CSE543 Deep Learning Homework 1

January 23, 2026

- Deadline: February 5th.
- Include your name and UW NetID in your submission. You only need to submit one PDF file for the text answer (including the inline questions for the programming task) to Canvas, and a zip file for the code to Gradescope.
- Homework must be typed. You can use any typesetting software you wish (LaTeX, Markdown, MS Word etc).
- You may discuss assignments with others, but you must write down the solutions by yourself.

1 Auto-differentiation (10 points)

Consider the following function:

$$f(w_1, w_2) = (\sin(2\pi w_1/w_2) + 3w_1/w_2 - \exp(2w_2)) \cdot (3w_1/w_2 - \exp(2w_2))$$

Suppose our program for this function uses the following evaluation trace:
Input: $z_0 = w \triangleq (w_1, w_2)$

1. $z_1 = w_1/w_2$.
2. $z_2 = \sin(2\pi z_1)$.
3. $z_3 = \exp(2w_2)$.
4. $z_4 = 3z_1 - z_3$.
5. $z_5 = z_2 + z_4$.
6. $z_6 = z_4 z_5$.

output: z_6.

Q1.1 **Reverse mode of auto-differentiation** (3 Points) Consider the reverse mode to compute the gradient df/dw, which is a two dimensional vector. Explicitly write out the reverse mode in this example. Write the pseudocode computing all the intermediate derivative as you would actually compute them in an implementation. You may assume you have evaluated the "trace" and have already stored $(z_0, \ldots, z_6)$. Your pseudocode for computing $\frac{dz_6}{dz_t}$ should *only* be written as a function of $z_1, \ldots, z_6$ and the derivatives $\frac{dz_6}{dz_{t+1}}, \ldots, \frac{dz_6}{dz_6}$ (as is specified by the reverse mode algorithm). For initialization, let $\frac{dz_6}{dz_6} = 1$. In particular, it must be written in the manner that the reverse mode is implemented taught in class (as to be an efficient algorithm). You should basically have the same number of lines of pseudocode as computing the original function.

Q1.2 (3 points) Now we consider another approach for auto-differentiation. This is a conceptually simpler way than the reverse mode. This question asks to compute $\frac{dz_6}{dw_1}$. The forward mode computes the derivative in a sequential manner, directly using the previous variables and the chain rule. The idea is as follows: at time t, we

(a) Compute $z_1, \ldots, z_6$ as usual.

(b) Compute $\frac{dz_t}{dw_1}$ for $t = 1, \ldots, 6$ using our previously computed derivative $\frac{dz_1}{dw_1}, \ldots, \frac{dz_{t-1}}{dw_1}$ using the chain rule:

$$\frac{dz_t}{dw_1} = \sum_{\tau<t} \frac{\partial z_t}{\partial z_\tau} \frac{\partial z_\tau}{\partial w_1}.$$

Explicitly write out the forward mode in our example, where you will write out the pseudocode computing all the intermediate derivatives as you would actually compute them in an implementation. You will be writing out a series of steps (the pseudocode) where you will be computing both z_t and its derivative $\frac{dz_t}{d_w}$. Your pseudocode for computing $\frac{dz_t}{dw_1}$ should only be written as a function of $z_1, \ldots, z_{t-1}$ and the previous derivatives $\frac{dz_1}{dw_1}, \ldots, \frac{dz_{t-1}}{dw_1}$.

Q1.3 (4 points) Now we consider forward mode of auto-differentiation for general functions.

Q1.3.1 (2 points) Now suppose we have a general function $f : \mathbb{R}^d \to \mathbb{R}$, and we want to compute $\frac{df}{dw}$. Describe how you would do this with the forward mode.

Q1.3.2 (2 points) Let T denote the computation time to compute $f(w)$. Obtain an upper bound of the computation time $\frac{df}{dw}$ in big-O notation in terms of T and d.
Remark: the time for computing $\frac{df}{dw}$ using the reverse mode is $O(T)$.

2 Implementation: Multi-layer NNs, Image Features, Optimizers (33 Points)

In this problem, you will practice writing backpropagation code, and training Neural Networks. The goals are as follows:

- Implement and apply a Two layer neural network classifier.
- Get a basic understanding of performance improvements from using higher-level representations as opposed to raw pixels, e.g. color histograms, Histogram of Oriented Gradient (HOG) features, etc.
- Understand Neural Networks and how they are arranged in layered architectures.
- Understand and be able to implement (vectorized) backpropagation.
- Implement various update rules used to optimize Neural Networks.

Q2.1 **Two-Layer Neural Network** (16 Points) The notebook `two_layer_net.ipynb` will walk you through the implementation of a two-layer neural network classifier.

Inline Q1: Now that you have trained a Neural Network classifier, you may find that your testing accuracy is much lower than the training accuracy. In what ways can we decrease this gap? Select all that apply.

1. Train on a larger dataset.
2. Add more hidden units.
3. Increase the regularization strength.
4. None of the above.

Q2.2 **Higher Level Representations: Image Features** (4 Points) The notebook `features.ipynb` will examine the improvements gained by using higher-level representations as opposed to using raw pixel values.

Q2.3 **Fully-connected Neural Network** (13 Points) The notebook `FullyConnectedNets.ipynb` will introduce you to our modular layer design, and then use those layers to implement fully-connected networks of arbitrary depth. To optimize these models you will implement several popular update rules.

Inline Q2: AdaGrad, like Adam, is a per-parameter optimization method that uses the following update rule:

```
cache += dw**2
w += - learning_rate * dw / (np.sqrt(cache) + eps)
```

John notices that when he was training a network with AdaGrad that the updates became very small, and that his network was learning slowly. Using your knowledge of the AdaGrad update rule, why do you think the updates would become very small? Would Adam have the same issue?

Important. Please make sure that the submitted notebooks have been run **and the cell outputs are visible**. Below are the steps for submission:

1. Open `collect_submission.ipynb` in Colab and execute the notebook cells. This notebook/script will generate a zip file of your code (`.py` and `.ipynb`) called `a1_code_submission.zip`. If your submission for this step was successful, you should see the following display message:

   `### Done! Please submit a1_code_submission.zip to Gradescope. ###`

2. Submit the zip file to Gradescope. Remember to download `a1_code_submission.zip` locally before submitting to Gradescope.

3. Ensure that you have answered the inline questions scattered throughout the notebooks **in the PDF file** instead of the notebooks.

3 Initialization for Leaky ReLU (13 Points)

Consider an H-layer neural network of the following form:

$$f(x, W^1, \ldots, W^{H+1}) = W^{H+1}\sigma(W^H\sigma(\cdots\sigma(W^1x)))$$

where $x \in \mathbb{R}^{d_1}$, $W^h \in \mathbb{R}^{d_{h+1}\times d_h}$ for $h = 1, \ldots, H$, and $W^{H+1} \in \mathbb{R}^{d_H}$. Here $\sigma(\cdot)$ is the leaky ReLU activation function: $\sigma(z) = \max\{0, z\} + \alpha \min\{0, z\}$. Let W^h_{ij} be the (i, j)-th entry of W^h. We initialize W^h_{ij} from a standard normal distribution with mean 0 and variance β_h.

Q3.1 (3 points) Let $z^h = W^h\sigma(W^{h-1}\sigma(\cdots\sigma(W^1x)))$. Derive β_h such that the variance for z^h is the same for all h.

Q3.2 (10 points) In this problem, you will experiment with fully-connected neural networks with different depths and αs in Leaky ReLU on the MNIST dataset, where you use only the images corresponding to the digits 0 and 1 (binary classification). Use logistic loss and stochastic gradient descent to train the neural network. You can tune the learning rate and the number of epochs. We recommend using PyTorch. If you plan to use PyTorch, you can use Dataloader to use the MNIST dataset directly. For this question, you can use any modules you want.

For this problem, we will fix the width $m = 256$ for all layers. Consider the four depths $H = 10, 20, 30, 40$ and four αs: $\alpha = 2, 1, 0.5, 0.1$. Plot 16 figures. Each figure corresponds to one pair of (H, α). Each figure has two curves: one corresponds to Kaiming initialization (normal distribution with variance $2/d_h$) and one corresponds to the normal distribution initialization with the variance scaling derived in Problem 4.1. Note you need to use the same learning rate and batch size for both initialization scalings. But you can use different learning rates and batches for different (H, α)s.

For each figure, you need to list all the hyperparameters used, either in the text or in the figure caption. Please attach your codes in your submission. Since this problem asks you to choose your own hyper-parameters, as long as your plots are reasonable, you will receive full credits. A example of plot is shown below.

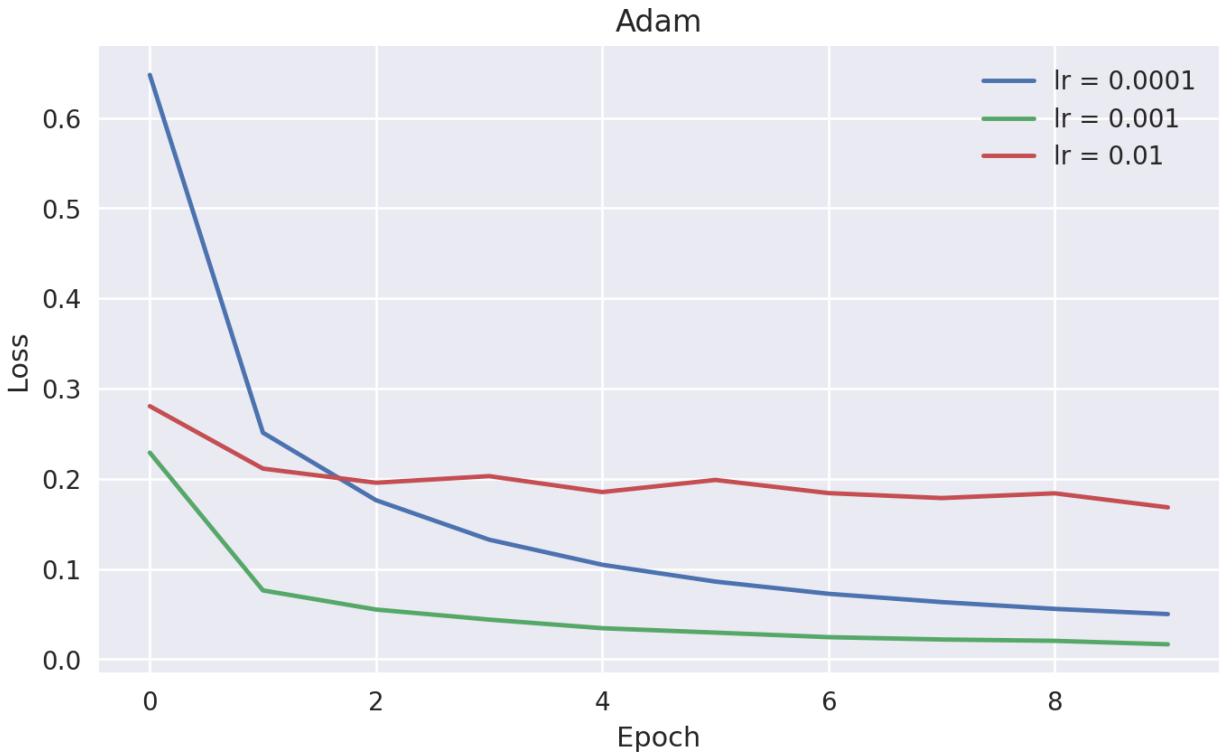

Figure 1: Depth = 30, $\alpha = 1$, learning rate = 0.001