ELEC/COMP 447/546
Assignment 4
Due: Mar. 22, 2023, Midnight

Introduction

This assignment will introduce you to PyTorch and neural networks. We have provided a Colab notebook located here with skeleton code to get you started. Colab comes with a free GPU. To activate the GPU during your session, click *Runtime* on the top toolbar, followed by *Change runtime type*, and select GPU under hardware accelerator. You will find the GPU useful for quickly training your neural network in Problem 2.

1.0 PyTorch (10 points)

In this problem, you will perform some basic operations in PyTorch, including creating tensors, moving arrays between PyTorch and Numpy, and using `autograd`. **Before starting, please read the following pages:**

1. Tensor basics
2. Autograd basics, these two sections:
    a. Differentiation in Autograd
    b. Vector Calculus using autograd

1.1 Basics of Autograd (5 points)

a. In the provided notebook, fill in the function `sin_taylor()` with code to approximate the value of the `sine` function using the Taylor approximation (defined here). You can use `numpy.math.factorial()` to help you.
b. Create a tensor `x` with value $\pi/4$. Create a new tensor `y = sin_taylor(x)`. Use `y.backward()` to evaluate the gradient of `y` at `x`. Is this value a close approximation to the exact derivative of `sine` at `x`?
c. Now, create a NumPy array `x_npy` of 100 random numbers drawn uniformly from $[-\pi, \pi]$ (use np.random.uniform). Create a tensor `x` from that array and place the tensor onto the GPU. Again, evaluate `y = sin_taylor(x)`. This time, `y` is a vector. If you run `y.backward()`, it will throw an error because `autograd` is meant to evaluate the derivative of a *scalar* output with respect to input vectors (see tutorial pages above). Instead, run either one of these two lines (they do the same thing):

```
y.sum().backward()
y.backward(gradient=torch.ones(100))
```

What is happening here? We are creating a 'dummy' scalar output (let's call it `z`), which contains the sum of values in `y`, and acts as the final scalar output of our

computation graph. Due to the chain rule of differentiation, `dz/dx` will yield the same value as `dy/dx`.

d. Get the gradient tensor `dz/dx` and convert that tensor to a Numpy array. Plot `dz/dx` vs. `x_npy`, overlaid on a cosine curve. Confirm that the points fall on the curve and put this plot in your report.

1.2 Image Denoising (5 points)

In this problem, you will denoise this noisy parrot image, which we denote `I`. To do so, you will create a denoising loss function, and use `autograd` to optimize the pixels of a new image `J`, which will be a denoised version of `I`.

a. In your Colab notebook, implement `denoising_loss()` to compute the following loss function:

$$loss = \|I - J\|_1 + \alpha\left(\left\|\frac{dJ}{dx}\right\|_1 + \left\|\frac{dJ}{dy}\right\|_1\right)$$

The first component is a *data term* making sure that the predicted image `J` is not too far from the original image `I`. The second term is a *regularizer* which will reward `J` if it is smoother, quantified using `J`'s spatial derivatives. We have provided you a function `get_spatial_gradients()` to compute the gradients.

b. Implement gradient descent to optimize the pixels of `J` using your loss function and `autograd`. Initialize `J` to be a copy of `I`. Try different values for the learning rate and α and find a combination that does a good job. Put the smoothed image `J`, along with the learning rate and α you used in your report.

c. ELEC/COMP 546 Only: Change the loss function to use L2 norms instead of L1. Does it work better or worse? Why?

Hints (you should use all of these in your solution):

- `torch.clone`: performs a deep copy of a tensor
- To require storing gradients for a tensor `x`, use: `x.requires_grad_(True)`
- In the code, you will see the statement `with torch.no_grad():`. Any statements written within that block will not update the computation graph. Put your gradient descent step within that block, since that operation should not update the graph.
- Remember to zero out the gradient buffer of `J` after each step using `J.grad.zero_()`.
- Remember to normalize the gradient to a unit vector before using it in your gradient descent step.

- To plot an image in tensor `J` using matplotlib, you will have to first detach it from the computation graph (to not track its gradients), move it from the GPU to the CPU, and convert to a NumPy array. You can do this in one line with: `J = J.detach().cpu().numpy().`

2.0 Training an image classifier (10 points)

In this problem, you will create and train your first neural network image classifier! Before starting this question, please read the following pages about training neural networks in PyTorch:

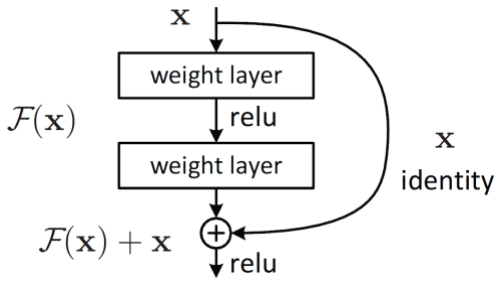
1. Data loading
2. Models
3. Training loop

We will be using the CIFAR10 dataset, consisting of 60,000 images of 10 common classes. Each image is of size 32 x 32 x 3. Download the full dataset as one *.npz* file here, and add it to your Google Drive. This file contains three objects: *X*: array of images, *y*: array of labels (specified as integers in [0,9]), and *label_names*: list of class names. Please complete the following:

a. Finish implementing the `CIFARDataset` class. See comments in the code for further instructions.
b. Add transforms: `RandomHorizontalFlip`, `RandomAffine` ([-5, 5] degree range, [0.8, 1.2] scale range) and `ColorJitter` ([0.8, 1.2] brightness range, [0.8, 1.2] saturation range). Don't forget to apply the `ToTensor` transform first, which converts a H x W x 3 image to a 3 x H x W tensor, and normalizes the pixel range to [0,1]. You will find the transform APIs in this page.
c. Implement a CNN classifier with the structure in the following table. You will find the APIs for Conv, Linear, ReLU, and MaxPool in this page. The spatial dimensions of an image should NOT change after a Conv operation (only after Maxpooling).

Layer	**Output channels for Conv/output neurons for Linear**
3 x 3 Conv + ReLU	50
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
MaxPool (2 x 2)	X
3 x 3 Conv + ReLU	100
Linear + ReLU	100
Linear	10

d. Implement the training loop.
e. Train your classifier for 15 epochs. The GPU, if accessible, will result in faster training. Make sure to save a model checkpoint at the end of each epoch, as you will use them in part f. Use the following training settings: batch size = 64, optimizer = Adam, learning rate = 1e-4.
f. Compute validation loss per epoch and plot it. Which model will you choose and why?
g. Run the best model on your test set and report:
    i. Overall accuracy (# of examples correctly classified / # of examples)
    ii. Accuracy per class
    iii. Confusion matrix: A 10 x 10 table, where the cell at row *i* and column *j* reports the fraction of times an example of class *i* was labeled by your model as class *j*. Please label the rows/columns by the object class name, not indices.
    iv. For the class on which your model has the worst accuracy (part ii), what is the other class it is most confused with? Show 5-10 test images that your model confused between these classes and comment on what factors may have caused the poor performance.
h. ELEC/COMP 546 Only: Change the last two Conv blocks in the architecture to Residual blocks and report overall accuracy of the best model. Recall that a residual block has the form:



Submission Instructions

All code must be written using Google Colab (see course website). Every student must submit a zip file for this assignment in Canvas with 2 items:

1. An organized report submitted as a PDF document. The report should contain all image results (intermediate and final), and answer any questions asked in this document. It should also contain any issues (problems encountered, surprises) you may have found as you solved the problems. **Please add a caption for every image specifying what problem number it is addressing and what it is showing**. The heading of the PDF file should contain:

    a. Your name and Net ID.
    b. Names of anyone you collaborated with on this assignment.
    c. A link to your Colab notebook (remember to change permissions on your notebook to allow viewers).
2. A pdf copy of your Colab notebook.

Collaboration Policy

I encourage collaboration both inside and outside class. You may talk to other students for general ideas and concepts, but you should write your own code, answer questions independently, and submit your own work.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.