The Wavelet RNN: Timeseries forecasting using multiresolution structure

by

Noam Finkelstein

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science.

Baltimore, Maryland

August, 2017

Abstract

In this work, we look to develop a method for handling time series data that can take advantage of structure in the signal at multiple resolutions. We draw inspiration from past work on wavelet decompositions and recent architectures in the recurrent neural network (RNN) literature. By combing key elements from each tradition, we develop the Wavelet RNN (WRNN) method.

Past research has proven that wavelet decompositions are a useful prism through which to work with time series data, especially with signals that display a multiresolution structure. Traditional wavelet transforms such as the discrete wavelet transform (DWT) and the stationary wavelet transform (SWT), while very useful in many settings, have properties that make them difficult to work with in a forecasting setting with arbitrary-length data.

We develop the sequential discrete wavelet transform (SDWT), which is closely related to the DWT and the SWT but is easier to work with for forecasting problems with non-aligned, arbitrary-length data. We then explore two different ways of applying neural networks to the SDWT to obtain forecasts that are aware of the

ABSTRACT

multiresolution structure in the data.

We also note that there has been an increasing interest in certain kinds of model interpretability. Towards that end, we develop a toolkit for visualizing the influence of each past point in a timeseries on a forecast. We use this toolkit to explore some of the properties of the WRNN.

We compare the WRNN to RNN models that are specifically designed with mutliscale structure in mind, as well as to an ARIMA model. Our experiments are conducted on simulated data, as well as on heart rate data obtained from a publically available database. We find the WRNN is competative with the results of state-of-the-art models.

Advisor: Dr. Suchi Saria

Reader: Dr. John Muschelli

Acknowledgments

I would like to thank my thesis advisor Dr. Suchi Saria for her mentorship, and for challenging me to think in new and difficult ways at every turn.

I would also like to thank my academic advisor and thesis reader Dr. John Muschelli for his statistical insight, and his guidance and support throughout the year.

Thank you to Peter Schulam for invaluable conversations throughout this project, and for lending me his sharp debugging skills.

I would also like to thank Mary Joy Argo for incredible organizational skills, and for her help in navigating departmental rules.

Finally, thank you to my friends and family for everything else.

Contents

CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction and Related Work

In this work, we develop a wavelet and recurrent neural network based method for time series forecasting, called Wavelet RNN (WRNN). We also develop a toolkit for evaluating the importance of past points in the time series to forecasts.

This introduction will frame the problems we hope to address, explore related work, and provide an overview of the methods themselves.

1.1 Wavelet RNN

A central motivation for the development of WRNN is the recognition that many important time series data in clinical medicine exhibit multiscale or multiresolution structure.

For example, heart rate data may have minute-scale structure that is influenced

by a patient's immediate surroundings, hour-scale structure that tracks short-term treatment effects, day-scale structure following the circadian rythm, and longer term structure due to some underlying disease. We are interested in developing a model that can take advantage of this underlying multiresolution structure to enhance its predictions. We also believe that introspecting into such a model can provide inside into the nature of the dataset.

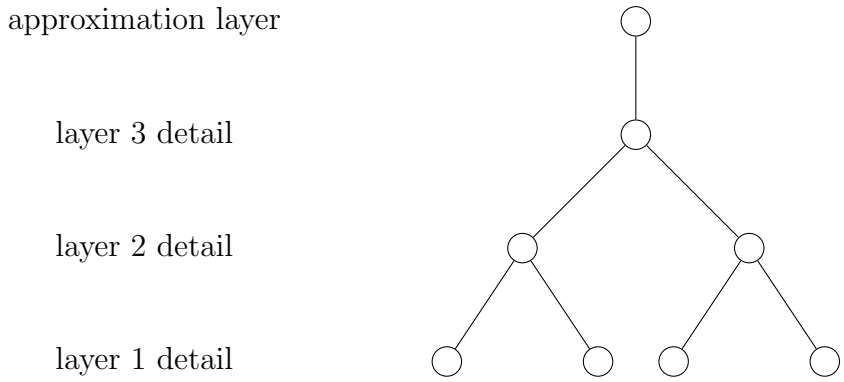
1.1.1 Wavelet background

Wavelets have frequently been used to explore the multiresolution structure of a wide class of signals.

At its most basic level, a wavelet is any wave-like function with finite support and mean zero. This differentiates it from something like a sine wave, which has infinite support.

The discrete wavelet transform is often discussed in terms of a mother wavelet ψ, which is the wavelet function, and some scaling function. The choice of which ψ to use depends on the nature of the signal - different wavelet functions are adept at capturing different kinds of structure in the signal through a DWT. The mother wavelet is then scaled by powers of two. For each scale, it is shifted so as to cover the signal. The resulting shifted and scaled functions are called the daugher wavelets. For some scale j and shift k, the daughter wavelet is given by:

Figure 1.1: A simple DWT coefficient structure



This figure shows the structure of a level 3 DWT of a length 8 signal. When the DWT is considered as a tree, as in [2], each approximation coefficient has a single child, and each detail coefficient has two children. If the DWT was only carried out to level 2, there would be 2 approximation coefficients. In this work, we will always execute the DWT to the maximum possible level.

$$\psi_{j,k}(t) = \frac{1}{\sqrt{2^j}}\psi\left(\frac{t - k2^j}{2^j}\right) \tag{1.1}$$

These daugher wavelets are then convolved with the signal to produce the detail coefficients of the discrete wavelet transform. [1] goes into great detail about the theoretical underpinnings of the DWT and other wavelet-based techniques.

Figure 1.1 shows the structure of coefficients that result from a DWT. This structure will be useful to keep in mind, as it will form the basic building block of the SDWT.

From a practical perspective, it turns out that this procedure is equivalent to repeatedly putting the signal through a low pass and high pass filter that form a Quadrature Mirror Filter pair [3]. The results of passing the signal through the highpass filter, and then downsampling, are called the detail coefficients. The results of passing the signal through the low pass filter and then downsampling are called the approximation coefficients. This procedure can be repeated on the approximation coefficients until there are no longer enough approximation coefficients to continue, or it can be discontinued at any level.

Figure 1.2 is a visualization of the results of a DWT on a simple simulated signal.

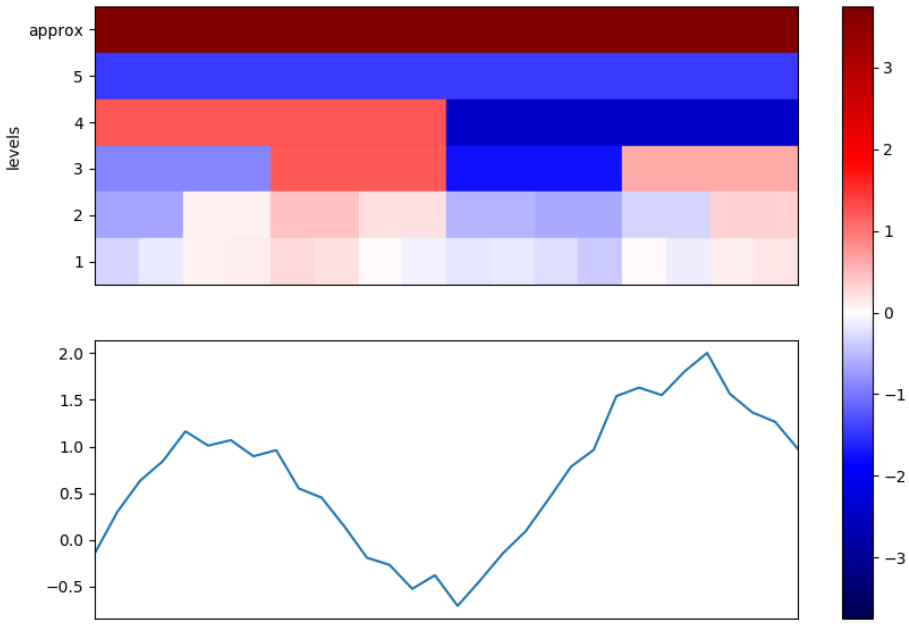
The stationary wavelet transform (SWT) is performed in a very similar manner to the DWT. The only difference is that instead of downsampling the signal to match the scaled filter at higher levels of the transform, we dilate the filter to match the signal length. For this reason, the SWT is also sometimes called the undecimated wavelet transform.

For a transform with J levels of a signal with T timesteps, the DWTs starting at timepoint $T - 2^J + 1$ will not have enough of a signal to operate on. In the SWT, the front of the signal is wrapped around to the back to make up for this difference. Note that this is only reasonable if the signal is stationary.

Figure 1.3 is a visualization of an SWT performed on that same simulated signal.

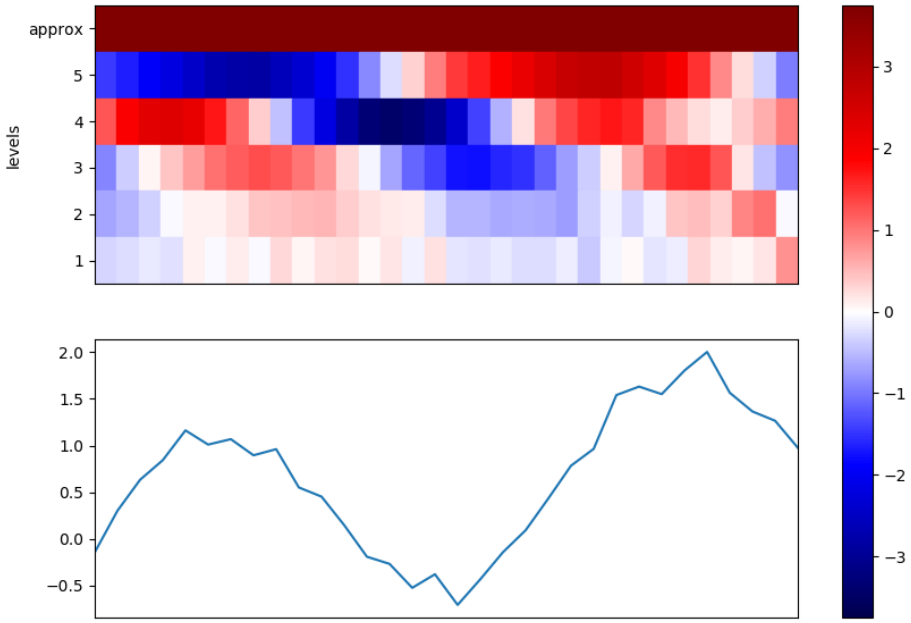
In chapter 2 we describe the shortfalls of both the DWT and the SWT for our purposes, and develop the sequential discrete wavelet transform (SDWT), a closely

Figure 1.2: A DWT decomposition



In this figure, the heatmap above the signal represents the values of the detail and approximation coefficients of the DWT. Because we are taking the DWT to the maximum depth, there is only one approximation coefficient that applies to the whole signal. Each detail coefficient is represented by a single rectangle that spans the part of the signal to which it applies. The colorbar to the right of the image shows the map between colors and coefficient values. White corresponds to zero-valued coefficients, red to positive coefficients and blue to negative coefficients. The intensity of the color in a given rectangle corresponds to the scale of the coefficient.

Figure 1.3: An SWT decomposition



This figure is similar to 1.2. Because of the nature of the SWT, the rectangles representing the coefficients cannot span the entire part of the signal to which they are related. Instead, they're located directly above the first timestep to which they're related, and their influence stretches 2^j timesteps, where j is the level of the coefficient.

related transform that is easier to use on variable length data.

All images in this section were generated with the dwtviz library, which is discussed in section 6.2.

1.1.2 Modeling with wavelets

The discrete wavelet transform has been used in a wide range of signal processing tasks. It has been used extensively for functional regression [4], clustering tasks [5], signal denoising [2], and more. By nature, wavelets are especially helpful in understanding the activity of a signal at multiple resolutions.

[6] and others demonstrate that the discrete wavelet transform on a wide class of signals has certain desireable properties. [2] noted that many of these properties were not adequately exploited by wavelet-based methods. In particular, the coefficients of a discrete wavelet transform tend to be sparse. Additionally, they tend to occur in clusters, so that if the parent or sibiling of a detail coefficient is non-zero, that coefficient is likely to be non-zero.

[2] captures this structure by introducing the idea of markov trees, which specifically capture the dependence of children on parents in the coefficient tree. [7] exploits this structure by expanding markov trees to a functional ANOVA setting.

We take advantage of this structure in our WRNN method.

1.1.3 Neural networks

Artifical neural networks are a class of models initially inspired by the workings of the human brain. Neural networks connect an input to an output through a number of neurons, each of which is put through some activation function, which is usually non-linear. Fitting a neural network to data involves learning the optimal weights for the connections between the inputs and neurons, neurons and other neurons, and neurons and the outputs. If no non-linear functions are used, neural networks can only represent linear combinations of the inputs, so fitting them becomes a form of linear regression.

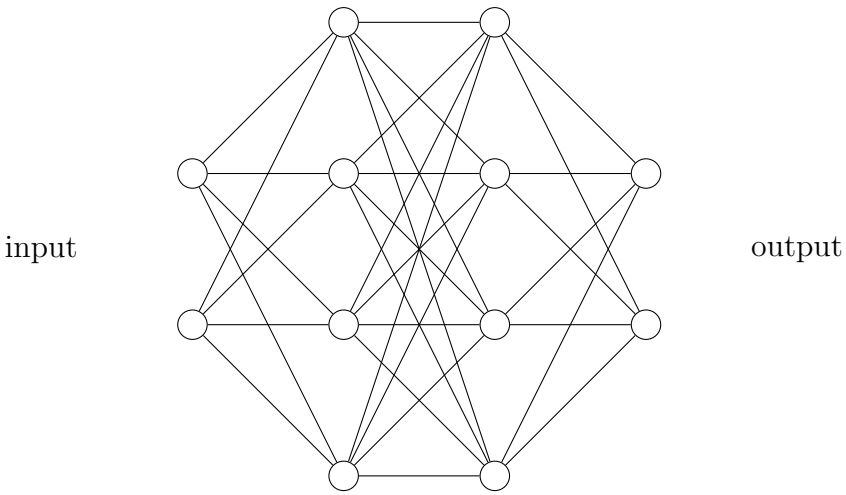
The neurons between the input and output are aligned into 'hidden layers'. A network in which each node is connected to every node in the following layer is called a fully connected network.

A basic fully connected neural network is illustrated in figure 1.4.

Simple neural networks, as described above, will only work with fixed-length inputs. To extend a similar idea to sequences of arbitrary length, the recurrent neural network was developed.

The basic idea of a recurrent neural network is that the network maintains some state that summarizes the relevant information in the sequence seen up to the current time. At each step, it accepts both that state and the input for that step. These two are fed into a neural network that maintains the same architecture between steps. This neural network then produces the new state as well as the output for that step.

Figure 1.4: Neural Network



This figure illustrates a simple fully connected neural network with two hidden layers, and input and output sizes of 2.

In the simplest case, the neural network at each step has a single hidden layer. The hidden state and the input at some time t are concatenated and treated as a single input vector x, and the output from the network is split into the next network state and the output at time t.

RNNs are most often not used directly on time series data. Instead they are often used on symbolic data. For example, they are used in natural language processing for tasks such as language modeling. In these contexts, the idea of multiscale structure is still valuable.

Recurrent neural networks tend to have difficulty retaining information from the distant past of a sequence because of the vanishing gradient problem [8]. Simply put the gradient of the weights with respect to points in the distant past goes to zero.

Much work has been done on overcoming the vanishing gradient problem. Notably, the long short term memory (LSTM) network [9] approach has seen widespread adoption.

More recent approaches have tended to focus on the notion of multiscale structure. In other words, they seek to allow more recent information to have an immediate impact while keeping track of longer-term information through a separate mechanism.

[10] introduces the clockerwork RNN. This RNN operates very similarly to the simple RNN described above, but the hidden layer is partitioned into some number of modules, each of which is updated according to its own clock speed. As such, the network is forced to use information that has not been updated in quite some time.

[11] develops the hierarchical and multiscale long short term memory (HMLSTM) network. This network operates much like a stacked LSTM, with the addition of a boundary detection neuron that learns to fire when there is a boundary at that level of the underlying data. The authors apply this network to text data, and find that at the highest resolution it detects word boundaries, and at the second layer it detects plausible phrase boundaries.

[12] uses a partially connected convolutional neural network. Instead of using a RNN to handle sequential data, a Wavenet model with J hidden layers uses the most recent 2^J points in the sequence by dilating the receptive window between layers. There are strong resonances between the Wavenet achitecture and the idea of the discrete wavelet transform.

1.1.4 Multiscale & Multiresolution Terminology

It is useful at the outset to understand exactly what we mean when we talk about multiscale or multiresolution structure.

In the wavelet literature, the term 'multiresolution' is used to refer to the fact that each daughter wavelet captures unique information. The information captured at each scale represents a single resolution, such that the DWT is refered to as a multiresolution analysis [13].

In the neural network literature, researchers tend to refer to 'multiscale' structure [11], [10]. This tends to simply mean that there is relevant information at variably

distant points in the input sequence.

As will become evident, we make use of both concepts in our method, through our use of wavelet transforms and recurrent neural networks. In practice, these ideas are coterminous; activity at higher resolutions happens at shorter time scales. We will therefore use the term 'multiresolution' to refer to the general idea captured by both terms.

1.1.5 Wavelet RNN Method

Our wavelet RNN method combines work in the tradition of wavelets and neural networks. We develop the sequential discrete wavelet transform (SDWT), which is a variation on the discrete wavelet transform (DWT) that permits of handling variable-length sequences. We train the Wavelet RNN to predict the next coefficient at each layer of the SDWT. We then take the inverse SDWT to acquire the one-step-ahead prediction.

We believe the WRNN takes advantage of this representation to bias the model to pick up multiresolution structure. This is in some ways similar to the point made in [14], which speculates that using a different wavelet-based transform called the Scattering Transform on image samples provides a better representation of the data than the raw image.

As compared to a classic RNN or LSTM [9], the WRNN seems to do better at overcoming the vanishing gradient problem [8]. The main reason for this is that the

approximation coefficients of the SDWT, which can be thought of as a sort of moving average, change more slowly and are influenced by more of the time signal than the high resolution coefficients. By operating directly on the SDWT, the neural networks are forced to be aware of activity at both resolutions.

1.2 Model Interpretability

A large part of the goal of developing a prediction technique that accounts for multiresolution structure in time series data is to discover recurring patterns, or motifs [15], that depend on activity at different resolutions in the history of the time series.

We were inspired by [16], which develops a method that can be applied to any classifier to detect which features of a sample are important to the classifier's prediction of its class. In a similar vein, [17] develops a method that can be applied to any convolutional neural network to provide visual explanations for classification decisions.

Towards that end, we built a tool that provides a post-hoc explanation [18] for the model's prediction. Our influence plots show visualizations of the gradient of the prediction with respect to every previous observation in the time series. This allows us to understand which historical points the model 'looks at' when it makes its predictions, and how each of those points affects the outcome.

Chapter 2

Sequential Discrete Wavelet Transform

This chapter develops the sequential discrete wavelet transform (SDWT).

2.1 Motivation

After exploring a number of possible representations for time series data, we adapted the discrete wavelet transform. As discussed above, the properties of the discrete wavelet transform have been well studied, and it is frequently used in the analysis of audio and visual data.

Dispite the convenience of the DWT, it has one major issue that makes it unsuitable for use in this case. It is not shift invariant, which means that an identical

feature in the time series will show up differently depending on where in the time series it occurs. Figure 2.1 demonstrates this lack of shift invariance. This is not an obstacle for dealing with time-aligned data, and the DWT has proven to be a very useful in functional regression for such data [4], [19]. However, we are interested in dealing with observational data in a clincal medicine context. Such data streams are rarely the same length, let alone time aligned in any meaningful sense.

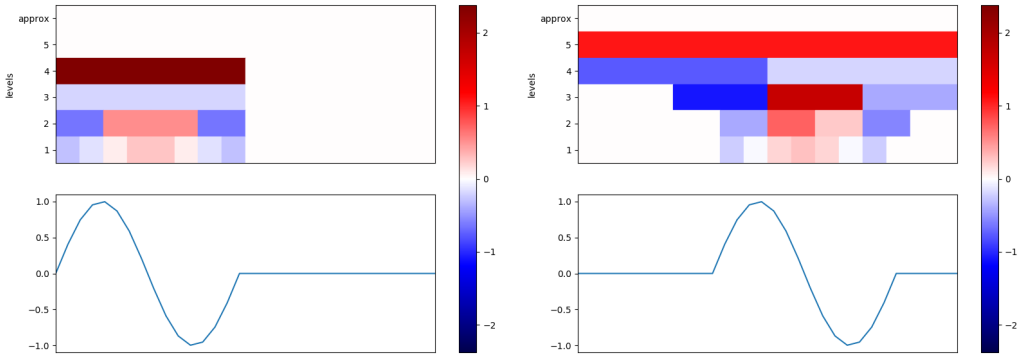
Further, not all lengths of time series are equally convenient to work with. Time series that have a length that is a power of two, for example, are easiest to work with for the Haar wavelet.

More recently, the merits of the undecimated wavelet transform, also known as the stationary wavelet transform, have been recognized in statistical settings [20]. The stationary wavelet transform is shift invariant, as demonstrated in figure 2.4 but, as the name implies, it works best on stationary signals. Additionally, because it contains redundant information, there is no concensus on how it should be inverted. Like the DWT, the SWT with the Haar wavelet works best on signals that have a length that is a power of 2.

The major benefit of the SWT over the DWT is that it is shift invariant in the sense that if we consider the signal as a ring buffer, the coefficients calculated by the SWT will be shifted versions of each other regardless of where in the signal we start.

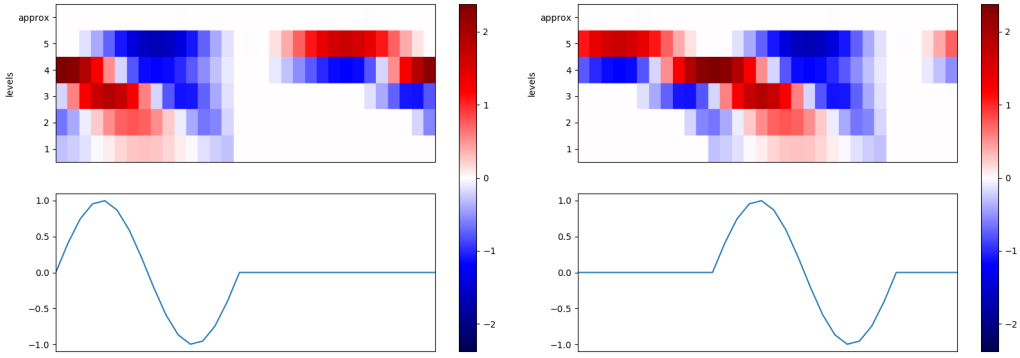
The SWT is equivalent to doing a DWT at each time point, and wrapping the signal as necessary to copmlete the DWT at later time points. This wrapping behavior

Figure 2.1: DWT of shifted signal



This figure demonstrates the clear lack of shift invariance of the DWT.

Figure 2.2: SWT of shifted signal



Unlike figure 2.1, this figure demonstrates that the coefficients of the SWT take on the same values regardless of where in the signal the decomposition is started. The heatmaps in these figures are simply shifted versions of each other.

is problematic for data that is not inherently stationary.

In a prediction setting, it is not immediately clear how either the DWT or the SWT could be used for forecasting into the future. Predicting one step ahead at each layer of the DWT would produce something that the inverse DWT could not invertable. It is possible, for example, to predict the coefficients of a new DWT that begins after the current one, but due to the nature of the DWT it is difficult to ensure continuity between the existing signal and the prediction. Forecasting through the SWT faces similar problems.

These reasons explain why wavelets have not been used in a forecasting setting before, and why they only work with time-aligned signals in regression or ANOVA settings.

The SDWT solves the shift invariance issue of the DWT without adopting the questionable wrapping approach of the SWT. It also provides an intuitive way to think about prediction, which will allow us to bring the multiresolution power of the wavelet transform into the forecasting setting with the WRNN.

2.2 Definition

For the purposes of this work, we develop the sequential discrete wavelet transform (SDWT). To conduct an SDWT, we take the DWT at steps 1 through $T - 2^J + 1$. Because of the nature of the DWT, many coefficients in the DWTs will be repeated

Figure 2.3: A visualization of the SDWT

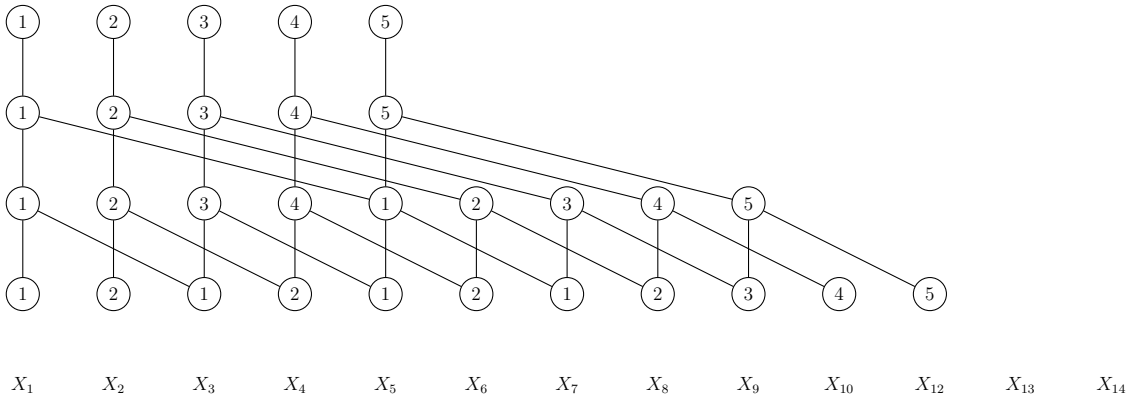


Figure 2.3: This graph demonstrates the relationship of the SDWT to the DWT. Each tree in the figure is a DWT, and the number in each node of the tree is the first DWT for which that coefficient had to be calculated. The nodes in the top layer are the approximation coefficients, the nodes in the second to top layer are the level 3 detail coefficients, and so forth. Note that after $t = 4$, each subsequent DWT needs only one single coefficient at each layer to be complete. To predict X_{14}, we would need to predict one additional coefficient at each level, and then take the inverse SDWT.

across DWTs. This is illustrated in 2.3.

The results of the SDWT will be $J + 1$ vectors, one of approximation coefficients, and one for each level of detail coefficients. Unlike the simple DWT it will not be a tree, and unlike the SWT it will not be a matrix.

For a signal of length T and a SDWT of level J, there will be $T - 2^J + 1$ approximation coefficients, and the same number of detail coefficients at level J. Then level j of the transform will have $T - \sum_{i=0}^{j-1} 2^i$ detail coefficients.

Figure 2.3 shows how the SDWT is contructed. Conceptually we take the DWT starting at each timestep from T to $T - 2^J + 1$, and keep the coefficients from the DWT that we haven't already calculated in previous DWTs.

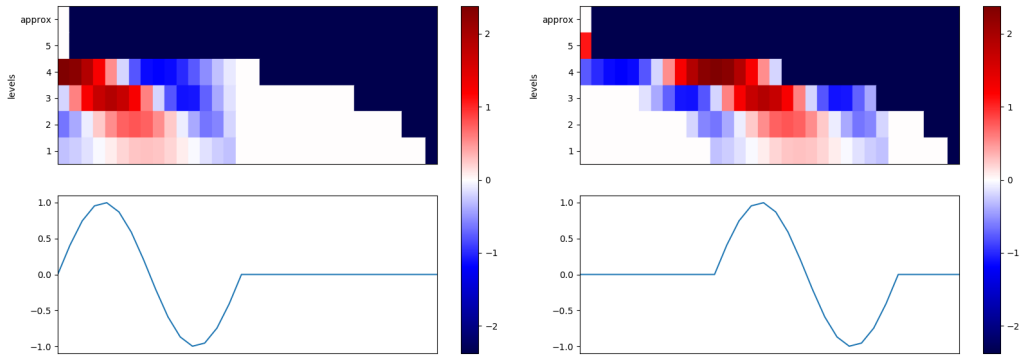
The inverse SDWT (ISDWT) is performed by extracting the first DWT tree from the coefficients acquired through the SDWT and inverting it to obtain the first 2^J elements of the signal. The element at the following timestep is obtained by extracting the second DWT tree, inverting it [13], and appending the last element to the exisiting signal. This is repeated until there are no further coefficients in the SDWT.

Note that the SDWT is a subset of the coefficients in an SWT when the original signal has a length that is a power of 2. Unlike the SWT, we stop taking DWTs once we run out of timesteps in the original signal, so that there is no need to wrap the end of the sequence back around to the front. Also unlike the SWT, it is always possible to take the level J SDWT when the signal length is greater than 2^J.

The SDWT has several desireable properties. Much like the SWT, it is invariant to some manner of shifts. In particular, if some subset of the time series lasts for 2^j timesteps, SDWT coefficients at layer j and lower will by construction be identical anywhere the feature reoccurs.

For comparison, figure 2.4 is the SDWT version of figures 2.1 and 2.2. It is worth noting that the SDWT shift invariance property holds even for sequences that the SWT and DWT cannot properly decompose.

Figure 2.4: DWT of shifted signal



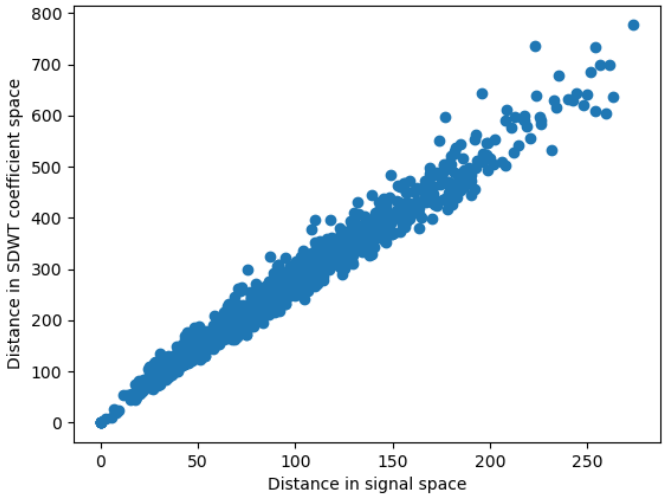
This figure demonstrates the nature of the SDWT's shift invariance. In comparison with the SWT, the blacked out coefficients do not exist in the SDWT.

2.3 Stability

We are also interested in the SDWT's stability [13]. If a signal changes by a small amount, we want the SDWT coefficients to change by a similarly small amount. If this relationship is not smooth, we will encounter many difficulties. First, small errors in predicting the SDWT coefficients may result in massive mistakes in the time-domain predictions. Second, we would need very powerful networks to learn anything in the SDWT representation for even a simple class of functions in the time domain.

To verify that the SDWT is stable in the manner described, we run experiments on signals simulated from several different generative processes. In each case, we map the relationship between the sum of differences between points in the time domain and the sum of differences in the corresponding SDWT coefficients.

Figure 2.5: Stability of SDWT



This figure demonstrates the nature of the relationship between the distance between the raw signals and the distance between the SDWT coefficients of those signals. As we would hope, the relationship is roughly linear, with increasing variability as the signals become more dissimilar.

We observe that across different classes of signals, the relationship between these distances is well represented by an affine function.

Figure 2.5 represents this relationship for a set of signals generated by the process described in section 6.3, using a limited number of underlying 'motifs' and minimal noise.

Chapter 3

Wavelet RNN

We have now set the stage to discuss the wavelt RNN method.

The basic idea behind the wavelet RNN is as follows. For each incoming signal, we take the level J SDWT of that signal. This gives us one sequence of approximation coefficents at the lowest level, and then J levels of detail coefficients.

As mentioned in section 2.2, to do one-step-ahead prediction on the timeseries by way of the SDWT, we simply need to predict one step ahead for the approximation coefficient layer and for each of the detail coefficient layers.

Below, we describe two variations on this basic method. In one, we use a single RNN, and in the next, we use multiple coordinated RNNs.

3.1 Single Network WRNN

One approach is to train a single LSTM to jointly predict the next coefficients of the SDWT at every layer, padding the beginnings of the shorter layers out with zeros as necessary. The downside of this technique is that we do not necessarily exploit the structure of a wavelet transform. We know from past research that parent and children nodes in the DWT have a special relationship this model doesn't bias the network towards exploiting the structure. Instead it must learn that relationship for itself.

However, this technique is also capable of picking up on other relationships than the ones between parent and child. This flexibility seems to require more training data than the multi network model, described in the next section, but it also tends to produce less noisy forecasts.

3.2 Multi Network WRNN

A simplistic approach to the multinetwork technique would be to predict forward each layer of the SDWT independently with an LSTM. Our experiments revealed that prediction under this technique suffers from a lack of alignment between layers. This lack of coordination between different resolutions translates into poor predictions in the time domain. Additionally, this technique in no way takes advantage of the highly structured relationship between parent and children coefficients exploited by [2]

and [7].

Instead, we adopt the following approach. We fit a simple LSTM to the approximation coefficients. Then, we fit an LSTM with a 2-dimensional input for each of the detail coefficients, where the prediction for the DWT parent coefficient in the layer above, as well as the actual immediate predecessor in the current level, were concatenated to form the input. This process is laid out in figure 3.1.

The multi network WRNN tends to be difficult to train, and in some cases will produce very noisy forecasts. However, it seems to perform better than the single network variant in low data settings.

Figue 3.2 shows an interesting failure case for undertrained LSTM and Multi WRNN models. Both underestimate the upwards trajectory at the end of the signal, but the WRNN catches the higher-resolution activity fairly well, despite missing the big picture. This failure case lends credence to our intuition that the WRNN should pick up and learn from activity and different resolutions more easily than a traditional RNN. We see this kind of behavior more with the Multi WRNN than with the Single Network WRNN because predictions of the coefficients at different layers of the SDWT are more independent of each other.

Figure 3.1: A visualization of the WRNN

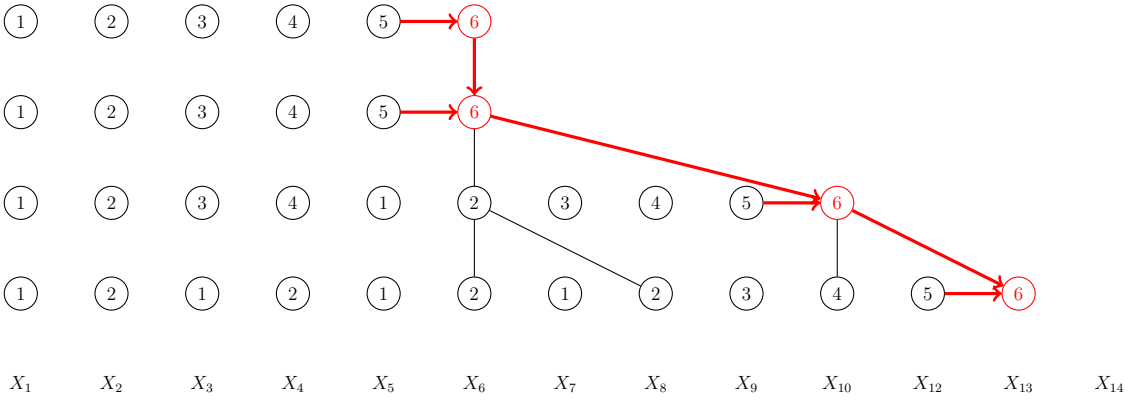
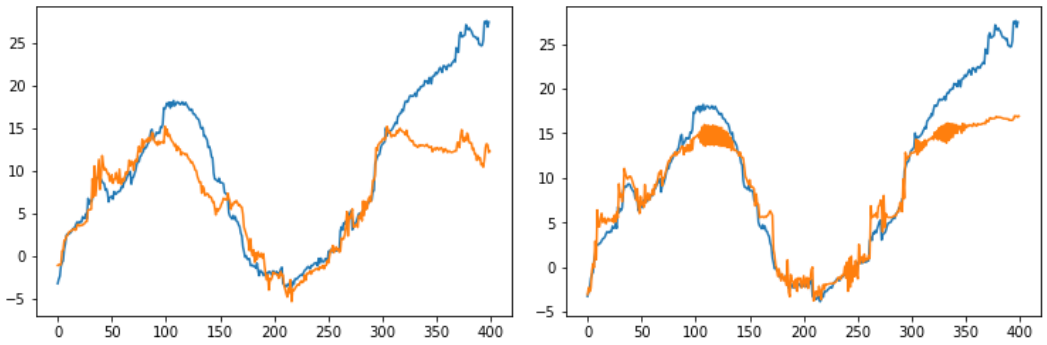


Figure 3.1: This figure shows how the WRNN technique is used for one-step-ahead prediction. The red arrows represent which coefficients are used as inputs into the RNN at each layer, and the red nodes represent the predicted coefficients. Thus, at the approximation coefficient layer, the only input is the most recent approximation coefficient, and the RNN's internal state. At the top detail coefficient layer, the inputs are the most recent detail coefficient at that layer, and the predicted approximation coefficient. The indicated DWT tree is then extracted and inverted, providing the prediction for $X_{6:14}$, from which we extract the prediction for X_{14}.

Figure 3.2: WRNN and LSTM Failure Case

(a) WRNN Failure Case

(b) LSTM Failure Case



These images show failure cases of an undertrained LSTM network and an undertrained WRNN network. In both the blue signal is the truth, and the orange signal is the one-step-ahead prediction overlay.

Chapter 4

Model Interpretability

We are interested in understanding what exactly influences a forecasting model's predictions, and by how much. To that end, we developed influence plots that show the gradient of a forecast with respect to each of the previous timesteps.

Points with near-zero gradients can be thought of as having very little effect on the prediction in that if they were moved, the prediction would remain nearly unchanged. Points with large gradients can be thought of as having substantial influence on the prediction of interest. If the gradient is positive, a higher observed value for that timestep would lead to a higher prediction, and vice versa.

Other techniques designed to improve the interpretability of model predictions such as [16], or models designed to be interpretable such as [21], have conducted experiments with humans to evaluate the effects of providing an explanation for the model's decisions. Due to resource limitations, we were not able to conduct such

experiments.

From personal experience, though, we do find that these tools help with model diagnostics, and provide insight into the differences between different models. We also find that in simple signals, the visual explanation provided by the influence plots accorded with the known generative process.

In forecasting problems, the most recent observations are the most important. This tendency is clearly reflected in our influence plots. We are also interested in finding out which features the models finds important when making its one-step-ahead predictions over the duration of the time series. Our software can generate an animation that shows these influence plots at each step of the time series. Figures 4.1 and 4.2 are static versions of these animations. Each plot shows the importance of every point in the time series for the one-step-ahead prediction at a given timestep.

Figures 4.1 and 4.2 show the gradient of all past timesteps for one-step-ahead prediction for t=33 to t=49 for the WRNN model and an LSTM network. In the WRNN case, the influence of large sections of the time series is substantial through the predictions. In the LSTM case, the influence of past sections of the time series fades much more quickly.

This shows, at a minimum, that the WRNN model is making use of more historical information than a traditional LSTM.

These plots also give us some insight into which past features of the time series the model considers to be important for its predictions at different time points. This

Figure 4.1: Influence plots for WRNN

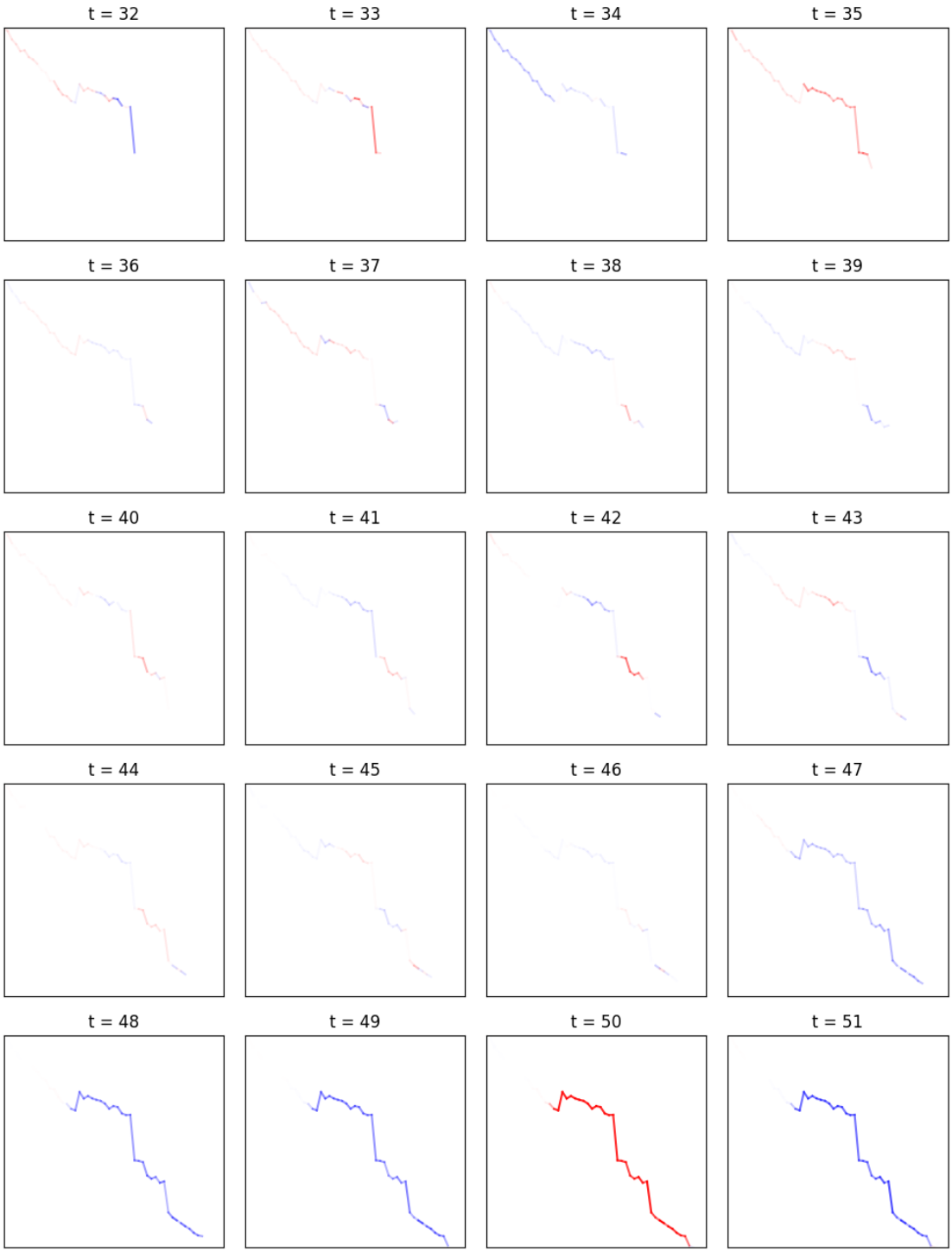
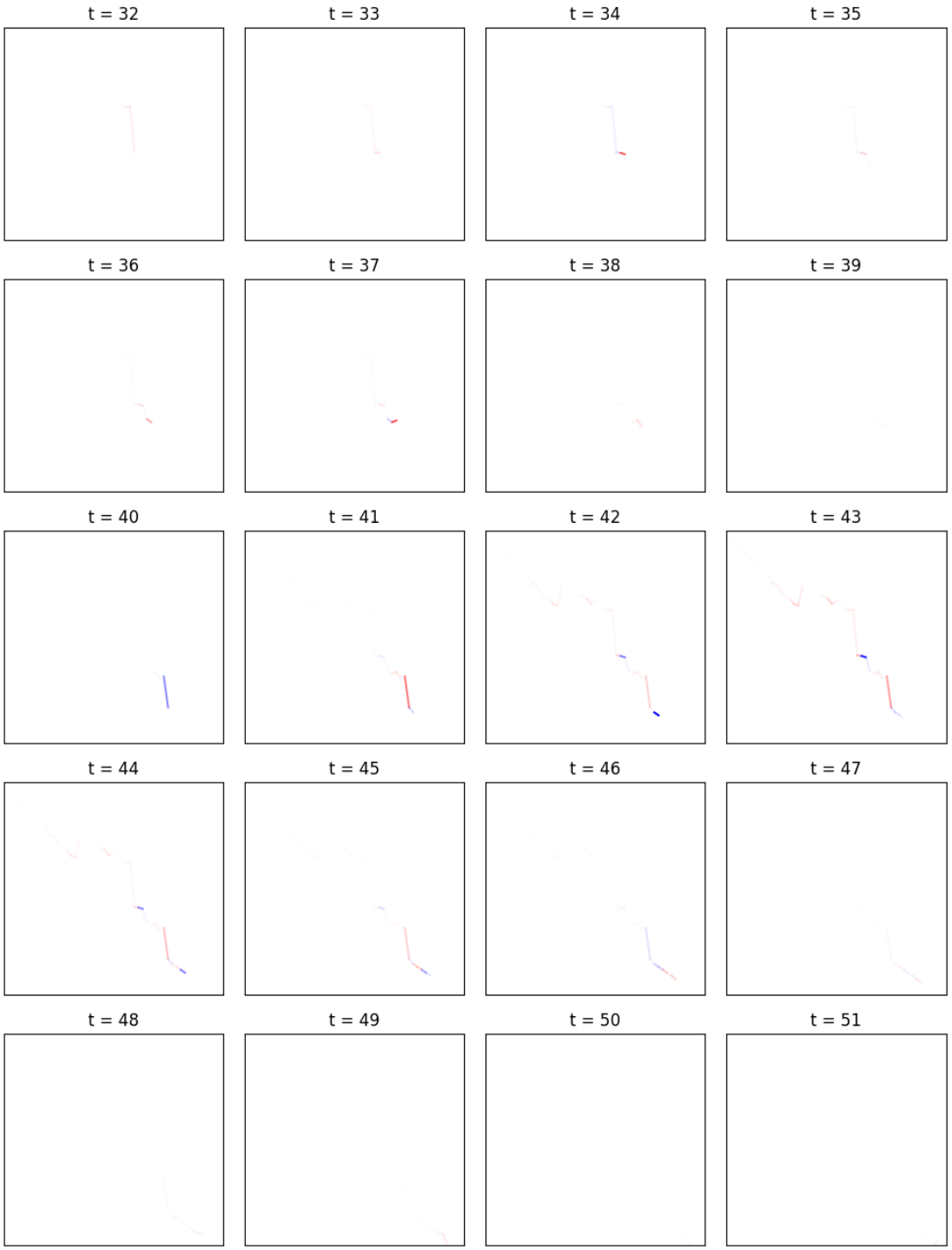


Figure 4.2: Influence plots for LSTM

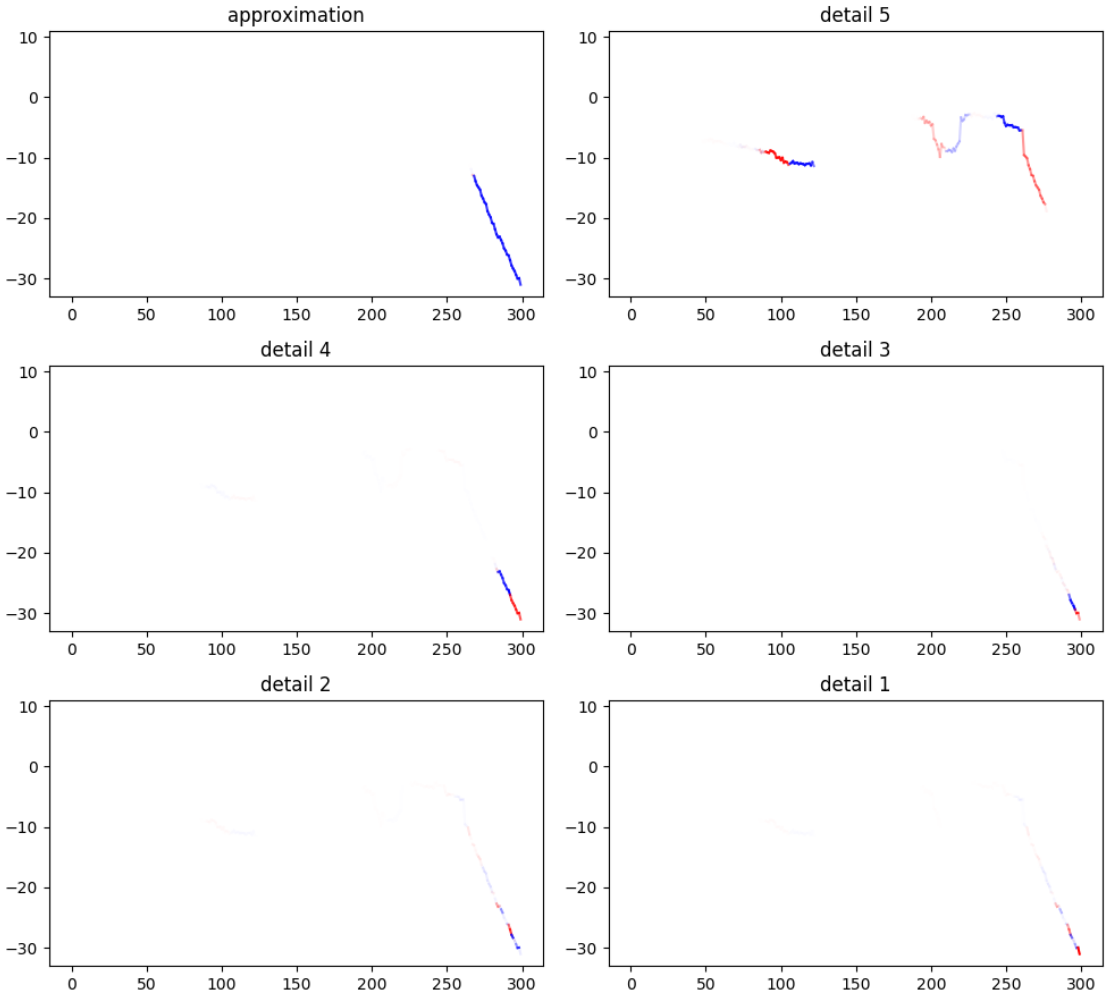


kind of information can be used to explain, or interpret, model predictions in a live setting.

We can also examine influence plots by the resolution of the time series. In figure 4.3, we look at the influence of each previous point with respect to the prediction at each layer of the SDWT. We would expect to see that the approximation coefficient and level 5 detail coefficient have the widest window by which they're influenced, and that the higher resolution coefficients are influenced by shorter term activity.

One avenue for future work suggested by these plots is the development of a clustering method that groups important activity at different resolutions together into motifs. This could in some ways be considered a multiresolution extension of the work in [15], and would be of use in signal classification and prediction.

Figure 4.3: Multiresoluion influence plots for WRNN



This figure shows the influence of each point of the time series on the predictions at each level of the SDWT.

Chapter 5

Experiments and Results

In this chapter, we run experiments on simulated data and real data.

We compare the WRNN, in both its single network 3.1 and multinetwork 3.2 forms, to a number of other methods. We use a basic LSTM [9], the hierarchical multiscale lstm [11], the clockwork RNN [] and a simple ARIMA model. We developed our own implementation of the hmlstm, described in section 6.4. We used the implementation of the clockwork RNN contained in the theanets package, which is built on the theano [22] library. We used the forecast [23] package in R for the ARIMA models.

The experiments on real data and the experiments on simulated data both compare the models' average loss on one-step-ahead predictions at every timestep of every signal. We used 64 hidden units in each recurrent neural network. We use two layers in the HMLSTM model. We use 1, 4, 16, and 32 as clockspeeds in the clockwork RNN. We use the "auto.arima()" function in the forecast package to determine the order

of every ARIMA model. To make one-step-ahead predictions, we fit a new ARIMA model at each timestep. We train the HMLSTM, the CWRNN and the LSTMS directly against squared error between the signal and the prediction. We train the WRNN methods against squared error between the predicted SDWT coefficients and the raw SDWT coefficients.

5.1 Simulated Data Experiments

The data were generated using the generate package, written specifically for this work and described further in section 6.3. The package generates data that share randomly generated structure across multiple time scales. We believe this makes it a useful dataset to test the WRNN on, as one of our goals is to be able to learn precisely such structure. In our first experiment, we compare the quality of one-step-ahead predictions on simluated data. These results are listed in table 5.1.

Next, we add gaussian noise to the data and evaluate one-step-ahead predictions on the noisy data. Table 5.2 shows results for gaussian noise with standard deviation equal to 1, table 5.3 shows results for standard deviation equal to 5.

We find it instructive to look at examples of how these models tend to predict. Figure 5.1 shows the one-step-ahead predictions of each method evaluated in this chapter on an out of sample simulated signal.

Table 5.1: One-Step-Ahead Evaluation

method	loss
lstm	0.314930
wrnn_multi	0.460940
wrnn_single	2.176626
hmlstm	0.310478
cwrnn	0.294125
arima	0.304199

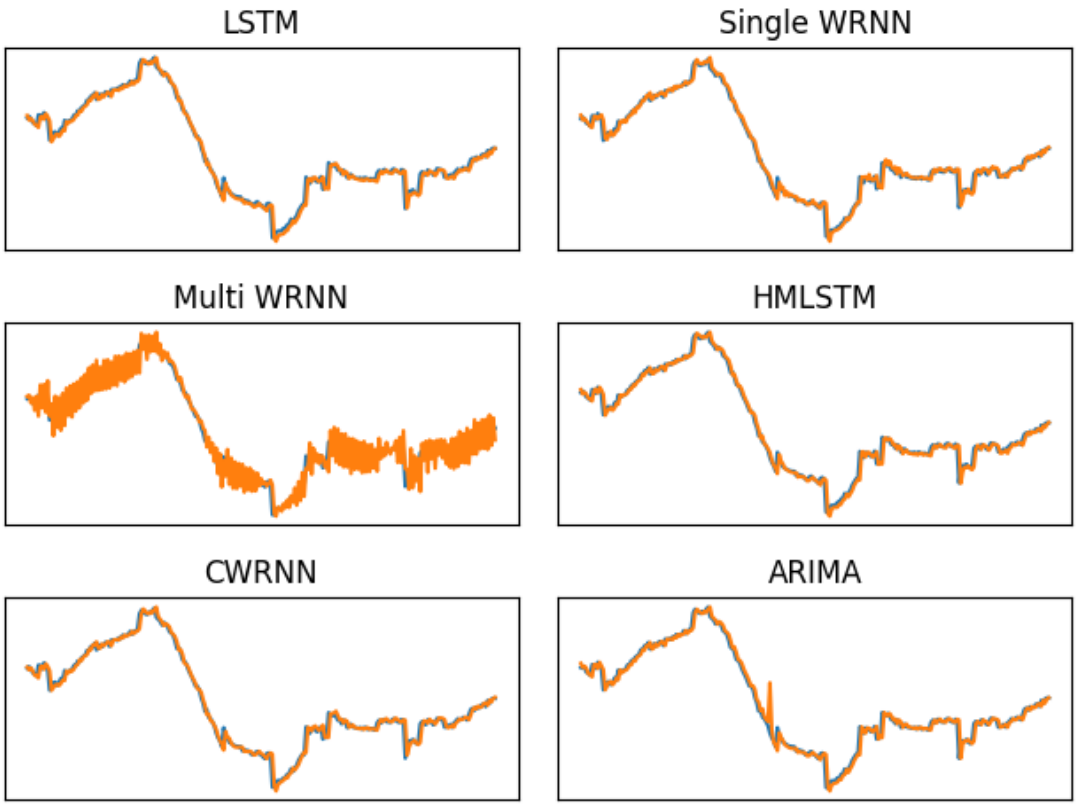
Mean squared error of one step ahead prediction on simulated data.

Table 5.2: One-Step-Ahead Evaluation, Noisy data (standard deviation = 1)

method	loss
lstm	1.314859
wrnn_multi	1.637971
wrnn_single	2.737518
hmlstm	1.387885
cwrnn	1.290917
arima	0.859301

Mean squared error on one step ahead prediction on simulated data with gaussian noise.

Figure 5.1: One-step-ahead predictions on simulated interval data



This figure shows the one-step-ahead predictions of each method for a randomly chosen signal from the test set. The predictions are in orange, and the true signal is in blue. Each method follows the signal closely, with the exception of the Multi WRNN, which makes noisy predictions.

Table 5.3: One-Step-Ahead Evaluation, Noisy data (standard deviation = 5)

method	loss
lstm	18.868434
wrnn_multi	13.059073
wrnn_single	9.189421
hmlstm	29.632380
cwrnn	21.201608
arima	5.185149

Mean squared error on one step ahead prediction on simulated data with gaussian noise.

5.2 Real data Experiments

We also tested our method on cardiological data, downloaded from physionet [24]. The data was originally collected as part of the CAST [25] trial. This trial was designed to test the effects of three different treatments of ventricular premature complexes. High-frequency heart rate information was collected from most participants both before treatment and after treatment. The raw data takes the form of an annotated ECG for each patient.

For our purposes, we are intersted in the RR interval. The RR interval is the distance between two consecutive R peaks in an ECG. The average number of RR intervals in any given minute is also know as the heart rate.

Table 5.4: RR-Interval One-Step-Ahead Evaluation

method	loss
lstm	25.082348
wrnn_multi	113.328871
wrnn_single	109.582777
hmlstm	24.320284
cwrnn	22.194518
arima	22.486587

Mean squared error on one step ahead prediction of heart rate data.

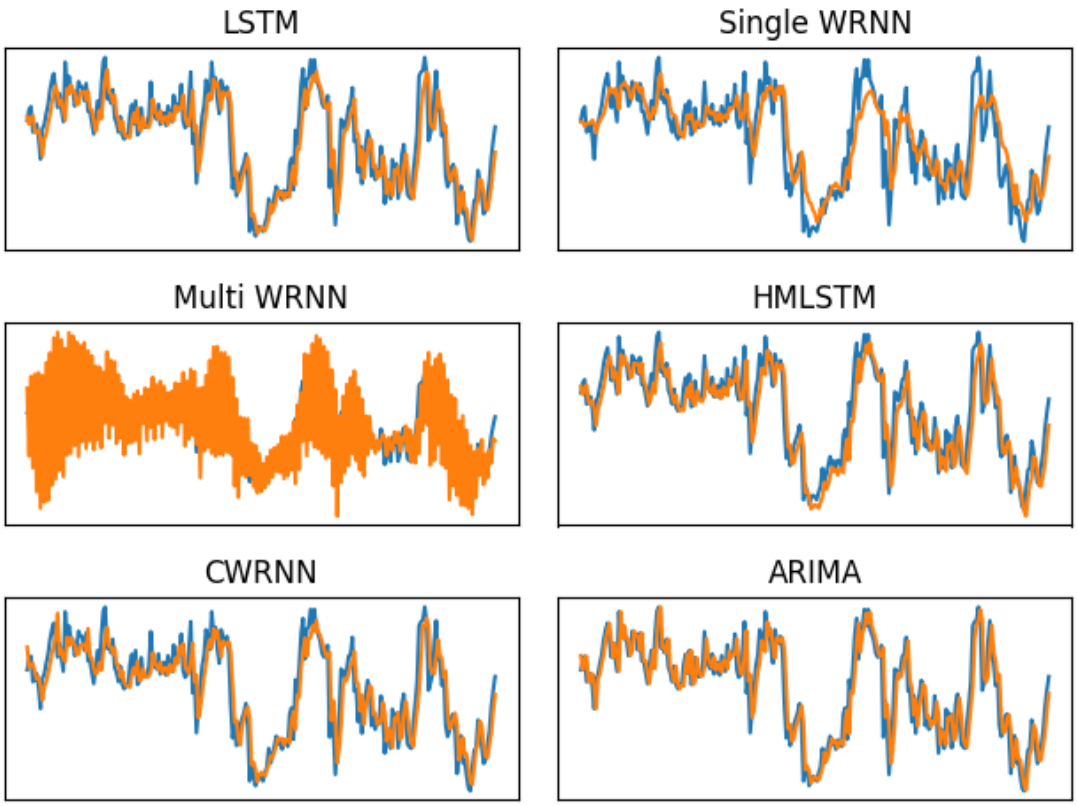
We used the ann2rr tool from the WFDB toolbox to extract RR interval information, and remove readings that were annotated as invalid. This gave us the heart rate corresponding to every RR interval recorded by the ECG.

For computational reasons, we were not able to use all the data. We downsampled the heart rate information by a factor of 16, effectively averaging heartrate over every 16 beats. We also were only able to use approximately information from approximately 4800 beats per patient.

We used 80% of the data for training each model and 20% for evaluation purposes.

In figure 5.2 the HMLSTM and CWRNN predictions seem quite similar. In particular, they don't tend to hit the short-term highs and lows. They also both seem to be hedging a little against the upwards trend by staying below it, especially at the

Figure 5.2: One-step-ahead predictions on RR interval data



One-step-ahead predictions from each method are shown in orange, overlaid on the true signal in blue.

tail end of the signal, which makes sense because most samples in this data do not have an upwards trend. The ARIMA predictions by contrast do tend to chase those high and low points more actively, which also results in more variability between timepoints.

We note that compared with other network architectures with similar complexity, the single network WRNN does not perform well. This is likely because the WRNN is predicting a length 6 vector instead of a scalar value.

Chapter 6

Software

Over the course of conducting research into the subject matter covered in this thesis, we developed several python libraries to facilitate experiments and conceptual exploration. These include warp, generate, dwtviz, hmlstm, sdwt, and wrnn. The first four of these are open source, and the last two will be open sourced at some time in the future, depending on whether and when the corresponding research is published.

In this chapter, we provide an overview of these packages. For more detailed documentation and source code, please view the github repositories.

6.1 warp

This package provides an array of functions that manipulate functions. The important functions in the package are called 'elongated', 'compress', 'add_noise', 'functional_scale', and 'functional_warp'.

Each of these accepts a function that is to be manipulated, along with some parameters to describe that manipulation, and returns a new function. The function passed as a parameter must accept a single parameter, which should be a number or a numpy vector, and returns the same.

In addition, this package includes a number of convenience functions for constructing parameterized distortion functions, and inverting arbitrary distortion and scaling functions.

6.2 dwtviz

This package provides an array of visualization tools for the discrete wavelet transform, the stationary wavelet transform, and the sequential discrete wavelet transform.

For examples of the kinds of visualizations this library can generate, please see section 1.1.1.

6.3 Signal Generation

The generate package provides an easy way to simulate random signals with shared structure at multiple resolutions across signals.

This function is built upon the warp 6.1 package. It generates some number of 'motifs', which are functions with finite, random support. These represent the shared structure amongst the generated signals. Each motif has some distribution over 'scale', i.e. how long that motif can last in a signal, and 'amplitude'. Each signal is created by optionally designating each timestep as the start of one of the underlying motifs. Once a motif is so designated, the scale and amplitude of that instance of the motif are determined by sampling from the appropriate distributions. Additionally, a distortion function is randomly sampled for each motif instance.

The user can determine how often motifs begin in the generated signals and how much noise is added to the signals.

The 'generate_data' function is available if the user is interested in viewing the underlying motifs and the distributional parameters that determine the distributions from which the scale, amplitude and distortion functions are sampled.

The 'generate_signals' function is available if the user is only interested in the genearted signals.

6.4 hmstlm

This package is an implementation of the hierarchical multiscale long short term memory network, described in [11]. The package also includes utility functions for preprocessing data, and utility functions for visualizing the predictions made and boundaries detected by the network.

This package is built on the tensorflow [26] framework.

6.5 sdwt

The sdwt package performs the SDWT described in chapter 2 on any signal, to any level.

6.6 wrnn

The wrnn package implements both the single network and multinetwork versions of the WRNN model described in chapter 3. Additionally, it implements many of the ideas described in chapter 4 on interpretability. All the influence plots in that chapter were generated by the wrnn package.

This package uses the autograd package for training the wrnn, and also for calculating gradients to use in the influence plots. There is also a tensorflow [26] version available for more computationally intensive tasks, where multiple cores or a GPU

might be needed. However influence plots can not be derived for models trained using the tensorflow implementation.

Chapter 7

Conclusion and Next Steps

In summary, we develop WRNN as a strategy to exploit multiresolution structure in time series data, and develop tools to aid with model interpretability in forecasting settings.

We find that the WRNN is competative with state of the art neural networks in the one-step-ahead forecasting task. We note, however, that the WRNN can be especially tricky to train, due to the fact that it increases the problem from one dimension to $J + 1$ dimensions.

We view this work as a starting point for further exploration of the multiresolution structure present in medical time series; there are many avenues by which it could be expanded. First, we are interested in enriching the SDWT itself. In particular, we are intersted in exploring ways to be more selective in which resolutions are included in the transform, rather than mandatorily included every resolution 2^j for $j \in \{1, ..., J\}$.

This would also give us scope to expand the scope of each DWT in the SDWT without incurring undue computational penalties.

Future work with the question of model interpretability in forecasting will entail refining the set of tools offered, and considering the effects of the influence plot explanations on human interactions with models. We are interested in applying statistical methods to these gradients to characterize the structure underlying the data in some way. For example, we may be able to cluster the kinds of subsignals that are important at different resolutions into a set of 'motifs'.

Future work with the WRNN can proceed in several directions. First, we would like to extend the method to multi-step-ahead prediction. We are hopeful that its multiresolution nature will lend itself well to making appropriate predictions as the forecasting horizon is extended.

Additionally, we are interested in expanding the WRNN to allow for classifying time series, and for predicting the class that a timeseries will take on in the future. In the medical setting, we might be interested in classifying the patient whose data we are looking at as diseased or healthy. We have several ideas about how to incorporate classification in the WRNN architecture. Early experiments indicate that feeding each layer's LSTM hidden state at each timepoint into a feed forward classifying neural network may work well. This architecture borrows from ideas in [11].

We would like to expand the model to a multi-signal setting in a principled manner. In the medical domain, a single patient produces many streams of data, some of which

are highly correlated because they are reflective of the same underlying systems. We are hopeful that we can extend the WRNN idea to capture multiresolution structure across signals.

Bibliography

[1] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way.* Elsevier Science, 2008.

[2] M. S. Crouse, R. D. Nowak, and R. G. Baraniuk, "Wavelet-based statistical signal processing using hidden markov models," 1998.

[3] D. Fugal, *Conceptual Wavelets in Digital Signal Processing: An In-depth, Practical Approach for the Non-mathematician.* Space & Signals Technical Pub., 2009.

[4] J. S. Morris and R. J. Carroll, "Wavelet-based functional mixed models," *B*, pp. 179–199, 2006.

[5] S. Ray and B. Mallick, "Functional clustering by bayesian wavelet methods," *Journal of the Royal Statistical Society B*, vol. 68, pp. 305–332, 2006.

[6] I. Johnstone and B. W. Silverman, "Wavelet threshold estimators for data with correlated noise," 1997.

BIBLIOGRAPHY

[7] L. Ma and J. Soriano, "Efficient functional ANOVA through wavelet-domain Markov groves," *ArXiv e-prints*, Feb. 2016.

[8] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 2, pp. 107–116, Apr. 1998. [Online]. Available: http://dx.doi.org/10.1142/S0218488598000094

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[10] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork rnn," pp. 1863–1871, 2014. [Online]. Available: http://jmlr.org/proceedings/papers/v32/koutnik14.pdf

[11] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," 2016.

[12] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," 2016. [Online]. Available: https://arxiv.org/abs/1609.03499

[13] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet

representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674–693, 1989.

[14] S. Mallat, "Group invariant scattering," *Communications on Pure and Applied Mathematics*, 2012.

[15] S. Saria, A. Duchi, and D. Koller, "Discovering deformable motifs in continous time series data," *International Joint Conference on Artificial Intelligence*, 2011.

[16] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," pp. 1135–1144, 2016.

[17] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," 2016.

[18] Z. C. Lipton, "The mythos of model interpretability," 2016.

[19] Y. Zhao, R. T. Ogden, and P. T. Reiss, "Wavelet-based lasso in functional linear regression," *Journal of Computational and Graphical Statistics*, vol. 21, no. 3, pp. 600–617, 2012.

[20] G. Nason and B. Silverman, "The stationary wavelet transform and some statistical applications," *Wavelets & Statistics: Lecture Notes in Statistics*, 1995, publisher: Springer-Verlag Other: Eds. A.Antoniadis & G.Oppenheim.

[21] B. Kim, R. Khanna, and S. Koyejo, "Examples are not enough, learn to criticize! criticism for interpretability," *Advances in Neural Information Processing Systems*, 2016.

[22] T. D. Team, "Theano: A python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[23] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: the forecast package for R," *Journal of Statistical Software*, vol. 26, no. 3, pp. 1–22, 2008. [Online]. Available: http://www.jstatsoft.org/article/view/v027i03

[24] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000 (June 13), circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

[25] P. Stein, P. Domitrovish, K. Schechtman, and J. Rottman, "Clinical and demographic determinants of heart rate variability in patients post myocardial infarction: insights from the cardiac arrhythmia suppression trial (cast)," *Clinical Cardiology*, 2000.

BIBLIOGRAPHY

[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

Vita

Noam Finkelstein was born in New York City on 19 July 1990. Following his graduation from Tenafly High School in 2008, he attended Deep Springs College, a junior college in California, where he studied philosophy and literature. He completed his undergraduate education at Mansfield College, Oxford University, from which he graduated with a degree in Politics, Philosophy and Economics in 2013. Following several years working as a software developer and research engineer, he enrolled in the ScM program in the department of biostatistics at Johns Hopkins University in the fall of 2016. While enrolled in the program he conducted research on computational methods in clinical medicine. In the fall of 2017, he will enroll in the PhD program in the department of computer science at Johns Hopkins University.