INTERNATIONAL JOURNAL OF SOCIAL Sciences and Information Technology
ISSN 2412-0294 www.ijssit.com

http://www.ijssit.com

# TOWARDS OPTIMIZATION OF THE GATED RECURRENT UNIT (GRU) FOR REGRESSION MODELING

[1*] **Zachary Kirori**
*zkirori@kyu.ac.ke*

[2**] **Edwin Ireri**
*eireri@kyu.ac.ke*

[1,2] *Kirinyaga University, Kenya*

***Abstract****: Deep Machine Learning takes place by adjusting the weights of deep neural networks – brain-like computational structures in order to optimize a given cost function. There exists a deep learning task where every neural model typically performs better. This paper explores optimization strategies for the GRU neural network such as dropout, gradient clipping and stacking ensembles of neural layers amongst others. The Recurrent Neural Networks (RNNs) are suited for classification and prediction problems based on time-series data. It has proven a challenging task for an ordinary RNN to learn long sequences leading to the invention of two other variants: Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU). Whereas both of these compare well on most learning tasks, the structure of a LSTM is more complex with three gates and in effect has to compute and keep the cell state ($C_t$). This makes GRU simpler to implement and typically converges faster making it more appropriate for online learning. The results of experiments indicate an improved convergence rate and better performance guarantees of the GRU over LSTM for typical regression tasks.*

***Keywords****: Deep Neural Networks, Recurrent Neural Networks, Deep Learning Optimization, Long Short Term Memory, Gated Recurrent Unit*

## 1. INTRODUCTION

The origin of modern stateful deep neural networks such as Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) is the Recurrent Neural Network (RNN) appropriate for regression modeling of deep learning tasks such as prediction and classification of time-series data. The primary distinction between a typical deep neural network such as a Convolution Neural Network (CNN) and a RNN is that rather than completely feed-forward connections, a recurrent network might have connections that feed back into prior layers (or into the same layer) facilitating RNNs to maintain memory of past inputs and model problems in time, Aditya Rawal [1]. Figure 1 below shows a typical RNN. RNNs can be unfolded in time and trained with standard back-propagation algorithm or its variant, the back-propagation through time (BPTT) as indicated in figure 2.

However, during back propagation, in an ordinary recurrent neural network, the gradient shrinks as it back propagates through time – a phenomenon typically referred to as the 'vanishing gradient'. This has the effect of reducing the gradient value such that it eventually becomes extremely small, thus inhibiting learning. The end effect is for such a network to forget state in longer sequences, thus having a short-term memory. This led to the invention of LSTM and GRU that have abilities to model longer sequences. Whereas LSTM memory cell contains three gates, GRU has two gates: an update gate and a reset gate, getting rid of the output

gate present in the LSTM model making it much more computationally efficient than LSTM.

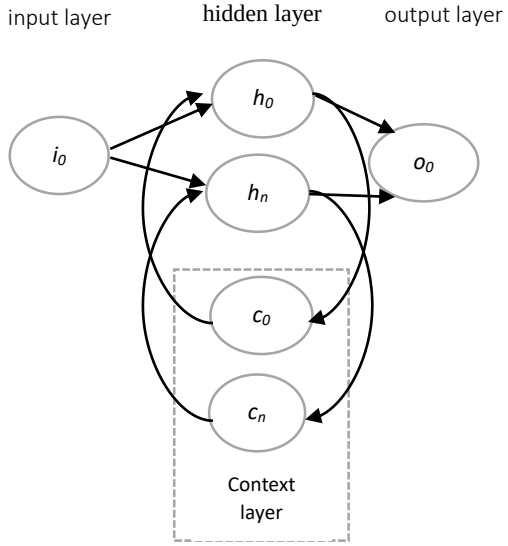


*Fig. 1: A typical RNN (Source: Self)*

The training algorithm, commonly BPTT, optimizes these weights based on the resulting network output error. For many applications, the GRU has performance similar to the LSTM, but being simpler means fewer weights and faster execution. The update gate indicates how much of the previous cell contents to maintain. The reset gate defines how to incorporate the new input with the previous cell contents. A GRU can model a standard RNN simply by setting the reset gate to 1 and the update gate to 0. Figure 2 below is a typical representation of a GRU cell.

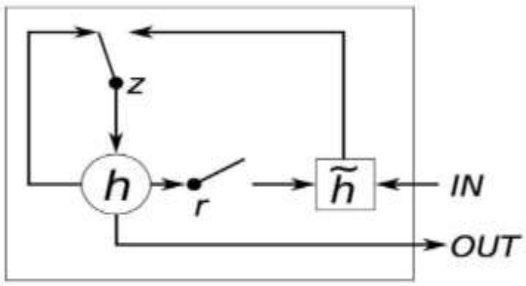


*Figure 2: GRU Cell (Source: Self)*

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state of the art results based on recurrent neural networks are commonly achieved with these two networks. LSTM's and GRU's are found in speech recognition, speech synthesis and text generation among other modern machine learning tasks, Apeksha Nagesh Shewalkar [3].

## 2. MATHEMATICAL GRU MODEL

Based on the GRU cell in figure 2 above we can unfold the structure in order to define a mathematical model representing the network. The unfolded GRU cell is represented by figure 3 next page.

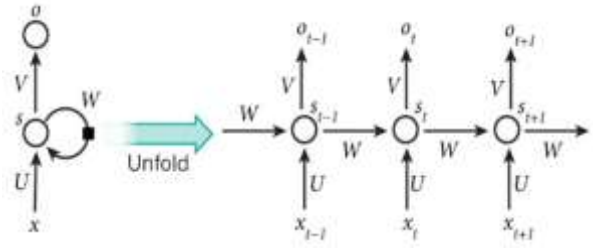


*Figure 3: Unfolding a GRU (Source: Self)*

The key idea of GRUs is that the gradient chains do not vanish due to the length of sequences. This is done by allowing the model to pass values completely through the cells. The model is defined by the following set of equations:

$$\begin{aligned} z_t &= \sigma(W^z x_t + U^z h_t - 1 + b^z) \\ r_t &= \sigma(W^r x_t + U^r h_t - 1 + b^r) \\ \sim h_t &= \tan^{-h}(W^h x_t + U^h h_t - 1 \circ r_t + b^h) \\ h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sim h_t \end{aligned} \quad (1)$$

The input to hidden connections is parametrized by a weight matrix *U*, hidden-to-hidden recurrent connections parametrized by a weight matrix *W* and hidden-to-output by weight matrix *V* with a bias vectors *b*. In the definitions, ∘ represents the Hadamard product, for element-wise multiplication; σ(x) is the Sigmoid function that has the effect of squishing values between 0 and 1 – useful in updating or forgetting data as any number multiplied by 0 is 0, causing values to disappears or be "forgotten". Equally, multiplying a number by 1 retains the value hence propagating that value forward into other layers in the network. A tan-h

function ensures that the values stay between -1 and 1, thus regulating the output of the neural network.

$z_t$ functions as a filter for the previous state. If it is low (near 0), then a substantial amount of the previous state value is reused. The size of the input vectors at the current state $x_t$ has less influence on the output. If $z_t$ is high, then the output at the current step is largely influenced by the current input $x_t$, and less by the size of the hidden layer vectors from the previous state $h_{t-1}$. $r_t$ functions as forget gate (or reset gate) hence allowing the cell to forget certain information of the state.

## 3. RELATED RESEARCH

With its better comparative computational power compared to its earlier derivatives, a typical GRU neural network is normally used to model streams of fast data such as those found in online deep learning as well as other resource-critical applications. Several research studies approach the optimization of the classical GRU from different angles. In their separate studies in batch normalization of RNNs and LSTM networks, C´esar L. et al., [5] and Tim Cooijmans et al., [8] confirmed that normalized neworks out-perform the un-normalized ones in convergence rates. Improved results of path-normalized RNNs were reported by Behnam Neyshabur et al., [6] for rectified linear unit (ReLU).

Weight initialization was reported to improve the performance of an LSTM network Quoc V. Le et al., [15]. Re-inforced learning was experimented for an LSTM and found to exhibit average performance on a corpus of songs database Natasha Jaques et al., [14]. The professor forcing algorithm was experimented by Alex Lamb et al., [11] and found to out-perform teacher forcing on RNNs. C´esar L. et al., [5] performed a dropout experiment on a typical LSTM and found it to have improvements over other networks.

The experimental results of feature pre-training on TimeNet was reported by Apeksha Nagesh Shewalkar [3] and and found to compare well with most networks with improved convergence rates and parameter tuning. David Krueger et al., [9] experimented on zone-out and dropout regularization techniques for regular RNNs on a MNIST dataset and found them to exhibit improvements on the chosen task. While reporting on an optimization technique based on adaptive learning rate Kingma D. P. and Ba J. L [13] claimed that adaptive learning was better suited for training recurrent networks as compared to gradient descent. In their work, Bao W et al., [4] constructed a time-series model based on stacked of autoencoders for the LSTM network and reported improvements over regular networks.

Outside optimization, several research studies have reported on modeling the time series problem. Dymitr Ruta et al., [22], carried research in time series ensembles while Tamal Datta Chaudhuri et al., [17] experimented on model based approach to time series modeling. The works of Enzo Busseti et al., [10] and Thiyanga S Talagala et al., [18] extended deep learning research for time series tasks in their individual studies. The results of a time series experiment are reported by G'abor Petneh'azi [11] and that of a multi-dimensional time series using GRUs are reported by Yifan Guo et al., [19].

## 4. OUR APPROACH

Our approach in tuning the Gated Recurrent Unit (GRU) network is geared towards improving its representation power at the same time reaping from its efficient computational load.

The tuning activity involved:

1. Improving its representational power. Theoretically, it is possible to improve the representational power of a neural network using ensembles, Jasmine Collins et al., [12]. In our work, we experiment on this through stacked ensembles of a basic GRU network. Figure 4 below demonstrates the architecture of the stacked GRU ensemble.

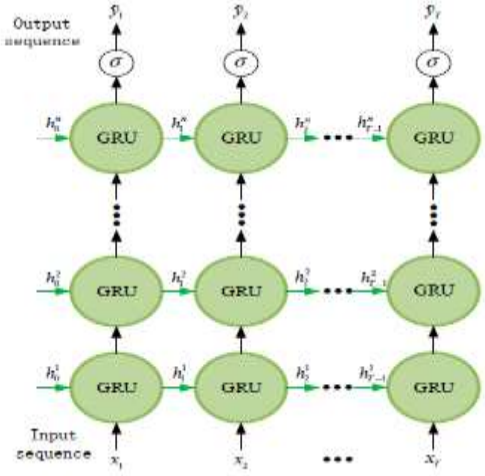


*Figure 4: Stacked GRU Architecture (Source: Self)*

From the figure above, the input of each GRU unit in the recurrent layer is the output of the hidden layer of the upper layer GRU unit changing the equations in 1 to:

$$\left.\begin{aligned} z_t^i &= \sigma\left(W_z^i \cdot \left[h_{t-1}^i, h_t^{i-1}\right]\right) \\ r_t^i &= \sigma\left(W_r^i \cdot \left[h_{t-1}^i, h_t^{i-1}\right]\right) \\ \tilde{h}_t^i &= \tanh\left(W^i \cdot \left[r_t^i \odot h_{t-1}^i, h_t^{i-1}\right]\right) \\ h_t^i &= z_t^i \odot h_{t-1}^i + \left(1 - z_t^i\right) \odot h_t^{i-1} \end{aligned}\right\} \quad (2)$$

2. Improving its learning abilities through a careful selection of the activation function. In their reported study, Chigozie Enyinna Nwankpa et al., [7] emphasizes on the importance of using the right activation functions for any given task. In our case, the Sigmoid and hyperbolic tangent ($tan^{-h}$) were selected. Sigmoid typically constraints values to [0,1] while tan+h moderates to a scale of [-1,+1]. The two equations are shown in figure 5 below:

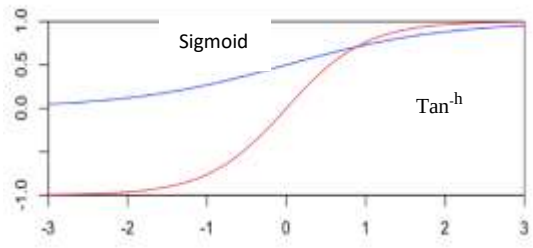


*Figure 5: Activation functions (Source: Self)*

Sigmoid: $\sigma(x)=\frac{1}{(1+\exp(-x))}$ (3)

$Tan^{-h}(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ (4)

3. Application of gradient clipping and recurrent layer dropout regularization strategies. One method of taming the vanishing gradient as well as the exploding gradient problem is gradient clipping, Sekitoshi Kanai et al., [16]. Another typically applied technique is dropout regularization on recurrent layers, Srivastava N. [20]. These techniques were added to improve the generalization power of the network. The diagram in figure 6 below demonstrates dropout effect in a neural network.

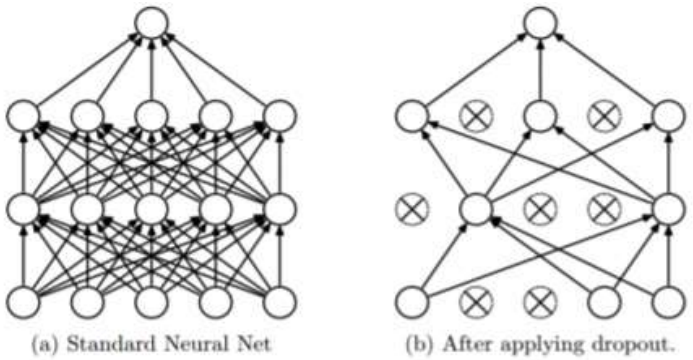


Figure 6: Dropout Effect in Neural Network (Source: [21])

## 5. EXPERIMENT

### 5.1 Introduction

In this section, we present the experimental setup that includes the task description in 5.2, the dataset used in 5.3, the Deep Machine Learning (DML) platform selected in 5.4 and the experimental setup in 5.5.

### 5.2 Task Description

The DML task selected for this study is the regression modeling of time series data representing the number of airline passengers arriving at an international airport. This is a prediction problem where given a year and a month,

the task is to predict the number of international airline passengers arriving at an airport in units of 1,000. The data was collected over a period of 144 months. The time series prediction is phrased as a regression problem where given the number of passengers (in units of thousands) this month, last month and previous months, we determine the number of passengers the following month.

The initial pre-processing step is to convert the given dataset into the required window of several months in the past. For purposes of the experiment, a window of three (3) months was selected as it was found to optimize the results. In this regard, the first column contains four (4) months' (t-3) passenger count before the current month. Subsequently, t-2, t-1 and t represent the remaining window period up to the present month. The next month's (t+1) passenger count, is the target prediction. For the sake of this experiment, a combination of the various reported enhancement techniques for a typical GRU are selected and the performance measures of accuracy and root mean squared error (RMSE) examined.

### 5.3 Dataset

The dataset used for this experiment is publicly available for free from the DataMarket webpage as a .CSV downloadable file with the filename "*international-airline-passengers.csv*".

### 5.4 Deep Learning Platform

The selected development platform consisted of a set of Python DML libraries and frameworks available for the experiment under the permissive MIT license namely: Keras and Tensorflow. Tensorflow is one of the two numerical backend platforms in Python that provide the basis for Deep Learning research and development. Keras runs on Python 2.7 or 3.5 and can seamlessly execute on Graphical Processing Units (GPUs) and Central Processing Units (CPUs) based on available hardware the underlying frameworks.

### 5.5 Experiment

As noted earlier, the time series problem was phrased as a regression problem with a window size of three (3) recent time steps that were used to make the prediction for the next time step given the current time step. In this case the input variables are t-3, t-2, t-1, t and the output variable is t+1. The code below was used import all of the functions and classes used to model this problem in the Science Python (SciPy) environment within the Keras deep learning library.

```
# Importing the base Neural Network (i.e. GRU)
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import GRU
```

*Algorithm 1: Importing the base Neural Network*

With time series data, the sequence of values is important. The method that was used for purposes of stratified cross validation was to split the ordered dataset into train and test datasets. This was necessary in order get an idea of the skill of the model on new unseen data. The code below was used to calculate the index of the split point and separates the data into the training dataset with two thirds (2/3) or roughly 67% of the available observations used to train the model, leaving the remaining one third (1/3) or roughly 33% for testing the model.

```
# split into train and test sets
size_Train = int(len(dataset) * 0.67)
size_Test = len(dataset) - size_Train
train_Set, test_Set = dataset[0:size_Train,:], dataset[size_Train:len(dataset),:]
```

*Algorithm 2: Splitting dataset*

Next a function to create a new dataset was defined in accordance with the window size as described above. The function takes two arguments, the dataset which is a Python Number array that we want to convert into a dataset and the look-back which is the number of previous time steps to use as input variables to predict the next time period. This has the role to create a dataset where X is the number of passengers at a given time (t) and Y is the number of passengers at the next time (t + 1). The value of the look-back argument was set to three (3) to conform to the selected window size. The following code was applied to reshape the dataset by overriding the default look-back value with the selected window size as below.

```
# reshape dataset

look_back = 3

trainX, trainY = create_dataset(train_Set, look_back)

testX, testY = create_dataset(test_Set, look_back)
```

*Algorithm 3: Reshape dataset*

The effect of the above code on the first few rows of the dataset is seen in table 1 below.

Table 1: Reshaped Dataset

| S/No | X3 | X2 | X1 | X | Y |
|---|---|---|---|---|---|
| 1 | 112 | 118 | 132 | 129 | 121 |
| 2 | 118 | 132 | 129 | 121 | 135 |
| 3 | 132 | 129 | 121 | 135 | 148 |
| 4 | 129 | 121 | 135 | 148 | 148 |

The parameters that were found to optimize the network capacity are a hidden layer of 4 neurons, an output layer of 1 neuron, 100 epochs and a batch size of size 2. For this experiment we define a variable 'optimizer' for the training algorithm which was any of the following: ADAM, RMSPROP, ADADELTA, ADAGRAD, SGD, ESGD, NADAM and ADAMAX.

```
# create and fit the GRU model

model = Sequential()

model.add(GRU(8, input_dim=look_back, activation='relu'))

model.add(Dense(1))

model.compile(loss='mean_squared_error',

model.fit(trainX, trainY, epochs=100, batch_size=2, verbose=2)
```

*Algorithm 4: Fitting GRU*

*Table 2: Model Summary*

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_64 (Dense)             (None, 12)                48
dense_65 (Dense)             (None, 8)                 104
dense_66 (Dense)             (None, 1)                 9
=================================================================
Total params: 161
Trainable params: 161
Non-trainable params: 0
_________________________________________________________________
```

Once the model is fitted, the subsequent activity is to estimate its performance on the train and test datasets. The role of this is to provide a point of reference when comparing new models. The technique applied was the Mean Squared Error (MSE) and the Root Mean Squared Error (RMSE) as illustrated in the code below.

```
# Estimate model performance

trainScore = model.evaluate(trainX, trainY, verbose=0)

percTr = (1000.0-(math.sqrt(trainScore)))/10

print('Train Score: %.2f MSE (%.2f RMSE) Accuracy %.2f %%' % (trainScore, math.sqrt(trainScore), percTr))
```

*Algorithm 5: Estimate model performance*

## 6. RESULTS

Finally, predictions were generated using various optimization choice parameters starting with the bare (un-optimized) GRU model for both the train and test datasets to get a visual indication of the skill of the model. Table 3 below indicates the

model shape and the parameter information for the un-optimized GRU while Figure 7 shows output predictions on training and test sets for the bare GRU. The RMSE scores for both the training and test scores are 23.37 and 47.71 respectively.

*Table 3: GRU Network shape and parameters*

| **Layer (type)** | **Output Shape** | **Param #** |
|---|---|---|
| gru_24 (GRU) | (2, 4) | 72 |
| dense_17 (Dense) | (2, 1) | 5 |

Total params: 77
Trainable params: 77
Non-trainable params: 0

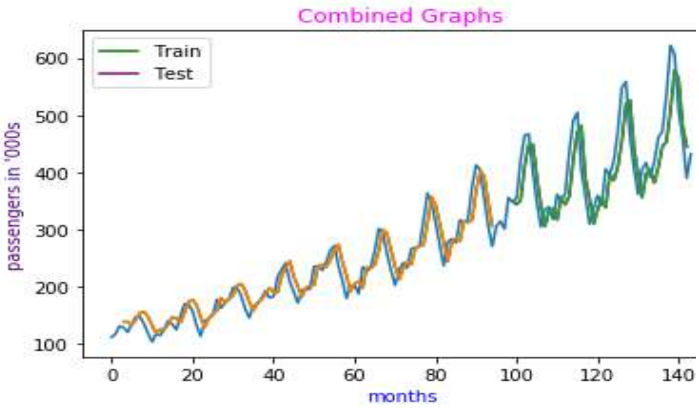


*Figure 7: Bare GRU training and test graph*

The initial optimization step was to create a stacked ensemble from the base GRU network. This procedure gave RMSE scores of 20.39 and 60.51 respectively for both the training and the test sets. Table 4 is an illustration of the network architecture while figure 8 is the graphical output for this experiment.

*Table 4: Stacked GRU ensemble architecture*

| Layer (type) | Output Shape | Param # |
|---|---|---|
| gru_16 (GRU) | (2, 3, 4) | 72 |
| gru_17 (GRU) | (2, 4) | 108 |
| dense_12 (Dense) | (2, 1) | 5 |

Total params: 185
Trainable params: 185
Non-trainable params: 0

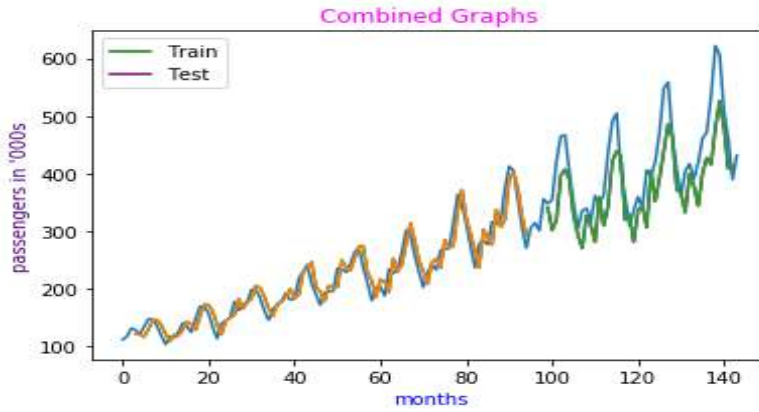


*Figure 8: Stacked GRU training and test graph*

Further, the stacked ensemble was enhanced using a dropout value of 0.2. This gave improved RMSE scores of 21.78 and 46.49 for the training and testing sets respectively. Table 5 illustrates this new network architecture while figure 9 is the resulting graphical output.

*Table 5: Stacked GRU ensemble with dropout*

| Layer (type) | Output Shape | Param # |
|---|---|---|
| gru_8 (GRU) | (2, 3, 4) | 72 |
| gru_9 (GRU) | (2, 4) | 108 |
| dropout_1 (Dropout) | (2, 4) | 0 |
| dense_8 (Dense) | (2, 1) | 5 |

Total params: 185
Trainable params: 185
Non-trainable params: 0

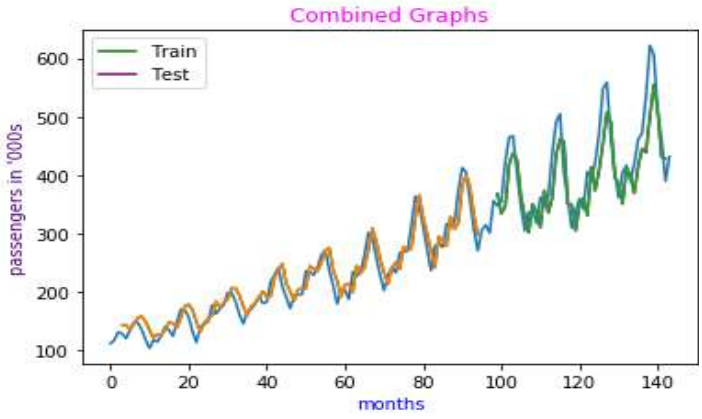


*Figure 9: Stacked GRU with dropout training and test graph*

Finally, the resulting network was a further enhanced by clipping at 1.0. This produced mild improvements on the test RMSE score of 46.48 while the training RMSE was 21.79. In table 6, the resulting network architecture is presented; while figure 10 summarizes the graphical output.

*Table 6: Stacked GRU ensemble with dropout*

| Layer type | Output Shape | Param # |
|---|---|---|
| gru_18 (GRU) | (2, 3, 4) | 72 |
| gru_19 (GRU) | (2, 4) | 108 |
| dropout_5 (Dropout) | (2, 4) | 0 |
| dense_13 (Dense) | (2, 1) | 5 |

Total params: 185
Trainable params: 185
Non-trainable params: 0

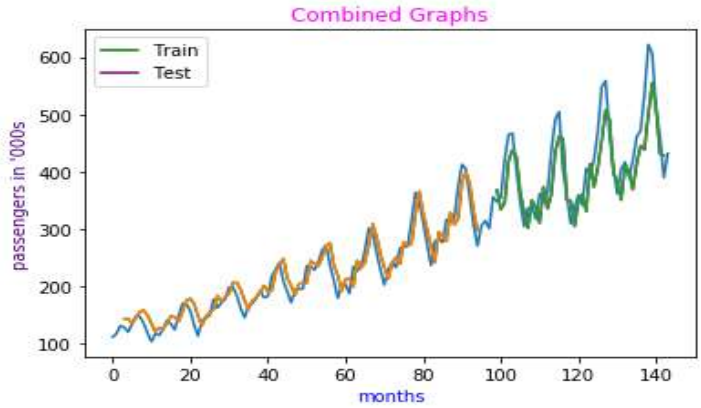


*Figure 10: Enhanced GRU training and test graph*

## 7. SUMMARY

### 7.1 Findings

Using the performance scores given by RMSE scores, we can see the average error on the training dataset reduced from 23.37 which is approximately 23 passengers to 21.79 which is approximately 22 passengers (in thousands per month) on the training set. Similarly, the average error on the unseen test set dropped from 47.71 equivalent to roughly 48 passengers to 46.48 which is approximately 47 passengers (in thousands per month).

### 7.2 Conclusion

In this research, we have experimented on enhancement techniques for specialized recurrent neural network – the gated recurrent unit (GRU) in a sequence data-modeling problem. From the

results above, we have been able to improve the network capacity especially on the test dataset. This is a clear indication that optimization procedures are key to training deep machine learning networks.

### 7.3 Future Work

In order to take full advantage of optimization procedures for the stated network, it is important to perform further experiments on value fine-tuning as well as experiment on additional optimization techniques. Some of the notable ideas would be to introduce model pre-training in order to take advantage of transfer learning. Additionally, it would also be important to perform a hyper-parameter search procedure in order to determine whether there other important architectural features that could be optimized.

## REFERENCES

*[1] Aditya R. (2018). Discovering Gated Recurrent Neural Network Architectures. PhD Dissertation*

*[2] Amit A. & Shubham P. (2017). Evaluation of Gated Recurrent Neural Networks on Deep Sentence Classification. International Journal of Computer Science Trends and Technology (IJCST) – Vol 5 Issue 4. Retrieved: http://www.ijcstjournal.org/volume-5/issue-4/IJCST-V5I4P26.pdf*

*[3] Apeksha N. (2018). Comparion of RNN, LSTM and GRU on Speech Recognition Data. MSc Dissertation.*

*[4] Bao W, Yue J. & Rao Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLoS ONE 12(7): e0180944. https://doi.org/10.1371/journal.pone.0180944*

*[5] Behnam N., Yuhuai W., Ruslan S. & Srebro N. (2016). Path-Normalized Optimization of Recurrent Neural Networks with ReLU Activations. arXiv:1605.07154v1 [cs.LG]*

*[6] C´esar L., Pereyra G., Brakel P., Zhang Y., Bengio Y. (2015).Batch Normalized Recurrent Neural Networks. arXiv:1510.01378v1 [stat.ML]*

*[7] Chigozie E., Ijomah W., Gachagan A. & Marshall S. (2018). Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. arXiv:1811.03378v1 [cs.LG]*

*[8] Cooijmans T., Ballas, N., César L., Ça˘glar G., & Courville A. (2017). Recurrent Batch Normalization. arXiv:1603.09025v5 [cs.LG]*

*[9] David K. et al., (2017). Zoneout: regularizing rnns by randomly Preserving hidden activations. arXiv:1606.01305v4 [cs.NE]*

*[10] Enzo B., Osband I, & Scott Wong. (2012). Deep Learning for Time Series Modeling. CS 229 Final Project Report.*

*[11] G'abor P. (2019). Recurrent Neural Networks for Time Series Forecasting. arXiv:1901.00069v1 [cs.LG]*

*Proceedings of Machine Learning Research 95:97-112, 2018 ACML*

*[12] Jasmine C., Jascha S. & Sussillo D. (2017). Capacity And Trainability In Recurrent Neural Networks. Published as a conference paper at ICLR 2017. arXiv:1611.09913v3 [stat.ML]*

*[13] Kingma D. P., & Ba J. L., (2015). Adam: A Method for Stochastic Optimization, Published as a conference paper at ICLR Priyanka G., Pankaj M., Lovekesh V. & Gautam S. (2016). Using Features from Pre-*

*trained TimeNet for Clinical Predictions. Under review as a conference paper at ICLR*

*[14] Natasha J., Shixiang G., Richard E., Douglas E. (2017). Tuning recurrent neural networks with reinforcement Learning. Under review as a conference paper at ICLR*

*[15] Quoc V. L., Navdeep J., Hinton G. (2015). A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. arXiv:1504.00941v2 [cs.NE]*

*[16] Sekitoshi K., Yasuhiro F., Sotetsu I. (2017). Preventing Gradient Explosions in Gated Recurrent Units. 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA*

*[17] Tamal D. & Chaudhuri I. (2016). Artificial Neural Network and Time Series Modeling Based Approach to Forecasting the Exchange Rate in a Multivariate Framework. Journal of Insurance and Financial Management, Vol. 1, Issue 5, 92-123*

*[18] Thiyanga S., Rob J. & Athanasopoulos G. (2018). Meta-learning how to forecast time series. Working paper. Retrieved: http://business.monash.edu/econometrics-and-businessstatistics/research/publications*

*[19] Yifan G. et al., (2018). Multidimensional Time Series Anomaly Detection: A GRU-based Gaussian Mixture Variational Autoencoder Approach. Proceedings of Machine Learning Research 95:97-112*

*[20] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. & Salakhutdinov R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. (15)1929-1958*