DSDE-Homework-5_Time_series_forecasting_DataInGD

ใช้ LSTM และ GRU ในการ predict ราคาหุ้น
โดย input เป็นราคา GOOG ใน csv file

โหลด ข้อมูลจาก csv

```
# Load data
df = pd.read_csv('GOOG.csv', index_col="Date")
df = df.drop(['Adj Close'], axis=1)
```

ดู row

```
df.loc['2019-02-26']
```

```
>>
Open      1.105750e+03
High      1.119510e+03
Low       1.099920e+03
Close     1.115130e+03
Volume    1.471300e+06
Name: 2019-02-26, dtype: float64
```

ใช้ plotly ในการสร้างกราฟแสดงข้อมูล

```
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"

plot_template = dict(
    layout=go.Layout({
        "font_size": 18,
        "xaxis_title_font_size": 24,
        "yaxis_title_font_size": 24})
)

fig = px.line(df['Open'], labels=dict(
    created_at="Date", value="Open", variable="Sensor"
))
fig.update_layout(
  template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")
)
fig.show()
```

สร้าง target value

```
target_col = "Open"
features = list(df.columns.difference([target_col]))

forecast_lead = 1
target = f"{target_col}_lead{forecast_lead}"

df[target] = df[target_col].shift(-forecast_lead)
df = df.iloc[:-forecast_lead]
```

แบ่ง test val test

```
test_start = "2019-01-01"
val_start = "2018-01-01"

df_train = df.loc[:val_start].copy()
```

```
df_val = df.loc[val_start:test_start].copy()
df_test = df.loc[test_start:].copy()

print("Test set fraction:", len(df_test) / len(df))
```

```
>> Test set fraction: 0.050157563025210086
```

ทำ standardize

```
target_mean = df_train[target].mean()
target_stdev = df_train[target].std()

for c in df_train.columns:
    mean = df_train[c].mean()
    stdev = df_train[c].std()

    df_train[c] = (df_train[c] - mean) / stdev
    df_val[c] = (df_val[c] - mean) / stdev
    df_test[c] = (df_test[c] - mean) / stdev
```

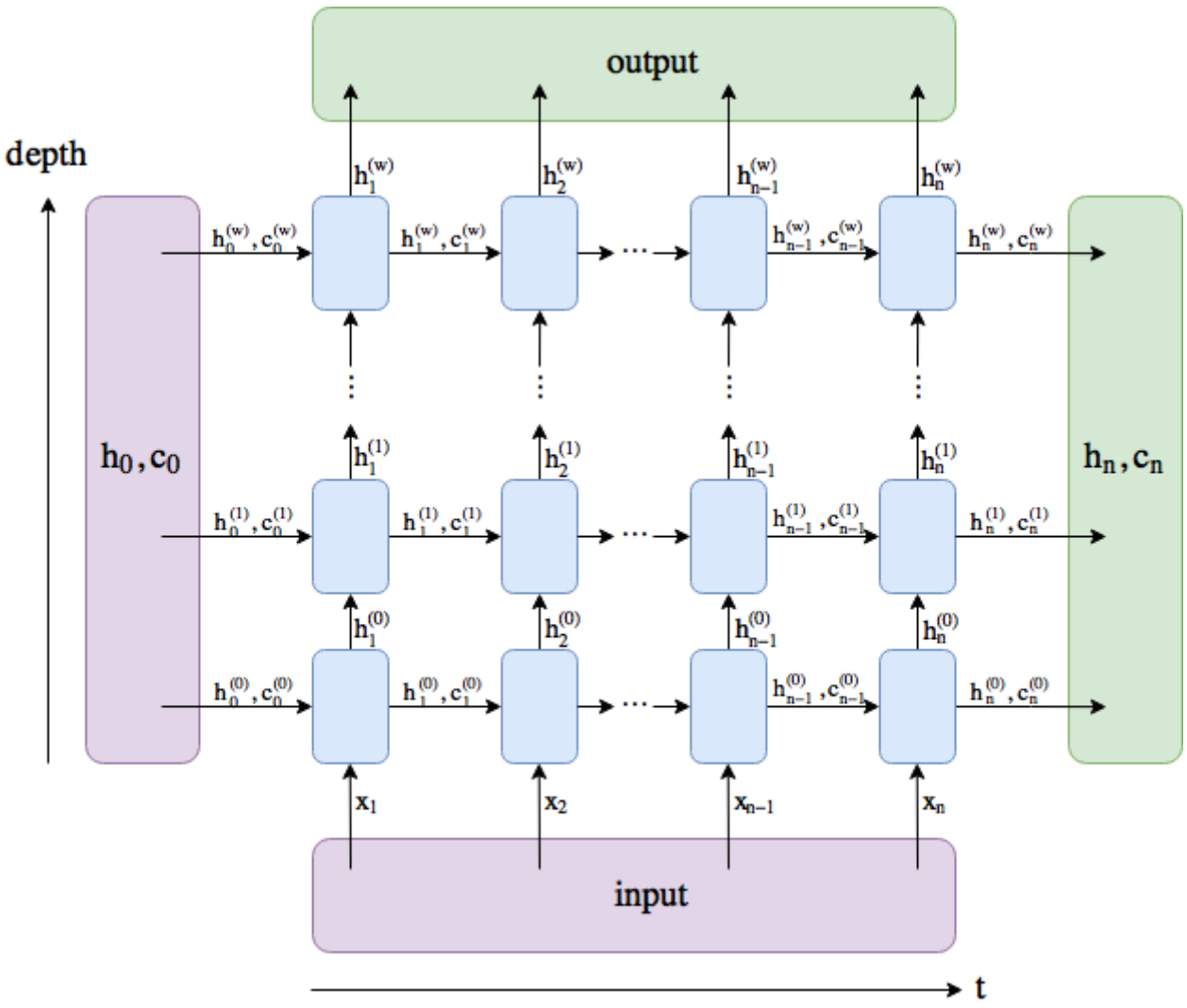
สร้าง DataLoader class สำหรับ PyTorch

```
import torch
from torch.utils.data import Dataset

class SequenceDataset(Dataset):
    def __init__(self, dataframe, target, features, sequence_length=5):
        self.features = features
        self.target = target
        self.sequence_length = sequence_length
        self.y = torch.tensor(dataframe[self.target].values).float()
        self.X = torch.tensor(dataframe[self.features].values).float()

    def __len__(self):
        return self.X.shape[0]

    def __getitem__(self, i):
        if i >= self.sequence_length - 1:
            i_start = i - self.sequence_length + 1
            x = self.X[i_start:(i + 1), :]
        else:
            padding = self.X[0].repeat(self.sequence_length - i - 1, 1)
            x = self.X[0:(i + 1), :]
            x = torch.cat((padding, x), 0)

        return x, self.y[i]
```

นำ data loader ที่สร้างมาใช้กับ train val test

```
torch.manual_seed(101)

batch_size = 32
sequence_length = 60

train_dataset = SequenceDataset(
    df_train,
    target=target,
    features=features,
    sequence_length=sequence_length
)
val_dataset = SequenceDataset(
    df_val,
    target=target,
    features=features,
    sequence_length=sequence_length
)
test_dataset = SequenceDataset(
    df_test,
```

```
    target=target,
    features=features,
    sequence_length=sequence_length
)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

X, y = next(iter(train_loader))

print("Features shape:", X.shape)
print("Target shape:", y.shape)
```



เลือก device เป็น GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

สร้าง ShallowRegressionLSTM

```
from torch import nn

class ShallowRegressionLSTM(nn.Module):
    def __init__(self, num_features, hidden_units):
        super().__init__()
        self.num_features = num_features  # this is the number of features
        self.hidden_units = hidden_units
        self.num_layers = 4

        self.lstm = nn.LSTM(
            input_size=num_features,
            hidden_size=hidden_units,
            batch_first=True,
            num_layers=self.num_layers
        )

        self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)

    def forward(self, x):
        batch_size = x.shape[0]

        # initialize the hidden and cell state of the LSTM layer
        h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()
```

```
        c0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()

        _, (hn, _) = self.lstm(x, (h0, c0))
        out = self.linear(hn[-1]).flatten()  # get the output of the last hidden layer
        return out
```

กำหนด parameter

```
learning_rate = 5e-4

num_hidden_units = 60



model = ShallowRegressionLSTM(num_features=len(features), hidden_units=num_hidden_units)

model.to(device)

loss_function = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

print summary

```
from torchinfo import summary
summary(model, input_size=(32, 60, 4))
```

```
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
ShallowRegressionLSTM                              [32]                      --
├─LSTM: 1-1                                        [32, 60, 60]              103,680
├─Linear: 1-2                                      [32, 1]                   61
==========================================================================================
Total params: 103,741
Trainable params: 103,741
Non-trainable params: 0
Total mult-adds (M): 199.07
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.92
Params size (MB): 0.41
Estimated Total Size (MB): 1.37
==========================================================================================
```

เริ่ม train model

```
from tqdm.notebook import tqdm
def train_model(data_loader, model, loss_function, optimizer):
    num_batches = len(data_loader)
    total_loss = 0
    model.train()

    for X, y in data_loader:
        X = X.to(device)
        y = y.to(device)
        output = model(X)
        loss = loss_function(output, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / num_batches
    print(f"Train loss: {avg_loss}")
```

```python
def test_model(data_loader, model, loss_function, best_val_loss):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
        for X, y in data_loader:
            X = X.to(device)
            y = y.to(device)
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    if avg_loss < best_val_loss:
        best_val_loss = avg_loss
        torch.save(model.state_dict(), 'model.pth')
        print('Save new best model')
    return best_val_loss
```

Evaluation

```python
def predict(data_loader, model):

"""Just like `test_loop` function but keep track of the outputs instead of the loss

function.

"""

output = torch.tensor([])

model.eval()

with torch.no_grad():

for X, _ in data_loader:

X = X.to(device)

y_star = model(X)

output = torch.cat((output, y_star.detach().cpu()), 0)


return output
```

โหลด model ที่ train เสร็จแล้ว

```python
PATH = './model.pth'
model.load_state_dict(torch.load(PATH))
```

predict ผลลัพธ์

```python
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)

ystar_col = "Model forecast"
df_train[ystar_col] = predict(train_eval_loader, model).numpy()
df_val[ystar_col] = predict(val_loader, model).numpy()
df_test[ystar_col] = predict(test_loader, model).numpy()

df_out = pd.concat((df_train, df_val, df_test))[[target, ystar_col]]

for c in df_out.columns:
    df_out[c] = df_out[c] * target_stdev + target_mean
```

```
print(df_out)
```

```
            Open_lead1  Model forecast
Date
2004-08-19    50.316402       52.646271
2004-08-20    55.168217       52.799896
2004-08-23    55.412300       54.844269
2004-08-24    52.284027       53.112549
2004-08-25    52.279045       52.587830
...                 ...             ...
2019-09-27  1220.969971     1203.750732
2019-09-30  1219.000000     1200.191284
2019-10-01  1196.979980     1195.909912
2019-10-02  1180.000000     1181.464844
2019-10-03  1191.890015     1176.163696
```

print evaluation metrics

```
import numpy as np

import math

from sklearn.metrics import mean_squared_error


def MAPE(Y_actual,Y_Predicted):

mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100

return mape


print( 'MPAE =', MAPE(df_test['Open_lead1'], df_test['Model forecast']) )

print( 'RMSE =', math.sqrt(mean_squared_error(df_test['Open_lead1'], df_test['Model forecast'])) )
```

```
>> MPAE = 1.9260324553098662 RMSE = 0.08593983988067332
```

plot กราฟ forecase

```
fig = px.line(df_out, labels={'value': "Open", 'created_at': 'Date'})

fig.add_vline(x=val_start, line_width=4, line_dash="dash")

fig.add_vline(x=test_start, line_width=4, line_dash="dash")

# fig.add_annotation(xref="paper", x=0.75, yref="paper", y=0.8, text="Test set start", showarrow=False)

fig.update_layout(

template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")

)
```

```
fig.show()
```

train GRU

```
from torch import nn

class ShallowRegressionGRU(nn.Module):
    def __init__(self, num_features, hidden_units):
        super().__init__()
        self.num_features = num_features  # this is the number of features
        self.hidden_units = hidden_units
        self.num_layers = 4

        self.gru = nn.GRU(
            input_size=num_features,
            hidden_size=hidden_units,
            batch_first=True,
            num_layers=self.num_layers
        )

        self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)

    def forward(self, x):
        batch_size = x.shape[0]

        # initialize the hidden and cell state of the LSTM layer
        h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()

        _, hn = self.gru(x, h0)
        out = self.linear(hn[-1]).flatten()  # get the output of the last hidden layer
        return out
```

ประกาศ parameters

```
learning_rate = 5e-4

num_hidden_units = 60



model = ShallowRegressionGRU(num_features=len(features), hidden_units=num_hidden_units)

model.to(device)

loss_function = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

summary model

```
from torchinfo import summary

summary(model, input_size=(32, 60, 4))
```

```
==========================================================================================
Layer (type:depth-idx)                             Output Shape              Param #
==========================================================================================
ShallowRegressionGRU                               [32]                      --
├─GRU: 1-1                                         [32, 60, 60]              77,760
├─Linear: 1-2                                      [32, 1]                   61
==========================================================================================
Total params: 77,821
Trainable params: 77,821
Non-trainable params: 0
Total mult-adds (M): 149.30
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.92
Params size (MB): 0.31
Estimated Total Size (MB): 1.26
==========================================================================================
```

train GRU

```
def train_model(data_loader, model, loss_function, optimizer):
    num_batches = len(data_loader)
    total_loss = 0
    model.train()

    for X, y in data_loader:
        X = X.to(device)
        y = y.to(device)
        output = model(X)
        loss = loss_function(output, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / num_batches
    print(f"Train loss: {avg_loss}")

def test_model(data_loader, model, loss_function, best_val_loss):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
        for X, y in data_loader:
            X = X.to(device)
            y = y.to(device)
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    if avg_loss < best_val_loss:
        best_val_loss = avg_loss
        torch.save(model.state_dict(), 'model_gru.pth')
        print('Save new best model')
    return best_val_loss
```

ประกาศ function predict

```
def predict(data_loader, model):

"""Just like `test_loop` function but keep track of the outputs instead of the loss

function.

"""

output = torch.tensor([])

model.eval()

with torch.no_grad():

for X, _ in data_loader:

X = X.to(device)

y_star = model(X)

output = torch.cat((output, y_star.detach().cpu()), 0)


return output
```

โหลด GRU ที่ train แล้ว

```
PATH = './model_gru.pth'

model.load_state_dict(torch.load(PATH))
```

predict

```
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)

ystar_col = "Model forecast"
df_train[ystar_col] = predict(train_eval_loader, model).numpy()
df_val[ystar_col] = predict(val_loader, model).numpy()
df_test[ystar_col] = predict(test_loader, model).numpy()

df_out = pd.concat((df_train, df_val, df_test))[[target, ystar_col]]

for c in df_out.columns:
    df_out[c] = df_out[c] * target_stdev + target_mean

print(df_out)
```

print evaluation metrics

```
import numpy as np

import math

from sklearn.metrics import mean_squared_error


def MAPE(Y_actual,Y_Predicted):

mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100

return mape


print( 'MPAE =', MAPE(df_test['Open_lead1'], df_test['Model forecast']) )
```

```
print( 'RMSE =', math.sqrt(mean_squared_error(df_val['Open_lead1'], df_val['Model forecast'])) )
```

```
>> MPAE = 1.9992090562917835 RMSE = 0.08901432835864326
```

plot graph

```
fig = px.line(df_out, labels={'value': "Open", 'created_at': 'Date'})

fig.add_vline(x=val_start, line_width=4, line_dash="dash")

fig.add_vline(x=test_start, line_width=4, line_dash="dash")

# fig.add_annotation(xref="paper", x=0.75, yref="paper", y=0.8, text="Test set start", showarrow=False)

fig.update_layout(

template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")

)

fig.show()
```