CSCI 561 homework3

Justin Zeng

April 22 2024

1 Introduction

This report details the hyperparameter tuning undertaken during the development of my neural network model. Each section describes a different set of hyperparameters, the rationale behind their selection, and the impact of various settings on model performance.

2 Experiment Setup

I started with building a baseline neural network with 2 layers hidden layers and 2 activation RELU layers with 512 and 256 as the parameters. I also ran with 100 batch size and 100 epochs as well as learning rate of 1. I used a softmax cross entropy loss for the loss function.

During data preprocessing, I checked different column's correlation to 'BEDS' and removed several unnecessary columns such as 'LONGITUDE', 'LATITUDE', and various address-related fields to focus on more relevant predictors. This is crucial for maintaining model performance by reducing noise and computational complexity. I also checked the different 'TYPE' of properties and found out that those listed as 'Pending' or 'Coming Soon' had low correlation to the number of 'BEDS' and thus filtered them out, ensuring that the model only trains on available and actionable data.

One of the key steps involves one-hot encoding the 'TYPE' column, which categorizes properties into different types such as condos, houses, etc. This transformation is essential for treating categorical data as numerical inputs for the model. After encoding, the train and test datasets are aligned to ensure they have the same features, filling missing columns with zeros where the training set had categories absent in the test data.

The missing numeric values are filled with the median of the respective column, a strategy that helps maintain a robust dataset without skewing it significantly. Additionally, the 'BEDS' column, which is the target variable, is separated from the features, allowing for clean datasets ready for model training and prediction.

The data is also normalized the data which involves scaling the features to have a mean of zero and a standard deviation of one and thus speeding up training.

3 Network Architecture

3.1 Number of Layers and Nodes per Layer

I experimented with 2 different set ups:

- **2 hidden layers and 2 activation layers:** This configuration led to the model achieving up to 0.2 validation accuracy, indicating under fitting and will need more nodes or more layers. I then tried increasing the nodes to 1024 and 512 but there were no significant results.
- **4 hidden layers and 4 activation layers:** This configuration led to the model achieving better validation accuracy than 2 hidden layers and 2 activation layers. But showed signs of overfitting with validation acurracies at 0.9 with only a 0.3 test accuracy. I then had to play around with the number of nodes. I first used 1024, 512, 256, and 128 which gave me consistent validation accuracies in the 0.3s. I then experiemented with minimizing the number gap by only removing the 1024 and replacing it with another 512. I've also experimented with using much lower number of nodes such as 64, 32, 32, 16 which brought the validation accuracy back to 0.2s. I ended up having 2 512 layers, 1 256 layer, and 1 64 layer which I experiemented with because it was closer to the number of class I was predicting (around 50ish bedrooms). This final configuration led to validation accuracies up to 0.5 and had similar test accuracy of 0.47.

3.2 Training split results

This is the results of each training split using the same hyper parameters as my final code.

	Split 1	**Split 2**	**Split 3**	**Split 4**	**Split 5**
Train Time	38.50s	26.85s	21.46s	30.63s	52.08s
Best Epoch	25	17	17	21	41
Train Loss	1.3272	1.4356	1.4591	1.4558	1.3304
Validation Loss	1.4379	1.5304	1.4207	1.4416	1.3616
Final Test Loss	1.5054	1.5659	1.5364	1.4927	1.3954
Train Accuracy	0.5083	0.4821	0.4723	0.4778	0.5019
Validation Accuracy	0.4921	0.4775	0.4945	0.4829	0.5015
Final Test Accuracy	0.4563	0.4599	0.4649	0.4726	0.4676

Table 1: Best Epoch, Losses, and Accuracy per Split

4 Learning Rate

- **Low learning rate:** I researched some common learning rates and found 0.0003 (3e-4) frequently recommended as 'the most optimal'. After implementing this adjustment, the model demonstrated significantly improved stability and I achieved validation accuracies of up to 0.51, which marked a substantial increase compared to previous configurations.
- **High learning rate:** Initially, I started with a learning rate of 1, which proved to be too large. This high learning rate caused the model's training process to be unstable, failing to converge and resulting in notably low test and validation accuracies, ranging between 0.15 to 0.20. The rapid updates to weights were too large, preventing the model from adjusting properly to minimize the loss function.

5 Activation Functions

- **RELU:** Initially, I selected the Rectified Linear Unit (RELU) activation function as it was the first activation function we were introduced to in our lecture. Early in the testing phase, I observed some problematic behaviors: the model's predictions were heavily biased, either predicting only 0s or 27s. Additionally, the output sometimes resulted in non-numerical values, specifically showing up as NaN (Not a Number), which indicated that the neurons were dying out, possibly due to the zero gradient problem in areas where the input is negative.
- **LeakyRELU:** To address these issues, I implemented the Leaky Rectified Linear Unit (LeakyRELU) activation function. This variant of RELU allows a small, non-zero gradient when the unit is inactive and the input is less than zero, thus mitigating the dying neuron problem. After incorporating LeakyRELU, the model showed improved stability in predictions across different data splits, and the instances of biased or non-numerical predictions were substantially reduced.

6 Epochs and Batch Sizes

In the process of tuning epochs and batch sizes, I encountered various challenges that influenced model training efficiency and accuracy. Initially, I experimented with a large batch size of 100 and a small batch size of 5. The larger batch size led to lower accuracies due to less noise in the gradient estimation, which can sometimes help escape local minima. However, the smaller batch size was taking too long to process, often exceeding 5 minutes per training session, which was not efficient.

Similarly, for epochs, a large setup of 200 epochs was impractical as it significantly increased training time without commensurate gains in model performance. A smaller count of 50 epochs seemed insufficient as it often triggered early stopping due to quick convergence. Ultimately, I settled on a batch size of 10 and 100 epochs as a balanced approach, improving training time while maintaining reasonable accuracy.

- **Small epochs and small batch size:** Early stopping often triggered. Final Test Accuracy: 0.4627, Test Loss: 1.5345 at Best Epoch: 6. Training completed in 30.40s. Epoch 6: Training Loss = 1.4542, Training Accuracy = 0.4692, Validation Loss = 1.5294, Validation Accuracy = 0.4696.
- **Large epochs and small batch size:** This combination provided slightly better performance with early stopping still a factor. Final Test Accuracy: 0.4804, Test Loss: 1.4111 at Best Epoch: 18. Training completed in 53.20s. Epoch 18: Training Loss = 1.2984, Training Accuracy = 0.4992, Validation Loss = 1.4884, Validation Accuracy = 0.4680.
- **Small epochs and large batch size:** This configuration was less effective, resulting in lower accuracy and higher losses. Epoch 50: Training Loss = 1.8151, Training Accuracy = 0.4500, Validation Loss = 1.8521, Validation Accuracy = 0.4072. Final Test Accuracy: 0.4339, Test Loss: 1.8318 at Best Epoch: 50. Training completed in 11.63s.
- **Large epochs and large batch size:** Also resulted in early stopping. Final Test Accuracy: 0.4427, Test Loss: 1.5977 at Best Epoch: 70. Training completed in 21.20s. Epoch 70: Training Loss = 1.5411, Training Accuracy = 0.4641, Validation Loss = 1.6134, Validation Accuracy = 0.4587.

7 Conclusion

Throughout the hyperparameter tuning process, I tested various configurations to determine their effect on model performance.

- **Activation Functions:** The switch from RELU to LeakyRELU significantly improved model stability and performance. While RELU led to problems like dying neurons and biased predictions (only predicting specific values like 0s or 27s), LeakyRELU helped in maintaining a slight activity in the neurons when the input was less than zero, thereby providing better performance and less biased outputs.
- **Learning Rates:** A high initial learning rate of 1 caused the model to fail in converging, resulting in low accuracies between 0.15 and 0.2. Reducing the learning rate to 0.0003 led to a marked improvement in both convergence and accuracy, achieving validation accuracies up to 0.51. This underscored the importance of choosing an optimal learning rate to balance the speed of convergence with the stability of the training process.
- **Epochs and Batch Sizes:** A balanced approach of using medium-sized batch sizes (10) and moderate epoch counts (100) offered the best performance. Smaller batch sizes, although providing more frequent updates, took excessively long and sometimes resulted in early stopping without sufficient training. Larger batch sizes and epoch counts either led to rapid, suboptimal convergence or increased computational load without proportional gains in performance.
- **2 Hidden Layers and 2 Activation Layers:** Achieved a maximum validation accuracy of 0.2, indicating underfitting. Increasing node numbers to 1024 and 512 did not yield significant improvements.
- **4 Hidden Layers and 4 Activation Layers:** Initially displayed high validation accuracies (0.9), but suffered from severe overfitting as evidenced by a much lower test accuracy (0.3). A variety of node configurations were tested, with a final setup of two 512 layers, one 256 layer, and one 64 layer, which achieved balanced validation and test accuracies of 0.5 and 0.47, respectively, providing a good fit without extreme overfitting or underfitting.

Overall, the best outcomes were achieved with the LeakyRELU activation function, a learning rate of 0.0003, and a balanced epoch-to-batch size ratio. These settings provided an optimal mix of efficiency, accuracy, and training stability, indicating that careful tuning of hyperparameters is crucial to maximizing model performance.