Homework 1: Gradient Descent and Optimization CPE 487/587

Instructor: Rahul Bhadani

Due: Jan 25, 2026, 11:59 PM
50 Points

You are allowed to use a generative model-based AI tool for your assignment. However, you must submit an accompanying reflection report detailing how you used the AI tool, the specific query you made, and how it improved your understanding of the subject. You are also required to submit screenshots of your conversation with any large language model (LLM) or equivalent conversational AI, clearly showing the prompts and your login avatar. Some conversational AIs provide a way to share a conversation link, and such a link is desirable for authenticity. Failure to do so may result in actions taken in compliance with the plagiarism policy.

Additionally, you must include your thoughts on how you would approach the assignment if such a tool were not available. Failure to provide a reflection report for every assignment where an AI tool is used may result in a penalty, and subsequent actions will be taken in line with the plagiarism policy.

Submission instruction:

Upload a .pdf on Canvas with the format `CPE487587-LastFirst-HW-XX`. For example, if your name is Sam Wells, your file name should be `CPE487587-LastFirst-HW-XX.pdf`. If there is a programming assignment, then you should include your source code along with your PDF files in a zip file `CPE487587-LastFirst-HW-XX.zip`. Your submission must contain your name and UAH Charger ID or your UAH email address. Please number your pages as well.

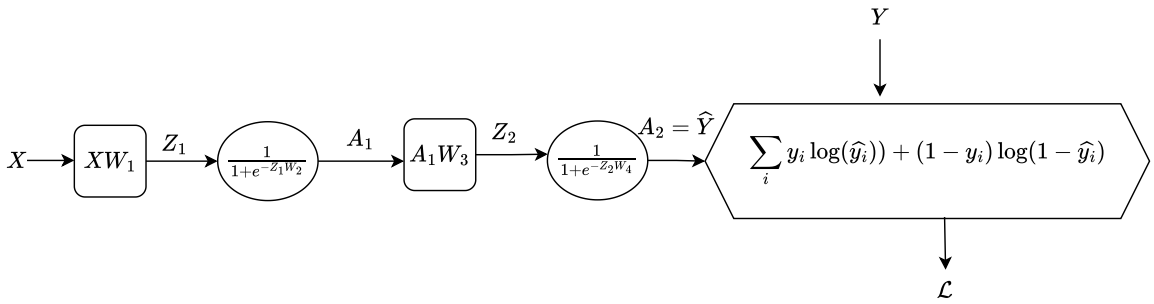
All CPE 587 students are required to submit their response in a Latex-generated PDF.

Before you attempt, please read the entire problem end-to-end.

For this assignment, you will only report GitHub commit hash as a submission and nothing else, and this supercedes the previous submission instruction.

1 Finding Optimal Weights using Gradient Descent

Consider a sequence of linear and nonlinear operations on a feature matrix from n samples and d features as shown in Figure 1.



The goal of this homework is to implement gradient-descent-based optimization using the power of automatic differentiation provided by the PyTorch library. We will find optimal weights to create a model that performs binary classification. For this problem consider a data set as $\{\mathcal{D} = (X, Y)\}$, where $X \in \mathbb{R}^{n \times d}$ is the feature matrix and $Y \in \mathbb{R}^{n \times 1}$ is the true label vector for all n samples.

Consider the following requirements for the sequence of linear and non-linear actions above:

1. $W_1 \in \mathbb{R}^{d \times 48}$.
2. $W_2 \in \mathbb{R}^{48 \times 16}$.
3. $W_3 \in \mathbb{R}^{16 \times 32}$.
4. $W_4 \in \mathbb{R}^{32 \times 1}$.

Your code should meet the following requirements:

1. You should generate a random matrix X of size $n \times d$ that will serve as a feature matrix. Use `torch.randn`. It should be a torch Tensor of data type Float32. **(2 Points)**
2. You should generate a label vector of size $n \times 1$ following the rule that if the sum of features for one data sample is greater than 2, it should have a label of 1, otherwise 0. **(2 Points)**
3. Weight matrice W_1 should be initialized from Gaussian distribution of 0 mean and standard deviation $\sigma = \sqrt{\frac{2}{d}}$. Similarly, it would be $\sigma = \sqrt{\frac{2}{48}}$ for W_2 and so on. **(2 Points)**

4. Select learning rate of $\eta = 0.001$ and perform training for upto 10000 epochs using gradient descent. Following the Chapter 2 course notebook for an example. **(2 Points)**

5. Your code should check for the presence of a GPU and create GPU tensors if the machine has a GPU; otherwise, CPU. The training portion should take advantage of a GPU. **(2 Points)**

1.1 Deliverables

1. Write a python function `binary_classification` in a file `two_layer_binary_classification.py`. The function should be able to take as an argument the following:

   (a) Number of features d, number of samples n, number of epochs $epochs$ with default value of 10000, learning rate η with a default value of 0.001.

   (b) It should return updated weight matrices after the completion of the training, and a loss vector that keeps the historical loss value for every epoch.

   (20 Points)

2. You should create a new subpackage `deepl` in your UV package and add `two_layer_binary_classification.py` as a module to the subpackage. **(2 Points)**

3. Appropriately initialize `__init__.py` so that you may be able to import your package. **(2 Points)**

4. Create a new subfolder called `scripts` in the root directory of your package, where you will add `binaryclassification_impl.py` demonstrating the use of the function `binary_classification`. **(2 Points)**

5. Your script should retrieve the loss vector and create a plot of the loss value against epochs. Your script should save the plot as a pdf file with the file number `crossentropyloss_20260115133203.pdf` where your script should automatically replace 20260115133203 by the actual timestamp YYYYMMDDhhmmss, Y = Year, M = Month, D = Day, h = hour, m = minutes, s = seconds. **(10 Points)**

6. Add a README.md to the UV project root directory explaining how to install your package, run your code, and get the result. **(2 Points)**

7. Commit all the files to GitHub and report the commit hash as a part of the submission. I will only evaluate your code to the point of the commit hash. **(2 Points)**

No Jupyter Notebook or Google Colab link is required for the submission.