18.dilated-cnn-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  \
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]: def position_encoding(inputs):
    T = tf.shape(inputs)[1]
    repr_dim = inputs.get_shape()[-1].value
    pos = tf.reshape(tf.range(0.0, tf.to_float(T), dtype=tf.float32), [-1, 1])
    i = np.arange(0, repr_dim, 2, np.float32)
    denom = np.reshape(np.power(10000.0, i / repr_dim), [1, -1])
    enc = tf.expand_dims(tf.concat([tf.sin(pos / denom), tf.cos(pos / denom)],␣
 ↪1), 0)
    return tf.tile(enc, [tf.shape(inputs)[0], 1, 1])

def layer_norm(inputs, epsilon=1e-8):
    mean, variance = tf.nn.moments(inputs, [-1], keep_dims=True)
    normalized = (inputs - mean) / (tf.sqrt(variance + epsilon))
    params_shape = inputs.get_shape()[-1:]
    gamma = tf.get_variable('gamma', params_shape, tf.float32, tf.
 ↪ones_initializer())
    beta = tf.get_variable('beta', params_shape, tf.float32, tf.
 ↪zeros_initializer())
```

```
    return gamma * normalized + beta

def cnn_block(x, dilation_rate, pad_sz, hidden_dim, kernel_size):
    x = layer_norm(x)
    pad = tf.zeros([tf.shape(x)[0], pad_sz, hidden_dim])
    x =  tf.layers.conv1d(inputs = tf.concat([pad, x, pad], 1),
                          filters = hidden_dim,
                          kernel_size = kernel_size,
                          dilation_rate = dilation_rate)
    x = x[:, :-pad_sz, :]
    x = tf.nn.relu(x)
    return x

class Model:
    def __init__(
        self,
        learning_rate,
        num_layers,
        size,
        size_layer,
        output_size,
        kernel_size = 3,
        n_attn_heads = 16,
        dropout = 0.9,
    ):
        self.X = tf.placeholder(tf.float32, (None, None, size))
        self.Y = tf.placeholder(tf.float32, (None, output_size))

        encoder_embedded = tf.layers.dense(self.X, size_layer)
        encoder_embedded += position_encoding(encoder_embedded)

        e = tf.identity(encoder_embedded)
        for i in range(num_layers):
            dilation_rate = 2 ** i
            pad_sz = (kernel_size - 1) * dilation_rate
            with tf.variable_scope('block_%d'%i):
                encoder_embedded += cnn_block(encoder_embedded, dilation_rate,
                                              pad_sz, size_layer, kernel_size)

        encoder_output, output_memory = encoder_embedded, encoder_embedded + e
        g = tf.identity(encoder_embedded)

        for i in range(num_layers):
            dilation_rate = 2 ** i
            pad_sz = (kernel_size - 1) * dilation_rate
            with tf.variable_scope('decode_%d'%i):
                attn_res = h = cnn_block(encoder_embedded, dilation_rate,
```

```
                                         pad_sz, size_layer, kernel_size)

            C = []
            for j in range(n_attn_heads):
                h_ = tf.layers.dense(h, size_layer // n_attn_heads)
                g_ = tf.layers.dense(g, size_layer // n_attn_heads)
                zu_ = tf.layers.dense(
                    encoder_output, size_layer // n_attn_heads
                )
                ze_ = tf.layers.dense(output_memory, size_layer // n_attn_heads)

                d = tf.layers.dense(h_, size_layer // n_attn_heads) + g_
                dz = tf.matmul(d, tf.transpose(zu_, [0, 2, 1]))
                a = tf.nn.softmax(dz)
                c_ = tf.matmul(a, ze_)
                C.append(c_)

            c = tf.concat(C, 2)
            h = tf.layers.dense(attn_res + c, size_layer)
            h = tf.nn.dropout(h, keep_prob = dropout)
            encoder_embedded += h

        encoder_embedded = tf.sigmoid(encoder_embedded[-1])
        self.logits = tf.layers.dense(encoder_embedded, 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 = test_size
```

```
epoch = 300
dropout_rate = 0.8
future_day = test_size
learning_rate = 5e-4
```

[8]:
```
def forecast():
    tf.reset_default_graph()
    modelnn = Model(
        learning_rate, num_layers, df_log.shape[1], size_layer, df_log.shape[1],
        dropout = 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, _, loss = sess.run(
                [modelnn.logits, modelnn.optimizer, modelnn.cost],
                feed_dict = {modelnn.X: batch_x, modelnn.Y: batch_y},
            )
            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

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

```
            },
        )
        output_predict[k + 1 : k + timestamp + 1] = out_logits

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

    for i in range(future_day):
        o = output_predict[-future_day - timestamp + i:-future_day + i]
        out_logits = sess.run(
            modelnn.logits,
            feed_dict = {
                modelnn.X: np.expand_dims(o, axis = 0)
            },
        )
        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())
```

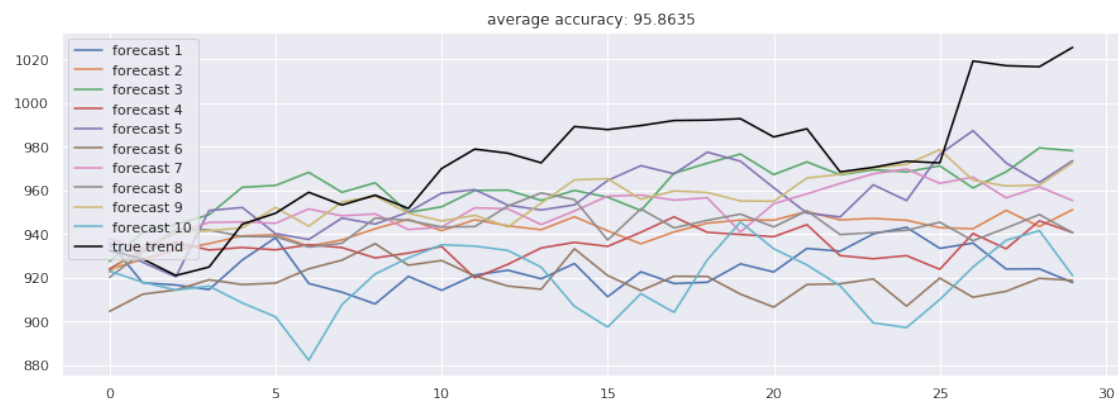
```
WARNING: Logging before flag parsing goes to stderr.
W0829 00:04:33.873839 140104212150080 deprecation.py:323] From <ipython-
input-6-1aeaade5f897>:44: dense (from tensorflow.python.layers.core) is
deprecated and will be removed in a future version.
Instructions for updating:
Use keras.layers.dense instead.
W0829 00:04:33.883059 140104212150080 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

simulation 1

W0829 00:04:34.265801 140104212150080 deprecation.py:323] From <ipython-
input-6-1aeaade5f897>:4: to_float (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.cast` instead.
W0829 00:04:34.294613 140104212150080 deprecation.py:323] From <ipython-
input-6-1aeaade5f897>:24: conv1d (from tensorflow.python.layers.convolutional)
is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.keras.layers.Conv1D` instead.
W0829 00:04:36.600379 140104212150080 deprecation.py:506] From <ipython-
input-6-1aeaade5f897>:82: calling dropout (from tensorflow.python.ops.nn_ops)
with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
train loop: 100%|      | 300/300 [00:14<00:00, 20.69it/s, acc=93,
cost=0.0106]

simulation 2

train loop: 100%|      | 300/300 [00:14<00:00, 20.99it/s, acc=97.6,
cost=0.00116]

simulation 3

train loop: 100%|      | 300/300 [00:14<00:00, 20.94it/s, acc=95.2,
cost=0.00553]

simulation 4

train loop: 100%|      | 300/300 [00:14<00:00, 20.97it/s, acc=95.4,
cost=0.00442]

simulation 5

train loop: 100%|      | 300/300 [00:14<00:00, 21.88it/s, acc=95.6,
cost=0.00393]

simulation 6

train loop: 100%|      | 300/300 [00:14<00:00, 21.01it/s, acc=95.3,
cost=0.00454]

simulation 7

train loop: 100%|      | 300/300 [00:14<00:00, 21.05it/s, acc=96.7,
cost=0.00229]

simulation 8
```

```
train loop: 100%|      | 300/300 [00:14<00:00, 21.01it/s, acc=97.1,
cost=0.00178]

simulation 9

train loop: 100%|      | 300/300 [00:14<00:00, 20.80it/s, acc=95.3,
cost=0.00492]

simulation 10

train loop: 100%|      | 300/300 [00:14<00:00, 20.94it/s, acc=90.6,
cost=0.0192]
```

[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()
```



[ ]:

```

```