Jack Stevenson
CS 434H, Fall 2022

1. Prove Bernoulli Naïve Bayes has a linear decision boundary like Logistic Regression

$$\frac{P(y=1|x_1,\ldots,x_d)}{P(y=0|x_1,\ldots,x_d)} > 1$$

$$\frac{\theta_1 \Pi_{i=1}^d \theta_{i1}^{x_i}(1-\theta_{i1})^{1-x_i}}{\theta_0 \Pi_{i=1}^d \theta_{i0}^{x_i}(1-\theta_{i0})^{1-x_i}} > 1$$

$$log\ \frac{\theta_1 \Pi_{i=1}^d \theta_{i1}^{x_i}(1-\theta_{i1})^{1-x_i}}{\theta_0 \Pi_{i=1}^d \theta_{i0}^{x_i}(1-\theta_{i0})^{1-x_i}} > log(1)$$

$$log(\theta_1) - log(\theta_0) + log(\Pi_{i=1}^d \theta_{i1}^{x_i}(1-\theta_{i1})^{1-x_i}) - log(\Pi_{i=1}^d \theta_{i0}^{x_i}(1-\theta_{i0})^{1-x_i}) > 0$$

$$log\frac{\theta_1}{\theta_0} + log\frac{\Pi_{i=1}^d \theta_{i1}^{x_i}(1-\theta_{i1})^{1-x_i}}{\Pi_{i=1}^d \theta_{i0}^{x_i}(1-\theta_{i0})^{1-x_i}} > 0$$

Adding a value C to represent the product of the likelihoods of each $x_i = 0, 1$. There is probably a more correct way to approach this. We divide both sides by C to eliminate this value.

$$log\frac{\theta_1}{\theta_0} + C\ log\frac{\theta_1^{\sum x_i}(1-\theta_1)^{1-\sum x_i}}{\theta_0^{\sum x_i}(1-\theta_0)^{1-\sum x_i}} > 0$$

$$log\frac{\theta_1}{\theta_0} + log(\theta_1^{\sum x_i}) + log(1-\theta_1)^{1-\sum x_i} - log(\theta_0^{\sum x_i}) - log(1-\theta_0)^{1-\sum x_i} > 0$$

$$log\frac{\theta_1}{\theta_0} + (\sum x_i)log(\theta_1) + (1-\sum x_i)log(1-\theta_1) - (\sum x_i)log(\theta_0) - (1-\sum x_i)log(1-\theta_0) > 0$$

$$log\frac{\theta_1}{\theta_0} + (\sum x_i)(log(\theta_1) - log(\theta_0)) + (1-\sum x_i)(log(1-\theta_1) - log(1-\theta_0)) > 0$$

$$log\frac{\theta_1}{\theta_0} + (\sum x_i)[log(\theta_1) - log(\theta_0) - log(1-\theta_1) + log(1-\theta_0)] + [log(1-\theta_1) - log(1-\theta_0)] > 0$$

$$log\frac{\theta_1}{\theta_0} + \sum x_i log\frac{\theta_1(1-\theta_0)}{\theta_0(1-\theta_1)} + log\frac{1-\theta_1}{1-\theta_0} > 0$$

$$log\frac{\theta_1(1-\theta_1)}{\theta_0(1-\theta_0)} + \sum x_i log\frac{\theta_1(1-\theta_0)}{\theta_0(1-\theta_1)} > 0$$

$$b = log\frac{\theta_1(1-\theta_1)}{\theta_0(1-\theta_0)},\ w_i = log\frac{\theta_1(1-\theta_0)}{\theta_0(1-\theta_1)}$$

2. Differences between Bernoulli Naïve Bayes and Logistic Regression
   There are a greater number of restrictions on the weights and biases that can be used in Naive Bayes than there are on those used in Logistic Regression. As shown above, the bias and weight terms used must follow a more structured formula regarding the likelihood of different outcomes of classification.

3. Compute the derivative for Softmax(z)

$$Softmax(z) = p_i = \frac{e^{z_i}}{\sum_{k=1}^{d} e^{z_k}}$$

We are finding the partial derivative of the expression above with respect to z_i. We have two cases to analyze: one where $i = j$ and one where $i \neq j$. First, the case where $i = j$:

$$\frac{\partial p_i}{\partial z_i} = \frac{f'g - fg'}{g^2}$$

$$f = e^{z_i},\ g = \sum_{k=1}^{d} e^{z_k}$$

The derivative of f with respect to z_j is e^{z_i}. The derivative of g with respect to z_i is e^{z_i}, the only term containing z^i.

$$\frac{\partial p_i}{\partial z_i} = \frac{(e^{z_i} \sum_{k=1}^{d} e^{z_i}) - (e^{z_i} e^{z_i})}{(\sum_{k=1}^{d} e^{z_i})^2}$$

$$\frac{\partial p_i}{\partial z_i} = \frac{e^{z_i}(\sum_{k=1}^{d} e^{z_i} - e^{z_i})}{(\sum_{k=1}^{d} e^{z_i})^2}$$

$$\frac{\partial p_i}{\partial z_i} = \frac{e^{z_i}}{\sum_{k=1}^{d} e^{z_i}} \times \frac{\sum_{k=1}^{d} e^{z_i} - e^{z_i}}{\sum_{k=1}^{d} e^{z_i}}$$

$$i = j: \ \frac{\partial p_i}{\partial z_j} = \frac{e^{z_i}}{\sum_{k=1}^{d} e^{z_k}} \times (1 - \frac{e^{z_i}}{\sum_{k=1}^{d} e^{z_k}}) = Softmax(z_i) \times (1 - Softmax(z_i))$$

Next, the case where $i \neq j$:

$$\frac{\partial p_i}{\partial z_j} = \frac{f'g - fg'}{g^2}$$

$$f = e^{z_i},\ g = \sum_{k=1}^{d} e^{z_k}$$

The derivative of f with respect to z_j is zero. The derivative of g with respect to z_i is e^{z_i}, the only term containing z^i.

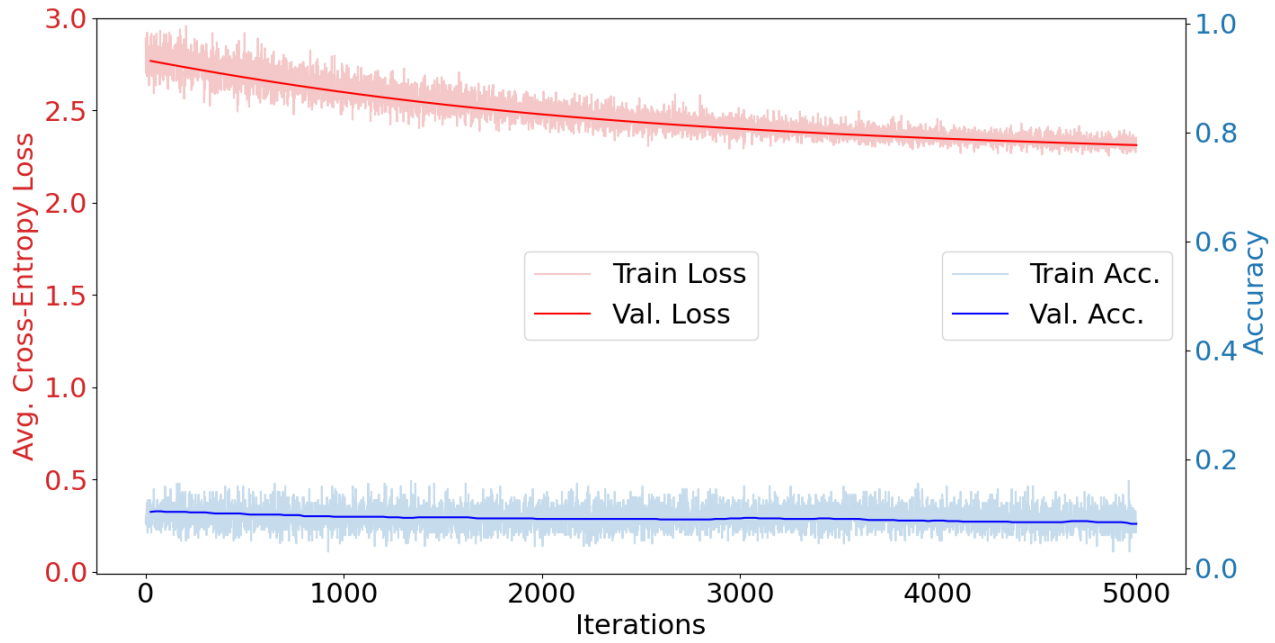
$$\frac{\partial p_i}{\partial z_j} = \frac{(0 \times \sum_{k=1}^{d} e^{z_i}) - (e^{z_i} e^{z_j})}{(\sum_{k=1}^{d} e^{z_i})^2}$$

$$\frac{\partial p_i}{\partial z_j} = \frac{-e^{z_i} \times e^{z_j}}{\sum_{k=1}^{d} e^{z_i} \times \sum_{k=1}^{d} e^{z_i}}$$

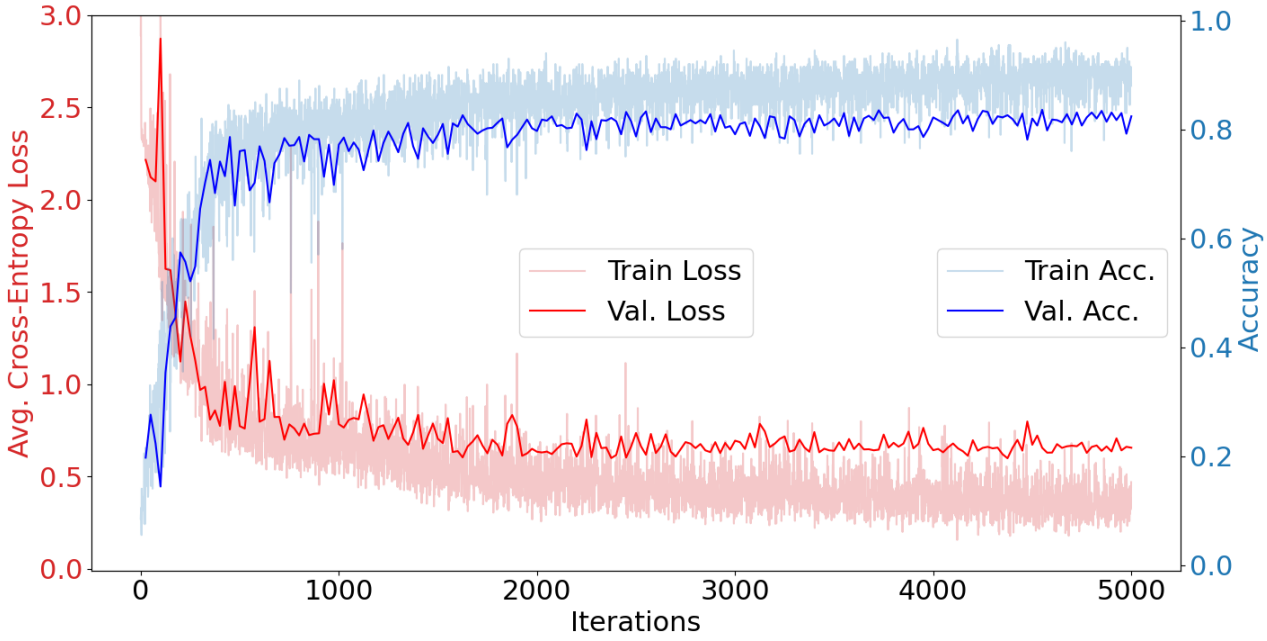
$$i \neq j: \frac{\partial p_i}{\partial z_j} = \frac{-e^{z_i}}{\sum_{k=1}^{d} e^{z_i}} \times \frac{e^{z_j}}{\sum_{k=1}^{d} e^{z_i}} = -Softmax(z_i) \times Softmax(z_j)$$

4. Learning Rate

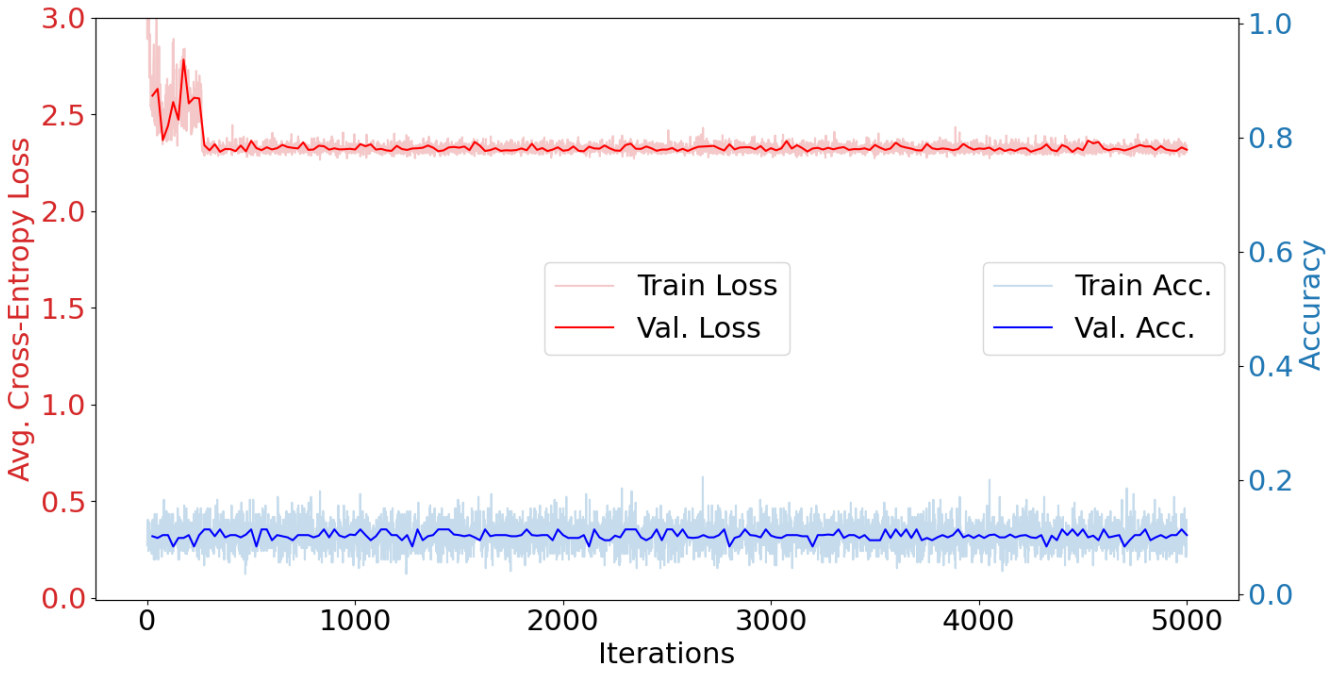
   With a small step size of 0.0001, the accuracy stayed below 10%. This is probably because it got stuck at a non-optimal local minimum. With a larger step size of 5, there was a lot more noise, but it reached around the same 90% accuracy as a step size of 0.01. A step size of 10 never achieved greater than 10% accuracy. This is probably because it was overcorrecting and was never able to find a true minimum. An accuracy of 10% is also no better than random for 10 digits, so 0.0001 and 10 produced very poor accuracies. If the maximum number of epochs was increased, I think a step size of 5 would find a better optimum, but 0.0001 and 10 would still produce mediocre results.



A step size of 0.0001



A step size of 5

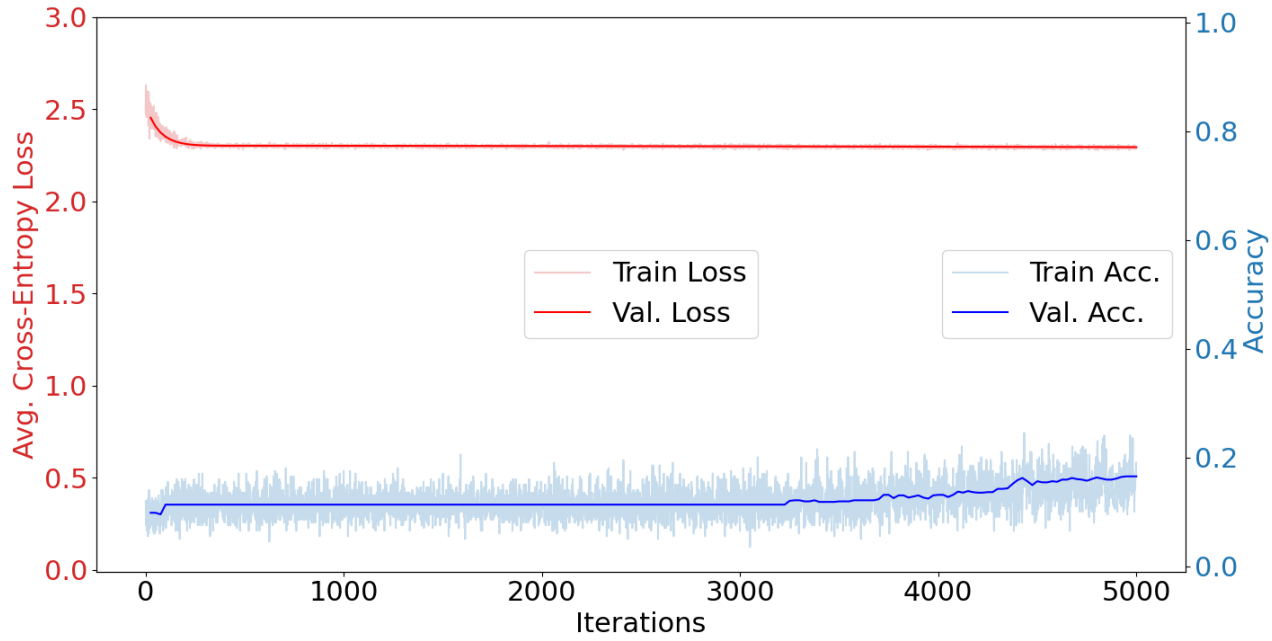


A step size of 10

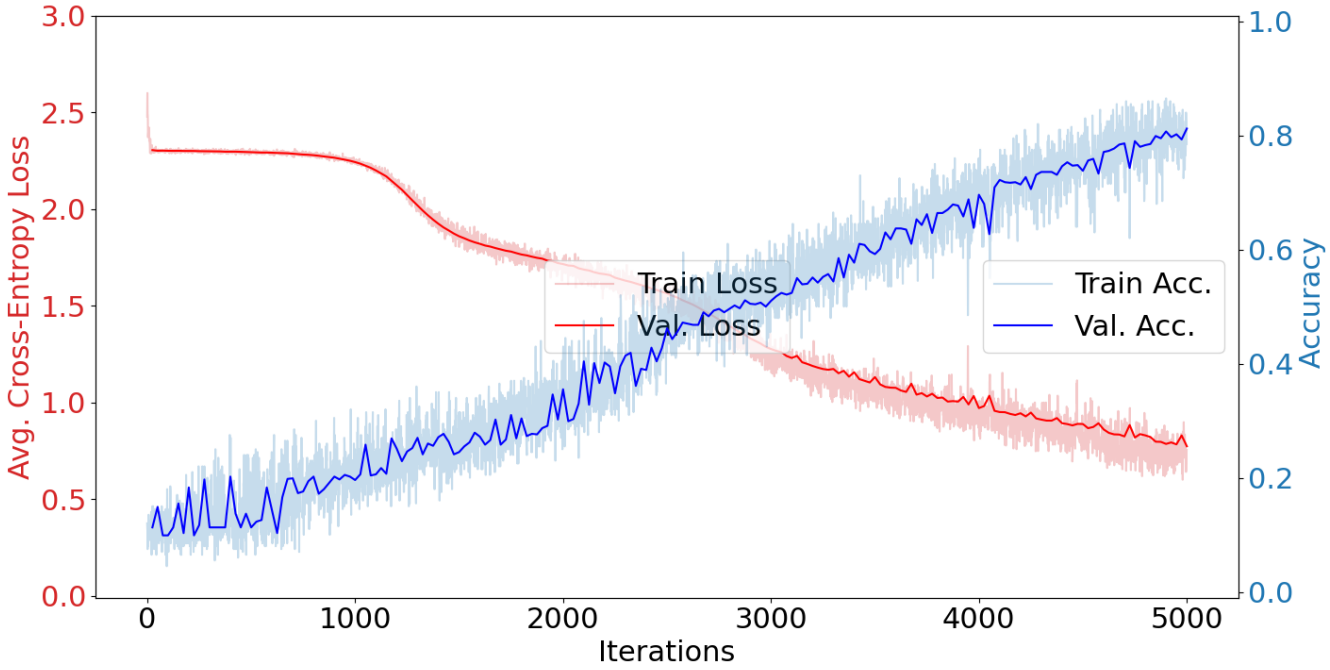
5. ReLU's and Vanishing Gradients

   For sigmoid activation with 5 layers and a step size of 0.01, the curve is much smoother and flatter than the one with the default parameters. It never reaches a satisfying result. It is interesting how validation accuracy seems to rise towards the end while validation loss remains constant. For sigmoid activation with 5 layers and a step size of 0.1, a satisfying accuracy is achieved. There seemed to be little to no difference between the train accuracy and the validation accuracy. It had a lot more noise than the previous

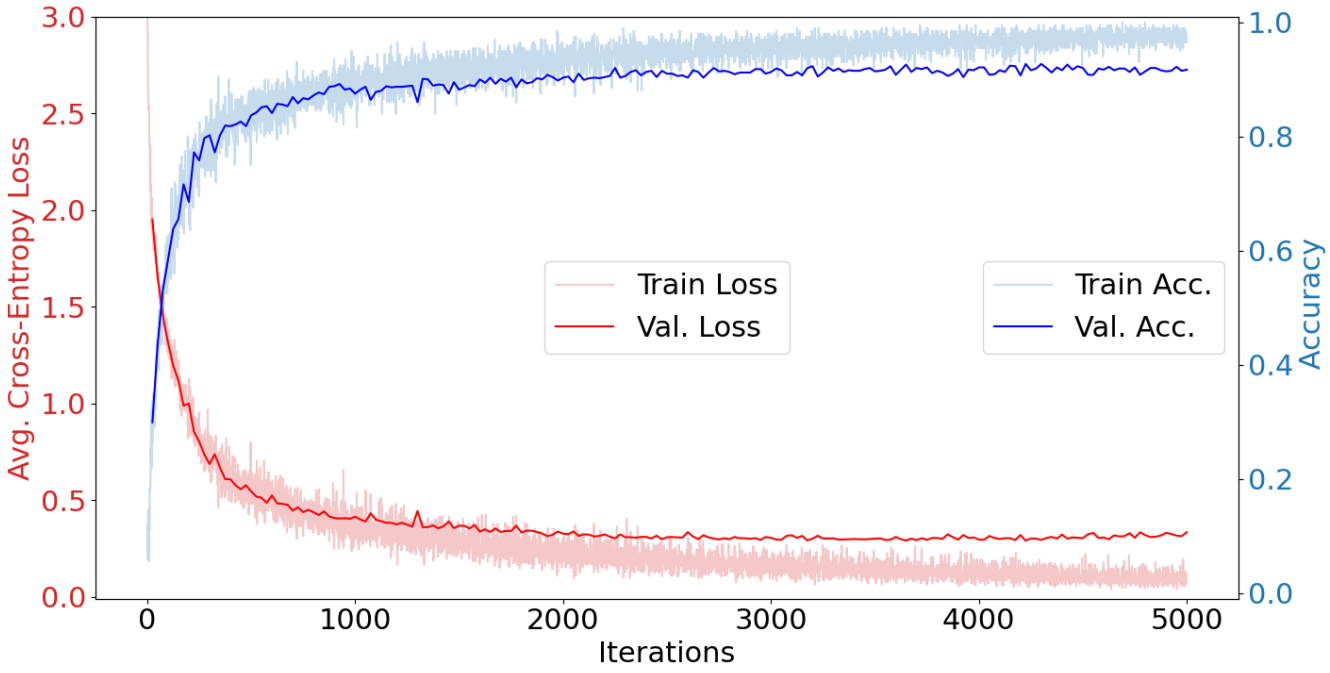
graph. The accuracy probably would have improved with more epochs. ReLU with 5 layers and a step size of 0.01 had the best profile by far. It reached a satisfying result much more quickly and there is a clear relationship between loss and accuracy. It is interesting that there is more of a separation between train and validation accuracy towards the end. I think the increase in learning rate between 1 and 2 is due to a false local optimum that was reached by a small step size. The third test outperformed the first as well. The maximum value of the derivative of ReLU is much greater than that of the sigmoid function. This might allow for faster optimization.



Sigmoid activation with 5 layers and a step size of 0.01



Sigmoid activation with 5 layers and a step size of 0.1



ReLU activation with 5 layers and a step size of 0.01

6. Measuring Randomness
   102: 87.3%
   10: 87.9%
   0: 89.8%
   500: 88.4%
   5000: 88.3%
   2 ^ 32 - 1: 87.9%

   I didn't really notice a significant difference when I changed the random seed value. There is a chance that the values I chose were not representative of the true range created with different random seeds. A very low random seed of 0 produced the best result. As the seed gets very large, the accuracy seems to trend downward. I'm not certain how to interpret these answers, as there is not a clear trend from the very lowest to the very highest random seed. In general, I would assume there is a benefit to considering multiple seeds during hyperparameter selection along with batch size, number of epochs, step size, number of layers, layer width, and activation function.

Debriefing

I spent around 7 hours on this assignment.

I found it easier than the last several assignments, but that was largely because it had less work to do. I found the concepts a bit more complex than the last assignment.

I worked alone, but I attended office hours and used some online resources (primarily Stack Overflow to learn more about Softmax and to read about hyperparameter selection for the Kaggle competition).

I am not 100% on neural networks, but none of the content is far above my level. I understand the high-level concepts. I intend to study the content more in the next few weeks.