Homework 2
600.482/682 Deep Learning
Fall 2023

Due Wednesday Sep 20 11:59 pm EST

Submission Instructions. Please submit the following **two items** to Gradescope (entry code BBVDNN):

1. Your report (LaTeX generated PDF) to `homework2-report`
2. A Zip file containing: (1) your Python Jupyter Notebook (.ipynb) and (2) an exported PDF of the notebook (with all cell outputs)* to `homework2-notebook`

   * On Google Colab, you may use "File" - "Print" - Save as PDF.

   * On Visual Studio, you may use "Jupyter: Export to PDF" in the Command palette.

Problems

1. The goal of this problem is to minimize a function given a certain input using gradient descent by breaking down the overall function into smaller components via a computational graph. The function is defined as:

$$f(x_1, x_2, w_1, w_2) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2)}} + 0.5(w_1^2 + w_2^2)\,.$$

(a) Please calculate $\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}$. You may use sigmoid function $\sigma(x)$ in your expression.

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial w_1}\frac{1}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}} + \frac{\partial f}{\partial w_1}0.5(\omega_1^2+\omega_2^2)$$

$$= \frac{\partial}{\partial j}\frac{1}{1+e^{-j}} + \frac{\partial}{\partial \omega_1}0.5\omega_1^2 + \frac{\partial}{\partial \omega_1}0.5\omega_2^2$$

$$= \frac{e^{-j}}{(1+e^{-j})^2}\frac{\partial j}{\partial \omega_1} + \omega_1$$

Where the derivative of j is calculated as $\frac{\partial j}{\partial \omega_1}(\omega_1 x_1 + \omega_2 x_2) = x_1$

$$\frac{\partial f}{\partial \omega_1} = \frac{e^{-(\omega_1 x_1+\omega_2 x_2)}}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}}x_1 + \omega_1 \tag{1}$$

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial w_2}\frac{1}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}} + \frac{\partial f}{\partial w_2}0.5(\omega_1^2+\omega_2^2)$$

$$= \frac{\partial}{\partial j}\frac{1}{1+e^{-j}} + \frac{\partial}{\partial \omega_1}0.5\omega_1^2 + \frac{\partial}{\partial \omega_1}0.5\omega_2^2$$

$$= \frac{e^{-j}}{(1+e^{-j})^2}\frac{\partial j}{\partial \omega_2} + \omega_2$$

Where the derivative of j is calculated as $\frac{\partial j}{\partial \omega_2}(\omega_1 x_1 + \omega_2 x_2) = x_2$

$$\frac{\partial f}{\partial \omega_2} = \frac{e^{-(\omega_1 x_1+\omega_2 x_2)}}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}}x_2 + \omega_2 \tag{2}$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_1}\frac{1}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}} + \frac{\partial f}{\partial x_1}0.5(\omega_1^2+\omega_2^2)$$

$$= \frac{\partial}{\partial j}\frac{1}{1+e^{-j}} + \frac{\partial}{\partial x_1}0.5\omega_1^2 + \frac{\partial}{\partial x_1}0.5\omega_2^2$$

$$= \frac{e^{-j}}{(1+e^{-j})^2}\frac{\partial j}{\partial x_1} + \omega_2$$

Where the derivative of j is calculated as $\frac{\partial j}{\partial x_1}(\omega_1 x_1 + \omega_2 x_2) = 0$

$$\frac{\partial f}{\partial x_1} = \frac{e^{-(\omega_1 x_1 + \omega_2 x_2)}}{1 + e^{-(\omega_1 x_1 + \omega_2 x_2)}}\omega_1 \tag{3}$$

$$\begin{aligned}
\frac{\partial f}{\partial x_2} &= \frac{\partial f}{\partial x_2}\frac{1}{1+e^{-(\omega_1 x_1+\omega_2 x_2)}} + \frac{\partial f}{\partial x_1}0.5(\omega_1^2+\omega_2^2)\\
&= \frac{\partial}{\partial j}\frac{1}{1+e^{-j}} + \frac{\partial}{\partial x_2}0.5\omega_1^2 + \frac{\partial}{\partial x_2}0.5\omega_2^2\\
&= \frac{e^{-j}}{(1+e^{-j})^2}\frac{\partial j}{\partial x_2} + \omega_2
\end{aligned}$$

Where the derivative of j is calculated as $\frac{\partial j}{\partial x_2}(\omega_1 x_1 + \omega_2 x_2) = 0$

$$\frac{\partial f}{\partial x_2} = \frac{e^{-(\omega_1 x_1 + \omega_2 x_2)}}{1 + e^{-(\omega_1 x_1 + \omega_2 x_2)}}\omega_2 \tag{4}$$

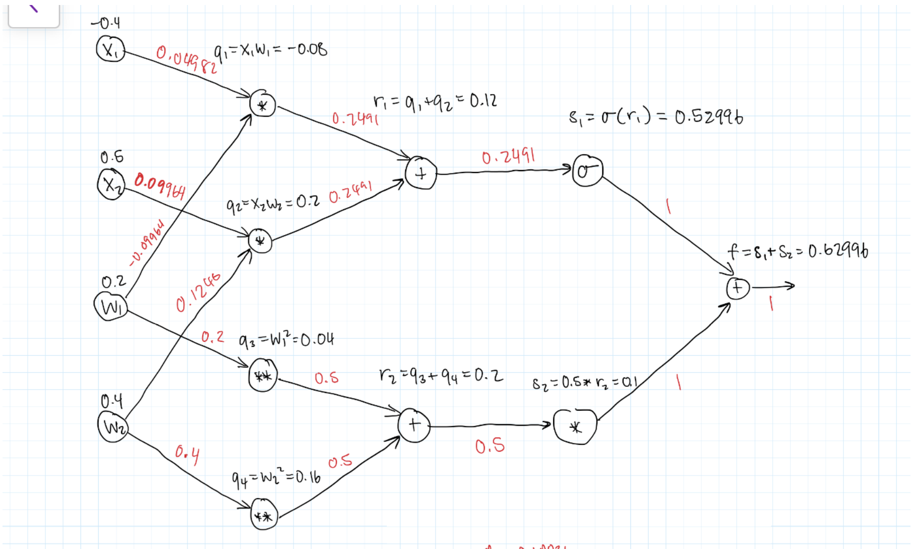
(b) Start with the following initialization: $w_1 = 0.2, w_2 = 0.4, x_1 = -0.4, x_2 = 0.5$, draw the computational graph. Please use backpropagation as we did in class. You may use sigmoid function as a node in the graph.
You can draw the graph on paper and insert a photo into your report.
The goal is for you to practice working with computational graphs. You must include the intermediate values in both forward and backward pass in your graph.

$$\frac{\partial f}{\partial \omega_1} = \frac{e^{-((0.2)(-0.4)+(0.4)(0.5))}}{1+e^{-((0.2)(-0.4)+(0.4)(0.5))}}(-0.4) + 0.2 = 0.10035$$

$$\frac{\partial f}{\partial \omega_2} = \frac{e^{-((0.2)(-0.4)+(0.4)(0.5))}}{1+e^{-((0.2)(-0.4)+(0.4)(0.5))}}(0.5) + 0.4 = 0.52455$$

$$\frac{\partial f}{\partial x_1} = \frac{e^{-((0.2)(-0.4)+(0.4)(0.5))}}{1+e^{-((0.2)(-0.4)+(0.4)(0.5))}}(0.2) = 0.0498$$

$$\frac{\partial f}{\partial x_2} = \frac{e^{-((0.2)(-0.4)+(0.4)(0.5))}}{1+e^{-((0.2)(-0.4)+(0.4)(0.5))}}(0.4) = 0.0996$$



$$\frac{df}{dx_1} = \frac{df}{ds_1}\frac{ds_1}{dr_1}\frac{dr_1}{dq_1}\frac{dq_1}{dx_1}$$

$$\frac{df}{dx_2} = \frac{df}{ds_1}\frac{ds_1}{dr_1}\frac{dr_1}{dq_2}\frac{dq_2}{dx_2}$$

$$\frac{df}{dw_1} = \frac{df}{ds_2}\frac{ds_2}{dr_2}\frac{dr_2}{dq_3}\frac{dq_3}{dw_1} + \frac{df}{ds_2}\frac{ds_1}{dr_1}\frac{dr_1}{dq_1}\frac{dq_1}{dw_1}$$

$$\frac{df}{dw_2} = \frac{df}{ds_2}\frac{ds_2}{dr_2}\frac{dr_2}{dq_4}\frac{dq_4}{dw_1} + \frac{df}{ds_2}\frac{ds_1}{dr_1}\frac{dr_1}{dq_2}\frac{dq_2}{dw_2}$$

w_1.grad = 0.10036
w_2.grad = 0.5248
x_1.grad = 0.0498
x_2.grad = 0.0996

(c) John would like to train a binary classification model that classifies whether a particular fruit is an apple or not based on two features: its color (x_1) and its roundness (x_2). w_1 and w_2 are the weights he gives to the two features. John tries to use the function $f(x_1, x_2, w_1, w_2)$ as the loss function for a single data point in his optimization problem. Will this formulation of loss function work? Briefly explain why or why not.

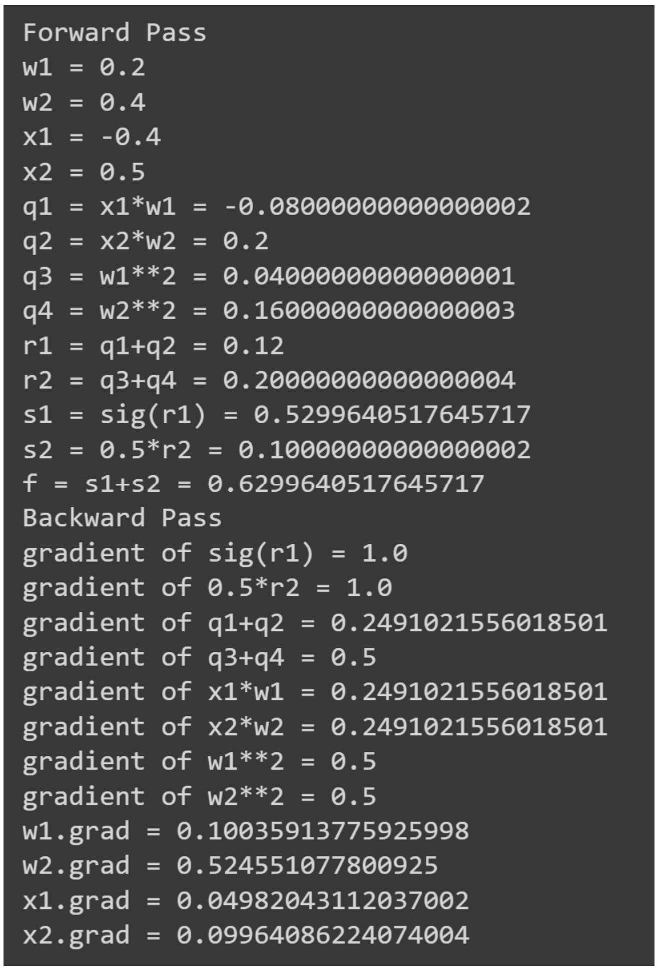
This formulation of a loss function will not work very well. This is a binary classification which means it either is the object or it isn't. We want a loss function that will generate probabilities between 0 and 1. The function $f(x_1, x_2, \omega_1, \omega_2)$ will not always provide probabilities between those two values. These input features are also too broad and the function does not account for the class separation between an apple and a non-apple.

2. Doing the computations above is a lot of work, already for such a simple function as in Problem 1. Clearly, it will be desirable to automate the forward and backward propagation process. In contemporary frameworks, such as TensorFlow and PyTorch, backpropagation is handled automatically for all standard operations, which is clearly very convenient. This feature is commonly referred to as “AutoGrad” and becomes possible due to one of the key observations we learned in class: difficult expressions can be broken down into simple sub-problems, the analytic gradients of which can be synthesized via chain rule. In this problem, you will implement your own AudoGrad structure.

   (a) Consider the Jupyter Notebook we provide (Homework2.ipynb) and follow the instructions therein to implement the missing operations and functionality. (To use Jupyter Notebook, we highly recommend uploading the .ipynb file to Google Colaboratory (`https://colab.research.google.com/`), where you may edit and run your notebook online. If that option is not feasible for you, you can create a local Python 3.6+ environment. The only required external dependencies are `numpy`, `matplotlib`, and `jupyter` which you can install using the package manager of your choosing (e.g. `pip`, `conda`))

   (b) Use the function and output derived in Problem 1 to test whether your AutoGrad structure works as intended.

   Attach a screenshot of your notebook’s printed intermediate values for this test case (both during the forward and backward pass) to your report.



```
Forward Pass
w1 = 0.2
w2 = 0.4
x1 = -0.4
x2 = 0.5
q1 = x1*w1 = -0.08000000000000002
q2 = x2*w2 = 0.2
q3 = w1**2 = 0.04000000000000001
q4 = w2**2 = 0.16000000000000003
r1 = q1+q2 = 0.12
r2 = q3+q4 = 0.20000000000000004
s1 = sig(r1) = 0.5299640517645717
s2 = 0.5*r2 = 0.10000000000000002
f = s1+s2 = 0.6299640517645717
Backward Pass
gradient of sig(r1) = 1.0
gradient of 0.5*r2 = 1.0
gradient of q1+q2 = 0.2491021556018501
gradient of q3+q4 = 0.5
gradient of x1*w1 = 0.2491021556018501
gradient of x2*w2 = 0.2491021556018501
gradient of w1**2 = 0.5
gradient of w2**2 = 0.5
w1.grad = 0.10035913775925998
w2.grad = 0.524551077800925
x1.grad = 0.04982043112037002
x2.grad = 0.09964086224074004
```

3. Your goal in this exercise is to implement a linear classifier using the AutoGrad class you implemented in Problem 2, train it on a 2-dimensional toy dataset, and show the results tested on the **same** dataset. Recall that we formulate a linear classifier as $f(x, W, b) = Wx + b$. To quantify the model’s performance, you will pass its output through a softmax function and then use Cross Entropy loss to measure agreement with the target output.

   Write a Python program based on your AutoGrad structure that iteratively computes a small step in the direction of negative gradient. The dataset (HW2_Q3_dataset) can be downloaded here (`https://piazza.com/class_profile/get_resource/lljvas0yc7034d/lm4r6jsufxw3j6`).

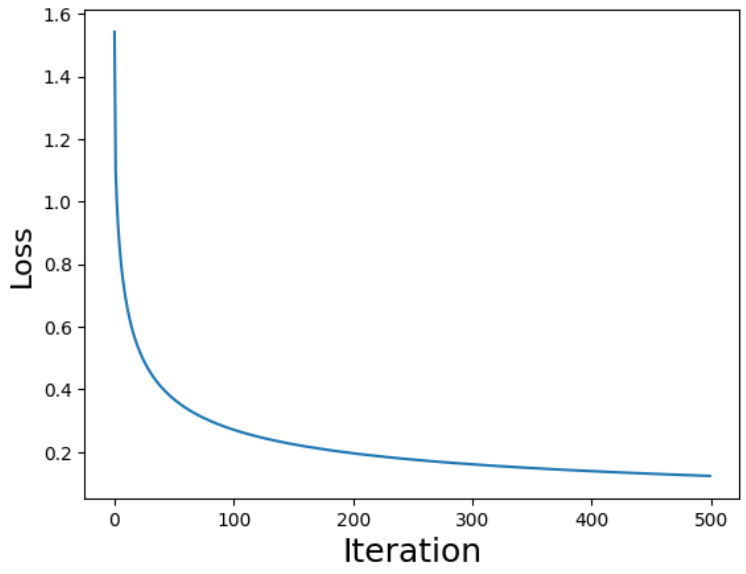
Please go through the README.txt carefully before you start. Initialize your optimization with all elements of W being small random numbers and $b = 0$. As shown in the visualization of the data provided in the link above, you can reasonably expect to see your algorithm converge to a solution that linearly separates the given data samples.

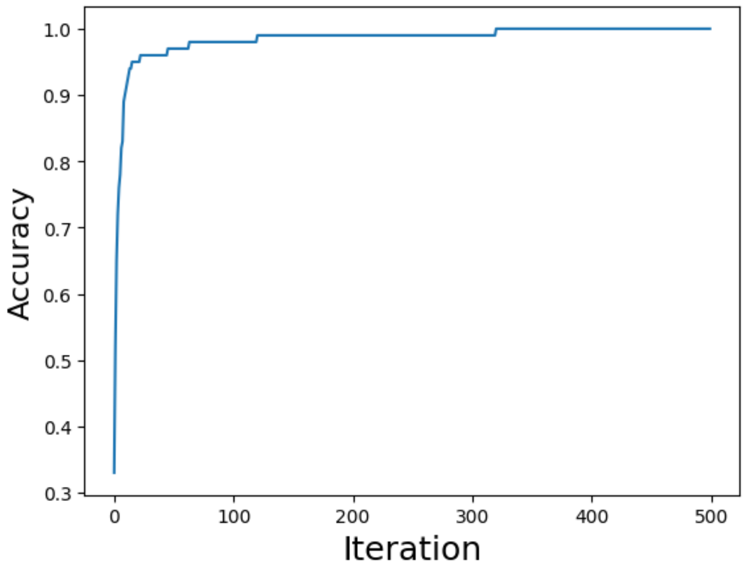
(a) Please plot the following and attach them in the report:

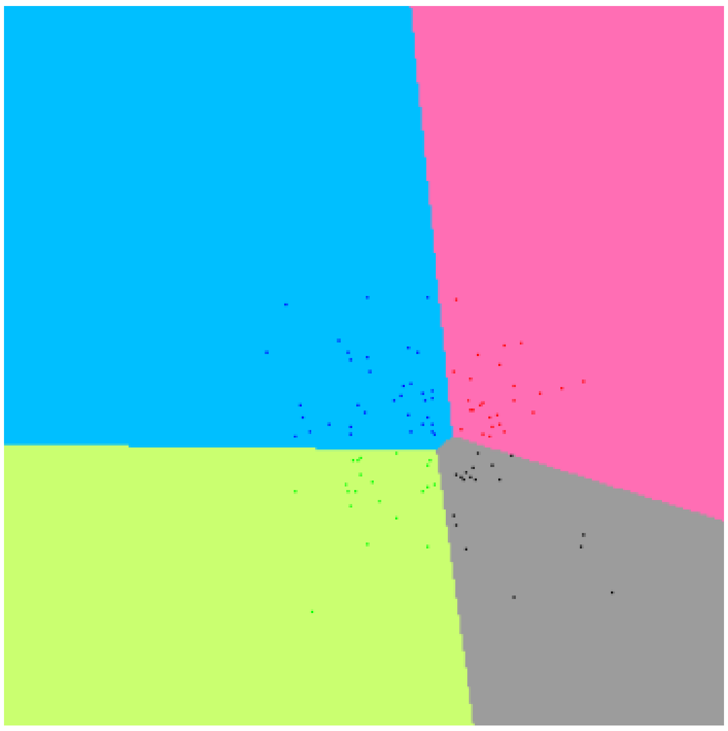
i. Cross Entropy loss with respect to training iterations (loss curve)

ii. Training accuracy with respect to training iterations (accuracy curve)

iii. Visualization of the ground truth labels and your model's decision boundaries (decision boundary visualization)







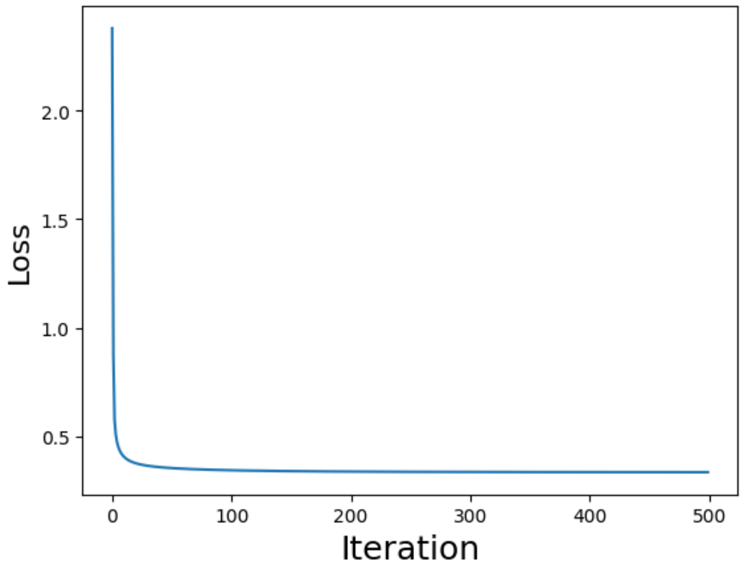
(b) Report your final classification accuracy. Briefly discuss your result based on the visualization of data distribution.

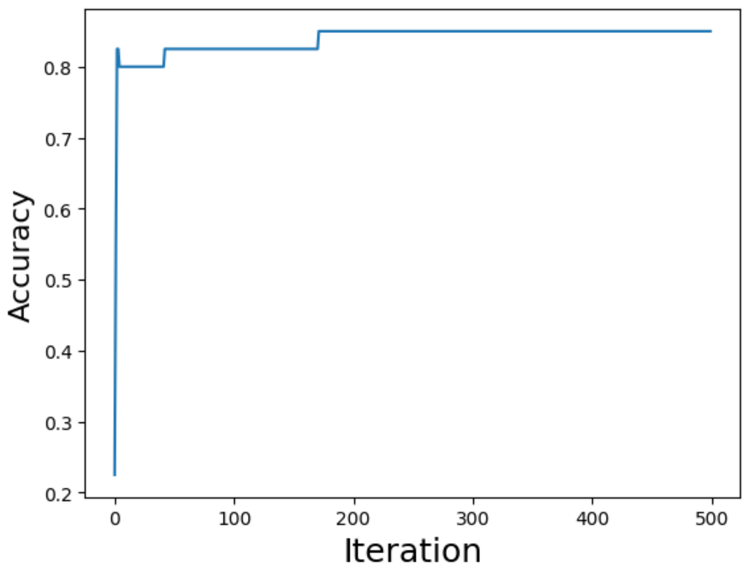
My final accuracy was 100%. This appears to be correct since the data points are in the right section based on their color. I had an issue where the data points were in different sectors and the color did not match the sector they were in but it ended up being an issue with my plot points and not the way I was training my data. If I didn't have 100% accuracy, I would expect to have blue data points in the pink sector, etc.

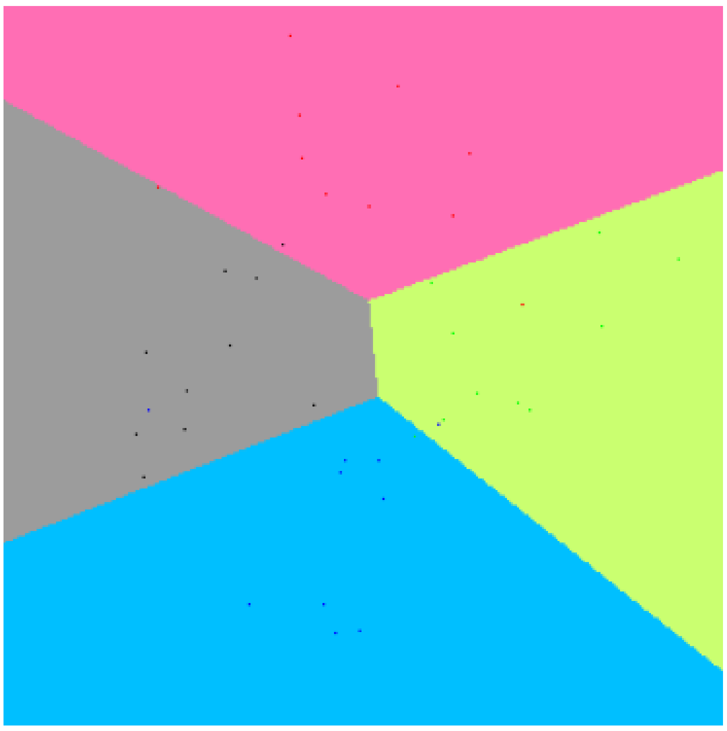
4. In this problem, we would like you to compare the performance of single-layer and multi-layer classifiers on a different dataset, again using your AutoGrad structure.

(a) Please use HW2_Q4_dataset for this question. This data is two dimensional. Create another linear model (same as Problem 3) for this dataset and run your classification training.

Report your final classification accuracy. As in Problem 3, please attach plots of the i) loss curve, ii) accuracy curve and iii) decision boundary visualization. Is this dataset linearly separable?



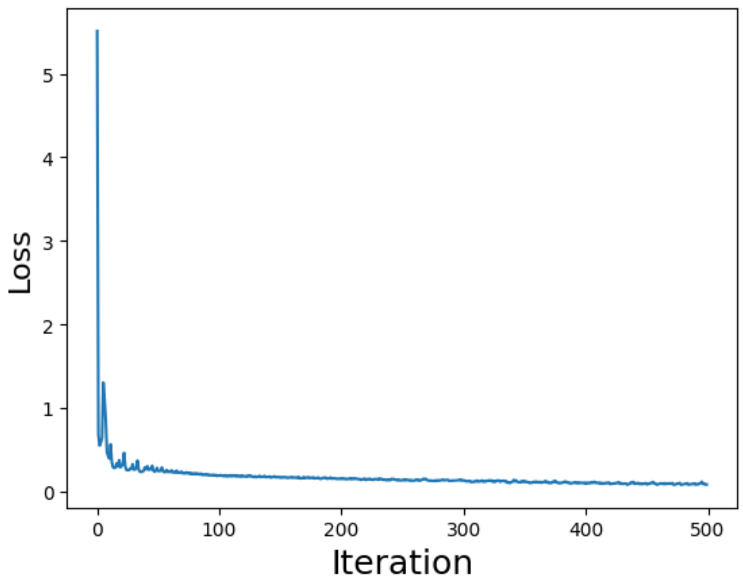


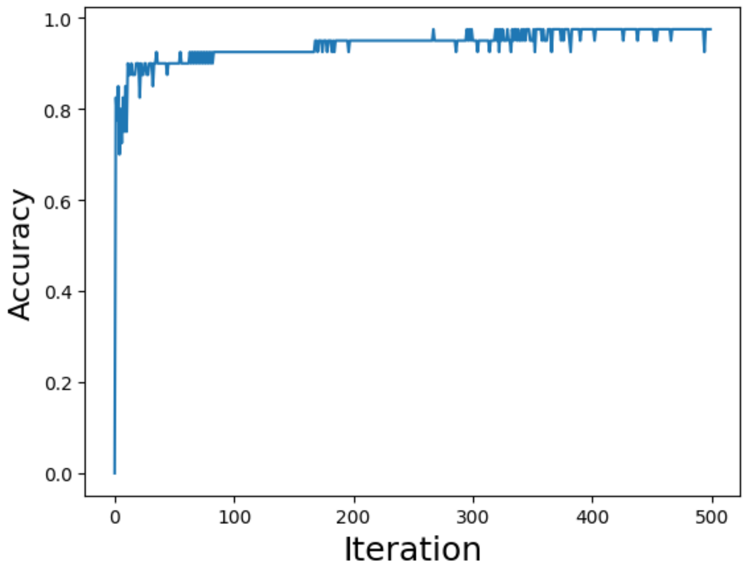


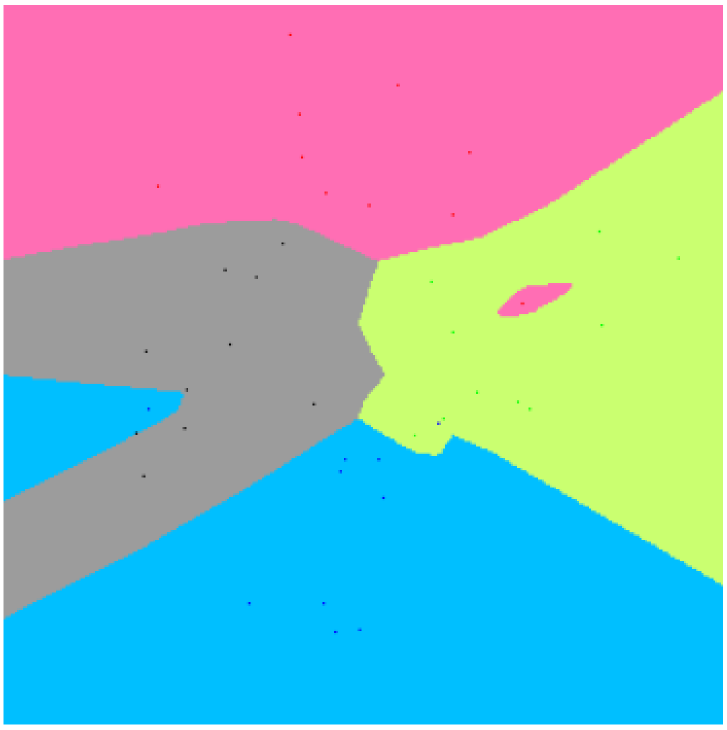
This data set is not linearly separable because it is not possible to draw a single line or a hyperplane through the picture and separate the data points into two classes.

(b) Now add one more layer of perceptrons.* Use ReLU as activation functions for the first hidden layer.** Connect outputs with a softmax function. Initialize your optimization procedure with all elements of W being small random numbers and $b = 0$. Use Cross Entropy loss. Train this model on the same two dimensional dataset in (a).

Report your classification accuracy. Attach plots of the i) loss curve, ii) accuracy curve and iii) decision boundary visualization in the report.


Loss
0
1
2
3
4
5
0
100
200
300
400
500
Iteration


Accuracy
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
500
Iteration



(c) Compare your results, including the classification accuracy and decision boundary plot, between 4(a) and 4(b). Briefly explain why introducing an additional layer causes these changes.

My accuracy for part b was only 85% and I had very clean lines for my decision boundary. Looking closely at the picture for part b, I can see that there are data points in a sector that don't match their color which means that the model isn't fitting the data correctly. Once I added another layer to the neural network, my accuracy went up to 97.5%. This accuracy is reflected in the decision boundary and now you can see that for the most part, the background matches where the data points are. I believe this is happening because, with the extra layer, the neural network is able to better fit the training data. The network may also be able to connect the data to the labels better with this additional layer.

* If the output of a single layer is $f(x, W_1, b_1)$, then adding another layer basically means that the output is $g(f(x, W_1, b_1), W_2, b_2)$: the second layer uses the first layer's output as its input. Therefore, the output dimension of f should match the expected input dimension of g. In our case, the output dimension from the ReLU activation in the first layer should match the input dimension of the second layer.

** The ReLU activation function is a nonlinear function defined as:

$$\text{ReLU}(x) = \max(0, x)$$

If we use ReLU as our activation function for the first layer, instead of $f(x, W_1, b_1) = W_1x + b$, we have

$$f(x, W_1, b_1) = \text{ReLU}(W_1x + b) = \max(0, W_1x + b)$$

The non-linearity of the ReLU function adds non-linearity to the model.