Erasmus University Rotterdam
Erasmus School of Economics
Master Thesis Quantitative Finance

Machine Learning in Time Series Forecasting: A Comparative Analysis of LSTM and GRU Ensembles

Ariel Levi (653596)

Supervisor:	A. Alfons
Second assessor:	P. Vallarino
Date:	9th April 2024

Abstract

This thesis investigates the comparative performance of Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models within an ensemble framework for prediction of insurance data, focusing on both claim numbers and amounts. The study utilizes various configurations of ensemble models to assess their forecasting capabilities. Through multiple Diebold-Mariano tests, significant differences between the two ensemble models are observed, yet both exhibit competitive forecasting abilities under different scenarios. This underscores the effectiveness of ensemble methods in addressing individual model weaknesses, thus providing valuable insights into their comparative analysis.

Acknowledgements

I would like to express my deepest gratitude to Charl Marais, who served as both my company supervisor for this thesis and manager during my internship at Allianz Benelux. I am deeply grateful for his unwavering support and guidance throughout this transformative journey. Additionally, I extend my sincere thanks to my esteemed colleagues, Erwin van Dongen and Feysel Negash, for their invaluable contributions and mentorship, which have greatly enriched my research experience.

Special thanks are due to my university supervisor, associate professor Andreas Alfons, whose exceptional analytical insights and expertise in machine learning have been a constant source of solid guidance.

Additionally, I express my heartfelt appreciation to everyone within Allianz Benelux who assisted me with my work. The support and collaboration of each colleague have been integral to my professional growth and the completion of this thesis.

Contents

1 Introduction **3**

2 Literature Review **5**

3 Methodology **9**

4 Data **18**

5 Results **23**

6 Discussion **31**

References **32**

A Property SME Data **36**

B Diebold-Mariano Test **38**

C PW Visualizations - Midcorp **40**

D Additional Results - Sensitivity Check **41**

E Hyperparameter Optimization **45**

1 Introduction

Throughout the years the topic of time series forecasting has been a hot topic within the scientific community as developments in forecasting can lead to a better view of the future. Having a better understanding of the future has always fascinated human beings, with ancient cultures worldwide having stories of prophets being mediators between an institution who has full knowledge of the future and the people who have limited knowledge of future events. In the real world, forecasting gets its application by being used by meteorologists, logistics experts and many other professions around the world. A prevalent example of time series forecasting being able to make a difference, is in the business world (Poll, Polyvyanyy, Rosemann, Roglinger & Rupprecht, 2018). Businesses usually have lots of different time series datasets which need to somehow be extrapolated into the future. This can be problematic because not every employee is a time series professional and there is not always time to choose between all the various different models if the amount of time series needed to be forecasted is very large. That is why in these situations the employees will usually pick one model and use it for most if not all of their forecasting needs (Taylor & Letham, 2018).

Out of all the recent developments in time series forecasting, most of the significant ones seem to be stemming from Machine Learning (ML), which diverge from traditional time series forecasting approaches by automating feature extraction and selection, managing complex patterns and relationships while also addressing non-linearity and handling non-stationarity ((Hall, 2018), (Bala & Singh, 2019)). These methods adjust to abrupt changes and irregularities due to their adaptability, while their iterative learning process enhances generalization. Additionally, ML requires more substantial datasets, while traditional statistical methods remain competitive with smaller data amounts (Makridakis, Spiliotis & Assimakopoulos, 2018b).

A subspace of ML is Neural Networks ((McCulloch & Pitts, 1943), (Rosenblatt, 1958)), known for its exceptional capabilities to capture non-linear relationships through multiple hidden layers, which enables them to excel in tasks ranging from image recognition to natural language processing ((Krizhevsky, Sutskever & Hinton, 2012), (Sutskever, Vinyals & Le, 2014)). In a time series setting these Neural Networks are adapted such that they can be iterated over time series observations, these Neural Networks are therefore called Recurrent Neural Networks (RNN)

(Elman, 1990). While various forms of RNNs exist, the most commonly used in practice are Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) (Jozefowicz, Zaremba & Sutskever, 2015), with the latter bearing a close resemblance to the former, although having a somewhat more simplified cell structure.

In an ensemble framework, forecasts significantly improve with a notable reduction in variance due to the aggregation of predictions from diverse base models, which collectively smooth out individual forecasting errors (Livieris, Pintelas, Stavroyiannis & Pintelas, 2020). Additionally, bias can be effectively managed through the application of regularization methods (Krogh & Hertz, 1991). It is notable however that it has not been investigated how the performance differs in an ensemble setting of multiple LSTMs vs multiple GRUs. While the performance difference of a single LSTM and a single GRU can be statistically insignificant (Chung, Gulcehre, Cho & Bengio, 2014), it could turn out to be significant in this ensemble setting. Additionally, the difference in computational time will be examined, as this is crucial in practical situations where time and computer resources are limited.

As such the research question being investigated will be: ***"To what extent does LSTM and GRU model performance differ in a memory ensemble setting?"***. This will be achieved through a systematic analysis of both LSTM and GRU models within an ensemble framework. The investigation will encompass data analysis, hyperparameter tuning, model training, and rigorous evaluation to determine the efficiency and accuracy of LSTM compared to GRU ensembles in time series forecasting. Emphasis will be placed on forecasting accuracy and ensemble weights to conduct a comparative analysis.

The thesis will be written in the setting of time series forecasting, focusing on a business framework. This research is conducted in collaboration with Allianz, Europe's biggest insurance firm, which has provided access to a vast amount of time series datasets from the data team within the Allianz Benelux Property division. The availability of a dedicated virtual client will facilitate the execution of heavy computational tasks. Regarding the thesis structure, a comprehensive literature review will be presented in Chapter 2, detailing prior research in this field. Following this, Chapter 3 will thoroughly explain the ML techniques applied. The data utilized for model analysis will be introduced in Chapter 4. Subsequently, Chapter 5 will display and analyze the findings of this research. Finally, the results will be extensively discussed and contextualized in Chapter 6.

2 Literature Review

Time series forecasting has a rich history, evolving from simple methods to sophisticated models. Initially, techniques like moving averages were fundamental, involving average calculations over a specified period to predict future values. Such methods, despite their simplicity, laid the groundwork for more advanced approaches. The late 19th and early 20th centuries witnessed a significant leap with the development of linear regression techniques (Molnar, Casalicchio & Bischl, 2020). These methods introduced the concept of fitting historical data trends by error minimization to predict future outcomes, a principle that remains central to every forecasting model that followed such as the Autoregressive Integrated Moving Average (ARIMA) (Box, Jenkins & Reinsel, 2015) and later the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models (Bollerslev, 1986). Through these monumental developments lots of variations were created all with different characteristics and advantages.

The transition to ML in the forecasting space served as a paradigm shift (Alzubi, Nayyar & Kumar, 2018). This development can mainly be attributed to the increasing complexity and volume of data in various domains in combination with the exponential growth of computational power (Fradkov, 2020). This shift necessitated models that could handle large datasets and capture complex, non-linear relationships, which traditional models like ARIMA and GARCH were less equipped to manage. As such, the evolution of ML models to capture non-linear relationships in complex datasets led to immense improvements in accuracy ((Molnar et al., 2020), (Fradkov, 2020)). These advancements consequently led from that point on for forecasting competitions to include lots of models which make use of ML (Makridakis, Spiliotis & Assimakopoulos, 2018a), with that consolidating its presence in the forecasting space.

There are many different ML methodologies available for time series forecasting, each with their own unique approaches. Gaussian Processes (GP) based models deal more with probabilistic forecasting, making them valuable for scenarios where understanding the range and distribution of future possibilities is as crucial as predicting a single outcome (Schulz, Speekenbrink & Krause, 2018). XGBoost and LightGBM on the other hand are tree-based models which focus more on handling large feature sets well by aggregation of simpler decision trees ((Chen & Guestrin, 2016), (Ke et al., 2017)). Support Vector Machines excel at classification by fitting

an optimal hyperplane in a high-dimensional space (Hearst, Dumais, Osuna, Platt & Scholkopf, 1998). The main class of ML models which seem to perform really well on time series, especially financial time series (Fischer & Krauss, 2018), are some sort of the aforementioned Recurrent Neural Network (RNN), which are Neural Networks that have been adapted for time series using sequential data and other techniques (Elman, 1990). The problem with vanilla RNNs is that with lengthy time series they suffer from the vanishing or exploding gradient problem, which make it fail to incorporate long-term dependencies (Pascanu, Mikolov & Bengio, 2013). Therefore the Long-Short Term Memory (LSTM) model was created by Hochreiter and Schmidhuber (1997). This model managed to solve both the vanishing and exploding gradient problem by splitting the stored information entering the RNN cell into long and short term memory by adding gates into the cell which control the flow of information.

Recent advancements in these ML forecasting methods have significantly improved forecasting accuracy. However, they do come with limitations, particularly in their practical application (Molnar et al., 2020). One major challenge is the tuning of hyperparameters, a complex and time-consuming task, especially if the ML method employed has lots of hyperparameters ((Snoek, Larochelle & Adams, 2012), (Kaastra & Boyd, 1996)). Additionally, training the model can take a very long time when leveraging the fact that ML techniques benefit from an ensemble approach to minimize the variability inherent in single-model forecasts, necessitating repetitive training of models to create ensembles ((Guan & Plotz, 2017), (Borovkova & Tsiamas, 2019)). This issue is compounded for users who want to use ML without access to advanced CPUs or cloud computing (Han, Mao & Dally, 2015). Another significant obstacle of ML models is their "black box" nature (Lipton, 2018), which obscures the understanding of how input variables are transformed into predictions mainly due to the lack of transparency in their internal decision-making processes and intricate computational dynamics.

To address the challenge of model complexity, an RNN model based on LSTM, known as the Gated Recurrent Unit (GRU), was introduced by Chung, Gulcehre, Cho and Bengio (2015). The main difference between LSTM and GRU lies in GRU having two gates which control the flow and storage of the time series, while LSTM has three of these gates. With this modification come advantages such as simplicity, but most importantly, does it reduce the amount of model parameters, which reduces training durations (Chung et al., 2015). The interesting thing about this simplification is, as discussed in the introduction, that there is no clear consensus on whether it reduces forecasting accuracy or not (Di Persio & Honchar, 2017).

Another huge development which must be mentioned when discussing the feasibility of LSTMs and GRUs is the development of transformers (Vaswani et al., 2017) which is now the state of the art deep learning method. Its performance seems to be unmatched and is now used among all aspects of our lives, ranging from NLP to large language models like GPT-4 and BERT (Rahman & Watanobe, 2023). Due to this phenomenon RNN based models have been very useful in combination with transformers, such as in Dai et al. (2019) where RNN attributes help with various aspects of improving upon vanilla transformers, which returns us to RNNs. Throughout the times various extensions of the LSTM model have made their way, like the bidirectional LSTM (Schuster & Paliwal, 1997) which takes the dynamics from both flows of time into account for training model parameters and also the peephole connection LSTM (Gers, Schmidhuber & Cummins, 2000) which improved capturing long term dependencies. What all of these extensions or flavours of LSTM have in common is that they complicate LSTMs, where GRU is an alteration that simplifies it instead (Chung et al., 2015). This is why GRU is such a big topic in current RNN forecasting papers and continues to be investigated and compared against LSTMs ((Di Persio & Honchar, 2017), (Hewamalage, Bergmeir & Bandara, 2021)).

Hence it is not a surprise that comparative analyses between LSTM and GRU models are a big topic within RNN forecasting papers. The relationship between these two seems to be nuanced with each paper adding some new insights. The first research paper on this topic by Chung et al. (2014) compares non-gated with gated RNNs and concludes that gated RNNs perform much better than non gated RNNs. The performance differences between LSTMs and GRUs are suspected to be task dependent however. Later Jozefowicz et al. (2015) found through their analysis of LSTM variations that GRU performs better than LSTMs on a plethora of tasks, except for LSTM managing to outperform on NLP tasks. In a comparative analysis for Google stock returns Di Persio and Honchar (2017) found that LSTM outperform GRU models when dropout is introduced, but without dropout GRU models again outperform LSTM models. All this indicates that the simpler architecture of GRUs does not necessarily mean less performance with even the opposite holding true in some cases. While it seems that the models are somewhat similar, showing their differences empirically can be quite a task and continues to be a field of research until this day.

Almost every single ML model, including LSTMs and GRUs, requires careful optimization of many different hyperparameters, with their optimization being essential for the model's performance (Snoek et al., 2012). Hyperparameter tuning is the answer to this challenge. While grid search is a common and traditional method, newer techniques like Bayesian Optimization and Random Search offer more efficient ways to find the best settings (Yu & Zhu, 2020). Bayesian Optimization iteratively updates the hyperparameters after each test (Snoek et al., 2012), and Random Search tries random combinations, which can be surprisingly effective (Bergstra & Bengio, 2012). However, despite these advancements, grid search remains widely used in practice (Yu & Zhu, 2020).

An approach used in combination with hyperparameter tuning is called ensembling which combines multiple models, such as LSTMs, with a range of important hyperparameters or another model characteristic, to generate diverse forecasts. These forecasts are then aggregated to produce a more accurate final prediction, as discussed by Borovkova and Tsiamas (2019). Furthermore, the paper implements the use of multiple LSTM models regarding different subsets of data to let the models capture different aspects of the time, this is then weighted by the recent performance of the ensemble parts. This methodology effectively mitigates the weaknesses inherent in individual models and reduces variance, which leads to enhanced and more robust overall performance. Now, with having discussed the long history and recent developments within the space of ML and RNN forecasting, it is time to discuss the methodology behind the conducted research and what new techniques are implemented.

3 Methodology

The foundation of this research lies in the core premise of the Neural Network (NN), the most basic form of NNs is the single hidden layer feedforward NN (Rosenblatt, 1958) which consists of three different layers, the input layer which receives the data, the hidden layer which processes the data and the output layer which provides the final result. Each of these layers consists of separate neurons, also called nodes. The strength of the connection between two nodes are determined through weights, these weights alongside their respective biases need to be trained to minimize the difference between the predictions and the realized values, this is done using a loss function during the process of backpropagation.

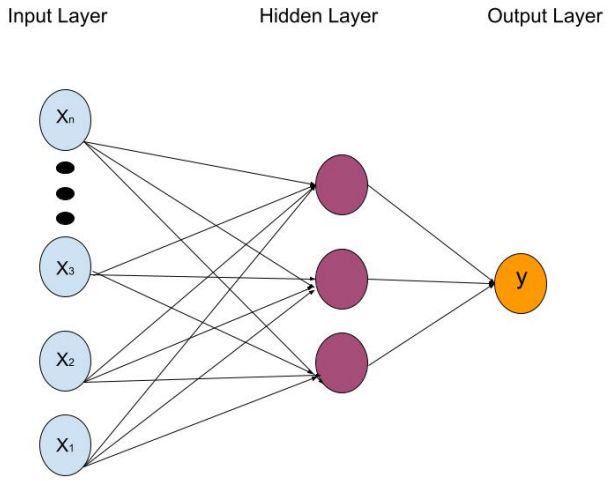


Figure 3.1: Single hidden layer feedforward NN

To demonstrate the single layer feedforward NN, Figure 3.1 is displayed with three hidden layer neurons. Deconstructing this figure mathematically there is the input layer which consists of n inputs, here represented as input vector $\mathbf{X} = [x_1, x_2, \ldots, x_n]^T$ then each connection has its own weight represented as a weight matrix $\mathbf{W^{(1)}}$ with entries such as $w_{1,1}^{(1)}$ and $w_{1,2}^{(1)}$ to display the connection from x_1 to the first and second hidden neuron respectively. Additionally each neuron has a bias $\mathbf{b^{(1)}} = [b_1^{(1)}, b_2^{(1)}, b_3^{(1)}]$, from which the hidden layer transformations can be written as $\mathbf{H} = f(\mathbf{W}^{(1)T}\mathbf{X} + \mathbf{b}^{(1)})$ where $f(.)$ is an activation function taken of the linear transformation

of $\mathbf{X}$. Note that the choice of activation function depends highly on the problem, which will be discussed in the next section. Finally the output can be calculated as $y = g(\mathbf{W}^{(2)T}\mathbf{H} + \mathbf{b^{(2)}})$ where $g(.)$ is another activation function and $\mathbf{W}^{(2)T}$ and $\mathbf{b^{(2)}}$ present the bias vector and weight matrix regarding the output connections.

The main method of calculating these biases and weights is through an algorithm called backpropagation (Rumelhart, Hinton & Williams, 1986). Backpropagation is a cornerstone algorithm in NN training, which operates through a two-phase process. Initially, in the forward pass, data flows from the input to the output layer, culminating in a prediction. During this stage, the prediction is compared to the realized values, and the error is calculated using a designated loss function, which in our case will be the Mean Squared Error (MSE). The backward pass, integral to backpropagation, involves the calculation of the loss function's gradient with respect to each weight. This is accomplished using the chain rule of calculus, enabling the gradient, and thus the error signal, to be 'propagated' backwards through the network. This process guides the adjustment of weights, aiming to minimize the prediction loss. Consequently, through iterative improvements, backpropagation allows the NN to learn from its missteps and incrementally enhance its prediction accuracy. This process is these days simply known as training a NN model or just training.

Weights and biases form the basis of linear transformations of NNs however they do not capture non linearity which NNs are so famous for. That happens through the activation functions which, in the hidden layers, are usually chosen to be non-linear to capture complex relationships in the data. The two activation functions most commonly used for RNN forecasting are the sigmoid activation function (3.1) and the hyperbolic tangent activation function (3.2).

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

While these may look like complex mathematical functions, their interpretation is very straightforward. The sigmoid activation function transforms the input x to a value between 0 and 1 which can be seen a filter to its input. The hyperbolic tangent function transforms x to a value between -1 and 1, which can be seen as a type of non-linear normalization to inputs.

The most basic type of NN adjusted for time series is the vanilla RNN (Figure 3.2), also called the Elman RNN (Elman, 1990), which works with NN layers being connected through a feedback loop making the network recurrent. This adjustment fits perfectly in a time series setting, because now each data point gets their own cell which has input nodes from the hidden state of the previous cell h_{t-1} and the current new observation x_t, which is a sequence of predefined length. The newly produced hidden state, h_t can serve as either the forecast for the next observation or as input for the next cell. The fact that RNN models process sequential data, enables them to capture temporal dependencies.

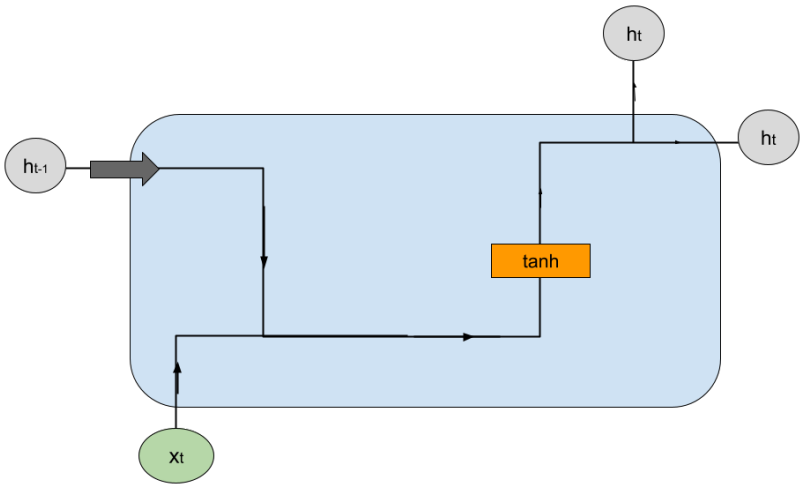


Figure 3.2: RNN cell

It is easy to grasp how the previous hidden state relates to the current hidden state through the formula $h_t = \tanh(\mathbf{W_{hh}}h_{t-1} + \mathbf{W_{xh}}x_t + b_h)$ where $\mathbf{W_{hh}}$ and $\mathbf{W_{xh}}$ are the weight matrices of the connection between the current hidden state and the previous hidden state and the connection between the current hidden state and the current input respectively. h_t and h_{t-1} are the current and previous hidden state and there is a bias term b_h. Finally when a prediction needs to be made the hidden state goes through another weight and bias related to the output: $y_t = \mathbf{W_{hy}}h_t + b_y$.

The method's limited utility, as mentioned before, stems from the vanishing gradient problem (Pascanu et al., 2013). This phenomenon arises when the weights associated with distant observations in the dataset's historical context become progressively diminished, resulting in their influence on the current hidden state h_t to vanish. This results in the RNNs being unable to retain information over a large number of time steps, which makes it unable to capture patterns

that span long over time. The cause of this problem lies in the partial derivatives calculation of the aforementioned backpropagation process. For RNNs the gradient depends on products of derivatives of the tanh activation function across many time steps, which if they are all smaller than one, their product will edge toward zero. It is also important to mention that when the gradient compounds in the opposite direction, the furthest observations have an extreme impact on the hidden state, this phenomenon is referred to as the exploding gradient problem.

The LSTM model (Hochreiter & Schmidhuber, 1997) (Figure 3.3) solves these problems of the vanilla RNN by introducing a different way to control the flow of information from the past. The main way it does this is by splitting the memory into short term memory, which will still be called the hidden state h_t, and long term memory which is called the cell state c_t where t is the current time step. The flow of memory is controlled through three gates:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.4}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.5}$$

the forget gate f_t, which controls how much of the previous cell state should be retained or discarded. The input gate i_t, which manages the flow of new information from the current input into the cell state. Finally there is the output o_t gate, which decides what amount of information from the updated cell state should contribute to generating the hidden state and the final output of the LSTM unit. Additionally in the above formulas W_f, W_i, W_o and b_f, b_i, b_o represent the weight matrices and bias vectors of the respective forget, input and output gates. Important to note is that usually the input gate is seen as two units in one with the cell state update split, defined by

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.6}$$

With $\tilde{C}_t$ being the candidate cell state while W_C and b_C denoting the respective weight matrix and bias vector of the candidate cell state. This candidate cell state is then moderated by the input gate which then results in cell state C_t of the current cell in time step t.

$$C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t \tag{3.7}$$

C_t and the output gate are then used for calculation of the hidden state of the current time step

h_t

$$h_t = o_t \circ \tanh(C_t) \tag{3.8}$$

where the current hidden state and cell state will be used for the calculations of the next LSTM cell in time step $t + 1$. Note that in Equations 3.8 and 3.7 $\circ$ denotes the Hadamard product.

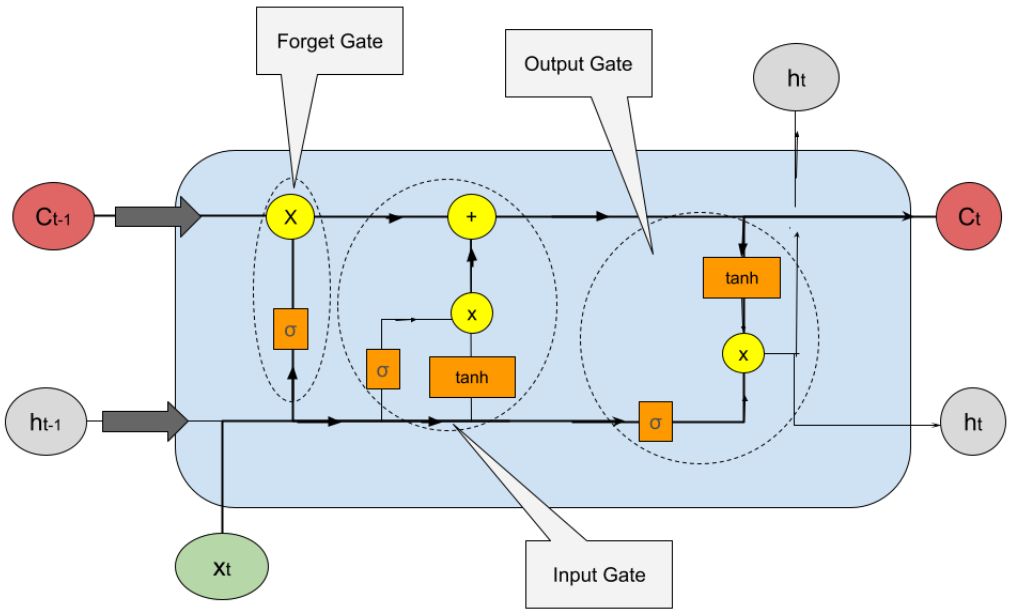


Figure 3.3: LSTM cell

The GRU model (Chung et al., 2015) (Figure 3.4) tackles the issue of the LSTM model being quite complex and simplifies it while still keeping its core elements. The first major change has been that the cell state has been removed which means that the hidden state h_t in a GRU setting contains both long and short term memory. Furthermore, significant transformations have been applied to the gate architecture, manifesting in the presence of two pivotal gates:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{3.9}$$

$$u_t = \sigma(W_u \cdot [h_{t-1}, x_t] + b_u) \tag{3.10}$$

The reset gate r_t helps the model decide how much of the previous hidden state should be forgotten or reset based on the current input and the update gate u_t which controls the balance between incorporating the new input and retaining the previous hidden state. In the above formulas W_r, W_u and b_r, b_u stand for the weight matrices and bias vectors of the reset and

update gate respectively. Next the candidate hidden state $\tilde{h}_t$ and with that the current hidden state h_t are determined by:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \circ h_{t-1}, x_t] + b_h) \tag{3.11}$$

$$h_t = u_t \circ h_{t-1} + (1 - u_t) \circ \tilde{h}_t \tag{3.12}$$

where the current hidden state could be seen as a weighted average between the previous hidden state and the candidate hidden state with the weights being u_t which ranges from 0 to 1 because of the sigmoid activation function. Note that for the above Equations 3.11 and 3.12 , W_h and b_h are the weight matrix and bias vector relating the candidate hidden state. Additionally $\circ$ denotes again the Hadamard product.

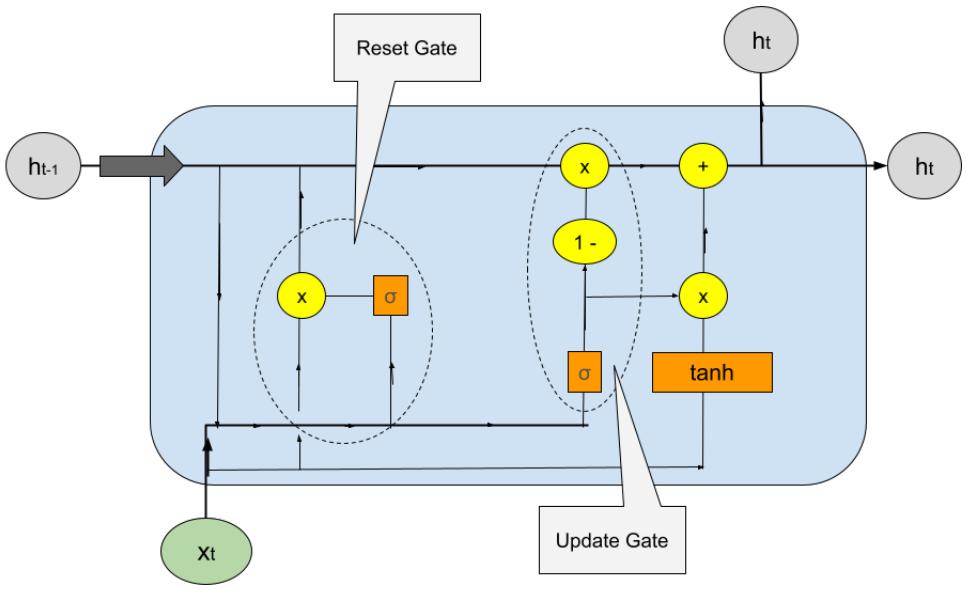


Figure 3.4: GRU cell

3.1 Ensembling

The use of LSTM ensemble techniques, as cited from Borovkova and Tsiamas (2019), is expected to further highlight the differences between LSTM and GRU models, providing a nuanced understanding of their respective strengths and limitations in forecasting applications. However unlike that paper which uses ensembles with varying time horizons and features, the ensemble

techniques here will make use of taking a range of a selected number of nodes.

The number of nodes within an RNN framework plays a slightly different role than in traditional NNs, where the amount of nodes is more a sign of model complexity, and for RNNs, where it is more of an indication of how much memory the model can store. Choosing an optimal number of nodes can be interpreted as a bias-variance tradeoff. The bias-variance tradeoff mainly involves adjusting a model to decrease bias at the cost of increasing variance, or reducing variance at the expense of increasing bias. When the bias is too high the model will not be able to properly capture the underlying data. Conversely, a model with excessive variance may learn the training data too closely, also known as overfitting, which results in poor performance on unseen data. As such, choosing a high number of nodes and thus remembering too much unnecessary information can lead to overfitting however it is still necessary for the model to remember important characteristics and patterns at least to some degree. This is how a plethora of models with different memory capabilities can paint a nuanced picture on the workings of LSTM and GRU ensembles, with each model capturing a different aspect of the data based on its memory capabilities.

To have an ensemble that fully captures different aspects of memory an exponential range of values has been used. While there are no fixed rules on the amount of models within an ensemble, it was deemed appropriate to go for a wider range with not too many models to ensure that each model covers different aspects of the forecast. Subsequently, the number of nodes for the ensemble setup have been set to 8, 16, 32, 64 and 128. Here it was made sure that each model has twice as much memory which guarantees variation within the ensemble. Besides the LSTM or GRU layer, the models also consists of two dense layers and an output layer, with the dense layers helping to capture data specific dynamics.

So the LSTM and GRU ensembles will contain a varying number of nodes but how will their weights be determined? For this question the most straightforward answer is the equally weighted (EW) ensemble. Since for this research the ensemble consists of five models the weights are all 0.2 since $w_8 = w_{16} = w_{32} = w_{64} = w_{128} = \frac{1}{5} = 0.2$ where w_x is the ensemble weight of a model with x nodes in its LSTM or GRU layer. This ensemble weighting ensures that the weight of each model is equal to eachother which has proved to result in competitive results despite its simplicity (Borovkova & Tsiamas, 2019). One other popular method is some sort of performance weighted (PW) ensemble, here the weights are adjusted based on the performance of the last

period in time. The way it is implemented in this LSTM and GRU setting is by adjusting the weight of each individual model based on its forecasting accuracy of the previous year in relation to the performance of its peers. Last year's data is also called the validation data, which will be discussed more in depth in the next section. So in notation, for a performance weight of a model with x nodes: $w_x = \left(\frac{\text{RMSE}_x}{\Sigma_i \text{RMSE}_i}\right)^{-1}$, where RMSE is a measure for forecasting accuracy described below (3.14).

An additional technique employed due to extensive data resources is called Transfer Learning (TL) which is a method that enriches model dynamics by pre-training models using datasets with similar characteristics (Weiss, Khoshgoftaar & Wang, 2016), in our case this will be done by pre-training the LSTM and GRU models with comparable data from a different source, after which they will be continued to be trained and forecasted on the original data sources. This technique will allow the models to extract features from the comparable data used for pre-training, which contributes to the robustness and adaptability of the models if the models manage to capture relevant features from the TL data.

The following section explores the steps taken to optimize model training with scheduled checkpointing, this involves training the model with the Adam optimizer (Kingma & Ba, 2014). Using the MSE loss function during training, the accuracy of the model is closely monitored through a validation dataset for each iteration of training. Next the model is only saved when the forecasting accuracy, with regards to the validation data, increases during checkpointing. Because training the model takes a lot of time to run, especially when low learning rates are used for model precision, it is important to let the model know at what epoch to start checkpointing. In the current case it was decided to start checkpointing ten epochs before the total amount of training epochs and two epochs for pre-training, where the total amount of epochs is seen as a hyperparameter to be tuned. For the full procedure and results of hyperparameter tuning, please refer to Appendix E.

To accurately assess the differences between LSTM and GRU ensembles and to address the research question, each ensemble will be judged based on its ability to forecast the next period at every time point within the respective test dataset. The comparative analysis between the two will include examining the differences in the types of weights used, EW or PW, and differences in whether TL was used or not during training. The analysis will be conducted comparatively on different datasets, which will be described further in the upcoming data section. All these

differences and its impact on forecasting performance will have to be analyzed numerically and interpreted. Specifically the weights within the PW ensembles can also provide valuable insights as these offer a glance into the performance of lower memory versus higher memory models in different contexts.

So to effectively evaluate the results, a combination of computational and accuracy based performance measures are utilized. Execution time, measured in minutes and seconds, serves as the main measure of computational efficiency.

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{3.13}$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{3.14}$$

Besides that, with forecasting accuracy being of vital importance, the Root Mean Squared Error (RMSE) is employed. RMSE (3.14), is particularly good at penalizing large forecasting errors, thereby ensuring that significant errors are more heavily accounted for. To make sure that the results still true with a different measure, the Mean Absolute Percentage Error (MAPE) (3.13) will shed a different light through it being a relative error measure which makes it easier to interpret due to it being scale independent as opposed to RMSE. In the above formulas y_i is the true observation, $\hat{y}$ is its respective forecast and n is the total number of observations within the test dataset. Additionally the Diebold-Mariano (DM) test (Diebold & Mariano, 2002) will be implemented to test whether the LSTM and GRU ensembles differ significantly from eachother in terms of forecasting accuracy, see Appendix B for an in depth analysis of the DM test. Also will the ensembles be compared against a Random Walk (RW) as a performance check, where for the RW, the forecast of period $t + 1$ is the observed value of period t.

4 Data

Within the Allianz Benelux Group, I have attained data related to the property insurance market from the property insurance division within Allianz called Allianz Property. Companies pay Allianz Property for coverage in the event of property damage, such as fire eruption, theft, water damage and many others. The data that is retrieved from operating this business is utilized for my thesis research. It is important to note that the data has been subjected to a min-max normalization:

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{4.1}$$

where x_{norm} is the normalized value, x is the original value, while x_{min} and x_{max} are the minimum and maximum values of training dataset. The min and max of the training dataset have been used to avoid data leakage to the non-observed entries in the dataset. The normalization has been applied to safeguard the data against potential misuse by competitors, but also to take care of scaling issues by scaling the training data to [0,1]. This constant scale makes hyperparameter tuning more efficient since every dataset has the same scale such that the hyperparameters do not need to be scaled differently for different datasets.

From running business operations there are multiple sources of data coming in every week. The datasets used in this research for forecasting will be the number of claims, which corresponds to how many claims have been filed throughout the year and the claim amounts, which corresponds to the size of the claims throughout the year. This comes down to two distinct datasets. Important to note is that the data is gathered and forecasted in an accumulative fashion. All this means that the datasets contain yearly seasonality but the ML models can handle this type of non-stationarity easily by design (Hewamalage et al., 2021).

Since the company is extremely big in size, company data is stored differently within different departments. Firstly it is important to know that Allianz Property is a multinational company which means that it operates in different countries. The two biggest countries of operation are Belgium (BE) and the Netherlands (NL), which are separate entities within the business. These two entities each have departments which all have their own different data sources. While there

are many different departments within these two entities both have a department called SME which stands for Small/Medium Enterprises and both have a department called Midcorp. The main difference between the SME and Midcorp departments is that Midcorp involves itself more with companies which are larger in terms of revenue and manpower, while SME deals more with smaller and less established enterprises. From now on the SME department within the NL entity will be called NL SME, the Midcorp department within the BE entity will be called BE Midcorp, the Midcorp department of the NL entity will be called NL Midcorp and finally the SME department of the Belgian entity will be called BE SME.

From the aforementioned four departments, every single department has the two aforementioned datasets which are used throughout the paper: claim amounts and claim numbers, so this comes down to eight datasets used in total. During the first and foremost part of the research only NL Midcorp and BE Midcorp data will be used, afterwards NL SME and BE SME datasets will be used as a sensitivity check to verify how much the results hold up with different data sources. Due to the reasons mentioned earlier in the methodology section, the Dutch (NL) datasets will be pre-trained with their respective Belgian (BE) counterpart data as part of the TL process. For example, NL Midcorp number of claims data will be pre- trained using BE Midcorp number of claims data. Important to note is that when TL is not applied then the BE data is also not used.

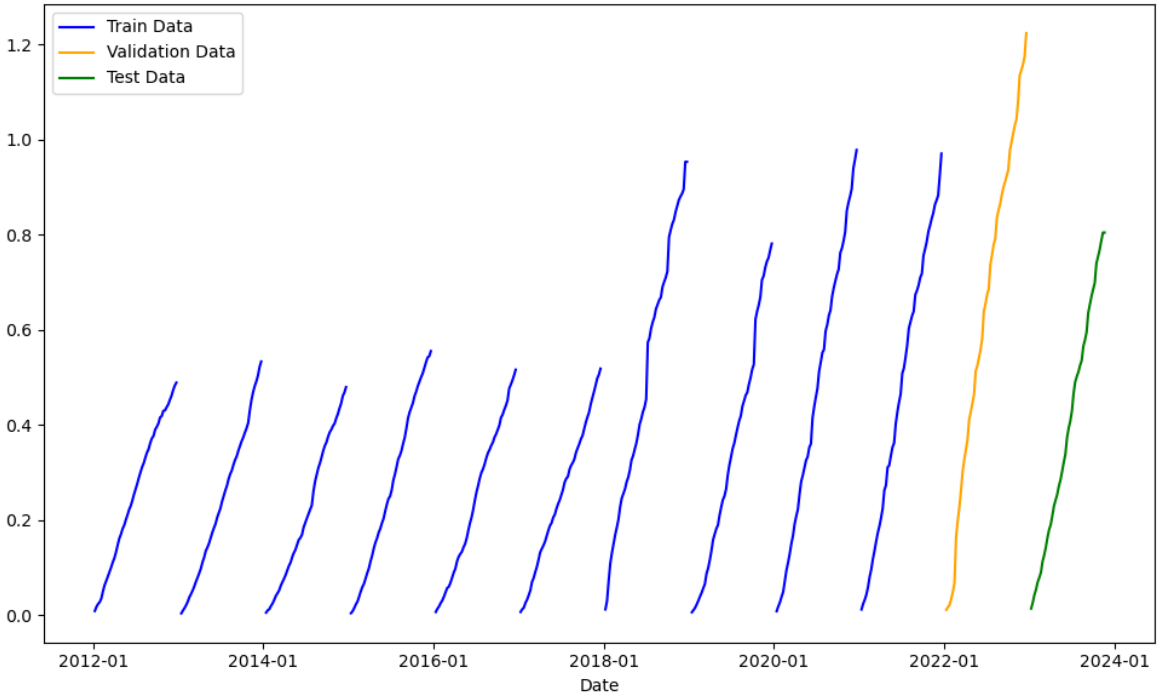
The time frame of the weekly available data for all datasets will include the years 2012 up to and including 2023. To make sure the data is handled properly, it is split into a training, validation and test set while most importantly retaining the chronological order of the data within the splits. After careful consideration it was decided to make use of 2012 up to and including 2021 as the training dataset, 2022 as the validation and 2023 up to the month November as the test dataset. For all eight datasets, the training set consists of 522 weekly observations and 52 for the validation set, while the test data consists of 47 observations because at the time of writing the final observations of 2023 were not yet published within the company. Since there was at most one missing value in a training set, the missing values were excluded from the analysis. As such 84,1% of the data will be used for training purposes. In most cases this will actually be an under representation because the BE SME and BE Midcorp datasets from 2012 to 2021 will also be used for pre-training using TL. Naturally, no BE SME or BE Midcorp data from 2022 or 2023 will be used during pre-training.

So we have in essence four Dutch datasets which will be forecasted, NL Midcorp number of claims, NL SME number of claims, NL Midcorp claim amounts and NL SME claim amounts. Every single one of these datasets has their Belgian counterpart which will be used for pre-training the models. With the help of checkpointing the models will be trained on their training dataset and then validated against the validation dataset to verify with unseen data whether training actually decreased the MSE and thus improved model accuracy on the unseen validation set. Subsequently using univariate forecasting the trained model will forecast the test observations one by one using a sequence of the current and previous three weeks as input and attain a forecast for the next test observation as output.

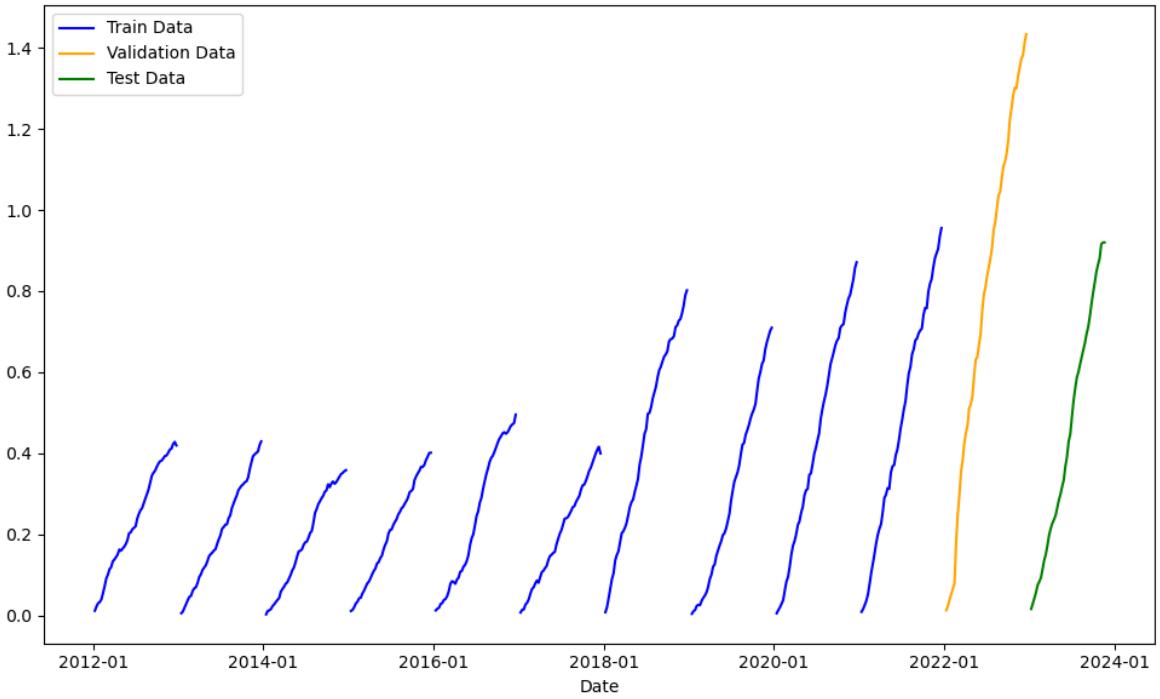
Regarding the fact that so many different datasets are used, it is important to know the differences between the composition of the data. For that reason the data has been put on display for both NL and BE Midcorp datasets at Figure 4.1 and 4.2 below. please refer to Appendix A for both NL SME and BE SME visualizations.

As mentioned before the data has been minmax normalized. This has been done on the training set to determine the minimum and maximum values and then that scale has been applied to the validation and test datasets. This is why the data ranges mostly from 0 to 1, however it also happens, for example in Figure 4.1a, that the range exceeds 1. This happens only when the scale is applied to values in the validation or test dataset which are above the maximum value of the train dataset.

Besides the minmax normalization, the data is also preprocessed using a log1p transformation denoted by $x_{trans} = \log(1 + x_{norm})$, where log is the natural logarithm, x_{norm} is the previously normalized data and x_{trans} is the newly transformed data. This transformation gives the usual advantages of log transformations such as mitigating skewness and stabilizing variance in the data (Curran, 2018), but also works well for the range of $x_{norm} \in [0, 1]$ thereby ensuring the transformed values to remain non-negative. Finally the transformed values are temporarily batched into monthly sequences consisting of four weeks to predict the next weekly period when being fed into the models. For every dataset's test and validation set the first four sequences are removed to counter data leakage from the neighbouring split.

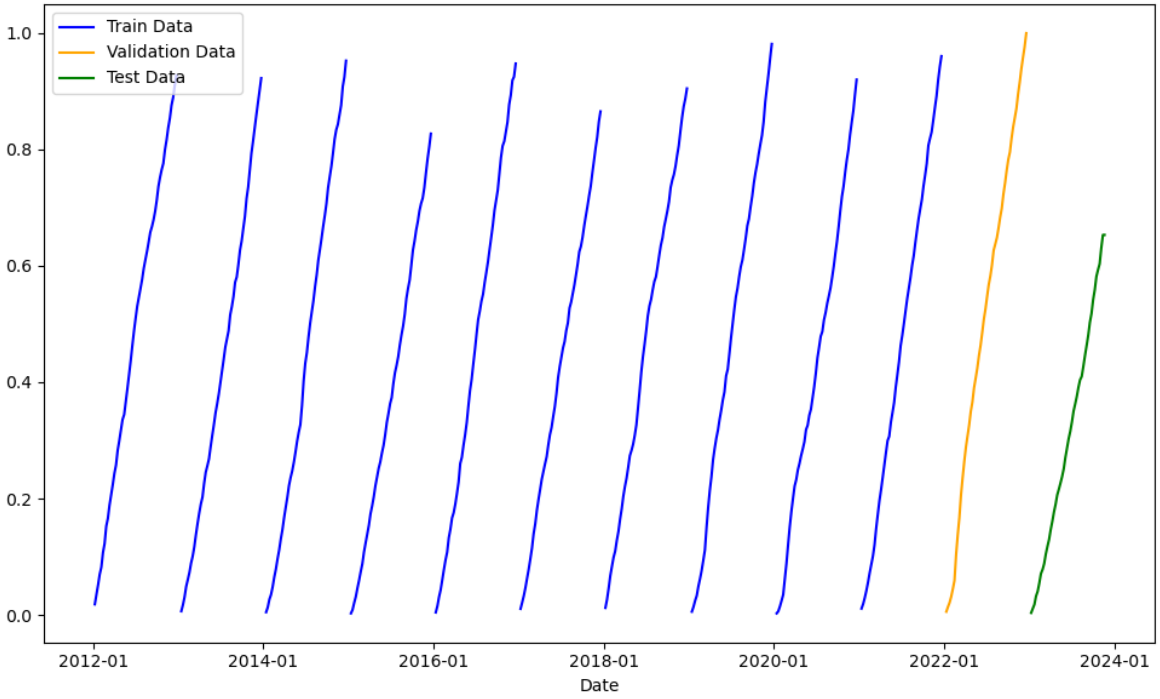


(a) Number of Claims

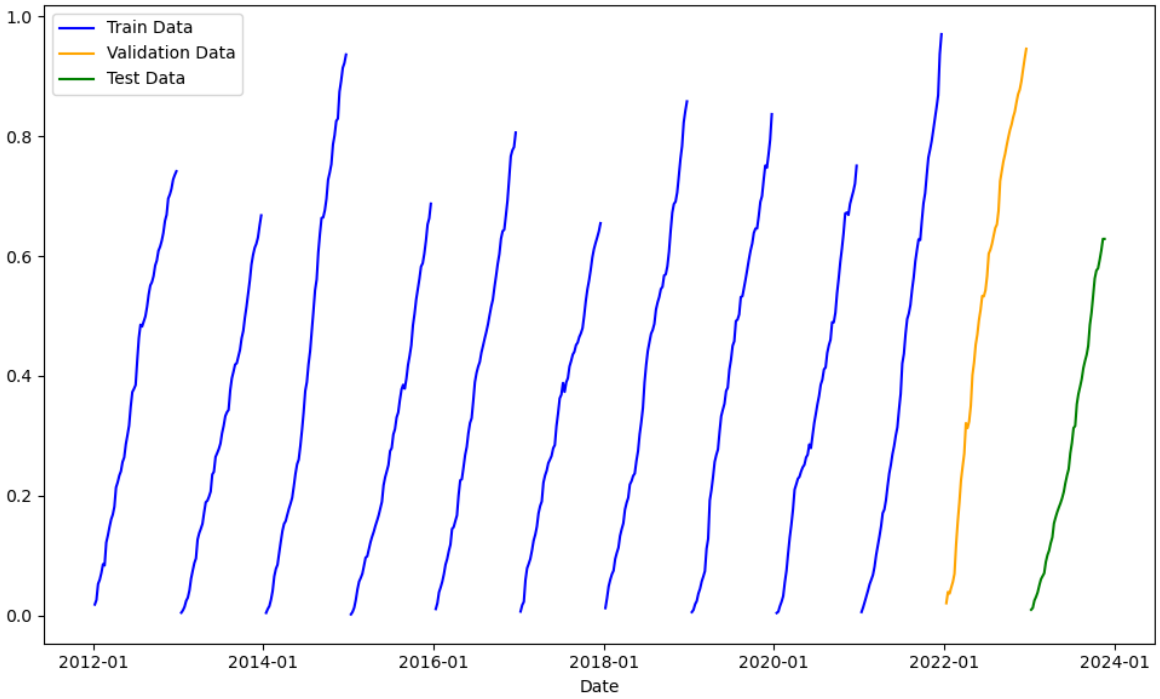


(b) Claim Amount

Figure 4.1: NL Midcorp Normalized Data



(a) Number of Claims



(b) Claim Amount

Figure 4.2: BE Midcorp Normalized Data

5 Results

This section presents the produced results and comparative analysis regarding LSTM and GRU ensembles, focusing on their performance in forecasting insurance claims data. For the first part of the results, the performance of the ensemble models will be analyzed for NL Midcorp data. When TL is used the models are pre-trained with BE SME data as discussed earlier. Within the tables, the datasets regarding number of claims will be referred as Claim_NR and in the same fashion claim amount data will be referred to as Claim_AM. The claim amount data within NL Midcorp which will be used for the main results are in Figure 4.1a and Figure 4.1b.

For each dataset besides accounting for the impact of using TL or not using TL, which for clarity will be referred to within the tables as No TL or TL, an additional matter of consideration is the way the ensembles are weighted. That is why equal weights, referred to as EW and performance weights referred to as PW will be modelled separately. Additionally the weights of the PW ensembles will be displayed in separate tables to provide extra insight into how the PW ensemble differed from the EW ensemble and how this influences forecasting accuracy.

Before beginning the modelling procedure all the ensembles have been collectively tuned based on their underlying data and model. This lead to eight different tuning results, including the tuning results for the sensitivity check. See Appendix E for those results.

Comparative Analysis

First of all to verify whether the ensembles produced quality forecasts they are compared to a random walk (RW). The RW of Claim_NR resulted in a MAPE of 5.51 and the RW of Claim_AM resulted in a higher MAPE OF 5.57. Looking at Table 5.1 it is seen that all MAPE values of the analysis are lower than the RW MAPE values, This demonstrates the potential of GRUs and LSTMs. When comparing the claim number RW with the LSTM using PW and not using TL, the LSTM shows a MAPE reduction of more than 50%.

Table 5.1: MAPE Forecasting Accuracy

	EW Claim_NR	**PW Claim_NR**	**EW Claim_AM**	**PW Claim_AM**
LSTM (No TL)	2.77	2.68	4.87	5.17
GRU (No TL)	3.29	3.24	4.34	4.19
LSTM (TL)	3.54	3.73	5.17	4.68
GRU (TL)	3.17	3.20	4.91	5.23

Note: All values are Mean Average Percentage Error (MAPE)

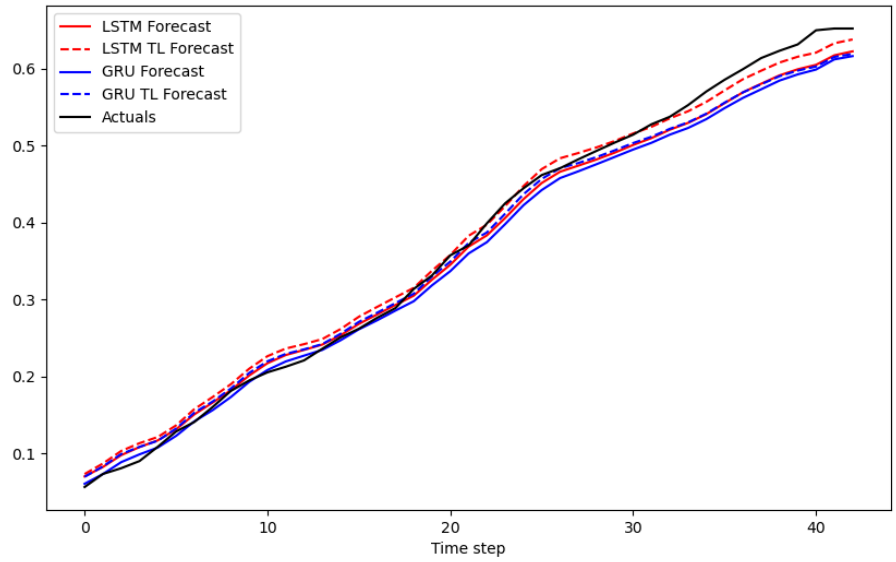
Table 5.2: RMSE Forecasting Accuracy

	EW Claim_NR	**PW Claim_NR**	**EW Claim_AM**	**PW Claim_AM**
LSTM (No TL)	1.25e-02	1.22e-02	1.85e-02	1.69e-02
GRU (No TL)	1.49e-02	1.43e-02	2.22e-02	2.07e-02
LSTM (TL)	8.78e-03	9.02e-03	1.38e-02	1.22e-02
GRU (TL)	1.04e-02	1.09e-02	1.86e-02	1.81e-02

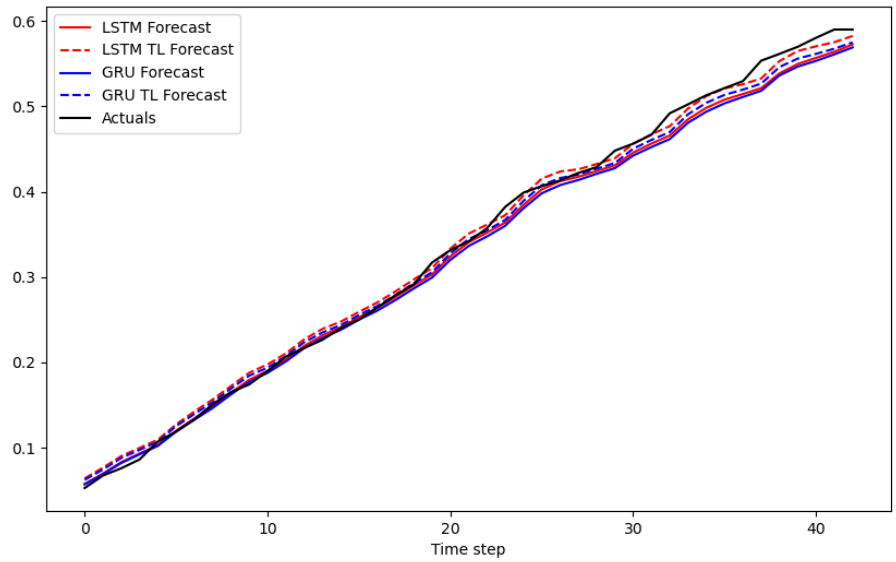
Note: All values are Root Mean Squared Errors (RMSE)

To bring all the above numbers to life, the visualizations of the forecasts alongside their actual values have been displayed below in Figure 5.1. To keep the figures tidy the visualizations for the PW forecasts have been put for display in Appendix C.

Regarding the accuracy of the forecasts it is observed that the different accuracy measures provide slightly different insights. When comparing the accuracy of the LSTM ensembles versus the GRU ensembles, it is observed that in Table 5.1 that for Claim_NR data when TL is not implemented, LSTM attains a lower MAPE while for Claim_AM the roles seem to be reversed with GRU showing higher accuracy. Here an interesting observation exception that in the CLaim_AM case when TL is not applied, the GRU ensemble improves greatly from the introduction of PW, with it resulting in a 4.1% lower MAPE compared to the EW, while LSTM shows a reduction in accuracy from the introduction of PW, with a 6.2% MAPE increase.



(a) Number of Claims



(b) Claim Amount

Figure 5.1: Midcorp EW Forecasting

The comparison between the ensembles where TL is used and the ensembles were it is not used also shows some interesting results. The MAPE values of the TL ensembles seem to achieve lesser accuracy than its non TL counterpart. When viewed from Table 5.2 however the opposite result emerges with every ensemble benefiting from TL. Since RMSE punishes larger errors much more than the MAE this could be a sign that the model is making less large errors, but

the average error of TL ensembles in turn are on a slightly larger magnitude.

As of the implementation of PW, the tables point in the same direction as the implementation of TL with it showing more accuracy gains in the RMSE context than in the MAPE context. In Table 5.1, PW reduces MAPE values in 4 out of 8 cases with respect to their EW counterpart. In Table 5.2 the PW reduces the RMSE values in 6 out of 8 cases.

Table 5.3: Performance Weightings

	8	**16**	**32**	**64**	**128**
LSTM - Claim_NR - NO TL	0.182	0.187	0.184	0.208	0.239
GRU - Claim_NR - NO TL	0.162	0.187	0.207	0.194	0.251
LSTM - Claim_AM - NO TL	0.138	0.173	0.164	0.252	0.274
GRU - Claim_AM - NO TL	0.158	0.190	0.176	0.202	0.274
LSTM - Claim_NR - TL	0.129	0.216	0.258	0.296	0.101
GRU - Claim_NR - TL	0.147	0.256	0.259	0.188	0.150
LSTM - Claim_AM - TL	0.224	0.231	0.181	0.288	0.076
GRU - Claim_AM - TL	0.138	0.222	0.303	0.188	0.150

Table 5.3 depicts how the ensembles assign performance weights based on each part of the ensembles' capability to accurately forecast the validation data. In Table 5.3 numbers above 0.200 indicate an increase in weight relatively to the EW model and numbers below 0.200 indicate a reduction. Taking a look at a case where forecasting accuracy did not change much as a consequence of changing from EW to PW, LSTM - Claim_NR - NO TL, it is observed that the model changes weights a lot, with the ensemble placing a heavy emphasis on the 128 node model.

Interesting to see is that its GRU counterpart, GRU - Claim_NR - NO TL does the same thing but even to a higher degree and therefore both ensembles manage to attain improved accuracy as a result of PW.

Overall there does not seem to be a big observable difference in weights between the LSTM ensembles and the GRU ensembles, the biggest insights here seems to be that the ensemble prefers the 128 node model when No TL is used and lower node models when TL is used. However besides LSTM- Claim_AM - TL, none of the models seem to favour the model with 8 nodes, relatively to the EW setting.

Table 5.4: Diebold-Mariano Tests

	EW Claim_NR	**PW Claim_NR**	**EW Claim_AM**	**PW Claim_AM**
DM-Stat (No TL)	-6.05	-5.83	-4.32	-3.30
P-Value (No TL)	1.45e-09	5.63e-09	1.57e–05	0.00097
DM-Stat (TL)	-1.94	-1.65	-2.41	-3.13
P-Value (TL)	0.052	0.099	0.016	0.0017

Taking a significance level of 0.05 , the Diebold-Mariano tests, where the forecast errors of every LSTM and its GRU counterpart were taken as inputs, resulted in the null hypothesis of equal accuracy often being rejected. Only in the two scenarios of claim numbers with TL, under both EW and PW, did the ensembles not reject the null hypothesis. This is quite an interesting development as even though in both cases the RMSE values for these ensembles are more than 10% lower for the GRU ensemble there is not enough cause to suggest that this difference is significant. However all the other differences in accuracy between the ensembles are shown to be significant.

Sensitivity analysis

Now the comparative analysis will be redone using NL SME data and BE SME data when TL is used. As before the tuning results are in Appendix E. As for the RW we get similar

MAPE values, for claim amount data the MAPE of the RW is 5.57 and for the number of claims dataset the MAPE of the RW is 5.68. This leads us to the first divergence from the main Midcorp analysis and that is that for most ensembles, besides the LSTM (TL) - Claim_AM ensemble, it is observed that the RW shows lower MAPE values. This could be interpreted as both models struggling to keep the average error down due to more noisy data, see Appendix D for the visualizations of the SME forecasts. Looking at the RMSE values from Table 5.6 it is observed that the RMSE values of the sensitivity analysis are lower than for the previous RMSE values associated with Midcorp, which is a favourable sign.

Table 5.5: MAPE Values - SME

	EW Claim_NR	**PW Claim_NR**	**EW Claim_AM**	**PW Claim_AM**
LSTM (No TL)	9.55	9.56	8.42	8.73
GRU (No TL)	7.63	7.26	5.92	6.38
LSTM (TL)	7.54	6.91	4.94	4.91
GRU (TL)	6.51	6.22	7.22	8.06

Note: All values are Mean Average Percentage Error (MAPE)

Table 5.6: RMSE - SME

	EW Claim_NR	**PW Claim_NR**	**EW Claim_AM**	**PW Claim_AM**
LSTM (No TL)	1.02e-02	1.02e-02	1.45e-02	1.50e-02
GRU (No TL)	7.90e-03	7.50e-03	9.55e-03	1.04e-02
LSTM (TL)	8.35e-03	7.79e-03	9.27e-03	8.94e-03
GRU (TL)	6.73e-03	6.45e-03	1.23e-02	1.37e-02

Note: All values are Root Mean Squared Errors (RMSE)

Regarding the difference in accuracy between LSTM and GRU ensembles we can see from

both MAPE and RMSE values that the GRU ensembles show higher accuracy than the LSTM ensemble in 6 out of 8 cases. With the cases of LSTM outperformance being Claim_AM TL. This is in itself is not in line with the previous analysis, but what remains true is that the difference in accuracy between the ensembles remains very dependent on the situation, with no clear winner on all fronts. Interestingly the Diebold-Mariano test this time rejects the null hypothesis every single time based on a significance level of 0.05, this shows that the differences between the accuracy of the two models are very much significant. See the Diebold Mariano test results in Appendix D

The TL reducing MAPE and RMSE values also seems to be indicative of the tables, with these values being reduced in all cases except for the GRU Claim_AM where, for both MAPE and RMSE, forecasting accuracy seems to deteriorate from introducing TL. The accuracy gains attained from PW also remains mixed with 4 to 5 out of 8 cases showing a clear boost in accuracy when compared with EW.

From the performance weights attained from the sensitivity test in Appendix D, it is seen that the ensembles provide this time weights that show a more balanced preference for the number of nodes within the ensembles. For cases where the implementation of PW resulted in relatively minor accuracy changes, the PW still seems to make use of all parts of the ensemble, with LSTM - CLAIM_NR - NO TL showing increases in weight for the model with 8 nodes relatively to the EW setting, while LSTM - CLAIM_AM - NO TL showing a strong preference for the 128 nodes part of the ensemble. Even though in these two cases this did not result in an increase in accuracy it still shows that the ensemble makes use of the different parts to capture the validation dataset, and this then in about half of the times results in increased accuracy.

Finally to wrap up the analysis the execution times are compared. Here the execution time is defined as the amount of time it took for all the GRU and LSTM ensembles to run together. For the comparative analysis, the LSTM ensembles took about 9 minutes and 36 seconds while the GRU ensembles took about 8 minutes and 30 seconds to run, this represents about 11.4% less time to run the GRU ensembles. For the sensitivity check it took for the LSTM 9 min and 40 sec and for the GRU 9 min and 36 sec to run, which represents a 0.3% decrease in execution time. These execution time results would be more meaningful with access to the cloud and more similar hyperparameters however it is impressive to see that GRU indeed indicates to be more efficient than LSTM with regards to computational speeds.

In conclusion, after having executed all the ensembles and observing all the accuracy measures and weights, it is shown that the LSTM and GRU ensembles have varying performance and have different memory preferences depending on the situation. The different data sources and different ways of looking through the data allowed to view the results through various angles. From that it became clear that the ensembles were TL was used are quite useful at improving ensemble forecasting accuracy. The PW ensembles were useful for giving different insights into the inner workings of the ensemble, even though the ensembles could not always translate it into enhanced performance.

The choice of LSTM versus GRU, seems to be a very context dependent choice with the different ensembles showing significant differences in their accuracy. Additionally the shorter training durations and more easily to understand cell architecture give GRU ensembles a positive signal. But this preference should be regarded as a weak one due to the slightly more complicated structure of LSTMs becoming an advantage in certain scenarios.

6 Discussion

In this paper a distinct framework of comparing LSTM with GRU models has been proposed. This has been achieved via ensembling both models with respect to the number of nodes, thus creating an ensemble which captures different aspects of the data due to it having access to various memory capabilities. With the help of different datasets, ensemble setups, accuracy measures, the use of TL and analyzing the performance weights, it was possible to comparatively analyze the ensemble models through different ensemble configurations.

The conducted research shows that the choice of LSTM versus GRU, seems to be a very context dependent choice, like Chung et al. (2014) hypothesized, with the different ensembles configurations showing significant differences in their accuracy. The main analysis showed the LSTM ensembles achieving slightly higher accuracies, while the GRU ensembles achieved higher accuracies in the sensitivity check conducted on different datasets. In both cases however, the other inferior ensemble was still showing competitive, and sometimes even better, performance. Through Diebold-Mariano tests the differences in accuracy were deemed to be significant in the vast majority of cases thus exemplifying the importance of the choice between the two.

The implementation of TL often lead to increases in accuracy, especially in terms of RMSE, where pre-training consistently lead to accuracy gains. The PW ensemble setting did in about half of the times lead to accuracy increases, however it always made ample use of the different memory capabilities of the ensemble.

Finally, the adoption of this ensembling technique has proven to be an effective strategy for evaluating these two RNN models. This approach has effectively mitigated the inherent randomness that often complicates the comparison of similar, yet contrasting models. Through the utilization of diverse datasets and methodologies, a multifaceted analysis was conducted, resulting in a more resilient and dependable comparison. For future research I wish to see this ensemble framework changed from number of nodes to sequence lengths ensembles to see how that changes the LSTM vs GRU analysis.

References

Alzubi, J., Nayyar, A. & Kumar, A. (2018). Machine learning from theory to algorithms: an overview. In *Journal of physics: conference series.*

Bala, R. & Singh, R. P. (2019). Financial and non-stationary time series forecasting using lstm recurrent neural network for short and long horizon..

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, *13*(2).

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, *31*(3), 307–327.

Borovkova, S. & Tsiamas, I. (2019). An ensemble of lstm neural networks for high-frequency stock market classification. *Journal of Forecasting*, *38*(6), 600–619.

Box, G. E. P., Jenkins, G. M. & Reinsel, G. (2015). *Time series analysis: forecasting and control.* John Wiley & Sons.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling.

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International conference on machine learning.*

Curran, D. (2018). Explorations in statistics: the log transformation. *Advances in physiology education*, *42*(2), 343–347.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V. & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Diebold, F. X. & Mariano, R. S. (2002). Comparing predictive accuracy. *Journal of Business & economic statistics*, *20*(1), 134–144.

Di Persio, L. & Honchar, O. (2017). Recurrent neural networks approach to the financial forecast of google assets. *International Journal of Mathematics and Computers in Simulation*, *11*, 7–13.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Fischer, T. & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669.

Fradkov, A. L. (2020). Early history of machine learning. *IFAC-PapersOnLine*, *53*(2), 1385–1390.

Gers, F. A., Schmidhuber, J. & Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, *12*(10), 2451–2471.

Guan, Y. & Plotz, T. (2017). Ensembles of deep lstm learners for activity recognition using wearables. *Proceedings of the ACM on interactive, mobile, wearable and ubiquitous technologies*, *1*(2), 1–28.

Hall, A. S. (2018). Machine learning approaches to macroeconomic forecasting. *The Federal Reserve Bank of Kansas City Economic Review*, *103*(63), 2.

Han, S., Mao, H. & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*.

Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J. & Scholkopf, B. (1998). Support vector machines. *IEEE Intelligent Systems and their applications*, *13*(4), 18–28.

Hewamalage, H., Bergmeir, C. & Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, *37*(1), 388–427.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Jozefowicz, R., Zaremba, W. & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. , 2342-2350.

Kaastra, I. & Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, *10*(3), 215–236.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, *30*.

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Krogh, A. & Hertz, J. (1991). A simple weight decay can improve generalization. *Advances in neural information processing systems*, *4*.

Lipton, Z. C. (2018). The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, *16*(3), 31–57.

Livieris, I. E., Pintelas, E., Stavroyiannis, S. & Pintelas, P. (2020). Ensemble deep learning models for forecasting cryptocurrency time-series. *Algorithms*, *13*(5), 121.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2018a). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, *34*(4), 802–808.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. , *13*(3).

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*, 115–133.

Molnar, C., Casalicchio, G. & Bischl, B. (2020). Interpretable machine learning–a brief history, state-of-the-art and challenges. In *Joint european conference on machine learning and knowledge discovery in databases* (pp. 417–431).

Pascanu, R., Mikolov, T. & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Poll, R., Polyvyanyy, A., Rosemann, M., Roglinger, M. & Rupprecht, L. (2018). *Process forecasting: Towards proactive business process management*. Springer.

Rahman, M. & Watanobe, Y. (2023). Chatgpt for education and research: Opportunities, threats, and strategies. *Applied Sciences*, *13*(9), 5783.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536.

Schulz, E., Speekenbrink, M. & Krause, A. (2018). A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology*, *85*, 1–16.

Schuster, M. & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), 2673–2681.

Snoek, J., Larochelle, H. & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, *25*.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, *27*.

Taylor, S. J. & Letham, B. (2018). Forecasting at scale. *The American Statistician*, *72*(1), 37–45.

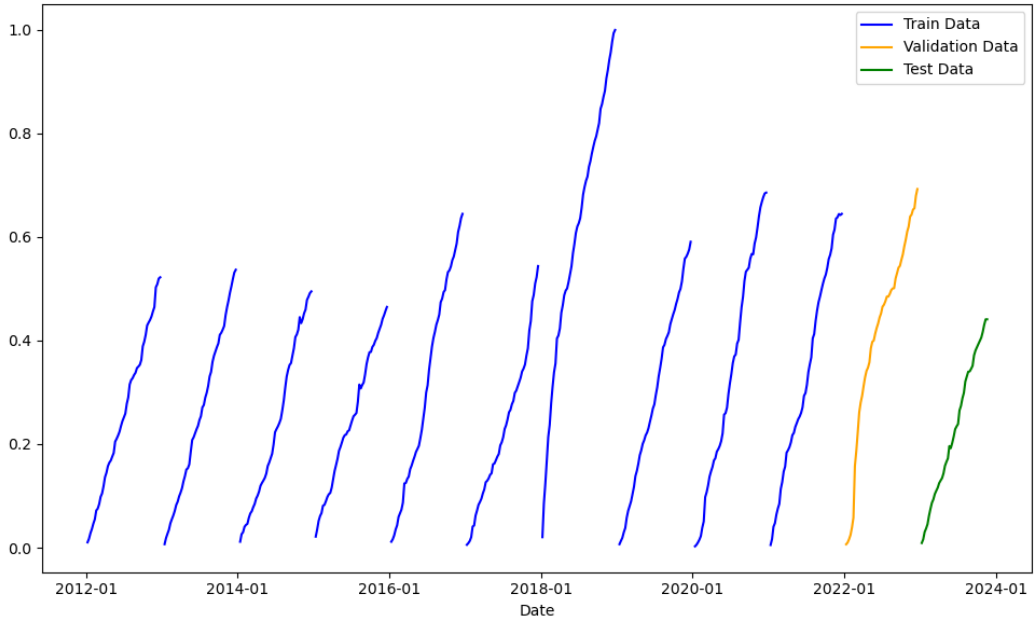
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. & Gomez, A. N. (2017). Attention

is all you need. *Advances in Neural Information Processing Systems*, *30*.

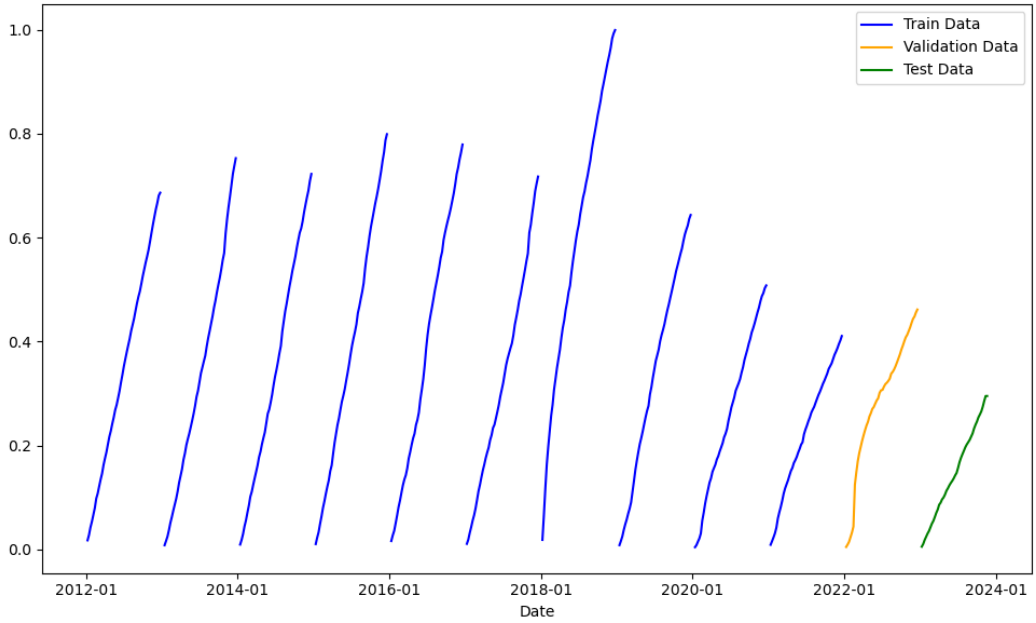
Weiss, K., Khoshgoftaar, T. M. & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, *3*(1), 1–40.

Yu, T. & Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications. *arXiv:2003.05689*.

A Property SME Data

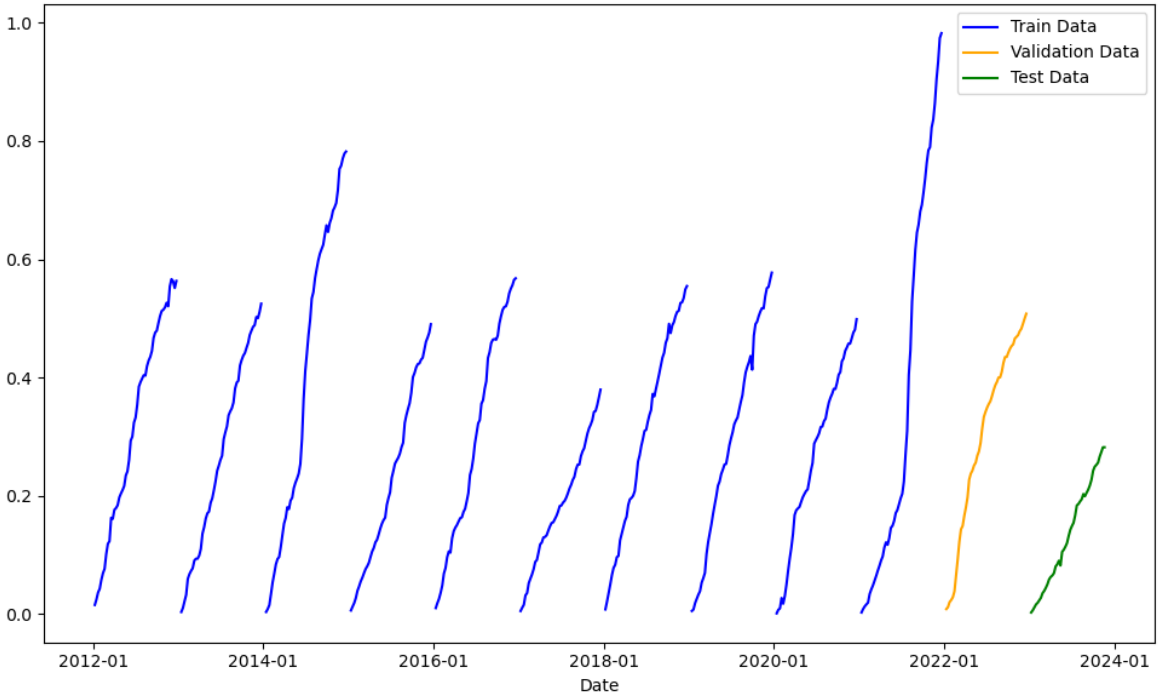


(a) Claim Amount

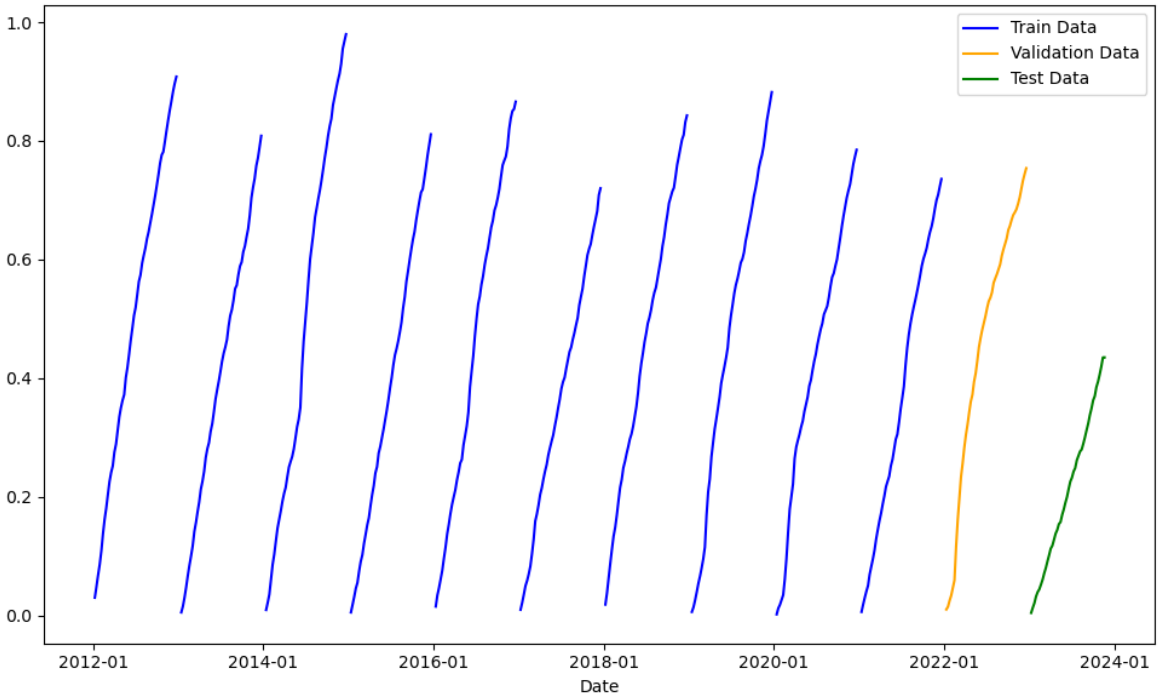


(b) Number of Claims

Figure A.1: NL SME Normalized Data



(a) Claim Amount



(b) Number of Claims

Figure A.2: BE SME Normalized Data

B Diebold-Mariano Test

The Diebold-Mariano (DM) test, which was introduced by Diebold and Mariano (2002) is a statistical test that compares two sets of predictions with the actual values to compare the accuracy of both forecasting methods. The null hypothesis of the test is that the two forecasting models have equal predictive accuracy while the alternative hypothesis is that the two forecasting models have different predictive accuracies.

The first step is choosing a loss function, where it was decided to go with the most common option for DM tests which is the squared error loss function:

$$L_a(t) = (y_t - \hat{y}_{a,t})^2$$

$$L_b(t) = (y_t - \hat{y}_{b,t})^2$$

If a represents model a and b represents model b, then $L_a(t)$ and $L_b(t)$ are their respective squared error loss functions at time t. Then y_t is the actual value at t while $\hat{y}_{a,t}$ and $\hat{y}_{b,t}$ are its predictions from both models at time step t. The next step involves calculating the loss differential, d_t and its average value $\overline{d}$. The loss differential is the difference between the loss functions of the two models at each time t. expressed as:

$$d_t = L_a(t) - L_b(t)$$

$$\overline{d} = \frac{1}{T}\sum_{t=1}^{T} d_t$$

Here, T represents the total number of observations. The standard error of the loss differential, $\mathrm{SE}(\overline{d})$ and with that the Diebold-Mariano test statistic DM, are then determined as

$$\mathrm{SE}(\overline{d}) = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(d_t - \overline{d})^2}$$

$$DM = \frac{\overline{d}}{\mathrm{SE}(\overline{d})}$$

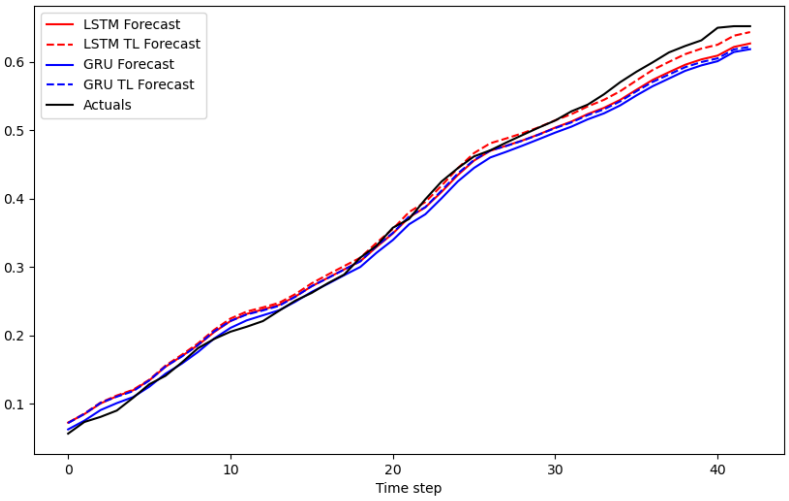
This statistic is used to test the null hypothesis that the two forecasting models, a and b, have equal predictive accuracy against the alternative hypothesis that they do not. The decision to reject or not reject the null hypothesis is based on the computed p-value. The p-value indicates the probability of observing a test statistic as extreme as the DM statistic, under the null hypothesis that the two forecasting models have equal predictive accuracy.

To calculate the p-value, the distribution of the DM statistic under the null hypothesis must be known. Under the null hypothesis and certain regularity conditions, which are out of scope for this thesis, the DM statistic asymptotically follows a standard normal distribution. Therefore, the p-value can be calculated using the cumulative distribution function (CDF) of the standard normal distribution. Using these assumptions the two-tailed p-value is calculated as:

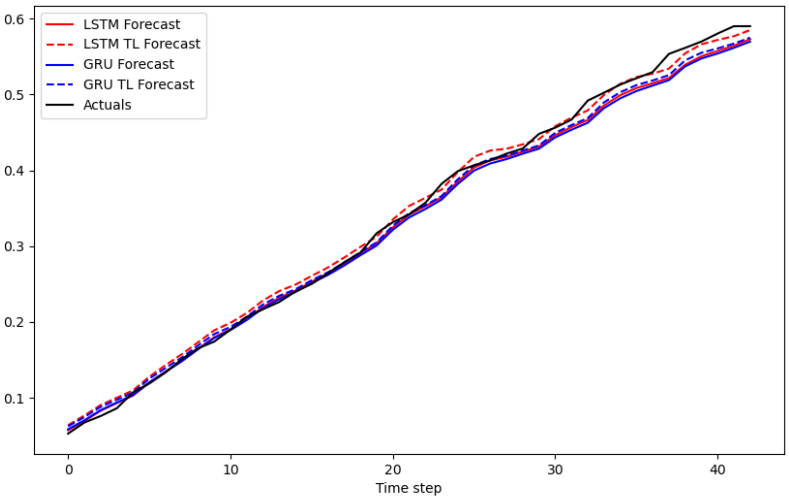
$$p\text{-value} = 2 \times (1 - \Phi(|DM|))$$

where Φ denotes the CDF of the standard normal distribution, and $|DM|$ is the absolute value of the DM statistic. A p-value smaller than the significance level of 0.05 indicates strong evidence against the null hypothesis, suggesting that the predictive accuracies of the two models are different.

C PW Visualizations - Midcorp



(a) Claim Amount



(b) Number of Claims

Figure C.1: Midcorp PW Forecasting

D Additional Results - Sensitivity Check

Table D.1: Performance Weights - SME

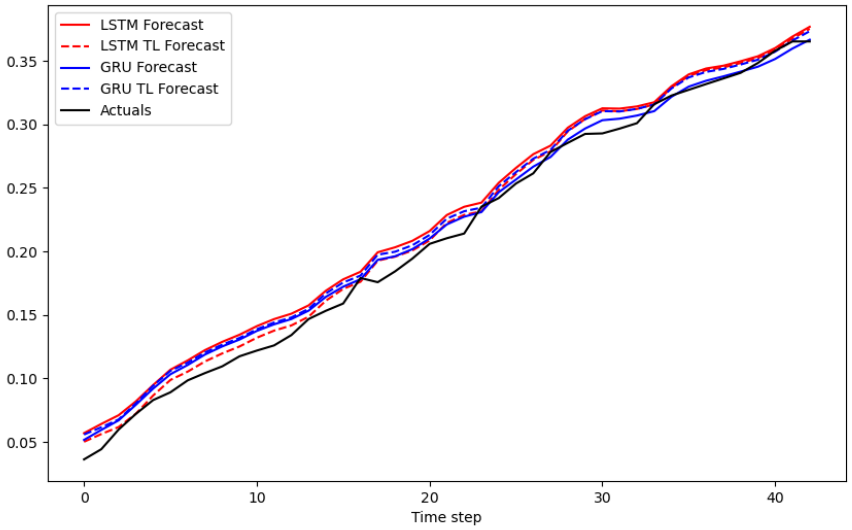
	8	**16**	**32**	**64**	**128**
LSTM - Claim_NR - NO TL	0.258	0.151	0.244	0.175	0.172
GRU - Claim_NR - NO TL	0.197	0.164	0.183	0.218	0.238
LSTM - CLAIM_AM - NO TL	0.207	0.130	0.209	0.207	0.246
GRU - CLAIM_AM - NO TL	0.183	0.146	0.175	0.264	0.232
LSTM - Claim_NR - TL	0.194	0.172	0.203	0.150	0.282
GRU - Claim_NR - TL	0.198	0.225	0.155	0.163	0.260
LSTM - CLAIM_AM - TL	0.136	0.276	0.159	0.205	0.225
GRU - CLAIM_AM - TL	0.225	0.215	0.134	0.249	0.177

Diebold-Mariano Tests

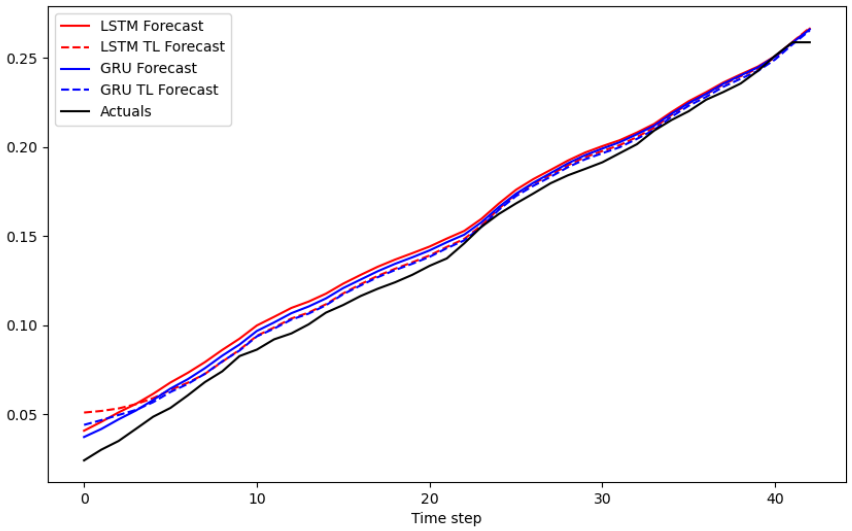
Table D.2: Diebold-Mariano Tests

	EW **Claim_NR**	**PW** **Claim_NR**	**EW** **Claim_AM**	**PW** **Claim_AM**
DM-Stat (No TL)	8.56	7.73	10.09	11.14
P-Value (No TL)	1.12e-17	1.05e-14	6.26e-24	8.00e-29
DM-Stat (TL)	2.72	2.11	-5.47	-7.34
P-Value (TL)	0.0065	0.035	4.59e-08	2.10e-13

Visualizations

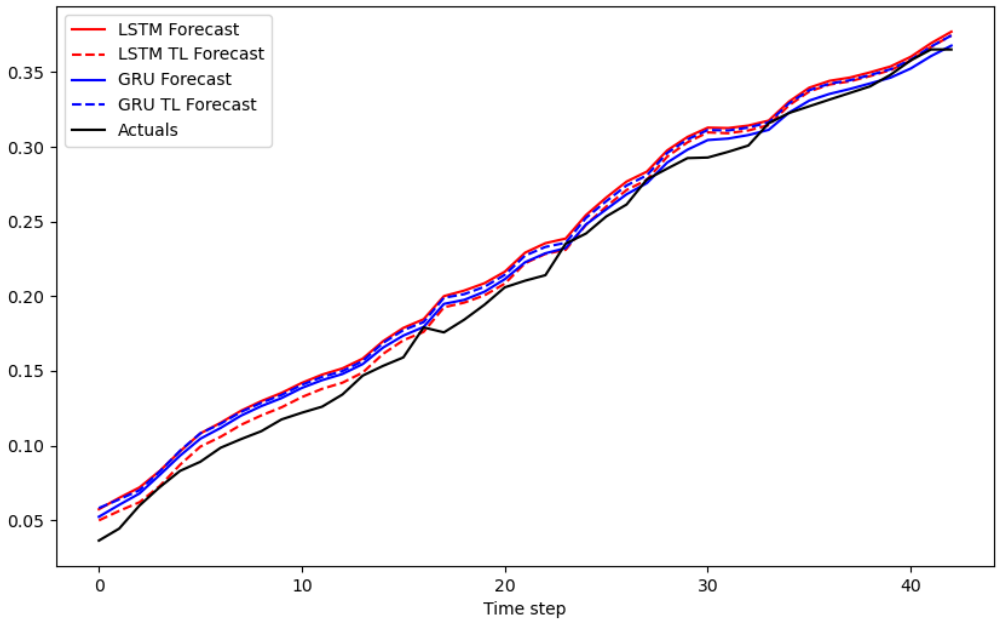


(a) Claim Amount

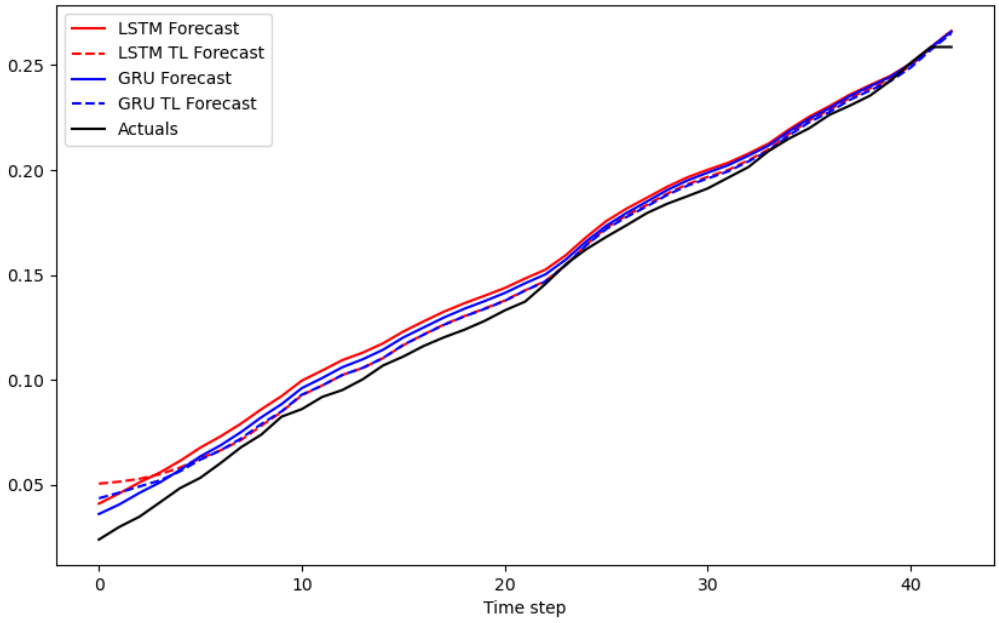


(b) Number of Claims

Figure D.1: SME EW Forecasting



(a) Claim Amount



(b) Number of Claims

Figure D.2: SME PW Forecasting

E Hyperparameter Optimization

Hyperparameter tuning plays a pivotal role in enhancing model performance and robustness, especially in complex time-series forecasting tasks. This section delves into the important process of hyperparameter tuning for LSTM and GRU ensemble models, focusing on optimizing the L2-penalty of the LSTM or GRU layer and dropout rates of the dense layers, which both play the crucial role of regularization.

The tuning process for the ensembles turned out to be quite a challenge as the parts of the ensemble with lower memory capabilities had to capture enough trends in the data, while making sure that the higher memory parts of the ensemble did not start to overfit, that is why the 8 nodes model and 128 nodes model were the primary focus of the tuning and if these two were correctly tuned the ensemble parts in between were assumed to be correctly tuned.

The hyperparameters from the LSTM ensembles are tuned separately from the GRU ensembles, since they require different hyperparameter setups as their cell architecture is completely different. Additionally were the hyperparameters tuned separately based on data as some datasets required the ensembles to be trained more deeply while others less so.

The steps taken to manually tune the hyperparameters were identical for all situations. First taking a large number of 120 epochs, an optimal learning rate was determined such that the respective 8 node part of the LSTM or GRU started to overfit using the training loss. Next the dropout and l2 losses were introduced to make sure that the 128 node model was restricted in its degree of overfitting while still making sure that the 8 node model could still manage to capture a high degree of the data trends. Afterwards the number of epochs, for both LSTM and GRU ensembles, are reduced to ensure the ensemble as a whole does not overfit. For every step of the tuning process the validation and training loss are constantly monitored. Finally when TL is used the pre-training model is given increased regularization to heavily reduce overfitting in the pre-training process.

Tuning Results

First of all for the main analysis of SME data. The tuning results are divided among using LSTM or GRU and the two datasets. For all four of them including in the identical four splits in the sensitivity check, the pre-training models were set to have dense layers with a dropout rate of 0.2 and the LSTM or GRU layer was set to have an L2 penalty of 0.1 . These values are relatively high as the TL data could not be used to train the models too deeply.

Next the results for all the splits will be presented:

Table E.1: Hyperparameter Settings

Dataset	**LSTM**	**Epochs**	**Dropout Rate**	**L2 Strength**	**Learning Rate**
CLAIM_AMOUNT	0	100	0	0.0001	0.00025
CLAIM_AMOUNT	1	100	0	0.0001	0.0005
NR_OF_CLAIMS	0	90	0	0.0002	0.00025
NR_OF_CLAIMS	1	90	0	0.0001	0.00045

Table E.2: Hyperparameter Settings - Sensitivity Check

Dataset	**LSTM**	**Epochs**	**Dropout Rate**	**L2 Strength**	**Learning Rate**
CLAIM_AMOUNT	0	100	0	00025	0.00025
CLAIM_AMOUNT	1	100	0	0.0002	0.0007
NR_OF_CLAIMS	0	110	0	0.0002	0.0002
NR_OF_CLAIMS	1	110	0	0.0001	0.00035

Please note that in the above tables an LSTM of 1 represents an LSTM ensemble and an LSTM of 0 represents a GRU ensemble.