CS577 – Deep Learning
Homework 1 – Task 3

Niti Wattanasirichaigoon
A20406934

Problem Statement

Load the crime data from the UCI repository "Communities and crime". Prepare the data you load as a tensor suitable for neural network. Normalize features as needed. Explain the steps you perform in preparing the data and justify them. Write a function "load_crime_data" to load the data and split it into training and testing subsets. Repeat the steps you performed in task 1 except that in this case you are required to perform k-fold cross validation. To report the performance of the cross validation average, report the validation error of all the folds and plot it as a function of epochs. Retrain the final model on all training data (i.e. all folds) and test the final performance on test set.

Proposed Solution

As always we start by studying out dataset and plan an appropriate solution to the problem.

1. **Data Preparation:**
   Looking into the dataset [1], I learned that it has a total of 128 attributes (122 predictors, 5 non-predictors and one response). The response or GOAL is the crime rate, *ViolenceCrimesPerPop*. Since this response is a continuous value, we are working on a regression problem.
   First we will have to remove the non-predicting features as described in the repository (state, county, community, community name, and fold) which are the first 5 columns from our dataset. We do not have to normalize the other numeric features because it had already been done, but we need to replace missing values. The dataset will be split into 80-20 for training and testing. The entire process will be written as a function, returning the predictors and labels for both training and testing.

2. **Model Initialization:**
   With 122 predicting features remaining, 16-32 nodes for the input and hidden layer appears reasonable. Unlike the classification problems in task 1 and 2, we use the mean squared error (MSE) as our loss function. We can use either MSE or MAE (mean absolute error) as our metric (MSE penalizes far away predictions more, while MAE can be easily

interpreted as how many units is the model's guess wrong by on average). The final output layer of our model does not need an activation function.

3. **Training and Validation:**
   I performed k-fold cross validation with k = 5 in model training, using a batch size of 32 and trained over 40 epochs (the sizes are smaller than that in task 2 because here we only have under 2000 data points). They will be trained over different combinations of number of hidden layers, nodes per layer, and epochs. I will be evaluating the models based on their cross validation MAE.

4. **Final Model Evaluation:**
   The best combination of hyper parameters that generated the model with best MAE will then be used to train our final model. The test data will be used to evaluate our model's performance based on the test MAE.

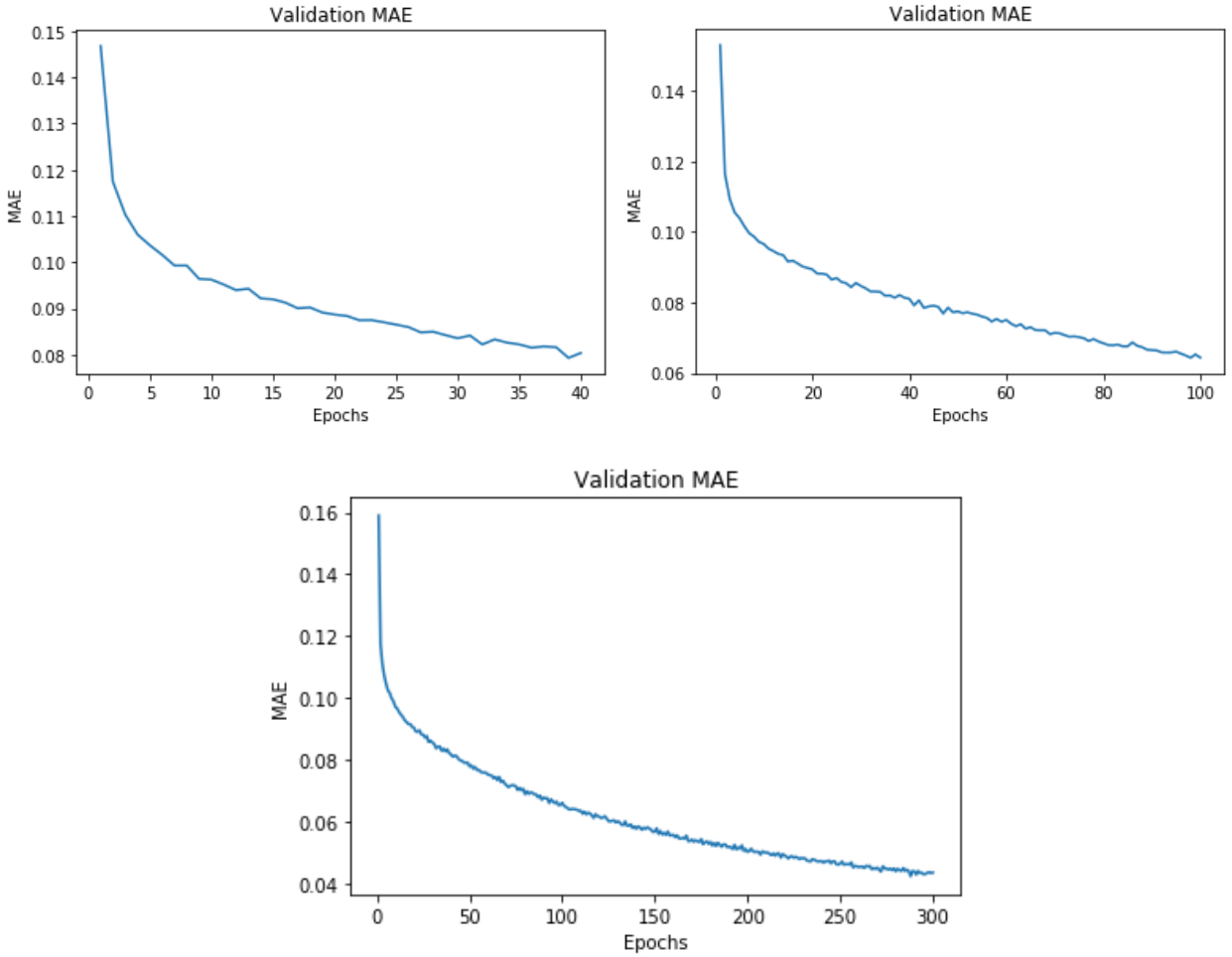
Implementation Details

Several changes were made during implementation, these include:

1. The missing values in the dataset were strings of '?' and not NaN values, I had to replace these with *np.nan* before being able to conduct imputation. Since the values have already been normalized (mean = 0), I decided the missing values should be replaced by the median instead. Replacing values by the mean would nullify any weights placed on these data points.
2. The model's MSE continued to decrease after 40 epochs. I increased it to 40 epochs so we can be certain that the MSE can stabilize.
3. Since I am using batches of 16 and 40 epochs, the graph of average validation MAE is smooth enough to observe (we do not have to apply smoothing).

Results and Discussion

The initial model was trained using batch sizes of 32 over 40 epochs. We can see that the validation MAE still continues to decrease at 40 epochs. To see where the validation MAE stops decreasing, I increased the epochs to 100 and then 300.



The cross-validation MAE is the average MAE between all folds, of which the CV-MAE of the model with 300 epochs is **0.1152**, even less than the model with 100 epochs. This suggest too many epochs will *overfit* our training fold as well. The validation MAE stopped decreasing after it reached around epochs. I then experimented with different models, and recorded their cross-validation MAEs.

Model	Layer Nodes	Epochs	CV-MAE
1.	Input(32), Hidden(32)	300	0.1152
2.	Input(32), Hidden(32)	100	0.1068
3.	Input(32), Hidden(32)	80	0.0990
4.	Input(32), Hidden(32)	60	0.1052
5.	Input(32), Hidden(16)	80	0.1013

6.	Input(16), Hidden(16)	80	0.0986
7.	Input(16), Hidden(16) , Hidden(8)	80	0.0976
8.	Input(16), Hidden(16), Hidden(8), Hidden(4)	80	0.0961

The initial model yielded the least CV-MAE at 80 epochs. I further tested it with different hyper parameters. The best model turned out to be model 6, with an input layer of 16 nodes, followed by hidden layers of 16, 8 and 4 nodes in succession, trained at 80 epochs. I recompiled the best model and evaluated on the test set. The **test MSE and MAE are 0.0190 and 0.092** respectively.

```
#4 Final Model Evaluation
test_mse, test_mae = model.evaluate(x_test, y_test)
print(test_mse, test_mae)
```

```
398/398 [==============================] - 0s 304us/step
0.019025002320732304 0.09198509156703949
```

We learned that in the regression setting, the complicated models tend to generate more accurate models (less MAE) while too many epochs during training can also cause overfitting of the training fold.

References

1. UCI Repository – Communities and Crime Data set. https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime