Predicting the Future Price of Financial Assets

To What Extent is it Possible to Predict Financial Time Series Forecasting with Gated Recurrent Units and Bayesian Optimization?

SUBJECT: Math

WORD COUNT: 3212

Table of Contents

Introduction

"We're right 50.75 percent of the time . . . but we're 100 percent right 50.75 percent of the time, … You can make billions that way." (Zuckerman). Said Robert Mercer, the former co-CEO of Renaissance Technologies, a quantitative hedge fund that has 165 billion dollars under management (Whale Wisdom). He's addressing their use of machine learning…

Machine learning is a popular topic in business and finance today and is becoming more applied as each day goes by. Financial companies like JP Morgan, Goldman Sachs and Two Sigma use machine learning for applications like Portfolio Management, Algorithmic Trading, Fraud Detection, Risk Management and many others (Cheung).

Unlike linear models, artificial neural networks and other machine learning models do not have a specific function that they follow. Because of this, not all the results of a model can be fully understood. As the function that a deep learning model gives out is much more complex than a simple $y = mx + c$ equation, where no-one would not be able to comprehend the equation that a deep learning model comes up with but the equations that do come out are fully usable, which is why it is also called a black-box function.

This is why I wanted to use neural networks to explore my question "To What Extent is It Possible to Predict Financial Time Series Forecasting with Gated Recurrent Units and Bayesian Optimization?". I did get results that prove that my approach is a possibility, and while we can explain what steps the model goes through to create said output, we do not know why exactly it would choose specific values for variables such as weights.

My goal for this project was to create a machine learning program that trains a model on past financial data using a neural network to predict the price of a financial instrument such as a stock or exchange-traded fund (ETF) one day ahead. I gave the neural network two different datasets. The first dataset $Y_1 = \{y_1, y_2, \ldots, y_n\}$ is for the neural network to train and create a model which will be a function $f(x)$. I then used the second dataset

$Y_2 = \{y_1, y_2, \ldots, y_n\}$ to test the accuracy of the function that was created by the neural network.

The creation of the model can be compared to trying to set a line of best fit to multiple data points, which is what actually happens in linear regression.

For this project, I use Python 3 due to its many application programming interface (API) packages such as Tensorflow, an API package for machine learning developed by Google.

Data

The data that I used for training and testing is the data of the three largest financial indices (Standard and Poor's 500, Nasdaq-100, and Russell 2000) that exist. This is because these three indices have a relatively high amount of data ranging back to the 1990s compared to other financial products such as different indices or company stocks by themselves. This is very important as in machine learning the more data there is to be analysed during the learning process of the model the better results will come out in the end.

Table 1 shows an example of the dataset being used. This particular one is for the Nasdaq 100 composite showing the Open, High, Low, Close and Volume with their corresponding dates only showing the first and last three rows instead of all 5696 rows.

Table 1 - Nasdaq 100 dataset

Date	**Open**	**High**	**Low**	**Close**	**Volume**
1999-03-10	44.340526	44.367629	43.608744	44.28632	5232000
1999-03-11	44.611555	44.869034	43.635847	44.503143	9688600
1999-03-12	44.340528	44.367631	43.066686	43.419025	8743600
...	...	...	...	...	...
2021-10-21	374.160004	377.470001	373.850006	377.269989	25469300
2021-10-22	375.980011	376.970001	372.390015	374.100006	36231900
2021-10-25	375.559998	376.23999	373.559998	375.390015	9026866

All data for this project was sourced from Yahoo Finance as they provide a Python 3 API that allows me to download daily stock price information for free.

After all the required data is gathered, it is then divided and scaled. The division of data is required to first train the model with the neural network, and then test the aforementioned trained model on data that it has not yet seen. It is very important that for training, the model has no access to the test data, as otherwise there would be a data leakage problem meaning test results would be worthless as when the neural network is training the model, it can apply the function $f(x)$ to the newest data that is meant for testing meaning all testing results would become useless. It would be like giving a student the answers to a test that they will be taking the next day.

The division of data itself follows the 70/30 rule where the first 70% of all the data is used for training and the other 30% are used for testing.

This data also needs to be scaled between 0 and 1 so that the model accepts all data as the same so that higher values are not interpreted as more important than lower ones. An example of this can be seen in Table 1, where volume data is in its millions but close data which we want to predict is in its tens or hundreds. If the data is not scaled the model would presume volume data to be more important than it actually is.

For scaling I will be using the Min-Max scaling or normalization approach which has the following formula:

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

("ML | Feature Scaling – Part 2.")

Where:

- X_i is the current value being scaled
- $min(X)$ is the lowest value in the whole data array (i.e. lowest close price)

- $max(X)$is the highest value in the whole data array (i.e. highest close price)
- X_{new} is the transformed data value ranging from 0 to 1

After the data is scaled it goes from the normal values as seen in Table 1, to values between 0 and 1 as seen in Table 2.

Table 2 - Scaled Nasdaq 100 dataset

Date	Open	High	Low	Close	Volume
1999-03-10	0.079304	0.079681	0.077666	0.080561	0.00758
1999-03-11	0.080295	0.080141	0.080874	0.080561	0.002681
1999-03-12	0.078312	0.080874	0.079092	0.081343	0.012441
...	...	...	...	...	...
2021-10-21	0.997701	0.996294	0.990848	0.991396	0.076921
2021-10-22	0.980222	0.985307	0.974044	0.976154	0.084169
2021-10-25	0.980483	0.978496	0.981171	0.978987	0.047176

Model

To train the model that will be used to predict the price of the ETF one day ahead I will be using Gated Recurrent Units (GRU) which is a form of a recurrent neural network (RNN). An RNN in a neural network that takes an input such as stock indice data.

To make a single model, multiple GRU cells are used in a chain-like fashion where the output of one GRU cell $h_t$becomes $h_{t-1}$for the next GRU cell.

A GRU has two important gates:

The first is the update gate which decides what information learned from the data is required then kept, and what information is no longer necessary.

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j$$

(Junyoung et al.)

Where:

- z_t is the update gate
- σ is the sigmoid function
- W_z and U_z are weights that the GRU decides for the model based on unknown variables.
- x_t is the input vector which in this case is the scaled data
- h_{t-1} is the output vector from the GRU cell before

The reset gate r_t^j with a sigmoid activation is used to decide what information that has been learned in the past is useful information, and what can be discarded. This is done the following way.

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})$$

(Junyoung et al.)

Where:

- r_t is the reset gate
- Other variables are the same as above.

After this, the current memory has to be calculated. This is the information taken from the reset gate, and from GRU cells that have come before it.

$$h_t' = PReLU(Wx_t + r_t \odot Uh_{t-1})$$

(Junyoung et al.)

Here the input x_t is multiplied with the weight W, and h_{t-1} is multiplied by the weight U. These two products are then used to calculate the Hadamard product ($\odot$) which creates an output matrix of the same size as Wx_t and Uh_{t-1}. In layman's terms, this means that the model decides what to keep and what information to get rid of.

After this, the activation function is applied. In this case I have chosen it to be a Parametric Rectified Linear Unit (PReLU) as from tests of multiple Rectified Linear Unit (ReLU) activations it has shown to have the best MAPE scores[1] of less than 3%. Activation functions let the output of the neural network be non-linear. The PReLU activation function is as follows:

$$f(y_i) = max(0, y_i) + a_i min(0, y_i)$$

(He, Kaiming, et al.)

Where:

- y_i is
- a_i is the "coefficient controlling the slope of the negative part"

Finally, the vector h_t which holds all the information gathered in the GRU needs to be calculated as it takes into account the information gathered in GRU cells before, and information gathered by the update gate. This is the final step and is known as the hidden state.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

(Junyoung et al.)

Where:

- All variables are the same as mentioned before.

[1] See Results and Analysis

Here the Hadamard product ($\odot$)is calculated between the update gate z_t and $h_{t-1}$and summed with the Hadamard product ($\odot$) between $(1 - z_t)$ and $h_t^{'}$whose value was calculated in the step before. This then gives us $h_t$which is the output vector. If this is not the last step h_t becomes $h_{t-1}$and the whole process is repeated over as many GRU units as are either chosen by the user, or by Bayesian Optimization in this case.

Bayesian Optimization

To further improve the results of the model being created I have chosen to apply Bayesian Optimization. The benefit of this is that there are millions of different values for hyperparameters such as dropout, learning rate and even the number of GRU cells that can be used. Furthermore, each financial index being predicted may benefit from different hyperparameters meaning that even if it is possible to find the best values for one ETF, others may not benefit from such values.

My solution to the choice of millions of different hyperparameters is to use a hyperparameter tuner - Bayesian Optimization - so that the algorithm itself finds the perfect hyperparameters for each of the different financial indices. This is where Bayesian Optimization comes in. The idea first came from Boris Banushev and his article titled "stockpredictionai.", though there the author chooses to use it only for some hyperparameters such as learning rate and dropout, the plan for the model being created is to leave all values up to the algorithm so that it finds the best values for each hyperparameter. The downside of this method is that it not only takes a lot more time but also requires a lot more computational power.

In theory Bayesian optimization is quite simple. Given our dataset $Y = \{y_1, y_2, ..., y_n\}$ random points are chosen which then create a surrogate function which is an approximation of our whole dataset.

This is done over multiple iterations where a black box regressor is trained over given points that increase with each iteration (OptimizationGeeks).

Preventing Overfitting

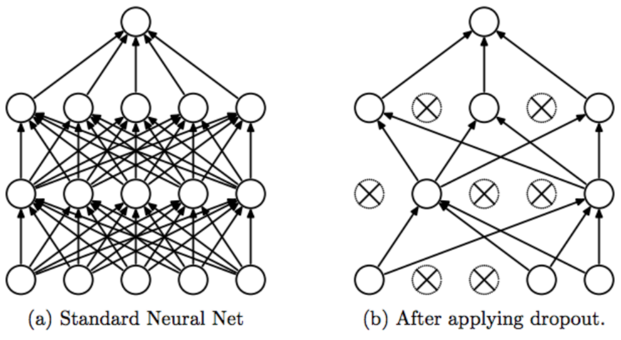
Overfitting is a term used when a model is fit exactly to its training data. “When this happens, the algorithm, unfortunately, cannot perform accurately against unseen data, defeating its purpose. Generalization of a model to new data is ultimately what allows us to use machine learning algorithms every day to make predictions and classify data” (IBM Cloud Education).

Compared to the financial data that is available, ETFs are the one with the most available data for finance, but compared to data being used in machine learning for other applications it is insufficient which leads to a high risk of overfitting.

Here are the techniques being used to prevent such a problem:

- Separating dating into training and testing datasets which were mentioned before. As, if “the training data has a low error rate and the test data has a high error rate, it signals overfitting”(IBM Cloud Education).
- One of the main strategies that I use is called a dropout which selects random units with their connections from the GRU and knocks them out.

Figure 01 - Dropout Example



(Srivastava, et al.)

- Model checkpoints are simple save states that the algorithm makes when an epoch has an improved result which in this case and in most others is the lowest loss possible which most likely means that it has reached its lowest point of gradient descent and if it keeps training there could be diminishing returns to the result, which is prevented by saving the best result.
- Early stopping is a technique where when a model has not had an improved loss for in my case 10 epochs, it stops training the model as there most likely is nothing else the model can extrapolate from the data, and the gradient descent will have finished. It has also shown me that even if I have my algorithm set for 100 epochs, all three of the indices while training stopped between 25 and 40 epochs, showing how fast my model was able to learn to reach very positive results.

Gradient Descent and Learning Rate

Gradient descent is the process through which the machine learning algorithm minimizes the function that is being trained, making the model more precise. It works by adjusting its "parameters iteratively to minimize a given function to its local minimum" (Donges). For its purpose I will be using the Adam loss function.

The learning rate of a machine learning model are the iterative steps sizes which gradient descent takes during optimization. It is the step size the algorithm takes to the best minimum of the Adam loss function. Usually, a learning rate is set to a specific number such as 0.001 and it does not change. A different approach was proposed not long ago where with each step the learning rate either becomes bigger or smaller called a cyclical learning rate (Smith). The positive of this is that at the start of the learning process the model is able to take larger steps towards the optimal loss function so that convergence is a lot faster, but at the end, it can take smaller steps to perfectly align with the gradient minimum (DeepLearningAI).

Results and Analysis

My results were really close to actual values, as seen by Figures 1 through 3, and testing scores.

Figure 02 - S&P 500 (SPY) Real price and prediction

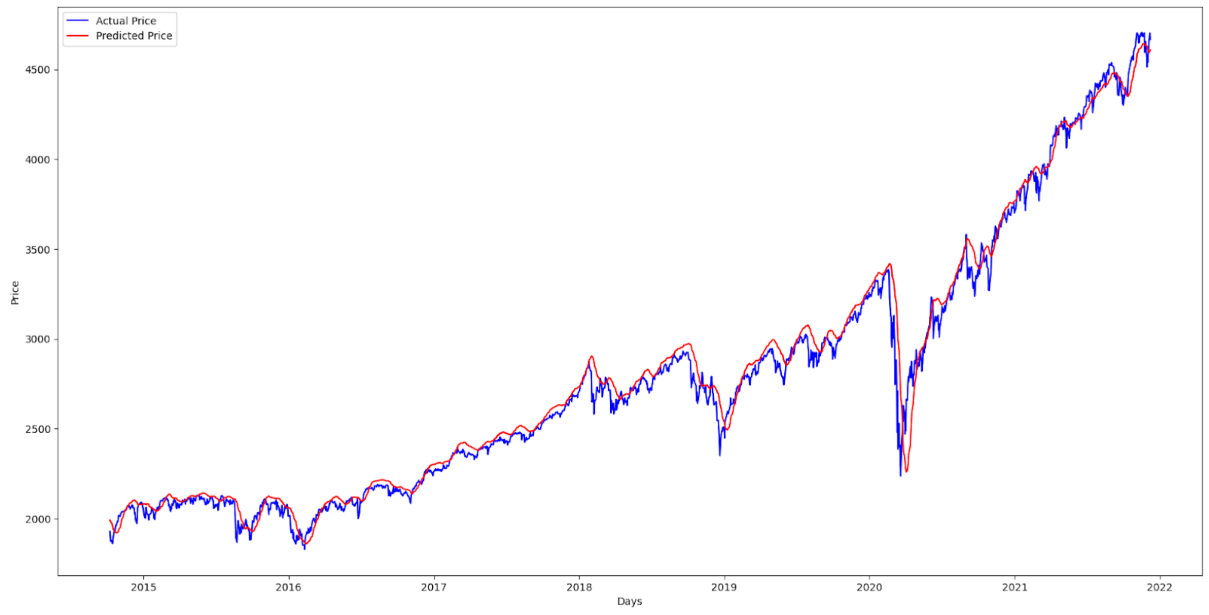


Figure 03 - Nasdaq 100 (QQQ) Real price and prediction

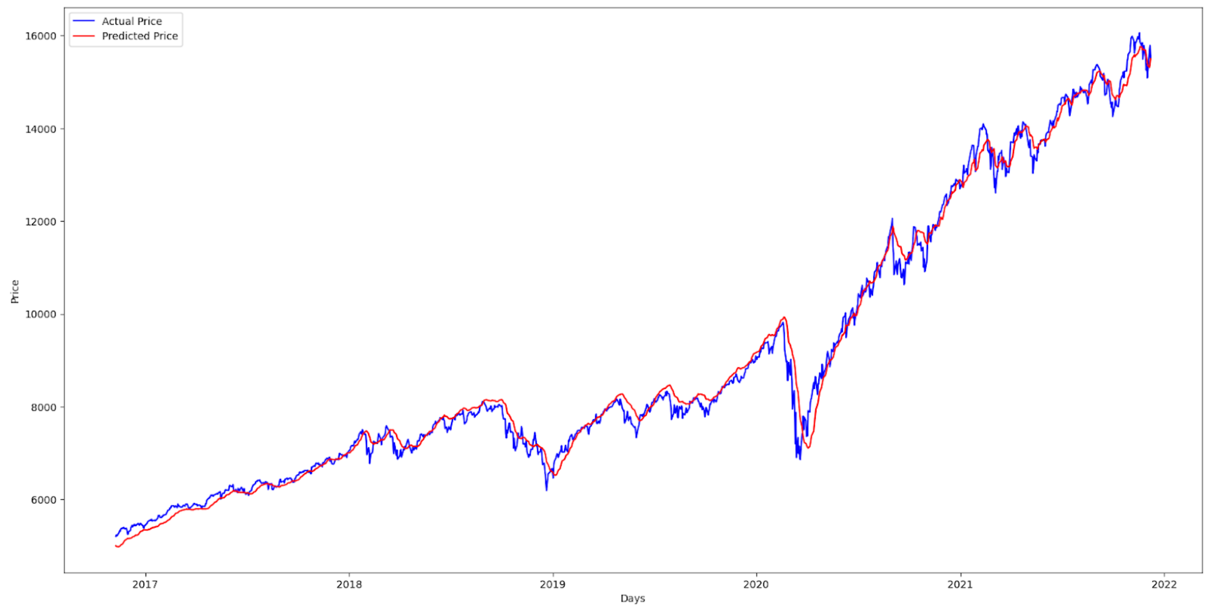
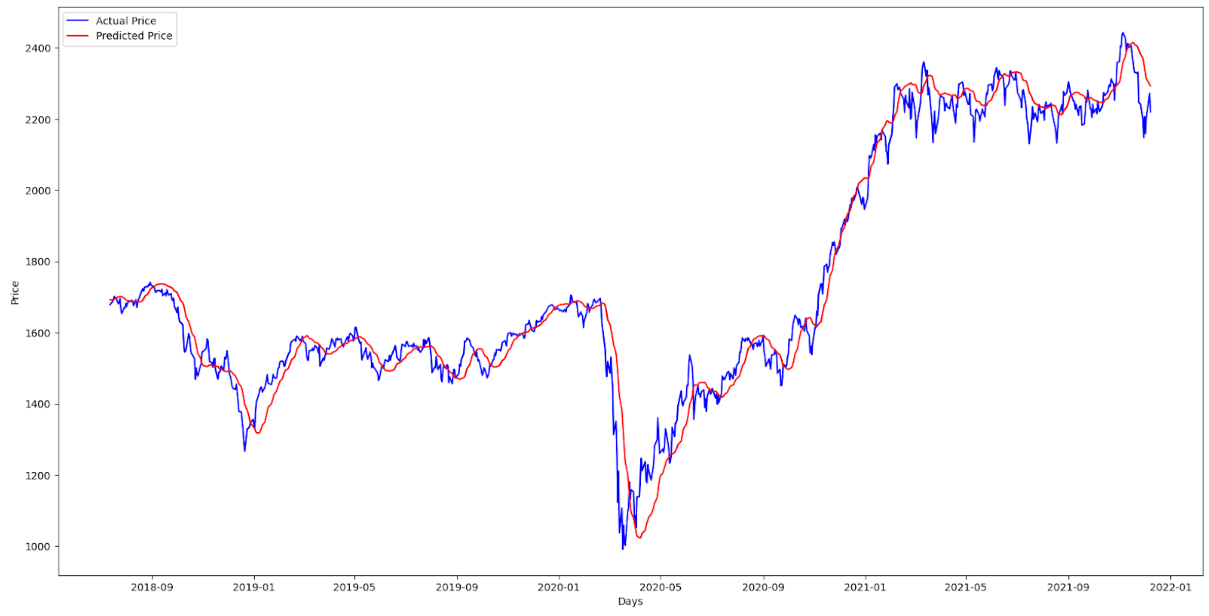


Figure 04 - Russell 2000 (IWM) Real price and prediction



I have chosen to use two types of testing measures to evaluate the predicted data of the future compared to real one day ahead data.

MAPE or Mean Absolute Percentage Error as it provides the complete accuracy for the whole prediction and will account if the model decides that the best way of prediction is to move the past prices up by one. The lower the MAPE value is the better the result as it shows that the error is smaller. The following is the formula:

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

(Stephanie. "Mean Absolute Percentage Error (MAPE)." Statistics How To)

Where:

- $N$is the number of points
- $y_i$is the real value
- $\hat{y}_i$is the predicted value

While the MAPE score is a very good indicator of how successful my model is, I wanted another to make sure actually how accurate it was as the MAPE could be fooled by a day where the change in price is not substantial and does not make a big impact on MAPE even if the day was positive but the algorithm predicted a negative so I decided to come up with an accuracy score.

The accuracy score is a metric of when the model says that the price will either go up or down compared to if it actually went up. I calculate it by having:

$$\%\Delta_{real}\ compared\ to\ \%\Delta_{predicted}$$

If both are negative or positive it counts as a positive result, on the other hand, if the model has a positive $\%\Delta$and the actual is a negative $\%\Delta$ it counts as a negative result and vice versa.

These are the values that I then received for the indices that I had selected:

Indice	Best MAPE	Best Accuracy
S&P 500	1.93	62.60%
QQQ	2.06	67.54%
IWM	2.98	61.80%

It is interesting to see that the S&P 500 index with the lowest MAPE score does not actually have the lowest accuracy score that the QQQ index has, and vice versa. Currently, I am not sure why the results for the IWM index are so much worse compared to S&P 500 and QQQ. In my opinion, this is because the Russell 2000 index has the least data points at 8634 rows only meaning it has less data to learn from. The other reason that I can think of is the larger abundance of stocks in the IWM index, as Nasdaq and S&P 500 have 100 and 500 stocks respectively included in it but the IWM has 2000 which could create a lot more noise in the data, leading to the model being less precise..

Conclusion

Through the essay, the steps taken to predict the future price of a financial instrument are laid out. The result that was received was very encouraging with all accuracies being above 60% showing that the success rate is better than an average which would be at 50%.

By the completion of the project, a research paper was released about a new form of machine learning called Bayesian Neural Networks which let the user quantify the uncertainty of the output of a neural network (Jospin, et al.). This would not only provide more data that could be analysed to understand the success of my model, but it would also let any user understand more of what is going on in the model, opening up the black box function. As currently we have no understanding whatsoever about what is exactly happening while the model is being trained.

While the results are exceedingly positive, there is a very large reliance on the black box function that no person can understand to make these predictions, and while a single price prediction algorithm has no real consequences to a human's life, the fact that the biggest financial institutions on earth that control the financial futures of millions of people rely on a technique that no one can fully understand may be frightening.

Works Cited

Banushev, Boris. "stockpredictionai." *GitHub*, 9 Jan. 2019, github.com/borisbanushev/stockpredictionai. Accessed 3 Apr. 2021.

"Bayesian Optimization (Bayes Opt): Easy explanation of popular hyperparameter tuning method." *YouTube*, uploaded by Optimization Geeks, 25 Jan. 2021, www.youtube.com/watch?v=M-NTkxfd7-8. Accessed 25 Aug. 2021.

Cheung, KC. "10 Applications of Machine Learning in Finance." *Algorithm-XLab*, 20 Aug. 2020, algorithmxlab.com/blog/applications-machine-learning-finance/. Accessed 13 Nov. 2021.

Cho, Kyunghyun, et al. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. 3 Sept. 2014, arxiv.org/pdf/1406.1078.pdf. Accessed 10 Aug. 2021.

Donges, Niklas. "Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms." *Builtin*, 23 July 2021, builtin.com/data-science/gradient-descent. Accessed 16 Dec. 2021.

Doshi, Akash, et al. *Deep Stock Predictions*. 8 June 2020. *arXiv*, arxiv.org/pdf/2006.04992.pdf. Accessed 3 July 2021.

He, Kaiming, et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 6 Feb. 2015, arxiv.org/pdf/1502.01852.pdf. Accessed 15 Dec. 2021.

Hochreiter, Sepp, and Jurgen Schmidhuber. *Long Short-Term Memory*. 1997, www.bioinf.jku.at/publications/older/2604.pdf. Accessed 15 June 2021.

IBM Cloud Education. "What is overfitting?" *IBM*, 3 Mar. 2021, www.ibm.com/cloud/learn/overfitting. Accessed 16 Dec. 2021.

"Is it a good practice to always scale/normalize data for machine learning?" *Stats Stack Exchange*, 7 Jan. 2016, stats.stackexchange.com/questions/189652/is-it-a-good-practice-to-always-scale-normalize-data-for-machine-learning. Accessed 2 Sept. 2021.

Jospin, Laurent Valentin, et al. *Hands-on Bayesian Neural Networks – a Tutorial for Deep Learning Users*. 30 Sept. 2021, arxiv.org/pdf/2007.06823.pdf#:~:text=WHAT%20IS%20A%20BAYESIAN%20NEURAL,(ANNs)%20is%20to%20repre%2D. Accessed 16 Dec. 2021.

Junyoung, Chung, et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 11 Dec. 2014. *arXiv*, arxiv.org/pdf/1412.3555.pdf. Accessed 25 Aug. 2021.

"Learning Rate Decay (C2W2L09)." *YouTube*, uploaded by DeepLearningAI, 25 Aug. 2017, youtu.be/QzulmoOg2JE. Accessed 16 Dec. 2021.

"ML | Feature Scaling – Part 2." *GeeksforGeeks*, 5 July 2021, www.geeksforgeeks.org/ml-feature-scaling-part-2/. Accessed 12 Sept. 2021.

Smith, Leslie N. *Cyclical Learning Rates for Training Neural Networks*. 4 Apr. 2017. *arXiv*, arxiv.org/pdf/1506.01186.pdf. Accessed 2 Sept. 2021.

Srivastava, Nitish, et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Edited by Yoshua Bengio, 2014, jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf. Accessed 12 Dec. 2021.

Stephanie. "Mean Absolute Percentage Error (MAPE)." *Statistics How To*, 25 Apr. 2021, www.statisticshowto.com/mean-absolute-percentage-error-mape/. Accessed 26 Aug. 2021.

Whale Wisdom. whalewisdom.com/filer/renaissance-technologies-llc. Accessed 13 Nov. 2021.

Zuckerman, Gregory. *The Man Who Solved the Market: How Jim Simons Launched the Quant Revolution*. PORTFOLIO, 2019.