Homework 10

Due: Thursday, April 10, 2025 at 12:00pm **(Noon)**

Problem 1

(16 points)
Consider a 2-layer neural network h which takes inputs $\mathbf{x} \in \mathbb{R}^d$, has a hidden layer of size m, and produces scalar outputs. The network has 4 parameters:

- $W_{0,1} \in \mathbb{R}^{m,d}$, representing the weights between layers 0 and 1,
- $\mathbf{b}_1 \in \mathbb{R}^m$, representing the first layer's bias,
- $W_{1,2} \in \mathbb{R}^{1,m}$, representing the weights between layers 1 and 2,
- and $b_2 \in \mathbb{R}$, representing the second layer's bias.

For an input $\mathbf{x}$, the output of the first layer of the network is

$$\mathbf{a}_1 = W_{0,1} \cdot \mathbf{x} + \mathbf{b}_1, \qquad \mathbf{o}_1 = \sigma(\mathbf{a}_1)$$

where σ is our choice of activation function. For this problem, use the sigmoid function.[1]

$$\sigma := \sigma_{\text{sigmoid}}(a) = \frac{1}{1+e^{-a}}$$

Then, the output of the second layer is

$$o_2 = W_{1,2} \cdot \mathbf{o}_1 + b_2,$$

As usual, we'll take our loss function L to be the squared error loss, given by

$$L(o_2; y) = (o_2 - y)^2$$

where $y \in \mathbb{R}$ is a real-valued label and o_2 is the network's output (in other words, this is $L(h(\mathbf{x}); y)$).

In this problem, you're asked to calculate partial derivatives of L with respect to each of the network's parameters. Note that $(W_{a,a+1})_{ij}$ is the entry in $W_{a,a+1}$ at row i and column j; similarly, $(\mathbf{b}_1)_i$ is the ith coordinate of $\mathbf{b}_1$. (Note that $1 \le i \le m$ and $1 \le j \le d$.)
(*Hint*: Make use of the chain rule.)

a. Calculate $\dfrac{\partial L}{\partial b_2}$. Show your work.

b. Calculate $\dfrac{\partial L}{\partial (W_{1,2})_{1i}}$. Show your work.

c. Calculate $\dfrac{\partial L}{\partial (\mathbf{b}_1)_i}$. Show your work.

d. Calculate $\dfrac{\partial L}{\partial (W_{0,1})_{ij}}$. Show your work.

[1] Notice that σ is a map $\mathbb{R} \to \mathbb{R}$, so applying it to vectors doesn't make much sense. Here, we really mean to apply the activation function coordinate-wise; this is a common shorthand. We could say equivalently $\mathbf{h}_i = \sigma((W\mathbf{x} + \mathbf{b}_1)_i)$.

Solution:

a. This is straightforward since we are only dealing with reals. Observe that

$$\frac{\partial L(o_2)}{\partial b_2} = \frac{\partial L(o_2)}{\partial o_2} \cdot \frac{\partial o_2}{\partial b_2} \tag{1}$$

$$= [2(o_2 - y)] \cdot \left[\frac{\partial (W_{1,2} \cdot \mathbf{o}_1) + b_2}{\partial b_2}\right] \tag{2}$$

$$= 2(o_2 - y) \cdot \frac{\partial b_2}{\partial b_2} \tag{3}$$

$$= 2(o_2 - y) \tag{4}$$

b. Recall that $W_{1,2}$ is functionally a column vector and $\mathbf{o}_1$ is a row vector. Now,

$$\frac{\partial L(o_2)}{\partial (W_{1,2})_{1i}} = \frac{\partial L(o_2)}{\partial o_2} \cdot \frac{\partial o_2}{\partial (W_{1,2})_{1i}} \tag{5}$$

$$= 2(o_2 - y) \cdot \frac{\partial}{\partial (W_{1,2})_{1i}} \langle W_{1,2}, \mathbf{o}_1 \rangle \tag{6}$$

$$= 2(o_2 - y) \left[\frac{\partial}{\partial (W_{1,2})_{1i}} \sum_i (W_{1,2})_{1,i} (\mathbf{o}_1)_i\right] \tag{7}$$

$$= 2(o_2 - y)(\mathbf{o}_1)_i \tag{8}$$

c. This one is a little longer. Consider the following decomposition by the chain rule.

$$\frac{\partial L(o_2)}{\partial (\mathbf{b}_1)_i} = \frac{\partial L(o_2)}{\partial o_2} \cdot \frac{\partial o_2}{\partial (\mathbf{o}_1)_i} \cdot \frac{\partial (\mathbf{o}_1)_i}{\partial (\mathbf{a}_1)_i} \cdot \frac{\partial (\mathbf{a}_1)_i}{\partial (\mathbf{b}_1)_i} \tag{9}$$

$$= 2(o_2 - y) \cdot \frac{\partial}{\partial (\mathbf{o}_1)_i} \langle W_{1,2}, \mathbf{o}_1 \rangle \cdot \frac{\partial \sigma(\mathbf{a}_1)_i}{\partial (\mathbf{a}_1)_i} \cdot \frac{\partial (\mathbf{a}_1)_i}{\partial (\mathbf{b}_1)_i} \tag{10}$$

$$= 2(o_2 - y) \cdot (W_{1,2})_i \cdot \sigma(\mathbf{a}_1)_i \cdot (1 - \sigma(\mathbf{a}_1)_i) \cdot \frac{\partial}{\partial (\mathbf{b}_1)_i} (W_{0,1} \cdot \mathbf{x} + \mathbf{b}_1) \tag{11}$$

$$= 2(o_2 - y) \cdot (W_{1,2})_i \cdot \sigma(\mathbf{a}_1)_i \cdot (1 - \sigma(\mathbf{a}_1)_i) \tag{12}$$

where we use the fact that $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$ in (11).

d. From (c), we have

$$\frac{\partial L(o_2)}{\partial (\mathbf{b}_1)_i} = \frac{\partial L(o_2)}{\partial (\mathbf{a}_1)_i} \cdot \frac{\partial (\mathbf{a}_1)_i}{\partial (\mathbf{b}_1)_i} = \frac{\partial L(o_2)}{\partial (\mathbf{a}_1)_i}$$

because the second term evaluates to 1. This gives

$$\frac{\partial L(o_2)}{\partial (\mathbf{a}_1)_i} = 2(o_2 - y) \cdot (W_{1,2})_i \cdot \sigma(\mathbf{a}_1)_i \cdot (1 - \sigma(\mathbf{a}_1)_i) \tag{13}$$

We also have

$$\frac{\partial (\mathbf{a}_1)_i}{\partial (W_{0,1})_{ij}} = \frac{\partial}{\partial (W_{0,1})_{ij}} \left(\langle (W_{0,1})_{ij}, \mathbf{x} \rangle + (\mathbf{b}_1)_i\right) \tag{14}$$

$$= \mathbf{x}_j \tag{15}$$

Putting everything together using the chain rule,

$$\frac{\partial L(o_2)}{\partial (W_{0,1})_{ij}} = \frac{\partial L(o_2)}{\partial (\mathbf{a}_1)_i} \cdot \frac{\partial (\mathbf{a}_1)_i}{\partial (W_{0,1})_{ij}} \tag{16}$$

$$= 2(o_2 - y) \cdot (W_{1,2})_i \cdot \sigma(\mathbf{a}_1)_i \cdot (1 - \sigma(\mathbf{a}_1)_i) \cdot \mathbf{x}_j \tag{17}$$

Programming Assignment

Introduction

For this assignment, you'll be implementing a single-layer and a double-layer feed forward neural network using stochastic gradient descent (SGD). Then, you'll compare the performance of your models on the UCI Wine Dataset, which you used in HW2. We're interested in predicting the quality of a wine (scored out of 10) given various attributes of the wine (for example, acidity, alcohol content). See Chapter 20 in the textbook for more detail.

Neural Networks

To make the parameterization a little cleaner, we'll write h_W to denote the output of our neural network, where W generically denotes all the weights and biases (if any) in that neural network. Recall that we're using squared error loss, which can be more generally given as

$$L(h_W(\mathbf{x}); \mathbf{y}) = \|h_W(\mathbf{x}) - \mathbf{y}\|_2^2 = \sum_{i=1}^{m} (h_W(\mathbf{x}_i) - \mathbf{y}_i)^2$$

where $h_W(\mathbf{x}_i)$ is the predicted value for the ith example and y_i is a corresponding label. Our goal is again to find the set of weights and biases that minimizes the loss function; you'll be using SGD to find this argmin in both cases.

Your task is to implement `OneLayerNN` and `TwoLayerNN` in `models.py`:

- `OneLayerNN`: A single-layer neural network that is equivalent to linear regression:

  $$h_W(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

  where $\mathbf{w}$ is the weight vector. For `OneLayerNN` only, we are assuming that the last feature of each example is 1, so no separate bias term is needed.

  Recall that we do not apply an activation function to the output of our final output layer.[2] Using the squared loss, the ERM hypothesis will have weights

  $$\mathbf{w} = \arg\min_{\mathbf{w}} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

  Even though we know how to solve this optimization in closed form, in this assignment we will use stochastic gradient descent.

- `TwoLayerNN`: For this model, you'll need to implement a fully connected hidden layer and backpropagation.

  For an input $\mathbf{x}$, the first layer of the network is computed by

  $$\mathbf{a}_1 = W_{0,1} \cdot \mathbf{x} + \mathbf{b}_1$$

  The activation function will be applied on the output of the first layer to generate the output

  $$\mathbf{o}_1 = \sigma(\mathbf{a}_1)$$

[2] or, rather, we can say that the output layer's activation function is the identity function.

and the output of the second layer is

$$o_2 = W_{1,2} \cdot \mathbf{o}_1 + b_2$$

In your implementation, you will take in the activation function $\sigma(a)$ as a parameter to `TwoLayerNN`. You will need to pass in the derivative of the activation function $\sigma'(a)$ for training. Doing so will ease swapping between different choices of activation functions, which can be explored for extra credit as a part of the Written Report. However, to complete the assignment, the sigmoid activation function is sufficient.

$$\sigma := \sigma_{\text{sigmoid}}(a) = \frac{1}{1 + e^{-a}}$$

Training Your Neural Networks

The objective of training a neural network is to find a set of weights and biases that minimize our model's training loss. If these weights are all initialized to the same constant value, they will all learn the same features. To avoid this, we should initialize our weights randomly. In particular, `np.random.uniform` may be useful.

When training the neural networks, first compute the output by calling `forward_pass` on an example. Then update the weights (and any biases) by calling `backward_pass`. We will train on one example at a time (equivalent to a batch size of 1).

We also expect for `TwoLayerNN` that you implement backpropagation as outlined in lecture 18 i.e. computing all the outputs in the forward pass and saving them for use in the backward pass so that backpropagation achieves $\mathcal{O}(|E|)$ complexity, where E is the network's edge set. You must implement backpropagation in the `backprop` method.

Computing Gradients

Please refer to the slides of the Backpropagation lecture for the definitions of the gradient computations to use in backpropagation.

Important Note: External libraries that trivialize the implementation are prohibited. More specifically, you may not use `numpy.linalg.lstsqnp` or similar functions in your implementation, and, of course, you may not use any neural network libraries/ deep learning frameworks. This should be completed using only Python and Numpy, as given in the stencil.

Stencil Code & Data

You can find and download the assignment here: HW10 on Github. If you have any problems, please consult the Download and Submission Guide.
We have provided the following stencil code:

- `main.py` is the entry point of program which will read in the dataset, run the models and print the results.
- `models.py` contains the `OneLayerNN` model and the `TwoLayerNN` model which you will be implementing.

You should not need to add any code to `main.py`. If you do for debugging or other purposes, please make sure all of your additions are commented out in the final handin. All the functions you need to fill in reside in `models.py`, marked by `TODO`s. To run the program, run `python main.py` in a terminal with the course environment set up.

We have provided unit tests for the functions that compute gradients for the 2-layer neural network. To enable these unit tests, uncomment `test_gradients` in the `main` function of `main.py`.

Your program assumes the data is formatted as follows: The first column of data in each file is the labels y and all other columns are the input features $(x_1, x_2, \ldots, x_n)$. We have taken care of all data preprocessing, as usual. If you're curious and would like to read about the dataset, you can find more information **here**, but it is strongly recommended that you use the versions that we've provided in the course directory to maintain consistent formatting.

Written Report

Guiding Questions

- Compare the average loss of the two models. Provide an explanation for what you observe. (4 points)
- Comment on each of your parameter choices (the learning rate, the hidden layer size and the number of epochs for training). (4 points)
- Among machine learning techniques, neural networks have a reputation for being 'opaque boxes', where the logic of their decision making is difficult or impossible for humans to interpret. (4 points)

  a) If an 'opaque box' model gives an answer that disagrees with a human expert, which answer should be believed? Are there circumstances where we should believe one party more often than the other?

  b) Companies often hide the logic behind their products by making their software closed-source. Are there differences between companies selling closed source software, where the logic is known but hidden, versus companies selling 'black box' artificial intelligence software, where the logic might be altogether unknown? Is using one type of software more justifiable than using the other?

  Please credit any outside sources you use in your answer.

- (Extra Credit) Train your neural network using the ReLU activation function provided on Slide 15 from Lecture 18. Comment on the results.
- (Extra Credit) Construct a fake dataset that has really low training error on the 2-layer neural network and high training error the 1-layer neural network. In a file `generate.py`, you should write a function `generate_data` that returns arrays X and Y that can be passed to the train functions of either model. Make sure to use these exact file/function names because the autograder will look for these in your submission. And make sure the function returns `X, Y = generate_data()`. Comment on your approach and the results in your report.

Grading

Loss Targets

We are expecting the following training losses for each of the models on the Wine dataset:

- `OneLayerNN`: < 0.55
- `TwoLayerNN`: < 0.50

As always, we will be grading your code based on correctness and not based on whether or not you meet these targets. We will also be testing your NN on two hidden toy datasets.

Hyperparameters

To verify the correctness of your implementation, check that your model satisfies the above training loss benchmarks using the given hyperparameters. After your model has satisfied these benchmarks, feel free to change the values of the hyperparameters. We will use the hyperparameter values that you choose when testing your model on the wine datasets. However, we also test your model on synthetic datasets with our own hyperparameter values, and so we strongly suggest that you first verify the correctness of your implementation before modifying hyperparameters.

Breakdown

Written Question	16%
L2 Loss & Sigmoid Derivative	4%
1-Layer Neural Network	20%
2-Layer Neural Network	48%
Report	12%
Total	100%

Handing In

Programming Assignment

You will hand in both the written assignment and the coding portion on gradescope, separately.

1. Your written assignment should be uploaded to gradescope under "Homework 10".
2. Submit your hw10 github repo containing all your source code and your project report named **report.pdf** on gradescope under "Homework 10 Code". **report.pdf** should live in the root directory of your code folder; the autograder will check for the existence of this file and inform you if it is not found. For questions, please consult the download/submission guide.

Anonymous Grading

You need to be graded anonymously, so do not write your name anywhere on your handin.

Obligatory Note on Academic Integrity

Plagiarism—don't do it.

As outlined in the Brown Academic Code, attempting to pass off another's work as your own can result in failing the assignment, failing this course, or even dismissal or expulsion from Brown. More than that, you will be missing out on the goal of your education, which is the cultivation of your own mind, thoughts, and abilities. Please review this course's collaboration policy and, if you have any questions, please contact a member of the course staff.