SOCIETY OF ACTUARIES®

Innovation and Technology

A Tour of AI Technologies in Time Series Prediction

July 2019

A Tour of AI Technologies in Time Series Prediction

AUTHOR Victoria Zhang, FSA, ACIA
Research Actuary
Society of Actuaries

CONTENTS

Acknowledgments

The author would like to thank the Society of Actuaries (SOA) Actuarial Innovation and Technology Steering Committee for sponsoring this paper. The author would also like to thank all the members of the Project Oversight Group (POG) for providing governance and advisory oversight on this project. In addition, the author gratefully acknowledges the assistance of Dale Hall, Mervyn Kopinsky, and Korrel Crawford for their assistance with various aspects of this project.

Project Oversight Group Members:

Lai Shan (Erica) Chan, FSA

Christopher James Coulter, FSA, MAAA

Lemeng (Sophie) Feng, ASA, ACIA

Carl Ghiselli, FSA, MAAA

Timothy J. Luedtke, FSA, MAAA

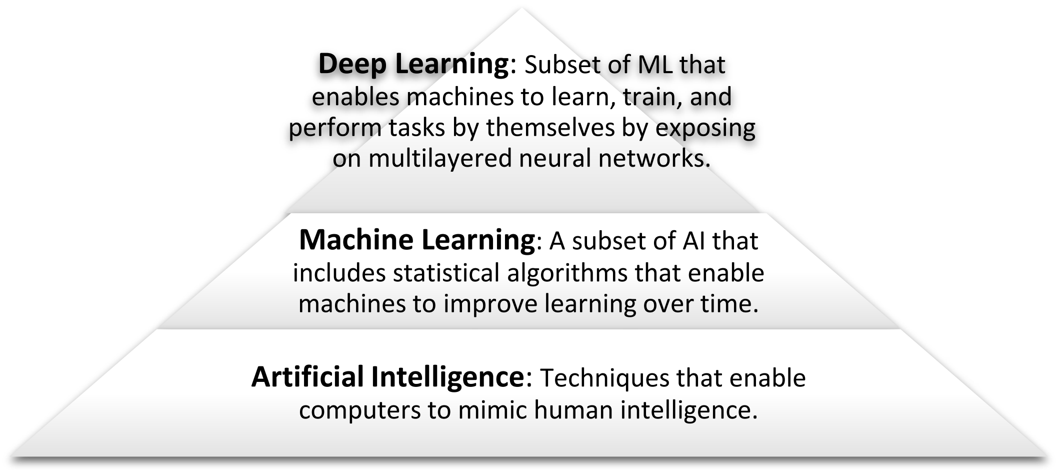
Michael Cletus Niemerg, FSA, MAAA

Executive Summary

Over the past few years, Artificial Intelligence (AI) technologies such as Machine Learning (ML) and Deep Neural Networks (DNN) or Deep Learning (DL) have become a very hot topic in many areas. The emerging field of DNNs was created around the concept of biological neural networks and has been widely applied in many fields. AI technologies have been used for computer vision, speech recognition, autonomous driving, etc. and have demonstrated remarkable results. McKinsey predicts that AI techniques have the potential to create between \$3.5T and \$5.8T in value annually across nine business functions in 19 industries[1]. However, even with the buzzwords around for a few years, AI is still very new to the actuarial field.

AI is the simulation of the human intelligence process by machines. This associated intelligence is demonstrated in planning, learning, reasoning, and problem-solving. Machine learning (ML) is a significant portion of AI, where a computer system is trained with a large amount of data to learn how to carry out a specific task. It is the science and engineering of getting computers or robots to learn and act like humans do and improve their learning over time in an autonomous fashion. This involves techniques from statistics, computer science, and mathematics. DL can be viewed as a more advanced version of ML, with higher dimensions of representability to solve complicated problems that were difficult to tackle with traditional ML models. In this report, we will go through some main categories of ML and DL models and explain the principles of them and how to use those models to address the real-world time series problems.

Figure 1
STRUCTURE OF AI TECHNOLOGIES


Deep Learning: Subset of ML that enables machines to learn, train, and perform tasks by themselves by exposing on multilayered neural networks.
Machine Learning: A subset of AI that includes statistical algorithms that enable machines to improve learning over time.
Artificial Intelligence: Techniques that enable computers to mimic human intelligence.

[1] Columbus, Louis. "Sizing The Market Value of Artificial Intelligence." Forbes, Forbes Magazine, 30 Apr. 2018 https://www.forbes.com/sites/louiscolumbus/2018/04/30/sizing-the-market-value-of-artificial-intelligence/#5f8806c2ffe9

To actuaries, there are numerous advantages to using AI models:

1. Strong representability. They have the ability to handle a tremendous amount of data, which could be in extremely high dimensions. Multiple layers of neural networks allow the model to detect relationships in higher dimensions; these relationships may not be visible in traditional modeling.

2. Better accuracy. Compared to traditional predictive modeling, AI models have far better predicting power thanks to the recent advancements in the DNNs. The DNN models have even outperformed human beings in areas such as image recognition and complicated board games such as Go.

3. Fast adaptability. AI models are quite dynamic, and they can evolve themselves as more data is fed in. What makes an AI model useful is that the algorithm can "learn" and adapt its output based on the new information. The model could update itself based on the error between its prediction and the new data and will improve its future prediction.

In this report, we will try to provide an in-depth review of current ML and DL models and will explain how those models work and their possible applications in the actuarial field. Recent studies and successes from the industry will also be discussed. We intend to bring more actuaries on board regarding those technologies and start to think about how to bring them into our daily work.

The second purpose of this report is to demonstrate how to use AI technology for ***time series prediction***. Time series prediction plays a vital role for insurance companies. From assumption setting in pricing, valuation, and asset liability management strategies, small improvements in time series predictions can result in significant financial impact. Currently, most insurance companies use predictive modeling techniques for time series predictions, but the performance is not very good and is not dynamic enough for environment changes. After the introduction of different ML and DNN models, we will focus on models that can be used to solve time series problems, including:

- Machine Learning models for time series classification,

- Deep Learning models for time series prediction,

- Recurrent Neural Network models for time series prediction, as well as anomaly detection

The models are uploaded to the GitHub site: https://github.com/rranxxi/soa_research_ai_time_series, so readers can download the program and utilize it for different purposes.

The report will be concluded with a discussion of the advantages and challenges actuaries are facing with AI technology. It intends to provide insights into this emerging field and how actuaries would adjust to this thriving technology.

Chapter 1: Introduction to Machine Learning and Potential Applications in Time Series Prediction

At a high level, there are three types of machine learning models: supervised learning, unsupervised learning, and reinforcement learning. In this chapter, we will go through the basic algorithms of supervised learning models and discuss their potential actuarial applications. At the end of this chapter, we will showcase how to use these models to solve time series classification problems.

1.1 SUPERVISED LEARNING MODELS

Supervised learning applies where there is input training data X and output ground truth label Y, and the algorithm will learn the mapping function $Y = f(X)$. It is called supervised learning because the training process can be thought of as a teacher supervising the learning process. The correct answers are known, and the algorithm iteratively makes predictions on the training data and is corrected by the teacher. The learning process stops when the prediction error is smaller than a predefined threshold.

There are two major categories of supervised learning algorithms: classification and regression.

Classification is the task of approximating a mapping function from input data to discrete output labels. We use classification algorithms widely in our daily lives, such as spam email classification, credit card fraud detection, customer segmentation, and disease detection and classification.

Regression is another significant part of a supervised learning technique. Instead of predicting discrete labels as in classification, regression focuses on predicting a continuous quantity. Therefore, some classification algorithms could also be adapted for regression tasks. Generally speaking, regression can be linear or non-linear. Regression could help us, for example, predict the financial markets or find correlations among different indexes or stocks.

1.1.1 INTRODUCTION OF SUPERVISED LEARNING ALGORITHM

To help us understand the supervised learning algorithms, we introduce the six most popular supervised learning-based algorithms. We will be going through them, discuss how they work, and compare their performance on the same artificially created datasets to visualize the different impact on the same data.

Naïve Bayes Classifier

Naïve Bayes Classifier is a simple, yet very effective classifier. It is based on applying Bayes' Theorem with strong independent assumptions between different features. Our human brains use Naive Bayes classifier all the time. For instance, if I tell you I have a fruit that is sweet, red, and round, what would you think it is more likely to be: an apple or a banana? We all agree it's likely an apple as we have not yet seen a red banana. In this example, we use our common sense to come up with an answer. Naïve Bayes Classifier solves this kind of problem with probability.

Recall Bayes Theorem:

$$\boldsymbol{P}\,(\mathbf{A}|\mathbf{B}) = \frac{\boldsymbol{P}(\boldsymbol{B}|\boldsymbol{A})}{\boldsymbol{P}(\boldsymbol{B})} * \boldsymbol{P}(\boldsymbol{A}) \qquad Posterior = \frac{Prior * likelihood}{evidence}$$

The Naïve Bayes Classifier calculates the probability of each label given known features and selects the outcome with the highest probability. This is called maximum a posterior (MAP) hypothesis. In the problem mentioned above, we could calculate the probability of being an apple as

$$\Pr(apple \mid red, sweet, round) = \frac{\Pr(red, sweet, round \mid apple) * \Pr(apple)}{\Pr(red, sweet, round)}$$

Where $\Pr(apple \mid red, sweet, round)$ is proportional to $\Pr(red, sweet, round \mid apple)$.

Without any prior information, the probability of picking an apple or being red, sweet, and round is constant; then we could drop the term $\Pr(apple)$ and $\Pr(red, sweet, round)$. To determine which fruit it is likely to be, we are essentially comparing $\Pr(red, sweet, round \mid apple)$ and $\Pr(red, sweet, round \mid banana)$. We could use the Naïve Bayes Classifier for problems such as text classification, spam detection, and sentiment analysis on social media.

K-Nearest Neighbor (KNN)

"Birds of a feather flock together." KNN is a non-parametric algorithm, which does not make any assumption on the underlying data distribution. This algorithm is built around the observation that similar things are near to each other, so could use the values from its neighbors to regress or classify an input X.

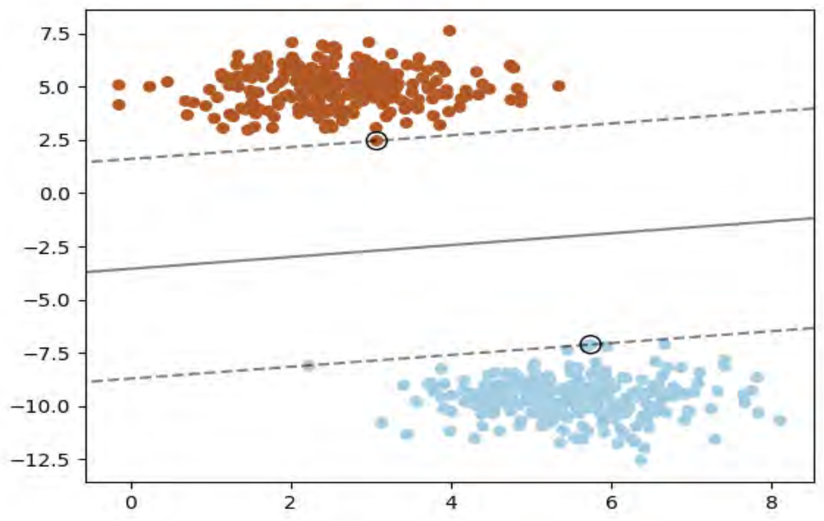
When using KNN for classification jobs, we will map X to a label based on the majority of its K-nearest neighbors. As for the regression case, a KNN regression model, instead of using the label of the majority of the k-nearest neighbor, will use the weighted average value of those neighbors as its prediction results. To a large extent, using a relatively small amount of neighbors k suppresses the effects of noise, but also makes the classification boundaries less distinct and the prediction results might be more prone to error.

The KNN algorithm has demonstrated its power in many applications, including credit rating and voting predictions. The KNN has the benefit of simplicity during the training and effectiveness in prediction. However, we should be mindful about its cons as well: it is very computationally expensive and highly memory demanding during the prediction, and it is quite sensitive to outliers within the dataset.

Support Vector Machine (SVM)

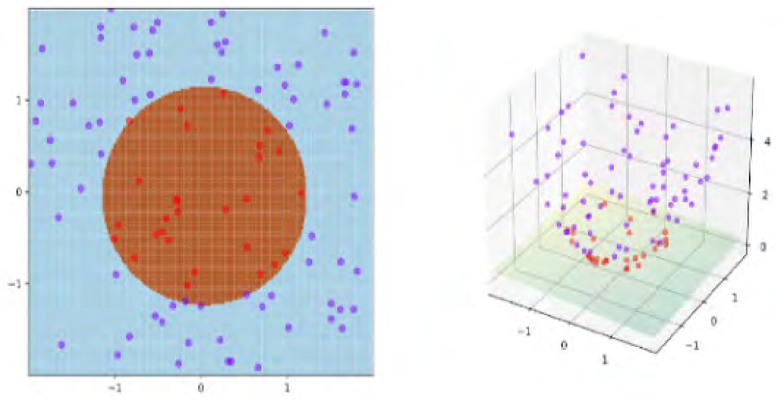
The best way to explain SVM is by an example: suppose we have some samples from two classes: blue and brown. SVM is to find an optimal decision boundary which will maximize the distance from this decision boundary to the nearest point of each class. The benefit of choosing such a decision boundary is quite apparent, since the larger this distance is, the more robust our model will be when applying it to the unseen data.

Figure 2
SVM CLASSIFICATION IN 2-D SPACE



The above example does not demonstrate the true power of SVM as our blue and red samples are clearly linearly separable, and it is not particularly challenging to find such decision boundaries. What if our samples for two classes look like this?

Figure 3
SVM CLASSIFICATION IN 3-D SPACE



The SVM model would still be able to find an optimal decision boundary by using a technique called kernel trick. What it does is lift the data from its original lower dimension space into a higher dimensional space where the samples are linearly separable and will compute the decision boundaries in this higher dimension space.

In the above example, the original data was in a cartesian space $[\boldsymbol{x_1}, \boldsymbol{x_2}]$. If we transform the data into a polar space $[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_1^2}, \boldsymbol{x_2^2}]$, then the samples can be nicely separated.

Decision Trees
The Decision Tree algorithm is the systemic classification approach of conducting a series of test questions to build an optimal decision tree structure. Starting from the root of the tree (raw data), the tree branches by the decision nodes (features or attributes chosen for splitting) and eventually lead to the leaf nodes, which could be the classification labels or some continuous values for regression models.

One of the widely used methods to build a decision tree classifier is to use the concept of **entropy** and **information gain** to determine which features or attributes should be used for branching, and when the splitting should stop. Entropy is a way of measuring the magnitude of information, and it will control how and when to split from a parent node. Entropy is calculated as the negative summation of probability times the log of the probability of each item, which can be computed mathematically as below:

$$\mathbf{H} = -\sum_{\boldsymbol{x}} \boldsymbol{P(x)}\,\boldsymbol{log}(\boldsymbol{P(x)}),\text{ where } P(x) \text{ is the probability distribution of a variable } x.$$

If all samples belong to the same class, then $\boldsymbol{P(x) = 1}$, so we will have the entropy 0. Thus, there is no information in the data. On the other side, if all items are evenly distributed between classes, then we will have the max entropy, which is 1.

$$\begin{aligned} &Information\ Gain\ (n) \\ &\quad = Entropy(parent) \\ &\quad - \big([weighted\ average] * \ entropy\ (children\ for\ feature)\big) \end{aligned}$$

When a parent node is divided into sub-nodes, the sub-nodes are called "children." The goal of a decision tree is to maximize the information gained on each tree node. When the information gain is 0, that means the feature does not increase our information, so it is not worth using to split the tree. It is worth mentioning that there are a few other metrics that can be used to train the data, like variance reduction or Gini impurity, but the way of splitting is quite similar.

When applying decision trees for regression, instead of entropy, we use a mean square error (MSE) or similar metrics to determine the split. One of the common methods of a regression decision tree is the *classification and regression tree* (CART) method. The method starts by searching for every distinct value of all its predictors and splitting the values of predictors that minimizes the overall sum of square error (SSE).

Assume there are two labels, 1 and 2, in one dataset, which could be split into two exclusive groups S_1 and S_2, then we have $\boldsymbol{SSE} = \sum_{\boldsymbol{i} \in S_1}(\boldsymbol{y}_i - \overline{\boldsymbol{y_1}}) + \sum_{\boldsymbol{i} \in S_2}(\boldsymbol{y}_i - \overline{\boldsymbol{y_2}})$ where $\overline{\boldsymbol{y_1}}$ and $\overline{\boldsymbol{y_2}}$ are the average values of the dependent variables in the group S_1 and S_2.

For group S_1 and S_2, the method will recursively split the prediction values within the group. The method stops when the remaining sample size falls below a certain threshold or hits a predefined depth.

Decision Trees are easy to explain and great for visualizations. They can be applied from business management to engineering. They also have the benefit of being systematic and require little effort for data preparation. However, there is a high probability of overfitting with Decision Trees. Potential bias may also exist if there are many categories. These two significant disadvantages can be mitigated through a method called bagging, which will be discussed in the next random forest algorithm.

Random Forests

Random Forests are an ensemble technique that takes multiple decision trees and merges them with the bagging method. Instead of relying on one individual decision tree, a random forest combines the prediction results from multiple decision trees to get more robust and accurate results.

The bagging method involves training each decision tree with a random set of training data points; furthermore, at each split, a subset of features is selected. This increases the diversity of the forest, which leads to the name "random forest." The final prediction result is the average of all the individual decision tree estimations.

Compared to the decision tree approach; there are many advantages of the random forest:

- Reduce overfitting and improve prediction accuracy: one of the major drawbacks of decision trees is that they are prone to overfitting. As the tree splits deeper and deeper, the model becomes too catered towards the training dataset and performs poorly when deployed for new data. Random forests reduce overfitting by considering a subset of data for each tree node and generalize the results of the forest.

- Lower variance, less bias: because we are averaging all the trees in the random forest, we are averaging the variance of decision trees as well so that the variance of the random forest is lower.

- Robust to outliers: since each decision tree in the random forest selects a subset of data, the number of outliers in each tree will be much smaller, and so is their influence. Thus, a random forest is more immune to the outliers.

- Fast in training and prediction: the random forest algorithm is highly parallelizable, which means the computation can be split and run in parallel at the same time and could be ideal for running on Graphics Processing Units (GPU). In addition, the fact that each decision tree is only working on a subset of features will speed up the calculation as well.

Gradient Boosted Trees (GBT)

GBT is another ensemble learning method that fuses output from multiple decision trees. It is different from a random forest as GBT builds one tree at a time, and each new tree is meant to correct errors made by the previous tree.

There is a total loss function defined to track the difference between prediction and ground-true results. In addition to this total loss function, there is usually a regularization term as well to avoid overfitting. When handling an imbalanced dataset where there is way more samples of some labels compared to others, there is a high chance of bias if we try to fit the data with a random forest. The model will predict the majority of the data and still achieve a high accuracy score. Taking the above apple and banana example again - assume there are 100 apples and 1 banana in a black box. If you are asked to guess the type of one fruit randomly picked from this black box without watching or touching it, you could have extremely good accuracy if you always guess it is an apple. On the other hand, gradient boosting is a sequential process, and it focuses on the error every time there is an incorrect prediction. Therefore, gradient boosting fits better for imbalanced data than random forests.

In real life, gradient boosting is ideal for applications such as anomaly detection, credit card transaction monitoring, insurance claim fraud, and cybersecurity. All of these data are highly imbalanced as the chances of an anomaly are very low. However, that is exactly where we want to focus on when modeling the data.

1.1.2 WHICH MODEL SHOULD I USE?

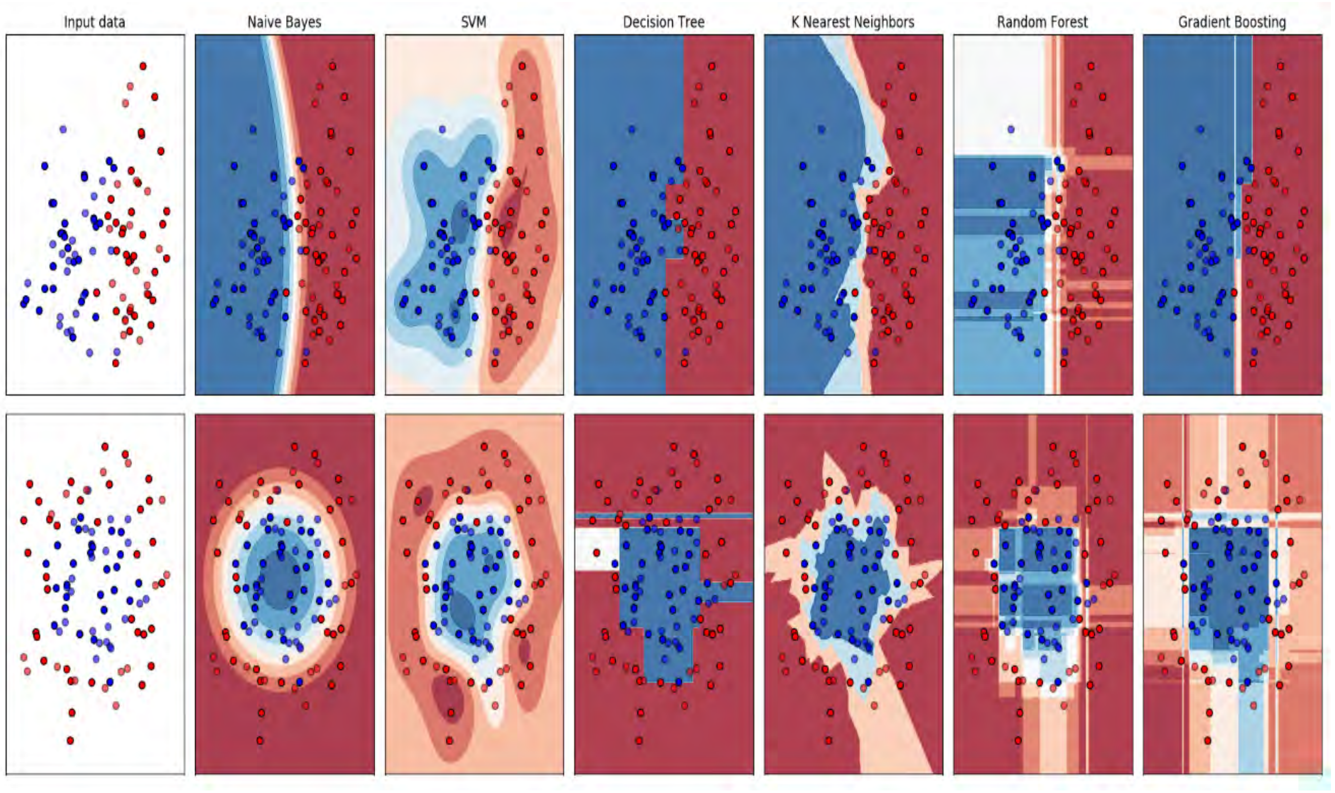
Depending on the characteristics of the input training data, different classifiers might create significantly different shapes of decision boundaries. In the following images, we artificially created two datasets by the scikit-learn framework and applied all the classifiers we have discussed to the two datasets, and then visualized their decision boundaries. In each of the datasets, there are two categories of samples, which are marked as red or blue in color in the leftmost image.

In the first dataset, the red and blue samples are linearly separable with some noise added intentionally. As we can see from the top part of the images, almost all the classifiers perform reasonably well for this kind of data sample.

Things get more interesting for the second dataset where the data can only be separated by a circular-shaped decision boundary. The simple classifier, such as Naive Bayes, incurs quite a few mistakes, whereas the more sophisticated models such as SVM, Decision Tree, Random Forest, and Gradient Boosted Tree all work quite well. From these two examples, we find that the k-nearest neighbors' model is especially interesting. Although it is a simple model, its accuracy is always remarkably good compared to some more advanced models.

A natural question we might ask is what the guideline is to pick a model. Unfortunately, there is no silver bullet here. It is a trial and error process; we usually need to try different models and manually compare their results to know which model is the best fit for our data.

Figure 4
CLASSIFICATION RESULTS BY SIX (6) DIFFERENT ML ALGORITHMS



1.1.3 ACTUARIAL APPLICATIONS OF SUPERVISED LEARNING

Classification

Underwriting is the heart of insurance. The ability to evaluate and accept the risk sets the foundation of the business. Insurance companies still rely on large underwriting teams to process new policies; there are lots of manual touch points and redundancies in the process. Machine learning classification has the advantage of being robust, accurate, and fast in response. The technique could be leveraged in the insurance underwriting process. Supervised classification models could replace the traditional underwriting process easily. Internal data could be used as training data, and the process of underwriting could be reduced to a few seconds. Classification algorithms such as Boosted Trees or Deep Neural Networks (DNN) could be trained on internal historical data and predict risk class labels.

Together with other AI techniques, we could assess more data. Third-party data, such as medical records, financial data from banks, social media, etc., could all be leveraged to build a sophisticated underwriting model. Compared to traditional underwriting models where all the information is from the customer's questionnaire and medical testing, an AI underwriting model tends to be more comprehensive yet cost-effective.

Regression
The regression models could be deployed in a few areas: it could be used in the actuarial valuation model, assumption setting, experience studies, and projections.

Compared to traditional predictive modeling, a regression model based on neural networks could consist of a huge number of neurons and adapt higher dimension data in the valuation. In traditional predictive modeling, we need to handpick a small piece of information (sex, age, smoking habit) from the input data and build a particular model around the features selected. However, the information available to us could be vast, and we just would not know how to incorporate it into traditional predictive modeling. With deep neural networks, we could make the best use of all the information available to us, and the networks will pinpoint the information relevant to the output of interest.

Time series projection is another area where ML regression plays a game-changing role. Investment-related insurance products all face market risk. The projected fund value of a UL or annuity product could be dramatically different over time and change the reserve, which is why a time series prediction is so critical to these products. In Chapters 2 and 3, we will demonstrate how to build AI models to address time series prediction problems.

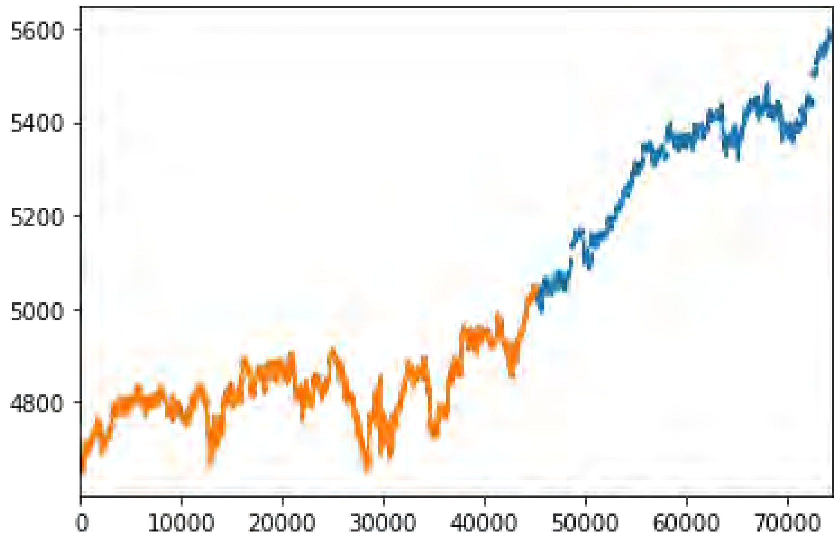
1.2 ML MODEL FOR TIME SERIES CLASSIFICATION

In this section, we will provide a concrete example of how to use different classifiers to model a time series classification problem. The task is to classify whether the NASDAQ index trend is going up or down or will stay stationary based on the last T data points. Stock markets are infamous for being unpredictable. However, this does not stop people from trying. This is another try. We will leverage the classifiers we have covered so far to see whether we can at least predict the thread of it. In Chapter 3, we will try to use recurrent neural networks to predict the price.

1.2.1 DATA PREPARATION

This dataset can be downloaded from http://cseweb.ucsd.edu/~yaq007/NASDAQ100_stock_data.html, which has the NASDAQ 100 index value collected every minute covering the period from July 26, 2016 to April 28, 2017, 191 days in total. Each day contains 390 data points from the opening to the closing of the market. The values of the data are visualized in the below figure.

Figure 5
NASDAQ 100 DATA FROM JULY 26, 2016 TO APRIL 28, 2017



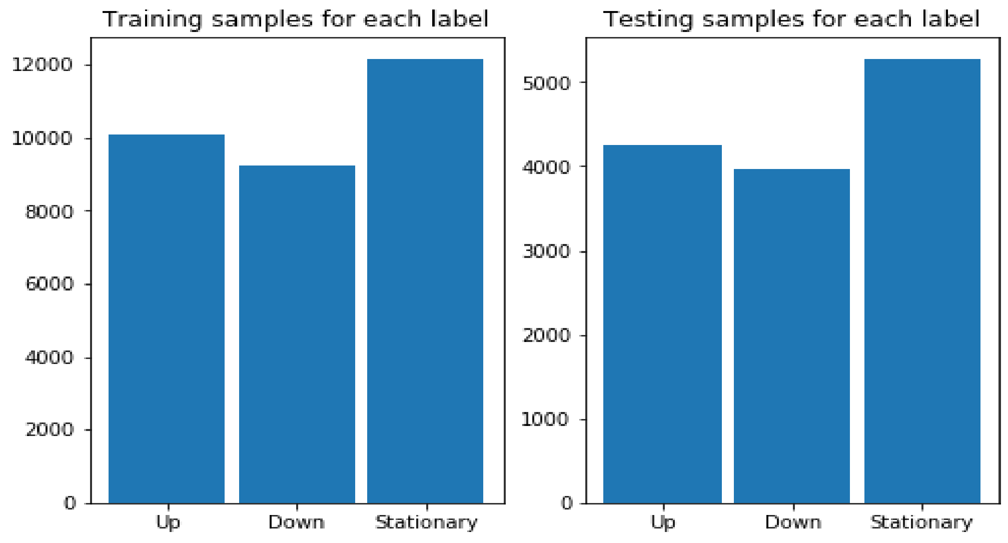
From Figure 5, we can observe that, in the first 45,000 samples, the NASDAQ index goes both up and down, but between 45,000 to the end of the dataset, the trend is mostly up. To make sure we have balanced trends, we only use the first part of the samples, which are plotted with orange color during this experiment.

We will also need to generate the classification labels Y(ground truth) from the input NASDAQ index. We define a threshold ε and a sliding window with length T, and for each time point together with all the preceding data within the sliding window, the current NASDAQ index could be one of three trends: up, down, or stationary (unchanged). Assume $\mathrm{m_t}$ is the average index price within the current sliding window, and $\mathrm{m_{t+T}}$ is the average price of the next sliding window, then label y can be derived based on the following equation:

$$\mathrm{y} = \begin{cases} 1, & \mathrm{m_t} < \mathrm{m_{t+T}} - \boldsymbol{\varepsilon} \\ -1, & \mathrm{m_t} > \mathrm{m_{t+T}} + \boldsymbol{\varepsilon} \\ 0, & \quad\quad\; else\ case \end{cases}$$

To ensure the number of samples for each label is balanced, the ε needs to be carefully selected. During our experiment, we set ε = 0.0055. We apply the conversion method throughout the selected samples X to generate the training labels Y. We also split the total X and the generated Y into two parts, the first 70% will be our training data, and the remaining 30% will be for testing and will be reserved to evaluate the classifier's accuracy after the training is complete. The graph below provides a visual presentation of the distribution of the three labels within both the training and testing data. We see that the up and down trends are almost evenly distributed and the stationary label has slightly more than the other two labels. We could further tweak the above threshold ε to have even more balanced labels; however, from our later experiments, this setup is already enough to show that the ML classifiers could effectively classify the NASDAQ trend.

Figure 6
TRAINING AND TESTING SAMPLE DISTRIBUTION

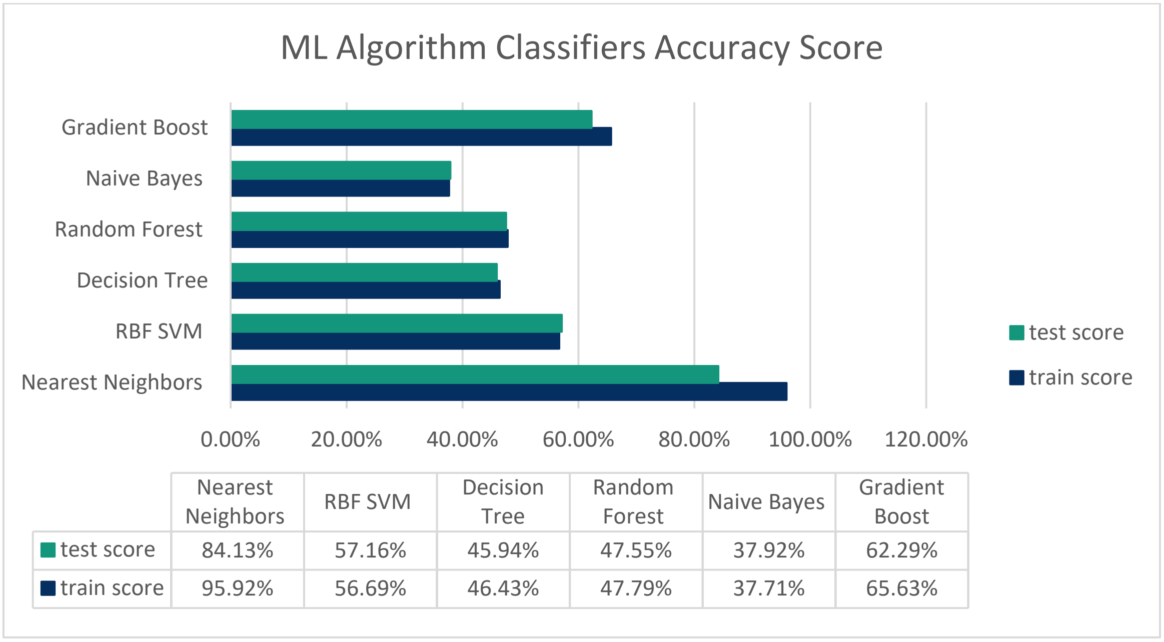


1.2.2 CLASSIFICATION AND RESULTS ANALYSIS

The classification was done by all six classification algorithms discussed earlier: K-Nearest Neighbors, SVM, Decision Tree, Random Forests, Naïve Bayes, and Gradient Boost. After training all of the six classifier models with the same data, we applied the trained models to the reserved 30% testing data, which had never been seen by those models to evaluate the classification accuracy. The results have been summarized in the below Figure 7. The ending result is showing that among the six algorithms, three appear to be effective as they achieved an accuracy score higher than 50%. K-Nearest Neighbors has the highest accuracy for about 84% of the testing data, followed by Gradient Boosted Trees and SVM. As we discussed earlier, the algorithm of K-Nearest Neighbors is simple, but it is very compelling in predicting the future trend. Gradient Boosting performed better than Decision Trees and Random Forests. However, all six algorithms seem to avoid overfitting issues as the accuracy scores on the testing dataset are relatively close to the ones from the training data. If the model falls into the overfitting trap, we would expect to see a high accuracy score in the training dataset but low accuracy score for testing data. The models can be downloaded from GitHub site at:

https://github.com/rranxxi/soa_research_ai_time_series/tree/master/trend_classfication.

Figure 7
CLASSIFICATION ACCURACY SCORE BY SIX (6) ML CLASSIFIERS



	Nearest Neighbors	RBF SVM	Decision Tree	Random Forest	Naive Bayes	Gradient Boost
🟩 test score	84.13%	57.16%	45.94%	47.55%	37.92%	62.29%
■ train score	95.92%	56.69%	46.43%	47.79%	37.71%	65.63%

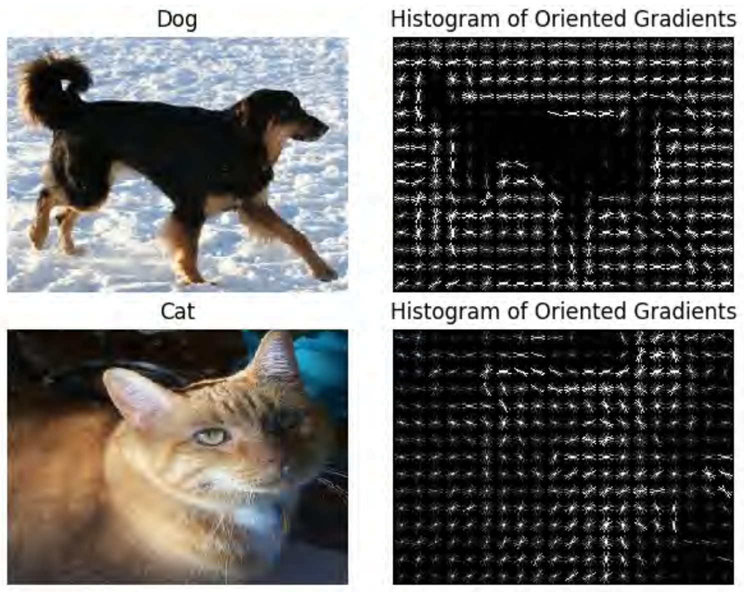
The power of predicting the future trend of time series data opens enormous potentials and opportunities for actuaries. Actuaries need to make decisions based on outlooks; these outlooks are usually based on experience, actuarial judgments, and sometimes a little extra conservatism. With an ML model, these outlooks are no longer blurry to us; it sets more confidence in the future outlooks.

Chapter 2: Introduction to Deep Learning and Applications in Time Series Prediction

In this chapter, we are going to explore a little bit about deep learning. Deep learning is an advanced form of machine learning. One of the most significant differences between deep learning and machine learning is the former's model size is typically much larger, and there are much more parameters needed to be trained. Another crucial difference is deep learning could automatically learn the relevant features from the training data to make a prediction, compared to a typical machine learning model where we need to hand engineer the features from the training data first and then feed them into the model. Therefore, before the invention of deep learning, a lot of effort was spent in machine learning research on feature engineering, not the design of models.

For example, when we use SVM to perform an image classification task to tell whether an image is a dog or cat, we need to first handcraft some features and extract those features from the images. After that, the original images are no longer required. Features could be the gradient information of an image as we have demonstrated in the below image, where we extracted the gradient information from a dog image and a cat image. If we want to train an SVM classifier, we will need to convert all our dog images and cat images by the same feature transformations before feeding them to train the SVM.

Figure 8
EXAMPLE OF IMAGE RECOGNITION WITH SVM CLASSIFIER


Dog
Histogram of Oriented Gradients
Cat
Histogram of Oriented Gradients

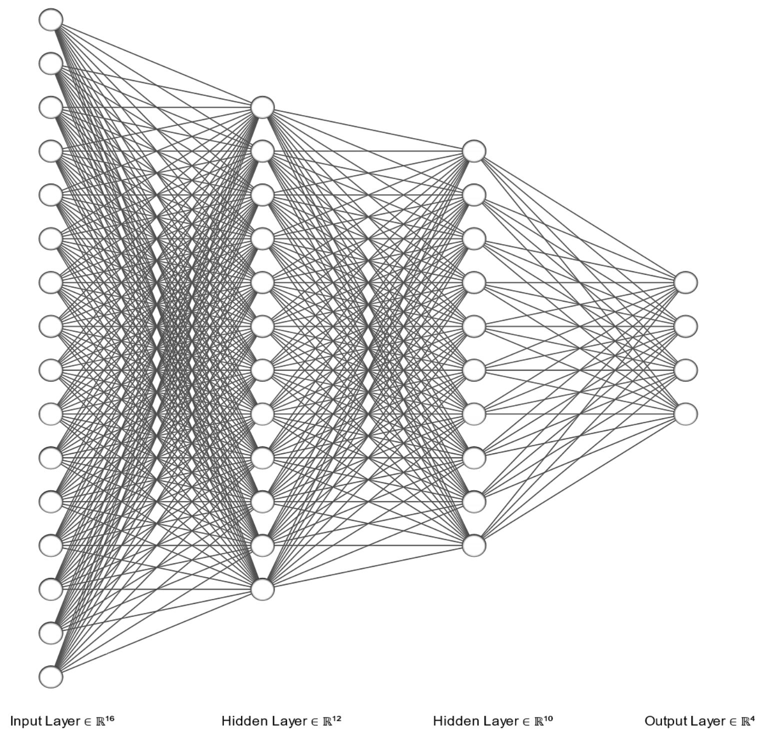
However, if we want to solve the same problem by using deep learning, we simply need to feed the whole image into the neural networks, and the networks will automatically learn what kind of features are important to tell whether it is a dog or cat.

2.1 DEEP NEURAL NETWORKS (DNN)

Deep Neural Networks (DNNs) gained much attention since Alex Krizhevsky successfully applied them and won the ImageNet Large Scale Visual Recognition Challenge in 2012. A deep neural network will look similar to Figure 9, where we have a Multiple Layer Perceptron (MLP), and each node in the network is a neuron except the first input layer. The input layer is the training data, and the output layers are the results we want to predict. Between the input and output layers, there are multiple hidden layers. In each layer other than the input layer, the nodes are called neurons, which will use a non-linear activation function to map the input to the output. One of the reasons why deep neural networks are so powerful is that they typically have a lot of hidden layers and a massive number of neurons. This allows them to learn a very high dimensional relationship between input and output, which could be an ideal fit for financial data since they can seem entirely random sometimes.

The lines between neurons are called connections, which represent how the data flows between different neurons. In the case of an MLP, each neuron has a connection to all the neurons in the previous layer, and we call those dense layers or fully connected layers. Taking the above dog and cat image classification problem as an example, to use an MLP as a classifier, we would need to flatten the 2-D image into a 1-D vector and would only have one neuron at the end of the MLP, which will output the labels indicating dog or cat. As we have already pointed out, there is no process for extracting features from images.

Figure 9
ILLUSTRATION OF A DEEP LEARNING MODEL


Input Layer $\in \mathbb{R}^{16}$
Hidden Layer $\in \mathbb{R}^{12}$
Hidden Layer $\in \mathbb{R}^{10}$
Output Layer $\in \mathbb{R}^{4}$

Each circular node represents a neuron, and the lines denote the connectivity between neurons. Neurons will apply a weight matrix $\boldsymbol{W}$ and a bias term $\boldsymbol{b}$ to the input data $\boldsymbol{X}$ it receives and will then apply an activation function $\boldsymbol{\sigma}$ to the results before passing the results $\boldsymbol{Y}$ to the neurons in the next layer. This whole process could be summarized as

$$Y = \sigma(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{b})$$

The activation function controls whether the current neuron should be active or not and how much it should be active. The range of the output of activation functions usually is in the range of $[-1,1]$ or $[0,1]$ and the choice of activation functions is usually a non-linear function such as a sigmoid function, Rectified Linear Unit (ReLU), or tanh function. The use of non-linear activations is one of the reasons the DNN models are capable of learning the non-linear relationships within the data and, thus, have better ability to represent the training data.[2]

Like machine learning, deep learning can be grouped into supervised learning, unsupervised learning, and reinforcement learning. For insurance applications, there is a recent success story from AXA Japan about applying deep learning in pricing.[3] AXA Japan's R&D team developed a deep learning model to predict significant claims, i.e., by entering the information of individual policies, we want to predict the likelihood that they will result in a substantial claim in the future. AXA entered over 70 input features including age, region, annual insurance premium, age of car, etc. into the deep learning model. The model is a fully connected neural network with three hidden layers. AXA used data in Google Compute Engine to train the model and Cloud Machine Learning Engine's HyperTune feature to tune hyperparameters. The resulting accuracy rate was about 78%.

2.2 DNN EXAMPLE OF TIME SERIES PREDICTION

In this section, we are going to demonstrate how to use deep neural networks (DNNs) to predict bitcoin prices. The DNN we are using in this section is mostly a feed-forward network. We are going to use two types of DNNs: Multi-layer Perceptron (MLP) and Convolutional Neural Networks (CNN).

2.2.1 MULTI-LAYER PERCEPTRON (MLP) VS. CONVOLUTIONAL NEURAL NETWORKS (CNN)

MLPs are probably one of the oldest DNN models and we have shown their network structure above in Figure 9. In an MLP, every neuron of the current layer will have one connection of every other neuron of the next layer. That is why the layers of an MLP are also called a fully connected or dense layer.

[2] Hornik, Kurt; Stinchcombe, Maxwell; White, Halbert. "Multilayer feedforward networks are universal approximators." University of California, San Diego . March 1989. https://www.sciencedirect.com/science/article/pii/0893608089900208

[3] Sato, Kaz. "Using Machine Learning for Insurance Pricing Optimization." March 2017. Google Cloud Platform https://cloud.google.com/blog/products/gcp/using-machine-learning-for-insurance-pricing-optimization

CNNs were invented to process 2-D images. However, in recent years, they are also widely used in time series data such as natural language processing or human activity recognition from accelerometer data recorded by smartphones or other wearable devices.[4]

It is natural to think that MLPs should be useful for time series applications. In some sense, MLPs are a lot like the traditional regression problem where we have a model to regress the input data. The difference is there is no need to design the mathematical model by hand; we only need to define how many neurons we need in the MLP, and the weights of the connections between different neurons describe the mathematical relationship to regress the training data. Nevertheless, the problem with MLPs is that the connections are so dense that it is very computationally demanding and slow, and it is also quite easy to overfit the training data.

As for a 1-D CNN, it is designed to find the pattern of a fixed segment of time series data. For each to-be-found pattern, we define a set of parameters, and we can predefine how many patterns we would like to reveal from each segment by experience. Therefore, CNNs require fewer connections between neurons across layers.

In the below experiment, we will present how to use both MLP and CNN in time series application with bitcoin history data.

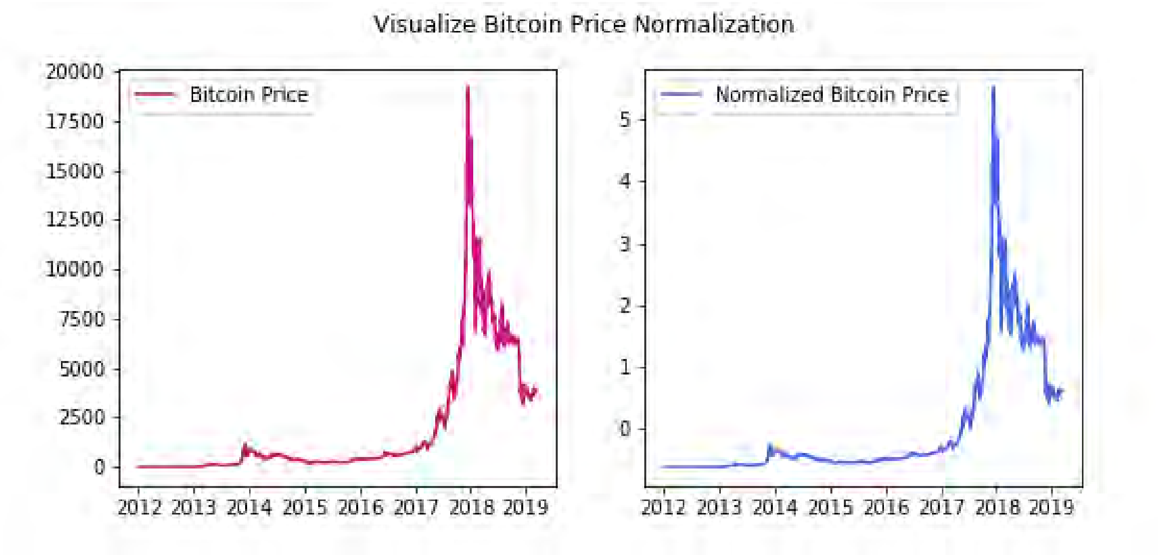
2.2.2 DATA PREPARATION

The dataset used is the bitcoin historical data from January 2012 to March 2019 and can be downloaded from the Kaggle website: https://www.kaggle.com/mczielinski/bitcoin-historical-data. The original dataset contains the bitcoin history data updated every minute with Open, High, Low, and Close prices. In this simple demonstration, we average the coin prices observed each day and use both an MLP and CNN to predict the next five days of bitcoin prices using the past 15 days prices.

There are 2,627 valid data points from January 2012 to March 2019. The original data is illustrated in the left figure below. It can be observed that the price is relatively flat between 2012 and 2017 and then rises quite high at the beginning of 2018 before falling later. The original data needs to be normalized first so that the model will treat every section of the data equally. Otherwise, the section with a large scale will have a bigger impact on the model and the model might overfit to those sections.

[4] Ackermann, Nils. "Introduction to 1D Convolutional Neural Networks in Kera for Time Sequences." Goodaudience, The Medium Spet 2018. https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf

Figure 10
BITCOIN PRICE NORMALIZATION



The time series bitcoin data does not explicitly have the ground truth label y (ground truth refers to the target variable being measured in training or testing), and both CNNs and MLPs are supervised, learning models. Therefore, we need to generate ground truth labels. As the goal here is to use the last 15 days of bitcoin prices to predict the following five days, both the MLP and CNN are used to try to find a mapping function $f(\cdot)$, which will perform the below mapping:

$\hat{y} = f(X[t_0 \cdots t_{14}])$, and the ground truth $y = X[t_{15} \cdots t_{19}]$ and the goal is to make sure the mean square error $\frac{1}{N}\Sigma\|y - \hat{y}\|^2$ is within some acceptable range. The 2,627 samples are split into two parts: the first 70% is used as training data, and the remaining 30% as testing data to evaluate the model accuracy after the training is complete.

2.2.3 CREATING THE MODEL

The MLP model used in this task consists of multiple dense layers (or fully connected layers), where each neuron in the previous layer will have a connection to every neuron in the next layer. Due to the use of dense layers, there are many parameters to be trained in the MLP model. In the MLP model we built, there are 21,509 parameters to be trained.

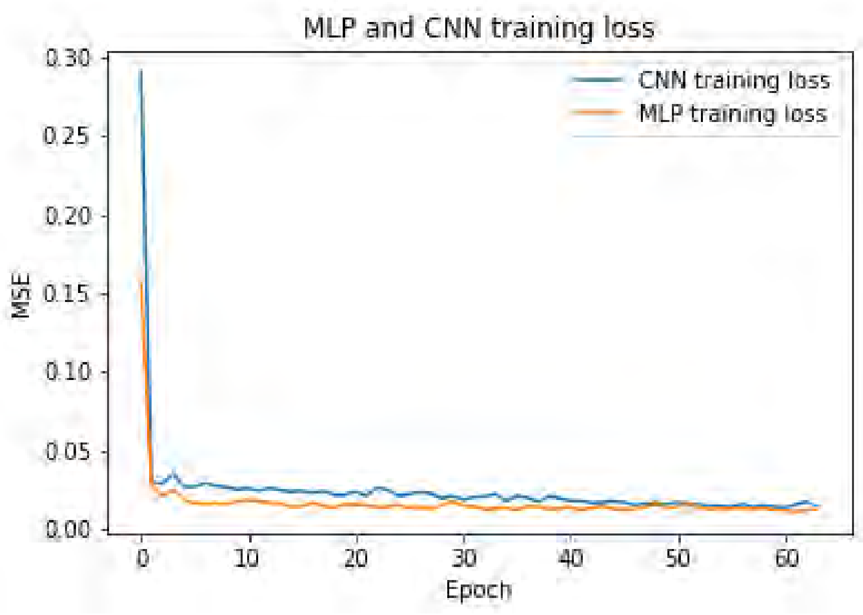
For the CNN model, multiple 1-D convolutional layers are used. Inside a convolutional layer, a set of weights (usually called filters or kernels) is systemically applied across the input data. For 2-D image detection, the filter is a 2-D dimensional array of weights. The filter is designed to detect the specific features of the input data, and the output of this operation is called a "feature map." Each convolutional layer will learn specific patterns from the data, and the next layer of the convolutional layer will learn more high dimension patterns from earlier layers. A Max-pooling layer is added right after the last convolutional layer to avoid overfitting. A dense layer is also used to make sure the output dimension is five. CNN models are quite effective at identifying the repeated patterns in the time series data; therefore, the required number of parameters is significantly smaller than for an MLP. In the CNN model we built, there are only ~9,000 parameters. For more details, please refer to the model on the GitHub site:

https://github.com/rranxxi/soa_research_ai_time_series/tree/master/bitcoin_cnn.

2.2.4 TRAINING THE MODEL

Both the CNN and MLP models were trained for around 64 epochs (an epoch is one complete presentation of the dataset to be learned to a learning machine). The MSE after each epoch of training is shown in Figure 15 for both models. The MSE of both models plateaued after around 20 epochs, which indicates that both of the models converged to their optimal state. Overall, MLPs tends to have slightly smaller MSEs than CNN models during the training stage. Next, we want to visit the testing results of the two models.

Figure 11
TRAINING LOSS COMPARISON OF MLP AND CNN MODELS IN BITCOIN TIME SERIES PREDICTION



2.2.5 RESULTS ANALYSIS

After the models were trained, we used them to predict bitcoin prices. We have a sliding window, which includes the 15 days of bitcoin data used both as the model input and predictor of the bitcoin price for the following five days. Then, we moved this sliding window by five days to predict the next five days. It is not surprising that, in the first 1,828 days, the predictions from both of the models have almost no errors since this part of the data has been used as training data. What is interesting is that, even with the testing data section, the predictions from both of the models are excellent. The MSE of the CNN model is 0.039 and the MSE of the MLP model is 0.0479, although the CNN model uses less than half of the parameters required by the MLP model. This result demonstrates that CNNs should be the preferred model in this time series analysis. This is, in fact, not counterintuitive. We human beings are also trying to recognize the patterns from the time series data to help us predict the future, and CNNs excel in identifying the pattern in both 1-D sequential data and 2-D image data. The fact that CNNs are proficient in recognizing pattern sets is a great advantage in time series predictions.

Figure 12
CNN/MLP PREDICTION ERROR

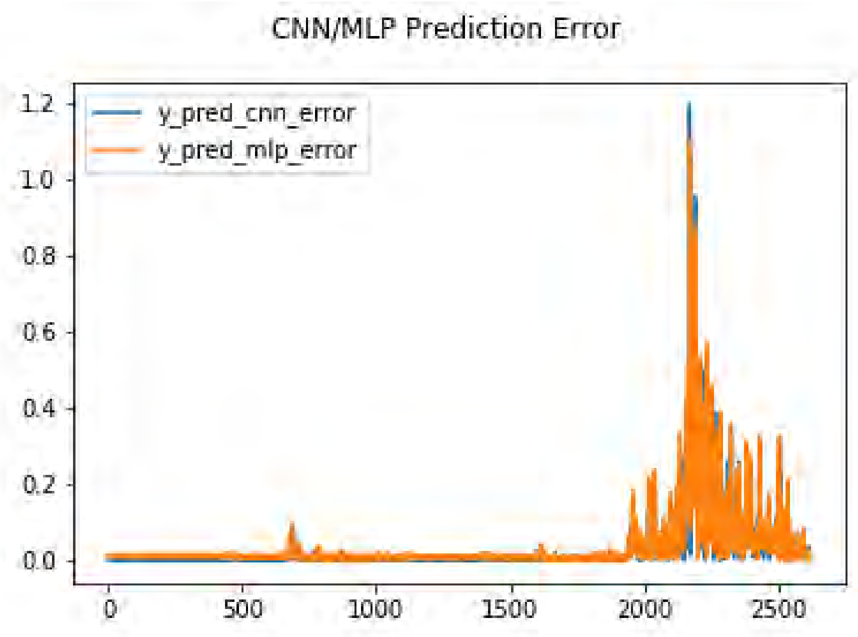
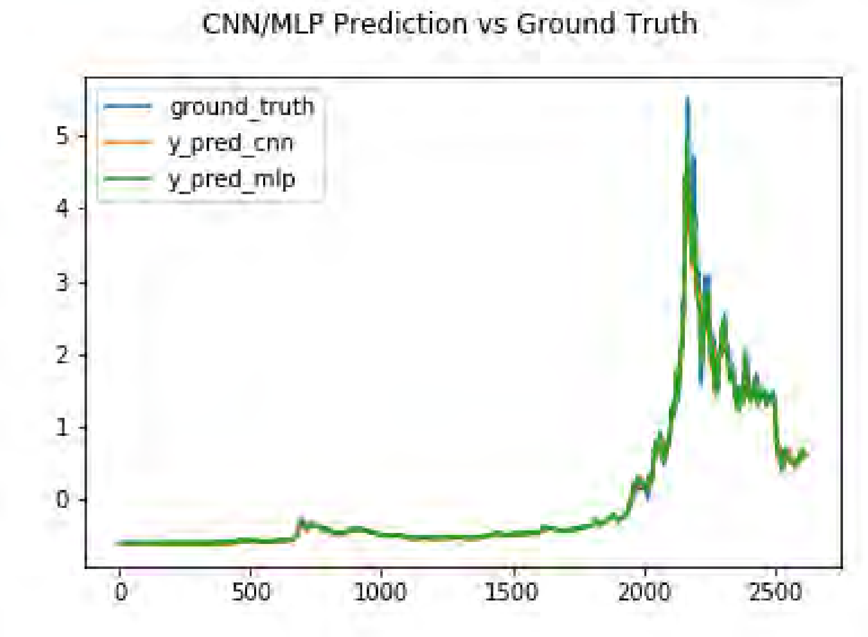


Figure 13
CNN/MLP PREDICTION VS GROUND TRUTH



Chapter 3: Time Series Prediction with Recurrent Neural Networks (RNN)

In this chapter, we are going to explore the method of using RNNs to perform time series predictions and how to use them for anomaly detection in the time series data. We will first introduce RNNs and follow that with one special type of RNN, the Long Short-Term Memory (LSTM) model. We will provide a step-by-step guide about how to use the LSTM model for time series predictions, as well as anomaly detection.

3.1 RECURRENT NEURAL NETWORKS (RNN)

RNNs are very close to our daily life. They are used by Apple Siri, Google Voice, Google Translate, etc. RNNs are a special type of deep learning model designed to recognize interesting patterns in time series data, such as stock markets, interest or exchange rates, natural languages, etc.

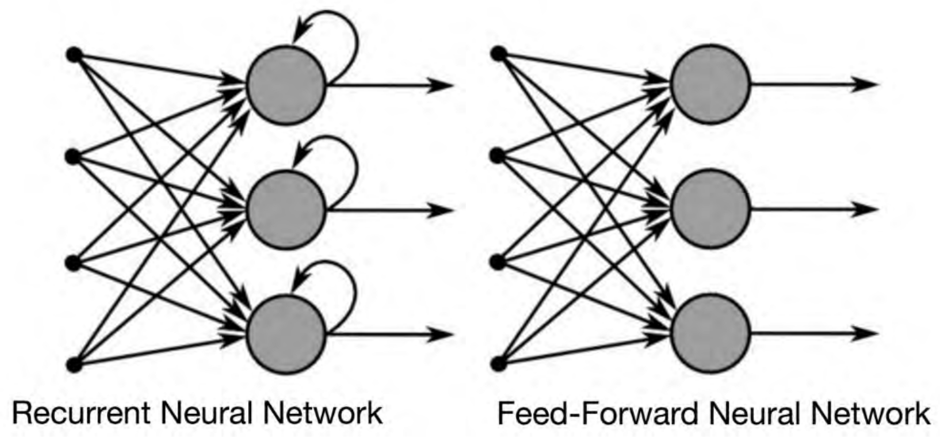
Different from the deep learning networks we have seen previously, RNNs have an internal memory and they can remember the important things they have observed and leverage that information to predict the future.

In a feed-forward neural network, we have an input $\boldsymbol{X}$ and a ground truth $\boldsymbol{Y}$. The goal is to learn the mapping function from $\boldsymbol{X}$ to $\boldsymbol{Y}$. In the most simple feed-forward network, this mapping will look like $\boldsymbol{Y} = \sigma(\mathbf{WX} + \mathbf{b})$, where $\boldsymbol{W}$ and $\boldsymbol{b}$ are the weight matrix and bias vector, respectively, and the σ is a non-linear activation sigmoid function $\frac{1}{1+e^{-x}}$, which allows the model to learn the non-linear relationship between the training data $\boldsymbol{X}$ and the ground truth $\boldsymbol{Y}$.

The input of the RNN will depend not only on the training data $\boldsymbol{X}$, but also on its past output. It maintains an internal state (or memory) $\boldsymbol{h}_t$, which is a high-level summarization of all the data it has observed. At each time step t, this state is updated according to the update function $\boldsymbol{h}_{t+1} = f(\boldsymbol{h}_t, \boldsymbol{X}_t)$, where $\boldsymbol{X}_t$ is the input data at time step t.

In a simple RNN, the mapping function f can be parameterized by the weight matrices $\boldsymbol{W}$ and $\boldsymbol{U}$, and its state can be updated according to $\boldsymbol{h}_{t+1} = \tanh(\boldsymbol{W}\boldsymbol{h}_t + \boldsymbol{U}\boldsymbol{X}_t)$, where $tanh$ is the non-linear activation function. During the training, we will feed the RNN a sequence of data and will learn W and $\boldsymbol{U}$ from those training sequences.

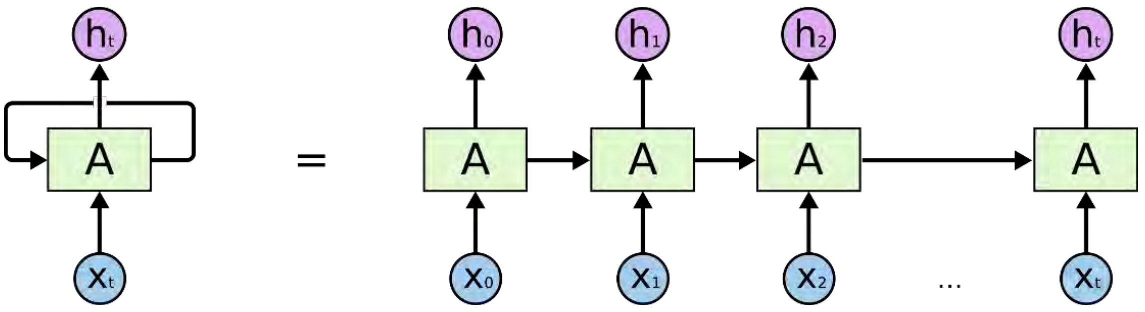
Figure 14
ILLUSTRATION OF RECURRENT NEURAL NETWORK AND FEED-FORWARD NEURAL NETWORK



3.2 VANISHING AND EXPLODING GRADIENT ISSUES

The RNN could be unrolled over time and it will look more like a feed-forward neural network. Because the output at the current time will also depend on the previous output, there is a backward propagation through time (BPTT) to update the weights of RNN during the training stage. However, BPTT might cause a fundamental flaw in the RNN. When the gradient is small, it will diminish during the BPTT, thus causing a vanishing gradient issue. On the other hand, if the gradient is large, it will explode during the BPTT and lead to the exploding gradient problem. An RNN variation named Long Short-Term Memory (LSTM) is more widely used as it mitigates those issues.

Figure 15
STRUCTURE OF RNN[5]

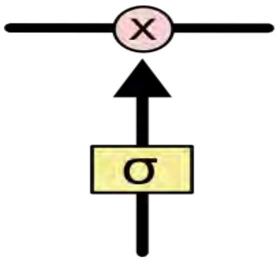


[5] Figure Source: Stanford CS231n course note http://cs231n.github.io/

3.3 LONG SHORT-TERM MEMORY (LSTM) MODEL

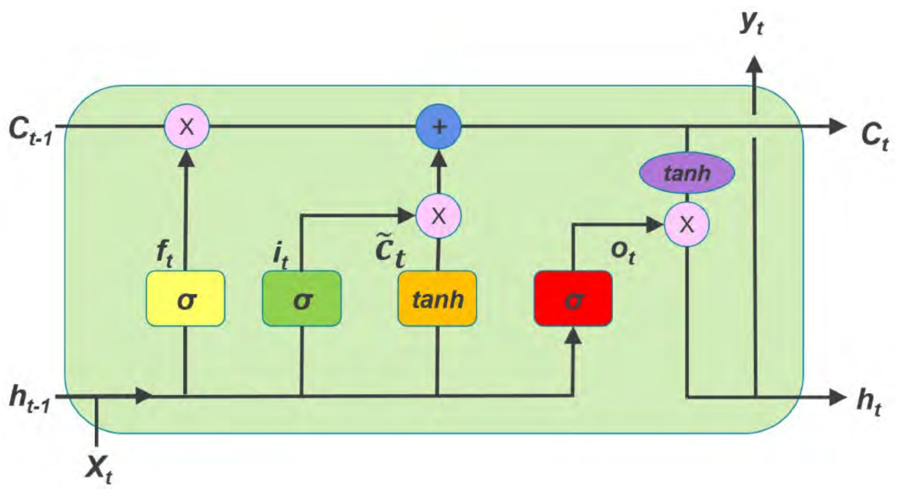
Due to the vanishing and exploding gradients issues, RNNs are hard to train and unable to memorize long-term information. In addition to that, the RNN model transforms the current information entirely by applying a mapping function and does not treat important information any differently. The LSTM extends RNN's memory and can selectively remember or forget information by structures called cell states and gates. Cell states are similar to the memory system of RNNs and gates are the valve to decide what information needs to be stored into memory and what information is required for output. A gate is illustrated below. It has a sigmoid function and a point-wise multiplication operation. The sigmoid output is within the range of [0, 1], where 0 means discard this information, and one means keep all the current information.

Figure 16
GATE OF LSTM



If we unroll the LSTM over time, it will be similar to the figure below. At each time step t, the cell state $\boldsymbol{C}_t$ will be updated based on the input training data $\boldsymbol{X}_t$ and the previous output $\boldsymbol{h}_{t-1}$. The different gates will determine what new information needs to be added into the cell state and to the output from the cell state. So, let us have a look at how each gate inside the LSTM works.

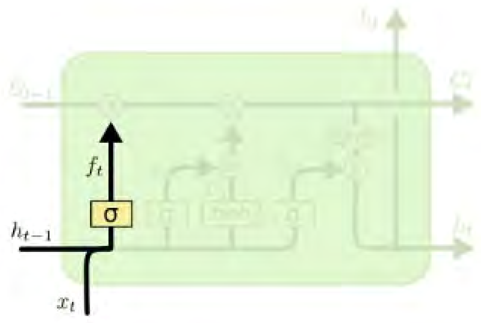
Figure 17
ILLUSTRATION OF LSTM STRUCTURE[6]



[6] Figure Source: Stanford CS231n course note http://cs231n.github.io/

Forget Gate $\boldsymbol{f_t}$: to forget the past information and reset the memory. In the time-series data, some past information might not be important to help us predict the future. The forget gate will take $\boldsymbol{h}_{t-1}$ and $\boldsymbol{X}_t$ and output a weight between [0,1] to scale the previous cell state and remove the unnecessary information. Forget gates can be described by equation $f_t = \sigma\left(W_f\ [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_f\right)$, where σ is the sigmoid activation function, $\boldsymbol{W}_f$ and $\boldsymbol{b}_f$ are the weight matrix and bias vector, respectively, which will be learned from the input training samples.

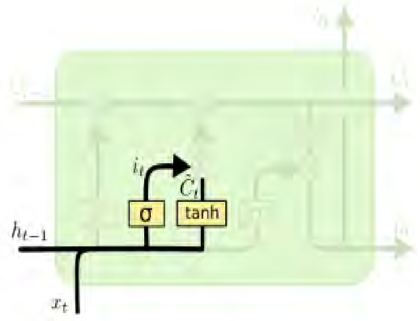
Figure 18
FORGET GATE OF LSTM MODEL[7]



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Input Gate $\boldsymbol{i_t}$: to add valuable information to its internal state. It makes sense that at each point of the time-series data, there may be some new important information that the LSTM would like to store. The input gate can be mathematically described as $\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i\ [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_i)$, where $\boldsymbol{W}_i$ and $\boldsymbol{b}_i$ are the weight matrix and bias vector, respectively. They will be learned from the input training samples. At each time step with the new information $\boldsymbol{X}_t$, we can compute a candidate cell state $\widetilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_C\ [\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_C)$, where $\boldsymbol{W}_C$ and $\boldsymbol{b}_C$ are the weight matrix and bias vector, respectively, to be trained.

Figure 19
INPUT GATE OF LSTM[8]



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

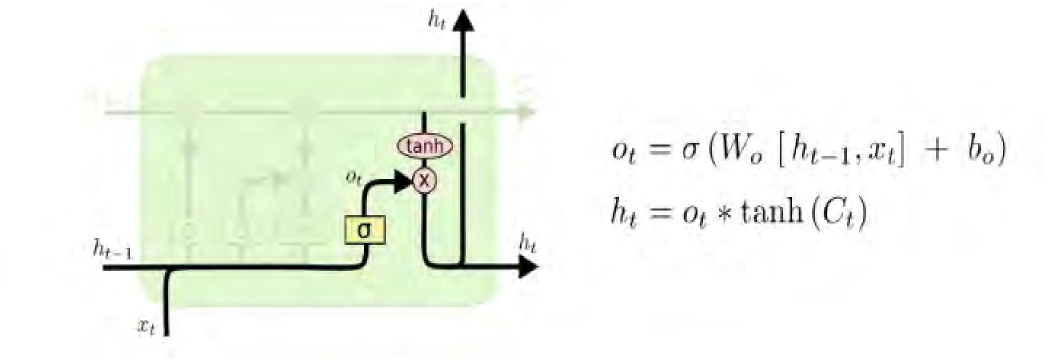
[7] Figure Source: Stanford CS231n course note http://cs231n.github.io/
[8] Figure Source: Stanford CS231n course note http://cs231n.github.io/

The forget and input gates are essentially two weights to control the contribution of the old cell state $\boldsymbol{C}_{t-1}$ and the new information $\widetilde{\boldsymbol{C}}_t$ to the new cell state $\boldsymbol{C}_t$, which will be updated by the equation:

$$\boldsymbol{C}_t = \boldsymbol{f}_t * \boldsymbol{C}_{t-1} + \boldsymbol{i}_t * \widetilde{\boldsymbol{C}}_t$$

Output Gate $\boldsymbol{O}_t$: controls how much information to reveal from its internal state $\boldsymbol{C}_t$ to the output. With the updated cell state, the LSTM has a fused version of its past information and the new information; what's left is how to make a prediction. In a similar fashion to the forget and input gates, the output gate will compute a weight $\boldsymbol{O}_t$ with previous output $\boldsymbol{h}_{t-1}$ and current input $\boldsymbol{X}_t$: $\boldsymbol{O}_t = \sigma(W_o\,[\boldsymbol{h}_{t-1}, \boldsymbol{X}_t] + \boldsymbol{b}_o)$, where $\boldsymbol{W}_o$ and $\boldsymbol{b}_o$ are the weight matrix and bias vector, respectively, to be trained. The current output can be simply calculated by applying a non-linear $\tanh$ function to the current cell state $\boldsymbol{C}_t$ and scaled by the output gate $\boldsymbol{O}_t$: $\boldsymbol{h}_t = \boldsymbol{O}_t * \tanh(\boldsymbol{C}_t)$.

Figure 20
OUTPUT GATE OF LSTM[9]


h_t
tanh
o_t
σ
h_{t-1}
h_t
x_t
$o_t = \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right)$
$h_t = o_t * \tanh\left(C_t\right)$

With the forget, input, and output gates, the LSTM optimally decides what information to store into its internal state $\boldsymbol{C}_t$ and how to make a prediction based on it. All the parameters of those gates will be updated during the training stage. Once the training is complete, the parameters of the three gates will be frozen to make predictions.

3.4 TIME SERIES PREDICTION WITH LSTM

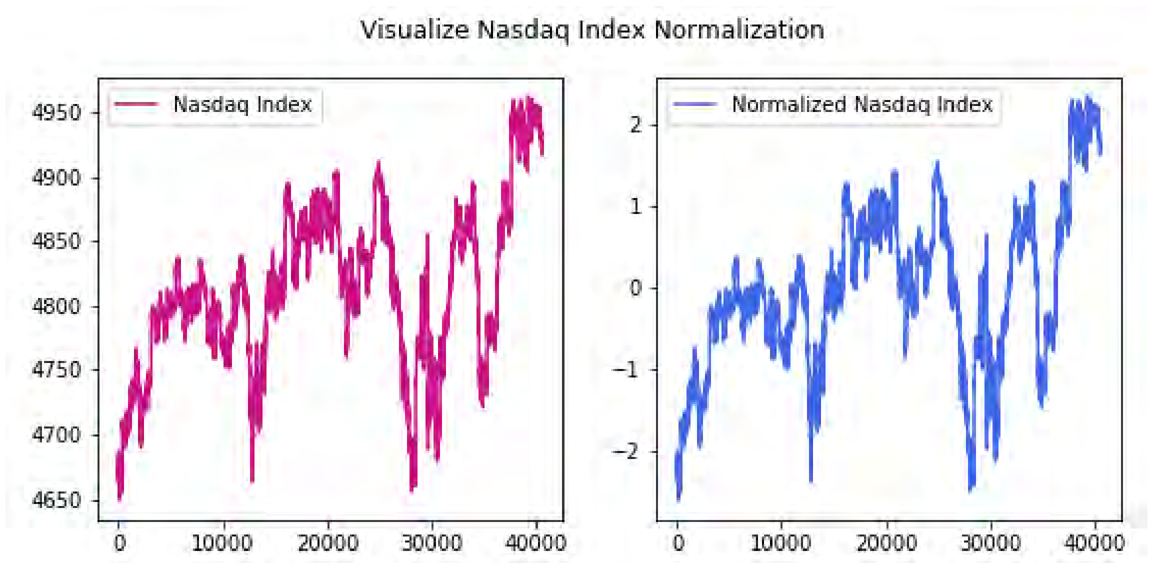
In this section, we are going to design an LSTM network to predict a dataset based on the NASDAQ index. This dataset includes 105 days of stock data starting from July 26, 2016, to December 22, 2016. Each day contains around 390 data points of most of the NASDAQ 100 stocks and its index value. The dataset can be download from this link: http://cseweb.ucsd.edu/~yaq007/NASDAQ100_stock_data.html. We will use the stock price and current NASDAQ index value to predict the future index value.

[9] Figure Source: Stanford CS231n course note http://cs231n.github.io/

3.4.1 DATA PREPARATION

We will be using the same data as in section 1.4 for the NASDAQ 100. The stock prices of the NASDAQ 100 have quite different scales. If we treat them equally, the model might put more weights on the stocks with more substantial value. We will apply the standard normalization by the formula $\frac{X-\mu}{\sigma}$, where μ is the mean and σ is the standard deviation of each stock and the index. To demonstrate the normalization process, we plot the NASDAQ index value before and after the normalization process as below. Each stock is also being normalized in the same fashion.

Figure 21
DATA COMPARISON OF NORMALIZATION

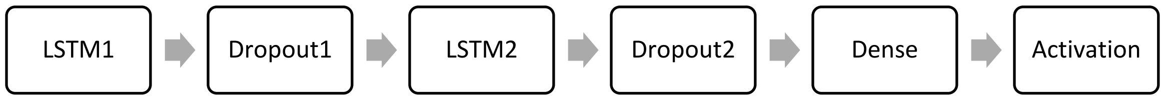


Similar to the previous modeling, we will also need to split all the data into a 70%/30% ratio for training and testing.

3.4.2 CREATING THE MODEL

Next, we need to create a model and train it with the training data. The model consists of two LSTM layers and there is one dropout layer after each LSTM layer, which will randomly disable some neurons during the training to avoid model overfitting. We stack multiple LSTM layers together to allow the model to learn to complicate temporal dynamics between the index and all its stocks. The output of the second dropout layer will connect to a dense layer (fully connected layer) before sending it to the activation function, which will map the dense layer output to a prediction value. We are using the mean-squared error function as our loss function to force the model to make a prediction as close to the training data as possible. The model looks like below:

Figure 22
STACK LSTM MODEL STRUCTURE



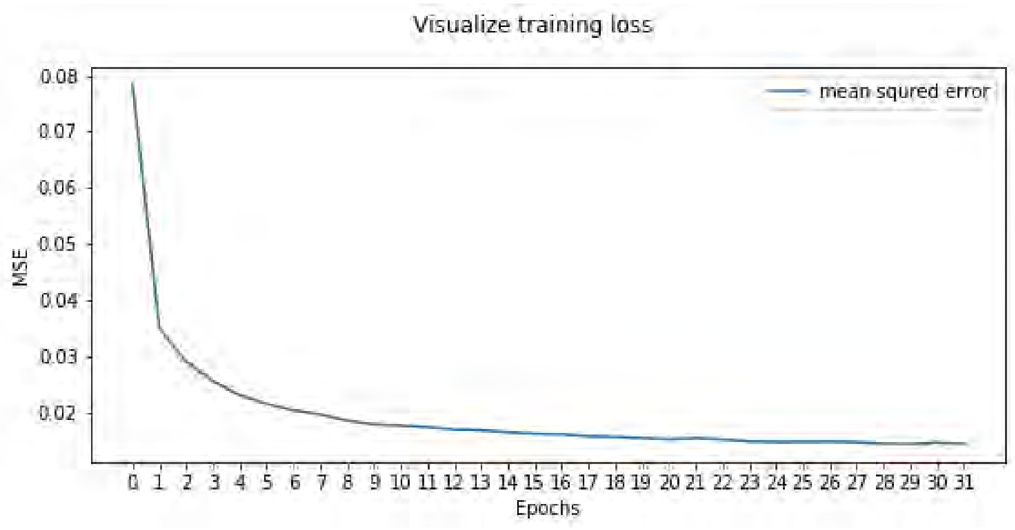
The model is available at:
https://github.com/rranxxi/soa_research_ai_time_series/tree/master/stock_prediction.

3.4.3 ROLLING WINDOW TRAINING

Although the LSTM, in theory, could memorize long-term features, in reality, the events happening right now likely depend only on data within a limited time window. For example, it is unlikely that the stock price from a year ago would be helpful for us to predict the stock price at 3:00 A.M. next Monday. Also, the forget gate in the LSTM has the same vanishing/exploding gradient issue with RNN. If we do not limit the window size, the model might take very long to train, and the convergence might be very slow. Therefore, we limit the LSTM to only use the stocks and index value in the past T time points for training.

The model is trained with 32 epochs, and we can see from Figure 27 that the MSE error after each epoch is descending, which indicates that the model is converging to its optimum value, and its prediction accuracy is improving over each epoch. Also, we notice that after around 20 epochs, the error has already plateaued. At this point, more rounds of training will not improve the model performance and may cause the model to overfit the training data. That is why we only train for 32 epochs.

Figure 23
TRAINING LOSS OF LSTM MODEL

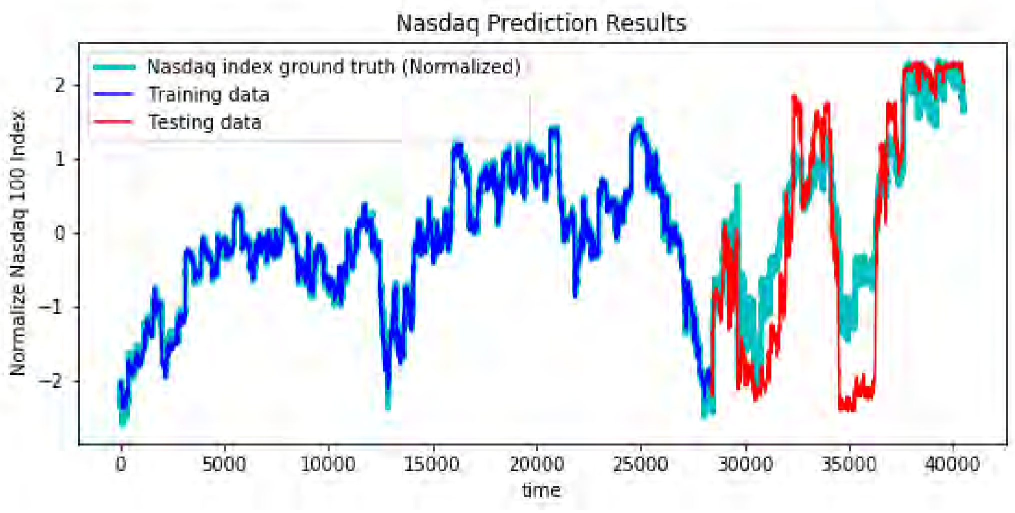


3.4.4 VISUALIZING THE RESULTS

After the model has been trained, we will freeze all the parameters in the model and use it to make predictions. The following figure shows the results when we apply the model to the entire dataset. It is expected that the prediction results will be exceptional in the training data section as the model has seen those data during the training. In the testing data section, the model has predicted the trend correctly, but failed to predict the correct scale. The red line (predicted results) appears to have the same trend with the light blue line (ground truth). However, the scale is off a bit. This is not surprising to us as the stock market is notoriously hard to predict. Even though we could not predict the exact value of the index, we could know the trend ahead of time.

LSTM models have proven their capability of predicting time series data. This model intends to provide a starting point for the readers to perform a similar time series prediction. Nevertheless, depending on the nature of the input data and the issues to be resolved, the program needs to be further adjusted and tested.

Figure 24
NASDAQ PREDICTION RESULTS



3.4.5 LSTM VS. TRADITIONAL FORECASTING METHODS

Time series forecasting is an essential subject in business and finance. There are several traditional forecasting techniques - one of the most notable statistical models is the Autoregressive Integrated Moving Average (ARIMA) with several variations. ARIMA has been a popular method for time series prediction for a long time; however, there are significant limitations to the ARIMA model. For instance, in a simple ARIMA model, it is hard to model the nonlinear relationship between variables. Furthermore, it is assumed that there is a constant standard deviation in errors of ARIMA, which usually is not satisfied in reality.

S.S. Namin and A.S. Namin conducted a study to compare the performance of LSTM vs. ARIMA models in time series forecasting in 2018.[10] Their study was conducted on many major indexes and shows that the

---

[10] Namin, Sima Siama, Namin Akbar Siamo, “FORECASTING ECONOMIC AND FINANCIAL TIME SERIES: ARIMA VS. LSTM “, Texas Tech University , March, 2018 https://arxiv.org/ftp/arxiv/papers/1803/1803.06386.pdf

LSTM outperformed the ARIMA model by about 85% on average in terms of reducing the error rate of forecasting results. As we noted earlier, the design of multiple layer neural networks allows high dimension data processing, which could uncover correlations that are typically not seen with traditional forecasting techniques.

Compared to traditional modeling techniques, the beauty of an RNN model is that it provides not only the data but also the previous state. Take a speech recognition example: if I say, “I lived in Paris for about eight years, mon français est bon.” In an RNN model, after studying the context of the first part of my speech, the state of the first part will be carried forward to the second part of my sentence and it could recognize that the second part is in French. However, traditional predictive modeling would fail to recognize this. Furthermore, an LSTM model is capable of learning from more extended memory, not just the previous state. In the speech recognition example above, if I start talking about how much I like the city, the model could automatically figure out I am talking about Paris based on the prior context.

This is not saying that RNN is better than traditional statistical modeling in all circumstances. ARIMA has been found to better model linear relationships, while RNN is better at modeling non-linear relationships. A classical method like ARIMA yields better results in forecasting short-term and univariate problems where LSTM is better for long-term forecasting. We can say the classical method is good at predicting simple, short-term, univariate problems, while RNN is good at more complex predictions.

3.5 TIME SERIES ANOMALY DETECTION WITH LSTM

Anomaly detection is critical in time series data, especially for the financial industry. It could help us remove outliers from the data. Anomaly detection has a wide range of applications such as fraud detection, health monitoring, web traffic monitoring, and so on. Anomaly detection in time series data can also help us identify the signals before some events happen. In this section, we are going to use the Numenta Anomaly Benchmark (NAB) dataset to demonstrate that the LSTM model can also be used to detect anomalies in time series data.

3.5.1 WHY LSTM

In theory, a classifier can also be used for anomaly detection as long as we have labels for each time point to know whether the current time point is an outlier or not. However, labeling every time point in practice is very costly. Moreover, the number of anomalies is typically way less than the regular data points. Thus, it is highly unbalanced. Training a machine learning based classifier with unbalanced labels is very difficult as the model might be overfitted with the classifier with more labels. RNN or LSTM are natural fits for time series anomaly detection. We are trying to train an LSTM model so it can predict future data. If we see a vast difference between our prediction and the incoming data, then we would suspect that the current data might be an outlier.

3.5.2 DATASET

Numenta Anomaly Benchmark (NAB)[11] is a new scale used for evaluating algorithms for anomaly detection in streaming, real-time applications. It is comprised of over 50 labeled real-world and artificial time-series data files, plus a novel scoring mechanism designed for real-time applications. We select the NYC taxi data from this dataset. It includes the number of taxi passengers between July 1, 2014, and January 31, 2015, and it consists of aggregating the total number of taxi passengers into 30-minute buckets. In this NYC taxi data, five anomalies occurred: during the NYC marathon, Thanksgiving, Christmas, New Year's Day, and a snowstorm. We will build an LSTM model to try to detect those anomalies.

3.5.3 DATA PREPARATION

As we have discussed previously, the first step to train a machine learning model is to normalize the data so that the data has a zero mean and a normalized scale. The model we used is pretty much the same as with the NASDAQ index prediction; the difference is we only have the number of passengers at previous time points and we are trying to predict the number in the next 30 minutes. Here, we are also using a rolling window approach and will use the last N time points to predict the next time point. N is a tunable hyper-parameter and is set to 50 during the training. Mathematically, we are trying to find a mapping function f, which will map the current LSTM state h_t and past N data to predict x_{t+1}, where $x_{t+1} = f(h_t, x_{t-n}, \dots, x_t)$.

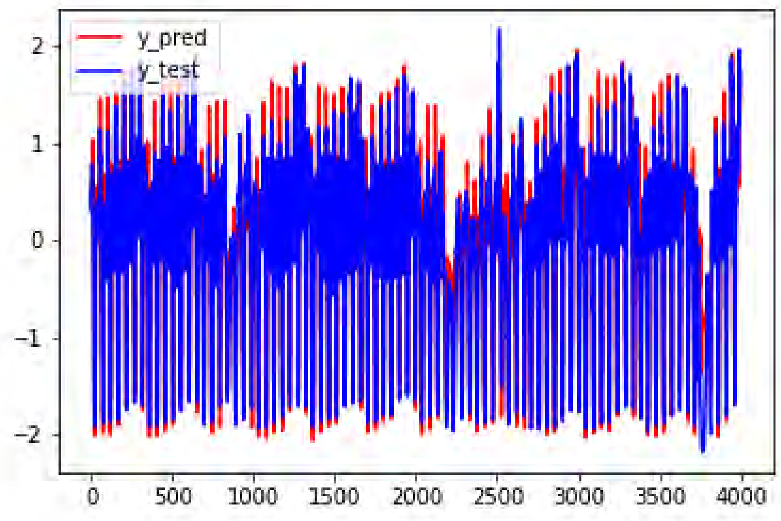
The whole data is split into two parts, the first 70% will be used for training, and the remaining 30% will be used as testing data to evaluate the model.

3.5.4 ANOMALY DETECTION WITH A TRAINED MODEL

We trained the model with 25 epochs and stopped the training once the error between the prediction and training data was small enough. We first used the trained model to predict the future number of taxi passengers and visualized the prediction results below. From the plot, the prediction looks pretty close to the test data, which proves that the model fits both the training and test data quite well.

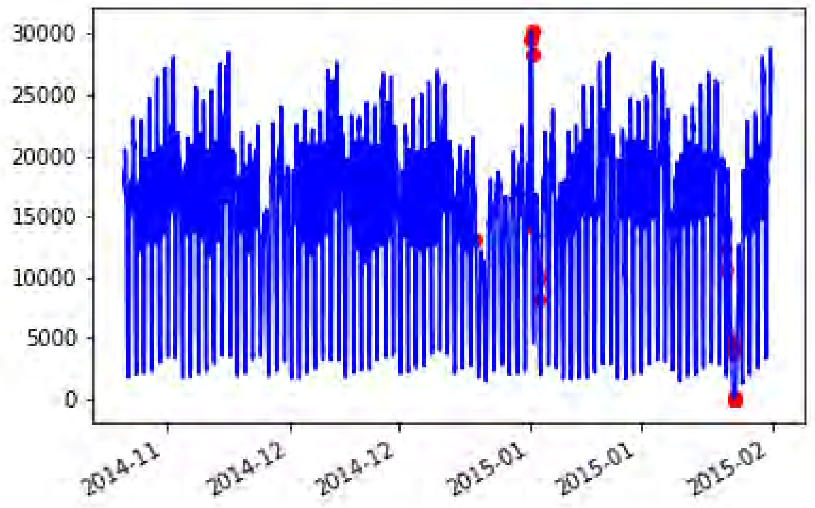
[11] The Numenta Anomaly Benchmark: https://github.com/numenta/NAB.

Figure 25
PREDICTED RESULTS OF NYC TAXI DATA WITH LSTM MODEL



Next, we detected the outliers from the test data. We defined that there were about 0.5% of outliers. We computed the error between the prediction and original test data, and the top 0.5% with the most significant errors were the suspicious outliers. It is interesting to see that we correctly detected the outliers around Christmas, New Year's Day, and the snowstorms (2015-01-23 to 2015-01-30), but failed to detect the NYC marathon (2014-11-02) and Thanksgiving (2014-11-27). There are some adjustments we could make to improve the model's performance, but the results are good enough to show that we could use LSTM to detect most of the anomalies successfully.

Figure 26
ANOMALY DETECTION RESULTS OF NYC TAXI DATA



Chapter 4: Conclusion and Recommendations

Artificial intelligence (AI) technology has extraordinary potential in the actuarial field. This report provided a high-level overview of AI and how it could be applied to actuarial work. We reviewed different kinds of machine learning algorithms, discussed potential actuarial applications of different AI models, and touched on some current success from the industry. We tied back to actuarial work with some real-life examples and provided results following each algorithm/model. Our hope is that, after reading this paper, actuaries will think more about "how could I bring AI into my daily work?"

Time series prediction is a critical topic in the business and finance areas. To actuaries, a better time series prediction could mean a more accurate valuation result, more effective asset management, or even more strategic business planning. The second goal of this report was to provide some concrete examples of solving different time-series related problems. We compared six supervised learning models for solving time series classification problems, implemented MLP and CNN deep learning models to predict Bitcoin prices, and provided an in-depth discussion about the LSTM model and how to apply it for stock index prediction, as well as for anomaly detection. All the models were built for the purpose of providing a clear starting point for actuaries to solve time-series related problems; however, we believe as AI develops, there will be more models that are better suited for the job.

To insurance companies, there are many challenges and obstacles in adapting AI technologies:

i. First, there are censorship and privacy concerns. After news spread about Facebook's use of private data in 2018, questions were raised as to how much privacy we should have after accepting AI into our daily lives? We know we will have a better sense of someone's mortality if we monitor his/her social media posts and subscriptions, grocery purchases, etc., but should we be allowed to do that?

ii. Secondly, there are regulatory concerns. Machine learning is a giant black box to regulators. There are no pre-defined mathematical equations, and the output from the program can constantly change due to new input data feeds. This certainly raises concerns from a regulatory perspective; regulators want all actuarial models to be transparent, well understood, and justified for control purposes.

iii. Lastly, there is a significant initial investment and ongoing cost. Insurance companies need to invest in both knowledge and hardware to adapt AI technology. There is a significant knowledge gap between actuaries and computer scientists. We cannot do the work without knowing both sides of the story. Continuous study and development are needed for all actuaries. From the hardware perspective, insurance companies need to upgrade their Graphics Processing Units (GPU) for model training and testing.

While AI can be a useful tool for actuaries, it is hard to imagine that a machine could think up worst case scenarios, provide business interpretations based on observed data, and continuously ask innovative questions about the products. Thus, actuaries will still need to intervene and interface with machines. The actuarial profession has a long history and actuaries are known for their continuous studying and being innovative. The emerging field of AI technology sets another opportunity for us to step to the next level.

About The Society of Actuaries

The Society of Actuaries (SOA), formed in 1949, is one of the largest actuarial professional organizations in the world dedicated to serving more than 32,000 actuarial members and the public in the United States, Canada and worldwide. In line with the SOA Vision Statement, actuaries act as business leaders who develop and use mathematical models to measure and manage risk in support of financial security for individuals, organizations, and the public.

The SOA supports actuaries and advances knowledge through research and education. As part of its work, the SOA seeks to inform public policy development and public understanding through research. The SOA aspires to be a trusted source of objective, data-driven research, and analysis with an actuarial perspective for its members, industry, policymakers, and the public. This distinct perspective comes from the SOA as an association of actuaries, who have a rigorous formal education and direct experience as practitioners as they perform applied research. The SOA also welcomes the opportunity to partner with other organizations in our work where appropriate. i

The SOA has a history of working with public policymakers and regulators in developing historical experience studies and projection techniques as well as individual reports on health care, retirement, and other topics. The SOA's research is intended to aid the work of policymakers and regulators and follow certain core principles:

Objectivity: The SOA's research informs and provides analysis that can be relied upon by other individuals or organizations involved in public policy discussions. The SOA does not take advocacy positions or lobby specific policy proposals.

Quality: The SOA aspires to the highest ethical and quality standards in all of its research and analysis. Our research process is overseen by experienced actuaries and nonactuaries from a range of industry sectors and organizations. A rigorous peer-review process ensures the quality and integrity of our work.

Relevance: The SOA provides timely research on public policy issues. Our research advances actuarial knowledge while providing critical insights on key policy issues, and thereby provides value to stakeholders and decision makers.

Quantification: The SOA leverages the diverse skill sets of actuaries to provide research and findings that are driven by the best available data and methods. Actuaries use detailed modeling to analyze financial risk and provide distinct insight and quantification. Further, actuarial standards require transparency and the disclosure of the assumptions and analytic approach underlying the work.

Society of Actuaries
475 N. Martingale Road, Suite 600
Schaumburg, Illinois 60173
www.SOA.org