Problem 1

Initially, I had my batch size set to 60, with a convergence threshold of 0.0001. This gave an accuracy of 83.2 percent, with 209 epochs. If I were to try different batch sizes of 1, 5, 10, 40, 75, and 99:

Batch 1: Test Accuracy: 92.2 percent, Number of Epochs: 15
Batch 5: Test Accuracy: 91.8 percent, Number of Epochs: 82
Batch 10: Test Accuracy: 89.1 percent, Number of Epochs: 107
Batch 40: Test Accuracy: 84.2 percent, Number of Epochs: 167
Batch 75: Test Accuracy: 83.6 percent, Number of Epochs: 282
Batch 99: Test Accuracy: 83.9 percent, Number of Epochs: 392

a. In general, although, as can be seen above with the batches of different sizes, it appears that a faster convergence occurred resulted in a higher testing accuracy, this does not necessarily mean that the number of gradient updates are lower. That is, a "faster" convergence may not be actually occurring. With smaller batch sizes, gradient descent is a 'noisier' process. Therefore, this leads to two potential trade-offs between speed of convergence and accuracy. Due to the gradient descent process becoming "noisier" because of the loss being changed several times, it is possible that the convergence speed could actually decrease. However, at the same time, it is also possible that accuracy, too, could decrease, as there may also be a greater amount of uncertainty in relation to the weights. Therefore, a quicker convergence could potentially result in a lower accuracy due to uncertainty in weights, or, greater test accuracy could also signify a slower convergence due to the amount of data being processed during each iteration.

b. Since my convergence threshold was set to an extremely low value (0.0001), it is clear that it took longer to converge. However, with a higher threshold (say, 0.1), convergence occurs faster. Unfortunately, increasing the time it takes to converge also simultaneously decreases the accuracy, as can be seen with the batches above. The larger the batch size, the longer it took for convergence to be reached (as can be seen with the larger number of epochs), and the lower the accuracy. Thus, I noticed that larger batch sizes result in lower accuracy, which I believe because, as there is more data to be computed with each iteration (when there are more batches), this causes less information to be considered, therefore decreasing accuracy.

Problem 2

Looking at the Google Colab, the following categories were one-hot encoded: work class, marital status, occupation, relationship, race, and native country.

To begin with, it is easier to represent work class with true/false binary values (1 or 0), given that there are several possible attributes associated with one's work class, all of which can be answered with a simple 'true' or 'false'. Having '1' correspond to true, and '0' to false, would allow for several of the workclass features to be created into a single vector, with each vector representing a person. Similarly, the same would apply for occupation, relationship, race and native country. All of the mentioned categories are composed of several different attributes, all of which may or may not be marked as 'true (1)' or 'false (0)'. In converting all of these features into separate vectors (for each individual in the data set), with each vector containing either 1's or 0s representing the presence of one attribute or another correlating to that feature, this allows for the training data to become more useful, as it could be easily re-scaled. Probabilities can easily be computed, thanks to one-hot encoding.

Although it is easier to use an enumeration to represent all of the possible values for each feature, having to handle that form of categorical data slows the process of computing probabilities, as the enumerations of data are not easily converted into values that can be quickly converted and calculated by the program. For example, using the values 1,2,3,4, and 5 to represent different colors would potentially cause bugs, given that colors are essentially 'equal' in a sense. Enumerating thus gives an ordering for different feature categories, whereas one-hot encoding allows for categories to be equally considered. With one-hot encoding, this speeds up the process by simplifying the representations of each feature and its attributes, therefore allowing probabilities to easily be calculated. Moreover, this also ensures that values are balanced.

Problem 3

When my model was normalized, with a batch size set to 60 and convergence threshold of 0.0001, my test accuracy was at 83.9 percent, with 209 epochs. However, after running it with the unnormalized data, my test accuracy went down to 34.1 percent, with only 23 epochs. Having the data be unnormalized negatively impacted the accuracy because, when the data no longer was scaled appropriately (i.e., normalized), this likely gave more weight to certain features over others. Not having the values be within a common range affected the model's ability to find any accurate or proportional relationships amongst the data.

Moreover, the SGD optimizer cannot tune all the weights for all the features at the same time if the features have different scales (as a result of being unnormalized. Due to there only being one alpha parameter for each range of values, this also makes it difficult for the model to properly make predictions.

Problem 4

When I ran the data on the given file, my test accuracy actually went up to 84.4 percent, with 264 epochs. Given that both models, regardless of the race/sex attributes being included, were essentially equally accurate, with only a less than one percent difference, it is clear that race and/or sex do not have any significant impact on education level. This therefore implies that there is little to no correlation between race/sex and education level, and that the model was still accurate (if not more accurate) without those features being included in the data.