BME646 and ECE60146: Homework 3

Spring 2023
Due Date: 11:59pm, Feb 06, 2023

Turn in typed solutions via BrightSpace. Additional instructions can be found at BrightSpace. Late submissions will be accepted with penalty: **-10 points per-late-day, up to 5 days**.

1 Introduction

The main goal of this homework is for you to develop a greater appreciation for the step size optimization logic that you will use for training deep neural networks. In the programming tasks, you will first run example scripts demonstrating the effects of using a vanilla Stochastic Gradient Descent (SGD) optimizer and see its shortcomings. Subsequently, you will be tasked to augment the vanilla SGD optimizer with momentum (SGD+) and Adaptive Moment Estimation (Adam). For more information on the topics covered in this HW, please refer to Prof. Kak's slides on Autograd [1].

2 Becoming Familiar with the Primer

1. Download the `tar.gz` archive and install version 1.0.9 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-1.0.9.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
   ```

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

3. 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.

4. 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.

   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.

5. Now comes the hard part of this homework:

   If you'd look at the code in Version 1.0.9 of the Primer, you will notice that it does NOT use any step optimizations in SGD. 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,

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

   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 consists of adding step-size optimization to training with the one-neuron and multi-neuron networks in the CGP primer. In order to fully appreciate what that means, it is recommended that you carefully review the material in the section "Step Size Optimization for SGD" in the Week 3 slides by your instructor [1]. This section consists of the Slides 103 through 115. As you will see in these slides, the two major components of step-size optimization are: (1) using momentum; and (2) adapting the step sizes to the gradient values of the different parameters. (The latter is also referred to as dealing with sparse gradients.) Both of these are incorporated in what's currently the world's most popular step-size optimizer: Adam (Adaptive Moment Estimation).

- **SGD with Momentum (SGD+)**: 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 for individual learnable parameters. 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:

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

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

- **Adaptive Moment Estimation (Adam)**: During the last few years, Adam has become one of the most widely used step-size optimizers for SGD in deep learning owing to its efficiency and robust performance especially on large datasets. The key idea behind Adam is a joint estimation of the momentum term and the gradient adaptation term in the calculation of the step sizes. It does so by keeping the running averages of both the first and second moment of the gradients, and take both moments into account for calculating the step size. The equations below demonstrate the key logic:

$$\begin{aligned} m_{t+1} &= \beta_1 * m_t + (1 - \beta_1) * g_{t+1}, \\ v_{t+1} &= \beta_2 * v_t + (1 - \beta_2) * (g_{t+1})^2, \\ p_{t+1} &= p_t - \mathrm{lr} * \frac{\hat{m}}{\sqrt{\hat{v}_{t+1} + \epsilon}}, \end{aligned} \tag{3}$$

where the definitions of the bias-corrected moments $\hat{m}$ and $\hat{v}$ can be found on Slide 115 of [1]. In practice, β_1 and β_2, which control the decay rates for the moments, are generally set to 0.9 and 0.99, respectively.

3 Programming Task

- Your main programming task is two-fold: implementing SGD+ and Adam based on the basic SGD you see in `one_neuron_classifier.py` and `multi_neuron_classifier.py`.

  As explained in Section 2, the Steps 1-4 are for you to become familiar with Version 1.0.9 of the Primer. Prof. Kak's slides on Autograd 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+ as well as Adam.

- Note that for the implementation of both SGD+ and Adam, **modifying the main module file `ComputationalGraphPrimer.py` is NOT recommended.** Instead, you should create subclasses that inherit the `ComputationalGraphPrimer` class provided by the module. In your subclasses, create or override any class methods as your implementation requires. Also, it should be stressed that you are not allowed to use PyTorch's built-in SGD optimizer.

- 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.

4 Submission Instructions

Include a typed report explaining how did you solve the given programming tasks.

1. Your pdf must include a description of

   - A description of both SGD+ and Adam in your own words with key equations.

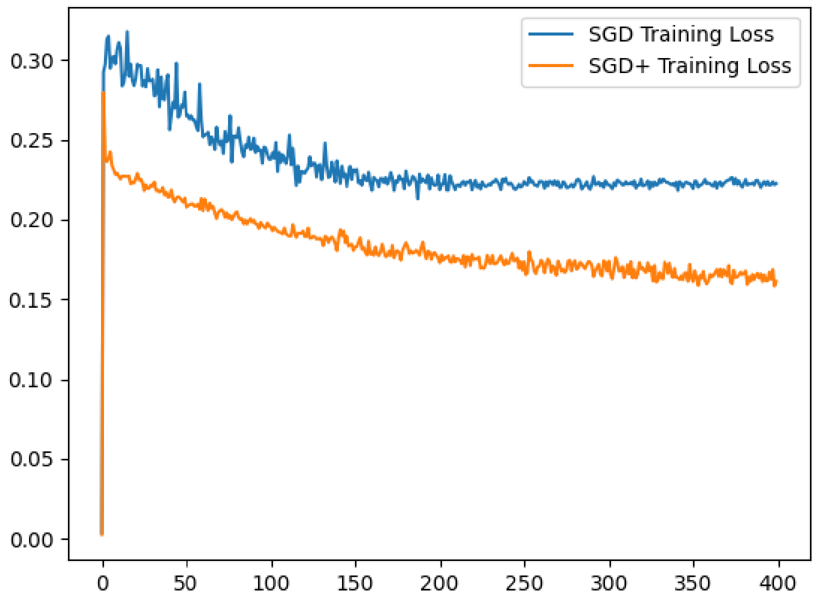

Figure 1: Sample comparative plot (SGD+ vs SGD) for the one-neuron network. Your results could vary depending on your choice of the training parameters. All the plot formatting related options are also flexible.

- For the one-neuron classifier, a plot of training loss vs iteration comparing all three optimizers (SGD, SGD+, Adam). Another of the same plot but with a different learning rate of your choice.
- The same comparative plots with two different learning rates for multi-neuron.
- Discuss your findings comparing the performance of the three optimizers in one or two paragraphs.
- Your source code. Make sure that your source code files are adequately commented and cleaned up.

2. Turn in a zipped file, it should include (a) a typed self-contained pdf report with source code and results and (b) source code files (only .py files are accepted). Rename your .zip file as hw3_<First Name><Last Name>.zip and follow the same file naming convention for your pdf

report too.

3. For all homeworks, you are encouraged to use `.ipynb` for development and the report. If you use `.ipynb`, please convert it to `.py` and submit that as source code.

4. You can resubmit a homework assignment as many times as you want up to the deadline. Each submission will overwrite any previous submission. **If you are submitting late, do it only once on BrightSpace.** Otherwise, we cannot guarantee that your latest submission will be pulled for grading and will not accept related regrade requests.

5. The sample solutions from previous years are for reference only. **Your code and final report must be your own work.**

6. To help better provide feedbacks to you, make sure to **number your figures**.

References

[1] Autograd for Automatic Differentiation and for Auto-Construction of Computational Graphs. URL https://engineering.purdue.edu/DeepLearn/pdf-kak/AutogradAndCGP.pdf.