

# A Quantitative Performance Analysis of LSTMs, GRUs, Single & Bidirectional RNNs in Classification & Regression Problems

Anirudh Sekar

## Abstract:

Single Directional RNNs were invented in the 1980s, with the main feature being that these networks could utilize data from previous timestamps to make its next prediction, essentially allowing the sequencing of data to become relevant to the prediction the model makes. In the 1990s, the Long Short-Term Memory Recurrent Neural Network was invented, allowing for a more complex analysis of the sequential data, and provided a solution to the Vanishing and Exploding Gradient Problem, faced by Single Directional RNNs. Around this time, the Bidirectional RNN was introduced, capturing both forward and backward temporal dependencies in sequential data. Recently, in 2014 the Gated Recurrent Unit was invented, allowing for complex analysis like the LSTM, but utilizes a simpler architecture, making it much more efficient. With these four different types of Recurrent Neural Networks each with their own sets of advantages and disadvantages, it may seem intimidating or even overwhelming to decide which architecture is the best fit for any given problem at hand. By conducting a quantitative performance analysis between all these different architectures, we will be able to gain an understanding how each architecture handles sequences of data and long-term dependencies and develop a more comprehensive understanding of their capabilities. This understanding can lead to more informed decisions when selecting a model architecture for specific tasks such as time series forecasting and image classification.

## Overview of Single Directional RNNs:

A Single Directional Recurrent Neural Network (RNN) [34] is a type of neural network architecture that is mainly used to detect patterns in a sequence of data, such as handwriting, genomes, text, or numerical time series. RNNs have cycles and transmit information back into themselves called feedback loops, allowing them to consider previous inputs, unlike Feedforward Neural Networks [38] [37]. It is important to consider that an RNN creates a prediction for each timestep, but it just doesn't output that prediction until all the sequential data is used.

Calculations for an RNN follow the formula below where  $f_1$  and  $f_2$  represent activation functions:

$$a_t = f_1(W_a a_{t-1} + W_x x_t + b_a) \quad [55]$$

$$y_t = f_2(W_y a_t + b_y)$$

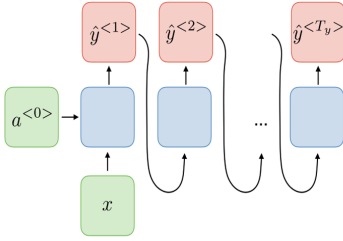
## Vanishing and Exploding Gradient Problem:

The vanishing-exploding gradient problem refers to issues in recurrent neural networks where gradients become too small (vanishing) or too large (exploding), hindering the training process. Techniques like normalized initialization and intermediate normalization layers have been used to address these problems, but they are not always effective and can lead to other issues [31]. This problem happens due to the weight of the feedback loop on each subsequent

iteration. When weight is greater than 1, the gradient grows larger and larger, causing it to “explode.” On the other hand, if the weight is less than 1, the gradient gets smaller and smaller, causing it to “vanish.”

### Architecture:

The architecture of a Single Directional Recurrent Neural Network is shown below, having the traits discussed earlier such as a feedback loop connecting to the operation in the following timestamp:



[3]

### Overview of Bidirectional RNNs:

A bidirectional recurrent neural network [24] is a type of neural network that can be trained in both positive and negative time directions simultaneously, allowing it to use input information beyond a preset future frame. It has been shown to provide better results in regression and classification experiments compared to other approaches [24] [40].

### Training:

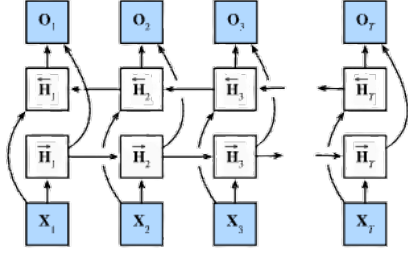
Training a Bidirectional Neural Network requires training for both the forward hidden state and the backward hidden state, causing it to take more time to train a Bidirectional RNN than a Single Directional RNN. The formulas for updating the forward and backward hidden state are as follows:

$$\overrightarrow{H}_t = \phi \left( X_t W_{xh}^{(f)} + \overrightarrow{H}_{t-1} W_{hh}^{(f)} + b_h^{(f)} \right)$$

$$\overleftarrow{H}_t = \phi \left( X_t W_{xh}^{(b)} + \overleftarrow{H}_{t+1} W_{hh}^{(b)} + b_h^{(b)} \right) \quad [53]$$

### Architecture:

We simply implement two unidirectional RNN layers chained together in opposite directions and acting on the same input. For the first RNN layer, the first input is  $x_1$  and the last input is  $x_t$ , but for the second RNN layer, the first input is  $x_t$  and the last input is  $x_1$ . To produce the output of this bidirectional RNN layer, we simply concatenate together the corresponding outputs of the two-underlying unidirectional RNN layers, represented by the architecture shown below.



[53]

### Overview of LSTMs:

Long Short-Term Memory Networks (LSTMs) [16] are an expansion of the “Hopfield” network, the first recurrent neural network. LSTMs were introduced by Jürgen Schmidhuber and Sepp Hochreiter in 1997, greatly improving the efficiency and the practicality of recurrent neural networks [1]. Today, LSTMs are known for their capabilities within prediction of complex sequential data, and they are frequently used in Time Series Forecasting, and was one of the most influential inventions within the field of Deep Learning and Neural Networks. LSTMs utilize a gated architecture containing three gates: the input gate, the output gate, and the forget gate, which help the model handle long-term dependencies in sequential data [28] [11] [25]. By incorporating both long-term memory and short-term memory, called the cell state and hidden state, respectively, LSTMs effectively mitigate the vanishing and exploding gradient problem, a challenge encountered by single-directional RNNs.

### Forget Gate:

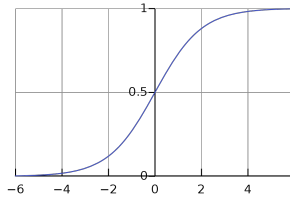
The forget gate in the LSTM model plays a crucial role in determining which information from the previous cell state should be retained or discarded. It computes a value between 0 and 1 based on the previous hidden layer state and the current input to determine what information to keep or discard, with 1 indicating retention and 0 indicating removal [46] [27] [49]. This value is calculated through a series of mathematical operations, modeled by the equation shown below.

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f) \quad [47]$$

### Sigmoid Function:

The sigmoid function puts the values of the data between zero and one, making the function very applicable and useful in calculations regarding the LSTM, but it is also useful in problems such as Logistic Regression and Logistic Growth [29].

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



[29]

### Input Gate:

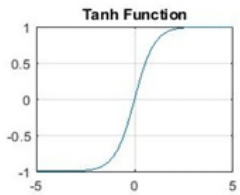
The input gate of the LSTM controls the flow of information into the cell state. It does this by determining which parts of the input are important to keep for future cell state updates [29]. This is done through a series of mathematical calculations modeled by the equation shown below:

$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i) \quad [29]$$

By using the weights of the hidden state and input, adding it to the bias, and then running it through the sigmoid activation function, the LSTM can modify the cell state, making it useful to analyze future data. This process helps the LSTM model learn long-term dependencies in sequential data.

### Tanh Function:

The Hyperbolic Tangent function (Tanh) helps solve the problem faced by the sigmoid function of not being centered at zero by having a range between -1 and 1. The Tanh function is slightly more popular than the sigmoid function for multi-layer neural networks due to better training performance [13]. The Tanh function is used to calculate the cell state, making this function very impactful and useful within the Deep Learning and Machine Learning space. The formula for the Tanh Function and its graph is shown below:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad [29]$$
The graph shows the Tanh function, which is an S-shaped curve centered at the origin (0,0). The x-axis ranges from -5 to 5, and the y-axis ranges from -1 to 1. The curve approaches -1 as x goes to negative infinity and approaches 1 as x goes to positive infinity. The title of the graph is "Tanh Function".

### New Information:

The New Information needed to be passed to the long-term memory is represented by the formula below:

$$N_t = \phi(x_t * W_c + h_{t-1} * U_c) \quad [36]$$

The new information that needs to be passed to the cell state is a function of a hidden state at the previous timestamp and input at the current timestamp. By using the tanh activation function, we can store the value of the new information between -1 and 1. If the value is negative, the information is subtracted from the cell state, and if the value is positive, the information is added to the cell state at the current timestamp [36]. However, the new information won't be added directly to the cell state. Instead, it goes through this next formula, creating the new cell state:

$$c_t = x_t * f_t + i_t * N_t \quad [36]$$

### Output Gate:

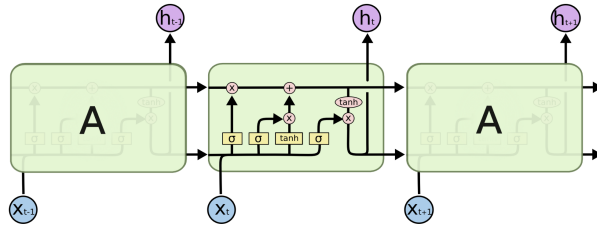
The output gate of the LSTM controls the information that is passed onto the next time step. The output gate value is also important to determine the value of the new hidden state and the new cell state. The output gate function is calculated as shown below:

$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o) \quad [29]$$

This output gate affects the possibilities of the new hidden state and cell state, which is important to note for future timestamps within the LSTM. The way the new hidden and cell states are developed is based around the values of the output gate, input gate, and forget gate, essentially utilizing all the previous operations that we have performed to create new values.

### Architecture:

The architecture of an LSTM looks like the diagram shown below, containing all the features that we stated before:



[30]

### GRUs:

#### Overview:

The gated recurrent unit (GRU) [9] was proposed to make each recurrent unit adaptively capture dependencies of different time scales. Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having separate memory cells [10]. Unlike LSTMs, GRUs only have 2 gates. The GRU neural network combines the forget gate and input gate into the update gate, simplifying the architecture and reducing computational costs. It has an input gate that determines what new information should be stored in the cell state. The gating signal ranges from 0 to 1, controlling forgetting and inputting for generating output [22] [18] [52]. In addition to this, the GRU only has a hidden state, unlike the LSTM which has both a cell state and a hidden state, as stated previously. The update gate value is crucial for deciding how much of the previous hidden state should be updated with new information. This helps the GRU network to effectively retain important information and discard irrelevant details, but in a more efficient way than the LSTM.

**Update Gate:**

The update gate of a GRU receives input and sorts them out before handing over to memory. If the gate reaches zero, the hidden layer is the result of MLP, standing for Multi-layer Perceptron, another term for a feedforward artificial neural network, and the present ones will be assigned as input [52] [32]. The main purpose of the update gate is to calculate the amount of required information to add to the current content of information, the hidden state [52]. The formula used to calculate the value of the update gate is shown below.

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \quad [52]$$

**Reset Gate:**

The purpose of the reset gate is to calculate the amount of data to be removed. Other than resetting this data, the reset gate also issues memory spaces to store values of the hidden state, update gate, and future hidden states [52]. The formula to calculate the reset gate is very similar to the update gate, shown below.

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \quad [52]$$

**Candidate State:**

The Candidate State is the proposal for the new hidden state at the current time step. It uses the reset gate to filter parts of the hidden state and combines it with the input. This candidate state is then used with the update gate to understand how much of the old memory to keep and how much of the Candidate State to add to the hidden state. The Candidate State is calculated using the reset gate, as shown below.

$$\tilde{h}_t = \phi(W_h * x_t + U_h * (h_{t-1} * r_t)) \quad [22]$$

(Note: All matrix products are Hadamard Products, element-wise multiplication)

**Hidden State:**

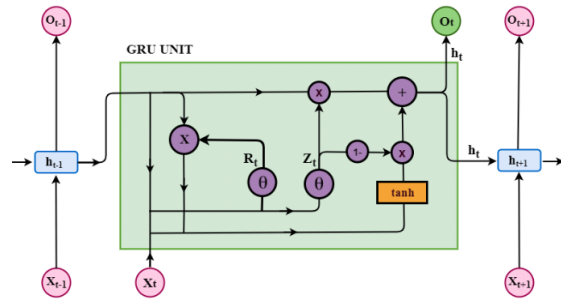
The reason why we calculated the Update Gate, Reset Gate, and Candidate State was to change the Hidden State. This hidden state will be what is used in the next iteration. If there is not another iteration, the hidden state is the predicted value returned by the Gated Recurrent Unit. The new hidden state is calculated using the following formula shown below.

$$h_t = (1 - z_t)h_{t-1} + z_t * \tilde{h}_t \quad [22]$$

(Note: All matrix products are Hadamard Products, element-wise multiplication)

### Architecture:

The architecture of a GRU looks like the diagram shown below, containing all the features that we stated before:



[20]

### Classification:

Classification in machine learning is the process of categorizing data into different classes or categories based on certain features or attributes. It involves training a model to predict the class labels of new data points based on the patterns learned from the training data [15]. Multi-class classification is a subset of classification where there are more than 2 categories within the data.

### Regression:

Regression is a statistical tool used to investigate relationships between variables, such as the causal effect of one variable on another. It is often used to estimate the quantitative effect of causal variables on the influenced variable and assess the statistical significance of these relationships. Multiple regression involves expressing the response variable as a function of two or more explanatory variables, allowing for the analysis of interactions between variables and correlation among them [42] [8] [41].

### Loss Functions:

Common loss functions for classification problems include mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and cross-entropy. These functions are optimized for speed and are commonly used in neural networks for classification tasks [14].

### Activation Functions and Optimizers:

Common activation functions for both classification and regression problems include Sigmoid, Tanh, Step Functions, and ReLU [4] [35] [5] [19] [21]. These functions introduce nonlinearities into neural networks and are commonly used in regression tasks. Common optimizers for classification models in Tensorflow include Adam and Adadelta, which have been found to be the most successful optimizers for various deep learning architectures [35]

[5], while common optimizers for regression problems in TensorFlow include Adam, RMSprop, and Stochastic Gradient Descent (SGD) [51][5][33].

### **Adam Optimizer:**

The Adam optimizer is a popular adaptive optimization method widely used in deep learning frameworks. It has a very fast convergence and there are variants of Adam such as Adabound, RAdam, and Adabelief that have been proposed and show better performance than the original Adam optimizer [7] [50]. The Adam optimizer performs stepwise optimization on a random loss function. It uses gradient update rules for parameters to reach the optima, with decay rates for moving mean index and a learning rate [45] [50]. The formula for the Adam Optimizer is as follows:

$$x_t = \delta_1 * x_{t-1} - (1 - \delta_1) * g_t$$

$$y_t = \delta_2 * y_{t-1} - (1 - \delta_2) * g_t^2 \quad [26]$$

$$\Delta \omega_t = -\eta \frac{x_t}{\sqrt{y_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta \omega_t \nu$$

### **Normalization**

Normalization in Machine Learning refers to scaling or transforming data to ensure each feature contributes equally. Different normalization techniques like Min-Max, Z-Score, and Unit Normalization can be used to enhance accuracy in ML algorithms. Some algorithms may not be significantly affected by normalization. Data normalization is crucial for improving data quality and algorithm performance in machine learning [2] [12] [44].

### **Min-Max Normalization:**

Min-Max Scaling is a data scaling algorithm widely used in Machine Learning. It linearly transforms original data to a specified interval, typically [-1, 1] or [0, 1]. However, it can be sensitive to outliers and may cause issues when unseen/testing samples fall outside the training data range. The Min-Max algorithm is one of the two commonly used data scaling algorithms in Machine Learning, with the other being the Z-score algorithm [6] [43].

### **Data:**

To conduct our analysis, we utilized 2 classification problems and 2 regression problems, each with one large dataset and one small dataset: One performing Image Classification using the MNIST Dataset [48] and one performing multi-class classification on the Iris Dataset [17]. For our regression problems, we used three different types of time-series forecasting, one problem involving weather temperature [23] and another involving the prediction of shampoo sales [39].



## Methodology:

When conducting our analysis, we utilized industry-grade frameworks including TensorFlow to create neural networks, Pandas to process data, NumPy to deal with the heavier mathematics, and Matplotlib to visualize our results. In addition to this, we made sure to test each of these models in the same way. In each of our trials, all 4 of our models: LSTM, GRU, Bidirectional RNN, and Single Directional RNN, have the same number of epochs, batch size, optimizer, and learning rate. For each of the problems, our models used the Adam Optimizer, a batch size of 8, and 25 epochs except for the MNIST dataset, in which we used 10 epochs instead of 25. For our regression problems, we are using the Mean Squared Error Loss Function to train our model. For our classification problems, our loss function is Sparse Categorical Cross Entropy.

## Metrics:

All the models that we tested were: Bidirectional RNN, Single-Directional RNN, Long Short-Term Memory RNN, and the Gated Recurrent Unit. To measure the performance of the models in the regression problems, we utilized common regression metrics, including Mean Squared Error, Mean Absolute Error, Root Mean Squared Error, and a graphical representation of the predicted and actual data. The amount of time taken for the model to train (execution time) was also considered for all the models, for both regression and classification. For classification, metrics used, in addition to training time, were the Accuracy Score, Logarithmic Loss, Multilabel Confusion Matrix, and the F1 score. To see which model performed the best in each metric, we have highlighted the best performing model for each metric in all the tables following.

## Performance on the MNIST Dataset:

	SINGLE DIRECTIONAL	BIDIRECTIONAL	LSTM	GRU
ACCURACY	0.8063	0.9405	0.9887	0.989
F1	0.8029494874	0.9400167322	0.9886386398	0.9889473488
LOG LOSS	0.6103989579	0.1917382062	0.04162692352	0.04007196339
TIME	481.6236937	1589.727481	1409.119676	1306.045482

The confusion matrices for each of the models on the MNIST dataset are shown below. (from left to right)

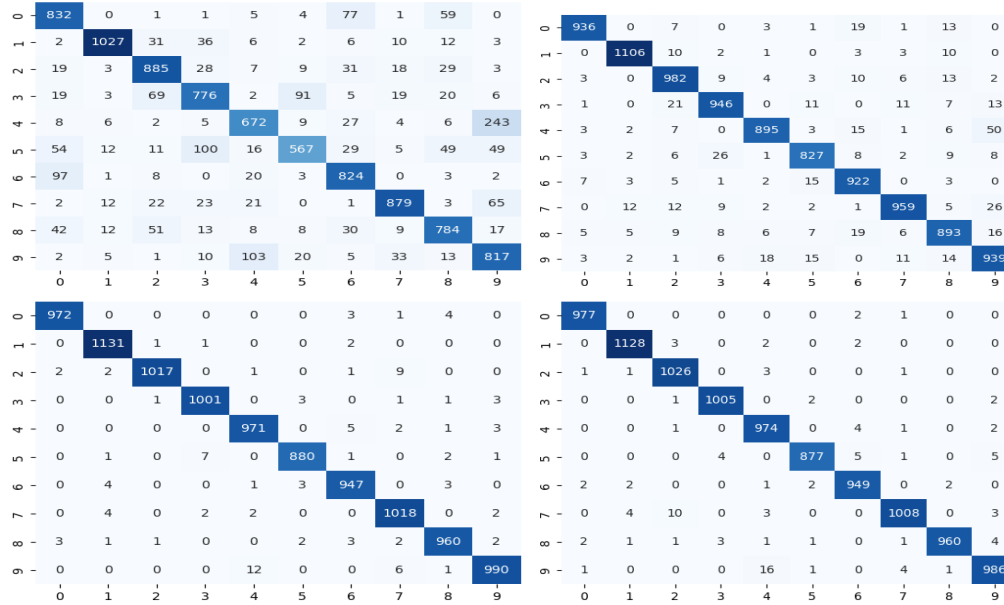


Figure 1: (top left) Confusion matrix for the performance of the Single Directional RNN (top right) Confusion Matrix of the Bidirectional RNN's performance (bottom left) Confusion Matrix for the performance of the LSTM (bottom right) Confusion Matrix for the performance of the GRU

#### Performance on the IRIS Dataset:

	SINGLE	BIDIRECTIONAL	LSTM	GRU
ACCURACY	1	1	0.9666666667	0.9333333333
F1	1	1	0.9666666667	0.9333333333
LOG LOSS	0.01974530316	0.003212514221	0.07997073092	0.2122203226
TIME	2.84154167	5.139798294	5.543126582	5.062231292

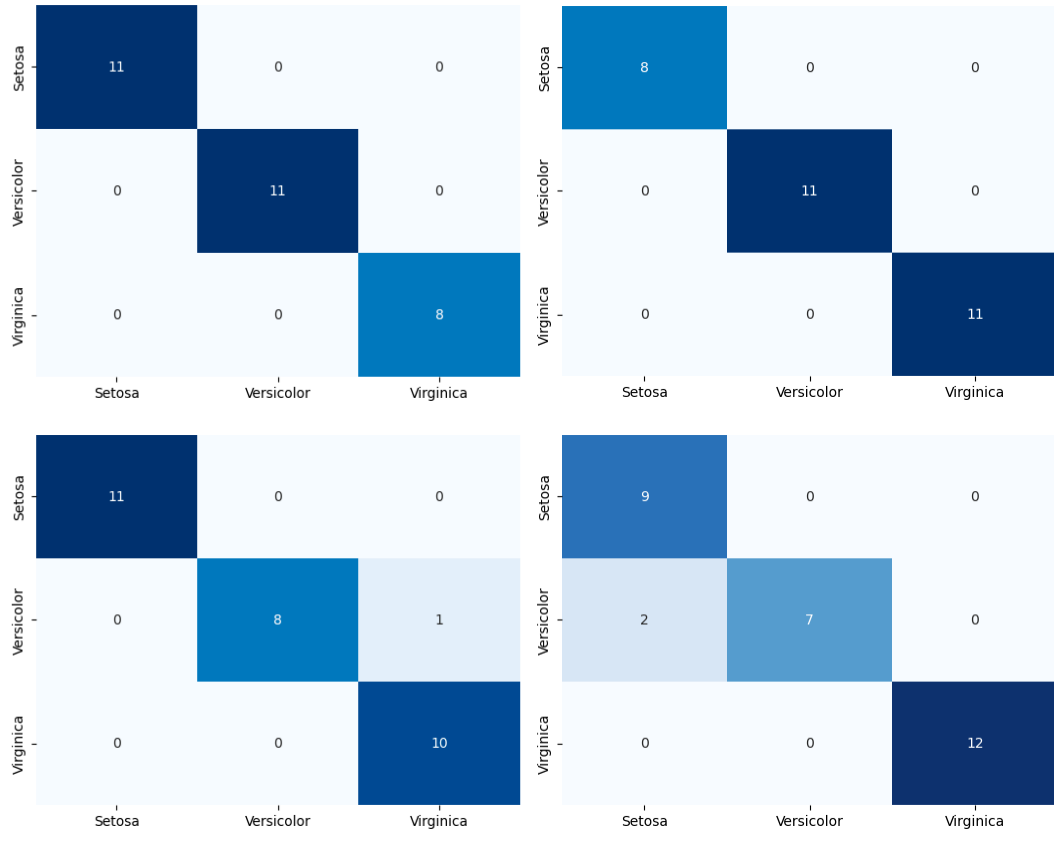


Figure 2: (top left) Confusion matrix for the performance of the Single Directional RNN (top right) Confusion Matrix of the Bidirectional RNN's performance (bottom left) Confusion Matrix for the performance of the LSTM (bottom right) Confusion Matrix for the performance of the GRU

#### Performance on the Shampoo Sales Dataset:

	SINGLE	BIDIRECTIONAL	LSTM	GRU
RMSE	0.3236235251	0.2888664952	0.1537707698	0.1817329278
MSE	0.104732186	0.08344385205	0.02364544963	0.03302685703
MAE	0.2759165327	0.2461796343	0.1321951168	0.1499704509
TIME	2.156040836	3.45449025	4.429650253	4.553116706

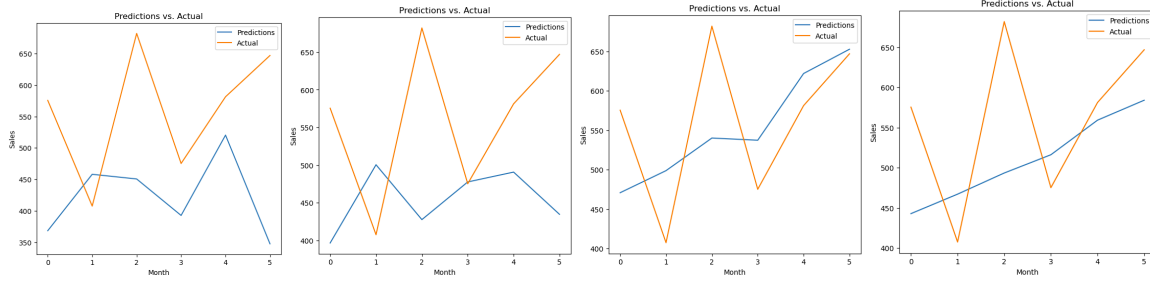


Figure 3: (from left to right) Diagrams of the Predictions made by the Single Directional RNN, Bidirectional RNN, LSTM, and GRU on the Shampoo Sales Dataset.

### Performance on the Austin Weather Temperature Dataset:

	SINGLE	BIDIRECTIONAL	LSTM	GRU
RMSE	0.1180091724	0.09521841258	0.08931683749	0.0905442842
MSE	0.01392616518	0.009066546336	0.007977496833	0.008198267401
MAE	0.08847029507	0.06739126891	0.06471606344	0.064629208
TIME	10.81444959	19.58799583	20.01417925	21.10600954

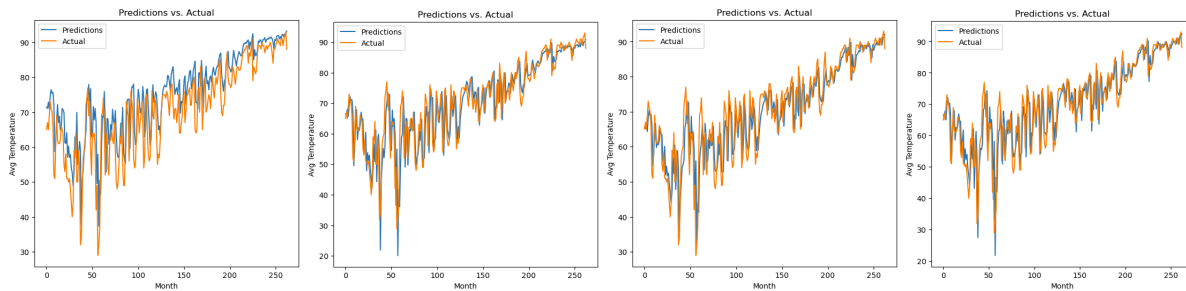


Figure 4: (from left to right) Diagrams of the Predictions made by the Single Directional RNN, Bidirectional RNN, LSTM, and GRU on the Shampoo Sales Dataset.

### Analysis:

With all our data being collected, we can start to answer questions using the data that we have collected.

1. Is there a model that performs significantly better than the others in each task?
2. Which type of model performs the best with a small amount of data?
3. Which model has the best time to performance ratio for each task?
4. Which model architecture performs the best for regression and which architecture is the best for classification?

5. Which model architecture would be the best overall for both regression and classification problems in terms of both performance and time?

### **Is there a model that performs significantly better than the others in each task?**

In each task, there were specific models that exceeded the performance of the other models, but we also need to analyze the frequency of a specific model architecture having high performance. In the Shampoo Sales Dataset, the LSTM had the lowest Mean Absolute Error at around 0.132 and the GRU had the 2nd lowest Mean Absolute Error at around 0.149. In the Austin Weather Temperature Dataset, the GRU attained the lowest Mean Absolute Error with a score of 0.646, with the LSTM having the 2nd lowest error receiving a MAE of 0.647. Within the MNIST dataset, the GRU performed with the best accuracy, having an accuracy of 98.9% while the LSTM performed with the 2nd best accuracy with a score of 98.87%. Finally, in the Iris Dataset, the Bidirectional Model performed the best with a perfect accuracy and a Logarithmic Loss of 0.003 and the Single Directional Model performed the 2nd best with a perfect accuracy and a logarithmic loss of 0.019. Based on these results, we can see that there wasn't any model that performed significantly better than the others in every task due to the proximity of the best performing and second-best performing model. Something to consider however is the frequency of each model architecture performing either the best or the 2nd best in each test. Out of the 2 classification tests and 2 regression tests, the GRU performed the best in 1 classification task, 1 regression task, and performed 2nd best in the other regression task. The LSTM performed the 2nd best in 1 classification task, 1 regression task, and performed the best in the other regression task. Because both the GRU and LSTM both have 3 test cases in which they both place top 2, they are easily the most consistent models in terms of sole accuracy. Though both models placed top 2 in performance for 3 out of the 4 test scenarios, the GRU has been the best performing model more frequently than the LSTM, allowing for the GRU to have a slight edge in terms of performance over the LSTM. Though the GRU does have a slightly better performance than the LSTM across both regression and classification problems, there was no model that performed significantly better than the others in every single task.

### **Which type of model performs the best with a small amount of data?**

Two of the datasets that each of the models were tested on had a relatively small amount of data. These two datasets were the Iris Dataset and the Shampoo Sales Dataset. On the Shampoo Sales Dataset, the LSTM performed the best with the GRU performing the 2nd best. On the Iris Dataset, the Bidirectional Model performed the best with the Single Directional performing the 2nd best. Because this varying performance depends on the problem, it is tough to pinpoint one specific model that performs the best on both problems with a small amount of data. On the Iris Dataset, the LSTM performed 3rd best, which is the same as the Bidirectional Model on the Shampoo Sales Dataset. Though the placements of the LSTM and Bidirectional Model were the same, we also must account for the sheer accuracy pulled off by the Bidirectional model, having perfect predictions and a Log Loss of 0.003. This is extraordinary compared to what the LSTM did in the Shampoo Sales dataset, so the Bidirectional Model has the slight edge compared to the LSTM, meaning that the Bidirectional Model performs the best with a small amount of data with the LSTM coming in 2nd in terms of performance with small amounts of data.

### **Which model has the best time to performance ratio for each task?**

In terms of the relationship between time and performance, each task had different models that exceeded the others. To measure this effectively, for the regression tasks, we used the median value within the Mean Absolute Error to Time Ratio between all the models and for classification tasks, we calculated the Accuracy to Time Ratio. We used the median for the error to time ratio because when Error is divided by the Time, both the error and the time need to be minimized, so the highest and lowest values will not suffice. To find the value that is best minimizing both Time and Mean Absolute Error, we will use the Median of the error ratio between all the models in the dataset. In the Austin Weather Temperature Dataset, the Median value of the Error to Time ratio was achieved in between the Bidirectional RNN and the LSTM, each contributing to the median value of 0.0033. For the Shampoo Sales Dataset, the median value was received in between the Bidirectional RNN and the GRU, with a median value of 0.052 being attained. For the classification models, the highest accuracy to time ratio for the MNIST dataset was achieved surprisingly by the Single Directional RNN, with a score of 0.0017 due to a fast execution time with a solid accuracy. This also held true for the Iris Dataset, where the Single Directional Model also had the highest accuracy to time ratio with a score of 0.35. This is primarily because the Single Directional Model executed at almost double the speed compared with the other models and maintained average accuracy, allowing for it to excel in comparison to the other models. The Bidirectional Model and the Single Directional Model performed exceptionally in terms of this metric, but the Single Directional Model outperformed the Bidirectional Model in most of the cases in which the Single Directional Model performed the best. In addition to this, when the Bidirectional Model performed with the best Time-Performance Ratio, the ratio for the Single Directional Model was not far behind. Because of this, the Single Directional Model has the edge in terms of the Time-Performance metric.

### **Which model architecture performs the best for regression and which architecture is the best for classification?**

In our tests within our regression tasks, we have previously identified that in the Shampoo Sales Dataset, the LSTM performed the best with the GRU performing the 2nd best and in the Austin Weather Temperature Dataset, the GRU attained the lowest Mean Absolute Error, and the LSTM attained the second lowest Mean Absolute Error. For the classification tasks, the Bidirectional Directional Model performed the best on the Iris Dataset and the GRU had the highest accuracy on the MNIST dataset. On the Austin Weather Temperature Dataset, the difference in error between the LSTM and GRU was only 0.0001, while the error difference on the Shampoo Sales Dataset was much higher at a value of 0.01. Because of this, it is safe to say that the LSTM is the better model for the regression problems because it performs better with small amounts of data and performs very close to the best with large amounts of data as well. For the classification problems, specifically the MNIST dataset, the GRU performed with the best accuracy and the LSTM performed with the 2nd best accuracy. Finally for the Iris Dataset, the Bidirectional RNN performed the best with the Single Directional RNN coming in 2nd. Because all the models placed either the best or 2nd best in either the Iris Dataset or MNIST dataset, it is tough to easily pick the best performing model. Though it is tough, it is fair to say that the Bidirectional Model performed the best overall in terms of classification performance because the Bidirectional Model performed the best on the Iris Dataset and received very good accuracy on the MNIST, with only a 4% difference between the best performing model and the Bidirectional RNN

on the MNIST dataset. In addition to this, the Bidirectional RNN performed the best on the Iris Dataset, showing that it is a good model for tasks with a small amount of data. Because of the strong performance of the Bidirectional Model on both the MNIST dataset and Iris Dataset, it did the best overall on the classification problems compared to the other model architectures and the GRU came in 2nd for classification problems.

**Which model architecture would be the best overall for both regression and classification problems in terms of both performance and time?**

To calculate which model architecture would be the best overall, we created a formula to calculate a composite score based on performance and time. We created four categories: best performance on Small Data, Time Performance Ratio, Best Regression, and Best Classification. Each of our categories had a weight, all of which were 25% of the overall score. We also only considered the top 2 performing within each of these metrics so that we could utilize the best performing models within each dataset. The formula that we used to calculate the model's overall score is shown below.

$$Model\ Score = \sum_{i=1}^4 \frac{5 - m_{p_i}}{16}$$

Utilizing this model score metric, where  $m_{p_i}$  represents the model placement on that category, either 1 or 2 because we are only using the top 2 performing models in each category, we can make calculations showing the overall score of each model. To recap which models performed first or second in each calculation, we can refer to the table below.

	Best Performing	2 <sup>nd</sup> Best Performing
Performance on Small Data:	Bidirectional	LSTM
Time-Performance Ratio:	Single Directional	Bidirectional
Best Regression:	LSTM	GRU
Best Classification:	Bidirectional	GRU

**Scores:**

First, we will measure the Single Directional RNN's performance overall. The Single Directional RNN only placed in the top 2 for the Time Performance Category, where the Single Directional RNN placed the best in classification and the Bidirectional Model placed the best in regression. The weight for this category was 25%, and the Single Directional Model placed 1st, getting the Single Directional Model a score of 0.25. For the Bidirectional Model, points were accumulated from Performance on Small Data, Time Performance Ratio, and Best Classification. The Bidirectional Model's performance on these categories gave the Bidirectional Model a score of 0.6875. For the LSTM, points were awarded in the Performance on Small Data Category and the Best Regression Performance Category. From these categories, the LSTM performed with a score of 0.4375, tying with the value of the Bidirectional Model. Finally, the GRU earned points from the Best Regression and Best Classification Categories, earning a final composite score of 0.3725. Based on our data, we can make the following projection.

### Overall Performance of each Model Architecture

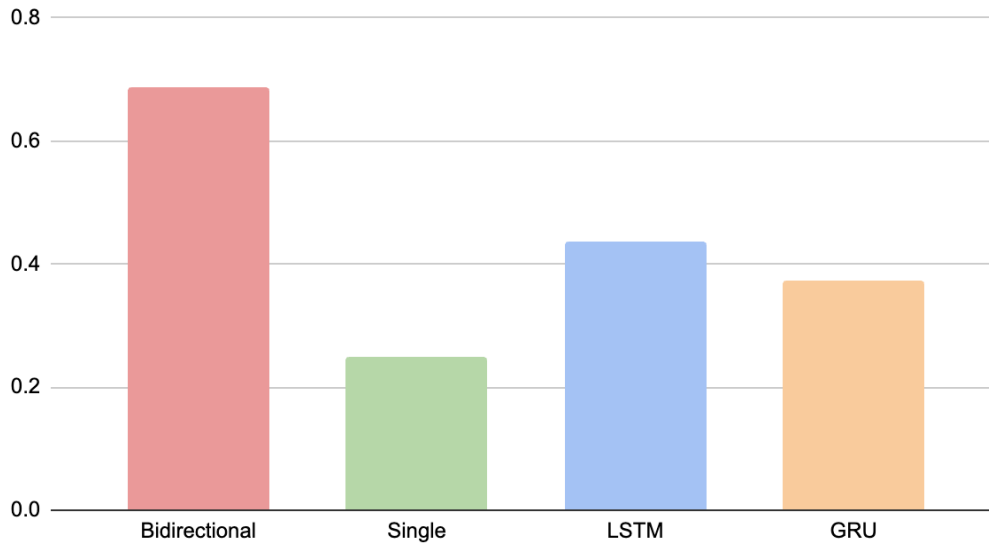


Figure 5: Bar graph showing the respective values of each model architecture according to the metric developed. This graph shows that the Bidirectional Architecture ranked the highest, with the LSTM ranked 2<sup>nd</sup>, GRU ranked 3<sup>rd</sup> and Single Directional RNN ranked 4<sup>th</sup>.

### Conclusion:

In this paper, we ran the Long Short Term Memory Model, Gated Recurrent Unit, Single Directional Recurrent Neural Network, and the Bidirectional Neural Network through rigorous testing in problems involving skills in both classification and regression. This work provides background on each of these model architectures and conducts a performance analysis. We measured the performance on these models, utilizing metrics such as Mean Squared Error, Mean Absolute Error, Training Time, F1 Score, and Logarithmic Loss. We were able to narrow down which model performed the best in all metrics. By utilizing an algorithm that we created in order to measure overall performance, taking into account all the metrics measured, we were able to find that the Bidirectional RNN had the best overall performance on both regression and classification in relation to time, the LSTM performed the 2<sup>nd</sup> best due to its capabilities to perform well on small amounts of data, the GRU performed the 3<sup>rd</sup> best mainly because of its strong performance on classification and regression problems but inefficiency in terms of time, and lastly, the Single Directional Model had the 4<sup>th</sup> best performance due to its fast execution but poor predictive capability.

### References

- [1] “Deep Learning’s Origins and Pioneers.” McKinsey & Company, 8 May 2018, [www.mckinsey.com/featured-insights/artificial-intelligence/deep-learning-origins-and-pioneers](http://www.mckinsey.com/featured-insights/artificial-intelligence/deep-learning-origins-and-pioneers).



- [2] 18th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2023): Salamanca, Spain, September 5–7, 2023, Proceedings, Volume 2. Germany, Springer Nature Switzerland, 2023.
- [3] Amidi, Afshine, and Shervine Amidi. "CS 230 - Recurrent Neural Networks Cheatsheet." Stanford.edu, 2019, stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.
- [4] Apicella, Andrea et al. "A survey on modern trainable activation functions." *Neural networks : the official journal of the International Neural Network Society* 138 (2020): 14-32 .
- [5] Bernico, Michael. *Deep Learning Quick Reference: Useful Hacks for Training and Optimizing Deep Neural Networks with TensorFlow and Keras*. United Kingdom, Packt Publishing, 2018.
- [6] Cao, Xi Hang & Stojkovic, Ivan & Obradovic, Zoran. (2016). A robust data scaling algorithm to improve classification accuracies in biomedical data. *BMC bioinformatics*. 17. 359. 10.1186/s12859-016-1236-x.
- [7] Chandra, Kartik et al. "Gradient Descent: The Ultimate Optimizer." *Neural Information Processing Systems* (2019).
- [8] Chatterjee, Samprit, and Hadi, Ali S.. *Regression Analysis by Example*. Germany, Wiley, 2015.
- [9] Cho, Kyunghyun et al. "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches." *SSST@EMNLP* (2014).
- [10] Chung, Junyoung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." *ArXiv abs/1412.3555* (2014): n. pag.
- [11] Cui, Zhiyong & Ke, Ruimin & Wang, Y.. (2018). Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction.
- [12] Dalwinder Singh, Birmohan Singh, Investigating the impact of data normalization on classification performance, *Applied Soft Computing*, Volume 97, Part B, 2020, 105524, ISSN 1568-4946, <https://doi.org/10.1016/j.asoc.2019.105524>.
- [13] Datta, Leonid. "A Survey on Activation Functions and their relation with Xavier and He Normal Initialization." *ArXiv abs/2004.06632* (2020): n. pag.
- [14] Ebert-Uphoff, Imme et al. "CIRA Guide to Custom Loss Functions for Neural Networks in Environmental Sciences - Version 1." *ArXiv abs/2106.09757* (2021): n. pag.
- [15] *Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies*. Netherlands, IOS Press, 2007.
- [16] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to Forget: Continual Prediction with LSTM," in *Neural Computation*, vol. 12, no. 10, pp. 2451-2471, 1 Oct. 2000, doi: 10.1162/089976600300015015.
- [17] Fisher, R. A.. (1988). *Iris*. UCI Machine Learning Repository. <https://doi.org/10.24432/C56C76>.
- [18] Gao, Yuan & Glowacka, Dorota. (2016). Deep Gate Recurrent Neural Network.
- [19] Hou, Le et al. "Neural Networks with Smooth Adaptive Activation Functions for Regression." *ArXiv abs/1608.06557* (2016): n. pag.

- [20] I. Bibi, A. Akhunzada, J. Malik, J. Iqbal, A. Musaddiq and S. Kim, "A Dynamic DL-Driven Architecture to Combat Sophisticated Android Malware," in *IEEE Access*, vol. 8, pp. 129600-129612, 2020, doi: 10.1109/ACCESS.2020.3009819.
- [21] J. Pomerat, A. Segev and R. Datta, "On Neural Network Activation Functions and Optimizers in Relation to Polynomial Regression," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 6183-6185, doi: 10.1109/BigData47090.2019.9005674.
- [22] Li, Xuechen et al. "Application of Gated Recurrent Unit (GRU) Neural Network for Smart Batch Production Prediction." *Energies* (2020): n. pag.
- [23] M, Gruben. "Austin Weather." Kaggle, 15 Aug. 2017, [www.kaggle.com/datasets/grubenm/austin-weather](http://www.kaggle.com/datasets/grubenm/austin-weather). Accessed 6 Apr. 2024.
- [24] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," in *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673-2681, Nov. 1997, doi: 10.1109/78.650093.
- [25] Madhavan, Samaya, and Tim M. Jones. "Deep Learning Architectures." IBM Developer, 8 Sept. 2017, [developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/](http://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/). Accessed 6 Apr. 2024.
- [26] Mehmood, Faisal, Shabir Ahmad, and Taeg Keun Whangbo. 2023. "An Efficient Optimization Technique for Training Deep Neural Networks" *Mathematics* 11, no. 6: 1360. <https://doi.org/10.3390/math11061360>
- [27] Mohammad Ehteram, Mahdie Afshari Nia, Fatemeh Panahi, Alireza Farrokhi, Read-First LSTM model: A new variant of long short term memory neural network for predicting solar radiation data, *Energy Conversion and Management*, Volume 305, 2024, 118267, ISSN 0196-8904, <https://doi.org/10.1016/j.enconman.2024.118267>.
- [28] Monner, Derek & Reggia, James. (2011). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural networks : the official journal of the International Neural Network Society*. 25. 70-83. 10.1016/j.neunet.2011.07.003.
- [29] Nwankpa, Chigozie & Ijomah, W. & Gachagan, Anthony & Marshall, Stephen. (2020). Activation Functions: Comparison of trends in Practice and Research for Deep Learning.
- [30] Olah, Christopher. "Understanding LSTM Networks." Colah's Blog, 27 Aug. 2015, [colah.github.io/posts/2015-08-Understanding-LSTMs](https://colah.github.io/posts/2015-08-Understanding-LSTMs).
- [31] Philipp, George et al. "The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions." *arXiv: Learning* (2017): n. pag.
- [32] R. Dey and F. M. Salem, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 2017, pp. 1597-1600, doi: 10.1109/MWSCAS.2017.8053243.
- [33] Ramsundar, Bharath, and Zadeh, Reza Bosagh. *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. Taiwan, O'Reilly Media, 2018.
- [34] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature* 323, 533–536 (1986). <https://doi.org/10.1038/323533a0>

- [35] Saleem, Muhammad & Potgieter, J. & Arif, Khalid. (2020). Plant Disease Classification: A Comparative Evaluation of Convolutional Neural Networks and Deep Learning Optimizers. *Plants*. 9. 1319. 10.3390/plants9101319.
- [36] Saxena, Shipra. "What Is LSTM? Introduction to Long Short-Term Memory." *Analytics Vidhya*, 4 Jan. 2024, [www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm](http://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm). Accessed 6 Apr. 2024.
- [37] Schmidt, Robin M.. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview." *ArXiv abs/1912.05911* (2019): n. pag.
- [38] Sherstinsky, Alex. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica D: Nonlinear Phenomena*. 404. 132306. 10.1016/j.physd.2019.132306.
- [39] Sinha, Samriddhi. "Sales of Shampoo over a Three Year Period." *Kaggle*, 30 July 2017, [www.kaggle.com/djokester/sales-of-shampoo-over-a-three-year-period?rvi=1](http://www.kaggle.com/djokester/sales-of-shampoo-over-a-three-year-period?rvi=1). Accessed 6 Apr. 2024.
- [40] Su, Yuanhang & Kuo, C.. (2018). On Extended Long Short-term Memory and Dependent Bidirectional Recurrent Neural Network. *Neurocomputing*. 356. 10.1016/j.neucom.2019.04.044.
- [41] Sykes, Alan O. *An Introduction to Regression Analysis*. 1993, [chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=1050&context=law\\_and\\_economics](http://chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=1050&context=law_and_economics).
- [42] Ter Braak, C. J. F., and C. W. N. Looman. "Regression." *Data analysis in community and landscape ecology*. Cambridge University Press, 1995. 29-77.
- [43] Tsamardinos, Ioannis & Brown, Laura & Aliferis, Constantin. (2006). The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm. *Machine Learning*. 65. 31-78. 10.1007/s10994-006-6889-7.
- [44] Wahid, Md Ferdous et al. "Subject-independent hand gesture recognition using normalization and machine learning algorithms." *J. Comput. Sci.* 27 (2018): 69-76.
- [45] Wang, Youming & Xiao, Zhao & Cao, Gongqing. (2022). A convolutional neural network method based on Adam optimizer with power-exponential learning rate for bearing fault diagnosis. *Journal of Vibroengineering*. 24. 10.21595/jve.2022.22271.
- [46] Wei, Daqian & Wang, Bo & Lin, Gang & Liu, Dichen & Dong, Z.Y. & Liu, Hesun & Liu, Yilu. (2017). Research on Unstructured Text Data Mining and Fault Classification Based on RNN-LSTM with Malfunction Inspection Report. *Energies*. 10. 406. 10.3390/en10030406.
- [47] Westhuizen, Jos & Lasenby, Joan. (2018). The unreasonable effectiveness of the forget gate.
- [48] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.
- [49] Yu, Ruiguo & Gao, Jie & Yu, Mei & Lu, Wenhuan & Xu, Tianyi & Zhao, Mankun & Zhang, Jie & Zhang, Ruixuan & Zhang, Zhuo. (2018). LSTM-EFG for wind power forecasting based on sequential correlation features. *Future Generation Computer Systems*. 93. 10.1016/j.future.2018.09.054.
- [50] Yuan, Wei and Kai-Xin Gao. "EAdam Optimizer: How  $\epsilon$  Impact Adam." *ArXiv abs/2011.02150* (2020): n. pag.

- [51] Zaccone, Giancarlo, and Karim, Md. Rezaul. Deep Learning with TensorFlow: Explore Neural Networks and Build Intelligent Systems with Python, 2nd Edition. India, Packt Publishing, 2018.
- [52] Zainuddin, Zulkarnain & A., P. & H., Hasan. (2021). Predicting machine failure using recurrent neural network-gated recurrent unit (RNN-GRU) through time series data. Bulletin of Electrical Engineering and Informatics. 10. 870-878. 10.11591/eei.v10i2.2036.
- [53] Zhang, Aston, et al. Dive Into Deep Learning. United Kingdom, Cambridge University Press, 2023.
- [54] Zvornicanin, Enes. "Differences Between Bidirectional and Unidirectional LSTM." Baeldung on Computer Science, 18 Mar. 2024, [www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm](http://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm).