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”

What will the next sample input in the series be?

Trained Network

Answer (approx.)
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

current output

Neural Network
Function Approximator

current input

Δ Δ Δ Δ

previous input previous input previous input previous input

unit time delays

previous input previous^2 previous^3
Example: Adaline Mimic

Adaline with linear output

current output

Current input
Training the Adaline Mimic

desired output

+ - error

current output

Adaline

Δ Δ Δ Δ

current input

adjust weights
Training the Adaline Mimic

- Recall the Adaline training rule:

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

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

Output Signal of System

Input

Desired Output
% 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);
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?

predicted next input

Neural Network
Function Approximator

current
input

Δ

Δ

Δ

Δ

previous
input

previous
input

previous
input

previous
input

unit time delays

Δ

Δ

Δ

Δ

previous
input

previous
input

previous
input

previous
input

2

3

2

3
Example: Adaline Predictor

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

desired next input → + - error → predicted next input

Adaline

current input → Δ Δ Δ Δ

adjust weights
Training the Adaline Predictor

- Recall the Adaline training rule:

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

- 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.
signal to be predicted (2 sine waves of different frequencies)
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.
% 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.
% 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);
```
% 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

\[
\text{desired next output} = \text{predicted next input} + \text{error}\]

Adaline

= output

adjust weights

next input

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

Here the filter was trained with a sine wave.

red = desired
blue = actual

MSE Training: 0.0975
MSE free-running: 0.56

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

well-trained in this region

beginning to drift
The same Filter at 1.75 x frequency

Adaptive Filter learning a sine wave

MSE training: 0.09759916
MSE free-running: 0.5600102

Applet started.
Noise-Reduction Scenario
Filter Learns to Predict the Noise

- **Speaker**
  - Signal
  - Noise

- **Mic A**
  - Signal with noise
  - Second signal correlated with noise
  - Filtered output approximates signal

- **Mic B**
  - Noise

- **Adaptive Filter**
  - Approximation to noise at Mic A
  - Training error

(correlated, but we don’t know the exact amplitude or phase)
Adaptive Filter Component

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

\[ \text{Adaptive Filter} = \frac{\Delta \Delta \Delta \Delta}{\text{training error}} \]

- Adaline

- Input

- Output

- Training error

- Diagram representation of adaptive filter component.
ANC Audio Demo from Ariz. State Univ.
http://www.eas.asu.edu/~dsp/grad/anand/java/ANC/ANC.html

function being learned
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)
To make matters more difficult, the amount of interference from the noise signal $p$ is not known and may change with time. Here is a measured signal $t$ consisting of the information signal $x$ and an unknown quantity ($1.2$) of the noise signal $p$.

$$\texttt{>> t = x + 1.2 * p;}$$

We would like to remove the interference from the measured signal $t$ so as to recover the original information signal $x$. 

**Measured Signal (with noise)**
Initializing Filter

We will use the function `INITLIN` to create a linear network to discover the relationship between the noise $p$, and the interference it contributes to $t$.

```plaintext
>> [w,b] = initlin(p,t);
```
Adapting using Widrow-Hoff

The function ADAPTWH can be used to adaptively train the network to predict the measured signal t given the original noise signal p. A learning rate of 0.01 is used:

\[ [a,e] = \text{adaptwh}(w,b,p,t,0.01); \]

ADAPTWH returns the network output a and error e throughout the 6 second interval.
Interference Estimated by Filter

Because the network was only given the original noise $p$ as input, it could only estimate the component of the measured signal due to the noise signal $x$. I.e. the network learns to estimate the interference.

The plot above shows the network output $a$, which is an adaptive estimate of the amount of the noise signal $p$ which is in the corrupted signal $x$. 
Estimated Information
(= Noisy Signal - Estimated Noise)

Because the network can only estimate the noise component of the measured signal, its error forms an estimate of the information signal.

The plot above shows the network error $e$, which is an adaptive estimate of the original information signal $x$. 
Error after Filtering

(= Estimated Info - Actual Info)

To see how well the network error estimates the original signal, we can take a look at their difference over time.

\[ d = x - e; \]

The plot above shows that the difference between the network estimate of the information signal and the original signal decreased quickly over the 6 second interval.
ANC (Active Noise Cancellation) Headphones


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 $\nu(k)$. 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

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

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)
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 Cancellation: Principles and Applications, © December 1975, IEEE.
Telephone Echo Cancellation
References

Widrow & Stearns, 1985

Haykin, 1995
Filter form is also called a **FIR (Finite Impulse Response)** filter. In statistics, it is called an **MA (Moving Average)** filter.

\[ \sum \]
\[ w_1 \quad w_2 \quad w_3 \quad w_n \]

- current input
- \( \Delta \)
- \( \Delta \)
- \( \Delta \)
- \( \Delta \)
When we add feedback, we have an IIR (Infinite Impulse Response) filter. In statistics, it is called an ARMA (AutoRegressive Moving Average) filter.
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”.

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

just a way of saying “delay”
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 Santa 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


Problem: Back up a truck so that

\[(x_{\text{trailer}}, y_{\text{trailer}}) = (x_{\text{dock}}, y_{\text{dock}}), \text{ given initial values for } (x_{\text{trailer}}, y_{\text{trailer}}, x_{\text{cab}}, y_{\text{cab}}, \theta_{\text{trailer}}, \theta_{\text{cab}}) \]
Truck-Backer Problem
Training the Truck-Backer

- The truck moves in small time increments $\Delta$

- 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 $t + \Delta$)

- 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

\[(x_{\text{trailer}}, y_{\text{trailer}}, x_{\text{cab}}, y_{\text{cab}}, \theta_{\text{trailer}}, \theta_{\text{cab}}) \rightarrow \theta_{\text{steering}} \rightarrow \text{Truck} \rightarrow (x'_{\text{trailer}}, y'_{\text{trailer}}, x'_{\text{cab}}, y'_{\text{cab}}, \theta'_{\text{trailer}}, \theta'_{\text{cab}})\]
Truck-Controller Combo

\((x_{\text{trailer}}, y_{\text{trailer}}, x_{\text{cab}}, y_{\text{cab}}, \theta_{\text{trailer}}, \theta_{\text{cab}})\)

\((x'_{\text{trailer}}, y'_{\text{trailer}}, x'_{\text{cab}}, y'_{\text{cab}}, \theta'_{\text{trailer}}, \theta'_{\text{cab}})\)

Controller

\(\theta_{\text{steering}}\)

Truck

to be trained

already trained
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

$T = \text{Truck (already trained, weights fixed)},$  
$C = \text{Controller (being trained)}$

$(x'_{\text{trailer}}, y'_{\text{trailer}}, x'_{\text{cab}}, y'_{\text{cab}}, \theta'_{\text{trailer}}, \theta'_{\text{cab}})$

$\text{state}_0 \rightarrow T \rightarrow C \rightarrow \text{state}_1 \rightarrow T \rightarrow C \rightarrow \text{state}_2 \rightarrow T \rightarrow C \rightarrow \cdots \rightarrow \text{state}_{n-1} \rightarrow T \rightarrow C \rightarrow \text{state}_n \rightarrow \text{goal state}$

$\theta_{\text{steering}}$

error (for backprop)
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
Simulations

initial state

time lapse

final state
Simulations

initial state

time lapse

final state