

## 4.gru

September 29, 2021

```
[1]: import sys
import warnings

if not sys.warnoptions:
    warnings.simplefilter('ignore')
```

```
[2]: import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from datetime import datetime
from datetime import timedelta
from tqdm import tqdm
sns.set()
tf.compat.v1.random.set_random_seed(1234)
```

```
[3]: df = pd.read_csv('../dataset/G00G-year.csv')
df.head()
```

```
[3]:
```

|   | Date       | Open       | High       | Low        | Close      | Adj Close  | \ |
|---|------------|------------|------------|------------|------------|------------|---|
| 0 | 2016-11-02 | 778.200012 | 781.650024 | 763.450012 | 768.700012 | 768.700012 |   |
| 1 | 2016-11-03 | 767.250000 | 769.950012 | 759.030029 | 762.130005 | 762.130005 |   |
| 2 | 2016-11-04 | 750.659973 | 770.359985 | 750.560974 | 762.020020 | 762.020020 |   |
| 3 | 2016-11-07 | 774.500000 | 785.190002 | 772.549988 | 782.520020 | 782.520020 |   |
| 4 | 2016-11-08 | 783.400024 | 795.632996 | 780.190002 | 790.510010 | 790.510010 |   |

|   | Volume  |
|---|---------|
| 0 | 1872400 |
| 1 | 1943200 |
| 2 | 2134800 |
| 3 | 1585100 |
| 4 | 1350800 |

```
[4]: minmax = MinMaxScaler().fit(df.iloc[:, 4:5].astype('float32')) # Close index
df_log = minmax.transform(df.iloc[:, 4:5].astype('float32')) # Close index
```

```
df_log = pd.DataFrame(df_log)
df_log.head()
```

```
[4]:      0
0  0.112708
1  0.090008
2  0.089628
3  0.160459
4  0.188066
```

## 0.1 Split train and test

I will cut the dataset to train and test datasets,

1. Train dataset derived from starting timestamp until last 30 days
2. Test dataset derived from last 30 days until end of the dataset

So we will let the model do forecasting based on last 30 hours, and we will going to repeat the experiment for 10 times. You can increase it locally if you want, and tuning parameters will help you by a lot.

```
[5]: test_size = 30
simulation_size = 10

df_train = df_log.iloc[:-test_size]
df_test = df_log.iloc[-test_size:]
df.shape, df_train.shape, df_test.shape
```

```
[5]: ((252, 7), (222, 1), (30, 1))
```

```
[6]: class Model:
    def __init__(
        self,
        learning_rate,
        num_layers,
        size,
        size_layer,
        output_size,
        forget_bias = 0.1,
    ):
        def lstm_cell(size_layer):
            return tf.nn.rnn_cell.GRUCell(size_layer)

        rnn_cells = tf.nn.rnn_cell.MultiRNNCell(
            [lstm_cell(size_layer) for _ in range(num_layers)],
            state_is_tuple = False,
        )
        self.X = tf.placeholder(tf.float32, (None, None, size))
```

```

        self.Y = tf.placeholder(tf.float32, (None, output_size))
        drop = tf.contrib.rnn.DropoutWrapper(
            rnn_cells, output_keep_prob = forget_bias
        )
        self.hidden_layer = tf.placeholder(
            tf.float32, (None, num_layers * size_layer)
        )
        self.outputs, self.last_state = tf.nn.dynamic_rnn(
            drop, self.X, initial_state = self.hidden_layer, dtype = tf.float32
        )
        self.logits = tf.layers.dense(self.outputs[-1], output_size)
        self.cost = tf.reduce_mean(tf.square(self.Y - self.logits))
        self.optimizer = tf.train.AdamOptimizer(learning_rate).minimize(
            self.cost
        )

def calculate_accuracy(real, predict):
    real = np.array(real) + 1
    predict = np.array(predict) + 1
    percentage = 1 - np.sqrt(np.mean(np.square((real - predict) / real)))
    return percentage * 100

def anchor(signal, weight):
    buffer = []
    last = signal[0]
    for i in signal:
        smoothed_val = last * weight + (1 - weight) * i
        buffer.append(smoothed_val)
        last = smoothed_val
    return buffer

```

```

[7]: num_layers = 1
    size_layer = 128
    timestamp = 5
    epoch = 300
    dropout_rate = 0.8
    future_day = test_size
    learning_rate = 0.01

```

```

[8]: def forecast():
        tf.reset_default_graph()
        modelnn = Model(
            learning_rate, num_layers, df_log.shape[1], size_layer, df_log.
            ↪shape[1], dropout_rate
        )
        sess = tf.InteractiveSession()
        sess.run(tf.global_variables_initializer())

```

```

date_ori = pd.to_datetime(df.iloc[:, 0]).tolist()

pbar = tqdm(range(epoch), desc = 'train loop')
for i in pbar:
    init_value = np.zeros((1, num_layers * size_layer))
    total_loss, total_acc = [], []
    for k in range(0, df_train.shape[0] - 1, timestamp):
        index = min(k + timestamp, df_train.shape[0] - 1)
        batch_x = np.expand_dims(
            df_train.iloc[k : index, :].values, axis = 0
        )
        batch_y = df_train.iloc[k + 1 : index + 1, :].values
        logits, last_state, _, loss = sess.run(
            [modelnn.logits, modelnn.last_state, modelnn.optimizer, modelnn.
↪cost],
            feed_dict = {
                modelnn.X: batch_x,
                modelnn.Y: batch_y,
                modelnn.hidden_layer: init_value,
            },
        )
        init_value = last_state
        total_loss.append(loss)
        total_acc.append(calculate_accuracy(batch_y[:, 0], logits[:, 0]))
    pbar.set_postfix(cost = np.mean(total_loss), acc = np.mean(total_acc))

future_day = test_size

output_predict = np.zeros((df_train.shape[0] + future_day, df_train.
↪shape[1]))
output_predict[0] = df_train.iloc[0]
upper_b = (df_train.shape[0] // timestamp) * timestamp
init_value = np.zeros((1, num_layers * size_layer))

for k in range(0, (df_train.shape[0] // timestamp) * timestamp, timestamp):
    out_logits, last_state = sess.run(
        [modelnn.logits, modelnn.last_state],
        feed_dict = {
            modelnn.X: np.expand_dims(
                df_train.iloc[k : k + timestamp], axis = 0
            ),
            modelnn.hidden_layer: init_value,
        },
    )
    init_value = last_state
    output_predict[k + 1 : k + timestamp + 1] = out_logits

```

```

if upper_b != df_train.shape[0]:
    out_logits, last_state = sess.run(
        [modelnn.logits, modelnn.last_state],
        feed_dict = {
            modelnn.X: np.expand_dims(df_train.iloc[upper_b:], axis = 0),
            modelnn.hidden_layer: init_value,
        },
    )
    output_predict[upper_b + 1 : df_train.shape[0] + 1] = out_logits
    future_day -= 1
    date_ori.append(date_ori[-1] + timedelta(days = 1))

init_value = last_state

for i in range(future_day):
    o = output_predict[-future_day - timestamp + i:-future_day + i]
    out_logits, last_state = sess.run(
        [modelnn.logits, modelnn.last_state],
        feed_dict = {
            modelnn.X: np.expand_dims(o, axis = 0),
            modelnn.hidden_layer: init_value,
        },
    )
    init_value = last_state
    output_predict[-future_day + i] = out_logits[-1]
    date_ori.append(date_ori[-1] + timedelta(days = 1))

output_predict = minmax.inverse_transform(output_predict)
deep_future = anchor(output_predict[:, 0], 0.3)

return deep_future[-test_size:]

```

```

[9]: results = []
for i in range(simulation_size):
    print('simulation %d'%(i + 1))
    results.append(forecast())

```

WARNING: Logging before flag parsing goes to stderr.

W0811 22:46:29.978655 140681713489728 deprecation.py:323] From <ipython-input-6-1b755385b006>:12: GRUCell.\_\_init\_\_ (from tensorflow.python.ops.rnn\_cell\_impl) is deprecated and will be removed in a future version.

Instructions for updating:

This class is equivalent as tf.keras.layers.GRUCell, and will be replaced by that in Tensorflow 2.0.

W0811 22:46:29.981659 140681713489728 deprecation.py:323] From <ipython-input-6-1b755385b006>:16: MultiRNNCell.\_\_init\_\_ (from

tensorflow.python.ops.rnn\_cell\_impl) is deprecated and will be removed in a future version.

Instructions for updating:

This class is equivalent as tf.keras.layers.StackedRNNCells, and will be replaced by that in Tensorflow 2.0.

simulation 1

W0811 22:46:31.758260 140681713489728 lazy\_loader.py:50]

The TensorFlow contrib module will not be included in TensorFlow 2.0.

For more information, please see:

- \* <https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-sunset.md>

- \* <https://github.com/tensorflow/addons>

- \* <https://github.com/tensorflow/io> (for I/O related ops)

If you depend on functionality not listed there, please file an issue.

W0811 22:46:31.762153 140681713489728 deprecation.py:323] From <ipython-input-6-1b755385b006>:27: dynamic\_rnn (from tensorflow.python.ops.rnn) is deprecated and will be removed in a future version.

Instructions for updating:

Please use `keras.layers.RNN(cell)`, which is equivalent to this API

W0811 22:46:32.109607 140681713489728 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/init\_ops.py:1251: calling VarianceScaling.\_\_init\_\_ (from tensorflow.python.ops.init\_ops) with dtype is deprecated and will be removed in a future version.

Instructions for updating:

Call initializer instance with the dtype argument instead of passing it to the constructor

W0811 22:46:32.121295 140681713489728 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/rnn\_cell\_impl.py:564: calling Constant.\_\_init\_\_ (from tensorflow.python.ops.init\_ops) with dtype is deprecated and will be removed in a future version.

Instructions for updating:

Call initializer instance with the dtype argument instead of passing it to the constructor

W0811 22:46:32.136707 140681713489728 deprecation.py:506] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/rnn\_cell\_impl.py:574: calling Zeros.\_\_init\_\_ (from tensorflow.python.ops.init\_ops) with dtype is deprecated and will be removed in a future version.

Instructions for updating:

Call initializer instance with the dtype argument instead of passing it to the constructor

W0811 22:46:32.395149 140681713489728 deprecation.py:323] From <ipython-input-6-1b755385b006>:29: dense (from tensorflow.python.layers.core) is deprecated and will be removed in a future version.

Instructions for updating:

Use keras.layers.dense instead.

```
train loop: 100%|      | 300/300 [02:10<00:00,  2.45it/s, acc=97.1,  
cost=0.00211]
```

simulation 2

```
train loop: 100%|      | 300/300 [02:09<00:00,  2.48it/s, acc=96.2,  
cost=0.00432]
```

simulation 3

```
train loop: 100%|      | 300/300 [02:08<00:00,  1.88it/s, acc=96.7,  
cost=0.00239]
```

simulation 4

```
train loop: 100%|      | 300/300 [02:05<00:00,  2.12it/s, acc=96.4,  
cost=0.00307]
```

simulation 5

```
train loop: 100%|      | 300/300 [02:12<00:00,  1.95it/s, acc=96.9,  
cost=0.00227]
```

simulation 6

```
train loop: 100%|      | 300/300 [02:15<00:00,  2.64it/s, acc=96.4,  
cost=0.00318]
```

simulation 7

```
train loop: 100%|      | 300/300 [02:10<00:00,  2.59it/s, acc=96.9,  
cost=0.00256]
```

simulation 8

```
train loop: 100%|      | 300/300 [02:11<00:00,  2.19it/s, acc=97.2,  
cost=0.00188]
```

simulation 9

```
train loop: 100%|      | 300/300 [02:14<00:00,  2.48it/s, acc=96.9,  
cost=0.00226]
```

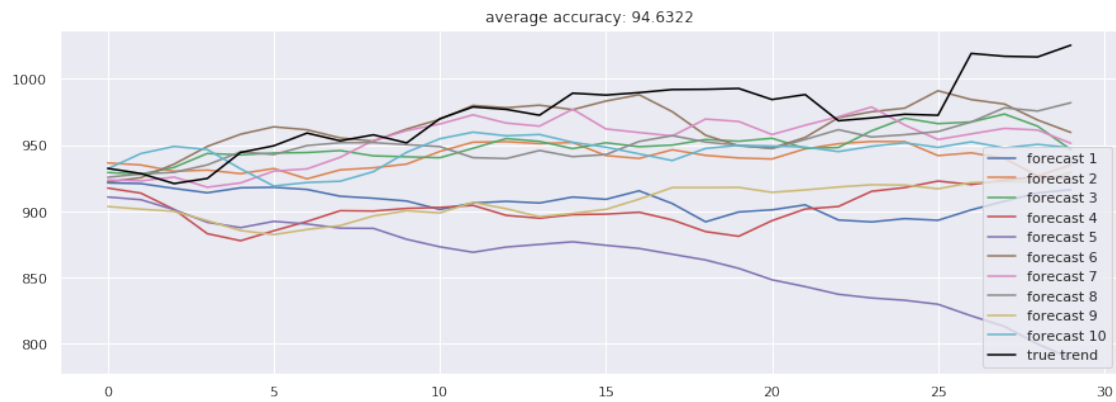
simulation 10

```
train loop: 100%|      | 300/300 [02:06<00:00,  2.48it/s, acc=97.5,  
cost=0.00167]
```

```
[10]: accuracies = [calculate_accuracy(df['Close'].iloc[-test_size:].values, r) for r_u  
    ↪in results]

plt.figure(figsize = (15, 5))
for no, r in enumerate(results):
    plt.plot(r, label = 'forecast %d'%(no + 1))
```

```
plt.plot(df['Close'].iloc[-test_size:].values, label = 'true trend', c = 'black')
plt.legend()
plt.title('average accuracy: %.4f'%(np.mean(accuracies)))
plt.show()
```



[ ]: