Statistical and Machine Learning (01.113)
Homework 2

Due on 28 FEB, 3 PM

Problem 1 (2 points)

The entropy of a discrete probability distribution, which is always greater than or equal to zero, is given by

$$\mathrm{Ent}(p) = -\sum_{i=1}^{n} p_i \log p_i, \quad \sum_{i=1}^{n} p_i = 1.$$

(a) Use Lagrange multipliers to find the probability distribution which maximizes entropy.

(b) Which probability distribution minimizes entropy?

Solution. (a) The Lagrangian is given by

$$L(p_1, \ldots, p_n, \lambda) = -\sum_{i=1}^{n} p_i \log p_i + \lambda \left(\sum_{i=1}^{n} p_i - 1 \right).$$

Setting the gradient to zero, we get

$$\frac{\partial L}{\partial p_i} = -\log p_i - 1 + \lambda = 0 \implies \lambda = 1 + \log p_i, \quad \forall i = 1, \ldots, n,$$
$$\frac{\partial L}{\partial \lambda} = \sum_{i=1}^{n} p_i - 1 = 0 \implies \sum_{i=1}^{n} p_i = 1.$$

This means that $p_i = p_j$ for all i, j, and hence the uniform distribution maximizes entropy.

(b) For any i, the distribution $p_i = 1$, $p_j = 0$ for all $j \neq i$ has entropy 0.

■

Problem 2 (2 points)

Let A be an n x n matrix, B be an n x p matrix, C be a p x n matrix and D be a p x p matrix. Show that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix},$$

where

$$M = \left(A - BD^{-1}C\right)^{-1}.$$

Solution. We have

$$\begin{aligned} AM + B(-D^{-1}CM) &= A(A - BD^{-1}C)^{-1} - B\left(D^{-1}C\left(A - BD^{-1}C\right)^{-1}\right) \\ &= \left(A - BD^{-1}C\right)\left(A - BD^{-1}C\right)^{-1} = I_n, \end{aligned}$$

$$\begin{aligned} &A\left(-MBD^{-1}\right) + B\left(D^{-1} + D^{-1}CMBD^{-1}\right) \\ &\quad = A\left(-\left(A - BD^{-1}C\right)^{-1}BD^{-1}\right) + BD^{-1} + BD^{-1}C\left(A - BD^{-1}C\right)^{-1}BD^{-1} \\ &\quad = -\left(A - BD^{-1}C\right)\left(\left(A - BD^{-1}C\right)^{-1}BD^{-1}\right) + BD^{-1} \\ &\quad = -BD^{-1} + BD^{-1} = \mathbf{0}, \end{aligned}$$

$$CM - DD^{-1}CM = CM - CM = \mathbf{0},$$

and

$$C\left(-MBD^{-1}\right) + D\left(D^{-1} + D^{-1}CMBD^{-1}\right) = DD^{-1} = I_p.$$

■

Problem 3 (1 point)

Derive a formula relating $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

Solution.

$$\begin{aligned} \frac{1}{2}\left(\tanh(x) + 1\right) &= \frac{1}{2}\left(\frac{e^x - e^{-x}}{e^x + e^{-x}} + \frac{e^x + e^{-x}}{e^x + e^{-x}}\right) \\ &= \frac{e^x}{e^x + e^{-x}} \\ &= \frac{1}{1 + e^{-2x}} = \sigma(2x). \end{aligned}$$

■

Problem 4 (5 points)

We will use PyTorch to train a Convolutional Neural Network (CNN) to improve classification accuracy on the Fashion MNIST dataset. This dataset comprises 50,000 training examples and 10,000 test examples of 28x28-pixel monochrome images of various clothing items. Let us begin by importing the libraries:

```
import numpy
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
```

There are a total of 10 classes enumerated in the following way:

```
labels = {
        0 : "T-shirt",
        1 : "Trouser",
        2 : "Pullover",
        3 : "Dress",
        4 : "Coat",
        5 : "Sandal",
        6 : "Shirt",
        7 : "Sneaker",
        8 : "Bag",
        9 : "Ankle boot"
         }
```

(a) Define your model by inheriting from the "nn.Module" using the following format:

```
class CNN(nn.Module):
        def __init__(self):
                super(CNN, self).__init__()
                # initialize layers here

        def forward(self, x):
                # invoke the layers here

        return ...
```

(b) Complete the main function below; the test and train functions will be defined in later parts.

```
def main():
        NUM_EPOCHS = # Complete here
        LRATE = # Complete here

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        train_dataset = datasets.FashionMNIST('../data', train=True,
                        download=True, transform=transforms.ToTensor())
        test_dataset = datasets.FashionMNIST('../data', train=False,
                        download=True, transform=transforms.ToTensor())

        ##### Use dataloader to load the datasets
        train_loader = # Complete here
        test_loader  = # Complete here

        model = CNN().to(device)
        optimizer = optim.SGD(model.parameters(), lr=LRATE)

        for epoch in range(1, NUM_EPOCHS + 1):
                test(model, device, test_loader)
                train(model, device, train_loader, optimizer, epoch)

        test(model, device, test_loader)


if __name__ == '__main__':
        main()
```

(c) Complete the training function by defining the model output and the loss function. Use the optimizer's step() function to update the weights after backpropagating the gradients. (Remember to clear the gradients with each iteration.)

```
def train(model, device, train_loader, optimizer, epoch):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
                data, target = data.to(device), target.to(device)

                # Fill in here

                if batch_idx % 100 == 0:
                print('Epoch:', epoch, ',loss:', loss.item())
```

(d) In the test function, define the variable "pred" which predicts the output, and update the variable "correct" to keep track of the number of correctly classified objects so as to compute the accuracy of the CNN.

```
def test(model, device, test_loader):
        model.eval()
        correct = 0
        exampleSet = False
        example_data = numpy.zeros([10, 28, 28])
        example_pred = numpy.zeros(10)

        with torch.no_grad():
                for data, target in test_loader:
                        data, target = data.to(device), target.to(device)

                        # fill in here

                        if not exampleSet:
                                for i in range(10):
                                        example_data[i] = data[i][0].to("cpu").numpy()
                                        example_pred[i] = pred[i].to("cpu").numpy()
                                exampleSet = True

        print('Test set accuracy: ', 100. * correct / len(test_loader.dataset), '%')

                for i in range(10):
                        plt.subplot(2,5,i+1)
                        plt.imshow(example_data[i], cmap='gray', interpolation='none')
                        plt.title(labels[example_pred[i]])
                        plt.xticks([])
                        plt.yticks([])
                plt.show()
```

You must achieve more than 80% accuracy to get full credit. Upload your final script in your Dropbox folder and name it as "**HW2.py**'.

Solution.

```
import numpy
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

labels = {
        0 : "T-shirt",
        1 : "Trouser",
        2 : "Pullover",
        3 : "Dress",
        4 : "Coat",
        5 : "Sandal",
        6 : "Shirt",
        7 : "Sneaker",
        8 : "Bag",
        9 : "Ankle boot"
}

##### Define the CNN
class CNN(nn.Module):
        def __init__(self):
                super(CNN, self).__init__()
                self.conv1 = nn.Conv2d(1, 20, 5, 1)
                self.conv2 = nn.Conv2d(20, 50, 5, 1)

                self.fc1 = nn.Linear(4*4*50, 500)
                self.fc2 = nn.Linear(500, 10)

        def forward(self, x):
                x = F.relu(self.conv1(x))
                x = F.max_pool2d(x, 2, 2)
                x = F.relu(self.conv2(x))
                x = F.max_pool2d(x, 2, 2)

                x = x.view(-1, 4*4*50)
                x = F.relu(self.fc1(x))
                x = self.fc2(x)

                return F.log_softmax(x, dim=1)

##### Training model
def train(model, device, train_loader, optimizer, epoch):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
                data, target = data.to(device), target.to(device)

                optimizer.zero_grad()
                output = model(data)
                loss = F.nll_loss(output, target)
                loss.backward()
                optimizer.step()
                if batch_idx % 100 == 0:
                        print('Epoch:', epoch, ', loss:', loss.item())

##### Testing model
def test(model, device, test_loader):
        model.eval()
        correct = 0
        exampleSet = False
        example_data = numpy.zeros([10, 28, 28])
        example_pred = numpy.zeros(10)
```

```
        with torch.no_grad():
                for data, target in test_loader:
                        data, target = data.to(device), target.to(device)

                        ##### Defining 'pred' and updating 'correct'
                        output = model(data)
                        pred = output.argmax(dim=1, keepdim=True)
                        correct += pred.eq(target.view_as(pred)).sum().item()
                        #####
                        if not exampleSet:
                                for i in range(10):
                                        example_data[i] = data[i][0].to("cpu").numpy()
                                        example_pred[i] = pred[i].to("cpu").numpy()
                                exampleSet = True

        print('Test set accuracy: ', 100. * correct / len(test_loader.dataset), '%')

        for i in range(10):
                plt.subplot(2,5,i+1)
                plt.imshow(example_data[i], cmap='gray', interpolation='none')
                plt.title(labels[example_pred[i]])
                plt.xticks([])
                plt.yticks([])
        plt.show()


def main():
        ##### Choosing epochs and learning rate
        NUM_EPOCHS = 5
        LRATE = 0.01
        #####

        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        train_dataset = datasets.FashionMNIST('../data', train=True,
                        download=True, transform=transforms.ToTensor())
        test_dataset = datasets.FashionMNIST('../data', train=False,
                        download=True, transform=transforms.ToTensor())

        ##### Load the dataset
        train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64,
                        shuffle=True, num_workers=1)
        test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1000,
                        shuffle=True, num_workers=1)
        #####

        model = CNN().to(device)
        optimizer = optim.SGD(model.parameters(), lr=LRATE)

        for epoch in range(1, NUM_EPOCHS + 1):
                test(model, device, test_loader)
                train(model, device, train_loader, optimizer, epoch)

        test(model, device, test_loader)


if __name__ == '__main__':
        main()
```

■