Nickhil Tekwani || CS6220 || Hw 4

```
import torch
print(torch.__version__)
```

```
2.0.1+cu118
```

```
import tensorflow as tf
print(tf.__version__)
```

```
2.12.0
```

Problem 2

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader, random_split

# Download and load the MNIST dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])
mnist = datasets.MNIST(root='./data', download=True, train=True, transform=transform)

# Split the dataset 80/20
train_size = int(0.8 * len(mnist))
test_size = len(mnist) - train_size
train_dataset, test_dataset = random_split(mnist, [train_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 164596962.06it/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 39113882.41it/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz
100%|██████████| 1648877/1648877 [00:00<00:00, 45912497.99it/s]Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./da


Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 11839980.59it/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

```
class MNISTNet(nn.Module):
    def __init__(self):
        super(MNISTNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = nn.ReLU()(x)
        x = self.conv2(x)
        x = nn.ReLU()(x)
        x = nn.MaxPool2d(2)(x)
        x = self.dropout1(x)
```

```
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        output = nn.LogSoftmax(dim=1)(x)
        return output

model = MNISTNet()

optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9, nesterov=True)
criterion = nn.CrossEntropyLoss()
```

```
def train_model(model, train_loader, optimizer, criterion, epochs):
    for epoch in range(epochs):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

# Training for 10 epochs for simplicity
train_model(model, train_loader, optimizer, criterion, 10)
```

```
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            outputs = model(data)
            _, predicted = outputs.max(1)
            total += target.size(0)
            correct += predicted.eq(target).sum().item()

    accuracy = 100 * correct / total
    print(f"Test Accuracy: {accuracy:.2f}%")

evaluate_model(model, test_loader)
```

```
Test Accuracy: 98.84%
```

Part B

```
import gensim
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors

# File paths
glove_input_file = '/content/glove.6B.100d.txt'
word2vec_output_file = '/content/glove.6B.100d.word2vec.txt'

# Convert GloVe file to word2vec format
glove2word2vec(glove_input_file, word2vec_output_file)

# Load the converted model
glove_model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=False)
```

```
<ipython-input-9-ebc2cb1e96a6>:10: DeprecationWarning: Call to deprecated `glove2word2vec` (KeyedVectors.load_word2vec_forma
  glove2word2vec(glove_input_file, word2vec_output_file)
```

```
MAX_SEQ_LEN = 200

def doc2ind(doc, glove_model):
    indices = [glove_model.key_to_index[word] for word in doc if word in glove_model.key_to_index]
    # pad sequences with zeros if they are shorter than MAX_SEQ_LEN
    padded_indices = indices + [0] * (MAX_SEQ_LEN - len(indices))
    # truncate sequences if they are longer than MAX_SEQ_LEN
```

```
    return padded_indices[:MAX_SEQ_LEN]

from sklearn.datasets import fetch_20newsgroups
from gensim.utils import simple_preprocess
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
import torch.optim as optim


# Fetch a subset of the 20NG dataset
categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)

ng_text = newsgroups_train.data

tokens = [simple_preprocess(text) for text in ng_text]
ng_vector_idx = torch.LongTensor([doc2ind(doc, glove_model) for doc in tokens])

# Prepare the embedding layer
weights = torch.FloatTensor(glove_model.vectors)
glove_emb = nn.Embedding.from_pretrained(weights)
glove_emb.weight.requires_grad = False
```

```
# Assuming num_classes to be the number of categories in the subset of 20NG dataset
num_classes = 5

class TextNet(nn.Module):
    def __init__(self, embedding_layer, num_classes):
        super(TextNet, self).__init__()
        self.embedding = embedding_layer
        self.fc1 = nn.Linear(100, 50)
        self.fc2 = nn.Linear(50, num_classes)

    def forward(self, x):
        x = self.embedding(x)
        x = x.mean(dim=1)
        x = self.fc1(x)
        x = nn.ReLU()(x)
        x = self.fc2(x)
        return nn.LogSoftmax(dim=1)(x)

model_ng = TextNet(glove_emb, num_classes)
optimizer_ng = optim.SGD(model_ng.parameters(), lr=0.01, momentum=0.9, nesterov=True)
criterion_ng = nn.CrossEntropyLoss()
```

```
# Splitting data for training and testing (80/20 split for simplicity)
train_size_ng = int(0.8 * len(ng_vector_idx))
test_size_ng = len(ng_vector_idx) - train_size_ng
train_dataset_ng = TensorDataset(ng_vector_idx[:train_size_ng], torch.tensor(newsgroups_train.target[:train_size_ng]))
test_dataset_ng = TensorDataset(ng_vector_idx[train_size_ng:], torch.tensor(newsgroups_train.target[train_size_ng:]))

train_loader_ng = DataLoader(train_dataset_ng, batch_size=64, shuffle=True)
test_loader_ng = DataLoader(test_dataset_ng, batch_size=64, shuffle=False)

# Training function for the TextNet
def train_model_ng(model, train_loader, optimizer, criterion, epochs):
    for epoch in range(epochs):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

# Evaluating function
def evaluate_model_ng(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            outputs = model(data)
            _, predicted = outputs.max(1)
```

```
            total += target.size(0)
            correct += predicted.eq(target).sum().item()

    accuracy = 100 * correct / total
    print(f"Test Accuracy for 20NG dataset: {accuracy:.2f}%")

# Train for a certain number of epochs (let's say 10 for simplicity)
train_model_ng(model_ng, train_loader_ng, optimizer_ng, criterion_ng, 10)

# Evaluate
evaluate_model_ng(model_ng, test_loader_ng)
```

```
Test Accuracy for 20NG dataset: 29.91%
```