Neural Network Model

Another model option we wanted to try out is the neural network, which is known for its ability to learn highly complex patterns in large volumes of data. While this option might be a bit overkill for our use case, prediction of the single taxi ride fare feature based on not *that* many input features, it was still interesting to see how it compares to the other models we have tried, and whether the substantially increased training time results in any kind of significant difference.

The model

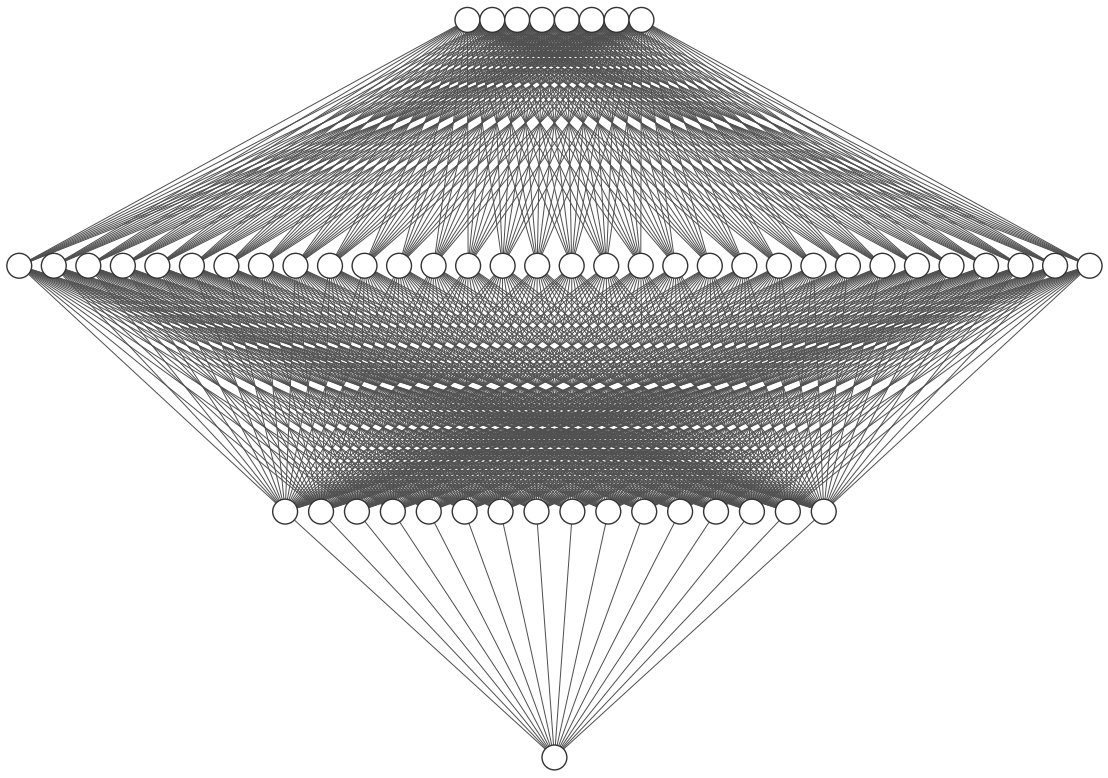
The neural network model is implemented using PyTorch with a feedforward architecture consisting of 4 linear layers:

- Input layer: 8 features (trip_distance, trip duration, pickup longitude,pickup latitude, hour, day_of_week, passenger_count, RateCodeID)
- Hidden layer 1: 32 neurons with ReLU activation and 10% dropout
- Hidden layer 2: 16 neurons with ReLU activation
- Output layer: 1 neuron for fare prediction

The network layers form a funnel shape, since we only want to regress a single value, so at each hidden layer, the data gets compacted more and more.

Just two hidden layers have been chosen to prevent overfitting. As an additional overfitting countermeasure, the model also uses a dropout layer, forcing the network handle broader patterns and not rely on single features.

To introduce non-linearity into the network, each hidden layer has its outputs passed through a ReLU function.



Batching

Batching is very important, both for training speed, and model performance.

We chose a batch size of 512, for which a single step provides enough data to broadly determine patterns in the dataset, but which is also not too large to stagnate learning. It also provides reasonable learning speed.

With this batch size, an epoch takes approximately 8.000 iterations.

Optimization algorithm

The model uses the Adam optimizer, which combines the benefits of the Momentum and RMSprop algorithms. It is computationally efficient, and has little memory requirements. We provide the optimizer with the MSE loss for each batch, which is well suited for regression tasks like this one.

Learning rate

The learning rate is a critical hyperparameter for every neural network. The higher the learning rate, the more agressively weights and biases get adapted each step. If the learning rate is too high, the model will likely not achieve a good acuraccy, since the weights cant be properly fine tuned. If the learning rate is too low, the model might not learn at all.

For proper training it is good to start out with a higher learning rate, and then gradually decrease it to allow fine tuning of the weights and biases. This has first been tried using an exponential learning rate scheduler, exponentially decreases the learning rate over time. This however introduced another hyperparameter to get right.

To combat this, we switched to the ReduceLROnPlateau learning rate scheduler. It can automatically detect when learning progress stagnates, and then lower the learning rate for more granular weight and bias tuning.

Model evaluation

The model's performance is evaluated for every 1000th batch, using the RMSE and L1 loss metrics.

After 10 training epochs, the neural network achieved strong performance on the taxi fare prediction task:

- RMSE: $0.89 (Root Mean Square Error)
- R^2 Score: 0.990 (99% of variance explained)
- Mean fare: $12.04

With an RMSE of just $0.89 on fares averaging $12.04, the model demonstrates high accuracy. The R^2 score of 0.990 indicates the model explains 99% of the variance in fare prices showing good predictive capability, despite the relatively simple architecture.