Statistical and Machine Learning (01.113) Homework 2

Adam Ilyas 1002010

March 7, 2019

1 Problem 1

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

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

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

where we solve for the critical points of the Lagrangian

$$\mathcal{L}(p, \lambda) = -\sum_{i=1}^{N} p_i \log(p_i) + \lambda \left(\sum_{i=1}^{N} p_i - 1 \right)$$

Solving for critical points of $\mathcal{L}(p, \lambda)$:

$$\frac{\delta \mathcal{L}(p, \lambda)}{\delta p_i} = -1 - \log(p_i) + \lambda = 0$$

$$\log(p_i) = \lambda - 1 \Rightarrow p_i = e^{\lambda - 1}$$

Here see the value of the same for all i. Next we solve for λ.

$$\frac{\delta \mathcal{L}(p, \lambda)}{\delta \lambda} = \sum_{i=1}^{N} p_i - 1 = 0$$

Substitute $p_i = e^{\lambda-1}$ into $\sum_{i=1}^{N} p_i - 1 = 0$, we get

$$\sum_{i=1}^{N} e^{\lambda-1} - 1 = Ne^{\lambda-1} - 1 = 0$$

$$e^{\lambda-1} = \frac{1}{N} \Rightarrow \lambda = 1 + \log(N)$$

Since value of p_i is the same for all i and $\sum_i p_i = 1$, at the critical point $p_i = \frac{1}{N}$ where N is the number of discrete values that x can take.

A uniform distribution. The maximum entropy configuration is corresponded to an equal distribution of probability acroiss the possible states.

(b) Which probability distribution minimizes entropy?

Ans. Distribution in which one of the states have absolute certainty that it will occur $p(X = j) = 1$ and all other states will not occur $p(X \neq j) = 0$. Thus the objective function

$$-p_j \log(p_j) - \sum_{i \neq j} p_i \log(p_i) = -1 \log(1) - \sum_{i \neq j} 0 \log(0)$$
$$= 0$$

Note* that $\lim_{x \to 0} p(x) \log p(x) = 0$ so we take $p(x) \log p(x) = 0$ when we encounter a value of x which $p(x) = 0$

2 Problem 2

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

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

where

$$M = \left(A - BD^{-1}C\right)^{-1} \tag{2}$$

First we multiple both sides of equation (1) by $\begin{bmatrix} A & B \\ C & D \end{bmatrix}$

$$\mathbf{I}_{n+p} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix} \tag{3}$$

$$\mathbf{I}_{n+p} = \begin{bmatrix} \mathbf{P}_{11} & \mathbf{P}_{12} \\ \mathbf{P}_{21} & \mathbf{P}_{22} \end{bmatrix} \tag{4}$$

We can then express the 4 partitions of $\mathbf{I}_{n+p}$ in terms of A, B, C, D.

Since $\mathbf{I}_{n+p}$ is an identity matrix: $\mathbf{P}_{11}$ and $\mathbf{P}_{22}$ should be identity matrices as well while $\mathbf{P}_{12}$ and $\mathbf{P}_{21}$ are zero matrices.

First, we consider the **top left** submatrix

$$\begin{aligned} \mathbf{P}_{11} &= AM - BD^{-1}CM \\ &= (A - BD^{-1})CM \\ &= M^{-1}M = \mathbf{I} \end{aligned} \tag{5}$$

which is indeed an identity matrix. Next week look at the **top right** submatrix

$$\begin{aligned} \mathbf{P}_{12} &= -AMBD^{-1} + BD^{-1} + BD^{-1}CMBD^{-1} \\ &= (-AM + \mathbf{I} + CMBD^{-1})BD^{-1} \\ &= (\mathbf{I} - \mathbf{P_{11}})BD^{-1} = 0 \end{aligned} \tag{6}$$

Since we found that $\mathbf{P}_{11}$ is an identity matrix from equation (5), we get $\mathbf{P_{12}}$ to be a zero matrix.

Next we look at the **bottom left** submatrix

$$\begin{aligned} \mathbf{P}_{21} &= CM - DD^{-1}CM \\ &= (\mathbf{I} - DD^{-1})CM = 0 \end{aligned} \tag{7}$$

Which is indeed a zero matrix. Lastly we look at the **bottom right** submatrix

$$\begin{aligned} \mathbf{P}_{22} &= -CMBD^{-1} + DD^{-1} + DD^{-1}CMBD^{-1} \\ &= -CMBD^{-1} + \mathbf{I} + DD^{-1}CMBD^{-1} \\ &= -CMBD^{-1} + \mathbf{I} + CMBD^{-1} \\ &= \mathbf{I} \end{aligned} \tag{8}$$

Thus: Right hand side of equation (3) is also a identity matrix. Thus by showing that

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

is an identity matrix, we know that

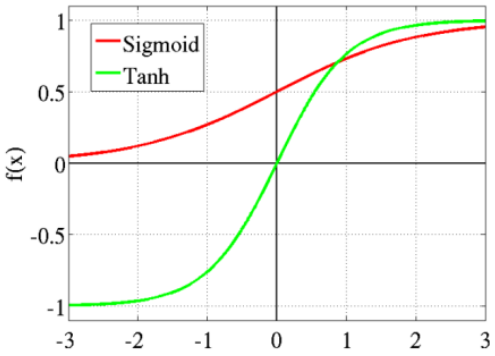
$$\begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix}$$

is exactly the inverse of

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

3 Problem 3

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



We observe that the *tanh* function is a scaled and shifted sigmoid function such that:

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

Another way to show which:

$$\text{Let } S = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

$$S + Se^x = e^x \Rightarrow S = e^x - Se^x$$

$$e^x = \frac{S}{1-S}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1} = \frac{(e^x)^2 - 1}{(e^x)^2 + 1}$$

4 Problem 4

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: 'Angle␣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 the layers
        self.layer1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1, padding=2),
```

```
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer2 = nn.Sequential(
            nn.Conv2d(
                in_channels=16,
                out_channels=32,
                kernel_size=5,
                stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

        self.fc = nn.Linear(7*7*32, 10)


    def forward(self, x):
        # invoke the layers
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)

        return out
```

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

```
def main():
    NUM_EPOCHS = 10
    LRATE = 0.015
    device = torch.device("cuda" if torch.cuda.is_available()
       else "cpu")
    print(f"Using␣{device}")
    # data
    data_path = './data'
    if not os.path.isdir(data_path):
        os.mkdir(data_path)

    train_dataset = datasets.FashionMNIST(data_path, train=True
       , download=True, transform=transforms.ToTensor())
```

```
    test_dataset = datasets.FashionMNIST(data_path, train=False
        , download=True, transform=transforms.ToTensor())

    batch_size = 100
    train_loader = torch.utils.data.DataLoader(dataset=
        train_dataset,batch_size=batch_size,shuffle=True);
    test_loader = torch.utils.data.DataLoader(dataset=
        test_dataset,batch_size=batch_size,shuffle=False);
    model = CNN().to(device)
    optimizer = optim.SGD(model.parameters(), lr=LRATE)

    # initialize weights using xavier initialize
    model.apply(xavier_init)
    for epoch in range(1,NUM_EPOCHS + 1):
        test(model,device,test_loader)
        train(model,device,train_loader,optimizer,epoch)
```

(c) Complete the training function by defining the model output and the loss function. Use the optimizers 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)

        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(model.parameters(), 0.001)

        # Foward pass
        outputs = model(data)
        loss = criterion(outputs,target)

        #Optimizer's step() function is used to update the
            weights after
        # backpropogating the gradients
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
        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, plot=False):
    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)

            outputs = model(data)
            _, pred = torch.max(outputs.data,1)
            correct += (pred == target).sum()

            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

    set_accuracy = (100*correct/len(test_loader.dataset)).item
        ()
    print(f'Test␣set␣accuracy:␣{set_accuracy}%')

    if plot:
        fig = plt.figure(figsize=(12,6));
        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.axis('off')
        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`.

```
Test set accuracy: 88%
Epoch: 7 ,loss: 0.16433095932006836
Epoch: 7 ,loss: 0.17118999361991882
Epoch: 7 ,loss: 0.14606855809688568
Epoch: 7 ,loss: 0.2389155775308609
Epoch: 7 ,loss: 0.10080623626708984
Epoch: 7 ,loss: 0.248300701379776
Test set accuracy: 88%
Epoch: 8 ,loss: 0.2833236753940582
Epoch: 8 ,loss: 0.34994930028915405
Epoch: 8 ,loss: 0.20306701958179474
Epoch: 8 ,loss: 0.13939650356769562
Epoch: 8 ,loss: 0.13270948827266693
Epoch: 8 ,loss: 0.21802589297294617
Test set accuracy: 89%
Epoch: 9 ,loss: 0.12236066907644272
Epoch: 9 ,loss: 0.15060991048812866
Epoch: 9 ,loss: 0.3397677540779114
Epoch: 9 ,loss: 0.07385330647230148
Epoch: 9 ,loss: 0.16223442554473877
Epoch: 9 ,loss: 0.16752071678638458
Test set accuracy: 89%
Epoch: 10 ,loss: 0.2295094132423401
Epoch: 10 ,loss: 0.19616113603115082
Epoch: 10 ,loss: 0.1733078509569168
Epoch: 10 ,loss: 0.2891373932361603
Epoch: 10 ,loss: 0.1814197152853012
Epoch: 10 ,loss: 0.32980501651763916
```