# PREDICTING STOCK PRICES WITH NEURAL NETWORKS

**Hannah Powers**
Rensselaer Polytechnic Institute
powerh@rpi.edu

**Pierce Phillips**
Rensselaer Polytechnic Institute
phillp2@rpi.edu

## ABSTRACT

Finding models that are able to handle internal and external shocks in historical stock price trends and have accurate forecasting abilities is an extremely important task to financial firms. Using the trivial approach of Martingale Property and the Efficient Market Hypothesis (EMH) poses many risks to the models accuracy in the long-term due to the fluctuations that these models can't accurately predict. That's where the power of neural networks have come into popularity and is the approach we investigated and hypothesized that building a robust neural network architecture would show historical trends in the data can be used to predict the next price. We found that GRU-based models not only perform the best long-term but also are on par with accuracy metrics with the Martingale baseline model on the short-term. The evaluations of these models show that using historical trends allows for accurate forecasting abilities and that predicting prices with shocks can not be trivially done generally.

## 1 INTRODUCTION

### 1.1 BACKGROUND

In many studies and in the world of finance, stock prices have been shown and understood to exhibit the Martingale Property. Therefore, if we were to say that stock prices were martingale then historical data and performance would provide little insight or knowledge into the present or future price. But in order for stocks to be martingale an external given rate of return is required to be present in the market, and LeRoy (1973) shows that stock prices only exhibit being martingale when risk-aversion within the market is present, and without this presence trivially predicting stock prices is too difficult and complex. We want to investigate the hypothesis that stock prices and the market aren't martingale and that historical performance does give insight into future performance.

Another key insight is that stock prices don't follow a fixed path or pattern based on historical data, therefore questioning the reliability of past performance as an indicator of future values. With all the external factors that can influence the price of a stock researchers looked towards the efficient market hypothesis (EMH) as a model to describe the pattern of stock prices. Takeuchi et al. (2011) though rejected the hypothesis that stock prices are martingale and showed that only using discounted prices under risk-neutral market assumptions does EMH exhibit a martingale process. Because of this researchers today see a need for more advanced statistical modeling techniques to be able to not only understand the movements of prices but the potential to predict their future performance without allowing for external shocks to significantly effect the models' accuracy.

### 1.2 MOTIVATION AND APPROACH

Time series forecasting has long been a main candidate for statistical modeling within the world of financial returns and prices. Machine learning has become one of the biggest candidates for modeling these future values, especially within deep learning. Even with simpler variables, utilizing deep learning models and methods on raw data have shown to be able to construct features that are complex and useful, and shown to be especially good at predicting complex, nonlinear and noisy stock market data (Hoseinzade & Haratizadeh, 2019; LeCun et al., 2015).

In this paper we explore the use of neural networks, specifically the Recurrent Neural Network (RNN) architecture, in accurately predicting stock prices based on historical data. We develop shallow models to investigate each performance against one another and then further explore the impact that depth has on these models in individual performance and in comparison with one another. We then explore a novel approach by stacking Gated Recurrent Units (GRU) layers fed into Long Short-Term Memory (LSTM) layers and then fed back into GRU layers for stock price prediction.

## 2 Related Work

With the ever-increasing use of neural networks in the world, it is no surprise that they are popular for predicting stock prices. Shah et al. (2019) report that the use of machine learning for the field shows promise and have started to dominate the field over previously used statistical models, such as Auto-Regressive Integrated Moving Average (ARIMA). Siami-Namini et al. (2019) displays the superiority of Recurrent Neural Networks (RNN) over ARIMA, achieving a significantly lower error across all their datasets. Given the time-series nature of stock prices, RNNs are a natural choice of model compared to other methods. Sethia & Raut (2018) notes that both Long Short Term Memory (LSTM) RNNs and Gated Recurrent Unit (GRU) RNNs outperformed both an Artifical Neural Network (ANN) model and a Support Vector Machine (SVM) model. We can also see them outperform linear models in Elliot et al. (2017).

Two other neural architectures that are becoming more popular for sequential data forecasting are Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs). Zhou et al. (2020) show that using a temporal convolutional network not only reduced the time consumption for training and testing but also performed better than RNNs on sequence prediction. One issue that many researchers find is that trying to combine, or stack, multiple different architectures creates challenges to tuning the hyperparameters to achieve reliable results. But Lin et al. (2021) notes that during normal market activity periods a GAN using GRU and CNN layers performed better than LSTM and RNN architectures on stock price prediction.

## 3 Models

All models are trained with an objective function of Mean Squared Error (MSE), that is

$$\min \frac{1}{N} \|\hat{P} - P\|_2^2$$

where $\hat{P}$ are the predicted prices, $P$ are the true prices, and $N$ is the number of instances in the training data. The functions used to calculate $\hat{P}$ for each model are described as follows.

### 3.1 Martingale

The Martingale model is derived from the Martingale Property (Elliot et al., 2017). The Martingale Property states that if we have a random variable with value $x_t$ at time $t$ that follows the sequence $S = \{x_0, x_1, ...\}$ then

$$\mathbb{E}[x_t | x_0, x_1, ..., x_{t-1}] = x_{t-1}.$$

Thus, the Martingale model, when provided a sequence of input, will predict the latest value of the sequence to be the next value.

### 3.2 Recurrent Neural Network (RNN)

RNNs were developed to use the sequential nature of certain tasks. We can see in Figure 1 that the sequence of data is fed into the model and the hidden state of the model after each input is fed is the hidden state for the next input. In Figure 1 the red cell is the same cell at all points, the hidden state is fed back into itself.

An RNN updates its hidden state with the following function

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b_t) \tag{1}$$

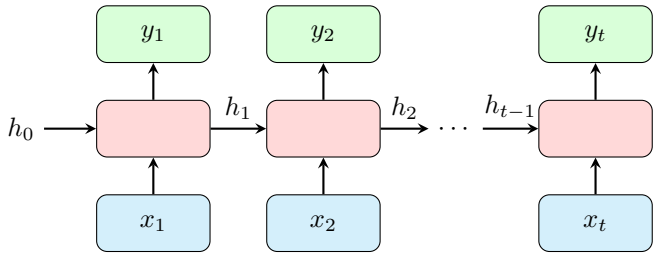


Figure 1: Diagram of unrolled RNN (Gittens, 2023)

where $\sigma$ is either the logistic sigmoid or rectified linear unit activation function, $W_x$ is the weights applied to the input $x$, $W_h$ is the weights applied to the hidden state $h$, and $b_t$ is a constant bias. For multilayer RNN, two or more RNNs are stacked with the 2nd and later RNNs taking the output of the previous RNN as input. The output of the RNN, $y_t$, is also the hidden state for the laster layer of the RNN, $h_t$.

For this project, we also apply a linear layer to the output of the RNN cell to generate a one-dimensional prediction.

### 3.3 Long Short-Term Memory (LSTM)

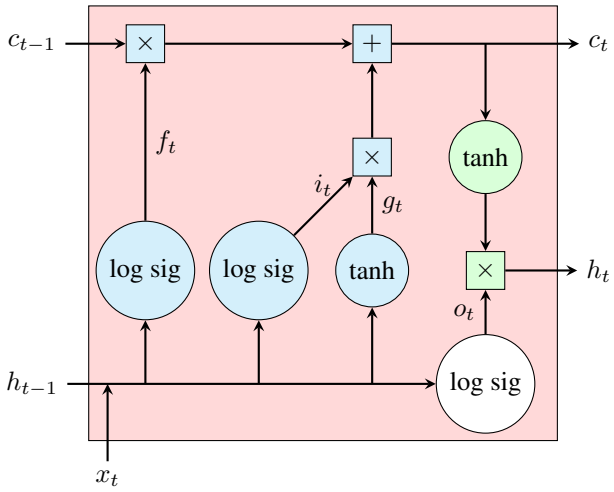


Figure 2: Diagram of LSTM cell (Gittens, 2023)

LSTM networks were introduced in order to enable an RNN to capture and preserve long-term information and dependencies that is usually lost in RNN's when dealing with sequential data. The design of an LSTM architecture resembles the same structure of the unrolled RNN in Figure 1 above but the difference lies in the structure of the RNN and LSTM cells, as seen in Figure 2 below.

An LSTM cell takes in current input $x_t$, previous hidden state $h_{t-1}$ and previous cell state $c_{t-1}$. It uses Equation 1 as in an RNN on four distinct instances: with the logistic sigmoid activation function for $i_t$, $f_t$, and $o_t$ and with the $\tanh$ function for $g_t$, as we can see in Figure 2. We use log sig to denote the logistic sigmoid function in Figure 2. The input gate is represented by $i_t$, the forget gate by $f_t$, the cell gate by $g_t$, and the output gate by $o_t$. It then takes

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

to get the new cell state and

$$h_t = o_t \odot \tanh(c_t)$$

to get the new hidden state, where $\odot$ represents element-wise multiplication. Like with multilayer RNNs, multilayer LSTMs take the hidden state previous layers as inputs, with each layer additionally being multipled by dropout $\delta_t$ which is a Bernoulli random variable.

### 3.4 GATED RECURRENT UNITS (GRU)

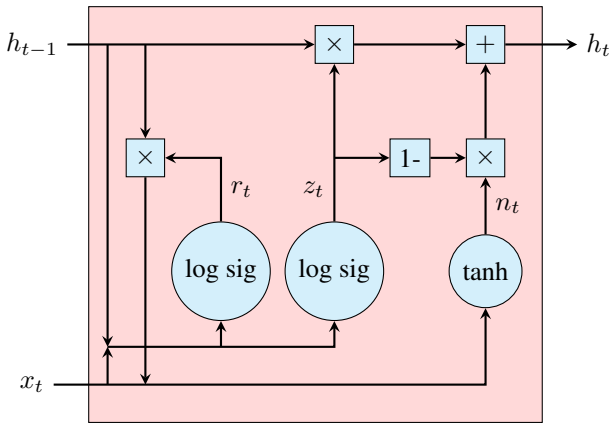


Figure 3: Diagram of GRU cell (Zhang et al., 2021)

A GRU is also a special kind of RNN whose structure is similar to LSTM with gated mechanisms to control what information is shared between cells. It differs though in having only two gates, an update and reset gate. These gates decide what information should and shouldn't be passed along to the next cell and can be trained to not only remember information from many previous cells but also dispose of information that is not useful for prediction.

The reset gate $r_t$ and update gate $z_t$ is uses Equation 1 with a logistic sigmoid activation. The new gate is calculated as

$$n_t = \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{t-1} + b_{hn})).$$

The new hidden state is then found by

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{t-1}.$$

Multilayer GRUs perform dropout like multilayer LSTMs.

Both GRUs and LSTMs solve the vanishing and exploding gradient problems. GRUs though have fewer parameters than LSTMs due to having one less gate and also lack a cell state so they have to store any long and short-term information in its hidden state (Lin et al., 2021).

### 3.5 LS-GRU


Figure 4: Diagram of ls-GRU model

Our long-short Gated Recurrent Unit (ls-GRU) model features three stacked GRUs where the final hidden state is fed into two stacked LSTMs and then again the final hidden state is fed back into three stacked GRUs. Due to the GRUs following the trends in the data well but slightly under-predicting the price and LSTMs following the trend well too while over-predicting the price we felt stacking the two approaches would yield a model that follows the data trends and being robust to handle upper and lower swings in prices due to the strengths of the GRUs and LSTMs models respectfully.

All 4 models have the same learning rate, size of each hidden state (50 neurons per layer) and linear transformation output function in order to allow for comparison within it's own model architecture but also between all other architectures.

# 4 EXPERIMENTS

## 4.1 DATA

We decided to focus on pulling data on various stock tickers to measure accuracy as well as adaptability of our models in price prediction. For training we used the closing price of Apple Inc (AAPL) only as any other variable wouldn't give any additional insight or benefit to our models performance. We scale the dataset by the MinMaxScaler function from scikit-learn which transforms each feature into a given range by

$$X_{std} = \frac{X - min(X)}{max(X) - min(X)}$$

$$X_{scaled} = X_{std} * (max - min) + min$$

where $X$ is an array of the all the closing prices and $min$ and $max$ are our lower and upper values of the range of values we're scaling our features to, which we chose to be -1 and 1 respectfully. Scaling allows our model to generalize better so it may learn on data from one stock and test on data from another, both of which may have very different values in stock price. We are training on from January $1^{st}$, 2000 to January $1^{st}$, 2017 with a bin size of 14 days where all but the last day is used as the training days and the final day in each bin are the target price we are trying to predict. This gives pairs of data where $x \in \mathbb{R}^{13}$ and $y$ is a single scalar value. We chose 14 days for the bin size due to the fact that our models will have long term memory already of the previous bins but this smaller viewing window allows for the short term to have a bigger effect on the next predicted price than with a bigger bin size. After training we use the closing prices from January $1^{st}$, 2017 to January $1^{st}$, 2020 as the validation dataset to see how well our model does on the remaining prices for AAPL in the full dataset. We then used Wells Fargo (WFC), Tesla (TSLA), Genera Mills (GIS) and American Tower Corp (AMT) to see how well the models do on new data from different market industries on the entire timeframe from 2000 to 2020.

## 4.2 EVALUATION

We use a number of metrics to evaluate the performance of these models. We use both the $R$-squared and root mean squared error measures from scikit-learn defined as follows:

$$RMSE = \sqrt{\frac{1}{|D|} \sum_{(p,\hat{p}) \in D} (p - \hat{p})^2}$$

and

$$R^2 = 1 - \frac{\sum_{(p,\hat{p}) \in D} (p - \hat{p})^2}{\sum_{(p,\hat{p}) \in D} (p - \overline{p})^2}$$

where $p$ is the true price, $\hat{p}$ is the predicted price, $\overline{p}$ is the mean price of the observed data, $D$ is the dataset, and $|D|$ indicates the size of the dataset. We also use the optimism and pessimism ratios defined in Sethia & Raut (2018).[1] We have

$$OR = \frac{\sum_{(p,\hat{p}) \in D} I(\hat{p} > 1.015p)}{|D|} \quad \text{(Opt Ratio)}$$

$$PR = \frac{\sum_{(p,\hat{p}) \in D} I(\hat{p} < 0.985p)}{|D|} \quad \text{(Pes Ratio)}$$

where $I(\cdot)$ is the indicator function that returns 1 when true and 0 otherwise. Ideally, a good model will have small values for $OR$ and $PR$. Should a model have a high optimism ratio, it would mean it tends to predict prices over what they should be. If that model were then used for choosing stocks to buy then it may overestimate the profit to be made, if any at all. Similarly, a model with a high pessimism ratio may mean a stock is shorted when it could have been kept or never bought at all. A low value for both ratios means the model is reliable and not likely to lead towards unnecessary risks or losses.

| | | RMSE | R-Squared | OR | PR |
|---|---|---|---|---|---|
| | **Martingale** | **0.015912** | **0.997501** | 0.525896 | 0.600266 |
| **1-Layer** | **RNN** | 0.255759 | 0.35445 | 0.102922 | 0.934263 |
| | **GRU** | 0.044402 | 0.980543 | 0.233732 | 0.87251 |
| | **LSTM** | 0.255294 | 0.356797 | **0.035193** | 0.995352 |
| | **ls-GRU** | **0.034726** | **0.988099** | 0.923639 | **0.126826** |
| **2-Layer** | **RNN** | 0.410256 | -0.661027 | **0.051129** | 0.995352 |
| | **GRU** | **0.030074** | **0.991074** | 0.710491 | **0.363214** |
| | **LSTM** | 0.235926 | 0.450689 | 0.094954 | 0.962151 |
| | **ls-GRU** | **0.032112** | **0.989823** | 0.870518 | **0.204515** |
| **3-Layer** | **RNN** | 0.47496 | -1.226288 | **0.02988** | 0.996016 |
| | **GRU** | 0.088512 | 0.922684 | 0.425631 | 0.661355 |
| | **LSTM** | 0.26359 | 0.314317 | **0.066401** | 0.971448 |
| | **ls-GRU** | 0.073291 | 0.946989 | 0.438247 | 0.726428 |

Table 1: Model results on the validation dataset for the metrics defined in Section 4.2

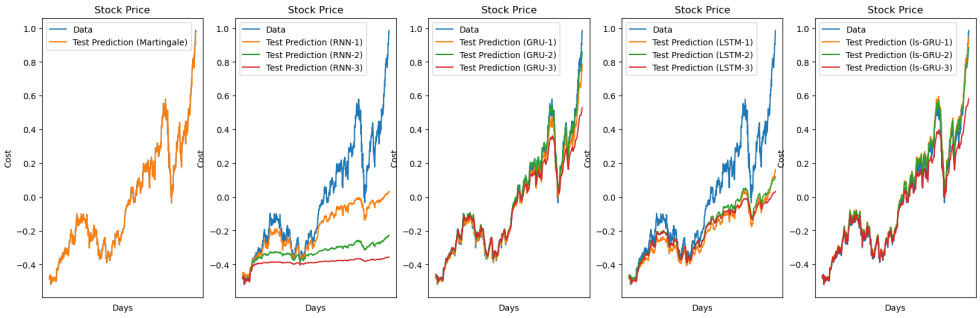


Figure 5: The actual vs. predicted stock price for all nine model types with varied layers for AAPL

### 4.3 Results

From our experimental results, down below in Table 5, we found that the GRU and our ls-GRU models performed the strongest compared to the other two models and did well to follow the trends of the Martingale process seen in the first model above in Figure 5 that is used for our baseline comparison. Once we began testing our models on the testing stock tickers we found similar results as the validation results but the ls-GRU model was now able to achieve the highest average $R^2$ over the GRU models by about 0.5%-1% throughout the additional analyses. The testing data results are listed in Appendix A. Varying the depth of the models also yielded a positive relationship for most of the different testing datasets for GRU and ls-GRU in terms of depth and accuracy while a negative relationship is more prominent for RNN and LSTM. Both RNN and LSTM over-predicted the closing price in every dataset when looking at the entirety of the dataset time window. No model was able to achieve both a low OR and PR but the Martingale, 3-layer GRU, and 3-layer ls-GRU almost balance the ratios where prices weren't skewed toward over or under-predicting, but the GRUs were the best model at balancing these ratios overall. The highest performing model on the validation set was the 2-layer GRU with a final $R^2$ of 91.1% and both the GRU and ls-GRU models continued to achieve $R^2$ above 95% for all the depth layers tested. These results show promise that a GRU-based model is able to effectively track the trends and shocks to historical stock price data and accurately predict the next closing price.

## 5 Conclusion

In this paper we have shown the difference in abilities for differing RNN architectures to capture and remember trends in time-series data and their predictive power. The baseline Martingale model tended to perform the best in the short-term with relatively minor shocks present in the data. But

[1]Code for our research project can be found at https://github.com/hpowers57/MLOpt_ResearchProject.

GRU-based models performed the strongest across all depths and testing datasets in the long-term with more shocks and volatility. We believe that the storage of the hidden and cell state that would normally be in LSTM being stored in the hidden state of the GRU allows for the model to remember key long-term trends present in the short-term while still allowing for the more recent trends to have a strong influence. Stacking GRU and LSTM models together also seemed to help balance out the over-prediction of the GRU only models and the under-prediction of the LSTM only models into our ls-GRU model that, on the new testing data, performed the strongest. These models could be very useful in applications that require complex long and short-term memory retention from time-series data that is influenced by minor and major external and internal shocks that can't be trivially predicted.

## REFERENCES

Aaron Elliot, Cheng Hsu, and Jennifer Slodoba. Time series prediction: Predicting stock price. *arXiv*, 2017.

Prof. Alex Gittens. Lecture notes for machine learning and optimization, April 2023.

Ehsan Hoseinzade and Saman Haratizadeh. Cnnpred: Cnn-based stock market prediction using a diverse set of variables. *Expert Systems with Applications*, 129:273–285, 2019.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.

Stephen F. LeRoy. Risk aversion and the martingale property of stock prices. *International Economic Review*, 14:436–446, 1973.

HungChun Lin, Chen Chen, GaoFeng Huang, and Amir Jafari. Stock price prediction using generative adversarial networks. *Journal of Computer Science*, 17:188–196, 2021.

Akhil Sethia and Purva Raut. Application of lstm, gru and ica for stock price prediction. *International Conference on ICT for Intelligent Systems*, 2:479–487, 2018.

Dev Shah, Haruna Isah, and Farhana Zulkernine. Stock market analysis: A review and taxonomy of prediction techniques. *International Journal of Financial Studies*, 7:26, 05 2019.

Sima Siami-Namini, Neda Tavakoli, and Akbar S. Namin. The performance of lstm and bilstm in forecasting time series. *IEEE International Conference on Big Data*, pp. 3285–3292, 2019.

Kei Takeuchi, Akimichi Takemura, and Masayuki Kumon. New procedures for testing whether stock price processes are martingales. *Computational Economics*, 37:67–88, 2011.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*, chapter 10. arXiv, 2021.

Kun Zhou, Wenyoung Wang, Teng Hu, and Kai Deng. Time series forecasting and classification models based on recurrent with attention mechanism and generative adversarial networks. *Sensors*, 20:7211, 2020.

## A APPENDIX

| | | RMSE | R-Squared | OR | PR |
|---|---|---|---|---|---|
| | **Martingale** | 0.022662 | 0.996962 | 0.530397 | 0.538968 |
| **1-Layer** | **RNN** | 0.308436 | 0.43729 | 0.233805 | 0.828981 |
| | **GRU** | 0.05426 | 0.982585 | 0.28543 | 0.831772 |
| | **LSTM** | 0.290203 | 0.501851 | 0.116604 | 0.939406 |
| | **ls-GRU** | 0.036576 | 0.992087 | 0.858282 | 0.204106 |
| **2-Layer** | **RNN** | 0.458919 | -0.24574 | 0.146701 | 0.920072 |
| | **GRU** | 0.034539 | 0.992944 | 0.625274 | 0.420171 |
| | **LSTM** | 0.278112 | 0.542494 | 0.195934 | 0.874626 |
| | **ls-GRU** | 0.034525 | 0.992949 | 0.778952 | 0.284632 |
| **3-Layer** | **RNN** | 0.524168 | -0.625161 | 0.105043 | 0.950568 |
| | **GRU** | 0.109712 | 0.928803 | 0.45007 | 0.642615 |
| | **LSTM** | 0.311712 | 0.42527 | 0.148894 | 0.901136 |
| | **ls-GRU** | 0.088282 | 0.9539 | 0.449073 | 0.709587 |

Table 2: Model results on the WFC dataset for the metrics defined in Section 4.2

| | | RMSE | R-Squared | OR | PR |
|---|---|---|---|---|---|
| | **Martingale** | 0.030263 | 0.996967 | 0.621849 | 0.643277 |
| **1-Layer** | **RNN** | 0.269407 | 0.759626 | 0.203782 | 0.977311 |
| | **GRU** | 0.054342 | 0.99022 | 0.448739 | 0.837395 |
| | **LSTM** | 0.265075 | 0.767295 | 0.304622 | 0.984034 |
| | **ls-GRU** | 0.046063 | 0.992973 | 0.839076 | 0.437395 |
| **2-Layer** | **RNN** | 0.426173 | 0.398492 | 0.297479 | 0.987815 |
| | **GRU** | 0.047928 | 0.992392 | 0.683613 | 0.583193 |
| | **LSTM** | 0.247635 | 0.796907 | 0.285294 | 0.978571 |
| | **ls-GRU** | 0.047859 | 0.992414 | 0.783613 | 0.490336 |
| **3-Layer** | **RNN** | 0.490949 | 0.201745 | 0.14958 | 0.993277 |
| | **GRU** | 0.100492 | 0.966555 | 0.480672 | 0.785714 |
| | **LSTM** | 0.277458 | 0.745044 | 0.276891 | 0.968908 |
| | **ls-GRU** | 0.082154 | 0.977648 | 0.47479 | 0.829832 |

Table 3: Model results on the TSLA dataset for the metrics defined in Section 4.2

| | | RMSE | R-Squared | OR | PR |
|---|---|---|---|---|---|
| | **Martingale** | 0.015426 | 0.998965 | 0.643014 | 0.682081 |
| **1-Layer** | **RNN** | 0.249237 | 0.729766 | 0.420371 | 0.834363 |
| | **GRU** | 0.041955 | 0.992343 | 0.4118 | 0.890572 |
| | **LSTM** | 0.24087 | 0.747604 | 0.406618 | 0.786924 |
| | **ls-GRU** | 0.030725 | 0.995893 | 0.854295 | 0.45286 |
| **2-Layer** | **RNN** | 0.385805 | 0.352483 | 0.326091 | 0.955152 |
| | **GRU** | 0.026548 | 0.996934 | 0.662747 | 0.563484 |
| | **LSTM** | 0.226902 | 0.776029 | 0.37931 | 0.844728 |
| | **ls-GRU** | 0.028807 | 0.99639 | 0.792705 | 0.498505 |
| **3-Layer** | **RNN** | 0.442609 | 0.147773 | 0.205302 | 0.971098 |
| | **GRU** | 0.084528 | 0.968918 | 0.45007 | 0.774965 |
| | **LSTM** | 0.253937 | 0.719479 | 0.295196 | 0.887383 |
| | **ls-GRU** | 0.069414 | 0.979039 | 0.533187 | 0.836755 |

Table 4: Model results on the GIS dataset for the metrics defined in Section 4.2

| | | RMSE | R-Squared | OR | PR |
|---|---|---|---|---|---|
| | **Martingale** | 0.009233 | 0.999528 | 0.795894 | 0.820809 |
| **1-Layer** | **RNN** | 0.174603 | 0.831359 | 0.590193 | 0.837752 |
| | **GRU** | 0.031292 | 0.994583 | 0.600558 | 0.934822 |
| | **LSTM** | 0.167032 | 0.845666 | 0.599163 | 0.774566 |
| | **ls-GRU** | 0.019953 | 0.997798 | 0.907913 | 0.625673 |
| **2-Layer** | **RNN** | 0.262885 | 0.617709 | 0.537373 | 0.95894 |
| | **GRU** | 0.018811 | 0.998043 | 0.798286 | 0.702212 |
| | **LSTM** | 0.15945 | 0.85936 | 0.609129 | 0.860474 |
| | **ls-GRU** | 0.018624 | 0.998081 | 0.860474 | 0.672314 |
| **3-Layer** | **RNN** | 0.301969 | 0.495589 | 0.272872 | 0.984453 |
| | **GRU** | 0.065655 | 0.976155 | 0.667132 | 0.789914 |
| | **LSTM** | 0.177366 | 0.82598 | 0.497508 | 0.92346 |
| | **ls-GRU** | 0.054203 | 0.983748 | 0.754634 | 0.856089 |

Table 5: Model results on the AMT dataset for the metrics defined in Section 4.2