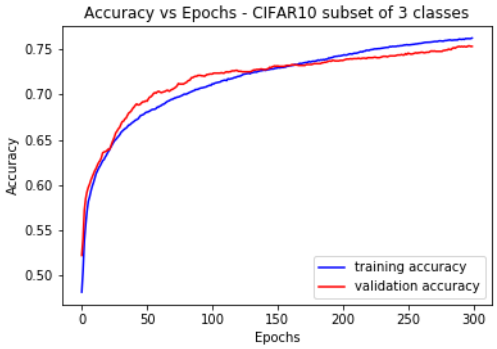
Tuan Tran – HW1 – CS595

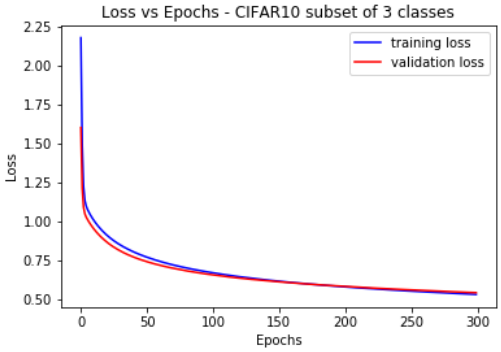
A. Overall program design:

- impl folder contains DenseNeuralNet.py file which contains all helper functions (including functions required by task 2 and 3 like load_spam_data) and also contains the implementation of the neural network model as a class.
- 3 jupyter notebooks corresponding to running the 3 tasks.

B. Task 1:

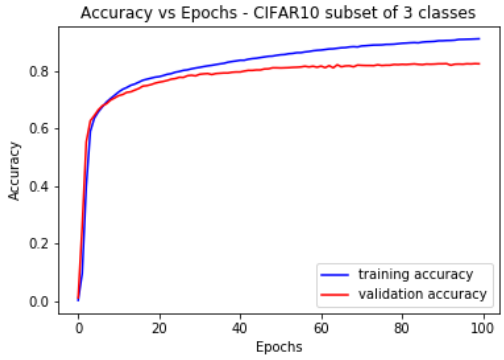
- **Data preprocessing**: since the data comes from keras, little preprocessing was needed. The only thing needs to be done was select a subset of 3 classes (I selected y=1, 2, 3) and split into train/validation/test set
- **Network design:** For the first task, since the number of training, validating and testing instances are in the 5 digits range, and since the feature dimension (after vectorize) is 3072, I decided to start with a reasonably complex network where number of hidden layers = 4 to 6, number of units per layer = 256. I chose categorical crossentropy as the loss function as since our target has more than 2 values (and thus our problem is categorical classification) and has been one-hot encoded. For evaluation metric, I used categorical accuracy for similar reason. For optimizer, I chose SGD first with default learning rate of 0.001, since it is among the most fundamental ones and has proven to work quite well and to converge to a small value reasonably well.
- **Issue:** My initial model suffers from an issue where the training and validation loss does not go down at all and accuracy was bad. My speculation was that the problem lies in the learning rate. I speculated that the learning rate was a bit high and as a result gradient descent is taking larger than necessary step and thus will jump around and not able to minimize the loss. I started decreasing the learning rate and I arrived at learning rate = 0.000001 where the model was finally stable
- **Tuning and result:** After the model is stable, I started playing around with other hyperparameters to reach the best model. The configuration (trained for 300 epochs) that was able to achieve 75.3% accuracy on test set is: hidden layers = 10, hidden units per layer = 512, batch size = 512, learning rate = 0.000001, loss = categorical crossentropy, optimizer = SGD. I trained the model for 300 epochs and achieved a 75.3 % accuracy on test set: (below are graphs for model trained with SGD)

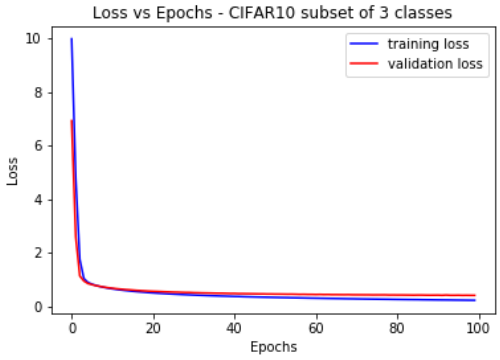




However, considering that my network is quite complex, and Adam has been known to perform well for deeper network, I decided to switch optimizer to Adam and play around with the

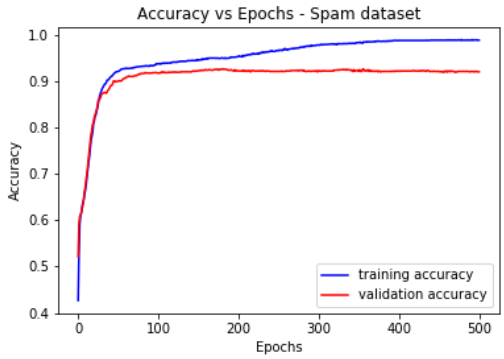
hyperparameters. Ultimately, I came to the best performing model with mostly the same hyperparameter as the model above (hidden layers = 10, units = 512, lr = 0.000001, etc). I trained the network with Adam, for 100 epochs (it overfits if more than that) and achieved a **86.7% accuracy** on test set, a **11.4% increase in accuracy** over the previous model trained with SGD: (below are graphs for model trained with Adam)

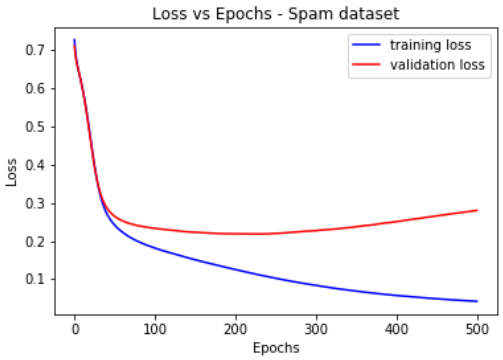




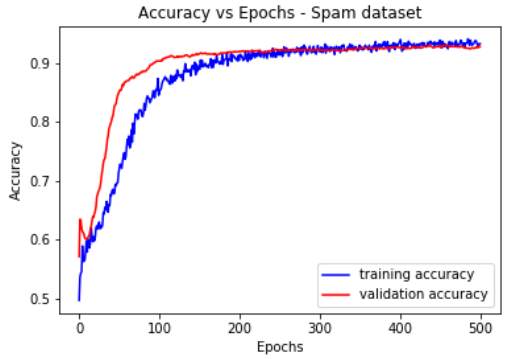
C. Task 2:

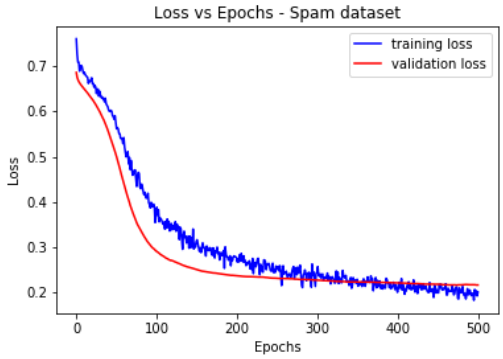
- **Data preprocessing**: after loading the data in as a 2-d tensor, I first shuffle the data (shuffle the rows) since when loaded in, all y=1 instances were on top and all y=0 instances were on the bottom. Afterwards I proceed to split the data into training and test set by the ratio of 60/40 and then further split training data into validation set by the same ratio of 60/40 (I tried with 80/20 and the result was the same). After that, I normalized the data as there were features in the range of 0 to 1 while there were ones that went up to 4 digits. I split the data first into training/validating/testing then normalize so that I do not introduce any potential information about the training set distribution to the validation and test set.
- **Network design:** Since there were only 1656 training instances with 57 features, I decided to go for a reasonably simple model that is still complex enough to do the task. Output activation was sigmoid, and loss was binary crossentropy since this is a binary classification. I play around with the hyperparameters a little bit and the best performing model I found is: hidden layer = 4, hidden units = 64, trained for 500 epochs and 128 batch size. The model was trained with SGD with a learning rate of 0.01.
- **Issue:** at learning rate = 0.01, the model was overfitting:





To prevent this overfit, I decide to try both decreasing the learning rate and adding dropout to the network. I found that adding dropout led to a consistent 3% increase in accuracy over just simply decreasing the learning rate. The following graphs are the results of dropout:

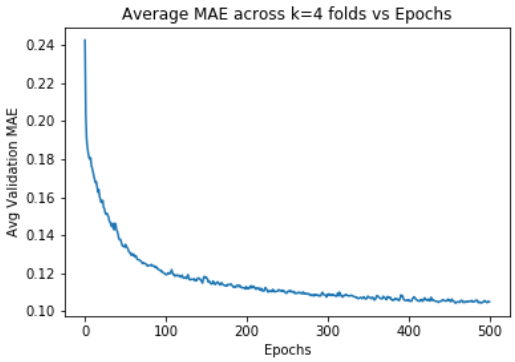


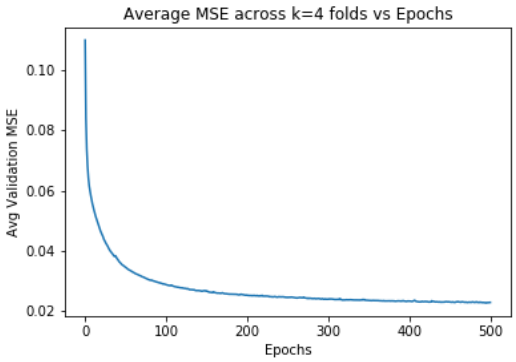


- **Tuning and result**: I played around with the hyperparameters but none beats the current model configuration. Adding more layers or units didn't help probably because the problem was simple enough that we did not need an overly complex architecture. I also played around with other optimizer and found little difference as the network itself is pretty shallow. Overall, the best model (with dropout) gives an **accuracy of 92.99%** on the test set

D. Task 3:

- **Data preprocessing:** first the data was loaded in as a 2-d tensor. Overall, there are 1994 instances, with 128 attributes. There are 5 non-predictive features and 23 missing data features. I then remove the first 5 features since they are not predictive. Of the 23 missing data features, there are 22 with a total of over 1600 missing values each (out of 1994 instances) and thus these 22 features were removed. The removed features are:
  'LemasSwornFT', 'LemasSwFTPerPop', 'LemasSwFTFieldOps', 'LemasSwFTFieldPerPop', 'LemasTotalReq', 'LemasTotReqPerPop', 'PolicReqPerOffic', 'PolicPerPop', 'RacialMatchCommPol', 'PctPolicWhite', 'PctPolicBlack', 'PctPolicHisp', 'PctPolicAsian', 'PctPolicMinor', 'OfficAssgnDrugUnits', 'NumKindsDrugsSeiz', 'PolicAveOTWorked', 'PolicCars', 'PolicOperBudg', 'LemasPctPolicOnPatr', 'LemasGangUnitDeploy', 'PolicBudgPerPop'
  Of the 23, 1 feature (OtherPerCap) has 1 missing value, thus this value was imputed; that is, this value got replaced with the mean of all the values of that feature. After cleaning the data, I split the data into training and test set by the ratio of 60/20. No normalization was needed as the data was already normalized. Finally I casted the element type of the data into float to ensure type consistency
- **Network design**: there were a total of 1196 training instances, so similar to task 2, I decide to build a reasonably simple network with 3 hidden layers and 128 units each. There was no output activation (so linear activation) and loss was mean squared error, metrics was mean absolute error since this is a regression problem. The model was trained with SGD with a learning rate of 0.001
- **Tuning and result**: I played around with the hyperparameters (increasing model complexity, changing optimizer, learning rate, etc) but nothing beats the current model configuration. The network was trained for 500 epochs with batch size 64 and without any dropout:





Overall, the best model (with dropout) gives a **MAE of 0.096** on the test set.