9.vanilla-2path

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/GOOG-year.csv')
df.head()
```

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-11-02	778.200012	781.650024	763.450012	768.700012	768.700012	1872400
1	2016-11-03	767.250000	769.950012	759.030029	762.130005	762.130005	1943200
2	2016-11-04	750.659973	770.359985	750.560974	762.020020	762.020020	2134800
3	2016-11-07	774.500000	785.190002	772.549988	782.520020	782.520020	1585100
4	2016-11-08	783.400024	795.632996	780.190002	790.510010	790.510010	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 days, 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.BasicRNNCell(size_layer)

        with tf.variable_scope('forward', reuse = False):
            rnn_cells_forward = tf.nn.rnn_cell.MultiRNNCell(
                [lstm_cell(size_layer) for _ in range(num_layers)],
                state_is_tuple = False,
            )
```

```
            self.X_forward = tf.placeholder(tf.float32, (None, None, size))
            drop_forward = tf.contrib.rnn.DropoutWrapper(
                rnn_cells_forward, output_keep_prob = forget_bias
            )
            self.hidden_layer_forward = tf.placeholder(
                tf.float32, (None, num_layers * size_layer)
            )
            self.outputs_forward, self.last_state_forward = tf.nn.dynamic_rnn(
                drop_forward,
                self.X_forward,
                initial_state = self.hidden_layer_forward,
                dtype = tf.float32,
            )

        with tf.variable_scope('backward', reuse = False):
            rnn_cells_backward = tf.nn.rnn_cell.MultiRNNCell(
                [lstm_cell(size_layer) for _ in range(num_layers)],
                state_is_tuple = False,
            )
            self.X_backward = tf.placeholder(tf.float32, (None, None, size))
            drop_backward = tf.contrib.rnn.DropoutWrapper(
                rnn_cells_backward, output_keep_prob = forget_bias
            )
            self.hidden_layer_backward = tf.placeholder(
                tf.float32, (None, num_layers * size_layer)
            )
            self.outputs_backward, self.last_state_backward = tf.nn.dynamic_rnn(
                drop_backward,
                self.X_backward,
                initial_state = self.hidden_layer_backward,
                dtype = tf.float32,
            )

        self.outputs = self.outputs_backward - self.outputs_forward
        self.Y = tf.placeholder(tf.float32, (None, output_size))
        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_forward = np.zeros((1, num_layers * size_layer))
        init_value_backward = 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_forward = np.expand_dims(
                df_train.iloc[k : index, :].values, axis = 0
            )
            batch_x_backward = np.expand_dims(
                np.flip(df_train.iloc[k : index, :].values, axis = 0), axis = 0
            )
            batch_y = df_train.iloc[k + 1 : index + 1, :].values
            logits, last_state_forward, last_state_backward, _, loss = sess.run(
                [
                    modelnn.logits,
                    modelnn.last_state_forward,
                    modelnn.last_state_backward,
                    modelnn.optimizer,
```

```
                    modelnn.cost,
                ],
                feed_dict = {
                    modelnn.X_forward: batch_x_forward,
                    modelnn.X_backward: batch_x_backward,
                    modelnn.Y: batch_y,
                    modelnn.hidden_layer_forward: init_value_forward,
                    modelnn.hidden_layer_backward: init_value_backward,
                },
            )
            init_value_forward = last_state_forward
            init_value_backward = last_state_backward
            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_forward = np.zeros((1, num_layers * size_layer))
    init_value_backward = np.zeros((1, num_layers * size_layer))

    for k in range(0, (df_train.shape[0] // timestamp) * timestamp, timestamp):
        batch_x_forward = np.expand_dims(
        df_train.iloc[k : k + timestamp, :], axis = 0
        )
        batch_x_backward = np.expand_dims(
            np.flip(df_train.iloc[k : k + timestamp, :].values, axis = 0), axis␣
↪= 0
        )
        out_logits, last_state_forward, last_state_backward = sess.run(
            [
                modelnn.logits,
                modelnn.last_state_forward,
                modelnn.last_state_backward,
            ],
            feed_dict = {
                modelnn.X_forward: batch_x_forward,
                modelnn.X_backward: batch_x_backward,
                modelnn.hidden_layer_forward: init_value_forward,
                modelnn.hidden_layer_backward: init_value_backward,
            },
        )
        init_value_forward = last_state_forward
```

```
        init_value_backward = last_state_backward
        output_predict[k + 1 : k + timestamp + 1, :] = out_logits

    if upper_b != df_train.shape[0]:
        batch_x_forward = np.expand_dims(df_train.iloc[upper_b:, :], axis = 0)
        batch_x_backward = np.expand_dims(
            np.flip(df_train.iloc[upper_b:, :].values, axis = 0), axis = 0
        )
        out_logits, last_state_forward, last_state_backward = sess.run(
            [modelnn.logits, modelnn.last_state_forward, modelnn.
↪last_state_backward],
            feed_dict = {
                modelnn.X_forward: batch_x_forward,
                modelnn.X_backward: batch_x_backward,
                modelnn.hidden_layer_forward: init_value_forward,
                modelnn.hidden_layer_backward: init_value_backward,
            },
        )
        init_value_forward = last_state_forward
        init_value_backward = last_state_backward
        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_forward = last_state_forward
    init_value_backward = last_state_backward

    for i in range(future_day):
        o = output_predict[-future_day - timestamp + i:-future_day + i]
        o_f = np.flip(o, axis = 0)
        out_logits, last_state_forward, last_state_backward = sess.run(
            [
                modelnn.logits,
                modelnn.last_state_forward,
                modelnn.last_state_backward,
            ],
            feed_dict = {
                modelnn.X_forward: np.expand_dims(o, axis = 0),
                modelnn.X_backward: np.expand_dims(o_f, axis = 0),
                modelnn.hidden_layer_forward: init_value_forward,
                modelnn.hidden_layer_backward: init_value_backward,
            },
        )
        init_value_forward = last_state_forward
        init_value_backward = last_state_backward
        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.
W0813 01:22:40.361487 140690772289344 deprecation.py:323] From <ipython-
input-6-04b2b1d463f4>:12: BasicRNNCell.__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.SimpleRNNCell, and will be replaced
by that in Tensorflow 2.0.
W0813 01:22:40.364087 140690772289344 deprecation.py:323] From <ipython-
input-6-04b2b1d463f4>:17: 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

W0813 01:22:40.688215 140690772289344 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.

W0813 01:22:40.691791 140690772289344 deprecation.py:323] From <ipython-
input-6-04b2b1d463f4>:30: 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
W0813 01:22:40.883351 140690772289344 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
W0813 01:22:40.890208 140690772289344 deprecation.py:506] From
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/rnn_cell_impl.py:459: 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
W0813 01:22:41.052329 140690772289344 deprecation.py:323] From <ipython-
input-6-04b2b1d463f4>:54: 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 [01:08<00:00,  4.43it/s, acc=73.8,
cost=0.155]

simulation 2

train loop: 100%|      | 300/300 [01:09<00:00,  4.32it/s, acc=75,
cost=0.151]

simulation 3

train loop: 100%|      | 300/300 [01:08<00:00,  4.41it/s, acc=75.2,
cost=0.14]

simulation 4

train loop: 100%|      | 300/300 [01:08<00:00,  4.36it/s, acc=71,
cost=0.186]

simulation 5

train loop: 100%|      | 300/300 [01:09<00:00,  4.33it/s, acc=84,
cost=0.0574]

simulation 6

train loop: 100%|      | 300/300 [01:09<00:00,  4.34it/s, acc=72.3,
cost=0.166]

simulation 7

train loop: 100%|      | 300/300 [01:08<00:00,  4.44it/s, acc=80.4,
cost=0.0918]

simulation 8

train loop: 100%|      | 300/300 [01:07<00:00,  4.45it/s, acc=79.7,
cost=0.101]

simulation 9
```

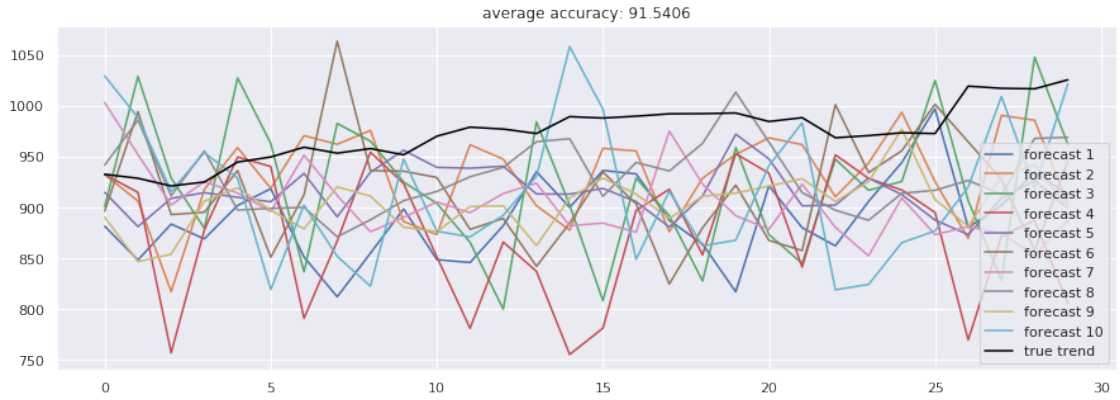
```
train loop: 100%|      | 300/300 [01:05<00:00,  4.61it/s, acc=80.6,
cost=0.088]

simulation 10

train loop: 100%|      | 300/300 [01:08<00:00,  4.40it/s, acc=70.8,
cost=0.194]
```

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



[ ]: