ML - Coursework 2

Leo Unoki, Karan Obhrai, Jack Qiu, Alexander Goncharov

- **A description of your model, along with justification of your choices.**

The default model can be summarised as follows:

```
Sequential(
  (0): Linear(in_features=13, out_features=100, bias=True)
  (1): ReLU()
  (2): Linear(in_features=100, out_features=50, bias=True)
  (3): ReLU()
  (4): Linear(in_features=50, out_features=25, bias=True)
  (5): ReLU()
  (6): Linear(in_features=25, out_features=12, bias=True)
  (7): ReLU()
  (8): Linear(in_features=12, out_features=1, bias=True)
)
```

With default values:

- Adam Optimizer with learning rate 0.001
- Batch size: 4096
- Loss function: Mean Squared Error
- Epoch of 2000

The choice of the number of output features for each layer was fairly arbitrary, though they were guided by hyperparameter tuning later on. We reason that deeper models are generally better than shallow models because they provide more parameters for the model to use to estimate the relationship between the input features and output value.

We use the ReLU activation function because it has better convergence properties and is less likely to suffer from vanishing gradients than other activation functions discussed in lectures.

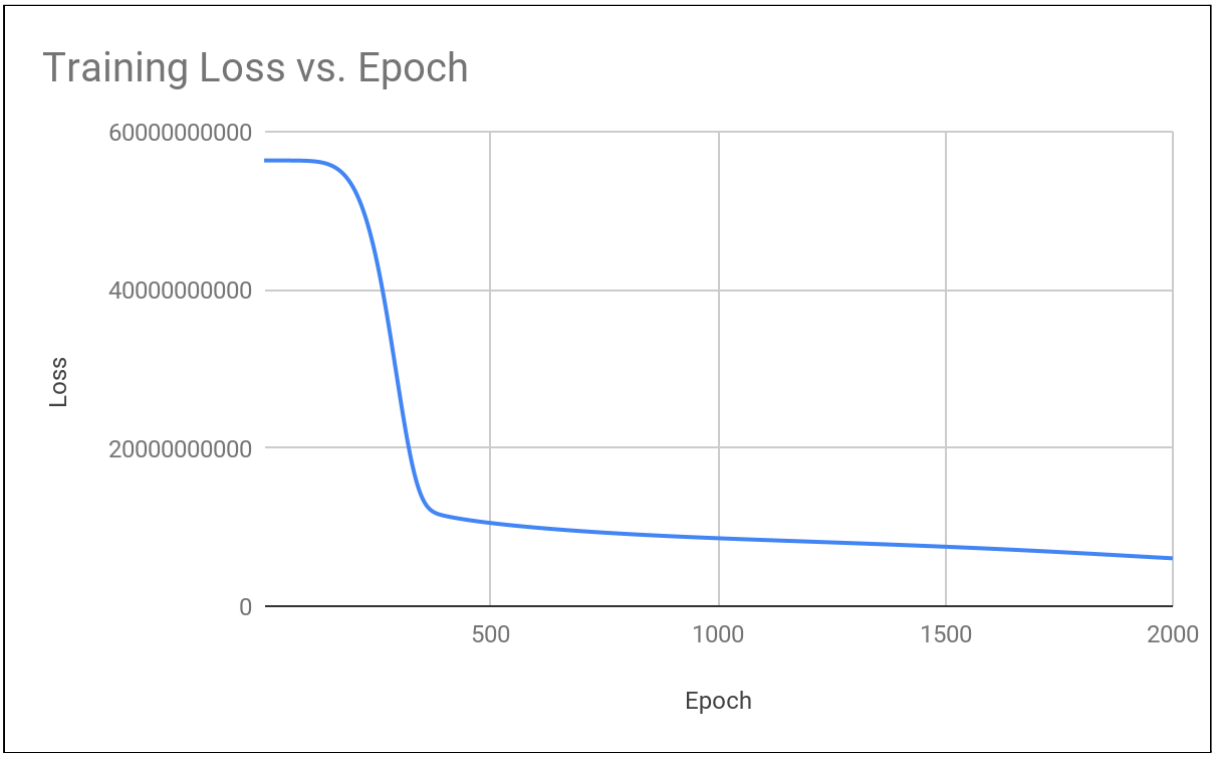
Choosing the right batch size is generally difficult. Smaller batch sizes generally train faster because there are more updates to model network parameters, whereas large batch sizes have better estimates of gradients because there are more samples involved. Though ultimately, whether or not our gradient is a good estimate of the true gradient really depends on whether our dataset is a good estimate of the true dataset. In a sense, our dataset can be thought of as a mini-dataset of the true dataset of housing prices and the actual features that determine the price.

As time is not a constraint as the model trains reasonably fast, we assume the housing dataset is a good reflection of the true dataset and thus reason that larger batch sizes will estimate gradients better and thus pick a batch size of 4096.

We set a relatively high epoch number to maximise learning. RMSE on the test dataset suggests that the model is not overfitting.

We use the mean squared error as a loss function due its suitability for regression problems.

We use the Adam optimizer because it's reasonably state-of-the-art, meaning it has relatively state-of-the-art performance of benchmark datasets. We note that Adam outperforms Stochastic Gradient Descent generally in convergence.



- **A description of the evaluation setup.**

Before training, we shuffle and split the dataset in a train/test split of 90/10. We both train and perform hyperparameter tuning with cross-validation on the train split and evaluate models on the test split.

We score based on RMSE error.

- **Information about the hyperparameter search you performed along with the resulting findings**

We hyperparameter search on the number of epochs, batch size and model layers.

```
param_grid = [{
    'layers': [[100, 50, 25, 1], [100, 50, 25, 12, 1], [200, 100, 50, 25, 1], [25, 12, 6, 1]],
    'batch_size': [512, 1024, 4096],
    'nb_epoch': [1000, 2000, 3000]
}]
```

We do not search on learning rate as we consistently find that a learning rate of 0.001 for the Adam optimizer yields the best results, this is corroborated by findings in the original Adam paper (https://arxiv.org/pdf/1412.6980.pdf)

We leverage sklearn's GridSearchCV which uses grid search together with cross-validation to find the most optimal hyperparameters.

We find that the most optimal parameters are:

- 100, 50, 25, 12, 1 corresponding to the model summary shown on first page
- Batch size of 512
- Number of epochs at 2000

With RMSE error of 53922.754828064135

We see that smaller batch sizes have better scores, though we note that the score improvement between 512 and 1024 was very marginal.

Ultimately, we could probably perform even finer-grained parameter searching, though we must note that that we've adjusted the parameter grid multiple times already, making sure that the optimal parameters were not the highest/lowest value in the category such that the potential addition of another higher/lower value could have been more optimal.

- **The final evaluation of your best model.**

We run our trained model on the test split to obtain a RMSE value of 71057.41957959987

Our best model was one found during hyperparameter search mentioned above.