Part 1

Q1: For the single-neuron classifier that you instantiate, what is the full name of the tensor object that contains the weights, and what is the name of the object that contains the bias?

The full name of the tensor object that contains the weights is called fc1.weight, and the full name of the tensor object that contains the bias is called fc1.bias.

Q2: What is the name of the tensor object that contains the calculated gradients of the weights and the bias? (This was not covered in class).

They are stored in smallSNC.fc1.weight.grad and smallSNC.fc1.bias.grad respectively. (I named my SNC smallSNC, and I named my first layer fc1.)

Q3: Which part of your code computes gradients (i.e. give the line of code that causes the gradients to be computed). Explain, in a general way, what this line must cause to happen to compute the gradients, and how PyTorch 'knows' how to compute the gradients.

The line that compute the gradients is

```
loss.backward()
```

To compute the gradients, this function must use the predicted output of the SNC and the labels of this particular dataset to find $dloss/dw_i$ *and* $dloss/db$ for all the training data. Pytorch knows these because when we calculate loss, we entered the prediction and the labels in this line.

```
loss = lossFunction(input = predict.squeeze(), target = trainingDataLabelT.float())
```

Q4: Give the training/validation plot versus epoch, and the accuracy vs. epoch for the three cases required in Assignment 2 at the end: a too-slow learning rate, a too-fast learning rate, and a 'just-right' learning rate.

Figure 1. Accuracy and Loss function for learning rate that is too low.

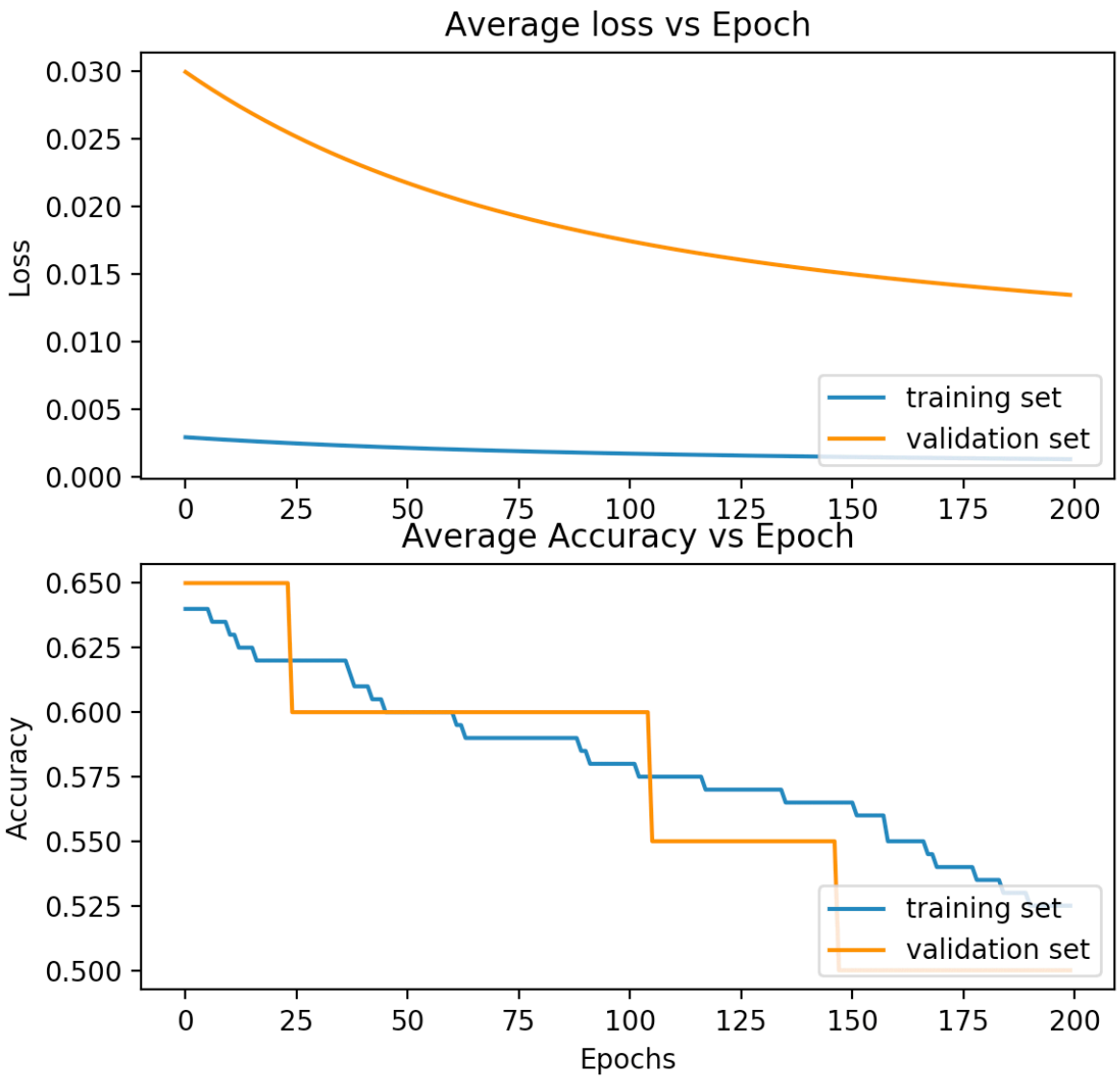


Figure 2. Accuracy and Loss function for learning rate that is just right.

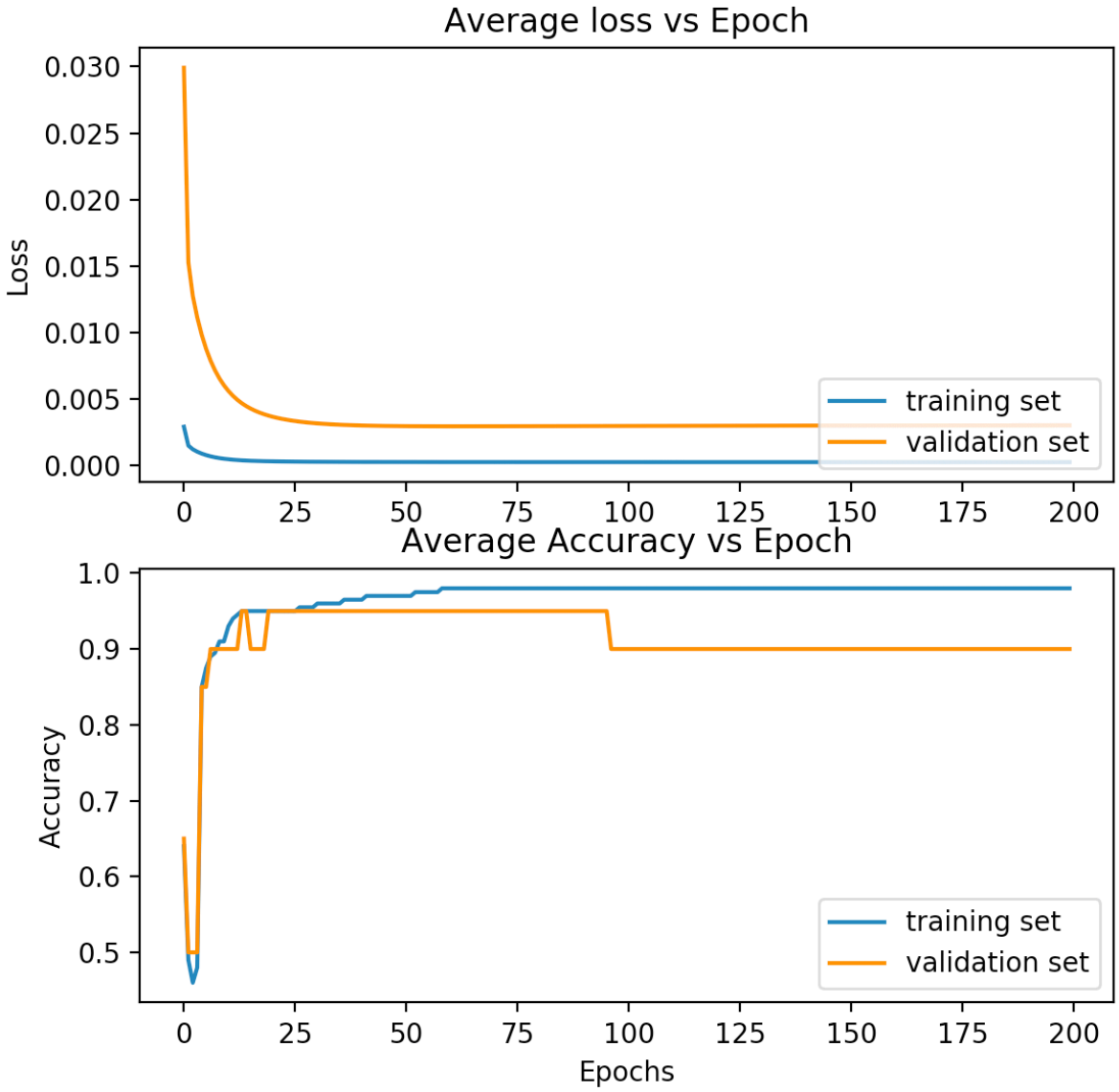
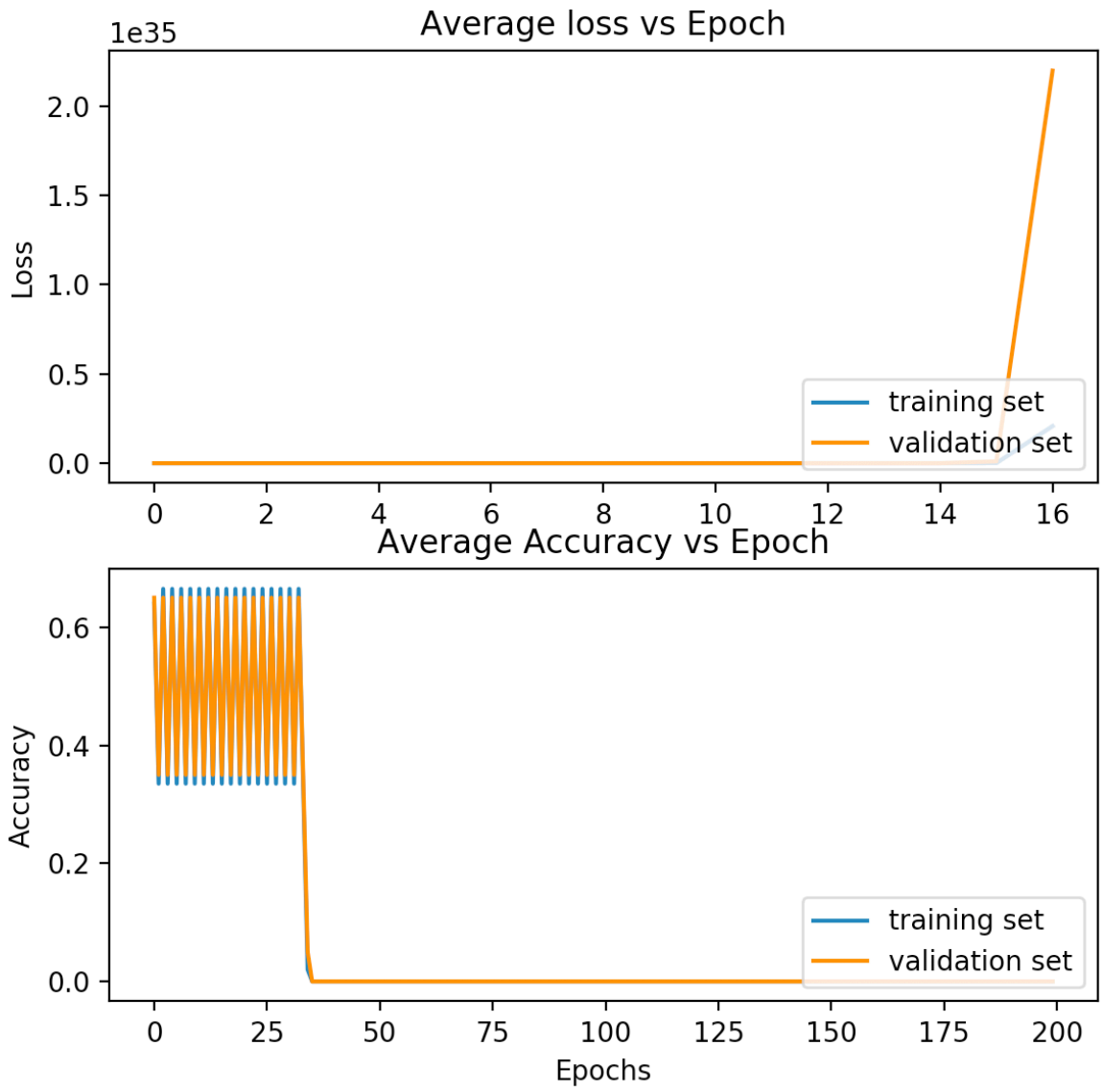


Figure 3. Accuracy and Loss function for learning rate that is too high.



Part 3

Q 3.2.1: How many high income earners are there in the dataset? How many low income earners?

There are 37155 low income earners and 11687 high income earners.

Q 3.2.2: Is the dataset balanced (i.e. does it have roughly the same number of each of the classes/labels - in this case high income vs. low income)? What are some possible problems with training on an unbalanced dataset?

The dataset is not balanced, as there are almost four times as many low-income earners than high income earners. This might cause the training to be biased towards identifying people with low income, while not being good identifying people with high income. The imbalanced data would also make the accuracy less meaningful. In this case of this dataset, even if the model only predicts below 50k, it would still have a 75% accuracy, which is not very meaningful.

Q 3.3.1: How many samples (rows) were removed by the above cleaning process? How many are left?

3620 rows are removed, and 45222 rows are left.

Q 3.3.2: Do you think this is a reasonable number of samples to throw out?

I think this is a reasonable number of samples, as the adult dataset is quite large. 3620 rows account for only 7.4% of the total data. However, if the dataset is a lot smaller, other measures to deal with the missing data should be taken to ensure that there is still some data left.

Q 3.5.1: What is the minimum age of individuals and the minimum number of hours worked per week for the dataset?

The minimum age is 17 years old and the minimum number of hours worked per week is 1 hours per week.

Q 3.5.2: Are certain groups over or under-represented? For example, do you expect the results trained on this dataset to generalize well to different races?

From the pie charts (on the next page), we can see that in terms of race, whites are over-represented while other races are under-represented, for this reason, I don't expect this result to generalize well to other races.

Q 3.5.3: What other biases in the dataset can you find?

Some other biases include:

- The dataset has way more male than female, the female population is under-represented

- The majority of the dataset work for private companies.

Figure 4. Ratio of work classes for entries in the adult dataset

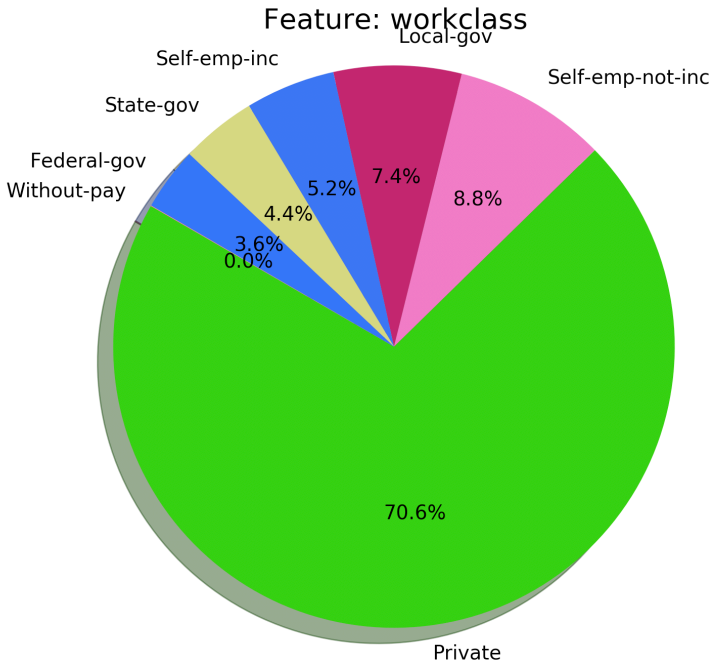


Figure 5: Ratio of race for entries in the adult dataset

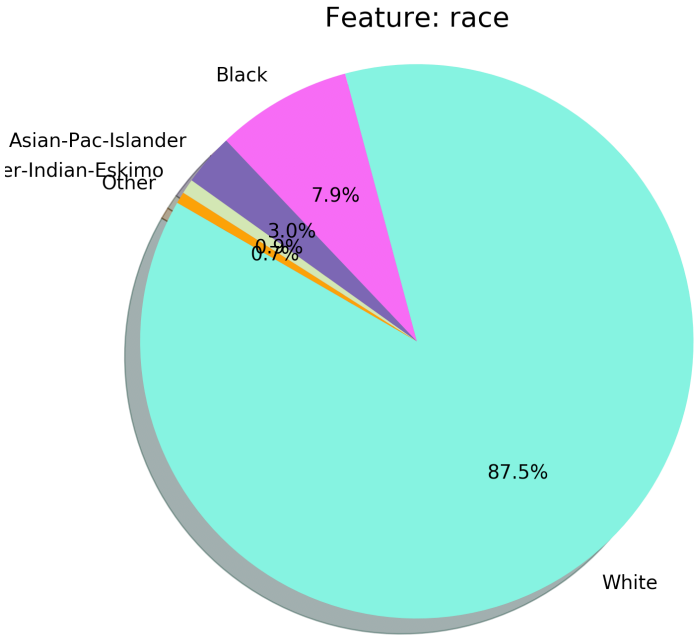


Figure 6: Ratio of education for entries in the adult dataset

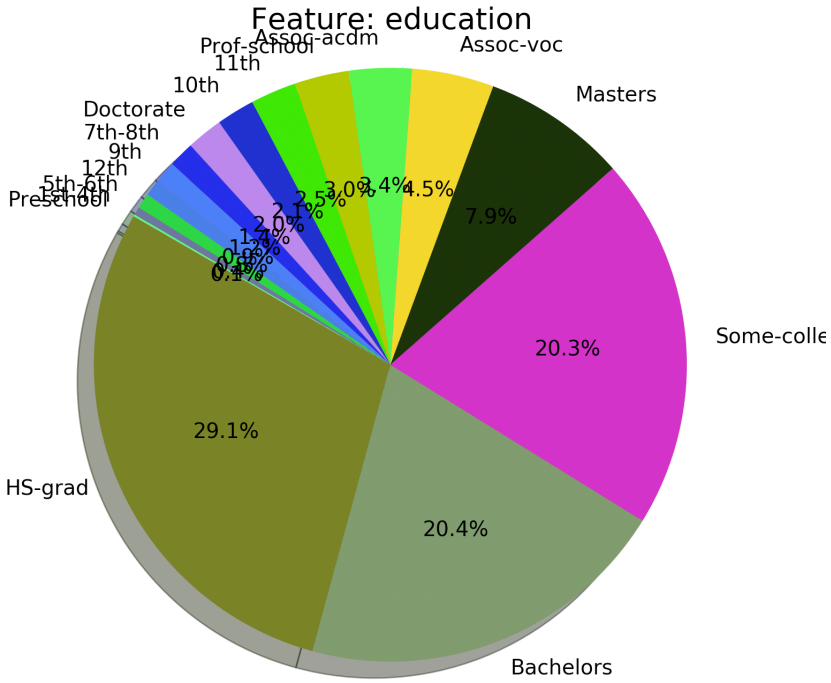


Figure 7: Binary bar chart to show the effect of work class on income level

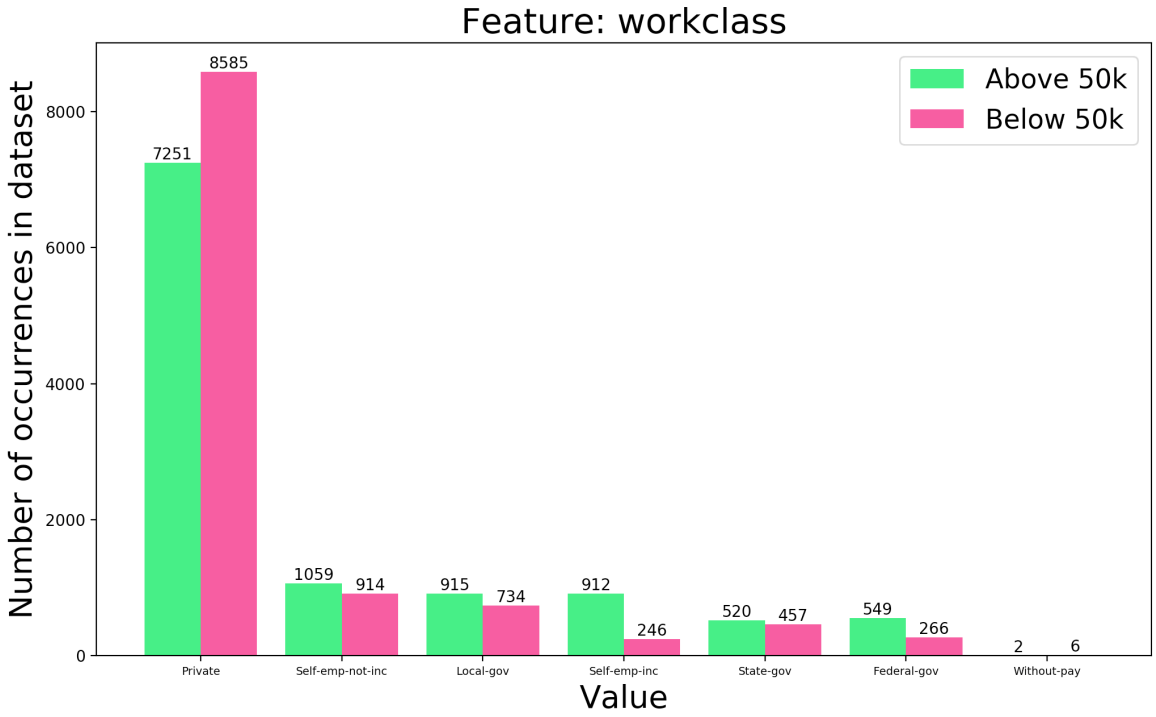


Figure 8: Binary bar chart to show the effect of race on income level

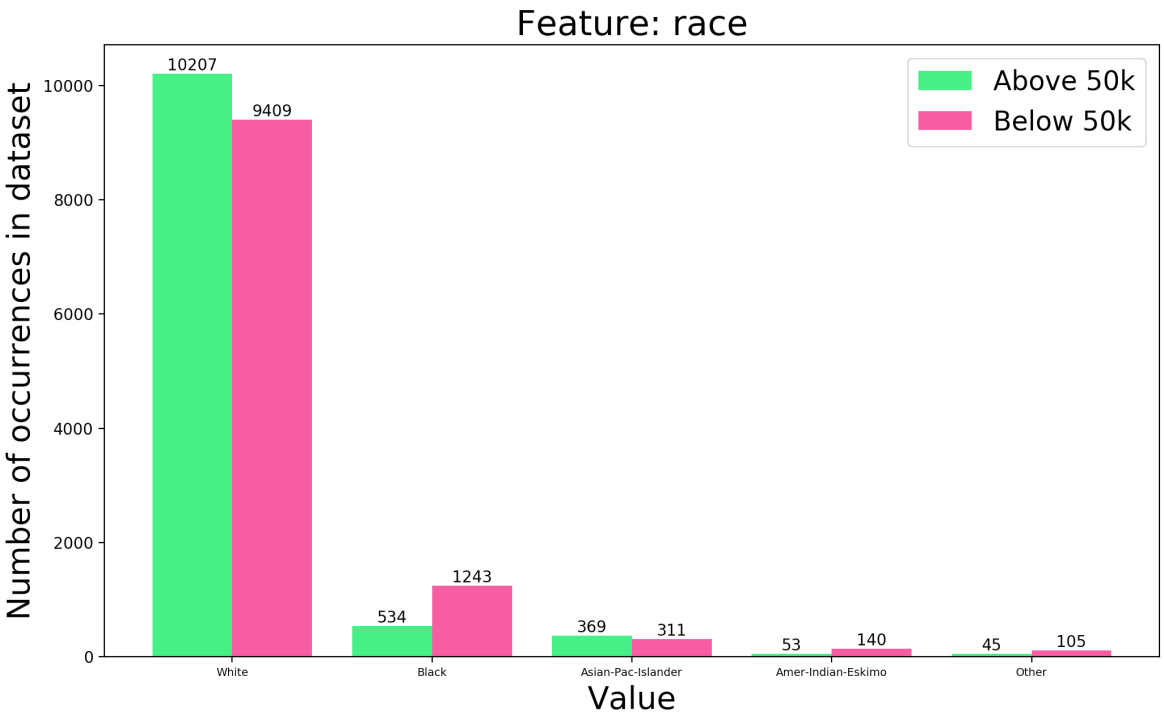
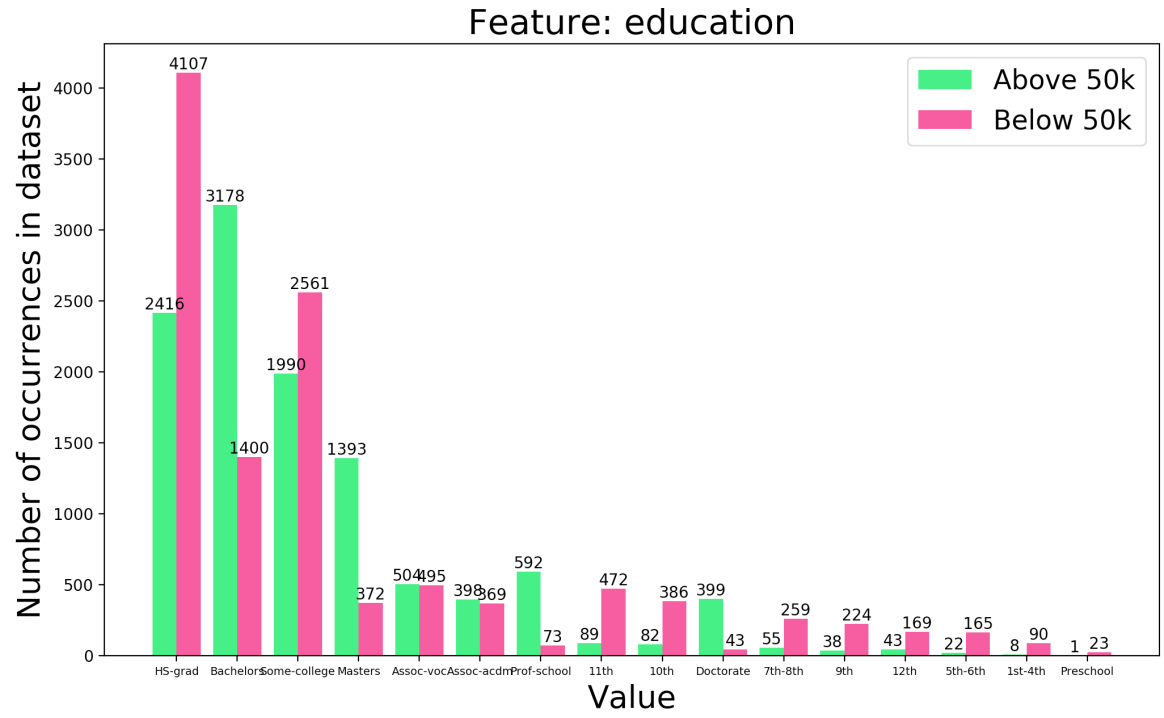


Figure 9: Binary bar chart to show the effect of education on income level



Q 3.5.4: List the top three features, that you identify using this method, that are very useful for distinguishing between high and low salary earners?

The top three features are education, marital status and occupation. Relationship seem to also be useful in identifying high vs low income, but I have omitted it here because it seems to convey the same information as marital status.

Q 3.5.5: Suppose you're told that someone has high school-level of education. How likely would they be to earn above 50K? Given their education is "Bachelors" and nothing else about a person what is the best assumption as to whether they earn above or below 50K?

For someone who has high school level education, meaning that they have graduated high school, considering only about 37% of them earn above 50k. It is not very likely for them to earn above 50k. Meanwhile, about 70% of the people with a bachelor's degree earn above 50k, this indicates that having a bachelor's education would make a person more likely to earn above 50k.

Q 3.5.6: Now suppose you're given a new sample, think about how you would classify (by eye) the person as a high or low income earner. You don't need to come up with an explicit algorithm or submit anything for this question (but you can if you want to think about this problem).

I can examine the categories further and see if certain combinations of features would have a better correlation with the salary, for example a married person with a bachelor's degree might prove to be very likely to have a salary above 50k.

Q 3.6.1: What are some disadvantages of using an integer representation for categorical?

If we use integer to represent categorical data, some categories will have higher value than the other. And since gradient calculation depends on the value of the input, using integers would mean some categories (here I'm using category to indicates the categories within a type of data, for example the categories in "country of origin" would be china, U.S. Canada and etc.) modify the weights faster than the other. The classifier might be tend to look for key categories instead of looking at the data holistically.

Q 3.6.2: What are some disadvantages of using un-normalized continuous data?

Continuous data can be degrees of magnitude apart between categories, for example age range between 1 to 99 but income level can range between 10,000 to 500,000. And since gradient calculation depends on the size of the input, when adjusting the weights, the classifier would be adjusting the weights of some categories (here the categories means the type of data like age, income, height) a lot faster than the other. Which places a stronger bias on categories with value of large degree of magnitudes than those with low degree of magnitude. This would mislead the classifier to be more prone to using certain categories to make predictions even though it might not improve accuracy.

Part 4

Q 4.2.1: Why is it important to shuffle the data during training? What problem might occurr during training if the dataset was collected in a particular order (e.g. ordered by income) and we didn't shuffle the data?

One of the most important reason of using mini-batches is to avoid falling into the local minimum. But if the element of randomness is taken out of mini-batches, then the model might go for the local minimum again.

Q 4.3.1: Give a justification for your choice of the size of the first layer. What should the size of the second (output) layer be?

I chose to let the first layer to have 15 neurons, because there are 15 categories. The size of the second layer should be 1 because we are doing binary classification

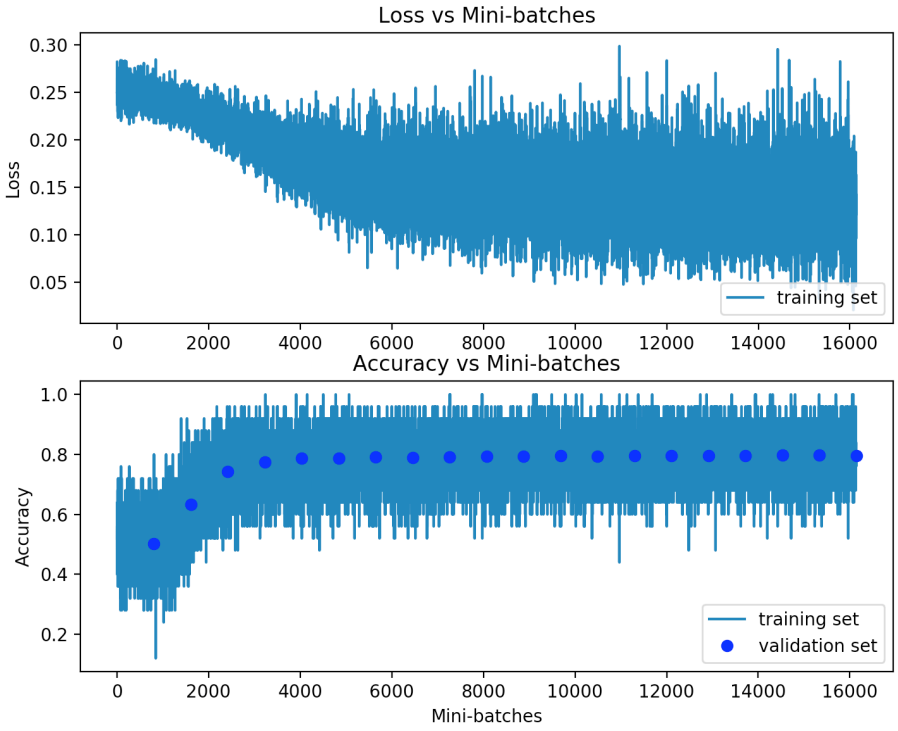
Q 4.3.2: Why do we think of the output of the neural network as a probability between 0 and 1? What do the specific values of 0 and 1 mean if they are output from the network?

Since this neural network is a binary classifier with the two expected output 0 and 1, and that the output ranges from 0 to 1, it make sense to interpret the output as how certain we feel about the result being 1. Specific values of 0 means we are very certain that the output is 0 (<= 50k) and the value of 1 means we are very certain that the output is 1 (> 50k).

Q 4.6.1:

For my model, using a batch size of 25, hidden layer of 15, and learning rate of 0.003, I was able to get an accuracy around 80% for my validation data. The graph is shown below in figure 10.

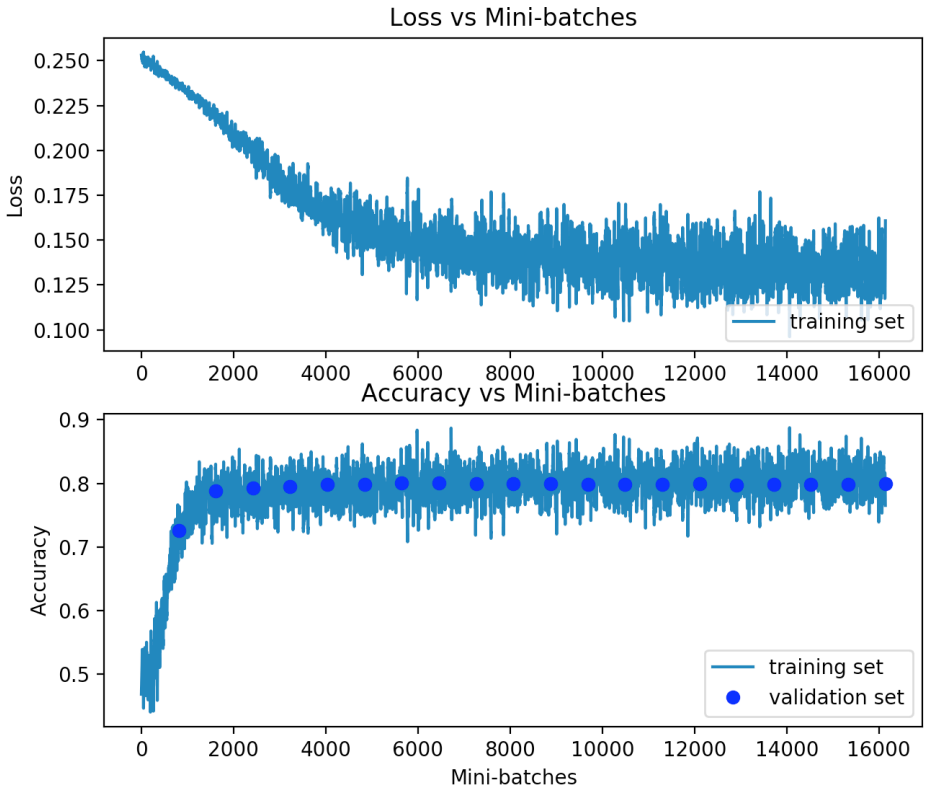
Figure 10: Loss and accuracy of Minibatches



Q 4.6.2:

Through using the savgol_filter, I was able to reduce the noise by a significant amount. A high level of smoothing using the savgol_filter is done by using a high window_length and a low polyorder. In my graph I used the window_length of 25 and the polyorder of 2.

Figure 11: Loss and accuracy of Minibatches, smoothed using savgol_filter with 25 as the window length and 2 as the polyorder

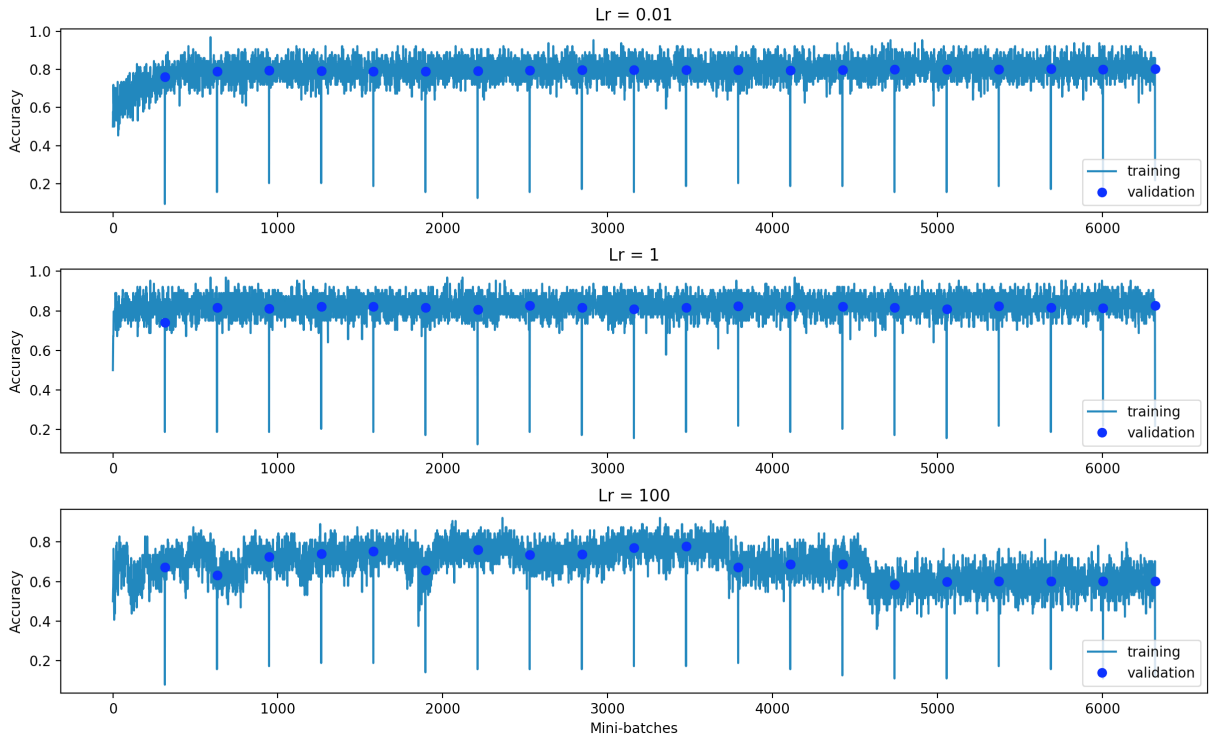


Part 5

Table 1. Validation accuracy for different learning rates with 64 as the batch size, 64 neurons in the hidden layer, and 20 epochs.

Learning Rate	Training Accuracy	Validation Accuracy
0.001	0.779	0.777
0.01	0.813	0.803
0.1	0.829	0.823
1	0.837	0.822
10	0.839	0.811
100	0.500	0.502
1000	0.500	0.498

Figure 12: training and validation accuracy for learning rate of 0.01, 1, and 100, with 64 as the batch size, 64 neurons in the hidden layer, and 20 epochs.



Q 5.1.1: Which learning rate works the best?

Under the current set of hyperparameter, that is:

> Number of epoch = 20, batch size = 64, 1 hidden layer of 64 neuron, and relu as the activation function, and random seed of one.

A learning rate of 0.1 achieved the best validation accuracy.

Q 5.1.2: What happens if the learning rate is too low? Too high?

Much like the SNC, when the learning rate is too low, the MLP learns very slowly. When the learning rate is too high, unlike the SNC, the MLP does not oscillate, nor does it converge to a higher accuracy, it seems like it only changes accuracy randomly.

Figure 13: training and validation accuracy for different batch sizes, with 0.1 as the learning rate, 64 neurons in the hidden layer, and 5, 20, and 30 epochs respectively.

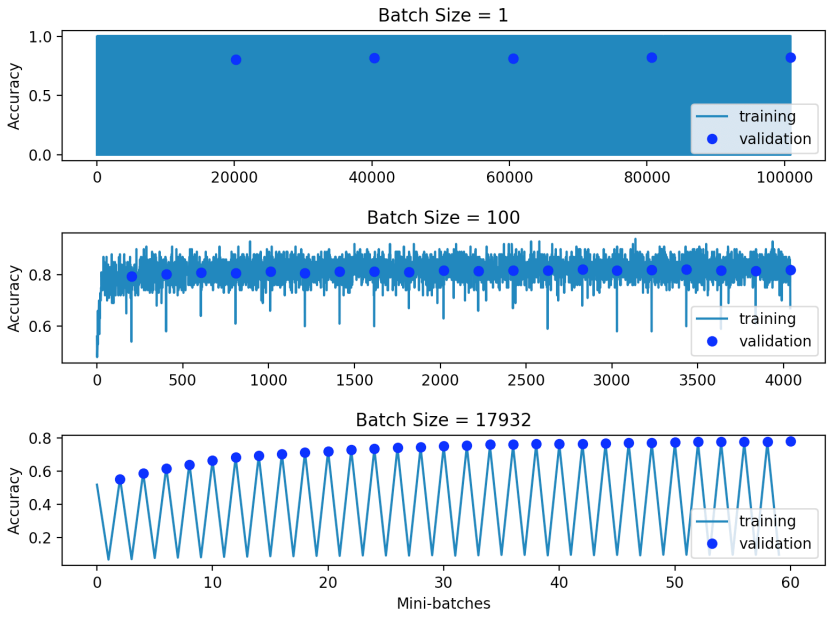
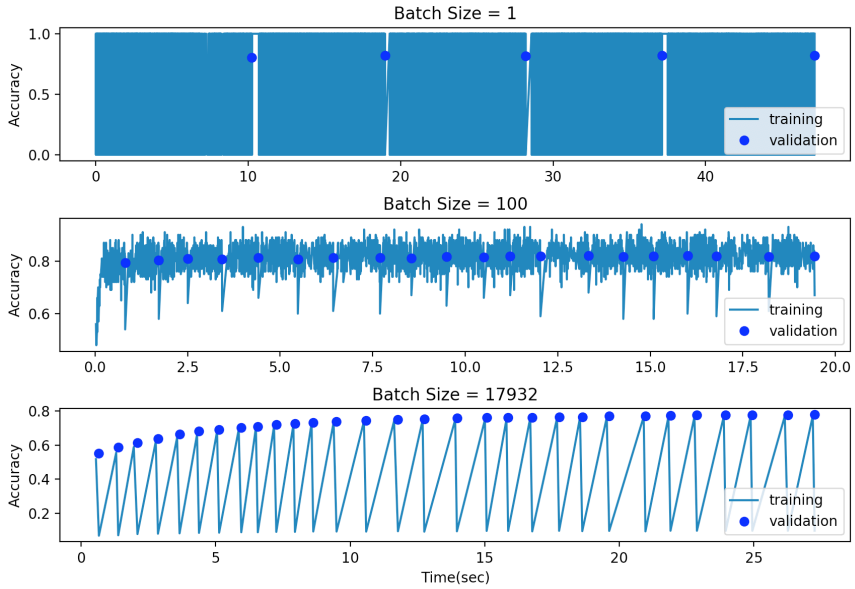


Figure 14: training and validation accuracy for different batch sizes, with 0.1 as the learning rate, 64 neurons in the hidden layer, and 5, 20, and 30 epochs respectively, displayed against time



Q 5.3.1: Which batch size gives the highest validation accuracy?

The batch size of 100 gave the highest validation accuracy, with a value of 82.2%.

Q 5.3.2: Which batch size is fastest in reaching a high validation accuracy in terms of the number of steps? Which batch size is fastest in reaching its maximum validation accuracy in terms of time?

The batch size of 17932 is fastest in reaching the high validation accuracy in terms of number of steps. The batch size of 100 is fastest in reaching the high validation accuracy in terms of time.

Q 5.3.3: What happens if the batch size is too low? Too high?

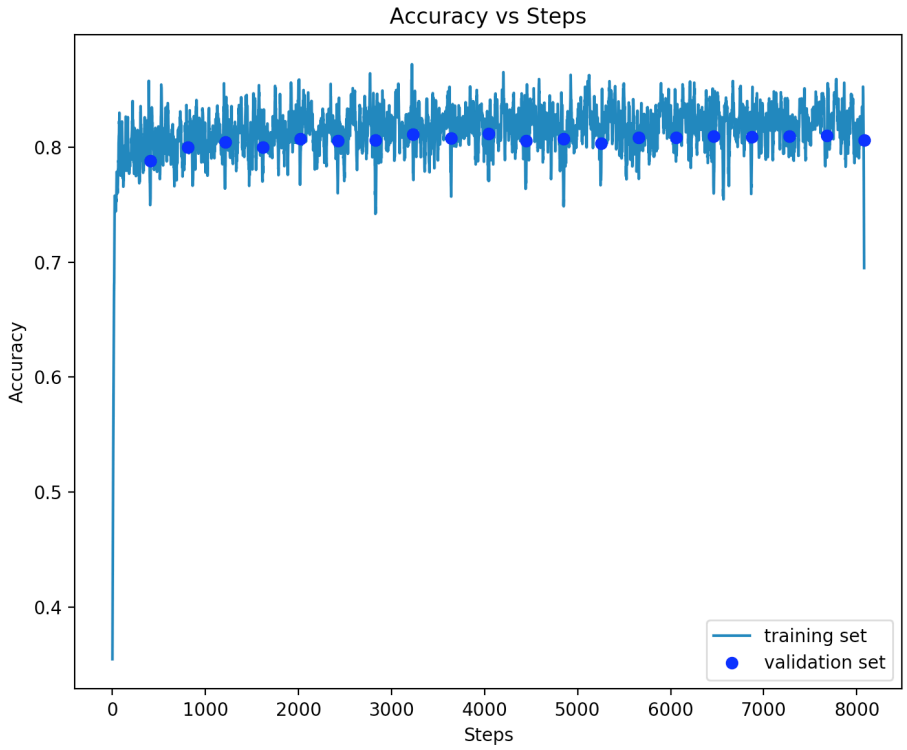
When the batch size is too low, the accuracy oscillates because the gradient computed based on only one training example is not very reliable, so the weights are adjusted very rapidly but seemingly-randomly. In the case of a very high batch size, the training would lack the merit of a stochastic gradient descent, as each step involves a large portion of the dataset. Therefore, it would be more likely to fall for local minimums as supposed to global minimum.

Q 5.3.4: State the advantages and disadvantages of using a small batch size. Do the same for large batch size. Make a general statement about the value of batch size to use (relative to 1 and the size of the dataset

The advantage of using a small batch size is that the model would be less likely to fall for a local minimum. The disadvantage is that the model is expansive to train since it would require a lot of gradient calculations. A large batch size takes less time to train as gradients are computed less often, but it would be more likely for it to be stuck on local minimums. To get the most out of both worlds, the batch size between 10 to size/10 would be reasonable.

Q 5.4.1: Graph for underfitting

Figure 15: training and validation accuracy for MLP with no hidden layer, 0.1 learning rate, 20 epochs and 50 as batch size



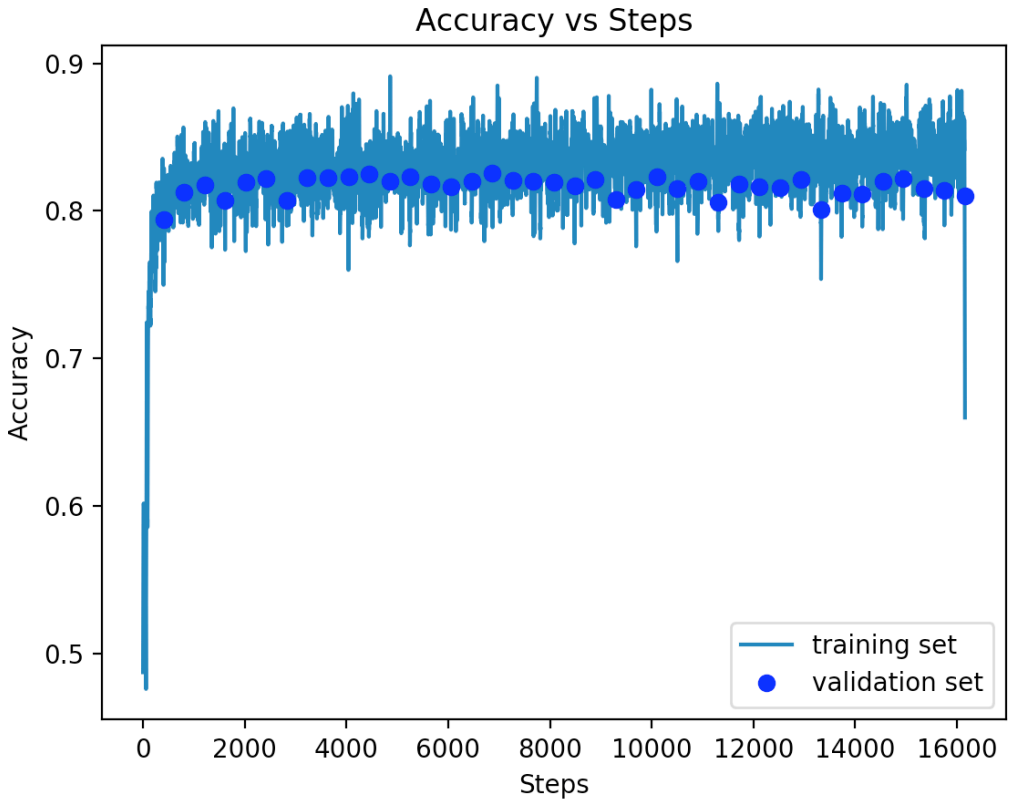
Q 5.4.2: What validation accuracy does the small model achieve? How does this compare to the best model you've trained so far? Is this model underfitting?

The small model is able to achieve an accuracy of 81%, which is slightly less than the larger model (82%) when put under the same set of hyperparameters.

I think the model is not underfitting, as the validation accuracy the model achieved is relatively high, and the training accuracy is slightly higher than the validation accuracy from the figure. To answer why, I would like to draw an analogy with the SNC from assignment two. In assignment two, there are 9 inputs, while in the adult data set, there are 14 inputs before the one-hot encoding. In essence, the two problems might have similar complexity. Therefore, in the case of this assignment, a SNC might be good enough.

Q 5.5.1 Overfitting

Figure 16: training and validation accuracy for MLP with 3 hidden layers of 64 , 0.1 learning rate, 20 epochs and 50 as batch size

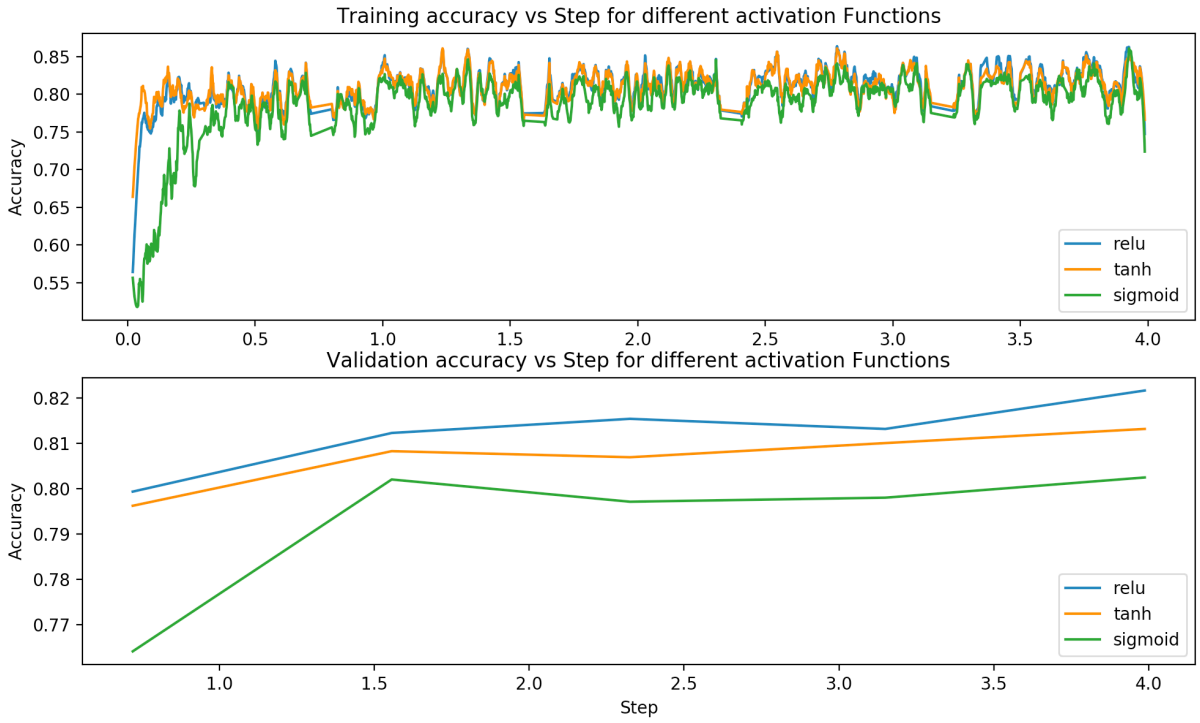


Q 5.5.2: What validation accuracy does the large model achieve? How does this compare to the best model you've trained so far? Is this model over-fitting?

The large model achieved a validation accuracy of around 82%, which is similar to the best model from before. I think the model is overfitting. In figure 16, we can see that the validation accuracy is significantly less than the training accuracy towards the later epochs.

Q 5.6.1

Figure 17: training and validation accuracy for MLP with different activation functions



From the graph, we can see that the relu function and the tanh function has very similar shape. On the other hand, the sigmoid function seems to train a bit slower than the tanh and relu function.

5.6.2: Is there a difference between the activation functions in terms of how long they take to run?

Table 2: runtime of different activation functions with 5 epochs, 20 as batch size

	relu	tanh	sigmoid
time spent	5.741394	5.169744	5.341916

From the table, we can see that the relu function takes the longest to run, while sigmoid and tanh are faster.

5.7:

My hyper parameter is mostly done using a grid search. First, from part 5.2, I found that the learning rate of 0.1 works best, so I stuck with it despite changing other hyper parameters. I started my search with finding the right number of neurons in the hidden layer

Table 3: using grid search to find the batch size and number of neurons that produces the highest validation accuracy

Validation Accuracy		Number of Neuron in hidden Layer 1					
		5	**10**	**15**	**20**	**30**	**50**
Batch size	**10**	0.822	0.821	0.819	0.819	0.821	0.818
	20	0.822	0.810	0.822	0.818	0.823	0.817
	100	0.821	0.816	0.822	0.822	0.822	0.819
	200	0.818	0.822	0.821	0.818	0.819	0.814
	500	0.810	0.809	0.814	0.814	0.810	0.810
	1000	0.803	0.802	0.806	0.808	0.809	0.802
	2000	0.800	0.797	0.799	0.800	0.802	0.797

After determining that a number between 15 and 30 works best, I took 20 as the number of neuron and started testing different number of epochs and mini-batch sizes.

Table 4: using grid search to find the batch size the least number of epochs that produces the highest validation accuracy

Validation Accuracy		Number Of Epoch					
		1	**3**	**5**	**10**	**20**	**30**
Batch size	**10**	0.810	0.823	0.822	0.822	0.820	0.817
	20	0.811	0.819	0.825	0.821	0.821	0.818
	100	0.792	0.806	0.813	0.816	0.819	0.818
	200	0.788	0.795	0.803	0.809	0.815	0.815
	500	0.731	0.787	0.793	0.802	0.807	0.811
	1000	0.505	0.775	0.785	0.794	0.798	0.807
	2000	0.498	0.651	0.769	0.786	0.794	0.798

In this search I found that having relatively small batch size can achieve high validation accuracy with very few epochs. To confirm my finding, I also compared the run time, shown in table 5.

Table 5: using grid search to find the batch size the least number of epochs that has the lowest runtime.

Time Spent		Number Of Epoch					
		1	**3**	**5**	**10**	**20**	**30**
Batch size	**10**	1.69	7.47	8.61	12.99	25.24	39.38
	20	1.23	2.30	3.61	7.81	16.17	23.69
	100	0.71	1.48	2.57	6.68	13.62	20.42
	200	0.47	0.90	1.98	4.78	7.91	13.31
	500	0.43	1.13	2.17	4.26	9.46	10.55
	1000	0.36	0.89	1.48	3.45	6.84	10.75
	2000	0.49	1.01	1.99	4.63	11.49	13.86

Therefore my final set of best parameters are

Batch size = 20

Epochs = 5

Learning rate = 0.1

Number of Neuron in hidden layer = 20

Random seed = 1