**Artificial Neural Network**

Introduction

One of most important thing one do when an individual buys a house is to compare it with market’s median price.

We built a neural network classifier to identify whether a house with given attributes is likely to be above or below the market’s median price with an F-score of 0.9283 and 92.81% accuracy.

Problem definition

We worked on ‘housepricedata.csv’ as provided. This data set comprises of 10 features each of 1460 houses and a binary class indicating whether the price of the house is above or below the market’s median price. The given features of a house are:

|  |  |
| --- | --- |
| **Variable** | **Nature** |
| LotArea | Continuous |
| OverallQual | Discrete |
| OverallCond | Discrete |
| TotalBsmtSF | Continuous |
| FullBath | Discrete |
| HalfBath | Discrete |
| BedroomAbvGr | Discrete |
| TotRmsAbvGr | Discrete |
| Fireplaces | Discrete |
| GarageArea | Continuous |

And, discrete class variable AboveMedianPrice which is one of 1(above) or 0(below).

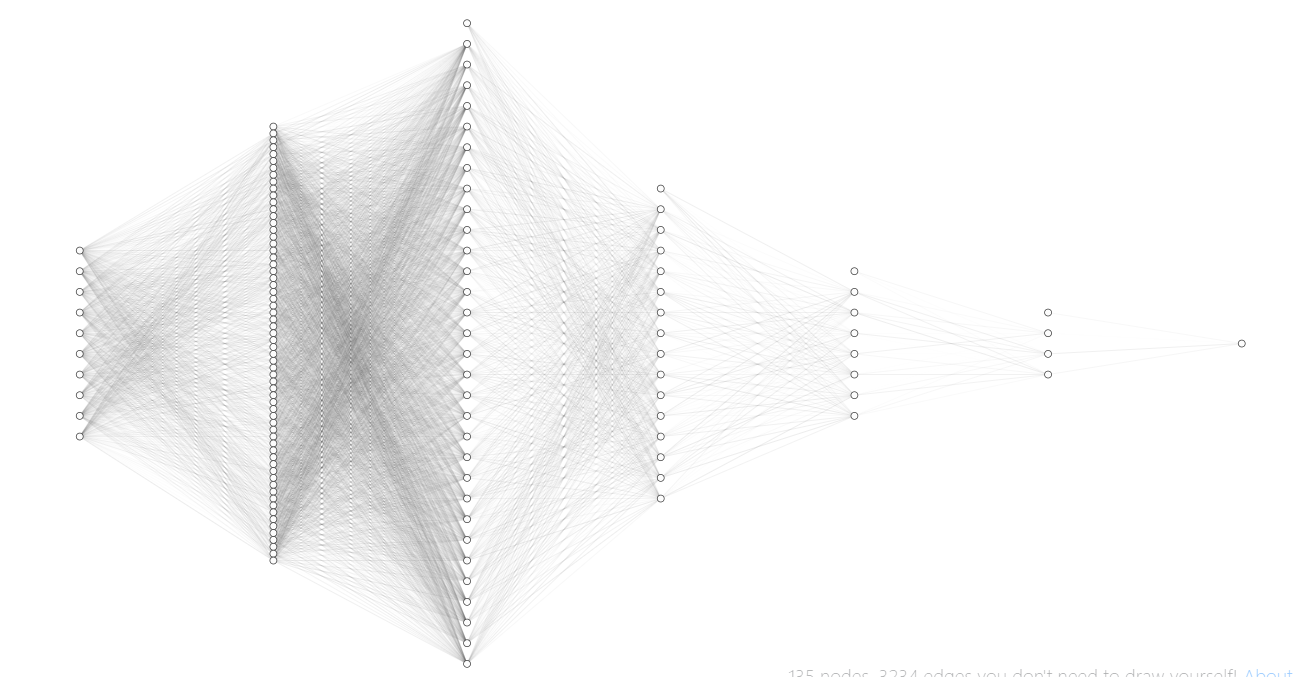
Methodology

We stored the data set in a 2-dimensial numpy array of shape (1460,11) and segregated the dataset into feature transposed array of shape (10,1460) and targeted variable of shape (1,1460). We then scaled the features to bring down their mean and variance to 0 and 1 each respectively. A careful 80-20 split is followed after that to ensure relative mapping doesn’t change.

We designed a multilayer feed forward deep network of 6 layers.

Input layer comprises of 10 nodes corresponding to each input feature of the house and output layer consists of just 1 node predicting 0 or 1. Specifications of layers are as follows:

|  |  |  |
| --- | --- | --- |
| Layer | Nodes | Activation function |
| Input layer | 10 | - |
| Hidden Layer 1 | 64 | ReLU |
| Hidden Layer 2 | 32 | ReLU |
| Hidden Layer 3 | 16 | ReLU |
| Hidden Layer 4 | 8 | ReLU |
| Hidden Layer 5 | 4 | ReLU |
| Output Layer | 1 | ReLU |

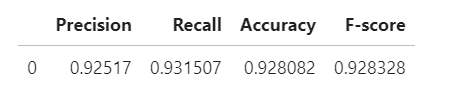


We defined our binary Cross-entropy cost function including L2 regularization loss (parameterized λ2) averaged over 1168 training examples. An adaptive learning rate optimization algorithm – **Adam** is used to find the weights and biases minimizing this cost function. We used python’s scientific computing library Numpy for ***Vectorized*** implementation allowing each epoch to pass through complete training set at once without 2 explicit for-loops which speeded up the training up to 300 times. One explicit for-loop is (mandatory) for forward and backward propagation from layers to layers. Since, we are using ReLU as our activation function in hidden units **He-initialization**is used. We embedded orthogonally **L2 regularization** and **dropout** in case high variance is to be seen. Since, the number of training examples is very low and we used vectorized implementation we choose **mini-batch size** as the batch itself. We embedded **learning rate decay** in the model as well. We used accuracy and f1-score as our optimizing metric and training time as satisficing metric. Hyper-parameter search was done using scaled-based grid followed by course to fine random sampling scheme grid. A caviar approach of training many networks was used to find the best-fitting model considering number of hyperparameters.

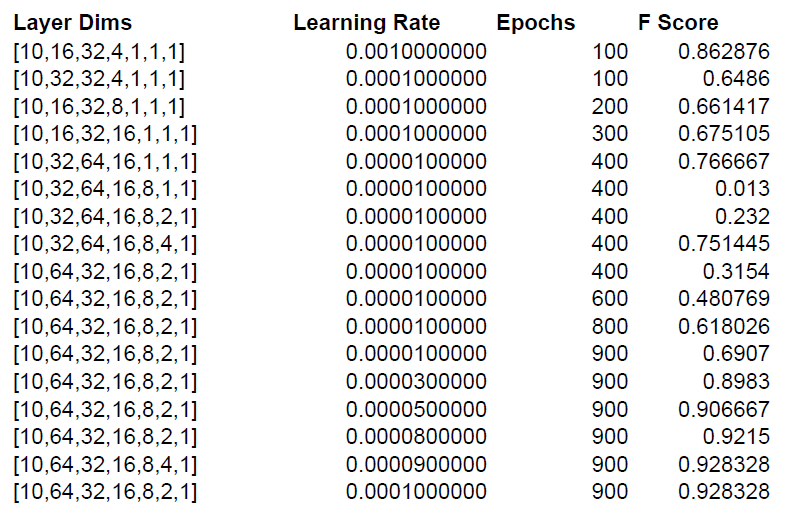
Results

1. Without regularization:

We obtained a classifier with impressive metrics as followed:

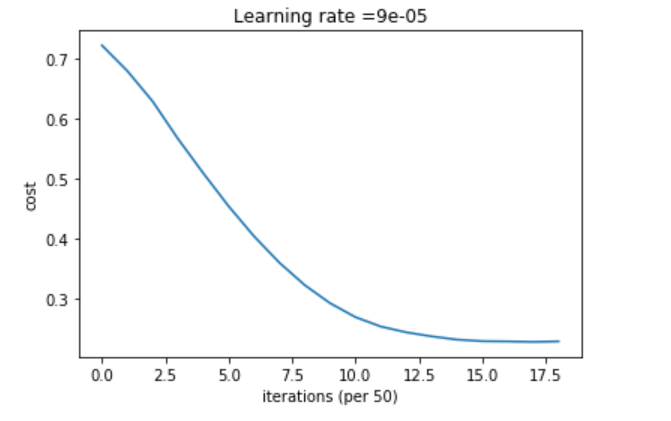


Hyper-parameter search (Metric: F1-score),



Shows that, our best hyper-parameter are:

|  |  |
| --- | --- |
| Layers | [10,64,32,16,8,4,1] |
| Learning Rate | 0.00009 |
| Epochs | 901 |
| Lambda | 5e-2 |
| Beta1 | 0.9 |
| Beta2 | 0.9 |
| Decay\_Rate | 0 |
| Seed of randomisation | 3 |



|  |  |
| --- | --- |
| Learning Rate : | 0.00009 |
| Epochs : | 901 |

1. Different initialization:

Different seed of randomisation was used for different He-initializations, and the results varied way too much for model with our tuned hyper-parameters.

We found seed=3 as our best He-intializations.

1. With L2 regularization:  
     
   Different values of L2 regularization parameter was tried, and we observed *high value* of lambda hurt the performance but low value suited.

Though, small value of in order <10-2 worked well as variance **is** observed, 0.05 being best.

1. Different Beta1 ( of Adam ):
2. Different Beta2 ( of Adam ):

1. Dropout:  
   We implemented dropout regularization which only hurt the performance for every value of probability of keeping the layers. So we didn’t include it in our final fiited model.

References

1. <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>
2. http://fa.bianp.net/blog/2019/evaluate\_logistic/
3. <https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/>
4. <https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2>
5. <https://medium.com/usf-msds/deep-learning-best-practices-1-weight-initialization-14e5c0295b94>
6. <http://alexlenail.me/NN-SVG/index.html>