Model Architecture and Methodology

Objective: The goal of the model is to predict the median house price in the California housing market. The model's effectiveness is measured using metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the R-squared (R2) value.

Architecture: The first layer consists of the number of features in the dataset which is 13. It consists of two hidden layers with 60 and 80 neurons respectively. Finally, the last layer has only 1 neuron as it predicts a numerical value about the price of a house. This architecture was chosen to provide sufficient complexity for capturing the underlying patterns in the data without overfitting.

Activation Functions:

- ➢ First Hidden Layer: Sigmoid activation function. Sigmoid was selected for its ability to handle non-linear relationships in the data, which is common in housing prices.
- ➢ Second Hidden Layer: ReLU (Rectified Linear Unit) activation function. ReLU was chosen as the second layer to introduce non-linearity while remaining computationally efficient and helping in mitigating the vanishing gradient problem.
- ➢ Last Layer : Linear as it predicts a numerical value about the price of a house.

Optimizer: The Adam optimizer is used with a learning rate of 0.005. Adam is known for its efficiency in computing gradients and its adaptability in adjusting learning rates, making it suitable for datasets with varying features like housing data.

Epochs: 1500 epochs to ensure that the model has ample opportunity to learn from the data.

Batch Size: A batch size of 100, providing a balance between computational efficiency and the ability to generalize.

The model was refined iteratively, testing various combinations of layers, neurons, and activation functions. This approach ensures that the final model is tailored to the specific characteristics of the California housing dataset.

Model Evaluation

Metrics for Evaluation: The model's performance was evaluated using MSE, RMSE, MAE, and R2. This selection was made based on various considerations. The MSE and RMSE provide information on the average model prediction error. The MAE offers a clear measure of prediction accuracy. Lastly, the R2 indicates the proportion of variance in the dependent variable that is predictable from the independent variables, helping in understanding the model's explanatory power; the closer it is to 1, the better the model is.

Balance Between Complexity and Generalization: The chosen architecture and hyperparameters strike a balance between a model that is complex enough to capture key patterns in the data and one that can generalize well to unseen data.

Adaptability and Efficiency: The use of Adam as an optimizer and the specific learning rate was decided based on the model's adaptability to different types of data and computational efficiency, crucial for handling large datasets like housing prices.

Conclusion

This neural network model is a result of careful consideration and iterative testing. It is designed to effectively predict median house prices in the California market by the given dataset but also in more general datasets as it will be explained later on.

Evaluation Setup

To evaluate our neural network, we begin by splitting the dataset into a training, evaluation, and testing set. The training set makes up 80% of the whole dataset and the remaining 20% is split equally between validation and testing. The models were trained on the training set and then the evaluation metrics were computed for the validation test and compared with each other to select the best hyperparameters. Then the final selected model was assessed on its ability to predict the labels of the test set which was in no way involved in the making of the model. While these sets are created randomly, we defined a seed which shuffles the dataset in the same manner for each neural network. This enables all different neural networks to be trained and evaluated on the same data, making our comparisons of the hyperparameters impartial. For the evaluation phase, the metrics outlined earlier are used. While we acknowledge that these metrics exhibit concurrent fluctuations, we use them collectively to provide a more generalized evaluation.

Additionally, we filled the empty values of the column 'total_bedroom' with the median of the training set's median of that column and used one hot encoding for the categorical column of 'ocean_proximity'. Furthermore, we normalized our features' values with the Standard Scaler of sklearn. This practice facilitates enhanced learning by the neural network, helps mitigate the possibility of overfitting, and can help the model generalize better, making the training process more efficient and reliable. Further, for the gradient descent mechanism, we applied the mini-batch gradient descent. This method is a common practice and can help minimize noise.

To begin our evaluation, we ran the neural network using various values for the hyper parameters to get an initial understanding of the effect of each hyper parameter on the results of our neural network, as shown in Fig. 1. These initial tests helped us to gain a better understanding of which hyper parameters needed to be reviewed and tested in more detail. From this data, we concluded that we should use a relatively small neural network, 1 to 3 layers, and that 1000-1500 epochs would be enough to train it.

layers	neurons	activation funcs	epochs	batch size	learning rate	final_train (MSE)	final_test(MSE)	rmse	mae	r2
2	10, 10	sigmoid, relu	50	100	0.01	7107620960	7167410233	84660.5589	58076.77418	0.4664548751
2	50, 50	sigmoid, relu	50	100	0.01	3558833855	3639567990	60328.83216	41440.75708	0.7290689811
2	10, 10	sigmoid, relu	100	100	0.01	4625869479	4633904344	68072.7871	48786.21756	0.6550501518
2	50, 50	sigmoid, relu	100	100	0.01	3210398408	3334892756	57748.53034	39207.83282	0.7517491375
2	70, 70	sigmoid, relu	100	100	0.01	3170658634	3294404080	57396.89957	38949.35441	0.7547631321
2	70, 70	sigmoid, relu	150	100	0.01	2948753927	3075850436	55460.35013	37265.51545	0.7710323601
2	100, 100	sigmoid, relu	100	100	0.01	3064134951	3204910898	56611.93247	37980.14402	0.7614250434
2	100,100	sigmoid, relu	150	100	0.01	2872488315	3028197856	55029.06374	36920.75011	0.7745796388
2	100,100	sigmoid, relu	175	100	0.01	2733407800	2933362514	54160.52543	36267.76504	0.7816392228
2	100,100	sigmoid, relu	160	100	0.01	2797440992	3010503471	54868.0551	36602.94883	0.7758968165
2	150, 80	sigmoid, relu	130	100	0.01	2550655540	2899534895	53847.32951	35908.03905	0.7841573654
2	180, 60	sigmoid, relu	150	100	0.01	2794518825	2984121058	54627.10919	36460.70012	0.7778607348
2	180, 60	sigmoid, relu	500	100	0.01	2168970509	2831768675	53214.36531	36116.7486	0.7892019122
2	180, 60	sigmoid, relu	750	100	0.01	2001432346	2803614750	52949.17138	35550.26018	0.7912977026
2	180, 60	sigmoid, relu	1000	100	0.01	1971558688	2895885560	53813.43289	36789.11231	0.7844290235
2	180, 60	sigmoid, relu	1500	100	0.01	1832201611	2937576084	54199.41036	37157.26582	0.7813255629
2	400, 300	sigmoid, relu	1500	100	0.01	916323076.4	4652878872	68212.01413	43873.81317	0.6536376797

Fig. 1: Preliminary Test Runs

Hyperparameter Tuning

We then created many methods in the functionRegressorHyperparameterSearch that tune the hyperparameters to find the best model. Optimal performance was measured using mainly mean squared error (MSE), alongside the other metrics explained above. The first layer which took the input always had 13 neurons (1 for each variable after the categorical variable was transformed), and the final layer always had 1 neuron, both having the linear activation function. All of our tuning was about the hidden layers between the input and output layer. Firstly, multiple combinations of layers and neurons were tested to find an optimal architecture for the model, as displayed in Fig. 2, and we concluded that a model with 2 layers and either [50,50], [60,180], and [180,60] performed the best. It seems that 3 layers is a lot for our kind of problem and can result in an overfitting phenomenon since they need a lot more epochs. Also, 1 layer with a huge number of neurons performs worse than the 2 layers with a relatively small number of neurons.

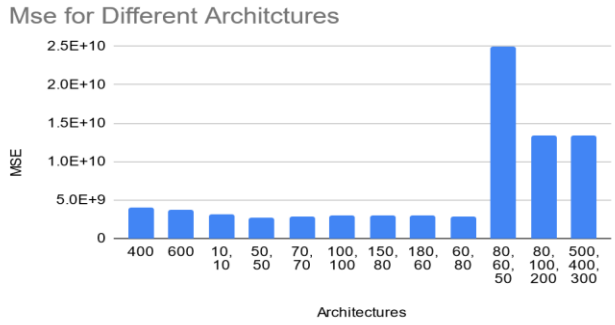

Fig. 2: MSE for Different Architectures

We then tested varying activation functions for these three architectures, as shown in Fig. 3, and found that the ones that performed best were the activation functions of [sigmoid, relu] and [relu, relu], which had very similar MSE values. In all the different models we derived the same conclusions, that the tanh activation function is not a proper solution to our problem, that sigmoid can only be used if it is followed by the relu and that using relu in both layers can produce great results.

[50,50] [60, 80] [180, 60]

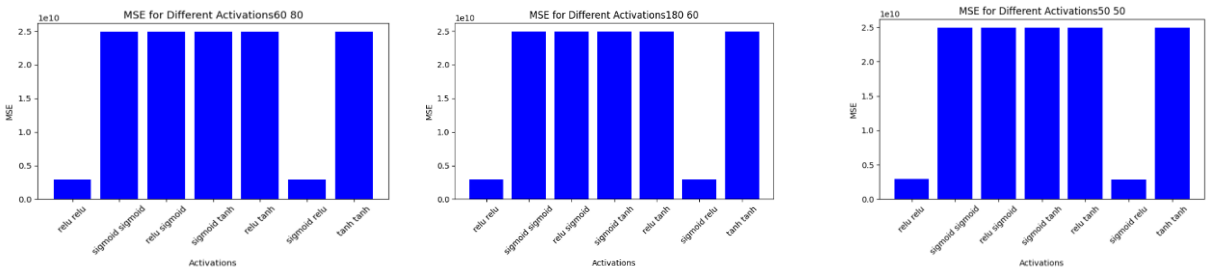

Fig. 3: Best performing models - Activation Functions

Using these architectures and activation functions, we tested varying learning rates, as seen in Fig. 4. The rates that resulted in the lowest MSEs in all the different networks were 0.01 and 0.005 as these values are not extremely low which can cause slow convergence, nor that they are extremely high that can lead to overshooting, so the network can learn and generalize in a very stable way.

[50, 50] [Sigmoid, ReLU] [60,80] [Sigmoid, ReLU] [180,60] [ReLU, ReLU]

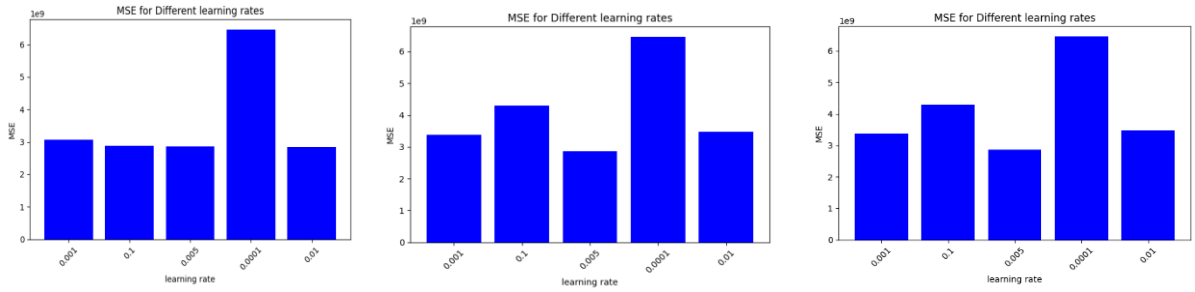

Fig. 4: Best performing models - Learning Rate

Lastly, we tested combinations of the previously mentioned hyperparameters against batch size, as shown in Fig. 5, finding that a batch size of 100 resulted in the model with the lowest MSE for all the different neural networks.

[50, 50] [Sigmoid, ReLU] LR=0.01 [60,80] [Sigmoid, ReLU] LR=0.005 [180,60] [ReLU,ReLU] LR=0.005

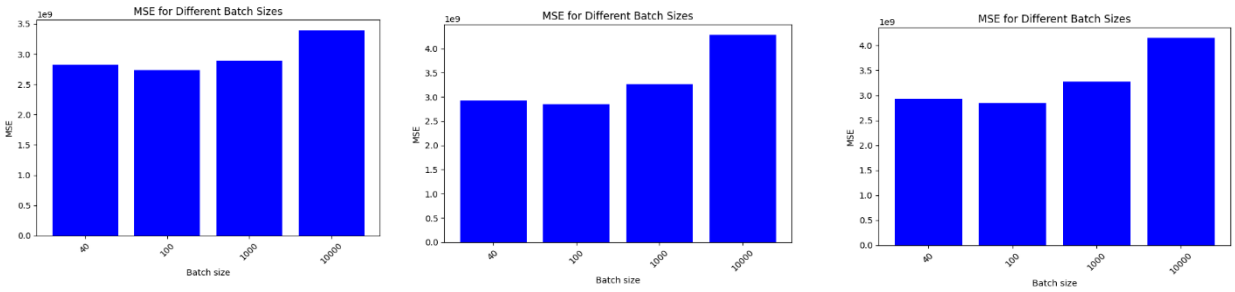

Fig. 5: Best performing models: Batch Size

From this whole process, we concluded into 3 different neural networks, and we ran k-fold evaluation, with the training and testing dataset, across these networks to find the best-performing one. This approach accounts for the random initialization of neural network weights and the stochastic nature of training and validation data, providing reliable means to choose hyperparameter sets independently of inherent randomness. The best neural network comprised

of 2 layers of 60 and 80 neurons, with activation functions sigmoid and relu, respectively, with a batch size of 100 and a learning rate of 0.005.

Finally, this network was trained for a different number of epochs ranging from 0 to 1500 and the RMSE was used to evaluate the model as displayed in Fig 6. From now on we used RMSE as the "Labts" evaluation uses this metric. We can observe that the model is not being overfitted on the training data, as in both datasets and the RMSE scores are decreasing, providing a good, generalized model that can achieve great predictions in both trained and validation data. The network has effectively converged, as evidenced by the negligible disparity in RMSE for the two datasets. For the number of epochs, we chose the value of 1100, as it is a local minimum for the validation RMSE, and it provides a great model with a smaller number of epochs which needs less training time.

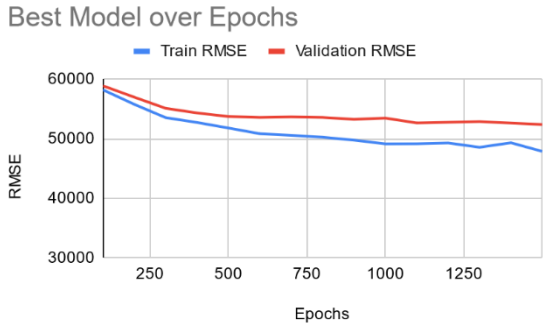

Fig 6: Best Neural Network Model over Epochs

Final Evaluation of Best Model

After this hyperparameter tuning we concluded to our final model, which has 2 hidden layers of 60 and 80 neurons, with activation functions of sigmoid and relu, with a batch size of 100, a learning rate of 0.005 and trained for 1100 epochs. In order to ascertain its optimal performance, the model underwent evaluation on a distinct testing set that remained entirely untouched during the entirety of the hyperparameter tuning process. In the unseen data, it evaluates an RMSE of 55002, MAE of 36967, r2 score of 0.78 and MSE of 3025222633, so it is obvious that the procedure was generic and resulted in a great neural network that can effectively work well on completely unseen data too. Later we ran it on "LabTS" and had an RMSE of 49704, which is a little more than half of the suggested threshold.