Computer Vision - Assignment 2

Submitted by : Bharathi Priyaa Thangamani
bt978@nyu.edu

Question 2: Visualization of Images in CIFAR and MNIST

The first 100 images from MNIST and CIFAR are printed as below :

```
print("Printing the first 100 Images of MNIST")
image.display{image = train.data[{{1,100}}], legend = 'First 100 MNIST', scaleeach =true}
print("Printing the first 100 Images of CIFAR")
local train_cifar = torch.load(base_data_path .. 'cifar-10-torch/data_batch_1.t7', 'ascii')
print({train_cifar})
local first100_cifar =
train_cifar.data:permute(2,1):reshape(10000,3,32,32)[{{1,100},{},{},{}}]
print({first100_cifar})
image.display{image = first100_cifar, legend = 'First 100 CIFAR', scaleeach =true}
```

Question 3 - Training a Single Layer Network on MNIST

Training on MNIST with num_training = 1000 and num_test = entire data set with default parameters

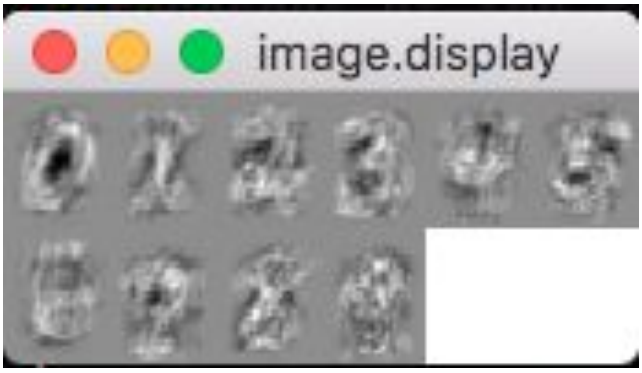
OUTPUT:

train | epoch = 1 | lr = 0.1000 | loss: 24318.9866 | error: 659.0000 - valid | validloss: 11820.6604 | validerror: 369.0000 | s/iter: 0.3278
train | epoch = 2 | lr = 0.1000 | loss: 2953.2705 | error: 197.0000 - valid | validloss: 3244.0897 | validerror: 273.0000 | s/iter: 0.3198
train | epoch = 3 | lr = 0.1000 | loss: 1251.4822 | error: 130.0000 - valid | validloss: 1713.2571 | validerror: 168.0000 | s/iter: 0.3047
train | epoch = 4 | lr = 0.1000 | loss: 1075.8950 | error: 117.0000 - valid | validloss: 2980.4362 | validerror: 234.0000 | s/iter: 0.2977
train | epoch = 5 | lr = 0.1000 | loss: 725.1646 | error: 97.0000 - valid | validloss: 1760.7965 | validerror: 189.0000 | s/iter: 0.3566
train | epoch = 6 | lr = 0.1000 | loss: 1124.6872 | error: 130.0000 - valid | validloss: 1726.1159 | validerror: 184.0000 | s/iter: 0.3058
train | epoch = 7 | lr = 0.1000 | loss: 695.9517 | error: 100.0000 - valid | validloss: 2663.6160 | validerror: 227.0000 | s/iter: 0.3017
train | epoch = 8 | lr = 0.1000 | loss: 443.7026 | error: 69.0000 - valid | validloss: 2404.2786 | validerror: 241.0000 | s/iter: 0.3211
train | epoch = 9 | lr = 0.1000 | loss: 528.0508 | error: 103.0000 - valid | validloss: 2357.3168 | validerror: 240.0000 | s/iter: 0.3209
train | epoch = 10 | lr = 0.1000 | loss: 288.7460 | error: 59.0000 - valid | validloss: 1893.3424 | validerror: 216.0000 | s/iter: 0.3196

| test | error: 1799.0000

Image of Network weights :

```
--Code to plot out the network weights
image.display(network.weight:reshape(10,28,28))
```



Observation of a single output:
Network output
-45120.3976
-6400.1844
-17597.0256
-9334.3788
27051.5978
14564.6961
20664.5290
-11052.8246
14109.2715
12990.2683
Target 5

When there is a single linear layer, after 10 epochs the network tries to identify the target correctly. If we observe the network output (where the target is 5)for a single sample, we see that weights assigned for numbers other than the target are deviated from the target by a big factor. This is good !

3.b : Reducing number of Training samples to 50

Output:
train | epoch = 1 | lr = 0.1000 | loss: 75.9566 | error: 49.0000 - valid | validloss: 25531.8894 | validerror: 42.0000 | s/iter: 0.0301
train | epoch = 2 | lr = 0.1000 | loss: 24232.5543 | error: 37.0000 - valid | validloss: 30423.1796 | validerror: 36.0000 | s/iter: 0.0267
train | epoch = 3 | lr = 0.1000 | loss: 32898.4370 | error: 35.0000 - valid | validloss: 45004.0684 | validerror: 37.0000 | s/iter: 0.0261
train | epoch = 4 | lr = 0.1000 | loss: 31499.4229 | error: 28.0000 - valid | validloss: 40767.8756 | validerror: 35.0000 | s/iter: 0.0300
train | epoch = 5 | lr = 0.1000 | loss: 29687.8886 | error: 21.0000 - valid | validloss: 25453.0172 | validerror: 32.0000 | s/iter: 0.0271
train | epoch = 6 | lr = 0.1000 | loss: 17650.9830 | error: 19.0000 - valid | validloss: 17170.1305 | validerror: 29.0000 | s/iter: 0.0269
train | epoch = 7 | lr = 0.1000 | loss: 2853.3474 | error: 9.0000 - valid | validloss: 9893.9695 | validerror: 32.0000 | s/iter: 0.0275
train | epoch = 8 | lr = 0.1000 | loss: 701.9030 | error: 4.0000 - valid | validloss: 11064.1703 | validerror: 31.0000 | s/iter: 0.0257
train | epoch = 9 | lr = 0.1000 | loss: 57.6007 | error: 1.0000 - valid | validloss: 11009.6059 | validerror: 33.0000 | s/iter: 0.0267
train | epoch = 10 | lr = 0.1000 | loss: 0.0000 | error: 0.0000 - valid | validloss: 11009.6059 | validerror: 33.0000 | s/iter: 0.0294
| test | error: 4920.0000

Observation :
We observe that with the decrease in number of training examples, the cross entropy loss starts decreasing more rapidly than when the training examples were 50. Note that we've set out batchsize to be 100. So, in the case where we have just 50 training examples, our network learns the weights very quickly and after epoch 8, the cross entropy loss becomes zero while our valid loss is still decreasing. Moreover, the test error is much higher than expected. This implies overfitting by our neural network where the network learns/memorizes weights for training dataset.

Qn.4 - Training a Multi-Layer Network on MNIST

4.a) Adding an extra layer of non-linearity

```
local network = nn.Linear(nin, nout)
local criterion = nn.CrossEntropyCriterion()
local linear1 = nn.Linear(nin, 1000)
local tanh1 = nn.Tanh()
local linear2 = nn.Linear(1000, nout)
local network = nn.Sequential()
network:add(linear1)
network:add(tanh1)
network:add(linear2)
```

OUTPUT:

train | epoch = 1 | lr = 0.1000 | loss: 6.4171 | error: 756.0000 - valid | validloss: 10.0675 | validerror: 738.0000 | s/iter: 0.6068
train | epoch = 2 | lr = 0.1000 | loss: 13.8094 | error: 765.0000 - valid | validloss: 10.9738 | validerror: 719.0000 | s/iter: 0.5608
train | epoch = 3 | lr = 0.1000 | loss: 14.1912 | error: 746.0000 - valid | validloss: 13.3064 | validerror: 634.0000 | s/iter: 0.6075
train | epoch = 4 | lr = 0.1000 | loss: 12.4027 | error: 718.0000 - valid | validloss: 11.9849 | validerror: 788.0000 | s/iter: 0.5791
train | epoch = 5 | lr = 0.1000 | loss: 11.9304 | error: 691.0000 - valid | validloss: 13.9729 | validerror: 838.0000 | s/iter: 0.5591
train | epoch = 6 | lr = 0.1000 | loss: 14.2846 | error: 654.0000 - valid | validloss: 8.9597 | validerror: 617.0000 | s/iter: 0.5476
train | epoch = 7 | lr = 0.1000 | loss: 12.0146 | error: 654.0000 - valid | validloss: 11.8275 | validerror: 864.0000 | s/iter: 0.5618
train | epoch = 8 | lr = 0.1000 | loss: 13.2489 | error: 659.0000 - valid | validloss: 9.3180 | validerror: 657.0000 | s/iter: 0.5863

train | epoch = 9 | lr = 0.1000 | loss: 10.0176 | error: 629.0000 - valid | validloss: 9.0417 | validerror: 623.0000 | s/iter: 0.5670
train | epoch = 10 | lr = 0.1000 | loss: 6.7360 | error: 585.0000 - valid | validloss: 8.3640 | validerror: 649.0000 | s/iter: 0.5511
| test | error: 6351.0000

Observation for a single test case :

Network output
 1.6862e+01
-3.7324e+00
-3.6708e+01
 4.1042e+00
-8.7163e-04
 4.2690e+00
 4.2498e+00
 1.2150e+01
-1.1048e+01
 8.1282e+00
[torch.DoubleTensor of size 10]
Target 1
[torch.LongTensor of size 1]

The decrease in performance could be attributed by the below :

1) Adding of non -linearity; By adding a Tanh layer, I observed that the weights from Layer 1 are getting mapped to +1/-1 by the Tanh layer, which feeds this as input to another linear layer. Thus, information is lost by adding the non-linear layer in the middle. Taking binary inputs, Linear layer 2, tries to classify each input into a set of 10 categories. Thereby, errors from the middle TanH layer are propagated to the other layers and further back propagated. The system doesn't incrementally learn over time and hence the fluctuation in the error gradient.

4.b) Setting learning rate to 10

output:
epoch = 1 | lr = 10.0000 | loss: 2858.9775 | error: 898.0000 - valid | validloss: 6527.7105 | validerror: 907.0000 | s/iter: 0.6606 epoch = 2 | lr = 10.0000 | loss: 3852.2270 | error: 894.0000 - valid | validloss: 5667.3754 | validerror: 893.0000 | s/iter: 0.6400 epoch = 3 | lr = 10.0000 | loss: 4120.9749 | error: 884.0000 - valid | validloss: 5286.7695 | validerror: 893.0000 | s/iter: 0.6370
epoch = 4 | lr = 10.0000 | loss: 4195.3231 | error: 882.0000 - valid | validloss: 4016.7115 | validerror: 895.0000 | s/iter: 0.6040 epoch = 5 | lr = 10.0000 | loss: 4427.4451 | error: 907.0000 - valid | validloss: 3254.6909 | validerror: 865.0000 | s/iter: 0.5908 epoch = 6 | lr = 10.0000 | loss: 3377.1946 | error: 880.0000 - valid | validloss: 3739.0180 | validerror: 900.0000 | s/iter: 0.5810

epoch = 7 | lr = 10.0000 | loss: 3722.4775 | error: 857.0000 - valid | validloss: 3473.2450 | validerror: 900.0000 | s/iter: 0.5825 epoch = 8 | lr = 10.0000 | loss: 3868.5825 | error: 884.0000 - valid | validloss: 3759.8474 | validerror: 830.0000 | s/iter: 0.5962 epoch = 9 | lr = 10.0000 | loss: 3560.1132 | error: 844.0000 - valid | validloss: 3169.8184 | validerror: 900.0000 | s/iter: 0.5429 epoch = 10 | lr = 10.0000 | loss: 3481.1997 | error: 865.0000 - valid | validloss: 3316.7666 | validerror: 888.0000 | s/iter: 0.5571 | test | error: 8949.0000

Observation : By setting the learning rate to 10, we are making the system take bigger steps in the gradient descent in (hopefully) the direction which will decrease its errors. A big learning rate(step size) could cause the subsequent weight updates to cross over the local minima and thereby contributing to error fluctuations. Also since, the network is not already incrementally learning , any increase in learning rate will further confuse the network to be taking big steps in the wrong direction/right direction and hence the error increases and also fluctuates.

Qn 5 : Training a Convolutional Network on CIFAR

5.a) code of convolutional neural network

```
local net1 = nn.Sequential()
net1:add(nn.SpatialConvolution(3, 16, 5, 5))
net1:add(nn.Tanh())
net1:add(nn.SpatialMaxPooling(2,2,2, 2))
net1:add(nn.SpatialConvolution(16, 128, 5, 5))
net1:add(nn.Tanh())
net1:add(nn.SpatialMaxPooling(2,2,2, 2))
net1:add(nn.View(128*5*5))
net1:add(nn.Linear(128*5*5, 64))
net1:add(nn.Tanh())
net1:add(nn.Linear(64,10))
```

OUTPUT

train | epoch = 1 | lr = 0.1000 | loss: 2.2508 | error: 10373.0000 - valid | validloss: 2.0972 | validerror: 2311.0000 | s/iter: 49.6493
train | epoch = 2 | lr = 0.1000 | loss: 2.0601 | error: 9181.0000 - valid | validloss: 1.9359 | validerror: 2220.0000 | s/iter: 44.6699
train | epoch = 3 | lr = 0.1000 | loss: 1.9347 | error: 8519.0000 - valid | validloss: 1.8207 | validerror: 2020.0000 | s/iter: 52.2987
train | epoch = 4 | lr = 0.1000 | loss: 1.8543 | error: 8096.0000 - valid | validloss: 1.7897 | validerror: 1974.0000 | s/iter: 50.4146
train | epoch = 5 | lr = 0.1000 | loss: 1.8387 | error: 8061.0000 - valid | validloss: 1.8852 | validerror: 2057.0000 | s/iter: 52.3665
train | epoch = 6 | lr = 0.1000 | loss: 1.7806 | error: 7829.0000 - valid | validloss: 1.7264 | validerror: 1904.0000 | s/iter: 50.6307

train | epoch = 7 | lr = 0.1000 | loss: 1.7246 | error: 7483.0000 - valid | validloss: 1.6913 | validerror: 1817.0000 | s/iter: 51.3753
train | epoch = 8 | lr = 0.1000 | loss: 1.6825 | error: 7356.0000 - valid | validloss: 1.5958 | validerror: 1726.0000 | s/iter: 51.7653
train | epoch = 9 | lr = 0.1000 | loss: 1.5772 | error: 6870.0000 - valid | validloss: 1.6230 | validerror: 1756.0000 | s/iter: 53.4943
train | epoch = 10 | lr = 0.1000 | loss: 1.5864 | error: 6911.0000 - valid | validloss: 1.5974 | validerror: 1732.0000 | s/iter: 52.0662
train | epoch = 11 | lr = 0.1000 | loss: 1.5256 | error: 6566.0000 - valid | validloss: 1.5818 | validerror: 1703.0000 | s/iter: 56.2381
train | epoch = 12 | lr = 0.1000 | loss: 1.5057 | error: 6439.0000 - valid | validloss: 1.5219 | validerror: 1654.0000 | s/iter: 52.7660
train | epoch = 13 | lr = 0.1000 | loss: 1.4695 | error: 6347.0000 - valid | validloss: 1.6132 | validerror: 1741.0000 | s/iter: 55.1031
train | epoch = 14 | lr = 0.1000 | loss: 1.4502 | error: 6319.0000 - valid | validloss: 1.5622 | validerror: 1658.0000 | s/iter: 54.2195
train | epoch = 15 | lr = 0.1000 | loss: 1.3830 | error: 5926.0000 - valid | validloss: 1.5554 | validerror: 1691.0000 | s/iter: 53.7581
train | epoch = 16 | lr = 0.1000 | loss: 1.3990 | error: 5980.0000 - valid | validloss: 1.6478 | validerror: 1763.0000 | s/iter: 50.8267
train | epoch = 17 | lr = 0.1000 | loss: 1.3281 | error: 5707.0000 - valid | validloss: 1.4586 | validerror: 1570.0000 | s/iter: 53.3583
train | epoch = 18 | lr = 0.1000 | loss: 1.2690 | error: 5498.0000 - valid | validloss: 1.6053 | validerror: 1754.0000 | s/iter: 54.0060
train | epoch = 19 | lr = 0.1000 | loss: 1.2517 | error: 5315.0000 - valid | validloss: 1.5010 | validerror: 1622.0000 | s/iter: 53.2883
train | epoch = 20 | lr = 0.1000 | loss: 1.2222 | error: 5222.0000 - valid | validloss: 1.5477 | validerror: 1639.0000 | s/iter: 53.0378
| test | error: 1639.0000

Printing out network weights

5.b) Breakdown of parameters

```
--- Qn 5: Print out the parameters for each Layer
for i  = 1, 10 do
  local params = network:get(i):parameters()
  if params ~= nil then
    local weights = params[1]
    local bias = params[2]
    print("Layer ", i, " Weights:", weights:nElement(), "Biases:", bias:nElement())
  end
end
```

Layer 1 Weights: 1200 Biases: 16
Layer 4 Weights: 51200 Biases: 128
Layer 8 Weights: 204800Biases: 64
Layer 10 Weights: 640 Biases: 10

Total number of parameters : 258,178