1a. (Summary of experiments below)

Experiment	Classes	N	Hidden_Layer_Sizes	Max_Iter	Accuracy (our model)	Accuracy (sklearn mlp)	Accuracy (sklearn logreg)
1	(0,1)	10000	(8,)	200	0.993	0.998	1
2	(0,1)	10000	(8,8)	200	0.995	0.998	1
3	(0,1)	10000	(100,)	200	0.995	0.995	1
4	(0,1)	10000	(2,)	200	0.995	0.995	1
5	(0,1)	10000	(8,)	50	0.991	0.998	1
6	(0,1)	100	(8,)	200	0.5	0.833	1
7	(0,1)	1000	(8,)	200	1	1	1
8	(5,6)	10000	(8,)	200	0.979	0.976	0.989

The general trends in these experiments line up with what we've learned in class, although there were definitely some surprises here. I saw that adding a hidden layer tended to increase the accuracy of our model, which made sense since adding a layer should allow our network to learn more complicated things. I observed that adding nodes to the hidden layer also improved the accuracy of our model as expected, although it decreased the accuracy of the sklearn model. Perhaps this was due to overfitting. Decreasing the number of nodes actually increased the accuracy of our model (maybe telling us we didn't need very many nodes to begin with) but did decrease the accuracy of the sklearn model as I would have expected. Decreasing max_iter also caused a decrease in the performance of our model, which makes sense: given less chance to optimize, it produced less optimal results. This change with a lower max_iter did not happen for the sklearn multilayer perceptron, indicating that perhaps the sklearn MLP had already converged in 50 iterations due to a better gradient descent algorithm. I also observed that decreasing the available data to 100 elements drastically reduced the performance of both networks, which I expected to happen as neural networks tend to get more accurate given more data. It was interesting to note that in the N=1000 case, both networks actually were perfectly accurate; I'm not too sure why that happened other than maybe that those 1000 data points happened to be particularly amenable to classification. Finally, I tried distinguishing between two digits that I thought might look more alike than 0 and 1 (I chose 5 and 6) and I did see that this choice decreased the accuracy of both MLPs and even decreased the accuracy of the sklearn logistic regression. This was expected, since I did think those shapes would look more alike and therefore harder for the models to classify.

1b. (Summary of experiments shown below)

Experiment	N	Hidden_Layer_Sizes	Max_Iter	sklearn_kwargs	Accuracy (sklearn mlp)	Accuracy (sklearn logreg)
1	10000	(32,)	100		0.912	0.895
2	10000	(32,)	100	activation=logistic	0.925	0.895
3	10000	(32,)	100	alpha=0	0.905	0.895
4	10000	(32,)	100	alpha=0.01	0.9	0.895
5	10000	(32,)	200		0.906	0.897

6	10000	(32,)	50	0.889	0.913
7	10000	(2,)	100	0.108	0.895
8	10000	(100,)	100	0.917	0.895
9	10000	(32,32,32)	100	0.9	0.895

I ended up doing more than 9 experiments because I wanted to test too many things (oops). I noticed that the logistic activation tended to improve accuracy over Relu, that changing alpha in either direction caused a decrease in performance (probably because the decreased alpha led to more overfitting, while the increased value of alpha led to underfitting), and that increasing the number of nodes in the hidden layer tended to increase performance. The settings that I found to work the best overall were the standard settings with the logistic activation function, although this wasn't too much better than the default settings. A couple of interesting things: first, setting max_iter to 200 actually decreased the performance of our MLP. I guess the model just happened to do better when it didn't converge as well? It did also get worse for a lower value of max_iter, but for that lower value of max_iter, the performance of the logistic regression model actually improved. Maybe the logistic regression model was slightly overfitting for 100 iterations – I'm not entirely sure. Another thing I noticed was that adding extra hidden layers decreased the performance of the MLP. I think this was probably due to some overfitting that was allowed by introducing a more complicated network.

1c.

Our current model cannot handle ten-class classification because our current classification loss function (binary cross entropy loss) is designed only to handle the loss from classifying two objects. We would need to implement some sort of n-fold cross entropy loss function in order to be able to classify n digits.