Knowing MLPs

Please read the following before moving on

Welcome to the world of Multilayer Perceptrons (MLP)! You already know that MLPs are feedforward neural networks consisting of multiple layers of nodes or neurons. These networks are well-suited for a wide range of classification and regression tasks, thanks to their ability to learn complex, non-linear relationships between inputs and outputs.

In this assignment, we will be working with the MNIST dataset to explore the importance of different MLP components. The MNIST dataset consists of 70,000 handwritten digit images, each of which is 28x28 pixels in size. Our goal is to use an MLP to classify these images into one of 10 categories (0-9).

To improve the performance of our MLP, we will experiment with various techniques such as Dropout, Batch Normalization, Loss Functions, Stochastic batch and mini-batch gradient descent, and more. Please note, you must use mini-batch unless explicity specified.

In addition, we will experiment with different optimization algorithms such as stochastic gradient descent, Adam, and RMSprop to find the optimal weights and biases for our MLP. We will use stochastic batch and mini-batch gradient descent, which involve updating the weights and biases of the network based on a small batch of randomly sampled training examples, to speed up the training process and reduce memory usage.

By the end of this assignment, you will have gained a deeper understanding of the various components that make up an MLP and their importance in achieving high performance in classification tasks. You will have gained hands-on experience in experimenting with these components and learned how to fine-tune an MLP to achieve the best possible performance on the MNIST dataset. So, let's get started!

👊Pro-tip: Do not re-write any results so as to re-use them in later experiments for tabulation and plotting.

💭Trivia: Did you know code written using ChatGPT is easy to catch?

Step zero: Import Libraries

In [ ]:

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd

import matplotlib
import seaborn as sns
%matplotlib inline

import torch
import torchvision
from torchvision import datasets
from torchvision import transforms
```

```
from torch.autograd import Variable
import torch.nn as nn
```

Step one: Using a PyTorch Dataset

Load MNIST dataset from `torchvision.datasets`

In [ ]:
```
transform = transforms.Compose([
                    transforms.ToTensor(),
            ])

trainset = #TODO use datasets.MNIST
testset = #TODO
```

In [ ]:
```
trainloader_minibatch = torch.utils.data.DataLoader(trainset, batch_size=64
                                                    , shuffle=True, num_workers=2)
trainloader_stochastic = torch.utils.data.DataLoader(trainset, batch_size=1
                                                     , shuffle=True, num_workers=2)
testloader = #TODO
```

In [ ]:
```
pbar = tqdm(total=len(trainloader_minibatch))
for idx, (data,label) in enumerate(trainloader_minibatch):
    print(idx,data.size(),label.size())
    pbar.update(1)
    break
pbar.refresh()
```

Step two: Define a MLP Model and without any bells and whitsles...

... along with a CrossEntropy loss criterion

Do not use Dropout, BN or any other thing. Use ReLU for hidden layers.

⚠️ Do not use SoftMax in the output as nn.CrossEntropyLoss combines SoftMax and NLLLoss.

In [ ]:
```
class SimpleMLP(nn.Module):
    def __init__(self):
        super(SimpleMLP,self).__init__()
        # code here
    def forward(self,x):
        # code here
        return output
```

Step three: Define the following optimizers using nn.optim

1. SGD
2. SGD with momentum
3. SGD with L2 regularization
4. RMSprop
5. Adam

In [ ]:
```
# code here
```

Step four: Run the SimpleMLP using different optimizers and plot train and test loss for each optimizer.

Explain the results.

Report final accuracy, F1 score and other relavant metrics in a tabular form on test and train datasets.

In [ ]:
```
# code, plots and explainiation here
```

In [ ]:
```
# sample code: need not rely on this
EPOCHS = 25

train_loss = []
val_loss = []

mlp.train()

for epoch in range(EPOCHS):
    pbar = tqdm(total=len(trainloader))
    out_loss = 0
    for batch_idx, (data, target) in enumerate(trainloader):
        # do things

        if torch.cuda.is_available():
            data,target = data.cuda(),target.cuda()

        out = mlp(data)
        # do things
        out_loss += loss.cpu().data.item()
        # do things
        pbar.update(1)
        pbar.desc= f'Loss: {loss.item()}'
    train_loss.append(out_loss/len(trainloader))
    with torch.no_grad():
        out_loss = 0
        for batch_idx, (data, target) in enumerate(testloader):
            # do things

        val_loss.append(out_loss/len(testloader))
    print()
    pbar.refresh()
    pbar.close()

plot_losses(train_loss,val_loss)
final_metrics = get_metrics_somehow(mlp,trainloader,testloader)
```

Step five: Using SimpleMLP and Adam optimizer, train models using 2 different lr_schedulers.

Select 2 of **MultiplicativeLR**, **MultiStepLR**, **LinearLR** and **ExponentialLR**

Compare the results among different LR schedulers and the original model which didn't employ any LR scheduler. Compile results in a tabular form. Plot losses for each. Explain results.

In [ ]:
```
# code and analysis here
```

Step six: Define 3 models with following changes:

1. Add BatchNorm
2. Add Dropout
3. Add BatchNorm and Dropout

In [ ]:
```
# code here
```

Step seven: Train the above models and compare with SimpleMLP.

Use your choice of optimizer, use no lr_scheduler so as to re-use the previous results.

Perform analysis. You've got the drill by now.

In [ ]:
```
# code here
```

Step eight: Mini-batch vs Stocastic

Now that you might have a clear winner in your mind regarding which model and settings perform the best, train it on mini-batch and stocastic and compare time taken, loss curve, accuracy etc.

Perform an analysis like never before!

In [ ]:
```
# code here
```

And most importantly!!! What did we learn?

Discuss any and all learnings here. The discussions must be all-encompassing so that we know what did you learn.

Please do not copy from your friend or copy-paste from the internet. We can see repetitions during evaluations.

In [ ]:
```
# text here
```