Imperial College London

Department Of Computing
Introduction To Machine Learning

Coursework 2: Artificial Neural Networks

---

team members:

salim al-wahaibi

alicia law

marcos-antonios charalambous

wei jie chua

Contents

1 Introduction

This report explores the architecture and evaluation of a neural network model trained on the California House Prices Dataset. The report describes the pre-processing done on the dataset, the hyperparameter tuning of the neural network and the final architecture of the best performing neural network found. The report also explores the evaluation setup used to evaluate and approximate the generalised error of the model.

2 Model Description

This section describes the implementation of a neural network architecture for regression. Firstly, the data is split into training and test sets, which is then followed by their respective pipeline pre-processing, as shown in Section 2.1 and 2.2. The model initialisation via the constructor of the neural network is in Section 2.3 and finally the model training process is described in Section 2.4.

2.1 Pre-processing Training Dataset

Pre-processing of input data (x) comprises of the following steps:

1. **Splitting the data**
   The input data was split into categorical data and numerical data.
2. **Handling missing data**
   All missing numerical data were replaced with the mean values of their respective columns.
   All missing textual data were replaced with the modes of their respective columns.
   These values used were determined according to the training data and stored as attributes of the Regressor class so that the same values could be used to pre-process the test set.
3. **Min-max normalisation**
   Min-max normalisation was applied such that all input datasets ranged from [0,1]. The transformation values used for this normalisation were also stored as attributes of the Regressor class so that the same values could be used to normalise the test set.

The target data (y) was also normalised during pre-processing. This was important as it contained a large range of values, ranging from [14999, 500001] with a mean of 207251. This had many implications on model training:

1. As the Mean Squared Error (MSE) loss function was used, if y was not normalised, any small percentage deviation in price predictions would result in large losses due to the squaring effect. This was observed to reduce the efficiency of gradient descent, with significantly larger epochs and learning rates required to reach an optimal solution.
2. It was important to assess the effectiveness of different activation functions, such as Sigmoid and Tanh, during hyperparameter tuning. For these activation functions to be effective, output data must be limited to [-1,1] and [0,1] for Tanh and Sigmoid, respectively.

Hence, min-max normalisation was also applied to target data (y) such that it ranged from [0,1].

2.2 Pre-processing Test Dataset

The test dataset underwent the same series of pre-processing steps as listed in Section 2.1. However, all missing data in the test dataset were replaced with the means/modes of the training dataset computed during pre-processing. Similarly, the min-max normalisation of the input and target test data were carried out using the same transformation values used to normalise the training dataset. This was done to ensure the test dataset remains unbiased.

2.3 Constructor

The constructor of the neural network initialises the *PyTorch* network layers using the parameters supplied {hidden_neurons, hidden_activations, output_activation}. Within the network, an ADAM loss optimiser is initialised with a Mean Squared Error (MSE) loss function. Other parameters included in the initialisation are the number of epochs that determine the optimisation's stopping criterion and the learning rate used. These parameters are supplied via the constructor to be tuned via the hyperparameter search in Section 4.

2.4 Model Training

The model's training process starts by pre-processing the supplied training data to handle any missing data and for data normalisation. At each epoch, the output is computed by the network. The loss associated with the output is computed by comparing the prediction along with its true value for each training data. This is followed by back-propagation, whereby the gradients from the output layer are computed up until the input layer through using chain gradients $\delta F_{loss}/\delta weights$. These are then used to tune and update the weights for the next round of iterations $W_{new} = W_{old} - \sigma * \delta F_{loss}/\delta weights$. The above process is then repeated for the number of epochs specified.

3 Evaluation Setup

Section 3.1 describes the prediction method and Section 3.2 describes its evaluation.

3.1 Prediction

The prediction method takes in a series of test data. The input test data is first pre-processed according to the steps listed in Section 2.2. This is followed by forward computation through the network to get the predicted output. As the predicted output would have been normalised to an approximate range of [0,1], an inverse transformation is applied to undo the scaling, based on the training data's feature ranges to provide the final housing price prediction.

3.2 Evaluation

To evaluate the trained neural network and approximate the true error, we compute the R^2 (coefficient of determination) regression score, taking the predictions made using the test data (Section

3.1) and comparing them with the true test output values. The R^2 value measures how closely the observed outcomes are predicted by the model, with an R^2 score of 1 representing a perfect fit.

4 Hyperparameter Search

Hyperparameter search was conducted to obtain the best combination of hyperparameters that would optimise the performance of the neural network. The hyperparameters assessed and methodologies used are described in Section 4.1, and the findings are shown in Section 4.2.

4.1 Methodology

The following parameters were explored:

1. Optimiser: AdaGrad, SGD, RMSProp and Adam
2. Activation Functions: ReLU, Tanh, Sigmoid
3. Learning Rate: 0.01 - 0.5
4. Epoch: 1000 - 3000
5. Number of layers: 2 - 50
6. Number of neurons: 2 - 30, randomly generated/start from max to min number of neurons

Hyperparameter tuning was conducted based on the following guidelines:

1. **Combined grid search and random search**
   Grid search was used to iterate across a range of learning rates (lr) and the number of layers(n) with a 'divide and conquer' strategy. Initially, the search was carried out over the full range listed above, using large intervals. The search was then repeated a number of times, each time with a narrower search range and smaller intervals as we converge to the optimal solution.

   For each combination of lr and n, the number of neurons and activation function assigned to each layer were assigned randomly, rather than using grid search to reduce computational cost. The results from our final search can be seen in the table in Appendix A.

2. **Decreasing number of neurons with layer depth**
   This is a common trend when designing feedforward neural networks and was used as a strategy for neuron assignment in our network due to observable model improvement when doing so. This is because initial layers take in noisy data and require more neurons to encode feature information properly. However, as the network progressively approaches the output layer, it builds a more generalised representation of the inputs, removing noise while consolidating important features. Hence, lesser neurons were required with depth.

3. **Final Selection**
   The final selection takes the architecture that yields the highest R^2 Score closest to 1.

4. **Prevent overfitting**
   Many hyperparameters, such as the epoch, number of layers and number of neurons, could result in overfitting. To prevent this, the validation and test datasets were always compared to ensure no significant deviation in performance was observed.

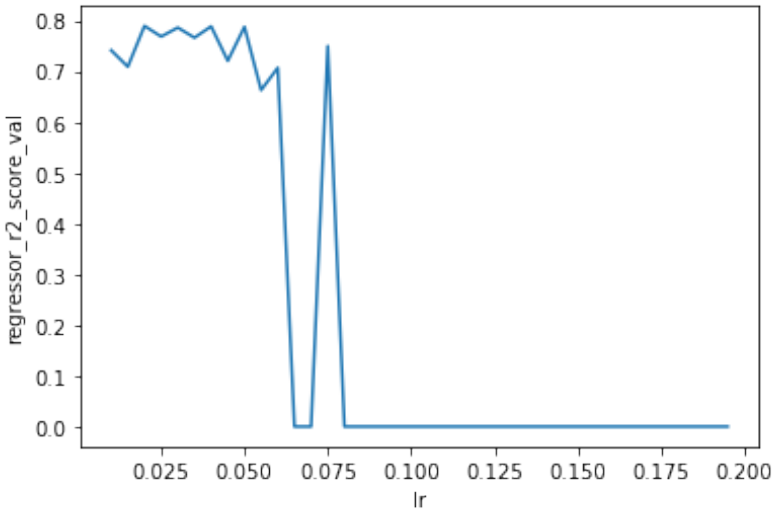
4.2 Findings

The findings observed during hyperparameter search for the optimal architecture are:

1. **Optimiser**
   The best optimisers found were AdaGrad and Adam, giving consistently higher R^2 scores.

2. **Activation Function**
   ReLU was observed to provide significantly better results over Tanh and Sigmoid.

3. **Learning Rate**
   The best learning rate is found to be around 0.03, which has an R^2 score of 78 %. R^2 decreases dramatically as we increase the learning rate.



4. **Number of layers and epochs**
   At 1000 epoch, increasing the number of layers beyond three resulted in reduced performance. However, as we increase the epoch to 3000, the R^2 score can be improved with a deeper layer, up until eight layers of neurons.

5. **Number of neurons**
   The number of neurons is found to observe a very interesting phenomenon. The model performs best when the initial layers have a significantly larger number of neurons (e.g. 30), which slowly reduces to 10 neurons at the later layers. Completely randomising the number of neurons between layers resulted in poor performance.

5 Final Evaluation and Architecture

5.1 Best Model Architecture

The architecture and evaluation of our best model along with the hyperprameters after the hyper prameter search are:

- Neural Network:
  - Number of neurons: Hidden layer= [27, 21, 19, 19], Output layer= 1
  - Number of layers: 1 input, 4 hidden, 1 output
  - Activation function: ['relu', 'relu', 'relu', 'relu'], ['identity']
- Epochs: 3000
- Learning Rate: 0.02
- Optimisor: Adam is used as optimisor of the neural network

The summary of the model is shown below:

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
TorchNeuralNetwork                       --
    Sequential    : 1-1                      --
            Linear    : 2-1                    336
            ReLU    : 2-2                      --
            Linear    : 2-3                    450
            ReLU    : 2-4                      --
            Linear    : 2-5                    304
            ReLU    : 2-6                      --
            Linear    : 2-7                    153
            ReLU    : 2-8                      --
            Linear    : 2-9                    40
            ReLU    : 2-10                     --
            Linear    : 2-11                   10
            ReLU    : 2-12                     --
            Linear    : 2-13                   3
            Identity    : 2-14                 --
==========================================================================
Total params: 1,296
Trainable params: 1,296
Non-trainable params: 0
==========================================================================
```

5.2 Best Model Evaluation

- R^2 Score: 0.7834
- RMSE: 54,483

Appendix

Appendix A: Hyperparameter Search Results

	num_of_layers	hidden_activations	hidden_neurons	lr	RMSE train	RMSE val	RMSE test	R2 train	R2 val	R2 test	best_model
model0	2	['relu', 'relu', 'identity']	[27, 23, 1]	0.02	54795.90372	53159.21322	57337.0352	0.777301787	0.778353603	0.760151199	
model1	2	['relu', 'relu', 'identity']	[24, 19, 1]	0.02	57083.04664	56347.5051	60654.24394	0.758323271	0.750969254	0.731595697	
model2	2	['relu', 'relu', 'identity']	[23, 20, 1]	0.02	50659.43514	51320.22905	54432.65049	0.809655106	0.793423574	0.783834665	
model3	3	['relu', 'relu', 'relu', 'identity']	[28, 20, 16, 1]	0.02	58376.9091	56139.94139	60083.09235	0.747243264	0.752800552	0.736626764	
model4	3	['relu', 'relu', 'relu', 'identity']	[28, 25, 21, 1]	0.02	51874.26805	52986.30364	55792.84794	0.800416558	0.779793145	0.772896332	
model5	3	['relu', 'relu', 'relu', 'identity']	[21, 16, 8, 1]	0.02	54339.12891	53069.15957	56305.0124	0.780999105	0.779103921	0.768707685	
model6	4	['relu', 'relu', 'relu', 'relu', 'identity']	[23, 22, 21, 14, 1]	0.02	54467.6302	52723.86937	57256.43123	0.779962093	0.781969055	0.76082508	
model7	4	['relu', 'relu', 'relu', 'relu', 'identity']	[30, 29, 25, 18, 1]	0.02	50550.23949	52938.62781	56975.65924	0.810474793	0.78018924	0.763165043	
model8	4	['relu', 'relu', 'relu', 'relu', 'identity']	[27, 21, 19, 19, 1]	0.02	48821.77335	50386.24368	54483.4257	0.823214091	0.800874191	0.783431195	yes
model9	5	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[22, 22, 22, 19, 15, 1]	0.02	52870.77565	53701.93737	57279.50493	0.792674888	0.773804743	0.760632272	
model10	5	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[23, 15, 7, 2, 2, 1]	0.02	116115.3746	112945.1792	117083.8573	-5.52891E-14	-0.000550434	-0.000140767	
model11	5	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[25, 22, 19, 19, 11, 1]	0.02	52707.2432	53245.23368	57344.00952	0.793955442	0.777635701	0.760092846	
model12	6	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[30, 25, 19, 14, 9, 5, 1]	0.02	51338.47565	51853.61405	55221.72393	0.80451813	0.789107251	0.777522033	
model13	6	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[22, 22, 20, 18, 10, 6, 1]	0.02	53661.8641	53777.97086	57385.0575	0.786424193	0.773163776	0.759749263	
model14	6	['relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[24, 16, 7, 5, 5, 2, 1]	0.02	96885.00755	93609.8175	97980.44195	0.303800511	0.312699936	0.299600424	
model15	7	['relu', 'relu', 'relu', 'relu', 'relu', 'relu', 'identity']	[25, 20, 16, 11, 10, 3, 3, 1]	0.02	53519.0563	52152.0716	56972.74019	0.787559439	0.786672563	0.76318931	