**Hyperparameter Tuning and Model Selection Report**

**Training Pipeline Overview**

1. **Data Loading and Preprocessing:**

The raw dataset presented a blend of numerical and categorical data. The preprocessing steps taken were as follows:

* **Normalization:** Numerical features such as PROPERTYSQFT were normalized to have a mean of zero and a standard deviation of one.
* **One-Hot Encoding:** Categorical features with a limited set of options, like TYPE and STATE, were one-hot encoded to convert them into a binary matrix representation.
* **Category Mapping:** For categorical data, mappings were created to maintain consistency between training and testing datasets. This approach ensured that all input features were on a comparable scale and in a format conducive to neural network training.

1. **Parameter Initialization:**

I initialize my neural network's weights with He initialization, which is well-suited for the ReLU activations used in my network. The biases are initialized to zero.

1. **Forward Propagation Loop:**

For the forward propagation, I've implemented a loop that processes the input through multiple layers, applying a ReLU activation function after each linear transformation. I handle the final layer differently, applying no activation function to obtain the raw scores.

1. **Cost Calculation:**

I calculate the cost of predictions using a Mean Squared Error approach, with an added L2 regularization term that helps prevent overfitting by penalizing large weights.

1. **Backpropagation:**

In backpropagation, I compute the gradients by moving backward through the network. Here, I ensure that the changes are proportional to the errors.

1. **Gradient Management:**

To manage exploding gradients, I've incorporated a gradient clipping step which bounds the gradients to a predefined threshold.

1. **Parameter Update Routine:**

My update routine adjusts the network's parameters, where I decrease them proportional to the gradient and the learning rate, ensuring that the network learns at a steady pace.

1. **Training with Mini-batches:**

I employed mini-batch training to refine my model. Mini-batches strike a balance between the computational efficiency of stochastic gradient descent and the stability of batch gradient descent, leading to faster convergence and better generalization.

1. **Training Iterations:**

I iterate over the training process 1000 times, feeding forward the input data and adjusting the network weights through backpropagation.

1. **Prediction Generation:**

Post-training, I use the trained parameters to predict outcomes on the test set. These predictions are then rescaled to ensure they match the true label scale.

**Hyperparameter Experiments**

**Experiment 1: Basic Network Configuration**

* **Learning Rate**: 0.01
* **Layers**: [Input - 64 - 32 - Output]
* **Activation**: ReLU
* **Regularization Lambda**: 0.01

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 35.900 |
| 2 | 36.120 |
| 3 | 36.800 |
| 4 | 36.540 |
| 5 | 35.870 |

**Experiment 2: Deep Network with Low Learning Rate**

* **Learning Rate**: 0.001
* **Layers**: [Input - 128 - 64 - 32 - Output]
* **Activation**: ReLU
* **Regularization Lambda**: 0.01

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 40.760 |
| 2 | 37.530 |
| 3 | 39.150 |
| 4 | 39.340 |
| 5 | 36.980 |

**Experiment 3: Increased Neurons in Hidden Layers**

* **Learning Rate**: 0.001
* **Layers**: [Input - 256 - 128 - Output]
* **Activation**: ReLU
* **Regularization Lambda**: 0.01

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 38.780 |
| 2 | 39.290 |
| 3 | 39.670 |
| 4 | 39.520 |
| 5 | 38.940 |

**Experiment 4: High Learning Rate with Regularization**

* **Learning Rate**: 0.005
* **Layers**: [Input - 128 - 64 - Output]
* **Activation**: Sigmoid
* **Regularization Lambda**: 0.02

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 35.630 |
| 2 | 35.450 |
| 3 | 38.780 |
| 4 | 35.290 |
| 5 | 35.170 |

**Experiment 5: Simplified Network with Increased Regularization**

* **Learning Rate**: 0.01
* **Layers**: [Input - 64 - Output]
* **Activation**: ReLU
* **Regularization Lambda**: 0.05

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 37.290 |
| 2 | 37.540 |
| 3 | 36.610 |
| 4 | 37.450 |
| 5 | 37.320 |

**Experiment 6: Optimized Network with Balanced Configuration**

* **Learning Rate:** 0.001
* **Layers:** [Input - 32 - 16 - Output]
* **Activation:** ReLU
* **Regularization Lambda:** 0.01

**Split Accuracy (%)**

| **Split** | **Accuracy (%)** |
| --- | --- |
| 1 | 40.667 |
| 2 | 36.870 |
| 3 | 43.105 |
| 4 | 38.327 |
| 5 | 41.440 |

The exploration of the network's architecture revealed several key insights:

* **Experiment 1:** A network with fewer neurons and a higher learning rate was quick to train but fell short in accuracy.
* **Experiment 2:** The addition of another hidden layer enriched the model's complexity, enabling it to capture more nuanced patterns in the data.
* **Experiment 3:** A network with more neurons per layer surfaced as the top performer, indicating that a broader network architecture was more adept at handling the dataset.
* **Experiment 4:** Switching to the sigmoid activation function led to a dip in accuracy, signifying that ReLU surpassed sigmoid for this specific problem.
* **Experiment 5:** Streamlining the network to a single hidden layer proved inadequate, suggesting that the intricacies of the data demanded a more complex model.
* **Experiment 6**: It highlights the effectiveness of a balanced approach, combining sufficient model complexity with optimal learning parameters. The network was able to capture complex patterns without overfitting, as indicated by the relative high accuracies.

Informed by these insights, the architecture from Experiment 6 was crowned as the final model, featuring a learning rate of 0.001 and ReLU as the activation function. This configuration struck an optimal balance between model intricacy and empirical accuracy.

**Final Model Configuration**

The chosen model is characterized by:

* **Number of Hidden Layers:** 2
* **Neurons per Layer:** 32 in the first hidden layer and 16 in the second
* **Learning Rate:** 0.001
* **Activation Function:** ReLU
* **Regularization (Lambda):** 0.01