaptively.  The output signal and
%   the error signal are returned, along with new network.


%   Adapting begins...please wait...

[net,y,e]=adapt(net,P,T);
```

applin2: actual output vs. target



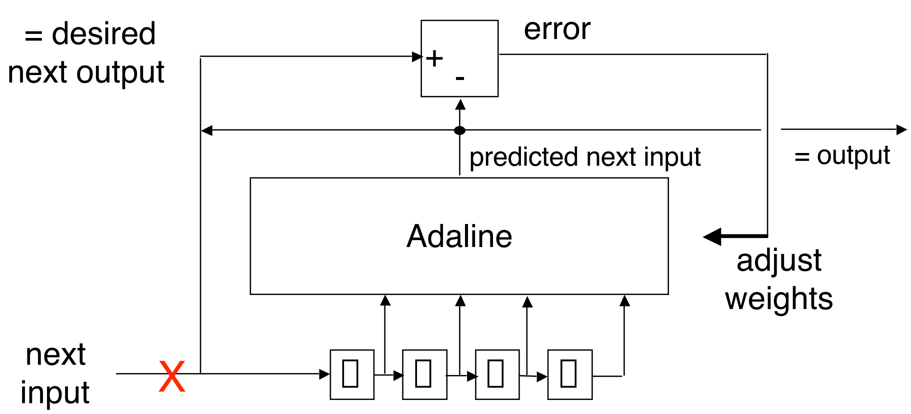
applin2: error = target - output



Once the Network has Been Trained

- it can use its *own* output as the next input.
- That is, it can “run free”, predicting the full output **sequence.**
- Since the output was only an approximation, the accuracy of the predicted output will *deteriorate* with time.

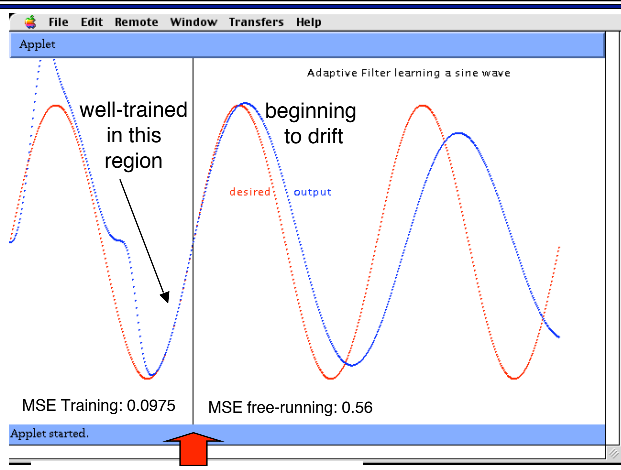
Free-Running Mode



Free-Running after Training (applet: cd /cs/cs152/af; go)

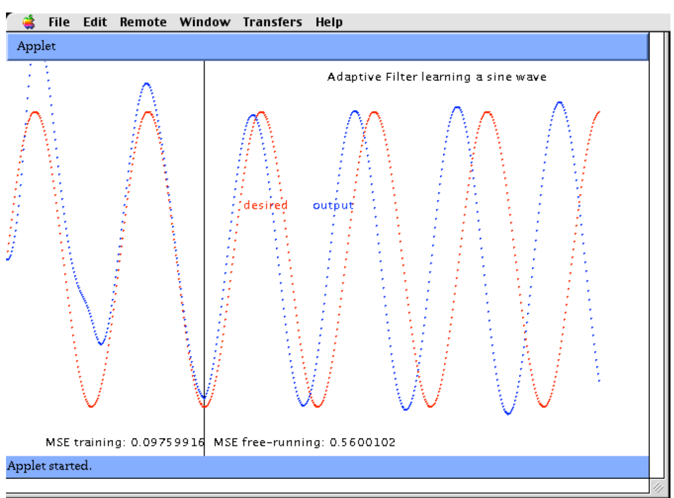
Here the filter was trained with a sine wave

red = desired
blue = actual



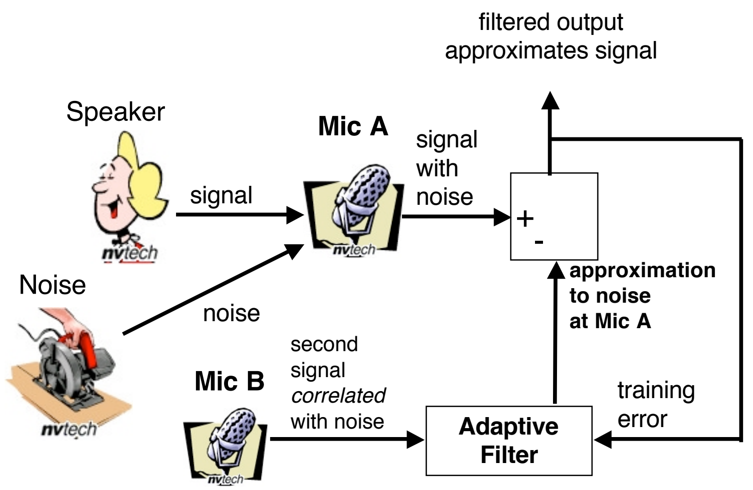
Here the sine wave was removed and the output (prediction) fed back into the input

The same Filter at 1.75 x frequency



Noise-Reduction Scenario

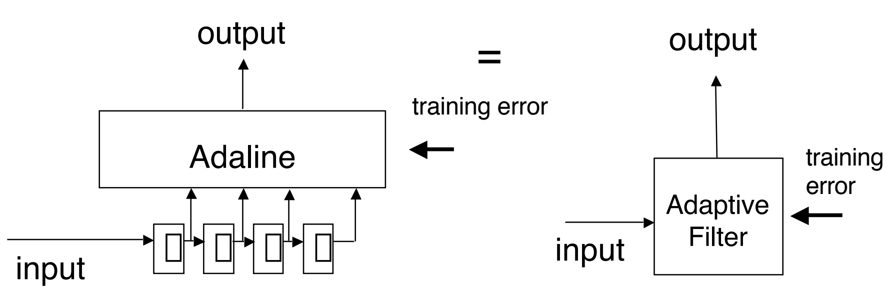
Filter Learns to Predict the Noise



(*correlated*, but we don't know the exact amplitude or phase)

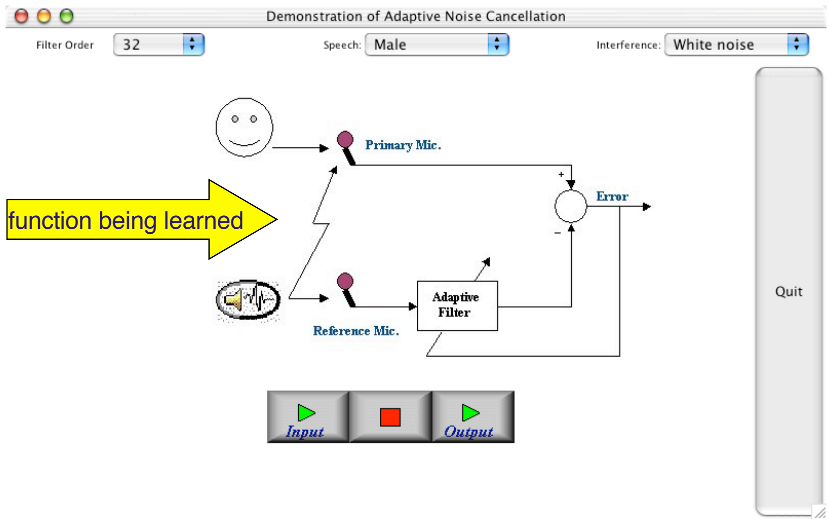
Adaptive Filter Component

- Adaptive filter component learns produce output from input as guided by training error signal

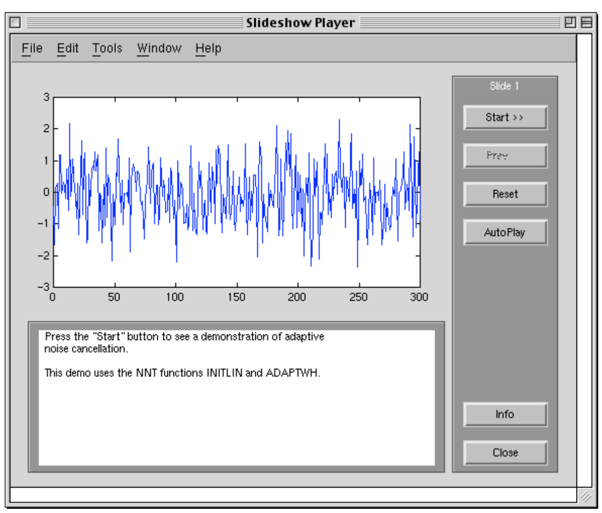


ANC Audio Demo from Ariz. State Univ.

http://www.eas.asu.edu/~dsp/grad/anand/java/ANC/ANC.html

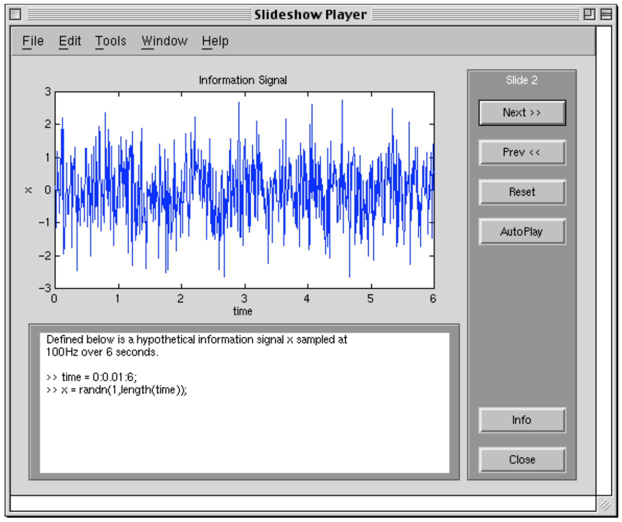


applin5 Demo(no longer exists)



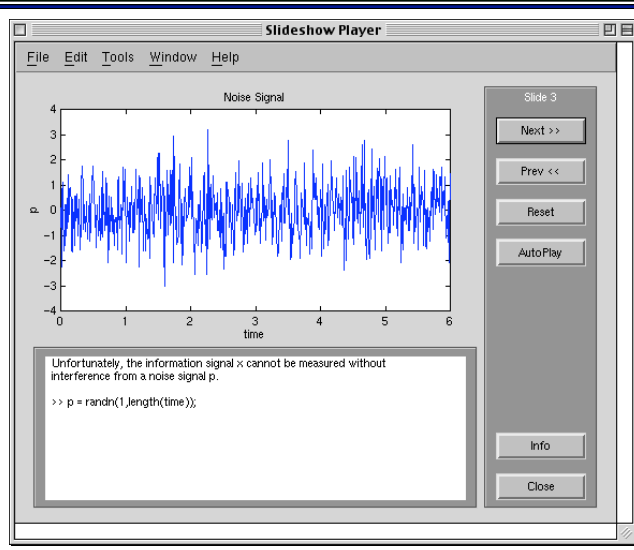
Information Signal

(without noise, not usually known, but we're creating it)

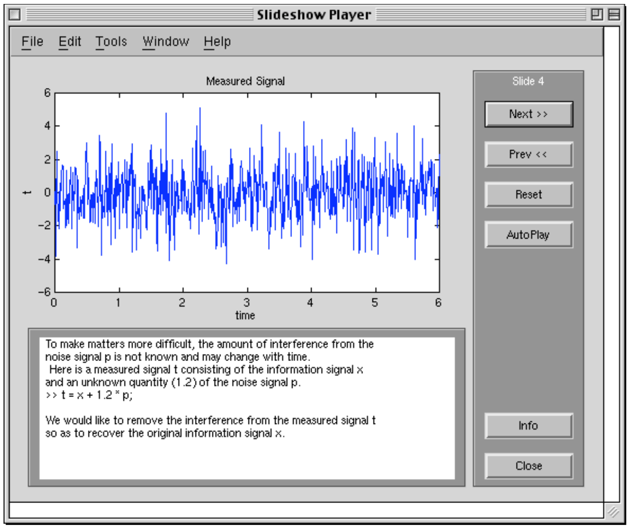


Noise Signal

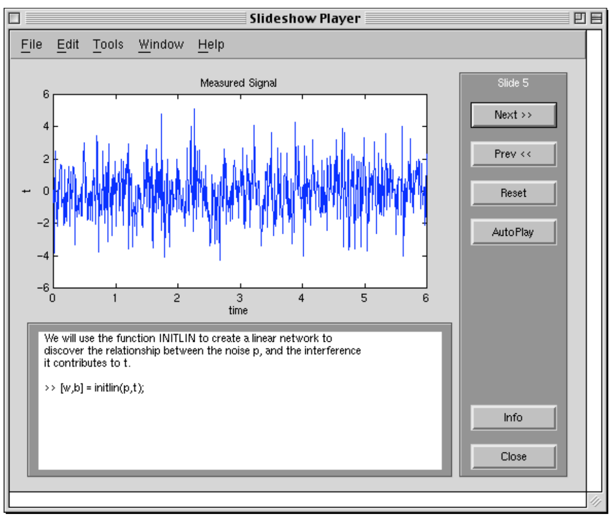
(not usually known, trying to learn it)



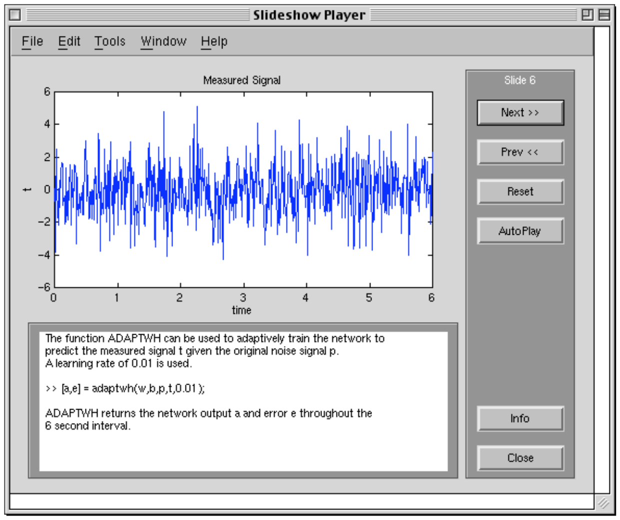
Measured Signal (with noise)



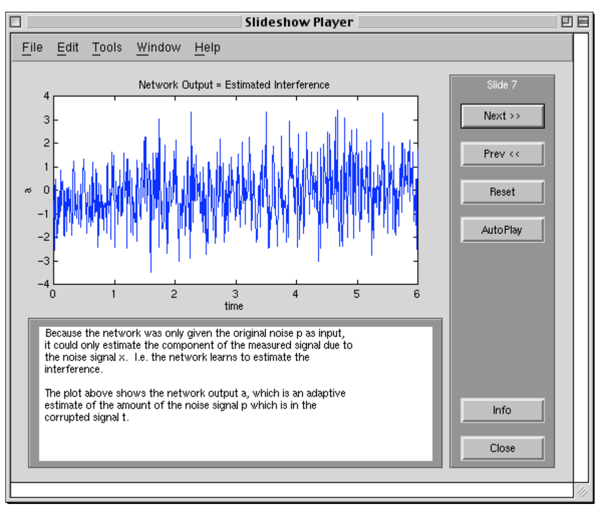
Initializing Filter



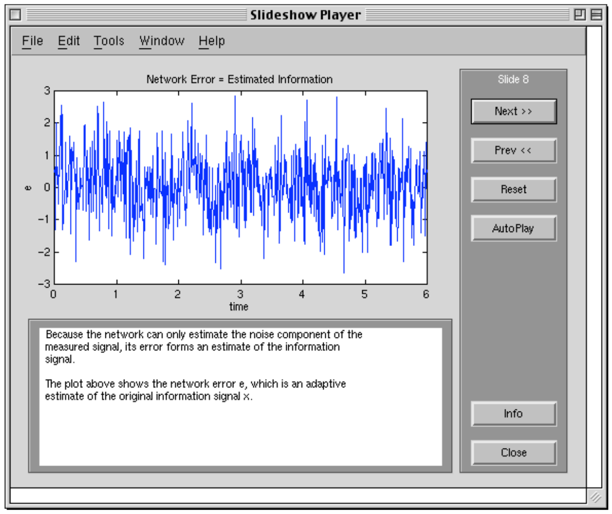
Adapting using Widrow-Hoff



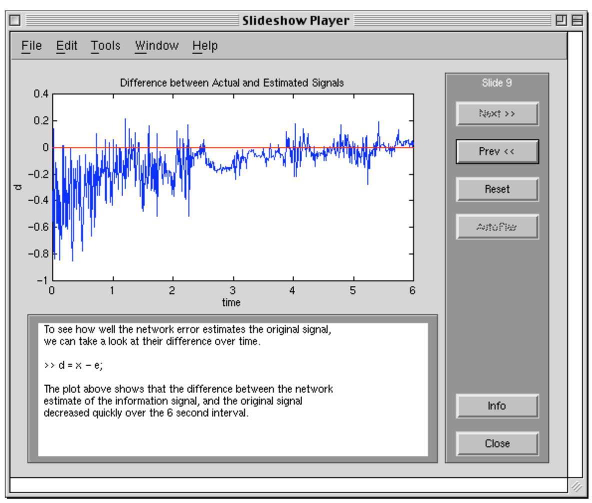
Interference Estimated by Filter



Estimated Information
(= Noisy Signal - Estimated Noise)



Error after Filtering
(= Estimated Info - Actual Info)



ANC (Active Noise Cancellation) Headphones

cf. http://www.nalanda.nitc.ac.in/industry/appnotes/Texas/computing/spra160.pdf

THE FILTERED-X LMS ALGORITHM

The filtered-X LMS algorithm developed by Widrow [8] seeks the controller coefficients (weight vector) of $C(q^{-1}, k)$, which minimize the mean-squared error, $\xi = E[e^2(k)]$. The mean-squared error is the average power of the error microphone signal. To accomplish this task, a gradient method is used. In the feedforward configuration, the component of $e(k)$ that is correlated with $x(k)$ is removed, leaving only $v(k)$. It is this feature that allows the selectivity property in an ANC system.

The controller weight vector, $\theta_C(k) = [c_0(k), c_1(k), \ldots, c_{n_C}(k)]^T$ is adjusted in the direction of the gradient

$$\nabla = \frac{\partial}{\partial \theta_C} E[e^2(k)] \quad . \tag{1}$$

Because the exact gradient is unavailable, an estimate must be used. In the LMS algorithm, the instantaneous value of the error squared, $e^2(k)$, is used

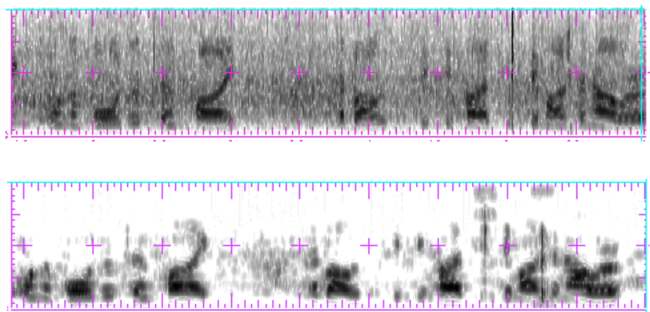
Contextual Nomenclature

- Classical filters *don't adapt*
  - (Lowpass / Highpass / Bandpass) filters
- Adaptive filters adapt
  - LMS filter (least-mean-squared)
  - RLS filter (recursive least squares, based on pseudo-inverse, not as stable)
  - Kalman filter (based on a stochastic state-space model)

Eric Wan (OGI) NN Audio demo

http://www.cse.ogi.edu/Neural/noise/noise.html

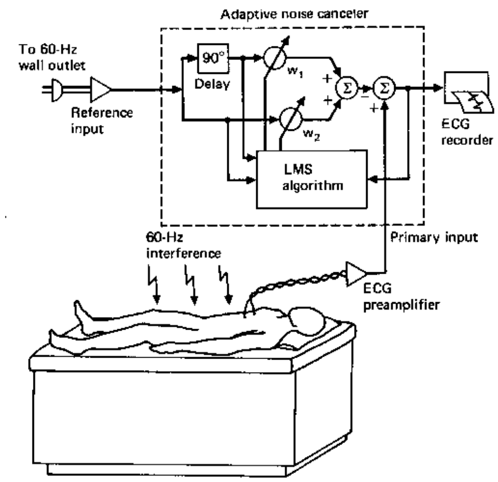
spectra before and after neural-network filtering



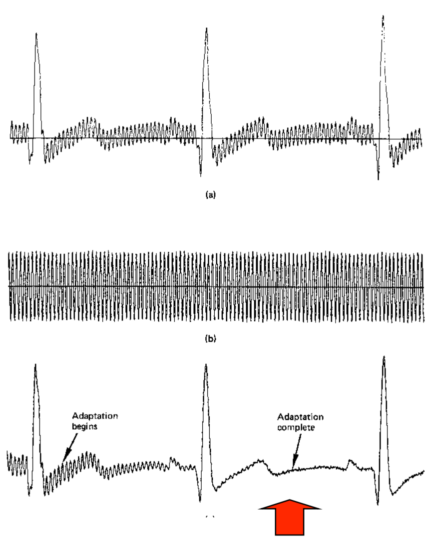
Other Applications

- EKG filtering (60 Hz noise)
- Fetal monitoring (baby's heart - mother's heart)
- Telephone echo cancellation
- Conference telephones

60 Hz Noise in EKG



60 Hz Noise in EKG



Fetal Heartbeat Monitoring

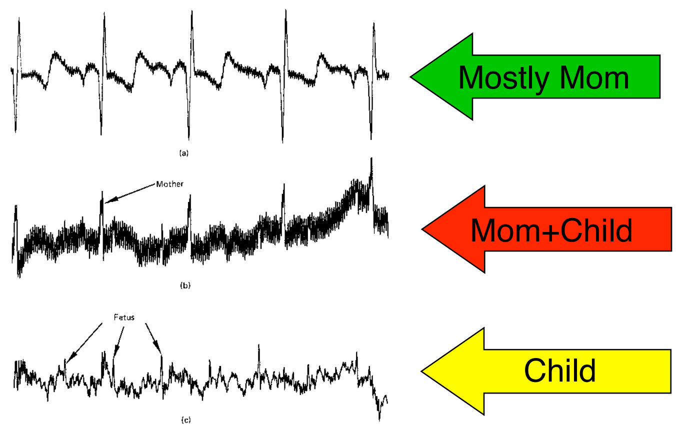
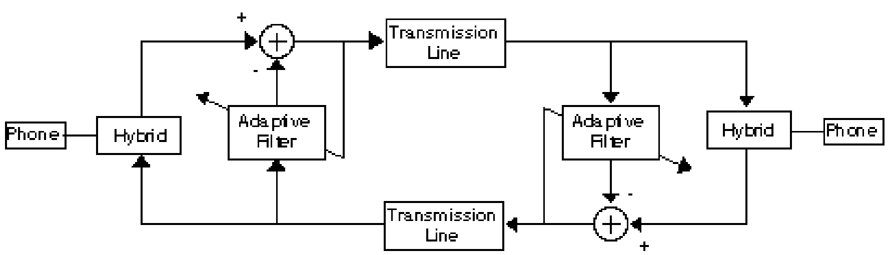
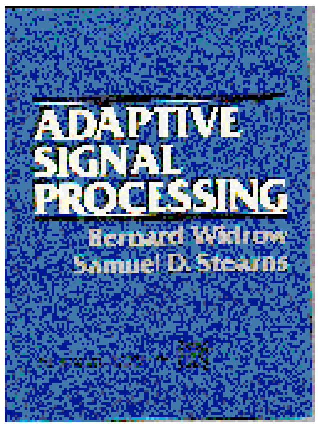


Figure 12.21 Result of wide-band fetal ECG experiment (bandwidth, 0.3–75 Hz; sampling rate, 512 Hz): (a) reference input (chest lead); (b) primary input (abdominal lead); (c) noise canceller output. From B. Widrow et al., *Adaptive Noise Canceling: Principles and Applications*, © December 1975, IEEE.

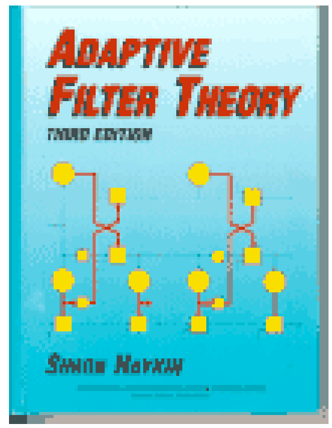
Telephone Echo Cancellation



References



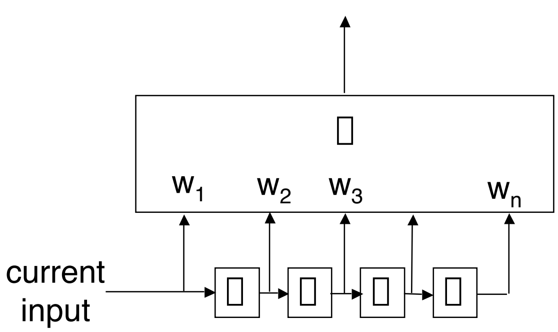
Widrow & Stearns, 1985



Haykin, 1995

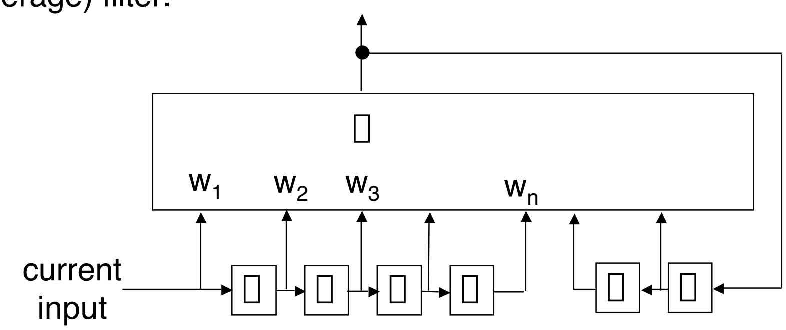
Additional Nomenclature

Filter form is also called a **FIR** (**F**inite **I**mpluse **R**esponse) filter.
In statistics, it is called an **MA** (**M**oving **A**verage) filter.



Additional Nomenclature

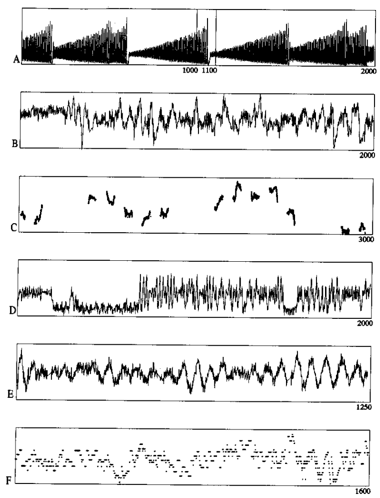
When we add **feedback**, we have an **IIR** (**I**nfinite **I**mpluse **R**esponse) filter.
In statistics, it is called an AR**MA** (**A**uto**R**egressive **M**oving **A**verage) filter.



Classical Fitting of Time-Series

- A given ARMA's coefficients can be fit to generate ***approximately*** a given time series by using least-squares estimation on a set of simultaneous linear equations known as the "Yule-Walker equations".
- Reference: Weigend and Gershenfeld (eds.), Time series prediction, Addison-Wesley, 1994.

Sante Fe Institute Competition
6 unknown time series



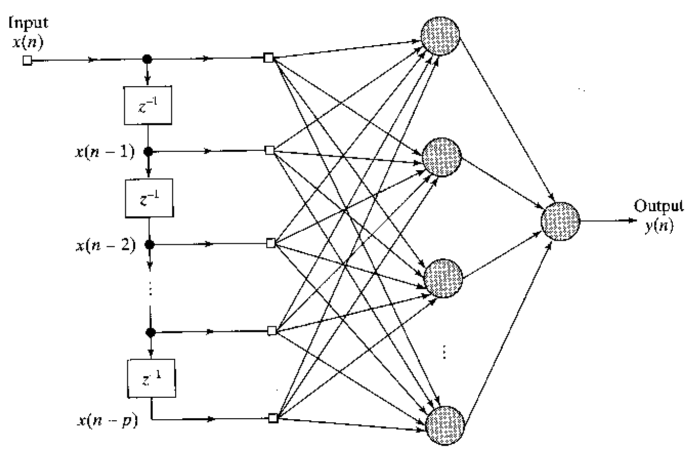
Other Models

- Time-Lagged Feed-Forward Networks, Time-Delay Neural Networks (TLFF, TDNN)
- FIR-Multi-layer networks (FIRNET)
- Backpropagation through time (BPTT)
- Real-Time Recurrent Learning (RTRL)
- Elman nets, Jordan nets
- Temporal difference method (TD(λ))

Time-Lagged Feed-Forward Networks (TLFF)

- An extension of the "Adaline" adaptive filter model
- Use an arbitrary feed-forward net (MLP) in place of the Adaline
- Train using ordinary backpropagation, analogous to LMS

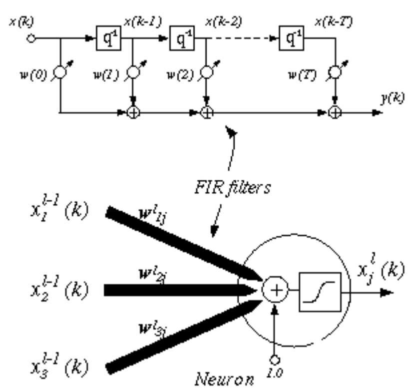
Time-Lagged Feed-Forward Networks (TLFF)

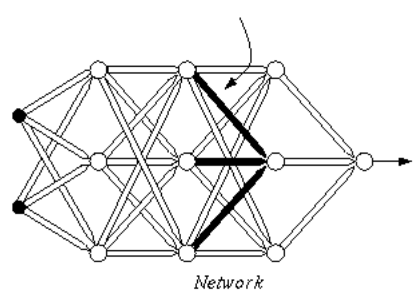


Another Approach: FIR Backpropagation

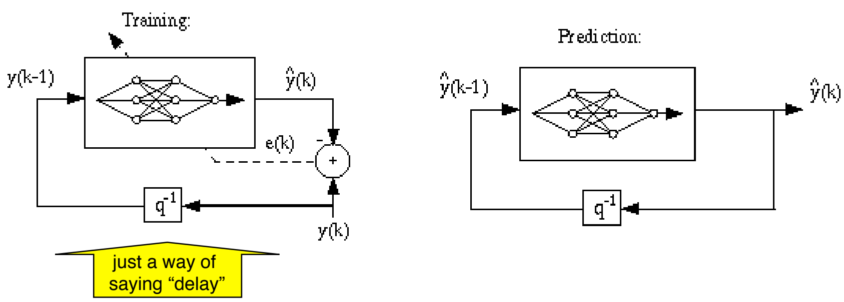
- Eric Wan (Stanford, OGI) came up with the idea of putting FIR filters **inside** a backprop network.
- In place of each single weight there is an entire FIR filter.
- Wan developed the training algorithm for such networks.

Wan's FIR Backprop Net

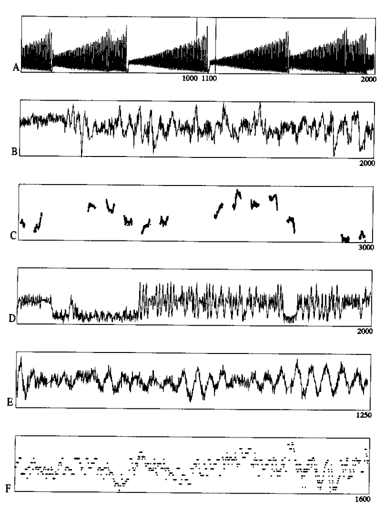




Recall: Training vs. Prediction



Sante Fe Institute Competition 6 unknown series



- A A clean physics experiment. 1000 points of the **fluctuations in a far-infrared laser**, approximately described by three coupled non-linear DE's.
- B Physiological data from a patient with sleep apnea. 34,000 points of heart rate, chest volume, blood oxygen concentration, and EEG state of a sleeping patient.
- C High-frequency **currency exchange rate** data. Ten segments of 3,000 points each of the exchange rate between the Swiss franc and the U.S. dollar, 1-2 minutes apart.
- D Numerically generated series. A driven particle in a 4-dimensional nonlinear multiple-well potential (9 degrees of freedom) with a small nonstationary drift in the depths.
- E Astrophysical data from a **variable white dwarf** star. 27,704 points in 17 segments of the time variation of the intensity.
- F J.S. Bach's final **(unfinished) fugue** from *The Art of the Fugue*.

Wan's Entry in Sante Fe Institute Competition

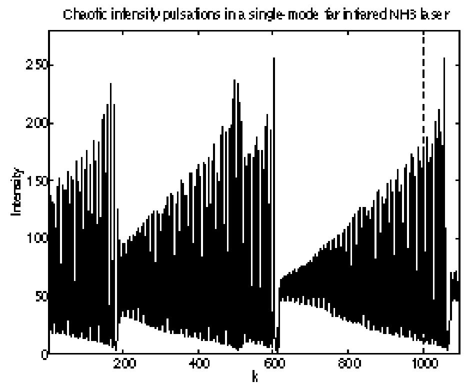
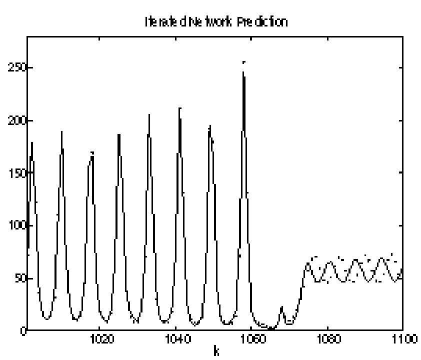
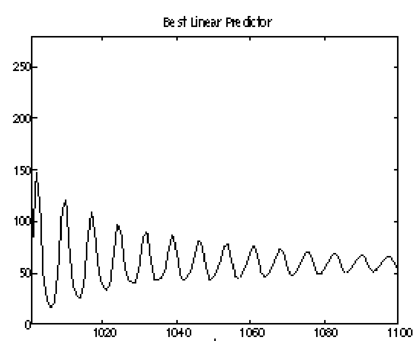


Fig. 3: 1000 points of laser data.

Wan's Prediction Expanded in Sante Fe Institute Competition





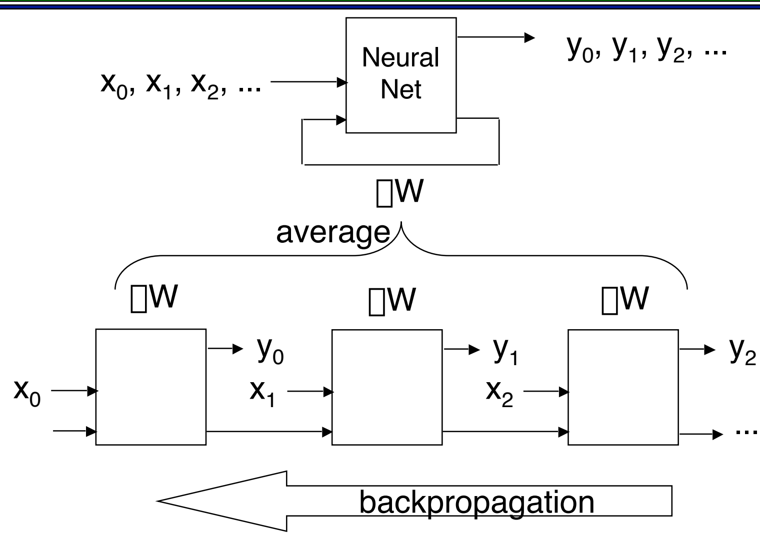
Solid line is actual

dashed line is predicted.

Backpropagation through time (BPTT)

- Unlike TLFF (Time-Lagged Feed-Forward), input samples are not kept in explicit delay lines.
- Instead, input fed sequentially into network (also used in FIR backprop).
- Training is **as if** the network were **unrolled** to accommodate the entire sequence of input samples.
- Only one set of weights is actually used in operation; the weight changes are **averaged** across stages to get the actual weight change

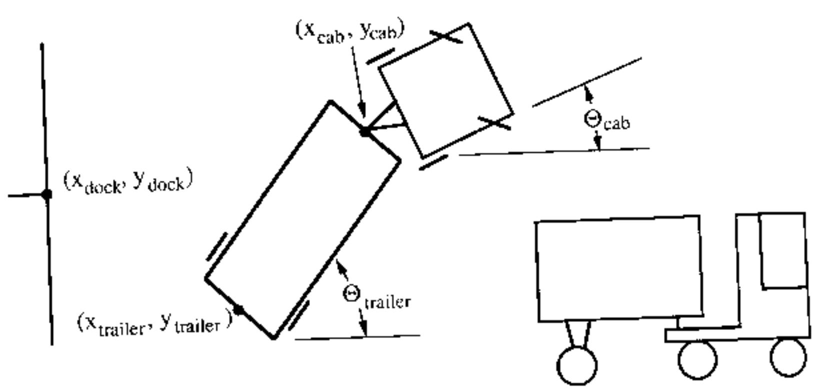
Backpropagation through time (BPTT)



BPTT Application

- The Truck Backer-Upper, D. Nguyen and B. Widrow
- reprinted in Miller, Sutton, and Werbos (eds.), Neural Networks for Control, MIT Press, 1990.
- Problem: Back up a truck so that $(x_{trailer}, y_{trailer}) = (x_{dock}, y_{dock})$, given initial values for $(x_{trailer}, y_{trailer}, x_{cab}, y_{cab}, \theta_{trailer}, \theta_{cab})$

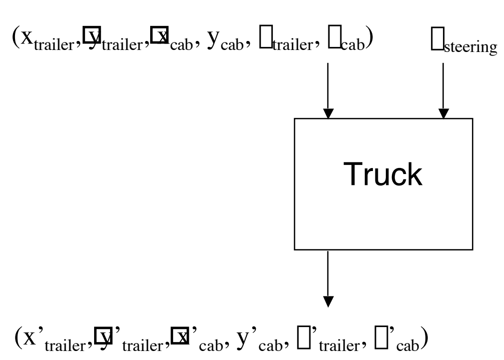
Truck-Backer Problem



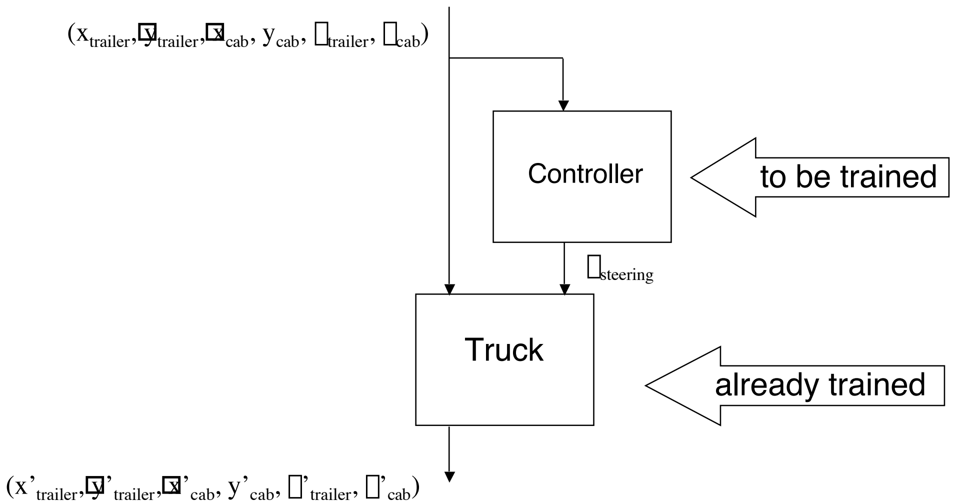
Training the Truck-Backer

- The truck moves in small time increments □
- A neural net is first trained to mimic the truck backing using **real truck dynamics**.
- Given the current state at a time t (which includes the steering angle), the network learns to determine the next state (at time t +□)
- This is done by starting the truck in a random state, observing the error between what the network does and the dynamic model, and adjusting the weights.

The function being learned



Truck-Controller Combo

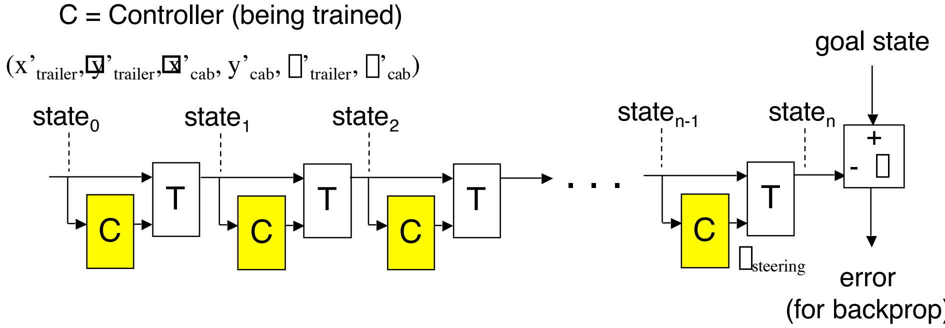


Training the Truck-Backer

- Starting from a random position, the controller backs up the truck one step at a time, until the goal is reached, or an obstacle (such as a side wall) is hit.
- An error value is produced by comparing the desired final state with the goal.
- The error value is backpropagated through the controller-truck combination to adjust the controller's weights, using BPTT.

BPTT for truck training

T = Truck (already trained, weights fixed),
C = Controller (being trained)



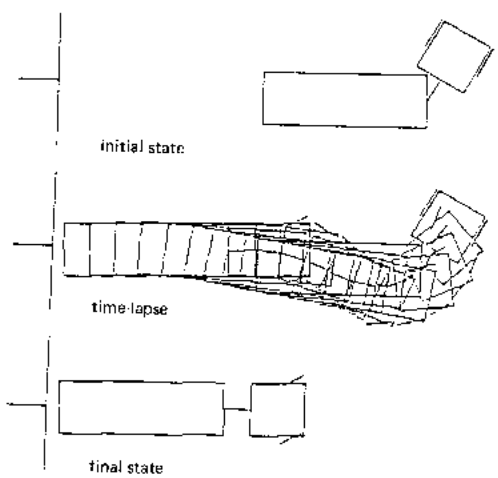
Network Statistics

- Truck Emulator:
  - 6-45-6 tansig-tansig network
- Controller
  - 6-25-1 tansig-tansig network

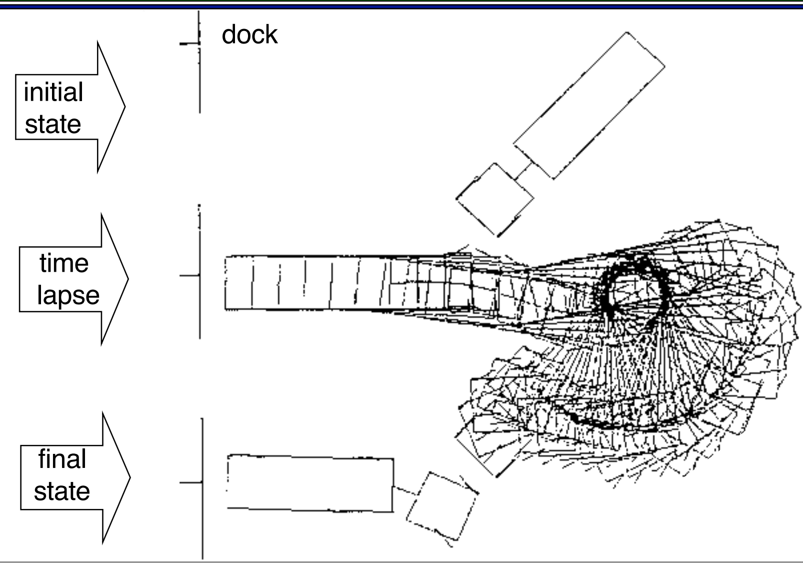
Training

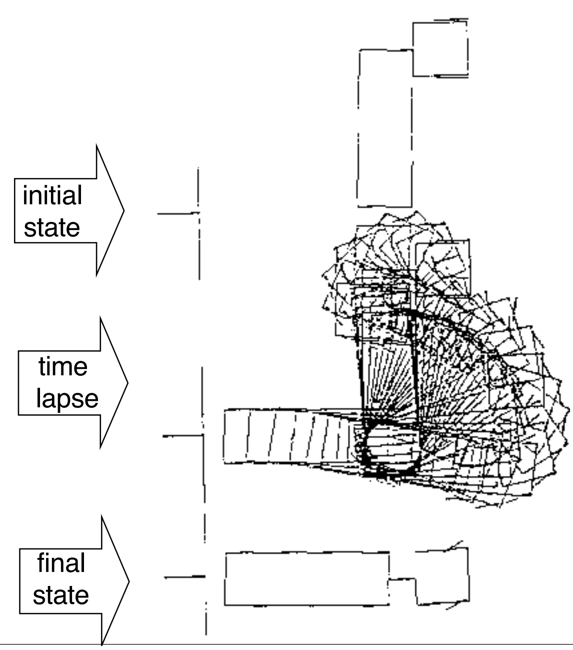
- 20,000 trials required to train
  - 16 lessons of 1000-2000 each
- Initially truck positioned very close to dock and in a nearly-correct position, so controller could **learn easy tasks first**.
- Final MSE was 3% of truck length, angle 7 degrees

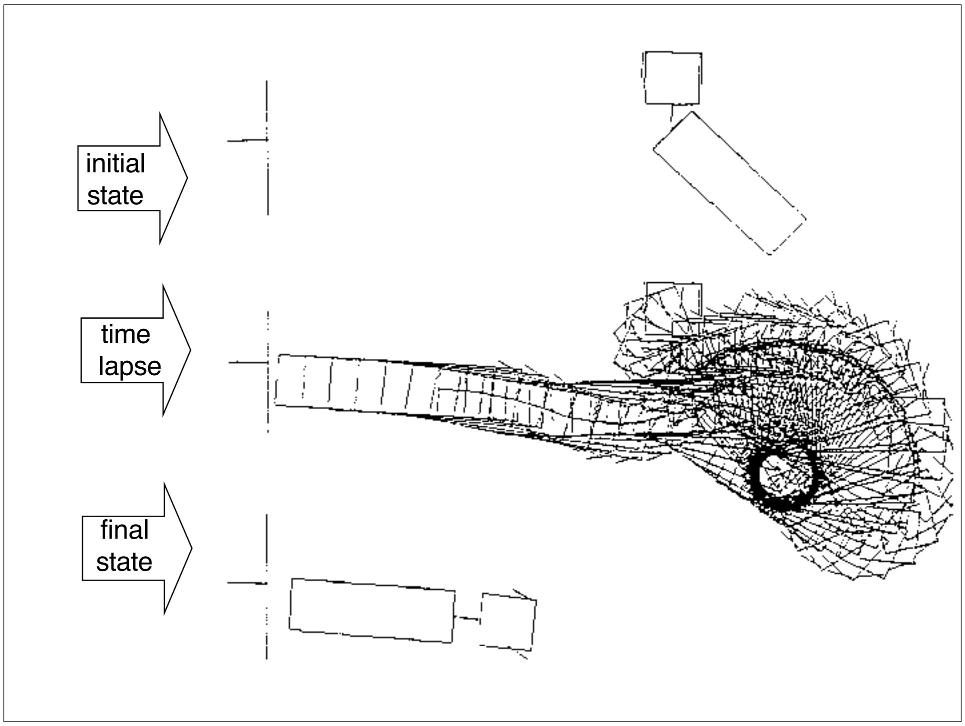
Simulations



Simulations






initial
state
time
lapse
final
state