Time Series Prediction with Artificial Neural Networks

by Jonathon Shlens
Swarthmore College
Computer Science Program, 1999

This research project investigates the ability of artificial neural networks (ANNs) to predict time series. Specifically, this project examined two applications: predicting electrical power demand and earthquake tremors in Los Angeles. The ANNs modeling power demand not only predicted the next day's peak power demand, but also generated a 24-hour profile of the demand for the next day. These ANNs successfully outperformed the predictive ability of a system used by the Los Angeles Department of Water and Power (LADWP). In addiiton, it also performed on par with industry-standard ANNs. The second application earthquake prediction was not able to predict earthquake tremors. Due to the voluminous amounts of data, better results may only be obtained through researching methods of on-line feature extraction and data preprocessing.

1 Introduction

The field of artificial neural networks (ANNs) has not only led to insights in the understanding of the brain; it has also produced numerous applications. One powerful application of ANNs is the ability to simulate highly complex systems—systems that can not be reliably predicted by traditional, algorithmic computer simulations. Electrical power demand and seismographic readings are both model examples of such systems. They contain multiple variables with relationships too intricate to state explicitly. Generating mathematical equations to describe this behavior is time-consuming. Furthermore, performance is poor because the equations can not deal with the complexity of the relationships in the system. For example, it is difficult to write an equation to predict how the time of sunrise and sunset will effect the electrical power demand at 8:00 am in Los Angeles during the summer.

This paper explores a different paradigm for modeling complex systems. Instead of generating explicit equations to model a system, this project exploits the inherent learning and pattern recognition ability of ANNs. Going beyond simple toy examples [11], we investigate how to optimize ANN performance given noise-ridden and complex data. Not constrained to the "biologically plausible" domain, improvements to ANN performance will at many times be purely engineered. However, without this constraint it is easy to see that at many times "the field of ANNs is more of an art than a science [7]."

2 Theory

2.1 Artificial Neural Networks

2.1.1 Multilayer Perceptrons

Because several in-depth discussions of ANNs exist [2, 7, 8], this section will be a brief overview focused towards this work's applications. A perceptron is a simple mathematical model of a single neuron. A multilayer perceptron corresponds to a sequential network of interconnected neurons as diagrammed below.

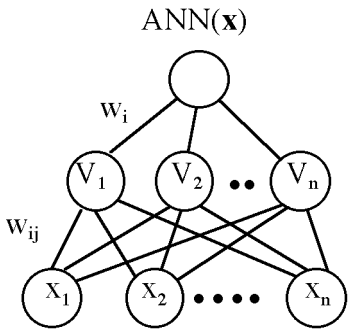

Figure 1: Schematic diagram of multilayer perceptron

The output voltage of neuron i, one circle, is $V_i = f(\sum_j w_{ij}x_j + b_i)$ where the following terms apply:

- f is a nonlinear transformation
- x_n is the n'th input to neuron i
- w_{ij} is the connection strength to neuron i from input x_j
- b_i is the bias of neuron i

The output of a perceptron is a nonlinear transformation applied to a weighted sum of inputs. The nonlinear transformation is often a sigmoid function $f(x) = \frac{1}{1+e^{-x}}$ because it is bounded as $\lim_{x\to\infty} = 1$ and because it's derivative is easy to express $f'(x) = f(x)[1 - f(x)]$. If we let x_n be the n'th input, the above figure (with one output) can be written as:

$$ANN(\vec{x}) = f(\sum_i w_i f(\sum_j w_{ij}x_j + b_i) + b)$$

The composition of sigmoid functions with adjustable parameters $\{w\}$ form a functionally complete system [2]. In other words, provided enough hidden nodes n, there exists a set of weights $\{w\}$ such that the above function will model any arbitrary function $F(\vec{x})$ to arbitrary precision e:

$$\forall \vec{x} : |F(\vec{x}) - ANN(\vec{x})| < e$$

Finding the appropriate weights to model the data could be a simple matter of trial-and-error, but a more systematic method is to use nonlinear regression.

2.1.2 Nonlinear Regression

Finding the set $\{w\}$ to best model $F(\vec{x})$ is a nonlinear optimization problem. It is first necessary to decide on a performance measure. A standard measure is the sum-squared error measure:

$$E = \frac{1}{2}\sum(F(\vec{x}) - ANN(\vec{x}))$$

which sums over all training patterns $\vec{x}$. Clearly, the ANN fits $F(\vec{x})$ best where E is minimized. We can now recast the problem as follows. For an ANN with k weights, we have a $k+1$ dimensional graph where the first k axes are the weight values $\{w\}$ and the $(k+1)$'th axis is E. Each point on the graph represents the performance of a particular set of weights $\{w\}$. This generates a k dimensional error surface much like below.

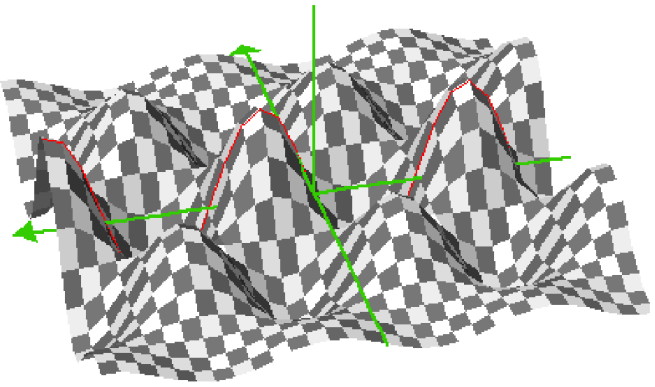

Figure 2: Example of 2-dimensional error surface where the z axis represents the error.

The above is a simplified example of a 2-dimensional error surface for a hypothetical ANN with two weights w_1 and w_2 on the horizontal plane and E on the vertical axis. This simple example demonstrates a key problem – there exist several local minima in E (i.e. where $\vec{\nabla}E = 0$). There are many well-researched solutions to this difficulty: simulated annealing, a pseudo-Newton's method approach, genetic algorithms, and gradient descent [1, 2, 3, 13]. This project will focus on the latter technique, gradient descent, with a few engineered improvements such as a momentum term and an approximate computation of the second derivative (Hessian) of E [8].

Starting at a random point on the error surface (i.e. a random set of $\{w\}$), the ANN will compute the partial derivative of the error with respect to each weight $-\frac{\partial E}{\partial w}$ after all training patterns and update each weight accordingly $w \rightarrow w + -\eta\frac{\partial E}{\partial w}$. It is notably difficult to compute $-\frac{\partial E}{\partial w}$ for the inner weights of $ANN(\vec{x})$. This is resolved by using the chain rule, termed back-propagation [12]. Depending on which set of random weights are selected, the ANN will finish at different local minima. This necessitates running the network multiple times to obtain several samples of minimal error values.

2.1.3 Recurrent Neural Networks

When time effects the value of the function being modeled, it is beneficial to add some time dependence in the ANN. Standard ANNs are limited because they sequentially process the data with no reference to time. A common method of adding time significance to an ANN is to make the network fully recurrent. This is analogous to the difference between combinatorial and sequential logic circuits. The main idea of this strategy is to provide a weighted feedback connection to all perceptrons and compute $-\frac{\partial E}{\partial w}$ summed over an entire time-series [3].

A partially recurrent network structure is a heuristic version of a fully recurrent network. A partially recurrent network only feeds back certain nodes and it retrains after each point in a time-series.

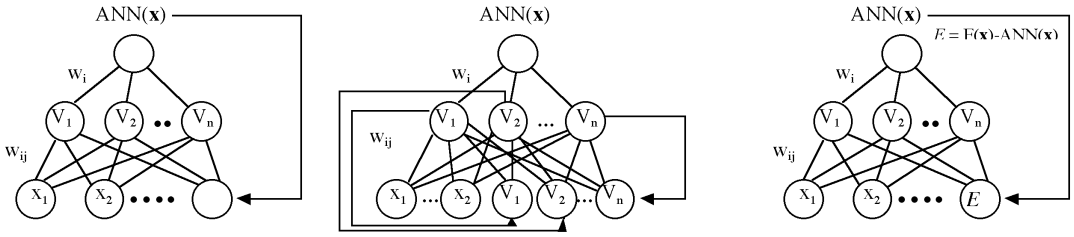

Figure 3: Schematic diagram of Jordan, Elman and residual-feedback style architectures

From left to right, these diagrams represent a Jordan-style, Elman-style and a residual-feedback style recurrent network. A Jordan-style network feeds back the previous output as an input [9], while an Elman style network feeds back the previous hidden nodes' values as inputs [6]. Both styles allow for the feedback weights to be modified via back-propagation in the exact same manner as other weights. The final style, residual feedback, is an architecture purely derived for time-series prediction [4]. This architecture feeds back the previous prediction's errors as an input. There is a drawback with the residual feedback style though; it can only predict one time step in advance because it requires the previous time step's target value.

2.1.4 Time Series Analysis

A time series is a set of discrete data recorded from real-world phenomena. A time-series $T = \{x_1, x_2, ..., x_{n-1}, x_n\}$ is termed *"of order p"* when $x_t = G(x_{t-1}, x_{t-2}, ..., x_{t-p}) + \epsilon_t$. The function G is the underlying pattern that determines the system while ϵ_t is the inherent noise or measurement error in the data.

Because of the random error term ϵ_t, we do not want an ANN to model the scatter plot of T but rather we want it to model the pattern G. The problem is we only have data for T and the magnitude and effect of ϵ_t is unknown. A common solution to this dilemma is to use cross-validation [2]. Cross-validation is performed by dividing the data into two sets, a training set and a testing set. Training is usually measured in epochs, the number of training set presentations. Typically as the number of the training epochs increases, we see the following graphs regardless of the type of ANN model:

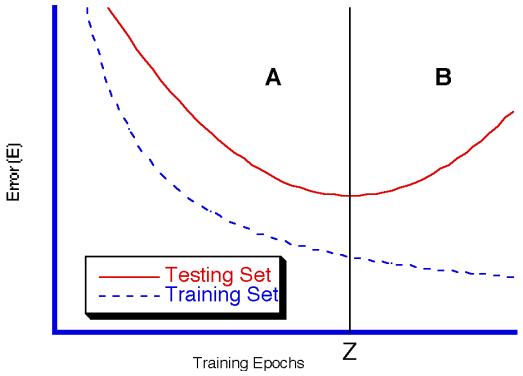

Figure 4: Training and testing error as a function of training epochs

The training-set error measures how well the ANN models the data T. Because we never train on the testing-set and the error terms ϵ_t are uncorrelated, the error of the testing-set is a measure of how well the model follows the common pattern G or *generalizes* between data sets. In section A of the graph, the testing-set error just begins to learn the pattern in the time series T, indicated by the decreasing error . At Z epochs the testing-set error is minimal – the ANN has discerned the function G at its optimal level. Beyond this minimum in section B the training-set error continues to decrease while the testing-set error increases.

In this section the ANN begins to model the noise terms in the model as well. The testing set increases because the ϵ_t terms are random. Consider the below figures.

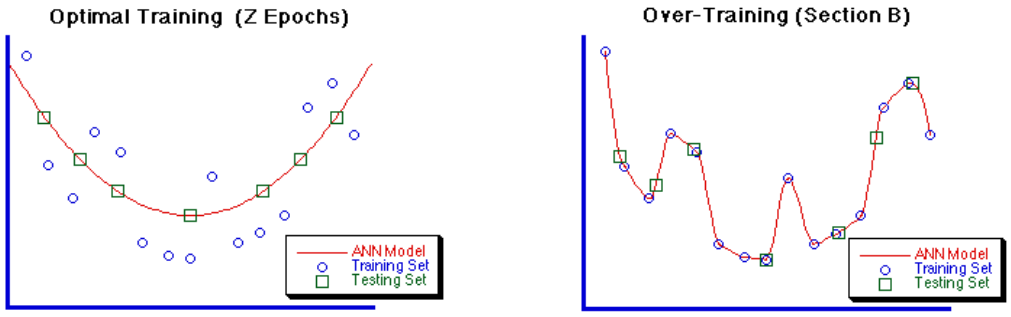

Figure 5: Example of optimal training and over training in an ANN model

Modeling the error (section B) in the training-set produces a "jagged" model which does not reflect the pattern in the data set. A "smooth" fit — corresponding to Z epochs in the previous figure — allows for a good generalization of the pattern G. The extra jaggedness of the over-fitted model is equivalent to the standard deviation of the error. The standard deviation or variance of the model therefore is a trade-off with how well the model fits the training data. The latter is measured by the mean squared error, termed the bias. This bias-variance trade-off is more formally derived in Bishop [2].

A more intuitive undertstanding of bias and variance is the following analogy: shooting paint pellets at a bull's eye. The mean of the percent error measures how close we are to the bull's eye. The standard deviation of the percent error measures how spread out the

paint splatters from where the pellet hits. Thus, the mean represents how accurate we are and the standard deviation represents how certain we are of our accuracy. In practice the bias is easy to minimize while minimizing the variance is usually the focus of ANN models.

3 Electrical Power Demand

3.1 Background

The need to predict electrical power demand is a vital concern for utility companies. By predicting the next day's demand, utility companies can optimally allocate their human resources and costs associated with starting up various power plants. For the LADWP and other companies, it is common practice to employ some method for predicting the next day's power demand. Because of the complexity of the data, the LADWP only tries to predict the peak, or maximum, power for the next day; they do not even attempt to predict a 24 hour profile of the next day's demand. Claiming a need for "industrial secrets," the LADWP will not divulge its predictive algorithm. However, from data supplied by the LADWP, we can ascertain the predictive ability of their model:

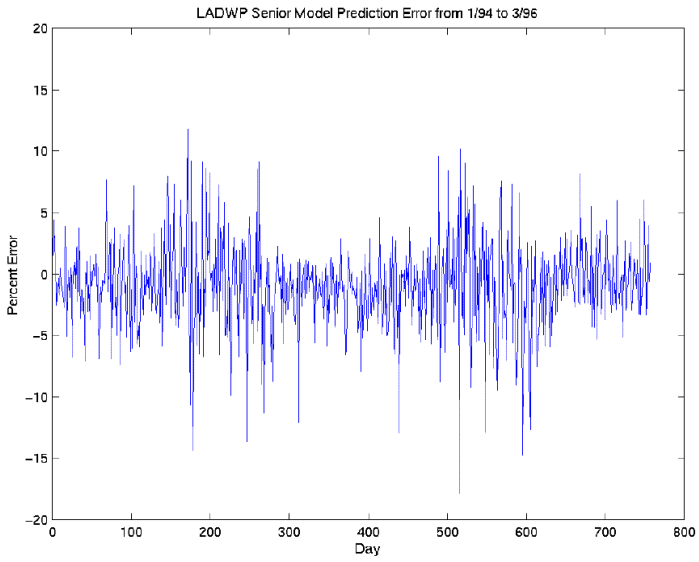

Figure 6: LADWP prediction error — bias = -0.91, variance = 3.54

With data for the predictive ability of the LADWP model, the first project goal was to produce a comparable model to predict the peak power demand. From this work we hoped to eventually build a model to predict a 24-hour profile. The peak power demand contains many of the complex relationships seen in a 24-hour profile of power demand. Therefore, it provides a good starting point from which to predict the 24-hour profile, the final goal of this project.

The peak power demand over one year in the Los Angeles area is not trivial. It is complex with noticeable spikes in the summer days and high frequency dips prevailing throughout the series.

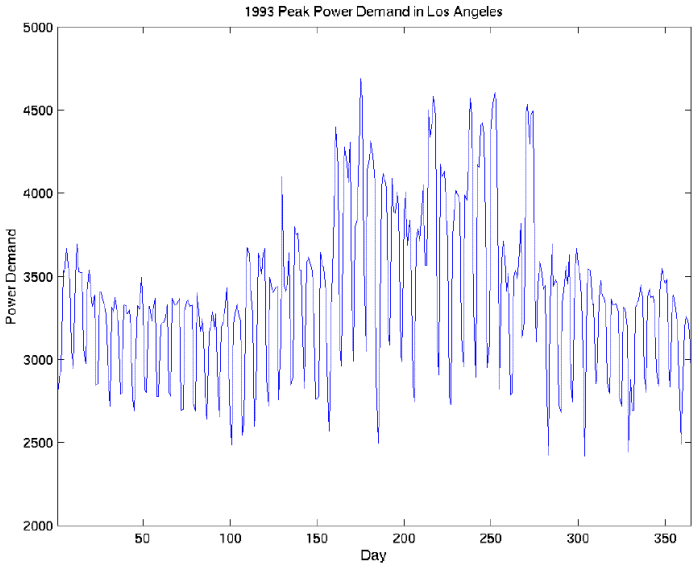

Figure 7: Peak power demand in Los Angeles during 1993

The problem can be simplified by first realizing that the high frequency spikes are the transitions between the weekends and weekdays. Dividing the series into these two groups makes this evident.

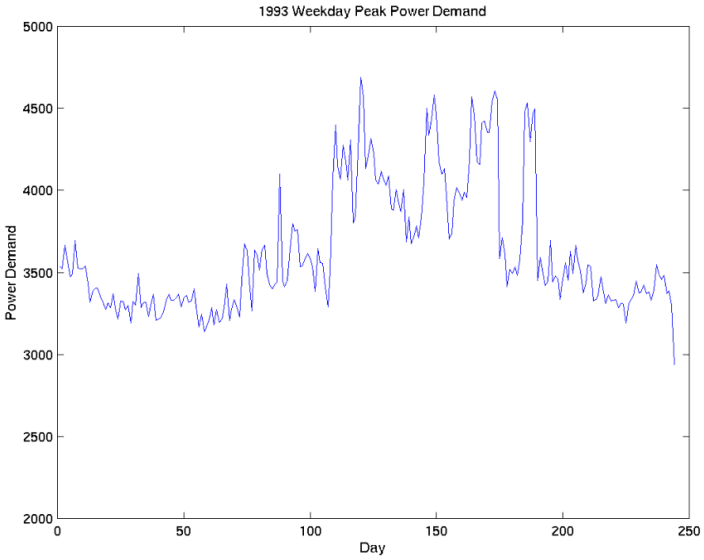

Figure 8: Weekday peak power demand in Los Angeles during 1993

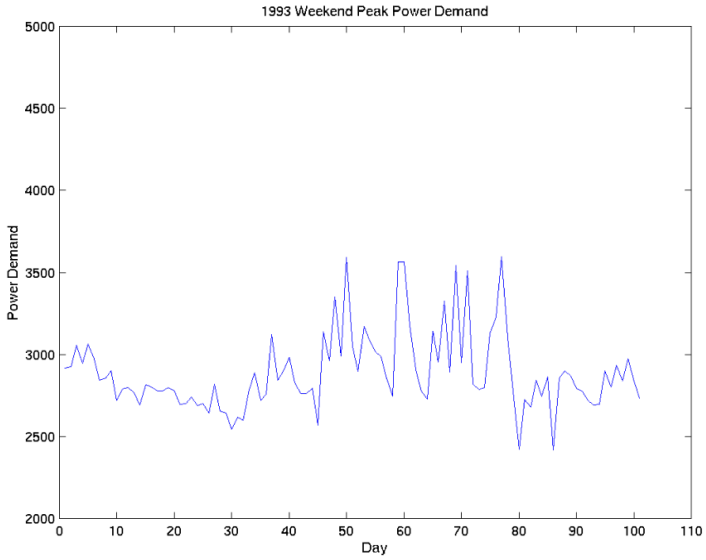

Figure 9: Weekend peak power demand in Los Angeles during 1993

Furthermore, counting the days on the x-axis appears to suggest high amplitude spikes throughout the summer and relative regularity during the winter. Dividing the data into areas as such (e.g. weekends/weekdays and summer/winter) is a powerful tool used throughout this paper.

Hourly power prediction is roughly 24 times more complex. For each day there exists 24 points signifying the power demand at each hour, versus 1 point/day for peak power demand. Below is a graph for the hourly demand for one week.

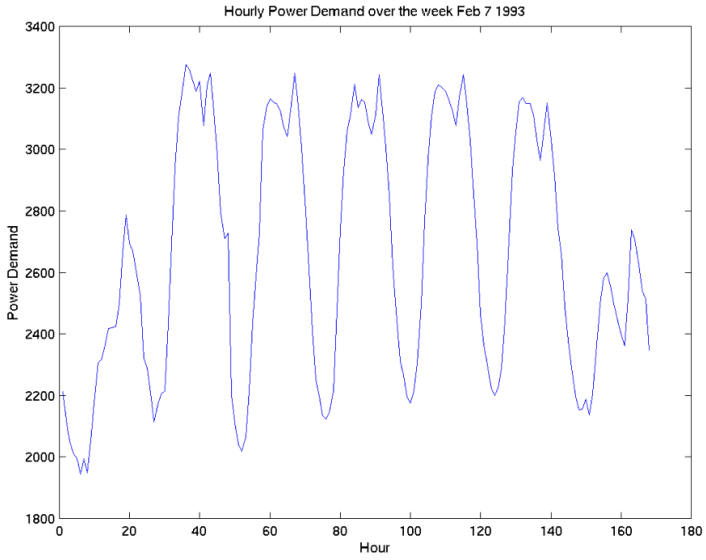

Figure 10: Hourly power demand during the week of Feburary 7, 1993

It is notable how this above graph relates to the previous peak power demand graph. The top points of the seven "bumps" (days) correspond to seven sequential points in

the peak power demand graph. Over the course of a weekday the power undergoes preciptious increases and decreases following a pattern related to the workday. Again, we can find a pattern by *dividing* the data by hour. Each hour contains far more regularity over the course of a year. More about exploiting this regularity will be discussed in the Methodology and Results section.

3.2 Methodology

The Methodology and Results section are closely realted for this application domain. Most of the work consisted of systematically testing each parameter or method in order to optimize the network's overall predictive ability. Thus, the final goal of the work was to in fact derive the best Methodology. With that in mind, this section will only state the "best" methodology. The Results section will highlight the data behind some of the more noteworthy searches performed in obtaining the Methodology.

3.2.1 Peak Power Prediction

Network Inputs

Presenting the ANN's with the optimal set of inputs is critical. Too few inputs gives too little information while too many inputs gives too much information (or dimensions) for the ANN to search through. Finding the optimal subset of previous power demand values, previous temperatures, predicted temperatures and other meteorological data is necessary. The optimal set consisted of the following normalized inputs:

- power demand for the previous day and one week ago
- the previous day's temperature
- the predicted temperature for the current day
- a weekday index and a seasonal cosine index

Any patterns which we discovered such as weekly and seasonal fluctuations need to be inputted in the form of indices. The weekday index consists of the following:

Sun	Mon	Tues	Wed	Thurs	Fri	Sat
1	-0.4	-1	-1	-1	-0.4	1

The idea behind the index is to allow the network to distinguish between weekdays and weekends and allow for the occasional three-day weekend. The seasonal cosine index consisted of a year-long period with peaks at winter and summer solstice. This allows the network to distinguish between the high amplitude spike period in the summer and the relative regularity in the winter.

Network Architectures

All of the networks used were 2-layer, feed-forward, back-propagation, partially-recurrent networks. The three types of recurrent networks were the aforementioned Elman-style, Jordan-style and residual-feedback style. The Results section will highlight the best performing, residual-feedback network in comparison to the other architectures.

Training Methods

For cross-validation the training set consisted of the peak power demands between Jan 1991-Dec 1993 and the testing set consisted of the demand between Jan 1994 -March 1996. We removed outlier days such as holidays and days from the Northridge earthquake (around Jan 18, 1994) from the data. These days had uncharacteristically low values and did not reflect the overall pattern of power demand.

By training on the training set and testing on the testing set, the network would in fact predict the power demand two years in advance[1].

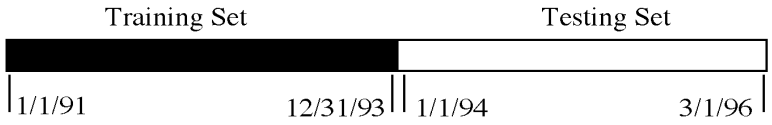

Figure 11: Cross-validation requires dividing the data into a training and testing set

In this application, it is of course best to minimize the forecast-advance or prediction horizon. Therefore, we used an online training method where by after each month the network retrained itself on the month it was just tested on. This minimizes the forecast-advance to only a month.

Data Division

An interesting methodology, which we will analyze in the Results section, was the use of data division. For many situations in the peak power prediction, there existed such stark differences in the data such that training a **separate** network on each set of data minimized the overall error of the network. As hinted at earlier, for peak power prediction we divided the network by: weekdays and weekends and by summer and winter – resulting in four networks for predicting the different parts of peak power demand.

3.2.2 Hourly Power Prediction

A tremendous suggestion by Connor et al [4] is to divide the data by hour. This generates 24 times as many networks to run but vastly simplifies the problem. By dividing the data by hour, we reduce the problem to the difficulty level of peak power prediction since there is now one value to predict for each day. We are also easily able to compare the prediction ability for each hour to the peak power prediction of the previous section (and the LADWP prediction model). Since the residual-feedback architecture only functions for one time step in advance[2], we would not be able to use a residual-feedback network to predict the power demand one day in advance. Having a separate network for each hour allows us to use a residual-feedback architecture. Using this suggestion all of the work from peak power prediction can easily be ported to hourly power prediction such as network inputs, network architecture, training methods and data division.

[1] At the end of training, an ANN will have been exposed to all of the actual data up until Dec 31, 1993. With this training, it will predict the power during Jan 1994 but it will also predict the power in March 96 – without having been retrained on any new information. In effect, the network is predicting as far as two years in advance

[2] A residual feedback network requires the previous actual value in order to predict the next time step. For example, if a residual feedback network tries to predict 11AM , it needs as an input the actual power demand at 10AM. Hence, it is restricted to predicting one hour in advance.

3.3 Results

3.3.1 Peak Power Prediction

The peak power demand provided a good 'proof-of-concept' starting point. Our goal here is to merely hone a methodology for later application to hourly prediction. In this preliminary part, we did not see it necessary to test all network architectures nor test all sections of the data set. Therefore, these results are limited to Elman-style network architectures and predicting weekends only. Below is a percent error plot of the best ANN model over the testing period for weekday peak power prediction.

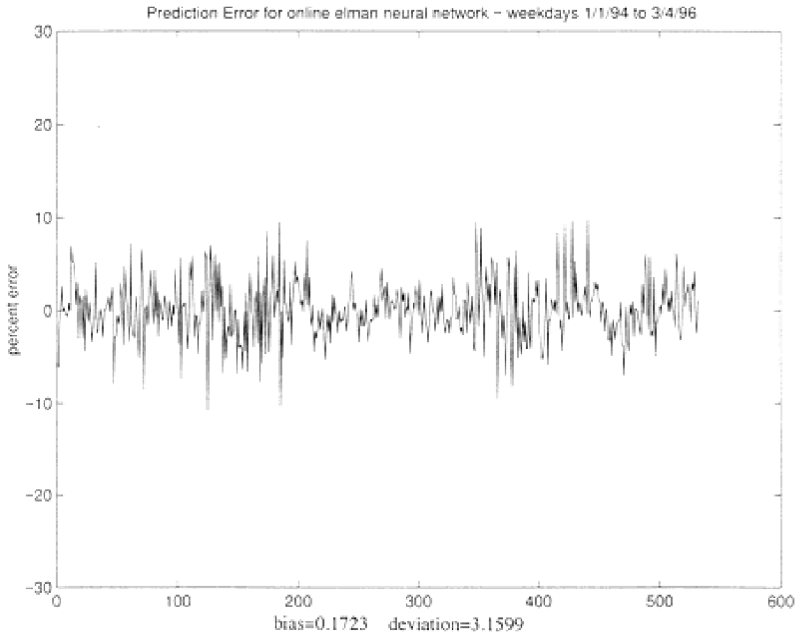

Figure 12: Prediction error for Elman-style network on weekdays

	Bias	Variance
Neural Net	0.17	3.16
LADWP	-0.64	3.12

Clearly, these results show that the ANN model reached the predictive ability of the LADWP model.

In addition to deriving the previous methodology, we gained some unexpected results from data division into winter and summer networks. Surprisingly, the networks learned summers best when trained on the full year where as the ANN predicted winters best when trained on winters alone.

	Summer Prediction		*Winter Prediction*	
(training on:)	bias	variance	bias	variance
full year	0.04	**3.59**	-0.16	3.56
winter only	2.23	4.60	0.31	**2.83**
summer only	0.86	4.04	n/a	n/a

The best training method for predicting summers and winters is **bold-faced**. Incidentally, training on only the winter to predict the summer (to check if the summer data is just useless) resulted in extremely poor performance. A possible explanation for these resultsis that the winter contains more regularity than the summer. The winter could provide some stability and regularity to the more erratic pattern in the summer. An analogous result surfaces in hourly prediction as well.

3.3.2 Hourly Power Demand

The major difference with peak power prediction is that different data divisions proved beneficial to the ANN. Unlike peak power demand, dividing the data by season provided no improvement in performance. There appeared to be more regularity across seasons for each hour than the peak power. In addition to hourly division, the only other data division used was division by weekday and weekend. This division functioned analogously to dividing by season in peak power demand.

DAYTIME (10 AM)	*Weekend Prediction*		*Weekday Prediction*	
(training on:)	bias	variance	bias	variance
full week	0.19	**4.11**	0.31	3.26
weekdays only	n/a	n/a	-0.01	**2.63**
weekends only	0.24	6.40	n/a	n/a

NIGHTTIME (4 AM)	*Weekend Prediction*		*Weekday Prediction*	
(training on:)	bias	variance	bias	variance
full week	0.50	**2.76**	0.06	**2.70**
weekdays only	n/a	n/a	0.01	3.09
weekends only	0.11	4.19	n/a	n/a

Again, it is best to predict weekdays by training on weekdays only but to train the entire year to predict weekends. This same training relationship is seen only during the work hours (7AM-7PM). The night hours contain such consistent regularity that weekday division is harmful.

Residual feedback proved to be the best network architecture for this application. It consistently, slightly outperformed Elman and Jordan style networks in several tests.

	Variance at predicting hour:			
	3AM	3PM	5PM	12AM
residual-feedback	**2.57**	**4.37**	**4.98**	**3.11**
Jordan-style	2.74	4.54	5.07	3.26
Elman-style	3.04	4.50	5.01	3.66

This sampling of hourly predictions is indicative of the performance improvement. There are occasions when the residual-feedback notably outperforms the others but on certain hours, it only predicts on par with the other styles.

The final results can now be summarized as follows:

	Bias	Variance	Mean of the absolute value
Neural Net Model	**0.11**	**3.21**	**2.36**
LADWP Model	-0.91	3.54	2.68
ANNSTLF	n/a	n/a	2.64

The above comparison needs some explanation. The neural network model predicts the power demand for all 24 hours in a day where as the LADWP model predicts only the peak power demand for that day. In other words, the neural net predicts a 24 hour profile (24 times as many points) better than the LADWP can predict just the peak power for that day. The third column and row is included in order compare this model with a recent model published in IEEE [10]. ANNSTLF is a neural network model used commonly by industry, which likewise predicts all 24 hours of power demand in advance. The ANNSTLF utilizes both different pre-processing techniques and network design. The ANNSTLF uses a hierarchy of non-recurrent neural networks — there is one hierarchy of networks for predicting each hour in the day. Three networks in the first hierarchy layer are trained specifically to extract the weekly, daily and hourly fluctuations, respectively. There is one network on the upper layer of the hierarchy. This network is actually an auto-regressive model which linearly combines the outputs of the lower level networks. The ANNSTLF model uses similar network inputs plus humidity. The number quoted for ANNSTLF is an average (mean) of the corresponding 5 test errors that the authors quoted in their paper (the five values ranged from 2.15 to 3.22). It should be strongly noted though that ANNSTLF was trained and tested on separate data. *This demonstrates that the model produced and examined in this research not only outperforms the LADWP but also is on par with the industry's best ANN solution.*

We can analyze the specific strengths and weaknesses of our model by examining specific subsets of the data. Dividing the prediction error of the model into weekday and weekend performances reveals that the neural network significantly outperforms the LADWP model on the weekends.

	Weekend Prediction		*Weekday Prediction*	
	bias	variance	bias	variance
Neural Net	**0.13**	**3.07**	**0.06**	**3.55**
LADWP Model	-0.63	3.12	-1.58	4.32

Both the standard deviation and the mean of the error for the ANN model are significantly below that of the LADWP during the weekends. For the weekdays the ANN only slightly improves on the LADWP performance.

Likewise, we can examine on which hours the ANN outperformed the LADWP model. Generally, on all hours the ANN attained a mean percent error below the LADWP model. What is most interesting is comparing the standard deviations of the error.

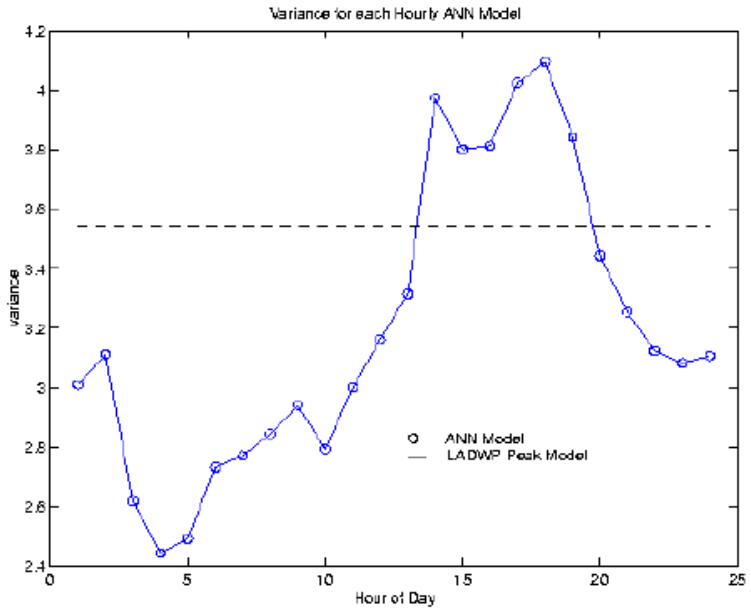

Figure 13: Prediction performance (variance) of ANN model divided by hour

The solid line is the hourly prediction and the horizontal line is the LADWP peak power prediction. This graph shows that these models outperform the LADWP peak power prediction during the morning and night hours but not during the late afternoon and evening.

4 Earthquake Prediction

4.1 Background

Seismographic readings (ground velocity measurements) for earthquakes often generate a precursor wave up to 30 seconds in advance of the main tremor. Below is the seismograph reading of a complete earthquake tremor. Note the motion before the large amplitude spikes.

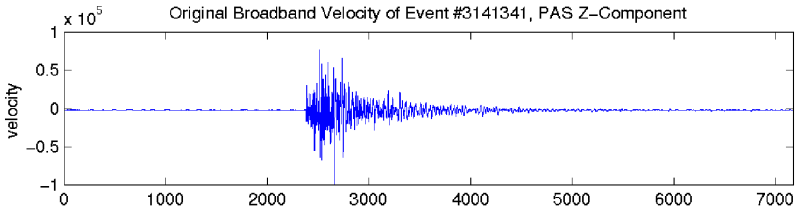

Figure 14: Seismograph velocity recordings of a typical earthquake

The motion prior to the main thrust of the earthquake is the precursor tremor. Previous geophysical research has shown a correlation between features of the precursor tremor and the strength and location of the main tremor. Already, studies have produced ANNs capable of predicting earthquake tremors with limited ability [5]. However, these studies have been more like "proof of concept" papers. They limit themselves to earthquakes

of certain magnitudes and locations and use unrealistic data transformations requiring foreknowledge of the entire earthquake. It is the hope of this work to produce a practical system not limited to those constraints. We will use the voluminous amounts of data available from the network of seismographs in the Los Angeles area in order to predict the epicenter and the strength of the oncoming main tremors.

4.2 Methodology

Because both applications deal with simulating time-series, predicting seismograph readings has a very similar methodology to predicting electrical power demand. One important difference with this application is that there is a slightly different goal. With power demand the only concern is minimizing the prediction error. With seismograph prediction we are concerned with two conflicting goals: both minimizing the prediction error and maximizing how far in advance the network can predict. We have termed the latter the "forecast-advance." A solution for these conflicting goals will be to create a group of networks, each predicting with a different forecast-advance. Generally, the smaller the forecast-advance, the more reliable the prediction. As the time to the onset of an earthquake approaches this group of networks will output an improving prediction for the oncoming main tremor.

The network receives a window of previous seismograph values. How far back this window is in time is a function of the specified forecast-advance for the network. Before inputting these values, a tremendous amount of pre-processing is performed. The seismograph samples at 20 points per second, generating waveforms of about 6000 points (see earlier figure). These graphs contain far too much fluctuation and high-frequency noise for the networks to process. We therefore need to produce an *online envelope* of the waveform as input. The phrase *online envelope* is emphasized because a traditional envelope uses the Hilbert transformation of the entire waveform and thus, can not be used in an online, functioning model. To solve this problem we created an algorithm based on peak-detection and interpolation, which generates the envelope as the waveform arrives.

There are two other problems with the data: computational time and scaling with orders of magnitude. 6000 data points per component per station per earthquake — in a data set of 3 components, 3 stations and 20 earthquakes — generates too many data points for the network to process in a reasonable amount of time. Hence, waveforms need to be down-sampled. Also, earthquakes vary by orders of magnitude (e.g. from 10^3 to 10^7) meaning that the large earthquakes would drown out the smaller earthquakes and quickly saturate the sigmoidal transfer functions. Therefore, before normalizing the seismograph data, we need to take the logarithm of the data in order to allow the network to deal with all magnitudes of earthquakes. Taking all of this into account, the seismograph data undergoes the following pre-processing steps:

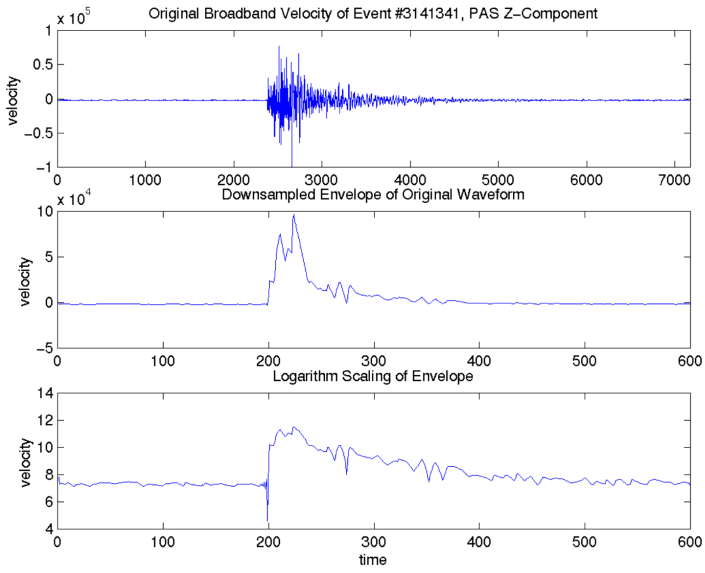

Figure 15: Pre-processing steps for earthquake seismograph

Envelope → Down-sampling (10:1) → Logarithmic scaling → Normalization

Finally, we also added the PGA, the peak ground acceleration, of the waveform sampled thus far. The PGA has been shown to have a strong correlation with the overall magnitude of the oncoming tremor. We again divided the data set into a training set and a testing set — the first for training and the second for online testing and updating.

A residual-feedback network can not be used in this application. One needs the previous prediction's errors in order to predict the next value in a residual-feedback network. Since we are predicting as far as 30 seconds in advance, how well the network performed will not be known for as much as 30 seconds. The large forecast-advance prevented us from using a residual-feedback architecture. In the previous application we found that the Jordan-style slightly outperformed Elman-style networks. For this reason we chose to use a Jordan-style network. More will be discussed about this in the Discussion section.

4.3 Results

The general strategy behind these results is to see how well a neural network can use the current readings at two locations in order to predict the seismograph readings at a third location. In addition to predicting the oncoming tremor, this model can also be used to predict the epicenter quickly. We can perform the latter by simply predicting in parallel the waveforms for all of the seismographs in the Southern California Seismic Network (SCSN). Whichever waveform has the largest peak provides a rough measure of the epicenter. For these preliminary tests we gathered the following data from the Southern California Earthquake Center Data Center:

- 20 Northridge aftershocks between 01/18/94 and 05/18/94 of magnitude 4 or greater
- Broadband data-stream (3-component velocity) recorded at 20 samples/second

- Station recordings at Pasadena (PAS), Calabassas (CAL) and University of Southern California (USC)

We performed two sets of tests on the data. We attempted to predict the current seismograph value at one station (PAS) from the previous seismograph readings at the two other stations. We also tried to predict the future value of one station (PAS) based on the past values of that same station (PAS). The second test provided a base test to compare against — specifically, whether the network can predict, at the minimum, the value of the same station it is receiving input from. Below is a chart of the mean and standard deviation of the errors for predicting PAS with different forecast-advance's.

	2.5 sec Forecast-Advance		*15 sec Forecast-Advance*	
(trained on:)	bias	variance	bias	variance
PAS	1.98	18.50	13.60	38.51
CAL and USC	4.35	20.35	5.84	30.24

The above overall error values only hints at the inability of the model. The real fault in the model can only be seen in the total error plot.

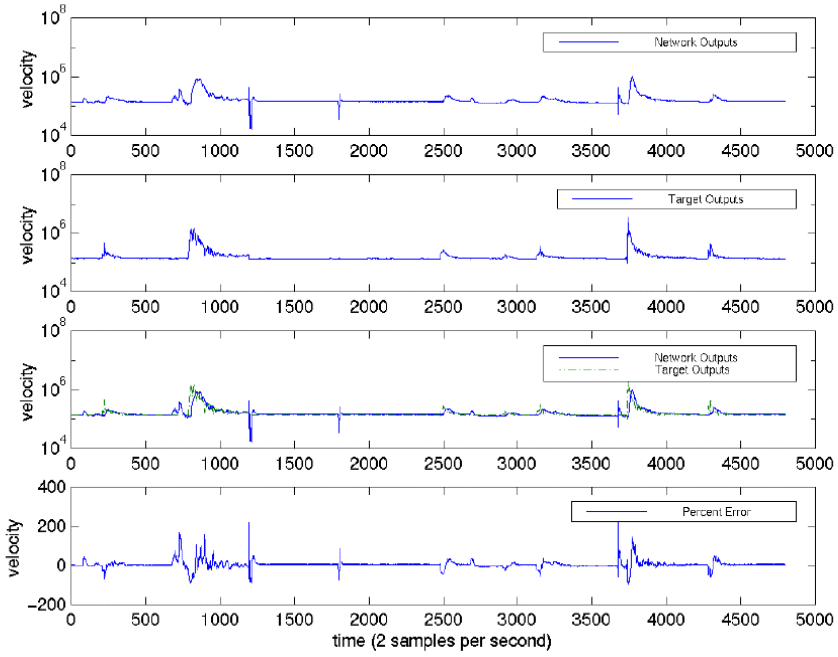

Figure 16: Prediction performance of ANN model on various earthquake tremors

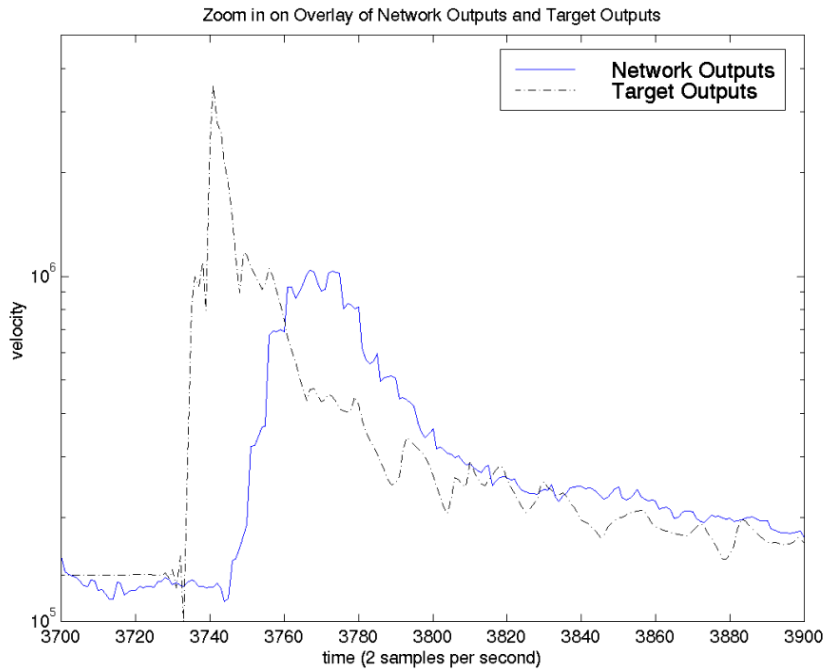

Figure 17: Zoom-in on overlay of prediction and actual tremor values

The first plot is an overlay of the target and network output over a testing set. The second figure is a zoom-in on the network output and error for predicting PAS from CAL and USC at 2.5 seconds in advance. The zoom-in on the graph is typical of how the network consistently "predicts" the onset of the main tremor late and usually underestimates it. Since the network is online, it retrains itself on the oncoming tremor. Thus, the network only starts predicting the main tremor upon its arrival. What these plots show, especially the zoom-in, is that the network is not predicting the earthquakes — **it is only "reacting" to them after the fact through retraining**. Even for the easier base test of predicting PAS from just PAS inputs, the network performs just as poorly. Thus, it appears that it does not matter where the input signal is arriving from because all signals appear to be just as (un)beneficial.

5 Discussion

The ability of ANNs to predict the behavior of a complex system such as electrical power demand demonstrates the applicability of this field of artificial intelligence research. Predicting electrical power consumption has proven to be a successful endeavor. Hopefully, in the future ANNs will likewise be as successful with predicting earthquakes. Predicting electrical power demand is a useful application and one that ANNs can fulfill quite well. Better results could still be obtained with better resources. Adding other meteorological inputs (humidity), reducing the forecast-advance and simply running more trials can lead to a better predictive ability.

More work needs to be performed before predicting earthquakes rises from a theoretical topic to one used in practice. Retesting the data on an Elman-style network could provide

a good start. Because of the complexity of the data, the hidden activations of the network could provide a more *meaningful* input than simply its previous prediction. However, most of the problems for the earthquake prediction model appear to be in pre-processing the data. Even after the already, voluminous amounts of pre-processing performed, we need to perform even more. The most notable problem is the logarithmic scaling. This non-linear transformation requires the input to be positive. In order to solve this problem, we first offset the velocity values so that the minimum velocity is above zero. Even with the offset the output appears to falsely characterize the inputs at points. In the previous figure we can see this by the large downward spike, which occurs at the minimum of the seismogram, even though there is no such equivalent dramatic change in the original data set; this is due to the nonlinearity of the transformation. What value to choose for the minimum offset requires prescience of the entire waveform, making the ideal offset unattainable.

We also need more feature extraction for the model. The PGA is a good example of other geophysical features to search for in the waveform. Other ideas include extracting the CAV (Cumulative Average Velocity) and spectral information. The CAV is difficult to scale, yet again because the CAV varies by orders of magnitude. The complete spectrum can not be inputted to the network so perhaps an enveloped, downsampled version needs to be used.

The first paper produced by Dowla using ANNs to predict earthquakes garnered better results than this work. There are two main reasons for this. Dowla's paper restricted the inputs to earthquakes between magnitude 4.1 and 4.5 with nearby epicenters, making the problem dramatically easier. They did not have to deal with scaling orders of magnitude for the time-series values nor the CAV input. Secondly the other paper used unrealistic pre-processing techniques which require prescience of the entire waveform before it arrives. These techniques included using a Hilbert transformation in the envelope, and pre-knowledge of the PGA for correct scaling. This work at the minimum shows that further study into feature extraction and preprocessing, especially when dealing with orders of magnitude, could be quite beneficial for this model. More collaboration with geophysicists and other experts in the field is needed in order to determine other possible feature extractions. As of right now, the ANNs are presented too much data to make sense of. By highlighting areas in the data relevant to earthquakes, we can make ANNs a more feasible solution to earthquake prediction.

6 Acknowledgements

Funding from the SURF office supported this research. I would especially like to thank Professor Rodney Goodman and John Lindal for their guidance and sponsorship over the past two summers. I would like to thank Professor Tom Heaton and Dr. Cheng Tong from the Geophysics Department for their support with earthquake prediction. Also I would like to thank my advisor Professor Lisa Meeden for tremendous class offerings. The LADWP provided the electrical power demand data and the following organizations provided the seismographic data: Southern California Earthquake Center Data Center, Seismological Laboratory — Caltech, Southern California Seismic Network, Terrascope Network — Caltech, U.S. Geological Survey, AMOES and

the ANZA seismic network, UCSD.

References

[1] R. Belew, J. McInerney, and N. Schraudolph. Evolving networks: using the genetic algorithm with connectionist learning. In Langton, Taylor, Farmer, and Rasmussen, editors, *Artificial Life II*. Addison-Wesley, 1991.

[2] C. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[3] M. Caudill and C. Butler. *Understanding Neural Networks: Computer Explorations*. MIT Press, Cambridge, 1993.

[4] J. Connor, R. Martin, and L. Atlas. Recurrent neural networks and robust time-series prediction. *Neural Networks*, 5:240–254, 1994.

[5] F. Dowla, S. Taylor, and R. Andersen. Seismic discrimination with artificial neural networks: Preliminary results and regional spectral data. *Bulletin of the Seismological Society of America*, 80:1346–1373, 1990.

[6] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[7] S. Haykin. *Neural Networks: A Comprehensive Foundation*. MacMillan, New York, 1994.

[8] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA, 1991.

[9] M. Jordan. Serial order: A parallel, distributed processing approach. In D. Rumelhart and J. Elman, editors, *Advances in Connectionist Theory: Speech*. Erlbaum, Hillsdale, 1989.

[10] A. Khotanzad and A. Abaye. ANNSTLF - a neural networks based electric load forecasting system. *IEEE Transactions on Neural Networks*, 8:843–845, 1997.

[11] M. Minsky. *Perceptrons*. MIT Press, Cambridge, 1969.

[12] D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, Cambridge, 1986.

[13] D. Whitley. Genetic algorithms and neural networks. In J. Periaux and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science*. John Wiley and Sons, 1995.