2.bidirectional-lstm

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  \
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 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.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(
```

```python
            [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)
        )
        self.outputs, self.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,
        )
        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,
            },
```

```python
        )
        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:]
```

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

```
WARNING: Logging before flag parsing goes to stderr.
W0812 10:20:04.613218 140016646534976 deprecation.py:323] From <ipython-
input-6-32a8ad1d5669>:12: LSTMCell.__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.LSTMCell, and will be replaced by
that in Tensorflow 2.0.
W0812 10:20:04.617547 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f579b74fd68>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:20:04.620435 140016646534976 deprecation.py:323] From <ipython-
input-6-32a8ad1d5669>: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.
W0812 10:20:04.623959 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f579b6efa20>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 1
```

```
W0812 10:20:04.949644 140016646534976 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.

W0812 10:20:04.954938 140016646534976 deprecation.py:323] From <ipython-
input-6-32a8ad1d5669>:42: bidirectional_dynamic_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.Bidirectional(keras.layers.RNN(cell))`, which is
equivalent to this API
W0812 10:20:04.955546 140016646534976 deprecation.py:323] From
/usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/rnn.py:464:
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
W0812 10:20:05.149145 140016646534976 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
W0812 10:20:05.156026 140016646534976 deprecation.py:506] From
/usr/local/lib/python3.6/dist-
packages/tensorflow/python/ops/rnn_cell_impl.py:961: 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
W0812 10:20:05.712592 140016646534976 deprecation.py:323] From <ipython-
input-6-32a8ad1d5669>:45: 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 [01:39<00:00,  3.04it/s, acc=97.8,
cost=0.00113]
W0812 10:21:46.695034 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f58216a3208>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:21:46.695935 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f5790ef9e10>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 2
```

```
train loop: 100%|     | 300/300 [01:40<00:00,  3.03it/s, acc=97.1,
cost=0.00187]
W0812 10:23:27.984155 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f57919d69e8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:23:27.985092 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f57008a7b70>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 3
```

```
train loop: 100%|     | 300/300 [01:40<00:00,  2.97it/s, acc=97.8,
cost=0.00118]
W0812 10:26:50.307250 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f5791f0d2e8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:26:50.308161 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56dbd71160>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 5
```

```
train loop: 100%|     | 300/300 [01:40<00:00,  2.97it/s, acc=97.1,
cost=0.00237]
W0812 10:28:31.638492 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56dbe750b8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:28:31.639337 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d982db38>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 6
```

```
train loop: 100%|     | 300/300 [01:37<00:00,  3.11it/s, acc=97.5,
cost=0.00143]
W0812 10:30:09.934609 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d99130b8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:30:09.935530 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d7b95320>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 7
```

```
train loop: 100%|      | 300/300 [01:39<00:00,  3.01it/s, acc=97.4,
cost=0.00163]
W0812 10:31:50.447502 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d7384cf8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:31:50.448328 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d56a2748>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 8
```

```
train loop: 100%|      | 300/300 [01:38<00:00,  3.05it/s, acc=96.6,
cost=0.00322]
W0812 10:33:30.276075 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d4e9bba8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:33:30.276944 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d2868da0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 9
```

```
train loop: 100%|      | 300/300 [01:38<00:00,  3.07it/s, acc=97.7,
cost=0.00133]
W0812 10:35:09.746517 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d290ccf8>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
W0812 10:35:09.747369 140016646534976 rnn_cell_impl.py:893]
<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at 0x7f56d03129b0>: Using a
concatenated state is slower and will soon be deprecated.  Use
state_is_tuple=True.
```

```
simulation 10
```

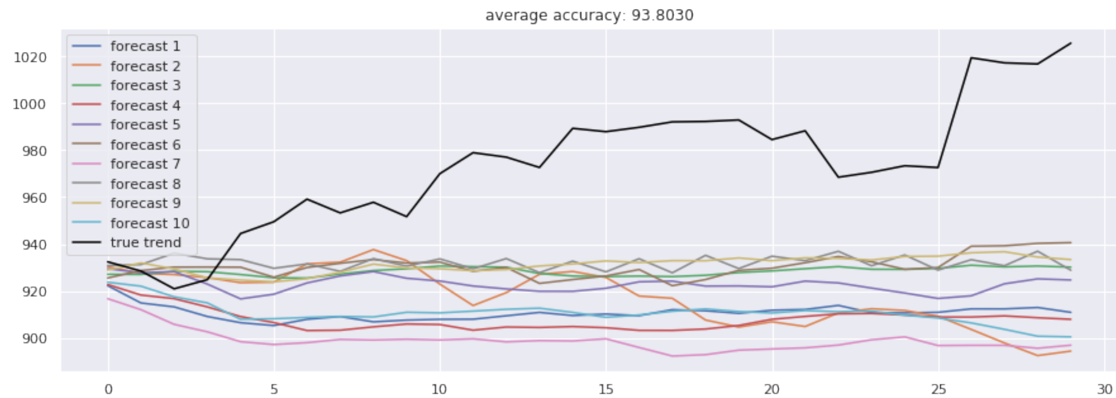
```
train loop: 100%|      | 300/300 [01:39<00:00,  3.03it/s, acc=97.5,
cost=0.00142]
```

[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()
```




```
[ ]:
```