import torch

import torch.nn as nn

from torch.utils.data import TensorDataset, DataLoader

from torchmetrics import Accuracy, Precision, Recall

batch\_size = 400

train\_loader = DataLoader(train\_data, shuffle=True, batch\_size=batch\_size)

test\_loader = DataLoader(test\_data, shuffle=False, batch\_size=batch\_size)

# Define the classifier class

class TicketClassifier(nn.Module):

def \_\_init\_\_(self, vocab\_size, embed\_dim, target\_size):

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

self.embedding = nn.Embedding(vocab\_size, embed\_dim)

self.conv = nn.Conv1d(embed\_dim, embed\_dim, kernel\_size=3, stride=1, padding=1)

self.fc = nn.Linear(embed\_dim, target\_size)

def forward(self, text):

embedded = self.embedding(text).permute(0, 2, 1)

conved = F.relu(self.conv(embedded))

conved = conved.mean(dim=2)

return self.fc(conved)

vocab\_size = len(word2idx) + 1

target\_size = len(np.unique(labels))

embedding\_dim = 64

# Create an instance of the TicketClassifier class

model = TicketClassifier(vocab\_size, embedding\_dim, target\_size)

lr = 0.05

criterion = nn.CrossEntropyLoss()

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

epochs = 3

# Train the model

model.train()

for i in range(epochs):

running\_loss, num\_processed = 0,0

for inputs, labels in train\_loader:

model.zero\_grad()

output = model(inputs)

loss = criterion(output, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

num\_processed += len(inputs)

print(f"Epoch: {i+1}, Loss: {running\_loss/num\_processed}")

accuracy\_metric = Accuracy(task='multiclass', num\_classes=5)

precision\_metric = Precision(task='multiclass', num\_classes=5, average=None)

recall\_metric = Recall(task='multiclass', num\_classes=5, average=None)

# Evaluate model on test set

model.eval()

predicted = []

for i, (inputs, labels) in enumerate(test\_loader):

output = model(inputs)

cat = torch.argmax(output, dim=-1)

predicted.extend(cat.tolist())

accuracy\_metric(cat, labels)

precision\_metric(cat, labels)

recall\_metric(cat, labels)

accuracy = accuracy\_metric.compute().item()

precision = precision\_metric.compute().tolist()

recall = recall\_metric.compute().tolist()

print('Accuracy:', accuracy)

print('Precision (per class):', precision)

print('Recall (per class):', recall)