Columbia University
STAT GR5245 Fall 2024
Python for Deep Learning
Ka Yi Ng

Homework Assignment 1

HOMEWORK GUIDELINE

Submit your completed homework as an **html** file onto CourseWorks by the specified due date and time. Please find instructions on CourseWorks (GR5245_HwkSubmit.ipynb) for how a Jupyter notebook (.ipynb file) can be converted into an html file using a Google Colab instance. Before you convert your file, please ensure that all lines of code have been executed in order to show the desired results. The TA will grade your homework based on what are shown in the html file and will not attempt to re-execute the lines of code in your homework.

Note: If you suspect there are typos in this homework, or some questions are wrong, please feel free to email the instructor.

QUESTION 1

In class 2, we discussed auto differentiation which is used to compute gradients. Here we will define a 2-layer neural network using variable tensors (without using nn.Module) and train it on the MNIST dataset. The aim is to examine the effect of different optimizers and initial values for the weight/bias parameters on the training.

(a) Load the MNIST training dataset from `torchvision.datasets`. Plot the first few image examples.

We will use the first 10000 images for the following parts.

(b) Flatten the 2D image examples to 1D arrays with 784 features. Rescale the feature values to range between 0.0 and 1.0.

(c) Declare variables required for building a 2-layer neural network where the input layer has 784 neurons, the hidden layer has 128 neurons and the output layer has 10 neurons. Initialize both the weight values W_1, W_2 and the bias values b_1, b_2 to zero.

$$h_1 = ReLU(XW_1 + b_1)$$

$$Y = h_1 W_2 + b_2$$

(d) Write a function that takes X as input, computes and returns the predicted values Y in the output layer of the neural network.

(e) Write a training loop that trains the 2-layer neural network using the prepared training set in part (b) and the function defined in part (d). Set the number of training steps to 1000. Use `torch.optim.SGD` with learning rate = 0.001. Use `torch.nn.CrossEntropyLoss` to define

the loss function to be minimized. Print the training loss for every 100 steps (ie. at 100th, 200th, …) to monitor the convergence speed.

(f) In the training loop, also compute and keep the training loss for every single step. Plot a graph to show how the training loss changes as the number of training steps increases.

(g) Train another neural network of the same architecture using `torch.optim.Adam` with learning rate = 0.001. Which one, SGD or Adam gives a better convergence speed in this case?

Note: for a proper comparison of convergence, please ensure that the same random seed is used in both cases. If you want to re-use the same weight variables, make sure that they are initialized to zero before training.

(h) In the previous parts, weights were initialized to zero. Now train a 3rd neural network of the same structure, but the weights are initialized with random samples of a normal distribution with mean 0 and standard deviation of 0.1. Use the optimizer as determined by part (g). Which weight initialization gives better convergence speed? Zero or random normal samples?

Note: https://arxiv.org/pdf/2102.07004.pdf provides a good reference of weight initialization methods being used.

(i) Plot the training loss for the above 3 training loops to compare the effect of using different optimizers and weight initialization.

--end of Assignment --