DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM)

AKASH SINGH

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

TRITA TRITA-ICT-EX-2017:124

KTH Information and
Communication Technology

Anomaly Detection for Temporal Data using Long Short-Term Memory (LSTM)

AKASH SINGH

Master's Thesis at KTH Information and Communication Technology
Supervisor: Daniel Gillblad
Examiner: Magnus Boman
Industrial Supervisors: Mona Matti, Rickard Cöster (Ericsson)

TRITA-ICT-EX-2017:124

Abstract

We explore the use of Long short-term memory (LSTM) for anomaly detection in temporal data. Due to the challenges in obtaining labeled anomaly datasets, an unsupervised approach is employed. We train recurrent neural networks (RNNs) with LSTM units to learn the normal time series patterns and predict future values. The resulting prediction errors are modeled to give anomaly scores. We investigate different ways of maintaining LSTM state, and the effect of using a fixed number of time steps on LSTM prediction and detection performance. LSTMs are also compared to feed-forward neural networks with fixed size time windows over inputs. Our experiments, with three real-world datasets, show that while LSTM RNNs are suitable for general purpose time series modeling and anomaly detection, maintaining LSTM state is crucial for getting desired results. Moreover, LSTMs may not be required at all for simple time series.

Keywords: LSTM; RNN; anomaly detection; time series; deep learning

Abstrakt

Vi undersöker Long short-term memory (LSTM) för avvikelsedetektion i tidsseriedata. På grund av svårigheterna i att hitta data med etiketter så har ett oövervakat angreppssätt använts. Vi tränar rekursiva neuronnät (RNN) med LSTM-noder för att lära modellen det normala tidsseriemönstret och prediktera framtida värden. Vi undersöker olika sätt av att behålla LSTM-tillståndet och effekter av att använda ett konstant antal tidssteg på LSTM-prediktionen och avvikelsedetektionsprestandan. LSTM är också jämförda med vanliga neuronnät med fasta tidsfönster över indata. Våra experiment med tre verkliga dataset visar att även om LSTM RNN är tillämpbara för generell tidsseriemodellering och avvikelsedetektion så är det avgörande att behålla LSTM-tillståndet för att få de önskade resultaten. Dessutom är det inte nödvändigt att använda LSTM för enkla tidsserier.

Keywords: LSTM; RNN; avvikelsedetektion; tidsserier; djupt lärande

Acknowledgements

I am grateful to Magnus Boman for his time, feedback, and genuine kindness. His guidance during times of struggle was essential in completing this thesis. I would like to thank my supervisor, Daniel Gillblad, for his inputs and ideas for the thesis as well as the future. My warmest regards to Mona Matti and Rickard Cöster for giving me the opportunity to do this thesis at Ericsson and their support. I would also like to extend my gratitude to the entire Machine Intelligence team at Ericsson, Kista, for welcoming me to the team, and their engagement during the project. I would like to appreciate the contribution of my opponents, Staffan Aldenfalk and Andrea Azzini, for their critique of my work. Special thanks to my family and friends for their continuous encouragement and motivation. Finally, I would like to thank my wife, Deepta, without whose love and support none of this would be possible.

Tack!
Thank You!

Contents

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BPTT	Back-propagation Through Time
CEC	Constant Error Carousel
DNN	Deep Neural Network
DRNN	Deep Recurrent Neural Network
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
HTM	Hierarchical Temporal Memory
LSTM	Long Short-Term Memory
MLE	Maximum Likelihood Estimation
MSE	Mean Squared Error
NN	Neural Network
PD	Probability Density
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

In this thesis project, we explore the use of LSTM RNNs for unsupervised anomaly detection in time series. The project was carried out within the Machine Intelligence research group at Ericsson AB, Sweden.

1.1 Anomaly Detection

Anomaly detection refers to the problem of finding instances or patterns in data that deviate from normal behavior. Depending on context and domain these deviations can be referred to as anomalies, outliers, or novelties [1]. In this thesis, the term used is anomalies. Anomaly detection is utilized in a wide array of fields such as fraud detection for financial transactions, fault detection in industrial systems, intrusion detection, and artificial bot listeners identification in music streaming services. Anomaly detection is important because anomalies often indicate useful, critical, and actionable information that can benefit businesses and organizations.

Anomalies can be classified into four categories [1]:

1. Point Anomalies: A data point is considered a point anomaly if it is considerably different from rest of the data points. Extreme values in a dataset lie in this category.

2. Collective Anomalies: If there is a set of related points which are normal individually but anomalous if taken together, then the set is a collective anomaly. Time series sequences which deviate from the usual pattern come under collective anomalies.

3. Contextual Anomalies: If a data point is abnormal when viewed in a particular context but normal otherwise it is regarded as a contextual anomaly. Context is often present in the form of an additional variable e.g. temporal or spatial attribute. A point or collective anomaly can be a contextual anomaly if some contextual attribute is present. Most common examples of this kind

are present in time series data when a point is within normal range but does not conform to the expected temporal pattern.

4. Change Points: This type is unique to time series data and refers to points in time where the typical pattern changes or evolves. Change points are not always considered to be anomalies.

These categories have a significant influence on the type of anomaly detection algorithm employed.

Anomaly detection is considered to be a hard problem [1], [2]. Anomaly is defined as a deviation from normal pattern. However, it is not easy to come up with a definition of normality that accounts for every variation of normal pattern. Defining anomalies is harder still. Anomalies are rare events, and it is not possible to have a prior knowledge of every type of anomaly. Moreover, the definition of anomalies varies across applications. Though it is commonly assumed that anomalies and normal points are generated from different processes.

Another major obstacle in building and evaluating anomaly detection systems is the lack of labeled datasets. Though anomaly detection has been a widely studied problem, there is still a lack of commonly agreed upon benchmark datasets [2]. In many real-world applications anomalies represent critical failures which are too costly and difficult to obtain. In some domains, it is sufficient to have tolerance levels, and any value outside the tolerance intervals can be marked as an anomaly. Though in many cases labeling anomalies is a time-consuming process and human experts with knowledge of the underlying physical process are required to annotate anomalies.

Anomaly detection for time series presents its own unique challenges. This is mainly due to the issues inherent in time series analysis which is considered to be of the ten most challenging problems in data mining research [3]. In fact, time series forecasting is closely related to time series anomaly detection, as anomalies are points or sequences which deviate from expected values [4].

1.2 Deep Learning

In recent years, deep learning has emerged as one of the most popular machine learning techniques, yielding state-of-the-art results for a range of supervised and unsupervised tasks. The primary reason for the success of deep learning is its ability to learn high-level representations which are relevant for the task at hand. These representations are learned automatically from data with little or no need of manual feature engineering and domain expertise. For sequential and temporal data, LSTM RNNs have become the deep learning models of choice because of their ability to learn long-range patterns.

Due to the problems in collecting labels, anomaly detection is mostly an unsupervised problem. Though most of the recent focus and success of deep learning research has been in supervised learning, unsupervised learning is expected to receive greater importance in the coming years [5].

1.3 Problem and Contribution

As discussed above there are several challenges inherent to anomaly detection. These are related to the definition of anomalies, evaluating anomaly detection algorithms, and availability of datasets. In the course of the project, we faced the same problems which influenced our choice of datasets and evaluation method. We discuss these choices in later sections. A different problem is related to the understanding of LSTM. Though LSTM has become the machine learning model of choice for sequential data, its working and limitations are still quite poorly understood. Owing to its complex architecture, LSTM is viewed as a black box with little clarity of the role and significance of the different components.

Our work has three contributions. First, we explain the need for LSTMs and study the LSTM architecture to illustrate why LSTMs are suitable for sequential and temporal data. Second, we explore how LSTMs can be used as general purpose anomaly detectors to detect a variety of anomalies. Third, we show how different parameters and architecture choices affect the performance of LSTMs.

1.4 Purpose and Goal

There have been only a few attempts at using LSTMs for unsupervised anomaly detection. In this thesis project we attempt to provide an improved understanding of LSTMs for time series modeling and anomaly detection. The aim is not to develop a superior anomaly detection algorithm, but to rather understand what makes LSTMs a good choice for time series modeling and anomaly detection, as well as their limitations for the purpose.

The goal of the thesis project is to help Ericsson in understanding the suitability of LSTMs for anomaly detection. Hence, the exploratory nature of the research presented here. The Machine Intelligence research group at Ericsson has been working with anomaly detection algorithms with the aim to incorporate them into Ericsson's numerous products and services. The outcome of the project would be an algorithm for time series anomaly detection, its implementation, and an analysis of its performance on different datasets.

1.5 Ethics and Sustainability

Anomaly detection is often used to safeguard against many illegal activities, e.g. fraud prevention/detection in financial transactions, and intrusion detection in security systems. Since we used publicly available datasets, there were no concerns regarding the collection of data from human subjects. Use of public datasets also allays privacy concerns. We do not claim superiority over other works, do not falsify any results or data, and take appropriate caution to avoid plagiarism and give proper citations wherever needed.

It is worth surveying the direct and indirect impact of anomaly detection techniques on the environment.

- First order effects: Anomaly detection systems are software systems, and there is no direct impact on environment or concerns regarding production, waste, harmful by-products, or pollution. Though anomaly detection systems will require computer hardware resources to be implemented, the energy consumption and other effects of such systems are negligible.
- Second order effects: Anomaly detection systems have a positive effect on organizations and parties that implement them. They can prevent monetary loss by detecting financial fraud, and other illegal activities in online systems. They also serve to reduce wastage and increase productivity by detecting faults in industrial machines, thereby allowing corrective actions to be taken in time.
- The anomaly detection algorithm used in this project is based on machine learning technologies. There is an ongoing debate about how increasing adoption of machine learning and artificial intelligence (AI) could be detrimental to society. These ill effects range from loss of many manual jobs to even an “all-knowing evil AI”. We do concede that some of these fears are not entirely unfounded, but discussion of such debates is beyond the scope of this report. As far as anomaly detection is concerned, it is mostly utilized when a manual inspection is not possible either due to the scale or the complexity of the task.

1.6 Methodology

In this project, we primarily used quantitative research methods as described in [6]. First, a literature study was carried out to find the main challenges in the research area as well as to make use of the recent research developments in the problem area. Experimental research methods were employed along with deductive approaches. The implementation of the algorithm was done in an exploratory and iterative nature, moving from simple to more sophisticated techniques. The algorithm was evaluated on a variety of datasets, to ascertain its efficacy. Rigorous quality assurance was performed to ensure that the software libraries were used

Table 1.1: Software libraries used in the project.

Library	Version
Keras	2.0.3
TensorFlow	1.0.0
sickit-learn	0.18.2
GPyOpt	1.0.3

correctly and the code did not suffer from any bugs or defects which could have affected the outcome of the experiments. The results were analyzed carefully to evaluate the utility of LSTMs for our purpose. We discuss the pros and cons of the proposed method, make recommendations about its use, and provide suggestions for future work.

Python programming language (version 2.7.12) was used to write the code for the project. The main software libraries used and their versions are listed in table 1.1. In order to support reproducibility of the research, the entire code base used for the project has been made available on GitHub[1].

1.6.1 Project Environment

The work was carried out within the Machine Intelligence research group at Ericsson at its premises in Kista, Sweden. The group focuses on applying machine learning and related technologies on problems relevant to Ericsson. I joined the team to work on a specific project that Ericsson is doing in collaboration with Swedish academia and industry. The project focuses on anomaly detection and predictive maintenance using sensor data from the manufacturing industry.

1.7 Delimitations

In this project, we focus only on unsupervised anomaly detection using LSTMs and do not discuss other alternative techniques. While multiple software packages provide implementations of LSTMs, we use Keras[2] with TensorFlow[3] backend. Keras provides an high-level API for neural networks enabling quick experimentation. Keras is easy to use but can be quite restrictive, and building custom implementations is not straightforward. This had a bearing on the way LSTMs are employed in the project and is explained in section 3.2. Another option was to directly use TensorFlow which provides a more flexible and lower level API as compared to Keras. However, TensorFlow's API had recently undergone significant changes. We found

[1]https://github.com/akash13singh/lstm_anomaly_thesis
[2]https://keras.io/
[3]https://www.tensorflow.org/

the documentation had not been updated and some important details were missing. Since the project did not require a custom LSTM implementation, we decided to use Keras. Reasons for other choices including the selection of datasets, metrics, and anomaly detection methods are provided in the corresponding sections.

1.8 Outline

The rest of the report is organized as follows: In chapter 2 we give an overview of the theory and concepts that are essential to understanding the work done in rest of the project. This chapter also includes a section on related work. Next, we introduce the model and methods use in this project in chapter 3. Then we present our experiments, results, and evaluation in chapter 4. A discussion on the method used, experiments, and results follows in chapter 5. Finally, we conclude the report in chapter 6 and also give some ideas for future work.

Chapter 2

Relevant Theory

2.1 Neural Networks

A neural network (NN) is a machine learning model inspired by the functioning and structure of a biological brain. An NN comprises of simple computational units called *nodes* or *neurons*. A neuron receives inputs along the incoming edges, multiplies the inputs by corresponding edge weights, and then applies a non-linear function called *activation function*, to the weighted sum and produces an output. The working of a neuron is illustrated in figure 2.1 and can be represented mathematically by the vector equation 2.1, where $\boldsymbol{x}, \boldsymbol{w}, b, \odot, f, y$ represent input vector, weight vector, neuron bias, element-wise multiplication, activation function, and neuron output respectively.

$$y(x) = f(\boldsymbol{w} \odot \boldsymbol{x} + b) \tag{2.1}$$

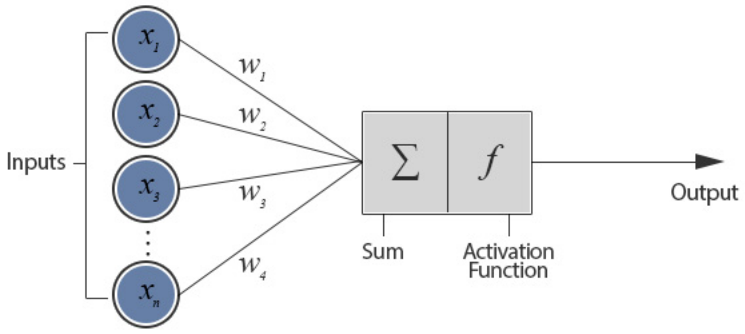


Figure 2.1: **Functioning of a neuron.** The output of a single neuron is a non-linear function of the weighted sum of its inputs. The non-linearity is introduced by the activation function. Image adapted from [7].

Typical activation functions include: *logistic sigmoid* (σ), *tanh*, and *rectified linear units (ReLU)* [5]. The functions are defined by equations 2.2, 2.3, and 2.4

respectively. For regression problems which require predicting continuous values, *linear activation* is used. Linear activation applies the identity function shown in equation 2.5.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.3}$$

$$ReLU(z) = max(0, z) \tag{2.4}$$

$$a(z) = z \tag{2.5}$$

The basic feed-forward network is shown in figure 2.2 and comprises of several neurons, also called *units*, organized in layers to form a network. Neurons in each layer are connected to all the neurons in the previous layer by a set of directed edges. Each edge has a corresponding weight associated with it. The first layer receives the input and is called the *input layer*. The last layer termed the *output layer* produces the output of the NN. The remaining layers are collectively referred to as *hidden layers*. Since the flow of information is from the input layer to the output layer, a hierarchy is implied in the layer structure. The input layer is also referred to as the bottom layer and the output layer as the top layer. The outputs of each neuron are calculated while moving up from the bottom layer until the output of the network is produced at the top layer.

Feed-forward NNs as described above are used for supervised learning tasks [5]. During training, the network is presented with input data along with outputs. A *loss function* which measures the distance between network output and the desired output is constructed to facilitate learning. The loss function commonly used for a regression problem (i.e. predicting a continuous value) is *mean squared error (MSE)*. MSE is computed as shown in equation 2.6, where N is the number of observations, y_i denotes the true value, and the predicted value is denoted by $\hat{y}_i$. MSE measures the averaged squared distance between the predicted values and true values. The difference between the true value and predicted value is also referred to as the *error* or *residual*.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.6}$$

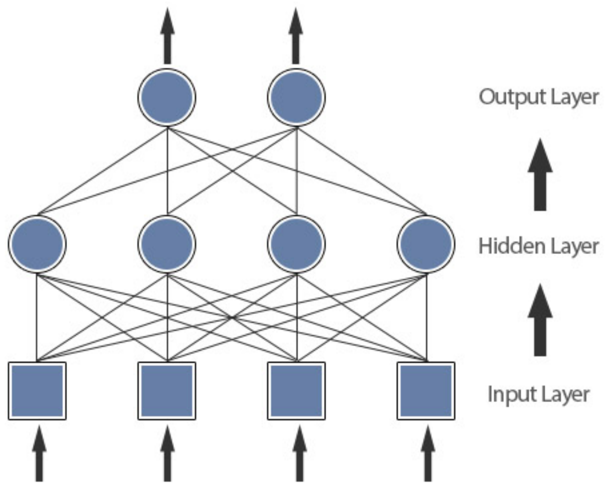


Figure 2.2: **A feed-forward NN with one hidden layer.** The neurons are represented by circles. Each neuron in a layer is connected to all the neurons in the previous (bottom) layer. The input layer nodes are not technically neurons as they forward the input signal without any processing. Image adapted from [7].

2.1.1 Training NNs

The learning problem is converted into an optimization (error minimization) exercise with the goal to minimize the loss function by tuning the parameters of the NN. The optimization algorithm used to train NNs is called *gradient descent*. Gradient descent involves calculating the gradients of the loss function with respect to the network parameters i.e. weights and biases. The method used to compute the gradients is called *back-propagation* [8] and is based on the chain rule of derivatives. The gradient is a measure of the change in the loss value corresponding to a small change in a network parameter. A scalar value called the *learning rate* (γ) is used to update the parameters (θ) in opposite direction of the gradient, according to equation 2.7. The process is done iteratively by making several passes over the training data. A pass over training data is called an *epoch* and after every epoch the parameters move closer to their optimum values which minimizes the loss function.

$$\theta = \theta - \gamma \frac{\partial L(\theta)}{\partial \theta} \tag{2.7}$$

If the dataset size is large, calculating the loss and gradient over the entire dataset may be too slow and computationally infeasible. Thus in practice, a variant of gradient descent called *stochastic gradient descent* (*SGD*) is commonly used. In SGD data is divided into subsets called *batches*, and the parameters are updated after calculating the loss function over one batch. Other popular variants are: RMSprop, AdaGrad, Adam [9]. In some of these variants, an additional parameter called *decay* is used to decrease the learning rate gradually as parameters approach the optimum values.

An often encountered problem in training NNs is *overfitting*. Overfitting occurs when the model tries to fit the noise in training data and is often the result of using a more complex model than required. In the presence of overfitting, model performs well on training data but poorly on new data. There are several ways to prevent overfitting. In *early-stopping* a small subset of training data is used as a validation set. After every epoch the value of the loss function on the training set is compared to the value on the validation set. If the loss on the validation set starts increasing even though the loss on the training set is decreasing, it is an indication of overfitting, and the model training can be stopped. Another method commonly used in deep learning is *dropout*. In dropout, a fixed percentage of NN connections are removed randomly in each training epoch.

It is important to note that network parameters (weights and biases) are learned by the training algorithm. On the other hand, parameters like learning rate, dropout, training batch size, decay, etc. are parameters of the learning algorithm and need to be set to appropriate values by the user. These latter parameters are collectively termed as *hyper-parameters*. For a detailed study of NNs, gradient descent, and back-propagation one is referred to chapters 5, 6, 7 and 8 of [10].

2.1.2 Deep Learning and Deep Neural Networks

A vital component of traditional machine learning pipelines is feature engineering [11]. Conventional machine learning algorithms require carefully designed features and do not perform well with raw data. However, feature engineering is not straightforward and requires considerable domain expertise. One of the primary reasons for the success of deep learning models is the ability to automatically learn high-level representations relevant for the task at hand. Deep neural networks (DNNs) are NNs with multiple hidden layers stacked together. Each layer is a non-linear module, which receives the output of its previous layer. Progressively more complex/abstract features are learned from bottom to top layer. Thus a DNN is similar to a processing pipeline where each layer does part of the task and hands its output to the next layer.

Deep learning techniques have given state-of-the-art results in a variety of domains from computer vision to language translation [10]. This success has been facilitated by many different factors: availability of large labeled datasets, advances made in computer engineering, distributed systems, and computational power including GPUs.

2.2 Need for RNNs for Sequential Data

Before studying RNNs it would be worthwhile to understand why there is a need for RNNs and the shortcoming of NNs in modeling sequential data.

One major assumption for NNs and in fact many other machine learning models is the independence among data samples. However this assumption does not hold for data which is sequential in nature. Speech, language, time series, video, etc. all exhibit dependence between individual elements across time. NNs treat each data sample individually and thereby lose the benefit that can be derived by exploiting this sequential information. One mechanism to account for sequential dependency is to concatenate a fixed number of consecutive data samples together and treat them as one data point, similar to moving a fixed size sliding window over data stream. This approach was used in [12] for time series prediction using NNs, and in [13] for acoustic modeling. But as mentioned in [12] the success of this approach depends on finding the optimal window size: a small window size does not capture the longer dependencies, whereas a larger window size than needed would add unnecessary noise. More importantly, if there are long-range dependencies in data ranging over hundreds of time steps, a window-based method would not scale. Another disadvantage of conventional NNs is that they cannot handle variable length sequences. For many domains like speech modeling, language translation the input sequences vary in length.

A hidden Markov model (HMM)[14] can model sequential data without requiring a fixed size window. HMMs map an observed sequence to a set of hidden states by defining probability distributions for transition between hidden states, and relationships between observed values and hidden states. HMMs are based on the Markov property according to which each state depends only on the immediately preceding state. This severely limits the ability of HMMs to capture long-range dependencies. Furthermore, the space complexity of HMMs grows quadratically with the number of states and does not scale well.

RNNs process the input sequence one element at a time and maintain a hidden state vector which acts as a memory for past information. They learn to selectively retain relevant information allowing them to capture dependencies across several time steps. This allows them to utilize both current input and past information while making future predictions. All this is learned by the model automatically without much knowledge of the cycles or time dependencies in data. RNNs obviate the the need for a fixed size time window and can also handle variable length sequences. Moreover, the number of states that can be represented by an NN is exponential in the number of nodes.

2.3 RNNs

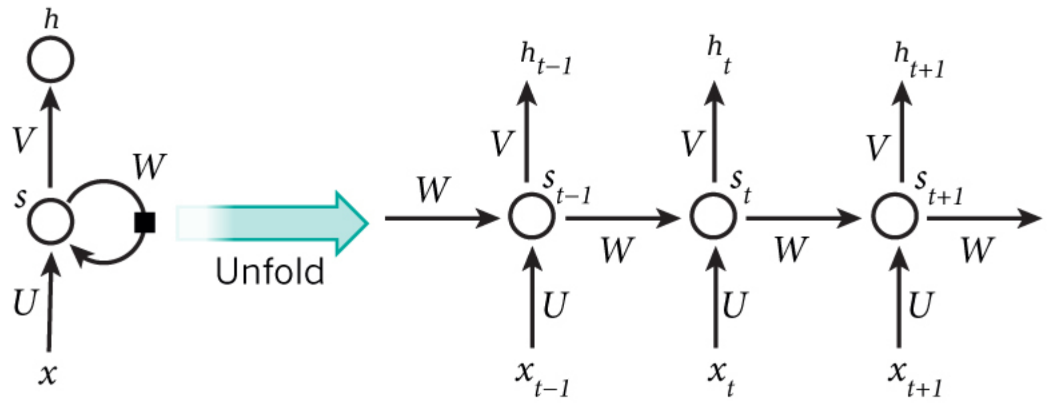


Figure 2.3: **A standard RNN**. The left hand side of the figure is a standard RNN. The state vector in the hidden units is denoted by s. On the right hand side is the same network unfolded in time to depict how the state is built over time. Image adapted from [5].

An RNN is a special type of NN suitable for processing sequential data. The main feature of an RNN is a state vector (in the hidden units) which maintains a *memory* of all the previous elements of the sequence. The most simple RNN is shown in figure 2.3. As can be seen, an RNN has a feedback connection which connects the hidden neurons across time. At time t, the RNN receives as input the current sequence element x_t and the hidden state from the previous time step s_{t-1}. Next the hidden state is updated to s_t and finally the output of the network h_t is calculated. In this way the current output h_t depends on all the previous inputs x_t' (for $t' \leq t$). U is the weight matrix between the input and hidden layers similar to a conventional NN. W is the weight matrix for the recurrent transition between one hidden state to the next. V is the weight matrix for hidden to output transition. Equations 2.8 summarize all the computations carried out at each time step.

$$\begin{aligned} s_t &= \sigma(Ux_t + Ws_{t-1} + b_s) \\ h_t &= softmax(Vs_t + b_h) \end{aligned} \tag{2.8}$$

The *softmax* in 2.8 represents the softmax function which is often used as the activation function for the output layer in a multiclass classification problem. The softmax function ensures that all the outputs range from 0 to 1 and their sum is 1. Equation 2.9 specifies the softmax for a K class problem.

$$y_k = \frac{e^{a_k}}{\sum_{k'=1}^{K} e^{a_{k'}}} for\ k = 1, \ldots, K \tag{2.9}$$

A standard RNN as shown in 2.3 is itself a deep NN if one considers how it behaves during operation. As shown on the right side of the figure, once the network is unfolded in time, it can be considered a deep network with the number of layers equivalent to the number of time steps in the input sequence. Since the same weights are used for each time step, an RNN can process variable length sequences. At each time step new input is received and due to the way the hidden state s_t is updated (equations 2.8), the information can flow in the RNN for an arbitrary number of time steps, allowing the RNN to maintain a memory of all the past information.

2.3.1 Training RNNs

RNN training is achieved by unfolding the RNN and creating a copy of the model for each time step. The unfolded RNN, on the right side of the figure 2.3, can be treated as a multilayer NN and can be trained in a way similar to back-propagation. This approach to train RNNs is called *back-propagation through time (BPTT)* [15].

Ideally, RNNs can be trained using BPTT to learn long-range dependencies over arbitrarily long sequences. The training algorithm should be able to learn and tune weights to put the right information in memory. In practice training RNNs is difficult. In fact, standard RNNs perform poorly even when the outputs and relevant inputs are separated by as little as 10 time steps. It is now widely known that standard RNNs cannot be trained to learn dependencies across long intervals [16], [17]. Training an RNN with BPTT requires back-propagating the error gradients across several time steps. If we consider the standard RNN (figure 2.3), the recurrent edge has the same weight for each time step. Thus back-propagating the error involves multiplying the error gradient with the same value over and over gain. This causes the gradients to either become too large or decay to zero. These problems are referred to as *exploding gradients* and *vanishing gradients* respectively. In such situations, the model learning does not converge at all or may take an inordinate amount of time. The exact problem depends on the magnitude of the recurrent edge weight and the specific activation function used. If the magnitude of weight is less than 1 and sigmoid activation (equation 2.2) is used, vanishing gradients is more likely, whereas if the magnitude is greater than 1 and ReLU activation (equation 2.4) is used exploding gradients is more likely [18].

Several approaches have been proposed to deal with the problem of learning long-term dependencies in training RNNs. These include modifications to the training procedure as well as new RNN architectures. In [18] it was proposed to scale down the gradient if the norm of the gradient crosses a predefined threshold. This strategy known as *gradient clipping* has proven to be effective in mitigating the exploding gradients problem. To deal with vanishing gradients problem [18] introduces a penalty term similar to the L1, L2 regularization penalties used to prevent overfitting in NNs. However, as noted using a constraint to avoid vanishing gradients, makes exploding gradients more likely. The LSTM architecture was introduced in

[19] to counter the vanishing gradients problem. LSTM networks have proven to be very useful in learning long-term dependencies as compared to standard RNNs and have become the most popular variant of RNN.

2.4 LSTM

LSTM can learn dependencies ranging over arbitrary long time intervals. LSTM overcome the vanishing gradients problem by replacing an ordinary neuron by a complex architecture called the LSTM unit or block. An LSTM unit is made up of simpler nodes connected in a specific way. The main components of the LSTM architecture introduced in [19] are:

1. Constant error carousel (CEC) : A central unit having a recurrent connection with a unit weight. The recurrent connection represents a feedback loop with a time step equal to 1. The CEC's activation is the internal state which acts as the memory for past information.

2. Input Gate: A multiplicative unit which protects the information stored in CEC from disturbance by irrelevant inputs.

3. Output Gate: A multiplicative unit which protects other units from interference by information stored in CEC.

The input and output gate control access to the CEC. During training, the input gate learns when to let new information inside the CEC. As long as the input gate has a value of zero, no information is allowed inside. Similarly, the output gate learns when to let information flow from the CEC. When both gates are closed (activation around zero) information or activation is trapped inside the memory cell. This allows the error signals to flow across many time steps (aided by the recurrent edge with unit weight) without encountering the problem of vanishing gradients. The problem of exploding gradients is taken care of by gradient clipping as discussed in section 2.3.1.

The standard LSTM as described above performed better than RNNs in learning long-range dependencies. However, in [20] a shortcoming was identified in it. On long continuous input streams without explicitly marked sequence start points and end points, the LSTM state would grow unbounded and eventually cause the network to become unstable. The model would fail to learn the cycles or sequences in data. The LSTM state would not be reset unless the input stream was manually separated into appropriately sized sequences. Ideally, the LSTM should learn to reset the memory cell contents after it finishes processing a sequence and before starting a new sequence. To solve this issue, a new LSTM architecture with *forget gates* was introduced in [20]. Forget gates learn to reset the LSTM memory when starting new sequences. A number of other modifications and variations of the LSTM architecture have been proposed, however as documented in [21] all the variants have similar performance. Since simpler architectures are preferable, we use LSTM with forget gates in this thesis project. We describe this architecture in more detail next.

2.4.1 LSTM with Forget Gates

The architecture of an LSTM unit with forget gates is shown in figure 2.4 and is the architecture used for rest of this report. The main components of the LSTM unit are:

1. Input: The LSTM unit takes the current input vector denoted by x_t and the output from the previous time step (through the recurrent edges) denoted by h_{t-1}. The weighted inputs are summed and passed through tanh activation, resulting in z_t.

2. Input gate: The input gate reads x_t and h_{t-1}, computes the weighted sum, and applies sigmoid activation. The result i_t is multiplied with the z_t, to provide the input flowing into the memory cell.

3. Forget gate: The forget gate is the mechanism through which an LSTM learns to reset the memory contents when they become old and are no longer relevant. This may happen for example when the network starts processing a new sequence. The forget gate reads x_t and h_{t-1} and applies a sigmoid activation to weighted inputs. The result, f_t is multiplied by the cell state at previous time step i.e. s_{t-1} which allows for *forgetting* the memory contents which are no longer needed.

4. Memory cell: This comprises of the CEC, having a recurrent edge with unit weight. The current cell state s_t is computed by forgetting irrelevant information (if any) from the previous time step and accepting relevant information (if any) from the current input.

5. Output gate: Output gate takes the weighted sum of x_t and h_{t-1} and applies sigmoid activation to control what information would flow out of the LSTM unit.

6. Output: The output of the LSTM unit, h_t, is computed by passing the cell state s_t through a tanh and multiplying it with the output gate, o_t.

The functioning of the LSTM unit can be represented by the following set of

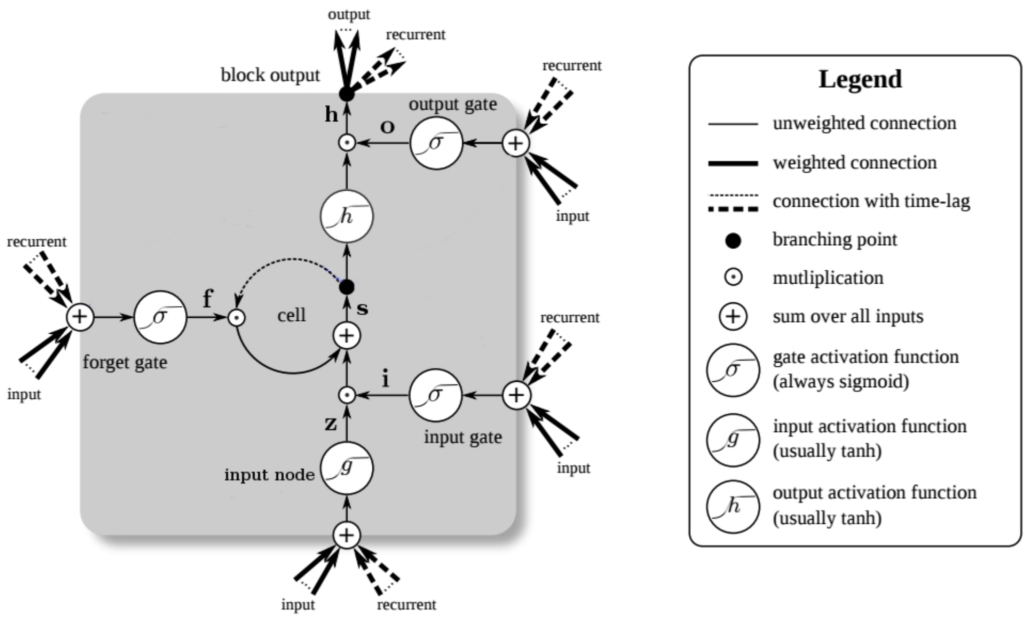


Figure 2.4: **A LSTM unit with forget gates.** A schematic diagram of the LSTM unit with forget gates as introduced in [20]. Image adapted from [21].

equations:

$$\begin{aligned}
z_t &= tanh(W^z x_t + R^z h_{t-1} + b^z) && \text{(input)} \\
i_t &= \sigma(W^i x_t + R^i h_{t-1} + b^i) && \text{(input gate)} \\
f_t &= \sigma(W^f x_t + R^f h_{t-1} + b^f) && \text{(forget gate)} \\
o_t &= \sigma(W^o x_t + R^o h_{t-1} + b^o) && \text{(output gate)} \\
s_t &= z_t \odot i_t + s_{t-1} \odot f_t && \text{(cell state)} \\
h_t &= tanh(s_t) \odot o_t && \text{(output)}
\end{aligned} \tag{2.10}$$

The W^*s are input weights, the R^*s are recurrent weights, and b^*s are the biases.

Note: *From here on we will use the terms RNN and LSTM RNN interchangeably to refer to an RNN with LSTM units. The vanilla RNN architecture presented in section 2.3 will be referred to as standard RNN. Also note that we always use LSTM units with a single memory cell.*

2.5 Deep RNNs

As mentioned in section 2.1.2 the success of deep learning models is due to the ability to learn a hierarchy of simple to complex (abstract) features facilitated by the stacking of several layers. An RNN can be considered a DNN when unrolled in time, with one layer for each time step and the same functions applied to each layer.

However, the purpose of depth in an RNN is different from a DNN. A DNN takes input at the bottom layer, process information through multiple hidden non-linear layers before producing an output. An RNN, on the other hand, takes input and produces an output at every time step with only one non-linear layer between the input and output. Thus in RNN depth only serves the purpose of maintaining a memory of old information, but does not provide hierarchical processing of information as in a DNN. An RNN falls short in two situations [22]: first, in case of complex sequential data which requires hierarchical information processing through many non-linear layers; second, when sequential data like speech or time series contain patterns that need to be processed at different time scales, but RNNs operate at a single time scale.

To deal with these shortcomings RNNs with multiple hidden layers or deep RNNs (DRNN) have been used for speech recognition in [23] and for acoustic modeling in [24]. DRNNs are also referred to as stacked RNNs, to indicate that multiple RNN layers have been stacked together. We use the two terms interchangeably. In a DRNN each layer can have multiple LSTM units, and the output sequence of one layer is fed as the input sequence for the next layer. The hidden states for each layer are computed by as per equation 2.11 [23]:

$$h_t^n = H(W^{h^{n-1}h^n} h_t^{n-1} + R^{h^n h^n} h_{t-1}^n + b_h^n) \tag{2.11}$$

where H is the LSTM function given by equations 2.10; n, ranging from 1 to N, denotes the nth layer of the network; and t denotes the time step. W denotes the feed-forward weights between two layers, and R denotes the recurrent weights from one time step to the next for the same layer. Network input is defined as $h_0 = x$ The network output denoted by y^t is computed as:

$$y_t = W^{h^N y} h_t^N + b_y \tag{2.12}$$

From equations 2.11 and 2.12 one can get some insight how a DRNN offers different time scales. The first layer will build a memory of the input signal. The next layer will develop a memory of the hidden state of the first layer, thus having a memory which goes "deeper" into the past and has also gone through one extra non-linear computation. And so forth for each subsequent layer [22].

2.6 Related Work

2.6.1 Anomaly Detection for Temporal Data

Anomalies in temporal data are contextual anomalies, with time providing the context. As an example consider the simple case of daily power demand of an office:

The demand will be high on weekdays and low on weekends. However, a high demand on a weekend or low demand on a weekday could indicate something unusual. Thus a high (low) value within normal range becomes unusual in context of day of the week. Depending on the domain and use case, one might be interested in point anomalies or collective anomalies. Another important consideration is the dimensionality of data. If the dataset is multidimensional with all features representing different time series, one can use methods for multivariate time series. However, it is equally common to employ methods that disregard the temporal aspect of the data and deal with finding point anomalies in multidimensional space. In this thesis, we consider only univariate datasets, but the model developed can be readily generalized to multivariate time series as well.

One approach for temporal anomaly detection has been to build prediction models and use the prediction errors (the difference between the predicted values and the actual values) to compute an anomaly score [4]. A wide variety of simple to complex prediction models have been employed. In [25] a simple window-based approach is used to calculate the median of recent values as the predicted value, and a threshold on the prediction errors is used to flag outliers. In [26] the authors build a one-step-ahead prediction model. A data point is considered an anomaly if it falls outside a prediction interval computed using the standard deviation of the prediction errors. The authors compare different prediction models: naive predictor, nearest cluster, multilayer perceptron, and a single-layer linear network.

A framework for online novelty detection which calculates a confidence score with each identified novelty is presented in [27]. The authors also develop a concrete algorithm using support vector regression to model the temporal data. A multivariate ARIMA model trained only on normal data (without any anomalies) is used in [28]. Piecewise linear models are used to model time series in [29]. A similarity measure is calculated to compare a new time series against a reference series. Unusual patterns are then highlighted by comparing the similarity measure to a threshold set using the standard deviation of the reference series. A probabilistic approach for anomaly detection in natural gas consumption time series is introduced in [30]. However, the prediction method predicts the consumption levels using other independent variables and does not use the temporal aspect of the data. The prediction model used is linear regression and a Bayesian maximum likelihood classifier trained on known anomalies is used to classify anomalies in new data.

2.6.2 RNNs for Anomaly Detection

Since we use RNNs as the prediction model, we review recent work on using RNNs for temporal anomaly detection in this section. Stacked LSTM RNNs are used for anomaly detection in time series in [31]. The model takes only one time step as input and maintains LSTM state across the entire input sequence. The model is trained on normal data and made to predict multiple time steps. Thus each observation has multiple predictions made at different times in the past. The multiple predictions are used to compute error vectors, which are modeled using a multivariate Gaussian distribution to give the likelihood of an anomaly. The model is tested on four real-world datasets including the power demand dataset on which it achieves a precision and recall of 0.94 and 0.17 respectively. The same approach is also used in [32] to detect anomalies in ECG datasets. As per the authors the results are promising and they find LSTMs to be a viable approach for anomaly detection in ECG signals.

In [33] LSTMs are employed to detect collective anomalies in network security domain. An LSTM RNN with a single recurrent layer is used as a prediction model. A prediction error greater than a set limit indicates a point anomaly. A circular array is maintained to store the most recent prediction errors. Based on the information in the circular array two metrics are calculated: first, the percentage of observations in the array that are anomalies; second, the sum of prediction errors in the array. If both are above specific thresholds the sequence corresponding to these observations is labeled as a collective anomaly. The model is used to detect anomalies in KDD 1999 dataset[1]. The authors report a recall of 0.86 with no false positives. They also note that it is possible to achieve a recall of 1.0 but at the cost of incurring a high number of false positives.

All of the above works use RNNs to model the normal time series pattern and do anomaly detection in an unsupervised manner, with labels only being used to set thresholds on prediction errors. RNNs have also been used for supervised anomaly detection by building models for time series classification. This is a viable approach only if there are sufficient labeled anomalies. LSTM RNNs are used for time series classification in [34] to find anomalies in Border Gateway Protocol data. The authors report that using fixed size input windows and down sampling the time series resulted in improved performance. Another interesting approach has been to combine RNNs with autoencoders for modelling normal time series behaviour. The resulting model reconstructs a time series sequence and the reconstruction errors are used for anomaly detection. This approach is used in [35] for acoustic novelty detection and in [36] for multi-sensor anomaly detection.

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Chapter 3

Methods and Datasets

3.1 The Anomaly Detection Method

The anomaly detection algorithm used in the project consists of two main steps. First, a summary prediction model is built to learn normal time series patterns and predict future time series. Then anomaly detection is performed by computing anomaly scores from the prediction errors.

3.1.1 Time Series Prediction Model

We use LSTM RNN as the time series prediction model. The model takes as input the most recent p values and outputs q future values. We refer to parameters p, q as *lookback* and *lookahead* respectively. The network consists of hidden recurrent layer/layers followed by an output layer. The number of hidden recurrent layers and the number of units in each layer vary for each dataset. Two consecutive recurrent layers are fully connected with each other. To avoid overfitting *dropout* is used between two consecutive layers. The output layer is a fully connected dense NN layer. The number of neurons in the output layer is equal to the *lookahead* value, with one neuron for each future value predicted. Since the model is used for regression we use linear activation in the output layer and MSE as the loss function. The prediction model is trained only on normal data without any anomalies so that it learns the normal behavior of the time series.

Predicting Multiple Time Steps Ahead: We experiment with predicting multiple time steps into the future (similar to [31]). With a *lookahead* of q at time t the model predicts the next q values of the time series i.e. $t+1$, $t+2$ ..., $t+q$. Predicting multiple time steps is done for two purposes. First, to showcase LSTM's capability as time series modelers, as predicting multiple future values is a harder problem as compared to one-step-ahead prediction. Second, predicting multiple time steps provides an early idea of the future behavior. It could even be possible to get an early indication of an anomaly. Consider a time series with a scale of 5 minutes. Predicting 6 time steps ahead can give us an idea about the behavior of the time series for next 30 minutes. If there is something unusual, e.g. an extreme value, early alerts can be sent out. However, anomaly detection can happen only when the real input value becomes available. Predicting multiple time steps comes at the cost of prediction accuracy. In our experiments, we use a lookahead of greater than 1 only if the prediction accuracy is still acceptable. The actual lookahead value used is chosen arbitrarily.

3.1.2 Anomaly Detection

Anomaly detection is done by using the prediction errors as anomaly indicators. Prediction error is the difference between prediction made at time $t-1$ and the input value received at time t. The prediction errors from training data are modeled using a Gaussian distribution. The parameters of the Gaussian, mean and variance, are computed using maximum likelihood estimation (MLE). On new data, the log probability densities (PDs) of errors are calculated and used as anomaly scores: with lower values indicating a greater likelihood of the observation being an anomaly. A validation set containing both normal data and anomalies is used to set a threshold on log PD values that can separate anomalies from normal observations and incur as few false positives as possible. A separate test set is used to evaluate the model.

3.1.3 Assumptions

The main assumption we make is that a prediction model trained on normal data should learn the normal time series patterns. When the model is used for prediction on new data, it should have higher prediction errors on regions with anomalies as compared to normal regions. This would enable us to use the log PD values of errors as anomaly scores and set a threshold to separate anomalies from normal data points. Another assumption is that the prediction errors follow a Gaussian distribution.

3.1.4 Algorithm Steps

The LSTM RNN is trained only on normal data to learn normal time series patterns and optimized for prediction accuracy. For this purpose, each dataset is divided into four subsets: a training set, N, with only normal values; validation set, V_N, with only normal values; a second validation set, V_A, with normal values and anomalies; and a test set, T, having both normal values and anomalies. The algorithm proceeds as follows:

1. Set N is used for training the prediction model. We used Bayesian optimization [37] to find the best values for hyper-parameters: lookback, dropout, learning rate, and the network architecture (number of hidden layers and units in each layer). We use a lookahead of more than 1 only if the prediction accu-

racy is still reasonable. If predicting multiple time steps is not required and one needs the best prediction accuracy, lookahead can be set to 1.

2. V_N is used for early stopping to prevent the model from overfitting the training data.

3. Prediction errors on N are modeled using Gaussian distribution. The mean and variance of the distribution are estimated using MLE.

4. The trained prediction model is applied on V_A. The distribution parameters calculated in the previous step are used to compute the log PDs of the errors from V_A. A threshold is set on the log PD values which can separate the anomalies, with as few false alarms as possible.

5. The set threshold is evaluated using the prediction errors from the test set T.

There was a modification required for the actual experiments. As per step 1 above, we train the model and optimize for prediction accuracy on set N. As per our assumption, the model learns the normal time series behavior and should have higher prediction errors on sets V_A, and T, thereby allowing us to do anomaly detection. However as discussed later in section 4.4.2 the parameters which were the best for prediction did not always give good results for anomaly detection. In such cases, we used the parameters from the optimization phase as a starting point. The model was further tuned manually, by repeating steps 1 to 4, until we could detect all anomalies in set V_A.

Bayesian optimization was done using the library GPyOpt[1]. The optimization procedure required us to provide candidate values for each hyper-parameter. Parameters like learning rate, dropout, etc. were given appropriate values as prescribed in literature. For the network architecture we provided three variants chosen arbitrarily.

3.2 Keras

We used Python programming language and Keras to code and implement the experiments for the project. Keras is an open source project which provides a high-level API to implement DNNs and runs on top of other deep learning libraries like TensorFlow and Theano[2]. We used Keras on top of TensorFlow. Keras was chosen as it is designed for fast prototyping and experimentation with a simple API. It allows to configure NNs in a modular way by combining different layers, activation functions, loss functions, and optimizers, etc. Keras provides out of the box solutions for most of the standard deep learning building blocks. However, if someone wants to build a custom or novel implementation, Keras API could be quite limited,

[1]https://sheffieldml.github.io/GPyOpt/
[2]http://deeplearning.net/software/theano/

and libraries like TensorFlow will be a better choice. Keras contains implementation of LSTM with forget gates as described in [20]. There are two important details, explained below, crucial for understanding how LSTMs implemented in Keras function.

3.2.1 BPTT Implementation

In Keras a modified version of BPTT is implemented. To unfold RNN across the entire input sequence consisting of hundreds or even thousands of time steps is computationally inefficient. Thus in Keras RNN is unfolded up to a maximum number of time steps. This parameter is provided through the input mechanism. Input data is fed in the form of a three-dimensional array of shape: (*batch_size*, *lookback*, *input_dimension*). The second argument, *lookback*, specifies the number of time steps for which the RNN is unfolded. The input data is divided into overlapping sequences with a time interval of one. Each sequence has *lookback* number of consecutive time steps and forms one training sample to the RNN model. During training, BPTT is done only over individual samples for *lookback* time steps.

3.2.2 State Maintenance

Keras provides two different ways of maintaining LSTM state.

1. Default Mode: Each sample in a batch is assumed to be independent, and state is only maintained over individual input sequences for *lookback* number of time steps.

2. Stateful Mode: In this option cell state is maintained among the various training batches. The final state of ith sample of the current batch is used as the initial state for ith sample of the next batch. Within a batch, individual samples are still independent. A common mistake when using stateful mode is to shuffle the training samples. However, to maintain state across batches a one-to-one mapping between samples of consecutive batches is assumed, and one should be careful not to shuffle samples.

The independence between individual samples in a batch may seem strange. The motivation for this implementation comes from language modeling and speech recognition tasks, which were key areas driving LSTM development and implementations. In many language modeling tasks, training samples are individual sentences and a short lookback value equal to maximum sentence length (in words) is enough to capture the necessary sequential dependencies. Thus different samples can be treated independently. However, for many domains and datasets, this behavior could be quite restrictive.

3.3 Datasets

As documented in [2], a major problem in anomaly detection research is a lack of labeled benchmark datasets. Many published works either use application specific datasets or generate synthetic datasets [38]. However, both approaches have their pitfalls. With an application specific dataset, it is difficult to judge how well an anomaly detection algorithm would generalize to different datasets. In case of synthetic datasets, there is no real-world validity of the anomalies and performance of the algorithm. To guard against these issues, we chose real-world datasets from different domains. These datasets have been used in previous works on anomaly detection.

3.3.1 Numenta Machine Temperature Dataset

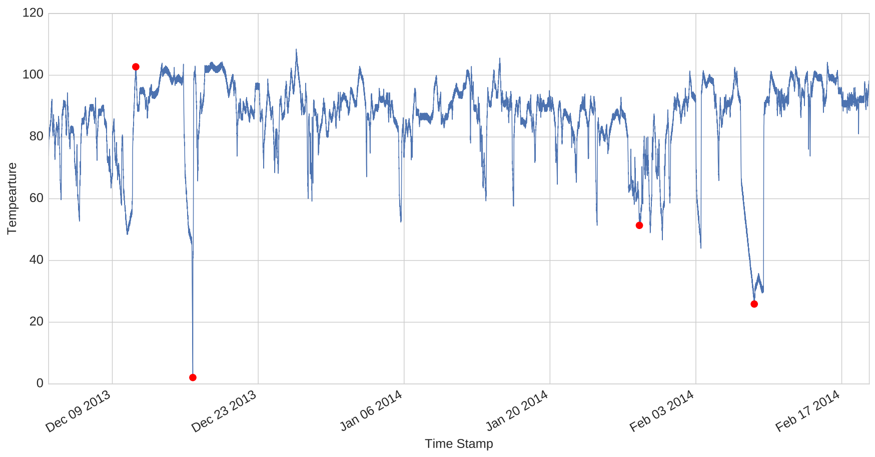
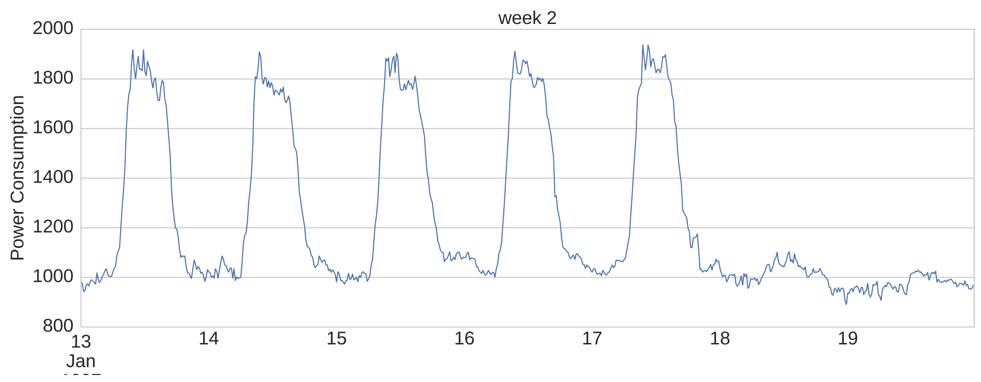


Figure 3.1: **Numenta's Machine Temperature Dataset.** This data contains temperature readings taken every 5 minutes. There are four known anomalies indicated by red markers. The X-axis shows time steps, and the Y-axis measures temperature.

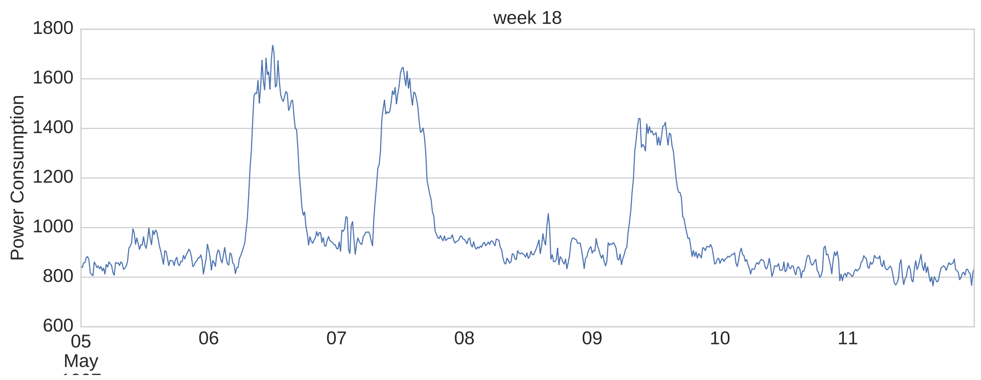
This dataset is taken from [39] and is available at Numenta's GitHub repository[3]. The dataset contains temperature sensor readings of an internal component of a large industrial machine. The readings are for the period between December 2, 2013, to February 19, 2014. There are a total of 22695 readings taken every 5 minutes. There are four anomalies with known causes. The data is shown in figure 3.1, with anomalies indicated in red. The first anomaly is a planned shutdown, and the fourth is a catastrophic failure. The other two anomalies are not visually discernible.

[3]https://github.com/numenta/NAB/tree/master/data

3.3.2 Power Demand Dataset



(a) A weekly cycle with high demand on weekends and low demand on weekdays.



(b) A week with anomalies as Monday and Thursday have low demand.

Figure 3.2: **Power Demand Dataset**. (a) shows a typical week (Monday to Sunday). (b) shows a week with anomalies. The X-axis shows date, and the Y-axis measures the power consumption. Recordings have been taken every 15 minutes.

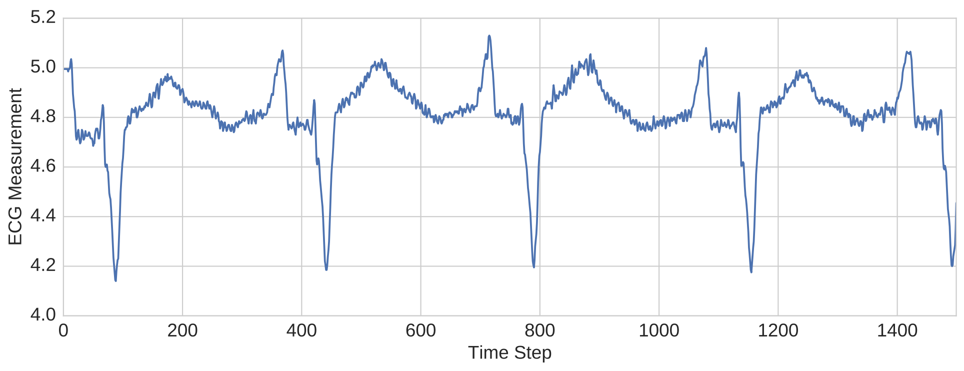
The second dataset records power demand of a Dutch research facility for the year 1997. The readings have been taken every 15 minutes, giving a total of 35040 observations. The data has a long weekly cycle of 672 time steps with five peaks and two lows corresponding to high power consumption on weekdays and low power consumption on weekends. The dataset has been used previously in [31], [40], and [41], where weekdays with a low power demand were considered anomalies. These weekdays coincided with holidays. Similarly, weekends with a high power demand can also be considered anomalies. We use the same approach. Examples of normal and anomalous days are shown in figure 3.2. The dataset can be downloaded from the webpage[4] accompanying [40].

[4]http://www.cs.ucr.edu/~eamonn/discords/

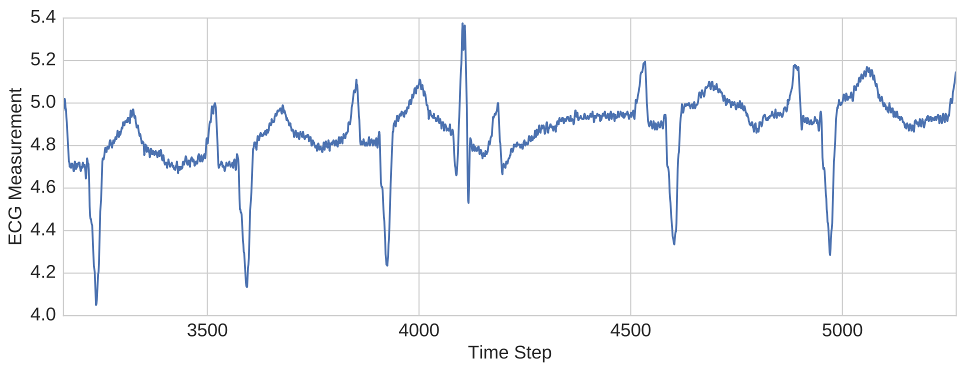
3.3.3 ECG Dataset

ECGs are time series recording the electrical activity of the heart. ECG datasets available at PhysioBank's archive[5] have been used for anomaly detection in [31], [40], [41], and [42] among others. In this report we use the dataset used in [40] as it provides labeled anomalies annotated by a cardiologist. A snippet of dataset showing normal patterns is shown in figure 3.3a. There are a total of 18000 readings with different kinds of anomalies. The three labeled anomalies identified as the most unusual sequences in [40] are shown in figures 3.3b and 3.3c. Though this dataset has a repeating pattern, the length of the pattern varies.

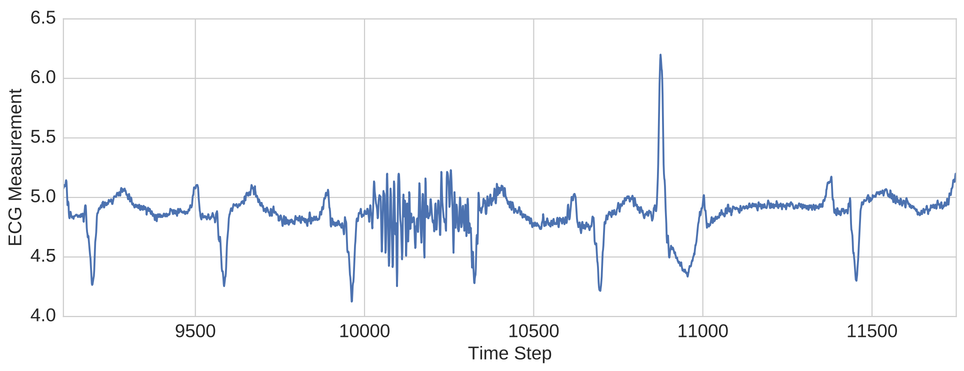
[5]https://physionet.org/physiobank/database/



(a)



(b)



(c)

Figure 3.3: **ECG Dataset**. (a) shows normal heartbeat pattern. (b) shows the first anomaly. (c) shows the other two anomalies. The X-axis shows time steps, while the Y-axis has the ECG measurements. These figures show snippets of the dataset so that the normal and anomalous heartbeats are easily visible

Chapter 4

Experiments and Results

4.1 Main Results

In this section the main results of anomaly detection on each dataset are presented. The prediction models used for different datasets are summarized in table 4.1

4.1.1 Data Pre-processing

For each dataset, the anomalies were divided into sets V_A and T. These sets were then augmented with normal data. Datasets which had a repeating cycle were divided in such a way that the cycles remained intact. The remaining data was divided into sets N and V_N. We also normalized the data to have zero mean and unit variance. The mean and standard deviation of the training set were used to normalize other sets. Finally, each set was transformed into the format required by the algorithm. So each input sample consisted of *lookback* number of time steps.

4.1.2 Numenta Machine Temperature Dataset

We divided the four anomalies equally into sets V_A and T, each of which had about 20% of data. The remaining data was used for training the prediction model.

Model Details: The LSTM RNN used had a lookback of 24, a lookahead of 12, two hidden recurrent layers with 80 and 20 LSTM units respectively, a dense output layer with 12 neurons, and a dropout of 0.1. We trained the prediction model with Adam optimizer using a learning rate of .05, a decay of 0.99, and a batch size of 1024. Training was done for 200 epochs with early stopping. As the data does not contain any repeating patterns we did not maintain LSTM state between batches. This model gave an MSE of 0.09 on N. Using set V_A a threshold of -11 was set on the log PD values. The threshold was then evaluated using set T.

Evaluation: The results of the anomaly detection on sets V_A and T are shown in figures 4.1 and 4.2 respectively. The threshold of -11 was necessary to detect the

Table 4.1: Details of the prediction models used for each dataset. Network architecture specifies the type of layers along with the number of units in each layer, and the dropout values. Linear activation has been used in the output layer. State propagation refers to LSTM state maintenance between batches.

	State Propagation	Network Architecture	Adam Optimizer	Lookback, Lookahead	Batch Size	MSE (Training)

Machine Temperature	No	Recurrent: {80}
		Dropout: 0.1
		Recurrent: {20}
		Dropout: 0.1
		Dense: {12}
		Linear Activation
			Learning Rate: 0.05
			Decay: 0.99
				24,12	1024	0.09

ECG	No	Recurrent: {60}
		Dropout: 0.1
		Recurrent: {30}
		Dropout: 0.1
		Dense: {5}
		Linear Activation
			Learning Rate: 0.1
			Decay: 0.99
				8,5	256	0.10

Power Demand	Yes	Recurrent: {300}
		Dropout: 0.2
		Dense: {1}
		Linear Activation
			Learning Rate: 0.01
			Decay: 0.99
				1,1	672	0.08

first anomaly in set V_A, but incurred a few false positives. On T the threshold of -11 detected the second anomaly but did not detect the first anomaly.

We compare the results of the LSTM algorithm to those of [39] which uses an anomaly detection algorithm based on Hierarchical Temporal Memory (HTM) [43]. The authors define fixed sized *anomaly windows* with each window centered around an anomaly. The earliest detection inside a window is counted as a true positive, whereas any detection outside the window is a false positive. A weighted scoring mechanism is used which gives a higher weight to early detection inside a window. The results of HTM algorithm are shown in figure 4.3. HTM detects three out of four anomalies and has five false positives. In the first anomaly window in figure 4.3c HTM does the detection much before the true anomaly occurs. This anomaly window covers the false positive detections made by the LSTM algorithm in figure 4.2. In fact most of the false positives of LSTM algorithm map to the anomaly windows used by HTM. Thus a window based detection approach can improve the performance of the LSTM algorithm and detect all anomalies with a low false positive rate. However, we did not develop such a detection method, as the particular detection method should be decided by the use case and purpose. It should be noted that HTM algorithm has a distinct advantage of using a small training window (purple shaded portion in figure 4.3a), whereas we used 70% of data for training purpose.

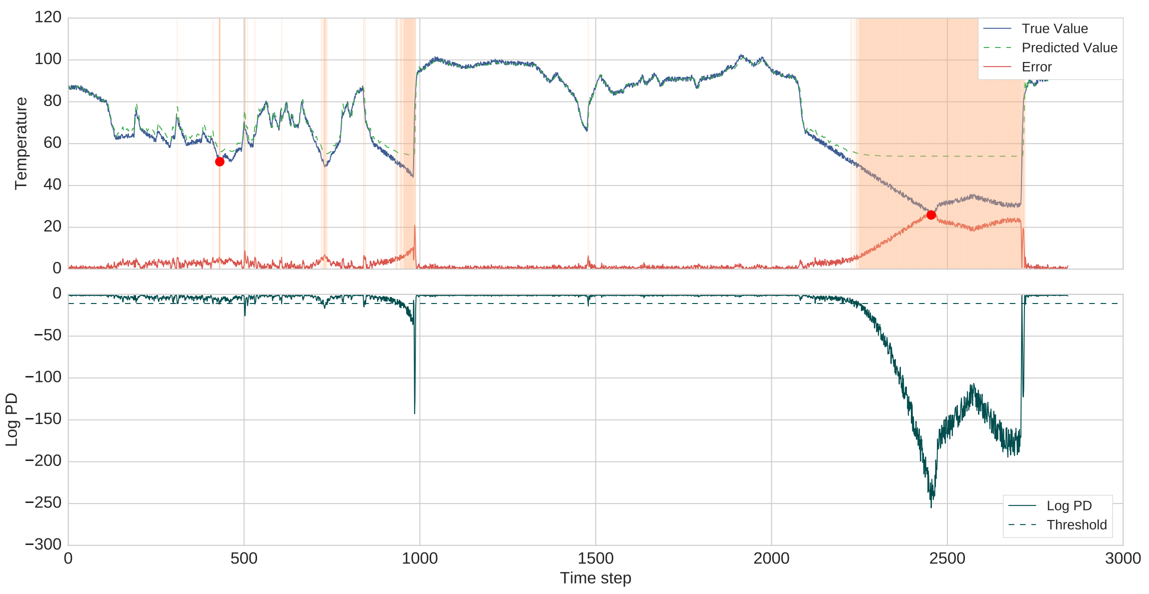


Figure 4.1: **Validation set results on machine temperature dataset.** The top plot shows the predictions done on V_A and the corresponding prediction errors. The bottom plot shows the log PD values of the prediction errors and the threshold set at -11. The X-axis shows time steps and the Y-axis has the corresponding metric value. There are two true anomalies highlighted by red markers. Shaded area in the top graph denotes detections made by the LSTM algorithm.

4.1.3 Power Demand Dataset

The dataset contains readings corresponding to 52 weeks. We divided the data such that set N consisted of 32 weeks and V_N consisted of 3 weeks. The remaining weeks were split between sets V_A and T. There were a total of nine anomalies with eight anomalies corresponding to weekdays with low demand and only one weekend as an anomaly with high demand. Out of the nine anomalies, four were allotted to V_A and the remaining to T.

Model Details: The RNN comprised of a single recurrent layer with 300 LSTM units, followed by a dense output layer with 1 neuron, and a dropout of 0.2. Look-back and lookahead were equal to 1. We trained the RNN for 50 epochs with early stopping and used Adam Optimizer with a learning rate of .01 and decay of 0.99, and a batch size of 672. Since the data has long cycles, we maintained the LSTM state between batches. The results on sets V_A and T are shown in figures 4.4 and 4.5 respectively. The prediction model gave an MSE of .08 on N. A threshold of -24 was set using the prediction errors on V_A.

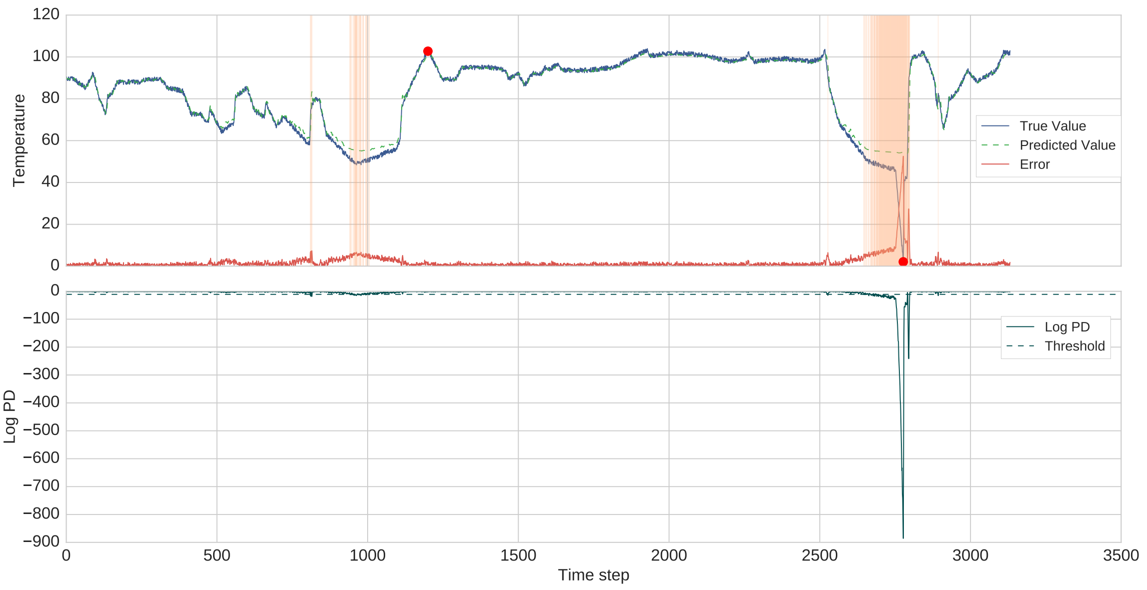
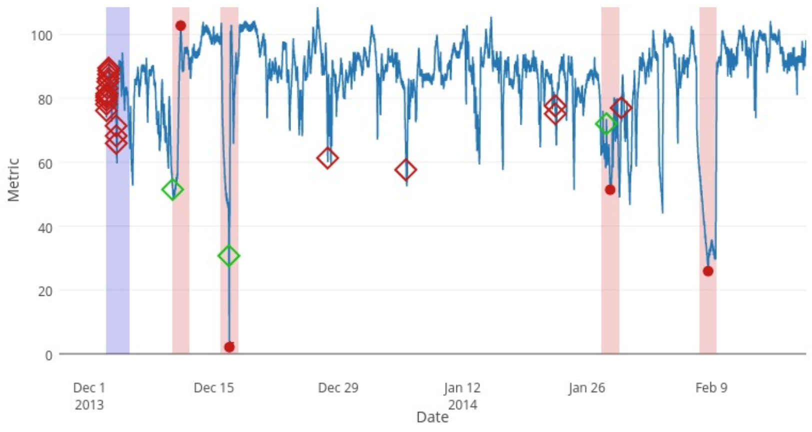


Figure 4.2: **Test set results on machine temperature dataset.** The top plot shows the predictions done on T and the corresponding prediction errors. The bottom plot shows the log PD values of the prediction errors and the threshold set at -11. The X-axis shows time steps and the Y-axis has the corresponding metric value. There are two true anomalies, shown as red markers. Shaded portion of the top graph are regions containing detections made by the LSTM algorithm.

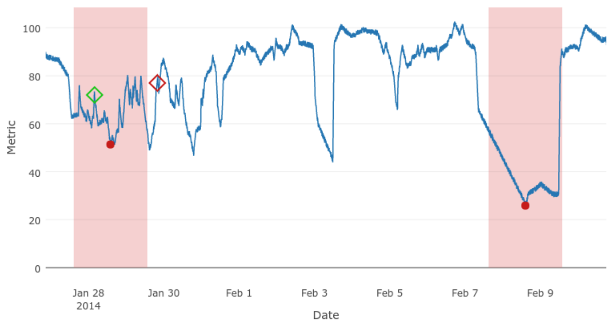
Evaluation: The results for the power demand dataset are shown in figures 4.4 and 4.5. The threshold of -24 detects all four anomalies in set V_A without incurring any false positives. All five anomalies in set T are detected, but there is one false positive. On closer inspection, this false positive turned out to be a sudden peak. This dataset has been used in [40], [41], [31] and [36]. [40] finds the three most unusual weeks, all of which happen to be weeks with two holidays. [31] and [36] use LSTM networks and treat entire weeks with a holiday as an anomaly. They both use *F-score* as the metric to evaluate performance. We did not use a specific metric for detection as this should be governed by the use case and because of the low number of anomalies.

4.1.4 ECG Dataset

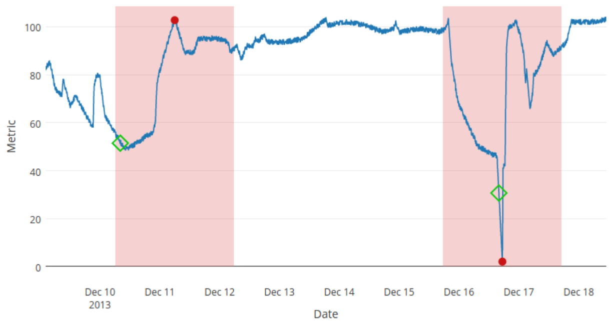
In ECG dataset there are only three labeled anomalies out of 18000 samples. Thus we do not use set V_A for finding threshold. Instead, the dataset was divided into a normal training set (N), a normal validation set (V_N) for early stopping, and a test set (T) containing all three anomalies.



(a)



(b)



(c)

Figure 4.3: **Results of HTM on Machine Temperature Data**. (a) shows the results on entire dataset. (b) shows the results corresponding to figure 4.1. (c) shows the results corresponding to figure 4.2. The X-axis shows time and the Y-axis shows the temperature. True anomalies are denoted by red markers. Green/red diamonds represent true/false positives. The pink shaded portions are anomaly windows. The purple shaded area in (a) is the testing window. Plots generated from code available at NAB GitHub repository.

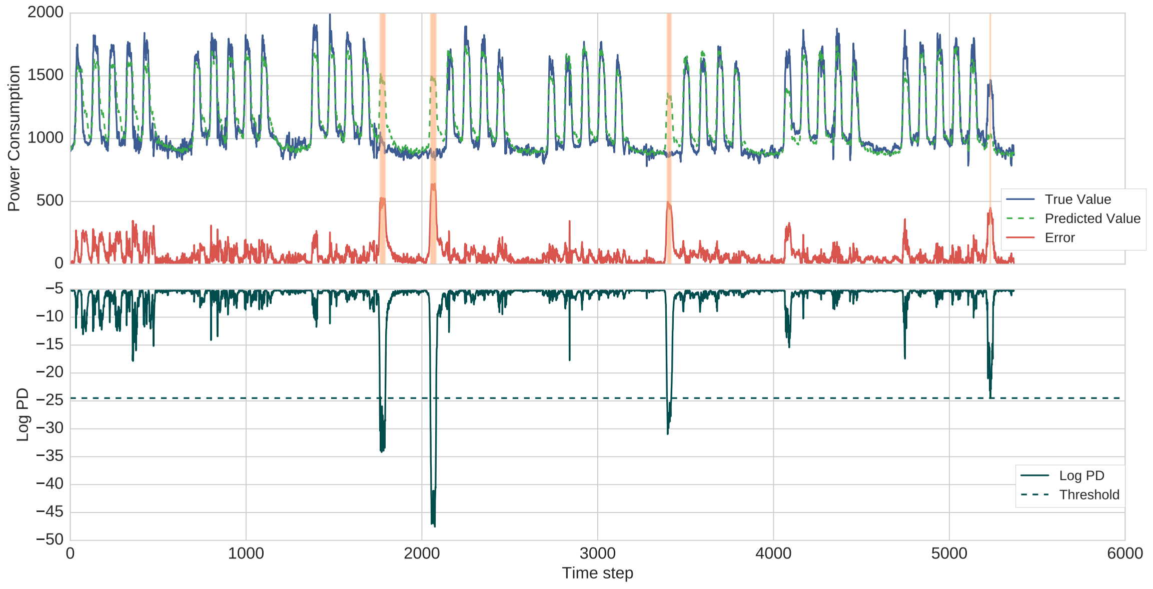


Figure 4.4: **Validation Set Results on Power Demand Dataset.** The top plot shows the predictions done on V_A and the corresponding prediction errors. The bottom plot shows the log PD values of the prediction errors and the threshold. The X-axis shows time steps and the Y-axis has the corresponding metric value. The plots contain eight weekly cycles with one anomaly each in cycles 3, 4 , 6, and 7. The first three anomalies correspond to weekdays having low demand and the last anomaly is a weekend (Saturday) having a high demand. Threshold is set at -24 to detect all anomalies. The shaded area in the top graph highlights regions with anomalies as detected by the algorithm.

.

Model Details: The prediction model used was a stacked RNN consisting of two hidden recurrent layers with 60 and 30 LSTM units respectively, a dense output layer with 5 neurons, a lookback of 8, a lookahead of 5, and a dropout of 0.1. LSTM state was not maintained between batches. We trained the RNN using Adam optimizer with a learning rate of 0.1, a decay of 0.99, and a batch size of 256. The model was trained for 50 epochs with early stopping. The MSE on set N was 0.10 and a threshold of -23 was set on the log PD values of prediction errors from set T.

Evaluation: The results of anomaly detection on set T are shown in figure 4.6. With a threshold of -23 the model detects all three anomalies. These results are similar to the results in [40], which finds the three most unusual sequences in this data. Furthermore, the rankings of the discords match the order of log PD values with a lower value indicating a more unusual sequence.

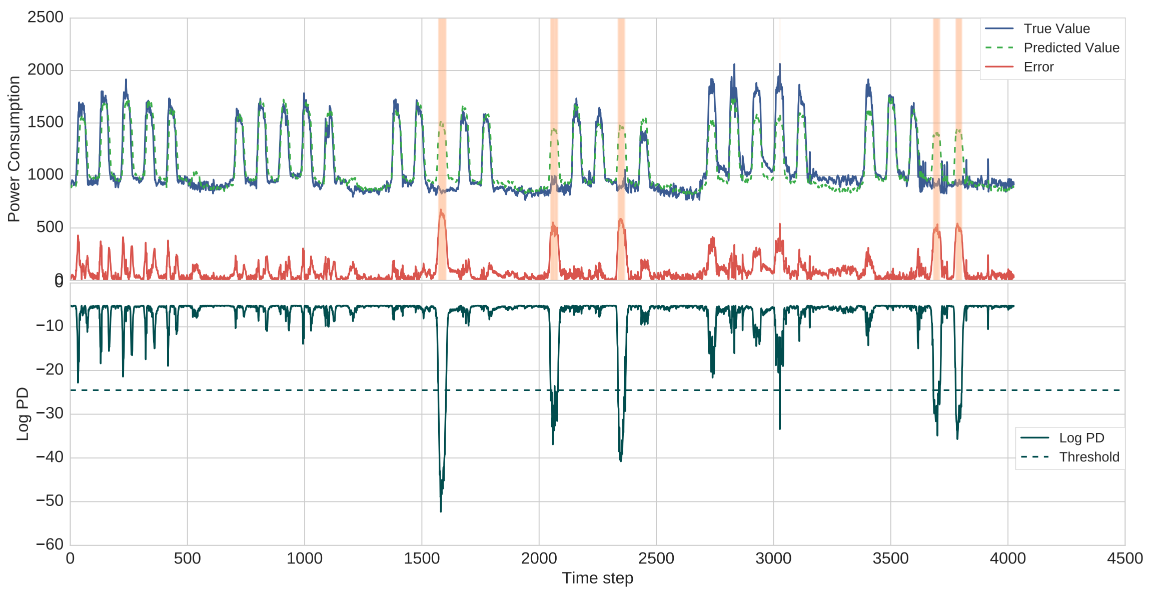


Figure 4.5: **Test Set Results on Power Demand Dataset.** The top plot shows the predictions done on T and the corresponding prediction errors. The bottom plot shows the log PD values of the prediction errors and the threshold set at -24. The X-axis shows time steps and the Y-axis has the corresponding metric value. The plots contain 6 weekly cycles with one anomaly in cycle 3 and two anomalies each in cycles 4 and 6. The anomalies detected by the algorithms are represented by the shaded regions. All 5 anomalies are detected with the set threshold but there is 1 false positive.

4.2 Maintaining LSTM State

An important factor in making LSTMs learn useful patterns from data is the treatment of LSTM state. We experimented with different ways of building LSTM state and their effect on the prediction and detection performance. We discuss the results in this section.

The machine temperature dataset does not seem to have any repeating pattern and indeed the performance of the model was found to be insensitive to how the state was maintained. The ECG dataset has a repeating pattern, of roughly 370 time steps, though the actual length of the pattern varies slightly. We expected that maintaining state between batches would give better results, but results were similar irrespective of how we handled the LSTM state. We believe even though ECG dataset had a long pattern, the relevant temporal relations were present only in few recent time steps.

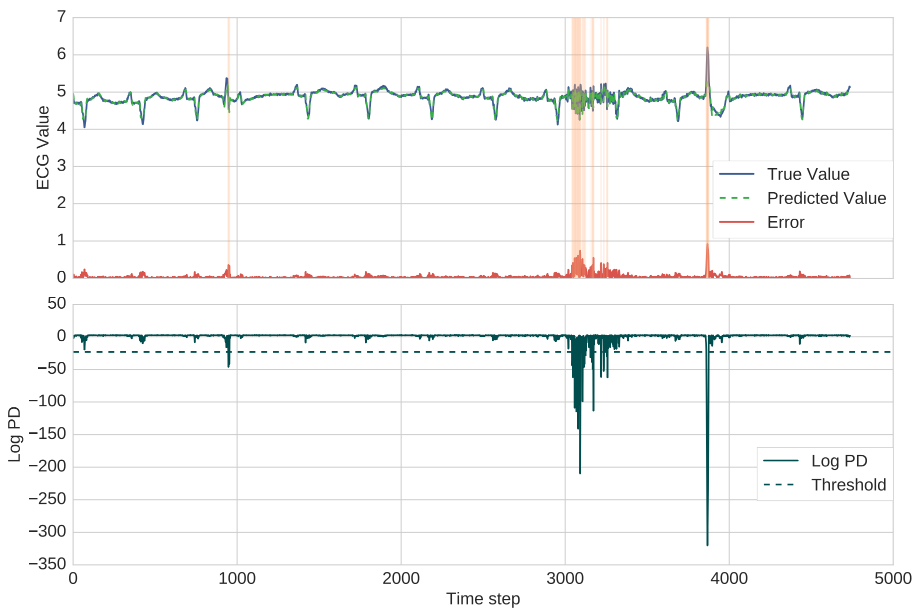
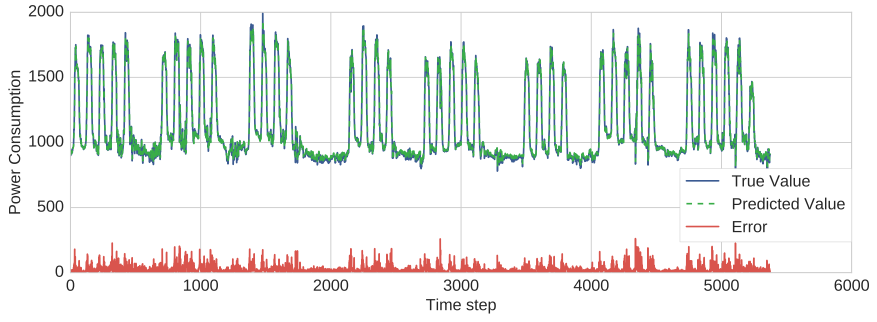
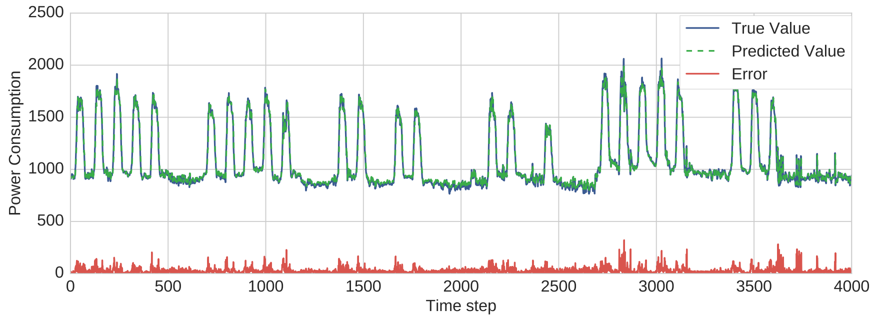


Figure 4.6: **Results for ECG dataset.** The top plot shows the predictions done on T and the corresponding prediction errors. The bottom plot shows the log PD values of the prediction errors and the threshold set at -23. The X-axis shows time steps and the Y-axis has the corresponding metric value. There are three anomalies. The anomalies detected by the algorithms are represented by the shaded regions.

For power demand dataset state maintenance was crucial to get the desired results. We first tried the default mode, where the state was reset after each batch. In all other respects, the prediction model was the same as section 4.1.3. The model had an MSE of 0.0283 on N. More importantly the MSE on sets V_A, and T, which have anomalies, was 0.017 and .019 respectively (contrast this with the results of section 4.1.3 where the MSE on N, V_A, and T was 0.08, 0.15, and 0.26 respectively). While these results denote a significant improvement in prediction, they are not useful for anomaly detection. For anomaly detection to work, we want a low MSE on set N and a higher MSE on sets V_A and T. This is because sets V_A and T contain anomalies and the model should have higher prediction errors on data with anomalies. The results using the default mode (no state maintenance between batches) are shown in figure 4.7. It is clear that the network fails to learn the weekly repeating pattern of five peaks and two lows. For example in figure 4.7a, the third weekly cycle has only four peaks and a low on Friday, but the model predictions follow the actual (anomalous) observations rather than predicting a high power consumption for Friday. Consequently, the resulting prediction errors provide no discrimination between anomalies and normal data points.



(a)



(b)

Figure 4.7: **Prediction on power demand dataset without maintaining state**. (a) shows the results on V_A and (b) shows the results on T. The model fails to learn the weekly pattern as the state information is not maintained. The X-axis has time steps and the the Y-axis shows power consumption.

On closer inspection we found that power demand data has multiple patterns at different time scales: there is a long weekly pattern comprising of 672 time steps, but there is also a daily pattern of 96 time steps. In order to learn these different patterns, LSTM state needs to build a memory of past information and provide a context to enable the model to make correct predictions depending on the day of the week. However, in the default mode the state is wiped clean after every batch. In the absence of any useful input from the state, model learning is influenced almost entirely by recent observations, and the weights are tuned to minimize the prediction errors. This causes the model to fit even anomalous data as it tries to follow the recent observations as much as possible.

Next, we experimented with the stateful mode provided by Keras. This approach was used for the final model presented in section 4.1.3. The state was maintained over all the training batches and was reset only at the start of a new epoch over training data. This allowed the model to build the state for an entire pass over the training data across all the training batches. Though the state was reset before each epoch, the weights of the model continue to be tuned over different epochs, and the LSTM gates learn to store relevant activations in the cell state. As a result, the state provides the necessary input for the model to learn the different patterns and make correct predictions.

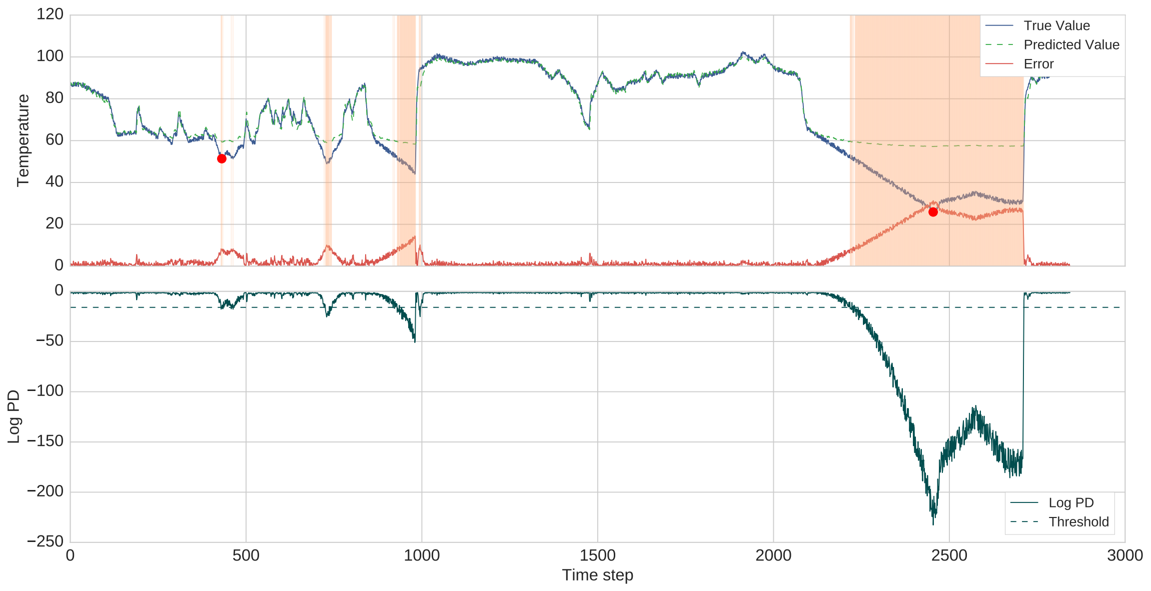
There were a couple of other interesting observations in the stateful mode. First, we found the performance to be sensitive to the batch size. A batch size roughly equal to the length of the weekly cycle was the most effective in learning the long-range pattern. Ideally, the LSTM model should be able to learn the patterns irrespective of the batch size. Second, the performance of the network was similar even if the state was not reset before each epoch. It is usually advised to reset the LSTM state after each epoch as the state may continue to grow and become unstable. However since the datasets we used are quite small (20000 to 40000 observations), and because of the particular Keras' constructs, it is hard to say anything conclusive.

4.3 Feed-forward NNs with Fixed Time Windows

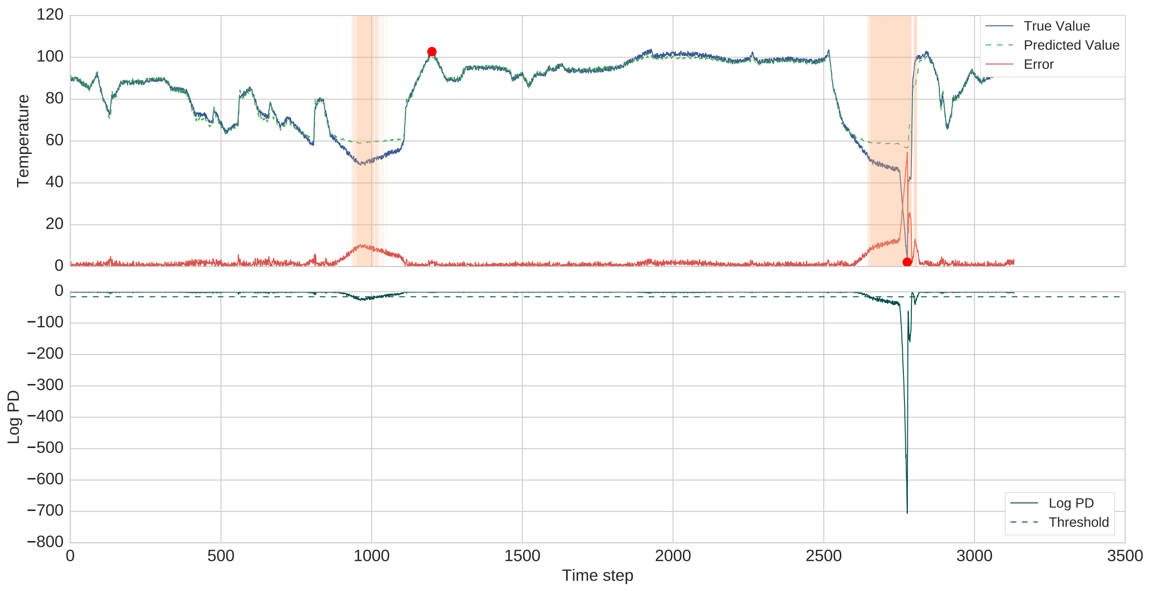
We noticed that state preservation across batches was required only for the power demand dataset. For other two datasets, the state was maintained only across *lookback* number of time steps for individual training samples. We still got reasonable prediction and anomaly detection results. This made us suspect if there was any advantage gained by using a model as complex as LSTM RNN and if feed-forward NNs with fixed size time windows could also capture the temporal relations. So we experimented with using a feed-forward NN as the prediction model.

Figures 4.8a and 4.8b shows the result on sets V_A and T respectively for machine temperature dataset. The prediction model used here has two hidden layers with 32 neurons each, with sigmoid activation, and no dropout. The model was trained with Adam optimizer using a learning rate of .01, batch size of 1024, and early stopping. All other details including the output layer, lookback, and lookahead were same as in the case of LSTM model. The model was trained for 100 epochs and had a MSE of .08 on set N. A threshold of -16 was set using V_A and gave similar results for anomaly detection as compared to LSTM model.

For ECG dataset the NN used had two hidden layers with 64 and 32 neurons respectively, with sigmoid activation, and no dropout. The model was trained with Adam optimizer using a learning rate of .001, batch size of 1024, and early stopping. All other parameters were same as earlier. The model was trained for 80 epochs and gave a MSE of .07 on set N. A threshold of -25 on log PD values of the errors on set T detected all anomalies. The results on ECG dataset are shown in figure 4.9 and are similar to the ones for LSTM model.



(a)



(b)

Figure 4.8: **Results using feed-forward NN on machine temperature data**. (a) shows the results on set V_A. (b) shows the results on set T. A threshold of -16 was set using V_A and the results are similar to figures 4.1 and 4.2. The X-axis shows time step and the Y-axis has the corresponding metric value

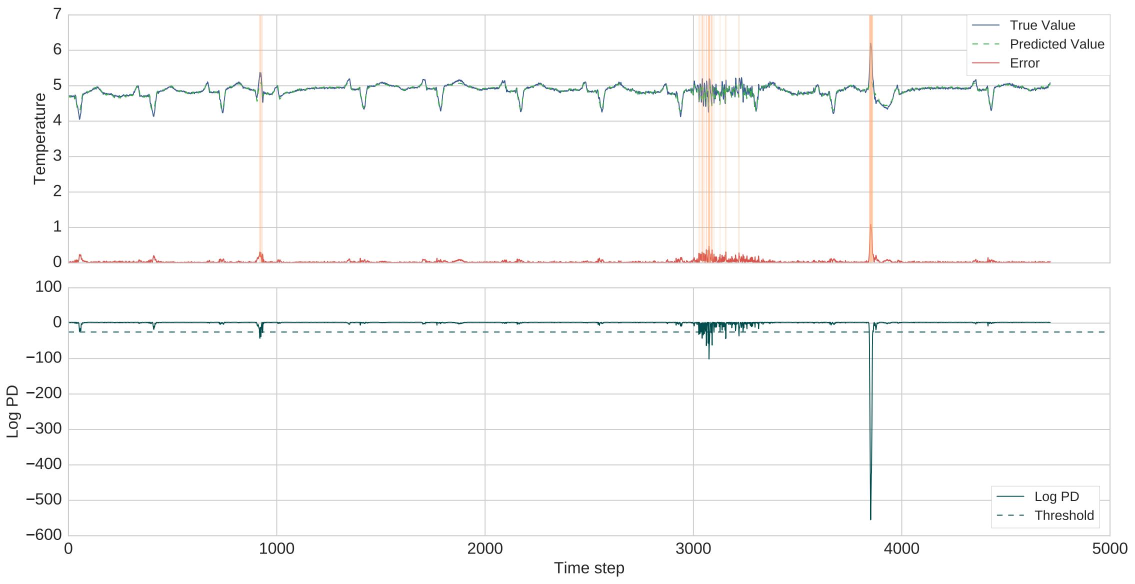


Figure 4.9: **Results using feed-forward NN on ECG data**. The results are on set T and a threshold of -25 detects all 3 anomalies. The X-axis shows time step and the Y-axis has the corresponding metric value.

For the power demand dataset the feed-forward NN gave a good prediction accuracy, but did not detect any of the anomalies. We tried different architectures and window sizes up to 100. This was similar to the problem encountered with the LSTM algorithm when we did not maintain state between batches. The NN failed to learn the long and multiple patterns present in the data.

4.4 Other Experiments

4.4.1 Effect of Lookback

The parameter lookback is the number of time steps for which the RNN is unfolded for back-propagation. There is no recommended value for which the RNN should be unfolded, and the optimal value depends on the nature of data and the temporal correlations present in it. Lookback values ranging from 8 to 200 have been used successfully for different tasks. While lookback value is an important parameter that influences learning, LSTM RNNs can still learn patterns longer than lookback length as the LSTM state can store information from any previous time step. In [44] it was shown that RNNs trained with back-propagation over a single time step were able to learn longer patterns for language modeling tasks. We found that

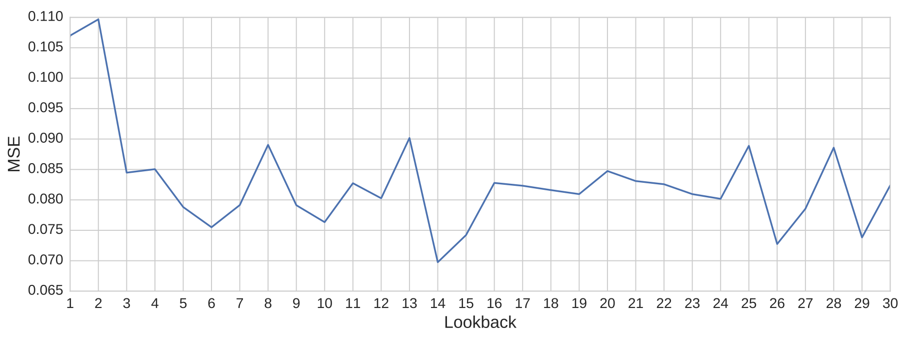


Figure 4.10: **Lookback and prediction accuracy.** The figure plots training MSE on ECG dataset for different values of lookback. Increasing lookback values after 14 do not reduce MSE. The X-axis shows lookback values and the Y-axis has the corresponding MSE values

increasing lookback values improved prediction performance up to a limit and no improvement was gained by increasing the lookback values further. Figure 4.10 plots the training MSE against lookback for ECG dataset on a single time step prediction task (lookahead equal to 1). The MSE reduces initially with a minimum of 0.07 for a lookback value of 14. Increasing lookback values after 14 do not give any further improvements. We also found the same behavior on other datasets.

4.4.2 Prediction Accuracy vs. Anomaly Detection

In some of our experiments, we experienced a trade-off between prediction performance and the results of anomaly detection. An RNN optimized for minimizing the prediction MSE was not the best for anomaly detection. The prediction errors were small throughout the set V_A (and T) making it difficult to find a threshold to separate anomalies from normal points without incurring too many false positives. One instance of this behaviour was on power demand dataset when we did not maintain state between batches. This is described in more detail in section 4.2.

An intuitive explanation for such phenomenon is as follows. When we are optimizing the model for prediction, a kind of overfitting occurs which cannot be prevented by early stopping as the MSE continues to decrease on both N and V_N. The weights of the model are tuned such that model predictions are influenced only by recent observations and model disregards any temporal relationships or information from its state. As a result, the model tries to fit even the anomalous observations to have a low prediction error. The model training is unsupervised with regards to anomaly and model has no target or feedback to guard against fitting the anomalies. The only remedy we found was to tune the model manually to detect anomalies on set V_A. However, it came at the cost of an increase in the MSE value. Manual tuning mostly involved decreasing the value of one or more of the following parameters: lookback value, the number of LSTM units in a recurrent layer, or the number of recurrent layers.

Chapter 5

Discussion

5.1 Anomaly Detection Datasets, Metrics, and Evaluation

The biggest challenge that we faced during the project was the selection of labeled datasets. Anomaly detection is considered an important problem owing to its applications and has been widely studied. Thus it was surprising that availability of labeled and benchmark datasets remains a major issue in anomaly detection research. One way to deal with this issue has been to take a dataset known to have only normal observations and introduce artificial anomalies by adding random noise to data points. But there are several problems with this approach. First, with artificial anomalies it is difficult to have any idea about the effectiveness of the algorithm on real-world data. Second, if there are sufficient labeled anomalies, it is better to use a supervised classification approach which is a much easier problem than unsupervised anomaly detection. Moreover in real-world too, anomalies are rare events and it could be argued that the development process should also reflect this fact.

We decided against using metrics like F-score, precision, recall, etc. to evaluate our methods. We believe, given the scarcity of anomalies, use of a metric to show good performance would be misleading. Since it is still necessary to demonstrate the effectiveness of our approach, we chose to compare our results to those of previous works. While we are aware that each dataset we use has too few anomalies, our purpose is to show the suitability of LSTM for temporal anomaly detection rather than claim a superior algorithm.

A related decision was not to develop a particular anomaly detection method, but instead use a thresholding approach to distinguish between anomalies and normal observations. Developing a specific anomaly detection approach, should be influenced by the intended purpose and domain of anomaly detection. For example in case of the power demand dataset there can be multiple ways to define an anomaly: a day could be an anomaly, an entire week could be treated as an anomaly, or one can even define individual observations to be anomalous, for real-time detection. ECG dataset has both point anomalies and collective anomalies. The machine temperature dataset comes with annotated point anomalies. But it is clear that whole regions around these points are unusual. The HTM algorithm used windows around these points for anomaly detection. The thresholding approach used by us can be utilized for point anomaly detection without too much modification. But for many applications, failures are preceded by unusual sequences. In such cases a collective anomaly detection method would be more useful. A possible approach, similar to [33], could be to define a sliding window over prediction errors, and use the sum of errors in a window to calculate anomaly scores.

Throughout this report we have commented on some of the main issues in anomaly detection research, including the evaluation of algorithms. The problem of evaluation is further compounded as different algorithms differ in the nature of anomalies they detect and have different trade-offs. This makes comparing different algorithms a futile task. There have been some attempts at addressing these problems. A criterion and procedure for creating anomaly detection datasets from classification datasets were presented in [38]. In [2] the authors studied different metrics and datasets that have been used for outlier evaluation. They emphasized the need for adoption of common benchmarks by the research community. As per the authors, any new outlier detection method should be supported by demonstrating its performance on a variety of datasets, with several metrics, and a wide range of parameters. However, in practice, it is common to claim a new superior algorithm by evaluating it on very specific datasets, with only a few metrics. The authors presented the results of 12 standard algorithms on over 20 datasets with different characteristics and prescribed to use these to evaluate new algorithms as well as datasets. They also touched upon the lack of reproducibility in existing research. While neither of these works claims to provide benchmark datasets for anomaly detection, we believe they represent important steps towards developing much needed common standards for anomaly detection evaluation.

5.2 Normality Assumption

One of the assumptions we made was the normal distribution of residuals or prediction errors. We conducted Shapiro-Wilk test for normality on the residuals of each dataset. Based on the outcome of the tests we rejected the hypothesis that the prediction errors were normally distributed. For all datasets, the errors have fat tails and are slightly skewed. However since we only use the normality assumption to derive a threshold to discriminate anomalies, the algorithm can still be used without any changes.

5.3 LSTMs vs Feed-forward NNs

From our analysis of LSTMs and the experiments we conducted there are several points that merit discussion. The most pressing one is perhaps the performance of feed-forward NNs vis-à-vis LSTM RNNs. On two of the three datasets, feed-forward NNs with fixed size time windows gave similar results to LSTMs for time series prediction and anomaly detection. When compared to feed-forward NNs, the theory and maths underlying the working of LSTMs are more complex and challenging to understand. Training and tuning LSTM networks is also a much more time-consuming and error-prone process. This raises a question if there is any advantage gained by using LSTMs over feed-forward NNs.

Feed-forward NNs do not have any explicit memory of the past. Their ability to model temporal relationships is limited by the size of input time window. Even for a particular time window, input data is treated as a multidimensional feature vector rather than as a sequence of observations. This method would only be suitable for simple time series which do not have long-range temporal dependencies and all the relevant information for the task is contained in only a few recent inputs. With the benefit of hindsight, it could be inferred that machine temperature, and ECG datasets constitute such simple time series. Visually these two datasets are different. While ECG dataset has a repeating pattern of 370 time steps, the machine temperature dataset does not seem to have any pattern and appears to be rather random.

The results on the power demand dataset assume greater significance in this light as they bear testimony to the superiority of LSTMs over feed-forward NNs in modeling complex time series. The power demand dataset not only has a long repeating pattern (670 time steps), but also exhibits different patterns at different time scales: a daily pattern, and a weekly pattern. We could not get any satisfactory anomaly detection results when using feed-forward NNs. Carefully maintaining LSTM cell state proved crucial in learning relevant patterns and detecting anomalies .

Some support for our reasoning is found in [45], where the authors applied LSTMs for prediction on two benchmark time series datasets. In their experiments, the LSTM model had access only to the current time step, while feed-forward NNs were given windows of consecutive time steps. It was found that for the specific datasets the LSTM model was outperformed by feed-forward NNs. The authors concluded that for many time series tasks only a few recent time steps are required and LSTM's ability to remember past information for long time periods does not provide any benefits. They suggested using LSTMs only when time window based methods do not provide desired results. It was also proposed to combine the two approaches by first training feed-forward NNs and then using LSTMs to further reduce the prediction errors.

It is difficult to know *a priori* the kind of relationships that exist in data, or if the data warrants a model as complex as LSTM. In general, it is always advisable to test simpler models before moving to complex ones. The principle of *Occam's razor* is just as applicable to machine learning as it is to other sciences. In this project, one of the aims was to study and explore LSTMs. It was only after our experiments with LSTMs we chose to use feed-forward NNs. However, there appears to be a pitfall of using LSTMs without trying simpler approaches. Much of this could be attributed to the recent popularity of LSTMs and their success on many sequential, and temporal problems. For e.g. [33] and [36] use LSTMs for time series anomaly detection, but do not compare them to simpler approaches.

5.4 LSTMs for Temporal Anomaly Detection

We conducted experiments with three different datasets. These datasets differ not only in the kind of patterns they have but also in the nature of anomalies. The machine temperature dataset comes with labeled anomalies. It does not have any repeating patterns and appears to be random. Two anomalies stand out and are extreme values. The other two seem to be quite arbitrary. In [39] it is mentioned that these latter anomalies are quite subtle and are preceded by small departures from the normal pattern which are not visually detectable. The authors highlight the utility of anomaly windows in detecting these anomalies. In our experiments too, the LSTM algorithm made detections around these two anomalies, though it did not detect the actual anomalies. However as discussed a window-based approach would give similar results to [39].

The ECG dataset also has annotated anomalies and contains both extreme values and unusual sequences. It has a repeating pattern, though the length of the pattern varies. Our method detected all the three anomalies. The power demand dataset has the longest dependencies, exhibits patterns at different time scales, and contains noise. Though there are no labels, by treating the week of the day as context, we can distinguish between normal and unusual patterns. By carefully maintaining the LSTM state the model learned the multiple patterns and all the anomalies were detected. Moreover, the anomaly scores gave a fair indication of the extent or probability of anomaly.

Based on our experiments we believe that LSTMs are effective temporal anomaly detectors. We were able to tune LSTMs for datasets with different characteristics and detect various types of anomalies. LSTMs' memory enables them to use context as well as input data while making predictions. Though as we experienced an LSTM network tuned for prediction accuracy is not necessarily a good anomaly detector. The key is to utilize LSTM memory to make it retain relevant past information.

We used LSTMs for multiple time steps prediction. For anomaly detection one time step prediction is sufficient. Multiple time step prediction was done to showcase the capability of LSTMs as time series modelers. Predicting multiple time steps can provide an early indication of unusual behavior and serve as an important first step in moving from anomaly detection to anomaly prediction and possibly predictive maintenance.

5.5 LSTMs: Evolution and Future

LSTMs were developed to enable RNNs to learn long-term dependencies. It was discovered that vanilla RNNs do not work well for tasks where the relevant events (often referred to as teacher signals) were more than a few time steps old. Vanilla RNNs are afflicted by the problem of exploding /vanishing gradients which prevents the model learning to converge to optimal parameters. LSTMs remove this problem by introducing a memory cell which ensures constant error flow and gating units to control information flow in and out of the cell. The original LSTM architecture worked well when the sequences in the input data had well defined boundaries but failed otherwise. The remedy was forget gates, which automatically refreshed LSTM memory as needed. Since then a number of variations have been proposed. The prominent one being peephole connections which enables LSTMs to learn precise intervals.

However, all this sophistication comes at the price of added complexity. It is widely acknowledged that LSTMs are difficult to train and optimize, as there are a lot of different parameters that are needed to be tuned. By using already trained models and known values of parameters one can get good results in practice, but it requires a great deal of effort to understand what makes LSTMs work and what are their limitations. Studies like [21], [46] have been conducted to understand the purpose of the various LSTM components and to analyze if the different variants of LSTM actually offer any advantage. A new simplified architecture called *gated recurrent unit (GRU)* was proposed in [47] and has become quite popular. This is another indication that the standard LSTM can be quite overwhelming and we will most likely see more simpler variants in future.

Chapter 6

Conclusion

In this thesis we developed an anomaly detection method for temporal data using LSTM networks. We started by studying LSTMs to explain their complex architecture, their ability to learn long-range dependencies, and different ways of maintaining LSTM state. Next, we developed an anomaly detection method based on a summary prediction model. We briefly touched upon the various issues prevalent in anomaly detection research. To circumvent these problems we selected three real-world datasets that have been used in previous research and contain different types of anomalies. Finally, we conducted experiments on these datasets. Based on our results we conclude that LSTMs are effective time series modelers and anomaly detectors. But we advise trying feed-forward NNs first as they are sufficient in many cases. LSTMs come with added complexity and will provide an advantage only when the data has long-range dependencies and multiple patterns.

6.1 Future Work

Some of the challenges in anomaly detection, including the selection of datasets and evaluation of algorithms, have been addressed in [2] and [38]. While none of the proposals are for time series, we believe it could still be beneficial to try them. The anomaly detection method used by us works directly for point anomalies. For detecting collective anomalies, a method which uses the sum of prediction errors in a sliding window can be developed. We would also like to work with more complex datasets, containing long-range dependencies and patterns at multiple time scales, where the advantage of LSTMs can be showcased more prominently. Lastly, it would be interesting to try GRUs as they simplify the LSTM architecture. GRUs merge the input and forget gates into a single update gate, and do not maintain an explicit memory cell. Owing to this simplified architecture GRUs are computationally more efficient than LSTMs. They have also been found to produce similar results to LSTMs [48] and are increasingly becoming a popular alternative.

Bibliography

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly Detection: A Survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. doi: 10.1145/1541880.1541882. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[2] G. O. Campos, A. Zimek, J. Sander, R. J. G. B. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle, "On the Evaluation of Unsupervised Outlier Detection: Measures, Datasets, and an Empirical Study," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 891–927, Jul 2016. doi: 10.1007/s10618-015-0444-8. [Online]. Available: http://dx.doi.org/10.1007/s10618-015-0444-8

[3] Q. Yang and X. Wu, "10 challenging problems in data mining research," *International Journal of Information Technology &; Decision Making*, vol. 05, no. 04, pp. 597–604, 2006. doi: 10.1142/S0219622006002258. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0219622006002258

[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. 26, no. 9, pp. 2250–2267, Sept 2014. doi: 10.1109/TKDE.2013.184

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. doi: 10.1038/nature14539. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[6] A. Håkansson, "Portal of Research Methods and Methodologies for Research Projects and Degree Projects," *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering FECS'13*, pp. 67–73, 2013. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-136960

[7] "Introduction to Artificial Neural Networks - Part 1." [Online]. Available: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7

[8] D. Williams and G. Hinton, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[9] S. Ruder, “An Overview of Gradient Descent Optimization Algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[11] P. Domingos, “A Few Useful Things to Know about Machine Learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[12] R. J. Frank, N. Davey, and S. P. Hunt, “Time Series Prediction and Neural Networks,” *Journal of Intelligent & Robotic Systems*, vol. 31, no. 1, pp. 91–103, 2001.

[13] A.-r. Mohamed, G. E. Dahl, and G. Hinton, “Acoustic Modeling using Deep Belief Networks,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[14] L. Rabiner and B. Juang, “An Introduction to Hidden Markov Models,” *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, Jan 1986. doi: 10.1109/MASSP.1986.1165342

[15] P. J. Werbos, “Backpropagation Through Time: What It Does and How to Do It,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[16] 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.

[17] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, “Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies,” 2001. [Online]. Available: http://www.bioinf.jku.at/publications/older/ch7.pdf

[18] R. Pascanu, T. Mikolov, and Y. Bengio, “On the Difficulty of Training Recurrent Neural Networks.” *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[19] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual Prediction with LSTM,” *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[21] K. Greff, R. K. Srivastava, J. KoutnÃk, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–11, 2017. doi: 10.1109/TNNLS.2016.2582924

[22] M. Hermans and B. Schrauwen, "Training and Analysing Deep Recurrent Neural Networks," in *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 190–198. [Online]. Available: http://papers.nips.cc/paper/5166-training-and-analysing-deep-recurrent-neural-networks.pdf

[23] A. Graves, A. r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013. doi: 10.1109/ICASSP.2013.6638947. ISSN 1520-6149 pp. 6645–6649.

[24] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014. [Online]. Available: http://193.6.4.39/~czap/letoltes/IS14/IS2014/PDF/AUTHOR/IS141304.PDF

[25] 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. doi: 10.1007/s10115-006-0026-6. [Online]. Available: http://dx.doi.org/10.1007/s10115-006-0026-6

[26] D. J. Hill and B. S. Minsker, "Anomaly Detection in Streaming Environmental Sensor Data: A Data-Driven Modeling Approach," *Environmental Modelling & Software*, vol. 25, no. 9, pp. 1014–1022, 2010.

[27] J. Ma and S. Perkins, "Online Novelty Detection on Temporal Sequences," in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: ACM, 2003. doi: 10.1145/956750.956828. ISBN 1-58113-737-0 pp. 613–618. [Online]. Available: http://doi.acm.org/10.1145/956750.956828

[28] R. S. Tsay, D. Peña, and A. E. Pankratz, "Outliers in Multivariate Time Series," *Biometrika*, vol. 87, no. 4, pp. 789–804, 2000.

[29] Y. Yao, A. Sharma, L. Golubchik, and R. Govindan, "Online Anomaly Detection for Sensor Systems: A Simple and Efficient Approach," *Performance Evaluation*, vol. 67, no. 11, pp. 1059–1075, 2010.

[30] H. N. Akouemo and R. J. Povinelli, "Probabilistic Anomaly Detection in Natural Gas Time Series Data," *International Journal of Forecasting*, vol. 32, no. 3, pp. 948–956, 2016.

[31] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long Short Term Memory Networks for Anomaly Detection in Time Series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89. [Online]. Available: https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf

[32] S. Chauhan and L. Vig, “Anomaly Detection in ECG Time Signals via Deep Long Short-Term Memory Networks,” in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oct 2015. doi: 10.1109/DSAA.2015.7344872 pp. 1–7.

[33] L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac, *Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks.* Springer International Publishing, 2016, pp. 141–152. ISBN 978-3-319-48057-2. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-48057-2_9

[34] M. Cheng, Q. Xu, J. Lv, W. Liu, Q. Li, and J. Wang, “MS-LSTM: A Multi-Scale LSTM Model for BGP Anomaly Detection,” in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on.* IEEE, 2016, pp. 1–6.

[35] E. Marchi, F. Vesperini, F. Weninger, F. Eyben, S. Squartini, and B. Schuller, “Non-Linear Prediction with LSTM Recurrent Neural Networks for Acoustic Novelty Detection,” in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015. doi: 10.1109/IJCNN.2015.7280757. ISSN 2161-4393 pp. 1–7.

[36] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, “LSTM-Based Encoder-Decoder for Multi-sensor Anomaly Detection,” *CoRR*, vol. abs/1607.00148, 2016. [Online]. Available: http://arxiv.org/abs/1607.00148

[37] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” in *Advances in Neural Information Processing Systems 25.* Curran Associates, Inc., 2012, pp. 2951–2959. [Online]. Available: http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[38] A. F. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, “Systematic Construction of Anomaly Detection Benchmarks from Real Data,” in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ser. ODD ’13. New York, NY, USA: ACM, 2013. doi: 10.1145/2500853.2500858. ISBN 978-1-4503-2335-2 pp. 16–21. [Online]. Available: http://doi.acm.org/10.1145/2500853.2500858

[39] A. Lavin and S. Ahmad, “Evaluating Real-Time Anomaly Detection Algorithms - The Numenta Anomaly Benchmark,” in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015. doi: 10.1109/ICMLA.2015.141 pp. 38–44.

[40] E. Keogh, J. Lin, and A. Fu, “HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence,” in *Fifth IEEE International Conference on Data*

Mining (ICDM'05), Nov 2005. doi: 10.1109/ICDM.2005.79. ISSN 1550-4786 pp. 8–.

[41] M. Jones, D. Nikovski, M. Imamura, and T. Hirata, "Anomaly Detection in Real-Valued Multidimensional Time Series," in *International Conference on Bigdata/Socialcom/Cybersecurity. Stanford University, ASE*, 2014. [Online]. Available: http://www.merl.com/publications/docs/TR2014-042.pdf

[42] M. C. Chuah and F. Fu, *ECG Anomaly Detection via Time Series Analysis.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 123–135. ISBN 978-3-540-74767-3. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74767-3_14

[43] J. Hawkins and D. George, "Hierarchical Temporal Memory: Concepts, Theory and Terminology," Technical report, Numenta, Tech. Rep., 2006. [Online]. Available: http://www-edlab.cs.umass.edu/cs691jj/hawkins-and-george-2006.pdf

[44] T. Mikolov, "Statistical Language Models Based on Neural Networks," *PhD thesis, Brno University of Technology.*, 2012. [Online]. Available: http://www.fit.vutbr.cz/~imikolov/rnnlm/thesis.pdf

[45] F. A. Gers, D. Eck, and J. Schmidhuber, *Applying LSTM to Time Series Predictable through Time-Window Approaches.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 669–676. ISBN 978-3-540-44668-2. [Online]. Available: http://dx.doi.org/10.1007/3-540-44668-0_93

[46] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An Empirical Exploration of Recurrent Network Architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 2342–2350.

[47] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *arXiv preprint arXiv:1406.1078*, 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[48] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *arXiv preprint arXiv:1412.3555*, 2014. [Online]. Available: https://arxiv.org/abs/1412.3555