Predict Stock Prices Using RNN: Part 2

July 22, 2017 · 9 min · Lilian Weng

In the Part 2 tutorial, I would like to continue the topic on stock price prediction and to endow the recurrent neural network that I have built in Part 1 with the capability of responding to multiple stocks. In order to distinguish the patterns associated with different price sequences, I use the stock symbol embedding vectors as part of the input.

Dataset

During the search, I found this library for querying Yahoo! Finance API. It would be very useful if Yahoo hasn't shut down the historical data fetch API. You may find it useful for querying other information though. Here I pick the Google Finance link, among a couple of free data sources for downloading historical stock prices.

The data fetch code can be written as simple as:

```
import urllib2
from datetime import datetime
BASE_URL = "https://www.google.com/finance/historical?"
           "output=csv&q={0}&startdate=Jan+1%2C+1980&enddate={1}"
symbol_url = BASE_URL.format(
    urllib2.quote('GOOG'), # Replace with any stock you are interested.
    urllib2.quote(datetime.now().strftime("%b+%d,+%Y"), '+')
)
```

When fetching the content, remember to add try-catch wrapper in case the link fails or the provided stock symbol is not valid.

```
try:
    f = urllib2.urlopen(symbol_url)
    with open("GOOG.csv", 'w') as fin:
            print >> fin, f.read()
    except urllib2.HTTPError:
        print "Fetching Failed: {}".format(symbol_url)
```

The full working data fetcher code is available here.

Model Construction

The model is expected to learn the price sequences of different stocks in time. Due to the different underlying patterns, I would like to tell the model which stock it is dealing with explicitly. Embedding is more favored than one-hot encoding, because:

1. Given that the train set includes N stocks, the one-hot encoding would introduce N (or $N-1$) additional sparse feature dimensions. Once each stock symbol is mapped onto a much smaller embedding vector of length k, $k \ll N$, we end up with a much more compressed representation and smaller dataset to take care of.
2. Since embedding vectors are variables to learn. Similar stocks could be associated with similar embeddings and help the prediction of each others, such as “GOOG” and “GOOGL” which you will see in Fig. 5. later.

In the recurrent neural network, at one time step t, the input vector contains `input_size` (labelled as w) daily price values of i-th stock, $(p_{i,tw}, p_{i,tw+1}, \dots, p_{i,(t+1)w-1})$. The stock symbol is uniquely mapped to a vector of length `embedding_size` (labelled as k), $(e_{i,0}, e_{i,1}, \dots, e_{i,k})$. As illustrated in Fig. 1., the price vector is concatenated with the embedding vector and then fed into the LSTM cell.

Another alternative is to concatenate the embedding vectors with the last state of the LSTM cell and learn new weights W and bias b in the output layer. However, in this way, the LSTM cell cannot tell apart prices of one stock from another and its power would be largely restrained. Thus I decided to go with the former approach.

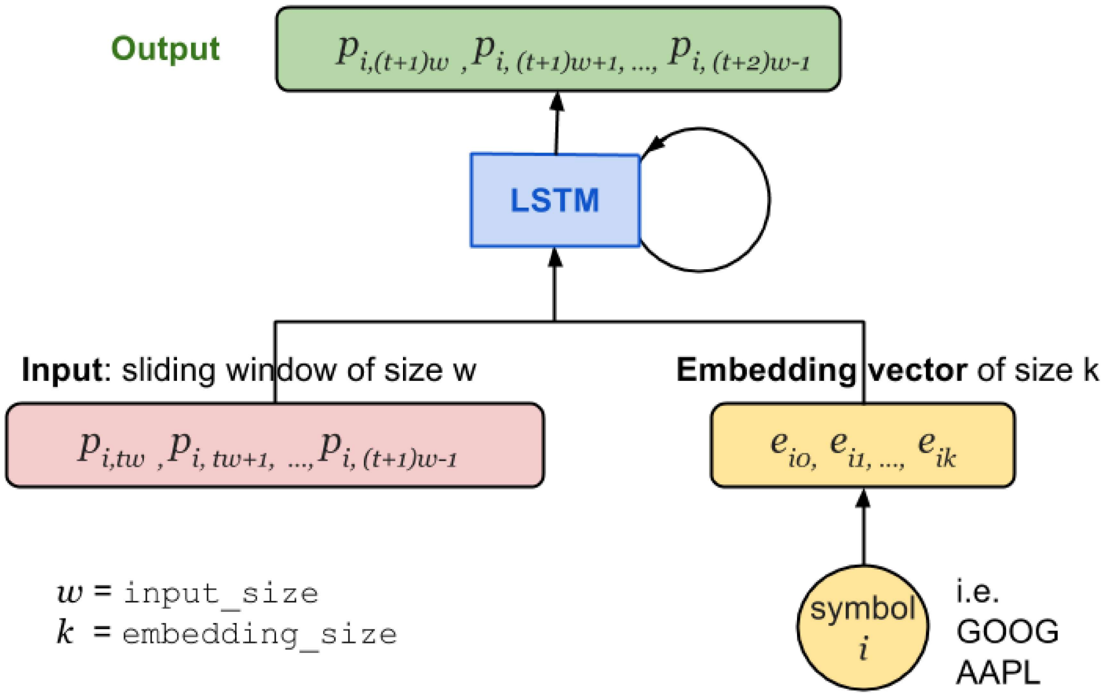

Output
$p_{i,(t+1)w}, p_{i,(t+1)w+1}, ..., p_{i,(t+2)w-1}$
LSTM
Input: sliding window of size w
$p_{i,tw}, p_{i,tw+1}, ..., p_{i,(t+1)w-1}$
Embedding vector of size k
$e_{i0}, e_{i1}, ..., e_{ik}$
w = input_size
k = embedding_size
symbol i
i.e.
GOOG
AAPL

Fig. 1. The architecture of the stock price prediction RNN model with stock symbol embeddings.

Two new configuration settings are added into `RNNConfig` :

- `embedding_size` controls the size of each embedding vector;
- `stock_count` refers to the number of unique stocks in the dataset.

Together they define the size of the embedding matrix, for which the model has to learn `embedding_size` $\times$ `stock_count` additional variables compared to the model in Part 1.

```
class RNNConfig():
    # ... old ones
    embedding_size = 3
    stock_count = 50
```

Define the Graph

— Let's start going through some code —

(1) As demonstrated in tutorial Part 1: Define the Graph, let us define a `tf.Graph()` named `lstm_graph` and a set of tensors to hold input data, `inputs` , `targets` , and `learning_rate` in the same way. One more placeholder to define is a list of stock symbols associated with the prices. Stock symbols have been mapped to unique integers beforehand with label encoding.

```
# Mapped to an integer. one label refers to one stock symbol.
stock_labels = tf.placeholder(tf.int32, [None, 1])
```

(2) Then we need to set up an embedding matrix to play as a lookup table, containing the embedding vectors of all the stocks. The matrix is initialized with random numbers in the interval [-1, 1] and gets updated during training.

```
# NOTE: config = RNNConfig() and it defines hyperparameters.
# Convert the integer labels to numeric embedding vectors.
embedding_matrix = tf.Variable(
    tf.random_uniform([config.stock_count, config.embedding_size], -1.0, 1.0)
)
```

(3) Repeat the stock labels `num_steps` times to match the unfolded version of RNN and the shape of `inputs` tensor during training. The transformation operation tf.tile receives a base tensor and creates a new tensor by replicating its certain dimensions multiples times; precisely the i-th dimension of the input tensor gets multiplied by `multiples[i]` times. For example, if the `stock_labels` is `[[0], [0], [2], [1]]` tiling it by `[1, 5]` produces `[[0 0 0 0 0], [0 0 0 0 0], [2 2 2 2 2], [1 1 1 1 1]]` .

```
stacked_stock_labels = tf.tile(stock_labels, multiples=[1, config.num_steps])
```

(4) Then we map the symbols to embedding vectors according to the lookup table `embedding_matrix` .

```
# stock_label_embeds.get_shape() = (?, num_steps, embedding_size).
stock_label_embeds = tf.nn.embedding_lookup(embedding_matrix, stacked_stock_labels)
```

(5) Finally, combine the price values with the embedding vectors. The operation tf.concat concatenates a list of tensors along the dimension `axis` . In our case, we want to keep the batch size and the number of steps unchanged, but only extend the input vector of length `input_size` to include embedding features.

```
# inputs.get_shape() = (?, num_steps, input_size)
# stock_label_embeds.get_shape() = (?, num_steps, embedding_size)
# inputs_with_embeds.get_shape() = (?, num_steps, input_size + embedding_size)
inputs_with_embeds = tf.concat([inputs, stock_label_embeds], axis=2)
```

copy

The rest of code runs the dynamic RNN, extracts the last state of the LSTM cell, and handles weights and bias in the output layer. See Part 1: Define the Graph for the details.

Training Session

Please read Part 1: Start Training Session if you haven't for how to run a training session in Tensorflow.

Before feeding the data into the graph, the stock symbols should be transformed to unique integers with label encoding.

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
label_encoder.fit(list_of_symbols)
```

The train/test split ratio remains same, 90% for training and 10% for testing, for every individual stock.

Visualize the Graph

After the graph is defined in code, let us check the visualization in Tensorboard to make sure that components are constructed correctly. Essentially it looks very much like our architecture illustration in Fig. 1.

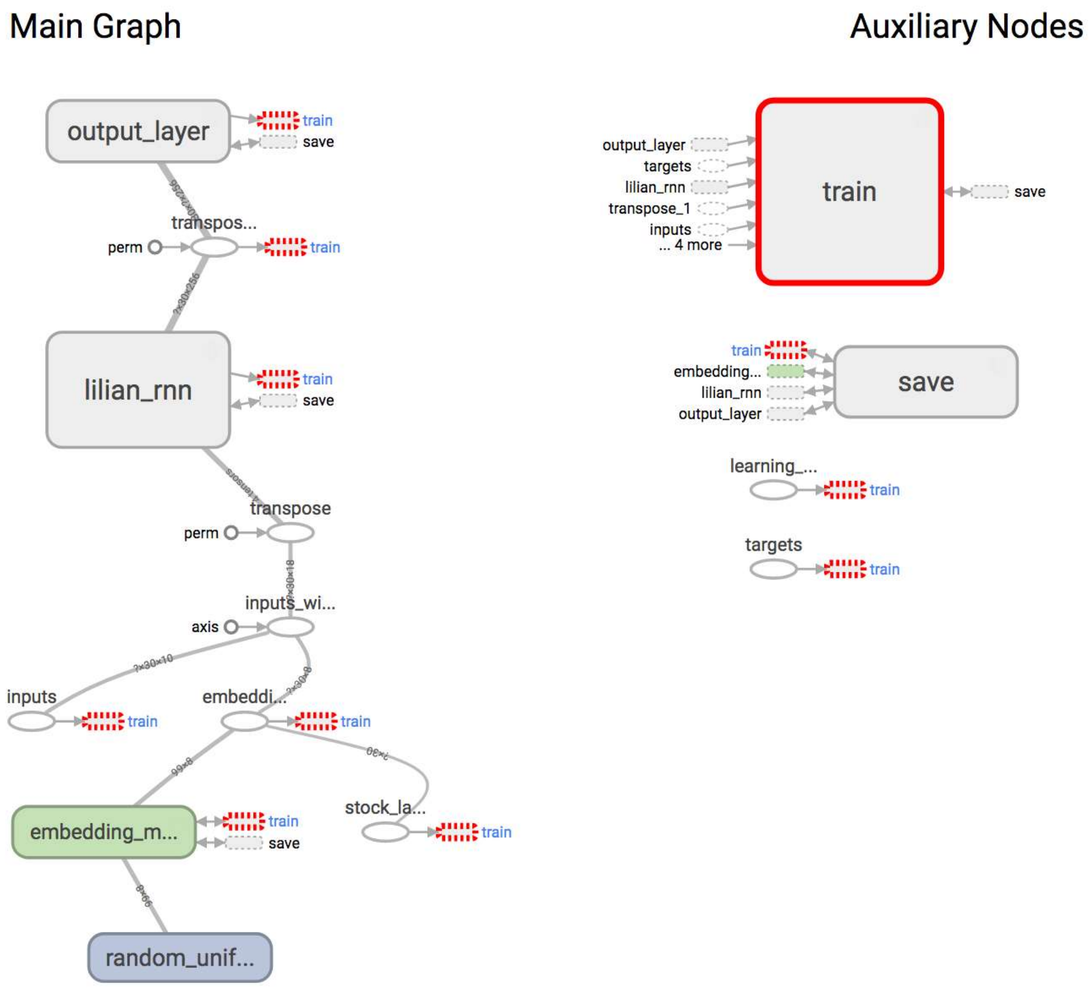


Fig. 2. Tensorboard visualization of the graph defined above. Two modules, "train" and "save", have been removed from the main graph.

Other than presenting the graph structure or tracking the variables in time, Tensorboard also supports **embeddings visualization**. In order to communicate the embedding values to Tensorboard, we need to add proper tracking in the training logs.

(0) In my embedding visualization, I want to color each stock with its industry sector. This metadata should stored in a csv file. The file has two columns, the stock symbol and the industry sector. It does not matter whether the csv file has header, but the order of the listed stocks must be consistent with `label_encoder.classes_` .

```
import csv
embedding_metadata_path = os.path.join(your_log_file_folder, 'metadata.csv')
with open(embedding_metadata_path, 'w') as fout:
    csv_writer = csv.writer(fout)
    # write the content into the csv file.
    # for example, csv_writer.writerows(["GOOG", "information_technology"])
```

(1) Set up the summary writer first within the training `tf.Session` .

```
from tensorflow.contrib.tensorboard.plugins import projector
with tf.Session(graph=lstm_graph) as sess:
    summary_writer = tf.summary.FileWriter(your_log_file_folder)
    summary_writer.add_graph(sess.graph)
```

(2) Add the tensor `embedding_matrix` defined in our graph `lstm_graph` into the projector config variable and attach the metadata csv file.

```
    projector_config = projector.ProjectorConfig()
    # You can add multiple embeddings. Here we add only one.
    added_embedding = projector_config.embeddings.add()
    added_embedding.tensor_name = embedding_matrix.name
    # Link this tensor to its metadata file.
    added_embedding.metadata_path = embedding_metadata_path
```

(3) This line creates a file `projector_config.pbtxt` in the folder `your_log_file_folder` . TensorBoard will read this file during startup.

```
    projector.visualize_embeddings(summary_writer, projector_config)
```

Results

The model is trained with top 50 stocks with largest market values in the S&P 500 index.

(Run the following command within github.com/lilianweng/stock-rnn)

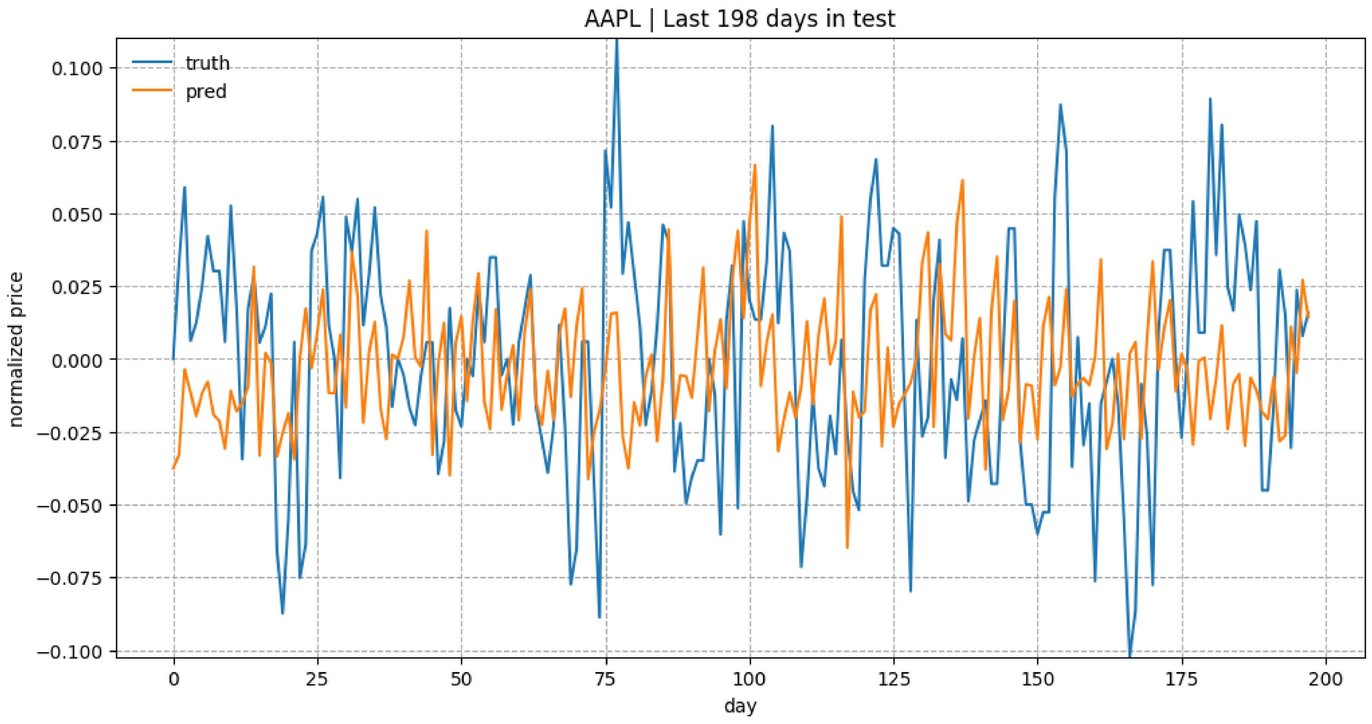
```
python main.py --stock_count=50 --embed_size=3 --input_size=3 --max_epoch=50 --train
```

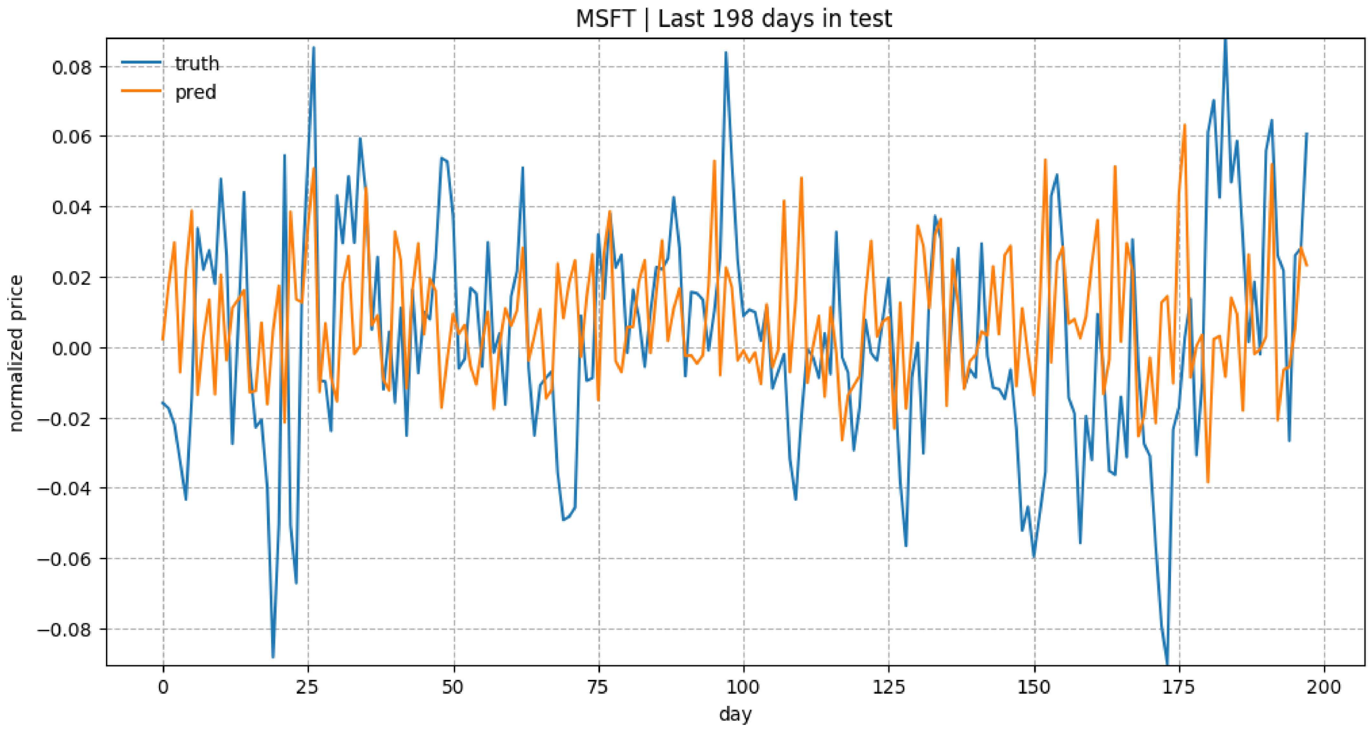
And the following configuration is used:

```
stock_count = 100
input_size = 3
embed_size = 3
num_steps = 30
lstm_size = 256
num_layers = 1
max_epoch = 50
keep_prob = 0.8
batch_size = 64
init_learning_rate = 0.05
learning_rate_decay = 0.99
init_epoch = 5
```

Price Prediction

As a brief overview of the prediction quality, Fig. 3 plots the predictions for test data of “KO”, “AAPL”, “GOOG” and “NFLX”. The overall trends matched up between the true values and the predictions. Considering how the prediction task is designed, the model relies on all the historical data points to predict only next 5 (`input_size`) days. With a small `input_size` , the model does not need to worry about the long-term growth curve. Once we increase `input_size` , the prediction would be much harder.


AAPL | Last 198 days in test
truth
pred
normalized price
day
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0
25
50
75
100
125
150
175
200


MSFT | Last 198 days in test
truth
pred
normalized price
day
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
0
25
50
75
100
125
150
175
200

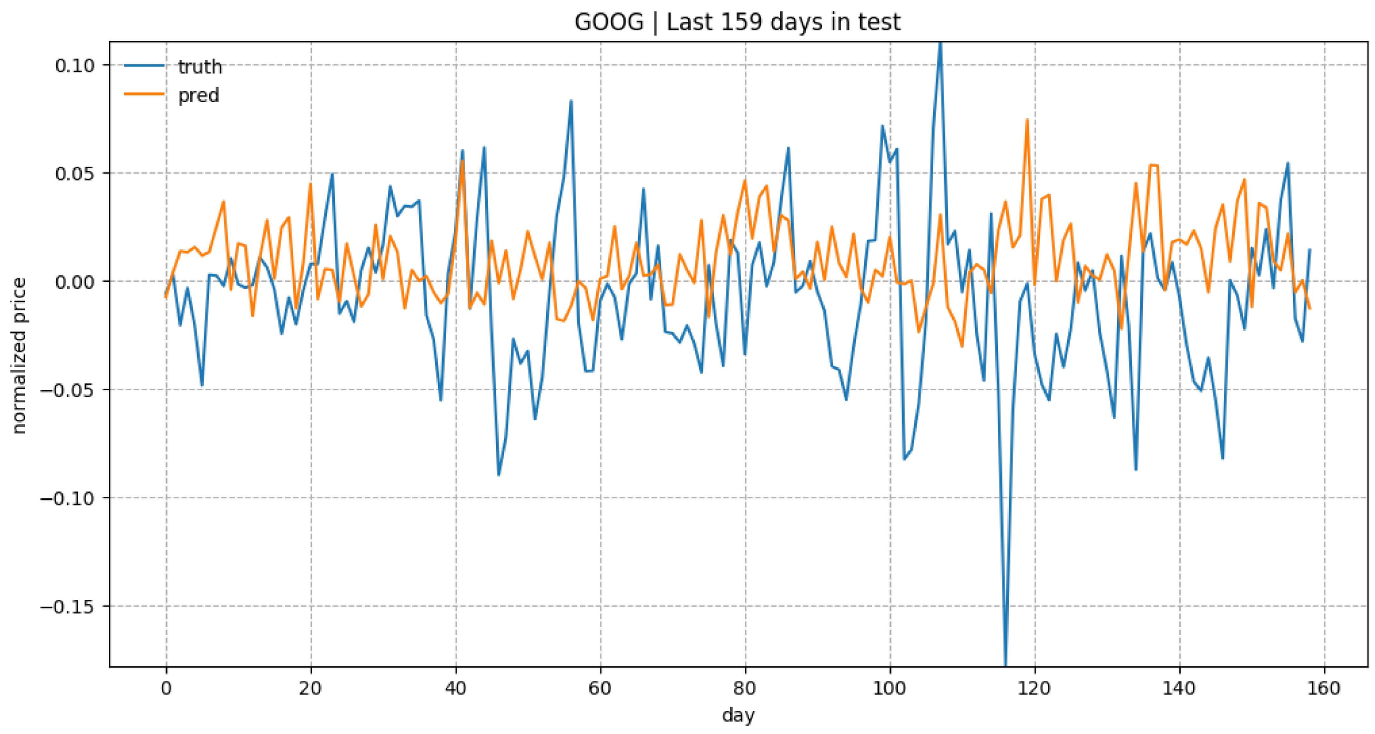


Fig. 3. True and predicted stock prices of AAPL, MSFT and GOOG in the test set. The prices are normalized across consecutive prediction sliding windows (See Part 1: Normalization. The y-axis values get multiplied by 5 for a better comparison between true and predicted trends.

Embedding Visualization

One common technique to visualize the clusters in embedding space is t-SNE (Maaten and Hinton, 2008), which is well supported in Tensorboard. t-SNE, short for "t-Distributed Stochastic Neighbor Embedding, is a variation of Stochastic Neighbor Embedding (Hinton and Roweis, 2002), but with a modified cost function that is easier to optimize.

1. Similar to SNE, t-SNE first converts the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities.
2. t-SNE defines a similar probability distribution over the data points in the low-dimensional space, and it minimizes the Kullback–Leibler divergence between the two distributions with respect to the locations of the points on the map.

Check this post for how to adjust the parameters, Perplexity and learning rate (epsilon), in t-SNE visualization.

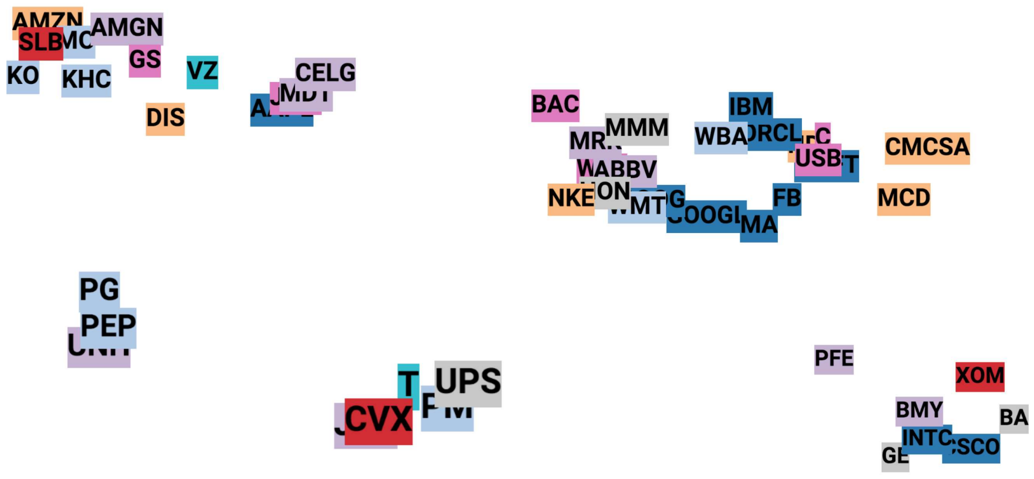
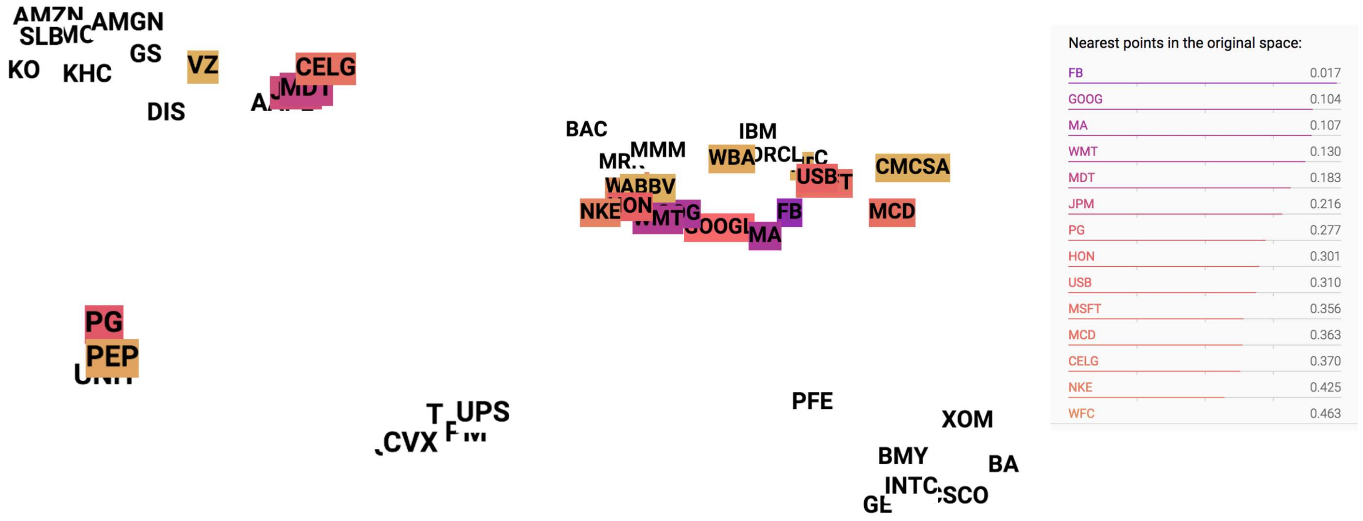


Fig. 4. Visualization of the stock embeddings using t-SNE. Each label is colored based on the stock industry sector. We have 5 clusters. Interstingly, GOOG, GOOGL and FB belong to the same cluster, while AMZN and AAPL stay in another.

In the embedding space, we can measure the similarity between two stocks by examining the similarity between their embedding vectors. For example, GOOG is mostly similar to GOOGL in the learned embeddings (See Fig. 5).



Nearest points in the original space:

FB	0.017
GOOG	0.104
MA	0.107
WMT	0.130
MDT	0.183
JPM	0.216
PG	0.277
HON	0.301
USB	0.310
MSFT	0.356
MCD	0.363
CELG	0.370
NKE	0.425
WFC	0.463

Fig. 5. "GOOG" is clicked in the embedding visualization graph and top 20 similar neighbors are highlighted with colors from dark to light as the similarity decreases.

Known Problems

- The prediction values get diminished and flatten quite a lot as the training goes. That's why I multiplied the absolute values by a constant to make the trend is more visible in Fig. 3., as I'm more curious about whether the prediction on the up-or-down direction right. However, there must be a reason for the diminishing prediction value problem. Potentially rather than using simple MSE as the loss, we can adopt another form of loss function to penalize more when the direction is predicted wrong.

- The loss function decreases fast at the beginning, but it suffers from occasional value explosion (a sudden peak happens and then goes back immediately). I suspect it is related to the form of loss function too. A updated and smarter loss function might be able to resolve the issue.

The full code in this tutorial is available in github.com/lilianweng/stock-rnn.

tutorial rnn tensorflow

«
How to Explain the Prediction of a Machine Learning Model?

»
Predict Stock Prices Using RNN: Part 1