tf.1_LSTM_vr.3

February 3, 2021

1 필요한 모듈을 가져오고 데이터를 로드합니다

[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]:
```
# 예측할 종목은 한양증권(001750) 입니다

df = pd.read_csv('C:\Jupyter_Project\HanyangSecurities.csv')
df.head()
```

[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-01-29	8740	9120	8740	9120	9120	96112
1	2020-01-30	9350	9570	9200	9370	9370	175760
2	2020-01-31	9380	9710	9380	9570	9570	226732
3	2020-02-03	9440	9440	9030	9140	9140	99485
4	2020-02-04	9140	9150	9020	9110	9110	30886

2 Normalization을 진행합니다

[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  0.793277
1  0.835294
2  0.868908
3  0.796639
4  0.791597
```

3 데이터를 분할하여 훈련 데이터를 생성합니다

[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))
```

4 딥러닝 네트워크를 학습시킵니다

[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)

        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 * 2 * 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 * 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.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 * 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.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())
```

```
simulation 1
WARNING:tensorflow:From <ipython-input-6-d01d21f09afe>: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.
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x0000014232E560C8>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:From <ipython-input-6-d01d21f09afe>: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.
WARNING:tensorflow:
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.

WARNING:tensorflow:From <ipython-input-6-d01d21f09afe>: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
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\tf-1\lib\site-
packages\tensorflow_core\python\ops\rnn_cell_impl.py:958: Layer.add_variable
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\tf-1\lib\site-
packages\tensorflow_core\python\ops\rnn_cell_impl.py:962: 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
WARNING:tensorflow:From <ipython-input-6-d01d21f09afe>: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.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\envs\tf-1\lib\site-
packages\tensorflow_core\python\layers\core.py:187: Layer.apply (from
tensorflow.python.keras.engine.base_layer) is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `layer.__call__` method instead.

train loop: 100%|| 300/300
[00:39<00:00,  7.55it/s, acc=97, cost=0.00313]

simulation 2
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
```

```
0x000001423242C908>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:39<00:00,  7.57it/s, acc=96.9, cost=0.0026]

simulation 3
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x0000014236F57748>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:39<00:00,  7.55it/s, acc=97.8, cost=0.00135]

simulation 4
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x00000142381BD848>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:40<00:00,  7.45it/s, acc=97.3, cost=0.00224]

simulation 5
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x00000142380A21C8>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.29it/s, acc=97.4, cost=0.00226]

simulation 6
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x00000142381E2608>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.28it/s, acc=97.4, cost=0.00197]

simulation 7
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x0000014239587B48>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.16it/s, acc=97.8, cost=0.00143]

simulation 8
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x000001423A8EEE88>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.21it/s, acc=97.3, cost=0.0023]
```

```
simulation 9
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x000001423A71EF08>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.19it/s, acc=96.8, cost=0.00289]

simulation 10
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x000001423CC89248>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.

train loop: 100%|| 300/300
[00:41<00:00,  7.19it/s, acc=97.1, cost=0.00253]
```

5 Visualize

vr.1,2 모델과는 다른 예측기법이 적용되었습니다. 실값(true trend)과 10개의 예측모델(forcast 1~10)이 그래프 상에 나열되어 있기 때문에 차트를 추종하는지의 여부는 확인하기 어렵지만 model loss만을 측정하는 앞선 두 모델과는 달리 average accurracy(정확도)를 측정하는데에 의의가 있다고 생각합니다

[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()
```