Stock Price Prediction using Stock Market Data and News

Collector: 1305063

Coder: 13050117

Trainer: 1305010

Writer: 1305011

Leader: 1305061

**Project Overview**

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. A wealth of information is available in the form of historical stock prices, company performance data and news data suitable for machine learning algorithms to process.

Can we actually predict stock prices with machine learning? Investors make educated guesses by analyzing data. They'll read the news, study the company history, industry trends and other lots of data points that go into making a prediction. The prevailing theories is that stock prices are totally random and unpredictable but that raises the question why top firms like Morgan Stanley and Citigroup hire quantitative analysts to build predictive models.

This project utilizes Deep Learning models, Long-Short Term Memory (LSTM) Neural Network algorithm, to predict stock prices. For data with timeframes recurrent neural networks (RNNs) come in handy but recent researches have shown that LSTM networks are the most popular and useful variants of RNNs.

We have used Keras to build a LSTM to predict stock prices using historical stock prices and news data and visualize both the predicted price values over time and the optimal parameters for the model.

**Problem Statement**

Given a dataset of:

* World News Headlines &
* Price history data of some companies

We are going to predict

* The closing price of each company for a day

**Performance Metrics**

For a company, we are going to measure

as our performance metric.

We then compute the average of all such errors for each company as our combined performance metric.

**Data Description**

The dataset consists of:

* Top 5 world news headlines from Kaggle &
* The opening, closing, adjusted closing, high and low price for 15 companies from Yahoo Finance

for approximately eight years (from 08-08-2008 to 01-07-2016)

The 15 American based multinational technology companies used in our data are as follows:

1. Baidu Inc
2. Adobe Systems Incorporated
3. Oracle Corporation
4. Amazon.com, Inc.
5. Alphabet Inc Class C
6. NVIDIA Corporation
7. Microsoft Corporation
8. NetEase Inc (ADR)
9. Electronic Arts Inc.
10. Apple Inc.
11. QUALCOMM, Inc.
12. Cisco Systems, Inc.
13. Texas Instruments Incorporated
14. Intel Corporation
15. IBM Common Stock

**Data preprocessing**

An important part of data processing in this project is textual data processing. First we accumulate all text of whole dataset in a list. We tokenize all words from that list. We count frequency of each words and collect 10000 most frequent words. We give each word a unique numeric id. Then for each of the 5 headlines of all days, we convert news headlines to a vector of numeric ids. For example, becomes . We also pad each vector with zeros to achieve a fixed size of 50. So, becomes. So, to clarify, each day contains 5 news headlines, and it is converted to a vector of size. In numpy, (250,). For all 1989 data points, we get a matrix of (1989, 250) for only news data.

We use min-max scaling to bring each column of the price data to interval.

For this project we fixed our look back window to 7 days. Our approach is to use news headlines of past 7 days including today and market prices for past 7 days excluding today to predict the closing price of today. To elaborate, let’s assume today is day. We use the news headlines (= news data for day) and price data ( = price data for day) to predict closing price of day. We process both news data and price data according to this.

For news data, we create 7 different list for 7 look back. That is, our model has 7 inputs for news data, each corresponds to all new headlines for a date (250 size vector). For a sample at timestamp t,

* Input 7 gets all news headlines of Day t
* Input 6 gets all news headlines of Day t – 1
* …
* Input 1 gets all news headlines of Day t – 6

To be able to do that we create 7 list, to feed 7 inputs in our model.

* List 1 contains news headlines of day 1, 2, 3, … of size (1981, 250)
* List 2 contains news headlines of day 2, 3, 4, … of size (1981, 250)
* …
* List 7 contains news headlines of day 7, 8, 9, … of size (1981, 250)

The price data however converted from (1989, 60) to (1981, 7, 60). Each 1981 data point of converted price data contains 60 prices for each past 7 days.

After that data is divided into 80%-10%-10% that is train, validation and test set.

**Model description**

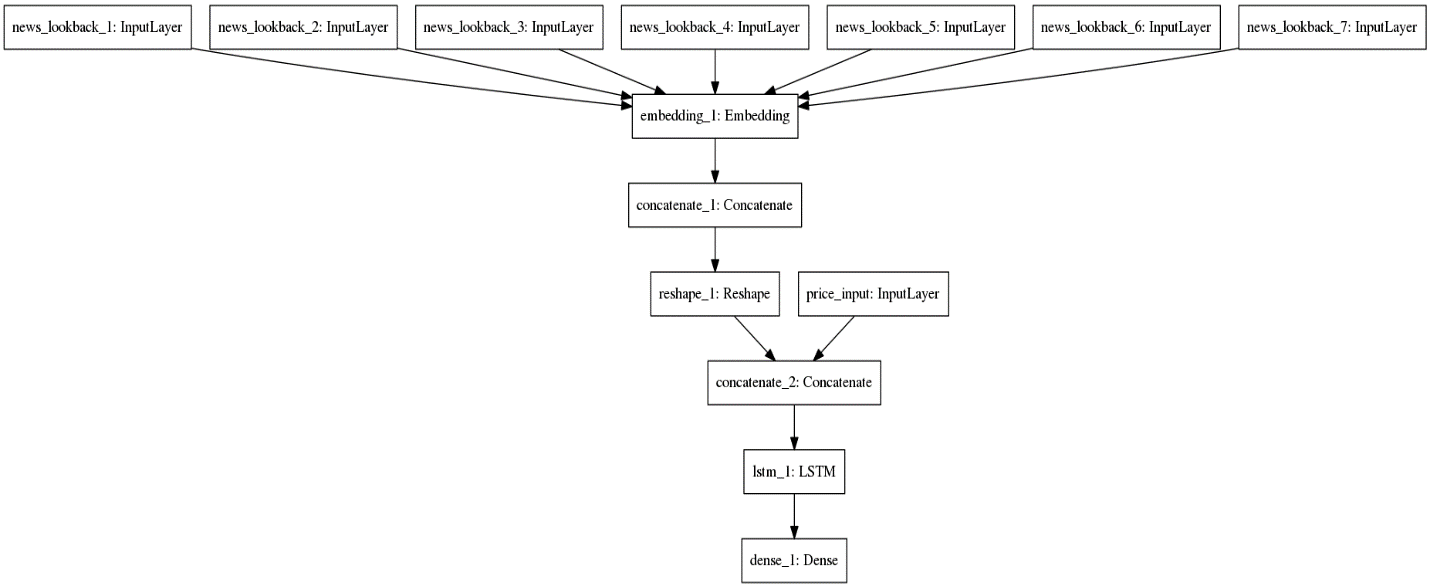
As our problem is a time series prediction problem, we are using Recurrent Neural Network. Our approach is to feed news data and price data together to a RNN cell. There are a number of options for an RNN cell – vanilla, Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM). Here we use LSTM.

As we processed news data and price data separately, we throw them separately into the network. So, we have different input lines for news and price. Also, before passing onto network, news data is just a bunch of word ids. To convert it into meaningful representation, we have a common embedding layer which trains simultaneously. Embedding layer tries to find best possible vector representation of each word which will minimize the loss.

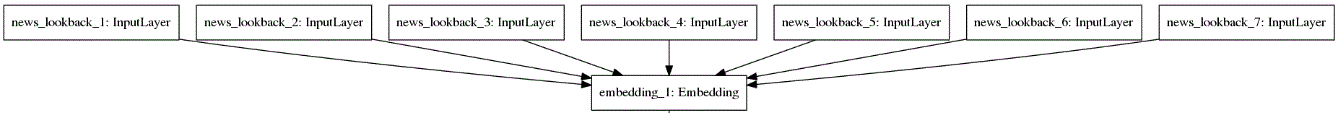
Then both data gets concatenate before entering recurrent cell. We get output from a fully connected dense layer attached in front of that recurrent cell.

**Architecture description**

Here is an overall view of our architecture. Let us break it down.



**News Input Layer**



As we have stated above, we have 7 input layers. Each layer corresponds to all new headlines for a date. For a sample at timestamp t,

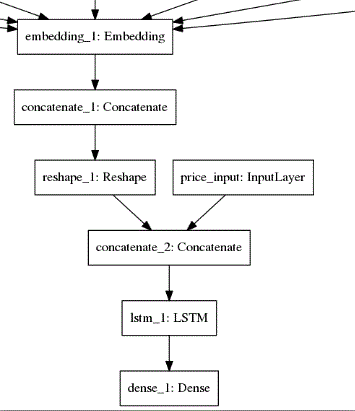
* Input Layer 7 gets all news headlines of Day t
* Input Layer 6 gets all news headlines of Day t – 1
* …
* Input Layer 1 gets all news headlines of Day t – 6

**Embedding Layer**

Each news data gets passed through a shared embedding layer. It converts each word id to a vector of length 4

**Concatenate Layer 1**

It concatenates the vectors of each word of each news for each headline for the last 7 days.

**Reshape Layer**

It reshapes the output of the Concatenate\_1 layer to align them correctly to concatenate with price data further down the architecture.

**Price Input Layer**

It takes Opening Price, Low Price, High Price and Adjacent Closing Price for the 15 companies. In total a vector of length 60.

**Concatenate Layer 2**

It concatenates news vector with price vector for a sample.

**LSTM**

It processes the output of concatenate\_2 layer and feeds it to the dense layer.

**Dense Layer**

It generates a vector for the closing prices of the 15 companies.

**Algorithm and techniques**

**Time series prediction**

Given a time series, predicting the next value is a problem that fascinated a lot of programmers for a long time. One may mistake it for regression. It is close, but not the same as regression. In a time series, each value is affected by the values just preceding this value.

For example, if there is a lot of traffic at 4.55 in a junction, chances are that there will be some traffic at 4.56 as well. This is called autocorrelation. If you are doing regression, you will only consider x(t) while due to autocorrelation, x(t-1), x(t-2), … will also affect the outcome. So we can think about time series forecasts as regression that factor in autocorrelation as well.

**Look back window**

For time series prediction there are a number of ways to feed neural network RNN cell (here, LSTM). The one of them is using look back window.

Idea is to to predict x(t), next value in a time series, we feed not only x(t-1), but x(t-2), x(t-3), …, x(t-w) to the model, where w = look back window.

**Data Scaling**

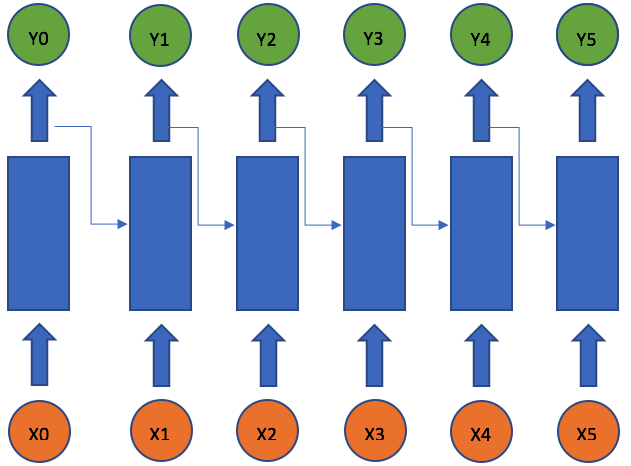
The data for a sequence prediction problem probably needs to be scaled when training a neural network, such as a Long Short-Term Memory recurrent neural network.

When a network is fit on unscaled data that has a range of values (e.g. quantities in the 10s to 100s) it is possible for large inputs to slow down the learning and convergence of the network and in some cases prevent the network from effectively learning a problem.

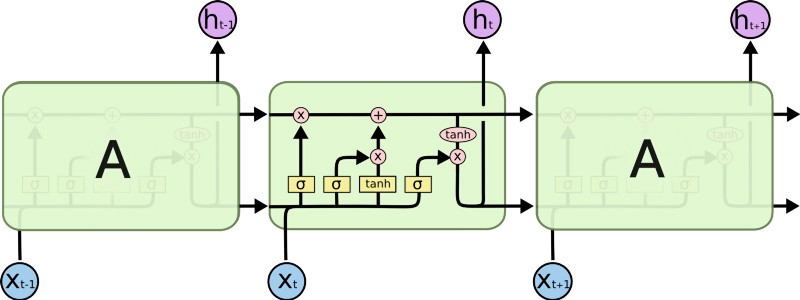
There are two types of scaling of s series that one may want to consider: normalization and standardization. We use normalization. Normalization is a rescaling of the data from the original range so that all values are within the range of 0 and 1.

Although, Normalization requires that we know or are able to accurately estimate the minimum and maximum observable values. We may be able to estimate these values from our available data. If a time series is trending up or down, estimating these expected values may be difficult and normalization may not be the best method to use on a problem.

**Long Short Term Neural Network**

A recurrent neural network deals with sequence problems. The simplest recurrent neural network can be viewed as a fully connected neural network if we unroll the time axes. One can build a deep recurrent neural network by simply stacking units to one another. A simple recurrent neural network works well only for a short-term memory.

As we have talked about, a simple recurrent network suffers from a fundamental problem of not being able to capture long-term dependencies in a sequence. This is a problem because we want our RNNs to analyze text and answer questions, which involves keeping track of long sequences of words. In late ’90s, LSTM was proposed by Sepp Hochreiter and Jurgen Schmidhuber, which is relatively insensitive to gap length over alternatives RNNs, hidden markov models, and other sequence learning methods in numerous applications.



This model is organized in cells which include several operations. LSTM has an internal state variable, which is passed from one cell to another and modified by Operation Gates –

* Forget Gate – controls flow of information from previous time step
* Input Gate – controls how much information should be taken from current time step
* Output Gate – controls how much information of the internal state is passed to output

**Word Embedding**

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.

Pretrained word embeddings are available. However, our model learns word embedding on the go to maximize performance.

**Optimizers**

* **RmsProp (Root Mean Square Propagation)**: It maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).
* **Adam (Adaptive moment estimation)**: Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. It is an update to RMSProp optimizer in which the running average of both the gradients and their magnitude is used. In practice Adam is currently recommended as the default algorithm to use, and often works slightly better than RMSProp.

**Hyper parameter tuning**

We tune our model considering 4 hyper parameters – optimizer, number of epochs, loss function and number of LSTM units.

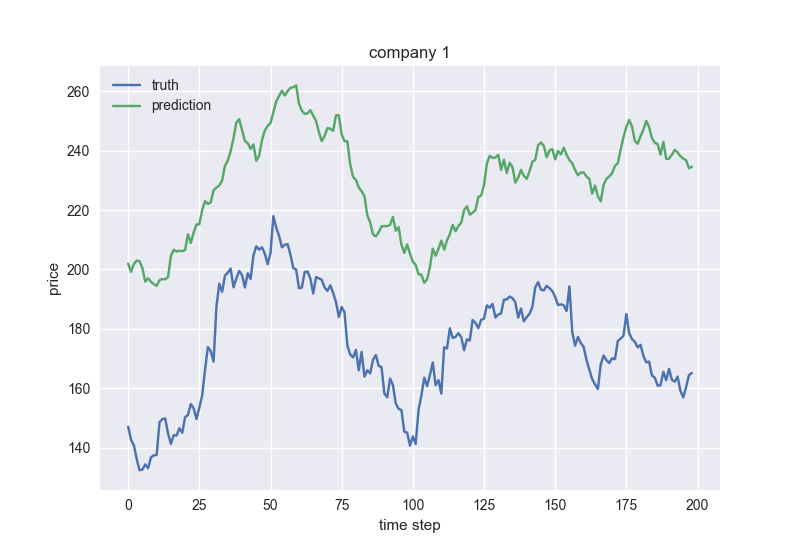
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Options** | **Optimizer** | **Epochs** | **Loss Function** | **LSTM unit** |
| adam | 20 | Mean squared error | 128 |
| rmsprop | 40 | Mean absolute error | 256 |

We check all 16 possible combination and select the one with least error score.

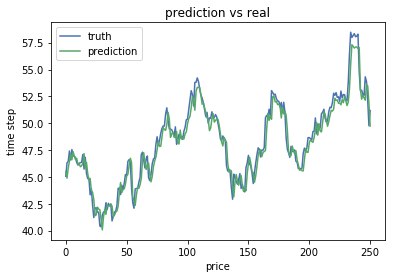
**Result description**

We show the result in Report.docx file. Analyzing the result, we see overall ‘rmsprop’ performed much better in almost cases.

**Conclusion**

As the results show, predicted price closely resembles the actual price for some companies, while deviates significantly for others. As below,

However, when we only consider historical price data and ignore the news headlines, the model predicts more precisely. As below,



It is important to understand that, though there may be news that drive share prices, for most cases it is the other way round i.e. price action that drives the headlines. Therefore, it might be fallible to try to predict the market based on news headlines. What moves the market is much more complex and it is even more complex these days when we have computer algorithms and high frequency trading making a major impact.