Leonardo Stocco Lanfranchi - Final Report - Math 466 - Spring 2025

Mathematical Integration of HMMs and GANs for Financial Data Generation

1. Abstract

The complexity and unpredictability of financial markets pose a persistent challenge for the application of machine learning techniques in the sector, especially in reinforcement learning frameworks where the simulation-to-reality gap hinders the capacity of a model to maintain its training performance in the real-world. This paper proposes a new hybrid architecture that combines Hidden Markov Models (HMMs) with Generative Adversarial Networks (GANs) to generate more comprehensive synthetic financial data. By using HMMs to identify latent market regimes—such as bull and bear markets—and training separate GANs within each regime, we aim to improve the realism and utilization potential of synthetic datasets. The framework consists of a five-step process that incorporates the Baum-Welch and Viterbi algorithms for state inference and data separation, followed by state-specific data generation. This approach addresses the limitations of traditional GANs in capturing dynamic market behavior and offers a more faithful environment for training and testing financial models. Ultimately, the proposed method seeks to support machine learning strategies to strive in financial market scenarios despite historical data limitations.

2. Introduction

Throughout the last centuries, financial markets have evolved to encompass the big majority of products the world has to offer. This structured marketplace of goods spans multiple

countries and product types, and has been getting more complex with the passage of time with the implementation of more exotic structured products, derivatives, and cryptocurrencies, for example. Still, the real complication of the financial markets has always been connected to its nature. Price movements of products are shaped not only by rational expectations and macroeconomic fundamentals, but also by behavioral shifts, information asymmetries, and latent market regimes that evolve over time, while also somewhat representing the current stance of the planet in numbers. Although many have tried, no one has succeeded in solving the markets.

During the last decades, the rise of machine learning allowed for many new techniques of analysis and strategy to be implemented in the markets. But unlike many other fields where machine learning thrived, it showed lackluster results in general. This phenomena can have many explanations, but one thing we can be certain– financial data is not the easiest to work with. Nevertheless, a group of researchers has been trying to challenge that assumption.

Sponsored by the AI4Finance foundation, a nonprofit organization dedicated to advancing AI in the finance industry, this group of researchers has developed a library called FinRL. It is the first open source framework for financial reinforcement learning. It facilitates tremendously the testing and training of models in financial data. Offering data from multiple markets and allowing for users to input their own data, it has received a good deal of notoriety in recent years. Yet, it still faces big challenges ahead. After all, the use of historical data environments to train and test models might create models that lack accuracy in real life given the simulation to reality gap. Low signal to noise ratio, survivorship bias of the data, model overfitting, delay in the data, partial observation of the true state of the market, multi-objective reward functions, and low interpretability are some of the current worries that the team has

regarding the quality of their library and the general usage of financial data for reinforcement learning purposes.

Another aspect that hinders the accuracy of our implementation is the difficulty in creating auxiliary data to support our models. Because of the dynamics of financial markets and the fact that underlying invisible forces sometimes affect the outcome, it is difficult to generate synthetic data that is trustworthy to use in these scenarios. On top of that, if we were to use non-realistic synthetic data in our models, it would worsen the effect of the challenges already imposed in it. However, if we succeeded in producing realistic data that captures market trends and the hidden forces that move the markets, some of the negative effects of these challenges could be mitigated.

In this paper, I will explore a possible solution to improve the way synthetic financial data is produced, aiming at creating the realistic datasets needed so that machine learning models can thrive in financial market scenarios, by using a combination of Hidden Markov Models (HMM) and Generative Adversarial Networks (GAN).

Before diving into how we would combine HMM and GAN, I will first explain these two frameworks.

3. Background

3.1 Hidden Markov Models

A hidden Markov Model is, like any Markov Model, a tool to represent states and their probabilities through a series of observations. The next state depends only on the previous state and nothing else. That is, if it is a first order Markov Model. If a Markov Model is of *n*-order, the

next state depends on the *n* previous states. It is also important to note that these models have discrete time stamps for the occurrence of the observations. All these characteristics are true for all Markov models– hidden or not. What differentiates a Hidden Markov model from the rest is that its states are hidden, meaning there is no way of truly observing them. Hence, the observations themselves are rooted on Stochastic Processes and the probability that they happen is dependent on our hidden state.

There are five elements in a Hidden Markov Model. We have the hidden states S, which are usually connected to the meaning or patterns of the phenomena we are modeling for. If we have Q states, we represent each of them as a number. At time *t*, we will have a Q x 1 vector where the only non-zero number is the Qth number that represents the state. Our observations V have a similar structure. Let's say we have O observations. Then, at time t, we have an O x 1 vector for each of them that has all zeros but the corresponding number to that observation. Our third element is the state transition model (*A*), a Q x Q grid with the probabilities from going from one state to the other. The chance of going from *k* at time *t*-1 to *j* at time *t* is A_{kj}. Hence, each row sums to one. The fourth element is the observation model *B*, which ties states to observations. It is an O X Q matrix. The probability of making observation *y* if we are in state *w*, both at time *t*, is A_{yw}. The fifth and last element of our HMM is the initial state distribution(*Z*), a Q x 1 vector with the probability of starting at each state. So if we are given these five elements, we can fully describe a Hidden Markov Model architecture, where our model M = (*A*, *B*, *Z*).

The joint probability of a sequence of states Q = (q_1,...,q_k) and observations O = (o_1,...,o_k) can be determined by the following formula:

$$P(O, Q \mid M) = Z_{q_1} \cdot B_{q_1}(o_1) \cdot \prod_{t=2}^{k} A_{q_{t-1}, q_t} \cdot B_{q_t}(o_t) \quad (1)$$

If we want, then, to compute the total likelihood of that sequence of observations, we need to marginalize over all possible hidden state sequences:

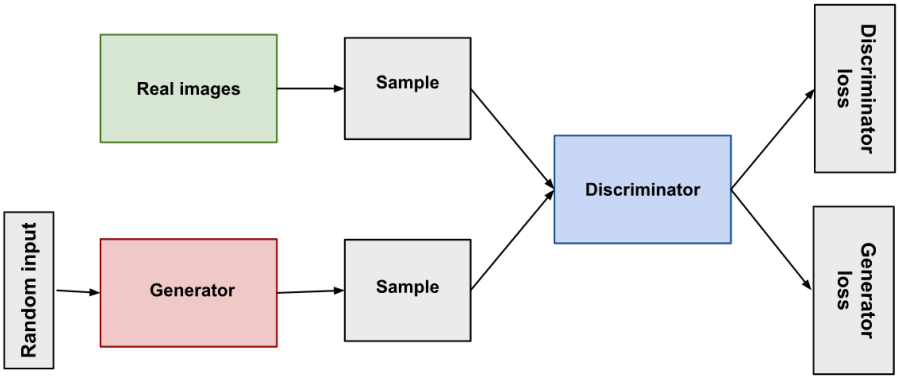
$$P(O \mid M) = \sum_{Q} P(O, Q \mid M) \quad (2)$$

When looking at real world applications, there are three main types of problems that a Hidden Markov Model can be used for. Firstly, if you have all five elements, you can compute the probability of observing a sequence of observations. That is solved by the Forward Algorithm. Secondly, if we have observed a specific sequence, we can find the most probable sequence of underlying hidden states using the Viterbi Algorithm. Thirdly, we can adjust the model parameters to maximize the likelihood of having each observation. This expectation maximization algorithm would modify our A, B, and Z to maximize the likelihood of finding all the observations our data has given our HMM parameters. To fulfil that, a Baum-Welch Algorithm is normally used. We will go more in depth into the mathematics of these when explaining our proposed framework. But before that, we need to talk about Generative Adversarial Networks.

3.2 Generative Adversarial Networks

The concept of a Generative Adversarial Network (GAN) was created in 2014 by Ian Goodfellow after an evening idea was the inspiration for a very successful overnight coding spree. Motivated by the problem of making computers generate images on their own, this new class of models hit the machine learning world by storm and has been thoroughly explored since. Its purpose is to generate new data samples that bear as much resemblance as possible to a particular dataset.

A GAN is made up of not one but two neural networks– normally called a generator *G* and a discriminator *D*. As the name suggests, these networks are implemented and trained in an adversarial process. While the generator learns to generate data that simulates the dataset, the discriminator receives real data and data from the generator and gauges if it is legitimate or not. The intuition behind it is for the generator to become better and better at fooling the discriminator, seeking to maximize its capacity of doing so. The image below shows an example of a general GAN structure.



Both the generator and the discriminator have loss functions that, given the nature of the problem, follow a binary cross entropy structure in discrete form. Its loss function for binary classification tasks, in this case, can be simplified. Only 0 or 1 can be its labels, where 0 labels fake data and 1 true data. In the formula below, y is the true label and $\hat{y}$ is the prediction probability that the label is one.

$$H(y,\hat{y}) = -\sum y\log(\hat{y}) + (1-y)\log(1-\hat{y}) \quad (3)$$

So the loss is small when there is a high $\hat{y}$ and the data is real (y = 1), or on the opposite case– low $\hat{y}$ and data is not real (y = 0). Now, if we adapt this binary cross-entropy to the specific context of both the discriminator and the generator, we get the following formulas,

$$L_D = -\sum_{x \in \chi, z \in \zeta} \log(D(x)) + \log(1 - D(G(z))) \tag{4}$$

$$L_G = -\sum_{z \in \zeta} \log(D(G(z)) \tag{5}$$

where *z* represents the latent data, *D*(x) represents the discriminator's assessment of real data, and *D*(*G*(*z*)) is the discriminator assessment for fake data, or data created by our generator.

Conceptually, this problem can be thought of as a minimax problem. Although discriminator and generator will optimize one at a time, we could say that the following value function is true to describe our setup:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{6}$$

Even though formulas (4) and (5) don't fully translate to (6) because in one case we are minimizing the loss and in the other we are maximizing the expected value, they are fundamentally related. In practice, as $y = \log x$ is steeper approaching $x = 0$ than y = log (1-x) is, the minimax approach is less efficient than the separate loss functions to truly get results out of the generative adversarial network. Still, the minimax value function can be simplified to very insightful formulas. For example, if we were to take the partial derivative of *V*(*D*,*G*) with respect to *D*(x), and simplify, we arrive at the optimal discriminator formula, which is:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \quad (7)$$

Hence, the optimal discriminator intuitively depends simply on the probability densities of the real data and the generated data. It approaches 1 when *x* is much more likely to be under the real data distribution and 0 when it is the other way around. When the generator perfectly matches the real data, it has value 0.5, which is the ideal GAN equilibrium.

Now, if we plug in the optimal discriminator back into the value function, we also find intriguing conclusions. After some derivations, we get that the value function we want to minimize is equal to:

$$V(G, D^*) = -\log 4 + 2 \cdot D_{JS}(p_{data} \parallel p_g) \quad (8)$$

As a consequence, we can observe that when the discriminator is optimal, training the generator is equivalent to minimizing the Jensen-Shannon divergence between the real and fake data. Its result can range from 0, when p_{data} and p_g are the same, to log 2, when the datasets are disjoint.

To contextualize, the Jensen-Shannon divergence is rooted on the Kullback-Leibner divergence, and its architecture makes it use the KL divergence to compare the average of the two datasets to the original datasets themselves. Unlike the KL divergence, which is stated in the formula below for continuous distributions *P* and *Q*,

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} dx \quad (9)$$

the JS divergence uses this building block to generate results that are always symmetric, bounded, and result in zero if the distributions are equal. The below formula describes JS divergences:

$$\mathrm{JS}(P \parallel Q) = \frac{1}{2} D_{\mathrm{KL}}\left(P \,\middle\|\, \frac{1}{2}P + \frac{1}{2}Q\right) + \frac{1}{2} D_{\mathrm{KL}}\left(Q \,\middle\|\, \frac{1}{2}P + \frac{1}{2}Q\right) \tag{10}$$

In conclusion, when our value function simplifies to depend on a JS divergence between our real and fake data, it intuitively means that we want the average of the real and the fake data to be as similar to the real data and to the fake data simultaneously. By minimizing this, the real and the fake data will be indistinguishable, and that is what a GAN is seeking to achieve.

4. Combination of HMM and GAN

In the past, there have already been successful implementations of GAN into financial data. Even big financial institutions like J.P. Morgan has released research papers with test results of GAN usage in an equity options strategy. The complexity and dynamism of the markets lead me to think, however, that if we were to combine Hidden Markov Models to our Generative Adversarial Network usage, the results could be even more pragmatic. That should be the case because some characteristics of the markets are dependent on these two states that the players in the market call bull or bear markets.

Bull and bear markets represent opposite phases of market sentiment and performance. A bull market is characterized by rising prices, investor confidence, and strong economic indicators such as low unemployment and growing GDP. During this period, optimism fuels demand for stocks, often leading to higher valuations and reduced volatility. In contrast, a bear market reflects widespread pessimism, falling prices, and often coincides with economic downturns or

recessions. Investors have the tendency to become risk-averse, volatility increases, and trading volumes may spike as people sell off assets. While bull markets are typically longer and driven by sustained growth, bear markets can be sharp and psychologically taxing, often amplifying short-term noise and uncertainty. Even though a generic generative adversarial network, because of its nature, should generate data that respects the change of bull and bear markets, it would be doing so through partial observations of the state of the market. By adding the Hidden Markov Model state, our synthetic data should respect more thoroughly the ups and downs of prices and, consequently, models trained in it should be able to perform better in real life.

To generate these batches of data, we would work in five steps. First, we would train the HMM on the real data we are given by using the Baum-Welch Algorithm. Second, we would use our trained HMM and the data sequence to find the most probable sequence of hidden states using the Viterbi algorithm. Third, we would separate our data into two based on their resulting hidden state. Fourth, we would run two separate GAN's, one on each dataset, to optimize our generator of bull market synthetic data and bear market synthetic data. Fifth and lastly, we would generate our finalized fake sequence of data by sampling a hidden state sequence from the HMM and for each of its time stamps, generating data based on the GAN corresponding to the current hidden state.

For this framework, our data would consist of the percentage price changes of each product involved in our project, the implied volatility at the time, and trading volume. These are easily accessible data points that are the most reliable to understand the current market scenario. To train our Hidden Markov Model, we would use the Baum-Welch Algorithm. This is a two step expectation maximization algorithm that iterates until convergence to find the best possible set of $M = (A, B, Z)$ given the data. Hence, the transition model, emission model, and initial

distribution are all modified until we either reach a threshold of minimum difference that an update is making or we reach a maximum number of iterations. In our specific case, we could test with hyperparameters to find a good number for the minimum difference and where we should stop the expectation maximization loop. The formula below represents the general intuition behind the Baum-Welch Algorithm, which is to find the model parameters M that maximize the likelihood of observing a given data sequence

$$M^* = \arg\max_M P(x_1, \ldots, x_T \mid M) \tag{11}$$

Even though it requires the usage of the Forward-Backward Algorithm. First, the Forward Algorithm uses recursion to find the probability of being in state *i* at time *t*. Then, the backward probabilities are the chance of seeing the future observations given we are in state *i* at time *t*. When combining both of these, we arrive at the probability of each hidden state at each time, which can be observed in the following generalistic formula,

$$\gamma_t(i) = P(q_t = s_i \mid x_1, ..., x_T) = \frac{\alpha_t(i) \cdot \beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j) \cdot \beta_t(j)} \tag{12}$$

where alpha represents the Forward Algorithm outputs and beta the Backward Algorithm outputs. Based on this inference and the transition probabilities, it enters its maximization step of the loop.

Now that we have our Hidden Markov Model trained, we would then find the most probable sequence of hidden states for our data. That is, we would use the Viterbi Algorithm and, based on our results, assign either the state *bull* or *bear* for each of our observations in the dataset. The engine behind the Viterbi's Algorithm dynamic programming structure can be roughly summarized by,

$$v_t(j) = \max_i \left[v_{t-1}(i) \cdot A_{ij} \right] \cdot B_j(x_t) \quad (13)$$

where v_k is the path probability at time k. So we are identifying the most likely path probability to state j at time t. The recursive nature of the formula makes it depend on the maximum of all previous v's, while also encompassing the transition and emission probabilities.

By the end of this process, we should have a state assigned to each and every of our dataset items. So, following the third and fourth part of our framework, we would separate the data into two subsets– one for each state– and conduct GAN into them separately. The GAN would be executed following what was described in its own section.

Finally, we would have our synthetic data for the two types of states, and now our mission would be to reconstruct datasets that both have the correct temporal structure of bull and bear markets, but would also have state of the art synthetic data for both bear and bull market days, respecting their respective characteristics. To do that, we would first use our Hidden Markov Model to sample hidden state sequences based on our HMM using recursion, the initial state distribution, and transition matrix to yield a full sequence of latent regimes. Then, we would use the two GANs we have already trained to generate synthetic data for all of our sequence based on their hidden state. With that, we would have our finalized batch of synthetic market data that could be used to train trading strategies.

5. Conclusion

The simulation-to-reality gap in financial market datasets has been hindering the advancement and implementation of machine learning strategies in the markets, but many believe an increase in the amount of data would aid in alleviating that. While Generative Adversarial Networks have already been shown to work and are a promising generator of

realistic synthetic data, it alone may disappoint when trying to capture the latent structures—like bull and bear markets—that govern financial time series. By integrating Hidden Markov Models with GANs, the proposed framework addresses this limitation head-on. The HMM identifies and preserves the underlying market regimes, while the GANs separately learn the distributional characteristics of each regime. Together, they enable the generation of synthetic data that is not only statistically realistic but also structurally aware of financial market dynamics. This hybrid approach offers a more faithful simulation environment for training and testing trading strategies, potentially reducing overfitting, enhancing trustworthiness on the data, and fostering generalizability. In doing so, it provides a novel path forward for improving the reliability and robustness of AI applications in finance.

6. Future work

In the future, the continuation of what is idealized in this paper would be to choose a set of market data to truly implement the hybrid HMM-GAN framework to create new simulation datasets and then test their effectiveness at simulating market data.

7. References

Liu, X.-Y., Xia, Z., Yang, H., Gao, J., Zha, D., Zhu, M., Wang, C. D., Wang, Z., & Guo, J. (2023). *Dynamic datasets and market environments for financial reinforcement learning*. arXiv. https://arxiv.org/abs/2304.13174

Degirmenci, A. (2014). *Introduction to Hidden Markov Models*. Harvard University.

https://scholar.harvard.edu/files/adegirmenci/files/hmm_adegirmenci_2014.pdf

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative adversarial networks*. arXiv.
[1406.2661] Generative Adversarial Networks

Tae, J. (n.d.). *The math behind GANs*. Jake Tae.
https://jaketae.github.io/study/gan-math/

Google Developers. (2025, February 26). *Overview of GAN structure*.
https://developers.google.com/machine-learning/gan/gan_structure

Wiese, M., Bai, L., Wood, B., & Buehler, H. (2019). *Deep hedging: Learning to simulate equity option markets*. arXiv.
https://arxiv.org/abs/1911.01700

Knight, W. (2018, February 21). *The GANfather: The man who's given machines the gift of imagination*. MIT Technology Review.
https://www.technologyreview.com/2018/02/21/145289/the-ganfather-the-man-whos-given-machines-the-gift-of-imagination/

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE, 77*(2), 257–286.
A tutorial on hidden Markov models and selected applications in speech recognition - Proceedings of the IEEE