Frank Tittiger

Estimating SDE Parameters with Neural Networks

Introduction

Stock prices, among other physical phenomena, display seemingly random behavior whilst also following a general trend in a certain direction. A common mathematical representation of this behavior is to model the change in stock prices according to the following stochastic differential equation (SDE):

$$dS(t) = \mu S(t)\ dt + \sigma S(t)\ dB(t)$$

Where $S(t)$ is the stock price, μ is the expected return of the stock, σ is the volatilty of the stock, and $B(t)$ is geometric Brownian motion (defined below). Together, $\mu S(t)\ dt$ is called the drift term and represents the general behavior of the stock over time, while $\sigma S(t)\ dB(t)$ is called the diffusion term, and is meant to encode the random movements stocks display on any given interval. The parameters, μ and σ, are of particular interest as they are often unknown but are extremely useful in application for things such as pricing derivatives, portfolio optimization, etc.

The purpose of this paper is to develop a method of estimating the parameters μ and σ from a given dataset that only contains only time and the values of the discrete changes in the stochastic process over time, under the assumption that the stochastic process is governed by the above SDE. To do this, synthetic datasets will be manufactured with controlled μ and σ parameters, which will then be used to train a neural network to estimate the μ and σ parameters. From here, testing on real stock datasets can then be used not only as a way to measure the efficacy of the model, but also to verify how well the SDE actually models stock pricing.

Geometric Brownian Motion

A key component of the SDE defined above is the $B(t)$ term, which represents geometric Brownian motion, and it is necessary to understand what this is in order to understand what the model we will build is doing. Geometric Brownian motion (GBM) is a continuous stochastic process on an interval of time that can be used to model the random behavior of some physical phenomena such as particle movement or, as in the interest of this paper, stock price movement. Formally, GBM is defined as any process, $B(t)$ that satisfies the following conditions:

- $B(0) = 0$
- $B(t)$ is continuous
- For any $t_1 < t_2 \leq t_3 < t_4$, $B(t_4) - B(t_3)$ is independent of $B(t_2) - B(t_1)$
- $B(t_2) - B(t_1)$ has mean 0 and variance $t_2 - t_1$. Equivalently, $B(t_2) - B(t_1) = \sqrt{t_2 - t_1} \cdot Z$ where Z is the standard normal random variable

Examples of what geometric Brownian motion look like are shown below, where each line represents a separate random walk using Brownian motion:

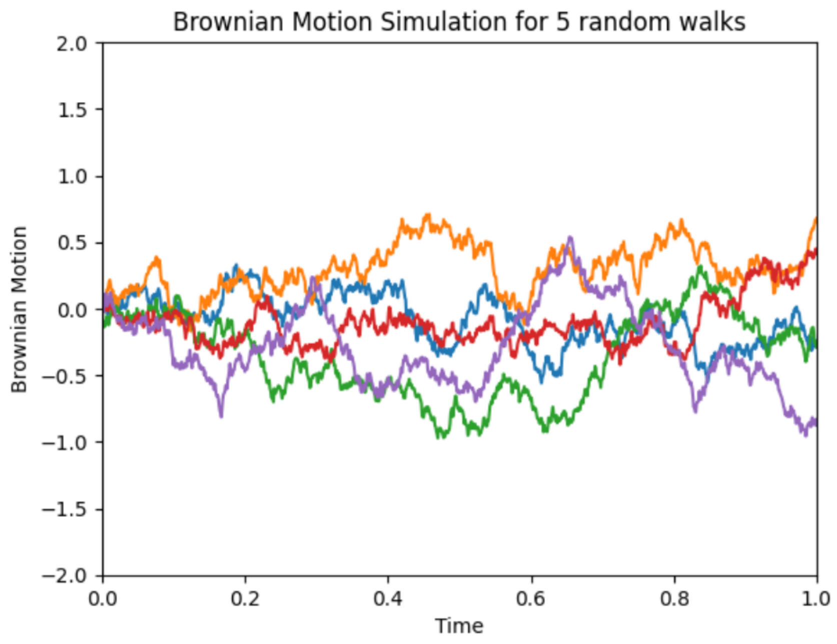

FIGURE 1. 5 sample simulations of $B(t)$

Dataset creation

Recalling that the purpose of this paper it to develop a model that estimates μ and σ, a dataset that replicates the SDE with given μ and σ parameters must be created. To do this, the GBM in diffusion term must first be simulated. Whilst $B(t)$ itself is continuous, in order to model its behavior, its time interval, T, will be discretized into n different sub-intervals. Computationally, $B(t)$ can then be modeled as an array, denoted as $B[t]$ (where t now represents discrete timesteps):

$$B[0] = 0$$

$$B[i] = B[i-1] + \frac{T}{n} \cdot Z$$

For $i \in \{1, 2, ..., n\}$ and Z standard normal. The implementation of this in python can be found in Cell 1 of the Jupyter notebook for this project, and the results of this form of modeling are what is shown above in Figure 1. From here, to use this discretized version of Brownian motion to transform the original SDE into a discrete form, note that the original SDE is solved by:

$$S(t) = S_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma B(t)}$$

From here, this equation is then discretized into another array of length n, denoted as $S[t]$, on the same interval T, by the following:

$$S[0] = S_0$$

$$S[i+1] = S[i]e^{(\mu - \frac{\sigma^2}{2})(\Delta t) + \sigma \Delta B}$$

Where S_0 is the original stock price on the interval, $\Delta t = t_{i+1} - t_i$ and $\Delta B = B[i+1] - B[i]$. This $S[t]$, combined with the discretized time interval T into n equal intervals, and labled with the chosen μ and σ, makes one sample for our dataset (note however, the log returns of S are stored, and not the actual $S[i]$ values, this is to ensure that the model is basing predictions on percent changes, and not on the magnitude of the stock values themselves). Note that each $S[t]$ allows us to provide μ, σ, and S_0, making it easy to accomplish supervised learning tasks. The implementation of creating one sample array is shown in cell 4 of the Jupyter notebook, and the plot of one sample is shown below:

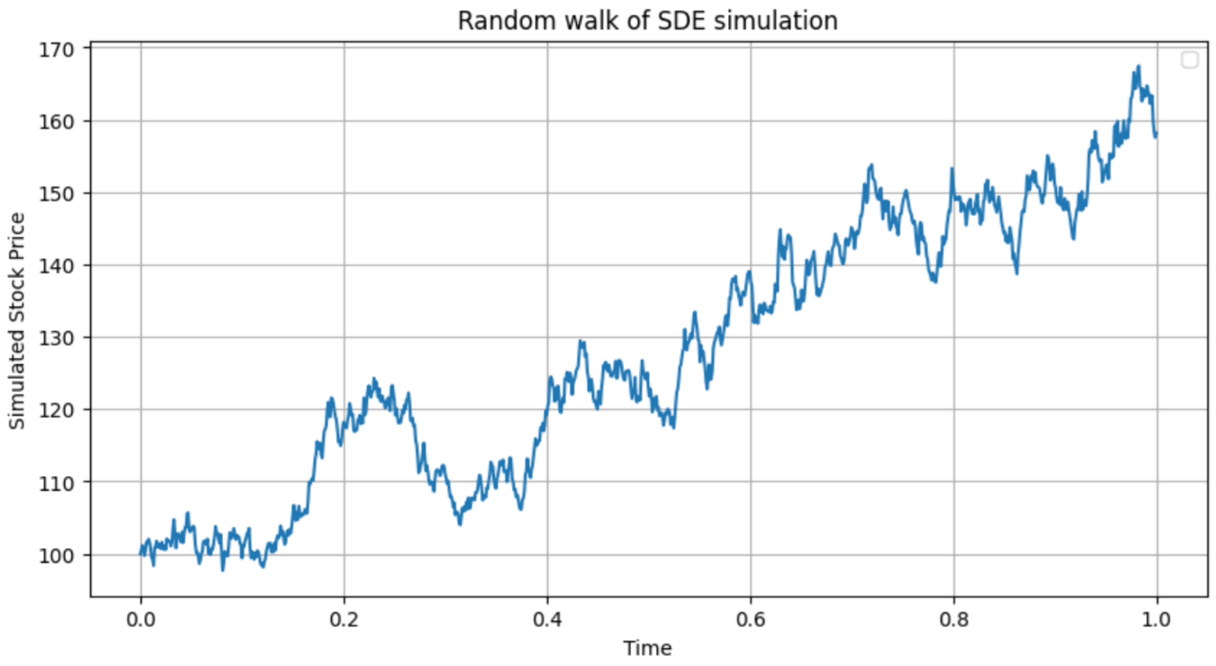

Figure 2. Graph of one SDE walk used to make a training data sample

From here, it is easy to generate a large synthetic training dataset by running the SDE simulation multiple times, and storing the results along with the chosen μ and σ parameters. For this project, a dataset of $1,000$ samples was created using the function found in cell 6. The μ and σ parameters for each sample were chosen from a uniform distribution from -0.15 to 0.15 and 0.02 to 0.30 respectively. For intuition, the form of each sample in the dataset is generally of the form: $\{\text{Time},\ \text{Log Returns},\ \mu,\ \sigma\}$ where Time is an array of length n where the i-th element is the subinterval (t_i, t_{i+1}), Log Returns is an array of length n where the i-th element is $\ln\left(\frac{S[i+1]}{S[i]}\right)$, and μ and σ are the corresponding parameters chosen to generate that samples unique S array. The dataset is then separated into train, validation, and test data, as seen in cell 7.

Model Creation

With a dataset created, developing the actual neural network could then be done. For the structure of the neural network, the input layer would be equal to n, the number of timestpes the original time interval T would be split into. For this case, n was fixed to be at 100. From here two different neural networks were created, one that using adam as its optimizer, and the other using stochastic gradient descent. Both neural networks had 3 hidden layers, each with 64 neurons and using the ReLu activation function, and 2 output layers using the linear activation function to predict the μ and σ values. The creation of these networks is implemented in cell 8. From here, to tune the model, different epoch and batch size combinations were tested to see which combination produced the least amount of error. Arbitrary epoch choices of 5, 10, 20, and 50 were chosen, and batch sizes of 16, 32, and 64. The results are shown in the figures below:

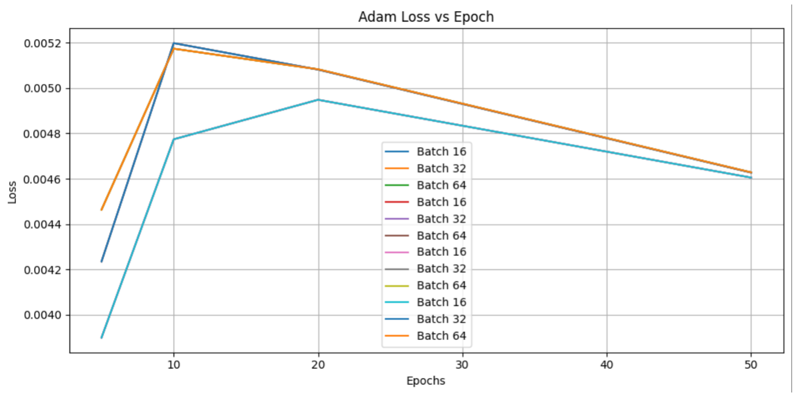

Figure 3. Neural Network with Adam Optimizer Results

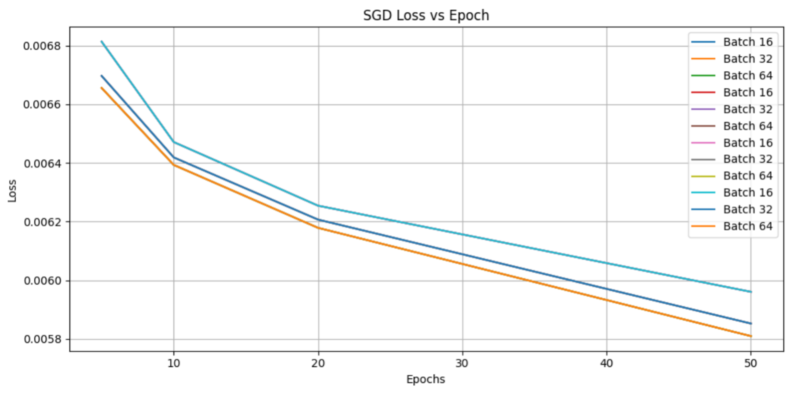

Figure 4. Neural Network with Stochastic Gradient Descent Results

Overall, the Neural Network that used Stochastic Gradient Descent with an epoch choice of 50 and batch size of 32 was chosen. From here, this specific model was then evaluated on the testing data and got an RSME score of 0.0063. Overall, a very low error score.

Test on actual stock data

Finally, with a model chosen, the neural network could then be evaluated on actual stock data. Simple stock data on the past 100 days of AAPL was pulled using code reused from a separate math finance class. From here, the data was then passed into the neural network, and the following (disappointing) predictions resulted as seen in cell 14:

$$\text{Predicted } \mu\text{: } -0.0184 \qquad \text{Actual } \mu\text{: } -0.0004$$

$$\text{Predicted } \sigma\text{: } 0.1632 \qquad \text{Actual } \sigma\text{: } 0.0356$$

There are a number of reasons why such a large difference in the predicted and actual values of μ and σ resulted. Among them, the ones that probably led to such a stark error as seen, are that μ and σ are assumed to be constant in our model, but this likely is not the case in practicality. Furthermore, while GBM is a useful tool for modeling stock prices, it does not describe their behavior perfectly, and a lot of assumptions go into assuming stock prices follow a GBM. Lastly, the model itself probably was not trained on enough samples and had too narrow a range of μ and σ values that were used to train it, but that was unfortunately a constraint of computational power.

Conclusion

The purpose of this paper was to create a neural network that could predict the μ and σ parameters of a specific SDE in the hopes of using it to estimate expected returns and volatility of stock prices. While the model was unsuccessful of being used in a practical case of stock price modeling, it none the less was successful in being able to estimate μ and σ parameters on synthetic datasets. Overall, it seems that while the model is able to predict parameters that follow certain conditions as prescribed by the SDE, it is not robust and is very sensitive to data that does is not generated from the equation directly. Nonetheless, there still seems to be room for improvement, specifically with regard to the tuning of the model. The loss trends in the epoch and batch testing did not seem to flatten out, so it is likely that increasing their range will increase model performance, but unfortunately that is not computationally feasible at this time with my current set up.