Dr. Tolga Soyata *© George Mason University, Department of ECE* *tsoyata@gmu.edu*

ECE655 (Advanced GPU Programming and Deep Learning) Project 3 (15 points)

Revised Deadline: Oct 12, 2025 11:59PM. 2 point penalty per each late day

- In this project, you will be using your knowledge from Lecture 4 and apply Linear Regression using numpy. You will be using multiple optimizers and loss functions.
- Document your results in a report, written in LaTeX (Overleaf). The report is expected to be 4-5 pages, which is packed with plots and tables.
- Your computer must have the full Anaconda package, PyTorch, and tensorboard packages installed to complete this project.
- Name your code `P3A.py`, `P3B.py`, etc, i.e., one program per part.
- Submit your report and code files under **Project 3 Submission Area**, *as a single ZIP file*, which must include:
  - Your Python source files and other necessary files for me to generate your results.
  - Your Report PDF and source files (the entire Overleaf directory)

PROJECT 3 DESCRIPTION:

In this project, you will download an open source linear dataset and perform linear regression on it by using the specified hyper-parameters. For your report, the best practice is to have **a)** one or more separate Python program files, and **b)** a separate section for each PART in your report.

PART A (2 point) DATA GENERATION AND VISUALIZATION:

Previously, you either measured diode I-V, generated synthetic data, or found the data on the web. It is time to start using open datasets, which are specifically designed to test linear models. Although the deep learning examples will require much larger datasets, the linear regression in this project will work with much smaller datasets (hundreds or even thousands of data points if possible). Go to this link and find your favorite linear dataset:

https://www.telusinternational.com/insights/ai-data/article/10-open-datasets-for-linear-regression

The data will be usually either in a Comma Separated Values (CSV) or Excel file (XLSX) format. These are straightforward file formats that are very easy to convert to the PyTorch tensor format, which is what you need. Commonly, the open source data provider gives you the Python code to convert the data into another format (numpy array or C array, or even PyTorch tensors). If not, typically all you have to do is to import a package that is aware of the file format, such as **`import csv.`** See: https://docs.python.org/3/library/csv.html).

You should be using the built-in **`Dataset`**, **`TensorDataset`**, and **`DataLoader`** classes. Split your dataset 80:20 for training vs. validation. Plot your data (training and validation). All of your data will be in CPU in PART A.

PART B (3 points) MODEL GENERATION:

Generate a linear model with the **`SGD`** optimizer, **`MSELoss`** loss function, batch size of **full**, which will give you only a single update per epoch. Run it for 100 epochs and plot the training vs. validation losses for 100 epochs using tensorboard. Everything will run in the CPU in PART B.

PART C (1 point) LOSS FUNCTION:

Keep everything the same, epochs=100. Change the loss function to **`L1Loss`**. Repeat the same experiment. Get a separate plot. This part is CPU only. Comment on the results. How do the results differ? What changed?

PART D (1 point) OPTIMIZER:

Keep everything the same, epochs=100. Full batch size. Change the loss function to **`MSELoss`** and the optimizer to **`Adam`**. There are multiple flavors of **`Adam`**. Choose the one you like. You will need to choose the parameters that **`Adam`** requires. Generally it is a good idea to keep the ones you are not sure about at their default value. The point is to analyze the impact of the optimizer. Get a separate plot. This part is CPU only. Comment on **`Adam`** vs. **`SGD`**. What changed?

PART E (1 point) BATCH SIZE:

Keep loss function = **`MSELoss`**, optimizer = **`SGD`**. Epochs=100. Try different batch sizes in [4, 8, 16, 32, 64]. Get a separate bar plot (x axis is batch size and y axis is the final loss value). This part is CPU only. Comment on your results. How does the batch size effect your results?

PART F (4 points) EPOCHS AND ALGORITHM RUNTIME (CPU):

Use the loss function **`MSELoss`**. Write a Python script that

- sweeps epochs from 20 to 1000, steps of 20 **50** runs
- Sweeps optimizer in [**`SGD`**, **`Adam`**] **2** runs
- Save the results of these 100 runs ($\mathbf{50} * \mathbf{2}$) in a numpy array. Each run will have a tuple: (finalized training loss, amount of time elapsed).

Draw a double-y-axis matplotlib plot that has epochs in the x axis and (loss, time elapsed) in the double y axis (y1 axis = loss, y2 axis = time elapsed). Your plot should have 2 curves. Ignore the validation loss in this part. This part is CPU only. Comment on your results.

PART G (3 points) EPOCHS AND ALGORITHM RUNTIME (GPU):

Repeat PART F using GPU only. Use Google Colab or your local PC. Your data and model must be completely in the GPU. The most important analysis of this part is how much GPU accelerates the training time for such an intense run. Your report should focus on this.

⇨ **General Note for PART F and G:**

The most important aspect of PART F and PART G is the automation of a significant amount of program runs. If you run them one by one, you will spend (more like waste) an incredible amount of time. However, if you automate them, it is like running a single program. You can start running it and have it report the results of the run in a file. This is how professional training sessions are done in the industry. Very similar to large circuit simulations done at INTEL, etc. They may take hours and may require running the same simulation with 100s of different parameters. Nobody runs them one by one. They automate it in exactly the same way you will ...

Considering that Python is infinitely friendly in terms of generating this kind of an automation program, this is your chance to harness the power of Python.

Remember that you can have lines in your for loop that look like the one shown below:

```
opti1=Adam(...)
opti2=SGD(...)

for optimizer in [opti1,opti2]:
   ...
```