11.bidirectional-lstm-seq2seq

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))
```

[10]:
```
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.LSTMCell(size_layer, state_is_tuple = False)

        backward_rnn_cells = tf.nn.rnn_cell.MultiRNNCell(
            [lstm_cell(size_layer) for _ in range(num_layers)],
            state_is_tuple = False,
        )
        forward_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_backward = tf.contrib.rnn.DropoutWrapper(
            backward_rnn_cells, output_keep_prob = forget_bias
        )
        forward_backward = tf.contrib.rnn.DropoutWrapper(
            forward_rnn_cells, output_keep_prob = forget_bias
        )
        self.backward_hidden_layer = tf.placeholder(
            tf.float32, shape = (None, num_layers * 2 * size_layer)
        )
        self.forward_hidden_layer = tf.placeholder(
            tf.float32, shape = (None, num_layers * 2 * size_layer)
        )
        _, last_state = tf.nn.bidirectional_dynamic_rnn(
            forward_backward,
            drop_backward,
            self.X,
            initial_state_fw = self.forward_hidden_layer,
            initial_state_bw = self.backward_hidden_layer,
            dtype = tf.float32,
        )

        with tf.variable_scope('decoder', reuse = False):
            backward_rnn_cells_decoder = tf.nn.rnn_cell.MultiRNNCell(
            [lstm_cell(size_layer) for _ in range(num_layers)],
            state_is_tuple = False,
            )
            forward_rnn_cells_decoder = tf.nn.rnn_cell.MultiRNNCell(
                [lstm_cell(size_layer) for _ in range(num_layers)],
                state_is_tuple = False,
            )
            drop_backward_decoder = tf.contrib.rnn.DropoutWrapper(
            backward_rnn_cells_decoder, output_keep_prob = forget_bias
            )
            forward_backward_decoder = tf.contrib.rnn.DropoutWrapper(
                forward_rnn_cells_decoder, output_keep_prob = forget_bias
            )
            self.outputs, self.last_state = tf.nn.bidirectional_dynamic_rnn(
                forward_backward_decoder, drop_backward_decoder, self.X,
                initial_state_fw = last_state[0],
                initial_state_bw = last_state[1],
                dtype = tf.float32
            )
```

```
        self.outputs = tf.concat(self.outputs, 2)
        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 * 2 * size_layer))
        init_value_backward = np.zeros((1, num_layers * 2 * 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.backward_hidden_layer: init_value_backward,
                    modelnn.forward_hidden_layer: init_value_forward,
                },
            )
            init_value_forward = last_state[0]
            init_value_backward = last_state[1]
            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 * 2 * size_layer))
    init_value_backward = np.zeros((1, num_layers * 2 * 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.backward_hidden_layer: init_value_backward,
                modelnn.forward_hidden_layer: init_value_forward,
            },
        )
        init_value_forward = last_state[0]
        init_value_backward = last_state[1]
        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.backward_hidden_layer: init_value_backward,
                modelnn.forward_hidden_layer: init_value_forward,
            },
        )
        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[0]
    init_value_backward = last_state[1]

    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.backward_hidden_layer: init_value_backward,
                modelnn.forward_hidden_layer: init_value_forward,
            },
        )
        init_value_forward = last_state[0]
        init_value_backward = last_state[1]
        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:]
```

[11]:
```
results = []
for i in range(simulation_size):
    print('simulation %d'%(i + 1))
    results.append(forecast())
```

```
W0813 22:30:03.664880 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c6a29f9e8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:30:03.666436 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c6a0dba58>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:30:03.827417 140106178451264 rnn_cell_impl.py:893]
```

```
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c6a0db5f8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:30:03.828239 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c6a0dbe48>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 1

W0813 22:30:03.988492 140106178451264 deprecation.py:323] From <ipython-
input-10-79385dfa86b9>:67: 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 [02:29<00:00,  2.03it/s, acc=96.4,
cost=0.00318]
W0813 22:32:35.430384 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c68a2de10>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:32:35.431268 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c689d96a0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:32:35.592414 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c689d91d0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:32:35.593283 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c680bc630>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 2

train loop: 100%|      | 300/300 [02:30<00:00,  2.00it/s, acc=98.1,
cost=0.000912]
W0813 22:35:08.073616 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c68a6df28>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:35:08.074523 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c40475208>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:35:08.237059 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bbbf9afd0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
W0813 22:35:08.237945 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bbbf56fd0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 3

train loop: 100%|      | 300/300 [02:31<00:00,  1.99it/s, acc=98.2,
cost=0.000814]
W0813 22:37:40.822348 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6c40403080>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:37:40.823194 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bba08cac8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:37:40.984025 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb9234278>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:37:40.984853 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb90d59e8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 4

train loop: 100%|      | 300/300 [02:31<00:00,  1.98it/s, acc=98.2,
cost=0.000732]
W0813 22:40:14.461846 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb230a7f0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:40:14.462596 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb31e4588>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:40:14.624744 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb2361198>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:40:14.625597 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb223d5c0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 5

train loop: 100%|      | 300/300 [02:28<00:00,  2.04it/s, acc=98.6,
cost=0.000437]
```

```
W0813 22:42:44.319911 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6baf458dd8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:42:44.320685 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb0355400>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:42:44.481708 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6baf534d30>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:42:44.482524 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6baf3f2c88>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 6
```

```
train loop: 100%|     | 300/300 [02:30<00:00,  2.00it/s, acc=98.8,
cost=0.000304]
W0813 22:45:16.281273 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bb04d4240>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:45:16.282183 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bad522c18>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:45:16.443265 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bad522940>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:45:16.444107 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6bac5314e0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

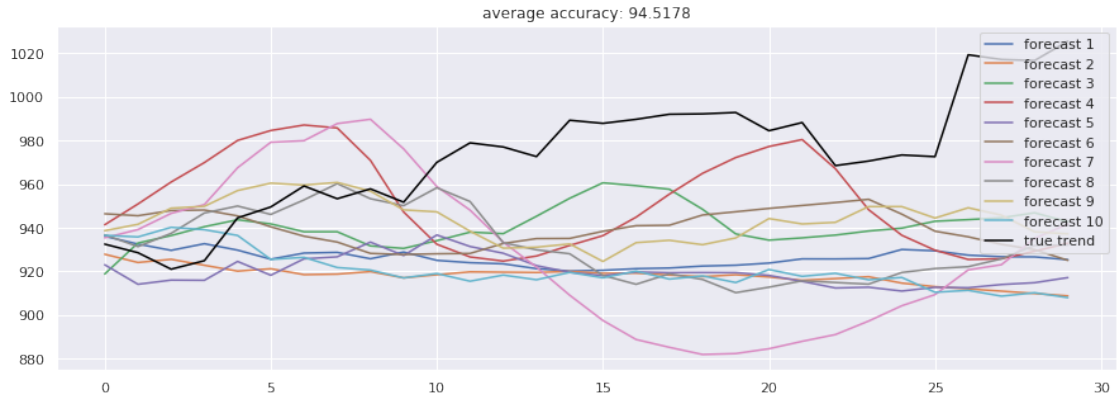
```
simulation 7
```

```
train loop: 100%|     | 300/300 [02:31<00:00,  1.99it/s, acc=98.5,
cost=0.000517]
W0813 22:47:49.574930 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba974ef28>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:47:49.575763 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6baa69bc18>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
W0813 22:47:49.735839 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba9794a20>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:47:49.736666 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba7e914a8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 8

train loop: 100%|     | 300/300 [02:31<00:00,  1.99it/s, acc=96.9,
cost=0.00272]
W0813 22:50:22.981087 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba69149b0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:50:22.982089 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba7828080>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:50:23.141866 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba69434e0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:50:23.142687 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba68067b8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.

simulation 9

train loop: 100%|     | 300/300 [02:31<00:00,  1.99it/s, acc=98.2,
cost=0.000705]
W0813 22:52:55.940449 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba3aacbe0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:52:55.941309 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba49a4080>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:52:56.101360 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba21a14e0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0813 22:52:56.102182 140106178451264 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f6ba39be898>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 10

train loop: 100%|      | 300/300 [02:27<00:00,  2.03it/s, acc=98.6,
cost=0.000453]
```

[12]:
```
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()
```



[ ]: