UNSUPERVISED ANOMALY DETECTION IN SEQUENCES USING LONG SHORT TERM MEMORY RECURRENT NEURAL NETWORKS

by

Majid S. alDosari

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computational Science

Committee:

Kirk D Borne	Dr. Kirk D. Borne, Thesis Director
Estela Blaisten	Dr. Estela Blaisten-Barojas, Committee Chair
Igor Griva	Dr. Igor Griva, Committee Member
Kevin M. Curtin	Dr. Kevin Curtin (acting), Department Chair
Donna Fox	Dr. Donna Fox, Associate Dean, Office of Student Affairs & Special Programs, College of Science
P. Agouris	Dr. Peggy Agouris, Dean, College of Science

Date: 4/20/2016

Spring Semester 2016
George Mason University
Fairfax, VA

Unsupervised Anomaly Detection in Sequences Using Long Short Term Memory Recurrent Neural Networks

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Majid S. alDosari
Bachelor of Science
Vanderbilt University, 2003
Master of Science
Vanderbilt University, 2012

Director: Dr. Kirk D. Borne, Professor
Department of Computational and Data Sciences

Spring Semester 2016
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my father, Saad F. al-Dosari, who supported me in this endeavor.

Acknowledgments

I appreciate Dr. Borne's lead, as well as encouraging enthusiasm, in this endeavor. Similarly, I am grateful to my other committee members, Dr. Blaisten-Barojas and Dr. Griva, who have given their time and input so that I may successfully complete my thesis. Also, I give special thanks to John Kaufhold of Deep Learning Analytics for being responsive to my questions regarding anything related to neural networks. His interest in and support of my work motivated me to do the best job that I can. Last but not least, I appreciate very much that Leif Johnson, author of the Theanets neural network package that I used, provided assistance beyond user support.

Table of Contents

List of Tables

List of Figures

Abstract

UNSUPERVISED ANOMALY DETECTION IN SEQUENCES USING LONG SHORT TERM MEMORY RECURRENT NEURAL NETWORKS

Majid S. alDosari

George Mason University, 2016

Thesis Director: Dr. Kirk D. Borne

Long Short Term Memory (LSTM) recurrent neural networks (RNNs) are evaluated for their potential to generically detect anomalies in sequences. First, anomaly detection techniques are surveyed at a high level so that their shortcomings are exposed. The shortcomings are mainly their inflexibility in the use of a context 'window' size and/or their suboptimal performance in handling sequences. Furthermore, high-performing techniques for sequences are usually associated with their respective knowledge domains. After discussing these shortcomings, RNNs are exposed mathematically as generic sequence modelers that can handle sequences of arbitrary length. From there, results from experiments using RNNs show their ability to detect anomalies in a set of test sequences. The test sequences had different types of anomalies and unique normal behavior. Given the characteristics of the test data, it was concluded that the RNNs were not only able to generically distinguish rare values in the data (out of context) but were also able to generically distinguish abnormal *patterns* (in context).

In addition to the anomaly detection work, a solution for reproducing computational research is described. The solution addresses reproducing compute applications based on Docker container technology as well as automating the infrastructure that runs the applications. By design, the solution allows the researcher to seamlessly transition from local (test) application execution to remote (production) execution because little distinction is made between local and remote execution. Such flexibility and automation allows the researcher to be more confident of results and more productive, especially when dealing with multiple machines.

Chapter 1: Introduction

In the modern world, large amounts of time series[①] data of various types are recorded. Inexpensive and compact instrumentation and storage allows various types of processes to be recorded. For example, human activity being recorded includes physiological signals, automotive traffic, website navigation activity, and communication network traffic. Other kinds of data are captured from instrumentation in industrial processes, automobiles, space probes, telescopes, geological formations, oceans, power lines, and residential thermostats. Furthermore, the data can be machine generated for diagnostic purposes such as web server logs, system startup logs, and satellite status logs.

Increasingly, these data are being analyzed. Inexpensive and ubiquitous networking has allowed the data to be transmitted for processing. At the same time, ubiquitous computing has allowed the data to be processed at the location of capture.

While the data can be recorded for historical purposes, much value can be obtained from finding anomalous data. However, it is challenging to manually analyze large and varied quantities of data to find anomalies. Even if a procedure can be developed for one type of data, it usually cannot be applied to another type of data.

Hence, the problem that is addressed can be stated as follows: find anomalous points in an arbitrary (unlabeled) sequence. So, a solution must use the same procedure to analyze different types of time series data.

The solution presented here comes from an unsupervised use of recurrent neural networks. A literature search only readily gives two similar solutions. In the acoustics domain, [1]

[①] In this document, the terms 'time series' and 'sequence' are used interchangeably without implication to the discussion. Strictly however, a time series is a sequence of time-indexed elements. So a sequence is the more general object. As such, the term 'sequence' is used when a general context is more applicable. Furthermore, the terms do not imply that the data are real, discrete, or symbolic. However, literature frequently uses the terms 'time series' and 'sequence' for real and symbolic data respectively. Here, the term 'time series' was used to emphasize that much data is recorded from monitoring devices which implies that a timestamp is associated with each data point.

transform audio signals into a sequence of spectral features which are then input to a denoising recurrent autoencoder. Improving on this, [2] use recurrent neural networks (directly) without the use of features (that are specific to a problem domain, like acoustics) to multiple domains.

This work closely resembles [2] but presenting a single, highly-automated procedure that applies to many domains is emphasized. First, some background is given on anomaly detection[②] that explains the challenges of finding a solution. Second, recurrent neural networks are introduced as general sequence modelers. Then, experiments will be presented to show that recurrent neural networks can find different types of anomalies in multiple domains. Finally, concluding remarks are given.

[②]Outlier, surprise, novelty, and deviation detection are alternative names used in literature.

Chapter 2: The Challenge of Anomaly Detection in Sequences

2.1 Introduction

The goal of this chapter is to show that the solution to the general problem of anomaly detection in time series is difficult. A typical general framework for anomaly detection in time series is explained with two advanced solutions as examples and their issues. The proposed solution, explained later in Chapter 5, fits into the framework providing a basis for comparison.

Describing the variety of solutions puts the difficulty of finding a general solution in context. Furthermore, few publications survey the problem of anomaly detection for time series in particular [3, 4].

Solutions have been fragmented across a variety of application domains such as communication networks [5–15], economics [16–18], environmental science [11, 19–27], industrial process [28–31], biology [32, 33], astronomy [32, 34], and transportation [35, 36]. The fragmentation of application domains led to a variety of problem formulations [4]. Furthermore, there is no good understanding of how the solutions in different application domains compare to each other [3]. Therefore, it is difficult to generalize the performance of a solution formulated in one application domain to its performance in another although they might have some commonalities.

Furthermore, adding to the difficulty of comparing solutions, the mechanics of anomaly detection have come from two disciplines with different priorities: statistics and computer science. Solutions from statistics focus on mathematical rigor while solutions from computer science consider computational issues [4].

[4] offers a survey of anomaly detection solutions in a variety of settings such as streaming data, distributed data, and databases. But to focus the solution presented here, the problem will be stated as follows: Given a finite time series $\boldsymbol{x}$,

$$\boldsymbol{x} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(T)}\}$$

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^v,$$

find points which can be considered anomalous where T is the length of the regularly spaced sequence in time[①], t, and v is the number of variables of $\boldsymbol{x}$. This statement is sensible only when anomalous points are a small part of the data. Furthermore, anomalies may not even be present.

So elements of any solution to this problem must answer the following questions:

1. What is normal (as an anomaly is defined as what is *not* normal)?
2. What measure is used to indicate how anomalous point(s) are?
3. How is the measure tested to decide if it is anomalous?

2.2 Anomaly Types

The presence of different anomaly types can be a challenge for anomaly detection techniques. What follow are qualitative descriptions of anomalies classified in a way most relevant to this work. However, a taxonomy of anomalies will never encompass all anomalies as well as defining anomalies as points that are not normal.

2.2.1 Point Anomalies

Point anomalies are single points of interest that can be classified as follows.

[①]Irregularly-spaced sequences might need a specific treatment. Of course, $\boldsymbol{x}$ can be treated as a sequence not indexed by time provided that some coherence is revealed over the index.

Simple: Simple point anomalies are just defined by their value. They are trivial to describe and detect. They are not of much interest in themselves but are mentioned because anomalies in more complicated time series can be 'converted' to resemble this simple type.

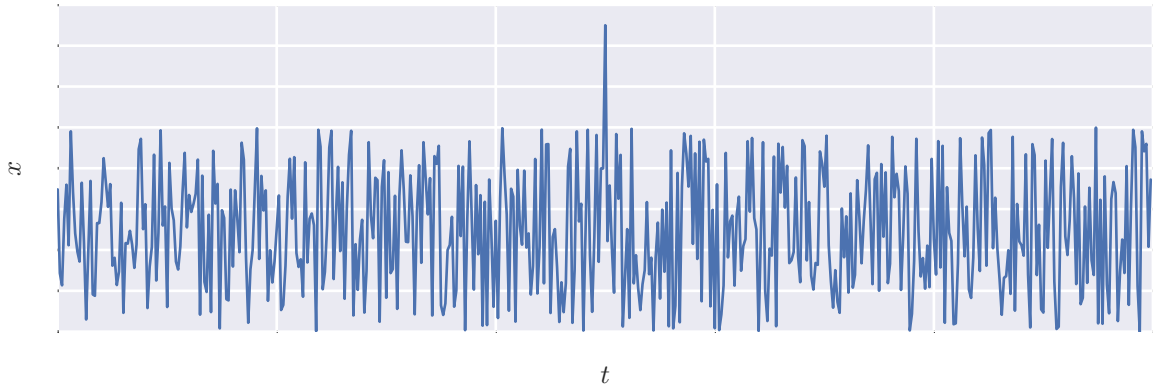

Figure 2.1: Simple point anomaly

Context: Some point anomalies are defined within a context. In Figure 2.2, the anomalous point's value is within the range of typical values. But if the cyclic nature of the series were removed, the anomaly is readily detected ('converted' to a simple point anomaly).

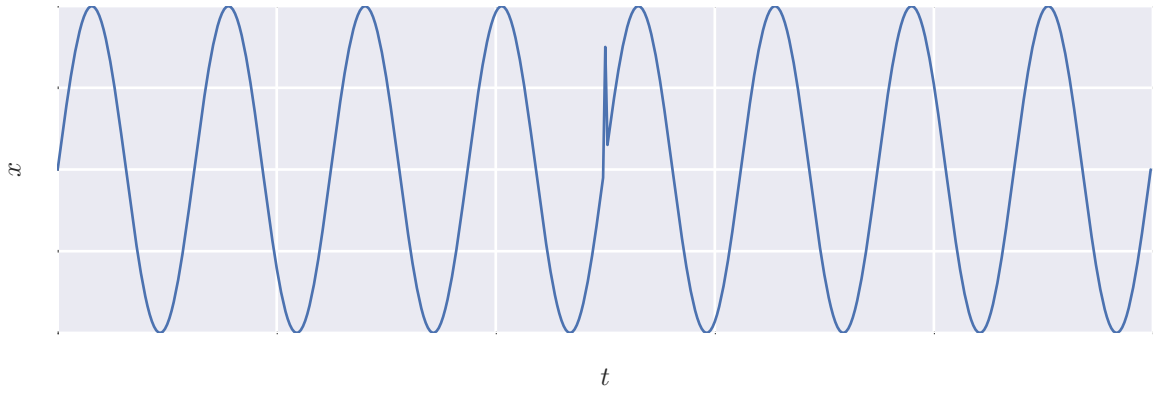

Figure 2.2: Anomaly in a periodic context

2.2.2 Discord

Anomalies over subsequences are called discords [3]. In Figure 2.3, about two cycles in a periodic time series are unlike the other cycles. The repeated units do not have to be periodic as in Figure 2.4.

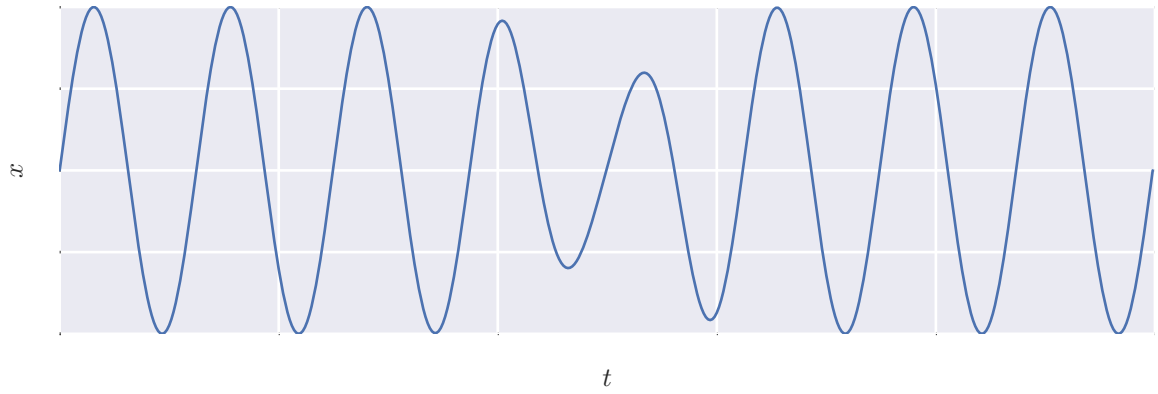

Figure 2.3: Discord anomaly in a periodic time series

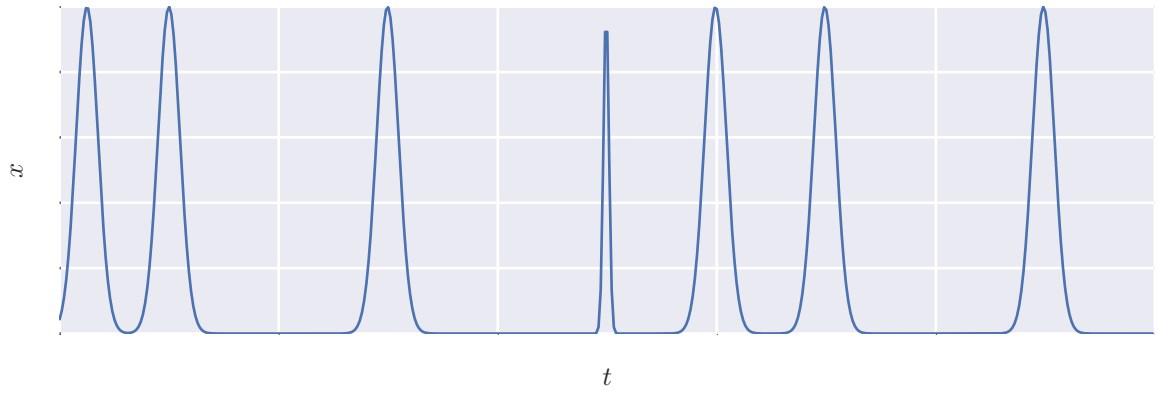

Figure 2.4: Discord anomaly in an aperiodic time series

2.2.3 Multivariate

Multivariate[2] time series add another element of complication for detecting anomalies in them and are not a focus in this work. [3] classifies multivariate time series according to a combination of periodicity and synchronicity: Variables in a time series can be synchronous and periodic, synchronous and aperiodic, asynchronous and periodic, asynchronous and aperiodic. Therefore, any deviation from these properties are anomalies.

2.3 Procedure

In this section, a procedure to find anomalies in time series will be outlined to help answer the questions posed in the introduction of this chapter. The focus will be on issues related to time series as opposed to the general anomaly detection problem. However, there will be a focus on the issues related to solving the general problem of finding anomalies in (an arbitrary) time series as opposed to finding anomalies in a particular application domain. But computational issues will not be emphasized.

By examining many detection techniques, a general procedure (Section 2.7 in [3]) can be gleaned:

1. Compute an anomaly score for an observation (a point or a subsequence in a time series). The anomaly score is a deviation from some 'normal' value such as the prediction from a model or similarity to other observations.

2. Aggregate the anomaly scores for many observations.

3. Use the anomaly scores to determine whether an observation can be considered anomalous. For example, an observation could be considered anomalous if its anomaly score exceeds three standard deviations from the mean of the anomaly scores.

[2] In this text, dimensionality of the time series will refer to the length of the time series as opposed to the number of its variables. Time series anomaly detection literature is inconsistent in the terminology used to refer to these two attributes.

At a high level, the procedure is rather straightforward. But, the process of finding what is normal is not. This section is devoted to explaining why finding anomalies in time series is particularly challenging. Addressing the stated problem, given an (single) arbitrary time series, the process of finding normal behavior involves the following steps:

1. Extract samples
2. Transform samples
3. Apply detection technique

2.3.1 Sample Extraction

Many instances of data are needed to inform what is normal. So, subsequences need to be extracted from a (long) sequence.

Samples can be extracted from a sliding a window over the time series. More precisely, beginning at step $t = 0$, sliding a window of width w over a time series x of length T one step at a time produces $p = T - w + 1$ windows, $\mathcal{X} = \{W_1, W_2, \ldots, W_p\}$[3].

Now, $\mathcal{X}$ contains all possible subsequences of $\boldsymbol{x}$ of length w and the value of p is typically not much less than T. Having many subsequences helps to localize the anomaly; so the window capturing the anomaly will have a higher anomaly score than adjacent windows.

But sometimes it is not desirable for computational reasons to process all subsequences. p can be reduced by introducing a 'hop', h, that skips h steps when advancing the window from the previous one ($h = 1$ gives all possible subsequences).

However, by introducing a large enough hop, anomalies could be missed. Consider the sequence *abccabcabc*. The second c is an anomaly. Now inspect the windows generated by various values of h in Table 2.1. The anomalous c is captured in a window when h is 1 or 2 but not when h is 3 or 4. As a general rule, when $h = 1$, an anomaly would never be missed. But when $h > 1$, there is a chance that anomalies would be missed.

[3] In literature, this form corresponds to a time series database [4]

Table 2.1: Point anomalies in sliding windows of width 3 for various hop sizes for the sequence $abc\underline{c}abcabc$ are underlined (The c in the first cab for $h = 1$ cannot be considered anomalous because it is not in its context). From [3].

hop (h)	Ordered Windows
1	abc, $bc\underline{c}$, $c\underline{c}a$, cab, abc, bca, cab, abc
2	abc, $c\underline{c}a$, abc, cab
3	abc, cab, cab
4	abc, abc

Another issue that needs to be considered when working with windows is that the window size must be large enough to capture an anomaly. Consider the sequence $aaabbbccc\underline{c}aaabbbcccaaabbbccc$ where the fourth c is anomalous. The window width must be at least 4 to capture the fourth c. In other words, the window size must be on the order of the anomalous behavior.

Now given $\mathcal{X}$ for some w, the problem may be posed as a multidimensional anomaly detection problem. In this setting, assuming that $\boldsymbol{x}$ is univariate, samples of $\mathcal{X}$ correspond to w dimensions. However, doing so largely ignores the temporal nature of $\boldsymbol{x}$. These issues will be discussed within the forthcoming parts of this chapter.

Finally, subsequent steps taken in the anomaly detection process may put restrictions on h and w. For example, if the window size is too large, there may not be enough samples to properly apply an anomaly detection technique.

2.3.2 Transformation

Anomalies can be more easily detected if the time series are analyzed in a different representation. Usually these representations are of lower fidelity but capture the essential characteristics of the time series in certain cases. As a general example, a real-valued time series could be or should be discretized into a finite set of symbols or numbers to make use of techniques from bioinformatics. Or, it could be transformed into a different domain such as the frequency domain to make use of techniques from signal processing. As an added benefit, the tranformed time series need less computation given their reduced representation.

More specifically, the Symbolic Aggregate approXimation (SAX) [37] is an example of time series discretization used to find anomalies in [32]. While the Haar transform represents a transformation to the frequency domain for the same purpose in [31, 38].

However, as a transformation only captures the essential characteristics of a time series, more subtle anomalies could be lost in the transformation process. For example, the anomaly in Figure 2.2 would be difficult to encode in terms of frequency because it is localized in time while oscillations (representing frequencies) are not.

Another issue to consider is the similarity of the arrangment of time series 'points' (like elements of $\mathcal{X}$ in $\mathbb{R}^w$) in the transformed space to that of the original space. Some anomaly detection techniques rely on a certain distribution of points in a space. Suppose an anomaly detection technique works in $\mathbb{R}^w$ by identifying points that are far away from some normal cluster of points. This arrangement should also be present in the transformed space. Anomaly detection techniques that rely on spaces are introduced in Section 3.1.

However, a recent empirical study [39] suggests that, in general, there is little to differentiate between numerous time series representations. Only spectral transformations applied to periodic series showed some advantage but only in certain cases.

2.3.3 Detection Technique

Choosing the anomaly detection technique is the final (involved) step in setting up an anomaly detection process. Detection techniques are discussed as a separate chapter to allow for more development.

Chapter 3: Detection Technique

The application of anomaly detection in a wide variety of domains has led to the development of numerous detection techniques. Each of these applications defines anomalies in a different way; some may only be interested in single anomalous points while others are more concerned about anomalous subsquences (discords). Furthermore, techniques are developed drawing on theory from statistics, machine learning, data mining, information theory, and spectral theory.

A highest-level categorization of these techniques could be as follows. The categories are not exhaustive but capture a wide variety of techniques discussed in literature.

Segmentation In segmentation-based techniques, the time series is first split into homogeneous segments. Then a finite-state automation is trained to learn transition probabilities between segments. So, a segmented anomalous time series should not have high transition probability [40–42].

Information Theory Information-theoretic techniques quantify a notion of information content such as entropy. So, a point is considered an anomaly if its removal reduces the information content significantly [43, 44]. That is, anomalous points increase disorder, or require more information to be represented in the sequence.

Proximity Techniques based on proximity map time series onto a space. It is expected that anomalous time series are 'different' because they are far from normal ones.

Model The difference between the (actual) values of a time series and its predicted values from a model indicate how anomalous they are.

Given the variety of techniques applied in different application domains, it is not always possible to use a solution developed for one problem and apply it to another. Finding a

general anomaly detection technique is difficult. To the author's knowledge, only one study [3] attempted to compare anomaly detection techniques over a wide variety of data. The study showed, as expected, varying performance of the techniques. A combination of time series characteristics and algorithm settings were sometimes used as explanations for the varying performance. These explanations do not help to objectively determine *a priori* what technique to use and how to adjust any parameters it might use.

It would be difficult to use information theoretic techniques because finding an information theoretic measure sensitive enough to detect a few anomalies is challenging [45]. Segmentation-based techniques require that a time series to be made of homogeneous segments. These conditions are deemed too restrictive to be able to solve the general problem. In addition, both are not well-studied. So, they are not further explored here.

This leaves model-based techniques and proximity-based techniques as potential solution categories. Both are widely studied. Furthermore, it is possible to make a theoretical comparison between model-based techniques and proximity-based techniques if they are evaluated as, respectively, generative and discriminative models [46]. Model-based techniques are usually preferred for anomaly detection [47] assuming enough training data are available.

Obviously, a good model is needed as well; recurrent neural networks will be introduced in the next chapter. However, proximity-based solutions are explored in this chapter as a benchmark for comparison as they are well-studied and have had numerous successful applications. Also, the hidden Markov model is introduced as an example of model-based solutions in this chapter as another benchmark.

3.1 Proximity

As previously mentioned, proximity-based techniques map time series as points of dimension w in some space using some distance measure. The distance measure is used to evaluate how close a test time series is to others; anomalous time series are those that are far (dissimilar) from those considered normal.

This implies that the time series 'points' are arranged in a certain way in the space. In

two dimensions, the simplest distribution is portrayed in Figure 3.1. Normal points, $\mathcal{N}_1$, are somewhat clustered. To test whether p_1 is an anomaly, it is easy to see, and calculate, that point p_1's nearest neighbor is larger than the nearest neighbor distances of all other points.

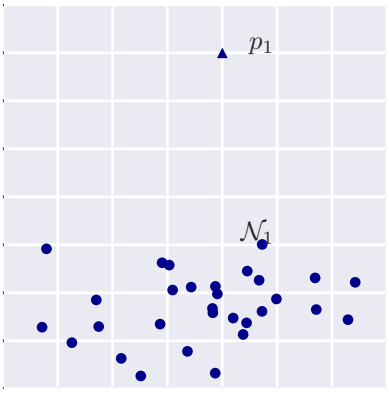

Figure 3.1: Simple anomaly distribution

Practically, this idealization never occurs. It is not as simple to objectively distinguish p_1 and p_2 from $\mathcal{N}_1$ and $\mathcal{N}_2$ in the situations depicted in Figure 3.2.

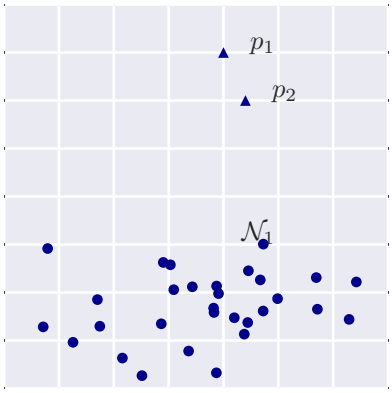

Figure 3.2 (a)

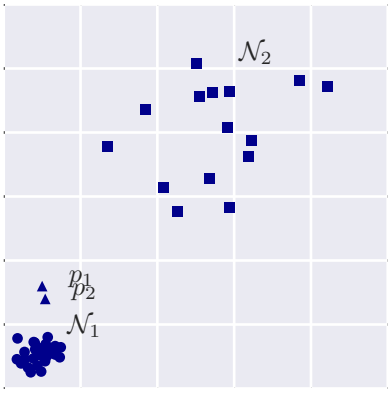

Figure 3.2 (b)

Figure 3.2: Complex anomaly distribution

3.1.1 Effects on Point Distribution

Having complex distributions is the purview of anomaly detection in general and not a problem particular to time series data. However, issues influenced by the temporal nature of the data will be the focus of this subsection.

Distance Measure

The choice of distance measure is crucial for anomaly detection because it captures some intuitive notion of similarity between a pair of time series. It is possible to use the 'ordinary' Eucilidean distance [48] but it does not always capture the desired similarity between a pair of time series. The comparison between time series should be invariant to the following factors:

Length

Sometimes it is desirable to be able to compare sequences of different lengths. Compare the sequence, *abababababab*, to a shorter one, *abababa*. There is reason to believe that both are at least highly similar.

Skew

Consider the sequence *abaababaaabaabab* and *aaababaababaabab* where occurrences of *b* are padded by an unpredictable number of *a*'s. The sequences can be considered similar even if they are not strictly periodic.

Dynamic Time Warping (DTW) [48] is capable of identifying similarity in such scenarios by 'warping' the sequences to minimize dissimilarity (DTW is also capable of comparing sequences of different lengths).

Translation

Typically, sequences that are shifted in time are considered similar. For example, *abababab* is similar to *bababababa*. Despite the sophistication of DTW, a warping transformation would not make these two sequences similar. Cross-correlation is a similarity measure that can handle translations [49].

Amplitude

Sometimes it is desirable to ignore differences in amplitude if two time series are otherwise similar. Distance measures are usually sensitive to differences in amplitude unless the time series are normalized [48].

So simple measures, like the Euclidean distance, are not able to accommodate these factors. However, recently, in a comprehensive study of similarity measures, [39] finds that, in large data sets, the clustering accuracy of 'elastic' measures, such as DTW, converges with that of the Euclidean distance. Nonetheless, the advantage of using DTW, as well as other elastic measures, is maintained for smaller data sets.

Furthermore, in addition to addressing the previously mentioned factors, the distance measure must be compatible with the time series representation [50]. So, for example, if a symbolic representation is used, then the distance measure must be appropriate for symbolic data.

Window Width

Section 2.3.1 introduced the use of sliding windows to extract samples of some width[①], w. Here, the effect of the choice of w will be explained when combined with a distance measure.

Also in Section 2.3.1, it was mentioned that w should be, at least, on the order of the length of the expected anomaly. But when w is too large the distance measure becomes less effective at measuring similarity because the proportion of the anomalous subsequence becomes small compared to w. Furthermore, pairs of points become more equidistant in high-dimensional space [51, 52] which makes distinguishing between time series difficult (though this could be mitigated by using a dimension-reducing transformation [53]).

Sliding Windows

The mere action of extracting samples with a *sliding* window (with a small h, in contrast to non-overlapping windows) can have a profound effect on the distribution of the samples

[①]Window 'size' or 'length' might also be used to refer to window width

in the space. The effects challenge assumptions about the organization of (sliding window) points for the task of anomaly detection.

[32] demonstrates that anomalous points are not necessarily located in sparse space. Conversely, repeated patterns are not necessarily located in dense space [54–56].

Furthermore, in a paper challenging the status quo, [57] makes the bold claim that clustering of time series with sliding windows is meaningless. It found no similarity between clusters found with a sliding window ($h << w$) and clusters found when the windows did not overlap ($h > w$) which invalidated the results of previously published papers per its argument. However, some constraints were identified that may allow some data sets to be clustered. But a decade after the paper's publication, the issue of clustering subsequences of time series is unresolved [58].

These findings place restrictions on selecting a proximity-based anomaly detection algorithm.

3.1.2 Data Classification

In the following subsections, data classification techniques (that facilitate anomaly detection) will be discussed. The techniques are not particular to time series data. However, finding anomalous windows of time series data is referred to as 'discord detection' in literature.

Local versus Global Techniques

First, it is instructive to make a general note about how data classification techniques work before they are categorized.

Proximity-based anomaly detection techniques can use a global context (of all) data points or a local context to test whether a point is an anomaly. Going back to Figure 3.1, it was mentioned that it is trivial to find the anomalous point using all data points. However, in Figure 3.2(b), it is not trivial to distinguish p_1 and p_2 as anomalous in this global view due to their proximity to $\mathcal{N}_1$. An anomaly detection approach that considers the (local) neighborhood of points p_1 and p_2 must be used.

3.1.3 Nearest-Neighbor

Nearest-neighbor techniques assume that normal points are in dense neighborhoods while anomalies are far from them.

In the beginning of Section 3.1, it was noted how simple it is to determine that point p_1 in Figure 3.1 is anomalous. However, given the discussion about sliding windows in Section 3.1.1, it is not as straightforward when dealing with data points generated from sliding windows.

The problem comes from adjacent windows being close to each other in space. Consider the windows of width equal to 3 for the sequence *abcabcXXXabcababc* [32] in Table 3.1. Obviously, the subsequence *XXX* stands out as the most anomalous after *bab*. But *XXX* has the nearest matches *cXX* and *XXa* one step away as can be seen in the subsequences generated by advancing a window on every step ($h = 1$). This makes the subsequence *bab* as more anomalous because it is different by two symbols from its nearest match *aba*. In contrast, when examining windows with no overlap ($h = 3$), *XXX* is far from all other subsequences.

Table 3.1: Neighbors of sliding windows of sequence *abcabcXXXabcababc*

$h = 1$	$h = 3$
abc *bca* *cab*	*abc*
abc *bcX* *cXX*	*abc*
XXX *XXa* *Xab*	*XXX*
abc *bca* *cab*	*abc*
aba *bab* *abc*	*aba*

However, it is still possible to use overlapping windows to find subsequences like XXX. The solution proposed by [32] is to use non-self matches. So when when calculating distances (similarity) to neighbors of XXX, only those that are at least one window width away are considered. In this calculation, XXX would not have any neighbors with the symbol X.

Furthermore, the solution in [32] improves upon the brute force discord search by employing a heuristic called HOT SAX: Heuristically Ordered Time series using Symbolic Aggregate approXimation. But the time series is discretized as part of its process.

If the issues with sliding windows can be mitigated, it is possible to use anomaly detection techniques that are not specific to time series. Nearest-neighbor techniques can use either the kth nearest neighbor (kNN) distance or the relative density of a region around a point in the calculation of its anomaly score.

For example, the kNN distance can be used to identify that p_1 and p_2 are anomalies in Figure 3.2(a). By choosing, say, $k = 4$, the fourth nearest neighbor of points p_1 and p_2 would be a point in *far* away in $\mathcal{N}_1$. While the fourth nearest neighbor for a point in $\mathcal{N}_1$ is always a *nearby* point within $\mathcal{N}_1$.

The HOT SAX technique was introduced as, what is essentially, a $k = 1$ kNN technique. But it can be generalized to an arbitrary k [34, 59].

However, kNN techniques cannot deal with data sets of varying density. Going back to Figure 3.2(a), if $\mathcal{N}_1$ were denser, so that distances between points in $\mathcal{N}_1$ were less than the distance between p_1 and p_2, then k can only equal to 2. Generally, k needs to be equal to the number of anomalous points as long as the clustering of $\mathcal{N}_1$ is denser than the clustering of p_k. But since some distances between between pairs of points in $\mathcal{N}_1$ are similar to the distance between p_1 and p_2, k has to be greater than 2. Furthermore, a further complication can be considered if another normal set of points, $\mathcal{N}_2$, is included as in Figure 3.2(b). Here, it is not simple to find a value of k that would not distinguish p_1 and p_2 due to the varying densities in $\mathcal{N}_1$ and $\mathcal{N}_2$.

But by using local density information, techniques based on the relative density of a point's region can mitigate this problem. For example, [60] defines a Local Outlier Factor

(LOF) for a point that is the ratio of the average local density of its k nearest neighbors to the local density of the point itself. The local density is defined as k divided by the volume of the smallest hypersphere that contains k nearest neighbors. In other words, a point is likely to be an anomaly if its neighbors are in dense regions while it is in a less dense region. Applied to Figure 3.2(b), the LOF for points in $\mathcal{N}_1$ and $\mathcal{N}_2$ is similar. But for points p_1 and p_2, if two nearest neighbors are considered, the second nearest neighbor would be the closest point in $\mathcal{N}_1$ where its local density is high compared to the local density of p_1 and p_2 resulting in a LOF that is higher than the LOF of points in $\mathcal{N}_1$ and $\mathcal{N}_2$. This is true for any $k > 2$.

In any case, nearest neighbor techniques rely on normal points being in denser regions. But given the discussion about the effect of sliding windows in Section 3.1.1, it is not clear how these techniques would be generally applicable to time series data.

[45] provides a more thorough treatment for general data types.

3.1.4 Clustering

Clustering algorithms are used to group similar points. Typically they are not used to find anomalies but they can be adapted to do so if some assumptions are made.

One assumption that could be made is that normal points belong to clusters while anomalies do not. This requires algorithms that do not force all points to belong to a cluster such as the well-known DBSCAN [61] algorithm where deviant points are considered noise. Still, such algorithms are optimized to find clusters. So anomalous points could incorrectly be included in a cluster.

Another assumption that could be made resembles nearest neighbor techniques; anomalous points are far from their nearest cluster's centroid while normal points are not. Any clustering algorithm could be used such as K-means [62]. However, this assumption would misclassify anomalous points that form clusters themselves.

A third assumption could be that normal points are in large and dense clusters while anomalous points are in small and sparse clusters. In similar fashion to LOF, described

previously, a Cluster-Based Local Outlier Factor (CBLOF) was introduced in [63]. But, again, given the discussion about sliding windows in Section 3.1.1, it is not clear how these techniques would be generally applicable to time series data.

[45] provides a more thorough treatment for general data types.

3.2 Models

Instead of comparing a point to other points, like in proximity-based solutions, a point can be compared to what is expected from a model.

For example, Hidden Markov Models (HMM) are advanced sequence modelers and therefore can be used for anomaly detection [64–67]. To use in anomaly detection, first, a HMM maximizes the probability of a set of training data. Then the probability of a test instance is calculated for comparison. But the comparison is only meaningful if the training data can be modeled by the hidden Markovian process.

Moreover, sequence modelers typically require training examples of fixed length which restricts the 'memory' of the model.

3.3 Conclusions

Some general comments can be made regarding the strengths and weaknesses of general anomaly detection techniques such as those in Section 11 of [45] for example. However this does not provide any rigorous and objective evaluation. An objective evaluation is needed in order to select the most appropriate algorithm for the problem *a prioi*. In [3] and [68] some qualitative explanation is given for the performance of several time series anomaly detection techniques over various data sets. Still, this does not objectively determine why one technique performs better.

Exceptionally however, [69] outlines a theoretical framework for local outlier detection in which different methods are assessed in this common view. Perhaps surprisingly, different methods were found to be similar including those used in quite different application domains.

Similarly, [70] unify anomalies identified based on some distance with anomalies identified based on a statistical model.

For the techniques discussed in Section 3.1, a number of issues specific to time series data (as well as issues not specific to time series data) combine in the process of using a proximity-based (discriminative) technique for anomaly discussion. The choices of similarity measure, sliding window width, sliding window hop, and classification technique must be compatible. Furthermore, defining a region for every normal behavior is difficult to begin with.

So the case is made for generative model-based solutions due to these complications. Models provide a better summary of data which gives it a robustness that is preferred for the task of anomaly detection [47]. Given this preference, proximity-based methods would only be preferable if modeling the time series is difficult (disregarding computational issues).

This work attempts to find anomalies in an arbitrary time series. So, an anomaly detection process is sought that:

- models arbitrary time series,
- minimizes the effects of window width,
- and requires as few parameters as possible.

This guides the discussion in the next two chapters. The next chapter shows how recurrent neural networks are general sequence modelers. In the following chapter, a procedure is outlined that addresses window width and model parameters.

Chapter 4: Recurrent Neural Networks

4.1 Introduction

In the previous chapter, an argument was made for the use of models to for the purpose of anomaly detection. In this chapter, recurrent neural networks [71], or RNNs, are introduced as powerful sequence modelers of arbitrary length.

RNNs have achieved state-of-the-art performance in supervised tasks such as handwriting recognition [72], and speech recognition [73]. RNNs can also be trained unsupervised to generate music [74], text [75, 76], and handwriting [76]. Given these impressive feats, RNNs can be effectively used for anomaly detection in sequences [1, 2].

Like HMMs, RNNs have hidden states. But, in contrast to HMMs, RNNs make a more efficient use of its hidden state [77]. In a HMM, the hidden state depends only on the previous state which must store possible configurations of the sequence. So to incorporate information about an increasing window of previous states, the hidden state must grow exponentially large making them computationally impractical. While in RNNs, the state is shared over time; the hidden state of a RNN contains information from an arbitrarily long window. The next section will explain this.

In addition, RNNs do not make a Markovian assumption. In fact, they are so general, that RNNs have been shown to be equivalent to finite state automata [78], equivalent to turing machines [79], and more powerful than turing machines [80] even.

In the next few sections, some essential concepts of RNNs will be presented as most applicable to this work. Consult the RNN chapter in [81] for an in-depth treatment.

The precision from mathematical expressions will be used to understand how RNNs operate but only concepts will be presented. To follow the expressions precisely, some conventions are followed: 1) Due to the numerous interacting quantities, a typographic

distinction is made between vectors ($\boldsymbol{v}$) and matrices ($\mathbf{M}$). 2) Variables are time indexed with superscripts enclosed in parentheses ($\boldsymbol{x}^{(t)}$) leaving subscripts available to index other quantities.

4.2 Recurrence

Recurrence explains how a RNN stores a distributed state. Consider a dynamical system driven by signal $\boldsymbol{x}^{(t)}$ as in Equation 4.1.

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \tag{4.1}$$

The state, $\boldsymbol{s}^{(t)}$, depends on the previous state, $\boldsymbol{s}^{(t-1)}$, through some function f parameterized by $\boldsymbol{\theta}$. There is no restriction on the number of previous time steps. For example, for four previous time steps

$$\boldsymbol{s}^{(t)} = f(f(f(f(\boldsymbol{s}^{(t-4)}, \boldsymbol{x}^{(t-3)}; \boldsymbol{\theta}), \boldsymbol{x}^{(t-2)}; \boldsymbol{\theta}), \boldsymbol{x}^{(t-1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).$$

So the composite function, g, can be written as depending on an arbitrary number of time steps, T.

$$\boldsymbol{s}^{(T)} = g_T(\boldsymbol{x}^{(T)}, \boldsymbol{x}^{(T-1)}, \boldsymbol{x}^{(T-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$$

In other words, the vector $\boldsymbol{s}^{(T)}$ contains a summary of the of the preceding sequence, $(\boldsymbol{x}^{(T)}, \boldsymbol{x}^{(T-1)}, \boldsymbol{x}^{(T-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$, through g_T.

It can be difficult to follow recurrent computations by looking at mathematical expressions. So recurrence can be graphically represented in two ways. One way, shown in Figure 4.1(a), shows the state feeding back into itself through its parameters representing Equation 4.1. The other way is to 'unfold' the recurrence in a flow graph as in Figure 4.1(b). Graphs offer a convenient way of organizing computations. The (unfolded) graph shows every hidden state, say $\boldsymbol{s}^{(t)}$, is dependent on the current input, $\boldsymbol{x}^{(t)}$, the previous state, $\boldsymbol{s}^{(t-1)}$, and (fixed)

parameters $\boldsymbol{\theta}$. So it should be obvious that $\boldsymbol{\theta}$ is shared over successive computations of $\boldsymbol{s}$.

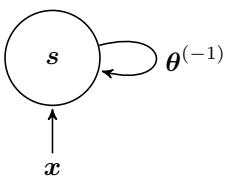

Figure 4.1 (a) cyclic

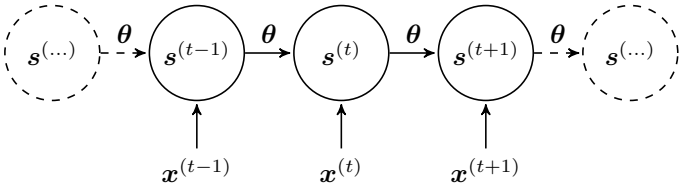

Figure 4.1 (b) acyclic (unfolded)

Figure 4.1: Recurrence graph views

4.3 Basic Recurrent Neural Network

The functionality of a basic RNN resembles that of a traditional neural network with the addition of a state variable shared over time.

The core RNN functionality can be formulated as Equations 4.2 describe. Each equation represents a 'layer' or, individually, in the scalar sense, 'nodes' in the network.

$$\boldsymbol{s}^{(t)} = \tanh(\mathbf{W}_{ss}\boldsymbol{s}^{(t-1)} + \mathbf{W}_{xs}\boldsymbol{x}^{(t)} + \boldsymbol{b}_s) \tag{4.2a}$$

$$\boldsymbol{o}^{(t)} = \mathbf{W}_{so}\boldsymbol{s}^{(t)} + \boldsymbol{b}_o \tag{4.2b}$$

The output at time t, $\boldsymbol{o}^{(t)}$, depends on current state, $\boldsymbol{s}^{(t)}$, which, in turn, depends on the previous state, $\boldsymbol{s}^{(t-1)}$, and the current input, $\boldsymbol{x}^{(t)}$, through associated weight matrices, $\mathbf{W}$. The weight matrices are subscripted with two variables to associate an input with an output. The input variable is the first subscript while the output variable is the second. For example, $\mathbf{W}_{so}$ connects the input from $\boldsymbol{s}$ to output, $\boldsymbol{o}$. Also, bias vectors, $\boldsymbol{b}$, allow values to be adjusted additively (offset). Finally, the hyperbolic tangent, tanh, provides a non-linearity, in the range (-1,1), that allows the RNN to summarize arbitrarily complex sequences.

A 'size' can be specified that measures the capacity of $\boldsymbol{s}$; the dimension of $\boldsymbol{s}$ is a

(free) parameter (obviously the dimensions of other quantities in Equation 4.2a need to be compatible). No size is associated with $\boldsymbol{o}$ because it is restricted by the dimension of a given output, $\boldsymbol{y}$ (read further).

The output, $\boldsymbol{o}$, needs to be compared to a 'target', $\boldsymbol{y}$, through a loss function, L. For predicting sequences as targets, the mean squared error is a common choice for the loss function. Squared differences are summed and averaged over and the number of variables of the sequence resulting in a scalar as shown in Equation 4.3.

$$L(\boldsymbol{o}, \boldsymbol{y}) = \frac{1}{TV} \sum_{t} \sum_{v} (\boldsymbol{o}_v^{(t)} - \boldsymbol{y}_v^{(t)})^2, \tag{4.3}$$

V and T are the number of variables and the length of the sequences respectively indexed by t and v respectively.

Equations 4.3 and 4.2, together, define the framework for a basic RNN. One way to enhance this is to 'stack' states so that the output from one state is input into another state in the same time step. So the last, lth, layer produces $\boldsymbol{o}$. There is evidence that stacking the state layers leads to the network being able to learn time series at multiple time scales [82, 83].

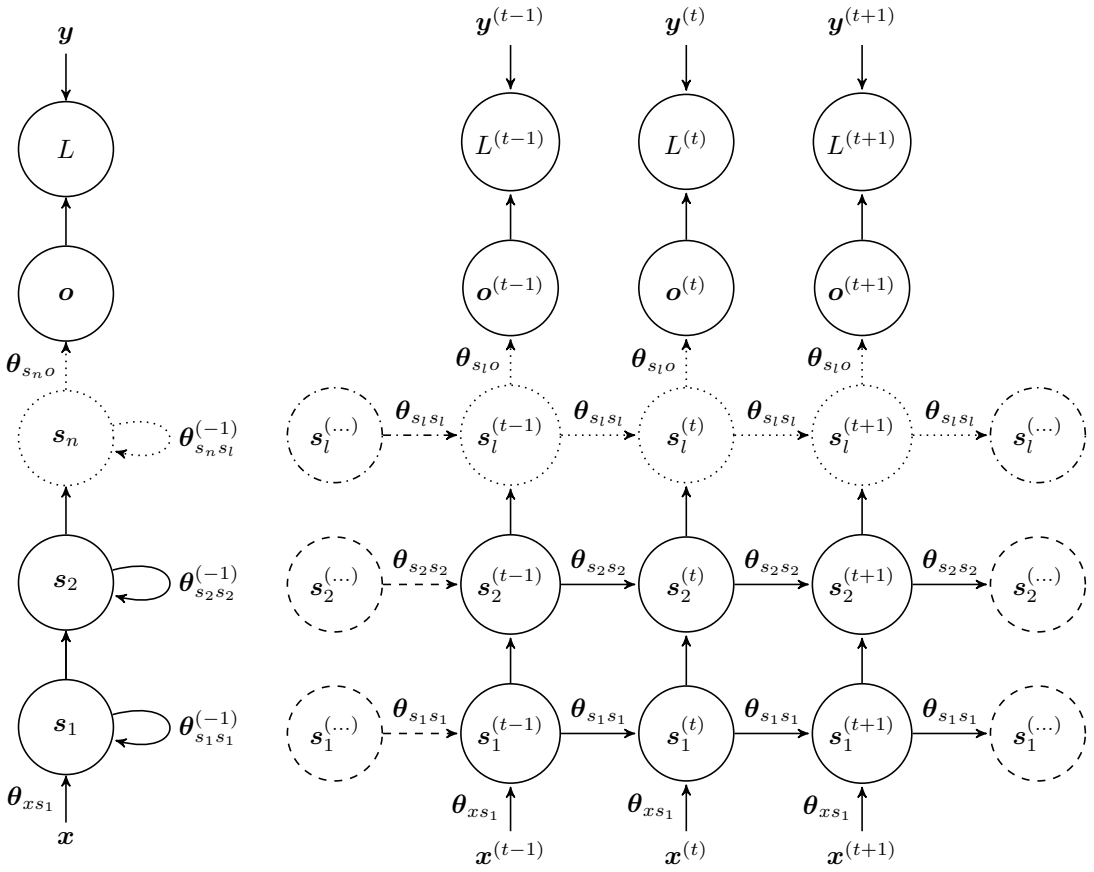

Figure 4.2 (a) cyclic

Figure 4.2 (b) acyclic

Figure 4.2: Recurrent neural network graph views. Bias terms are omitted for clarity but are included as elements in $\boldsymbol{\theta}$ along with elements from $\mathbf{W}$. The notation of $\boldsymbol{\theta}$ follows that of $\mathbf{W}$ but captures more variables by vectorizing each component for concatenation into a single vector. $\boldsymbol{\theta}_{ss} = (\boldsymbol{W}_{ss}, \boldsymbol{b}_s)$, $\boldsymbol{\theta}_{xs} = (\boldsymbol{W}_{xs})$, and $\boldsymbol{\theta}_{so} = (\boldsymbol{W}_{so}, \boldsymbol{b}_o)$ (operations to produce the vector are implied). Inter-layer parameters, $\boldsymbol{\theta}_{s_i s_{i+1}}$, are omitted as well to focus on operations in time (horizontally).

The stacking can be seen graphically in Figure 4.2. Notice that the unfolded view allows one to see that information can flow through many paths in time and layers (horizontally and vertically respectively). So 'depth' can be used to describe the number of operations in these two directions with depth in the time direction being typically much greater[①].

[①]This makes RNNs, among 'deep learning' architectures, the deepest networks!

4.4 Training

To train the network, the loss function is used to minimize (optimize) an objective function given pairs of training examples, $(\boldsymbol{x}, \boldsymbol{y})$, by adjusting the parameters, $\boldsymbol{\theta}$. Unfortunately, training neural networks in general, let alone recurrent neural networks, are challenging optimization and computational problems [84]. However, a flavor of the relatively simple mini-batch stochastic gradient descent (SGD) has remained successful in training a variety of networks [85] despite the availability of other, perhaps more sophisticated, optimization algorithms.

In plain mini-batch SGD, parameters are updated with a 'learning rate', α, for a selected 'mini-batch' example set (from the training set), M, according to Equation 4.4 until a convergence criterion is met.

$$\Delta\boldsymbol{\theta} = -\alpha \frac{1}{|M|} \sum_{(\boldsymbol{x}_m, \boldsymbol{y}_m) \in M} \frac{\partial L(\boldsymbol{o}, \boldsymbol{y}_m)}{\partial \boldsymbol{\theta}} \tag{4.4}$$

Unfortunately, RNNs can suffer acutely from the vanishing (and exploding) gradient problem which makes learning long range dependencies difficult [86–89].

The problem is understood through the calculation of $\partial L / \partial \boldsymbol{W}_{ss}$ for a 'history' of T points prior to t. An involved application of the (backpropagated) differential chain rule to the loss function using the *basic* (unfolded) RNN defined in Equations 4.2 leads to Equation 4.5.

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{ss}} = \sum_{i=0}^{T} \frac{\partial L^{(t)}}{\partial \boldsymbol{o}^{(t)}} \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{s}^{(t)}} \left(\prod_{j=i+1}^{T} \frac{\partial \boldsymbol{s}^{(j)}}{\partial \boldsymbol{s}^{(j-1)}} \right) \frac{\partial \boldsymbol{s}^{(i)}}{\partial \boldsymbol{W}_{ss}} \tag{4.5}$$

Equation 4.5 shows that there are multiplicative 'chains' of differentials for each preceding time step.

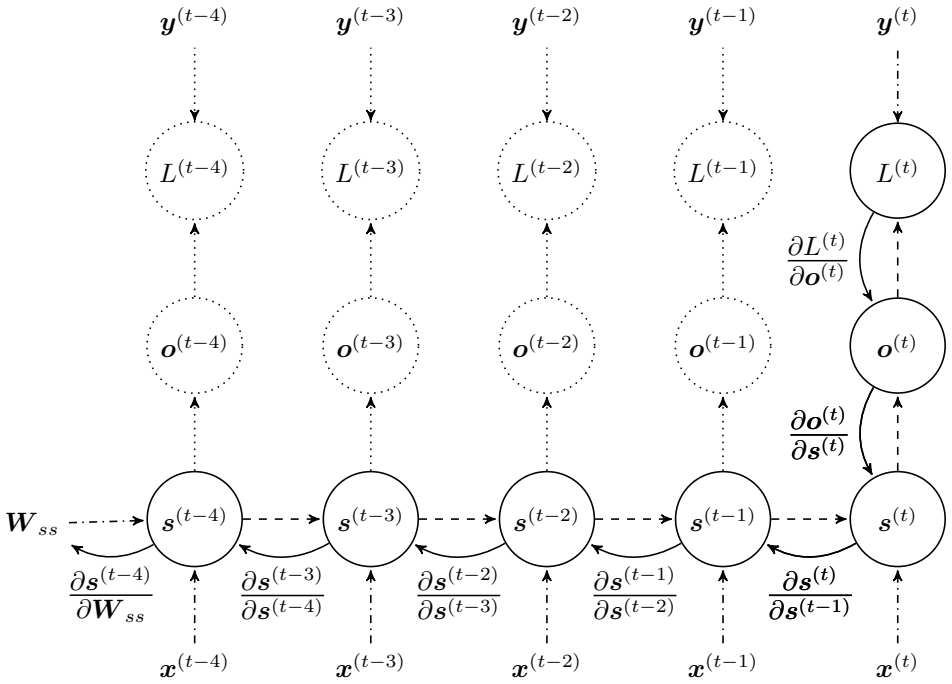

Figure 4.3: Partial derivative chain for a basic RNN for a history of 4 time steps (biases not shown). Several quantities are distinguished. Dash-dotted lines are the independent inputs, $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{W}_{ss}$, to the loss function, L, while the dashed lines represent intermediate quantities in the graph. The path formed by the curved solid lines follows the chain rule to get one summand for $\partial L^{(t)}/\partial \boldsymbol{W}_{ss}$. Dotted quantities are not involved in the calculation (for $T = 4$).

The computational graph in Figure 4.3 represents Equation 4.5 for $T = 4$. Now that the graph is associated with the equation, the origin of the vanishing gradient can be seen; the vanishing gradient is due to successive multiplications of the derivative of tanh which is bound in $(0, 1]$ (Again, see [86–89] for details).

[81] (sec. 10.7) lists ways in which the problem can be mitigated by applying different RNN architectures or enhancing the optimization process. For example, Hessian-free optimization [90] uses second order information that can make use of the ratio of small gradients to small curvature to more directly find optimal parameters. [91] uses Hessian-free optimization to train a basic RNN to achieve remarkable results. However, this comes at a computational cost. For practical reasons, it might be preferable to use well-established SGD-based methods

over second order methods [92, 93] as long as the RNN architecture facilitates learning long range dependencies; the Long Short Term Memory (LSTM) RNN [94] uses a 'memory cell' to recall (past) information only when needed thereby working around the vanishing gradient problem.

4.5 Long Short-Term Memory

LSTM models have achieved impressive benchmarks recently. Networks with LSTM elements have been used for text-to-speech synthesis [95], language identification from speech [96], large-vocabulary speech recognition [97], English-to-French text translation [98], identifying non-verbal cues from speech [99], Arabic handwriting recognition [100], image caption generation [101], video to text description [102], and generating realistic talking heads [103]. Furthermore, LSTM RNNs have remained competitive against other RNN types [104].

There are a few variants of LSTM but they all have a linear self-loop that gradients can flow through for a long time. LSTM variants have not shown to differ greatly in performance [105]. The equations describing the LSTM state involve significantly more steps than the basic RNN state equation, Equation 4.2. So, the mathematical procedure for the LSTM state is presented with a figure. The LSTM variant presented in Figure 4.4 is found in [76].

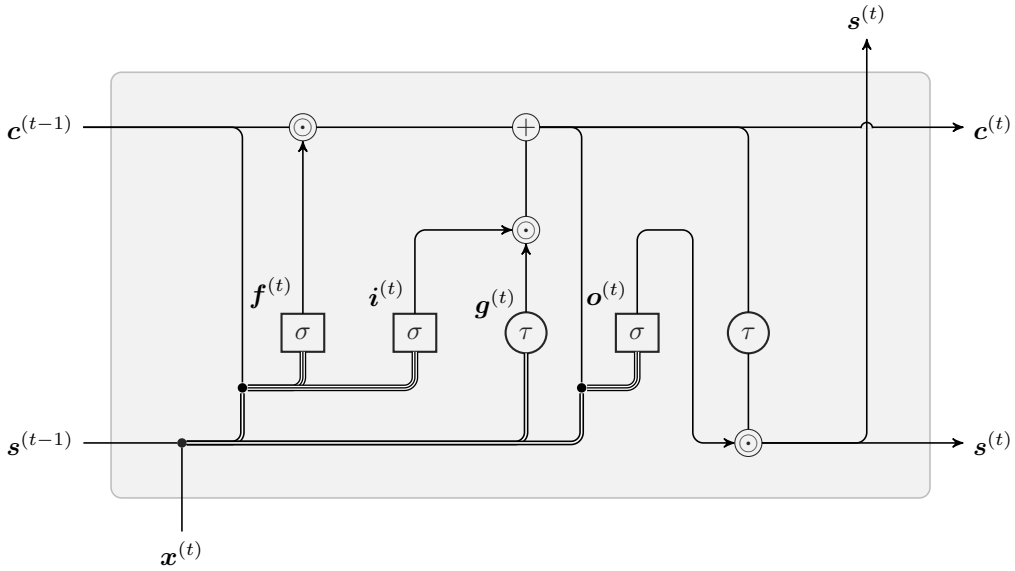

Figure 4.4 (a) calculation flow diagram (biases omitted, adapted from [106])

$$\boldsymbol{i}^{(t)} = \sigma(\mathbf{W}_i \cdot (\boldsymbol{x}^{(t)}, \boldsymbol{s}^{(t-1)}, \boldsymbol{c}^{(t-1)}) + \boldsymbol{b}_i) \tag{4.6a}$$

$$\boldsymbol{f}^{(t)} = \sigma(\mathbf{W}_f \cdot (\boldsymbol{x}^{(t)}, \boldsymbol{s}^{(t-1)}, \boldsymbol{c}^{(t-1)}) + \boldsymbol{b}_f) \tag{4.6b}$$

$$\boldsymbol{g}^{(t)} = \tanh(\mathbf{W}_c \cdot (\boldsymbol{x}^{(t)}, \boldsymbol{s}^{(t-1)}) + \boldsymbol{b}_c) \tag{4.6c}$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot \boldsymbol{g}^{(t)} \tag{4.6d}$$

$$\boldsymbol{o}^{(t)} = \sigma(\mathbf{W}_o \cdot (\boldsymbol{x}^{(t)}, \boldsymbol{s}^{(t-1)}, \boldsymbol{c}^{(t-1)}) + \boldsymbol{b}_o) \tag{4.6e}$$

$$\boldsymbol{s}^{(t)} = \boldsymbol{o}^{(t)} \odot \tanh(\boldsymbol{c}^{(t)}) \tag{4.6f}$$

Figure 4.4 (b) equations from [76]

Figure 4.4: Long Short-Term Memory layer. σ stands for the logistic sigmoid function while τ stands for tanh. Element-wise multiplication is represented by the $\odot$ symbol. For brevity and clarity, a (single) weight matrix combines the weights associated with more than one input to a single output (unlike the typical notation used in literature). For example, $\mathbf{W}_i \cdot (\boldsymbol{x}^{(t)}, \boldsymbol{s}^{(t-1)}, \boldsymbol{c}^{(t-1)}) = \mathbf{W}_{xi}\boldsymbol{x}^{(t)} + \mathbf{W}_{si}\boldsymbol{s}^{(t-1)} + \mathbf{W}_{ci}\boldsymbol{c}^{(t-1)}$ where the input vectors, $\boldsymbol{x}$, $\boldsymbol{s}$, and $\boldsymbol{c}$ are concatenated into a vector and weight matrices $\mathbf{W}_{xi}$, $\mathbf{W}_{si}$, and $\mathbf{W}_{ci}$ are augmented horizontally. In the diagram, the concatenation is represented by black dots combining single lines into more than one line. Furthermore, $\mathbf{W}_{ci}$, $\mathbf{W}_{cf}$, and $\mathbf{W}_{co}$ are diagonal matrices so the number of weights associated with each $\boldsymbol{c}$ equals the LSTM 'size'.

The size that can be attributed to the LSTM module is the dimension of $\boldsymbol{i}$, $\boldsymbol{f}$, $\boldsymbol{g}$, $\boldsymbol{c}$, $\boldsymbol{o}$,

or $\boldsymbol{s}$ (which are equal).

Very briefly, the central element in a LSTM module is the 'memory cell', $\boldsymbol{c}$, that works to keep information unchanged over many time steps (likened to a conveyor belt through time). Information flow is controlled through input, 'forget', and output 'gates', elements $\boldsymbol{i}$, $\boldsymbol{f}$, and $\boldsymbol{o}$ respectively. The input gate controls information into the memory cell. The output gate controls when information should flow to the next time step. The forget gate zeros out elements in the memory cell. The gates make their decisions based on the input, $\boldsymbol{x}$, the 'hidden' state, $\boldsymbol{s}$, and content already in the memory cell.

Unlike the recurrence of the basic RNN (Equation 4.2a), information can be kept for a long time in the recurrence chain of $\boldsymbol{c}$ (Equation 4.6d). There is a way for information to propagate without successive multiplication of fractions (as in the derivative of tanh): In Equation 4.6d, consider an element in (vector) $\boldsymbol{c}^{(t-1)}$ with a value of 1. It can be retained indefinitely as long as the corresponding element in $\boldsymbol{f}^{(t)}$ is 1 and the corresponding element in $\boldsymbol{i}^{(t)}$ is 0. In other words, the derivative is constant with respect to t.

Finally, note how the LSTM modules can be stacked (vertically) in similar fashion to the basic RNN. The output of the lower module, $\boldsymbol{s}^{(t)}$, becomes the input, $\boldsymbol{x}^{(t)}$, of the upper module.

With such configurations, the next chapter will show how LSTM RNNs can be used to model time series to find anomalies.

Chapter 5: Anomaly Detection Using Recurrent Neural Networks

5.1 Introduction

Chapters 2 and 4, separately, introduced anomaly detection in time series and recurrent neural networks as time series modelers. This chapter outlines and tests a procedure for finding anomalies using RNNs with the goal of mitigating many of the problems associated with anomaly detection that were discussed in Chapters 2 and 3. As much as possible, the same procedure is applied to each test time series to test the procedure's generality.

The procedure outline describes:

sampling: the time series used for training and its associated sampling process

recurrent autoencoders: the specific form of the RNN

training: the training algorithm used on the RNN

Bayesian optimization: the search for optimized parameters for the procedure

After describing the procedure, the anomaly scores of the test time series are evaluated for their ability to detect anomalies.

5.2 Sampling: Sliding Windows

Some univariate time series *with* anomalies were chosen or generated to test the anomaly detection process named as follows:

spike: two generated sequences that are simply evenly-spaced 'spikes' of the same height.

sine: a generated sinusoidal signal.

power demand: a building's electrical power usage data[①] referenced in the influencial HOT-SAX paper [32] (for anomaly detection in time series). The data was scaled down by dividing it by its median (to help in the training process).

electrocardiogram (ECG): a signal from the human heart taken from the PhysioNet [107] database. Finding anomalies in ECG data has been used as a benchmark in several studies at least [2, 3, 32, 108–111].

polysomnography ECG (PSG-ECG): human ECG data [107] taken during a sleep study[②].

The normal behaviour, as well as anomalous behaviour, of these time series can be seen in the plots presented in Section 5.5.

While there is only one 'source' time series to train on (in each test), the RNN needs to see many 'samples' from the source time series. The sliding window procedure, described in Section 2.3.1, is used to get samples of some window width. Additionally, since RNNs do not need a fixed window, sliding windows are obtained for more than one window width. Furthermore, the samples are collected into 'mini-batches' (of the same window width) to be more compatible with the training procedure. The window width is incremented (width skip) from some minimum until it is not possible to obtain a mini-batch (so the largest window will be close to the length of the time series).

Table 5.1 specifies the relevant sizes and increments for the sampling process. The values were adjusted manually until a 'reasonable' number of samples were found. However, the minimum window length was chosen such that meaningful dynamics were captured (regardless of the scale of the anomaly).

[①] `http://www.cs.ucr.edu/~eamonn/discords/power_data.txt`

[②] University College Dublin Sleep Apnea Database, `http://www.physionet.org/physiobank/database/ucddb/`, doi:10.13026/C26C7D, study id. ucddb002

Table 5.1: Time series sample specifications

series	length	min. win.	slide skip	width skip	batch size	⇒ **samples**
spike-1	800	100	20	20	10	51
spike-2	800	100	20	20	10	51
sine	1000	100	10	10	30	96
power	7008	100	100	20	30	252
ECG	3277	300	20	20	30	300
PSG-ECG	2560	300	20	20	30	165

5.3 RNN Setup: Autoencoder

The RNN needs to learn what normal time series behavior is. So an autoencoder is used which can learn expected behavior by setting the target, $\boldsymbol{y}$, to be the input, $\boldsymbol{x}$. The loss function is the MSE (Equation 4.3). Furthermore, to prevent the RNN from learning trivial identity functions, Gaussian noise is added to the input where the standard deviation is equal to 0.75 times the standard deviation of the whole time series.

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \mathcal{N}(0, (0.75\sigma_{\text{std}}(\boldsymbol{x}))^2)$$

Note the comparison in the loss function is between the uncorrupted signal, $\boldsymbol{x}$, and the output from the network, $\boldsymbol{o}$, given $\boldsymbol{x}$: $L(\boldsymbol{o}(\tilde{\boldsymbol{x}}), \boldsymbol{x})$.

With this setup, a denoising autoencoder, the data generating process, $p(\boldsymbol{x})$, is implicitly

learned [112].

5.4 Training: RMSprop

SGD with RMSprop [113] for parameter updates has been demonstrated to provide results that are similar to more sophisticated second-order methods but with significantly less computational cost [93]. Another benefit of RMSprop is that it is designed to work with mini-batch learning. Since the training data is highly redundant (from sliding windows), it is expected that computing the gradient (to update parameters) from a subset of the data (mini-batch) is more efficient than computing the gradient for all the data.

For each derivative in the gradient, RMSprop keeps an exponentially-weighted moving average of derivative magnitudes which normalizes the derivative by its root-mean-squared. More specifically, the (overall) RNN training procedure is outlined in the following box.

The Theanets [114] implementation[3] of RNNs and RMSprop was used. Theanets is based on the mathematical expression compiler, Theano [115]. Gradients were computed using Theano's automatic differentiation feature instead of explicitly-defined backpropagation [71].

While the training procedure optimizes $\boldsymbol{\theta}$, there are other parameters that could be adjusted to minimize the loss. The number of layers, l, and the 'size' of each layer, n, corresponding to the dimension of vector $\boldsymbol{s}$ (which equals the cell state memory dimension as well), were chosen as 'hyper-parameters' for optimization in a Bayesian optimization process. Bayesian optimization is suited for optimizing RNN training because 1) it tries to minimize the number of (expensive) objective function calls, which, in this case, is the training procedure, and 2) it considers the stochasticity of the function, which, in this case, the selection of training and validation data are random in addition to the training data shuffling on each epoch.

The Spearmint [116] package was used to drive the hyper-parameter optimization process. Spearmint takes a parameter search space and a maximum number of optimization iterations.

[3] With every calculation of $\boldsymbol{o}$, the RNN states are initialized to 0.

Training Procedure

{obtain mini-batches (as Section 5.2)}
$\mathcal{X} \leftarrow \text{sampling}(\boldsymbol{x})$

{randomly split mini-batches into training and validation sets}
$\mathcal{X}_t \leftarrow \text{choose}(75\%, \mathcal{X})$
$\mathcal{X}_v \leftarrow \mathcal{X} - \mathcal{X}_t$

{initialize}
$\boldsymbol{\theta} \leftarrow \mathbf{0}$ {initial RNN parameters are 0}
{set training parameters}
$\alpha = 10^{-4}$ {learning rate}
$h = 14$ {RMS 'halflife'}
$\gamma \leftarrow e^{\frac{-2}{h}}$
$r = 10^{-8}$ {RMS regularizer}
patience = 5 {stopping criterion parameter}
min_improvement = .005 {stopping criterion parameter}

for epoch i **do**
  $\mathcal{X}_t \leftarrow \text{shuffle}(\mathcal{X}_t)$
  for mini-batch M in $\mathcal{X}_t$ **do**
    for parameter p in $\boldsymbol{\theta}$ **do**
      {per-parameter, update p according to RMSprop [76] (indices to p omitted)}

$$f_{i+1} \leftarrow \gamma f_i + (1-\gamma)\frac{\partial L}{\partial p}$$

$$g_{i+1} \leftarrow \gamma g_i + (1-\gamma)\left(\frac{\partial L}{\partial p}\right)^2$$

$$p_{i+1} \leftarrow p_i - \frac{\alpha}{\sqrt{g_{t+1} - f_{t+1}^2 + r}}\frac{\partial L}{\partial p}$$

    end for
  end for
  {compute average loss on validation set}
  $v_i \leftarrow \frac{1}{|\mathcal{X}_v|}\sum_{\boldsymbol{x}_v \in \mathcal{X}_v} L(\boldsymbol{x}_v, \boldsymbol{o})$
  {stop when no improvement more than patience times}
  if $v_{min} - v_i > v_{min} \cdot \text{min\_improvement}$ **then**
    $v_{min} \leftarrow v_i$
    $i_{min} \leftarrow i$
  end if
  if $i - i_{min} >$ patience **then**
    STOP {return $\boldsymbol{\theta}$}
  end if
  $i \leftarrow i + 1$
end for

The process was programmed to save RNN parameters after every optimization iteration. However, the process was stopped until the RNN with the minimum validation loss was subjectively able to detect anomalies[④].

For every sample, Figure 5.1 shows the results of the Bayesian optimization process and the training progress of the best network. In the training progress figures, the validation and training losses are close throughout the training process. This is expected since the training and validation sets are highly redundant and perhaps the RNN, by nature, interprets them to be almost identical (invariance to translation and sequence length).

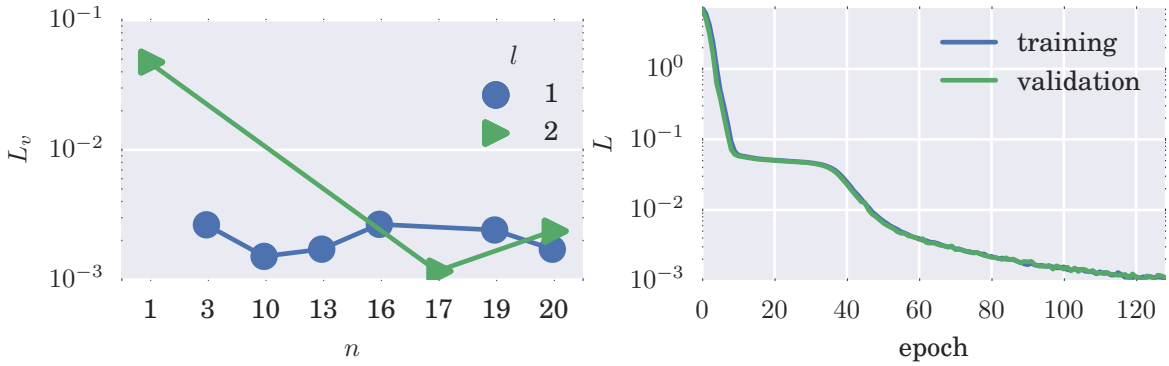

Figure 5.1 (a) spike-1

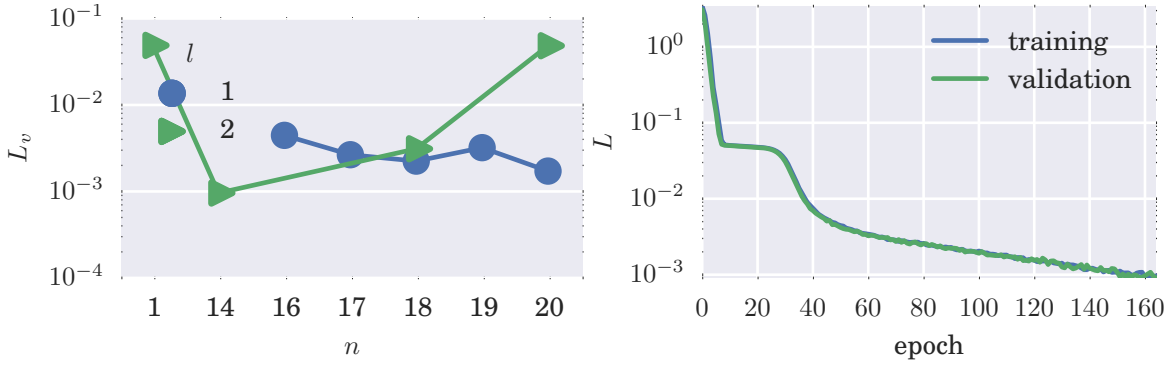

Figure 5.1 (b) spike-2

[④] Further optimization may have been possible with increasing RNN size but loss reduction diminished relative to the associated increase in computational expense.

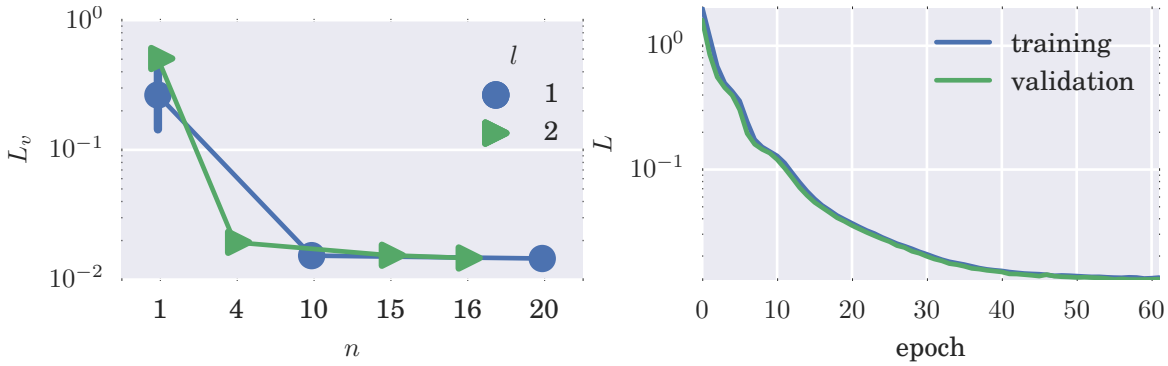

Figure 5.1 (c) sine

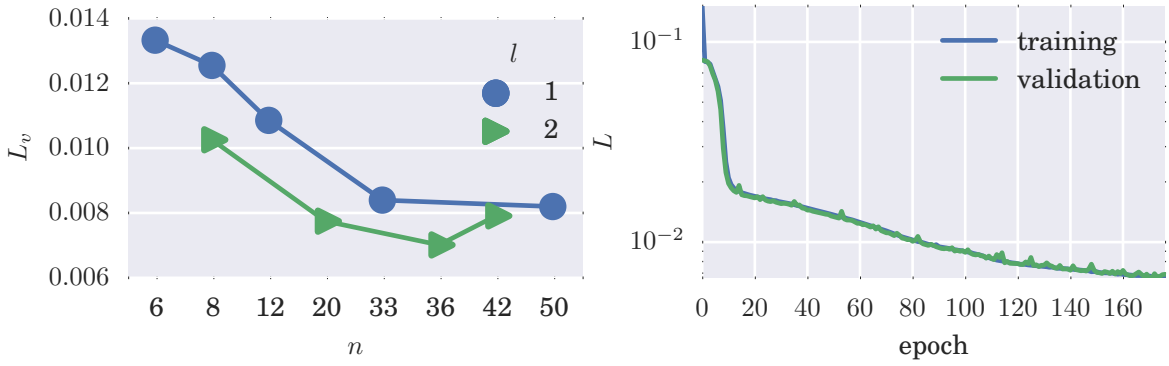

Figure 5.1 (d) power

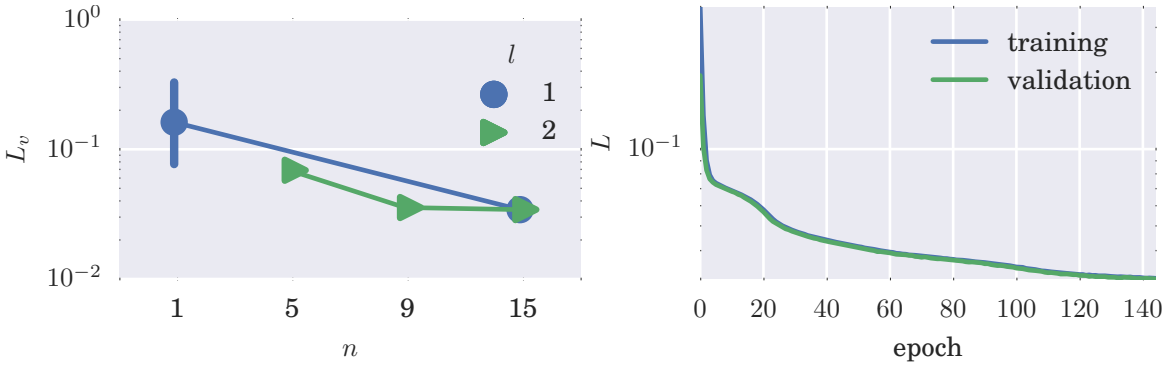

Figure 5.1 (e) ECG

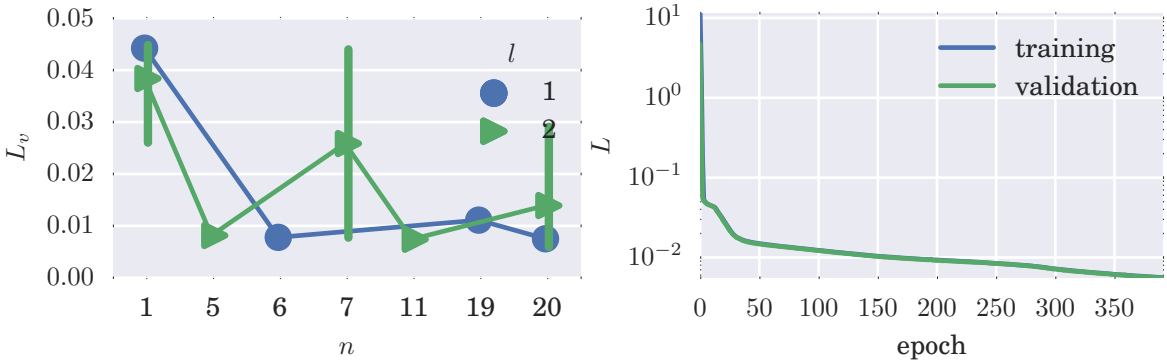

Figure 5.1 (f) PSG-ECG

Figure 5.1: Bayesian optimized validation loss (left) and training progress of best RNN (right). Each point in the Bayesian optimization plot is the average validation loss since more than on training session could have been launched with the same parameters. If so, the points have a 95% confidence interval bar through them.

5.5 Results and Discussion

The optimal RNN has some expectation of the typical series. So, a 'reconstruction error' is the difference between the test input and the output. Therefore, the square of the reconstruction error can be used as an anomaly score [117]. This section will show how the various calculations of error can be used to detect point anomalies *as well as* discord anomalies introduced in Section 2.2. However, the particular (objective) anomaly detection technique will not be discussed; only the anomaly score, that would be used in such a technique, is presented. Nonetheless, the argument is made for the ability of autoencoding RNNs to detect anomalies.

Although the order of the anomaly scores should not be important to the anomaly detection technique, sequences of error calculations can be made in two ways for a test series: 1) the MSE of a sliding window and 2) the squared error of every point in the test series where the whole sequence is input to the optimal RNN. While the MSE from a sliding window can detect both point anomalies and discords, it is preferable not to have to specify a window size. Ideally, high anomaly scores for points close together signal a discord while a lone high anomaly score signals a point anomaly. But both types of calculations are evaluated in this discussion.

Error calculations are presented graphically in Figure 5.2 for each series (see Figure caption after the last subfigure). The errors are summarized by their maximum, kernel density estimate, and the 5 percentile mark. Together, with the error plot, the efficacy of the RNN in distinguishing anomalies can be evaluated; normal data should correspond to the bulk of the error calculations while anomalies should correspond to extreme high values.

It should be noted that as the window size is increased, fractionally less points are included in the, normal, 95 percentile (and vice versa). The kernel density estimate might change significantly accordingly.

What follows is a discussion of the ability of the (optimal) RNN to detect anomalies referencing subfigures of Figure 5.2.

spikes-1

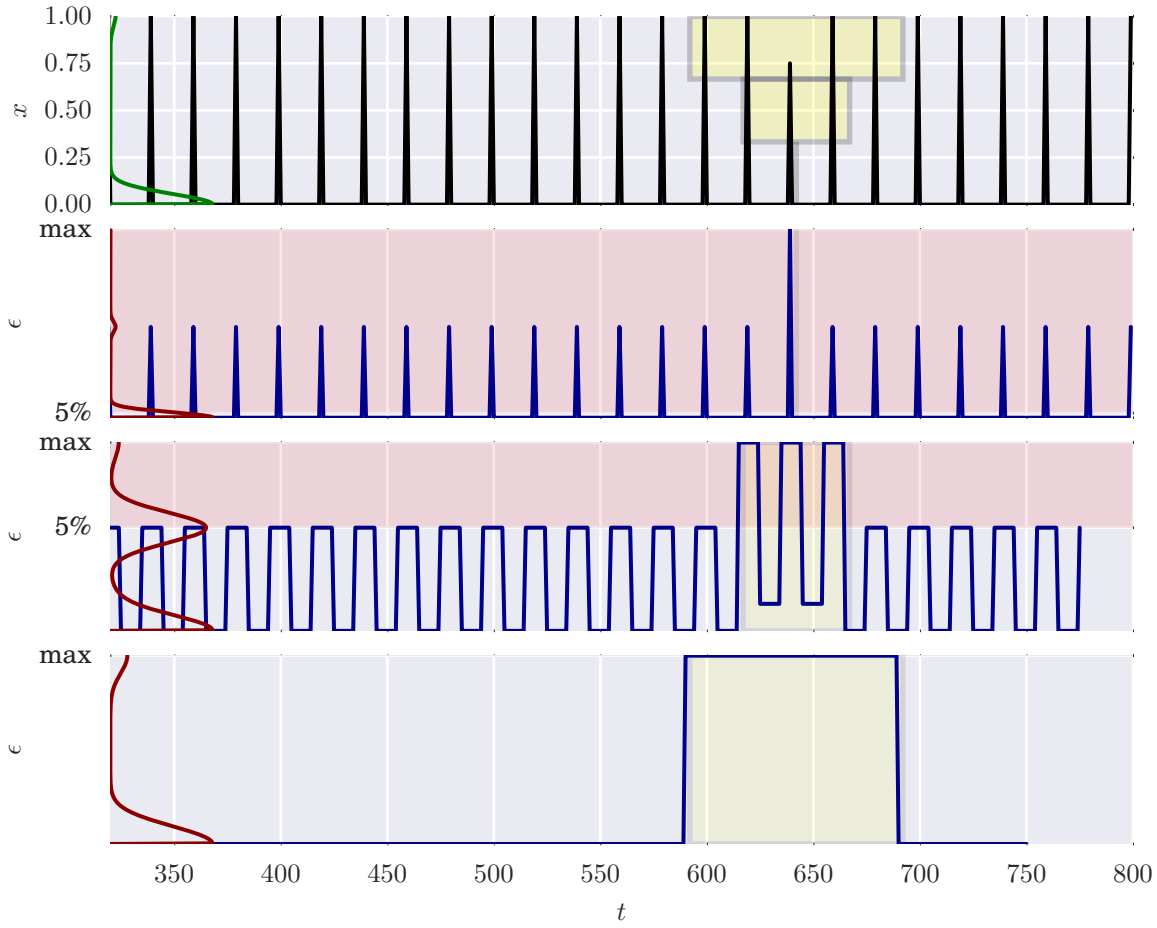

Figure 5.2 (a) spikes-1

This series can be considered discrete with a point anomaly. The point anomaly takes on an improbable value but is not extreme. Clearly, the anomalies are distinguished in all error calculations.

Note the error increases as soon as the sliding window (from the left) encounters the anomaly. This observation can be used to analyze the other series.

spikes-2

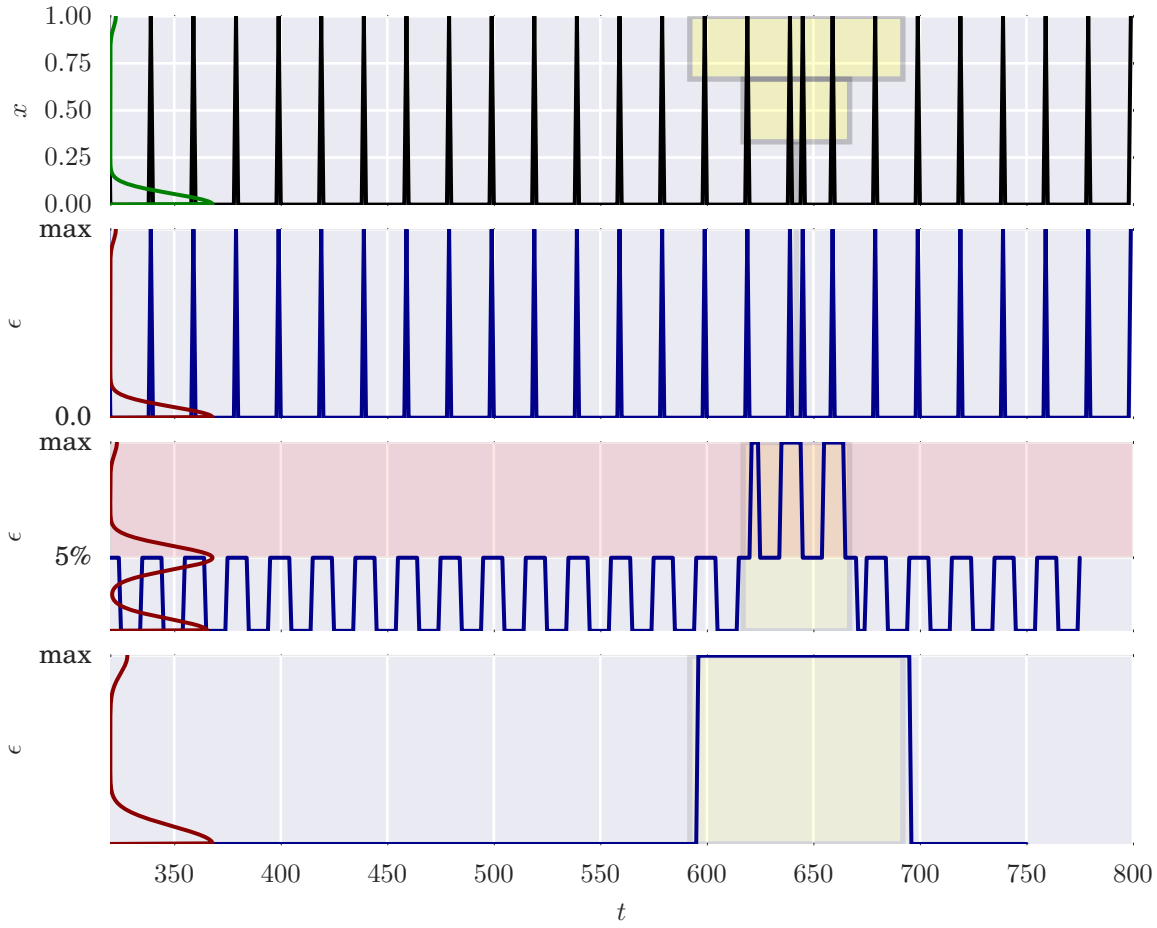

Figure 5.2 (b) spikes-2

This spike series tests if the error becomes significant where the spike is out of place; a discord. The anomaly is definitely detected in the windowed error plot but not the point error plot.

sine

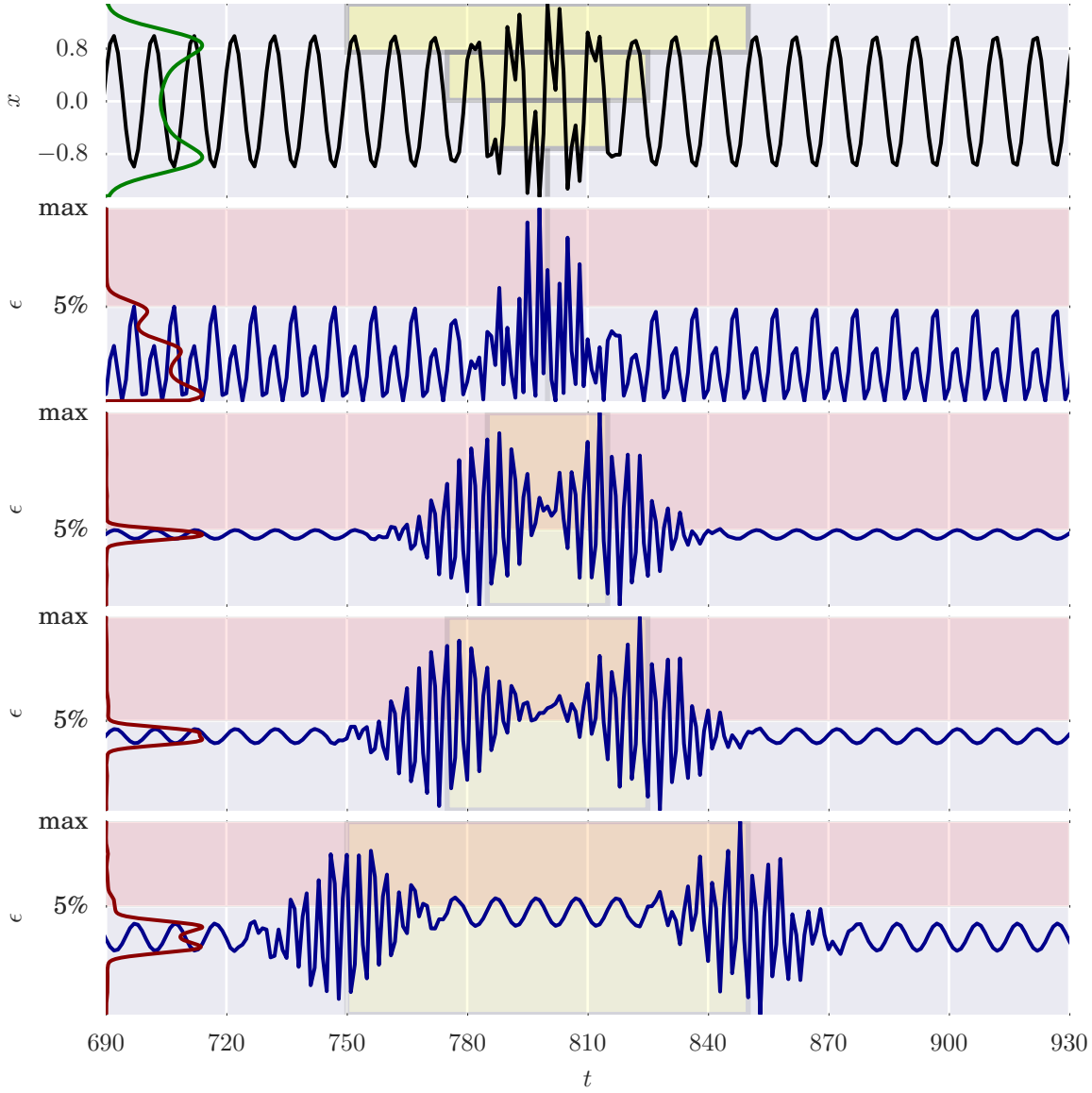

Figure 5.2 (c) sine

This series tests discord because no single point is anomalous. For the windowed errors, the anomalous and normal regions are distinguished by a range of values for each. But, there was a transition region with errors higher or lower than the anomalous region. In the point error plot, the extreme values correspond to extreme values in the test plot. So, given

these results, the windowed error can be more attributed to the extreme values of the test series than to its anomalous behavior. In any case the RNN found a tight expectation for typical values.

power

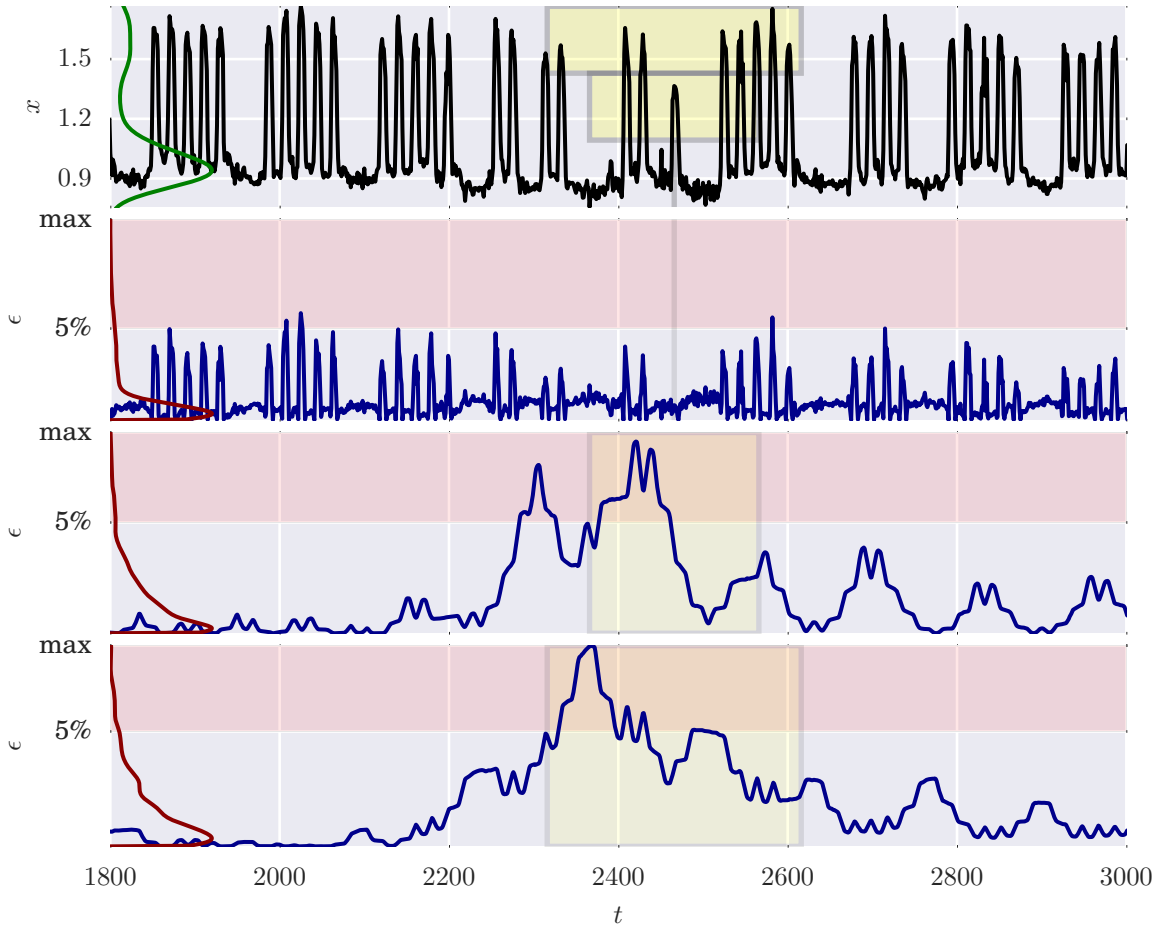

Figure 5.2 (d) power

The power series normally shows a demand profile over a five-day work week. The windowed error distribution has a long tail because high errors were given to normal areas of test series (not shown within the plot, possibly due to some drift in the series). In fact, the first windowed error plot does not contain the highest error unlike the second windowed error plot. Nonetheless, both windowed errors distinguished the anomalous region. But,

obviously, the point errors cannot be used to detect the anomalous region.

ECG

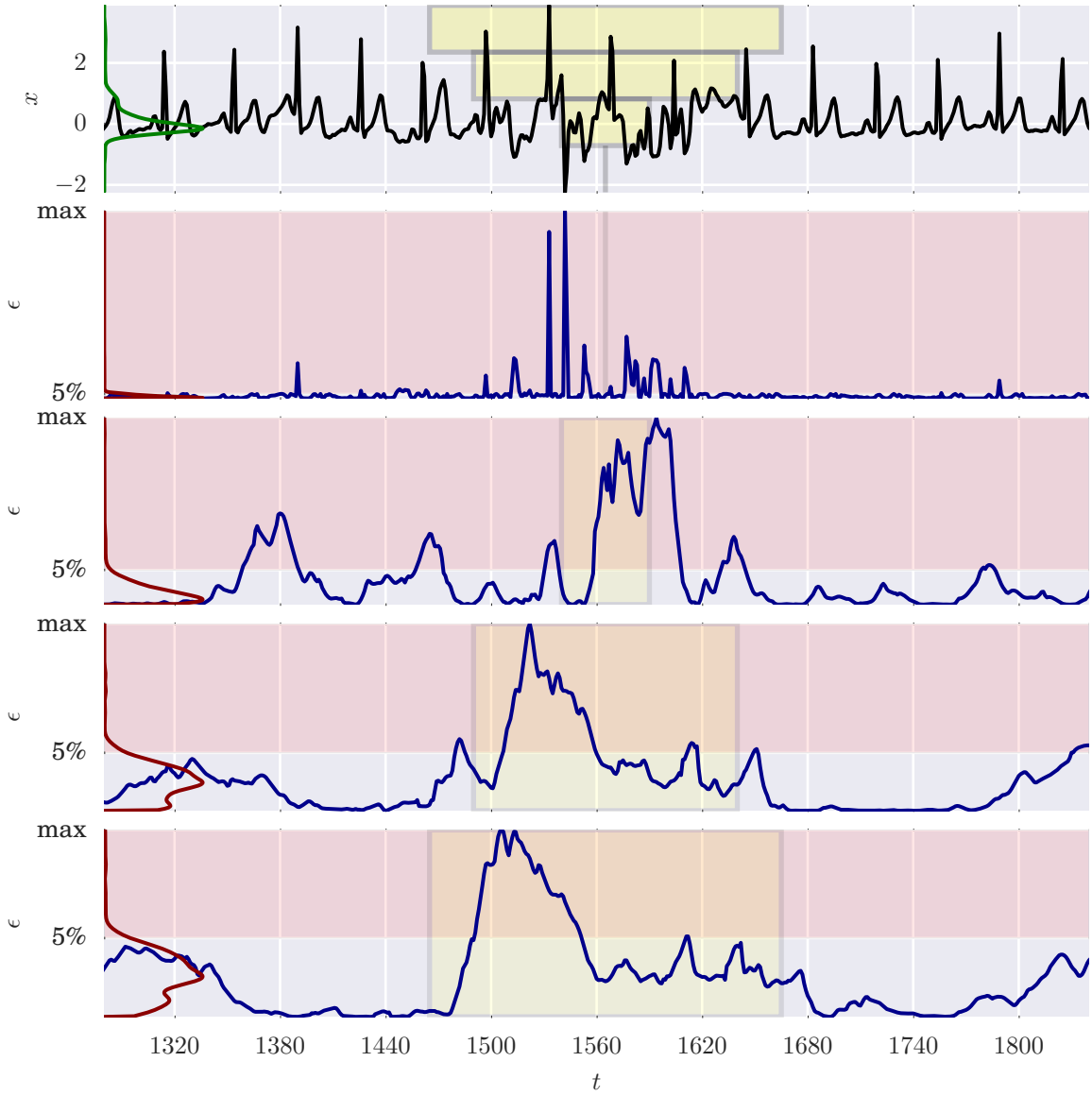

Figure 5.2 (e) ECG

The ECG is a challenging series because, while there is a repeating element, the element does not precisely have the same period. There is also noise in the signal that has to be distinguished from its significant features.

In the point error plot, the most prominent values correspond to points on the test series that are simply lower than a baseline. The same cannot be said about the error plot with the smallest window; although the areas with extreme values are the most prominent, areas that have different behavior, but not extreme in value, are distinguished as well. For example, a particularly subtle anomaly was detected within the highest 5 percentile at around $t = 1380$ despite the fact that the more anomalous *region* contributed extreme values. The same could be said about anomaly at around $t = 1470$. This window size was chosen to be about the size of the repeating unit since the effect of these anomalies on the error is diluted for larger windows. The larger windows detected the most anomalous region but it is not apparent whether the extreme error values were due to deviant values or deviant behavior.

PSG-ECG

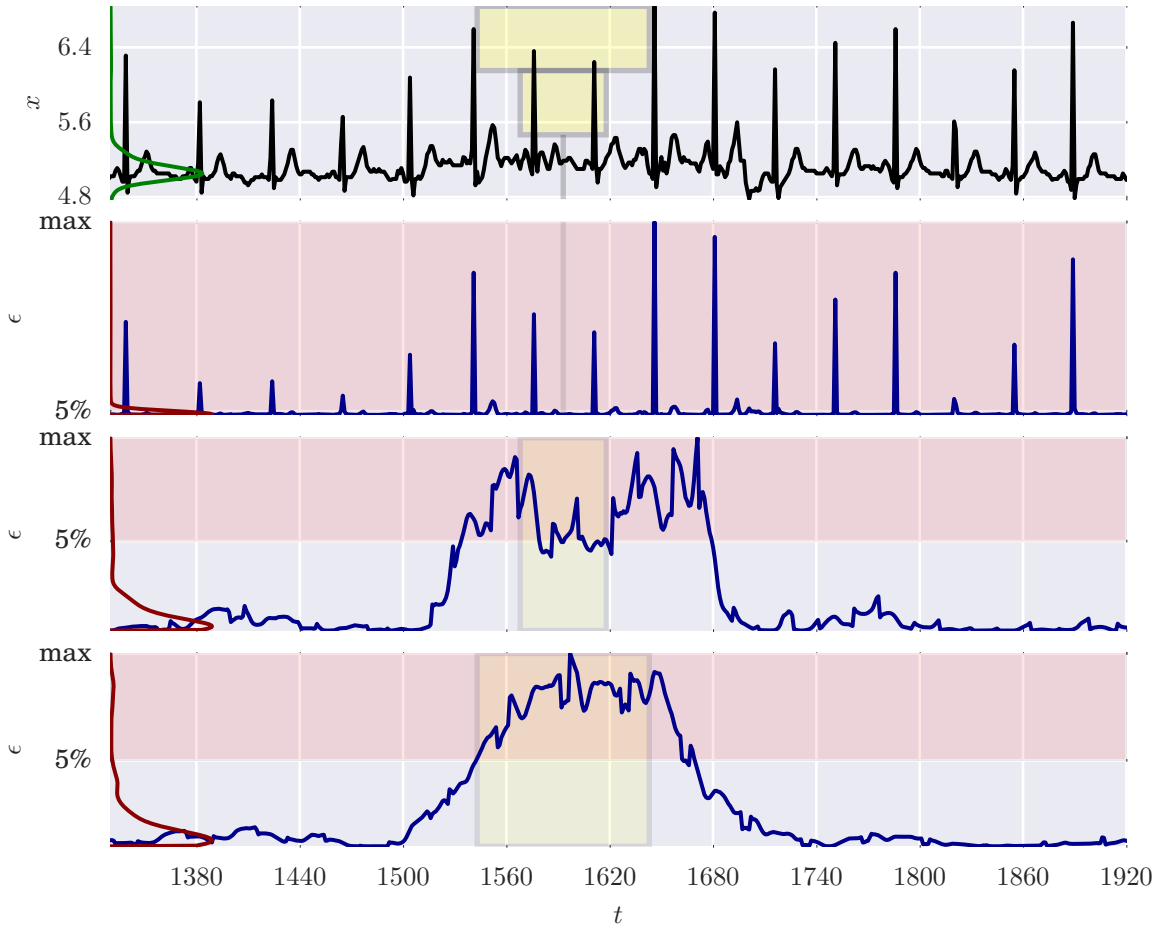

Figure 5.2 (f) PSG-ECG

Figure 5.2: Anomaly scores of series. In the top pane, a portion of the input series is shown with an anomaly. While the focus is on the anomaly, enough normal data is shown for subjective comparison. This is accompanied by a kernel density estimate, plotted on the ordinate (vertically), to evaluate the normality of the data *by value* as opposed to *behavior*. The lower panes show series of squared error, ϵ, where each pane is associated with a sliding window size. The window size is represented by a highlighted vertical span in the error plot as well as a corresponding box in the input series plot to give a sense of scale to the anomaly relative to the window. Sliding a window over the test series that calculates its MSE generates the error series (error calculation type 1). The locations of the MSE points correspond with the center of the sliding window. However, error plots with just a vertical line as a 'window' are just individual squared errors (error calculation type 2). Furthermore, a kernel density estimate for all the errors (including ones outside the plot) is plotted (vertically) on the ordinate. The highest 5 percentile and the maximum of (all) the errors are marked on the ordinate also.

In this ECG, the point errors do not reveal much about the anomalous region because every spike in the error plot plot corresponds to a normal spike in the ECG, normal or otherwise. However, the anomalous region is clearly detected in both windowed error plots. Again the error in the anomalous region is well beyond the normal range of error fluctuations albeit the test series values are slightly above average (as can be seen from the density estimate).

5.6 Conclusions

Given the results, which are influenced by the described training process and RNN architecture, some general statements can be made about how well the RNNs can find anomalies.

- Using the point errors, extreme, but not necessarily rare, values may be found.
- The windowed errors are a versatile way of finding both point and discord anomalies at multiple scales. However, this means that the window size must be about the same scale as the anomaly. Else, the effect of the anomaly would not be pronounced. Furthermore, the minimum window size must be long enough to contain meaningful dynamics.
- Pursuant to the previous two points, windowed errors are more reliable in finding anomalies.
- The training data can have some anomalous data.
- The RNNs were insensitive to variations in sequence length and translation.
- With minor variation, the same process can be used to find anomalies in a variety of series.

In general, subject to effective training, the RNNs 'learned' the typical *behavior* of the series which includes information about their typical values as well as their ordering.

Concludingly, the autoencoding recurrent neural network can be used to detect anomalies in sequences.

Chapter 6: Concluding Remarks

Now that proximity-based and model-based anomaly detection techniques have been introduced in Chapter 2, some comparisons can be made with them given the results from the previous chapter. From there, some qualified conclusions can be made. Mind that, from the start, the comparison is made with techniques that do not require labeled data. Also, the comparison is made as general statements of advantages of RNNs over the alternative.

Hidden Markov Models (model).

Chapter 4, explained how, fundamentally, RNNs store states more efficiently. By itself, this does not provide a functional advantage over HMMs, but this requires an HMM for every sequence length, unlike RNNs. Furthermore, while HMMs are powerful, RNNs are fundamentally sequence modelers.

HOT SAX (proximity).

The HOT SAX [32] technique (and its variants) is considered a proximity-based technique optimized for sequences that is sensitive to window size. While the results show in the previous chapter that window size is important, RNNs have the advantage that, the same RNN can be used to find anomalies at different scales. In HOT SAX, a comparison is made for many pairs of windows for one window size. This thorough comparison may be tolerable for short sequences, but a *trained* RNN can analyze a long sequence for anomalies based on a shorter sample. Furthermore, the mathematics of RNNs naturally accept multivariate sequences.

Through the previous discussion, the advantage of using autoencoding RNNs as described in this work, in comparison to other techniques, can be summarized in a few questions. A negative response to the following questions for the alternative gives RNNs an advantage.

- Is only the test sequence needed to determine how anomalous it is?[①] (Is a summary of the data stored?)
- Is it robust against some window length?
- Is it invariant to translation? (Is it invariant to sliding a window?)
- Is it fundamentally a sequence modeler?
- Can it handle multivariate sequences?
- Can the model prediction be associated with a probability [76]?
- Can it work with unlabeled data? If not, is it robust to anomalous training data?
- Does it not require domain knowledge?

Finding a technique with all these advantages is difficult. But, as mentioned in the Introduction chapter, the work in [2] is the closest to the work described here so some comparison is merited. In [2], the RNN is trained to predict a set length of values ahead for every (varying length) subsequence that starts at the first point[②]. Although this training setup was used to avoid dealing with windows (as an advantage), the choice of the prediction length remains arbitrary and its effect on finding anomalies at different scales is not studied. In this work, although windows were found to be needed to detect anomalies, the only choice made regarding their length was to set some minimum meaningful length for the training samples[③] (not the scale of the anomaly). In fact, specifying a window size for the prediction errors (as in Section 5.5) can be seen as an advantage because it allows detection of anomalies at different scales as a desired choice for the investigator. Furthermore, [2] uses normal data for training thereby not providing evidence that their process can tolerate

[①]This is related to generative versus discriminative models. Generative models are preferred for anomaly detection.

[②]Clarification provided in electronic exchange with author, P. Malhotra.

[③]Another way of seeing the difference in the mode of operation between the two RNN setups is by considering their mappings. In [2], an arbitrary subsequence is mapped to a fixed length sequence while in this work an arbitrary subsequence is mapped to itself.

anomalous data. But in contrast to this work, evidence for anomaly detection in multivariate time series is provided.

Unfortunately, the power of RNNs comes at high computational expense in the training process. Not only is there a cost in finding RNN parameters ($\boldsymbol{\theta}$), but there is also a cost in finding RNN hyper-parameters which can include parameters specifying RNN architecture as well as parameters specifying training algorithm parameters.

Given the results of this work and how it compares to other techniques, it can be concluded that autoencoding RNNs can be used to detect anomalies in arbitrary sequences, provided that an initial training cost can be managed.

6.1 Further Work

The text ends with a list of further work directions with potential to strengthen the case for using autoencoding RNNs in anomaly detection. As the list is mainly concerned with the RNNs, and much progress has been made in RNN research recently, the list is not exhaustive. Furthermore the rapid progress might render items in the list as outdated in the near future.

Better optimize presented work. More training epochs and more LSTM layers could have found more optimized parameters. Also, variations in the training data on the length scale of the sequence (trends) should be removed. These optimizations are important to effectively learn normal sequence patterns.

Use autocorrelation to determine a minimum window width. In the sampling process, the minimum window length was manually determined such that the length captured meaningful dynamics. This length can be systematically determined by using information from the sequence's autocorrelation.

Accelerate training.

Normalize input. Although not required, some carefully chosen normalization of data could help. Another normalization scheme to consider is found in a recent

paper [118] which suggests using normalization based on mini-batch statistics to accelerate RNN training.

Find an optimum mini-batch size. Some redundancy in the mini-batch is desired to make smooth progress in training. However, if the mini-batch size is too large (too redundant), a gradient update would involve more computations than necessary.

Use dropout to guard against overfitting. In this work, to guard against overfitting, a corrupting signal is added which depends on the value of the data. In dropout, regardless of the values of the data, a small portion of nodes in a layer can be deactivated allowing other nodes to compensate. Dropout was first applied to non-recurrent neural networks but recent study [119] explains how dropout can be applied to RNNs.

Experiment with different RNN architectures.

Experiment with alternatives to the LSTM layer. Over a basic RNN, the LSTM imposes more computational complexity as well as more storage requirements (for the memory cell). Gated Recurrent Units (GRU) [120] are gaining in popularity as a simpler and cheaper alternative to LSTM layers.

Experiment with bi-directional RNNs. Bi-directional RNNs [121] incorporate information in the forward as well as reverse direction. They have been successfully used with LSTM for phoneme classification [122].

Experiment with more connections between RNN layers. A better model might be learned if non-adjacent layers are connected [82] (through weights) because it allows for more paths for information to flow through.

Incorporate uncertainty in reconstruction error. The output from a RNN can be interpreted to have an associated uncertainty [76]. It follows that it should be possible to get high or low error signals associated with high uncertainty which should affect

the interpretation of the existence of an anomaly (see Reconstruction Distribution, Section 13.3, in [81]).

Objectively compare anomaly detection performance against other techniques over a range of data. While certain disciplines might have benchmark datasets to test anomaly detection, measuring the generality of a technique by evaluating its performance over a wide variety of data is not widespread[4]. To solve this problem, Yahoo recently offered a benchmark (labeled) dataset [123] which includes a variety of synthetic and real time series.

Methods based on non-linear dimensionality reduction might be competitive [124].

Find anomalies in multivariate sequences. The NASA Shuttle Valve Data [125] is an example which was used in [111] and the well-known HOT SAX [32] technique.

[4] Perhaps this is due to the difficulty in finding a general technique.

Chapter A: Reproducible Computational Infrastructure[♠]

A.1 Introduction

There has been much attention recently paid to reproducible computational research [126]. In some cases, just providing the computational code and data, along with some instructions, is sufficient to be able to reproduce a computational experiment. However, typically code relies on libraries and other support facilities which complicates the computational setup. So, just providing the computational code is not sufficient to ensure reproducibility (at least not easily). Some domains have managed this complexity somewhat by providing specific solutions. As examples, Galaxy is used in genome analysis [127], Madagascar in geophysics [128], WaveLab in signal processing [129], and Bioconductor in computational biology and bioinformatics [130]. These solutions can be seen as platforms onto which instructions can be provided to reproduce results.

However, these solutions do not address computational infrastructure setup in addition to being limited to their domains. 'Infrastructure' here means aspects related to both hardware and software. While the importance of hardware is not emphasized as much as software in reproducibility[①], it is best to think of hardware as clean slates onto which software is installed, beginning with the operating system. In fact, some computational code requires certain hardware like graphics processing units (GPUs). Furthermore, computational codes might interact with (non-computational) services provided by the operating system and/or non-computational services that perhaps are closely associated with the operating system. Therefore providing instructions, in the form of code, that specify the hardware *and* software

[♠]The code described in this chapter is referenced under DOI 10.5281/zenodo.45950

[①]This is because the quantitative programmer is usually highly removed from hardware details. The same cannot be said of software dependencies.

has much value for reproducibility. The benefit from having such instructions is not limited to ensuring integrity of results; the iterative work process is greatly enhanced because this level of reproducibility implies automation.

Many software tools from information technology (not specific to high-performance computing) automate infrastructure setup. As such, the results presented in Chapter 5, were obtained by using an automated process that used some of these information technology tools. Furthermore, while being motivated by a problem encountered in this work, the process has been separated out as an independent solution applicable to any computational problem.

The following sections describe the problem and, in turn, its solution.

A.1.1 Motivation

The Bayesian optimization process, explained in Section 5.4, requires the coordination of many components. The general problem is explained here but refer to Chapter B for the specific solution components. The components must satisfy the following requirements:

- the provisioning of a MongoDB database so that Spearmint could store its optimization progress in it
- the provisioning of a database to store RNN parameters after every training iteration
- the coordination of training runs on potentially multiple compute machines where a RNN training run is for certain hyper-parameters (a coarse form of parallelization)

Furthermore, two operational requirements can be added as well:

- There should be an automated (reproducible) process that sets up these components.
- The investigator should be able to seamlessly transition from testing and development on his/her local machine to 'production' execution on a remote machine. That is, the investigator's workflow should be uninterrupted.

So the challenge is twofold: the reproducibility of each component and the reproducibility of the setup. This implies that the setup should occur in clean, isolated environments.

A.2 Solution Elements

Some solutions solve certain parts of the previously listed requirements. They can be evaluated based on their degree of reproducibility and automation.

local virtual machine: VirtualBox. This software virtualizes all the workings of a machine into a process running on a (local) host operating system.

However, on its own, VirtualBox does not provide a systematic and generic way of provisioning the machine as well as provisioning software on the machine.

local virtual machine automation: Vagrant. By providing Vagrant with instructions in a file, virtual machine setup can be automated.

Vagrant can control VirtualBox by specifying virtual machine hardware as well a machine image (file) which typically includes an operating system installation at least. While a virtual machine provisioned automatically provides an isolated, reproducible environment for work, it needs to exist in the context of being a part of a network of machines. So, ideally, after a minimal initial provisioning, the virtual machine should be treated as just another machine regardless of the fact that it exists virtualized locally.

application reproducibility: Docker. From instructions in a file, Docker can create an isolated application image.

Docker has recently emerged as an easy yet powerful way to work with (isolated) application containers. Furthermore, the image execution is portable as long as the destination machine has compatible hardware architecture which has obvious advantages when working with multiple machines. Also, by persisting the image, the application can be started swiftly since the image is the result of potentially lengthy

software installation processes. In the context of computational research, it is possible to have isolated, reproducible containers for, as examples, databases, computational code, and computational task management. While Docker provides a great deal of application reproducibility, it is not involved in machine provisioning.

distributed Docker support: Weave and CoreOS. Weave and CoreOS facilitate the distributed operation of Docker containers.

Given that Docker was identified as an important solution component, it follows that Docker-specific solutions should be chosen that facilitate distributed application execution. Weave provides global private network addressing for each container. CoreOS is a Linux operating system designed for distributed, containerized applications. As such, it is delivered with minimal software and services as containers are assumed to be the primary method by which software and services are added. In the context of computational research, this ensures the fastest possible execution of computational code since the operating system is not running unnecessary processes.

remote machine facility: Amazon Web Services (AWS). AWS provides high-performance compute machines, including machines equipped with GPUs, running CoreOS.

What is important for the purpose of automation is that AWS provides a programmatic interface for the provisioning of machines. However, the choice of AWS is not critical because AWS can be substituted by other providers with comparable facilities.

global automation: Ansible. Ansible is the highest-level automation software that can orchestrate the infrastructure setup process (in full).

Ansible can be used to generically provision machines, local or remote, virtual or physical. Ansible can also be used to provision software, containerized or not. Furthermore, the provisioning of hardware and software can occur in a coordinated fashion.

A.2.1 Solution Stack

The solution elements can be organized into a 'stack' to help understand their place in an overall solution. As depicted in Table A.1, the stack represents a technology dependency where higher elements depend on lower elements. The goal is to be able to automatically recreate this technology stack anywhere. As such, Ansible is not shown because it is an automation tool that sets up the stack.

Table A.1: Container-oriented computational technology stack. Technologies in shaded cells are under the influence of automation by Ansible. The parentheses around x64 indicate that the hardware architecture is virtualized (under type-2 hypervisor). Table cells containing ellipses are immaterial to the discussion.

application	...	...
container network	Weave	
app. containerization	Docker	
operating system	CoreOS	
machine	(x64)	x64
hypervisor	VirtualBox	...
hypervisor interface	Vagrant	AWS
host operating sys.	Windows|OS X|Linux	...
hardware	x64	x64
	local	remote

The reproducibility of results from computation (at the application layer) is generally not influenced by technologies lower in the stack. In fact, they can be swapped with other technologies as long as the technology stack is compatible. But the automation ensures the compatibility of the stack.

For reproduciblity of results, it is more a matter of convenience that the layers above the hypervisor can be recreated locally and remotely. However, the technologies selected facilitate portability of execution in several ways which allows for easier collaboration and reproducibility of results starting at different levels. At the lowest level, the automation code can recreate the full stack, locally and remotely, perhaps with different hypervisors. So, at the container level, the highest level, Docker instructions can be used to recreate the

application image. Alternatively, the image (itself) can be transferred to any compatible machine for execution. At the application level, only the most independent codes are truly portable. Practically all modern computational codes have complex dependencies which are handled in a variety of ways. But by using Docker, dependencies are handled in the most general way.

In fact, the level of encapsulation that Docker offers has been compared to that of virtual machines (inaccurately[②]). But unlike a virtual machine, it does not suffer from the additional overhead incurred by running an operating system and possibly virtualizing hardware as well. In fact, the overhead of running Docker is negligible. So if the concern is just running a typical application, transferring a Docker image is preferable to transferring a virtual machine image[③].

But speaking of a 'stack' on a machine, individually, does not address distributed computing. Using multiple machines to accelerate computing is highly-desirable, if not essential, depending on the application. This is where automating the entire stack on any compute provider, homogeneously, becomes advantageous since the automation code embodies the distributed environment. Therefore, even the distributed environment can be reproducible.

A.2.2 Partial Solutions

Research-oriented cluster computing facilities were found to not satisfy the requirements previously mentioned. Typically, the machines are provided with an operating system installed with a restricted user account. This restricts the ease in which some software can be installed although this can be mitigated somewhat if Docker is installed. Most importantly, is that the use of research-oriented clusters does not facilitate a seamless transition from local development to remote execution because the local and remote environments do not match (unless the local environment is restricted to use the same technologies as the facility which would limit portability of the setup process). So treating cluster computing providers

[②] It is best to think of Docker containers as encapsulated *processes*.

[③] Applications requiring specialized hardware such as GPUs are not as transferable.

as just providers of machines is advantageous because it allows the investigator to generically automate the setup of the same computing environment on any facility.

While automating the setup on a research-oriented cluster is possible, StarCluster is notable for automating the setup on AWS. Although the setup is convenient, StarCluster otherwise is much like a research-oriented cluster computing facility but with more control given to the user since administrative privileges on the operating systems are granted. Not only is StarCluster restricted to AWS, but its technology stack is not designed for the primary usage of Docker. Furthermore, the argument holds that StarCluster does not facilitate the seamless transition from local development to remote execution due to the mismatch between the environments.

A.3 Solution

Section A.2.1 discussed the advantages of having a homogeneous computing stack on any hardware based on Docker solutions. This section will exhibit a 'base' infrastructure onto which complex distributed applications can run. The end result is that the investigator gains control of multiple machines all of which have almost identical environments. Figure A.1 depicts the base infrastructure in a technology layer-oriented view. Machines occupying the top row in the diagram are designated as 'compute' machines.

The general infrastructure setup procedure, controlled by Ansible, is as follows:

1. File Share Setup

The base infrastructure begins with setting up resources on a user's local machine to enable the execution of applications on remote machines.

Central to this is sharing local files which provides persistence of data and executables between instantiations of machines. The file share also provides convenience for developing code on any machine which is especially powerful when using version controlled code.

2. `init` Machine Setup

The purpose of the `init` machine is to provide coordination services and other low resource services to other machines in the network. In the base setup the `init` machine provides two services:

File Share Expose

Through the use of Weave, the `init` machine sets up, transparently, local *or* remote, access to the file share.

Docker registry

The init machine runs a Docker registry (of Docker images) so that any machine can download service and computation applications for execution.

The images are built from code (in '`Dockerfiles`') in an initialization process. The images can modularly build on one another. For example, app1 and app2 (shown in Figure A.1) can build on a common lib image.

3. Compute Machine Provisioning

After the `init` machine is set up, compute machines can be brought into the set of machines controlled by Ansible. Local compute[④] machines can be provisioned by Vagrant which are well-suited for testing. Remote compute machines can be acquired as well from AWS[⑤] for accelerated execution.

After the setup, the investigator can expect that the same commands will work on any machine. Moreover, the set up can be executed on any (local) machine. This is demonstrated in Chapter B.

More details on the setup and its use can be found in the code repository `https://github.com/majidaldo/personal-compute-cloud`.

[④] Non-virtualized local computing resources can be added to the network like app3 running on `localhost` as shown in Figure A.1. But such operation is not the purview of the automation code.

[⑤] Other providers can be integrated by creating and modifying Ansible scripts.

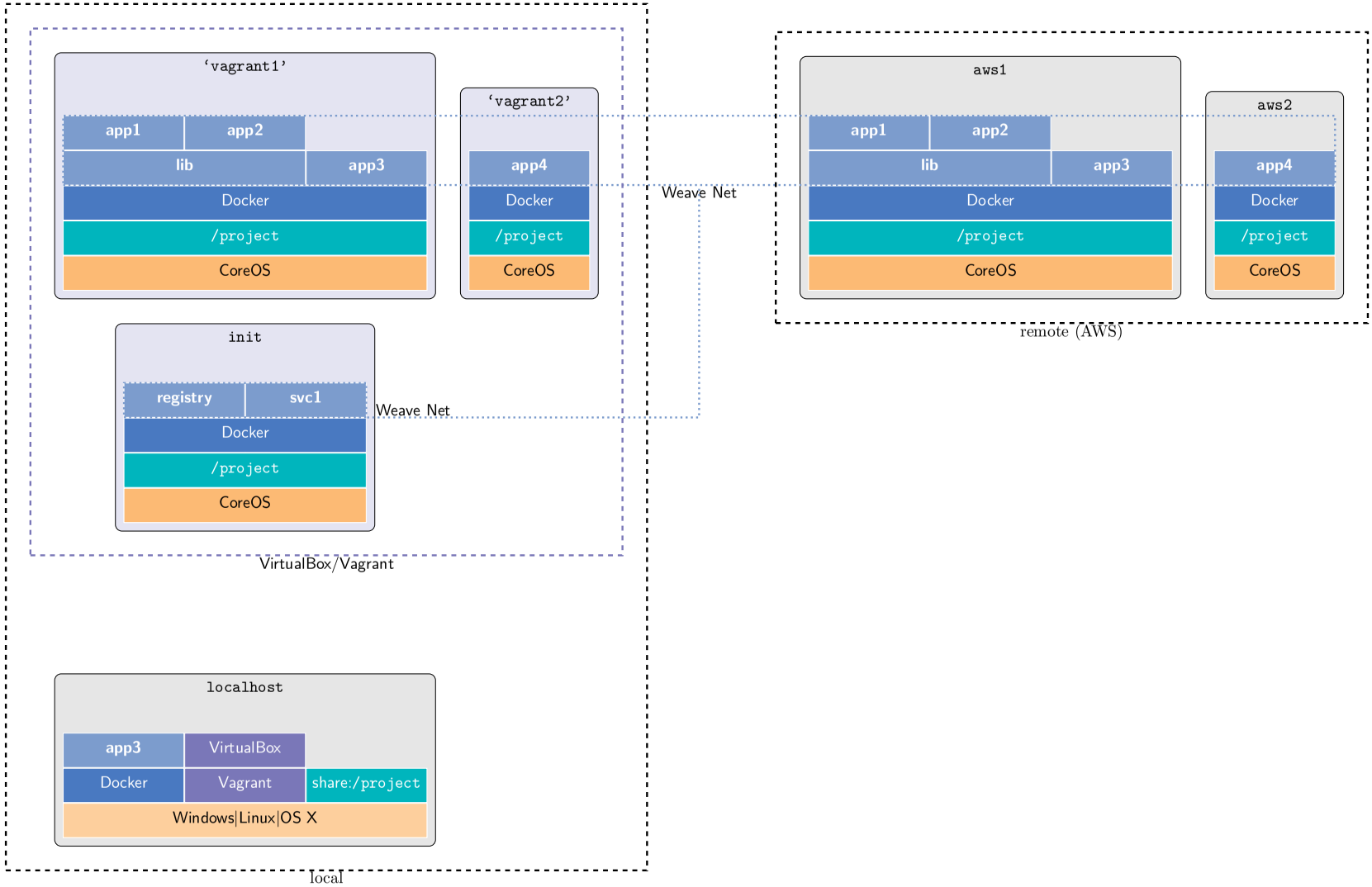

Figure A.1: Generic infrastructure for distributed computation based on Docker. Dashed boxes group networks. The (dotted) Weave network crosses network boundaries.

Chapter B: Reproducing Results

B.1 Introduction

This chapter shows the commands that were issued to obtain the results shown in Chapter 5. The commands build on the infrastructure setup explained in Chapter A. Also, the commands (themselves) specifically address requirements of the Bayesian optimization process driven by Spearmint as described in Section A.1.1. As such they are the highest-level commands that can be automated as well but were not to give the investigator some control over execution. But the control is only operational as similar results should be obtained regardless of how the computations are executed.

However, first, a set of 'manual' instructions are provided as well that are more typical of instructions accompanying computational publications (if at all!). The manual instructions, in contrast to the automated instructions, are not straightforward to extend for distributed execution. Nonetheless, the automated execution is just encapsulating the manual procedure with an automated one which embodies a distributed environment. So, becoming familiar with the manual process helps with understanding the automated process.

In any case, all software used was current as of January, 2016. So, if the software version is not identified, it can be assumed that it was current as of January 2016. So newer software *may* work.

B.2 Manual Execution

0. Install prerequisite software.

Install the following. Using the versions indicated ensures reproduction of results [①]:

[①] Each has its own, possibly multiple, methods of installation! Furthermore, the listed software can

Python:	2.7.11
NumPy:	1.10.1
SciPy:	0.16.0
scikit-learn:	0.16.1
MongoDB:	3.1.9
PyMongo:	2.8
psutil:	3.2
modified Spearmint:	Git commit ccc503ae08798cb5ed9fd6090310de89b6d9b39f from the repository at `https://github.com/majidaldo/Spearmint`
Theano:	Git commit fe58ada7cbf8b6fd031d9ad9b3c6c570b1717f9b from the official repository at `https://github.com/Theano/Theano`
Theanets:	Git commit ae588cfd6b7b04c02a603ecfee4ba14c68d46ca2 from the official repository at `https://github.com/lmjohns3/theanets`

1. Obtain main program code.

The main computational code is available at `https://github.com/majidaldo/tsad`. Use Git commit 37703f3d10cf5ff26cc72cc8b7dff639cffda78c.

2. Set up.

1. Configure RNN storage.

With local execution of MongoDB, the server host must be set to `localhost` in `config.yml` in the program code.

have their own prerequisites possibly with version restrictions. Therefore, using Conda from Continuum Analytics is recommended for installing Python-based computational software. Conda can be obtained from `http://conda.pydata.org/miniconda.html`.

```
---
rnndb: localhost
```

2. **Start MongoDB.**

   Usually, this involves some kind of invocation of the `mongod` command. Refer to MongoDB documentation for specific instructions.

   The MongoDB instance will store RNN parameters and Spearmint progress.

3. **Change to sequence directory.**

   Each sequence optimization is associated with two executables and a Spearmint configuration file in a directory in the `experiments` directory. The correspondence between the names used in Section 5.2 to the directory names is as follows:

spike-1	`spikelv`
spike-2	`spikereg`
sine	`sin`
power	`power`
ECG	`ECG`
ECG-PSG	`sleep`

4. **Configure Spearmint.**

   Ensure that Spearmint runs the non-distributed executable, `o.py`, by modifying the contents of the configuration file, `config.json`, if needed. `"o.py"` should be the value for the `main-file` attribute of the configuration file. Also, the `address` attribute for `database` should be set to `"localhost"` since it is assumed that a local MongoDB is running. With these settings, `config.json` should be as follows.

```
{

    "experiment-name": "power",
    "database": {"address":"localhost"},
    "language"    : "PYTHON",

    "resources" : {
        "my-machine" : {
```

```
            "scheduler"          : "local",
            "max-concurrent"     : 1,
            "max-finished-jobs" : 20
        }
    },

    "tasks": {
        "main" : {
            "type"       : "OBJECTIVE",
            "likelihood" : "GAUSSIAN",
            "main-file"  : "o.py",
            "resources"  : ["my-machine"]
        }
    },

    "variables" : {
        "nl" : {
            "type" : "INT",
            "size" : 1,
            "min"  : 1,
            "max"  : 2
        },
        "n" : {
            "type" : "INT",
            "size" : 1,
            "min"  : 1,
            "max"  : 10
        },
        "iter" : {
            "type" : "FLOAT",
            "size" : 1,
            "min"  : 0,
            "max"  : 1
        }
    }
}
```

In addition, the parameter space for the Bayesian optimization search can be set by changing the `min` and `max` under the `variables` attribute. `nl` is the variable name for number of RNN layers, l, while `n` is the number of nodes in a layer, n.

3. Run Spearmint.

```
python /path/to/Spearmint/spearmint/main.py .
```

B.3 Automated Execution[2]

1. Obtain automation code.

The code is available at `https://github.com/majidaldo/tsad-sys`. It is made of several submodules so `git` should be used with the `recursive` option.

```
git clone --recursive https://github.com/majidaldo/tsad-sys
```

Then, change into the code directory and checkout the appropriate version.

```
git checkout 8f20b9768cd4c4ae83bbb8e156e5d7aa9fbb2565
```

The automation code also contains the computational code as a submodule in the `tsad` directory.

2. Set up.

1. Set up automation.

Follow instructions according to the `README` file `https://github.com/majidaldo/personal-compute-cloud/tree/thesis0/README.md` to initialize the automation, provision the `init` machine, and provision compute machines. There is no need to change any system variable set in the automation code.

2. Start services.

`ssh` into the `init` machine to run services on it. The services (and compute applications) are all Docker applications. Each application corresponds to a directory within the `docker` directory in the automation code. The application directories contain shell scripts to run the application.

1. Start MongoDB.

Change into the `/project/docker/1111-mongodb` directory and execute `./run.sh`.

[2] Unfortunately the reproducibility of the automated computation is flawed because for most of the software listed in Step 0 of the manual execution instructions, the version is not specified in the code. Consequently, the most recent version is installed by default. However, the versions can be fixed easily by modifying the appropriate command in the file `docker/0100-computer/Dockerfile` in the automation code for CPU operation or `docker/0111-computer-gpu/Dockerfile` for GPU operation.

2. **Start the IPython controller[3].**

   Similarly, change into the `/project/docker/0200-ipycontroller` directory and execute `./run.sh`.

   The IPython controller handles the distribution of computational tasks across workers. Its data is configured for storage in `MongoDB` as well.

3. **Configure RNN storage.**

   Just like in manual execution, the RNNs are configured for storage in a MongoDB database. So similarly, confirm that the database host is set to `mongodb` in `tsad/config.yml`.

4. **Configure Spearmint.**

   Again, like in manual execution, change into a sequence directory in `tsad/experiments`. Ensure that Spearmint runs the *distributed* executable, `po.py`, by modifying the contents of the configuration file, `config.json`, if needed. `po.py` should be the value for the `main-file` attribute of the configuration. Furthermore, set the number of concurrent compute workers Spearmint controls by changing the value of the `max-concurrent` attribute.

   Also, the `address` attribute for `database` should be set to `"mongodb"` since it is assumed that a *remote* instance of MongoDB is running. With these settings, `config.json` should be as follows.

```
{

    "experiment-name": "power",
    "database": {"address":"mongodb"},
    "language"   : "PYTHON",

    "resources" : {
        "my-machine" : {
            "scheduler"         : "local",
            "max-concurrent"    : 5,
            "max-finished-jobs" : 20
        }
```

[3] The IPython controller is part of IPython Parallel. Version 4.0.2 of IPython Parallel was used.

```
    },

    "tasks": {
        "main" : {
            "type"       : "OBJECTIVE",
            "likelihood" : "GAUSSIAN",
            "main-file"  : "po.py",
            "resources"  : ["my-machine"]
        }
    },

    "variables" : {
        "nl" : {
            "type" : "INT",
            "size" : 1,
            "min"  : 1,
            "max"  : 2
        },
        "n" : {
            "type" : "INT",
            "size" : 1,
            "min"  : 1,
            "max"  : 10
        },
        "iter" : {
            "type" : "FLOAT",
            "size" : 1,
            "min"  : 0,
            "max"  : 1
        }
    }
}
```

In addition, the parameter space for the Bayesian optimization search can be set in the same manner described in manual execution.

3. Run Spearmint.

In distributed operation, Spearmint is run as a coordinating 'service' so it is recommended that the following steps are run on the `init` machine.

1. Enter compute environment.

Change into the `docker` directory that represents the computing environment,

`/project/docker/0100-computer`. Then, start the environment with the following command[4].

```
./run.sh ipykernel
```

Now, enter the environment by issuing the following Docker command.

```
docker exec -ti init-ipykernel-1 bash
```

Notice that the command includes `init` because the `run.sh` command assigns a name to the Docker container that includes the hostname.

2. Execute.

Now inside the Docker container, change into the sequence directory in `/data/experiments` and execute the following.

```
python ~/spearmint/spearmint/main.py .
```

4. Join compute workers.

The workers are IPython engines that connect to the IPython controller. Running compute workers on compute machines provisioned by the automation is straightforward. However, other workers can join as long as their environment is the same as that described in the manual execution section. Instructions are provided for both.

Join automated workers.

`ssh` into a compute worker and run the following in the `/project/docker/0100-computer` directory to start a compute container.

```
./run.sh ipyengine 1
```

The 1 indicates that just one worker will be started but the number can be increased.

A NVIDIA GPU-equipped machine can start a GPU accelerated worker by running the same command in the `/project/docker/0111-computer-gpu` directory.

[4]This runs an IPython kernel but it is irrelevant.

Computation progress can be monitored by issuing a `docker logs` command for the container.

```
docker logs compute-machine-name-ipyengine-1
```

The 1 stands for the first worker started from the previous `./run.sh` command. The container name can be found from the list output from the command, `docker ps`.

Manually join worker.

An `IPython` engine can join the group of workers if it establishes connectivity to the MongoDB database and the IPython controller. Assuming the worker is on the local machine setup as per the manual execution instructions in Section B.2, connect the worker as follows.

1. **Forward MongoDB and IPython controller network ports to the `init` computer.**

   Instead of communicating to local services, local communication is forwarded to services running on the `init` machine. Since the ports are forwarded from the local machine, in the main program code directory (say, `tsad`) confirm that the RNN code is communicating with MongoDB as if it were local; check the contents file `config.json` so that it is as follows.

   ```
   ---
   rnndb: localhost
   ```

   Then, also in the `tsad` directory, navigate to the `docker` directory. In the `docker` directory, the `port.sh` script forwards network ports. Forward the MongoDB port with `./port.sh 27017`. In another command console, forward (one of) the IPython controller ports with `./port.sh 4321`. Keep the consoles open.

2. **Start IPython engine.**

   From the `tsad` directory, start the IPython engine as follows.

```
ipengine \
--file=/path/to/tsad-sys/files/\
.ipython/profile_default/security/ipcontroller-engine.json \
--ssh="core@init"\
--sshkey=/path/to/.vagrant.d/insecure_private_key
```

The `tsad-sys` directory is the automation code directory. In this command, the IPython engine is making a SSH network tunnel to the IPython controller residing in the `init` machine. The IPython controller writes its (dynamic) network configuration to the `ipcontroller-engine.json` file on its start. This information is available on the (outside) local machine through the file share. For the location of the `insecure_private_key` file, check your Vagrant installation.

B.4 Reproduction of Figures

The figures shown in Chapter 5 are produced in a separate process associated with the production of this document. Figure production is not fully automated but the previously described computation environment can used as a basis for producing the figures by following the steps below.

1. Install data analysis and plot tools.

Install the following in a compute environment:

matplotlib	1.5.1
Seaborn	0.6.0
pandas	0.17.1

2. Obtain code for this document.

Download the code from `https://github.com/majidaldo/tsad-docs`. Then check-out the appropriate version as follows.

```
git checkout c524238ba7a4e514e6cb155afe42c3b4808048ed
```

3. Configure location of main computation code.

In the document code, change the path in `figs/tsad.py` to the location of the main computation code.

4. Generate figures.

Assuming MongoDB is running, from the `figs` directory execute the following command.

```
python figs.py all
```

The figures will be produced as files written to the same directory.

Bibliography

Bibliography

[1] E. Marchi, F. Vesperini, F. Eyben, S. Squartini, and B. Schuller, "A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, vol. 289021, no. 289021, apr 2015, pp. 1996–2000.

[2] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *European Symposium on Artificial Neural Networks*, no. April, 2015, pp. 22–24.

[3] D. Cheboli, "Anomaly Detection of Time Series," Ph.D. dissertation, The University of Minnesota, 2010. [Online]. Available: http://udc.umn.edu/handle/11299/92985

[4] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier Detection for Temporal Data : A Survey," *Ieee Transactions on Knowledge and Data Engineering*, vol. 25, no. 1, pp. 1–20, 2013. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6684530$\delimiter"026E30F$nhttp://www.morganclaypool.com/doi/abs/10.2200/S00573ED1V01Y201403DMK008http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6684530

[5] G. Jiang, H. Chen, and K. Yoshihira, "Modeling and tracking of transaction flow dynamics for fault detection in complex systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 4, pp. 312–326, 2006.

[6] B. Szymanski and Y. Zhang, "Recursive data mining for masquerade detection and author identification," *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, no. i, pp. 424–431, 2004.

[7] N. Ye, "A markov chain model of temporal behavior for anomaly detection," *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, no. 4, pp. 171–174, 2000.

[8] L. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," *Proceedings of ACM CSS Workshop on Data Mining Applied to Security Philadelphia PA*, pp. 1–25, 2001. [Online]. Available: http://freeworld.thc.org/root/docs/intrusion{_}detection/nids/ID-with-Unlabeled-Data-Using-Clustering.pdf

[9] G. Zhen, G. Jiang, H. Chen, and K. Yoshihira, "Tracking probabilistic correlation of monitoring data for fault detection in complex systems," in *International Conference on Dependable Systems and Networks*. IEEE, 2006, pp. 259–268.

[10] C. Warrender, S. Forrest, and B. Pearlmutter, "Detecting intrusions using system calls: alternative data models," in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*. IEEE Comput. Soc, 1999, pp. 133–145. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=766910

[11] F. Angiulli and F. Fassetti, "Detecting Distance-based Outliers in Streams of Data," in *Proceedings of the ACM Sixteenth Conference on Information and Knowledge Management*, 2007, pp. 811–820.

[12] T. Lane and C. E. Brodley, "Temporal sequence learning and data reduction for anomaly detection," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 295–331, 1999.

[13] ——, "An Application of Machine Learning to Anomaly Detection," *Computer Engineering*, pp. 366–377, 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.9537{&}amp;rep=rep1{&}amp;type=pdf

[14] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion Detection using Sequences of System Calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[15] K. Sequeira and M. Zaki, "ADMIT: anomaly-based data mining for intrusions," *ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 386–395, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=775103

[16] M. Gupta, J. Gao, Y. Sun, and J. Han, "Community trend outlier detection using soft temporal pattern mining," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 692–708.

[17] M. E. Otey, A. Ghoting, and S. Parthasarathy, "Fast Distributed Outlier Detection in Mixed-Attribute Data Sets," *Data Mining and Knowledge Discovery*, vol. 12, no. 2-3, pp. 203–228, 2006. [Online]. Available: http://www.springerlink.com/index/10.1007/s10618-005-0014-6

[18] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams," *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '03*, p. 336, 2003. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-77952383186{&}partnerID=tZOtx3y1

[19] D. J. Hill and B. S. Minsker, "Anomaly detection in streaming environmental sensor data: A data-driven modeling approach," *Environmental Modelling and Software*, vol. 25, no. 9, pp. 1014–1022, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.envsoft.2009.08.010

[20] D. J. Hill, B. S. Minsker, and E. Amir, "Real-time Bayesian anomaly detection in streaming environmental data," *Water Resources Research*, vol. 46, no. 4, pp. 1–16, 2010.

[21] D. Birant, "Spatio-Temporal Outlier Detection in Large Databases," *Journal of Computing and Information Technology*, vol. 14, no. 4, pp. 291–297, 2006. [Online]. Available: http://cit.zesoi.fer.hr/browsePaper.php?paper=952

[22] T. Cheng and Z. Li, "A multiscale approach for spatio-temporal outlier detection," *Transactions in GIS*, vol. 10, no. 2, pp. 253–263, 2006. [Online]. Available: http://discovery.ucl.ac.uk/76163/

[23] S. Yuxiang, X. Kunqing, M. Xiujun, J. Xingxing, P. Wen, and G. Xiaoping, "Detecting spatio-temporal outliers in climate dataset: a method study," in *Proceedings. 2005 IEEE International Geoscience and Remote Sensing Symposium, 2005. IGARSS '05.*, 2005, pp. 760–763. [Online]. Available: http://dx.doi.org/10.1109/igarss.2005.1525218

[24] E. Wu, W. Liu, and S. Chawla, "Spatio-temporal outlier detection in precipitation data," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5840 LNCS, pp. 115–133, 2010.

[25] W. Drosdowsky, "An Analysis of Seasonal Rainfall Anomalies - 1950-1987," *International Journal of Climatology*, vol. 13, pp. 1–30, 1993.

[26] R. Lasaponara, "On the use of principal component analysis (PCA) for evaluating interannual vegetation anomalies from Spot/Vegetation NDVI temporal series," *Ecological Modelling*, vol. 194, no. 4, pp. 429–434, 2006.

[27] C. T. Lu and L. R. Liang, "Wavelet fuzzy classification for detecting and tracking region outliers in meteorological data," *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pp. 258–265, 2004. [Online]. Available: http://dx.doi.org/10.1145/1032222.1032260

[28] S. Basu and M. Meckesheimer, "Automatic outlier detection for time series: An application to sensor data," *Knowledge and Information Systems*, vol. 11, no. 2, pp. 137–154, 2007. [Online]. Available: http://link.springer.com/article/10.1007/s10115-006-0026-6

[29] A. Nairac, N. Townsend, R. Carr, S. King, P. Cowley, and L. Tarassenko, "A system for the analysis of jet engine vibration data," *Integrated Computer-Aided Engineering*, vol. 6, no. 1, pp. 53–66, 1999. [Online]. Available: http://iospress.metapress.com/content/YWAUBUE89WBW8X9L

[30] D. Dasgupta and S. Forrest, "Novelty detection in time series data using ideas from immunology," *Proceedings of the International Conference on Intelligent Systems*, pp. 82–87, 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.3894{&}amp;rep=rep1{&}amp;type=pdf

[31] Y. Bu, T. Leung, A. Fu, E. Keogh, J. Pei, and S. Meshkin, "Wat: Finding top-k discords in time series database," *In Proceedings of the 2007 SIAM International Conference on Data Mining (SDM'07)*, pp. 26–28, 2007. [Online]. Available: http://www.cse.cuhk.edu.hk/{~}adafu/Pub/sdm07.pdf

[32] E. Keogh, J. Lin, and A. Fu, "HOT SAX: Efficiently finding the most unusual time series subsequence," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 226–233, 2005.

[33] L. Wei, E. Keogh, and A. Xi, "SAXually explicit images: Finding unusual shapes," *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 711–720, 2006.

[34] D. Yankov, E. Keogh, and U. Rebbapragada, "Disk aware discord discovery: Finding unusual time series in terabyte sized datasets," *Knowledge and Information Systems*, vol. 17, no. 2, pp. 241–262, 2008.

[35] X. Li, Z. Li, J. Han, and J.-g. Lee, "Temporal Outlier Detection in Vehicle Traffic Data," in *IEEE 25th International Conference on Data Engineering*, no. July, 2009, pp. 1319–1322. [Online]. Available: http://web.engr.illinois.edu/{~}hanj/pdf/icde09{_}xli.pdf

[36] Y. Ge, H. Xiong, Z.-h. Zhou, H. Ozdemir, J. Yu, and K. C. Lee, "TOP-EYE : Top- k Evolving Trajectory Outlier Detection," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010, pp. 1733–1736.

[37] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: A novel symbolic representation of time series," *Data Mining and Knowledge Discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[38] A. W.-c. Fu, O. T.-W. Leung, E. Keogh, and J. Lin, "Finding time series discords based on haar transform," in *Proceedings of the Second international conference on Advanced Data Mining and Applications*. Springer-Verlag, 2006, pp. 31–41.

[39] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, 2013.

[40] S. Salvador and P. Chan, "Learning states and rules for detecting anomalies in time series," *Applied Intelligence*, vol. 23, pp. 241–255, 2005.

[41] M. V. Mahoney and P. K. Chan, "Trajectory boundary modeling of time series for anomaly detection," 2005.

[42] P. K. Chan and M. V. Mahoney, "Modeling multiple time series for anomaly detection," *IEEE International Conference on Data Mining*, pp. 90–97, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1565666$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=1565666

[43] S. Muthukrishnan, R. Shah, and J. Vitter, "Mining deviants in time series data streams," *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, pp. 41–50, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1311192

[44] H. V. Jagadish, N. Koudas, and S. Muthukrishnan, "Mining Deviants in a Time Series Database," in *Proceedings of the 25th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., 1999, pp. 102–113.

[45] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A Survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, jul 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1541880.1541882

[46] B. Ng, "Survey of Anomaly Detection Methods," 2006. [Online]. Available: http://www.osti.gov/scitech/biblio/900157-VDshbd/

[47] M. Längkvist, L. Karlsson, A. Loutfi, M. L. Ã. Ngkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, no. C, pp. 11–24, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2014.01.008$\ delimiter"026E30F$npapers3://publication/doi/10.1016/j.patrec.2014.01.008

[48] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '02*, p. 102, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=775047.775062

[49] P. Protopapas, J. M. Giammarco, L. Faccioli, M. F. Struble, R. Dave, and C. Alcock, "Finding outlier light-curves in catalogs of periodic variable stars," *Monthly Notices of the Royal Astronomical Society*, vol. 369, no. 2, p. 16, 2005. [Online]. Available: http://arxiv.org/abs/astro-ph/0505495

[50] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Transactions on Database Systems*, vol. 27, no. 2, pp. 188–228, 2002.

[51] A. Hinneburg, C. C. Aggarwal, and D. a. Keim, "What is the Nearest Neighbor in High Dimensional Spaces?" *Proceedings of the 26th VLDB Conference*, pp. 506–515, 2000. [Online]. Available: http://www.informatik.uni-halle.de/{~}hinnebur/PS{_}Files/vldb2000{_}hd{_}similarity.pdf

[52] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful?" *Database TheoryICDT'99*, pp. 217–235, 1999. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-49257-7{_}15

[53] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases," *Knowledge and Information Systems*, vol. 3, no. 3, pp. 263–286, 2001. [Online]. Available: http://research.microsoft.com/pubs/79074/time{_}series{_}indexing.pdf

[54] S. Kitaguchi, "Extracting feature based on motif from a chronic hepatitis dataset," in *Proceedings of 18th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI04)*, 2004.

[55] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining KDD 03*, vol. 304, p. 493, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=956750.956808

[56] J. Bentley and R. Sedgewick, "Fast algorithms for sorting and searching strings," *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pp. 360–369, 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=314321

[57] E. Keogh, E. Keogh, J. Lin, and J. Lin, "Clustering of Time Series Subsequences is Meaningless:," 2004. [Online]. Available: http://citeseer.ist.psu.edu/670978

[58] S. Zolhavarieh, S. Aghabozorgi, and Y. W. Teh, "A Review of Subsequence Time Series Clustering." *TheScientificWorldJournal*, vol. 2014, p. 312521, 2014. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=4130317{&}tool=pmcentrez{&}rendertype=abstract

[59] E. Keogh, J. Lin, S.-H. H. Lee, and H. Van Herle, "Finding the most unusual time series subsequence: Algorithms and applications," *Knowledge and Information Systems*, vol. 11, no. 1, pp. 1–27, 2007.

[60] M. Breunig, H. Kriegel, R. Ng, and J. Sander, "Optics-of: Identifying local outliers," *Principles of Data Mining and ...*, pp. 262–270, 1999. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-48247-5{_}28

[61] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.2930

[62] G. Münz, S. Li, and G. Carle, "Traffic anomaly detection using k-means clustering," in *GI/ITG Workshop MMBnet*, 2007. [Online]. Available: http://www.decom.ufop.br/menotti/rp122/sem/sem3-luciano-art.pdf

[63] Z. He, X. Xu, and S. Deng, "Discovering cluster-based local outliers," *Pattern Recognition Letters*, vol. 24, no. 9-10, pp. 1641–1650, 2003.

[64] X. Zhang, P. Fan, and Z. Zhu, "A new anomaly detection method based on hierarchical HMM," in *Proceedings of the 8th International Scientific and Practical Conference of Students, Post-graduates and Young Scientists. Modern Technique and Technologies. MTT'2002 (Cat. No.02EX550)*, 2003, pp. 249–252. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1236299

[65] Y. Qiao, X. Xin, Y. Bin, and S. Ge, "Anomaly intrusion detection method based on HMM," *Electronics Letters*, vol. 38, no. 13, p. 663, 2002. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el{_}20020467

[66] Y. Qiao, X. W. Xin, Y. Bin, and S. Ge, "Anomaly intrusion detection method based on HMM," *Electronics Letters*, vol. 38, no. 13, pp. 663–664, 2002.

[67] G. Florez-Larrahondo, S. M. Bridges, and R. B. Vaughn, "Efficient Modeling of Discrete Events for Anomaly Detection Using Hidden Markov Models," *Information Security*, vol. 3650, pp. 506–514, 2005. [Online]. Available: http://dx.doi.org/10.1007/11556992{_}38

[68] V. Chandola, V. Mithal, and V. Kumar, "Comparative Evaluation of Anomaly Detection Techniques for Sequence Data," *2008 Eighth IEEE International Conference on Data Mining*, pp. 743–748, 2008.

[69] E. Schubert, A. Zimek, and H. P. Kriegel, "Local outlier detection reconsidered: A generalized view on locality with applications to spatial, video, and network outlier detection," *Data Mining and Knowledge Discovery*, vol. 28, no. 1, pp. 190–237, 2014.

[70] E. M. Knorr and R. T. Ng, "A unified approach for mining outliers," in *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 1997, p. 11.

[71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[72] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[73] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE*, no. 3, pp. 6645–6649, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778

[74] N. Boulanger-Lewandowski, P. Vincent, and Y. Bengio, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription," *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, no. Cd, pp. 1159–1166, 2012.

[75] J. Martens, "Generating Text with Recurrent Neural Networks," *Neural Networks*, vol. 131, no. 1, pp. 1017–1024, 2011. [Online]. Available: http://www.icml-2011.org/papers/524{_}icmlpaper.pdf

[76] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, pp. 1–43, 2013. [Online]. Available: http://arxiv.org/abs/1308.0850

[77] Zachary C. Lipton, "A Critical Review of Recurrent Neural Networks for Sequence Learning," pp. 1–35, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019v2

[78] M. L. Minsky, *Computation: Finite and Infinite Machines*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1967.

[79] H. Siegelmann and E. Sontag, "On the Computational Power of Neural Nets," *Journal of Computer and System Sciences*, vol. 50, no. 1, pp. 132–150, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000085710136

[80] ——, "Analog computation via neural networks," *[1993] The 2nd Israel Symposium on Theory and Computing Systems*, pp. 1–2, 1993.

[81] Y. Bengio, I. J. Goodfellow, and A. Courville, "Deep Learning," 2015. [Online]. Available: http://www.iro.umontreal.ca/{~}bengioy/dlbook

[82] M. Hermans and B. Schrauwen, "Training and Analyzing Deep Recurrent Neural Networks," *Advances in Neural Information Processing Systems*, pp. 190–198, 2013.

[83] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to Construct Deep Recurrent Neural Networks," *arXiv preprint arXiv:1312.6026*, pp. 1–10, 2013. [Online]. Available: http://arxiv.org/abs/1312.6026

[84] A. L. Blum and R. L. Rivest, "Training a 3-node neural network is NP-complete," *Neural Networks*, vol. 5, no. 1, pp. 117–127, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005800103

[85] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7700 LECTU, pp. 437–478, 2012.

[86] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," 1991. [Online]. Available: http://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Untersuchungen+zu+dynamischen+neuronalen+Netzen{#}0

[87] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[88] K. Doya, "Bifurcations in the learning of recurrent neural networks," *[Proceedings] 1992 IEEE International Symposium on Circuits and Systems*, vol. 6, pp. 1–4, 1992. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.40.5278{&}rep=rep1{&}type=pdf

[89] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, no. 2, 2013, pp. 1310–1318. [Online]. Available: http://jmlr.org/proceedings/papers/v28/pascanu13.pdf

[90] J. Martens, "Deep learning via Hessian-free optimization," *27th International Conference on Machine Learning*, vol. 951, pp. 735–742, 2010. [Online]. Available: http://www.cs.toronto.edu/{~}asamir/cifar/HFO{_}James.pdf

[91] J. Martens and I. Sutskever, "Learning recurrent neural networks with Hessian-free optimization," *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pp. 1033–1040, 2011.

[92] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in Optimizing Recurrent Networks," 2012. [Online]. Available: http://arxiv.org/abs/1212.0901

[93] Y. Dauphin, H. de Vries, and Y. Bengio, "Equilibrated adaptive learning rates for non-convex optimization," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1504–1512. [Online]. Available: http://papers.nips.cc/paper/5870-equilibrated-adaptive-learning-rates-for-non-convex-optimization.pdf

[94] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[95] Y. Fan, Y. Qian, F. Xie, and F. K. Soong, "TTS Synthesis with Bidirectional LSTM based Recurrent Neural Networks," in *Interspeech-2014*, 2014, pp. 1964–1968.

[96] J. Gonzalez-Dominguez, I. Lopez-Moreno, H. Sak, J. Gonzalez-Rodriguez, and P. J. Moreno, "Automatic Language Identification using Long Short-Term Memory Recurrent Neural Networks," in *Interspeech-2014*, 2014, pp. 2155–2159.

[97] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," *arXiv preprint arXiv:1402.1128*, no. Cd, 2014.

[98] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *NIPS*, p. 9, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215

[99] R. Brueckner and B. Schulter, "Social signal classification using deep blstm recurrent neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 4823–4827.

[100] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant, "The A2iA Arabic Handwritten Text Recognition System at the Open HaRT2013 Evaluation," in *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, apr 2014, pp. 161–165.

[101] O. Vinyals and A. Toshev, "Show and Tell : A Neural Image Caption Generator," *Cvpr*, 2015.

[102] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, and K. Saenko, "Translating Videos to Natural Language Using Deep Recurrent Neural Networks," *arXiv2014*, 2014. [Online]. Available: http://arxiv.org/abs/1412.4729

[103] B. Fan, L. Wang, F. K. Soong, and L. Xie, "PHOTO-REAL TALKING HEAD WITH DEEP BIDIRECTIONAL LSTM." IEEE Institute of Electrical and Electronics Engineers, apr 2015. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=238346

[104] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," in *Proceedings of The 32nd International Conference on Machine Learning*, 2015, pp. 2342–2350. [Online]. Available: http://jmlr.org/proceedings/papers/v37/jozefowicz15.html

[105] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A Search Space Odyssey," *arXiv*, p. 10, 2015. [Online]. Available: http://arxiv.org/abs/1503.04069

[106] Colah, "Understanding LSTM Networks," 2015.

[107] 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.

[108] Z. Li, W. Xu, A. Huang, and M. Sarrafzadeh, "Dimensionality reduction for anomaly detection in electrocardiography: A manifold approach," in *Wearable and Implantable Body Sensor Networks (BSN), 2012 Ninth International Conference on.* IEEE, 2012, pp. 161–165.

[109] L. Wei, N. Kumar, V. Lolla, E. Keogh, S. Lonardi, and C. Ann, "Assumption-free anomaly detection in time series," *Siam*, pp. 1–4, 2005. [Online]. Available: http://www.cs.ucr.edu/{~}wli/publications/WeiL{_}AnomalyDetection.doc$\delimiter"026E30F$nwww.cs.ucr.edu/{~}ratana/SSDBM05.pdf

[110] M. Chuah and F. Fu, "ECG Anomaly Detection via Time Series Analysis," *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops SE - 14*, vol. 4743, pp. 123–135, 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74767-3{_}14

[111] M. Jones, D. Nikovski, M. Imamura, and T. Hirata, "Anomaly Detection in Real-Valued Multidimensional Time Series," *ASEBSC*, pp. 1–9, 2014. [Online]. Available: http://ase360.org/handle/123456789/56

[112] Y. Bengio, L. Yao, G. Alain, and P. Vincent, "Generalized denoising auto-encoders as generative models," *Advances in Neural ...*, pp. 1–9, 2013. [Online]. Available: http://papers.nips.cc/paper/5023-generalized-denoising-auto-encoders-as-generative-models

[113] T. Tieleman and G. Hinton, "Neural Networks for Machine Learning (lecture 6.5)," 2012.

[114] L. Johnson, "theanets," 2015. [Online]. Available: http://theanets.readthedocs.org/en/stable/

[115] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math compiler in Python," in *9th Python in Science Conference*, no. Scipy, 2010, pp. 1–7. [Online]. Available: http://www-etud.iro.umontreal.ca/{~}wardefar/publications/theano{_}scipy2010.pdf

[116] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[117] S. M. Erfani, Y. W. Law, S. Karunasekera, C. a. Leckie, and M. Palaniswami, "Privacy-preserving collaborative anomaly detection for participatory sensing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8443 LNAI, no. PART 1, pp. 581–593, 2014.

[118] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch Normalized Recurrent Neural Networks," *arXiv preprint arXiv:1510.01378*, 2015.

[119] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *arXiv:1409.2329 [cs]*, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329$\delimiter"026E30F$nhttp://www.arxiv.org/pdf/1409.2329.pdf

[120] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv*, pp. 1724–1734, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[121] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=650093

[122] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM networks," *Proceedings of the International Joint Conference on Neural Networks*, vol. 4, pp. 2047–2052, 2005.

[123] N. Laptev and I. Flint, "Generic and Scalable Framework for Automated Time-series Anomaly Detection," *Kdd*, 2015.

[124] M. Lewandowski, J. Martinez-del Rincon, D. Makris, and J. C. Nebel, "Temporal extension of Laplacian Eigenmaps for unsupervised dimensionality reduction of time series," in *Proceedings - International Conference on Pattern Recognition*. Istanbul, Turkey: IEEE Institute of Electrical and Electronics Engineers, 2010, pp. 161–164.

[125] B. Ferrel and S. Santuro, "NASA Shuttle Valve Data," 2005. [Online]. Available: http://www.cs.fit.edu/{~}pkc/nasa/data

[126] V. Stodden, J. Borwein, and D. Bailey, "Setting the default to reproducible," *Computational Science Research. SIAM News*, vol. 46, pp. 4–6, 2013.

[127] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. Miller, W. J. Kent, and A. Nekrutenko, "Galaxy: A platform for interactive large-scale genome analysis," *Genome Research*, vol. 15, no. 10, pp. 1451–1455, 2005.

[128] S. Fomel, P. Sava, I. Vlad, Y. Liu, and V. Bashkardin, "Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments," *Journal of Open Research Software*, vol. 1, no. 1, p. e8, 2013. [Online]. Available: http://openresearchsoftware.metajnl.com/article/view/jors.ag/20

[129] J. Buckheit and D. Donoho, "WaveLab and Reproducible Research," *Wavelets and Statistics*, vol. 103, pp. 55–81, 1995. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4612-2544-7{_}5$\delimiter"026E30F$nhttp://link.springer.com/10.1007/978-1-4612-2544-7

[130] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang, "Bioconductor: open software development for computational biology and bioinformatics." *Genome biology*, vol. 5, no. 10, p. R80, 2004. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=545600{&}tool=pmcentrez{&}rendertype=abstract

Biography

EDUCATION

MSc. Mechanical Engineering 2008 - 2012
Vanderbilt University

Thesis: Thermal Properties of Yttrium Aluminum Garnett From Molecular Dynamics Simulations

BSc. Mechanical Engineering and Economics 1999 - 2003
Vanderbilt University

EXPERIENCE

Intern 2013
Continuum Analytics

Teaching Assistant 2011 - 2012
Vanderbilt University

Power Plant Operations Engineer 2006 - 2007
Saudi ARAMCO

Project Engineer 2003 - 2006
Saudi ARAMCO