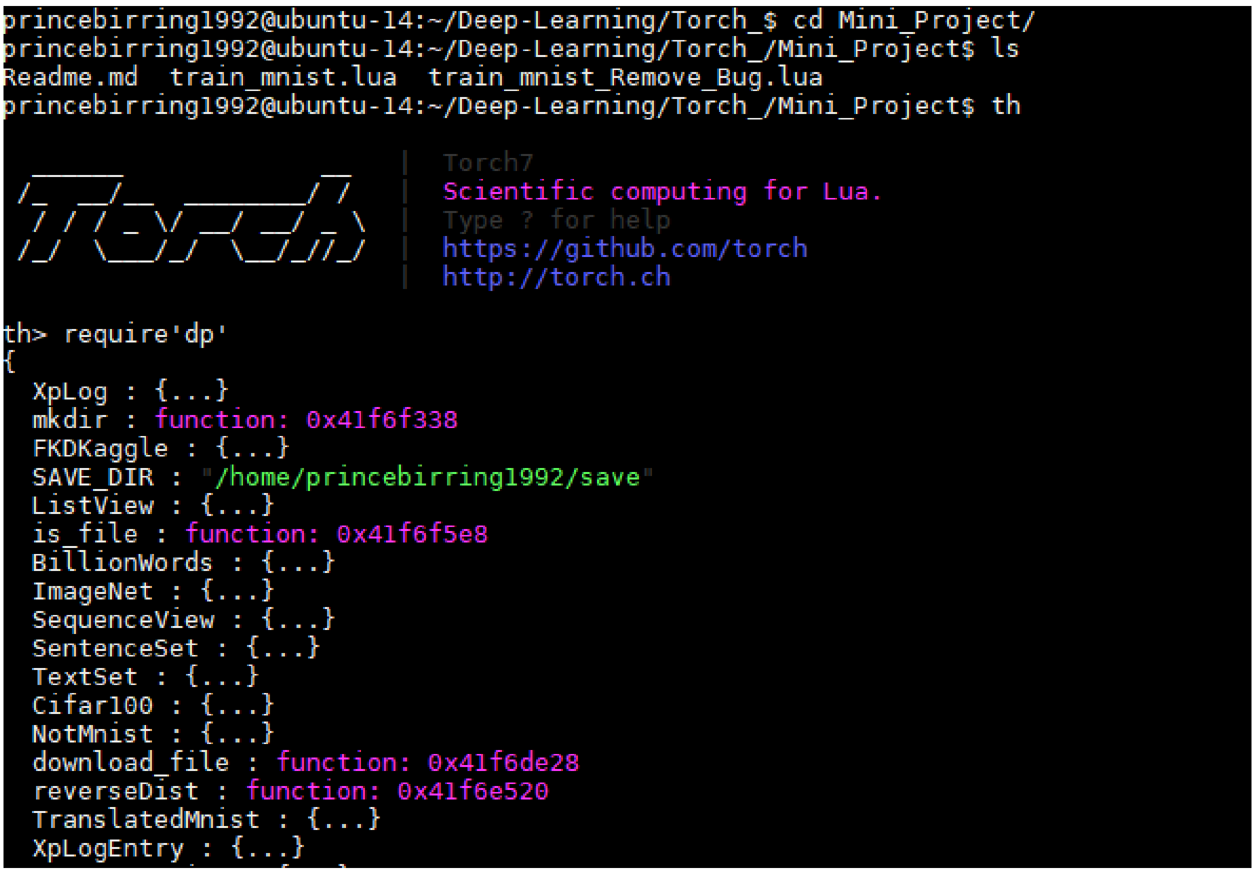
Machine Learning II

Mini Project #1

Prince Pal Birring

Training MLP networks with Torch on MNSIT dataset

1. We are using the dp package for Torch. If you are using GCP or HPC, dp should already be installed.



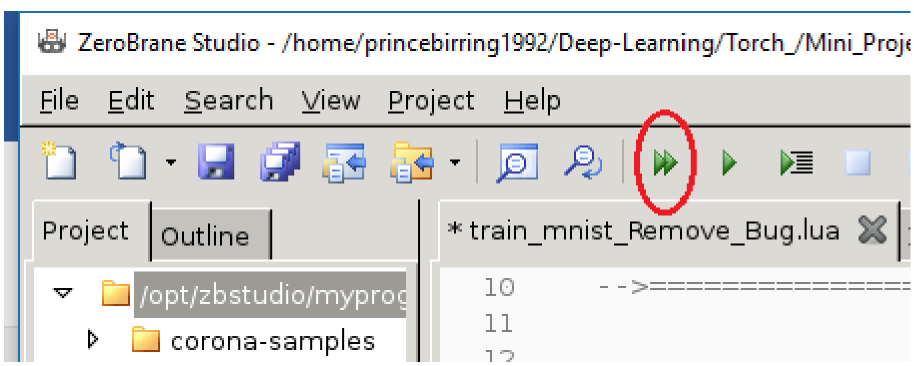
2. Downloads the trains_mnist.lua file from Github(See https://github.com/amir-jafari/Deep-Learning/tree/master/Torch_). It is based on the following website https://github.com/nicholas-leonard/slides/blob/master/torch7.md, which may want to review before proceeding.

```
princebirring1992@ubuntu-14:~$ ls
caffe                                                   cudnn-8.0-linux-x64-v6.0.tgz    install-14-0
cuda                                                    cudnn-8.0-linux-x64-v6.0.tgz.1  log
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb    data                            pycharm-comm
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb.1  demos                           pycharm-comm
princebirring1992@ubuntu-14:~$ git clone https://github.com/amir-jafari/Deep-Learning.git
Cloning into 'Deep-Learning'...
remote: Counting objects: 593, done.
remote: Compressing objects: 100% (263/263), done.
remote: Total 593 (delta 91), reused 347 (delta 87), pack-reused 240
Receiving objects: 100% (593/593), 8.29 MiB | 11.91 MiB/s, done.
Resolving deltas: 100% (177/177), done.
Checking connectivity... done.
princebirring1992@ubuntu-14:~$ ls
caffe                                                   cudnn-8.0-linux-x64-v6.0.tgz    demos
cuda                                                    cudnn-8.0-linux-x64-v6.0.tgz.1  install-14-0
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb    data                            log
cuda-repo-ubuntu1404-8-0-local-ga2_8.0.61-1_amd64.deb.1  Deep-Learning                   pycharm-comm
princebirring1992@ubuntu-14:~$ cd Deep-Learning/
princebirring1992@ubuntu-14:~/Deep-Learning$ ls
Caffe_  Readme.md  Torch_
princebirring1992@ubuntu-14:~/Deep-Learning$ cd Torch_/
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_$ ls
Demo  GPU_Test  Lua  Mini_Project  Readme.md  Torch  Website_Link
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_$ cd Mini_Project/
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_/Mini_Project$ ls
Readme.md  train_mnist.lua  train_mnist_Remove_Bug.lua
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_/Mini_Project$
```

3. Run the program in ZerobraneStudio and investigate and Verify its performance.
   Step1:

```
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_/Mini_Project$ zbstudio
princebirring1992@ubuntu-14:~/Deep-Learning/Torch_/Mini_Project$
```

   Step2:



   Result:

```
2017-11-04 18:14:15 (41.3 MB/s) - ‘mnist4.zip’ saved [15313661/15313661]


  inflating: ./test.th7
New maxima : 0.106600 @ 1.000000
New maxima : 0.110900 @ 3.000000
Test Accuracy : 0.105200
Program completed in 1330.89 seconds (pid: 2468).
```

Change:

```
local input, target = inputs[idx], targets:narrow(1,idx,1)
```

To:

```
local input, target = inputs[idx], targets[idx]
```

Results:

```
Output | Local console | Markers
New maxima : 0.879700 @ 5.000000
New maxima : 0.890500 @ 7.000000
New maxima : 0.905500 @ 8.000000
New maxima : 0.909300 @ 11.000000
New maxima : 0.911100 @ 12.000000
New maxima : 0.915900 @ 18.000000
New maxima : 0.916300 @ 22.000000
New maxima : 0.920900 @ 23.000000
Test Accuracy : 0.907300
Program completed in 1329.41 seconds (pid: 2977).
```

4. Modify the program to run on the GPU.

   Add the below code in the existing code to run the program on GPU.

```
require ‘cunn’

module: cuda()

criterion:cuda()
```

```lua
require 'dp'
require 'cunn'

-- Load the mnist data set
ds = dp.Mnist()

-- Extract training, validation and test sets
trainInputs = ds:get('train', 'inputs', 'bchw')
trainTargets = ds:get('train', 'targets', 'b')
validInputs = ds:get('valid', 'inputs', 'bchw')
validTargets = ds:get('valid', 'targets', 'b')
testInputs = ds:get('test', 'inputs', 'bchw')
testTargets = ds:get('test', 'targets', 'b')

-- Create a two-layer network
module = nn.Sequential()
module:add(nn.Convert('bchw', 'bf')) -- collapse 3D to 1D
module:add(nn.Linear(1*28*28, 20))
module:add(nn.Tanh())
module:add(nn.Linear(20, 10))
module:add(nn.LogSoftMax())
module:cuda()

-- Use the cross-entropy performance index
criterion = nn.ClassNLLCriterion()
criterion:cuda()
```

Results:

```
Output | Local console | Markers

New maxima : 0.881400 @ 6.000000
New maxima : 0.894200 @ 8.000000
New maxima : 0.894900 @ 9.000000
New maxima : 0.901500 @ 10.000000
New maxima : 0.908900 @ 13.000000
New maxima : 0.911200 @ 16.000000
New maxima : 0.914300 @ 17.000000
New maxima : 0.919200 @ 25.000000
Test Accuracy : 0.921600
Program completed in 1325.70 seconds (pid: 2882).
```

5. The program is set up to perform stochastic gradient descent. Modify the program to use minibatches. Experiment with different minibatches sizes.

```lua
require 'dpnn'
function trainEpoch(module, criterion, inputs, targets, batch_size)
   for idx=0,(inputs:size(1) - batch_size), batch_size do
--local idx = math.random(1,inputs:size(1))
      local input= inputs[{{idx + 1, idx +batch_size}}]

      local target = targets[{{idx + 1, idx + batch_size}}]
--targets:narrow(1,idx,1)
      -- forward
      local output = module:forward(input)
      local loss = criterion:forward(output, target)
      -- backward
      local gradOutput = criterion:backward(output, target)
      module:zeroGradParameters()
      local gradInput = module:backward(input, gradOutput)
      -- update
      module:updateGradParameters(0.9) -- momentum (dpnn)
      module:updateParameters(0.1) -- W = W - 0.1*dL/dW
   end
end

 bestAccuracy, bestEpoch = 0, 0
wait = 0
for epoch=1,30 do
   trainEpoch(module, criterion, trainInputs, trainTargets, 200)
   local validAccuracy = classEval(module, validInputs, validTargets)
   if validAccuracy > bestAccuracy then
```

Result:

```
Output | Local console | Markers
New maxima : 0.946700 @ 21.000000
New maxima : 0.947400 @ 22.000000
New maxima : 0.947800 @ 23.000000
New maxima : 0.948100 @ 24.000000
New maxima : 0.948400 @ 25.000000
New maxima : 0.948500 @ 26.000000
New maxima : 0.948600 @ 28.000000
New maxima : 0.949100 @ 29.000000
Test Accuracy : 0.951300
Program completed in 149.09 seconds (pid: 3038).
```

Mini Batches	Timing
100	103.55 Seconds
200	93.43 Seconds
500	95.67 Seconds
1000	96.01 Seconds
5000	93.88 Seconds
10000	98.39 Seconds

6. Experiment with different numbers of layers and different number of neurons. Maintain the total numbers of weights and biases in the networks, while increasing the number of layers in the network. Describe how the performance changes as the numbers of layers increases – both in terms of training time and performance.

Parameters before adding a layer

```
Output | Local console | Markers
Program starting as '"/opt/zbstudio/bin/linux/x64/lua" -e "io.stc
Program 'lua' started in '/opt/zbstudio/myprograms' (pid: 4218).
 15910
[torch.LongStorage of size 1]
```

```lua
-- Create a two-layer network
module = nn.Sequential()
module:add(nn.Convert('bchw', 'bf')) -- collapse 3D to 1D
module:add(nn.Linear(1*28*28, 19))
module:add(nn.Tanh())
module:add(nn.Linear(19, 33))
module:add(nn.Tanh())
module:add(nn.Linear(33, 10))
module:add(nn.LogSoftMax())
module:cuda()
```

Parameters after adding the layer and different numbers of neurons.

```
Output | Local console | Markers
Program starting as '"/opt/zbstudio/bin/linux/x64/lua" -e "io.std
Program 'lua' started in '/opt/zbstudio/myprograms' (pid: 4228).
 15915
[torch.LongStorage of size 1]
```

7. Try one other training function from the list on this page: https://github.com/torch/optim/blob/master/doc/alogs.md . Compare the performance with gradient descent.

8. Try different Transfer functions and do comparison study between those. Explain your results.

```
-- Create a two-layer network
module = nn.Sequential()
module:add(nn.Convert('bchw', 'bf')) -- collapse 3D to 1D
module:add(nn.Linear(1*28*28, 19))
module:add(nn.ReLU())
module:add(nn.Linear(19, 33))
module:add(nn.ReLU())
module:add(nn.Linear(33, 10))
module:add(nn.LogSoftMax())
module:cuda()
```

Results:

```
Output | Local console | Markers
New maxima : 0.953600 @ 8.000000
New maxima : 0.954200 @ 9.000000
New maxima : 0.955500 @ 10.000000
New maxima : 0.957400 @ 11.000000
New maxima : 0.958900 @ 12.000000
New maxima : 0.959400 @ 15.000000
New maxima : 0.959700 @ 16.000000
New maxima : 0.959800 @ 18.000000
Test Accuracy : 0.956600
Program completed in 122.21 seconds (pid: 5028).
```

Execution time increase after changing the transfer function from Tanh() to ReLU().

9. Make a chart or table to see the advantages between mini batch, full batch and stochastic gradient. Calculate the timing by using the system clock command.

	Time	Advantages
Full Batch = 50,000	121.65 Seconds	Process the entire batch as one as has high learning rate as compare to stochastic gradient.
Mini Batch = 10,000	108.73 Seconds	Mini-batch has very high learning rate as compare to Full batch and Stochastic Gradient.
Stochastic Gradient	1325.70 Seconds	