**BITS F464**

**Machine Learning - Assignment 2**

**NEURAL NETWORK**

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Praneetha Vaddamanu** | **2017AAPS0319H** |

**Problem Statement and Methodology**

The given task is to implement a neural network from scratch for the provided dataset, so as to predict whether the price of a house will be above or below the market’s median price. Thus, it is a binary classification task. A neural network is capable of determining a highly nonlinear boundary between the positive and negative classes, in feature space, and is implemented by gradient descent. The model learns a set of weights, which are used to predict the class for a new test sample. To optimize the learning of these weights, experimentation with different layers, number of nodes in each layer, layer activations, weight initializations, learning rates, and dropout is performed. Further, the results are recorded and analysed for test samples.

**Dataset Description and Preprocessing**

It is required to test the implementation on the dataset provided in datasets/housepricedata.csv.

The dataset consists of 1461 examples, and has 10 features :

1. LotArea
2. OverallQual
3. OverallCond
4. TotalBsmtSF
5. FullBath
6. HalfBath
7. BedroomAbvGr
8. TotRmsAbvGrd
9. Fireplaces
10. GarageArea

We take the dataset with examples stacked as columns, and after shuffling the entire dataset, perform a train : validation : test split of 0.75 : 0.15 : 0.15. Thus, the training set has 1021 examples, the validation set has 220 examples and the test set, 219 examples.

We experiment with two types of feature scaling, namely *min-max normalisation* and *standardisation*, which are defined as follows:

**Min-Max Normalisation**

Min-max normalisation constrains the dataset to lie between -1 and 1. However, it does not bring all parameters to the same distribution. The cost function may be scaled to fit between -1 and 1, however, it is not made symmetric across all dimensions.

**Standardisation**

where **,** are the mean and standard deviation of **X**, respectively.

When performed for each feature, standardization makes sure that all parameters are from a distribution with mean 0 and variance 1. Thus, it makes the cost function symmetric across all dimensions, and enables gradient descent to proceed stably and smoothly.

For these reasons, we choose to employ standardisation for feature scaling.

In addition, the parameters **,** are calculated after splitting the dataset, and the train, validation and test sets are separately scaled using these parameters. This is done so that all preprocessing uses information from the training data set, as we require the test set to be untouched in order to give an unbiased estimate.

In addition to this initialisation, we apply scaling to our weights. Scaling is done so as to bring weights to a variance of 1 in each layer, so as to tackle the exploding gradient problem.

Scaling employed is the standard , sometimes called **Xavier initialisation.**

Here, the denominator inside the square root is the number of nodes in the previous layer.

**Metrics for Performance Evaluation**

**Accuracy**

The accuracy calculated as,

captures how many data points (as a fraction of the total number of data points fed to the algorithm) were classified correctly. It ranges between 0 and 100 %. A higher accuracy is indicative of better performance by the model.

Relying only on accuracy to judge model performance however, may, in some cases, be misleading. Hence, we also calculate the F-score in order to better evaluate the model’s performance.

**F1-score**

The F1-score is calculated as,

where

,

and

.

Recall captures how many data points were correctly classified as positive points, as a fraction of the total number of actual positive data points in the dataset.

Precision captures the number of data points correctly classified as positive points, as a fraction of the total number of points classified as positive by the model.

Ideally, we would want a high recall, as well as high precision, but often there is a tradeoff between the two. In order to capture correctly the effect of both recall and precision on the evaluation of performance of the model, we use the *F1-score* which is the harmonic mean of recall and precision.

The F1-score ranges between 0 and 1. A higher F1-score is indicative of better performance by the model.

**Loss function**

As this neural network is a binary classification model, we use the binary cross entropy loss function for gradient descent :

where , are the actual and predicted values of the target variable respectively, *n* is the number of training examples.

**Weight Initialization**

We experiment with uniform and gaussian initialization of weights.

For uniform initialization, we constrain the weights to lie between -1 and 1. Gaussian initialization returns random float values for the weights, such that their mean is 0 and standard deviation is 1. Thus, the weights are initialized such that they are not far from zero, both on the positive and negative side.

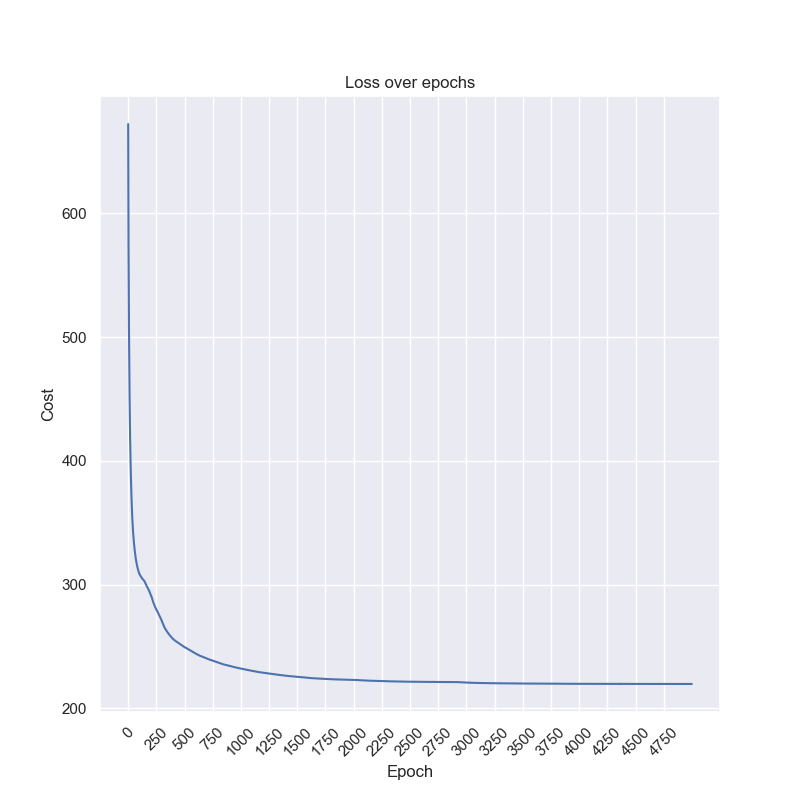
**Tuning the Neural Network**

1. **Configuration 1**

We first start with a basic neural network with a single hidden layer, with relu activation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Output** |
| **Number of Nodes** | **10** | **8** | **1** |
| **Activation** | **relu** | **relu** | **sigmoid** |

Running this for 5000 epochs of batch gradient descent (BGD), with a learning rate of 0.0005 seemed to give a nice loss curve which did not spike during training:



Throughout our analysis, we keep the number of epochs as 5000, and learning rate as 0.0005. This was determined after looking at the loss curves for several configurations of learning rates and layers. These values almost always never lead to unstable learning, except in the case of dropout (discussed later).

The results of this configuration are:

**Train accuracy = 91.77277179 %**

**Validation accuracy = 88.63636364 %**

**Val F1-Score = 0.89711934**

We attempt to further increase the validation accuracy, and try out more models. The results of the configurations are first presented, and they are later analysed.

1. **Configuration 2**

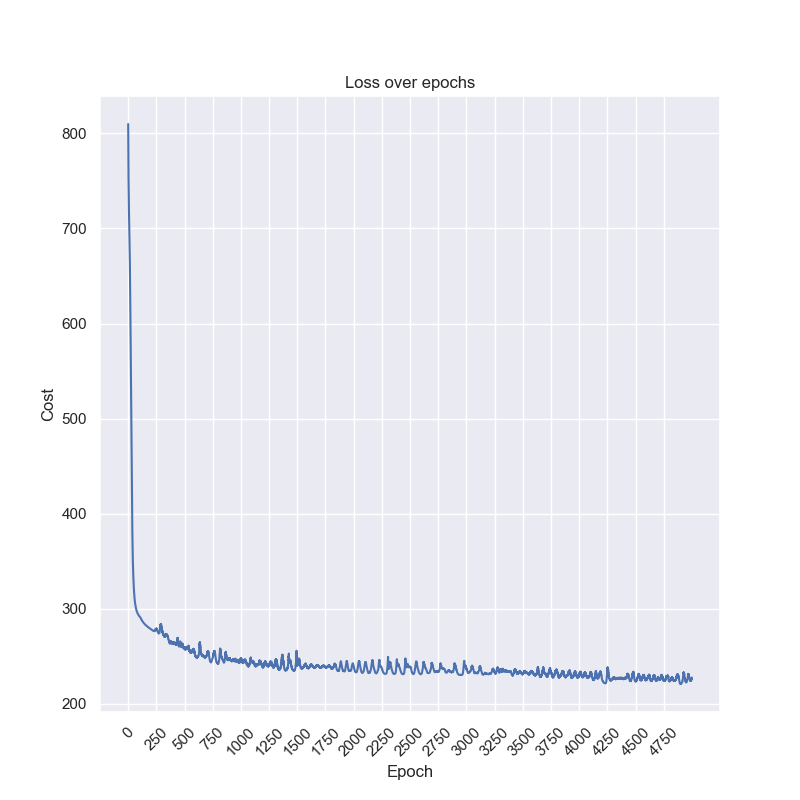
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Output** |
| **Number of Nodes** | **10** | **8** | **8** | **1** |
| **Activation** | **relu** | **relu** | **relu** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 90.59745348 %** (decreased slightly, perhaps due to slight spikes in loss curve below)

**Validation accuracy = 90.90909091 %**

**Val F1-Score = 0.91666667**

****

1. **Configuration 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Hidden 3** | **Output** |
| **Number of Nodes** | **10** | **8** | **8** | **8** | **1** |
| **Activation** | **relu** | **relu** | **relu** | **relu** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 92.75220372%**

**Validation accuracy = 88.63636364 %**

**Val F1-Score = 0.90118577**

This shows overfitting as train accuracy increased, but validation accuracy decreased. Hence we can either decrease the number of nodes, or the number of layers (without dropout).

1. **Configuration 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Output** |
| **Number of Nodes** | **10** | **6** | **4** | **1** |
| **Activation** | **relu** | **relu** | **relu** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 92.067 %**

**Validation accuracy = 90.45 %**

1. **Configuration 5**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Output** |
| **Number of Nodes** | **10** | **6** | **4** | **1** |
| **Activation** | **relu** | **tanh** | **tanh** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 92.36 %**

**Validation accuracy = 90 %**

This shows overfitting as train accuracy increased, but validation accuracy decreased. Hence we can either decrease the number of nodes, or the number of layers (without dropout).

1. **Configuration 6**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Output** |
| **Number of Nodes** | **10** | **6** | **4** | **1** |
| **Activation** | **tanh** | **tanh** | **tanh** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 96.278 %**

**Validation accuracy = 90.4545 %**

Using tanh activation throughout seems to have bettered training, due to which even validation accuracy has improved. There doesn’t seem to be much overfitting at this stage.

1. **Configuration 7**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Output** |
| **Number of Nodes** | **10** | **6** | **4** | **1** |
| **Activation** | **relu** | **relu** | **tanh** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 92.948 %**

**Validation accuracy = 89.091 %**

1. **Configuration 8**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Hidden 3** | **Output** |
| **Number of Nodes** | **10** | **6** | **4** | **4** | **1** |
| **Activation** | **relu** | **tanh** | **relu** | **relu** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 92.2625 %**

**Validation accuracy = 93.636 %**

1. **Configuration 9**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Hidden 3** | **Hidden 4** | **Hidden 5** | **Output** |
| **Number of Nodes** | **10** | **4** | **4** | **4** | **4** | **4** | **1** |
| **Activation** | **relu** | **relu** | **relu** | **relu** | **relu** | **relu** | **sigmoid** |

The results of this configuration are:

**Train accuracy = 90.6953 %**

**Validation accuracy = 93.6363 %**

**Val F1-Score = 0.94166667**

The validation accuracy has increased. It is suspected that this may be due to using several ReLU layers, since ReLU is thought to have a regularising effect.

**Results**

Out of all configurations above, the last configuration yielded highest validation accuracy, for **gaussian initialization, 5000 epochs, 0.0005 learning rate.**

The final result on test dataset is below:

**Test accuracy = 90.86757991 %**

**Test F1-Score = 0.90196078**

**Weight Initialization**

The above was the result of **gaussian initialization.**

For **uniform initialization**, we have:

**Train accuracy = 90.49951028 %**

**Val accuracy = 87.72727273 %**

**Test accuracy = 90.4109589 %**

**Val F1-Score = 0.89243028 %**

**Test F1-Score = 0.90047393 %**

Thus, in this instance, **gaussian initialization performed better.**

**Analysis**

tanh activation did not seem to work well with deeper networks, although it did do relatively well for small networks. ReLU activation had a regularising effect, leading to better validation accuracy and decrease in training accuracy, when stacked in a deep network. The scaling performed (also called Xavier initialization in certain texts) led to a smooth and stable gradient descent, allowing training to perform well. Increasing the number of layers and/or the number of nodes per layer was seen to increase the training accuracy, even leading to overfitting on the validation set.

**Adding Dropout**

Experimenting with adding dropout to **configuration 9 (the final one)** resulted in both the train and validation accuracies decreasing. This is suspected to be partly due to the fact that ReLU activation itself provides a regularisation effect of sorts and on further adding dropout to a neural network which is not heavily overfitting to begin with, the model is prevented from learning effectively.

**Configuration 10:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Layer** | **Input** | **Hidden 1** | **Hidden 2** | **Hidden 3** | **Hidden 4** | **Hidden 5** | **Output** |
| **Number of Nodes** | **10** | **4** | **4** | **4** | **4** | **4** | **1** |
| **Activation** | **relu** | **relu** | **relu** | **relu** | **relu** | **relu** | **sigmoid** |
| **Dropout:**  **keep\_prob** | **1** | **1** | **0.8** | **1** | **1** | **1** | **-** |

The results of this configuration are:

**Train accuracy = 87.17 %**

**Validation accuracy = 88.64 %**

**Test accuracy = 93.61 %**

**Val F1-Score = 0.89626556**

**Test F1-Score = 0.92929293**

Unexpectedly, the test accuracy has increased by 2%, although this could never have been gauged while looking at only the validation set, as we were, while tuning. However, this does illustrate that dropout certainly does decrease overfitting, and can in certain instances, increase test accuracy, as in this case.

**Conclusion**

A thorough experimentation with feature scaling, weight initialization, learning rate, number of epochs, number of layers, number of nodes per layer, layer activations, and dropout probabilities has been performed. With this high level of hyperparameters, it becomes increasingly more difficult to tune the neural network effectively. 10 of the configurations that have been tested are presented in the report, though the testing has been far more extensive. The values also highly depend on the *numpy.random* seed value, and were seen to fluctuate. Thus, the working and implementation of a neural network has been thoroughly understood.