EECS 4422/5323
Lab 5: Basic Deep Learning

Presented by: Calden Wloka

Outline

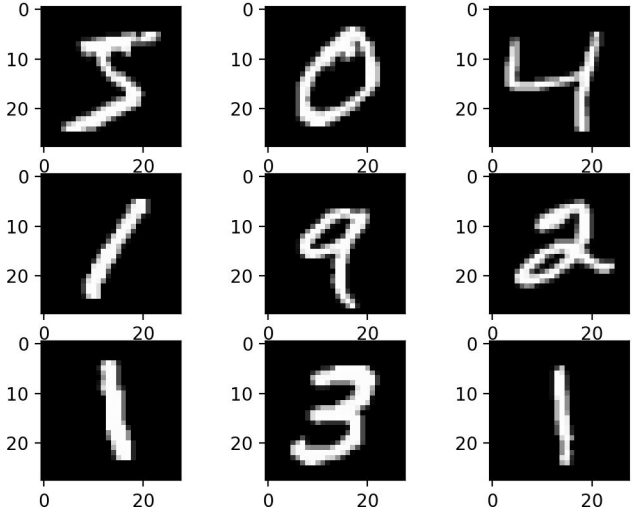
This lab will involve training small neural networks over a set of well-established datasets: MNIST and CIFAR10 using PyTorch

- Introduction to MNIST
- Defining a Basic Network
- Training
- Introduction to CIFAR10

MNIST

MNIST is a database of small images of handwritten numerical digits, 0-9.

- MNIST can be downloaded from: http://yann.lecun.com/exdb/mnist/
- Note that the native resolution of these images is 28x28 pixels
- Make sure you understand the role of each downloadable file



Defining a Basic Network - PyTorch

We will be using PyTorch to implement our network.

- PyTorch is a powerful deep learning library based in Python with many helpful functions and implementation tools (https://pytorch.org/)
- PyTorch can be run using a GPU or on the CPU; the networks we will be using today are small enough that CPU operations are not prohibitively slow
- Most of today's lab code has been adapted from the PyTorch tutorial: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

A Simple Network: LeNet

- You can find starter code in the file `LeNet_definition.py` which can be downloaded from the course webpage
- This code defines a LeNet class. There are two major components:
  - Constructor which initializes the network architecture and defines the layers
  - A `forward` function which describes how information flows through the network layers defined in the constructor
  - The code also includes a helper function for calculating how to *flatten* the elements in the network (turn them into a single dimensional array)
  - Make sure you understand this code; could you draw a network diagram for this network? Note that this network definition assumes 32x32 pixel input

Training the Network

- Normally you could use the `torchvision` package to access a dataloader for MNIST. However, this package is not available in the labs
- Therefore, you need to write your own data loader
  - It is often better to load data as you need it rather than load all images into memory at the start; for MNIST it may be possible to load all images (the images are small, after all), but for larger and more complex datasets this will not be possible, so it is good to understand how to write a dataloader
- The goal of your dataloader should be:
  - Iterate over a shuffled version of the training set, loading in the images and converting them to PyTorch tensors. Note you will also need to access their corresponding labels
  - Present images to the network in batches
  - Make sure the images are the correct input size - different options for this, and useful tools include `torch.nn.ZeroPad2d` or do the resize in numpy and before you convert to a PyTorch tensor

Connecting the Dataloader to Network Training

- Sample training code has been provided in comments in the `LeNet_definition` file
- Note that this training code is written assuming data output from the dataloader from `torchvision`, if your dataloader operates slightly differently, you'll need to update it
- Recommended batch size is 4

Probing Network Behaviour

- How does loss vary with training? Does it always go down, or do you sometimes jump around?
- Can you quantify performance by test category? Are some numbers harder than others?
- What happens if you change the batch size, learning rate, or momentum?

CIFAR10

- A dataset of very small colour images (32x32 RGB), which you can find here: https://www.cs.toronto.edu/~kriz/cifar.html
- Note that there is a Python version download
- Can you update your data loader to handle CIFAR10 instead of MNIST?
- How do you need to update the network to handle CIFAR10 input? What consideration needs to be made between MNIST to CIFAR10 data?