Attention based Model

February 18, 2021

[50]:

```
import numpy as np
import pandas as pd
import os, datetime
import tensorflow as tf
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from sklearn.preprocessing import MinMaxScaler

import matplotlib.pyplot as plt
plt.style.use('seaborn')

import warnings
warnings.filterwarnings('ignore')
```

[51]:

```
batch_size = 64
seq_len = 128

d_k = 64
d_v = 64
h = 8
d_ff = 2048
```

[52]:

```
stock = pd.read_csv('C:\Jupyter_Project\Hanyang_Securities_F.csv')
df = stock.dropna()

df['Volume'].replace(to_replace=0, method='ffill', inplace=True)
df.sort_values('Date', inplace=True)
df.tail()
```

[52]:

```
            Date   Open   High    Low  Close  Adj Close  Volume
5181  2021-02-01   9200   9480   9100   9380     9380.0   81355
5182  2021-02-02   9460   9810   9460   9700     9700.0  105755
5183  2021-02-03   9850  10200   9800   9990     9990.0  170966
5184  2021-02-04  10100  10200   9940  10150    10150.0  133504
5185  2021-02-05  10200  10800  10150  10650    10650.0  247224
```

[53]:

```
ratio = df['Adj Close']/df['Close']
ratio
```

[53]:

```
0       0.231324
1       0.231324
2       0.231324
3       0.231324
4       0.231324
          ...
5181    1.000000
5182    1.000000
5183    1.000000
5184    1.000000
5185    1.000000
Length: 5186, dtype: float64
```

[54]:

```
df['Adj Open'] = df['Open']*ratio
df['Adj High'] = df['High']*ratio
df['Adj Low'] = df['Low']*ratio
```

[55]:

```
df.drop(['Open','High','Low','Close'], axis=1, inplace=True)

df
```

[55]:

```
            Date     Adj Close  Volume      Adj Open      Adj High  \
0     2000-01-04   1619.266357   56800   1457.339721   1642.398734
1     2000-01-05   1549.868774   52100   1549.868774   1642.398253
2     2000-01-06   1457.339844   64900   1619.266493   1619.266493
3     2000-01-07   1473.532349   61800   1468.905874   1526.736814
4     2000-01-10   1503.603882   56100   1529.049486   1549.868617
...          ...           ...     ...           ...           ...
5181  2021-02-01   9380.000000   81355   9200.000000   9480.000000
5182  2021-02-02   9700.000000  105755   9460.000000   9810.000000
5183  2021-02-03   9990.000000  170966   9850.000000  10200.000000
5184  2021-02-04  10150.000000  133504  10100.000000  10200.000000
5185  2021-02-05  10650.000000  247224  10200.000000  10800.000000

           Adj Low
0      1457.339721
1      1529.049641
2      1445.773655
3      1457.339686
4      1457.339147
...            ...
5181   9100.000000
5182   9460.000000
5183   9800.000000
5184   9940.000000
5185  10150.000000
```

```
[5186 rows x 6 columns]
```

[56]:

```
df.rename(columns={'Date':'Date','Adj Open':'Open', 'Adj High':'High', 'Adj Low':
 ↪'Low', 'Adj Close':'Close'}, inplace=True)
```

[57]:

```
df = df[['Date', 'Open', 'High', 'Low', 'Close', 'Volume']]

df.head()
```

[57]:

```
         Date         Open         High          Low        Close  Volume
0  2000-01-04  1457.339721  1642.398734  1457.339721  1619.266357   56800
1  2000-01-05  1549.868774  1642.398253  1529.049641  1549.868774   52100
2  2000-01-06  1619.266493  1619.266493  1445.773655  1457.339844   64900
3  2000-01-07  1468.905874  1526.736814  1457.339686  1473.532349   61800
4  2000-01-10  1529.049486  1549.868617  1457.339147  1503.603882   56100
```

[58]:

```
df.index.values
```

[58]:

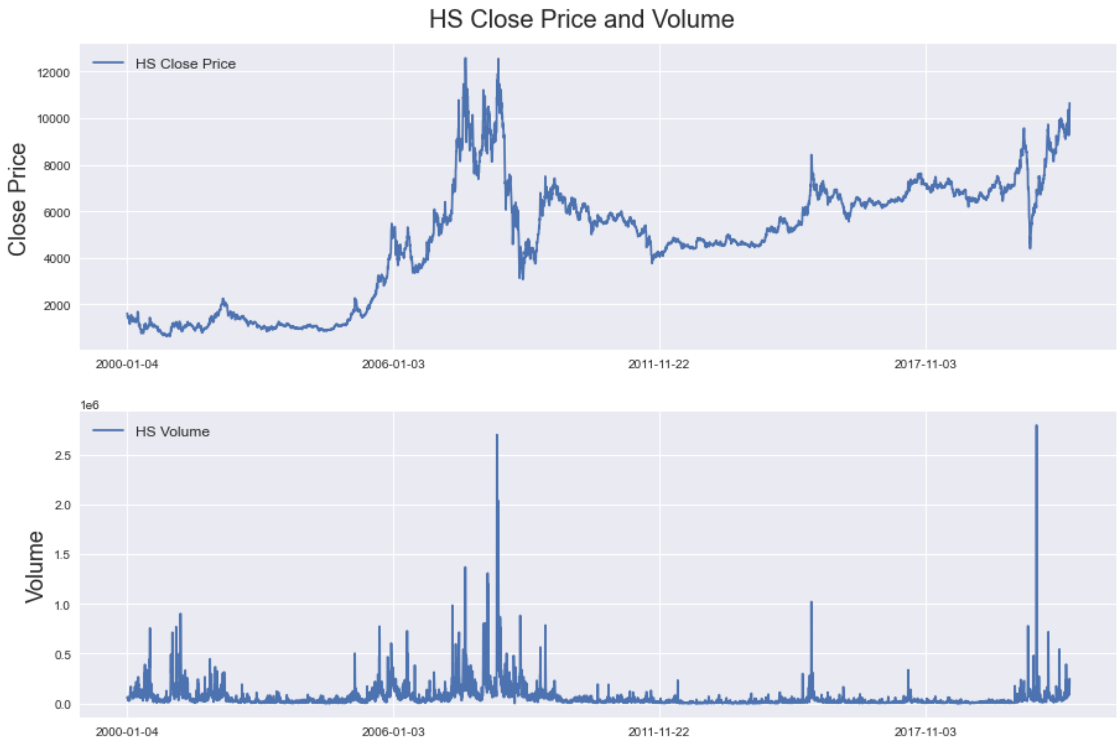
```
array([   0,    1,    2, ..., 5183, 5184, 5185], dtype=int64)
```

[59]:

```
fig = plt.figure(figsize=(15,10))
st = fig.suptitle("HS Close Price and Volume", fontsize=20)
st.set_y(0.92)

ax1 = fig.add_subplot(211)
ax1.plot(df['Close'], label='HS Close Price')
ax1.set_xticks(range(0, df.shape[0], 1464))
ax1.set_xticklabels(df['Date'].loc[::1464])
ax1.set_ylabel('Close Price', fontsize=18)
ax1.legend(loc="upper left", fontsize=12)

ax2 = fig.add_subplot(212)
ax2.plot(df['Volume'], label='HS Volume')
ax2.set_xticks(range(0, df.shape[0], 1464))
ax2.set_xticklabels(df['Date'].loc[::1464])
ax2.set_ylabel('Volume', fontsize=18)
ax2.legend(loc="upper left", fontsize=12)
```

[59]:

```
<matplotlib.legend.Legend at 0x26905243d60>
```



[60]:
```
df.head()
```

[60]:
```
         Date         Open         High          Low        Close  Volume
0  2000-01-04  1457.339721  1642.398734  1457.339721  1619.266357   56800
1  2000-01-05  1549.868774  1642.398253  1529.049641  1549.868774   52100
2  2000-01-06  1619.266493  1619.266493  1445.773655  1457.339844   64900
3  2000-01-07  1468.905874  1526.736814  1457.339686  1473.532349   61800
4  2000-01-10  1529.049486  1549.868617  1457.339147  1503.603882   56100
```

[61]:
```
scaler = MinMaxScaler()
scale_cols = ['Open', 'High', 'Low', 'Close', 'Volume']
df_scaled = scaler.fit_transform(df[scale_cols])

# 정규화가 완료된 데이터들은 pandas dataframe으로 변환합니다
# pandas는 시계열 자료에 대한 다양한 기능을 제공하여 LSTM에서 사용하는 window를 만들
때 유용합니다

df_scaled = pd.DataFrame(df_scaled)
df_scaled.columns = scale_cols

print(df_scaled)
```

```
          Open      High       Low     Close    Volume
0     0.069093  0.078420  0.072692  0.082280  0.020301
```

```
1     0.076891  0.078420  0.078924  0.076473  0.018620
2     0.082740  0.076587  0.071686  0.068730  0.023197
3     0.070068  0.069256  0.072692  0.070085  0.022088
4     0.075136  0.071089  0.072692  0.072601  0.020050
...        ...       ...       ...       ...       ...
5181  0.721622  0.699387  0.736878  0.731697  0.029080
5182  0.743534  0.725532  0.768164  0.758474  0.037804
5183  0.776402  0.756432  0.797711  0.782742  0.061119
5184  0.797472  0.756432  0.809878  0.796130  0.047725
5185  0.805899  0.803969  0.828128  0.837970  0.088383

[5186 rows x 5 columns]
```

[62]:

```
df_scaled.describe()
```

[62]:

```
              Open         High          Low        Close       Volume
count  5186.000000  5186.000000  5186.000000  5186.000000  5186.000000
mean      0.353266     0.336558     0.359212     0.350731     0.021491
std       0.216670     0.207100     0.219397     0.214807     0.043042
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.112288     0.107498     0.113192     0.111441     0.003833
50%       0.388059     0.367230     0.394341     0.384704     0.009001
75%       0.504751     0.478909     0.514688     0.500902     0.022008
max       1.000000     1.000000     1.000000     1.000000     1.000000
```

[63]:

```
times = sorted(df_scaled.index.values)
last_20pct = sorted(df_scaled.index.values)[-int(0.2*len(times))]
last_40pct = sorted(df_scaled.index.values)[-int(0.4*len(times))]
```

[64]:

```
df_train = df_scaled[(df_scaled.index < last_40pct)]  # Training data are 80% of␣
 ↪total data
df_valid = df_scaled[(df_scaled.index >= last_40pct) & (df_scaled.index <␣
 ↪last_20pct)]
df_test = df_scaled[(df_scaled.index >= last_20pct)]

# print proportions
print('train: {}% | validation: {}% | test {}%'.format(round(len(df_train)/
 ↪len(df_scaled),2),
                                                       round(len(df_valid)/
 ↪len(df_scaled),2),
                                                       round(len(df_test)/
 ↪len(df_scaled),2)))
```

```
train: 0.6% | validation: 0.2% | test 0.2%
```

[65]:

```
train_data = df_train.values
valid_data = df_valid.values
test_data = df_test.values
```

```
print('Training data shape: {}'.format(train_data.shape))
print('Validation data shape: {}'.format(valid_data.shape))
print('Test data shape: {}'.format(test_data.shape))

df_train.head()
```

```
Training data shape: (3112, 5)
Validation data shape: (1037, 5)
Test data shape: (1037, 5)
```

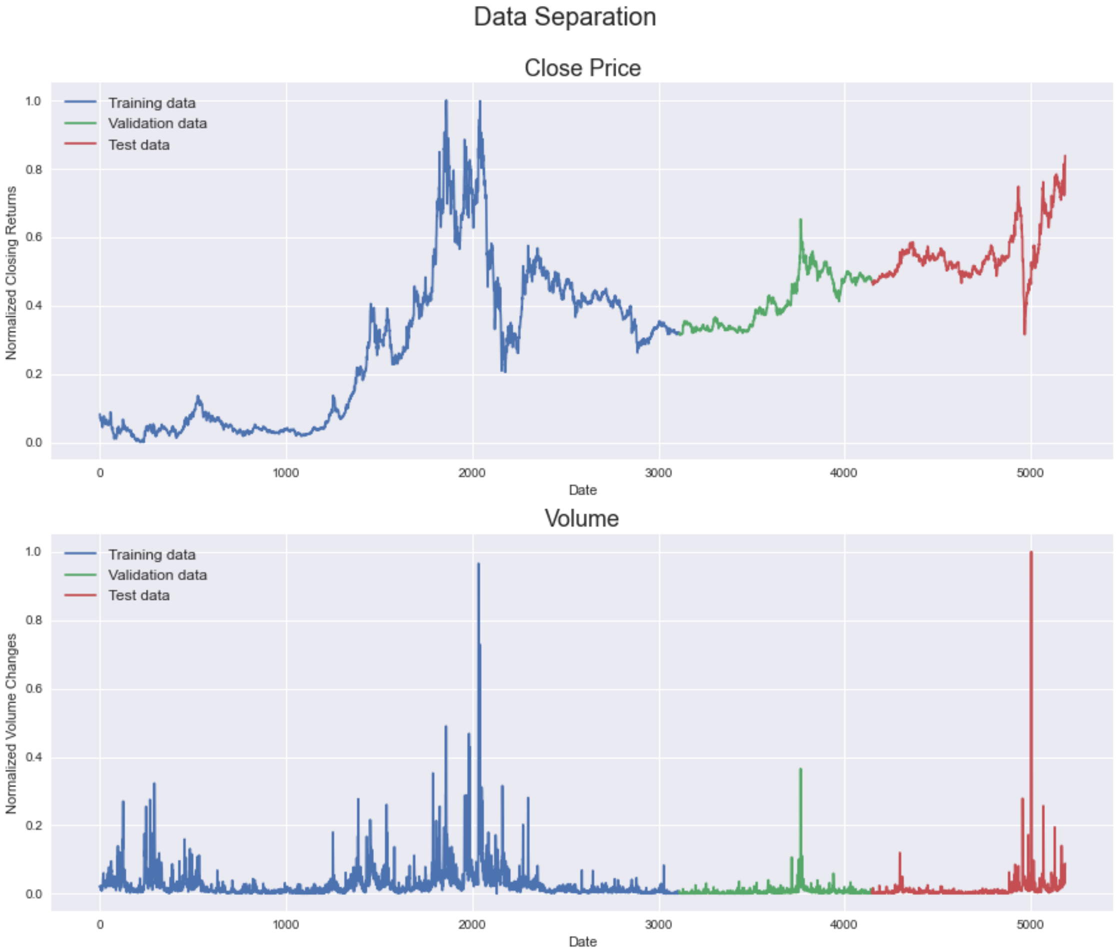
[65]:

```
       Open      High       Low     Close    Volume
0  0.069093  0.078420  0.072692  0.082280  0.020301
1  0.076891  0.078420  0.078924  0.076473  0.018620
2  0.082740  0.076587  0.071686  0.068730  0.023197
3  0.070068  0.069256  0.072692  0.070085  0.022088
4  0.075136  0.071089  0.072692  0.072601  0.020050
```

[66]:

```
fig = plt.figure(figsize=(15,12))
st = fig.suptitle("Data Separation", fontsize=20)
st.set_y(0.95)

###############################################################################

ax1 = fig.add_subplot(211)
ax1.plot(np.arange(train_data.shape[0]), df_train['Close'], label='Training␣
 ↪data')

ax1.plot(np.arange(train_data.shape[0],
                   train_data.shape[0]+valid_data.shape[0]), df_valid['Close'],␣
 ↪label='Validation data')

ax1.plot(np.arange(train_data.shape[0]+valid_data.shape[0],
                   train_data.shape[0]+valid_data.shape[0]+test_data.shape[0]),␣
 ↪df_test['Close'], label='Test data')
ax1.set_xlabel('Date')
ax1.set_ylabel('Normalized Closing Returns')
ax1.set_title("Close Price", fontsize=18)
ax1.legend(loc="best", fontsize=12)

###############################################################################

ax2 = fig.add_subplot(212)
ax2.plot(np.arange(train_data.shape[0]), df_train['Volume'], label='Training␣
 ↪data')

ax2.plot(np.arange(train_data.shape[0],
                   train_data.shape[0]+valid_data.shape[0]), df_valid['Volume'],␣
 ↪label='Validation data')

ax2.plot(np.arange(train_data.shape[0]+valid_data.shape[0],
                   train_data.shape[0]+valid_data.shape[0]+test_data.shape[0]),␣
 ↪df_test['Volume'], label='Test data')
ax2.set_xlabel('Date')
ax2.set_ylabel('Normalized Volume Changes')
ax2.set_title("Volume", fontsize=18)
ax2.legend(loc="best", fontsize=12)
```

[66]:
```
<matplotlib.legend.Legend at 0x2693b9bcfd0>
```



[67]:
```
# Training data
x_train, y_train = [], []
for i in range(seq_len, len(train_data)):
  x_train.append(train_data[i-seq_len:i]) # Chunks of training data with a␣
 ↪length of 128 df-rows
  y_train.append(train_data[:, 3][i]) #Value of 4th column (Close Price) of␣
 ↪df-row 128+1
x_train, y_train = np.array(x_train), np.array(y_train)

###############################################################################

# Validation data
x_valid, y_valid = [], []
for i in range(seq_len, len(valid_data)):
    x_valid.append(valid_data[i-seq_len:i])
    y_valid.append(valid_data[:, 3][i])
x_valid, y_valid = np.array(x_valid), np.array(y_valid)

###############################################################################

# Test data
x_test, y_test = [], []
for i in range(seq_len, len(test_data)):
    x_test.append(test_data[i-seq_len:i])
    y_test.append(test_data[:, 3][i])
x_test, y_test = np.array(x_test), np.array(y_test)

print('Training set shape', x_train.shape, y_train.shape)
print('Validation set shape', x_valid.shape, y_valid.shape)
print('Testing set shape' ,x_test.shape, y_test.shape)
```

```
Training set shape (2984, 128, 5) (2984,)
Validation set shape (909, 128, 5) (909,)
Testing set shape (909, 128, 5) (909,)
```

[68]:

```
class Time2Vector(Layer):
  def __init__(self, seq_len, **kwargs):
    super(Time2Vector, self).__init__()
    self.seq_len = seq_len

  def build(self, input_shape):
    '''Initialize weights and biases with shape (batch, seq_len)'''
    self.weights_linear = self.add_weight(name='weight_linear',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.bias_linear = self.add_weight(name='bias_linear',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.weights_periodic = self.add_weight(name='weight_periodic',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

    self.bias_periodic = self.add_weight(name='bias_periodic',
                                shape=(int(self.seq_len),),
                                initializer='uniform',
                                trainable=True)

  def call(self, x):
    '''Calculate linear and periodic time features'''
    x = tf.math.reduce_mean(x[:,:,:4], axis=-1)
    time_linear = self.weights_linear * x + self.bias_linear # Linear time␣
↪feature
    time_linear = tf.expand_dims(time_linear, axis=-1) # Add dimension (batch,␣
↪seq_len, 1)

    time_periodic = tf.math.sin(tf.multiply(x, self.weights_periodic) + self.
↪bias_periodic)
    time_periodic = tf.expand_dims(time_periodic, axis=-1) # Add dimension␣
↪(batch, seq_len, 1)
    return tf.concat([time_linear, time_periodic], axis=-1) # shape = (batch,␣
↪seq_len, 2)

  def get_config(self): # Needed for saving and loading model with custom layer
    config = super().get_config().copy()
    config.update({'seq_len': self.seq_len})
    return config
```

[69]:

```
class Scaled_Dot_Product_Attention(Layer):
  def __init__(self, d_k, d_v):
    super(Scaled_Dot_Product_Attention, self).__init__()
    self.d_k = d_k
    self.d_v = d_v

  def build(self, input_shape):
    self.query = Dense(self.d_k,
                       input_shape=input_shape,
                       kernel_initializer='glorot_uniform',
                       bias_initializer='glorot_uniform')

    self.key = Dense(self.d_k,
                     input_shape=input_shape,
                     kernel_initializer='glorot_uniform',
                     bias_initializer='glorot_uniform')

    self.value = Dense(self.d_v,
                       input_shape=input_shape,
                       kernel_initializer='glorot_uniform',
                       bias_initializer='glorot_uniform')

  def call(self, inputs): # inputs = (in_seq, in_seq, in_seq)
    q = self.query(inputs[0])
    k = self.key(inputs[1])

    attn_weights = tf.matmul(q, k, transpose_b=True)
    attn_weights = tf.map_fn(lambda x: x/np.sqrt(self.d_k), attn_weights)
    attn_weights = tf.nn.softmax(attn_weights, axis=-1)

    v = self.value(inputs[2])
    attn_out = tf.matmul(attn_weights, v)
    return attn_out

##############################################################################

class Multi_Head_Attention(Layer):
  def __init__(self, d_k, d_v, h):
    super(Multi_Head_Attention, self).__init__()
    self.d_k = d_k
    self.d_v = d_v
    self.h = h
    self.attn_heads = list()

  def build(self, input_shape):
    for n in range(self.h):
      self.attn_heads.append(Scaled_Dot_Product_Attention(self.d_k, self.d_v))

    # input_shape[0]=(batch, seq_len, 7), input_shape[0][-1]=7
    self.linear = Dense(input_shape[0][-1],
                        input_shape=input_shape,
                        kernel_initializer='glorot_uniform',
                        bias_initializer='glorot_uniform')

  def call(self, inputs):
    attn = [self.attn_heads[i](inputs) for i in range(self.h)]
    concat_attn = tf.concat(attn, axis=-1)
    multi_linear = self.linear(concat_attn)
    return multi_linear

##############################################################################
```

```
class TransformerEncoder(Layer):
  def __init__(self, d_k, d_v, h, d_ff, dropout=0.1, **kwargs):
    super(TransformerEncoder, self).__init__()
    self.d_k = d_k
    self.d_v = d_v
    self.h = h
    self.d_ff = d_ff
    self.attn_heads = list()
    self.dropout_rate = dropout

  def build(self, input_shape):
    self.attn_multi = Multi_Head_Attention(self.d_k, self.d_v, self.h)
    self.attn_dropout = Dropout(self.dropout_rate)
    self.attn_normalize = LayerNormalization(input_shape=input_shape,␣
 ↪epsilon=1e-6)

    self.ff_conv1D_1 = Conv1D(filters=self.d_ff, kernel_size=1,␣
 ↪activation='relu')
    # input_shape[0]=(batch, seq_len, 7), input_shape[0][-1] = 7
    self.ff_conv1D_2 = Conv1D(filters=input_shape[0][-1], kernel_size=1)
    self.ff_dropout = Dropout(self.dropout_rate)
    self.ff_normalize = LayerNormalization(input_shape=input_shape, epsilon=1e-6)

  def call(self, inputs): # inputs = (in_seq, in_seq, in_seq)
    attn_layer = self.attn_multi(inputs)
    attn_layer = self.attn_dropout(attn_layer)
    attn_layer = self.attn_normalize(inputs[0] + attn_layer)

    ff_layer = self.ff_conv1D_1(attn_layer)
    ff_layer = self.ff_conv1D_2(ff_layer)
    ff_layer = self.ff_dropout(ff_layer)
    ff_layer = self.ff_normalize(inputs[0] + ff_layer)
    return ff_layer

  def get_config(self): # Needed for saving and loading model with custom layer
    config = super().get_config().copy()
    config.update({'d_k': self.d_k,
                   'd_v': self.d_v,
                   'h': self.h,
                   'd_ff': self.d_ff,
                   'attn_heads': self.attn_heads,
                   'dropout_rate': self.dropout_rate})
    return config
```

[22]:

```
# val_loss가 10회 같을 시 early_stop, batch_size(=K)는 K문제 풀고 답보고 하는 식
# 위에서 모델을 구성한 후 compile 메서드를 호출하여 학습과정을 설정합니다
# optimizer : 훈련 과정을 설정한다
# loss : 최적화 과정에서 최소화될 손실 함수(loss function)을 설정합니다
# metrics : 훈련을 모니터링하기 위해 사용됩니다
# validation_data = 검증 데이터를 사용합니다. 각 에포크마다 정확도도 함께 출력됩니다
# 이 정확도는 훈련이 잘 되고 있는지를 보여줄 뿐이며 실제로 모델이 검증데이터를 학습하지는 않습니다
# 검증 데이터의 loss가 낮아지다가 높아지기 시작하면 overfitting의 신호입니다
# verbose / 0 : 출력 없음 / 1 : 훈련 진행도 보여주는 진행 막대 보여줌 / 2 : 미니 배치마다 손실 정보 출력

from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras import backend as K

def RMSE(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))

def soft_acc(y_true, y_pred):
    return K.mean(K.equal(K.round(y_true), K.round(y_pred)))

def MPE(y_true, y_pred):
    return K.mean((y_true - y_pred) / y_true) * 100

def MSLE(y_true, y_pred):
    return K.mean(K.square(K.log(y_true+1) - K.log(y_pred+1)), axis=-1)

def RMSLE(y_true, y_pred):
    return K.sqrt(K.mean(K.square(K.log(y_true+1) - K.log(y_pred+1)), axis=-1))

def R2(y_true, y_pred):
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()))
```

[29]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import optimizers
from keras.optimizers import Adam




def create_model():
  '''Initialize time and transformer layers'''
  time_embedding = Time2Vector(seq_len)
  attn_layer1 = TransformerEncoder(d_k, d_v, h, d_ff)
  attn_layer2 = TransformerEncoder(d_k, d_v, h, d_ff)
  attn_layer3 = TransformerEncoder(d_k, d_v, h, d_ff)


  '''Construct model'''
  in_seq = Input(shape=(seq_len, 5))
  x = time_embedding(in_seq)
  x = Concatenate(axis=-1)([in_seq, x])
  x = attn_layer1((x, x, x))
  x = attn_layer2((x, x, x))
  x = attn_layer3((x, x, x))
  x = GlobalAveragePooling1D(data_format='channels_first')(x)
  x = Dropout(0.1)(x)
  x = Dense(64, activation='relu')(x)
  x = Dropout(0.1)(x)
  out = Dense(1, activation='linear')(x)



  model = Model(inputs=in_seq, outputs=out)
  model.compile(loss = RMSE, optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999),␣
 ↪metrics=[soft_acc, 'mse', 'mae', RMSE, 'mape', MPE, MSLE, RMSLE, R2])
  return model


model = create_model()
model.summary()
filename = os.path.join('tmp', 'checkpointer.ckpt')

callback = tf.keras.callbacks.ModelCheckpoint(filename,
                                              monitor='val_loss',
                                              save_best_only=True, verbose=1)
early_stop = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=200,
                    callbacks=[callback, early_stop],
                    validation_data=(x_valid, y_valid))



###############################################################################
'''Calculate predictions and metrics'''

#Calculate predication for training, validation and test data
train_pred = model.predict(x_train)
```

```python
valid_pred = model.predict(x_valid)
test_pred = model.predict(x_test)

#Print evaluation metrics for all datasets
train_evaluate = model.evaluate(x_train, y_train, verbose=0)
valid_evaluate = model.evaluate(x_valid, y_valid, verbose=0)
test_evaluate = model.evaluate(x_test, y_test, verbose=0)
```

```
Model: "functional_3"
--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_2 (InputLayer)            [(None, 128, 5)]     0
--------------------------------------------------------------------------------
------------------
time2_vector_1 (Time2Vector)    (None, 128, 2)       512         input_2[0][0]
--------------------------------------------------------------------------------
------------------
concatenate_1 (Concatenate)     (None, 128, 7)       0           input_2[0][0]
time2_vector_1[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder_3 (Transfor (None, 128, 7)       46634
concatenate_1[0][0]
concatenate_1[0][0]
concatenate_1[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder_4 (Transfor (None, 128, 7)       46634
transformer_encoder_3[0][0]
transformer_encoder_3[0][0]
transformer_encoder_3[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder_5 (Transfor (None, 128, 7)       46634
transformer_encoder_4[0][0]
transformer_encoder_4[0][0]
transformer_encoder_4[0][0]
--------------------------------------------------------------------------------
------------------
global_average_pooling1d_1 (Glo (None, 128)          0
transformer_encoder_5[0][0]
--------------------------------------------------------------------------------
------------------
dropout_2 (Dropout)             (None, 128)          0
global_average_pooling1d_1[0][0]
```

```
--------------------------------------------------------------------------------
------------------
dense_2 (Dense)                 (None, 64)           8256        dropout_2[0][0]
--------------------------------------------------------------------------------
------------------
dropout_3 (Dropout)             (None, 64)           0           dense_2[0][0]
--------------------------------------------------------------------------------
------------------
dense_3 (Dense)                 (None, 1)            65          dropout_3[0][0]
================================================================================
==================
Total params: 148,735
Trainable params: 148,735
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
Epoch 1/200
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-29-19e5523f6ff3> in <module>
     45 early_stop = EarlyStopping(monitor='val_loss', patience=10)
     46 
---> 47 history = model.fit(x_train, y_train,

     48                       batch_size=batch_size,
     49                       epochs=200,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\e gine\training
 ↪py in _method_wrapper(self, *args, **kwargs)
    106   def _method_wrapper(self, *args, **kwargs):
    107     if not self._in_multi_worker_mode():  # pylint:␣
 ↪disable=protected-access
--> 108       return method(self, *args, **kwargs)
    109 
    110     # Running inside `run_distribute_coordinator` already.

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\e gine\training
 ↪py in fit(self, x, y, batch_size, epochs, verbose, callbacks, validation_split ␣
 ↪validation_data, shuffle, class_weight, sample_weight, initial_epoch,␣
 ↪steps_per_epoch, validation_steps, validation_batch_size, validation_freq,␣
 ↪max_queue_size, workers, use_multiprocessing)
   1096                 batch_size=batch_size):
   1097               callbacks.on_train_batch_begin(step)
-> 1098               tmp_logs = train_function(iterator)
   1099               if data_handler.should_sync:
   1100                 context.async_wait()
```

```
C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\d f_function.
↪py in __call__(self, *args, **kwds)
    778       else:
    779         compiler = "nonXla"
--> 780         result = self._call(*args, **kwds)
    781
    782       new_tracing_count = self._get_tracing_count()

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\d f_function.
↪py in _call(self, *args, **kwds)
    838         # Lifting succeeded, so variables are initialized and we can run␣
↪the
    839         # stateless function.
--> 840         return self._stateless_fn(*args, **kwds)
    841     else:
    842       canon_args, canon_kwds = \

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\f nction.
↪py in __call__(self, *args, **kwargs)
   2826     """Calls a graph function specialized to the inputs."""
   2827     with self._lock:
-> 2828       graph_function, args, kwargs = self._maybe_define_function(args,␣
↪kwargs)
   2829     return graph_function._filtered_call(args, kwargs)  # pylint:␣
↪disable=protected-access
   2830

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\f nction.
↪py in _maybe_define_function(self, args, kwargs)
   3211
   3212       self._function_cache.missed.add(call_context_key)
-> 3213       graph_function = self._create_graph_function(args, kwargs)
   3214       self._function_cache.primary[cache_key] = graph_function
   3215       return graph_function, args, kwargs

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\f nction.
↪py in _create_graph_function(self, args, kwargs, override_flat_arg_shapes)
   3063     arg_names = base_arg_names + missing_arg_names
   3064     graph_function = ConcreteFunction(
-> 3065         func_graph_module.func_graph_from_py_func(
   3066             self._name,
   3067             self._python_function,
C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\func_graph.
 ↪py in func_graph_from_py_func(name, python_func, args, kwargs, signature,␣
 ↪func_graph, autograph, autograph_options, add_control_dependencies, arg_names,
 ↪op_return_value, collections, capture_by_value, override_flat_arg_shapes)
    984         _, original_func = tf_decorator.unwrap(python_func)
    985
--> 986       func_outputs = python_func(*func_args, **func_kwargs)
    987
    988       # invariant: `func_outputs` contains only Tensors, CompositeTensors,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\def_function.
 ↪py in wrapped_fn(*args, **kwds)
    598         # __wrapped__ allows AutoGraph to swap in a converted function. We␣
 ↪give
    599         # the function a weak reference to itself to avoid a reference␣
 ↪cycle.
--> 600         return weak_wrapped_fn().__wrapped__(*args, **kwds)
    601     weak_wrapped_fn = weakref.ref(wrapped_fn)
    602

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\func_graph.
 ↪py in wrapper(*args, **kwargs)
    960           # TODO(mdan): Push this block higher in tf.function's call stack.
    961           try:
--> 962             return autograph.converted_call(

    963                 original_func,
    964                 args,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autograph\impl\api.
 ↪py in converted_call(f, args, kwargs, caller_fn_scope, options)
    594     try:
    595       if kwargs is not None:
--> 596         result = converted_f(*effective_args, **kwargs)
    597       else:
    598         result = converted_f(*effective_args)

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\engine\training
 ↪py in tf__train_function(iterator)
     14                 try:
     15                     do_return = True
---> 16                     retval_ = ag__.converted_call(ag__.ld(step_function),␣
 ↪(ag__.ld(self), ag__.ld(iterator)), None, fscope)
     17                 except:
```

```
     18                     do_return = False

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autogra h\impl\api.
 ↪py in converted_call(f, args, kwargs, caller_fn_scope, options)
    455   if conversion.is_in_whitelist_cache(f, options):
    456     logging.log(2, 'Whitelisted %s: from cache', f)
--> 457     return _call_unconverted(f, args, kwargs, options, False)
    458 
    459   if ag_ctx.control_status_ctx().status == ag_ctx.Status.DISABLED:

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autogra h\impl\api.
 ↪py in _call_unconverted(f, args, kwargs, options, update_cache)
    338   if kwargs is not None:
    339     return f(*args, **kwargs)
--> 340   return f(*args)
    341 
    342 


C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\e gine\training
 ↪py in step_function(model, iterator)
    794 
    795       data = next(iterator)
--> 796       outputs = model.distribute_strategy.run(run_step, args=(data,))
    797       outputs = reduce_per_replica(
    798           outputs, self.distribute_strategy, reduction='first')

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\distrib te\distribute
 ↪py in run(***failed resolving arguments***)
   1209       fn = autograph.tf_convert(
   1210           fn, autograph_ctx.control_status_ctx(), convert_by_default=Fals e)
-> 1211       return self._extended.call_for_each_replica(fn, args=args,␣
 ↪kwargs=kwargs)
   1212 
   1213   # TODO(b/151224785): Remove deprecated alias.

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\distrib te\distribute
 ↪py in call_for_each_replica(self, fn, args, kwargs)
   2583       kwargs = {}
   2584     with self._container_strategy().scope():
-> 2585       return self._call_for_each_replica(fn, args, kwargs)
   2586 
   2587   def _call_for_each_replica(self, fn, args, kwargs):
```

```
C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\distribute\distribute
↪py in _call_for_each_replica(self, fn, args, kwargs)
   2943         self._container_strategy(),
   2944         replica_id_in_sync_group=constant_op.constant(0, dtypes.int32)):
-> 2945       return fn(*args, **kwargs)
   2946
   2947   def _reduce_to(self, reduce_op, value, destinations, experimental_hints):

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autograph\impl\api.
↪py in wrapper(*args, **kwargs)
    253       try:
    254         with conversion_ctx:
--> 255           return converted_call(f, args, kwargs, options=options)
    256       except Exception as e:  # pylint:disable=broad-except
    257         if hasattr(e, 'ag_error_metadata'):

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autograph\impl\api.
↪py in converted_call(f, args, kwargs, caller_fn_scope, options)
    530
    531   if not options.user_requested and conversion.is_whitelisted(f):
--> 532     return _call_unconverted(f, args, kwargs, options)
    533
    534   # internal_convert_user_code is for example turned off when issuing a␣
↪dynamic

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\autograph\impl\api.
↪py in _call_unconverted(f, args, kwargs, options, update_cache)
    337
    338   if kwargs is not None:
--> 339     return f(*args, **kwargs)
    340   return f(*args)
    341

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\engine\training
↪py in run_step(data)
    787
    788       def run_step(data):
--> 789         outputs = model.train_step(data)
    790         # Ensure counter is updated only if `train_step` succeeds.
    791         with ops.control_dependencies(_minimum_control_deps(outputs)):

C:
↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\engine\training
↪py in train_step(self, data)
```

```
    754     # The _minimize call does a few extra steps unnecessary in most cases,
    755     # such as loss scaling and gradient clipping.
--> 756     _minimize(self.distribute_strategy, tape, self.optimizer, loss,
    757               self.trainable_variables)
    758 

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\keras\e gine\training
 ↪py in _minimize(strategy, tape, optimizer, loss, trainable_variables)
   2720       loss = optimizer.get_scaled_loss(loss)
   2721 
-> 2722   gradients = tape.gradient(loss, trainable_variables)
   2723 
   2724   # Whether to aggregate gradients outside of optimizer. This requires␣
 ↪support

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\b ckprop.
 ↪py in gradient(self, target, sources, output_gradients, unconnected_gradients)
   1065                           for x in nest.flatten(output_gradients)]
   1066 
-> 1067     flat_grad = imperative_grad.imperative_grad(
   1068         self._tape,
   1069         flat_targets,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\i perative_grad
 ↪py in imperative_grad(tape, target, sources, output_gradients, sources_raw,␣
 ↪unconnected_gradients)
     69         "Unknown value for unconnected_gradients: %r" %␣
 ↪unconnected_gradients)
     70 
---> 71   return pywrap_tfe.TFE_Py_TapeGradient(
     72       tape._tape,  # pylint: disable=protected-access
     73       target,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\eager\b ckprop.
 ↪py in _gradient_function(op_name, attr_tuple, num_inputs, inputs, outputs,␣
 ↪out_grads, skip_input_indices, forward_pass_name_scope)
    160       gradient_name_scope += forward_pass_name_scope + "/"
    161     with ops.name_scope(gradient_name_scope):
--> 162       return grad_fn(mock_op, *out_grads)
    163   else:
    164     return grad_fn(mock_op, *out_grads)
```

```
C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\ops\mat _grad.
 ↪py in _MeanGrad(op, grad)
    263     output_shape = array_ops.shape(op.outputs[0])
    264     factor = _safe_shape_div(
--> 265         math_ops.reduce_prod(input_shape), math_ops.
 ↪reduce_prod(output_shape))
    266   return math_ops.truediv(sum_grad, math_ops.cast(factor, sum_grad.
 ↪dtype)), None
    267

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\util\di patch.
 ↪py in wrapper(*args, **kwargs)
    199     """Call target, and fall back on dispatchers if there is a TypeError.
 ↪"""
    200     try:
--> 201       return target(*args, **kwargs)
    202     except (TypeError, ValueError):
    203       # Note: convert_to_eager_tensor currently raises a ValueError, not a

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\ops\mat _ops.
 ↪py in reduce_prod(input_tensor, axis, keepdims, name)
   2455   return _may_reduce_to_scalar(
   2456       keepdims, axis,
-> 2457       gen_math_ops.prod(
   2458           input_tensor, _ReductionDims(input_tensor, axis), keepdims,
   2459           name=name))

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\ops\gen math_ops.
 ↪py in prod(input, axis, keep_dims, name)
   6735     keep_dims = False
   6736   keep_dims = _execute.make_bool(keep_dims, "keep_dims")
-> 6737   _, _, _op, _outputs = _op_def_library._apply_op_helper(
   6738         "Prod", input=input, reduction_indices=axis, keep_dims=keep_dims,
   6739                 name=name)

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framewo k\op_def_libr
 ↪py in _apply_op_helper(op_type_name, name, **keywords)
    740       # Add Op to graph
    741       # pylint: disable=protected-access
--> 742       op = g._create_op_internal(op_type_name, inputs, dtypes=None,
    743                                  name=scope, input_types=input_types,
    744                                  attrs=attr_protos, op_def=op_def)
```

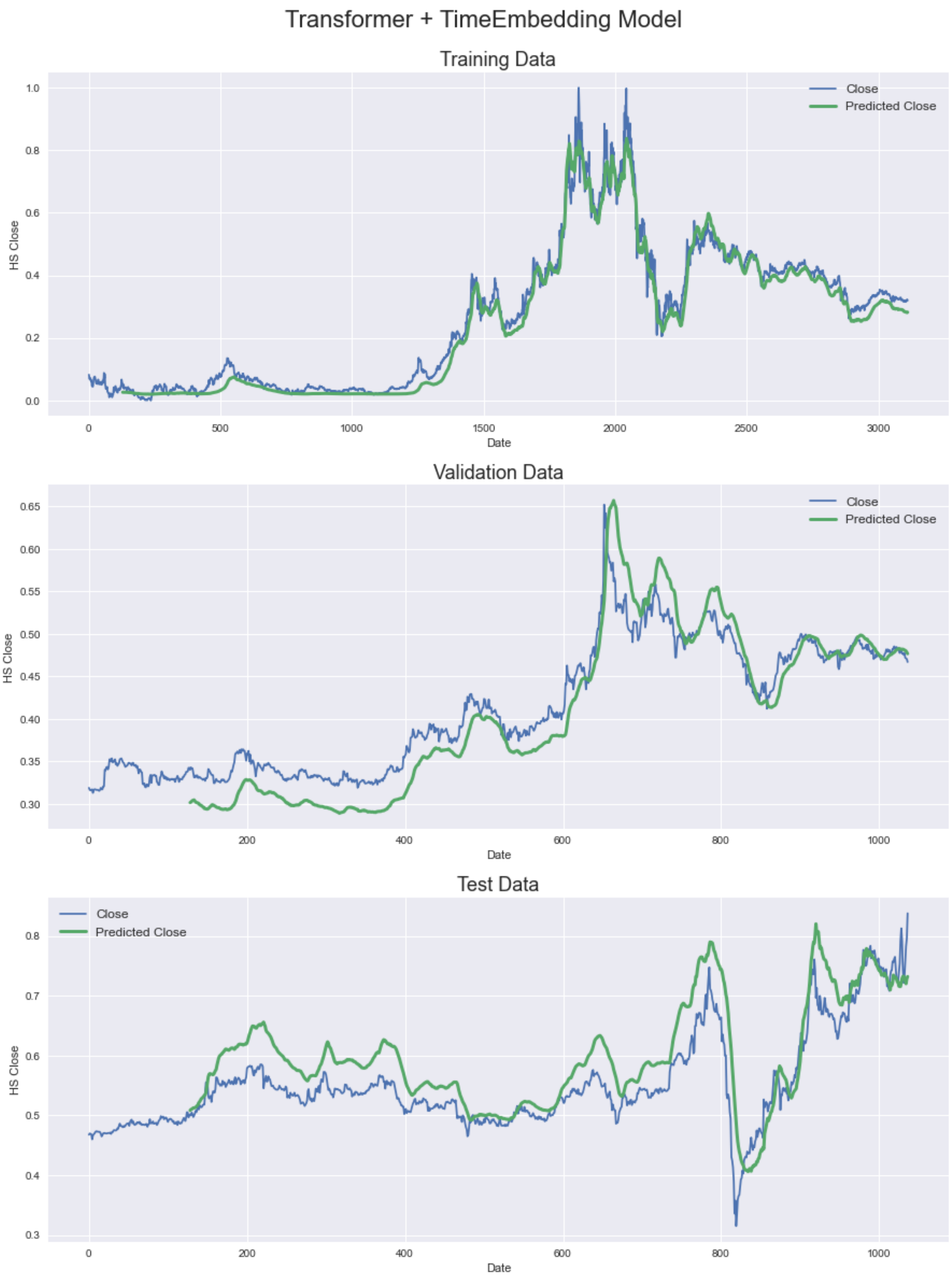
```
C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\func_graph.
 ↪py in _create_op_internal(self, op_type, inputs, dtypes, input_types, name,␣
 ↪attrs, op_def, compute_device)
    589       inp = self.capture(inp)
    590       inputs[i] = inp
--> 591     return super(FuncGraph, self)._create_op_internal(  # pylint:␣
 ↪disable=protected-access
    592         op_type, inputs, dtypes, input_types, name, attrs, op_def,
    593         compute_device)

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\ops.
 ↪py in _create_op_internal(self, op_type, inputs, dtypes, input_types, name,␣
 ↪attrs, op_def, compute_device)
   3475     # Session.run call cannot occur between creating and mutating the op.
   3476     with self._mutation_lock():
-> 3477       ret = Operation(
   3478           node_def,
   3479           self,

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\ops.
 ↪py in __init__(self, node_def, g, inputs, output_types, control_inputs,␣
 ↪input_types, original_op, op_def)
   1972       if op_def is None:
   1973         op_def = self._graph._get_op_def(node_def.op)
-> 1974       self._c_op = _create_c_op(self._graph, node_def, inputs,
   1975                                 control_input_ops, op_def)
   1976       name = compat.as_str(node_def.name)

C:
 ↪\ProgramData\Anaconda3\envs\muiiya\lib\site-packages\tensorflow\python\framework\ops.
 ↪py in _create_c_op(graph, node_def, inputs, control_inputs, op_def)
   1783   inputs = _reconstruct_sequence_inputs(op_def, inputs, node_def.attr)
   1784   # pylint: disable=protected-access
-> 1785   op_desc = pywrap_tf_session.TF_NewOperation(graph._c_graph,
   1786                                               compat.as_str(node_def.op),
   1787                                               compat.as_str(node_def.name))

KeyboardInterrupt:
```

[31]:

```
'''Display results'''

fig = plt.figure(figsize=(15,20))
st = fig.suptitle("Transformer + TimeEmbedding Model", fontsize=22)
```

```
st.set_y(0.92)

#Plot training data results
ax11 = fig.add_subplot(311)
ax11.plot(train_data[:, 3], label='Close')
ax11.plot(np.arange(seq_len, train_pred.shape[0]+seq_len), train_pred,␣
 ↪linewidth=3, label='Predicted Close')
ax11.set_title("Training Data", fontsize=18)
ax11.set_xlabel('Date')
ax11.set_ylabel('HS Close')
ax11.legend(loc="best", fontsize=12)

#Plot validation data results
ax21 = fig.add_subplot(312)
ax21.plot(valid_data[:, 3], label='Close')
ax21.plot(np.arange(seq_len, valid_pred.shape[0]+seq_len), valid_pred,␣
 ↪linewidth=3, label='Predicted Close')
ax21.set_title("Validation Data", fontsize=18)
ax21.set_xlabel('Date')
ax21.set_ylabel('HS Close')
ax21.legend(loc="best", fontsize=12)

#Plot test data results
ax31 = fig.add_subplot(313)
ax31.plot(test_data[:, 3], label='Close')
ax31.plot(np.arange(seq_len, test_pred.shape[0]+seq_len), test_pred,␣
 ↪linewidth=3, label='Predicted Close')
ax31.set_title("Test Data", fontsize=18)
ax31.set_xlabel('Date')
ax31.set_ylabel('HS Close')
ax31.legend(loc="best", fontsize=12)
```

```
[31]: <matplotlib.legend.Legend at 0x2693bf1e280>
```

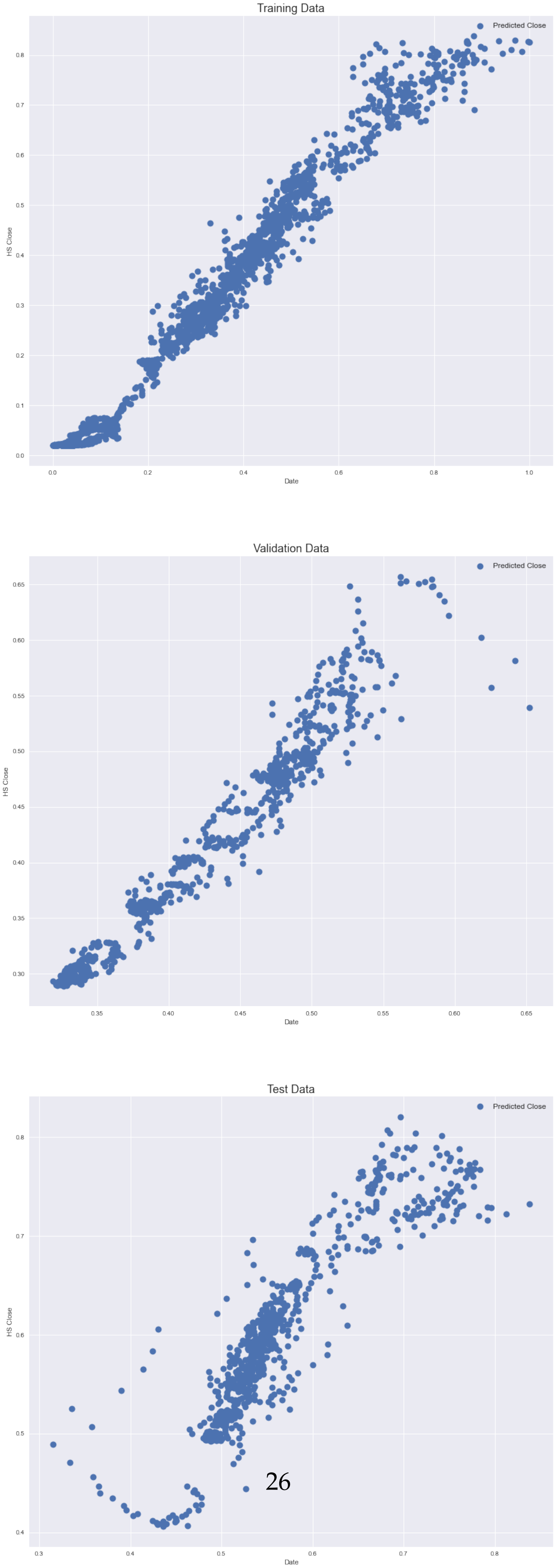


[48]:
```
# 원래값과 예측 값이 일치하면 직선에 가깝게 분포가 된다
```

```
%matplotlib inline
import matplotlib.pyplot as plt

'''Display results'''

fig = plt.figure(figsize=(15,45))
st = fig.suptitle("Transformer + TimeEmbedding Model", fontsize=22)
st.set_y(0.92)

#Plot training data results
ax11 = fig.add_subplot(311)
plt.scatter(np.asarray(y_train), train_pred, linewidth=3, label='Predicted␣
 ↪Close')
ax11.set_title("Training Data", fontsize=18)
ax11.set_xlabel('Date')
ax11.set_ylabel('HS Close')
ax11.legend(loc="best", fontsize=12)

#Plot validation data results
ax21 = fig.add_subplot(312)
plt.scatter(np.asarray(y_valid), valid_pred, linewidth=3, label='Predicted␣
 ↪Close')
ax21.set_title("Validation Data", fontsize=18)
ax21.set_xlabel('Date')
ax21.set_ylabel('HS Close')
ax21.legend(loc="best", fontsize=12)

#Plot test data results
ax31 = fig.add_subplot(313)
plt.scatter(np.asarray(y_test), test_pred, linewidth=3, label='Predicted Close')
ax31.set_title("Test Data", fontsize=18)
ax31.set_xlabel('Date')
ax31.set_ylabel('HS Close')
ax31.legend(loc="best", fontsize=12)
```

```
[48]: <matplotlib.legend.Legend at 0x2691d344550>
```

Transformer + TimeEmbedding Model



Training Data

Validation Data

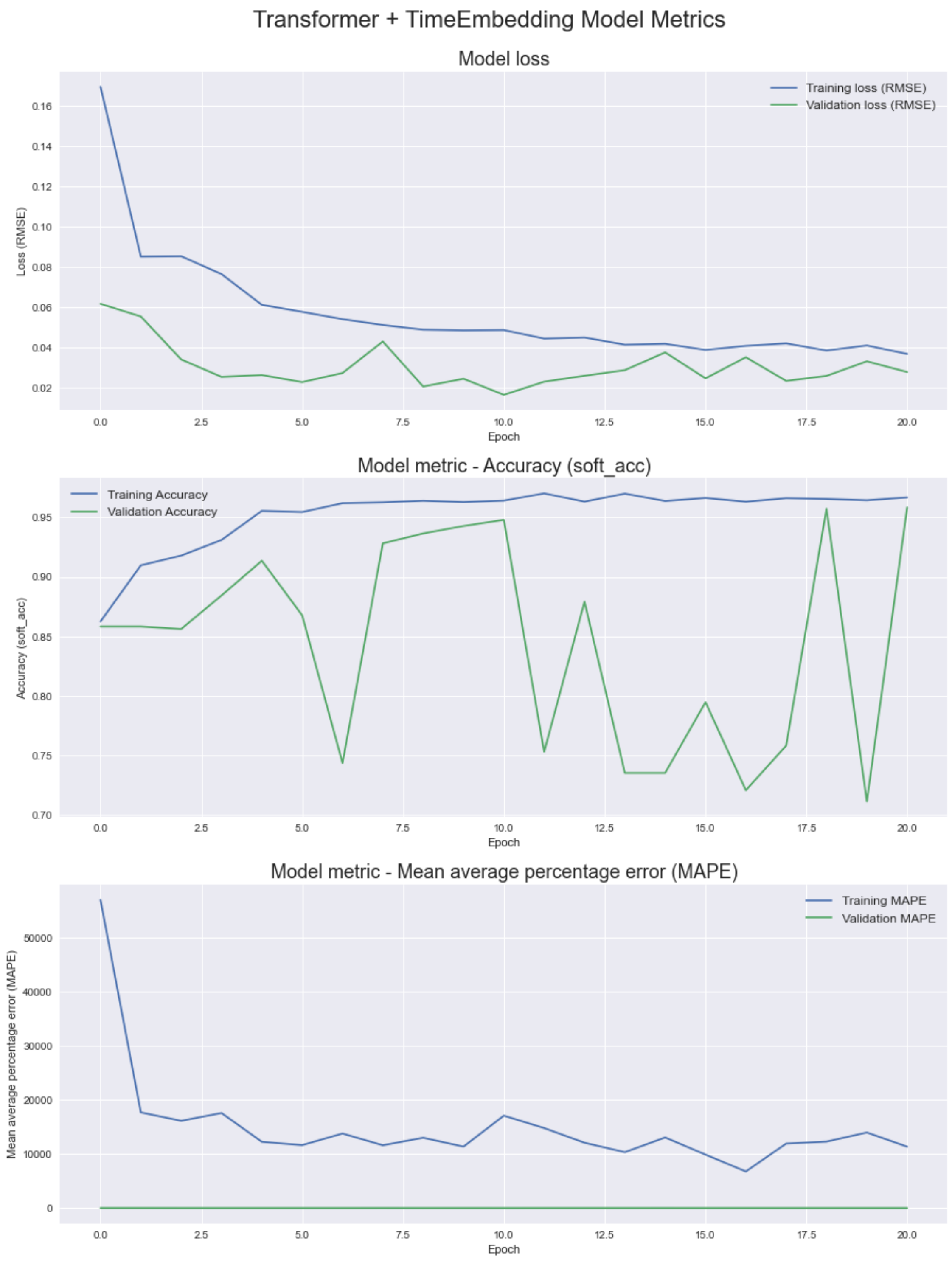
Test Data

[26]:

```
'''Display model metrics'''

fig = plt.figure(figsize=(15,20))
st = fig.suptitle("Transformer + TimeEmbedding Model Metrics", fontsize=22)
st.set_y(0.92)

#Plot Model Loss
ax1 = fig.add_subplot(311)
ax1.plot(history.history['loss'], label='Training loss (RMSE)')
ax1.plot(history.history['val_loss'], label='Validation loss (RMSE)')
ax1.set_title("Model loss", fontsize=18)
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss (RMSE)')
ax1.legend(loc="best", fontsize=12)

#Plot Model Acurracy
ax2 = fig.add_subplot(312)
ax2.plot(history.history['soft_acc'], label='Training Accuracy')
ax2.plot(history.history['val_soft_acc'], label='Validation Accuracy')
ax2.set_title("Model metric - Accuracy (soft_acc)", fontsize=18)
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy (soft_acc)')
ax2.legend(loc="best", fontsize=12)

#Plot MAPE
ax3 = fig.add_subplot(313)
ax3.plot(history.history['mape'], label='Training MAPE')
ax3.plot(history.history['val_mape'], label='Validation MAPE')
ax3.set_title("Model metric - Mean average percentage error (MAPE)", fontsize=18)
ax3.set_xlabel('Epoch')
ax3.set_ylabel('Mean average percentage error (MAPE)')
ax3.legend(loc="best", fontsize=12)
```

[26]:

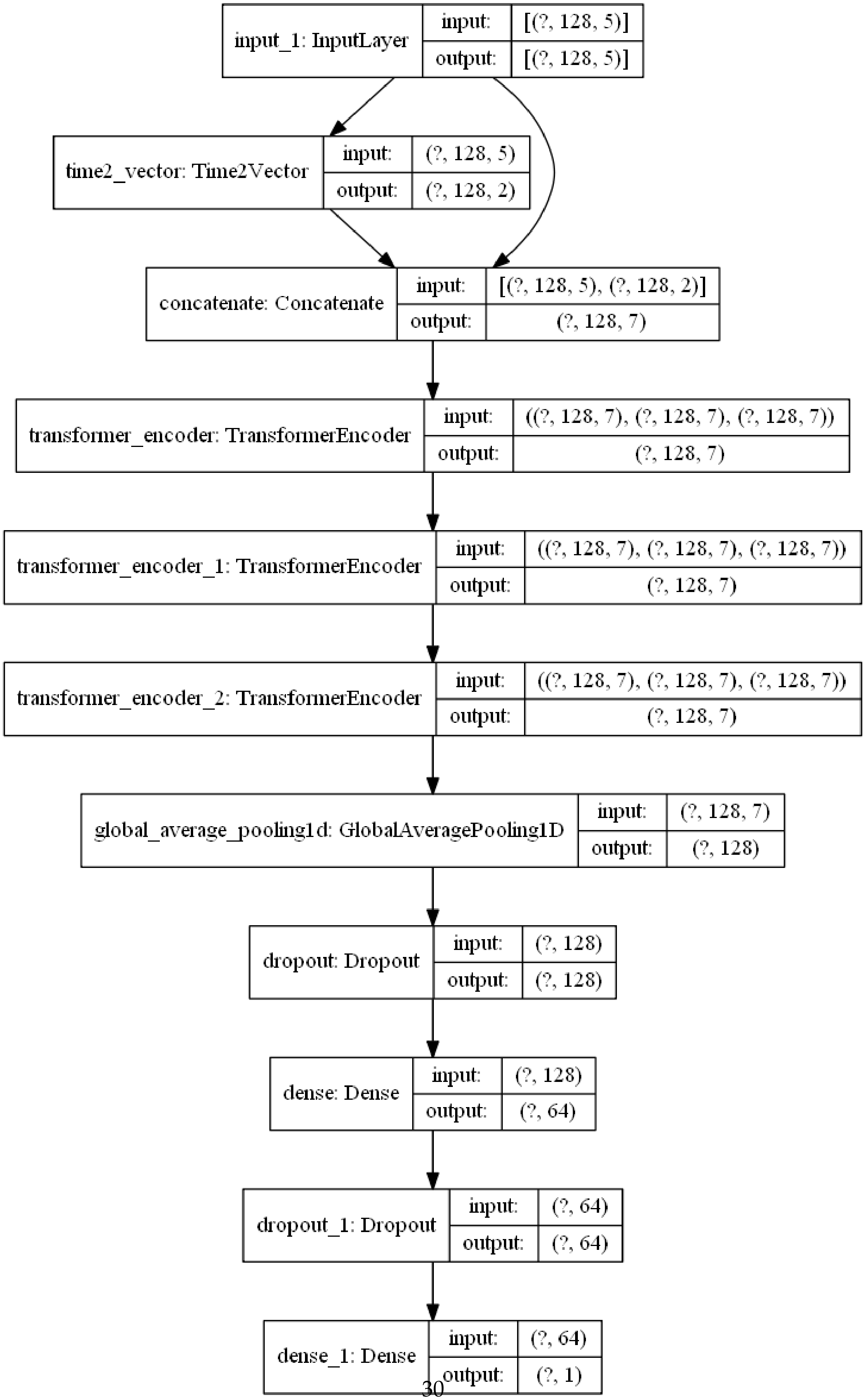
```
<matplotlib.legend.Legend at 0x2690438f040>
```



[24]:

```
tf.keras.utils.plot_model(
    model,
    to_file="HS_Transformer+TimeEmbedding.png",
    show_shapes=True,
    show_layer_names=True,
    expand_nested=True,
    dpi=96,)
```

[24]:


input_1: InputLayer
input:
[(?, 128, 5)]
output:
[(?, 128, 5)]
time2_vector: Time2Vector
input:
(?, 128, 5)
output:
(?, 128, 2)
concatenate: Concatenate
input:
[(?, 128, 5), (?, 128, 2)]
output:
(?, 128, 7)
transformer_encoder: TransformerEncoder
input:
((?, 128, 7), (?, 128, 7), (?, 128, 7))
output:
(?, 128, 7)
transformer_encoder_1: TransformerEncoder
input:
((?, 128, 7), (?, 128, 7), (?, 128, 7))
output:
(?, 128, 7)
transformer_encoder_2: TransformerEncoder
input:
((?, 128, 7), (?, 128, 7), (?, 128, 7))
output:
(?, 128, 7)
global_average_pooling1d: GlobalAveragePooling1D
input:
(?, 128, 7)
output:
(?, 128)
dropout: Dropout
input:
(?, 128)
output:
(?, 128)
dense: Dense
input:
(?, 128)
output:
(?, 64)
dropout_1: Dropout
input:
(?, 64)
output:
(?, 64)
dense_1: Dense
input:
(?, 64)
output:
(?, 1)

[26]:

```
model.summary()
```

```
Model: "functional_1"
--------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
==================
input_1 (InputLayer)            [(None, 128, 5)]     0
--------------------------------------------------------------------------------
------------------
time2_vector (Time2Vector)      (None, 128, 2)       512         input_1[0][0]
--------------------------------------------------------------------------------
------------------
concatenate (Concatenate)       (None, 128, 7)       0           input_1[0][0]
time2_vector[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder (Transforme (None, 128, 7)       46634
concatenate[0][0]
concatenate[0][0]
concatenate[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder_1 (Transfor (None, 128, 7)       46634
transformer_encoder[0][0]
transformer_encoder[0][0]
transformer_encoder[0][0]
--------------------------------------------------------------------------------
------------------
transformer_encoder_2 (Transfor (None, 128, 7)       46634
transformer_encoder_1[0][0]
transformer_encoder_1[0][0]
transformer_encoder_1[0][0]
--------------------------------------------------------------------------------
------------------
global_average_pooling1d (Globa (None, 128)          0
transformer_encoder_2[0][0]
--------------------------------------------------------------------------------
------------------
dropout (Dropout)               (None, 128)          0
global_average_pooling1d[0][0]
--------------------------------------------------------------------------------
------------------
dense (Dense)                   (None, 64)           8256        dropout[0][0]
```

```
--------------------------------------------------------------------------------
------------------
dropout_1 (Dropout)             (None, 64)           0           dense[0][0]
--------------------------------------------------------------------------------
------------------
dense_1 (Dense)                 (None, 1)            65          dropout_1[0][0]
================================================================================
==================
Total params: 148,735
Trainable params: 148,735
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

[36]:

```
import numpy as np


print('R2_Score')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[9], valid_evaluate[9], test_evaluate[9]))

print('Mean Squared Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[2], valid_evaluate[2], test_evaluate[2]))

print('Mean Absolute Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[3], valid_evaluate[3], test_evaluate[3]))

print('Root Mean Squared Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[4], valid_evaluate[4], test_evaluate[4]))

print('Mean Squared Logarithmic Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[7], valid_evaluate[7], test_evaluate[7]))

print('Root Mean Squared Logarithmic Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[8], valid_evaluate[8], test_evaluate[8]))
```

```
print('Mean Absolute Percentage Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[5], valid_evaluate[5], test_evaluate[5]))

print('Mean Percentage Error')
print('-' * 40)
print('train error: {} |\nvalid error: {} |\ntest error : {}\n'.
 ↪format(train_evaluate[6], valid_evaluate[6], test_evaluate[6]))
```

```
R2_Score
----------------------------------------
train error: -12.3204927444458 |
valid error: -15.43246841430664 |
test error : -19.464357376098633

Mean Squared Error
----------------------------------------
train error: 0.001218542456626892 |
valid error: 0.0009290337329730392 |
test error : 0.002639360260218382

Mean Absolute Error
----------------------------------------
train error: 0.02679363824427128 |
valid error: 0.025435589253902435 |
test error : 0.042303115129470825

Root Mean Squared Error
----------------------------------------
train error: 0.03006589598953724 |
valid error: 0.02718539535999298 |
test error : 0.045397695153951645

Mean Squared Logarithmic Error
----------------------------------------
train error: 0.0006790685001760721 |
valid error: 0.00046420295257121325 |
test error : 0.001039888127706945

Root Mean Squared Logarithmic Error
----------------------------------------
train error: 0.020755302160978317 |
valid error: 0.018145276233553886 |
test error : 0.026628771796822548

Mean Absolute Percentage Error
----------------------------------------
```

```
train error: 6683.10205078125 |
valid error: 6.372311115264893 |
test error : 7.651193618774414

Mean Percentage Error
----------------------------------------
train error: -inf |
valid error: 3.62542986869812 |
test error : -6.325413227081299
```

[ ]:

```

```