**Lab 5 Training a Convolutional Neural Network**

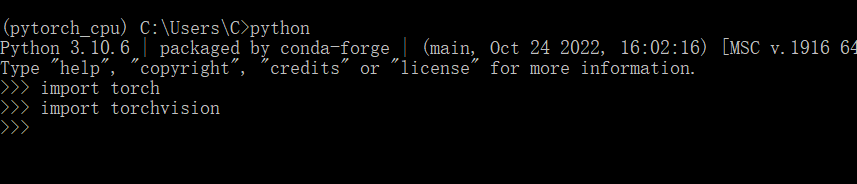
In this Lab, we will train a neural network on the CIFAR10 dataset. In Anaconda command prompt,

conda create -n pytorch\_cpu

conda install pytorch torchvision torchaudio cpuonly -c pytorch

conda install matplotlib

After installation of pytorch is done, activate the environment. Verify everything is working:



Open Spyder in Anaconda and execute mnist.py, the script will download the dataset online and perform training using the CPU. You will see the training is taking place. Take a look at the neural net we are using, class Net(). This is a Multi-level Perceptron with one hidden layer of 64 neurons. The input is flatten into 784x1 and the last layer has 32 neurons.

**Question 1** **[Data Visualization]:** Use the following command to answer the following questions. a) What is the size of the training set; b) What is the size of the test set. c) Copy and Paste the following command in Console and attach the results here.

figure = plt.figure(figsize=(10, 8))

cols, rows = 5, 5

for i in range(1, cols \* rows + 1):

sample\_idx = torch.randint(len(train\_data), size=(1,)).item()

img, label = train\_data[sample\_idx]

figure.add\_subplot(rows, cols, i)

plt.title(label)

plt.axis("off")

plt.imshow(img.squeeze(), cmap="gray")

plt.show()

Text

Description automatically generated

Text

Description automatically generated with medium confidence

**Question 2[Multi-Level Perceptron]:** Find the variable “num\_epochs” and change it to 20. Execute the program and wait until the training finishes. Then copy and paste the following command into the Console and show the plots. Illustrate what has been plotted and the trend of the curve.

Show the total execution time of training and go to your System settings to find your processor information. For example, my PC features an AMD Ryzen 1700 8-Core CPU of 3000 MHz and running 20 epoch takes 149s.

plt.plot(total\_train\_loss, label='Training loss')

plt.plot(total\_test\_loss, label='Test loss')

plt.legend()

plt.grid()

Text

Description automatically generated

Processor: Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz 3.00 GHz

Chart, line chart

Description automatically generated

**Question 3 [Convolutional Neural Network]:** We want to see how convolutional layers would help us increase the testing accuracy. The program contains another model Net2 with two convolutional layers of 5x5 and 3x3 followed by a max pooling layer. We add some dropout to avoid overfitting. Now we change “model = Net()” into “model = Net2()” and execute the program again for **10** epochs. Compare the results with Question 2 using MLP. Plot the same figure and compare the training and testing accuracy. Do we have an improvement in the testing accuracy? What about the execution time of the program.

It was initially at model = Net2(), and the I changed it to model = Net()

Text

Description automatically generated

Chart, line chart

Description automatically generated

In question 3, the training and test loss is higher on average than question 2. Training loss on question 3 makes significant drop after 5.

**Question 4 [Inference]:** After the model is trained, we want to perform inference on the trained model and evaluate the results. Copy and paste the following code into the Console for 5 different times and capture the output for each. Please screenshot the results in plots and attach the results below. Is the prediction correct most of the time?

testing\_data = enumerate(test\_loader)

batch\_idx, (images, labels) = next(testing\_data)

img = images[0]

img = img.to(device)

img = img.view(-1, 1, 28, 28)

print(img.shape)

# Since we want to use the already pretrained weights to make some prediction

# we are turning off the gradients

with torch.no\_grad():

logits = model.forward(img)

probabilities = F.softmax(logits, dim=1).detach().cpu().numpy().squeeze()

print(probabilities)

fig, (ax1, ax2) = plt.subplots(figsize=(6,8), ncols=2)

ax1.imshow(img.view(1, 28, 28).detach().cpu().numpy().squeeze(), cmap='viridis')

ax1.axis('off')

ax2.barh(np.arange(10), probabilities, color='b' )

ax2.set\_aspect(0.1)

ax2.set\_yticks(np.arange(10))

ax2.set\_yticklabels(np.arange(10))

ax2.set\_title('Prediction Probability')

ax2.set\_xlim(0, 1.1)

plt.tight\_layout()

Histogram

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

A picture containing shape

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Yes, the prediction is correct most of the time.

**Question 5 [Quantization]:** Recall that in order to run the neural nets on CPS devices with constrained resources, we often need to do **quantization**. Now let us see the effects of execution time, model size and accuracy trade-offs via converting the floating point 32 (fp32) into int8. Execute the following scripts respectively in the Console and compare the execution time, testing accuracy as well as the model size (the model size can be find in the workspace directory with the model name *model\_fp32* and *model\_int8*). Attach the screenshots of the results and discuss briefly.

start\_time = time.time()

test\_loss = test(model, device, test\_loader)

print("--- Total Execution Time of Fp32 model %s seconds ---" % (time.time() - start\_time))

torch.save(model.state\_dict(), "model\_fp32")

model\_int8 = torch.quantization.quantize\_dynamic(

model, # the original model

{torch.nn.Linear,torch.nn.Conv2d}, # a set of layers to dynamically quantize\

dtype=torch.qint8)

start\_time = time.time()

test\_loss = test(model\_int8, device, test\_loader)

print("--- Total Execution Time of INT8 model %s seconds ---" % (time.time() - start\_time))

torch.save(model.state\_dict(), "model\_int8")

Text

Description automatically generated

Both int8 and fp32 have approximately the same average loss and accuracy. The total execution time for fp32 is slightly faster than int8.

Model\_int8

Text

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Model\_fp32

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text, chat or text message

Description automatically generated

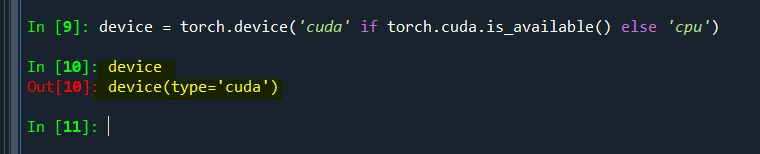
\*Both files have 13,295 lines of values. It is too big to screenshot the rest of them.

**Bonus Question [+2]:**

You can earn bonus question if you have a Nvidia GPU and installed CUDA. Create a new conda environment called pytorch\_gpu and install pytorch according your CUDA version.

<https://pytorch.org/get-started/locally/>

1. Confirm your installation is correct:



1. Run the program again and compare the execution time.