Name: Chaoran Li
Student ID: (UTD ID) 2021489307; (NET ID) cxl190012
Course Number: CS 6364.002

Homework 5 Writeup Part

Rewrite the code for last three question. Add more explanations for all five questions.

Q1 (Regression) Implement a neural network to train a regression model for the Boston housing data set https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155
Split the dataset to a training set (70% samples) and a testing set (30% samples). Report the root mean squared errors (RMSE) on the testing sets.
– You have to use PyTorch deep learning library.
– Two hidden layers: the first hidden layer must contain 16 units using ReLU activation function; the second layer must contain 32 units using tanh activation function.

This question is nearly the same as homework 4 except for using PyTorch. Below is my design for CNN:

```
96  class Model(nn.Module):
97      def __init__(self, _fin):
98          super(Model, self).__init__()
99          # Two hidden layers: the first hidden layer must contain 16 units using ReLU activation function;
100         # the second layer must contain 32 units using tanh activation function.
101         _f1, _f2, _fout = 16, 32, 1
102         self.l1 = nn.Linear(_fin, _f1)
103         self.act1 = nn.ReLU()
104         self.l2 = nn.Linear(_f1, _f2)
105         self.act2 = nn.Tanh()
106         self.out = nn.Linear(_f2, _fout)
```

I use MSELoss() as loss function and SGD as optimization algorithm. For SGD, I set alpha (learning rate) as 1e-3 and momentum (eta) as 0.9. After training, I got the RMSE of 9.0525 for training set and 9.6096 for testing set.

```
Training...

Input feature: 14
RMSE of training set:
9.052528
RMSE of testing set:
9.609648

Process finished with exit code 0
```

Compared with the result I got in homework 4, I got the better result which are 6.6301 for training set and 6.8568 for testing set. Even for using SGD with momentum, I got better result which are 7.8477 for training set and 8.3603 for testing set. This might be caused by too simple design in layer and optimization algorithm.

Q2 (Classification): Implement a neural network to train a classification model for the Titantic dataset: https://blog.goodaudience.com/machine-learning-using-logistic-regression-in-python-with-code-ab3c7f5f3bed.
Split the dataset to a training set (80% samples) and a testing set (20% samples). Report the overall classification accuracies on the training and testing sets and report the precision, recall, and F-measure scores for each of the two classes on the testing sets.
– You have to use PyTorch deep learning library.
– Two hidden layers: the first hidden layer must contain 5 units using ReLU activation function; the second layer must contain 3 units using tanh activation function.

This question is nearly the same as homework 4 except for using PyTorch. Below is my design for CNN:

```python
class Model(nn.Module):
    def __init__(self, _fin):
        super(Model, self).__init__()
        # Two hidden layers: the first hidden layer must contain 5 units using ReLU activation function;
        # the second layer must contain 3 units using tanh activation function.
        _f1, _f2, _fout = 5, 3, 1
        self.l1 = nn.Linear(_fin, _f1)
        self.act1 = nn.ReLU()
        self.l2 = nn.Linear(_f1, _f2)
        self.act2 = nn.Tanh()
        self.out = nn.Linear(_f2, _fout)
```

I use MSELoss() as loss function and SGD as optimization algorithm. For SGD, I set alpha (learning rate) as 1e-3 and momentum (eta) as 0.9. After training, I got the accuracy of 0.650 for training set and 0.696 for testing set.

```
Training...

Input feature: 7
The precision, recall rates, and the F1-score of training set:
(0.6501160873355823, 0.6591865357643759, 0.6408441009566062, None)
The precision, recall rates, and the F1-score of testing set:
(0.6960440958236167, 0.702247191011236, 0.6985793406339152, None)

Process finished with exit code 0
```

Compared with the result I got in homework 4, I got the better result which are 0.808 for training set and 0.795 for testing set. Even for using SGD with momentum, I got better result which are 0.726 for training set and 0.699 for testing set. This might be caused by too simple design in layer and optimization algorithm.

Q3 Given an input image (see the attachment), design two filters/kernels with size of 3 by 3 to detect all horizontal and vertical edges. To make it easier, you can first convert the image into gray scale using OpenCV. Your architecture only contains 1 convolutional layer, 1 pooling layer and 1 fully connected layer. Show the visualizations of the output of the convolutional layer, the pooled layer and the ReLU activated convolutional layer (not the fully connected layer) for both kernels.
What need to submit:
– Python programming code
– Output six figures (two kernels by three layers) in total.

I wasted most time in this question. Once, I wanted to train the model to get better kernels for edge detection. At last, I got my way back to this. For now, I use convolutional layer and max pooling layer which is just like the normal step for edge detection.
Kernels: (Up for horizontal and bottom for vertical edge detection)

```
Kernels:
[[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
[[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
Reshaped data:
(1, 300, 276) (82800,)

Process finished with exit code 0
```

The original image:

Convert into gray scale:

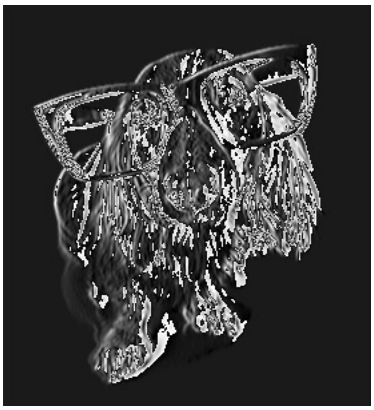
Implement with horizontal and vertical kernels: (left for horizontal and right for vertical edge detection)

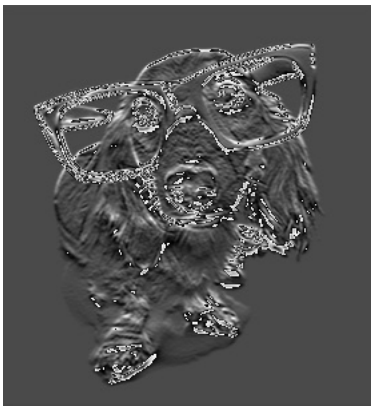
Combined above two into the final edge detection result:

Then, I use this as y value as input for our model. Below is the intermediate view during training.

```
99   ~/Desktop/CS6364ArtificalIntelligence/homework/HW5/question3.py
100      def __init__(self):
101          super(Model, self).__init__()
102          self.conv1 = nn.Conv2d(1, 2, 3, padding=1)
103          self.conv1.weight.data = torch.Tensor([[kernels[0].tolist()], [kernels[1].tolist()]])
104          self.act1 = nn.ReLU()
105          self.pool1 = nn.MaxPool2d(3, 1, padding=1)
```

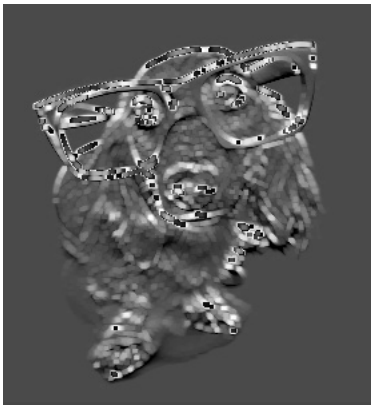
Convolutional layer: (left for horizontal and right for vertical edge detection)





ReLU activated layer: (left for horizontal and right for vertical edge detection)

Pooling layer: (left for horizontal and right for vertical edge detection)



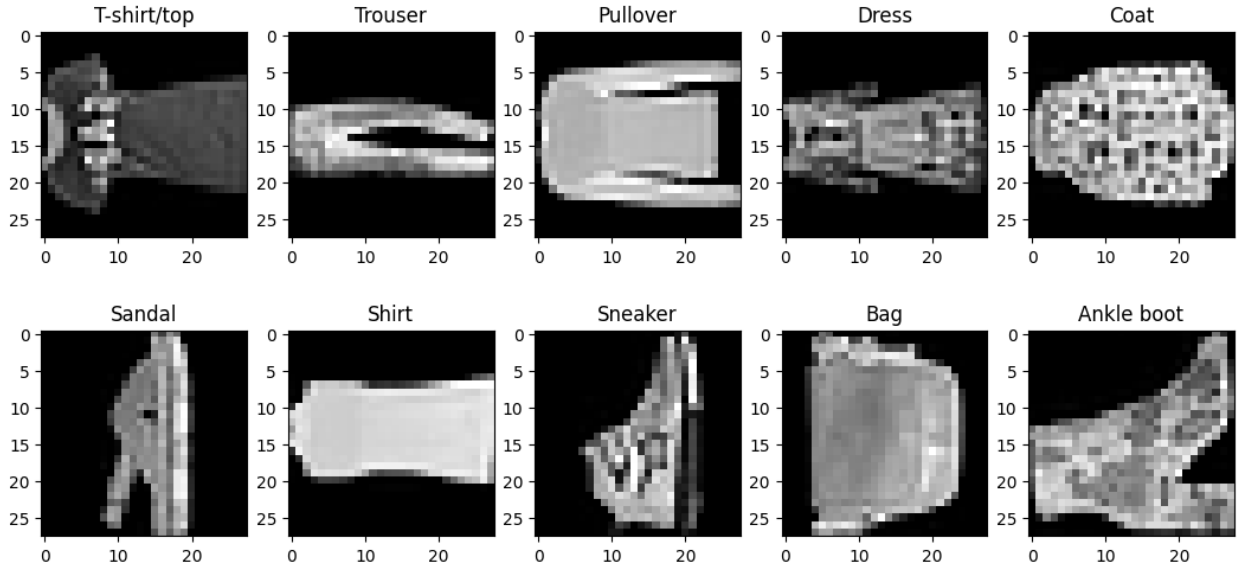
The result is not as good as expected. But I think the result is not bad as least. There is only one thing that still confused me is that the color for horizontal and vertical are different. In another word, the value for horizontal is smaller than the vertical. I would guess this is because I did not crop the image into a square shape.

Q4 Build an image classification model using Convolutional Neural Networks (CNN) in PyTorch. The data set (i.e. Fashion-MNIST) can be found here: https://github.com/zalandoresearch/fashion-mnist/tree/master/data/fashion. Information about this dataset can be found here:
https://github.com/zalandoresearch/fashion-mnist.
It consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
By building the classification model, I suggest you divide it into several subtasks:
1. Loading the data set.
– Please explore a few samples and visualize these images.
2. Creating a validation set and preprocessing the images.
– Split the training data set to a training set (90% samples) and a validation set (10% samples) – convert the images and the targets into torch format for both training and validation data sets.
3. Implementing CNNs using PyTorch.
– Define the architecture with just 2 convolutional layers to extract features from the images and then use a fully connected dense layer to classify those features into their respective categories (i.e. two Conv2d layers and a linear layer).
– Train the model for 25 epochs and show the validation losses by printing in console. You are expected to see that the validation loss is decreasing as epoch increases.
– Visualize the training and validation losses by plotting them.
– Show the accuracy of the model on the training and validation set. 4. Generating predictions for the test set.
– Load the test images.
– Do the pre-processing steps on these images similar to what you did for the training images. – Generate predictions for the test set.
What to submit:
– Python programming code
– Five image visualizations when loading the data set.
– Plotting visualization of the training and validation losses.
– Report the accuracy of your model on training and validation set.
– Your predictions for the test images.

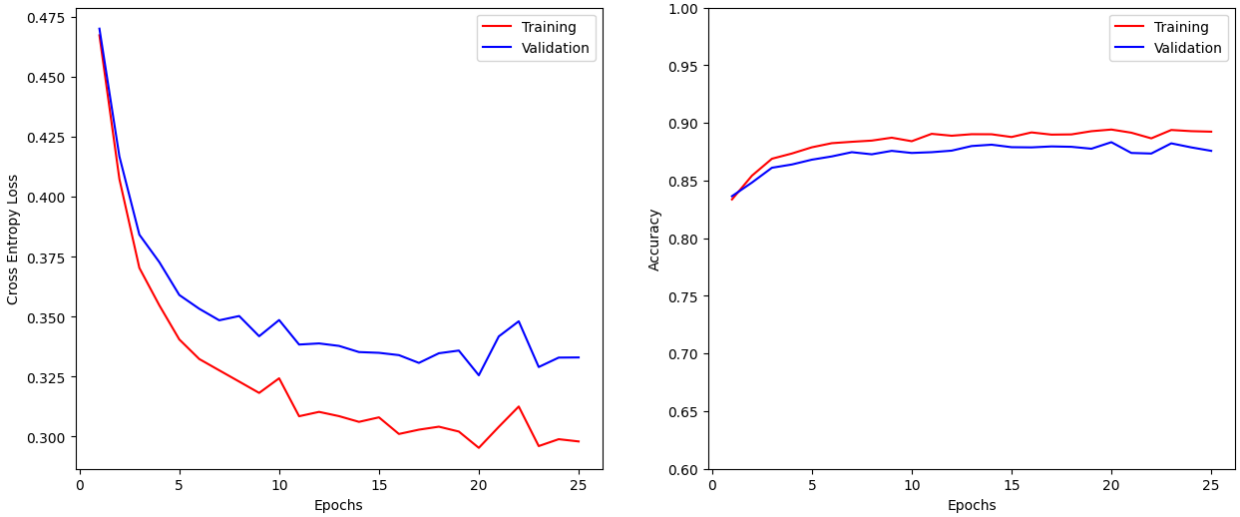
Below is my layer design:

```python
# train models
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        # input channel, output channel, kernel
        self.conv1 = nn.Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        self.bn1 = nn.BatchNorm2d(4, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True)
        self.act1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        self.conv2 = nn.Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
        self.bn2 = nn.BatchNorm2d(4, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True)
        self.act2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        self.fc = nn.Linear(in_features=196, out_features=10, bias=True)
```

For image visualization, since the data has ten categories, I show ten images instead. Each image is the first case for each category in training set.



Plotting loss and accuracy for training and validation sets:



After training, I use testing set to evaluate the final model:

```
Epoch 25:
Training Loss:0.29799482226371765
Validation Loss:0.3330153226852417
Training Accuracy:0.892462962962963
Validation Accuracy:0.8758333333333334

Testing:
Testing Loss:0.3536078929901123
Testing Accuracy:0.8751

Process finished with exit code 0
```

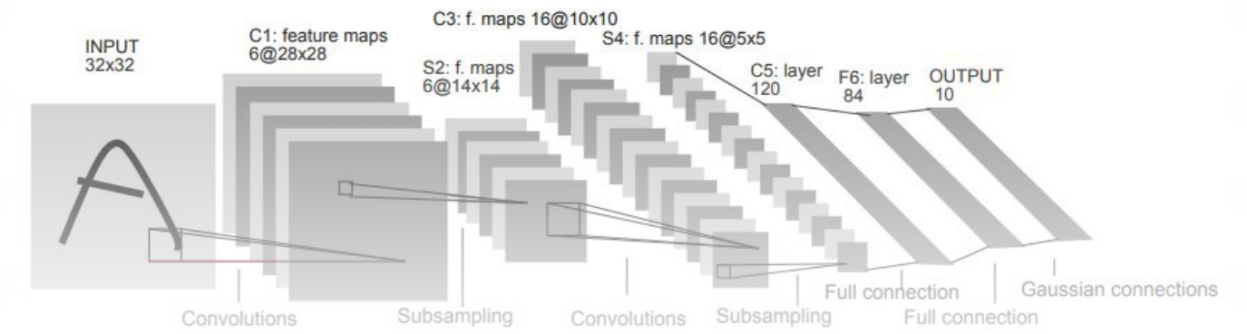
I use cross entropy loss as loss function and Adam as optimization function. The result is pretty good in some way. Since the layer design is fixed. Maybe we could use better optimization algorithm and loss function.

Q5 (Bonus Question): You will get an additional half point (0.5) if you can answer this bonus question correctly. That means, if you answer Q1-Q4 correctly, you get a full point (1.0) for this HW assignment. If you can answer Q1-Q5 correctly, you will get 1.5 points.
Following the step by step instructions below to build a LeNet-5 CNN architecture in Pytorch using Fashion- MNIST data set provided in Q2.
https://medium.datadriveninvestor.com/lenet-5-a-classic-cnn-architecture-c87d0b03560d
What to submit:

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): ReLU(inplace)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=196, out_features=10, bias=True)
  )
)
```

– Python programming code
– Visualization of plotting the training accuracy and loss after each epoch
– Your predictions for the test images.

I did not pay for the original url. I found the detailed layer design in other place.

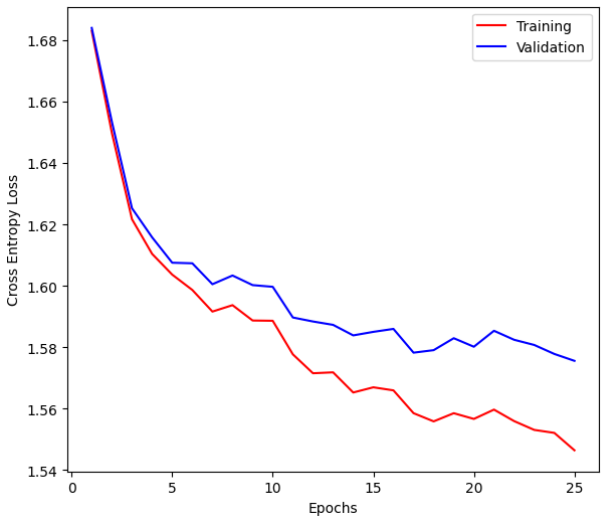


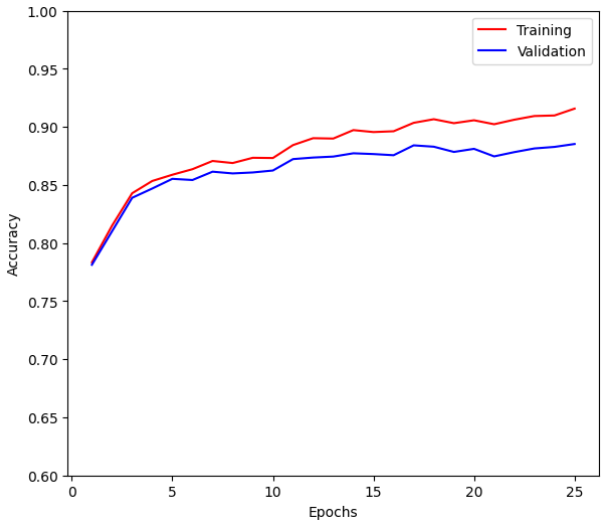
Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

However, the input for our case is 28 x 28. Hence, I add a padding for 2 in the first convolutional layer. By this, I can use the following layer design. Below is my layer design:

```
79  class Model(nn.Module):
80      def __init__(self):
81          super(Model, self).__init__()
82          # input channel, output channel, kernel
83          self.conv1 = nn.Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
84          self.act1 = nn.Tanh()
85          self.pool1 = nn.AvgPool2d(kernel_size=2, stride=2)
86          self.conv2 = nn.Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
87          self.act2 = nn.Tanh()
88          self.pool2 = nn.AvgPool2d(kernel_size=2, stride=2)
89          self.conv3 = nn.Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
90          self.act3 = nn.Tanh()
91          self.fc1 = nn.Linear(in_features=120, out_features=84, bias=True)
92          self.act4 = nn.Tanh()
93          self.fc2 = nn.Linear(in_features=84, out_features=10, bias=True)
94          self.act5 = nn.Softmax(dim=1)
95
```

Plotting loss and accuracy for training and validation sets:





After training, I use testing set to evaluate the final model:

```
Epoch 25:
Training Loss:1.5464342832565308
Validation Loss:1.5755860805511475
Training Accuracy:0.9157962962962963
Validation Accuracy:0.8853333333333333

Testing:
Testing Loss:1.5810539722442627
Testing Accuracy:0.8795

Process finished with exit code 0
```

I use cross entropy loss as loss function and Adam as optimization function. The result is worse than what we got in question 4. I think this might be caused by my padding and the margin is usually less important in computer vision.