**Homework 4: Intro to Deep Learning (Spring 2020)**

Name: Pratik Mistry

NetID: pdm79

RUID: 194008675

**Solution: Part 1**

* I used NN Sequential Module of Pytorch for training and testing the model.
* I used Cross Entropy Loss Layer in model training as Pytorch Cross Entropy Layer calculates the Softmax as well as the loss simultaneously. Also, since this problem was **MULTICLASS CLASSIFICATION** problem, I had used this layer while training for calculating loss.
* For Part 1, I have used CNN having layers as per the standard Lenet Architecture defined in slide 3 and 4 of lecture 10.
* As per architecture:
  + Starting layer is Convolution2d layer with input of 3 channels and output as 6 channel with Kernel filter size as 5,5 followed by Relu function **– c1 and relu1**
  + Then I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 **– s2**
  + Further, then comes Convolution2d layer with input as 6 channels and outputs as 16 channels with Kernel filter size as 5,5 followed by Relu function – **c3 and relu3**
  + Again, I perform sub-samping using MaxPool2d with kernel size as 2,2 and stride of 2 **– s4**
  + In the last convolution network, I have Convolution2d layer with input as 16 channels and outputs as 120 channels with Kernel filter size as 5,5 followed by Relu function **– c5, relu5**
  + Now very important task – I have to flatten the images i.e. output from convolution network so that they can be fed into fully connected layers. Thus, I have flattened the output of convolution network using torch.view() function in forward pass.
  + Then, once data is flattened, I fed it to fully connected Linear layer with input as 120 and output as 84 followed by Relu function. – **f6 and relu6**
  + Further, I again pass the data to Linear Layer with input as 84 and final output as 10– **f7**
  + This output is passed to final LogSoftmax layer – **sig7**
* Please find the below attached image of the output. The image shows the Training Loss for each epoch, training time taken, total images tested and model accuracy.
* As seen in the images, it took around **12585.094 seconds i.e. 209.75 minutes** for training the model with 50000 training and 10000 testing CIFAR-10 images in 50 epochs. The test accuracy of the model as shown in screenshots below is **63%** (with maximum accuracy of **64%** and minimum accuracy of **42%**).
* The working code is uploaded in the assignment submitted on Sakai with name of the file as: **lenet\_cifar\_fill\_simpleLenet.py**

**Output Screenshot:**

Output_Simple.tiff

**Highest and Lowest Accuracy Instances Screenshots**:

Highest_Simple.tiff Lowest_Simple.tiff

**Solution: Part 2**

* The model used in this part is same but only additional layer added here is Drop Out Layer with dropping rate of 0.5 (commonly rate used widely).
* As per previous architecture, I have added dropout layer after c3 and relu3 **– drop3**
* Please find the below attached image of the output. The image shows the Training Loss for each epoch, training time taken, total images tested and model accuracy.
* As seen in the images, it took around **12313.83 seconds i.e. 205.23 minutes** for training the model with 50000 training and 10000 testing CIFAR-10 images in 50 epochs. The test accuracy of the model as shown in screenshots below is **50%** (with maximum accuracy of **51%** and minimum accuracy of **35%**).
* The working code is uploaded in the assignment submitted on Sakai with name of the file as: **lenet\_cifar\_fill\_withDropout.py**

**Output Screenshot:**

Output_withDropout.tiff

**Highest and Lowest Accuracy Instances Screenshots**:

Highest_withDropout.tiff Lowest_withDropout.tiff

**Solution: Part 3**

* The model used in this part is same as Part 2, but only additional layer added here is Batch Normalization Layer with dropping rate of 0.5 (commonly rate used widely).
* As per previous architecture, I have added batch normalization layer between c1 and relu1 **– bn1**
* Please find the below attached image of the output. The image shows the Training Loss for each epoch, training time taken, total images tested and model accuracy.
* As seen in the images, it took around **12906.69 seconds i.e. 215.11 minutes** for training the model with 50000 training and 10000 testing CIFAR-10 images in 50 epochs. The test accuracy of the model as shown in screenshots below is **53%** (with maximum accuracy of **55%** and minimum accuracy of **37%**).
* The working code is uploaded in the assignment submitted on Sakai with name of the file as: **lenet\_cifar\_fill\_withDropout\_BatchNorm.py**

**Output Screenshot:**

Output_withDO_BN.tiff

**Highest and Lowest Accuracy Instances Screenshots**:

Highest_withDO_BN.tiffLowest_withDO_BN.tiff

**Conclusions and Analysis:**

* Simple Lenet model has the best accuracy with maximum of 64% compared to other two variations in model.
* Adding dropout layer in Simple Lenet model after the second convolution and relu layer decreases the overall testing and training time but it results in the worst accuracy compared to Simple and third variation in Lenet Model. The maximum accuracy achieved was 51%. The total time taken was 12313.83 seconds i.e. 205.23 minutes compared to total time taken for Simple Lenet model which was 12585.094 seconds i.e. 209.75 minutes.
* The main reason why total time taken after adding dropout layer decreased by approximately 4 minutes is because dropout layer randomly drops some neurons while training and deeper layers will have to train less data. Also, reason for lower accuracy is because the common rate of 0.5 would have caused model to drop important features from images during training to avoid overfitting. Another important reason I analyze is that dropout layer with rate 0.5 **after** **second** convolution and relu layer would have dropped more important features because after passing training images from convolution-relu-maxpool-convolution-relu layers the images would just have important features left in them some of which gets randomly dropped in dropout layer.
* Now, adding a batch normalization to existing Lenet model having dropout layer, increased some performance accuracy with maximum of 55% which is greater than Lenet with dropout layer but less than simple lenet model.
* The reason again why accuracy in third variation is bad than Simple lenet model is because of dropout layer as mentioned in above explanation. But, the reason of better accuracy than Lenet model with dropout layer is it normalizes the data before passing the data through further layers.
* Now, the time taken was 12906.69 seconds i.e. 215.11 minutes, which is quite large, compared to earlier two variations because in each epoch for all images, it has to do data preprocessing/normalization for all images. Also, other reason is batch normalization layer is added after first convolution layer and data after first layer will be huge (comprising of important or unimportant features) compared to deeper layers where only important features remains in data.
* Lastly, the reason why my machine takes very huge total time for training and testing in 50 epochs is because of low machine configurations. Below are the relevant details:

***Machine specifications are:***

* + Processor: 2 GHz Intel Core 2 Duo
  + Memory: 8 GB1067 MHz DDR3
  + Graphics: NVIDIA GeForce 9400M 256MB
  + OS: Mac OS Yosemite 10.10.5