DeepLearning_Q6_AmirPourmand

March 12, 2022

Name: Amir Pourmand ID: 99210259

In this notebook, we will be building and training LSTM and GRU to predict the stock market. You do not allow to use TensorFlow and Keras libraries.

0.1 1. Libraries and settings

[ ]:
```
# Import libraries
# Notice that it is important that which libraries you use, so you should import
# libraries just here in your code
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
from pandas import datetime
import math, time
import itertools
import datetime
from operator import itemgetter
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from math import sqrt
```

0.2 2. Load data

[ ]:
```
# Read data
df = pd.read_csv('NVDA.csv')
df
```

[ ]:
```
# Plot close prices ("<CLOSE>") based on dates ("<DATE>")

######################
data=pd.read_csv('NVDA.csv',index_col=2,parse_dates=True)
data[['<CLOSE>']].plot()
# your code
```

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

[ ]:
```
# function to create train, test data given stock data and sequence length
def load_data(stock, look_back):
    data_raw = stock.values # convert to numpy array
    data = []

    for index in range(len(data_raw) - look_back):
        data.append(data_raw[index: index + look_back])

    data = np.array(data);
    test_set_size = int(np.round(0.2*data.shape[0]));
    train_set_size = data.shape[0] - (test_set_size);

    x_train = data[:train_set_size,:-1,:]
    y_train = data[:train_set_size,-1,:]

    x_test = data[train_set_size:,:-1]
    y_test = data[train_set_size:,-1,:]

    return [x_train, y_train, x_test, y_test]

look_back = 60 # choose sequence length
x_train, y_train, x_test, y_test = load_data(df, look_back)
print('x_train.shape = ',x_train.shape)
print('y_train.shape = ',y_train.shape)
print('x_test.shape = ',x_test.shape)
print('y_test.shape = ',y_test.shape)
```

0.3 3. Build the structure of models

[ ]:
```
# Build model
#####################
# you can change these parameters to get better result
input_dim = 1
hidden_dim = 32
num_layers = 2
output_dim = 1


# Here we define our model as a class
class LSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
```

```
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)

        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out


class GRU(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_layers, output_dim):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.num_layers = num_layers
        self.gru = nn.GRU(input_dim, hidden_dim, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_dim)

        out, (hn, cn) = self.gru(x, (h0.detach(), c0.detach()))
        out = self.fc(out[:, -1, :])
        return out
```

[ ]:
```
# Train models
model = LSTM(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=output_dim,␣
 ↪num_layers=num_layers)

loss_fn = torch.nn.MSELoss()

optimiser = torch.optim.Adam(model.parameters(), lr=0.01)
print(model)
print(len(list(model.parameters())))
for i in range(len(list(model.parameters()))):
    print(list(model.parameters())[i].size())
num_epochs = 100
hist = np.zeros(num_epochs)

seq_dim =look_back-1

for t in range(num_epochs):
```

```
    y_train_pred = model(x_train)

    loss = loss_fn(y_train_pred, y_train)
    if t % 10 == 0 and t !=0:
        print("Epoch ", t, "MSE: ", loss.item())
    hist[t] = loss.item()

    optimiser.zero_grad()

    loss.backward()

    optimiser.step()
# your code

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

[ ]:
```
# Plot loss based on epochs

plt.plot(hist, label="Training loss")
plt.legend()
plt.show()
```

[ ]:
```
# make predictions

######################
# make predictions
y_test_pred = model(x_test)

# invert predictions
y_train_pred = scaler.inverse_transform(y_train_pred.detach().numpy())
y_train = scaler.inverse_transform(y_train.detach().numpy())
y_test_pred = scaler.inverse_transform(y_test_pred.detach().numpy())
y_test = scaler.inverse_transform(y_test.detach().numpy())



# your code

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


# Calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(y_train[:,0], y_train_pred[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(y_test[:,0], y_test_pred[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
######################
```

```
# your code

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

[ ]:
```
# Visualising the prediction results and compare LSTM and GRU models

######################
# Visualising the results
figure, axes = plt.subplots(figsize=(15, 6))
axes.xaxis_date()

axes.plot(df_ibm[len(df_ibm)-len(y_test):].index, y_test, color = 'red', label␣
↪= 'Real')
axes.plot(df_ibm[len(df_ibm)-len(y_test):].index, y_test_pred, color = 'blue',␣
↪label = 'Predicted')
#axes.xticks(np.arange(0,394,50))
plt.show()
# your code

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

Bonus (5%)

try denoising techniques and train models again after denoising the prices. Then compare the results with previous step and explain how much improvment you can make by denoising data.

[ ]:
```

```