12-2015

Neural Decomposition of Time-Series Data for Effective Generalization

Luke Godfrey
University of Arkansas, Fayetteville

Neural Decomposition of Time-Series Data
for Effective Generalization

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Luke B. Godfrey
University of Arkansas
Bachelor of Science in Computer Science, 2014

December 2015
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____

Dr. Michael S. Gashler
Thesis Director

_____

Dr. Wing Ning Li
Committee Member

_____

Dr. Xintao Wu
Committee Member

Abstract

We present a neural network technique for the analysis and extrapolation of time-series data called Neural Decomposition (ND). Units with a sinusoidal activation function are used to perform a Fourier-like decomposition of training samples into a sum of sinusoids, augmented by units with nonperiodic activation functions to capture linear trends and other nonperiodic components. We show how careful weight initialization can be combined with regularization to form a simple model that generalizes well. Our method generalizes effectively on the Mackey-Glass series, a dataset of unemployment rates as reported by the U.S. Department of Labor Statistics, a time-series of monthly international airline passengers, the monthly ozone concentration in downtown Los Angeles, and an unevenly sampled time-series of oxygen isotope measurements from a cave in north India. We find that ND outperforms popular time-series forecasting techniques including ARIMA, SARIMA, SVR with a radial basis function, Gashler and Ashmore's model, and echo state networks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The analysis and forecasting of time-series is a challenging problem that continues to be an active area of research. Predictive techniques have been presented for an array of problems, including weather [15], traffic flow [24], seizures [12], sales [8], and others [34, 19, 7, 35]. Because research in this area can be so widely applied, there is great interest in discovering more accurate methods for time-series forecasting.

One approach for analyzing time-series data is to interpret it as a signal and apply the Fourier transform to decompose the data into a sum of sinusoids [2]. Unfortunately, despite the well-established utility of the Fourier transform, it cannot be applied directly to time-series forecasting. The Fourier transform uses a predetermined set of sinusoid frequencies rather than learning the frequencies that are actually expressed in the training data. Although the signal produced by the Fourier transform perfectly reproduces the training samples, it also predicts that the same pattern of samples will repeat indefinitely. As a result, the Fourier transform is effective at interpolation but is unable to extrapolate future values. Another limitation of the Fourier transform is that it only uses periodic components, and thus cannot accurately model the nonperiodic aspects of a signal, such as a linear trend or nonlinear abnormality.

Another approach is regression and extrapolation using a model such as a neural network. Regular feedforward neural networks with standard sigmoidal activation functions do not tend to perform well at this task because they cannot account for periodic components in the training data. Fourier neural networks have been proposed, in which feedforward neural networks are given sinusoidal activation functions and are initialized to compute the Fourier transform. Unfortunately, these models have proven to be difficult to train [15].

Recurrent neural networks, as opposed to feedforward neural networks, have been successfully applied to time-series prediction [16, 17]. However, these kinds of networks make up a different

class of forecasting techniques. Recurrent neural networks also have difficulty handling unevenly sampled time-series. Further discussion about recurrent neural networks and other classes of forecasting techniques is provided in Chapter 2.

This thesis claims that effective generalization can be achieved by regression and extrapolation using a model with two essential properties: (1) it must combine both periodic and nonperiodic components, and (2) it must be able to tune its components as well as the weights used to combine them. We present a neural network technique called Neural Decomposition (ND) that demonstrates this claim. Like the Fourier transform, it decomposes a signal into a sum of constituent parts. Unlike the Fourier transform, however, ND is able to reconstruct a signal that is useful for extrapolating beyond the training samples. ND trains the components into which it decomposes the signal represented by training samples. This enables it to find a simpler set of constituent signals. In contrast to the fast Fourier transform, ND does not require the number of samples to be a power of two, nor does it require that samples be measured at regular intervals. Additionally, ND facilitates the inclusion of nonperiodic components, such as linear or sigmoidal components, to account for trends and nonlinear irregularities in a signal.

In Chapter 5, we demonstrate that the simple innovations of ND work together to produce significantly improved generalizing accuracy with several problems. We tested with the chaotic Mackey-Glass series, a dataset of unemployment rates as reported by the U.S. Department of Labor Statistics, a time-series of monthly international airline passengers, the monthly ozone concentration in downtown Los Angeles, and an unevenly sampled time-series of oxygen isotope measurements from a cave in north India. We compared against an autoregressive integrated moving average (ARIMA) model, seasonal ARIMA (SARIMA), support vector regression with a radial basis function (SVR), a model recently proposed by Gashler and Ashmore [15], and echo state networks. In all but one case, ND made better predictions than each of the other prediction techniques evaluated; in the excepted case, echo state networks performed slightly better than ND.

This paper is outlined as follows. Chapter 2 provides a background and reviews related works. Chapter 3 gives an intuitive-level overview of ND. Chapter 4 provides finer implementation-level

details. Chapter 5 shows results that validate our work. Finally, Chapter 6 discusses the contributions of this paper and future work.

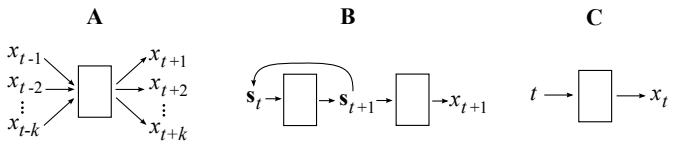

Figure 2.1: Three broad classes of models for time-series forecasting: (A) prediction using a sliding window, (B) recurrent models, and (C) regression-based extrapolation.

Chapter 2

Related Work

2.1 Models for Time-Series Prediction

Many works have diligently surveyed the existing literature regarding techniques for forecasting time-series data [10, 22, 38, 5, 13, 37, 9]. Some popular statistical models include Gaussian process [4] and hidden Markov models [26].

Autoregressive integrated moving average (ARIMA) models [39, 36] are among the most popular approaches. The notation for this model is ARIMA(p,d,q), where p is the number of terms in the autoregressive model, d is the number of differences required to take to make the time-series stationary, and q is the number of terms in the moving average model. In other words, ARIMA models compute the dth difference of $x(t)$ as a function of $x_{t-1}, x_{t-2}, \ldots, x_{t-p}$ and the previous q error terms.

Out of all the ARIMA variations that have been proposed, seasonal ARIMA (SARIMA) [3] is considered to be the state of the art "classical" time-series approach [24]. Notation for SARIMA is ARIMA$(p,d,q)(P,D,Q)[S]$, where p,d,q are identical to the normal ARIMA model, P,D,Q are analogous seasonal values, and S is the seasonal parameter. For example, an ARIMA(1,0,1)(0,1,1)[12] uses an autoregressive model with one term, a moving average model with one term, one seasonal difference (that is, $x'_t = x_t - x_{t-12}$), and a seasonal moving average with one term. This seasonal variation of ARIMA exploits seasonality in data by correlating x_t not only with recent observations

like x_{t-1}, but also with seasonally recent observations like x_{t-S}. For example, when the data is a monthly time-series, $S = 12$ correlates observations made in the same month of different years, and when the data is a daily time-series, $S = 7$ correlates observations made on the same day of different weeks.

In the field of machine learning, three high-level classes of techniques (illustrated in Figure 2.1) are commonly used to forecast time-series data [15]. Perhaps the most common approach, (A), is to train a model to directly forecast future samples based on a sliding window of recently collected samples [13]. This approach is popular because it is simple to implement and can work with arbitrary supervised learning techniques.

A more sophisticated approach, (B), is to train a recurrent neural network [21, 28]. Several recurrent models, such as LSTM networks [16, 17], have reported very good results for forecasting time-series. Echo state networks (ESNs) have performed particularly well at this task [23, 20, 30]. An ESN is a randomly connected, recurrent reservoir network with three primary meta-parameters: input scaling, spectral radius, and leaking rate [25]. Although they are powerful, these recurrent models are only able to handle time-series that are sampled at a fixed interval, and thus cannot be directly applied to unevenly sampled time-series.

Our model falls into the third category of machine learning techniques, (C): regression-based extrapolation. Models of this type fit a curve to the training data, then use the trained curve to anticipate future samples. One advantage of this approach over recurrent neural networks is that it can make continuous predictions, instead of predicting only at regular intervals, and can therefore be directly applied to irregularly spaced time-series. A popular method in this category is support vector regression (SVR) [33, 11]. Many models in this category decompose a signal into constituent parts, providing a useful mechanism for analyzing the signal. Our model is more closely related to a subclass of methods in this category, called Fourier neural networks (see Section 2.3), due to its use of sinusoidal activation functions. Models in the first two categories, (A) and (B), have already been well-studied, whereas extrapolation with sinusoidal neural networks remains a relatively unexplored area.

2.2 Inverse Discrete Fourier Transform

The discrete Fourier transform (DFT) maps a series of N complex numbers in the time domain to the frequency domain. The inverse DFT (iDFT) can be applied these new values to map them back to the time domain. More interestingly, the iDFT can be used as a continuous representation of the originally discrete input. The transforms are generally written as a sum of N complex exponentials, which can be rewritten in terms of sines and cosines by Euler's formula.

The DFT and the iDFT are effectively the same transform with two key differences. First, in terms of sinusoids, the DFT uses negative multiples of $2\pi/N$ as frequencies and the iDFT uses positive multiples of $2\pi/N$ as frequencies. Second, the iDFT contains the normalization term $1/N$ applied to each sum.

In general, the iDFT requires all N complex values from the frequency domain to reconstruct the input series. For real-valued input, however, only the first $N/2+1$ complex values are necessary ($N/2$ frequencies and one bias). The remaining complex numbers are the conjugates of the first half of the values, so they only contain redundant information. Furthermore, in the real-valued case, the imaginary component of the iDFT output can be discarded to simplify the equation, as we do in Equation 2.1. This particular form of the iDFT (reconstructing a series of real samples) can therefore be written as a real sum of sines and cosines.

The iDFT is as follows. Let R_k and I_k represent the real and imaginary components respectively of the kth complex number returned by the DFT. Let $2\pi k/N$ be the frequency of the kth term. The first frequency yields the bias, because $cos(0) = 1$ and $sin(0) = 0$. The second frequency is a single wave, the third frequency is two waves, the fourth frequency is three waves, and so on. The cosine with the kth frequency is scaled by R_k, and the sine with the kth frequency is scaled by I_k. Thus, the iDFT is sufficiently described as a sum of $N/2+1$ terms, with a $sin(t)$ and a $cos(t)$ in each term and a complex number from the DFT corresponding to each term:

$$x(t) = \sum_{k=0}^{N/2} R_k \cdot cos(\frac{2\pi k}{N}t) - I_k \cdot sin(\frac{2\pi k}{N}t) \tag{2.1}$$

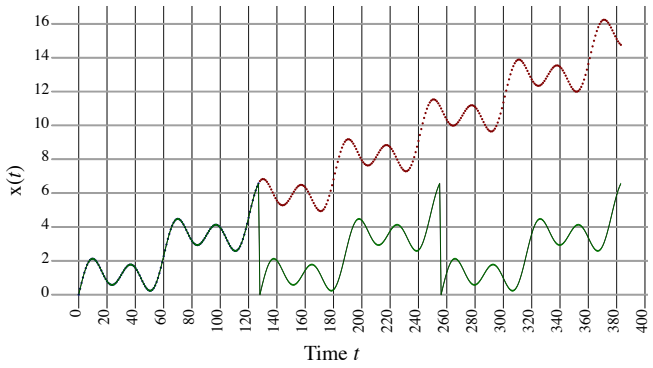

Figure 2.2: The predictive model generated by the iDFT for a toy problem with both periodic and nonperiodic components. Blue dots represent training samples, red dots represent testing samples, and the green line represents the iDFT. Two significant problems limit its ability to generalize: (1) The model repeats, ignoring the linear trend, and (2) The extrapolated predictions misalign with the phase of the continuing nonlinear trend.

Equation 2.1 is useful as a continuous representation of the real-valued discrete input. Because it perfectly passes through the input samples, one might naively expect this function to be a good basis for generalization. In order to choose appropriate frequencies, however, the iDFT assumes that the underlying function always has a period equal to the size of the samples that represent it, that is, $x(t+N) = x(t)$ for all t. Typically, in cases where generalization is desirable, the period of the underlying function is not known. The iDFT cannot effectively model the nonperiodic components of a signal, nor can it form a simple model for series that are not periodic at N, even if the series is perfectly periodic.

Figure 2.2 illustrates the problems encountered when using the iDFT for time-series forecasting. Although the model generated by the iDFT perfectly fits the training samples, it only has periodic components and so is only able to predict that these samples will repeat to infinity, without taking nonperiodicity into account. Our approach mimics the iDFT for modeling periodic data, but is also able to account for nonperiodic components in a signal (Figure 3.2).

2.3 Fourier Neural Networks

Use of the Fourier transform in neural networks has already been explored in various contexts. The term *Fourier neural network* has been used to refer to neural networks that use a Fourier-like

neuron [31], that use the Fourier transform of some data as input [27], or that use the Fourier transform of some data as weights [15]. Our work is not technically a Fourier neural network, but of these three types, our approach most closely resembles the third.

Silvescu provided a model for a Fourier-like activation function for neurons in neural networks [31]. His model utilizes every unit to form DFT-like output for its inputs. He notes that by using gradient descent to train sinusoid frequencies, the network is able to learn "exact frequency information" as opposed to the "statistical information" provided by the DFT. Our approach also trains the frequencies of neurons with a sinusoidal activation function.

Gashler and Ashmore presented a technique that used the fast Fourier transform (FFT) to approximate the DFT, then used the obtained values to initialize the sinusoid weights of a neural network that mixed sinusoidal, linear, and softplus activation functions [15]. Because this initialization used sinusoid units to model nonperiodic components of the data, their model was designed to heavily regularize sinusoid weights so that as the network was trained, it gave preference to weights associated with nonperiodic units and shifted the weights from the sinusoid units to the linear and softplus units. Use of the FFT required their input size to be a power of two, and their trained models were slightly out of phase with their validation data. However, they were able to generalize well for certain problems. Our approach is similar, except that we do not use the Fourier transform to initialize any weights (further discussion on why we do not use the Fourier transform can be found in Section 3.4).

Chapter 3

High Level Approach

In this chapter, we describe Neural Decomposition (ND), a neural network technique for the analysis and extrapolation of time-series data. This chapter focuses on an intuitive-level overview of our method; implementation details can be found in Chapter 4.

3.1 Algorithm Description

We use an iDFT-like model with two simple but important innovations. First, we allow sinusoid frequencies to be trained. Second, we augment the sinusoids with a nonperiodic function to model nonperiodic components. The iDFT-like use of sinusoids allows our model to fit to periodic data, the ability to train the frequencies allows our model to learn the true period of a signal, and the augmentation function enables our model to forecast time-series that are made up of both periodic and nonperiodic components.

Our model is defined as follows. Let each a_k represent an amplitude, each w_k represent a frequency, and each ϕ_k represent a phase shift. Let $g(t)$ be an augmentation function that represents the nonperiodic components of the signal.

$$x(t) = \sum_{k=1}^{N} \left(a_k \cdot sin(w_k t + \phi_k)\right) + g(t) \tag{3.1}$$

Note that the lower index of the sum has changed from $k = 0$ in the iDFT to $k = 1$ in our model. This is because ND can account for bias in the augmentation function $g(t)$, so the 0 frequency is not necessary. Therefore, only N sinusoids are required rather than $N + 2$.

If the phase shifts are set so that $sin(t + \phi)$ is transformed into $cos(t)$ and $-sin(t)$, the frequencies are set to the appropriate multiples of 2π, the amplitudes are set to the output values of the DFT, and $g(t)$ is set to a constant (the bias), then ND is identical to the iDFT. However, by choos-

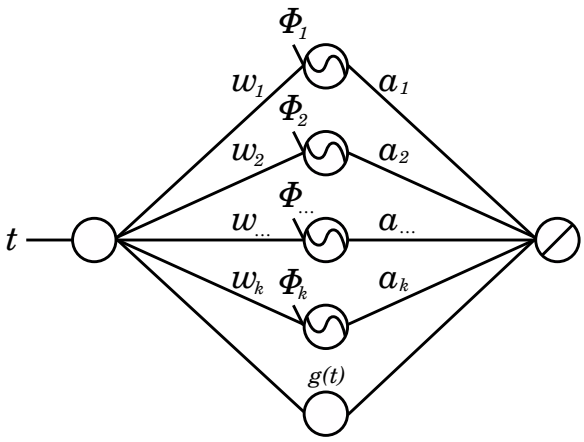

Figure 3.1: A diagram of the neural network model used by Neural Decomposition. For each of the k sinusoid units, w_i are frequencies, ϕ_i are phase shifts, and a_i are amplitudes, where $i \in \{1 \dots k\}$. The augmentation function $g(t)$ is shown as a single unit, but it may be composed of one or more units with one or more activation functions.

ing a $g(t)$ better suited to generalization and by learning the amplitudes and tuning the frequencies using backpropagation, our method is more effective at generalization than the iDFT. $g(t)$ may be as simple as a linear equation or as complex as a combination of linear and nonlinear equations. A discussion on the selection of $g(t)$ can be found in Chapter 4.

We use a feedforward artificial neural network with a single hidden layer to compute our function (see Figure 3.1). The hidden layer is composed of N units with a sinusoid activation function and an arbitrary number of units with other activation functions to calculate $g(t)$. The output layer is a single linear unit, so that the neural network outputs a linear combination of the units in the hidden layer.

We initialize the frequencies and phase shifts in the same way as the inverse DFT as described above. Rather than use the actual values provided by the DFT as sinusoid amplitudes, however, we initialize them to small random values (see Section 3.4 for a discussion on why). Weights in the hidden layer associated with $g(t)$ are initialized to approximate identity, and weights in the output layer associated with $g(t)$ are randomly perturbed from zero.

We train our model using stochastic gradient descent with backpropagation. This training process allows our model to learn better frequencies and phase shifts so that the sinusoid units more accurately represent the periodic components of the time-series. Because frequencies and

phase shifts are allowed to change, our model can learn the true period of the underlying function rather than assuming the period is N. Training also tunes the weights of the augmentation function.

ND uses regularization throughout the training process to distribute weights in a manner consistent with our goal of generalization. In particular, we use L^1 regularization on the output layer of the network to promote sparsity by driving nonessential weights to zero. Thus, ND produces a simpler model by using the fewest number of units that still fit the training data well.

By pre-initializing the frequencies and phase shifts to mimic the inverse DFT and setting all other parameters to small values, we reduce time-series prediction to a simple regression problem. Artificial neural networks are particularly well-suited to this kind of problem, and using stochastic gradient descent with backpropagation to train it should yield a precise and accurate model.

The neural network model and training approach we use is similar to those used by Gashler and Ashmore in a previous work on time-series analysis [15]. Our work builds on theirs and contributes a number of improvements, both theoretically and practically. First, we do not initialize the weights of the network using the Fourier transform. This proved to be problematic in their work as it used periodic components to model linear and other nonperiodic parts of the training data. By starting with weights near zero and learning weights for both periodic and nonperiodic units simultaneously, our model does not have to unlearn extraneous weights. Second, their model required heavy regularization that favored using linear units rather than the initialized sinusoid units. Our training process makes no assumptions about which units are more important and instead allows gradient descent to determine which components are necessary to model the data. Third, their training process required a small learning rate (on the order of 10^{-7}) and their network was one layer deeper than ours. As a result, their frequencies were never tuned, their results were generally out of phase with the testing data, and their training times were very long. Because our method facilitates the training of each frequency and allows a larger learning rate (10^{-3} in our experiments), our method yields a function that is more precisely in phase with the testing data in a much shorter amount of time. Thus, our method has simplified the complexity of the model's training algorithm, minimized its training time, and improved its overall effectiveness at time-series prediction. The

superiority of our method is demonstrated in Chapter 5 and visualized in Figure 5.1.

3.2 Comparison to iDFT

Neural Decomposition has a number of benefits over the iDFT for time-series prediction. One is that, unlike the FFT-approximated iDFT, ND does not require the number of samples to be a power of two. In order to use the FFT on any input size that is not a power of two, the input must be padded with zeros (or some other arbitrary placeholder) to make it a power of two in size. Although this is acceptable in some applications, it sabotages generalization by training a model to reconstruct these arbitrary values. The removal of a power of two restriction maximizes the amount of information that ND is able to effectively utilize.

Additionally, our approach does not make the generally false assumption that the input is periodic at N. The iDFT predicts that the input series will repeat itself indefinitely and cannot handle fractional periods. ND uses flexible frequencies that enable it to learn the actual period of the underlying function, even if the input series contains a fractional part of a signal's periodic components. Our method effectively harnesses the information provided by the entire input series, including the fractional part. The iDFT, however, is not able to use this extra information. In fact, the fractional part introduces unnecessary complexity into the model generated by the iDFT.

A third advantage ND has over the iDFT is that ND does not require samples taken at regular intervals. Although many real-world datasets are sampled regularly, there are a number of applications that are not. Any time-series data obtained from a mobile device, for example, may contain irregular samples due to power consumption or loss of signal [1].

The iDFT can be used to model time-series as a sum of sinusoids, which is ideal for periodic data. To model any nonperiodic components of a time-series, however, the iDFT has to use several sinusoid units, resulting in an unnecessarily complicated function representing the closed form of the series. ND is able to account for nonperiodic components of a signal using nonperiodic functions, resulting in a simpler model.

The most important benefit of our approach is that it is able to generalize. The iDFT yields

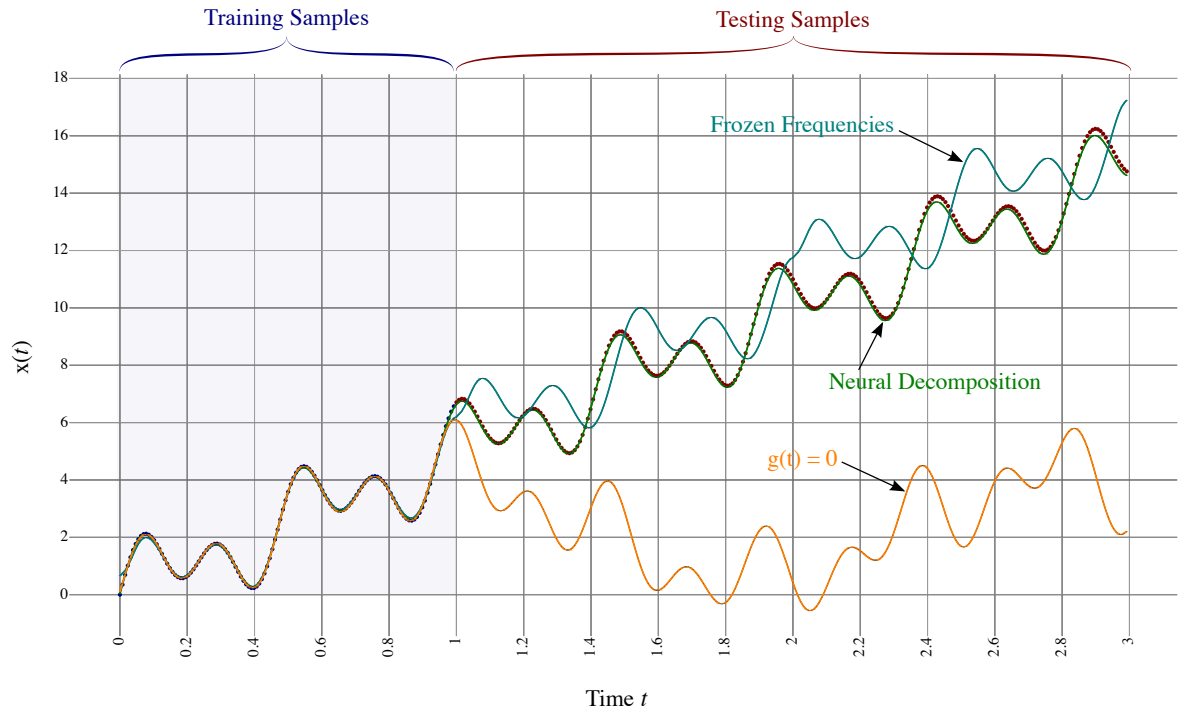

Figure 3.2: A comparison of Neural Decomposition with two algorithmic variations showing the importance of certain algorithm details. The data used here is the same data used in Figure 2.2. The full ND model, shown in green, fits very closely to the data that was withheld during training. The cyan curve shows predictions made when the basis functions, including sinusoidal frequencies, were frozen during training. Note that the predictions are out-of-phase, indicating that training these components is essential for effective generalization. The orange curve shows predictions made without including any nonperiodic components among the basis functions, that is, setting the augmentation function $g(t) = 0$. Although the predictions exhibit the correct phase, they fail to fit with the nonperiodic trend. This shows the importance of using heterogeneous basis functions.

a model that perfectly fits the input samples, but it generalizes poorly for nonperiodic data, or for data that is periodic at a point other than at N. Because it has flexible frequencies and can model nonperiodic components, ND can generalize for both periodic and nonperiodic time-series, regardless of where the periodicity is.

3.3 Toy Problem for Justification

Figure 3.2 demonstrates that flexible frequencies and an appropriate choice for $g(t)$ are essential for effective generalization. We compare three ND models using the equation $x(t) = sin(4.25\pi t) + sin(8.5\pi t) + 5t$ to generate time-series data. This is a sufficiently interesting toy problem because

it is composed of periodic and nonperiodic functions and its period is not exactly N (otherwise, the frequencies would have been multiples of 2π). We generate 128 values for $0 \leq t < 1.0$ as input and 256 values for $1.0 \leq t < 3.0$ as a validation set. Powers of two are not required, but we used powers of two in order to compare our approach with using the inverse DFT (approximated by the inverse FFT).

One of the compared ND models freezes the frequencies so that the model is unable to adjust them. Although it is able to find the linear trend in the signal, it is unable to learn the true period of the data and, as a result, makes predictions that are out of phase with the actual signal. This demonstrates that the ability to adjust the constituent parts of the output signal is necessary for effective generalization.

Another of the compared ND models has flexible frequencies, but uses no augmentation function (that is, $g(t) = 0$). This model can learn the periodic components of the signal, but not its nonperiodic trend. It tunes the frequencies of the sinusoid units to more accurately reflect the input samples, so that it is more in phase than the second model. However, because it cannot explain the nonperiodic trend of the signal, it also uses more sinusoid units than the true underlying function requires, resulting in predictions that are not perfectly in phase. This model shows the necessity of an appropriate augmentation function for handling nonperiodicity.

The final ND model compared in Figure 3.2 is ND with flexible frequencies and augmentation function $g(t) = wt + b$. As expected, it learns both the true period and the nonperiodic trend of the signal. We therefore conclude that an appropriate augmentation function and the ability to tune components are essential in order for ND to generalize well.

3.4 Toy Problem Analysis

In Figure 3.3, we plot the weights over time of our $g(t) = wt + b$ model being trained on the toy problem. Weights in Figure 3.3(a) are the frequencies of a few of the sinusoids in the model, initialized based on the iDFT, but tuned over time to learn more appropriate frequencies for the input samples, and weights in Figure 3.3(b) are their corresponding amplitudes. The training pro-

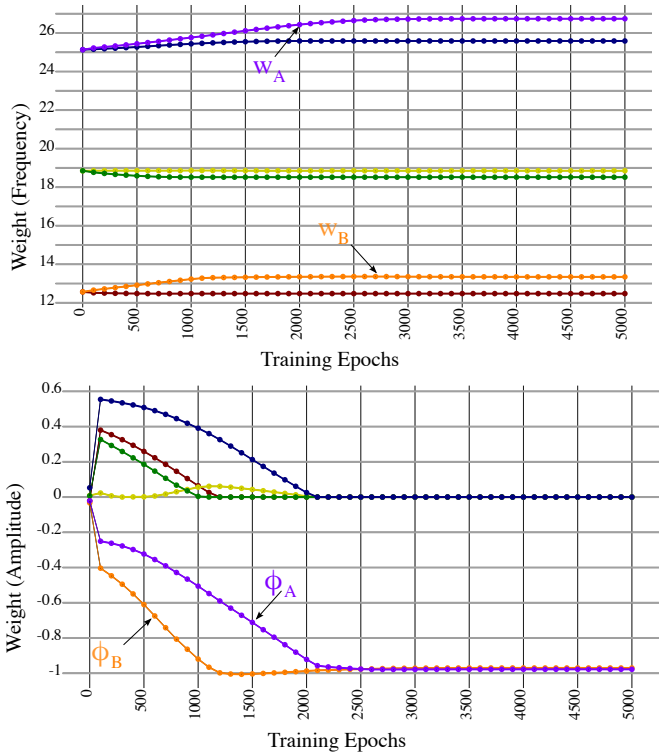

Figure 3.3: (Left) Frequencies of the basis functions of Neural Decomposition over time. (Right) Basis weights (amplitudes) over time on the same problem. Note that ND first tunes the frequencies (Left), then finishes adjusting the corresponding amplitudes for those sinusoids (Right) (w_A corresponds to ϕ_A and w_B corresponds to ϕ_B). In most cases, the amplitudes are driven to zero to form a sparse representation. After the amplitudes reach zero, the frequencies are no longer modified.

cess tunes frequencies w_A and w_B to more accurately reflect the period of the underlying function and adjusts the corresponding amplitudes ϕ_A and ϕ_B so that only the sinusoids associated with these amplitudes are used in the trained model and all other amplitudes are driven to zero. This demonstrates that ND tunes frequencies it needs and learns amplitudes as we hypothesized. It is also worth noting that after the first 2500 training epochs, no further adjustments are made to the weights. This suggests that ND is robust against overfitting, at least in some cases, as the "extra" training epochs did not result in a worse prediction.

Gashler and Ashmore utilized the FFT to initialize the sinusoid amplitudes so that the neural network immediately resembled the iDFT [15]. Using the DFT in this way yields an unnecessarily complex model in which nearly every sinusoid unit has a nonzero amplitude, either because it uses

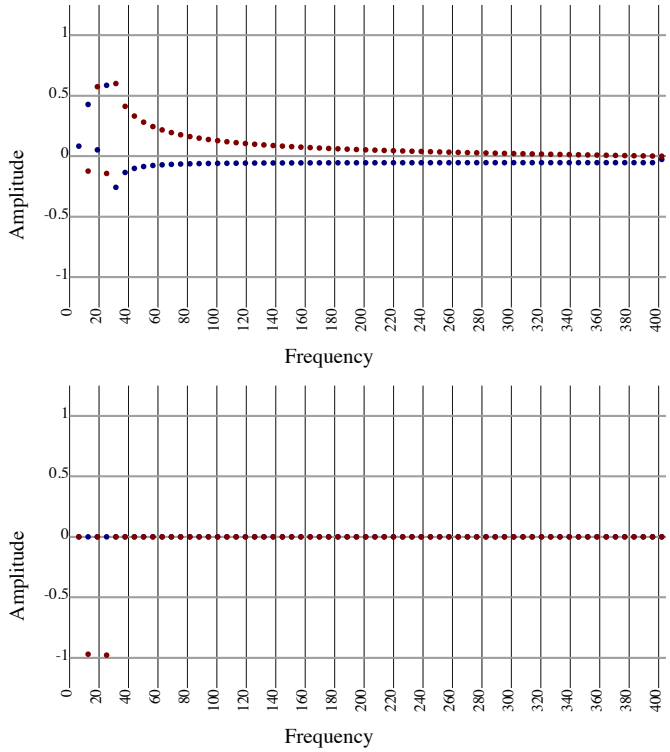

Figure 3.4: Frequency domain representations of the toy problem (amplitude vs frequency). (Left) Frequencies used by the iDFT. (Right) Frequencies used by ND.

periodic functions to model the nonperiodic signal or because it has fixed frequencies and so uses a range of frequencies to model the actual frequencies in the signal [31]. Consequently, the training process required heavy regularization of the sinusoid amplitudes in order to shift the weight to the simpler units (see Section 2.3). Training from this initial point often fell into local optima, as such a model was not always able to unlearn superfluous sinusoid amplitudes.

Figure 3.4 demonstrates why using amplitudes provided by the Fourier transform is a poor initialization point. The actual underlying function only requires two sinusoid units (found by ND), but the Fourier transform uses every sinusoid unit available to model the linear trend in the toy problem. Instead of tuning two amplitudes, a model initialized with the Fourier transform has to tune every amplitude and is therefore far more likely to fall into local optima.

ND, by contrast, does not use the FFT. Sinusoid amplitudes (the weights feeding into the output layer) and all output-layer weights associated with $g(t)$ are initialized to small random values. This allows the neural network to learn the periodic and nonperiodic components of the signal

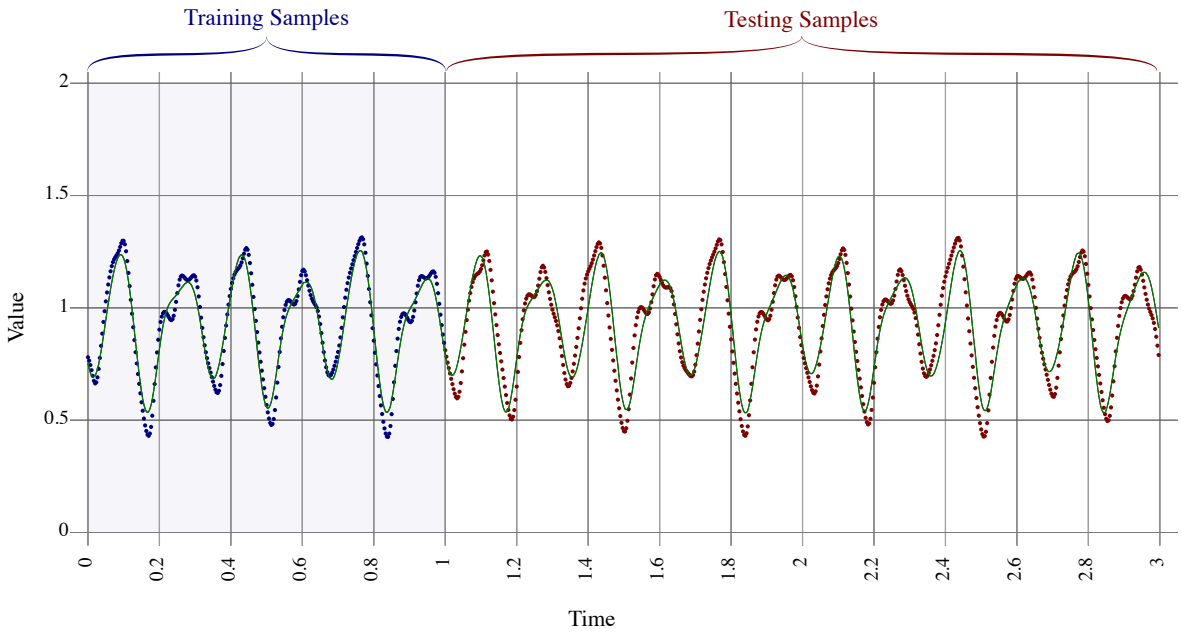

Figure 3.5: Neural Decomposition on the Mackey-Glass series. Although it does not capture all the high-frequency fluctuations in the data, our model predicts the location and height of each peak and valley in the series with a high degree of accuracy.

simultaneously. Without the hindrance of having to unlearn part of the DFT, the training process is better able to find near-optimal values for these weights. Figure 3.4 shows a comparison of our trained model with the frequencies used by the iDFT, omitting the linear component learned by ND.

3.5 Mackey-Glass Series for Justification

In addition to the toy problem, we applied ND to the Mackey-Glass series as a proof-of-concept. This series is known to be chaotic rather than periodic, so it is an interesting test for our approach that decomposes the signal as a combination of sinusoids. Results with this data are shown in Figure 3.5. The blue points on the left represent the training sequence, and the red points on the right half represent the testing sequence. All testing samples were withheld from the model, and are only shown here to illustrate the effectiveness of the model in anticipating future samples. The green curve represents the predictions of the trained model. Although it does not capture all the

high-frequency fluctuations in the Mackey-Glass series, it clearly exhibits shapes similar to those in the test set. Interestingly, neither the shapes in the test data nor those exhibited within the model are strictly repeating. This occurs because the frequencies of the sinusoidal basis functions that ND uses to represent its model may be tuned to have frequencies with no small common multiple, thus creating a signal that does not repeat for a very long time. Most notably, our model predicts the location and height of each peak and valley in the series with a high degree of accuracy. This demonstrates that ND can be effective for predicting chaotic series.

Chapter 4

Implementation Details

In this chapter, we provide a more detailed explanation of our approach. A high level description of Neural Decomposition can be found in Chapter 3. For convenience, an implementation of Neural Decomposition is included in the Waffles machine learning toolkit [14].

4.1 Topology

We use a feedforward artificial neural network as the basis of our model. For an input of size N, the neural network is initialized with two layers: $1 \rightarrow m$ and $m \rightarrow 1$, where $m = N + |g(t)|$ and $|g(t)|$ denotes the number of nodes required by $g(t)$. The first N nodes in the hidden layer have the sinusoid activation function, $sin(t)$, and the rest of the nodes in the hidden layer have other activation functions to compute $g(t)$.

The augmentation function $g(t)$ can be made up of any number of nodes with one or more activation functions. For example, it could be made up of linear units for learning trends and sigmoidal units to fit nonperiodic, nonlinear irregularities. Gashler and Ashmore have suggested that softplus units may yield better generalizing predictions compared to standard sigmoidal units [15]. In our experiments, we used a combination of linear, softplus, and sigmoidal nodes for $g(t)$. The network tended to only use a single linear node, which may suggest that the primary benefit of the augmentation function is that it can model linear trends in the data. Softplus and sigmoidal units tended to be used very little or not at all by the network in the problems we tested, but intuitively it seems that nonlinear activation functions could be useful in some cases.

4.2 Weight Initialization

The weights of the neural network are initialized as follows. Let each of the N sinusoid nodes in the hidden layer, indexed as k for $0 \leq k < N/2$, have a weight w_k and bias ϕ_k. Let each w_k represent a frequency and be initialized to $2\pi\lfloor k/2 \rfloor$. Let each ϕ_k represent a phase shift. For each even value of k, let ϕ_k be set to $\pi/2$ to transform $sin(t+\phi_k)$ to $cos(t)$. For each odd value of k, let ϕ_k be set to π to transform $sin(t+\phi_k)$ to $-sin(t)$. A careful comparison of these initialized weights with Equation 2.1 shows that these are identical to the frequencies and phase shifts used by the iDFT, except for a missing $1/N$ term in each frequency, which is absorbed in the input preprocessing step (see Section 4.3).

All weights feeding into the output unit are set to small random values. At the beginning of training, therefore, the model will predict something like a flat line centered at zero. As training progresses, the neural network will learn how to combine the hidden layer units to fit the training data.

Weights in the hidden layer associated with the augmentation function are initialized to approximate the identity function. For example, in $g(t) = wt + b$, w is randomly perturbed from 1 and b is randomly perturbed near 0. Because the output layer will learn how to use each unit in the hidden layer, it is important that each unit be initialized in this way.

4.3 Input Preprocessing

Before training begins, we preprocess the input data to facilitate learning and prevent the model from falling into a local optimum. First, we normalize the time associated with each sample so that the training data lies between 0 (inclusive) and 1 (exclusive) on the time axis. If there is no explicit time, equally spaced values between 0 and 1 are assigned to each sample in order. Predicted data points will have a time value greater than or equal to 1 by this new scale. Second, we normalize the values of each input sample so that all training data is between 0 and 10 on the y axis.

This preprocessing step serves two purposes. First, it absorbs the $1/N$ term in the frequencies

by transforming t into t/N, which is why we were able to omit the $1/N$ term from our frequencies in the weight initialization step. Second, and more importantly, it ensures that the data is appropriately scaled so that the neural network can learn efficiently. If the data is scaled too large on either axis, training will be slow and susceptible to local optima. If the data is scaled too small, on the other hand, the learning rate of the machine will cause training to diverge and only use linear units and low frequency sinusoids.

In some cases, it is appropriate to pass the input data through a filter. For example, financial time-series data is commonly passed through a logarithmic filter before being presented for training, and outputs from the model can then be exponentiated to obtain predictions. We use this input preprocessing method in two of our experiments where we observe an underlying exponential growth in the training data.

4.4 Regularization

Regularization is essential to the training process. Prior to each sample presentation, we apply regularization on the output layer of the neural network. Even though we do not initialize sinusoid amplitudes using the DFT, the network is quickly able to learn how to use the initialized frequencies to perfectly fit the input samples. Without regularizing the output layer, training halts as soon as the model fits the input samples, because the measurable error is near zero. By relaxing the learned weights, regularization allows our model to redistribute its weight over time. We find that regularization amount is especially important; too much prevented our model from learning, but too little caused our model to fall into local optima. In our experiments, setting the regularization term to 10^{-2} avoided both of these potential pitfalls.

Another important function of regularization in ND is to promote sparsity in the network, so that the redistribution of weight produces as simple a model as the input samples allow. We use L^1 regularization for this reason. Usually, the trained model does not require all N sinusoid nodes in order to generalize well, and this type of regularization enables the network to automatically discard unnecessary nodes by driving their amplitudes to zero. L^2 regularization is not an accept-

able substitute in this case, as it would distribute the weights evenly throughout the network and could, like the DFT, try to use several sinusoid nodes to model what would more appropriately be modeled by a single node with a nonperiodic activation function.

It is worth noting that we only apply regularization to the output layer of the neural network. Any regularization that might occur in the hidden layer would adjust sinusoid frequencies before the output layer could learn sinusoid amplitudes. By allowing weights in the hidden layer to change without regularization, the network has the capacity to adjust frequencies but is not required to do so.

Backpropagation with stochastic gradient descent tunes the weights of the network and accomplishes the redistribution of weights that regularization makes possible. In our experiments, we use a learning rate of 10^{-3}.

Chapter 5

Validation

In this chapter, we report results that validate the effectiveness of Neural Decomposition. In each of these experiments, we used an ND model with an augmentation function made up of ten linear units, ten softplus units, and ten sigmoidal units. It is worth noting that $g(t)$ is under no constraint to consist only of these units; it could include other activation functions or only contain a single linear node to capture trend information. We use a regularization term of 10^{-2} and a learning rate of 10^{-3} in every experiment to demonstrate the robustness of our approach; we did not tune these meta-parameters for each experiment.

In our experiments, we compare ND with ARIMA, SARIMA, SVR, Gashler and Ashmore's model [15], and ESN. We used the R language implementation for ARIMA, SARIMA, and SVR [29]. For the ARIMA models, we used a variation of the `auto.arima` method that performs a grid-search to find the best parameters. For SVR, we used the `tune.svm` method, which also performs a grid-search. We used Lukoševičius' implementation of ESN [25] and implemented a grid-search to find the best parameters. Although these methods select the best models based on the amount of error calculated using the training samples, the grid-search is a very slow process. Gashler and Ashmore's model did not require a grid-search for parameters because it has a default set of parameters that are automatically tuned during the training process. With ND, no problem-specific parameter tuning was performed.

In each figure, the blue points in the shaded region represent training samples and the red points represent withheld testing samples. The curves on the graph represent the predictions made by the three models that made the most accurate predictions (only two models are shown in the fourth experiment because only two models could be applied to an irregularly sampled time-series). The actual error for each model's prediction is reported for all experiments and all models in Table 5.1 and Table 5.2.

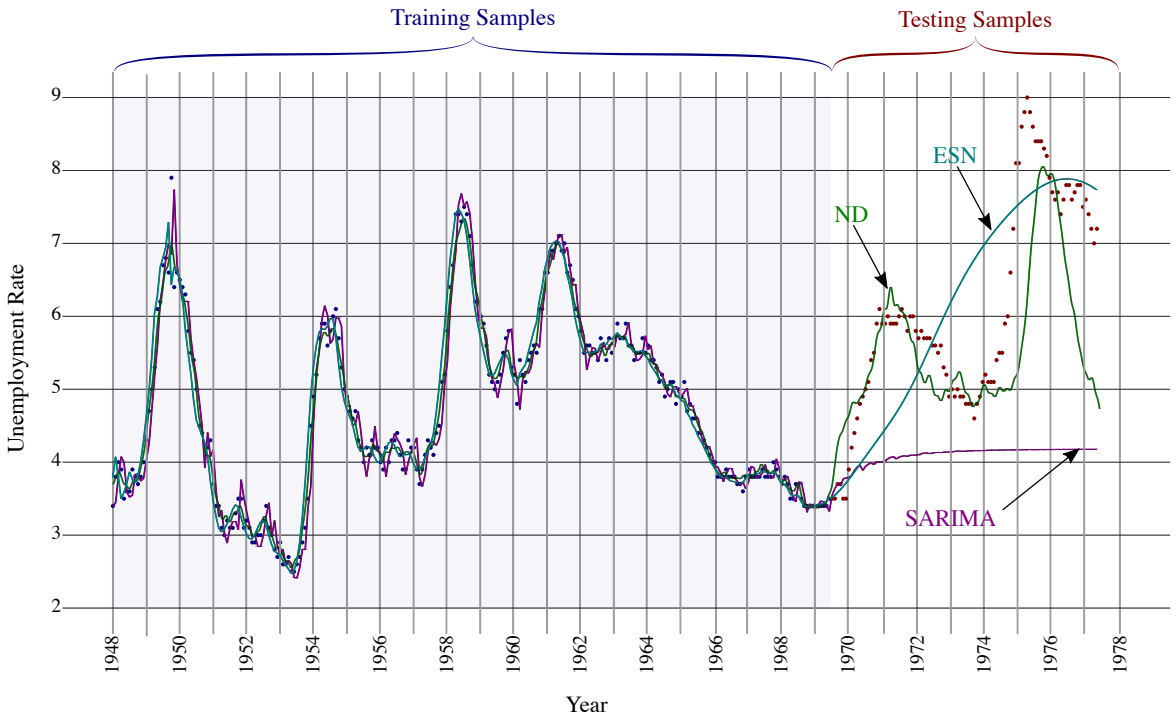

Figure 5.1: A comparison of the three best predictive models on the monthly unemployment rate in the US. Blue points represent the 258 training samples from January 1948 to June 1969 and red points represent the 96 testing samples from July 1969 to December 1977. SARIMA, shown in magenta, correctly predicted a rise in unemployment, but underestimated its magnitude, and did not predict the shape of the data well. ESN, shown in cyan, predicted a reasonable mean, but did not capture the dynamics of the data. Only ND, shown in green, successfully predicted both the depth and approximate shape of the surge in unemployment, followed by another surge in unemployment that followed.

In our first experiment, we demonstrated the effectiveness of ND on real-world data compared to widely used techniques in time-series analysis and forecasting. We trained our model on the unemployment rate from 1948 to 1969 as reported by the U.S. Bureau of Labor Statistics, and predicted the unemployment rate from 1969 to 1977. These results are shown in Figure 5.1. Blue points on the left represent the 258 training samples from January 1948 to June 1969, and red points on the right represent the 96 testing samples from July 1969 to December 1977. The three curves represent predictions made by ND (green), ESN (cyan), and SARIMA (magenta); ARIMA, SVR, and Gashler and Ashmore's model yielded poorer predictions and are therefore omitted from the figure. Grid-search found ARIMA(3,1,2) and ARIMA(1,1,2)(1,0,1)[12] for the

ARIMA and SARIMA models, respectively. ARIMA, not shown, did not predict the significant rise in unemployment. SARIMA, shown in magenta, did correctly predict a rise in unemployment, but underestimated its magnitude, and did not predict the shape of the data well. SVR, not shown, correctly predicted that unemployment would rise, then fall again. However, it also underestimated its magnitude. Gashler and Ashmore's model, not shown, predicted the rise and fall in unemployment, but underestimated its magnitude and the model's predictions significantly diverge from the subsequent testing samples. It is also worth noting that Gashler and Ashmore's model took about 200 seconds to train compared to ND, which took about 30 seconds to train. ESN, shown in cyan, predicted a reasonable mean value for the general increase in unemployment, but failed to capture the dynamics of the actual data. Results with Neural Decomposition (ND) are shown in green. ND successfully predicted both the depth and approximate shape of the surge in unemployment. Furthermore, it correctly anticipated another surge in unemployment that followed. ND did a visibly better job of predicting the nonlinear trend much farther into the future.

Our second experiment demonstrates the versatility of Neural Decomposition by applying to another real-world dataset: monthly totals of international airline passengers as reported by Chatfield [6]. We use the first six years of data (72 samples) from January 1949 to December 1954 as training data, and the remaining six years of data (72 samples) from January 1955 to December 1960 as testing data. The training data is preprocessed through a $log(x)$ filter and the outputs are exponentiated to obtain the final predictions. As in the first experiment, we compare our model with ARIMA, SARIMA, SVR, the model proposed by Gashler and Ashmore, and ESN. The predictions of the three most accurate models (ND, ESN, and SARIMA) are shown in Figure 5.2; ARIMA, SVR, and Gashler and Ashmore's model yielded poorer predictions and are therefore omitted from the figure. SVR, not shown, predicts a flat line after the first few time steps and generalizes the worst out of the four predictive models. The ARIMA model found by grid-search was ARIMA(2,1,3). ARIMA, not shown, was able to learn the trend, but failed to capture any of the dynamics of the signal. Grid-search found ARIMA(1,0,0)(1,1,0)[12] for the SARIMA model. Both SARIMA (shown in magenta) and ND (shown in green) are able to accurately predict the

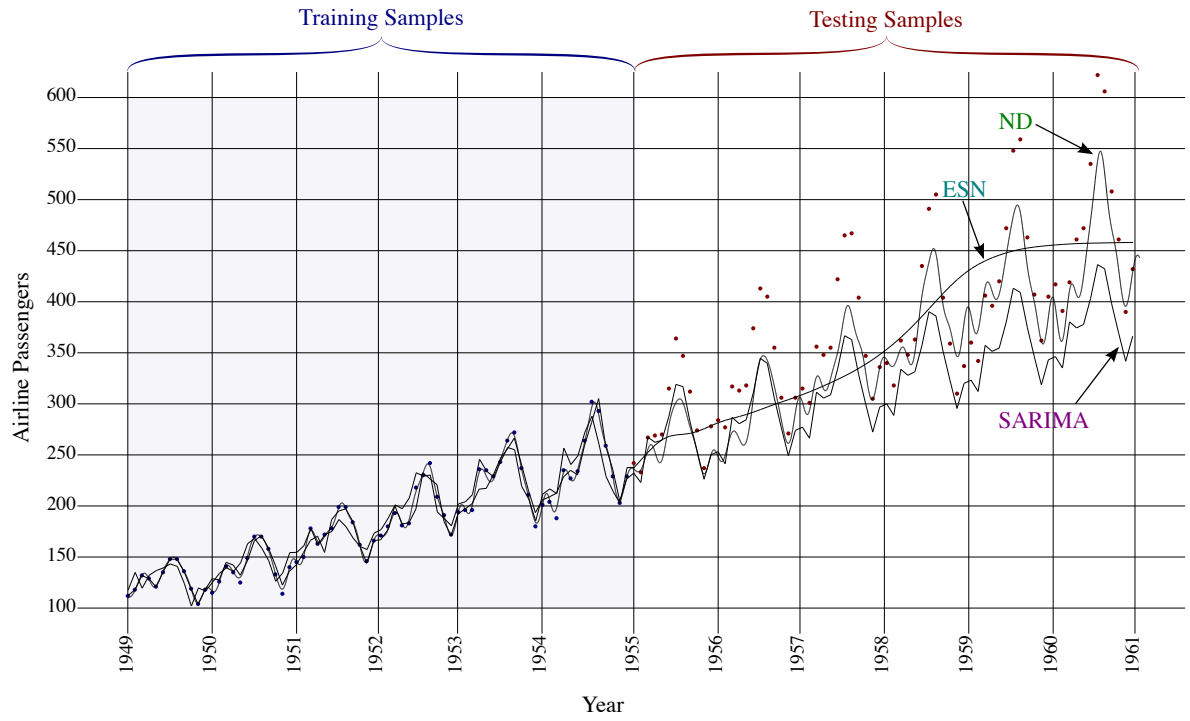

Figure 5.2: A comparison of the three best predictive models on monthly totals of international airline passengers from January 1949 to December 1960 [6]. Blue points represent the 72 training samples from January 1949 to December 1954 and red points represent the 72 testing samples from January 1955 to December 1960. SARIMA, shown in magenta, learns the trend and general shape of the data. ESN, shown in cyan, predicts a mean but does not capture the dynamics of the actual data. ND, shown in green, learns the trend, shape, and growth better than the other compared models.

shape of the future signal, but ND performs better. Unlike SARIMA, ND learns that the periodic component gets bigger over time. Gashler and Ashmore's model makes meaningful predictions for a few time steps, but appears to diverge after the first predicted season. ESN, shown in cyan, performs similarly to the ARIMA model, only predicting the trend and failing to capture seasonal variations.

The third experiment uses the monthly ozone concentration in downtown Los Angeles as reported by Hipel [18]. Nine years of monthly ozone concentrations (152 samples) from January 1955 to December 1963 are used as training samples, and the remaining three years and eight months (44 samples) from January 1964 to August 1967 are used as testing samples. The training data, as in the second experiment, is preprocessed through a $log(x)$ filter and output is exponenti-

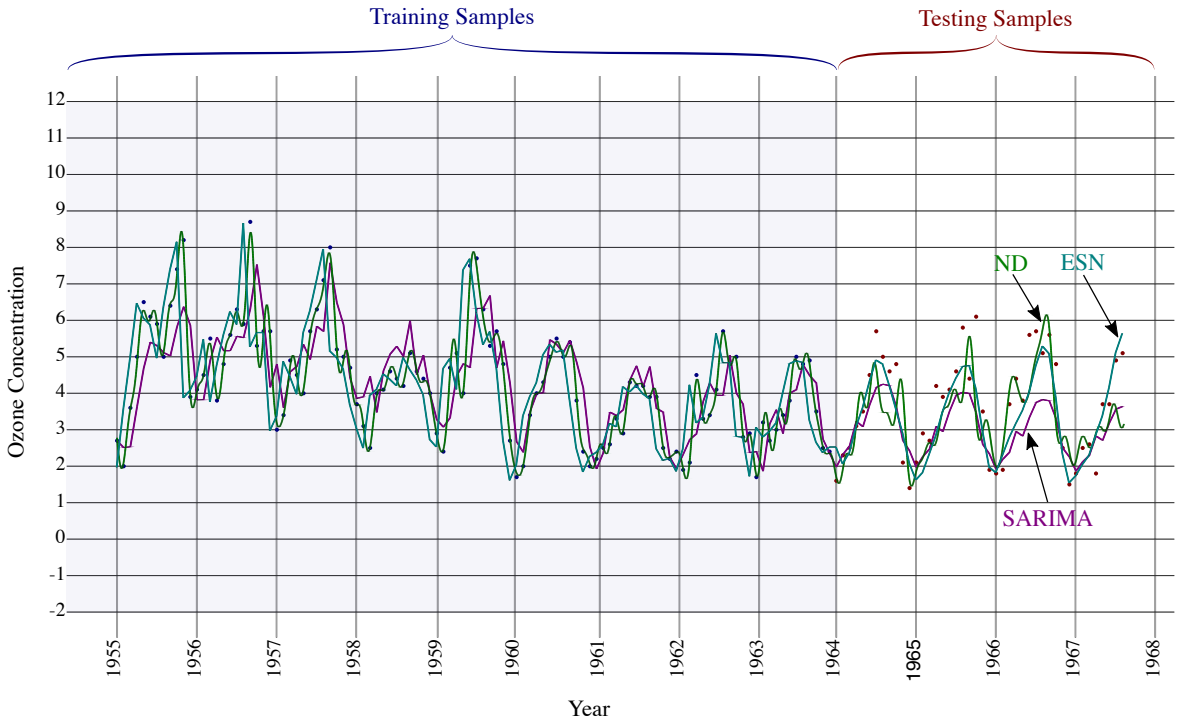

Figure 5.3: A comparison of the three best predictive models on monthly ozone concentration in downtown Los Angeles from January 1955 to August 1967 [18]. Blue points represent the 152 training samples from January 1955 to December 1963 and red points represent the 44 testing samples from January 1964 to August 1967. The compared models include SARIMA, ESN, and ND. All three of these models perform well on this problem. ESN's prediction, shown in cyan, has a smaller error than ND's prediction. ND's prediction, shown in green, has a smaller error than SARIMA's prediction (shown in magenta). ARIMA, SVR, and Gashler and Ashmore's model all performed poorly on this problem; rather than include them in this graph, their errors have been reported in Table 5.1 and Table 5.2.

ated to obtain the final predictions. Figure 5.3 compares the SARIMA, ESN, and ND models on this problem; ARIMA, SVR, and Gashler and Ashmore's model yielded poorer predictions and are therefore omitted from the figure. The ARIMA and SARIMA models found by grid-search were ARIMA(2,1,2) and ARIMA(1,1,1)(1,0,1)[12], respectively. ARIMA and SVR resulted in flat-line predictions with a high amount of error, and Gashler and Ashmore's model diverged in training and yielded unstable predictions. SARIMA (shown in magenta), ESN (shown in cyan), and ND (shown in green), on the other hand, all forecast future samples well. ESN yielded the most accurate predictions, and ND yielded the second most accurate predictions.

Our fourth experiment demonstrates that ND can be used on irregularly sampled time-series.

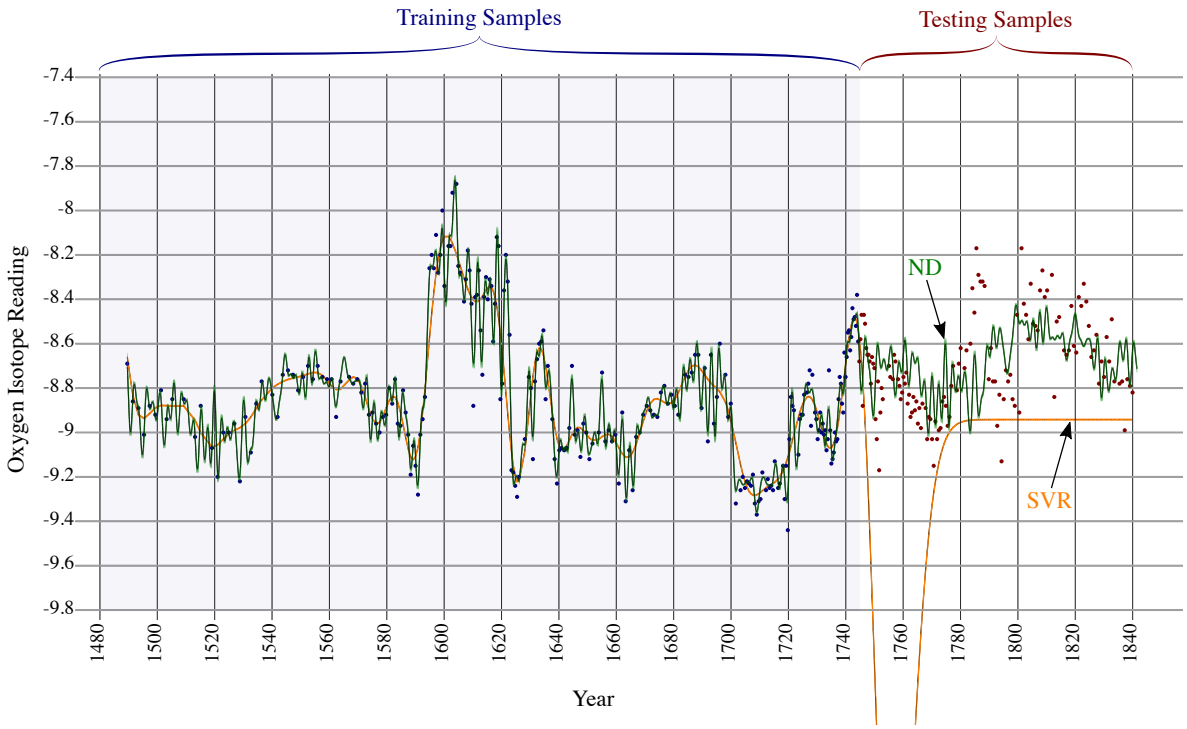

Figure 5.4: A comparison of two predictive models on a series of oxygen isotope readings in speleothems in India from 1489 AD to 1839 AD [32]. Blue points represent the 250 training samples from July 1489 to April 1744 and red points represent the 132 testing samples from August 1744 to December 1839. Because this time-series is irregularly sampled (the time step between samples is not constant), only SVR and ND could be applied to it. SVR, shown in orange, does not perform well, but predicts a steep drop in value that does not occur in the testing data, followed by a flat line. ND, shown in green, performs well, capturing the general shape of the testing samples.

We use a series of oxygen isotope readings in speleothems in a cave in India from 1489 AD to 1839 AD as reported by Sinha et. al [32]. Because the time intervals between adjacent samples is not constant (the interval is about 1.5 years on average, but fluctuates between 0.5 and 2.0 years), only ND and SVR models can be applied. ARIMA, SARIMA, Gashler and Ashmore's model, and ESN cannot be applied to irregular time-series because they assume a constant time interval between adjacent samples; these four models are therefore not included in this experiment. Figure 5.4 shows the predictions of ND and SVR. Blue points on the left represent the 250 training samples from July 1489 to April 1744, and red points on the right represent the 132 testing samples from August 1744 to December 1839. SVR, shown in orange, predicts a steep drop in value that does not exist in the testing data. ND, shown in green, accurately predicts the general shape of the

Table 5.1: Mean absolute percent error (MAPE) on the validation problems for ARIMA, SARIMA, SVR, Gashler and Ashmore, ESN, and ND. Best result (smallest error) for each problem is shown in **bold**.

Model	**Labor**	**Airline**	**Ozone**	**Speleothem**
ARIMA	39.42%	12.34%	39.50%	N/A
SARIMA	29.69%	13.33%	22.71%	N/A
SVR	25.14%	47.04%	49.53%	8.50%
Gashler/Ashmore	34.38%	19.89%	77.19%	N/A
ESN	15.73%	12.05%	**16.15%**	N/A
ND	**10.89%**	**9.52%**	21.59%	**1.89%**

testing data.

Table 5.1 presents an empirical evaluation of each model for the four real-world experiments. We use the mean absolute percent error (MAPE) as our error metric for comparisons [24]. MAPE for a set of predictions is defined by the following function, where x_t is the actual signal value (i.e. it is an element of the set of testing samples) and $x(t)$ is the predicted value:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{x_t - x(t)}{x_t}\right| \tag{5.1}$$

Using MAPE, we compare Neural Decomposition to ARIMA, SARIMA, SVR with a radial basis function, Gashler and Ashmore's model, and ESN. We found that on the unemployment rate problem (Figure 5.1), our approach yielded a model with a MAPE of 10.89%, a 14.15% improvement over the second best model, SVR, which had a MAPE of 25.14%. On the airline problem (Figure 5.2), our approach performed significantly better than other approaches. On the ozone problem (Figure 5.3), ESN was the best model, but ND and SARIMA also performed well. Table 5.2 presents the same data using the root mean square error (RMSE) metric.

Table 5.2: Root mean square error (RMSE) on the validation problems for ARIMA, SARIMA, SVR, Gashler and Ashmore, ESN, and ND. Best result (smallest error) for each problem is shown in **bold**.

Model	**Labor**	**Airline**	**Ozone**	**Speleothem**
ARIMA	2.97	75.32	1.33	N/A
SARIMA	2.41	67.54	1.06	N/A
SVR	2.18	209.57	1.83	1.078
Gashler/Ashmore	2.81	94.47	3.71	N/A
ESN	**1.09**	63.50	**0.705**	N/A
ND	**1.09**	**45.03**	0.99	**0.214**

Chapter 6

Conclusion

In this thesis, we presented Neural Decomposition, a neural network technique for time-series forecasting. Our method decomposes a set of training samples into a sum of sinusoids, inspired by the Fourier transform, augmented with additional components to enable our model to generalize and extrapolate beyond the input set. Each component of the resulting signal is trained, so that it can find a simpler set of constituent signals. ND uses careful initialization, input preprocessing, and regularization to facilitate the training process. A toy problem was presented to demonstrate the necessity of each component of ND. We applied ND to the Mackey-Glass series and was found to generalize well. Finally, we showed results that demonstrate that our approach is superior to popular techniques ARIMA, SARIMA, SVR, Gashler and Ashmore's model, and ESN for some time-series, including the US unemployment rate, monthly airline passengers, monthly ozone concentration in Los Angeles, and an unevenly sampled time-series of oxygen isotope measurements from a cave in north India. We predict that ND will similarly outperform these and other techniques on a number of other problems.

This work makes the following contributions to the current knowledge:

- It empirically shows why the Fourier transform provides a poor initialization point for generalization and how neural network weights must be tuned to properly decompose a signal into its constituent parts.
- It demonstrates the necessity of an augmentation function in Fourier and Fourier-like neural networks and shows that components must be adjustable during the training process, observing the relationships between weight initialization, input preprocessing, and regularization in this context.
- It unifies these insights to describe a method for time-series forecasting and demonstrates

that this method is effective at generalizing for some real-world datasets.

There are three primary areas of future work. First, a study is needed on selecting the augmentation function. Our work only used a linear augmentation function, but intuitively it seems that a more complex set of units would be able to fit a broader spectrum of time-series. Second, ND must be compared to other time-series models such as echo state networks and LSTM networks. We compared ND to a few widely used models, but a comparison to other neural network approaches remains to be done. Third, ND should be applied to new problems. The preliminary findings on the datasets in this thesis show that ND can generalize well for some problems, but the breadth of applications for ND not yet known. Some interesting areas to explore are traffic flow [24], sales [8], financial [34], and economic [22].

References

[1] Tarek Abdelzaher, Yaw Anokwa, Peter Boda, Jeffrey A. Burke, Deborah Estrin, Leonidas Guibas, Aman Kansal, Samuel Madden, and Jim Reich. Mobiscopes for human spaces. *Pervasive Computing, IEEE*, 6(2):20–29, 2007.

[2] Peter Bloomfield. *Fourier analysis of time series: an introduction*. John Wiley & Sons, 2004.

[3] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley, June 2008.

[4] Sofiane Brahim-Belhouari and Amine Bermak. Gaussian process for nonstationary time series prediction. *Computational Statistics & Data Analysis*, 47(4):705–712, 2004.

[5] Chris Chatfield. *Time-series forecasting*. CRC Press, 2000.

[6] Chris Chatfield. *The Analysis of Time Series: An Introduction*. Chapman and Hall/CRC, 6 edition, July 2003.

[7] Shyi-Ming Chen. Forecasting enrollments based on fuzzy time series. *Fuzzy Sets and Systems*, 81(3):311–319, August 1996.

[8] Tsan-Ming Choi, Yong Yu, and Kin-Fan Au. A hybrid sarima wavelet transform method for sales forecasting. *Decision Support Systems*, 51(1):130–140, 2011.

[9] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

[10] Georg Dorffner. Neural networks for time series processing. In *Neural Network World*. Citeseer, 1996.

[11] Harris Drucker, Chris J.C. Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems 9*, 1996.

[12] Christian E. Elger and Klaus Lehnertz. Seizure prediction by non-linear time series analysis of brain electrical activity. *European Journal of Neuroscience*, 10(2):786–789, February 1998.

[13] Ray J Frank, Neil Davey, and Stephen P Hunt. Time series prediction and neural networks. *Journal of Intelligent and Robotic Systems*, 31(1-3):91–103, 2001.

[14] M. S. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, 12:2383–2387, July 2011.

[15] Michael S. Gashler and Stephen C. Ashmore. Training deep fourier neural networks to fit time-series data. In *Intelligent Computing in Bioinformatics - 10th International Conference, ICIC 2014, Taiyuan, China, August 3-6, 2014. Proceedings*, pages 44–55, 2014.

[16] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Artificial Neural Networks—ICANN 2001*, pages 669–676. Springer, 2001.

[17] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[18] Keith W. Hipel and A. I. McLeod. *Time Series Modelling of Water Resources and Environmental Systems*. Elsevier, January 1994.

[19] Kyoung jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, September 2003.

[20] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[21] Ma Jun and Meng Ying. Research of traffic flow forecasting based on neural network. In *Intelligent Information Technology Application, 2008. IITA '08. Second International Symposium on*, volume 2, pages 104–108, December 2008.

[22] Iebeling Kaastra and Milton Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236, 1996.

[23] Decai Li, Min Han, and Jun Wang. Chaotic time series prediction based on a novel robust echo state network. *Neural Networks and Learning Systems, IEEE Transactions on*, 23(5):787–799, 2012.

[24] Marco Lippi, Matteo Bertini, and Paolo Frasconi. Short-term traffic flow forecasting: An experimental comparison of time-series analysis and supervised learning. In *IEEE Transactions on Intelligent Transportation Systems*, volume 14, pages 871–882, March 2013.

[25] Mantas Lukoševičius. *A practical guide to applying echo state networks*, volume 7700 of *Lecture Notes in Computer Science*, pages 659–686. Springer Berlin Heidelberg, 2 edition, 2012.

[26] Iain L MacDonald and Walter Zucchini. *Hidden Markov and other models for discrete-valued time series*, volume 110. CRC Press, 1997.

[27] Keiichiro Minami, Hiroshi Nakajima, and Takeshi Toyoshima. Real-time discrimination of ventricular tachyarrhythmia with fourier-transform neural network. *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING*, 46(2), February 1999.

[28] O. Nerrand, P. Roussel-Ragot, D. Urbani, L. Personnaz, and G. Dreyfus. Training recurrent neural networks: Why and how ? an illustration in dynamical process modeling., 1994.

[29] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[30] Zhiwei Shi and Min Han. Support vector echo-state machine for chaotic time-series prediction. *Neural Networks, IEEE Transactions on*, 18(2):359–372, 2007.

[31] Adrian Silvescu. Fourier neural networks. In *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, volume 1, pages 488–491, 1999.

[32] Ashish Sinha, Gayatri Kathayat, Hai Cheng, Sebastian F. M. Breitenbach, Max Berkelhammer, Manfred Mudelsee, Jayant Biswas, and R. L. Edwards. Trends and oscillations in the indian summer monsoon rainfall over the last two millennia. *Nat Commun*, 6, 02 2015.

[33] Haowei Su, Ling Zhang, and Shu Yu. Short-term traffic flow prediction based on incremental support vector regression. In *Natural Computation, 2007. ICNC 2007. Third International Conference on*, volume 1, pages 640–645, August 2007.

[34] Francis E.H. Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, August 2001.

[35] James W. Taylor, Patrick E. McSharry, and Roberto Buizza. Wind power density forecasting using ensemble predictions and time series models. *Energy Conversion, IEEE Transactions on*, 24(3):775–782, September 2009.

[36] William Wu-Shyong Wei. *Time series analysis*. Addison-Wesley publ, 1994.

[37] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[38] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[39] Guoqiang Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, January 2003.