ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING AND ARCHITECTURE

DEPARTMENT

Electrical, Electronic, and Information Engineering "Guglielmo Marconi" - DEI

SECOND CYCLE DEGREE IN
Electronic Engineering

MASTER THESIS
in
Signal Acquisition And Processing M

Anomaly detection by prediction for health monitoring of satellites using LSTM neural networks

CANDIDATE
WENLIANG XIANG

SUPERVISOR:
Prof. Dr. Riccardo Rovatti

Co-SUPERVISORS:
Prof. Dr. Mauro Mangia
Dr. Alex Marchioni
Dr. Filippo Martinini
Dr. Andriy Enttsel

Academic Year 2020/21

Session III

Abstract

Anomaly detection in satellite has not been well-documented due to the unavailability of satellite data, while it becomes more and more important with the increasing popularity of satellite applications. Our work focus on the anomaly detection by prediction on the dataset from the satellite, where we try and compare performance among recurrent neural network (RNN), Long Short-Term Memory (LSTM) and conventional neural network (NN). We conclude that LSTM with input length $p = 16$, dimensionality $n = 32$, output length $q = 2$, 128 neurons and without maximum overlap is the best in terms of balanced accuracy. And LSTM with $p = 128$, $n = 32$, $q = 16$, 128 and without maximum overlap outperforms most with respect to AUC metric. We also invent award function as a new performance metric trying to capture not only the correctness of decisions that NN made but also the amount of confidence in making its decisions, and we propose two candidates of award function. Regrettably, they partially meet our expectation as they possess a fatal defect which has been proved both from practical and theoretical viewpoints.

Contents

Chapter 1

Introduction

In this thesis project, we are going to explore the anomaly detection by prediction on the dataset from satellite. We define samples as anomalies if they do not conform the definition of normal behaviors. As a matter of fact, it is really difficult to provide an accurate definition of anomaly as it is usually application-dependent in a sense that the same event can be identified as anomaly in some scenarios but may also be labeled as a normal event in other cases, see in the figure 1.1. Here we cite one of the most commonly used definitions of anomalies in literature [1]:

> "An outlier[1] is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism."

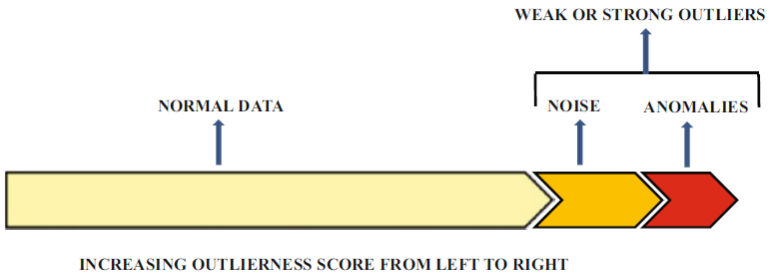


Figure 1.1: There is a gray region indicating the freedom of definition, and we regard this region as noise, the existence of this region reveals the difficulty of defining anomalies or normal behaviors that encompass all the possibilities, hence, the definition of anomalies are actually application-dependent. Figure from [1].

Anomalies are categorized into three types [5]:

- *Point Anomalies*. anomalous event is just an element (or sample) in dataset, for example, a peak that is found in time series during the peak detection is a point anomaly.
- *Contextual Anomalies*. the definition of anomaly depends on the context which contains two aspects:

[1] We use term *anomaly* and the term *outlier* interchangeably in our report.

1. Contextual attribute. it describes the position of the given data instance in the context, for instance, the time stamp is a contextual attribute for element in time series data.
2. Behavioral attribute. it characterize the features independent of context for the given data instance, e.g., the value of the data instance in time series.

and it is common to see that, a data instance can be determined as an anomaly in a context but nominal data in other context.

- *Collective Anomalies*. an anomalous event is a collection of some related samples, rather than each single sample. for example, an anomaly in ECG data is shown in figure 1.2

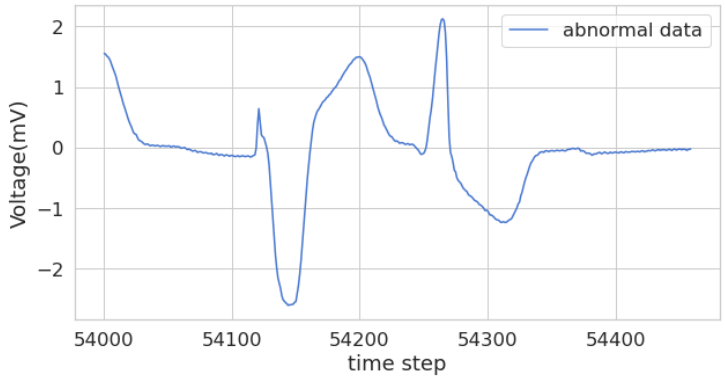


Figure 1.2: An anomaly in ECG data, it is actually a collection of ECG samples.

As is declared in the definition, anomalous events are created by some unusual mechanism, they reflects ill status of systems and gives rise to the degeneration of system performance as a consequence, or sometimes even results in a system-level crash if there is any fatal failure in critical components of system. Since systems nowadays becomes more and more complex and therefore fragile as well in some way, the occurrence of anomalies are somehow inevitable, from this viewpoint, detecting anomalies instead of preventing them from happening could be a better choice if we think of how much efforts we must make in order to completely get rid of them.

Anomaly detection has been one of the state-of-arts accompanied with many other challenges in addition to the problem of accurate definition we mentioned above, such as: the lack of sufficient anomalies because the amount of anomalies is rare in relative to nominal data, availability of labeled datasets for further study which usually is the main issue, from this perspective, unsupervised learning approaches are preferable in contrast to other methods such as supervised learning. Anomaly detection has been applied to many fields:

- Fraud Detection— In the field of economics, unusual consumption behaviors or transaction activities typically happen when credit cards are stolen from users. Such fraudulent financial activities can be easily recognized via anomaly detection given that behavioral pattern in that case has changed, and be blocked in order to protect users as well as financial institutions from further losses.

- Intrusion Detection— It aims to identify malicious activities happened in computer systems or networks, for example, intrusion against computer security system. It is one of the main issues that computer society concerns, and turns to be more and more relevant

to human's life as we are pacing towards a digitalized world. On the other hand, many challenges are emerging in the meantime, e.g., volume of data involved becomes larger and larger if we think of the applications of IoT.

- Structural Health Monitoring (SHM)— As a matter of fact, anomaly detection is consistently one of the concerns in the field of SHM. Basically, it detects anomalies throughout data collected from sensors embedded in the objects like buildings or bridges, evaluates structural condition accordingly and sends an alert when needed to eliminate any potential threats.

- Health Care— Undoubtedly, anomaly detection is also perfectly suitable to the applications related to health care if we consider them special cases of SHM in an extended sense where we would like to monitor human body's condition instead. For example, a dedicated hardware can be designed to perform 24-hour anomaly detection on real-time ECG signals from patients and once there is an anomaly arising, it can immediately warn doctors of any cardiac illness so that they would have more time to take appropriate actions and to save their life as well as improve survival rate.

A variety of methods have been proposed and well-developed to cope with anomaly detection regarding different situations. Many statistical approaches are invented given that anomaly detection was first investigated in the statistics community, meanwhile, with the rising interests in artificial intelligence, various neural network based techniques are also proposed for anomaly detection. for this reason, we are going to explore the possibility of utilizing a particular neural network, LSTM, to perform anomaly detection based on data from satellites and compare its performance with the conventional neural network. LSTM possesses some good properties attracting us to adopt it in anomaly detection: first, unlike conventional NN, it is mainly designed to learn the time dependency, this feature helps us as we are exploring time series data from satellites, also, the number of parameters LSTM needs is less than conventional NN thanks to the technique of parameter sharing, last but not least, the training is more stable during the training process since LSTM does not suffers from exploding gradient or vanishing gradient problems in contrast with recurrent neural networks (RNN).

Outline Our project focus on the data from satellite which has not been well-documented, and we will try different neural networks with various parameters to perform anomaly detection. This report will be organized as follows: In chapter 1, we briefly introduce the methods applied for anomaly detection starting from statistical fields where anomaly detection stems from and we stop at one of state-of-arts, neural network based methods. Then we formally introduce neural network based methods at next chapter with a toy case on ECG data which helps us obtain intuitive ideas for methods we are going to apply. We detailed describes our work on satellite data followed by the results in chapter 3 and 4. Finally, we end up with our conclusion in chapter 5.

Chapter 2

Methods for anomaly detection

Early studies on outlier detection mainly focused on the statistical characteristics of data set, while sometimes, most of data at hand does not subject to purely statistics laws, more universial methods are needed consequently. To the best of our knowledge, many outlier detection methods have been proposed for various applications and we will demonstrate some typical examples starting from statistical methods and end up with machine learning based methods.

2.1 Depth-based Method

As is addressed in the definition, an outlier is the observation that deviates most from normal data, intuitively speaking, it is the data point[1] that lies far away from clusters. A depth-based method is proposed to detect outliers by constructing a collection of contours. Suppose we want to find out outliers given the dataset, X say, according to its definition, outliers are supposed to be found far from normal clusters, see an example in the figure 2.1,

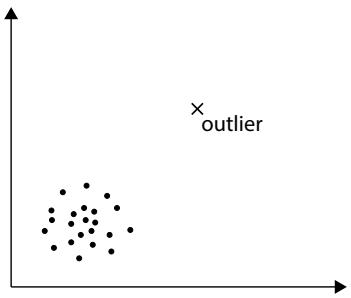


Figure 2.1: An outlier lies far away from the cluster.

This feature inspire us that if we regard all data points of X as vertices of mutually nested contours, the outlier is most likely to be discovered in the outmost contour. To capture this idea, we first confine our attention on 2-dimensional space and imagine we have an elastic band encompassing all data points of X initially, then it starts shrinking and finally becomes the outermost contour when it touches all the outermost data points, and if we removes those points that support the outermost contour, the band will shrink once again until it forms a new

[1] we use terms *point*, *sample*, and *element* interchangeably

outermost contour based the rest of data points as is shown in figure 2.2 and 2.3. This process will proceed consistently until all points are removed from the space.

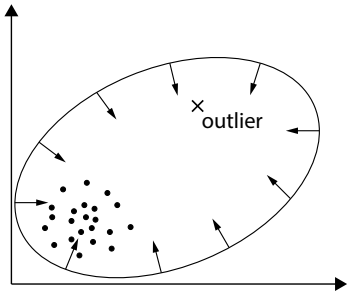


Figure 2.2: An elastic band encompasses all data points at the beginning and it tends to shrink without supports.

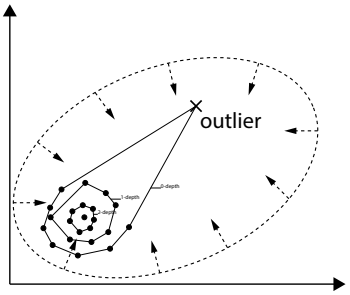


Figure 2.3: An elastic band forms the outermost contour once it touches outermost points.

This algorithm is represented below, where we denote $\mathcal{C}$ as the collection of contours and V is the vertex set of the outermost contour.

Algorithm 1 adapted from [1]

$S = X, \mathcal{C} = \{\ \emptyset\ \}, V = \emptyset$
while $S \neq \emptyset$ **do**
  Find the outermost contour
  Assign all data points that support the outermost contour to V
  $\mathcal{C} = \mathcal{C} \bigcup \{\ V\ \}, S = S \setminus V, V = \emptyset$
end while

Ida Ruts [34] proposed *ISODEPTH* algorithm to construct a collection of nested contours and assign depth number to each contour under the assumption that outliers only locate in lowest depth contour, namely, the outermost one. Compared to naive algorithm whose total time complexity is $O(N^5 \log N)$, ISODEPTH brings the overall complexity to $O(N^2 \log N)$ and this has been confirmed on synthetic datasets generated according to standard Gaussian distribution. Theodore Johnson extended Ida's work and created a fast algorithm, *FDC* [20], for the same purpose by restricting the computation on a small set of data points instead of the whole data points as is did in ISODEPTH, and it leads overall complexity to $O(N \log N + h \log^2 N + kh^3)$. It is worth mentioning that, for a large data set, k is typically not large ($\leqslant 100$ say) and h is at least 2 or 3 orders of magnitude smaller than N, which makes the final complexity of FDC approximately equal to $O(N \log N)$.

2.2 Deviation-based method

Deviation-based method focus on the variation of dissimilarity due to the occurrence of outliers, and an algorithm [2] is invented based on this motivation. This algorithm simulates the

mechanism similar to human beings: after seeing a sequence of homogeneous or similar data, a new point will be considered an outlier if it deviates a lot from the previous observations, said differently, it contributes the most dissimilarity to the whole data set. In order to quantitatively measure how much a set of points contributes to dissimilarity of the whole data set, *smoothing factor* (SF) is then introduced in the paper and as we can see, this notion is well-defined as we expect: for the normal data sets, their SF is small in contrast to SF of outlier sets, which are relatively high. Consider an example provided in the paper, where we have the whole set $X = \{\ 1, 4, 4, 4\ \}$. Apparently, $O = \{\ 1\ \}$ is an outlier set embedded in X and it can be proved more evidently if we compute SF of the power set of X constituting the table 2.1.

X	I	SF(I)
$\{\ 1, 4, 4, 4\ \}$	$\{\ \}$	0.00
$\{\ 1, 4, 4, 4\ \}$	$\{\ 4\ \}$	-0.94
$\{\ 1, 4, 4, 4\ \}$	$\{\ 4, 4\ \}$	-1.13
$\{\ 1, 4, 4, 4\ \}$	$\{\ 4, 4, 4\ \}$	1.69
$\{\ 1, 4, 4, 4\ \}$	$\{\ 1\ \}$	5.06
$\{\ 1, 4, 4, 4\ \}$	$\{\ 1, 4\ \}$	3.38
$\{\ 1, 4, 4, 4\ \}$	$\{\ 1, 4, 4\ \}$	1.69

Table 2.1: Table adapted from [2]

Likewise, we can also apply same concept to sequential problems by extending the definition of smoothing factor to the case of sequential data. In that case, SF indicates dissimilarity contribution of current data to preceding data set, in other words, how much current data point contributes to dissimilarity of preceding data set. For instance, suppose we have a sequence data reads as follows,

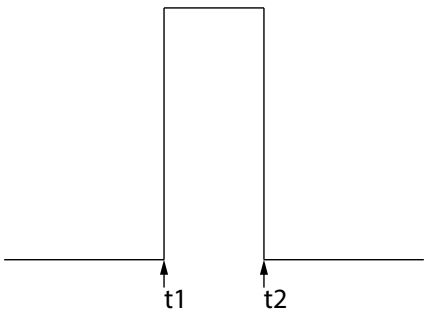


Figure 2.4: Large deviations happen at t_1 and t_2, which are identified as anomalies.

By definition, the SF at t_1 and t_2 shall be large compared to the SF at other time instance since data at t_1 and t_2 contributes most to dissimilarity of the preceding sequence. On the other hand, it is evident to see that now SF is sensitive to abrupt variation (mathematically speaking, discontinuous points) and therefore vulnerable to noise. Concerning the complexity issue, this algorithm possesses a linear computation time that makes it perfectly suitable to large data set. However, the effectiveness of algorithm depends on the dissimilarity functions we used to define smoothing factors, and if we have priori knowledge about outliers, we can design a very effective dissimilarity function for the application. Furthering experiments reveal the facts that

dissimilarity function is application-dependent and it is difficult to have a universial one that works well for all applications.

2.3 Distance-based method

Conventional methods for outlier detection lie in the field of statistics and focus their attention on statistical characteristics of data. Those methods require many discordancy tests to see if data fit any standard distribution when we lack some priori-knowledge or when we do not know whether data follows particular distributions. In the light of the unified notion of outliers that coincide with aforementioned outlier definition, regardless of data distribution, Edwin M. Knorr offers a distance-based approach and algorithms without need for many (not *all*) statistical discordancy tests [22] [23].

Specifically, a data point o is said to be an outlier if

$$\frac{\{\ x \mid d(x,o) > r\ \ ,\ \ x \in X\ \}}{\overline{\overline{X}}} \geqslant p$$

where

$d(x,o)$— distance from x to o, in this case, Euclidean distance is applied.

$\overline{\overline{X}}$— cardinality of set X, equivalently, number of data in x.

r— radius.

p— fraction.

Obviously, it is equivalently to say

$$\overline{\overline{\{\ x \mid d(x,o) \leqslant r\ \ ,\ \ x \in X\ \}}} \leqslant \overline{\overline{X}}(1-p) \triangleq N(1-p)$$

The simple algorithm based on this definition is nothing more than a brute-force algorithm: for every point of X, it counts the number of points in its neighborhood (technically, the ball with radius r) and to see if the amount exceeds $\leqslant N(1-p)$. If yes, the point is an outlier, otherwise, it is not. The pseudo-code (C++ style) is represented below

The complexity of simple algorithm is $O(kN^2)$ where k stands for dimensionality. Additionally, a block-oriented design can also be implemented to avoid the cost of building index for entire dataset: it divides buffer into two halves, then performs distance computation as well as outlier checking on one half and only those referred to as outlier candidates that do not pass the outlier checking need furthering checking on the other half. This block-oriented design ease the overload of I/O's but have same complexity to the simple algorithm.

However, if we follow this divide-and-conquer idea and simply divide the whole space into small cells instead of two blocks in block-oriented design as well as check outliers in the unit of cell, finally we will reach an extreme case of block-oriented design, that is, the cell-based algorithm whose complexity now is linearly proportional to N but exponentially with respect to dimensionality k. Take 2-D space for example, we first divide space into cells in terms of side length $l = \frac{r}{2\sqrt{2}}$, whereas r is the radius of ball, and we define the first layer neighbors of cell

for $\forall x \in X$ **do**
  counter = 0
  for $\forall y \in X$ **do**
    // count how many data points are inside the ball
    if $d(y, x) \leqslant r$ **then**
      counter++;
      if counter $> N(1-p)$ **then**
        // x is NOT an outlier
        break;
      end if
    end if
  end for
  if counter $\leqslant N(1-p)$ **then**
    // x is an outlier
  end if
end for

$C_{x,y}$ as

$$N_1(C_{x,y}) \triangleq \{\ C_{u,v} \mid u = x \pm 1\ \ ,\ \ v = y \pm 1\ \}$$

also the second layer neighbors of $C_{x,y}$ is

$$N_2(C_{x,y}) \triangleq \{\ C_{u,v} \mid u = x \pm i\ \ ,\ \ v = y \pm j\ \ ,\ \ i, j = 2, 3\ \}$$

A graphic representation is shown in figure 2.5

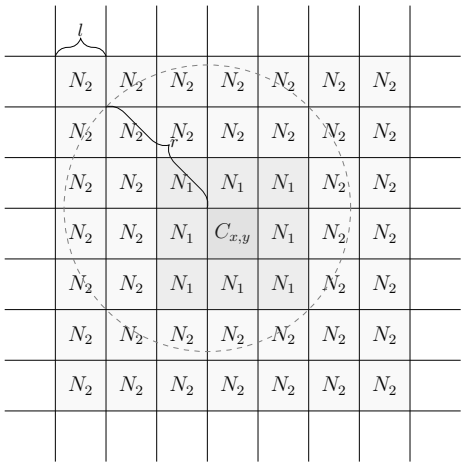


Figure 2.5: A graphic representation of the cell-based algorithm.

with propositions:

1. if there are $> N(1-p)$ data points in $C_{x,y}$ or $C_{x,y} \bigcup N_1(C_{x,y})$, then none of data points in $C_{x,y}$ is an outlier.
2. if there are $\leqslant N(1-p)$ data points in $C_{x,y} \bigcup N_1(C_{x,y}) \bigcup N_2(C_{x,y})$, then every data point in $C_{x,y}$ is an outlier.

Proof. 1. Since we know that $\forall a \in C_{x,y}$ or $a \in C_{x,y} \bigcup N_1(C_{x,y})$,

$$a \in \{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \}$$

holds for $\forall o \in C_{x,y}$. Thus, for $\forall o \in C_{x,y}$, we have

$$C_{x,y} \subset C_{x,y} \bigcup N_1(C_{x,y}) \subset \{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \}$$

Namely,

$$\overline{\overline{\{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \}}} \geqslant \overline{\overline{C_{x,y} \bigcup N_1(C_{x,y})}} \geqslant \overline{\overline{C_{x,y}}} > N(1-p)$$

holds for $\forall o \in C_{x,y}$, and by definition, o is not an outlier.

2. Similarly, for $\forall o \in C_{x,y}$, and $\forall a \in \{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \}$,

$$a \in C_{x,y} \bigcup N_1(C_{x,y}) \bigcup N_2(C_{x,y})$$

holds for $\forall o \in C_{x,y}$. Thus, for $\forall o \in C_{x,y}$, we have

$$\{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \} \subset C_{x,y} \bigcup N_1(C_{x,y}) \bigcup N_2(C_{x,y})$$

Namely,

$$\overline{\overline{\{\ t \mid d(t,o) \leqslant r\ \ ,\ \ t \in X\ \}}} \leqslant \overline{\overline{C_{x,y} \bigcup N_1(C_{x,y}) \bigcup N_2(C_{x,y})}} \leqslant N(1-p)$$

holds for $\forall o \in C_{x,y}$, and by definition, o is an outlier.

□

Then, once we have counted number of points in each cell, outliers can be detected according to two propositions. Clearly, those two propositions are merely sufficient conditions for declaring outliers, but if space partitioning is dense enough so that the cell volume is sufficiently small, then two propositions would become almost sufficient and necessary conditions for outlier detection. We can also extend cell-based algorithm to the case of high dimensional space, the only difference is side length of cell which, in that case, turns out to be $\frac{r}{2\sqrt{k}}$. It has $O(c^k + N)$ complexity, where c is a constant depending on dimensionality k.

In conclusion, the simple algorithm and its block-oriented design have complexity $O(kN^2)$ whilst cell-based algorithm has linear complexity exponential with respect to dimensionality k. As a result, the cell-based is preferable when dealing with low dimensional data (e.g., $k \leqslant 4$) and simple algorithm including its block-oriented design are superior when k is large.

2.4 Density-based method

Distance-based method utilizes a pair of global parameter, radius and fraction, to describe the whole dataset which may contain various cluster structure in different region, consequently, it lacks capability to capture detailed structure information of dataset. For instance, assume we have a dataset composed of two clusters C_1 and C_2 and three outliers, o_1, o_2, and o_3 respectively, see in figure 2.6. In this dataset, C_1 is dense and C_2 is sparse on the contrary, o_1 and o_2 lie far away from both cluster C_1 and C_2, whilst o_3 is relatively close to C_1. The problem is, the distance between o_3 and C_1 approximately equal to the average distance of any two points in cluster C_2, in that case, distance-based approach will fail to detect o_3. For this reason, Markus M. Breunig assign each point a *local outlier factor* (LOF) as the score of being an outlier by introducing the concept of density to dataset, LOF is local property in a sense that for a given point, it only depends on surrounding neighbors. Hence, it is capable of characterizing local structure of dataset. Moreover, he also presents a density-based algorithm based on this notion [3] and [4].

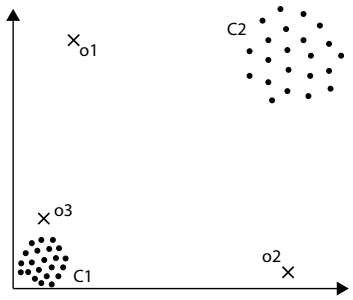


Figure 2.6: Three outliers and two clusters with different density, C_1 is dense in relative to C_2, adapted from [3]

Specifically, the higher the LOF, the more chance for a point to be an outlier. As for those data points in clusters, their LOF approximately equals to 1. In addition, the upper bound and lower bound on LOF are also given with proofs in [4], as well as the tightness of bounds. The performance of density-based algorithm finally reaches $O(N \log N)$ complexity at best.

2.5 Machine Learning based method

Machine learning (ML) based methods are powerful complement of conventional methods especially in the situation where we do not have sufficient priori knowledge, and typically, a ML-based model for outlier detection is trained in an unsupervised or semi-supervised manner as it is difficult and expensive to label massive data that involve expert knowledge.

2.5.1 Autoencoder-based method

With the extensive studies and wide applications of IoTs, the dimensionality of data has been growing consistently and eventually gives rise to the *curse of dimensionality* inevitably. Basically, the curse of dimensionality is a collection of phenomenon that happen when data dimen-

sion increases, for examples, the probability that two data are "similar" in terms of Euclidean metric tends to zero and data volume appears to "explode" exponentially [37].

Autoencoder (AE) based method stems from the motivation of dimensionality reduction. It was first introduced as a nonlinear principal component analysis (NL-PCA) in [24] in order to reduce dimensionality by extracting main features from data. A typical AE comprises two neural networks stacked sequentially [24] and perform identity mapping theoretically, as is represented in figure 2.7. The first neural network called encoder is a feature extractor, which aims to extract main features from input data and then represent them in latent space by means of nonlinear mapping f. Then, the second NN referred to as decoder exploits those main features to reconstruct input data while keeping reconstruction error as minimum as possible. Those main features in latent space can be regarded as a compressed representation of input data, and dimensionality reduction is therefore achieved.

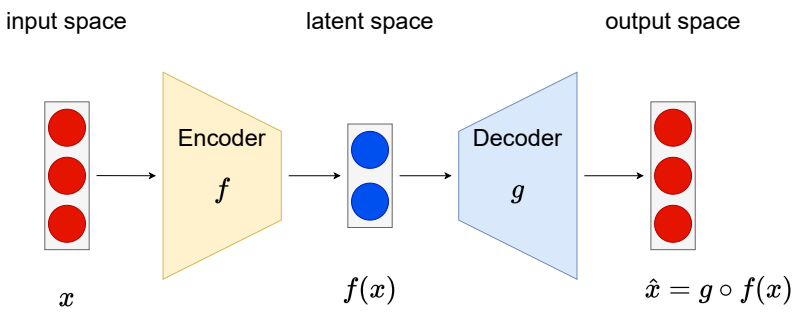


Figure 2.7: A typical architecture of an autoencoder.

Or formally speaking, an AE is a pair of mappings (f, g) such that

$$(f, g) = \arg\min_{f,g \in \mathcal{F}} L\left[x, g \circ f(x)\right] \quad , \quad x \in X$$

where $\mathcal{F}$ is the space of candidate functions, X is input space, and L stands for the pre-defined loss function, e.g., L_2-norm. On the other hand, it could also be the case that all data distribute in a manifold [2] embedded in a higher dimensional input space X, and it is of our interest to find out a low dimensional representation of this manifold. Some algorithms have been proposed for this purpose [36] [33] [16]. To better illustrate this idea, we represent an toy case to show how an AE capture the reduced-dimensional representation of input data in a manifold with very small reconstruction error.

Procedure:

1. Firstly, we know that a half sphere is 2-dimensional manifold embedded in 3-D space and we generate some synthetic samples that are uniformly distributed in a unitary half sphere 2.8.

$$X \triangleq \{\ [x_1, x_2, x3]^t \mid x_1^2 + x_2^2 + x_3^2 = 1 \quad , \quad x_3 \geqslant 0\ \}$$

[2] A (Hausdorff) space is said to be a manifold if it is locally homeomorphic to a Euclidean space, in other words, any point of manifold possesses a neighborhood that is homeomorphic to a open set of Euclidean space

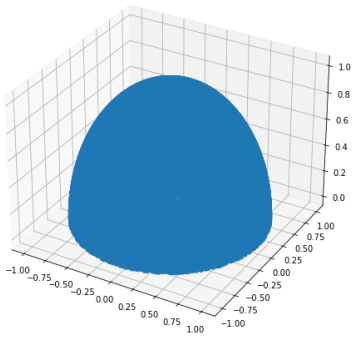


Figure 2.8: Synthetic samples are uniformly distributed in a half sphere

2. A natural low dimensional representation of this dataset is just take the first and second coordinates, because

$$f : X \to \mathbb{R}^2 \text{ where } \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}) = \begin{cases} y_1 = x_1 \\ y_2 = x_2 \end{cases}$$

and

$$g : \mathbb{R}^2 \to \mathbb{R}^3 \text{ where } \begin{bmatrix} \hat{x_1} \\ \hat{x_2} \\ \hat{x_3} \end{bmatrix} = g(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) = \begin{cases} \hat{x_1} = y_1 \\ \hat{x_2} = y_2 \\ \hat{x_3} = \sqrt{1 - (y_1^2 + y_2^2)} \end{cases}$$

in that case, the reconstruction error is zero and y_1, y_2 are sufficiently capable of reconstruct x_1, x_2 and x_3. So we model an AE with one layer NN both for encoder and decoder parts, the latent space is 2-dimensional space as we only need two coordinates for reconstruction. The number of neurons subjects to the constraint [24]

$$M_1 + M_2 \ll \frac{m(N-f)}{m+f+1}$$

whereas

M_1— number of neurons of the encoder, e.g., 10

M_2— number of neurons of the decoder, e.g., 10

f— dimensionality of latent space, i.e., 2

m— dimensionality of X, i.e., 3

N— amount of samples in X, e.g., 4000

And the results are shown in the figures 2.9 2.10 2.11 2.12 with resulting reconstruction error at $5e-4$ in the end.

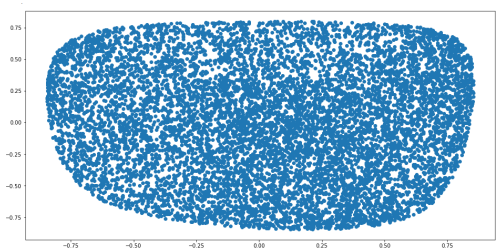


Figure 2.9: Data representation in latent space.

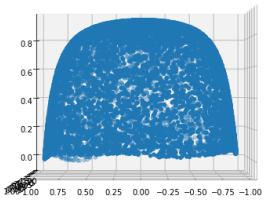


Figure 2.10: Left view of reconstructed data via AE.

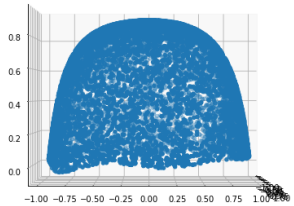


Figure 2.11: Front view of reconstructed data via AE.

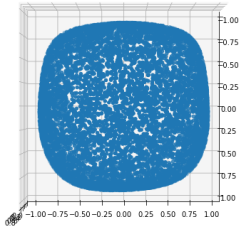


Figure 2.12: Top view of reconstructd data via AE.

3. As a comparison, we also tried PCA with reconstruction error 0.0276, see in the figures 2.13 2.14 2.15.

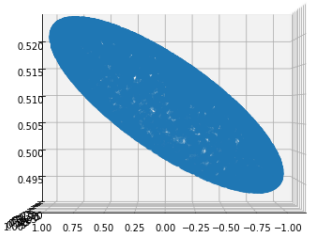


Figure 2.13: Left view of reconstructed data via PCA.

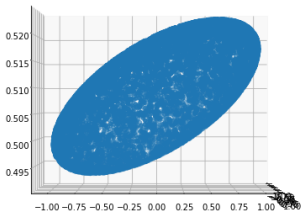


Figure 2.14: Front view of reconstructed data via PCA.

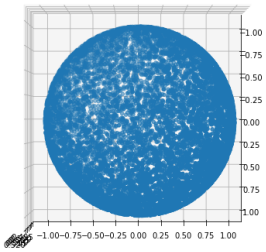


Figure 2.15: Top view of reconstructed data via PCA.

The experiment results reveal a fact that: as a nonlinear PCA, AE outperforms PCA in extracting main features from data.

2.5.2 Prediction-based method

Nowadays, massive unlabeled time series data is generated consistently and brings new issues ML-based methods:

- conventional NN assumes data is time-independent regardless of their correlation and it lacks capability of processing long-term sequence.
- unlabeled data makes it impossible to train NN in a supervised manner.
- In contrast to numerous and easy-acquired nominal data, the amount of anomalies is so rare on the contrary, which makes NN difficult to learn any knowledge from anomalies.

Recurrent Neural Network (RNN) is thus proposed to overcome those problems. Basically, it is trained to predict next q values of input time series, and thanks to its memory capacity, it can take not only the current input but also historical information in memory into account while performing prediction. Therefore, if we train RNN with only nominal time series to predict next q values, it will learn the normal behavior and understand underlying patterns finally. In that case, once a well-trained RNN encounters anomalies in the time series, the prediction error will be significantly large so that we can easily identify them. On the other hand, because of vanishing gradient problem that RNN might suffer from, Hochreiter proposed an improved RNN called Long Short-Term Memory (LSTM) in [17]. We are going to talk about RNN and LSTM more in details in the following chapters.

Chapter 3

Anomaly detection based on Neural Network

3.1 Neural Network

Neural Network (NN) is one of the most popular scientific topics since it was first introduced in 1943 [28]. It has been studied, well-developed for decades, and is still arousing more and more attention from academic, engineering as well as many other communities, meanwhile, consistently driving people to devote their enthusiasm and professional knowledge to this field.

3.1.1 Conventional Neural Network

The main idea of NN has been well explained in many literature, it was inspired by analogy with the mechanism that our brain neurons process bioelectric signal. As is stated in neurophysiology, neurons activities in human brain behaves in a way of "all-or-none" just like binary states. This feature inspire us that probably we can describe neural events by means of proposition logic and represent biological network, abbreviated as *network*, by logical expression. Indeed, [28] has proved that for any logic expression satisfying certain conditions, there exists a network behaving in the same fashion as is described by the logical expression, and vice versa. Moreover, It was also found that every logic expression can be characterized by many equivalent networks in the sense that they yield the same result, mathematically speaking, a logic expression corresponds to an equivalent class. For the sake of clarity, we will explain NNs following the steps:

1. we first start with elementary building blocks of NNs, that is, perceptrons.
2. we address the universal approximation theorem to answer the question "why NNs behave so well".
3. discuss the architecture of Deep Neural Network and list some activation functions.
4. explain the main idea of gradient descent we well as back propagation algorithm.

Perceptron

As the building blocks of NNs, perceptron was first created by Rosenblatt in 1958 [32], it is also a linear discriminant model demonstrated as follows,

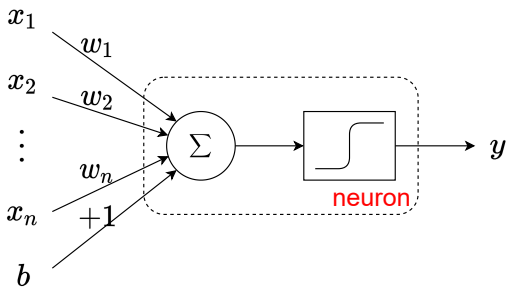


Figure 3.1: A perceptron model.

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \quad , \quad w_i, x_i, b \in \mathbb{R}, y \in \{-1, +1\}$$

or in a more compact form

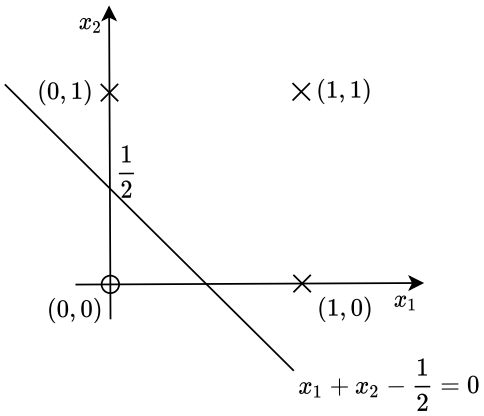
$$y = f(\vec{w}^t \vec{x} + b) \quad , \quad \vec{w}, \vec{x} \in \mathbb{R}^n, b \in \mathbb{R}, y \in \{-1, +1\}$$

where f is activation function defined as,

$$f(x) = \begin{cases} -1 & , x < 0 \\ +1 & , x \geqslant 0 \end{cases}$$

and w_i, x_i are weights and inputs respectively, likewise, $\vec{w}, \vec{x}$ are weight (column) vector and input (column) vector respectively. To show how a perceptron works, let us consider a simple case where we want to design a perceptron performing logic OR operation. Given two inputs x_1, x_2, the truth table of logic OR is

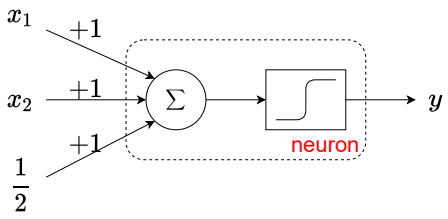
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



with discriminant expression:

$$w_1x_1 + w_2x_2 + b \underset{0}{\overset{1}{\gtrless}} 0 \iff w_1x_1 + w_2x_2 \underset{0}{\overset{1}{\gtrless}} -b$$

Obviously, $x_1 + x_2 - \frac{1}{2} = 0$ is one of the solution. Therefore, the perceptron that performs logic OR is designed as



$$f(x) = \begin{cases} -1 & , x < 0 \\ +1 & , x \geqslant 0 \end{cases}$$

Formally speaking, the perceptron model acting as a linear discriminant is a pair of weight and bias such that the loss is minimum, i.e.,

$$(\vec{w}, b) = \arg\min_{\substack{\vec{w}\in\mathbb{R}^n \\ b\in\mathbb{R}}} L\left(\vec{x}, y\right)$$

and the loss function L is defined as

$$L\left(\vec{x}, y\right) = -\sum_{j\in M}(\vec{w}^t\vec{x}_j + b_j)y_j \ , \ \vec{x}_j \in \mathbb{R}^n, y_j \in \{-1, +1\}$$

with M denoting the set of all misclassified samples.

The model is convergent guaranteed by perceptron convergence theorem and can be solved by means of stochastic gradient descent algorithm.

universal approximation theorem

universal approximation theorem is one of the cornerstone of NN theory, it states the fact that any NN with only one hidden layer, in general, can be considered a universal approximator in the sense that for any continuous function in compact space like $\mathbb{R}$, an NN can uniformly approximate it to arbitrarily small error [8]. Said formally,

Theorem 3.1.1. *Let $\mathcal{A}$ be the set of the form*

$$\mathcal{A} \triangleq \{\ g \mid g(x) = \sum_{i=1}^{N} \alpha_j\sigma\left(\vec{w}_j^t\vec{x} + b_j\right)\ \} \ , \ \vec{w}_j^t \in \mathbb{R}^n, \vec{x} \in I^n, \alpha_j, b_j \in \mathbb{R}$$

where

$$I^n \triangleq \underbrace{[0,1] \times [0,1] \times \cdots \times [0,1]}_{n \text{ times}} \ , \ \sigma(x) = \begin{cases} 0 & , x \to -\infty \\ +1 & , x \to +\infty \end{cases} , e.g., \sigma(x) = \frac{1}{1+e^{-x}}$$

then, $\mathcal{A}$ is dense in $C[I^n]$. Said differently, $\forall f \in C(I^n)$ and $\forall \varepsilon > 0$, there exists a function $g \in \mathcal{A}$ such that

$$|f(\vec{x}) - g(\vec{x})| < \varepsilon$$

holds for any $\vec{x} \in I^n$

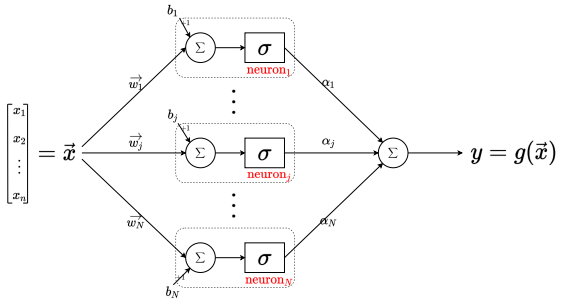


Figure 3.2: A NN with one hidden layer, input vector $\vec{x} = [x_1, x_2, \cdots, x_n]^t \in \mathbb{R}^n$ and output $y \in \mathbb{R}$.

A more general form of the theorem is given in [18], further experiments [26] also reveals the fact that the number of neurons in Deep Neural Network will grow linearly in order to achieve higher approximation accuracy in contrast to the shallow NN which grows exponentially.

Architecture and activation function

A typical NN usually is comprised of multiple hidden layers other than input and output layers to obtain sufficient generalization capability, for example, a NN with 3 hidden layers is represented in 3.3

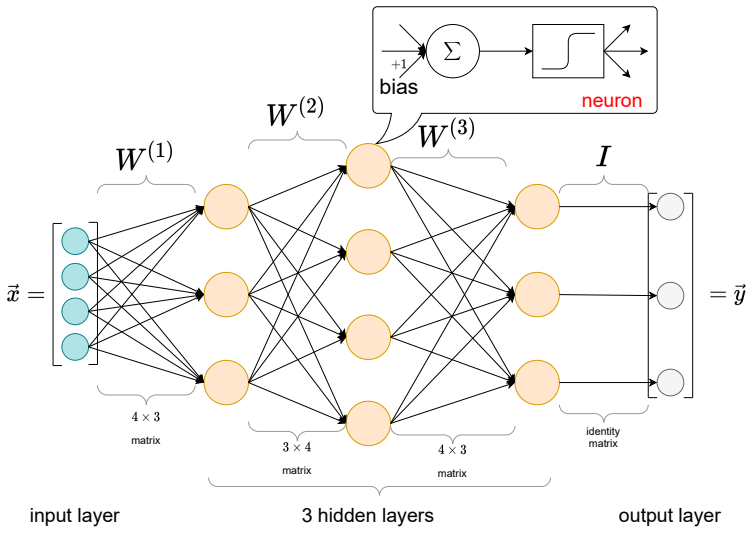


Figure 3.3: Deep neural network (DNN).

With regard to activation functions, they are a family of functions with different characteristics used for the purpose of introducing nonlinearity to NN. The choice for activation functions is application-dependent, and considering derivation operations that we are going to perform in gradient descent algorithm or in back propagation, it is better to exploit smooth and differentiable activation functions. In case if activation functions are absent in the model, the network degenerates to a composition of linear operators, which turns out to be a linear transformation. Here are some commonly used activation functions.

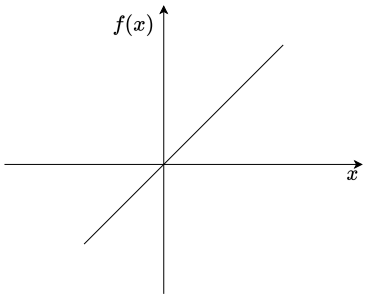


Figure 3.4: Linear function.

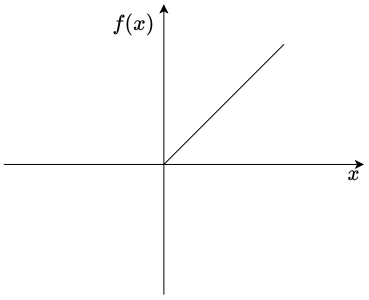
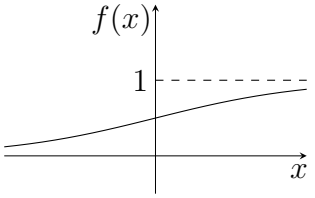
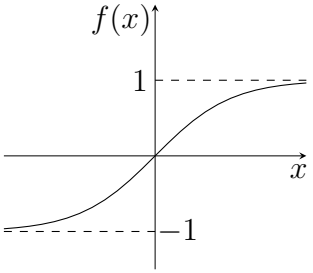


Figure 3.5: Relu function.



$f(x) = \frac{1}{1+e^{-x}}$

Figure 3.6: Sigmoid function.



$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Figure 3.7: Tanh function.

gradient descent and back propagation algorithm

For the simplicity of exposition, we consider the gradient descent in perceptrons, this algorithm can also be extended to general NN though. Suppose $\vec{w} \in \mathbb{R}^n$, and think of the optimization problem

$$\vec{w}^* = \arg\min_{\vec{w} \in \mathcal{W}} L(\vec{w})$$

we apply Taylor expansion to $L(\vec{w})$ at $\vec{w}^{(0)}$

$$L(\vec{w}) = L(\vec{w}^{(0)}) + \nabla_{\vec{w}} L(\vec{w}^{(0)})(\vec{w} - \vec{w}^{(0)}) + o\left(\left\|\vec{w} - \vec{w}^{(0)}\right\|\right) \quad , \quad \nabla_{\vec{w}} L(\vec{w}^{(0)}) = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_n} \end{bmatrix}\Bigg|_{\vec{w}=\vec{w}^{(0)}}$$

By Cauchy-Schwarz inequality, we know

$$\left|\nabla_{\vec{w}} L(\vec{w}^{(0)})(\vec{w}-\vec{w}^{(0)})\right|^2 \leqslant \left\|\nabla_{\vec{w}} L(\vec{w}^{(0)})\right\|^2 \left\|\vec{w}-\vec{w}^{(0)}\right\|^2$$

the equality holds i.f.f. $\vec{w}-\vec{w}^{(0)} = \pm\frac{\nabla_{\vec{w}} L(\vec{w}^{(0)})}{\left\|\nabla_{\vec{w}} L(\vec{w}^{(0)})\right\|}$, where "$-$" sign indicates steep descent direction. Therefore, the iteration equation reads,

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta\frac{\nabla_{\vec{w}} L(\vec{w}^{(k)})}{\|\nabla_{\vec{w}} L(\vec{w}^{(k)})\|} \qquad \eta\text{— learning rate}$$

Usually, we refer to this equation as normalized gradient descent and since the division by $\left\|\nabla_{\vec{w}} L(\vec{w}^{(k)})\right\|$ is just a scaling, we can also simply omit this factor and finally yields gradient descent iteration

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta\nabla_{\vec{w}} L(\vec{w}^{(k)}) \qquad \eta\text{— learning rate}$$

Remember, our main goal is to use gradient descent to find optimal weights for the network such that the loss is minimum, and back propagation is applied to compute gradients in the expression. Take the NN in fig 1.18 for example, we want to find an optimal tuple $W^* \triangleq \{W^{(1)}, W^{(2)}, W^{(3)}\}$ where

$$\begin{aligned}
W^{(1)} &= \left[\vec{w}_1^{(1)}, \vec{w}_2^{(1)}, \vec{w}_3^{(1)}\right] \ , \ \vec{w}_j^{(1)} \in \mathbb{R}^4 \\
W^{(2)} &= \left[\vec{w}_1^{(2)}, \vec{w}_2^{(2)}, \vec{w}_3^{(2)}, \vec{w}_4^{(2)}\right] \ , \ \vec{w}_j^{(2)} \in \mathbb{R}^3 \\
W^{(1)} &= \left[\vec{w}_1^{(3)}, \vec{w}_2^{(3)}, \vec{w}_3^{(3)}\right] \ , \ \vec{w}_j^{(3)} \in \mathbb{R}^4
\end{aligned}$$

assume loss function is defined as

$$L = \frac{1}{2}\left\|\vec{y}-\vec{t}\right\|_2^2 = \frac{1}{2}\sum_j (y_j - t_j)^2$$

according to the gradient descent algorithm, we need to solve $\frac{\partial L}{\partial w_{ij}^{(l)}}$ for each layer $l = 1, 2, 3$. By chain rule, we will know that the gradients for $l = 2$ involves gradients of $l = 3$, and same reason applied to $l = 1$, so we compute gradients of the last hidden layer $l = 3$ first

$$\frac{\partial L}{\partial w_{ij}^{(3)}} = \frac{\partial L}{\partial y_j}\cdot\frac{\partial y_j}{\partial h^{(3)}}\cdot\frac{\partial h^{(3)}}{\partial w_{ij}^{(3)}} = (y_j - t_j)\cdot 1\cdot f'\cdot f(h^{(2)})$$

where

$$\begin{aligned}
h^{(1)} &= {W^{(1)}}^t\vec{x} + b^{(1)} \\
h^{(2)} &= {W^{(2)}}^t f\left(h^{(1)}\right) + b^{(2)} \\
h^{(3)} &= {W^{(3)}}^t f\left(h^{(2)}\right) + b^{(3)} \\
\vec{y} &= f\left(h^{(3)}\right)
\end{aligned}$$

Then, we can apply chain rule layer by layer and backwards for $\frac{\partial L}{\partial w_{ij}^{(2)}}$ and $\frac{\partial L}{\partial w_{ij}^{(1)}}$.

3.1.2 Recurrent neural network

As was briefly discussed before, conventional NN has disadvantage in handling time series data due to the lack of memory effect, for this reason, recurrent neural network (RNN) is proposed to capture time dependency among sequential data and therefore it possesses the capability to memorize historical information. One way to implement memory effect is to share parameter across time by introducing a feedback loop into the network. Parameter sharing plays an role of memory effect in a sense that historical information can be stored in shared parameters. Figure 3.8 depicts a general architecture of RNN where we suppose the input sequence length is fixed to p.

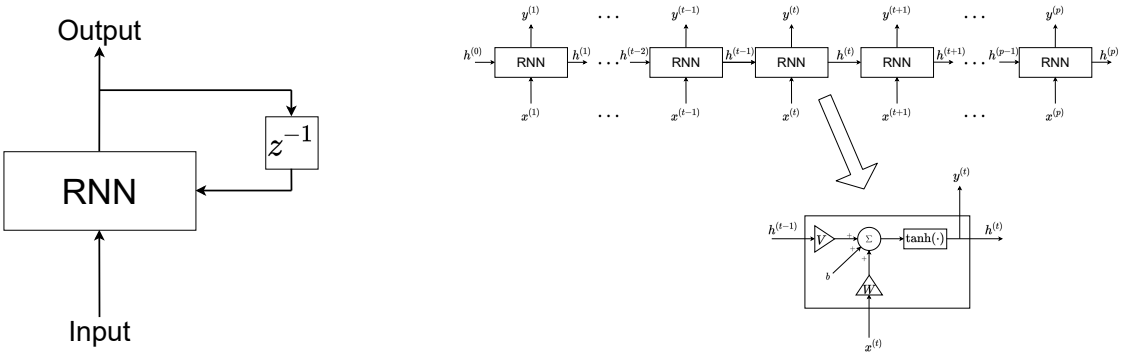


Figure 3.8: A classical RNN: compact form is on the left, diagram on the right corresponds unfolded form across time.

a classical RNN can characterized by a system of equations

$$\begin{cases} h^{(t)} &= \tanh\left(Vh^{(t-1)} + Wx^{(t)} + b\right) \\ y^{(t)} &= h^{(t)} \end{cases}$$

where

$$\begin{aligned} h^{(t)} &\text{— hidden state vector at } t \\ x^{(t)} &\text{— input vector at } t \\ y^{(t)} &\text{— output vector at } t \\ V, W, b &\text{— shared parameters} \end{aligned}$$

It is also worth noting that a RNN is essentially a nonlinear Kalman filter in the extended sense if we compare the equations above with the equations for Kalman filter as follows

$$\begin{cases} h^{(t)} &= T(t, t-1)h^{(t)-1} + x^{(t-1)} \\ y^{(t)} &= C(t)h^{(t)} + v^{(t)} \end{cases}$$

whereas now,

$$T(t, t-1) \text{— state transition matrix from } t-1 \text{ to } t$$
$$x^{(t)} \text{— process noise}$$
$$C(t) \text{— observation matrix at } t$$
$$v^{(t)} \text{— observation noise}$$
$$h^{(t)} \text{— state vector}$$
$$y^{(t)} \text{— observation vector}$$

Our main goal is to find optimal shared parameters such that the error calculated by loss functions is minimum. We first define the loss functions L [11].

$$L = \sum_{t=1}^{p} L^{(t)}$$

with

$$L^{(t)} = \frac{1}{2} \left\| y^{(t)} - t^{(t)} \right\|_2^2 = \frac{1}{2} \sum_{j=1}^{q} \left(y_j^{(t)} - t_j^{(t)} \right)^2 \ , \ y^{(t)}, t^{(t)} \in \mathbb{R}^q$$

and $t^{(t)}$ is the target vector at t.

In order to achieve this, we are going to proceed with the following steps:

1. compute $\frac{\partial L}{\partial h_j^{(t)}}$

As far as gradient descent is concerned, BP algorithm is no longer applicable to solve gradients of L with respect to time t, to cope with this issue, BPTT is represented in [31] and we demonstrate it roughly with a slightly different layout of RNN architecture as is in the figure 3.9.

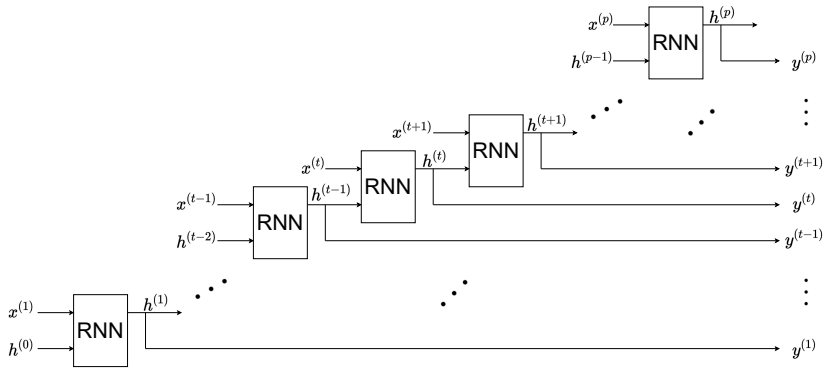


Figure 3.9: Graphic representation of BPTT, adapted from [31].

the derivative of L with respect to the hidden state h_j at last layer is

$$\frac{\partial L}{\partial h_j^{(p)}} = \frac{\partial L}{\partial L^{(p)}} \cdot \frac{\partial L^{(p)}}{\partial y_j^{(p)}} \cdot \frac{\partial y_j^{(p)}}{\partial h_j^{(p)}} = 1 \cdot (y_j^{(p)} - h_j^{(p)}) \cdot 1$$

Similarly, by exploiting chain rule we can also acquire $\frac{\partial L}{\partial h_j^{(t)}}$, $\frac{\partial L}{\partial h_j^{(t-1)}}$, ... layer by layer and backwards until it touches the first one $h_j^{(1)}$, once it is done, we are able to obtain $\frac{\partial L}{\partial w_{ij}}$, $\frac{\partial L}{\partial v_{ij}}$, and $\frac{\partial L}{\partial b_j}$ [12].

3.1.3 Long Short-Term Memory

Vanishing gradient is the problem that gradient tends to vanish when it is propagated backwards through many time steps. Specifically, we know the system of equations for RNN with loss function is

$$\begin{cases} L & = \sum_{t=1}^{p} L^{(t)} \ , \ L^{(t)} = \frac{1}{2} \left\| y^{(t)} - t^{(t)} \right\|_2^2 \\ h^{(t)} & = \tanh\left(V h^{(t-1)} + W x^{(t)} + b \right) \\ y^{(t)} & = h^{(t)} \end{cases}$$

where $L^{(t)}$ and $h^{(t)}$ are functions of w_{ij}. Thus,

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{p} \left(\frac{\partial L}{\partial h_i^{(t)}} \cdot \frac{\partial h_i^{(t)}}{\partial w_{ij}} \right)$$

where $\frac{\partial L}{\partial h_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \cdot \frac{\partial L^{(t)}}{\partial y_i^{(t)}} \cdot \frac{\partial y_i^{(t)}}{\partial h_i^{(t)}}$ is the expression we solved before, by applying chain rule, we get

$$\frac{\partial h_i^{(t)}}{\partial w_{ij}} = \frac{\partial h_i^{(t)}}{\partial w_{ij}} + \frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} \cdot \frac{\partial h_i^{(t-1)}}{\partial w_{ij}}$$

Similarly, if we apply chain rule recursively to equation above, we will get a set of equations

$$\begin{cases} \frac{\partial h_i^{(t-1)}}{\partial w_{ij}} & = \frac{\partial h_i^{(t-1)}}{\partial w_{ij}} + \frac{\partial h_i^{(t-1)}}{\partial h_i^{(t-2)}} \cdot \frac{\partial h_i^{(t-2)}}{\partial w_{ij}} \\ \frac{\partial h_i^{(t-2)}}{\partial w_{ij}} & = \frac{\partial h_i^{(t-2)}}{\partial w_{ij}} + \frac{\partial h_i^{(t-2)}}{\partial h_i^{(t-3)}} \cdot \frac{\partial h_i^{(t-3)}}{\partial w_{ij}} \\ & \vdots \\ \frac{\partial h_i^{(2)}}{\partial w_{ij}} & = \frac{\partial h_i^{(2)}}{\partial w_{ij}} + \frac{\partial h_i^{(2)}}{\partial h_i^{(1)}} \cdot \frac{\partial h_i^{(1)}}{\partial w_{ij}} \\ \frac{\partial h_i^{(1)}}{\partial w_{ij}} & = \frac{\partial h_i^{(1)}}{\partial w_{ij}} + 0 \end{cases}$$

and we substitute them for equation $\frac{\partial h_i^{(t)}}{\partial w_{ij}} = \frac{\partial h_i^{(t)}}{\partial w_{ij}} + \frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} \cdot \frac{\partial h_i^{(t-1)}}{\partial w_{ij}}$, yield

$$\begin{aligned} \frac{\partial h_i^{(t)}}{\partial w_{ij}} &= \frac{\partial h_i^{(t)}}{\partial w_{ij}} + \frac{\partial h_i^{(t)}}{\partial h_i^{(t-1)}} \left[\frac{\partial h_i^{(t-1)}}{\partial w_{ij}} + \frac{\partial h_i^{(t-1)}}{\partial h_i^{(t-2)}} \left(\frac{h_i^{(t-2)}}{\partial w_{ij}} + \frac{\partial h_i^{(t-2)}}{\partial h_i^{(t-3)}} \left(\cdots \left(\frac{\partial h_i^{(1)}}{\partial w_{ij}} + 0 \right) \cdots \right) \right) \right] \\ &= \sum \cdots \sum \prod_{k=1}^{t-1} \frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}} \end{aligned}$$

Finally we get

$$\frac{\partial L}{\partial w_{ij}} = \sum_{t=1}^{p} \frac{\partial L}{\partial h_i^{(t)}} \cdot \left(\sum \cdots \sum \prod_{k=1}^{t-1} \frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}} \right)$$

Intuitively, since we have exponential term $\prod_{k=1}^{t-1} \frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}}$ in the gradient expression where t is up to p (the whole sequence), it is easy to deduce that:

- if $\frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}} < 1$, then $\frac{\partial L}{\partial w_{ij}}$ converges to 0 when $p \to \infty$.
- if $\frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}} > 1$, $\frac{\partial L}{\partial w_{ij}}$ will explode when $p \to \infty$.

Long Short-Term Memory (LSTM) is thereby proposed to overcome vanishing gradient that might happen on RNN. The intuition is simple: keep $\frac{\partial h_i^{(k+1)}}{\partial h_i^{(k)}}$ constantly equal to 1, which results in the constant error flow, and we refer to this idea as *constant error carrousel* (CEC) [17]. Specifically, we insert an internal feedback loop into the cell in addition to some auxiliary gates: input gate, block input, and output gate. See in figure 3.10

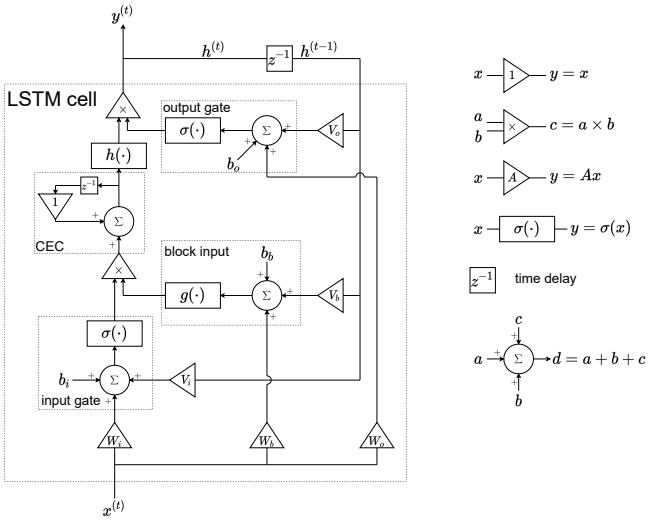


Figure 3.10: Architecture of LSTM cell: long-term memory is stored in CEC, input gate prevents CEC from perturbation by irrelevant input and output prevents other units from perturbation by contents in CEC, adapted from [15].

In that case, we isolate CEC from perturbation, as a result, the existence of CEC in the cell enables LSTM possessing temporal (short-term) memory while maintaining long-term memory meanwhile so that LSTM is more capable of handling sequence tasks compared to RNN. Nevertheless, it brings another issue when processing continual input stream. On one hand, LSTM does not know when the stream ends, so it will process stream all the time and fails to reset itself when necessary. Besides, due to the self-connected loop in CEC, continual processing can lead to internal states growing unboundedly and finally result in output saturated. On the other hand, the errors may also be trapped in the self-connected loop permanently and can not be backpropagated through time for updating weights. All these consequences finally results the network breaking down.

A novel functional component referred to as forget gate is then employed for the purpose of resetting itself as well as releasing memory resource at appropriate time. Basically, forget gate is added to the self-connected loop in CEC allowing network remove irrelevant historical information from the memory while keeping crucial long-term time dependency in the mean-time [9] [10]. See in figure 3.11.

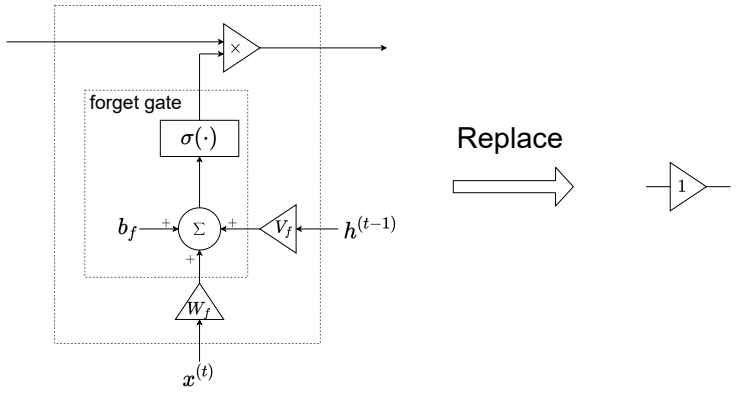


Figure 3.11: Unitary gain in CEC is replaced with the forget gate.

which finally yields the commonly used LSTM cell illustrated in Figure 3.12 below

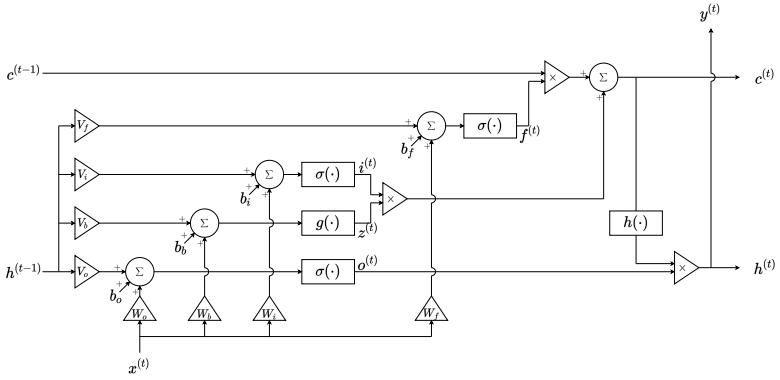


Figure 3.12: A LSTM cell, adapted from [15].

where the system of equations is,

$$\begin{cases} z^{(t)} &= g\left(V_b h^{(t-1)} + W_b x^{(t)} + b_b\right) \\ i^{(t)} &= \sigma\left(V_i h^{(t-1)} + W_i x^{(t)} + b_i\right) \\ f^{(t)} &= \sigma\left(V_f h^{(t-1)} + W_f x^{(t)} + b_f\right) \\ c^{(t)} &= c^{(t-1)} \cdot f^{(t)} + i^{(t)} \cdot z^{(t)} \\ o^{(t)} &= \sigma\left(V_o h^{(t-1)} + W_o x^{(t)} + b_o\right) \\ h^{(t)} &= h\left(c^{(t)}\right) \cdot o^{(t)} \\ y^{(t)} &= h^{(t)} \end{cases}$$

A more compact graphic representation of this LSTM cell is cited from google.

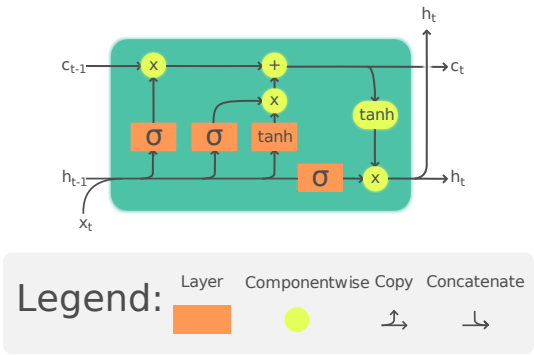


Figure 3.13: A commonly used figure from [7].

while every component can be interpreted in such an natural way:

- forget gate decides what information needs to be removed from memory.
- input gate and block input choose the information to be stored.
- output gate choose information to be output.

3.1.4 Related work

Memory effect and forget gate allow LSTM to remove irrelevant contents from memory and maintain both long-term data correlation in sequential data at the same time without need to specify the sequence end. Considering most real-world datasets comprising abundant nominal but only very few anomalies instances, Pankaj Malhotra [27] utilizes a predictor with multiple LSTM layers to perform anomaly detection regardless of sequence length and preprocessing. The network has two LTSM layers stacked sequentially and all units at two LTSM layers are fully connected, that is, each unit at lower layer is connected to all units at upper layer as is depicted in figure 3.14. Globally, the network is trained on nominal dataset in order to learn normal behaviors of data, and specifically, it will be used to predict future values for a given input while prediction error is difference between predictions and actual stream values. When move to real scenario, the prediction error is used as a score indicating anomalies, this network has been proved efficacious on four datasets.

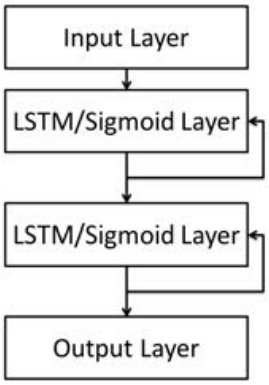


Figure 3.14: The LSTM network adopted in [27].

To better explain how it works, suppose a nominal time series $X = \{x^{(1)}, x^{(2)}, \cdots, x^{(N)}\}$ and $x^{(i)} \in \mathbb{R}^n, 1 \leqslant i \leqslant N$. At each time instance t, the network will take a certain input, $\{\cdots, x^{(t)}\}$ say, and predict next q values $\{\hat{x}^{(t+1)}, \hat{x}^{(t+2)}, \cdots, \hat{x}^{(t+q)}\}$ with $\hat{x}^{(i)} \in \mathbb{R}^d$, $t+1 \leqslant i \leqslant t+q$. For example, at time instance $t = 1$, the input is $\{\underbrace{\vec{0}, \vec{0}, \cdots, \vec{0}}_{(N-1)\times\vec{0}}, x^{(1)}\}$, and our model will predict next q values after $x^{(1)}$, namely, $\{\hat{x}^{(2)}, \hat{x}^{(3)}, \cdots, \hat{x}^{(q+1)}\}$, where $\hat{x}^{(i)} \in \mathbb{R}^d$, $2 \leqslant i \leqslant q+1$. And at next time instance $t = 2$, the input becomes $\{\underbrace{\vec{0}, \vec{0}, \cdots, \vec{0}}_{(N-2)\times\vec{0}}, x^{(1)}, x^{(2)}\}$, while the output now is $\{\hat{x}^{(3)}, \hat{x}^{(4)}, \cdots, \hat{x}^{(q+2)}\}$. Similarly, at any time t, we have input $\{\underbrace{\vec{0}, \vec{0}, \cdots, \vec{0}}_{(N-t)\times\vec{0}}, x^{(1)}, x^{(2)}, \cdots, x^{(t)}\}$ and output $\{\hat{x}^{(t+1)}, \hat{x}^{(t+2)}, \cdots, \hat{x}^{(t+q)}\}$. This process will proceed until the end where we have input $\{x^{(1)}, x^{(2)}, \cdots, x^{(N)} = X\}$. It is illustrated as follows

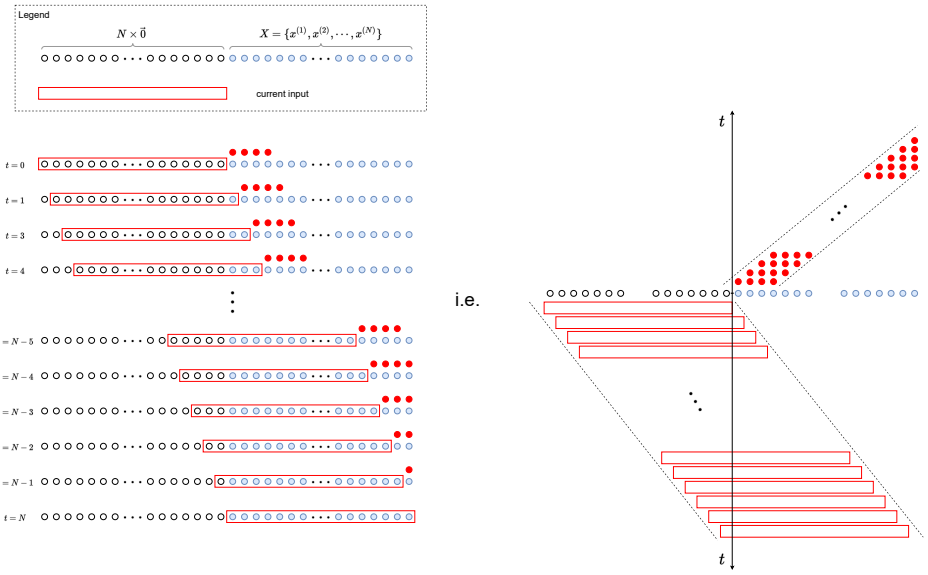


Figure 3.15: A graphic interpretation of how LSTM works across time steps. Note that, non-existing elements are replaced with $\vec{0}$'s.

Clearly, for any $q \leqslant t \leqslant N$, $x^{(t)}$ will be predict q times ($q = 4$ in the figure), thus, error vector for $x^{(t)}$ is given by

$$\vec{e} = \left[e_{11}^{(t)}, e_{12}^{(t)}, \cdots, e_{1q}^{(t)}, \cdots, e_{d1}^{(t)}, e_{d2}^{(t)}, \cdots, e_{dq}^{(t)}\right]^t$$

where $e_{ij}^{(t)}$ is the difference between $x_i^{(t)}$ and its prediction at time $(t-j)$. Once the training is completed and we set an appropriate anomaly threshold, the anomalies can be detected then.

Moreover, Kyle deploys an LSTM network using nonparametric dynamic thresholding to detect anomalies in high volume telemetry data without need of expert labels [19]. This network is also trained on nominal data in order to capture normal behavior of the system, after that, an unsupervised, nonparametric dynamic thresholding approach is applied to automatically identify whether a prediction error represents an anomaly. In order to provide an granular view of system as well as locating anomalies effectively once they are uncovered, each channel has a model used to predict a next value given an input vector. Since abrupt changes in

time series values often gives rise to unperfect predictions and results in sharp spikes in error values even if the behavior is normal, prediction error vector has to be smoothed before it is compared with the threshold identifying anomalies, besides, another method called anomaly scoring [25] can also be utilized to address the aforementioned issue related to unperfect prediction. And thanks to dynamic thresholds, extreme values that can be considered anomalies will be detected regardless of assumptions about error distribution. Furthermore, to mitigate false positives while easing memory and computation cost, a pruning procedure is carried out to exclude false anomalies caused by noise and priori knowledge, such as anomaly history or labeled data, also helps us set another threshold to balance precision and recall.

Similarly, a deep LSTM predictor is also applied to anomaly detection in [29] as well as ECG data [6], and does not require priori knowledge, expert-labelling, and preprocessing. The network is deep in a sense that it consists of three fully connected LSTM layers with dropout=0.2 for each LSTM layer to avoid overfitting, it is trained on nominal data as usual so as to learn higher level features. An anomaly is confirmed if both the anomaly score and window score are above their corresponding thresholds, whereas anomaly score is defined as likelihood of error vector which is assumed to comply with Gaussian distribution that can be estimated on validation data using Maximum Likelihood Estimation, and window score is amount of anomaly candidates while anomaly candidate is the data point whose anomaly score exceeds the threshold that is selected by maximizing F_β-*score*.

3.2 A toy case on ECG signal

In this chapter, we will explore how various parameters impact the performance of different type of neural networks. For the sake of simplicity, we only consider shallow networks, meaning that all networks in this chapter are composed of only one hidden layer, and we leverage those neural networks to detect anomalies in dataset by prediction. To do that, we will train them on nominal ECG data so that they will learn normal behaviors as well as underlying patterns from nominal data, each neural network will take past p ECG samples into consideration and predict next q values of them. All the experiments are carried out on dataset 'sel102' in database 'qtdb' comprising 225000 ECG samples, an example of it is given,

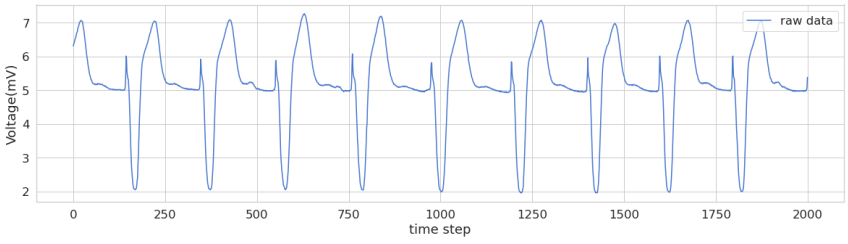


Figure 3.16: Raw ECG data, whereas each time step in x-axis is $\frac{1}{f_s}$ with $f_s = 250$ [Hz], and y-axis indicates voltage in mV.

Once raw data is acquired, we firstly standardize the whole dataset in terms of the mean μ and standard deviation σ estimated on nominal data by applying *StandardScaler()*, in that case, we explicitly exclude anomalies in computation for two reasons:

1. anomalies only represent abnormal behaviors and since they are minority in dataset, they do not represent major characteristics of dataset and therefore do not possess much statistical meaning as a consequence.

2. anomalies are so rare that their contribution to overall μ and σ is negligible even if we explicitly include them in computation.

After standardization, the dataset is then divided into 3 subsets: training set, test set, and anomaly set while the first two subsets consist of only nominal data and the anomaly set contains the only abnormal ECG signal.

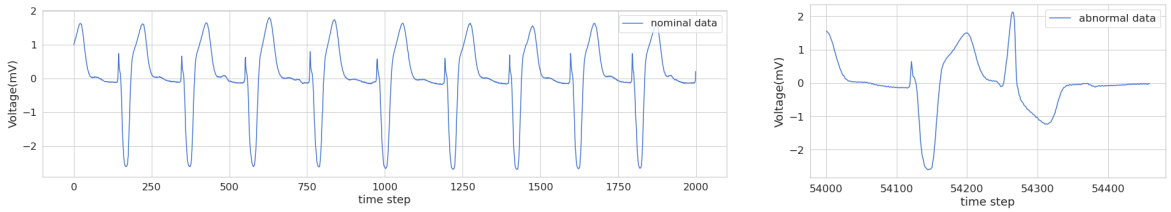


Figure 3.17: Nominal data after standardization shown on the left, while abnormal data after standardization represented on the right.

And we reshape those datasets in terms of input length p and output length q by *sub-dataset_generator()* so that they are compatible with the input shape and output shape of neural networks. Once it is done, training and test datasets will be applied for network training and validation with the loss defined as mean square error, parameter *dropout* and *patience* are also needed for the purpose of avoiding overfitting and early stopping. After that, test dataset is used once again for test (Note that, test dataset will be used twice, one for validation and the other in test stage.).

At each time step t, the network will take last p samples from time series, ECG signal in that case, as input

$$\text{Input} \triangleq \left\{ x^{(t-p+1)}, x^{(t-p+2)}, \cdots, x^{(t-1)}, x^{(t)} \right\} \quad , \quad x^{(t)} \in \mathbb{R}^n$$

and predicts the next q values as the output accordingly,

$$\text{Output} \triangleq \left\{ \hat{x}^{(t+1)}, \hat{x}^{(t+2)}, \cdots, \hat{x}^{(t+q-1)}, \hat{x}^{(t+q)} \right\} \quad , \quad \hat{x}^{(t)} \in \mathbb{R}^n$$

where $n = 1$ for the ECG experiments. The following graphics 3.18 3.19 demonstrate this idea with a detailed representation of RNN or LSTM,

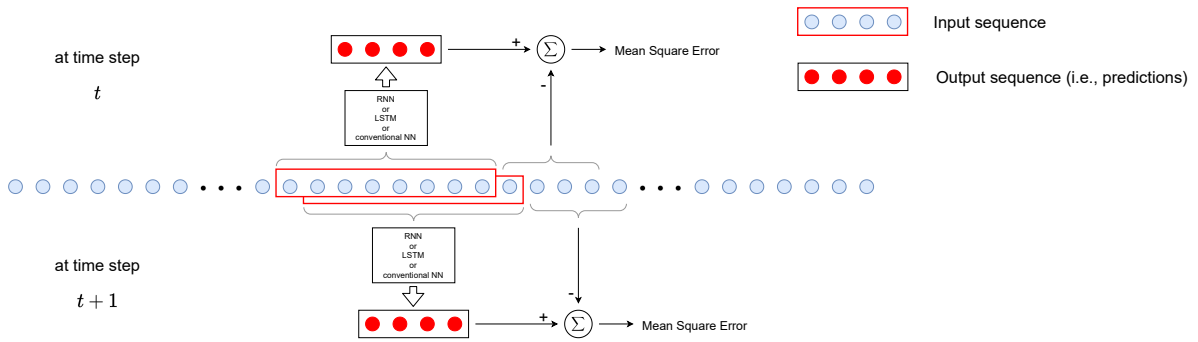


Figure 3.18: General idea of how network works, similar to [27].

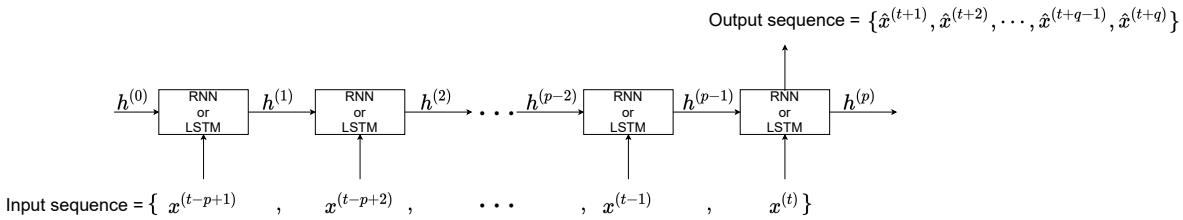


Figure 3.19: Unrolling RNN or LSTM network.

The mean square error at t, denoted as $e^{(t)}$, accordingly is:

$$e^{(t)} = \frac{1}{q}\sum_{i=1}^{q}\left(e_i^{(t)}\right)^2 \qquad , \quad e_i^{(t)} = \left(\hat{x}^{(t+i)} - x^{(t+i)}\right)$$

Keras will minimize the loss and ends up with an well-trained neural network, we will investigate how different parameters impact the performance of neural networks in the following sections.

3.2.1 Performance versus number of neurons

Network performance depends on generalization capability which is partially relevant to the number of neurons in networks, and we will explore its impacts as follows. In this experiment, we fix input length $p = 8$ and output length $q = 4$, and train three types of neural networks: RNN, LSTM, and conventional NN, for each type of neural network, the number of neurons is given by 2^n where n varies from 0 to 7. The experiment result is depicted in the figure 3.20,

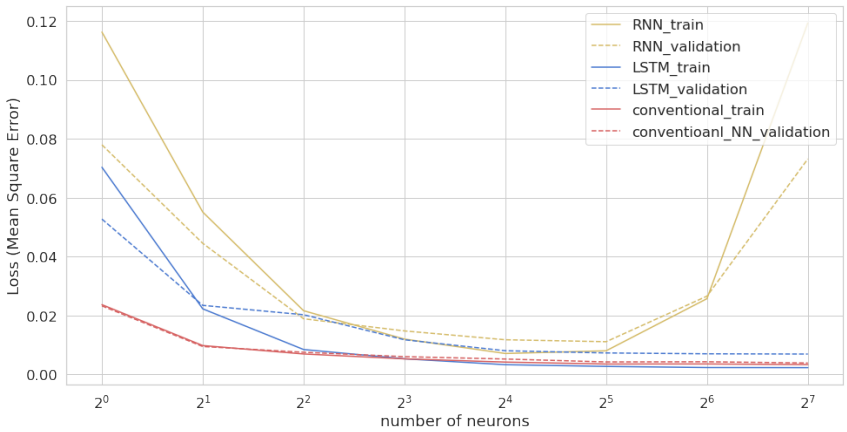


Figure 3.20: Loss versus number of neurons, and input length $p = 8$, output length $q = 4$ for each NN.

It is evident to say that the performance of conventional NN are superior than LSTM, both conventional NN and LSTM outperform RNN, apart from that, their performance also increase monotonically with the number of neurons increasing, this is because the generalization capability will increases if we have more neurons in the network. It is also worth notice that the

improvement of performance for LSTM and conventional NN are not so significant after 2^2 even if we continue to enlarge the number of neurons.

On the other hand, the performance of RNN increase with respect to the augmenting of number of neurons, but decreases after 2^5 even if the number of neurons is still augmenting. This is reasonable since RNN may have some troubles with training when the input length p is large, because the error will be backpropagated through many time step during this process.

3.2.2 Performance versus input length

In this experiment, we study how input length p impacts network performance, to do that, we firstly fix output length $q = 4$, number of neurons equals to 64 and we examine three types of networks: RNN, LSTM, and conventional NN. For each type of network, the input length p varies from 1 to 16. The result is shown as follows,

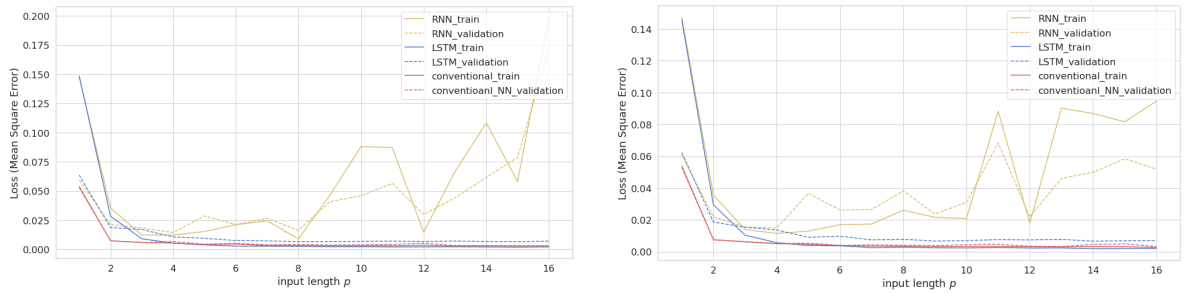


Figure 3.21: On the left, loss versus input length p with output length $q = 4$ and number of neurons fixed to 64. On the right, we have loss versus input length p with output length $q = 4$, but we fix the number of neurons per neurons constantly equals to 8 so that there is no worry about the lack of generalization capability.

In general, the performance of LSTM and conventional NN are better than RNN as is shown in the left graph, and they all increase while p is growing at the beginning mainly because longer input sequence can provide more information to networks so that they can make predictions better. However, the improvement of performance for LSTM and conventional NN is so tiny after $p = 4$. On the contrary, performance of RNN starts fluctuating dramatically after $p = 4$ and even worse when input length p consistently increasing. On the other hand, one may worry about whether generalization capability of networks will be limited by the number of neurons in the network when p is too large. To verify this idea, we additionally carried out an experiment with same setups, but the number of neurons per input length is fixed to 8. As a result, there is no remarkable difference observed compared to the outcomes from previous experiment, which means that generalization capability will not be limited by the number of neurons in the network even if p is large.

3.2.3 Performance versus output length

Moreover, we investigate how output length q impacts network performance by fixing input length $p = 8$ and number of neurons to 64, then we will evaluate the performance of three types of networks as usual, the output length q for each type of neural network shall vary from 1 to 16. The results are as represented in figure 3.22,

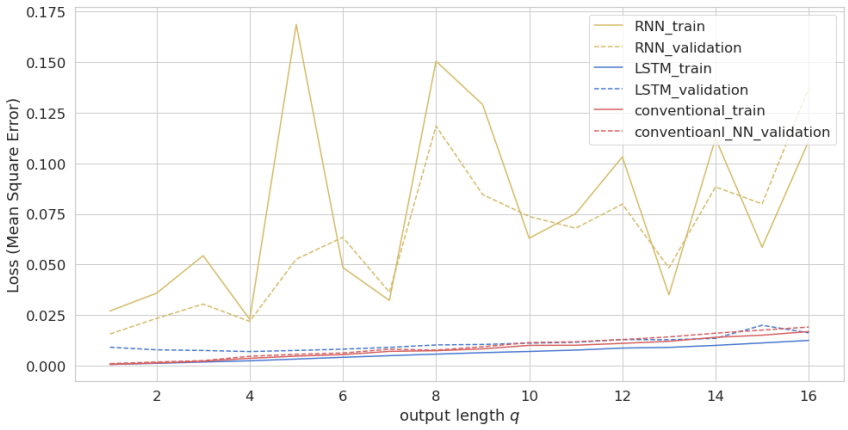


Figure 3.22: Loss versus output length q with input length p fixed to 8 and 64 neurons in the network.

As is expected, for LSTM and conventional NN, their loss tend to increase if output length q is increasing, and We provide an intuitive proof to explain this phenomenon. The mean square error at time t is a function of q (we omit mention of time t)

$$e = f(q) = \frac{1}{q}\sum_{i=1}^{q} E_i \qquad , \quad E_i = e_i^2$$

and since it is more difficult to forecast far distant future values, it is therefore reasonable to assume that E_i is monotonically increasing with respect to time index i, said differently, $E_{i+1} \geqslant E_i$, and in general, it is correct as is depicted in the figure 3.23.

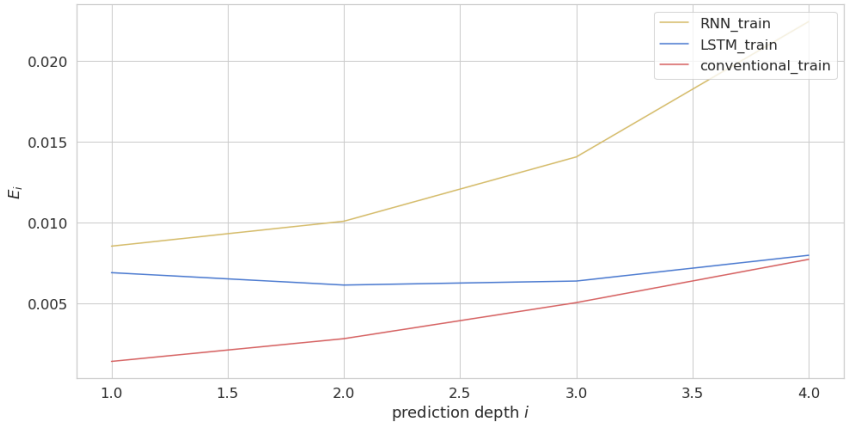


Figure 3.23: E_i versus time index i.

Our main goal is to prove that loss tends to increase when output length q is growing, in other words, $f(q)$ is monotonically increasing with respect to output length q, or said equivalently,

$$f(q+1) \geqslant f(q) \qquad , \quad \forall q \in \mathbb{N}$$

Proof.

$$\begin{aligned} f(q+1) - f(q) &= \frac{q}{q+1} \cdot \frac{1}{q} \left(\sum_{i=1}^{q} E_i + E_{q+1} \right) - \frac{1}{q} \sum_{i=1}^{q} E_i \\ &= \frac{\sum_{i=1}^{q} E_i}{q} \left(\frac{q}{q+1} - 1 \right) + \frac{E_{q+1}}{q+1} \\ &= -\frac{\sum_{i=1}^{q} E_i}{q(q+1)} + \frac{E_{q+1}}{q+1} \\ &= \frac{qE_{q+1} - (\sum_{i=1}^{q} E_i)}{q(q+1)} \end{aligned}$$

since we know $E_{i+1} \geqslant E_i$, for $\forall 1 \leqslant i \leqslant q$, thus

$$qE_{q+1} \geqslant \sum_{i=1}^{q} E_i$$

in other words,

$$f(q+1) - f(q) \geqslant 0$$

□

On the other hand, the profile of RNN is still terrible and no need to put any comments on it.

3.2.4 Performance versus type of neural networks

In this experiment, we will see how the type of neural networks impacts their performance. Since both RNN and LSTM have memory effect in contrast to conventional NN, and LSTM even has long-term memory, theoretically speaking, if we enlarge input length p, RNN and LSTM are capable of fully utilizing historical information when they are performing predictions, and we shall expect that the performance: LSTM $>$ RNN $>$ conventional NN in that case. Whilst if p is small, performance improvement of RNN and LSTM may not so distinguishable in contrast to conventional NN. To verify this idea, we fix output length $q = 4$ and 128 neurons for each network guaranteeing sufficient generalization capability, then we train RNN, LSTM, and conventional NN respectively with the input length p varying from 1 to 16 for each type of neural network. The results are represented below

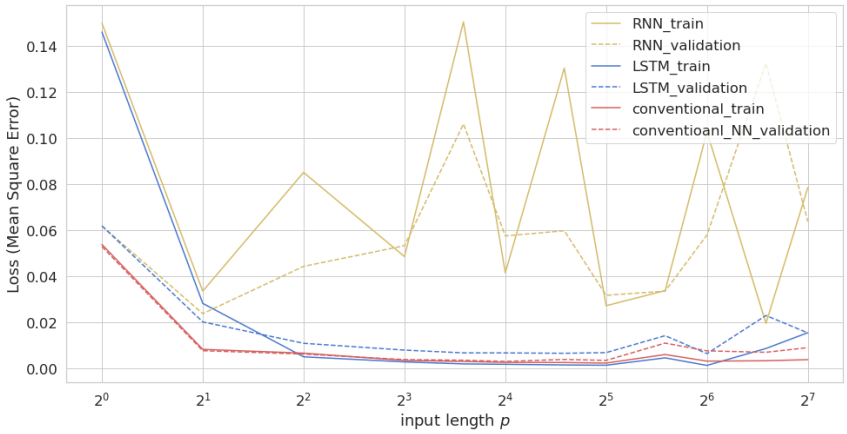


Figure 3.24: Loss versus NN type, fix output length to 4 and number of neurons to 128.

It is obvious that LSTM and conventional NN are superior than RNN, and performance of RNN is shitty as usual. Specifically, conventional NN is better than LSTM in a sense that it outperforms LSTM a lot when p is small, but this advantage vanishes as p increasing.

We conclude that conventional NN always wins because ECG possesses an obvious pattern that can be easily learnt using conventional NN without need of RNN or LSTM, another possible reason is: for RNN and LSTM, prediction takes historical information as well as previous outputs into account, but the fact is prediction on ECG data does not involves historical information and irrelevant to previous outputs (as RNN and LSTM also take previous outputs into consideration), thus, it suffices to precisely predict future values just based on present input sequence via conventional NN.

3.2.5 Examination on test dataset

In this experiment, we will try our models on test dataset containing an anomalous event with input length $p = 8$, output length q fixed to 4 and 1 separately. According to the outcome from the first experiment, the number of neurons we choose for RNN is 32, and 64 for LSTM and conventional NN, so that we can obtain a relatively good performance while maintaining smaller number of parameters in the meantime. Considering the case where $p = 8$ and $q = 4$ (same applied to the case where $p = 8$ and $q = 1$), once the models are well-trained and validated on training and test datasets respectively, we then feed them again with test dataset. Suppose the test dataset is a sequence of time series of length l

$$X = \{\, x^{(1)}, x^{(2)}, \cdots, x^{(l-1)}, x^{(l)} \,\}$$

Since we know that our model will take p consecutive samples from X at each time instance t, and predict next q values after them. An example of predictions are shown in the figure 3.25.

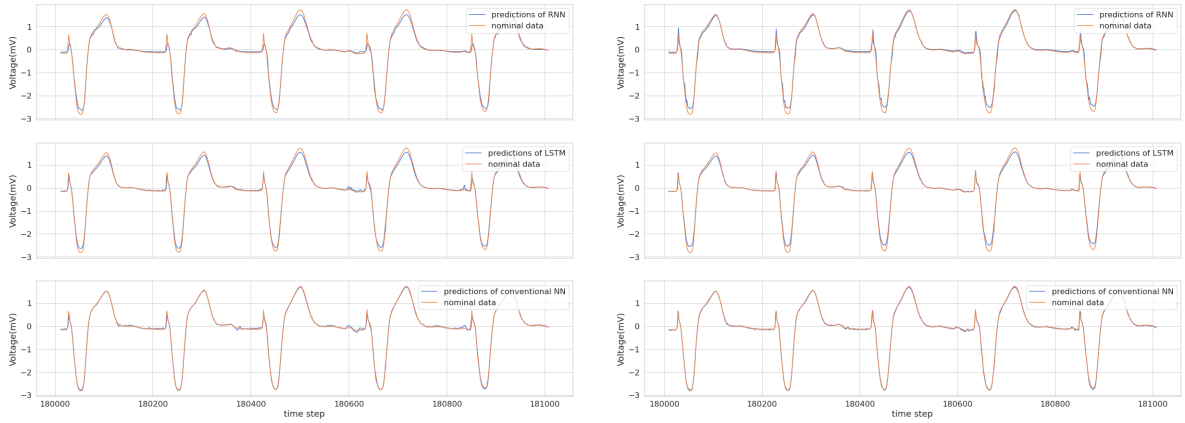


Figure 3.25: Prediction on test dataset, output length $q = 4$ on the left and $q = 1$ on the right.

As a result, for each $x^{(t)} \in X$, $p + q \leqslant t \leqslant l$, it will be predicted q times at $t - q$, $t - q + 1$, $\cdots$, and $t - 1$ respectively. see in the figure 3.26.

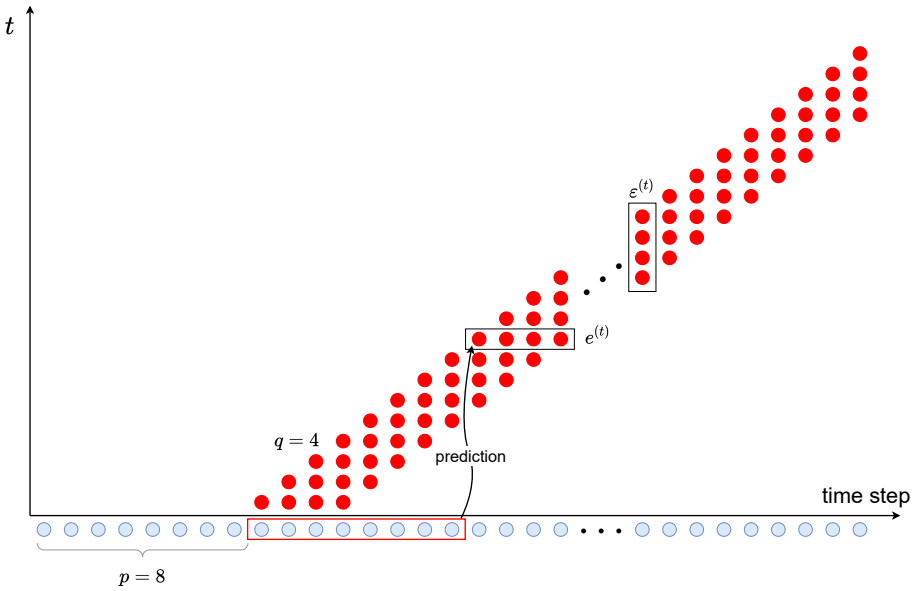


Figure 3.26: Graphic explanation about how NN perform prediction on test dataset X.

and we define error vector of $x^{(t)}$[1] as

$$\varepsilon^{(t)} = \left[\varepsilon_1^{(t)}, \varepsilon_2^{(t)}, \cdots, \varepsilon_q^{(t)}\right]^t$$

where each component $\varepsilon_i^{(t)}$ is the prediction error of $x^{(t)}$ at time $t - q + i - 1$, hence, mean square error (MSE) of $x^{(t)}$ reads

$$\text{MSE} = \frac{1}{q}\sum_{i=1}^{q}\left(\varepsilon_i^{(t)}\right)^2$$

[1]NOT aforementioned $e^{(t)}$! Watch out the different notations: $e^{(t)}$ is mean square error at t and $\varepsilon^{(t)}$ stands for the error vector of $x^{(t)}$

With the help of appropriate data layout and visualization, all those error vectors constitute a matrix of size $(l-p) \times q$, denoted as A, while each column is the error vector $\varepsilon^{(t)}$.

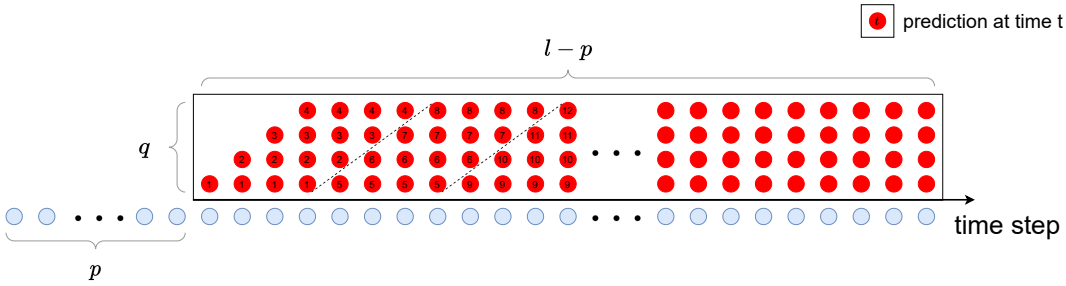


Figure 3.27: Prediction error matrix A.

Regarding the data layout, each $e^{(t)}$, $p \leqslant t \leqslant l$ is allocated to $t \pmod q$-th row with column index starting from t. elements in $e^{(t)}$ will be filled into $A_{t \pmod q,t}, A_{t \pmod q,t+1}, \cdots$, $A_{t \pmod q,t+q}$ consecutively. Then we compute MSE of $x^{(t)}$ for $p+q \leqslant t \leqslant l$ and plot the distribution of MSE after that in order to obtain the threshold for anomaly detection [35].

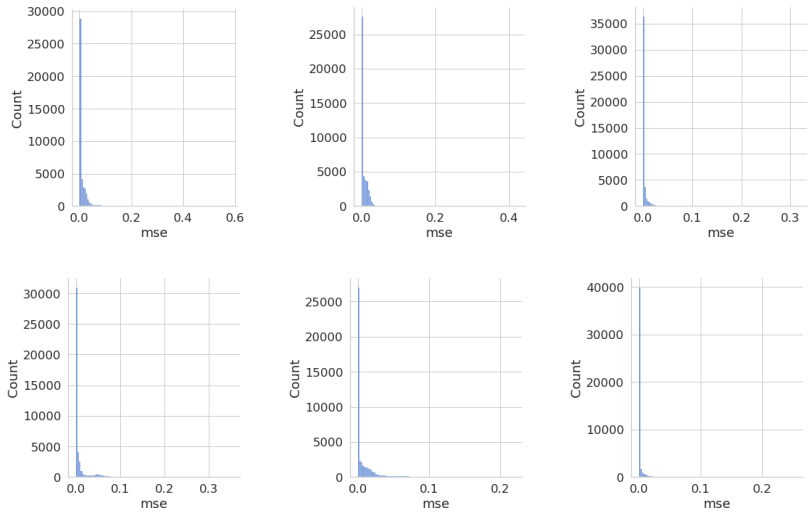


Figure 3.28: Top: MSE distribution of RNN, LSTM, and conventional NN with $q = 4$. Bottom: MSE distribution of RNN, LSTM, and conventional NN with $q = 1$

For the sake of simplicity, we take maximum MSE value as the threshold for anomaly detection. Then, we examine models then on the anomaly dataset, see in the figure 3.29.

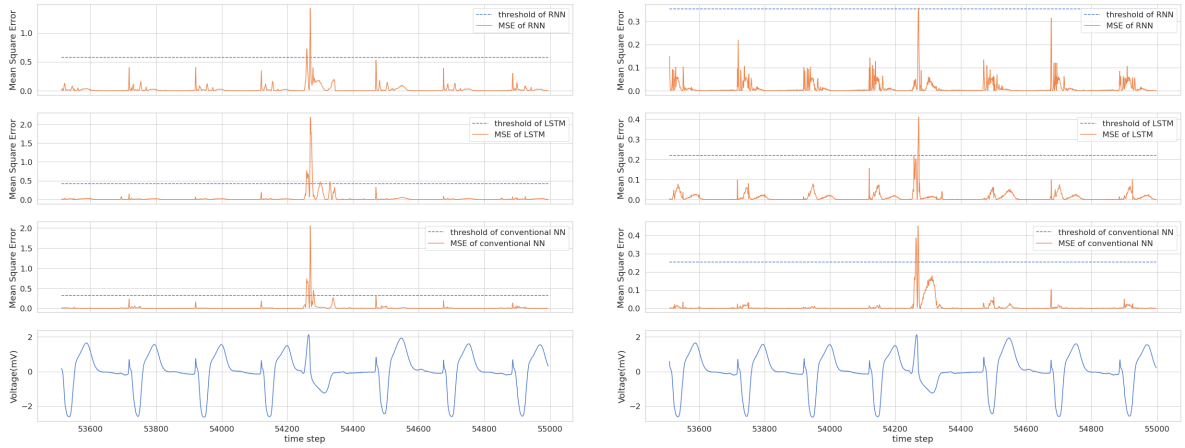


Figure 3.29: anomaly detection on anomaly dataset, $q = 4$ on the left and $q = 1$ on the right.

In the case of $q = 4$, it is obvious to see that RNN, LSTM and conventional NN all detect the only anomaly in ECG data, but the MSE of RNN while encountering the anomaly is not as distinguishable as LSTM's and conventional NN's. What's more, compared to conventional NN, LSTM possesses less MSE glitch on nominal data potentially reduce fault alarms as a consequence. Apart from that, LSTM is able to detect all abnormal waveform in contrast with conventional NN since MSE value in abnormal interval after the peak is still relatively high.

On the contrary, the performance of predictors with $q = 1$ is not as good as those with $q = 4$: MSE in the normal region also has some fluctuation, and MSE value is not as remarkable as it is in $q = 4$. This is reasonable because with the more future values that network wants to predict, the larger uncertainty we have to face which results in larger MSE in the end. That means, although a network with $q = 4$ will have larger MSE than that with $q = 1$ when dealing with nominal data as is observed in figure 3.22, however, when it confronts anomalous data, it also raise bigger MSE compared to that with $q = 1$, and this MSE now is much larger since MSE is quadratic to error, uncertainty, said differently, and as a result, MSE of anomaly for $q = 4$ is much larger than that for $q = 1$.

Another interesting observation is that all those networks are sensitive to spikes or abrupt changes of the waveform in a sense that their MSE is large every time when a spike appears in the waveform, examples are represented in the figure 3.30.

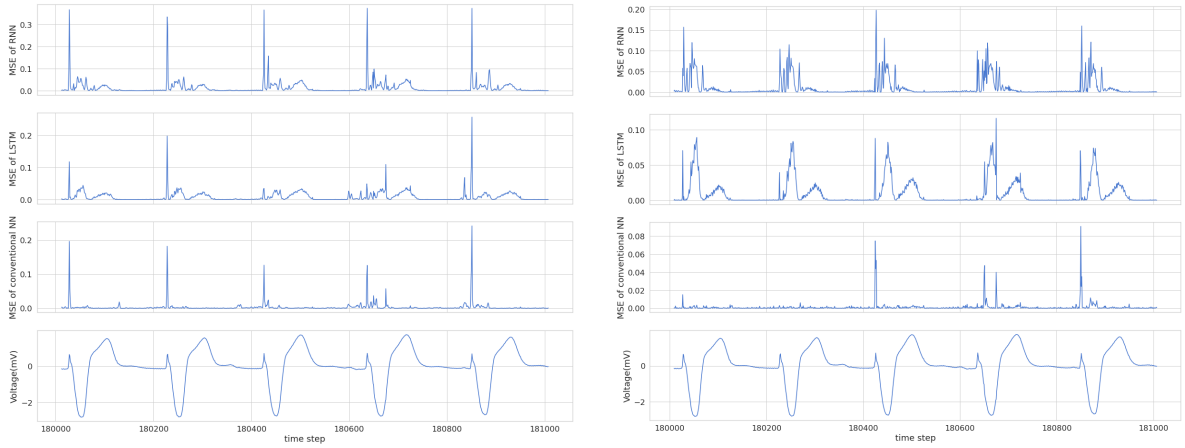


Figure 3.30: model is sensitive to spikes, $q = 4$ for left and $q = 1$ for right

In conclusion,

1. the performance of LSTM and conventional NN are better than RNN.
2. prediction on nominal data with conventional NN is a little bit superior than the one with LSTM in general as its prediction error is smaller.
3. with regard to anomaly detection on this ECG signal, LSTM is a better choice since it produces less glitches in nominal region compared with conventional NN.

Moreover, since we will investigate those networks with large input length in large-scale dataset, we want to evaluate performance to see if our models still works well when input length is large by scaling up parameters, and it yields the following results, which seem good as well.

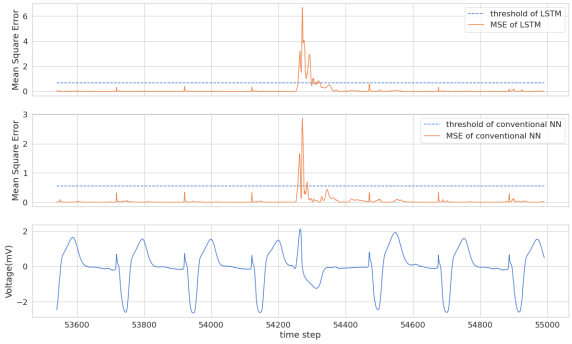


Figure 3.31: Anomaly detection via LSTM and conventional NN with input length $p = 32$, output length $q = 8$, and 256 neurons.

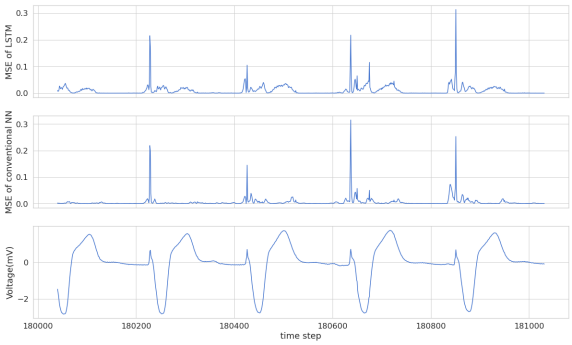


Figure 3.32: To show that networks on the left are sensitive to spikes in waveform.

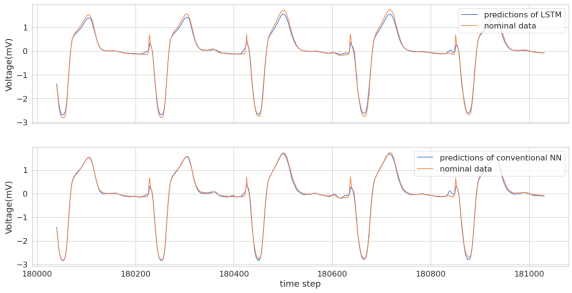


Figure 3.33: Prediction on test dataset using LSTM and conventional NN with input length $p = 32$, output length $q = 8$, and 256 neurons.

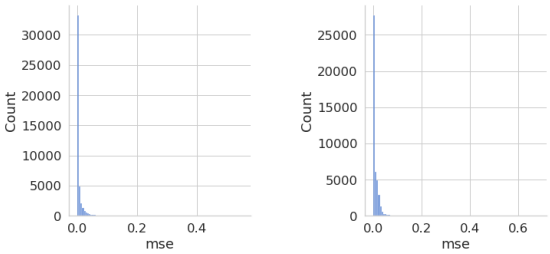


Figure 3.34: MSE distribution of those two networks on test dataset.

Chapter 4

Anomaly detection on satellite data

With the increasing popularity of satellite applications, a variety of satellites have been launched to orbit in aerospace for different purposes such as navigation, telemetry, even military usages. A typical satellite usually carries out many tasks using limited hardware resources embedded in it, including but not restrict to, attitude control, communication and power management, as a consequence, most advanced post processing that requires high computational resources have to done in the ground control station based on the massive data transmitted from in-orbit satellites. Nevertheless, we would also like to endow satellites with the ability of online anomaly detection in order to prevent system from catastrophic failure or crash, and we focus on the feasibility of this idea in this thesis first. Regrettably, satellite data is usually unavailable since they involves some legal privacy albeit they are indispensable for the study on anomaly detection. Fortunately, we get the telemetry data that can support us to explore anomaly detection which allows us to carry out this thesis project.

The telemetry data we are working on is a collection n_s signals for one year with T_s [s] sampling period. Said differently, each sample at any time step is composed of n_s quantities including: n_1 analogy quantities plus n_2 digital quantities. Once the dataset is at hand, we first standardize it by removing mean value μ and dividing by standard deviation σ,

$$x = \frac{(z - \mu)}{\sigma}$$

where μ and σ are estimated according

$$\mu = \frac{1}{N}\sum_{i=1}^{N} z_i \qquad , \quad N\text{— amount of samples}$$

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(z_i - \mu)^2$$

Temporally, we take 3 quantities into consideration forming a DataFrame, signal_1, signal_2, and signal_3. Whereas signal_1 and signal_2 are used as indicators of anomalies existing in signal_3: we define some signal_3 samples as anomalies if signal_1 equals to α or signal_2 is β at the same time instance. It is also worth notice that some anomalies in signal_3 do not meet the definition, but they only appear as adjacent neighbors of the anomalies we defined above. Once anomalies are determined, we then divide the DataFrame into two parts: first ten months

data is used in training stage among 80% of it aiming for training and 20% for validation, whilst last two months data is applied for performance evaluation in test stage. During training stage, we explicitly exclude anomalies by removing them from signal_3 and feed NNs only with the purely nominal signal_3 after they have been reshaped to fit to input and output shapes of NNs, by doing so, NNs can therefore learn the normal behaviors and perform prediction on nominal data very precisely, while on the contrary, we will keep anomalies in signal_3 and fill a few 'Nan' by linear interpolation at test stage before reshaping in order to avoid discontinuity on time series.

From now on, we confine ourselves on shallow neural networks, meaning that each NN has only one hidden layer either conventional NN or LSTM other than input layer and output layer, see the figures 4.1 4.2. NNs are trained on nominal signal_3 to predict future values given the input sequence, once the training stage is completed, they should have learnt only the normal behaviors underlying signal_3 and whenever they encounter anomalies in real scenario, a significant prediction error exceeding the threshold will be raised consequently.

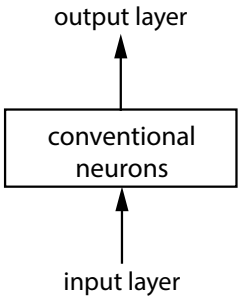


Figure 4.1: Architecture of shallow conventional NN.

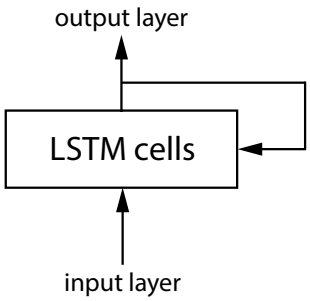


Figure 4.2: Architecture of shallow LSTM.

Without loss of generality, we consider each sample in the input sequence an n-dimensional vector, where n equals to 1 as we are using only one quantity, signal_3, to train NNs currently. Thus, the input sequence fed to NNs at each time instance t is a sequence of p samples:

$$\text{Input} \triangleq \left\{ x^{(t-p+1)}, x^{(t-p+2)}, \cdots, x^{(t-1)}, x^{(t)} \right\} \quad , \quad x^{(t)} \in \mathbb{R}$$

and the prediction of next q values after the input sequence constitutes the output sequence

$$\text{Output} \triangleq \left\{ \hat{x}^{(t+1)}, \hat{x}^{(t+2)}, \cdots, \hat{x}^{(t+q-1)}, \hat{x}^{(t+q)} \right\} \quad , \quad \hat{x}^{(t)} \in \mathbb{R}$$

A graphic representation of the network workflow is depicted

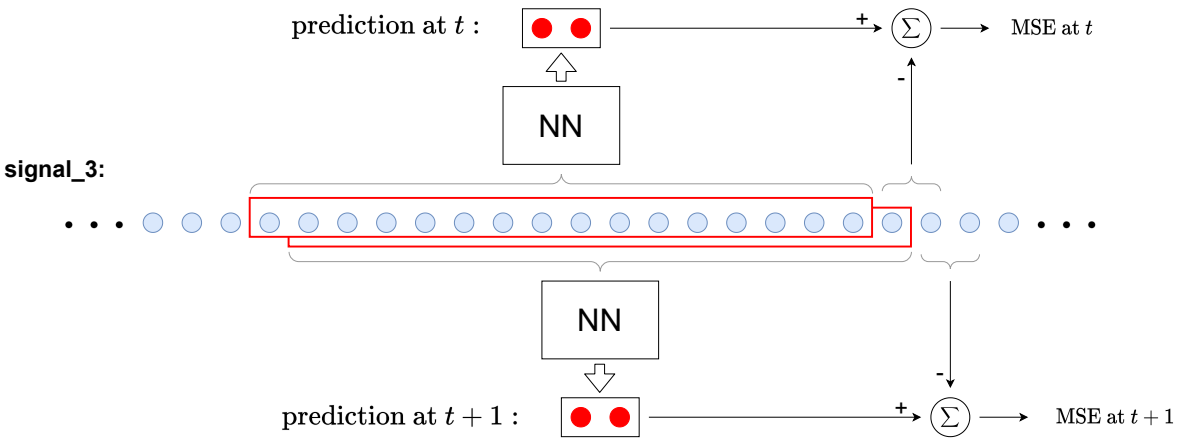


Figure 4.3: Workflow of NNs.

4.1 Workflow

4.1.1 Custom callbacks

It has been observed that when input length p and number of neurons for LSTM are too large, $p = 32$ with $q = 4$ ($n = 1$) and 512 neurons say, it is difficult and infeasible to train LSTM in a sense that there is always an abrupt jump in loss at some point, see the figure 4.4, because the amount of parameters N_{para} for LSTM is quadratic proportional to the number of neurons (i.e., LSTM states) M [21]. As a result, the training process of LSTM will be overwhelmed by numerous parameters, so we have to tradeoff between the number of neurons M indicating generalization capability and input length p.

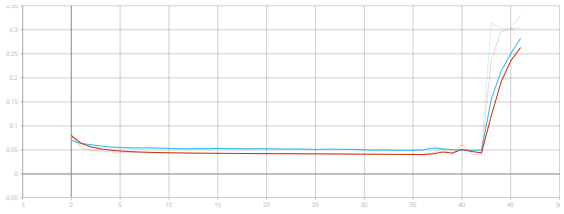


Figure 4.4: Training process of LSTM with $p = 32$ ($n = 1$), $q = 4$, 512 neurons and learning rate $1e-3$, as we can see, there is an abrupt jmup happens without custom callbacks.

$$\begin{aligned} N_{\text{para}} &= 4 \times (M(n+M)+M) \\ &\approx M^2 \end{aligned}$$

We try to fix this issue by reducing learning rate, and the results are represented: the reduction of learning rate does not helps us out, it only postpone the occurrence of jump instead of eliminate it, which we can see in the figures 4.5 4.6. In order to cope with it permanently, we propose a custom callbacks, which will be discussed after we make some clarification because one may easily think of the abrupt jump happens in loss as the exploding gradients, but it is not true indeed. To well explain it, let us first dive into the exploding gradient.

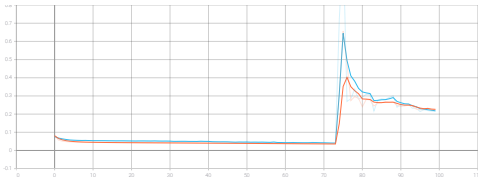


Figure 4.5: Training process of LSTM with $p = 32$ ($n = 1$), $q = 4$, 512 neurons, and learning rate $5e - 4$ without custom callbacks. reduction of learning rate puts off the occurrence of jump, but not prevent it from appearing.

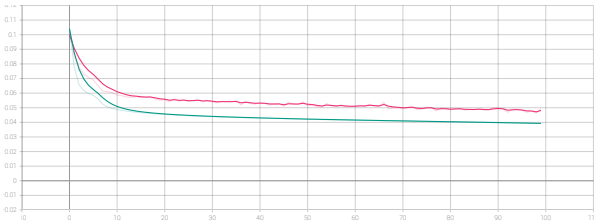


Figure 4.6: Training process of LSTM with $p = 32$ ($n = 1$), $q = 4$, 512 neurons, and learning rate $1e - 4$ without custom callbacks, we do not see the jump at this learning rate as the training process converges at very low speed and it has been stopped before the jump arises.

exploding gradient

From geometrical viewpoint, exploding gradient is the phenomenon that the gradient becomes too large at some point such that the updated weights at next step is far away from the current one as if we are casting it, this prevents network from converging to optimal [13] [14]. Think of the iterative equation of gradient descent we provide before,

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta \nabla_{\vec{w}}^{(k)} L \qquad , \quad \nabla_{\vec{w}}^{(k)} L \triangleq \nabla_{\vec{w}} L \left(\vec{w}^{(k)}, \vec{b} \right) \text{— loss function}$$

and consider the case where weights is around the cliff of loss function, see the figure 4.7.

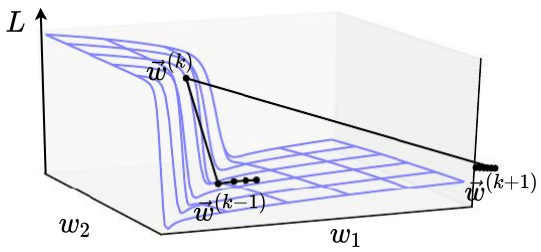


Figure 4.7: Bias $\vec{b}$ is neglected in the figure for the sake of simplicity, adapted from [13]

We apply the equation of gradient descent at some steps:

1. at step $k - 1$, the weights lies at the foot of a steep cliff and its gradient is small.
2. at step k, the weights $\vec{w}$ jump to the top of cliff which is exactly what happened to our case, and the gradient at this step, $\nabla_{\vec{w}}^{(k)} L$ turns to be very large.
3. at step $k + 1$, gradient exploding comes to play, the new updated weights $\vec{w}^{(k+1)}$ shall be far away from the previous weights $\vec{w}^{(k)}$ as if it is catapulted far from $\vec{w}^{(k)}$, but we do not have this problem.

Gradient clipping is an alternative way proposed to deal with exploding gradient [30], which simply clips the gradient when it exceeds pre-determined threshold $g_{\text{th}} \in \mathbb{R}$ as is illustrated in the figure 4.8.

Algorithm 2 gradient clipping

if $\left\|\nabla_{\vec{w}}^{(k)} L\right\| > g_{\text{th}}$ **then**

set gradient $= \frac{\nabla_{\vec{w}}^{(k)} L}{\left\|\nabla_{\vec{w}}^{(k)} L\right\|} \cdot g_{\text{th}}$

end if

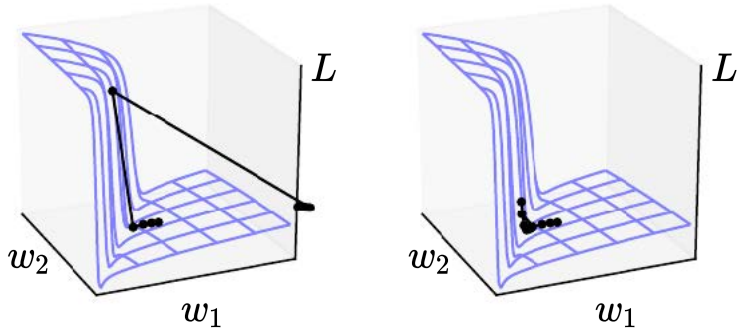


Figure 4.8: On the left, exploding gradient happens in the absence gradient clipping, and can be prevented with clipping shown on the right, adapted from [14].

This algorithm is well-defined since now the norm of new updated gradient reads

$$\left\| \frac{\nabla_{\vec{w}}^{(k)} L}{\left\|\nabla_{\vec{w}}^{(k)} L\right\|} \cdot g_{\text{th}} \right\| = \frac{\left\|\nabla_{\vec{w}}^{(k)} L\right\|}{\left\|\nabla_{\vec{w}}^{(k)} L\right\|} \cdot g_{\text{th}} = g_{\text{th}} < \left\|\nabla_{\vec{w}}^{(k)} L\right\|$$

On the other hand, considering that gradient at each step only gives us the steep descent direction of loss functions whilst its value is not what we concern, thus, gradient exploding can also be avoided (well, theoretically) if we apply normalized gradient descent algorithm shown in Chapter 2

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta \frac{\nabla_{\vec{w}}^{(k)} L}{\left\|\nabla_{\vec{w}}^{(k)} L\right\|}$$

Apart from it, Adam optimizer is also robust to this issue as it is updated by taking both 1st and 2nd moment into account:

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta \frac{r}{\sqrt{m} + \delta}$$

r— 1st moment

m— 2nd moment

which can be considered "normalization" in an extended sense. Nevertheless, we should keep in mind that none of three approaches, neither gradient clipping nor normalized gradient descent

algorithm or Adam optimizer can eliminate the abrupt jump of loss. As long as we are around steep cliff region, it is still very likely to happen and seems inevitable. To permanently ease this issue, we define our custom callbacks by adding two lines to the source code of *ReduceLROnPlateau()* allowing it to load the latest optimal weights if there is no remarkable improvement after a certain patience number of epochs in addition to reducing learning rate by the factor, its effectiveness is represented in the figure 4.9.

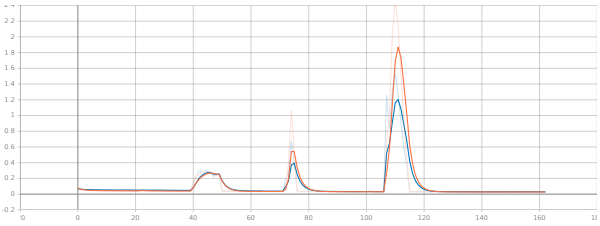


Figure 4.9: Training of LSTM with $p = 32$, $q = 4$ ($n = 1$) and 512 neurons where we adopt custom callbacks. Once there is no improved weights discovered after a patience number of epochs, the custom callbacks loads the latest optimal weights automatically and continues training.

4.1.2 Unbalancing issue

On the other hand, since we are feeding NNs with 1-dimensional signal_3 data, which implies the fact that if we unroll LSTM across time, only one value in the input sequence is available for LSTM to update the weights at each time step, and LSTM concatenates it with M (e.g., $M = 512$) states to form a real input vector [21]. In that case, our LSTM is extremely unbalancing as it uses very few values to update massive states during training process (say, the ratio of $\frac{M}{n} = 512$ is too large). Apart from that, the number of parameters is indeed quadratic proportion to M as is mentioned before, it means M dominates difficulty of training an LSTM compared to influence from dimensionality n. For this reason, we enhance the dimensionality of sample fed to LSTM by reshaping input sequence while lessening number of neurons M at the same time,

$$\text{Input} \triangleq \underbrace{\left\{ \begin{bmatrix} x^{(t-p-n+2)} \\ x^{(t-p-n+3)} \\ \vdots \\ x^{(t-p)} \\ x^{(t-p+1)} \end{bmatrix}, \begin{bmatrix} x^{(t-p-n+3)} \\ x^{(t-p-n+4)} \\ \vdots \\ x^{(t-p+1)} \\ x^{(t-p+2)} \end{bmatrix}, \cdots, \begin{bmatrix} x^{(t-n)} \\ x^{(t-n+1)} \\ \vdots \\ x^{(t-2)} \\ x^{(t-1)} \end{bmatrix}, \begin{bmatrix} x^{(t-n+1)} \\ x^{(t-n+2)} \\ \vdots \\ x^{(t-1)} \\ x^{(t)} \end{bmatrix} \right\}}_{p \text{ vectors}}$$

then we feed each n-dimensional vector to LSTM at corresponding time step as is depicted beneath:

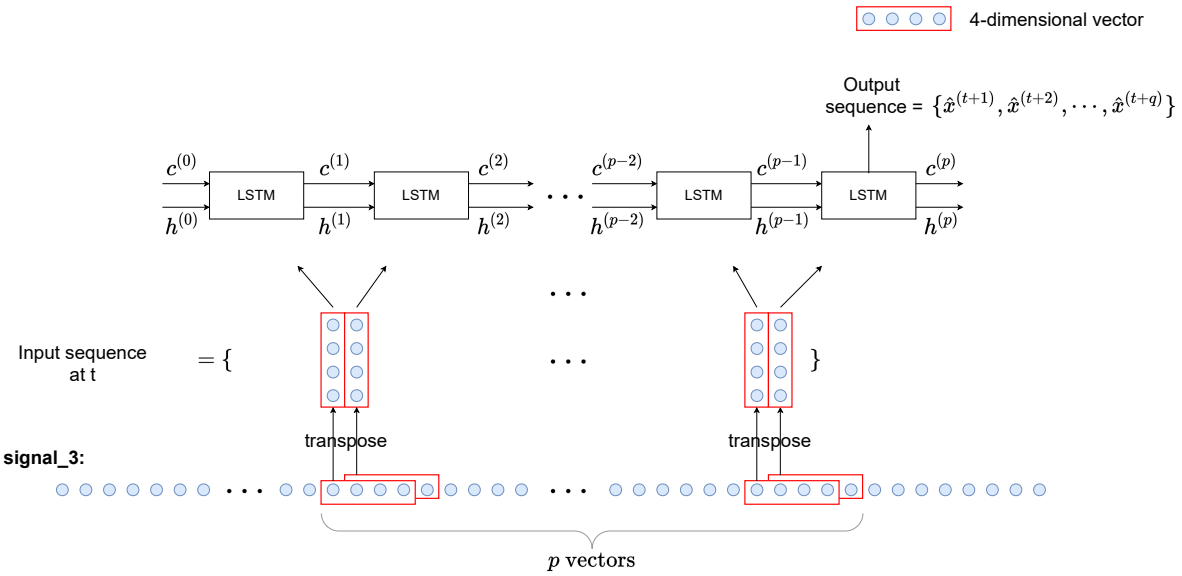


Figure 4.10: Enlarge dimensionality n by reshaping input sequence from signal_3.

4.2 Experiments

In the first experiment, we set input length $p = 16$, output length $q = 2$, and 128 neurons for all NNs, and we adjust n for LSTMs from 1 up to 32 in order to see if there is any help for improving performance of LSTM when we loosen the unbalancing effect, what's more, we also compare them with conventional NN to explore the performance among different type of NNs, the results are provided in section 5.1.

Furthermore, we reshape input sequence without maximum overlap as follows,

$$\text{Input} \triangleq \underbrace{\left\{ \begin{bmatrix} x^{(t-np+1)} \\ \vdots \\ x^{(t-n(p-1)-1)} \\ x^{(t-n(p-1))} \end{bmatrix}, \begin{bmatrix} x^{(t-n(p-1)+1)} \\ \vdots \\ x^{(t-n(p-2)-1)} \\ x^{(t-n(p-2))} \end{bmatrix}, \cdots, \begin{bmatrix} x^{(t-2n+1)} \\ \vdots \\ x^{(t-n-1)} \\ x^{(t-n)} \end{bmatrix}, \begin{bmatrix} x^{(t-n+1)} \\ \vdots \\ x^{(t-1)} \\ x^{(t)} \end{bmatrix} \right\}}_{p \text{ vectors}}$$

and explore the difference between non-overlap and overlapped strategies for LSTMs in the second experiment, with input length $p = 16$, output length $q = 2$ fixed for all LSTM under test. The results are given in section 5.2.

Since one typical period of signal_3 contains approximately 4096 samples, we therefore enlarge input length p up to 128 with maximum overlap and n only differing from 1 to 4 for the sake of simplicity to see if there is any improvement after we enlarge input size. But unfortunately, we are not able to carry out further assessment on test dataset because we lack sufficient memory space to generate test datasets for those NNs.

In the third experiment, we reshape input sequence fitting to $p = 128$ and $n = 32$ but without maximum overlap, the real input length therefore turns out to be $p \times n = 4096$ which implies that at least one period of signal_3 will be taken into consideration, besides, the difficulty of training LSTM is overcame as well owing to the fact that the input length p is 128 rather than 4096, unbalancing can also be avoided since n is comparable to the number of neurons, 128,

now. The results are in section 5.3.

4.2.1 Award function

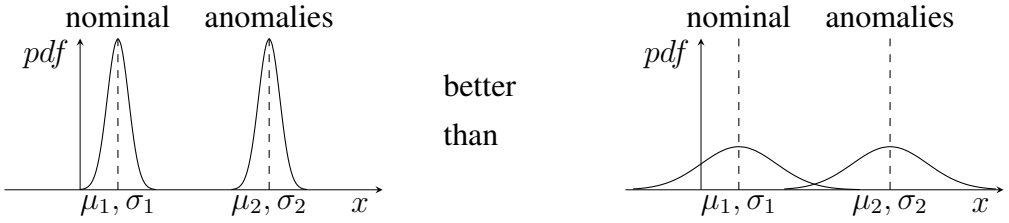
Balanced accuracy, BA for short, identifies each MSE as binary category, either positive or negative labels, despite of their value, hence, it looks more like a qualitative performance metric from this viewpoint. For this reason, we attempt to invent a quantitative-like metric for performance assessment, that is, award function, which takes not only the binary category of MSE but also MSE values into account. Without loss of generality, we define award function a class of functions $f : X^N \times Y^N \to \mathbb{R}$, satisfying:

1. It gives positive score when NN predicts correctly, in other words, it awards the NN. Additionally, the higher difference between MSE and threshold, the higher award will be (i.e., in the case of correct prediction, f is monotone increasing w.r.t. difference between MSE and threshold.).

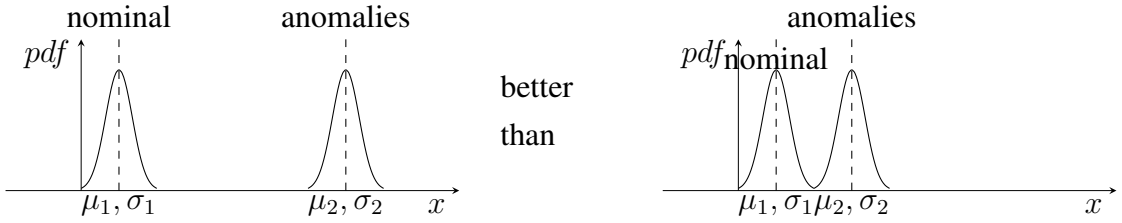
2. On the contrary, it gives negative score when NN predicts incorrectly, namely, it penalizes NN. And the higher difference between MSE and threshold $\Rightarrow$ award $\downarrow$ (i.e., in the case of incorrect prediction, f is monotone decreasing w.r.t. difference between MSE and threshold.).

3. Bonus: it may also possess the following desired properties which describes separability of MSE distribution:

   (a) $\sigma \downarrow \Rightarrow$ award $\uparrow$, a graphic interpretation is represented below.



   (b) $\Delta\mu = \mu_2 - \mu_1 \uparrow \Rightarrow$ award $\uparrow$.



The first award function we came up with is

$$award = \frac{1}{N} \sum_{i=1}^{N} y_i \left(x_i - \gamma \right)$$

N— number of samples, $N = N_+ + N_-$

N_+— amount of positive samples

N_-— amount of negative samples

y_i— label at i, $y_i = \begin{cases} +a, & \text{if anomaly at time step } i \\ -1, & \text{if nominal at time step } i \end{cases}$

x_i— MSE at i

γ— threshold

Properties:

- if $x_i < \gamma$ and $y_i = -1 \Rightarrow$ prediction is correct, positive award and monotonously increase with the difference.
- if $x_i > \gamma$ and $y_i = +a \Rightarrow$ prediction is correct, positive award and monotonously increase with the difference.
- if $x_i < \gamma$ and $y_i = +a \Rightarrow$ prediction is incorrect, negative award and monotonously decrease with the difference.
- if $x_i > \gamma$ and $y_i = -1 \Rightarrow$ prediction is incorrect, negative award and monotonously decrease with the difference.
- if $a = \frac{N_-}{N_+}$, $\Delta\mu$ is preserved:

$$
\begin{aligned}
\text{award} &= \frac{1}{N}\sum_{i=1}^{N} y_i\,(x_i - \gamma) \\
&= \frac{N_+}{N_+ + N_-}\cdot\frac{1}{N_+}\cdot\sum_{i=1}^{N_+} a\,(x_i - \gamma) - \frac{N_-}{N_+ + N_-}\cdot\frac{1}{N_-}\cdot\sum_{i=1}^{N_-}(x_i - \gamma) \\
&= \frac{aN_+}{N_+ + N_-}\Bigg(\underbrace{\frac{1}{N_+}\sum_{i=1}^{N_+} x_i}_{\mu_2} - \gamma\Bigg) - \frac{N_-}{N_+ + N_-}\Bigg(\underbrace{\frac{1}{N_-}\sum_{i=1}^{N_-} x_i}_{\mu_1} - \gamma\Bigg) \\
&= \frac{aN_+}{N_+ + N_-}(\mu_2 - \gamma) - \frac{N_-}{N_+ + N_-}(\mu_1 - \gamma) \\
&= \frac{a}{a+1}\Delta\mu
\end{aligned}
$$

but regardless of choice of γ.

- if $a = +1$, it reads,

$$\begin{aligned}\text{award} &= \frac{1}{N}\sum_{i=1}^{N} y_i\,(x_i - \gamma) \\ &= \frac{1}{N}\left[\sum_{i=1}^{N_+}(x_i - \gamma) - \sum_{i=1}^{N_-}(x_i - \gamma)\right] \\ &\downarrow N_- \gg N_+ \\ &\approx \frac{1}{N_-}\sum_{i=1}^{N_-}(\gamma - x_i)\end{aligned}$$

  that means, nominal data will dominate the overall award in a sense that it will contribute to the most part of the award, and shall cancel the contribution from anomalies, which is unwanted behavior.

Unfortunately, we can not read any information about σ_1 or σ_2 from this award function. Another defect of this award function is that it only focus on the difference between MSE and threshold, i.e., $(x_i - \gamma)$, but sometimes, the ratio of MSE to threshold sometimes is more convincible as performance metric rather than the difference and it inspires us to propose an alternative award function

$$\textit{award} = \frac{1}{N}\sum_{i=1}^{N} y_i \ln\frac{x_i}{\gamma}$$

N— number of samples, $N = N_+ + N_-$

N_+— amount of positive samples

N_-— amount of negative samples

y_i— label at i, $y_i = \begin{cases} +a, & \text{if anomaly at time step } i \\ -1, & \text{if nominal at time step } i \end{cases}$

x_i— MSE at i

γ— threshold

Properties:

- if $x_i < \gamma$ and $y_i = -1 \Rightarrow$ prediction is correct, positive award and monotonously increase with the ratio.
- if $x_i > \gamma$ and $y_i = +a \Rightarrow$ prediction is correct, positive award and monotonously increase with the ratio.
- if $x_i < \gamma$ and $y_i = +a \Rightarrow$ prediction is incorrect, negative award and monotonously decrease with the ratio.
- if $x_i > \gamma$ and $y_i = -1 \Rightarrow$ prediction is incorrect, negative award and monotonously decrease with the ratio.

- if $a = 1$,

$$\begin{aligned}
\text{award} &= \frac{1}{N}\sum_{i=1}^{N} y_i \ln\frac{x_i}{\gamma} \\
&= \frac{1}{N}\left(\sum^{N_+} \ln\frac{x_i}{\gamma} + \sum^{N_-} \ln\frac{\gamma}{x_i}\right) \\
&= \frac{1}{N}\left(\sum^{N_+} \ln x_i - \sum^{N_-} \ln x_i\right) + \frac{1}{N}\big(N_- - N_+\big)\ln\gamma \\
&\downarrow N_- \gg N_+ \\
&\approx -\frac{1}{N_-}\sum^{N_-} \ln x_i + \ln\gamma
\end{aligned}$$

Again, nominal data will dominate the overall award by cancelling the contribution from anomalies, which is unexpected. Moreover,

  - σ_2 is also preserved in a sense that: $\sigma_2 \downarrow \Rightarrow$ award $\uparrow$.

    Proof. Since

$$\begin{aligned}
\frac{1}{N_+}\sum^{N_+} \ln x_i &= \ln\ \sqrt[N_+]{x_1 x_2 \cdots x_{N_+}} \\
&= \ln\left(\prod^{N_+} x_i\right)^{\frac{1}{N_+}} \\
&\leqslant \ln\left(\frac{1}{N_+}\sum^{N_+} x_i\right) \triangleq \ln\mu_2
\end{aligned}$$

    As a consequence,

$$\begin{aligned}
\text{award} &= \frac{1}{N}\left(\sum^{N_+} \ln x_i - \sum^{N_-} \ln x_i\right) + \frac{1}{N}\big(N_- - N_+\big)\ln\gamma \\
&\downarrow \text{if } N_- = 0 \\
&= \frac{1}{N_+}\sum^{N_+} \ln x_i - \ln\gamma \\
&\leqslant \ln\mu_2 - \ln\gamma
\end{aligned}$$

    Thus, $\sigma_2 \downarrow \ \Rightarrow$ all x_i tend to $\mu_2 \ \Rightarrow\ \frac{1}{N_+}\sum^{N_+} \ln x_i \to \ln\mu_2 \ \Rightarrow$ award $\uparrow$ and tends to the upper bound. □

  - However, σ_1 is not preserved because

Proof.

$$\begin{aligned}
\text{award} &= \frac{1}{N}\left(\sum^{N_+} \ln x_i - \sum^{N_-} \ln x_i\right) + \frac{1}{N}(N_- - N_+)\ln\gamma \\
&\downarrow \text{if } N_+ = 0 \\
&= \underbrace{-\frac{1}{N_-}\sum_{i=1}^{N_-} \ln x_i}_{\leqslant \ln \mu_1} + \ln\gamma \\
&\geqslant \ln\gamma - \ln\mu_1
\end{aligned}$$

therefore, $\sigma_1 \downarrow \quad \Rightarrow \quad -\frac{1}{N_-}\sum^{N_-} \ln x_i \to -\ln\mu_1 \quad \Rightarrow$ award $\downarrow$ and goes to lower bound. □

- if $a = \frac{N_-}{N_+}$ and all predictions are correct, we also have

$$\begin{aligned}
\text{award} &= \frac{1}{N}\sum_{i=1}^{N} y_i \ln\frac{x_i}{\gamma} \\
&= \frac{1}{N_+ + N_-}\left(\sum_{i=1}^{N_+} a\ln\frac{x_i}{\gamma} - \sum_{i=1}^{N_-}\ln\frac{x_i}{\gamma}\right) \\
&= \frac{aN_+}{N_+ + N_-}\left(\frac{1}{N_+}\sum_{i=1}^{N_+}\ln x_i - \ln\gamma\right) - \frac{N_-}{N_+ + N_-}\left(\frac{1}{N_-}\sum_{i=1}^{N_-}\ln x_i - \ln\gamma\right) \\
&= \frac{a}{a+1}\cdot\frac{1}{N_+}\left(\sum_{i=1}^{N_+}\ln x_i - \frac{1}{a}\sum_{i=1}^{N_-}\ln x_i\right)
\end{aligned}$$

  regardless of choice of γ.

- But we do not have any information about μ_2 or μ_1 from this award function.

Nevertheless, this award function could also be suitable to our cases if the nominal MSE distribution are almost same to every NN, which means the contribution from nominal data to overall award are approximately equal.

The last award function we propose is equivalent to the balanced accuracy

$$\textit{award} = \frac{1}{N}\sum_{i=1}^{N} |y_i|\chi\big[y_i(x_i - \gamma)\big]$$

y_i— label at i, $y_i = \begin{cases} +a, & \text{if anomaly at time step } i \\ -1, & \text{if nominal at time step } i \end{cases}$

χ— step function, $\chi(z) = \begin{cases} +1, & \text{if } z > 0 \\ 0, & \text{if } z \leqslant 0 \end{cases}$

x_i— MSE at i

γ— threshold

TP— true positive

TPR— true positive rate

TN— true negative

TNR— true negative rate

Since we invent award function as a metric to describe the goodness of performance, of course we would expect an NN reaches as much award as possible under this circumstance, in that case, the problem is transformed to be an optimization issue, and we prove that

$$\max_{\gamma\in\mathbb{R}} \text{award} \Leftrightarrow \max_{\gamma\in\mathbb{R}} \text{BA}$$

Proof.

$$\begin{aligned}
\text{award} &= \frac{1}{N}\sum_{i=1}^{N} |y_i|\chi\big[y_i(x_i-\gamma)\big] \\
&= \frac{1}{N}\left[\sum^{N_+} a\chi(x_i-\gamma) + \sum^{N_-}\chi(\gamma-x_i)\right] \\
&= \frac{1}{N_+ + N_-}\big(a\cdot\#\text{ of TP} + \#\text{ of TN}\big) \\
&\downarrow \text{if } a = \frac{N_-}{N_+} \\
&= \frac{aN_+}{N_+ + N_-}\cdot\underbrace{\frac{\#\text{ of TP}}{N_+}}_{\triangleq\text{TPR}} + \frac{N_-}{N_+ + N_-}\cdot\underbrace{\frac{\#\text{ of TN}}{N_-}}_{\triangleq\text{TNR}} \\
&= \frac{a}{a+1}\big(\text{TPR} + \text{TNR}\big) \\
&\downarrow a \gg 1 \\
&\approx \text{TPR} + \text{TNR}
\end{aligned}$$

$\therefore \quad \max_{\gamma\in\mathbb{R}} \text{award} \;\Leftrightarrow\; \max_{\gamma\in\mathbb{R}} \text{TPR+TNR}.$ □

In addition, if $a = 1$,

$$\text{award} = \frac{\text{\# of TP} + \text{\# of TN}}{N_+ + N_-} \triangleq \text{accuracy}$$

we will use award function 1 and 2 to evaluate performance of NNs in the first experiment in order to examine their effectiveness as a new performance metric, the results are provided in section 5.4.

Chapter 5

Results and discussion

We represent results of all experiments launched in the last chapter.

5.1 Results of the first experiment

In the first experiment, input length p, output length q as well as number of neurons M are fixed to 16, 2 and 128 respectively for all NNs, while dimensionality n changes from 1 up to 32 to see if any improvement can be obtained by alleviating unbalancing issue, moreover, conventional NNs are added as counter parts of LSTMs for fair comparisons. Once training is done, we test them on test dataset and plot MSE at each time step in figure 5.1 to acquire some intuitive understanding.

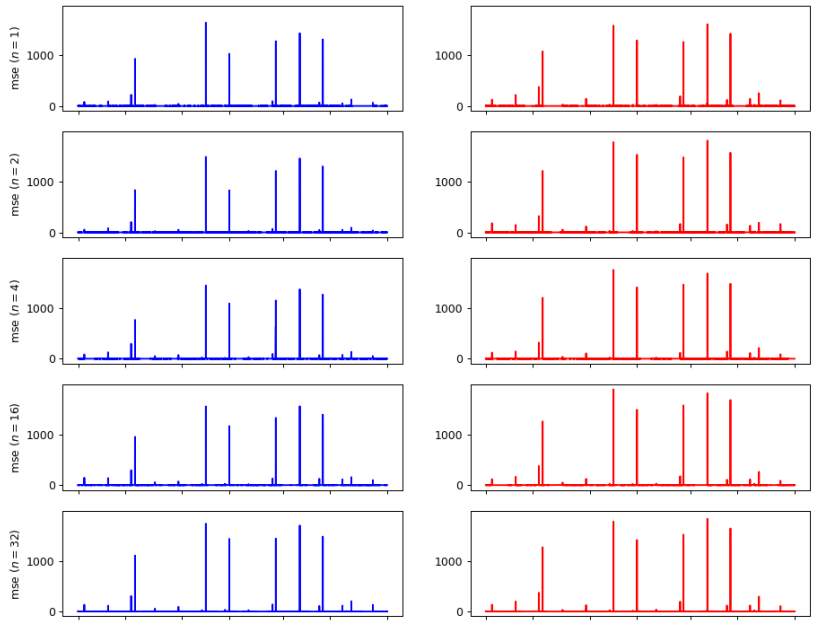


Figure 5.1: MSE at each step are assembled in this figure whereas all NNs are equipped with $p = 16$, $q = 2$, $M = 128$ and maximum overlap strategy. X-axis indicates time step and y-axis stands for corresponding MSE value, plots on the left column are conventional NNs with different dimensionality n while LSTMs are on the right.

Roughly speaking, both conventional NN and LSTM are able to detect anomalies aligned with the definition in test datasets, furthermore, MSE of anomalies provided by LSTM usually is higher in contrast to that of conventional NN, which is desired as it can easily exceeds the threshold and raise a sign for anomalous events. A detailed comparison is provided in 5.2.

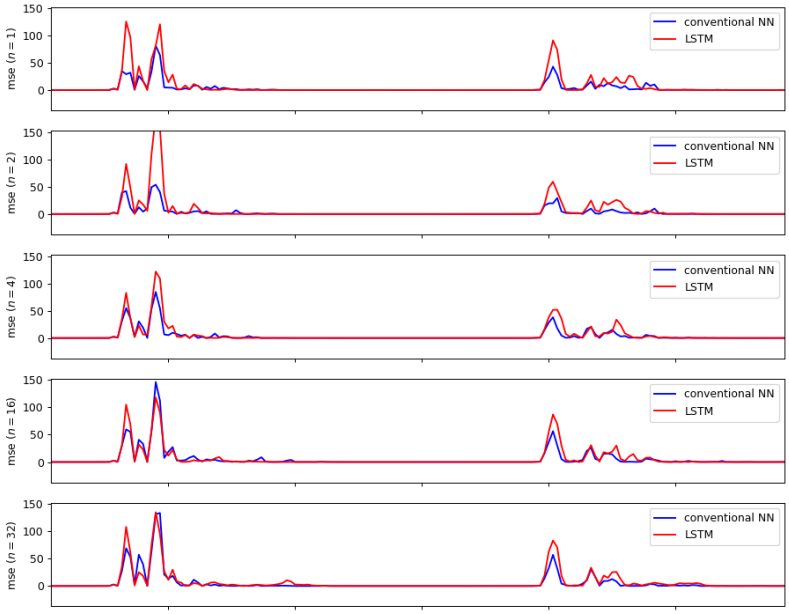


Figure 5.2: A glimpse of detailed comparison for conventional NN and LSTM, and it is clear to see MSE produced by LSTM is higher than by conventional NN.

Apart from that, they are also able to detect undefined anomalies whose value fluctuates fiercely but the corresponding signal_1 and signal_2 are normal, as is shown in 5.3.

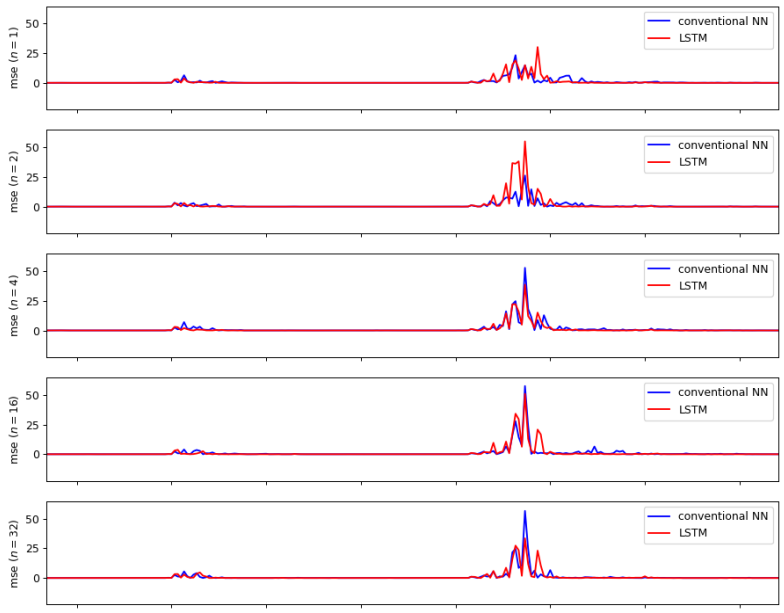


Figure 5.3: Both LSTM and conventional NN yield significant MSE for undefined anomalies, meaning that undefined anomalies can be detected as well, x-axis is time axis.

To quantitatively evaluate their performance, we then plot the histograms of MSE distribution on test dataset for each NN in figure 5.5, and we hope that MSE distribution of nominal data and that of anomalies are visually separable so that we can easily set a threshold to classify them into two categories: nominal data or anomaly. More mathematically, we would like to see $\Delta\mu$[1] as much large as possible while keeping both σ_1 and σ_2 [2] small at the same time, see an intuitive example figure 5.4.

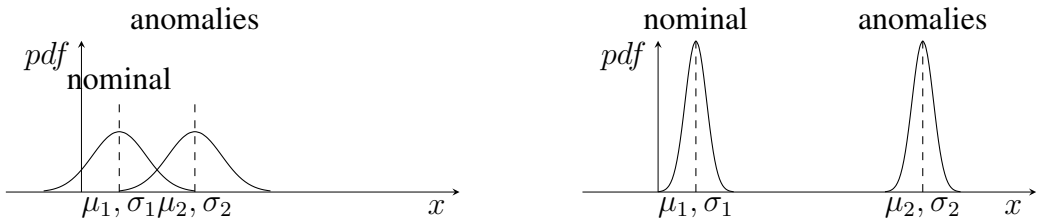


Figure 5.4: Apparently the classifier that produces probability density function (pdf) on the right is preferable compared to the left because we can easily find a proper threshold that partition anomalies and nominal.

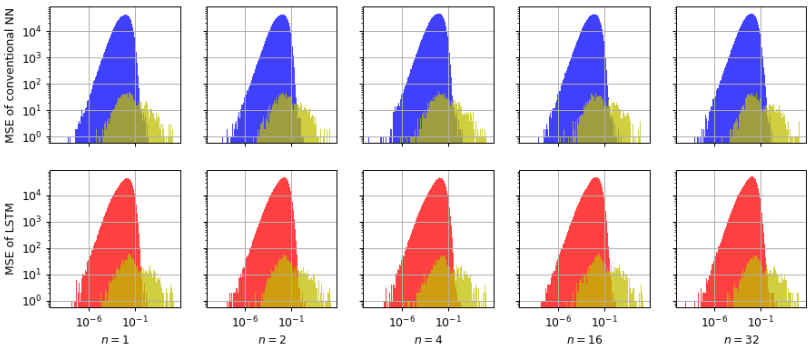


Figure 5.5: Histograms of MSE distribution on test dataset, whereas 'blue' indicates MSE distribution of nominal data given by conventional NNs, 'red' stands for LSTM nominal, and 'yellow' for anomalies.

Non-overlap area in MSE distribution for anomalies given by LSTM is bigger than that by conventional NN, The extensive overlap between nominal MSE distribution and anomaly MSE distribution also addresses the difficulty to find an appropriate threshold that can partition nominal data from anomalies. Nevertheless, we can still read some evidence that LSTM may outperforms conventional NN in a sense that the non-overlapped area of anomalies for LSTM in MSE distribution plots is slightly larger than that for conventional NN, which implies higher accuracy as it can recognize more anomalies in dataset. we can see it in the case of $n = 1$, $n = 2$ and $n = 4$ in the figure 5.5. μ and σ are also provided in the table 5.1 for the sake of completeness.

[1] $\Delta\mu = \mu_2 - \mu_1$, where μ_1 is mean of nominal MSE and μ_2 is mean of anomaly MSE.
[2] σ_1 is standard deviation of nominal MSE and σ_2 is standard deviation of anomaly MSE.

		$n=1$		$n=2$		$n=4$		$n=16$		$n=32$	
		nominal	anomaly	nominal	anomaly	nominal	anomaly	nominal	anomaly	nominal	anomaly
conventional NN	μ	0.01341	9.84148	0.01498	8.32260	0.01418	9.59606	0.01183	10.92414	0.01405	12.43549
	σ^2	0.000481	6790.186552	0.000514	5124.205154	0.000505	5959.668316	0.000381	7595.826844	0.000480	9963.150728
LSTM	μ	0.01856	12.38848	0.01963	13.21072	0.01942	12.23991	0.01601	13.87833	0.01454	14.10955
	σ^2	0.000763	8721.850122	0.000796	11049.528767	0.000788	9939.897480	0.000532	12171.552201	0.000496	11527.194517

Table 5.1: The statistics aggregate of μ and σ for each NN.

Actually, the overlapped region reflects a fact that anomalies is not well-defined in our case as there are some anomaly existing in test dataset but their waveform are very similar to nominal signal, and we show MSE for undefined anomalies in figure 5.6. We pick out 3 peaks of anomalies standing for large anomalies, medium and nominal-like anomalies respectively, they can also be observed according to MSE distribution in figure 5.7, with the peak positions in table 5.2. As a matter of fact, we do not have much prior knowledge about anomalies in real scenario before it arises, neither do we know the exact waveform of it, nor can we give an accurate definition for it. Of course, it can be expected to obtain considerable improvement for anomaly detection once we can define anomalies in a more accurate sense, for instance, just classify those anomalies that are similar to nominal data as nominal.

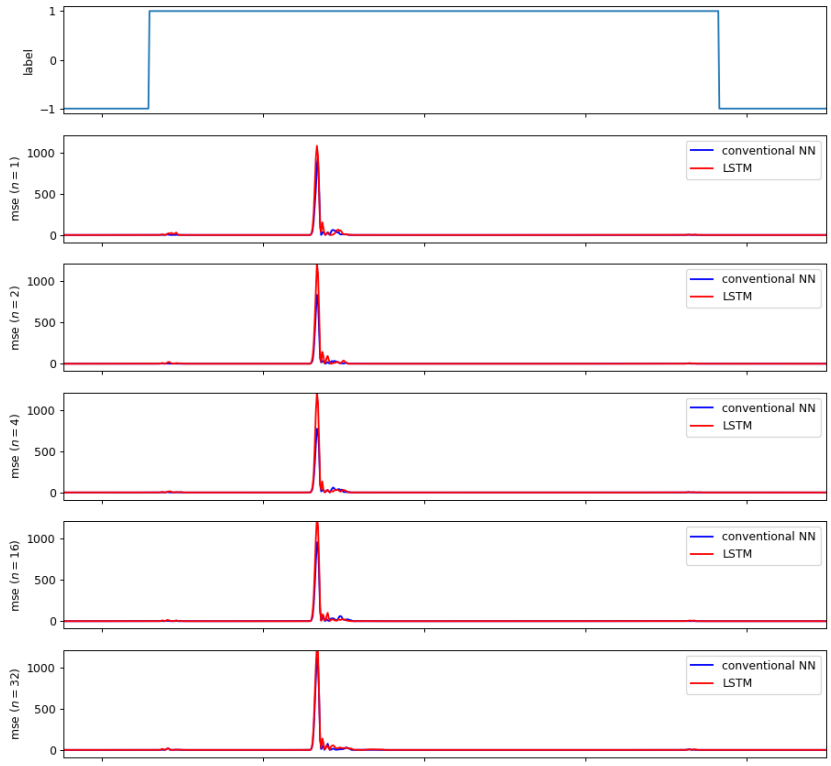


Figure 5.6: A glimpse of MSE on anomalies whereas x-axis is time axis, some MSE for anomalies are small and nominal-like, in fact, they are related to the definition of anomalies which is one of the main challenges for anomaly detection.

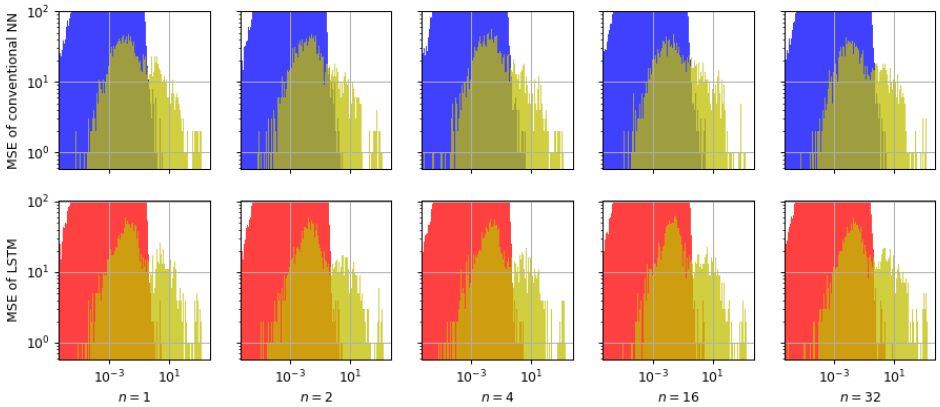


Figure 5.7: A detailed view of 3 peaks in the histograms of MSE distribution.

Network type	MSE peak position	$n = 1$	$n = 2$	$n = 4$	$n = 16$	$n = 32$
conventional NN	1st	0.0155667	0.0226003	0.0159584	0.00975566	0.00919072
	2nd	1.35853	1.27986	3.30763	2.02201	2.20032
	3rd	668.728	630.003	685.558	746.012	937.684
LSTM	1st	0.0155667	0.0195662	0.0328121	0.0231691	0.0291219
	2nd	2.79328	4.68427	4.41301	4.15746	3.39088
	3rd	772.429	970.888	1220.34	1149.67	1251.05

Table 5.2: Positions of peaks for each NN.

Ultimately, we adopt ROC curves in figure 5.8 as a figure of merit for performance assessment, and as we can see from AUC values, the performance of LSTM usually is better than conventional NN except for the LSTM with $n = 2$. What's more, LSTM with $n = 32$ outperforms the most among all the NNs in this experiment.

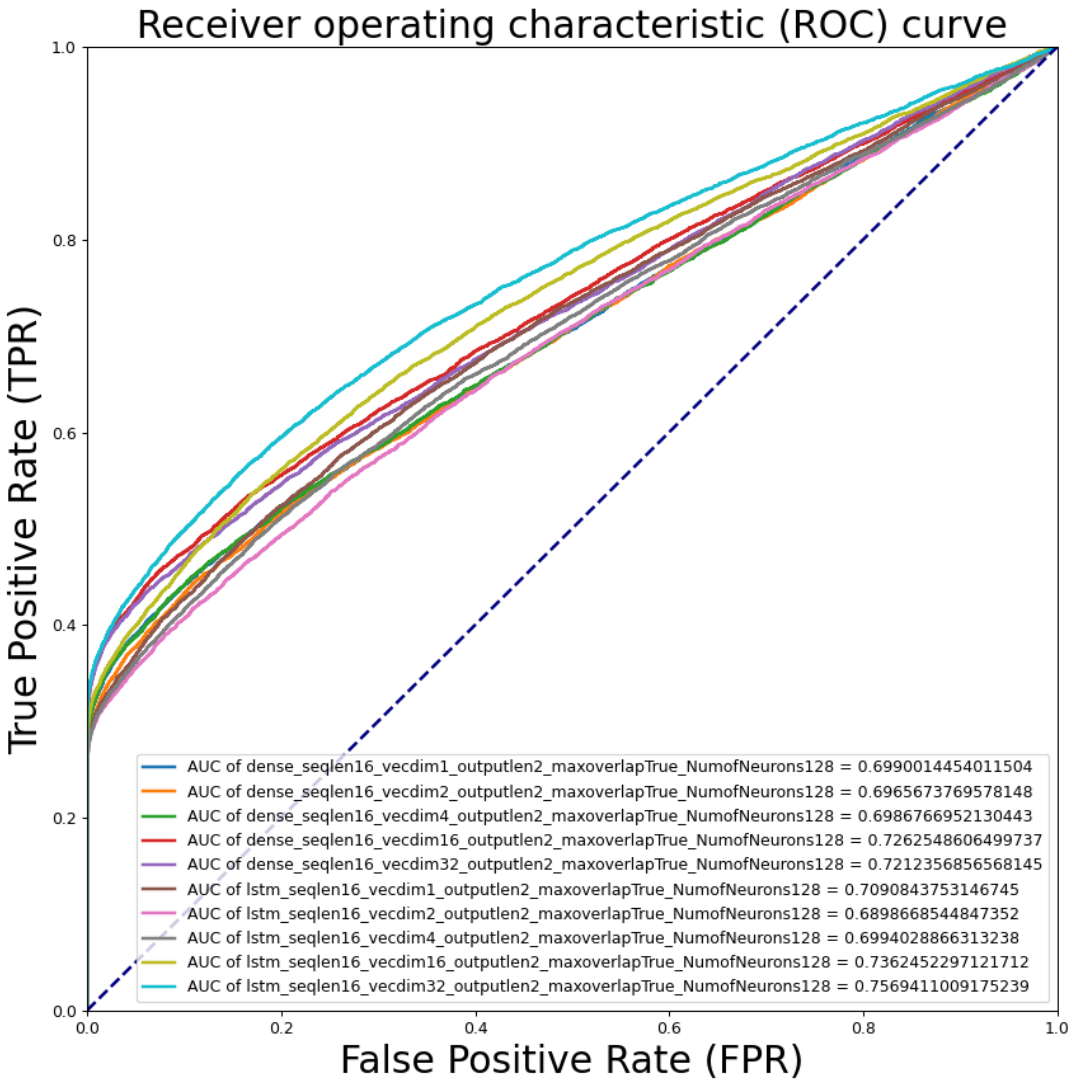


Figure 5.8: ROC curves: conventional NN versus LSTM with $p = 16$, $q = 2$, 128 neurons and maximum overlapped strategy applied to all, but n varying from 1, 2, 4, 16, 32.

5.2 Results of the second experiment

We investigate the difference of LSTM with or without maximum overlap, while dimensionality varies from 1 to 32, and fix input length $p = 16$, output length $q = 2$, number of neurons $M = 128$ for all NNs in this experiment. Likewise, we obtain the ROC curves in figure 5.9,

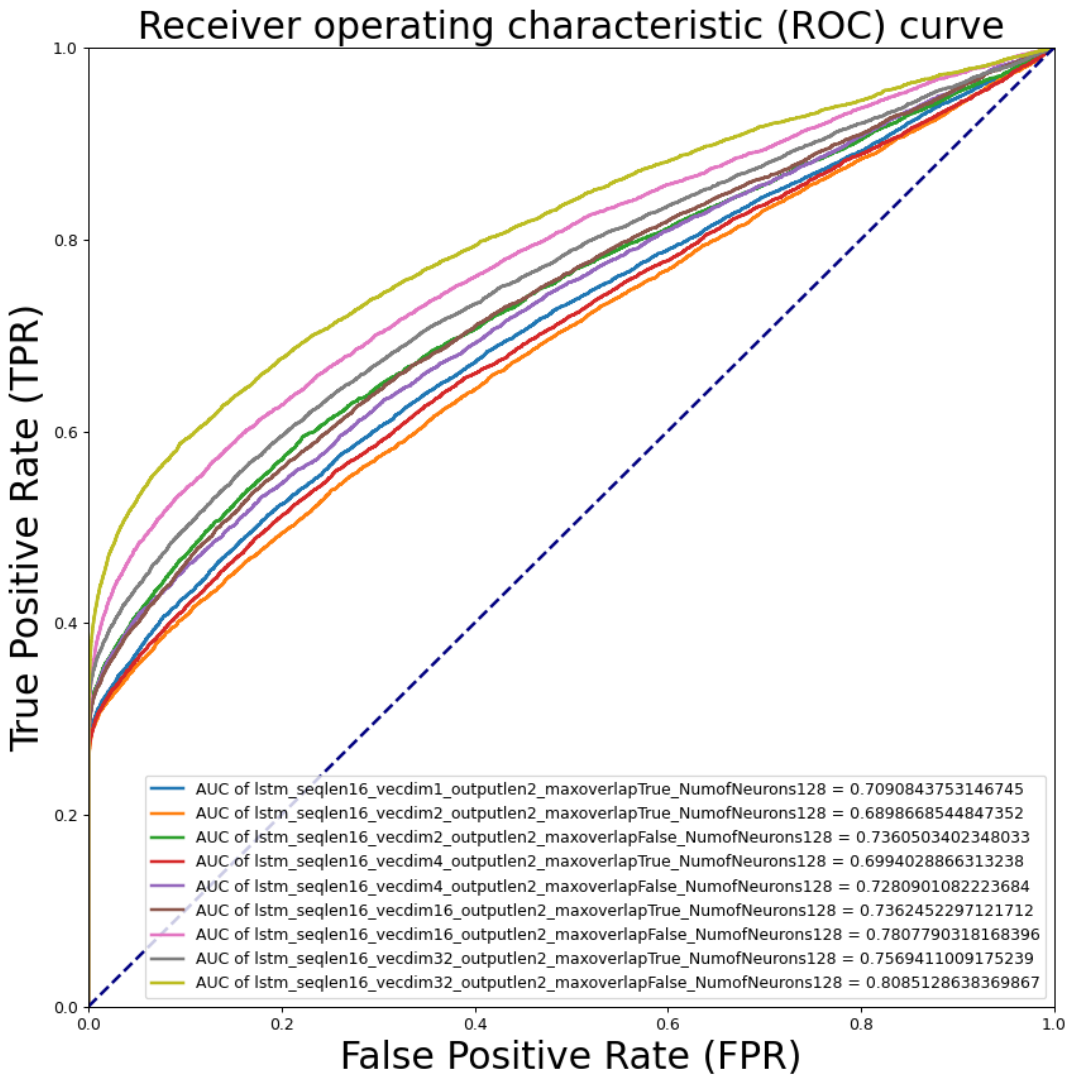


Figure 5.9: ROC curves: LSTMs with $p = 16$, $q = 2$, 128 neurons for all, but both maximum overlapped and non-overlapped strategies are adopted while n also varies from 1, 2, 4, 16, 32.

we can see that LSTMs without maximum overlap are all superior than those with maximum overlap with regard to their AUC scores. It may because LSTM takes longer input into account in relative to the one with maximum overlap, which actually provides more information to LSTM.

5.3 Results of the third experiment

The last ROC curves for LSTM and conventional NN without maximum overlap are also provided in figure 5.10,

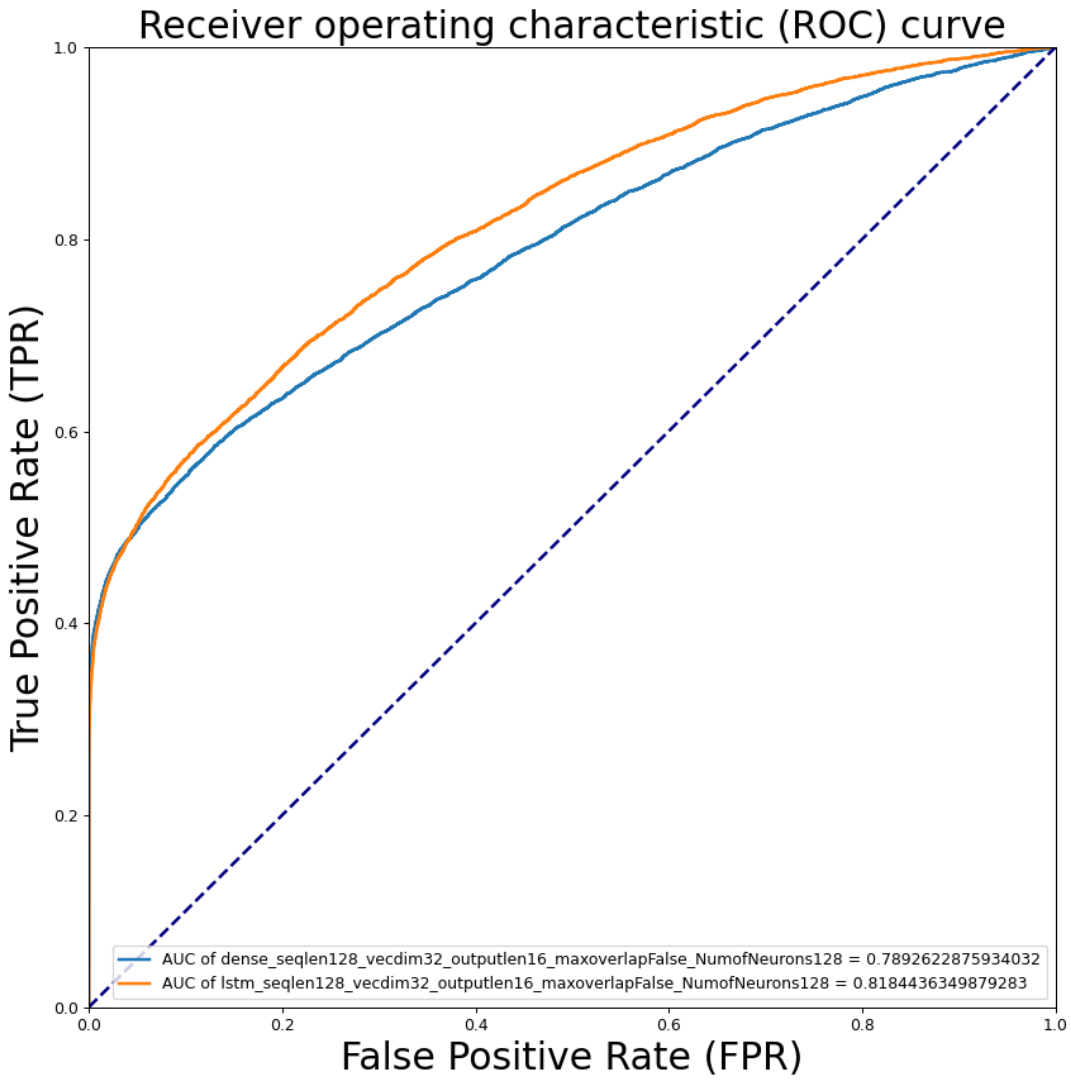


Figure 5.10: ROC curves: conventional NN versus LSTM, with $p = 128$, $n = 32$, $q = 16$, $M = 128$, and non-maximum overlapped strategy. it is also worth mentioning that the real sequence length in this experiment is $128 \times 32 = 4096$.

The best NN among all experiments in terms of AUC scores is discovered, which is the LSTM with $p = 128$, $n = 32$, $q = 16$, 128 neurons and without maximum overlap. Other than the AUC, the maximum balanced accuracy (BA) can also be applied as a metric of performance assessment, and they are listed in the table 5.3,

Network type	Maximum overlap (T/F)	input length p	dimensionality n	output length q	$\max_{\gamma}$ BA
conventional NN	True	16	1	2	0.6713859924178619
		17	1	2	0.667096038292062
		19	1	2	0.6717606348306576
		31	1	2	0.6908369503487524
		47	1	2	0.6866307016866247
	False	4096	1	16	0.7273249441182179
LSTM	True	16	1	2	0.664638530775286
		16	2	2	0.6546995054771664
		16	4	2	0.6583135609609423
		16	16	2	0.6830465352597754
		16	32	2	0.7003792551460092
	False	16	2	2	0.6872680630704903
		16	4	2	0.6787624996723588
		16	16	2	0.7200527344411967
		16	32	2	0.7461558636887556
	False	128	32	16	0.7362720949373311

Table 5.3: Balanced accuracy, $M = 128$ for all NNs in the table.

From BA viewpoint, LSTM with $p = 16$, $n = 32$, $q = 2$, 128 neurons and without maximum overlap is the best, followed by LSTM with $p = 128$, $n = 32$, $q = 16$, 128 neurons and without maximum overlap.

5.4 Apply award functions and discuss the problem

We examine award function 1 and 2 on conventional NN and LSTM while dimensionality n varying from 1 to 32 (same to parameters in 5.1) to capture effectiveness of award functions. We list awards for each NN in the table 5.4.

Network type	Maximum overlap (T/F)	input length p	dimensionality n	output length q	award1= $\frac{1}{N}\sum^{N} y_i\,(x_i - \gamma)$	award2= $\frac{1}{N}\sum^{N} y_i \ln \frac{x_i}{\gamma}$
conventional NN	True	16	1	2	0.09723124771961351	2.9493555748672993
		17	1	2	0.09999176934023832	2.8829075880569435
		19	1	2	0.09884591903288836	2.90818726231427
		31	1	2	0.08855055105398048	2.9318232961190938
		47	1	2	0.09378775437591744	2.787778138015646
LSTM	True	16	1	2	0.10461559978725694	2.7274586928001465
		16	2	2	0.10709689946505914	2.6282662262330843
		16	4	2	0.10760096671489174	2.67529253994409
		16	16	2	0.10084003017936187	2.7205822057824043
		16	32	2	0.09732405004654987	2.7536033213430766

Table 5.4: 128 neurons for all NNs in this experiment, threshold is taken from validation dataset such that MSE distribution residual is 0.01, $a = 1$.

First, we can observe that the performance in terms of award1 for LSTM is better than conventional NN except for the case of $n = 2$, this is almost coherent with conclusions given by AUC scores in 5.1 As for award2, the conclusion is totally opposite to AUC scores in 5.1 because if we look into the term $y_i \ln \frac{x_i}{\gamma}$ in award2 at each time step, see in figure 5.11 as well as details in figure 5.12

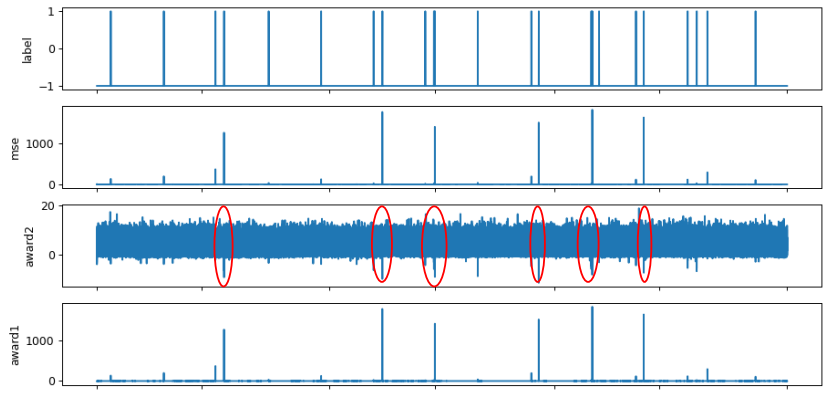


Figure 5.11: An global view of $y_i (x_i - \gamma)$ and $y_i \ln \frac{x_i}{\gamma}$ in award functions at each time step, where $a = 1$ and x-axis is time step.

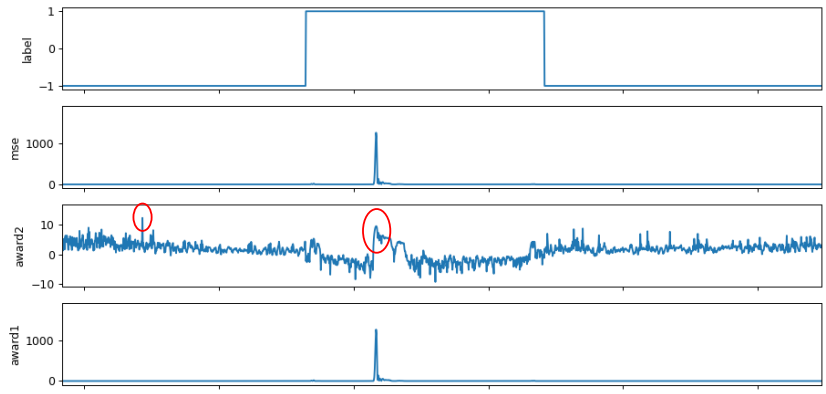


Figure 5.12: Detailed vision of $y_i (x_i - \gamma)$ and $y_i \ln \frac{x_i}{\gamma}$ at each time step with $a = 1$ and x-axis indicates time step.

we can observe that $y_i \ln \frac{x_i}{\gamma}$ in nominal region is comparable to the $y_i \ln \frac{x_i}{\gamma}$ in anomalous region, referring to red circles in award2 in figure 5.11 and figure 5.12. Moreover, since amount of nominal data is far more than anomalies, we will never see the contribution from anomalies to overall award as a consequence. In summary, award2 only reflects the contribution from nominal data, anomaly contribution is negligible. To cope with this issue, we then set $a = \frac{N_-}{N_+}$ and get the following results constituting table 5.5,

Network type	Maximum overlap (T/F)	input length p	dimensionality n	output length q	award1= $\frac{1}{N}\sum^{N} y_i\,(x_i-\gamma)$	award2= $\frac{1}{N}\sum^{N} y_i \ln\frac{x_i}{\gamma}$
conventional NN	True	16	1	2	9.813369179567468	2.0649584555681124
		17	1	2	8.2952031006636	1.9493533044764237
		19	1	2	9.567558729161982	2.0325395374335367
		31	1	2	10.896034018138574	2.3252996396447227
		47	1	2	12.40295429252144	2.3262278914286183
LSTM	True	16	1	2	12.35141987194856	2.1643876871966503
		16	2	2	13.171364312777479	1.956064231855534
		16	4	2	12.202218852403357	1.9955995717581614
		16	16	2	13.841631010573288	2.330033161439289
		16	32	2	14.074036803636176	2.582786706602275

Table 5.5: 128 neurons for all NNs in this experiment, $a = \frac{N_-}{N_+}$.

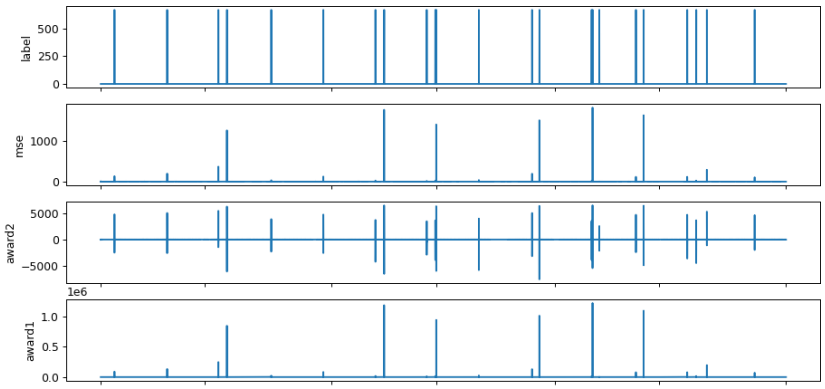


Figure 5.13: $y_i\,(x_i-\gamma)$ and $y_i \ln\frac{x_i}{\gamma}$ at each time step with x-axis standing for time step. $y_i \ln\frac{x_i}{\gamma}$ in nominal region is incomparable to $y_i \ln\frac{x_i}{\gamma}$ in anomalous region anymore when $a = \frac{N_-}{N_+}$, and we can also see considerable penalty when incorrect prediction happens at award2.

Based on table 5.5, we see that LSTM is superior than conventional NN in most of cases according to overall awards except the case of $n = 3$ in award2, we should also keep in mind that when $a = \frac{N_-}{N_+}$, both award1 and award2 are independent of the choice of threshold γ.

Given that award functions are created as an alternative metric for performance assessment, we would love to see that NNs hit the overall award as much high as possible, besides, once $\{y_i\}_{i=1}^{N}$ and $\{x_i\}_{i=1}^{N}$ are provided by NN, award function becomes a function of γ. In that case, the problem of maximizing award turns into finding an optimal threshold such that its corresponding award are maximum. We then examine the latter optimization problem for each NN above with $a = 1$ and observe that optimal threshold always coincides with the maximum threshold, which is reasonable because the optimization problem we are solving is:

$$\gamma^* = \arg\max_{\gamma\in\mathbb{R}} \text{award}$$

If we take the first award function for example, it yields

$$\max_{\gamma} \frac{1}{N} \sum_{i=1}^{N} y_i(x_i - \gamma)$$

and we know

$$\begin{aligned}
&\sum_{i=1}^{N} y_i(x_i - \gamma) \\
=&\sum^{N_+}(x_i - \gamma) + \sum^{N_-}(\gamma - x_i) \\
=&\left(\sum^{N_+} x_i - \sum^{N_-} x_i\right) + \underbrace{(N_- - N_+)}_{>0}\gamma
\end{aligned}$$

$\therefore$

$$\gamma^* = +\infty \quad \text{or} \quad \max_i x_i$$

Analogously, for the optimization problem,

$$\max_{\gamma} \frac{1}{N} \sum_{i=1}^{N} y_i \ln \frac{x_i}{\gamma}$$

we get

$$\gamma^* = +\infty \quad \text{or} \quad \max_i x_i$$

In addition, if we consider contribution to the award from anomalies and nominal data separately, e.g.,

$$\begin{aligned}
\text{award} &= \frac{1}{N_+}\sum^{N_+}(x_i - \gamma) + \frac{1}{N_-}\sum^{N_+}(\gamma - x_i) \\
&= \frac{1}{N_+}\sum^{N_+} x_i - \gamma + \gamma - \frac{1}{N_-}\sum^{N_-} x_i \\
&= \mu_2 - \mu_1 \\
&= \Delta\mu
\end{aligned}$$

Which is independent of the choice of γ. To ensure the existence of well-defined optimal threshold in the award function, we propose 4 candidates of award functions attempting to possess the desired properties, but they all fail in the end, and we will put them down in the

report for the sake of completeness. We denote

$$\sigma\text{— sigmoid function, } \sigma(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leqslant 0 \end{cases}$$

$$y_i\text{— label at } i,\ y_i = \begin{cases} +a, & \text{if anomaly at time step } i \\ -1, & \text{if nominal at time step } i \end{cases}$$

- $$\begin{aligned} \text{award} &= \frac{1}{N}\sum_{i=1}^{N}\sigma\big[y_i(x_i-\gamma)\big] \\ &= \frac{1}{N}\left[\sum^{N_+}\sigma(x_i-\gamma)+\sum^{N_-}\sigma(\gamma-x_i)\right] \end{aligned}$$

  where $\sum^{N_+}\sigma(x_i-\gamma)$ is minority of award which is negligible, and population of term $\sum^{N_-}\sigma(\gamma-x_i)$ is majority which dominates the award, if we maximize the award, normal term tends to select $\gamma^* = \max_i x_i$ at the cost of cancelling the effect from anomalies. and similar to this award function

  $$\begin{aligned} \text{award} &= \frac{1}{\sum^{N_+}\big(x_i-\min_i x_i\big)+\sum^{N_-}\big(\max_i x_i - x_i\big)}\sum_{i=1}^{N}\sigma\big[y_i(x_i-\gamma)\big] \\ &= \frac{1}{\sum^{N_+}\big(x_i-\min_i x_i\big)+\sum^{N_-}\big(\max_i x_i - x_i\big)}\left[\sum^{N_+}\sigma(x_i-\gamma)+\sum^{N_-}\sigma(\gamma-x_i)\right] \end{aligned}$$

  it also leads us to $\gamma^* = \max_i x_i$ in the end.

- if we take both contribution from anomalies and from nominal into account separately:

  $$\begin{aligned} \text{award} &= \frac{1}{N_+}\sum^{N_+}\sigma(x_i-\gamma)+\frac{1}{N_-}\sum^{N_-}\sigma(\gamma-x_i) \\ &\quad\downarrow \text{ in the absence of false positives and negatives, it yields} \\ &= \frac{1}{N_+}\sum^{N_+}(x_i-\gamma)+\frac{1}{N_-}\sum^{N_-}(\gamma-x_i) \\ &= \mu_2-\mu_1 \end{aligned}$$

regardless of the choice of γ as well. Also, we may come up with

$$\begin{aligned}
\text{award} &= \frac{1}{\sum_{N_+}\left(x_i - \min_i x_i\right)} \sum^{N_+} \sigma(x_i - \gamma) + \frac{1}{\sum_{N_-}\left(\max_i x_i - x_i\right)} \sum^{N_-} \sigma(\gamma - x_i) \\
&\quad \downarrow \text{ in the absence of false positives and negatives, it yields} \\
&= \frac{1}{\sum_{N_+}\left(x_i - \min_i x_i\right)} \sum^{N_+} (x_i - \gamma) + \frac{1}{\sum_{N_-}\left(\max_i x_i - x_i\right)} \sum^{N_-} (\gamma - x_i) \\
&= \text{constant} + \underbrace{\left[\frac{N_-}{\sum_{N_+}\left(x_i - \min_i x_i\right)} - \frac{N_+}{\sum_{N_+}\left(x_i - \min_i x_i\right)}\right]}_{\text{constant}} \gamma
\end{aligned}$$

and it ends up with either $\gamma^* = \max_i x_i$ or $\gamma^* = \min_i x_i$.

On the other hand, I think finding an optimal threshold according to probability density functions of anomalies and nominal data is analogous to finding an optimal hyperplane by SVM in a sense that it can be regarded as 1-dimensional case of SVM problem, hence, I would like to stick to this issue in near future.

Chapter 6

Conclusions

This thesis project focus on the anomaly detection by prediction on the dataset from the satellite. We adopted recurrent neural network (RNN) as well as Long Short-Term Memory (LSTM) to anomaly detection as they are capable to learn time dependency from time series compared to conventional NN. We first try a toy case on ECG data which helps us exclude the adoption of RNN in the following chapters since it brings us the worst performance. Then we move to the dataset from satellite where we tried conventional NNs and LSTM with different parameters and compare their performance in order to find an appropriate NN to perform anomaly detection on satellite data. We have confronted some challenges most of which are solved quite well except for the last one, more specifically:

- The difficulty of training LSTMs when the number of parameters and input sequence length are too large, and we propose our custom callbacks which will load the latest optimal weights whenever we have problems with training process.
- At each time step, the dimensionality of samples we feed to LSTM seems too small compared to the number of parameters it needs to tune, we lessen this unbalancing issue by different reshaping strategies, which also help to relieve the difficulty of training a LSTM with very large input length, and we conclude that
  - LSTM with input length $p = 16$, dimensionality $n = 32$, output length $q = 2$, 128 neurons and without maximum overlap is the best among all the NNs in terms of balanced accuracy.
  - However, LSTM with $p = 128$, $n = 32$, $q = 16$, 128 and without maximum overlap outperforms most with respect to AUC metric.
- Apart from that, we invent a new performance metric, award function, which takes not only the binary labels but also its values into account, and we compare it with conclusions drawn from AUC metric to see its effectiveness. Unfortunately, it is not perfectly-aligned with our expectation as it possesses a fatal defect which has been proved both from practical and theoretical viewpoints. And we attempted to fix the issue, but failed again, then we come up with another idea that will be examined in the near future.

Chapter 7

Acknowledgements

I am so grateful to my dear supervisor, friend, and our captain, Riccardo Rovatti! for his wisdom, encouragement as well as kindness. Indeed, I can not tell you how I miss the time I was in his class, he can always answer my questions with magic words that evoke my enthusiasm to dive in deeply and consistently. Without exaggeration, it is one of the happiest times I had ever been and I shall cherish it forever.

And also, I would like to introduce you my dear friends whom I really appreciate the time we have shared with together:

- Mauro Mangia, from whom I learned how to carry out scientific research, I would thank for his guidance, feedback, patience, as well as accompany all the time.
- Alex Marchioni, to whom I thank for his timely inspirations which always help me out as well as delicious Italian food he recommended.
- Filippo Martinini, Other than the professional advice, ideas, as well as guidance, many thanks for wonderful cooperation and music sharing.
- Andriy Enttsel, warmhearted help both from code level and mathematics explanation.

Last but not the least, my parents, my regards to their unconditional support and contribution to the family.

Finally, I would like to put forward a proposal: Probably we should put more attention to colors we used in scientific paper works if we think of the population of the color weakness in the world, for example, we can avoid using red and green in the same figure. Of course, it would never be possible to remove all the obstacles among people, but at least, we attempt to do so.

Bibliography

[1] Charu C Aggarwal. *Outlier analysis second edition*. Springer Switzerland, 2016.

[2] Andreas Arning, Rakesh Agrawal, and Prabhakar Raghavan. A linear method for deviation detection in large databases. In *KDD*, volume 1141, pages 972–981, 1996.

[3] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. Optics-of: Identifying local outliers. pages 262–270, 09 1999.

[4] Markus Breunig, Hans-Peter Kriegel, Raymond Ng, and Joerg Sander. Lof: Identifying density-based local outliers. volume 29, pages 93–104, 06 2000.

[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[6] Sucheta Chauhan and Lovekesh 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)*, pages 1–7, 2015.

[7] Guillaume Chevalier. Lstm cell. Schematic of the Long-Short Term Memory cell, a component of recurrent neural networks, May 2018.

[8] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[9] F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999.

[10] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 10 2000.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 374. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 378. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 288. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 409. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[16] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[19] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 387–395, 2018.

[20] Theodore Johnson, Ivy Kwok, and Raymond Ng. Fast computation of 2dimensional depth contours. pages 224–228, 01 1998.

[21] Murat Karakaya. Lstm: Understanding the number of parameters, Nov 2020.

[22] Edwin M. Knorr and Raymond T. Ng. A unified approach for mining outliers. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, page 11. IBM Press, 1997.

[23] Edwin M Knorr and Raymond T Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB*, volume 98, pages 392–403. Citeseer, 1998.

[24] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[25] Tianyu Li, Mary L. Comer, Edward J. Delp, Sundip R. Desai, James L. Mathieson, Richard H. Foster, and Moses W. Chan. Anomaly scoring for prediction-based anomaly detection in time series. In *2020 IEEE Aerospace Conference*, pages 1–7, 2020.

[26] Andrea Lörke, Fabian Schneider, Johannes Heck, and Patrick Nitter. Cybenko's theorem and the capability of a neural network as function approximator. Sep 2019.

[27] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94, 2015.

[28] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[29] Van Quan Nguyen, Linh Van Ma, Jin-young Kim, Kwangki Kim, and Jinsul Kim. Applications of anomaly detection using deep learning on time series data. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 393–396, 2018.

[30] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[31] Tony Robinson and F. Fallside. The utility driven dynamic error propagation network, 01 1987.

[32] F ROSENBLATT. The perceptron: A probabilistic model for information storage and organization in the brain1. *Psychological Review*, 65(6):19S8.

[33] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[34] Ida Ruts and Peter J. Rousseeuw. Computing depth contours of bivariate point clouds. *Computational Statistics & Data Analysis*, 23(1):153–168, 1996. Classification.

[35] Dhruvil Shah, Soham Khade, and Sudesh Pawar. Anomaly detection in time series data of sensex and nifty50 with keras. In *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, pages 433–438, 2021.

[36] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[37] Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2018.