BME695DL/ECE695: Homework 3

Spring 2021
Due Date: Monday, March 01,2021 (11:59pm)

Turn in your solutions via BrightSpace.

1 Introduction

The main goal of this homework is for you to develop a greater appreciation for the optimizations used for estimating the step size in the hyperplane spanned all the learnable parameters in an SGD based approach to updating the values of the parameters.

1.1 Steps

1. Download and install version 1.0.5 of your instructor's ComputationalGraphPrimer. You may not want to "sudo pip install" the Primer since that would not give you the Examples directory of the distribution that you are going to need for the homework. Here is the main documentation page for the Primer:

   `https://engineering.purdue.edu/kak/distCGP/ComputationalGraphPrimer-1.0.5.html`

2. Go to the Examples directory of the distribution and execute the following scripts:

   ```
   python3  one_neuron_classifier.py
   ```

   ```
   python3  multi_neuron_classifier.py
   ```

3. The final output of both these scripts is a display of the training loss versus the training iterations.

4. Now execute the following script in the Examples directory

   ```
   python3  verify_with_torchnn.py
   ```

   Unless you make changes to the script in the Examples directory, the loss vs. iterations graph that you will see is for a network that is a "torch.nn" version of the handcrafted network you get through the script "multi_neuron_classifier.py"

Compare mentally the output you get with the above call with what you saw for the second script in Step 2.

5. Now make a couple of changes to the file "verify_with_torchnn.py" in order to see the "torch.nn" based output for the one-neuron model. The changes you need to make are mentioned in the documentation part of the file "verify_with_torchnn.py".

   Again compare mentally the loss-vs-iterations for the one-neuron case with the handcrafted network vis-a-vis the torch.nn based network.

6. For both the one-neuron and the multi-neuron cases, you will see a dramatic improvement in the performance with the "torch.nn" based implementations of the network. A significant portion of this improvement can be attributed to the use of step optimization for the "torch.nn" based code.

7. Now comes the hard part of this homework:

   If you'd look at the code in Version 1.0.5 of the Primer, you will notice that it does NOT use any step optimizations for the gradient descent. In other words, the update steps in the Primer are based solely on the current value of the gradient of the loss with respect to the parameter in question. That is,

$$p_{t+1} = p_t - \text{lr} * g_{t+1} \tag{1}$$

   where p_t denotes learnable parameters from the previous time step, *e.g.*, layer weights at iteration t, and g_{t+1} denotes the corresponding gradient for the current time step $t+1$.

   Your work for this homework must at least bring in the notion of momentum for calculating the step size. In its simplest implementation, using momentum only involves remembering the step size used at the previous iteration and then making the current step-size decision based on the current value for the gradient and the previous value for the step size.

   In order the invoke the notion of momentum for step optimization, you have to compute the step updates separately. This makes it more convenient to base the current step size on both its previous value and the current value of the gradient, as shown below. In the formulas shown, the variable v is the step size and the first equation is the recursive update formula for its update. As you can see, for calculating the step size to use at the current iteration $t+1$, we use a fraction μ of its value at the previous iteration. [If you are puzzled by the appearance of the learning-rate in the first equation rather than in

the second, that's because the basic function of the learning rate is to control what fraction of the current value of the gradient to use at the current iteration.]:

$$\begin{aligned} v_{t+1} &= \mu * v_t - \text{lr} * g_{t+1}, \\ p_{t+1} &= p_t + v_{t+1}. \end{aligned} \tag{2}$$

The momentum scalar $\mu \in [0, 1]$ decides weight to the previous time step update. The v_0 is typically initialized with all zeros.

2 Programming Task

Your main programming task is to implement SGD+, a momentum enhanced version of the basic SGD you see in `one_neuron_classifier.py` and `multi_neuron_classifier.py`.

As explained in Section 1.1, the Steps 1-6 are for you to become familiar with Version 1.0.5 of the Primer. Prof. Kak's Week 4 lecture slides explain the basic logic of the implementation code for `one_neuron_classifier.py` and `multi_neuron_classifier.py`.

More specifically, your programming task is to create new versions of the one-neuron and multi neuron-classifiers that are based on SGD with momentum (SGD+) optimizer (Ref. Eq. 2).

Since you are going to be modifying the main module file `ComputationalGraphPrimer.py`, that raises the question of what it is you are expected to submit for this homework.

For submission, bundle all of the code you wrote for the SGD+ based implementation of the one-neuron classifier and place it in a file named `one_neuron_classifier_sgd_plus.py`.

Do the same for your SGD+ based implementation for the multi-neuron classifier, but this time place it in a file called `multi_neuron_classifier_sgd_plus.py`.

Your homework submission will consist of these two files.

Also note that, depending on how you decide to implement SGD+, you may need one or more helper functions of your own in the module file `ComputationalGraphPrimer.py`. Feel free to add those functions to your

own install of the module. Make sure you incorporate these helper functions in the two submission files named above.

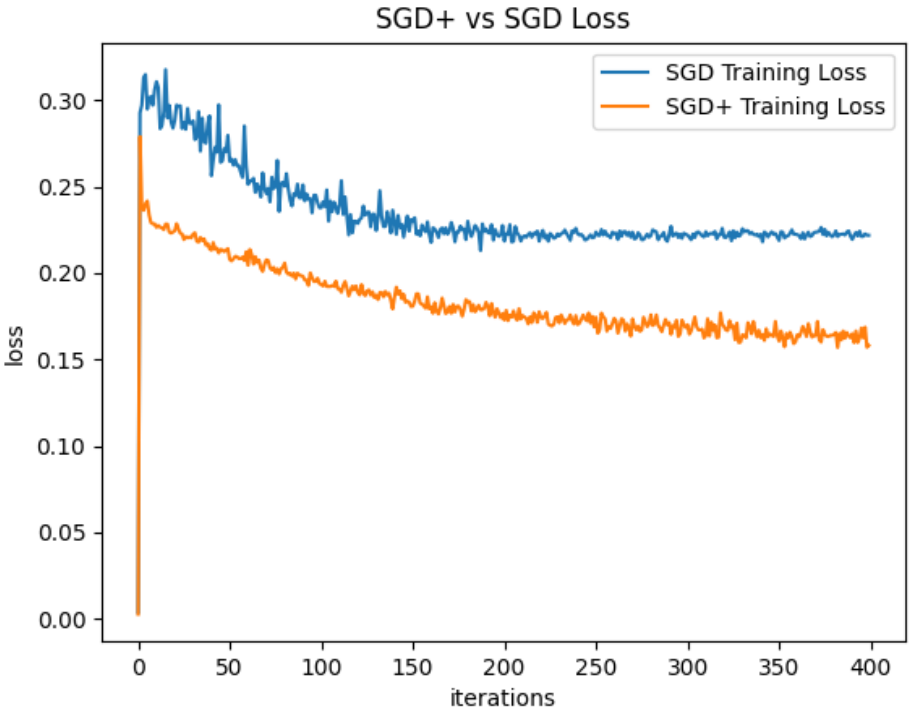

Figure 1: Sample comparative plot (SGD+ vs SGD) for the one-neuron network. Your results could vary depending on your choice of the training parameters. All the plot formatting related options are also flexible.

The important thing related to this homework is that you are not allowed to use PyTorch's built-in SGD optimizer.

The expected output from

1. `python3 one_neuron_classifier_sgd_plus.py` call is `one_neuron_loss.jpg`
2. `python3 multi_neuron_classifier_sgd_plus.py` call is `multi_neuron_loss.jpg`

Fig. 1 shows an example of the comparative plots from the one-neuron classifier. This plot is shown just to give you an idea of the improvement achieved from SGD+ over SGD. Your results could vary based on your choice of the parameters, such as learning rate, μ, batch size, number of iterations, etc. Note that loss-vs-epochs plots are also acceptable.

3 Submission Instructions

- Make sure to submit your code in Python 3.x and not Python 2.x.
- Create a .zip archive with the following four files:

  `one_neuron_classifier_sgd_plus.py`, `one_neuron_loss.jpg`,

  `multi_neuron_classifier_sgd_plus.py`, `multi_neuron_loss.jpg`.

  Name the .zip archive as `hw03_<Firstname><Lastname>.zip` (without any white spaces). Note that .rar file format is Windows specific so please do NOT submit your solutions in .rar format. **Your code must be your own work.**
- You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission.