import torch

import torch.nn as nn

import torch.optim as optim

import torch.nn.functional as F

import numpy as np

import pandas as pd

from torch.utils.data import DataLoader, TensorDataset

import csv

import random

class MLP(nn.Module):

def \_\_init\_\_(self):

super(MLP, self).\_\_init\_\_()

self.fc1 = nn.Linear(9, 128)

self.fc2 = nn.Linear(128, 64)

self.dropout = nn.Dropout(p=0.2) # Dropout probability

self.output = nn.Linear(64, 1)

self.relu = nn.ReLU()

def forward(self, x):

x = self.relu(self.fc1(x))

x = self.dropout(x) # Enable Dropout

x = self.relu(self.fc2(x))

x = self.dropout(x)

x = self.output(x)

return x

# Monte Carlo Dropout Prediction Function

def mc\_dropout\_predict(model, data, mc\_samples=50):

model.train() # Enable dropout

predictions = []

# Conduct mc\_samples predictions

for \_ in range(mc\_samples):

with torch.no\_grad():

pred = model(data)

predictions.append(pred.cpu().numpy()) # Convert to a numpy array to facilitate subsequent calculations

predictions = np.array(predictions) # Convert to a NumPy array (mc\_samples, N, 1)

mean\_pred = predictions.mean(axis=0).squeeze()

var\_pred = predictions.var(axis=0).squeeze()

return mean\_pred, var\_pred

# Select the 5% of the samples with the highest level of uncertainty

def select\_high\_uncertainty\_samples(mean\_pred, var\_pred, percentage=0.05):

# Sort and obtain the sample with the highest uncertainty

uncertainty\_sorted\_idx = np.argsort(var\_pred)[::-1] # Sort in descending order of variance

top\_k = int(len(var\_pred) \* percentage)

selected\_samples\_idx = uncertainty\_sorted\_idx[:top\_k]

return selected\_samples\_idx

# Expand the training set

def augment\_training\_data(train\_data, test\_data, selected\_samples\_idx):

# Select the sample with the highest level of uncertainty from the data pool that has never been seen before

test\_data\_np = test\_data.cpu().numpy()

selected\_samples = test\_data\_np[selected\_samples\_idx].copy() # Use .copy() to avoid the problem of negative strides

# Convert the selected samples back to Tensor and add them back to the training set

selected\_samples\_tensor = torch.tensor(selected\_samples, dtype=torch.float32)

new\_train\_data = torch.cat((train\_data, selected\_samples\_tensor), dim=0)

return new\_train\_data

# Expand training labels

def augment\_training\_labels(train\_labels, test\_labels, selected\_samples\_idx):

# Obtain the corresponding labels from the test set

test\_labels\_np = test\_labels.cpu().numpy()

selected\_labels = test\_labels\_np[selected\_samples\_idx].copy()

# Convert the selected labels back to Tensor and add them back to the training label set

selected\_labels\_tensor = torch.tensor(selected\_labels, dtype=torch.float32)

new\_train\_labels = torch.cat((train\_labels, selected\_labels\_tensor), dim=0)

return new\_train\_labels

# train model

def train\_model(model, train\_data, train\_labels, num\_epochs=50, learning\_rate=0.01):

optimizer = optim.Adam(model.parameters(), lr=learning\_rate)

criterion = nn.MSELoss()

model.train()

loss\_history = []

for epoch in range(num\_epochs):

optimizer.zero\_grad()

output = model(train\_data)

loss = criterion(output, train\_labels)

loss.backward()

optimizer.step()

loss\_history.append(loss.item()) # Record the loss of the current epoch

# Convert the loss history into a DataFrame

loss\_df = pd.DataFrame(loss\_history, columns=['Loss'])

print(f'Epoch {epoch + 1}/{num\_epochs}, Loss: {loss.item()}')

return model, loss\_df

# Main function

def iterative\_training(train\_data, train\_labels, test\_data, test\_labels, num\_iterations=10, mc\_samples=50):

model = MLP()

# Re-train the model

model, loss\_df = train\_model(model, train\_data, train\_labels)

loss\_df.to\_csv(f'Model training loss\\Active learning high-fidelity model training loss\_ Without active learning.csv',index=False, encoding='GBK')

# save model

torch.save(model.state\_dict(),f"The model generated during training\model\_iteration\_ when no active learning was employed.pth")

for i in range(num\_iterations):

print(f"Iteration {i + 1}/{num\_iterations}")

# Perform uncertainty estimation using Monte Carlo Dropout

mean\_pred, var\_pred = mc\_dropout\_predict(model, test\_data, mc\_samples)

results\_df = pd.DataFrame({

'mean\_pred': mean\_pred.squeeze().tolist(),

'var\_pred': var\_pred.squeeze().tolist()

})

results\_df.to\_csv(f"\The sample mean and variance in each iteration of active learning\Sample mean and variance\_ Round{i + 1}.csv", index=False, encoding='GBK')

# Select the 5% of the samples with the highest level of uncertainty

selected\_samples\_idx = select\_high\_uncertainty\_samples(mean\_pred, var\_pred, percentage=0.05)

df\_selected\_samples = pd.DataFrame(selected\_samples\_idx, columns=['Sample\_Index'])

df\_selected\_samples.to\_csv(f"\The sequence number of the data selected for the validation set each time\The selected highly uncertain sample\_Round{i+1}.csv",index=False, encoding='GBK')

# Expand the training set and training labels from the test set

train\_data = augment\_training\_data(train\_data, test\_data, selected\_samples\_idx)

train\_labels = augment\_training\_labels(train\_labels, test\_labels, selected\_samples\_idx)

train\_data\_np = train\_data.cpu().numpy()

train\_labels\_np = train\_labels.cpu().numpy()

train\_data\_and\_label\_df = pd.DataFrame(train\_data\_np)

train\_data\_and\_label\_df['labels'] = train\_labels\_np

train\_data\_and\_label\_df.to\_csv(f'Each training set\\ The (i + 1)th training set.csv', index=False)

model, loss\_df = train\_model(model, train\_data, train\_labels)

loss\_df.to\_csv(f'\Active learning for high-fidelity model training\_loss for the {i + 1}th round.csv',index=False, encoding='GBK')

# save model

torch.save(model.state\_dict(), f"\The model generated during training\model\_iteration\_{i + 1}.pth")

return model

train\_df = pd.read\_csv('train\_data.csv', encoding='GBK')

test\_df = pd.read\_csv('val\_data.csv',encoding='GBK')

train\_data = train\_df.iloc[:, :-2] # The last column is the label

train\_labels = train\_df.iloc[:, -2]

test\_data = test\_df.iloc[:, :-2] # The last column is the label

test\_labels = test\_df.iloc[:, -2]

train\_data = torch.tensor(train\_data.values, dtype=torch.float32)

train\_labels = torch.tensor(train\_labels.values, dtype=torch.float32).view(-1, 1)

test\_data = torch.tensor(test\_data.values, dtype=torch.float32)

test\_labels = torch.tensor(test\_labels.values, dtype=torch.float32).view(-1, 1)

model = iterative\_training(train\_data, train\_labels, test\_data, test\_labels)