fkkmy4qml

August 9, 2024

[ ]:
```
import tensorflow as tf
from scipy.io import loadmat
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Flatten, Conv1D, MaxPooling1D,␣
 ↪Dropout, BatchNormalization, LSTM, Reshape
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import os
import pandas as pd
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.losses import Huber
from tensorflow.keras.regularizers import l2
from tensorflow.keras.initializers import HeNormal
```

[ ]:
```
# Run this cell to connect to your Drive folder

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[ ]:
```
df_forFCST = pd.read_csv(  r'/content/gdrive/MyDrive/DataStore/salesD_smoothed.
 ↪csv' )
df_forFCST.set_index( 'ds', inplace = True )
df_forFCST.index = pd.to_datetime( df_forFCST.index )
df_forFCST.index
```

[ ]:
```
DatetimeIndex(['2019-01-01', '2019-01-02', '2019-01-03', '2019-01-04',
               '2019-01-05', '2019-01-06', '2019-01-07', '2019-01-08',
               '2019-01-09', '2019-01-10',
               …
               '2024-06-21', '2024-06-22', '2024-06-23', '2024-06-24',
```

```
               '2024-06-25', '2024-06-26', '2024-06-27', '2024-06-28',
               '2024-06-29', '2024-06-30'],
              dtype='datetime64[ns]', name='ds', length=2008, freq=None)
```

[ ]:
```python
def get_model(input_shape, wd = 3e-3):
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu',␣
 ↪input_shape=input_shape,  kernel_initializer=HeNormal()),
        MaxPooling1D(pool_size=2),
        #Conv1D(filters=64, kernel_size=3, activation='relu'),
        #MaxPooling1D(pool_size=2),

        Flatten(),

        Dense(256, activation='relu', kernel_regularizer=l2(wd)),
        Dense(256, activation='relu', kernel_regularizer=l2(wd)),
        Dense(1)
    ])
    return model
```

[ ]:
```python
def get_compile(model, lrate = 1e-4):
    optimizer = tf.keras.optimizers.SGD(momentum=0.9, learning_rate=lrate)

    model.compile(optimizer=optimizer,
              loss='huber',
                #loss = 'mae',
                  metrics = ['mae', 'mse'])

def get_checkpoint_every_epoch():
    return ModelCheckpoint(
        filepath='/content/gdrive/MyDrive/var/FTryModel/checkpoints_every_epoch/
 ↪checkpoint_{epoch:03d}.weights.h5',
        save_weights_only=True,
        save_freq='epoch',
    )

def get_checkpoint_best_only():
    return ModelCheckpoint(
        filepath='/content/gdrive/MyDrive/var/FTryModel/checkpoints_best_only/
 ↪checkpoint.weights.h5',
        save_weights_only=True,
        monitor='loss',
        save_best_only=True,
        mode='min',
    )

def get_early_stopping():
```

```
    return EarlyStopping(monitor='loss', patience=100, mode='min', min_delta=0.
 ↪005)
def get_lr_schedule():
    lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 2e-9 * 10**(epoch / 20))
    return lr_schedule

def get_test_accuracy(model, x_test, y_test):
    test_loss, test_mse, test_mae = model.evaluate(x=x_test, y=y_test,␣
 ↪verbose=0)
    print('losses: {acc:0.3f}'.format(acc=test_loss))
    print('mse: {acc:0.3f}'.format(acc=test_mse))
    print('mae: {acc:0.3f}'.format(acc=test_mae))
    predictions = model.predict(x_test)
    print('deviation:', abs((sum(predictions) - sum(y_test))) / sum(y_test),␣
 ↪'%')
    plt.plot(y_test, label='           ')
    plt.plot(predictions, label='            ')
    #plt.plot(val_loss, label='     ')
    plt.legend()
    plt.show()
def get_test_results(model, x_test, y_test):
    test_loss, test_mse, test_mae = model.evaluate(x=x_test, y=y_test,␣
 ↪verbose=0)
    predictions = model.predict(x_test, verbose = 0)
    return test_loss, test_mse, test_mae, (abs((sum(predictions) -␣
 ↪sum(y_test))) / sum(y_test))
```

[ ]:
```
features = ['y_mix', 'y_wo', 'coef']

#
look_back = 10  #
df_forFCST['y_mix_lag1'] = df_forFCST['y_mix'].shift(1)
df_forFCST['y_wo_lag1'] = df_forFCST['y_wo'].shift(1)
df_forFCST['coef_lag1'] = df_forFCST['coef'].shift(1)
features.extend(['y_mix_lag1', 'y_wo_lag1', 'coef_lag1'])

#
df_forFCST.dropna(inplace=True)

scaler = MinMaxScaler()
df_forFCST[features] = scaler.fit_transform(df_forFCST[features])
print(len(df_forFCST))
#
train_size = int(len(df_forFCST) * 0.9)
train_data = df_forFCST[:train_size]
val_data = df_forFCST[train_size:train_size + int(len(df_forFCST) * 0.05)]
```

```
test_data = df_forFCST[train_size + int(len(df_forFCST) * 0.05):]



#
def create_dataset(data, look_back):
    X, Y = [], []
    for i in range(len(data) - look_back - 1):
        a = data[i:(i + look_back), :]
        X.append(a)
        Y.append(data[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 10
X_train, y_train = create_dataset(train_data[features].values, look_back)
X_val, y_val = create_dataset(val_data[features].values, look_back)
X_test, y_test = create_dataset(test_data[features].values, look_back)
```

2007

[ ]:
```
input_shape=(X_train.shape[1], X_train.shape[2])

checkpoint_every_epoch = get_checkpoint_every_epoch()
checkpoint_best_only = get_checkpoint_best_only()
early_stopping = get_early_stopping()
#lr_shed = get_lr_schedule()
callbacks = [checkpoint_every_epoch, checkpoint_best_only, early_stopping]
```

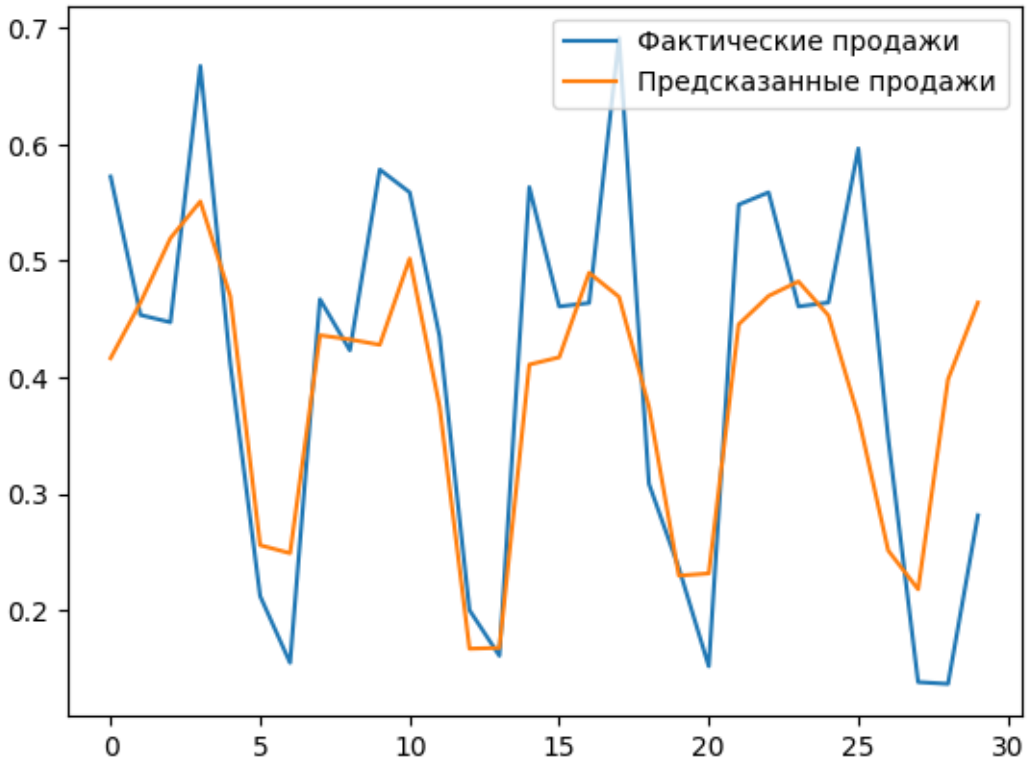
[ ]:
```
model = get_model(input_shape)
get_compile(model)
model.summary()

batchs = 50

history = model.fit(X_train, y_train, epochs = 100, verbose = 1,␣
 ↪validation_data = (X_val, y_val), batch_size = batchs)
```

[ ]:
```
get_test_accuracy(model, X_test, y_test)
#get_test_accuracy(model, X_test[-31: -1], y_test[-31:-1])
#get_test_accuracy(model, X_test[-61: -31], y_test[-61:-31])
#get_test_accuracy(model, X_test[-91: -61], y_test[-91:-61])
#get_test_accuracy(model, X_test[-121: -91], y_test[-121:-91])
get_test_accuracy(model, X_test[0: 30], y_test[0: 30])
get_test_accuracy(model, X_test[30: 60], y_test[30: 60])
get_test_accuracy(model, X_test[60: 90], y_test[60: 90])
```

losses: 1.479

```
mse: 0.116
mae: 0.025
3/3             0s 35ms/step
deviation: [0.00654752] %
```


```
losses: 1.473
mse: 0.086
mae: 0.012
1/1             0s 23ms/step
deviation: [0.04497806] %
```

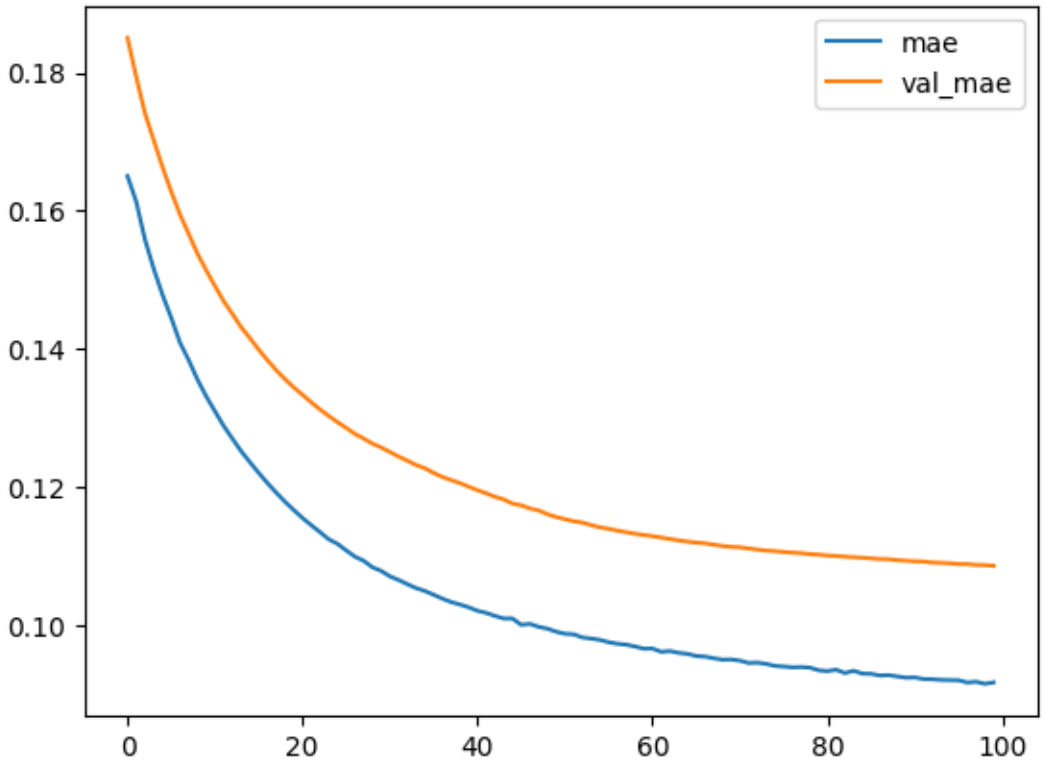



```
losses: 1.489
mse: 0.168
mae: 0.046
1/1             0s 35ms/step
deviation: [0.03428706] %
```


```
losses: 1.475
mse: 0.096
mae: 0.016
1/1             0s 39ms/step
deviation: [0.0260261] %
```

[ ]:
```
df = pd.DataFrame(history.history)
df.plot(y=['mae', 'val_mae'])
```

[ ]: <Axes: >



[ ]:
```
batchsi = [32, 50, 64, 96]
lratesi = [1e-4, 3e-4, 5e-4, 8e-4, 1e-3, 3e-3, 6e-3]
weedi = [5e-4, 8e-4, 1e-3, 2e-3, 3e-3, 5e-3, 8e-3]
bests = []
```

[ ]:
```
for batchs in batchsi:
    for lrate in lratesi:
        for weed in weedi:
            model = get_model(input_shape, weed)
            get_compile(model, lrate)
            history = model.fit(X_train, y_train, epochs=50, verbose = 0,␣
 ↪batch_size = batchs)
            test_loss, test_mse, test_mae, dev = get_test_results(model,␣
 ↪X_test, y_test)
            test_loss1, test_mse1, test_mae1, dev1 = get_test_results(model,␣
 ↪X_test[-31: -1], y_test[-31:-1])
            test_loss2, test_mse2, test_mae2, dev2 = get_test_results(model,␣
 ↪X_test[-61: -31], y_test[-61:-31])
            test_loss3, test_mse3, test_mae3, dev3 = get_test_results(model,␣
 ↪X_test[-91: -61], y_test[-91:-61])
```

```
            print(batchs, lrate, weed)
            print(get_test_results(model, X_test, y_test))
            if max(dev1, dev2, dev3) < 0.05:
                print(get_test_results(model, X_test[0: 30], y_test[0:30]))
                print(get_test_results(model, X_test[30: 60], y_test[30:60]),␣
 ↪end = '    ')
                print(get_test_results(model, X_test[60: 90], y_test[60:90]))
                bests.append((batchs, lrate, weed))
```

[ ]:
```
print(bests)
#batch_size lerning_rate (weight_decay)
#24 1e-4 in 9%
#50 5e-4 in 9%
#64 5e-4 in 9%
#64 5e-4 100ep 1e-3wd in 5% *
#64 5e-4 100ep 2e-3wd in 5% *
#with wd:
#32 0.0002 3e-3 in 3 % **
#50 0.0005 2e-3 in 3 % **
#64 0.0002 8e-4 in 4 %
#with henorm
#32 1e-4 3e-3 in 2% ***
#32 3e-4 5e-4 in 5% *
#50 1e-4 5e-4 in 5% *
#50 1e-4 5e-3 in 4% **
#64 1e-4 1e-3 in 4% **
```

[ ]:
```
model1 = get_model(input_shape)  # Create a new model
get_compile(model1)
model1.load_weights('/content/gdrive/MyDrive/var/FTryModel/
 ↪checkpoints_best_only/checkpoint.weights.h5')
get_test_accuracy(model1, X_test, y_test)
get_test_accuracy(model1, X_test[-31: -1], y_test[-31:-1])
```

[ ]:
```
# Define the learning rate array (adjust to match the number of epochs)
lrs = 2e-9 * (10 ** (np.arange(len(history.history['loss'])) / 20))

# Set the figure size
plt.figure(figsize=(10, 6))

# Set the grid
plt.grid(True)

# Plot the loss in log scale
plt.semilogx(lrs[90:180], history.history["loss"][90:180])

# Increase the tickmarks size
```

```
plt.tick_params('both', length=10, width=1, which='both')

# Set the plot boundaries
plt.xlabel("Learning Rate")
plt.ylabel("Loss")
plt.title("Learning Rate vs Loss")
plt.show()
```