Stock Movement Prediction using Hidden Markov Model and Recurrent Neural Network Architectures

Tsai Ying Liu
tsai.liu@mail.utoronto.ca

Zi You
zi.you@mail.utoronto.ca

Abstract

In recent years, sequential pattern recognition has achieved state of the art in numerous applications. Stock movement prediction is one of the learning tasks due to the sequential nature of time series. To forecast the stock movement, it is critical to model the dynamic structure. Therefore, we first introduce Hidden Markov Models (HMM) and Recurrent Neural Network (RNN), which have proven suitability to represent the arbitrary nonlinear dynamic systems by extracting patterns from temporal data. We then focus on the variants of the RNN, Long-Short term memory (LSTM) and Gated Recurrent Units (GRU), to solve the problem of long-term dependency in time series and provide better interpretation of the hidden dynamics. This paper aims to predict the stock price movements based on the HMM and different recurrent neural network architectures (RNN, LSTM, GRU). In particular, we consider the performance results and propose a reliable trading strategy. The experiments are based on a high-risk stock market and a mature stock index. The performance result shows the HMM with four observation sequences along with 6 states better represents the hidden states among other HMMs. Comparing among the RNN architectures, the GRU obtains up to 77% of accuracy and has the lowest prediction errors in terms of RMSE and MAE. Our comparative analysis indicates that the RNN architectures are better than the HMMs in both accuracy and profit test performances. Specifically, the GRU is the outstanding model in our scenario, which achieves the maximum cumulative profit.

Keywords: Hidden Markov Model, Recurrent neural networks, Long-Short term memory, Gated Recurrent Units, Machine learning, Financial forecasting

1 Introduction

By the Random-walk hypothesis [1], it indicates that stock price movement is a random process. The stock price at each time period is independent to its past price. In other words, there is no predictability for the stock return. In fact, to discover the patterns of the stock market has become one of the popular research areas for machine learning and deep learning. Sequential pattern recognition in stock movement prediction has showed promising performance.

In recent years, stock market behaviour has been influencing by complex and multidimensional factors that are hard to be evaluated. Therefore, having the capacity to interpret the dynamic nature of the stock market is critical for making a reliable trading decision. Using classical econometric models approaches to effectively discover the relations between elements of the sequence and capture the hidden trends is challenge.

Therefore, in the following section, we first introduce the probabilistic Markovian models, such as Hidden Markov model (E. Baum, 1966) [2] [3] and deterministic dynamical models, such as recurrent neural networks, since these are the widely used approaches for forecasting time series. They provide strong computational strength to generate the new observations in the fast-changing stock market environment. Moreover, they produce representation of the hidden context (state or layer), which is the powerful information for sequential classification.

However, the general a-priori assumptions of the HMM and the RNN simplify the sequential learning problems. The assumption of identically independent distribution lost the sequential correlations between data points. The associated hidden dynamics of stock market movement is hard to be interpreted with sufficient inference. Thus, Long-Short term memory (LSTM) (Graves and Schmidhuber) [4], a variant of recurrent neural network, will be introduced in the section 3.2. Even the LSTM has achieved outstanding performances in handwriting recognition [5], vocabulary speech recognition [7], audio detection [8], and visual recognition [9], it is also suitable for stock movement prediction. Firstly, the LSTM isn't restricted by any prior statistical assumption and it is able to capture more abstract patterns from the large dataset under the nature of neural network. Furthermore, it can capture the long-term dependency in time series because of its unique gate design within the hidden layers of recurrent neural networks. In the section 3.3, we will introduce the Gated Recurrent Units (GRU), which has a simpler gate design than the LSTM's.

This paper exploits the novel techniques for stock price prediction and proposes trading decision making (buy, sell or hold) based on Hidden Markov Models (HMM), recurrent neural network (RNN), Long-Short term memory (LSTM), and Gated Recurrent Units (GRU). These models are trained and tested on historical daily stock price and trading volume of the Canopy Growth Corporation (CGC) and S&P 500 index from January 2nd, 2015 to 8th November 2018. The experimental results indicate that all the approaches are feasible. In particular, the GRU is the outstanding model in our scenario, which achieves the maximum cumulative profit.

2 Hidden Markov Model (HMM)

The strong probabilistic framework of the Hidden Markov Model (HMM) (Baum and Petrie 1966) [2][3] provides representation of invisible "states" given observations that are visible but depend on these "states". The factors, such as inflation rate, interest rate, or S&P500 index, can be the potential observations that provide information for the HMM to capture the hidden states in stock market.

In section 2, we introduce the setting, assumptions and three typical problems of HMM. We discuss three algorithms, Forward algorithm, Viterbi algorithm and Baum-Welch algorithm, that can solve these problems and get the optimal parameters.

Model notation:

Number of observation: M
Number of states: N
Number of clock time: T (it also means the length of observation sequence)
Observation sequence: $=\left\{Q_t^{(l)}, t = 1,2,\dots T, l = 1,2,\dots L\right\}$ (l indicate different inputs in one observation.)
Hidden state: $Q = \{q_t, t = 1,2,\dots T\}$
Possible value of each state: $S = \{S_i, i = 1,2,\dots N\}$
Possible observation value per state: $V = \{v_k, k = 1,2,\dots M\}$
Transition matrix: $A = (a_{ij}), a_{ij} = P(q_t = S_j | q_{t-1} = S_i), i,j = 1,2,\dots N$
Initial probability of being in state S_i at time t=1: $P = P_i, P_i = P(q_1 = S_i), i = 1,2,\dots N$
Observation probability: $B = b_{ik},\ b_{ik} = P(Q_t = v_k | q_t = S_i) = b_i(O_t) = P(Q_t | q_t = S_i)$
Parameter of HMM: $\lambda \equiv \{\mathrm{A}, B, p\}$
Note: we assume the observation probability is Gaussian distribution, so we can write $b_i(O_t) = N(O_t = v_k, \mu_i, \sigma_i)$. As a result the parameter of HMM can be wrote as $\lambda \equiv \{\mathrm{A}, \mu_i, \sigma_i, p\}$

2.1 The Hidden Markov Model must satisfy following assumptions.

1. The hidden state must satisfy the first order Markov Property.
$$a_{ij} = P(S_t | S_{t-1,} S_{t-2} \dots S_1) = P(q_t = S_j | q_{t-1} = S_i)$$
State i at time t is a function of state j at $t-1$. In other word, the state i at time t only depend on state j at $t-1$.
2. The transition probability matrix is constant and it is independent of time at which the transitions happen.
$$P(q_{t_1+1} = S_j | q_{t_1} = S_i) = P(q_{t_2+1} = S_j | q_{t_2} = S_i)$$
t_1 and t_2 could be any value.
3. The output observation only depend on the state that produce the observation.

2.2 Three Problems of Hidden Markov Model.

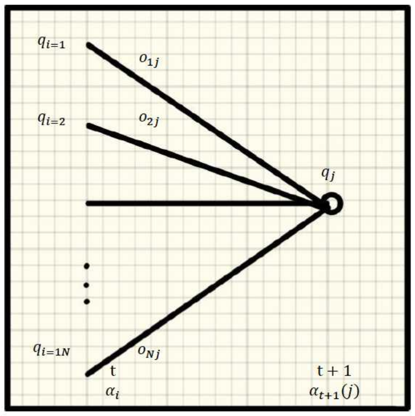
1. Model evaluation problem. For this problem, we want to compute the probability that an observation sequence was produced by model. If there are several models, then the model which produces highest probability can best match the observation.
2. Model estimate problem. Given a model and an observation sequence, find the most probable hidden state sequence.
3. Model training problem. Given a model and an observation sequence, learn the best HMM parameters.

2.2.1 Forward algorithm:

There are several algorithms to solve the evaluation problem. We adopted the forward algorithm here because compare with other method, forward algorithm can significantly reduce the amount of computation.

$\alpha_{t(i)}$ is forward variable.

Figure1. Sequence of operation required for the computation of the forward variable.



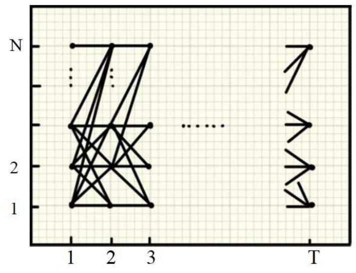
$(O_1, O_2, \dots O_t)$ is an observation sequence which stops at time t. $\alpha_{t(i)}$ means the joint probability of the a sequence is observed and state stop at q_i at time t under given a model.

$$\begin{cases} \alpha_1(i) = \pi_i b_i(O_1), & t = 1, \quad 1 \le i \le N \\ \alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t\,(i)\alpha_{ij}\right] b_j(O_{t+1})\,, & t = 1,2, \dots T-1, 1 \le j \le N \end{cases}$$

$\alpha_{t(i)}\alpha_{ij}$ means the probability of joint even that $O_1 \dots O_t$ is observed and state transfer form q_i to q_j at time t+1. Because there are N passible state value at time t, we need to make a summation in order to get the probability of q_j at time t+1 with all the possible previous state. $\left[\sum_{i=1}^{N} \alpha_{t(i)}\, \alpha_{ij}\right] b_j(O_{t+1})$ means the probability of q_j at time t+1 times the probability of O_{t+1} in state q_j. After we sum all these terminal forward variables together, we get the probability of an observation sequence given a model λ:

$$\mathrm{P(O|\lambda)} = \sum_{i=1}^{N} \alpha_T(i) = P(O_1, O_2, \dots O_T, i_T = q_i|\lambda)$$

Figure 2. Lattice Structure



2.2.2 Viterbi algorithm.

For the second problem, there are two methods are used to find the optimal state sequence. The first method is to choose a most likely individual state i_t. The advantage of this algorithm is that it allows us to maximize the expected number of correct states. However, it fails to consider the global structure. For example, if there is a situation that state I can't transfer to state j, then the state sequence which is produced by this method is impossible. In our experiment, we have adopted the Viterbi algorithm. This algorithm can find the single best path of states instead of finding individual state. This approach can better consider the global constrains.

First three steps are very similar to the forward algorithm:
Step1: initialization ($T = 1$)
$\delta_1(i) = \pi_1 b_1(O_{1j}) \quad 1 \le i \le N$
$\Psi_{1(i)=0}$
Stpe2: recursion ($2 \le t \le T, 1 \le i \le M$)
$\delta(i) = \max\limits_{1 \le j \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \quad 1 \le i \le N$
This equation maximizes the probability of O_t given the probability of state j which is reached at time t via state q_i at time $t-1$.
$\Psi_{(j)} = \arg\max\limits_{1 \le j \le N} [\delta_{t-1}(i) a_{ij}] \quad 1 \le i \le N$
This equation is used to find the state which can maximize the probability of joint event that $O_1 \ldots O_{t-1}$ is observed and q_j is reached at time t via state q_i at time $t-1$.
Step3: termination
$P^* = \max\limits_{1 \le j \le N} \delta_T(j)$
P^* is a maximal joint probability of $O_1 \ldots O_T$ and $q_i = i_T$. We need to find the optimal state sequence associated the given observation sequence by $i_T^* = \arg\max\limits_{1 \le j \le N} [\delta_T(j)]$.
Step4: path backtracking
$i_T^* = \Psi_{t+1}(i_{t+1}^*), t = T-1, T-2 \ldots 1$

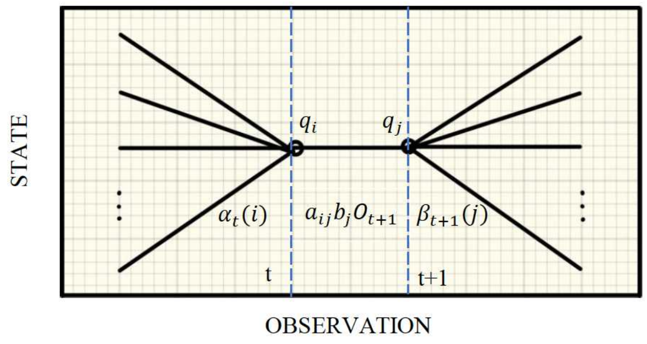
2.2.3 Baum-Welch algorithm.

Solving problem 3 is most difficult part in the process of HMM. We used the Baum-Welch algorithm to adjust model parameters so that the probability of observation sequence can be maximized.

$$\xi_t(\mathrm{i},\mathrm{j}) = \mathrm{P}(j_t = q_i, j_{t+1} = q_j | \mathrm{O}, \lambda) = \frac{\alpha_t(i) a_{ij} b_j O_{t+1} \beta_{t+1}(j)}{P(O|\lambda)}$$

We can present the equation as following figure. $\beta_{t+1}(j)$ means, given state q_j at time t+1 and model λ, the probability of a partial observation sequence $O_{t+1}, O_{t+2} \ldots O_T$.
It measures the probability of a path in state q_i at time t making a transition to state q_i at time $\xi_t(\mathrm{i},\mathrm{j})$ given an observation sequence and model λ.

Figure3. Computation required for the calculation of $\xi_t(\mathrm{i},\mathrm{j})$



$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j)$$

$\gamma_t(i)$ measures the probability of state q_i at time t under all possible state value at time $t + 1$.
$\sum_{t=1}^{T-1} \gamma_t(i)$: Expected number of transitions make from q_i. (Only consider T-1 period because there is no transition can be made at period T).
$\sum_{t=1}^{T-1} \gamma_t(i)$: expected number of transitions from q_i to q_j.
The Baum-Welch re-estimation formulas for A, B and π:
$\overline{\pi_\iota} = \gamma_1(i), \qquad 1 \leq i \leq N$

$$\overline{a_{\iota J}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T=1} \gamma_t(i)} = \frac{E\left[number\ of\ q_i \rightarrow q_j\right]}{E[number\ of\ transition\ make\ form q_i\ ]}$$
$$= E[probability\ of\ q_i \rightarrow q_j\ ]$$

$$\overline{b_J(k)} = \frac{\sum_{t=1, O_t=k}^{T} \gamma_t(j)}{\sum_{t=1}^{T=1} \gamma_t(j)} = \frac{E[number\ of\ time\ being\ in\ state\ j\ and\ O_t = k\ ]}{E[number\ of\ time\ being\ in\ state\ j\ ]}$$
$$= E[probability\ of\ O_t = k\ |state\ j\ ]$$

First, we need to randomly define a set of initial parameter λ. Then we re-estimate the model as $\overline{\lambda} \equiv \{\overline{\pi_\iota}, \overline{a_{\iota J}}, \overline{b_J(k)}\}$. The new $\overline{\lambda}$ satisfy $P(O|\overline{\lambda}|) > P(O|\lambda)$. During iteratively put $\overline{\lambda}$ into above re-estimation calculation, the observation of O can be continuously improved until the value of $\overline{\lambda}$ converge.

3. Recurrent Neural Networks Architectures

In section 3, we review three different recurrent neural networks architectures, RNN, LSTM, and GRU, for time series sequence learning. We focus on the scenario of the stock market. We illustrate their internal structures along with the computational procedures. We present their properties, limitations, and advantages when applying to time series data that is dependent.

To better demonstrate the powerful Gated RNNs, LSTM and GRU, in Sec. 3.2 and 3.3, we first present the background and the challenge of Recurrent Neural Networks (RNN) in Sec. 3.1.

3.1 Recurrent Neural Network

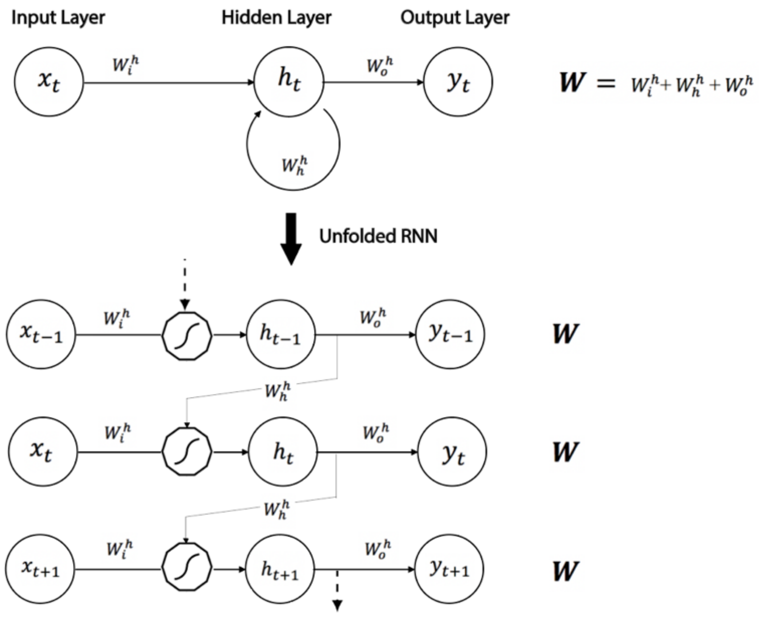
Recurrent neural network (RNN) is a powerful neural network family for sequential learning, which can be applied to time series prediction. RNN's cyclical connections present the temporal relationship of the time series to the input layer, and thereby influence the output. Its architecture of high dimensional hidden states can represent the complex and nonlinear dynamics of stock market. Furthermore, RNN has the capability to handle dependent inputs at some degrees. Here we introduce simple RNN, which has only a single and self-connected hidden layer, for clarity.

At each time t, where t = 1 to T, the output vectors $\widehat{y_t}$ are influenced by the recurrent connections between current input vectors x_t and previous hidden state vectors h_{t-1} . It will update the current hidden state vectors h_t by iterating the following equations. Bias weights of h_t are omitted for better illustrating.

$$h_t = tanh\, W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \qquad \hat{y_t} = \mathrm{softmax}\,(W_y h_t + b_y)$$

Gradient descent optimization is applied to find the optimal parameters**.** During the training time, gradient descent will keep adjusting the network weights W based on the loss functions L_k and gradient $\partial L_k / \partial W_k$ at each epoch k until the algorithm converges. The direct relation between weights and the loss function are found when unfolding RNN. The unfolded RNN architecture represents as an equivalent infinite, acyclic and directed graph rather than cyclic graph (Bianchi, 2017) [12]. At each replicated layer with same operation, the input weights matrix, the hidden weights matrix , and the output weights matrix are constructed to keep unfolded RNN sharing the same weight at each time step.

Figure 4. Unfolded RNN represents as an equivalent infinite, acyclic and directed graph with same weight matrices at each time interval.



Backpropagation through time can be used to compute the gradients $\partial L_k / \partial W_k$ in the temporal neural model, RNN. Loss function L_k evaluates the prediction error between actual future input x_k and desired output $y_k{}^*$ with regulation $R_\lambda(W_k)$. In addition, regulation reduces the issue of overfitting on the training data.

$$L_k = E\,(x_k\,, y_k{}^*) + R_\lambda(W_k)$$

The parameters can be estimated by minimizing the following error function, Mean Square Error (MSE).

$$MSE\,(y_k\,, y_k{}^*) = \frac{1}{|X_k|} \sum_{x \in X_k} (y_x - y_x{}^*)^2$$

We can update our parameters based on the Stochastic Gradient Descent (SGD), which is a widely used and safe gradient-update method. It converges to a local minimum if the learning rate η in the following update equation is sufficiently small. The learning rate η can be reducing by the methods of step decay, exponential decay or the fractional decay.

$$W_{k+1} = W_k + \eta \nabla_k (W_k)$$

Adam (Kingma and Ba) [19] is another better method to update the weights $\boldsymbol{W_k}$, which we implement in our experiment. Its default values of β_1, β_2, and ε are 0.9, 0.999 and 10^{-8} respectively.

$$\text{First moment: } m_k = \beta_1 m_{k-1} + (1-\beta_1) \nabla L_k (W_k^{(i)})$$
$$\text{Second moment: } v_k = \beta_2 v_{k-1} + (1-\beta_2) \nabla L_k (W_k^{(i)})^2$$

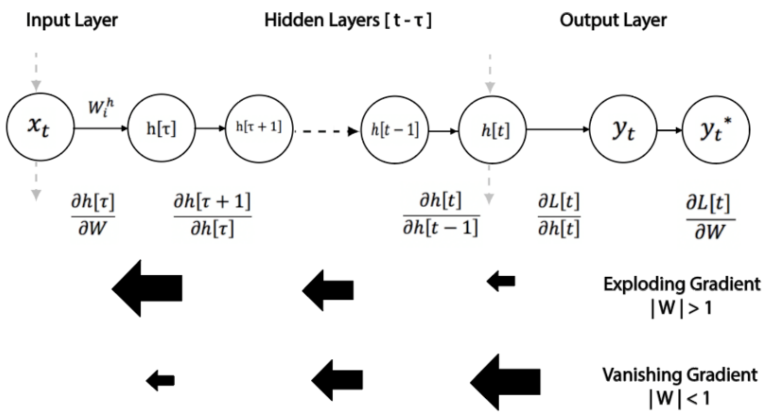
$$\widehat{m_k} = \frac{m_k}{1-\beta_1^k} \quad , \quad \widehat{v_k} = \frac{v_k}{1-\beta_2^k}$$

$$\boldsymbol{W_{k+1}} = \boldsymbol{W_k} + \frac{\boldsymbol{\eta}}{\sqrt{\widehat{v_k}+\varepsilon}} \widehat{m_k}$$

Large depth RNN has difficulty on extracting the long-term dependencies from sequential data. The unstable relationship between the parameters and the hidden states dynamics manifests RNN into the issue of vanishing and exploding gradient (Bengio et al., 1994). The partial derivatives $\partial h[t] / \partial h[\tau]$ of the states either decrease exponentially or grow exponentially when the number of layers $t - \tau$ increases. If there is large error estimation of the desired output $\mathbf{y_t}^*$, the back-propagation dynamics make it hard to modify the computation at the beginning of the sequence.

$$\frac{\partial L[t]}{\partial W} = \sum_{\tau} \frac{\partial L[t]}{\partial h[t]} \frac{\partial h[t]}{\partial h[t-1]} \cdots\cdots \frac{\partial h[\tau+1]}{\partial h[\tau]} \frac{\partial h[\tau]}{\partial W}$$

Figure5. The issue of exploding and vanishing gradients in RNN


Input Layer
Hidden Layers [t - τ]
Output Layer
x_t
W_i^h
h[τ]
h[τ + 1]
h[t − 1]
h[t]
y_t
y_t^*
$\frac{\partial h[\tau]}{\partial W}$
$\frac{\partial h[\tau+1]}{\partial h[\tau]}$
$\frac{\partial h[t]}{\partial h[t-1]}$
$\frac{\partial L[t]}{\partial h[t]}$
$\frac{\partial L[t]}{\partial W}$
Exploding Gradient |W| > 1
Vanishing Gradient |W| < 1

Fortunately, there are numerous solutions to solve exploding gradients, such as gradient clipping. We clip the gradient vector g if gradient norm $\|g\|$ is larger than the certain maximum threshold η to avoid extreme parameter changes (Pacanu et al., 2012).

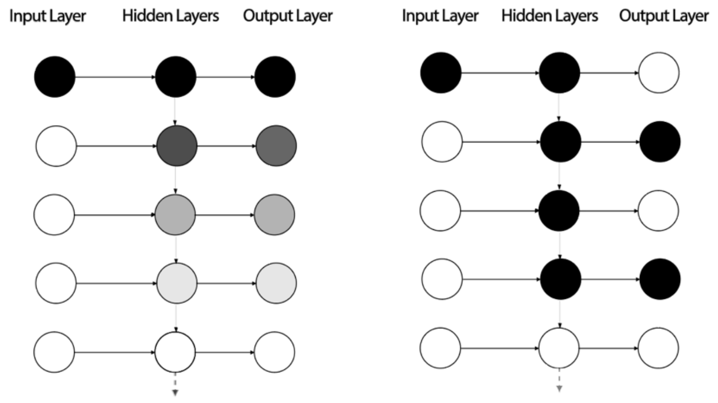
$$g \leftarrow \frac{\eta g}{\|g\|}$$

To solve vanishing gradients, we consider a fundamental change to the RNN's architecture by introducing the gated architecture in recurrent neural networks in Sec 3.2 and 3.3.

3.2 Long Short-Term Memory (LSTM) networks

Long-term dependencies. The vanishing gradient problem causes the sensitivity of past information decrease or even vanished after many time steps, which was illustrated in the figure6. In other words, RNN may lose the past information that is important to recognize the long-term behaviors of time series data. Therefore, RNN is not an ideal structure when we intent to interpret the hidden states in the stock market over long time periods.

Figure6. The issue of long-term dependencies. The unit with darker color has more sensitivity. **Left:** RNN **Right:** LSTM



Therefore, LSTM, the variation of the RNN, was introduced by Hochreiter and Schmidhuber (1991) to solve the issue of vanishing gradient. Its complex "memory blocks" in the hidden layers replaces the simple sigmoid units in RNN. The networks have the capacity to behave differently toward different information. It can decide to be more sensitive to the information that are helpful to learn the long-term dependencies of longer sequential data.

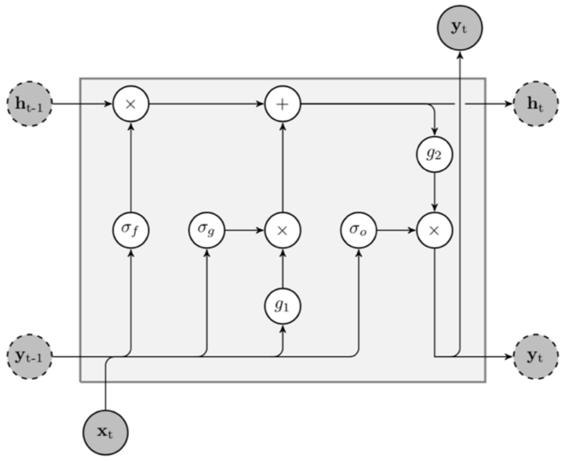
Each memory unit is composed with three multiplicative gates, which can forget, remember, or access the information over long time period. These gates are depended to the current input x_t and previous cell output h_{t-1}. The forget gate f, the input gate i, and the output gate o transform the memory context by the sigmoid function and tanh function, which compress the values within 0 to 1 and -1 to 1 respectively. This special architecture design allows gradients flow backwards successfully and control the model derivative not being vanished over many time steps. The forget gate decides what context information from previous cell state h_{t-1} should be forget and reset it to zero. The input gate then updates the value of cell state from c_{t-1} to c_t after deciding what value to store along with the candidate state g. The output gate *o* filters the part it decides to be as output. Bias weights are omitted for better illustrating.

$$\begin{pmatrix} f \\ i \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W \begin{pmatrix} \mathrm{x_t} \\ \mathrm{h_{t-1}} \end{pmatrix}$$

$$\mathrm{c_t} = \mathrm{f} \odot \mathrm{c_{t-1}} + \mathrm{i} \odot \mathrm{g}$$

$$\mathrm{y_t} = \mathrm{o} \odot \tanh(\mathrm{c_t})$$

Figure7. A recurrent cell in the LSTM architecture



3.3 Gated Recurrent Units (GRU)

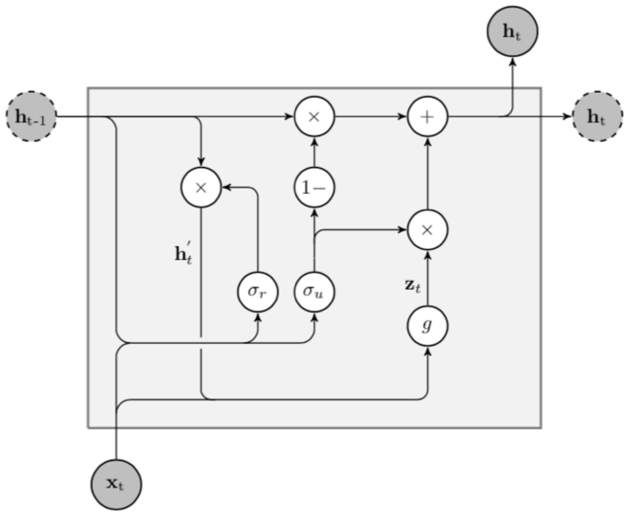
GRU, another gated RNNs, (Cho et al., 2014b; Chung et al., 2014, 2015a; Jozefowicz et al., 2015; Chrupala et al., 2015) simplify the procedure of LSTM by combining the forget gate and the input gate together. This single gating unit successfully controls what information to reset and what to update at the same time. It reduces the training time effectively. Bias weights are omitted for better illustrating.

$$\begin{pmatrix} \mathrm{r} \\ \mathrm{z} \end{pmatrix} = \begin{pmatrix} \mathrm{sigm} \\ \mathrm{sigm} \end{pmatrix} W \begin{pmatrix} \mathrm{x_t} \\ \mathrm{h_{t-1}} \end{pmatrix}$$

$$\widetilde{h_t} = \tanh\left(\mathrm{W_x h_t} + \mathrm{W_g}(\mathrm{r} \odot \mathrm{h_{t-1}})\right)$$

$$\mathrm{h_t} = (1 - \mathrm{z}) \odot \mathrm{h_{t-1}} + \mathrm{z} \odot \widetilde{h_t}$$

Figure8. A recurrent cell in the GRU architecture

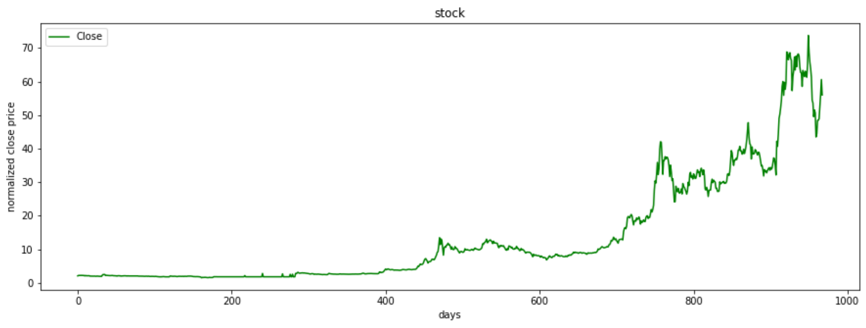


4 Experiments

Liquidity in the cannabis stock market is starting to pick up after legalization of marijuana. The growing cannabis market and companies make forecasting cannabis stock price movement become more valuable for active investors. We selected the Canopy Growth Corporation (CGC) as the representative of the cannabis industry, since it has the top trading volume among other competitors. We modeled HMM, RNNs, LSTM, and GRU to predict whether cannabis stock will go up or down in a period of daily regular trading hours. The prediction result is used for constructing reliable trading decisions (buy, sell, or hold).

Data Description. The dataset of historic daily stock price (opening, high, low, and closing price) of Canopy Growth Corporation (CGC) and S&P 500 index were collected from yahoofinance.com. We had a total of 963 records starting from January 2nd, 2015 to 8th November 2018. We predicted the difference between today and next day's closing prices.

Plot 1. The daily close stock price of CGC from 2015.01.02 to 2018.01.29



4.1 Hidden Markov Model Methodology

We use 80% of the data to train the models. In order to compare the cumulative profit that produced by the HMM with other models, we use10% of data as test set. Initially, we have created 3 models. In the first model we only used the Canopy's close price data as a single observation sequence. In the second model, we use Canopy's close, open, lowest and highest price as a mixed observation sequence (4-observation). In the last model, based on the second model, we add information of TSX index as observation (9-observation).

Table1. Dataset arrangement for training, validating and testing during the whole sample period.

dataset	Number of observations	Time period
Training dataset	771	2015.01.02 ~ 2018.01.29
Testing dataset	96	2018.06.16 ~ 2018.11.08

4.1.1 Model Selection

We use Akaike information criterion (AIC) and Bayesian information criterion (BIC) to choose the numbers of states.

$$AIC = -2\log(L) + 2k$$
$$BIC = -2\log(L) + klog(n)$$

L: Likelihood function of model.
k: Number of free parameters in the model.
n: Size of observation

Since $2k$ and $klog(n)$ are penalty term of AIC and BIC, the best model is the one which gets the minimum value of AIC and BIC.

Table2. AIC and BIC of the different state number.

Methods	AIC	BIC
2-state HMM	2954.282	2986.816
3-state HMM	2365.145	2430.212
4-state HMM	2045.505	2152.402
5-state HMM	1317.57	1475.592
6-state HMM	1318.615	1537.057

From the table above, we find both AIC and BIC indicate that 5 states is the optimal choice for the first model. For the second model, 6 states has minimal AIC and BIC. For the last model, 5 states has minimal AIC and BIC.

4.1.2 Model Evaluation

We get optimal model parameters through Baum-Welch algorithm.

First Model:

Table 3. Initial State Probability

Pr1	Pr2	Pr3	Pr4	Pr5
1/5	1/5	1/5	1/5	1/5

Table4. Transition Matrix

	To S1	To S2	To S3	To S4	To S5
Form S1	0.985	0.015	0.000	0.000	0.000
From S2	0.000	0.996	0.000	0.000	0.004
From S3	0.009	0.000	0.950	0.041	0.000
From S4	0.000	0.000	0.021	0.979	0.000
From S5	0.000	0.000	0.000	0.000	1.000

Table5. Response Parameters

	Average (Close)	Standard Deviation (Close)
S1	4.407	1.055
S2	9.790	1.577
S3	2.673	0.139
S4	1.905	0.147
S5	24.616	8.265

Second Model:

Table6. Initial State Probability

Pr1	Pr2	Pr3	Pr4	Pr5	Pr6
1/6	1/6	1/6	1/6	1/6	1/6

Table7. Transition Matrix

	To S1	To S2	To S3	To S4	To S5	To S6
Form S1	0.980	0.020	0.000	0.000	0.000	0.000
From S2	0.000	0.936	0.000	0.064	0.000	0.000
From S3	0.003	0.000	0.997	0.000	0.000	0.035
From S4	0.000	0.011	0.021	0.954	1.000	0.000
From S5	0.000	0.000	0.000	0.000	0.007	0.975

Table8. Response Parameters

	Average (Close)	Standard Deviation (Close)	Average (Open)	Standard Deviation (Open)	Average (Low)	Standard Deviation (Low)	Average (High)	Standard Deviation (High)
S1	3.851	0.275	3.858	0.259	3.783	0.278	3.914	0.261
S2	6.610	0.839	6.532	0.818	6.283	0.842	6.788	0.815
S3	2.135	0.383	2.144	0.385	2.109	0.374	2.164	0.396

S4	8.559	0.529	8.564	0.544	8.396	0.536	8.726	0.536
S5	24.791	8.200	24.478	8.067	23.488	7.462	25.619	8.599
S6	10.965	1.099	11.007	1.143	10.635	1.041	11.292	1.264

Third Model:

Table9. Initial State Probability

Pr1	Pr2	Pr3	Pr4	Pr5
0	0	0	0	1

Table10. Transition Matrix

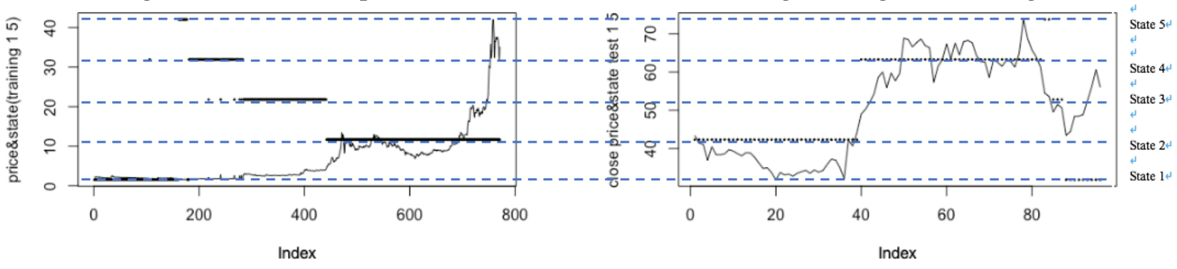
	To S1	To S2	To S3	To S4	To S5
From S1	0.978	0.000	0.022	0.000	0.000
From S2	0.009	0.958	0.033	0.000	0.000
From S3	0.017	0.030	0.953	0.000	0.000
From S4	0.000	0.016	0.000	0.952	0.032
From S5	0.000	0.000	0.000	0.008	0.992

Table11. Response Parameters

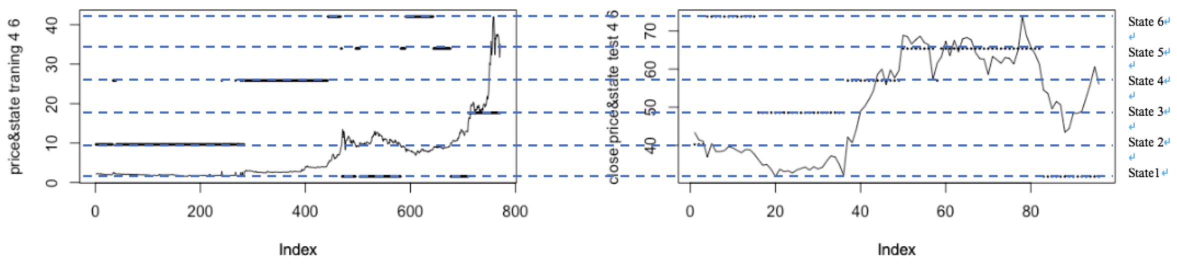
	Average (Close)	Standard Deviation (Close)	Average (Open)	Standard Deviatio n (Open)	Average (Low)	Standard Deviatio n (Low)	Average (High)	Standard Deviatio n (High)
S1	21.011	8.823	20.823	8.620	19.953	8.124	21.751	9.174
S2	8.188	0.867	8.139	0.831	7.947	0.864	8.367	0.877
S3	10.352	0.824	10.411	0.846	10.116	0.767	10.603	0.874
S4	3.967	0.662	3.960	0.648	3.860	0.625	4.054	0.681
S5	2.123	0.372	2.133	0.376	2.099	0.367	2.150	0.381
	Average (Volume)	Standard Deviation (Volume)	Average (TSX Close)	Standard Deviatio n (TSX Close)	Average (TSX High)	Standard Deviatio n (TSX High)	Average (TSX Low)	Standard Deviatio n (TSX Low)
S1	7076285.9	4661446.5	15993.64	316.328	16028.65	320.511	15955.42	322.179
S2	2237558.9	2754743.9	15147.15	250.257	15202.60	250.763	15096.57	249.550
S3	2734658.9	1884602.9	15453.33	185.260	15500.89	186.107	15404.27	182.023
S4	1649236.8	1492071.2	14553.62	368.225	14605.44	356.622	14498.23	382.962
S5	151181.3	225826.9	14021.37	847.186	14092.81	837.864	13942.73	859.752
	Average (TSX Volume)	Standard Deviation (TSX Volume)						
S1	188639900	51502607						
S2	185308033	46661481						
S3	222812567	92065200						
S4	195587807	48645432						
S5	217498390	70208433						

After we found all optimal parameters for each model, we use the Viterbi algorithm to find the hidden path of three models. Following plots describe how the state sequences in training set and test set looks like.

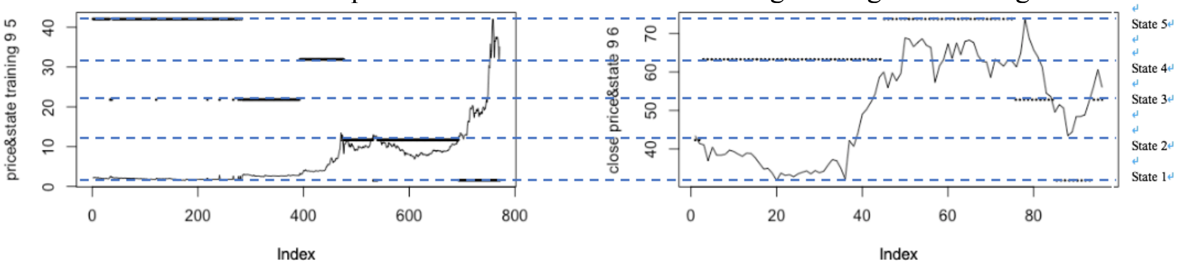
Plot2. Single observation sequence with 5 states- Left: In training set. Right: In testing set.



Plot3. 4-observation sequences with 6 states- Left: In training set. Right: In testing set.



Plot4. 9-observation sequence with 5 states- Left: In training set. Right: In testing set.



In plot 2 (left), there is a slowly increasing in close price when market in state 3, 4 and 5. If you buy stock at these periods, you will make profit. The situation which state 2 show is really ambiguous. In the period of state 5, there is a long-term decrease and a sharp increase. The model of 1-observation can't capture hidden state well. In plot 3 (left), the ambiguity in the first model have been significant improved. After we added five more observations, the trend of market in state 2 again becomes ambiguous. However, in the last 100 days, the performance of model 3 is better than model 1. Also, from above plots and transition matrixes we found that these states are very 'sticky', therefore it's hard to transfer from one state to another. This caused our experiment lack accuracy.

4.2 Recurrent Neural Networks Architectures (focus on LSTM) Methodology

Environment. We train and evaluate the RNN, LSTM, and GRU model in Python 3, Keras v 2.1.2 with TensorFlow v 1.5.0 as backend.

Dataset arrangement. In our experiment, 80% of the data is used to train the models and update the model parameters. The data of following three months is used to tune the hyper-parameters, which are the number of epochs, size of batches, and number of neurons, which we get an optimal model setting. Finally, we use this optimal model to predict the daily performance accuracy of each model.

Table12. Dataset arrangement for training, validating and testing during the whole sample period.

dataset	Number of observations	Time period
Training dataset	771	2015.01.02 ~ 2018.01.29
Validating dataset	96	2018.01.30 ~ 2018.06.15
Testing dataset	96	2018.06.16 ~ 2018.11.08

4.2.1 LSTM Evaluation

We present an accuracy performance analysis considering stock movement prediction problems. We tune the hyperparameters of LSTM and provided its optimal configuration. We discuss the effect of dropout to see if this technique can reduce overfitting during the training time. We indicate the optimizer and loss function that was used in training models. We then compare the performance of LSTM with RNN and GRU.

The goal of our experiments is to demonstrate that which model setting has the lowest prediction errors in terms of root mean squared error (RMSE) and mean absolute error (MAE).

Optimal Hyperparameters. We diagnose the hyperparameters that can highly influence the result and randomly choose the values of the hyperparameters that bring less effect to the result. We tune different numbers of epochs, sizes of batches, and numbers of neurons when training the LSTM model to find out the optimal configuration when predicting the difference between today and yesterday's stock price. The tuning procedures are evaluated on the validating dataset. There are 14 random configurations for LSTM are evaluated. The prediction accuracy of the optimal setting would be computed on the testing dataset.

We set our first 10 configurations as following and showed the accuracy result in the table. In the diagnostic of the number of epochs, we set 4 batches and 1 neuron numbers along with epochs at 100, 500, and 1000 respectively. In the diagnostic of the sizes of training batches, we set 1000 Epochs and 1 neuron along with batch (es) at 1, 2, and 4 respectively. In the diagnostic of the numbers of neurons, we have 1000 epochs and 1 batch along with neuron(s) at 1, 2, 3, and 4 respectively.

Table13. Accuracy (MSE) of different methods

Methods		MSE
4 batches 1 neuron	100 Epochs	3.63
	500 Epochs	3.71
	1000 Epochs	**3.62**
1000 Epochs 1 neuron	1 batch	**3.55**
	2 batches	3.61
	4 batches	3.69
1000 Epochs 1 batch	1 neuron	3.617
	2 neurons	3.596
	3 neurons	**3.415**
	4 neurons	6.382

The criteria tests' results indicate that LSTM with 1000 epochs, 1 batch and 3 neuron worked the best among other settings. Under this setting, we compared the performance of LSTM with RNN and GRU. In our experiment, due to the computational limitation of the environment, where each result in the above table took over an hour to compute, we did not perform the diagnostic of optimal configurations for RNN and GRU. In this situation, both our environment and experimental test harness need further improvement.

Dropout. The issue of overfitting occurs because nodes between large neural nets are easy to co-adapt at test time. Dropout, the regularization technique, is introduced to slow down learning and effectively reduce overfitting of the network. Dropout is implemented by randomly drop nodes from input layers or hidden layers of the neural networks with a given probability p at training time. The dropout probability p is normally between 0.2 and 0.5, which is not extremely low or large to effect learning of models. Dropout has been shown state-of-the-art results over other weight regularization methods, such as L1 and L2, on many deep learning applications (Nitish Srivastava and Geoffrey Hinton, 2014) [17]. Therefore, we will implement Dropout in our experiment.

Figure9: Left: Standard Neural Net, **Right:** Dropout Neural Net Model

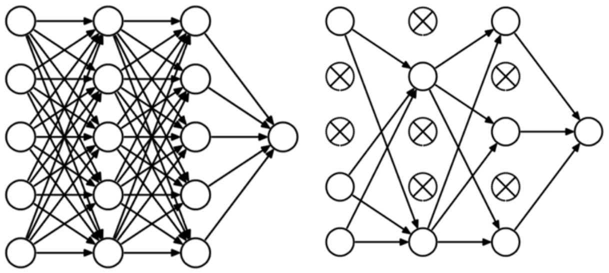


Table14. Accuracy (RMSE) of different methods

Methods	RMSE
LSTM + Dropout 0%	3.630
LSTM + Dropout 20%	3.6129
LSTM + Dropout 40%	**3.6120**
LSTM + Dropout 60%	3.6125

The RMSE of LSTM with 40 percentage dropout value indicates the best result among other dropout values. In addition, LSTM with no dropout technique has the highest RMSE, 3.630, which the system might be overfitting when taking all the units in the large networks. The results show that dropout value at 20% could be a great start, since once could see a decreasing of error from implementing no dropout to dropout. However, when LSTM drops out the most percent of the nodes in either the input or hidden layers, which was 60%, the networks might be under learning.

Optimization Algorithm. We use Adam as our optimizer with 0.001 learning rate to train each model, since it has lower training cost than SGD.

Loss Function. We select RMSE and MAE as our loss functions. RMASE score put more weight on large error values, while MAE just averaged the error values and made them all positive. In the following formulas, y_t is the actual value, $\widehat{y_t}$ is the predictive value at time t, and n is the number of period.

$$Root\ Mean\ Squared\ Error\ (RMSE) = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \widehat{y_t})^2}$$

$$Mean\ Absolute\ Error\ (MAE)\ =\ \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$$

4.2.2 Predictive Accuracy (Test) Results

Model setting. For each model, we have lengths of 9 for both input and output sequences, where we have 9 variables in our dataset. Each model with two layers is trained for 1000 epochs. Batch size is 1and the number of neurons are 3. The hyperparmeters are defined in section 4.2.2 under LSTM model.

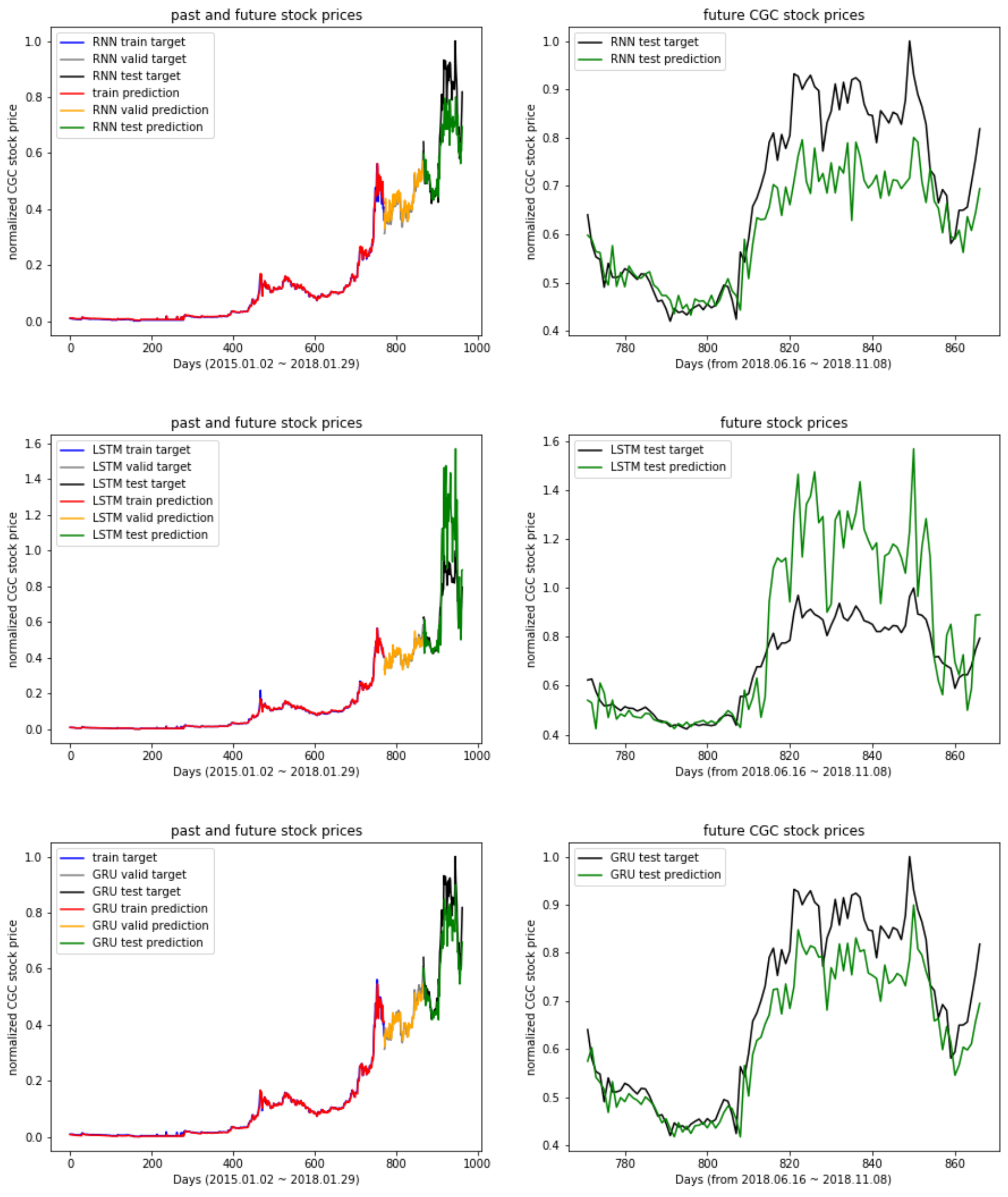
Table15. Accuracy performance of each model on the training, validation and test dataset

	Training dataset		Validation dataset		Test dataset	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
RNN	0.005	0.0037	0.017	0.013	0.1252	0.0961
LSTM	0.005	0.0024	0.017	0.013	0.3303	0.2317
GRU	**0.0047**	**0.0021**	**0.0157**	**0.0121**	**0.0814**	**0.0667**

In these three datasets, we can see the test dataset has larger errors than train set and validation set. It means that all three models possess low bias but slightly higher variance. Among all the models, GRU performs the best with the lowest RMSE and MAE on average in the train, validation, and test set respectively on stock prices dataset. However, surprisingly, LSTM shows the worst accuracy result.

We then make a hypothesis. In this specific scenario of stock movement prediction with only 9 features in the data of three-year period, a simpler Gate RNNs, GRU, outperforms both LSTM and RNN. Firstly, in order to prove that GRU outperforms LSTM, we can see LSTM has 0.2489 less of RMSE and 0.165 less of MAE compared with GRU in the test results. The complex memory units in every layer that LSTM possesses may make it less effectively to remember the important information over many time steps than GRU. In other words, the complexity of gate recurrent architecture is not needed for time series data type. It may be more necessary for more complex data formats, such as text, audio, or video data types, where LSTM has achieved outstanding performance in many research papers. Secondly, in our test result, we prove that GRU outperforms RNN. We indicate that RNN's simple hidden layer make itself hard to learn the long-term dependencies in the long sequential data. It might be more suitable for processing the data that does not require to remember the past information that is far away from recent information. Furthermore, we should also investigate more optimal hyperparameters to get more reliable performances.

Plot 5. 1000 epochs with 1 batch size and 3 neuron – **Left.** Display the actual data (black) and the prediction for training dataset (red), prediction for validating dataset (yellow), and prediction for testing dataset (green) **Right.** Display the actual data (black) and specifically the prediction for testing dataset (green) from RNN, LSTM, and GRU model respectively.



5 Results

5.1 Trading Strategy

In the stock market, there are increasing sectors that are in different scales influencing every day's stock price movement, especially for the high-risk cannabis industry. Therefore, with the potential among of active investors that are interested in this new business field and the computation advantage of machine learning, we presented a daily trading strategy to the daily traders that has large trading volume, which aim at predicting the difference between today and tomorrow's close price and maximizing the accumulative returns in the future 96 days.

Daily trading strategy assumption. Firstly, we assume the number of stocks that an investor hold is finite and constant depended on his/her investment in cannabis stock market. Secondly, an investor only makes trading actions (discussed in next paragraph) at the close price when receiving a buy signal at time t. Thirdly, in order to maximize daily return in control, if an investor holds stocks at time t, then he/she will automatically sell all stocks at time t+1 around closing hour before he makes another action at time t+1.

In addition, our models aren't designed to predict the return between the highest and the lowest price. It is hard to figure out when the price will reach the highest/lowest point in one day, even though we can maximize the profit by buying the stock at lowest price in day 1 and sell it at the highest price in day 2. Therefore, we set the everyday close price as our predicting price.

Daily trading action design. Here we define our trading action that was introduced in the second assumption. In RNN, LSTM and GRU models, we have obtained the optimal result of $\hat{y}_t$ in chapter 4. Therefore, when the close price $\hat{y}_{t+1}$ is greater than $\hat{y}_t$, we buy the stock at time t at close price and sell the stock at time t+1 at close price. Otherwise, we should not buy stock at time t. We didn't set sell signal because we assumed a person who hold stocks at time t will automatically sell all stocks at time t+1. The action signal in each time period are represented by notation A_t.

$$recurrent\ network\ models\colon\ A_t = \begin{cases} 1, & (\hat{y}_{t+1} - \hat{y}_t) > 0 \quad (buy\ signal) \\ 0, & otherwise(shouldn't\ buy\ signal) \end{cases}$$

In HMM model, according to the price movement, we have defined an action that should be taken in each states: We still use A_t to represent action signal:

$$model\ 1\colon\ A_t = \begin{cases} 1, & state3, state4, state5 \quad (buy\ signal) \\ 0, & state1, stae2(shouldn't\ buy\ signal) \end{cases}$$

$$model\ 2\colon\ A_t = \begin{cases} 1, & state4, state5, state6, \text{state3} \quad (buy\ signal) \\ 0, & state1, stae2(shouldn't\ buy\ signal) \end{cases}$$

$$model\ 3\colon\ A_t = \begin{cases} 1, & state3, state4, state5 \quad (buy\ signal) \\ 0, & stae2, stae5(shouldn't\ buy\ signal) \end{cases}$$

After we know which action we should take in each period, we can calculate how much profit we will make after n period. $A_t(y_{t+1} - y_t)$ is amount of profit you will make in one period. Summing all these daily profit, we will get the cumulative profit.

$$Profit = \sum_{t=1}^{n} A_i(y_{t+1} - y_t) * Stock \qquad \text{(Stock is amount of stock you hold)}$$

Table16. Accumulate profit procedure (as example)

Close Price	43.43	44.46	41.08	36.93	40.51	38.42	38.40	38.65	39.70
A_t	1	0	0	1	0	0	1	1	…
Cumulative profit (for one stock)	…	44.46-43.43= 1.03	1.03+0= 1.03	1.03+0= 1.03	1.03+(40.51-36.93) = 4.61	4.61+0= 4.61	4.61+0= 4.61	4.61+(38.65-38.40) = 4.86	4.86+(39.70-38.65) = 5.91

5.2 Profitability Test Results

We evaluate our profitability test results by the Mean correct forecast directions (MCFD) with 1(·). It is an economic metric that measures the accuracy of the correct sign between the actual close price and the predictive close price on average. When the signs are equal, the MCFD will be assigned as 1. In other words, both the actual close price and the forecast close price are in the same movement. Therefore, the larger the value of the MCFD is, the more accurate the stock movement prediction is.

$$Mean\ correct\ forecast\ directions\ (MCFD)\ =\ -\frac{1}{n}\sum_{t=1}^{n} 1(sign(y_t)\cdot sign(\widehat{y_t})) > 0$$

Table 17. Profitability accuracy (MCFD) of each model in CGC stock market from 2018.06.16 to 2018.01.29.

	MCFD
RNN	75%
LSTM	71%
GRU	**77%**

As we can see from table17, the MCFD of GRU has the highest percentage among RNN and LSTM. This result aligns with the previous accuracy performance (RMSE and MAE), which GRU performed the best in our model setting. Overall, GRU can better predict when the stock price will increase or decrease.

Table 18. Cumulative profit of each model in CGC stock market from 2018.06.16 to 2018.01.29.

	Cumulative Profit (per stock)
RNN	$ 27.64
LSTM	$27.12
GRU	**$28.33**
HMM MODEL1	$1.61
HMM MODEL 2	$12.57
HMM MODEL 3	$5.75

From the table 18, it shows that all the models have earned positive cumulative profit indicating that all the approaches are feasible. However, the amount of the profit they made are in wide range from $1.61 to $28.33. Looking at the HMMs, the HMM with 4 observation sequences and 5 states has the best performance among all the HMMs, which can better capture the truly market trend than other two models. On the other side, GRU performed the best in the Recurrent Neural Network family. In addition, we found that Recurrent Neural Networks outperformed all the HMM models.

There are many factors drive stock price up and down, whereas we only used few visible factors to predict it. As a result, the cumulative profit which is produced by HMM is not as good as RNN, LSTM and GRU do. Compare with other three models, another big problem of HMM is that its independent assumption is too strong, therefore it lack connection between historical information when it deals with time series observation. Without considering historical information, the HMM may result a poorer performance.

6 Conclusion

In our research, we studied HMM and three different recurrent neural networks architectures, which were RNN, LSTM, and GRU, for time series sequence learning. We focused on the scenario of the stock market. We predicted whether cannabis stock will go up or go down in a period of daily regular trading hours.

For HMM, we introduced its settings and assumptions. We computed the lowest values of AIC and BIC to select the optimal numbers of state for each model. We then used the Baum-Welch algorithm to optimize the model parameters. We presented Viterbi algorithm to find the state sequences with the highest probability. We also discussed Forward algorithm, which can solve the model evaluation problem by computing the probability that an observation sequence was produced by model. After we developed 4 Hidden Markov Models, we compared their performance by test data set. We found that the model with 4 observation sequences and 5 states can find the true hidden state most efficiently.

For the recurrent network architectures, we illustrated their internal structures along with the computational procedures. We presented their properties, disadvantages and advantages when applying to time series data that is dependent. In the RNN, we discussed that the gradient can be computed by backpropagation through time and can find the close-form relationship between network weights and the loss function by gradient decent optimization. Stochastic Gradient Descent (SGD) and Adam can be the optimizer to update the parameters of the networks. We then addressed the problems of exploring and vanishing gradients when the depth of RNN is large. The exploring gradients can be solved by gradient clipping, while the vanishing gradients can be solved by the gated architecture in recurrent neural networks, which are LSTM and GRU.

We presented an accuracy performance and profit performance considering the stock movement prediction problem. We tuned the hyperparameters of LSTM. The optimal configuration we found were 1000 epochs, 1 batch and 3 neuron. We discussed the effect of dropout and discovered that when dropout was at 40%, it can effectively reduce the overfitting during the training time. We use Adam optimizer and loss function of RMSE and MAE. We compared the accuracy performance of LSTM with RNN and GRU and made a hypothesis. In our scenario, a simpler Gate RNNs, GRU, outperformed both LSTM and RNN.

In the last part, we made the assumptions and constructed our daily trading strategy that aim at maximum the daily return of the close price in the next 96 days. We designed our trading action as buying low at time *t* and selling high at time *t+1*. Eventually, we evaluated our profitability test results by the Mean correct forecast directions (MCFD). The experimental result showed that all the models are feasible to earn certain amount of profits. Overall, we found that Recurrent Neural Networks family outperformed HMM models. We can make the assumption that GRU is the model that can earn more profits in the cannabis stock market among all the other models that were introduced in the experiment.

In conclusion, we found that although HMM has advantage on the efficient algorithms to generate new observations robustly, its assumptions have limited HMM's capacity of capturing the true hidden states in the dynamics. Moreover, the simple recurrent design of RNN may be efficient at processing the short-term sequential data. The RNN architecture needs to introduce more efficient gate design when applying to the long-term data. Furthermore, for more accuracy performance, we can further introduce other classical time series models, such as ARIMA or GARCH, to combine with the gated RNNs to see if the statistical models would be helpful to better represent the dynamic of stock market.

7 Reference

[1] Fama, Eugene F. (September–October 1965). "Random Walks In Stock Market Prices". Financial Analysts Journal. 21 (5): 55–59. doi:10.2469/faj.v21.n5.55. Retrieved 2008-03-21.

[2] Baum, Leonard E., and Ted Petrie. 1966. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. The Annals of Mathematical Statistics 37: 1554–63.

[3] Baum, Leonard E., and John Alonzo Eagon. 1967. An inequality with applications to statistical estiation for probabilistic functions of Markov process and to a model for ecnogy. Bulletin of the American Mathematical Society 73: 360–63.

[4] Felix A. Gers; Jürgen Schmidhuber; Fred Cummins (2000). "Learning to Forget: Continual Prediction with LSTM". Neural Computation. 12 (10): 2451–2471. doi:10.1162/089976600300015015.

[5] Doetsch P, Kozielski M, and Ney H. Fast and robust training of recurrent neural networks for offline handwriting recognition. In: 14th International conference on frontiers in hand- writing recognition (ICFHR), 2014, pp. 279–284. IEEE.

[6] Zaremba W, Sutskever I, and Vinyals O. Recurrent neural network regularization. arXiv preprint arXiv:1409.2329, 2014.

[7] Sak H, Senior A, and Beaufays F. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv preprint arXiv:1402.1128, 2014.

[8] Marchi E, Ferroni G, Eyben F, et al. Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks. In: IEEE international conference on acoustics, speech and signal processing (ICASSP), 2014, pp. 2164–2168. IEEE.

[9] Donahue J, Hendricks LA, Guadarrama S, et al. Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 2625–2634. IEEE.

[10] L. R. Rabiner and B. H. Juang Rabiner (1986). An introduction to hidden Markov models, IEEE ASSp Magazine

[11]Robert Mattila, Vikram Krishnamurthy, Bo Wahlberg, "Recursive identification of chain dynamics in Hidden Markov Models using Non-Negative Matrix Factorization", Decision and Control (CDC) 2015 IEEE 54th Annual Conference on, pp. 4011-4016, 2015.

[12] Bianchi, Filippo Maria & Maiorino, Enrico & Kampffmeyer, Michael & Rizzi, Antonello & Jenssen, Robert. (2017). Recurrent Neural Network Architectures. 10.1007/978-3-319-70338-1_3.

[13] Li, Xiaolin, Marc Parizeau, and Réjean Plamondon. 2000. Training Hidden Markov Models with Multiple Observations—A Combinatorial Method. IEEE Transactions on PAMI 22: 371–77.

[14] Tuyen, Luc & Nguyen Van, Hung. (2012). A Normal - Hidden Markov Model in forecasting stock prices. Journal of Computer Science and Cybernetics 1813-9663. 28. 206-216.

[15] Kavitha, G & Udhayakumar ,A & Nagarajan ,D. Stock Market Trend Analysis Using Hidden Markov Models

[16] Nguyet, Nguyen. An Analysis and Implementation of the Hidden Markov Model to Technology Stock Prediction

[17] Nitish Srivastava & Geoffrey Hinton &Alex Krizhevsky & Ilya Sutskever & Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research. JMLR:v15:srivastava14a, 15:1929-1958

[18] Chai S, Guo C, editors. The Co-integrating Relationship between Stock Index and Futures Prices. Inter- national Conference on New Trends in Information and Service Science; 2009.

[19] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 .