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?

Answer (approx.)
Time-Series Problems: “Predict the Future”

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

Neural Network Function Approximator

Example: Adaline Mimic

Adaline with linear output
Training the Adaline Mimic

Recall the Adaline training rule:

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

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.
```
% 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);
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?

Neural Network Function Approximator

predicted next input

current input

previous input
previous input
previous input

unit time delays

Example: Adaline Predictor

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

Adaline with linear output

current input

predicted next input
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.

applin2

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.

\[ \text{lr} = 0.1; \]
\[ \text{delays} = [1 \ 2 \ 3 \ 4 \ 5]; \]

\[ \text{net} = \text{newlin}(	ext{minmax}(\text{cat}(2,P{:})),1,\text{delays},\text{lr}); \]

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.

\[ [\text{net},\text{y},\text{e}] = \text{adapt}(\text{net},P,T); \]
applin2: actual output vs. target

Figure No. 1

applin2: error = target - output

Figure No. 1
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**

\[
\text{next input} \rightarrow \text{Adaline} \rightarrow \text{predicted next input} \rightarrow + \rightarrow \text{error} \rightarrow \text{predicted output} \rightarrow = \text{output} \rightarrow \text{adjust weights}
\]

= desired next output

= output

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

Here the filter was trained with a sine wave.

MSE Training: 0.0975
MSE free-running: 0.56

The same Filter at 1.75 x frequency
Noise-Reduction Scenario
Filter Learns to Predict the Noise

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

Function being learned

applyin5 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

THE FILTERED-X LMS ALGORITHM

The filtered-X LMS algorithm developed by Widrow [8] seeks the controller coefficients (weight vector) of \( b_0(x^{-1}) \), which minimize the mean-squared error, \( \xi = E\{e^2\} \). 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 \( e(t) \) that is correlated with \( u(t) \) is removed, leaving only \( c(t) \). It is this feature that allows the selectivity property in an ANC system.

The controller weight vector, \( \theta(t) = [\theta_0(t), \theta_1(t), \ldots, \theta_N(t)]^T \), is adjusted in the direction of the gradient

\[
\nabla_a = \frac{\partial}{\partial a} E\{e^2\}.
\]

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

Adaptive noise canceler

Reference input

To 60 Hz wall outlet

Adaptive noise canceler

Delay

LMS algorithms

Primary input

60 Hz interference

EGG preamplifier

EGG recorder

60 Hz Noise in EKG

[EGG signal with noise and after noise cancellation]
Fetal Heartbeat Monitoring

Mostly Mom

Mom+Child

Child

Telephone Echo Cancellation

![Diagram of Telephone Echo Cancellation](image-url)
**References**

- Widrow & Stearns, 1985
- Haykin, 1995

**Additional Nomenclature**

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

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

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(□))
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
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 nonlinear 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

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.
BPTT Application

- The Truck Backer-Upper, D. Nguyen and B. Widrow
- Problem: Back up a truck so that 
  \((x_{\text{trailer}}, y_{\text{trailer}}) = (x_{\text{dock}}, y_{\text{dock}})\), given initial values for 
  \((x_{\text{trailer}}, y_{\text{trailer}}, x_{\text{cab}}, y_{\text{cab}}, q_{\text{trailer}}, q_{\text{cab}})\)

Truck-Backer Problem

![Diagram of truck backer problem](image)
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 $t + △t$).
- 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

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

$$\begin{align*}
(x_{\text{trailer}}, y_{\text{trailer}}, x_{\text{cab}}, y_{\text{cab}}, q_{\text{trailer}}, q_{\text{cab}}) & \rightarrow \text{Controller} \\
& \rightarrow \text{Truck} \\
& \rightarrow (x'_{\text{trailer}}, y'_{\text{trailer}}, x'_{\text{cab}}, y'_{\text{cab}}, q'_{\text{trailer}}, q'_{\text{cab}})
\end{align*}$$

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

\[ (x'_{\text{trailer}}, y'_{\text{trailer}}, x'_{\text{cab}}, y'_{\text{cab}}, q'_{\text{trailer}}, q'_{\text{cab}}) \]

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

\[ \text{goal state} \]

\[ \text{error (for backprop)} \]

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

Initial state

Time lapse

Final state