Anomaly Prediction on Error Messages Data from Hospitals for ChipSoft

Layout: typeset by the author using LaTeX.
Cover illustration: Lisanne Coorengel

Anomaly Prediction on Error Messages Data from Hospitals for ChipSoft

Josephine de Kok
12672289

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisors
Dr. S. van Splunter
MSc. M. van Rood

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 30, 2022

1 Acknowledgement

I would like to express my deepest thanks to both my supervisors: Sander van Splunter from the UvA and Maarten van Rood from ChipSoft. They have helped and supported me during the entire process, from helping to give shape to the research to answering questions, providing feedback, and giving me some structure and reassurance when needed.
I would also like to express my gratitude to Jurgen Baas, who has given some great advice and feedback throughout this project. And, not to forget, Denise Monkau for the detailed feedback I got on my first complete version of the thesis.
Furthermore, I would like to thank ChipSoft for providing me with this opportunity and the UvA for allowing me to write my thesis in ChipSoft's AI team and for helping me gain the skills and knowledge in the past three years needed to conduct this research and write this thesis.
Then some special thanks to everybody in IS Kliniek, especially the people in my room and in Preop, along with Fieke Vermeer who first brought me into contact with Maarten van Rood.
I also appreciate everybody attending the scrum meetings, keeping up with the progress; And all students following the Afstudeerproject Bachelor KI in the fifth and sixth period of the school year of 2021-2022, asking and answering questions in the lectures and listening to my presentations and thereby also providing me with ideas and feedback.
Finally, this project would not have been possible without my friends and family supporting me the past three months and giving me all the advice, suggestions, guidance and support I could have needed.

Abstract

On behalf of ChipSoft, this thesis compares three models: Simple RNN, LSTM and Transformer on their ability to predict anomalies based on hospital error messages data. Furthermore, it explores possible improvements by incorporating uncertainty i.e. introducing dropout to the models. In order to make it possible to compare the models, this thesis first tackles the challenge of defining the anomalies through a Peaks Over Threshold method with a dynamic threshold.
The results show that for anomaly prediction on the hospital error messages data Transformer is the preferred model out of the three. Not only does it have the best average run score and the shortest runtime, it is also the most reliable method that gets satisfactory results on every run. On top of that, Transformers predictions are also by far the closest to the original values when predicting anomalies.
In terms of uncertainty within the models, dropout only lowered the scores. Possible explanations for the decreasing results are underfitting and/or the singularity of our data (one data-input per data point).

Keywords: Anomalies, Dynamic threshold, Anomaly prediction, Simple RNN, LSTM, Transformer, Dropout

Contents

2 Introduction

ChipSoft is a Dutch software company that develops software for the healthcare sector, mainly hospitals. As with every working software system, errors sometimes occur and they usually follow a certain pattern. On average, there are more errors during the day than the night since more people use the software during the day. The same applies to the weekdays, during which more errors occur than on the weekends. However, sometimes there is a sudden substantial increase in the number of errors which might happen when something crashes or if there is a bug in the software, this sudden increase is called an anomaly.

An anomaly is abnormal behaviour of the system which results in unexpected increases in error messages. A well-known implementation that pinpoints these increases is the Peaks Over Threshold (POT) method. It labels data points as anomalies if they are bigger than a given threshold. The problem is that there is not one static threshold for this particular case since the number of error messages differs greatly depending on the time and day. The solution is a dynamic threshold that changes based on the time and day. However, this does not solve the entire problem because it is still a challenge to determine the height of this dynamic threshold.

Note that in this definition of an anomaly we only look at peaks in the database, so heaps of error messages coming in. If the number of error messages coming in is considerably low, especially on a normally busy time, this could also be named an anomaly. It is abnormal to get next to no error messages and this might point to a problem with relaying the error messages for example. But since this would be an extremely rare event and because it is difficult to track whether the low numbers are a coincidence or low because of a problem, we do not take this into account.

Within ChipSoft these error messages are called JIPs[1]. The amount of JIPs a hospital gets is registered which means that if there is a sudden significant increase, this can be seen in the so-called "JIPS-melder". However, there is a lot of data and to check this manually is practically impossible. It should explicitly be pointed out that the data only takes note of the number of error messages and has no information about the substance nor the source of the error. As an exploratory initial solution to automatically check the data ChipSoft has a Seasonal Trend LOESS Decomposition (STL) implementation for anomaly detection. This type of time series analysis is used for identifying anomalies in data e.g. see [Zhang et al., 2021]. These anomalies are data points that deviate significantly

[1]JIPs is not an acronym, but was named after a dog

from the expected value i.e. they do not follow the "normal" behaviour of the majority of the data.

STL detects anomalies by finding seasonal and trend patterns in the data. Think of the temperature changes within a year. In the northern hemisphere the temperature reaches its peak in summer then gets colder in fall, gets very cold in winter and gets warmer again in spring, until it is at its peak of hotness in summer again. Because there are temperature changes within one year, to compare different years, you need to take the seasonal component out of the equation and this is what STL does. All the data is smoothed and the seasonal component is created and then subtracted from the raw data, thus eliminating seasonal variation.

Trend patterns are found similarly, and for ChipSoft this trend component contains some interesting information. An increasing trend component might indicate that there will be problems in the future since, over a long period of time, more and more error messages keep appearing. But to find this trend pattern, the data (this time without the seasonal component) is smoothed again and the trend component is created which is then subtracted from the data creating the remainder component. If the remaining values are close to zero, the seasonal and trend components were a very accurate way of describing the time series. Other remaining values can be seen as noise or anomalies.

STL is a strong model for anomaly detection since it has some significant advantages [Li et al., 2020], these include a strong resilience to outliers in the sequence, resulting in robust sub-series. And STL can deal with any seasonal frequency that is greater than one. Furthermore, STL's process is based purely on numerical methods, not making use of any mathematical modelling tools.

There are some disadvantages of STL, one of which is that STL is not an explainable method such as the explainable anomaly detection models such as EXAD and SHAP. [Song et al., 2018] [Jakubowski et al., 2021] This means that it can not be explained exactly how the seasonal and trend patterns and thus also the anomalies are found. More clarity on how data points are labelled as anomalies is preferred, this would also give more insight on the definition of an anomaly.

But by far the biggest disadvantage of STL is that it can only do anomaly detection based on data that already exists, which means that one can only detect an anomaly when it has passed (or is still currently happening). It is way more desirable to be able to predict anomalies instead of detecting them afterwards, as this enables the possibilities for proactive interventions. This future-oriented approach is possible with anomaly prediction which is predicting when anomalies will take place based on previous data.

There are different models that can be effective for time series anomaly prediction and in this thesis, we discuss and compare a few of them: Simple RNN, Long Short-Term Memory (LSTM) and Deep Transformer Networks (Transformer). Here we assume with endorsement of ChipSoft's tech-team, that predicting one hour into the future will be enough to proactively intervene, may anomalies occur. It is possible to predict further into the future, although this will most likely negatively impact the performance of the models, but examining this is beyond the scope of this paper. On top of the different methods, we also investigate the potential of incorporating uncertainty into the models, which leads us to the following research questions:

1. How can we define an anomaly within error messages data from a hospital?
2. Between Simple RNN, LSTM, and Transformer, what is the most effective model for time series anomaly prediction based on the F1-score?
3. Does incorporating uncertainty on top of anomaly prediction models improve their performance?

The F1-score has two components: precision and recall. Precision stands for the percentage of how many of the predicted anomalies were actually correct and recall stands for the percentage of how many of the actual anomalies were predicted by the models. They are calculated as follows:

$$Precision = \frac{TruePositives}{TruePositives+FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives+FalseNegatives}$$

$$F1 = \frac{2*Precision*Recall}{Precision+Recall} = \frac{2*TruePositives}{2*TruePostives+FalsePositives+FalseNegatives}$$

The reason to use the F1-score as an effectiveness measurement instead of accuracy is because we want to find anomalies. Anomalies are by definition extremely rare, which means that if the model were to label everything as a "normal" data point it would get fantastic scores when using accuracy even though this is not a good prediction. This problem does not occur when using the F1-score.

3 Background

In the background we provide context to Simple RNN, LSTM, Transformer, and Incorporating Uncertainty.

3.1 Simple RNN

A recurrent neural network (RNN) is a dynamic neural network that feeds signals from previous timesteps back into the network. So it uses an encoder-decoder based deep learning technique that uses reconstruction errors to detect and predict anomalies and is used specifically to deal with sequential data because of its memorization. [Mathonsi and Zyl, 2022] This feeding back of the input is what sets RNN apart from "normal" feed-forward neural networks (e.g. see figure 1) [2]

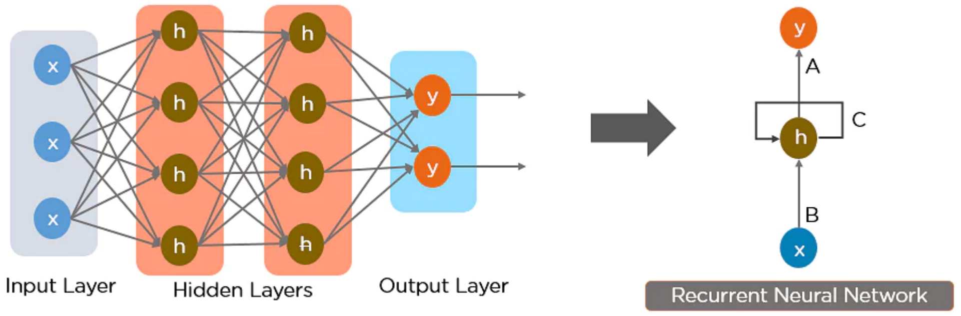

Figure 1: Feed-forward neural network vs a recurrent neural network

Basically, a RNN consists of a hidden state h and an optional output y which operates on the sequence length x = $(x_1, ..., x_T)$. The hidden state $\mathrm{h}_{\langle t \rangle}$ is updated by $\mathrm{h}_{\langle t \rangle} = f(\mathrm{h}_{\langle t-1 \rangle}, x_t)$ at each time step t, where f is the activation function. [Cho et al., 2014] For a regression problem such as ours, f should be linear.

This basic idea of introducing memory into the neural network helps Simple RNN extract information from a sequence of data points which is ideal for anomaly prediction. However, RNNs are known to suffer from the "vanishing error problem" which limits them from looking back further than ten timesteps because the signal that is fed back either "vanishes" or "explodes". [Staudemeyer and Morris, 2019] Essentially, since the gradient descent algorithm, which RNNs use, looks for the smallest gradients, the gradients gradually get smaller until they barely change. The model learns from the change in the gradients, so if that change is nihil, the model cannot learn anymore and thus faces the vanishing gradient problem causing it to not converge towards a good solution.[3]

[2]https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn

[3]https://www.guru99.com/rnn-tutorial.html

3.2 LSTM

Long Short-Term Memory (LSTM) is a version of a RNN, but uses some advanced techniques to solve the "vanishing error problem", a major obstacle Simple RNN deals with. The main trick of LSTM networks is memory blocks which control the input and output gates by preventing irrelevant information from entering or leaving. Memory blocks also have a forget gate which weighs the information inside the cells and can reset the state of the cell if the information becomes irrelevant which is an important part of the LSTM networks because it enables continuous prediction which prevents biases in the predictions. [Staudemeyer and Morris, 2019] LSTM's solution for the "vanishing error problem" that the Simple RNN struggles with, is the introduction of the constant error flow which enables LSTM to learn more than 1,000 timesteps, depending on the complexity of the network. [Staudemeyer and Morris, 2019] In short, LSTM is a more complex version of the Simple RNN but makes use of the same basic principle of a recurrent neural network where the network feeds signals from previous timesteps back into the network.

LSTM is a popular model, especially for Natural Language Processing (NLP), but also for anomaly detection (and prediction) because it has gotten some great results in comparison to other models. [Mathonsi and Zyl, 2022] [Li et al., 2020] [Wang et al., 2019] Because LSTM can not only learn from single data points but also from entire sequences of data it is well-suited for classification problems, preprocessing and making predictions based on time series. Anomaly prediction falls under making predictions based on time series which makes LSTM an excellent choice for this problem. The combination of anomaly prediction being a problem LSTM is well-suited for and it getting exquisite results in similar experiments makes LSTM a model with strong potential for the anomaly prediction on ChipSoft's error messages data from hospitals.

3.3 Transformer

Deep Transformer Network (Transformer) is a deep neural network that uses attention-based sequence encoders and focus-score based self-conditioning for robust multi-modal feature extraction and adversarial training to gain stability. [Tuli et al., 2022] So basically the attention is not divided equally between all data points but data points with higher focus scores get more attention. This focus score is determined by the deviation that the model calculates while trying to generate an approximate reconstruction of the input window. [Tuli et al., 2022]

Recurrent models cannot effectively model long-term trends because at each timestamp they first need to perform inference for all previous timestamps before they can proceed, but Transformer models can extract relations while ignoring distance [Meng et al., 2019] [Tuli et al., 2022] So unlike the slower and computationally expensive recurrent models, such as Simple RNN and LSTM, Transformer models allow single-shot inferences with the complete input series using position encoding. Furthermore, Transformer uses parallel computing transformers which is also why it can detect anomalies way faster than recurrent methods. [Wen et al., 2022]

Not only is Transformer significantly faster than Simple RNN and LSTM, but it has also been reported to get good and sometimes even better results on tasks such as anomaly detection. [Wen et al., 2022] [Meng et al., 2019] Because of this, Transformer has great potential with regard to anomaly prediction on ChipSoft's error messages data from hospitals.

3.4 Incorporating Uncertainty

By definition, uncertainty refers to working with imperfect, incomplete, and/or unknown information, and the two main types of uncertainty used in AI are: aleatoric and epistemic uncertainty. Aleatoric uncertainty, also known as data uncertainty, is not a property of the model, but of the data distribution. Epistemic uncertainty, on the other hand, which is also known as knowledge uncertainty is a property of the model and occurs due to inadequate knowledge. [Abdar et al., 2021]

Dropout is a form of epistemic uncertainty and is often introduced to models since it reduces overfitting which is a problem many models struggle with. Reducing the overfitting of a model can result in major improvements and it has been shown that dropout can significantly improve neural networks in many different fields such as speech recognition, document classification, computational biology, and anomaly detection. [Zhu and Laptev, 2017] [Srivastava et al., 2014]

As can be seen in figure 2, the key idea of dropout is to randomly drop units (along with their connections) from the neural network during training. [Srivastava et al., 2014] Note that this means that no values are dropped during inference. If one would like to use dropout in the training as well as the test phase, it is possible to use Monte Carlo Dropout. [Seoh, 2020]

[4]

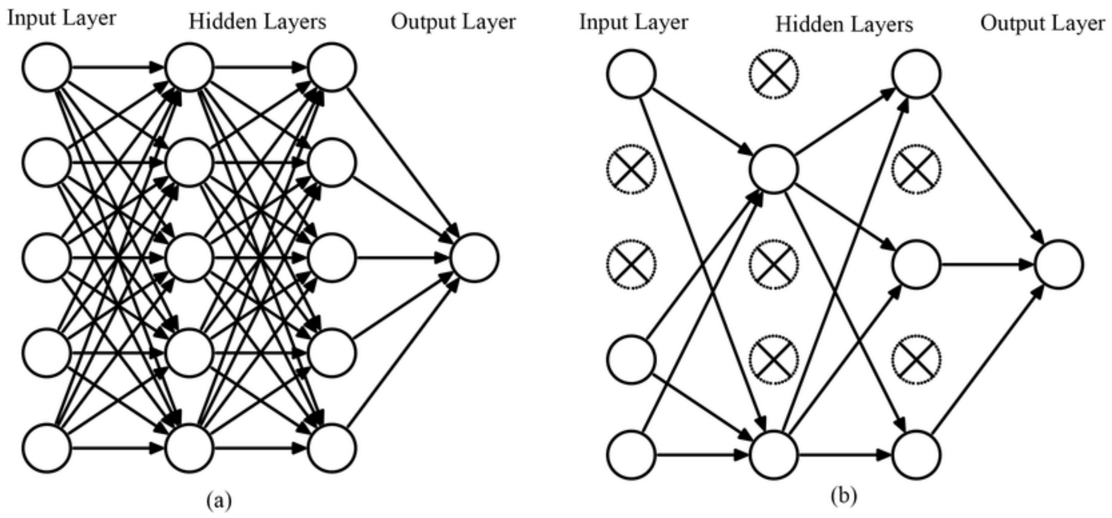

Figure 2: Neural network without dropout vs. neural network with dropout

[4]https://www.researchgate.net/figure/Dropout-Strategy-a-A-standard-neural-network-b-Applying-d
fig3_340700034

4 Method

In this section we first introduce the dataset, then we explain how to define anomalies specifically for the hospital error messages, and then lastly, we describe the implemented codes. We first give a basic overview of the different steps of the codes whereafter we expand on the important and more complex parts.

4.1 Dataset

The original dataset exists only out of 25139 rows containing the DateTime and Count (and as always a row ID), where the Count shows the number of error messages and the DateTime the corresponding hour, the dataset is thus an hourly aggregate of the number of error messages. After setting the thresholds two more columns are added to the dataset: Threshold and Anomaly. Threshold contains the original threshold (this threshold later gets transformed to be compatible with the transformed data points) and Anomaly is a boolean where False stands for a normal data point and True for an anomaly. Finally, after the transformations of the data points and the thresholds, the transformed thresholds are also added to the dataset. This transformation is simply centring the data to have a zero mean and is based on the anomaly detection implementation of Victor Schmidt.[5] He argues that neural networks learn better when data is preprocessed but since we are working with time-series we want to keep the data as close to the real world as possible.

All by all this means that per row we have the following columns:

Row ID	DateTime	Count	Threshold	Anomaly	Transformed threshold

These 25139 rows each contain information about 1 hour per day which means that the data spans over a time of 25139/24=1047,5 days ($\approx$ 2,87 years). The name of the hospital from which the data originates is confidential

In total, after calculating the dynamic threshold, around 0.5% of the whole dataset (128 out of 25139 data points) has the label: “anomaly” based on the chosen dynamic threshold. How these anomalies were exactly labelled is explained in defining anomalies.

[5] https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent

4.2 Defining Anomalies

In the introduction the definition for an anomaly was introduced: abnormal behaviour which results in unexpected increases in error messages. Specifically for the hospital domain, this definition is further refined in the remainder of this section.

In the hospital domain there is a strong difference between hourly behaviour in weekdays and hourly behaviour in weekends, e.g., due to polyclinic care being restricted to weekdays. A dataset is created in which all data points between 9.00 and 10.00 AM on a Saturday and a Sunday are put in one list and all data points between 9.00 and 10.00 AM on every weekday are put in another list, the same goes for the other 23 hours of the day. The resulting dataset consists of 48 lists of data points: 24 for the weekend, and 24 for weekdays. There is no substantial (long-term) increase or decrease in the data which allows for one dynamic threshold based on all data points per hour, if there was, the dynamic threshold would have had to be lowered or elevated as time goes by.

To pinpoint the anomalies in the dataset we need to determine the height of the dynamic threshold for every hour. For this, we had to make a few assumptions:

1. Within the weekend (so Saturday and Sunday) there is no significant difference in the number of error messages coming in.
2. Within the weekdays there is no significant difference in the number of error messages coming in.
3. For the weekend and weekdays separately there is no significant difference in the number of error messages coming in between the same hours on a different day.

For every one of these 48 lists, a threshold is determined which is done based on the distance of the biggest majority of the data points in that particular list vs. the few data points that are significantly bigger than the others. How much higher these points have to lie for them to be an anomaly is very subjective, one could choose for the threshold to be just above 90% of all data points in the list or above 99% of all data points in the list.The higher the threshold, the less data points are labelled anomalies. However, this same threshold is used for determining which of the predictions should be labelled anomalies which means that since the data points undergo transformations, the threshold should also go through that same process for the purpose of a fair evaluation.

The effect of the threshold is as follows: if the original data point is higher than the threshold (so it is labelled as an anomaly) then the predicted point will, if the model works well, also get a value higher than the (now transformed) threshold since it is essentially the same threshold. Now if we put the threshold higher for the original data and the original point is now under the threshold (so no anomaly), this same threshold will also be higher for the predicted values and thus will, if the model works well, the predicted value of the data point also be under the threshold now. The only difference between the higher and lower threshold is thus that more points will be labelled as an anomaly in both the original dataset and the predicted dataset if the threshold is higher, and fewer points will be labelled as an anomaly if the threshold is lower.

Note that this method of defining anomalies does, rightfully so, not divide the anomalies evenly over the database. Some parts have relatively loads of anomalies and others have none. It is not unusual that when one anomaly occurs this problem is not suddenly gone the hour after, it could take a few hours meaning that more data points in a row will be labelled anomalies. But since this happens more often, this is behaviour the models learn to recognize and then predict.

4.3 Predicting Anomalies With Simple RNN, LSTM, and Transformer

In short, Simple RNN, LSTM, and Transformer go through the following ten steps:

1. Import all needed packages/libraries
2. Load the data
3. Split the data
4. Preprocess the data
5. Compile the model
6. Train the model
7. Predict the values
8. Save data in csv-file
9. Calculate precision and recall
10. Plot the results

For all methods the steps are the same, it is the models and thus the layers of the neural networks that differ. For an in-depth explanation of the different models, see the experimental setup. The codes can be found in the GitHub repository. [6]

The data is split into a training and test set whereby 80% of the data will be used for training and 20% for testing. The preprocessing of the data is very minimal because we want the network to learn from data that is as close to the real world as possible. For an optimal working of the neural networks, we centre the data around 0, so we subtract the mean and divide the result by the standard deviation. Likewise, the threshold undergoes the same transformations to keep it compatible with the data points.

In terms of the different models, they undergo the same steps: First, the model is initialised, and thus the layers are created (for an in-depth explanation of the different layers, see the experimental setup), then the models are compiled using a Mean Square Error and the RMSprop optimizer and the last step before predicting, is the training of the models which is done using model.fit().

[6]GitHub: `https://github.com/MariekePop/Thesis_Anomaly_Prediction.git`

How exactly the function predict() works is as follows: by construction X_test is the list that contains the sequences, which means that X_test[0] is the first sequence, that is to say, the first 6 values of the original data points (see the 6 data points within the red dashed line in figure 3). By predicting based on X_test[0] we try to predict the 7th data point (the red outlined data point in figure 3). This is the associated target for this sequence and is stored in y_test[0]. The same principles are applied to the other sequences which means that: predict(X_test[1]) (see the 6 data points within the blue dashed line in figure 3) is the prediction of the 8th data point, which is the associated target being y_test[1] (the blue outlined data point in figure 3) and so on.

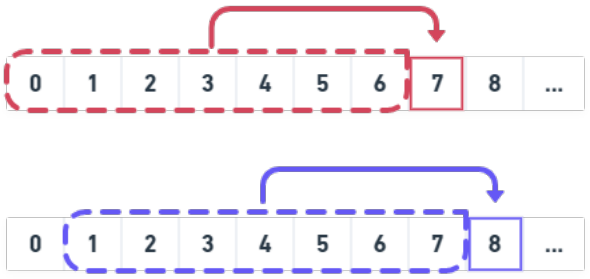

Figure 3: Examples of X_test in dashed line and Y_test fully outlined

The result of this predict(X_test) is a list of lists (a 2-dimensional numpy array) with one value, which gets reshaped so that it simply is a list of values (1-dimensional numpy array), which makes it easy to compare to the original data points.

In the GitHub repository[7], there is one main file per method that includes both the training and predicting and there is a split version in which one file only does the training and the other only predicts which could be used for a faster prediction. It is advised to train the model again when more data becomes available.

[7]GitHub: `https://github.com/MariekePop/Thesis_Anomaly_Prediction.git`

5 Experimental Setup

In the experimental setup we describe the architecture of the Simple RNN, LSTM and Transformer models and how dropout is used to introduce uncertainty to the models.
All the implementations are coded in python using the Keras API.[8]

5.1 Simple RNN

The Simple RNN code was inspired by the Keras recurrent tutorial by Victor Schmidt.[9]

For the implementation of Simple RNN a sequential layered model is used instead of a graphical layered model because each layer has one input tensor and one output tensor. A sequential model behaves very similarly to a list of layers which means layers can be added and removed from the model. The Simple RNN makes use of four layers (see figure 4): A Simple RNN input layer (with return sequence), a Simple RNN hidden layer with return sequence, and a Simple RNN hidden layer without return sequence, and lastly, a Dense output layer with a linear activation function. Dense layers are fully connected, which means that all the neurons of the layer are connected to every neuron of the layer before.
The reason the first two layers of the model return sequences is because all its information should be transferred to both the next layer and to itself for the next timestep. However, for the third layer, only the last sequence prediction is to be compared to the target, which means that it is not necessary to return the whole sequence. This means that for the third layer the full prediction of inputs (all seven values) is solely passed to the layer itself and not as an input to the next layer (which is the output layer), only the last prediction of the input in the sequence is given to the Dense output layer.

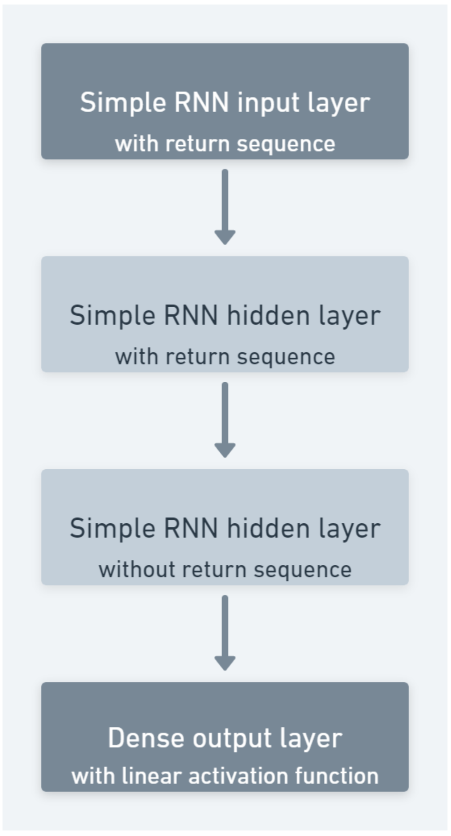

Figure 4: Layers of the Simple RNN model

For a more in-depth explanation of the essential RNN (and LSTM) fundamentals with mathematical explanations and proof, see: [Sherstinsky, 2020].

[8]https://keras.io/api/
[9]https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent

5.2 LSTM

The implemented LSTM code was also inspired by the Keras recurrent tutorial by Victor Schmidt.[10]

The implementation of LSTM and the Simple RNN are very similar because LSTM is a version of a RNN. So just like the Simple RNN implementation LSTM makes use of a sequential layered model instead of a graphical one because each layer has one input tensor and one output tensor. But in the LSTM model, the first layer is a LSTM input layer (with sequence return) instead of a Simple RNN input layer. Furthermore, the other layers in the LSTM model are two LSTM hidden layers, from which the first does return sequences and the second one does not, and a Dense output layer with a linear activation function (see figure 5).

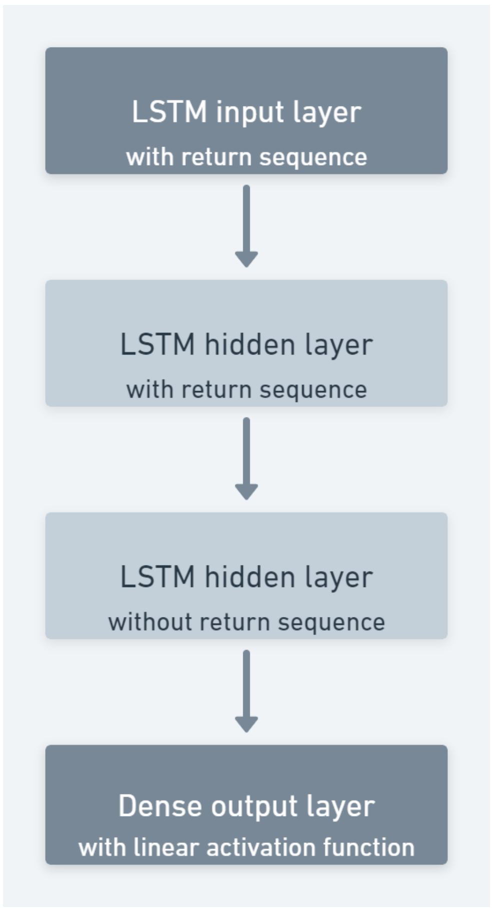

Figure 5: Layers of the LSTM model

[10]`https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent`

The structure of a LSTM layer is however way more complicated than a Simple RNN layer, for a detailed explanation this thesis refers to Olah[11]

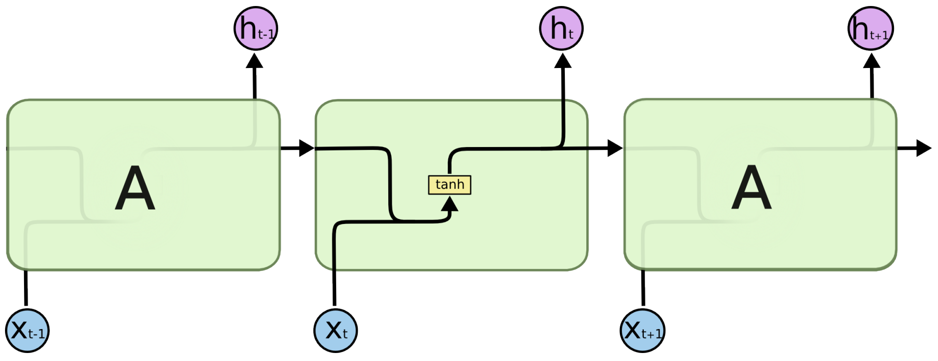

Figure 6: The structure of a Simple RNN layer

As can be seen in figure 6 the structure of a Simple RNN exists out of a single tanh layer. However, as can be seen in figure 7, LSTM has, instead of one single neural network layer, four different sublayers that interact with each other.

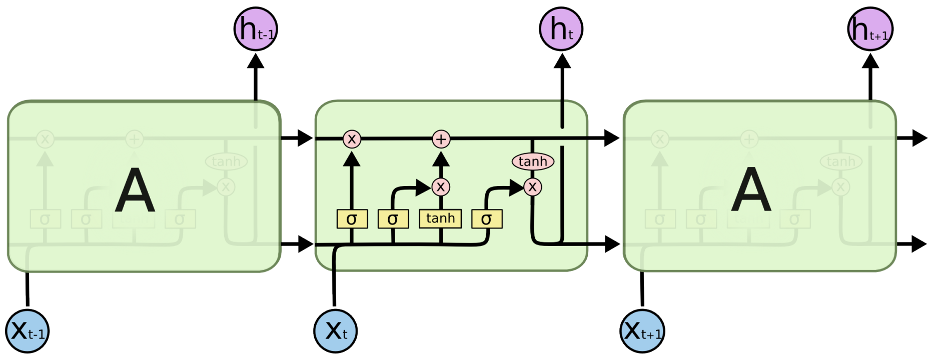

Figure 7: The structure of a LSTM layer

Every yellow box denotes a subneural network, the first is the (sigmoid) forget gate sublayer, the second is the (sigmoid) input gate sublayer, the third is the (tanh) sublayer that creates a vector of the new candidate values and the fourth and last sublayer (sigmoid) sublayer decides which of these candidate values are put into the output.

For a more in-depth explanation of the essential LSTM (and RNN) fundamentals with mathematical explanations and proof, see: [Sherstinsky, 2020].

[11]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

5.3 Transformer

The implemented Transformer code was inspired by the Keras Timeseries classification with a Transformer model tutorial by Theodoros Ntakouris.[12]

Compared to the Simple RNN and LSTM implementations, the Transformer model has more layers which are needed to make use of Transformers attention and focus-scores (see figure 8). Starting with the standard input layer, the model directly goes to two TransformerEncoder blocks. These blocks exist out of an attention and feed-forward component.
The attention (and normalisation) component goes as follows: First there is a MultiHeadAttention layer and then a LayerNormalization layer. The output of the LayerNormalization layer is added to the input of the attention component of the block. This will then function as input for the feed-forward component of the block which exists of: First two Conv1D layers whose output is given to a LayerNormalization layer. And again the output of this LayerNormalization layer is added to the input of the feed-forward component of the block.
This output then goes to a GlobalAveragePooling1D layer because we need to reduce the output tensor of the TransformerEncoder blocks down to one vector for each data point which is commonly done with a pooling layer. Then we add a Dense layer with a relu activation function and lastly a Dense output layer with a linear activation function.

For a mathematical explanation of the Transformer model, see Thickstun[13].

[12]`https://github.com/keras-team/keras-io/blob/master/examples/timeseries/timeseries_classification_transformer.py`

[13]`https://www.guru99.com/rnn-tutorial.html`

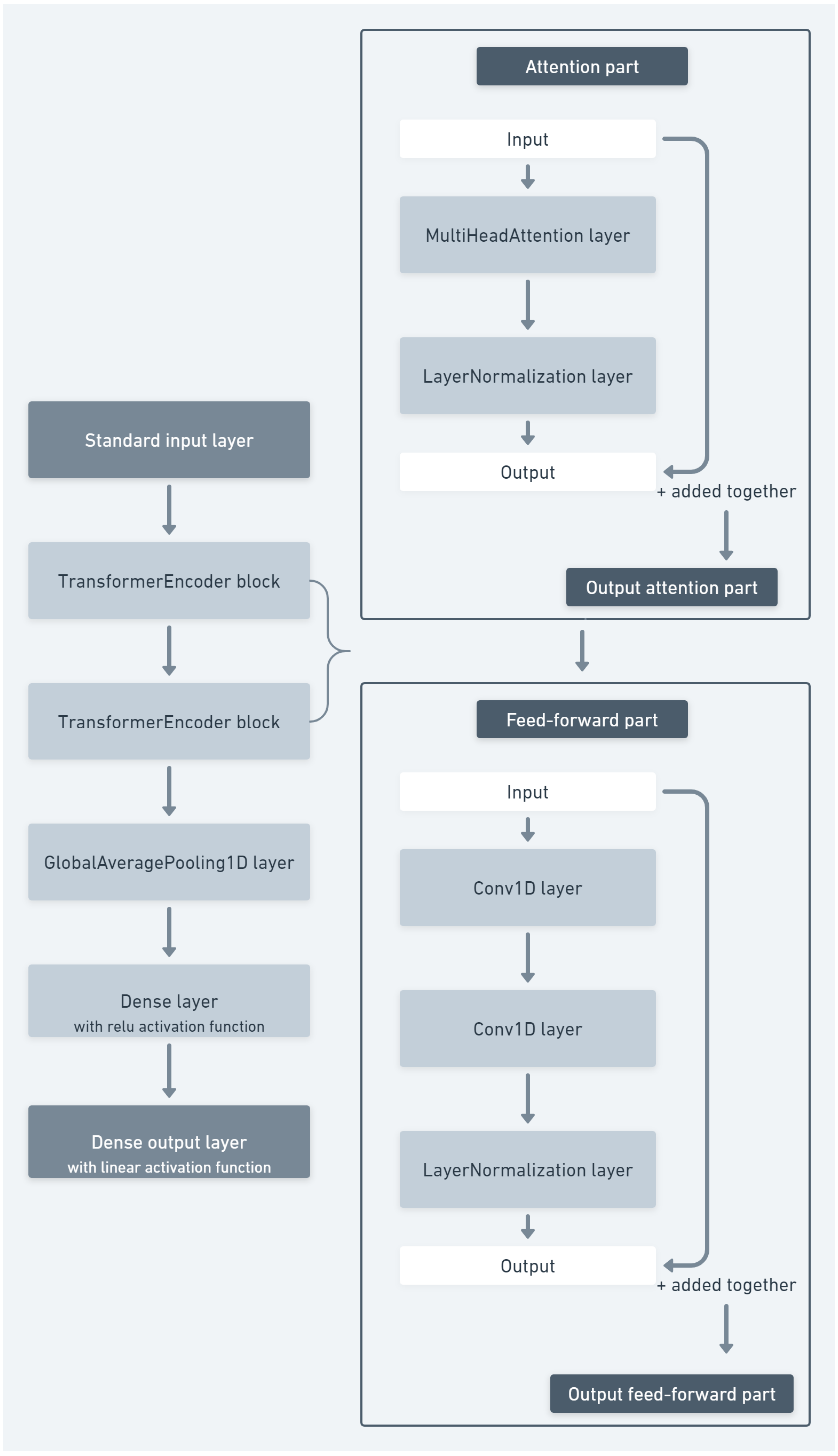

Figure 8: Layers of the Transformer model

5.4 Incorporating Uncertainty

In this thesis uncertainty is incorporated by introducing dropout to the three models. For both Simple RNN and Transformer dropout layers are added in between the different layers. Dropout can be applied to input neurons called the visible layer, but can also be applied to hidden neurons in the body of your network model. The dropout rate determines how many inputs are randomly excluded from each update cycle. The optimal dropout rate can considerably differ per model, the more a model overfits, the higher the optimal dropout rate is. The different dropout rates used for all three models are: 0.1, 0.2, and 0.35, meaning one in 10, one in five, and one in 2.86 inputs will be randomly dropped out with every dropout layer.

Within the LSTM model dropout is introduced differently than with the other two models. As explained before, every LSTM layer exists out of four sublayers, a (sigmoid) forget gate sublayer, a (sigmoid) input gate sublayer, a (tanh) sublayer to create vectors of the candidate values, and a(sigmoid) sublayer to decide on the final output. The trick of these sublayers is that they can all introduce dropout individually which is done by adding the dropout to the existing LSTM layer instead of adding separate dropout layers in between the LSTM layers.

For a structure such as a Simple RNN structure, there is no difference between the two options of adding dropout and also for Transformer dropout was introduced as a separate dropout layer.

6 Results

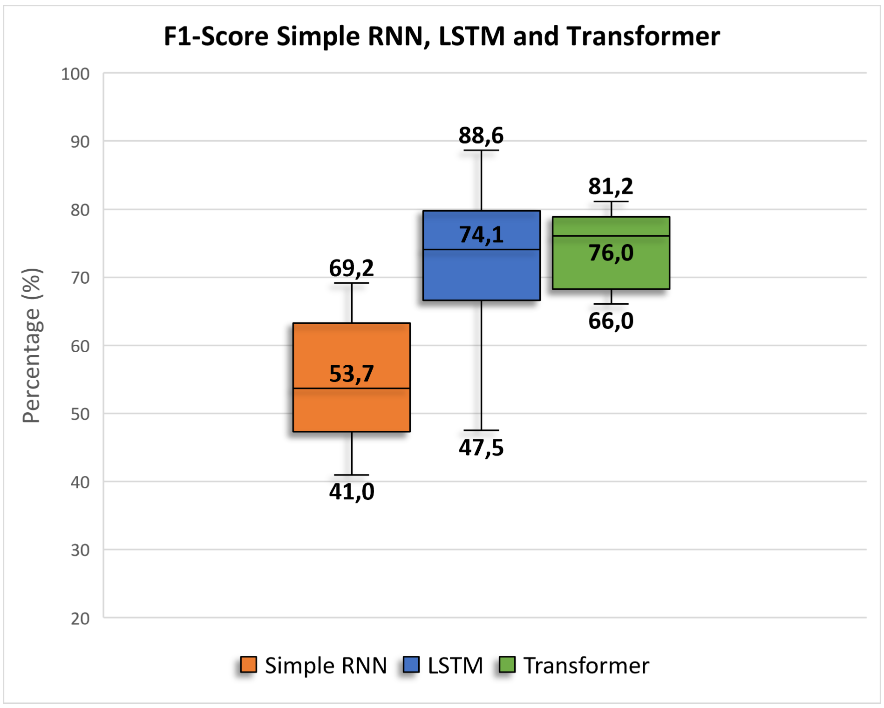

Figure 9: Overview of F1-scores for all methods, based on 10 runs per model

As shown in figure 9, LSTM and Transformer get significantly better results in comparison to Simple RNN. Where the average run of Simple RNN gets a F1-score of 53.7%, LSTM has an average run of 74.1%, and Transformer of 76.0%. The difference in scores of LSTM and Transformer can not so much be seen in the average run score, as they are very close, but is mostly visible in the span of the outcomes. The graph above is based on ten runs of every method, which means that for example for LSTM, the highest achieved run had a F1-score of 88.6% and the lowest was 47.5%. The difference between the highest and lowest achieving runs of the Transformer is way smaller i.e. 81.2% vs. 66.0%. As for Simple RNN, the highest was 69.2% which is already lower than the average run of the other two methods and the lowest was 41.0%.

Usually when a series of anomalies occurs (see figure 10), LSTMs predictions are on par with Transformer, and both are better (closer to yellow) than Simple RNN.

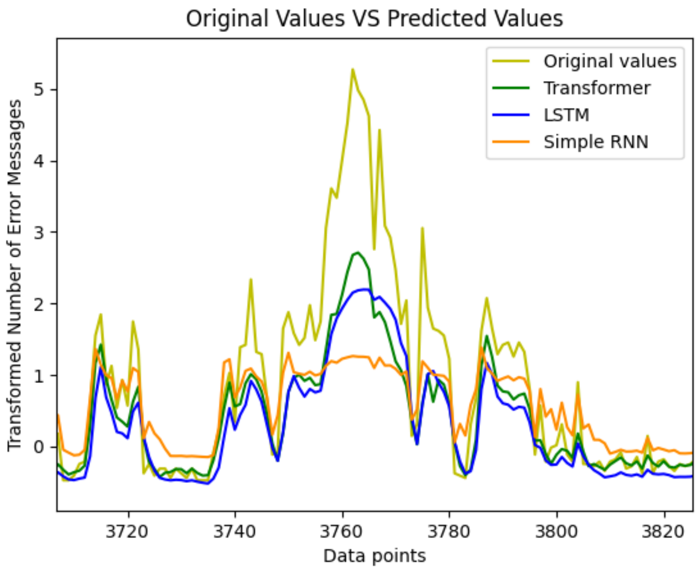

Figure 10: The original values and predictions of a series with anomalies

Most of the data does not contain any anomalies (only 128 out of 25139 data points are anomalies i.e. around 0.5% of the dataset). There is still some fluctuation in the number of error messages in these parts (i.e. see figure 11), but this is normal.

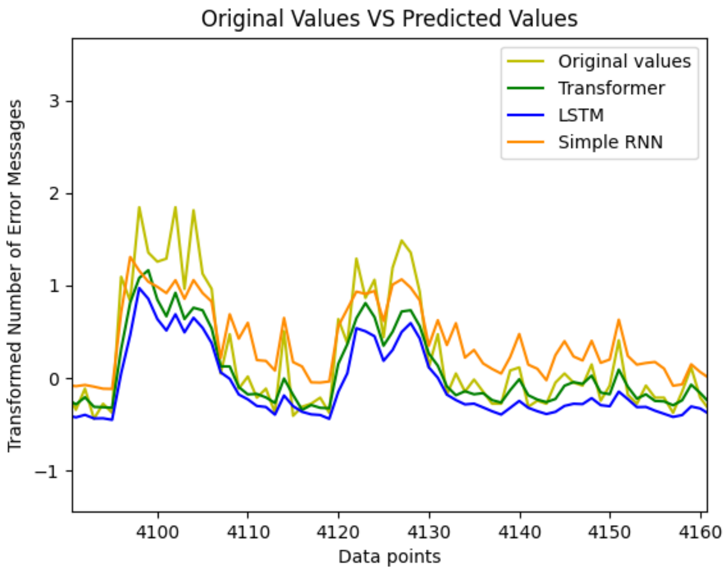

Figure 11: The original values and predictions of a series without anomalies

Lastly, there are instances where Transformer outperforms the other two models by far as can be seen in figure 12.

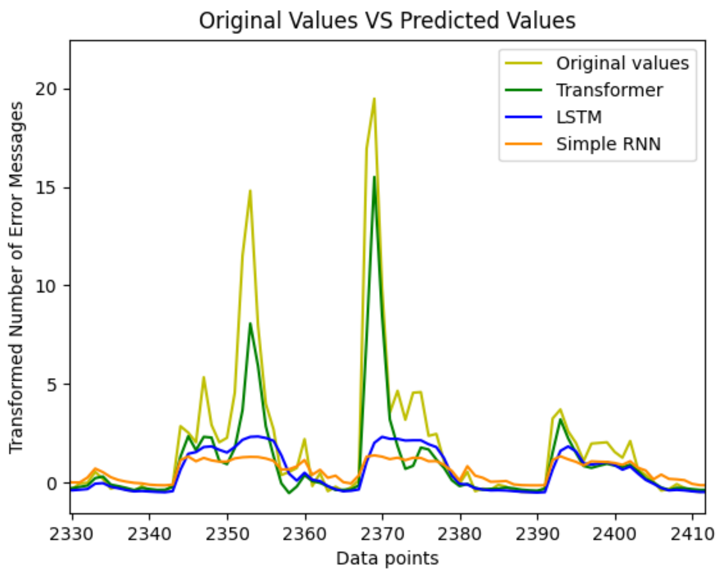

Figure 12: The original values and predictions of a series with anomalies where Transformer outperforms LSTM and Simple RNN

Solely comparing the performance on the predictions of the anomalies by adding the distance between the original transformed data point and the predicted transformed data point together for every anomaly, gives us table 13:

Difference Between the Predictions and the Original Values

Simple RNN	206.2
LSTM	211.6
Transformer	147.1

Figure 13: Total sum of the distance between the predictions and original (transformed) values of anomalies

Since the data points are transformed, these numbers are not representations of the exact number of error messages, but they do allow for comparison. The lower the number, the closer the predictions were to the original value of the anomaly, and thus the better the model predicted.

6.1 Results Incorporating Uncertainty

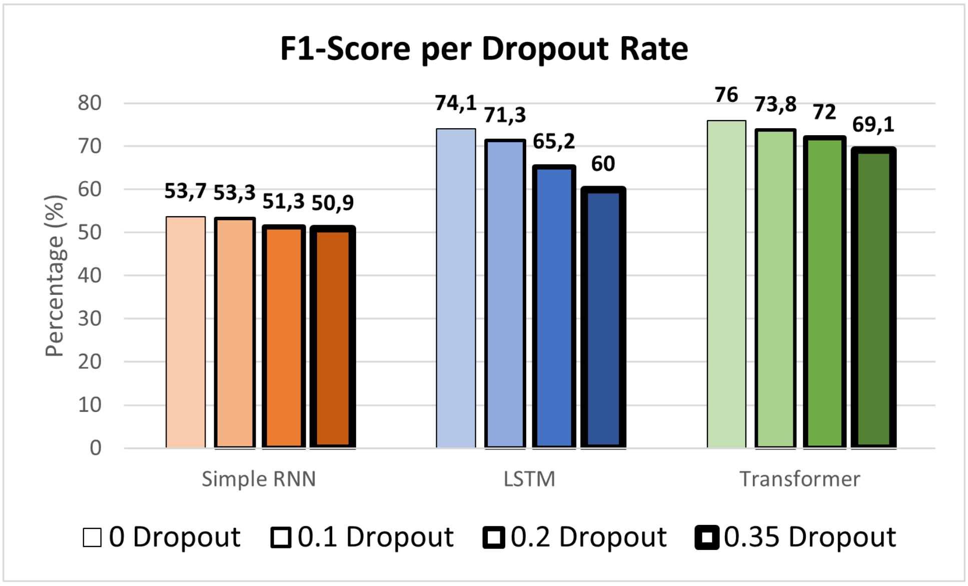

Figure 14: Overview of F1-scores for all methods, based on 10 runs per model per dropout rate

As for incorporating uncertainty using dropout, the results are poor. The best results were obtained with zero dropout and with every increase in the dropout rate, the results got lower.

6.2 Simple RNN

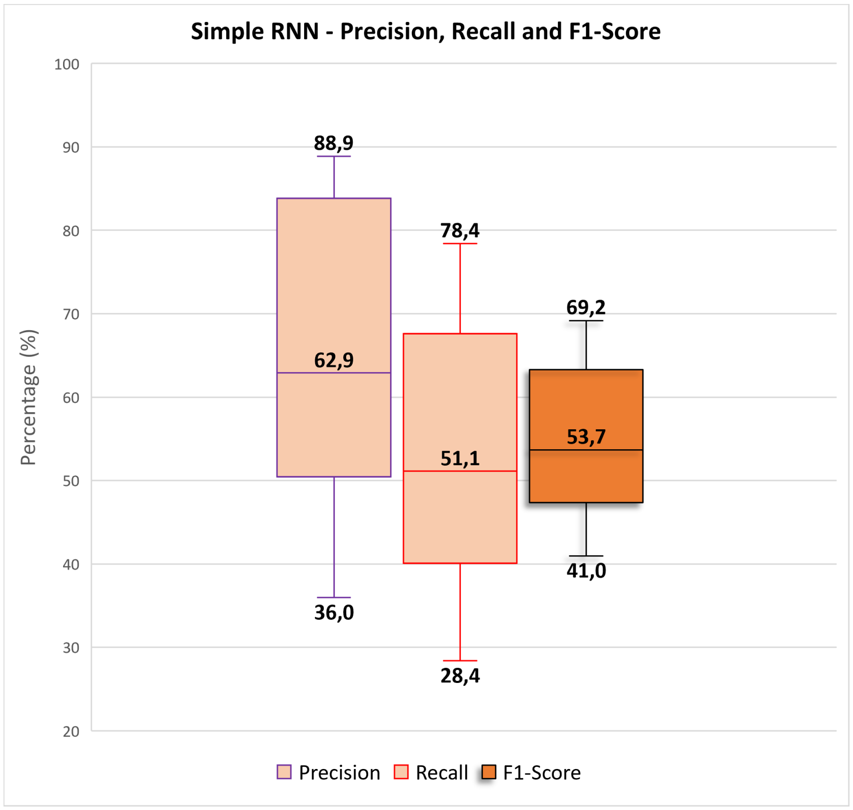

Figure 15: Results for Simple RNN based on 10 runs per model

Simple RNN has an average run of 53.7% in 125 seconds. Based on the results of the other models presented further below, Simple RNN had the lowest F1-scores of the three methods. There is quite the outcome span for both the precision and the recall which means that the scores of the run lay far apart, from 36.0% to 88.9% for precision and from 28.4% to 78.4% for recall. However the averages of precision and recall are not extremely far apart which would mean that the threshold was well chosen, but it could have been slightly better because the precision is both in best, average, and lowest run around 10% higher than the recall.

6.3 LSTM

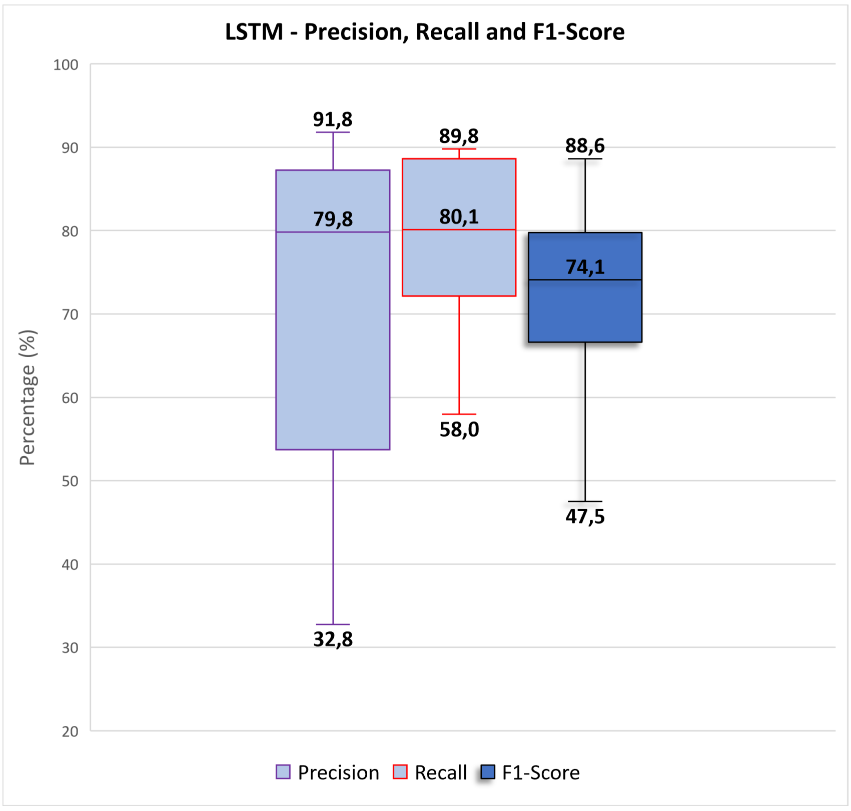

Figure 16: Results for LSTM based on 10 runs per model

With an average run of 74.1% in 252 seconds LSTM has gotten comparatively good results only being beaten by Transformer with a slightly higher average run of 76.0%. As for the precision and the recall, there is a very high outcome span, especially for the precision which has an outcome span of almost 60%. The averages however do get very similar scores on precision and recall indicating a well-chosen threshold.

6.4 Transformer

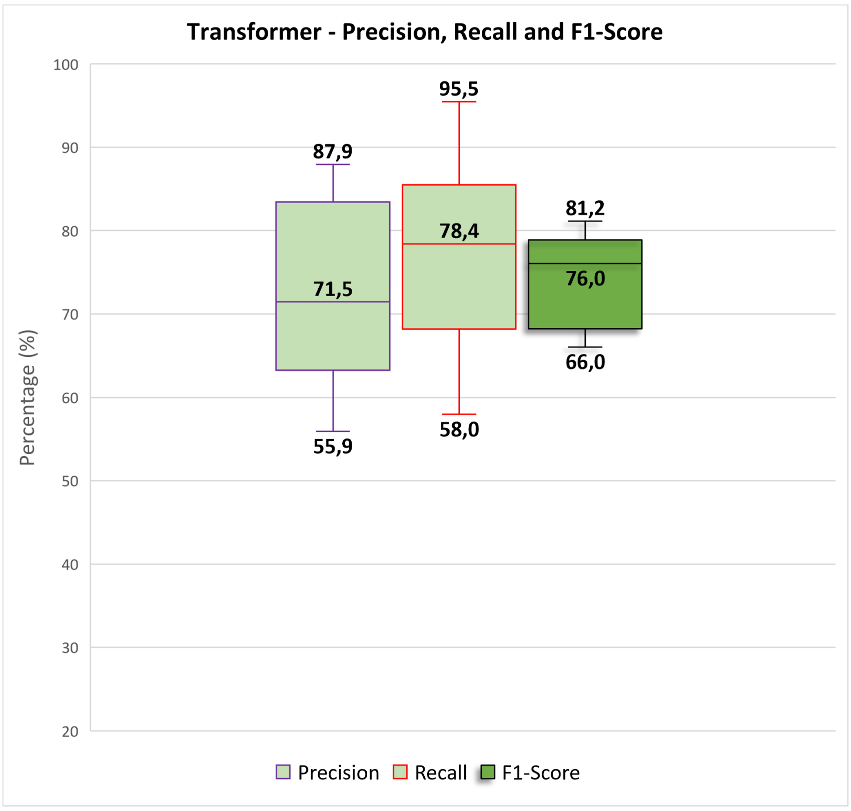

Figure 17: Results for Transformer based on 10 runs per model

With an average run of 76.0% in 102 seconds Transformer has gotten the best results out of the three methods. Overall for both precision and recall as well as the F1-score as a whole, there is a relatively small outcome span. Also, the averages of precision and recall are not very far apart, again indicating a well-chosen threshold.

7 Discussion

This section discusses the results of this thesis, structured per research question.

> First: How can we define an anomaly within error messages data from a hospital?
> Then: Between Simple RNN, LSTM, and Transformer, what is the most effective model for time series anomaly prediction based on the F1-score?
> And: Does incorporating uncertainty on top of anomaly prediction models improve their performance?

For a quick overview of the most important findings, please see the conclusion.

7.1 Define an Anomaly

Anomalies are exclusively defined as abnormal behaviour which results in unexpectedly high numbers of error messages. When this number of error messages is high enough to be seen as an anomaly is decided with the help of the Peaks Over Threshold (POT) method which labels data points as anomalies if they are bigger than a given threshold. Because of the difference in the normal number of error messages depending on the time and day, the threshold is dynamic which means that the anomalies are defined based on not only the number of error messages but also the time of the day and whether it is on a weekend or on a weekday. In total, around 0.5% of the whole dataset (128 out of 25139 data points) has the label: "anomaly" based on this dynamic threshold.

This POT method gives a lot of clarity on how data points are labelled as anomalies since the implementation is quite straightforward, but in reality, the dynamic threshold brought some problems along. Although the models, especially Transformer, are sufficiently good in finding the shapes, meaning: usually when the values go up in the original data, they also go up in the predicted values, the predicted values were often not high enough at the peaks. So to get the best results, the dynamic threshold had to be altered a bit. At first, we lowered the thresholds everywhere equally, but because the threshold was already very low in some places almost every score was named an anomaly by the models. The solution was lowering the dynamic threshold percentagewise which means that the high points of the dynamic threshold got lowered more where already low points of the dynamic threshold were lowered significantly less. This prevented the models from naming every value at those low dynamic threshold points an anomaly while lowering the threshold more for the peaks in the data.

7.2 Comparing the Models

Simple RNN getting the lowest scores is of no surprise since it is a way simpler model in comparison to the other two. Where Simple RNN is just a simple recurrent neural network, LSTM has the memory blocks and Transformer makes use of attention- and focus scores.

Between LSTM and Transformer, there is little difference in the F1-scores of their average runs. Transformer performs slightly better than LSTM, but because the scores of every run can vary quite a bit (this can be seen in the span of the outcomes), it could be that if LSTM predicts one round and Transformer predicts one round, LSTM performs better on that round. LSTM has gotten the highest-scoring prediction (88.6%) out of all runs from all models. Unfortunately, it is not a very stable implementation since it also has a run that scored 47.5%. Normally there is no labelled data which makes it impossible to check the performance of the model in that particular run and thus how reliable the predictions are. This unreliability of LSTM makes it a difficult model to work with.

A way more stable implementation is the Transformer model. The lowest score it has gotten is 66.0% and the average run has a higher score than LSTM, which makes Transformer a better method to work with. There is more assurance that the predictions are good and that this is not coincidentally a bad run which is more likely to happen with the unstable LSTM model. Not to mention, the Transformer model had a significantly shorter runtime.

One could say that if you run Transformer enough times it could also get a result well under 50% and that LSTM actually has a higher average run score if that one low score did not count which would then make LSTM a better method. However, during the runs performed to perfect the hyperparameters, it also occasionally occurred that LSTM got surprisingly low results whereas that with Transformer has not happened yet. This instability of LSTM is thus an actual problem and not a one-time thing that coincidentally occurred during the result runs.

Besides, when looking at the predictions of the anomalies Transformer was a lot closer to the actual values. Simple RNN scored 40.2% worse than Transformer and LSTM got even poorer results, scoring 43.8% worse than Transformer. This difference is massive, especially considering that the quantified measurement of performance (the F1-scores) of Transformer and LSTM are very similar. The comparison of the individual values added together gives results that are closer to what we can see on figure 12, i.e. Transformer being the superior model for this anomaly prediction task. It is however difficult to say if this way of comparing the performances of the models is better or worse than the quantified measurement method, but since for both Transformer was preferred, it is safe to say that for anomaly prediction on hospital error messages data Transformer is the best choice out of the three models.

All models could have better results if for every run we were able to use the optimal threshold for that run. While trying to find the best threshold for the models it seemed that the optimal threshold per run differed greatly, from 40% of the original threshold to almost 80% of the original threshold. This makes it impossible to use a threshold that works well for every run and the best solution was to take the threshold that on average gave the best results since we normally do not have labelled data which makes it impossible to calculate the optimal threshold per run. But when always using an optimal threshold the average run was well above the 80% for both LSTM and Transformer.

This set threshold can be altered in the code. By lowering the threshold the recall will be higher and the precision will be lower, by making the threshold higher the recall will be lower and the precision will be higher. The difference in the optimal thresholds per run is the cause for the large outcome span in the precision and recall. If the optimal threshold for that run is way lower than the set threshold, the recall is most likely to be very high and the precision very low. This also works the other way around. Because of this effect that the threshold has on the results, ChipSoft has all the freedom to alter the precision-recall ratio (the sensitivity) depending on what they think is important at that moment. When more people are working, the threshold can be put lower so more data points are labelled as anomalies. However, when there is a low capacity because employees are unavailable, the threshold can be put up so only predictions that are very high are named anomalies.

7.3 Incorporating Uncertainty

For these models under the current circumstances dropout only lowered the scores. This is most likely caused by underfitting. Most neural networks struggle with overfitting which is why dropout usually betters the performance, but if a model underfits dropout will only worsen the performance. Oftentimes when a model underfits it is because the data is too simplistic and adding features to the data could help. Expanding the dataset by adding more data to the trainingset is not a solution for underfitting. It might improve the results, but that is simply because there is more data, it does not help the model learn better from the data.

Minimalizing the underfitting could still increase the performance of the models, but there is also another explanation which might explain the poor results when using dropout. When working with pictures for example, every picture consists of many pixels. When applying dropout to those pixels you could improve the results by dropping some of them as this does not negatively affect the dataset. However, because we only have one data-input per data point, dropping one is simply reducing the size of the dataset. This means that there is less data to train on which only hinders the models from getting good results.

Expanding the dataset does considerably enhance the performances of the models which was shown when we changed the splitting ratio from 60% trainingset and 40% testset to an 80-20% ratio, the scores went up significantly (around 6%). As time goes by, the JIPs-melder will register more data which means that more data can be added to the trainingset. Also to compare the models we had to use a testset (of 5000 data points), but when only predicting the upcoming value, there is no need for a 5000 data points testset which means the data points can be added to the training set, improving the performance.

Another reason why dropout decreased the performance of the models could be: not training until convergence and having a network that is too small relative to the dataset.[14] However, for all three models the loss moves towards a minimum and thus stops decreasing. Of course, sometimes it slightly increases and then decrease a bit, but in general, not training until convergence does not apply. The same goes for the network complexity because adding more layers and/or increasing the number of units per layer or adding more TransformerEncoder blocks did not improve the results and just made the models slower.

Finally, the reason the performance of LSTM decreased more with a higher dropout

[14]`https://www.ibm.com/cloud/learn/underfitting`

rate compared to the other models, is because LSTM introduced dropout to all four of the sublayers and therefore introduced more dropout for the same rate.

8 Conclusion and Future Research

Here we first give a short overview of the most important findings of this thesis which are exhaustively explained and analysed in the discussion, then we examine further research possibilities.

8.1 Conclusion

1. How can we define an anomaly within error messages data from a hospital?

In conclusion, anomalies are defined through the Peaks Over Threshold (POT) method with a dynamic threshold. They are not only defined based on the number of error messages alone, but also on the time of the day and whether it is on a weekend or on a weekday. In total, around 0.5% of the whole dataset (128 out of 25139 data points) has the label: "anomaly" based on the dynamic threshold.

2. Between Simple RNN, LSTM, and Transformer, what is the most effective model for time series anomaly prediction based on the F1-score?

As for the models: Simple RNN got the lowest scores and LSTM and Transformer got similar F1-scores. Transformer had a slightly better average run, but LSTM got the highest-scoring prediction out of all runs from all models.
The preferred method out of the three in these circumstances is Transformer. Not only does it have the best average run score and the shortest runtime making it the fastest implementation of the three, it is also the most reliable method that gets satisfactory results on every run. On top of that, Transformer was also by far the closest to the original transformed data points when predicting anomalies. For a more in-depth analysis, see: Comparing the Models.

3. Does incorporating uncertainty on top of anomaly prediction models improve their performance?

In terms of uncertainty within the models, dropout only lowered the scores. Possible explanations for the decreasing results are underfitting and/or the singularity of our data (one data-input per data point). For more information, see: Incorporating Uncertainty.

8.2 Future Research

Besides using data with more features which would help with the underfitting of the models, there are other possibilities to better the performances of the models. One could try to combine the models with the STL implementation as combining STL with other models has been done before with anomaly detection and was quite successful. [Li et al., 2020] And since anomaly detection is quite similar to anomaly prediction, it could be profitable to also combine the models with the STL implementation for anomaly prediction.

It is also possible that a method besides Simple RNN, LSTM, and Transformer could work better, a possible option could be an ARIMA model or Time Series Forecasting which both have been reported to do well in anomaly detection. [Yu et al., 2016] [Mokhtari et al., 2022] As for the dropout, other techniques such as batch normalisation exist which could be used instead. Batch normalisation allows for much higher learning rates since it addresses internal covariant shift by normalising layer inputs. Internal covariant shift is a phenomenon best explained as follows: Because the distribution of each layer's inputs changes during training, as the parameters of the previous layers change, the training is slowed down because this needs careful parameter initialization and lower learning rates making it notoriously difficult to train models with saturating nonlinearities. [Ioffe and Szegedy, 2015] Internal covariant shift also acts as a regularizer so it could even completely replace dropout. However for this to work, first the underfitting of the models should be handled. Regularising an underfitting model will only make it perform less well.

Lastly, since ChipSoft provides software to a multitude of hospitals, it could be possible in the future to use the prediction models on hospitals where we do not have (enough) data from. If there is a large enough database with data from different hospitals along with data about the hospitals themselves. Then, if a new hospital is added, it might be possible to make predictions based on data from similar hospitals. It is very likely that this will not work in the first period after the hospital starts using the software because when working with a new (version of a) software system there are usually more difficulties resulting in more error messages and anomalies. This makes predicting very difficult since there is either not enough training data or the data is from a hospital that is already used to the software. However, after a while when there still is not enough data to make predictions, but the hospital is more stable with its use of the software, the data of other hospitals could be used. And as more data from the hospital itself is registered, the predictions will get better and fewer data from other hospitals will be needed.

References

[Abdar et al., 2021] Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., Fieguth, P., Cao, X., Khosravi, A., Acharya, U. R., Makarenkov, V., and Nahavandi, S. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297.

[Cho et al., 2014] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Jakubowski et al., 2021] Jakubowski, J., Stanisz, P., Bobek, S., and Nalepa, G. J. (2021). Explainable anomaly detection for hot-rolling industrial process. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10.

[Li et al., 2020] Li, Y., Bao, T., Gong, J., Shu, X., and Zhang, K. (2020). The prediction of dam displacement time series using stl, extra-trees, and stacked lstm neural network. *IEEE Access*, 8:94440–94452.

[Mathonsi and Zyl, 2022] Mathonsi, T. and Zyl, T. L. v. (2022). Multivariate anomaly detection based on prediction intervals constructed using deep learning. *Neural Computing and Applications.*

[Meng et al., 2019] Meng, H., Li, Y., Zhang, Y., and Zhao, H. (2019). Spacecraft Anomaly Detection and Relation Visualization via Masked Time Series Modeling. *2019 Prognostics and System Health Management Conference (PHM-Qingdao).*

[Mokhtari et al., 2022] Mokhtari, A., Ghorbani, N., and Bahrak, B. (2022). Aggregated traffic anomaly detection using time series forecasting on call detail records. *Security and Communication Networks*, 2022:1–9.

[Seoh, 2020] Seoh, R. (2020). Qualitative analysis of monte carlo dropout. *ArXiv*, abs/2007.01720.

[Sherstinsky, 2020] Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306.

[Song et al., 2018] Song, F., Diao, Y., Read, J., Stiegler, A., and Bifet, A. (2018). Exad: A system for explainable anomaly detection on big data traces. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 1435–1440.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Staudemeyer and Morris, 2019] Staudemeyer, R. C. and Morris, E. R. (2019). Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586.

[Tuli et al., 2022] Tuli, S., Casale, G., and Jennings, N. R. (2022). Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6):1201–1214.

[Wang et al., 2019] Wang, X., Zhao, T., Liu, H., and He, R. (2019). Power consumption predicting and anomaly detection based on long short-term memory neural network. In *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 487–491.

[Wen et al., 2022] Wen, Q., Zhou, T., Zhang, C., Chen, W., Ma, Z., Yan, J., and Sun, L. (2022). Transformers in time series: A survey. *ArXiv*, abs/2202.07125.

[Yu et al., 2016] Yu, Q., Jibin, L., and Jiang, L. (2016). An improved ARIMA-based traffic anomaly detection algorithm for wireless sensor networks. *International Journal of Distributed Sensor Networks*, 12(1):9653230.

[Zhang et al., 2021] Zhang, C., Sun, J., Lu, R., and Wang, P. (2021). Anomaly detection model of power grid data based on stl decomposition. In *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, volume 5, pages 1262–1265.

[Zhu and Laptev, 2017] Zhu, L. and Laptev, N. (2017). Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110, Los Alamitos, CA, USA. IEEE Computer Society.