UNIVERSIDAD
POLITÉCNICA
DE MADRID

Deep Learning: Assignment of Units 1 and 2

Constructing a deep neural network for the Median House Value classification problem

Submitted To:
Daniel Manrique
Martin Molina

Submitted By :
Michail Gongolidis
Alejandro Gonzalez Gonzalez

Introduction

The objective of this activity is to follow the construction process of a deep artificial neural network for the Median House Value classification problem. This problem consists on estimating the median house value of a block group in California using the California Housing Prices dataset.

The dataset that we have in our hands is consisted of 20,433 instances that are obtained with 8 attributes individually normalized with a min-max scaling. The 8 attributes correspond to latitude and longitude of the house, medianage, number of total rooms, total number of bedrooms, the population of the area the house belongs, households and the median income.

Our purpose is to predict the value of the house, but the prices are classified into 10 different classes, the first one starting at 15 thousand dollars and the last one ending at 500 thousand dollars.

Each class was labelled from 0 (the cheapest) to 9 (the most expensive), and one-hot encoded before the beginning of the assignment.

Finally, the dataset is pre-splitted in 3 parts, 80% of it belonging to the training set and the rest 20% is divided equally to test and development set.

Design Process

In this section we present the process we followed to reach our final optimal model.

Part 1 - Initial Hyperparameters

In the first part we tried to create a dummy model to test out Tensorflow and the nature of the problem. We found similarities with the problem presented in the class so we adapted the presented model to our problem.

We initialized our hyperparameters by setting the number of $epochs = 100$, $learningrate = 0.1$, $batchsize = 32$ and we introduced a neural network with 4 hidden layers with neurons per layer $[150, 75, 25, 10]$.

Afterwards, we declared the placeholders that are used to feed the actual training examples.We initialized the hidden layers as dense layers with activation function $Relu$ and the output layer with activation function $softmax$. In order to compute the cross-entropy (the sum of the loss for every sample) and the cost function (the mean of the cross entropy for all the set of data) the net-out is used as an input.

The goal is to reduce the cost function and the cross entropy, with that purpose we use an optimizer.Our initial optimizer is the SGD.

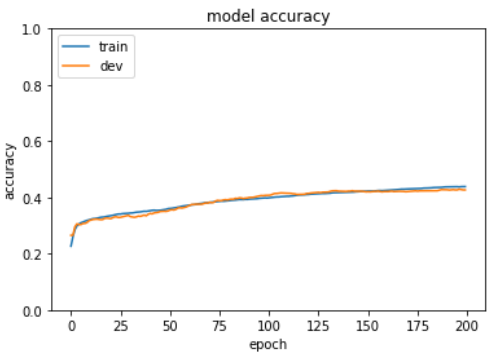
We run the session and after 1:13 seconds in a GTX 1060 graphics card we got $Testaccuracy$: 0.38258317 and $Testloss$: 1.5202003

We realized that we had really high bias and our model was oversiplifying.

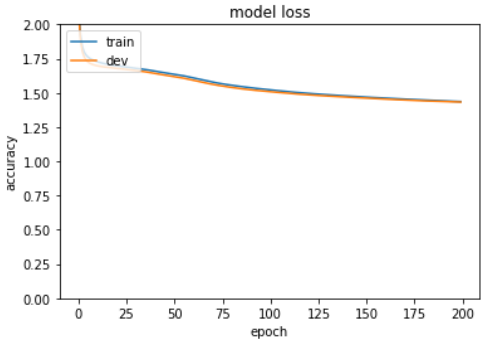
Part 2 - Architecture tuning

Our first improvement method was to change the architecture of our model. For this, we increased the number of epochs to 200 and decreased the learning rate to 0.001 . Our main change thought was the increase of the number of neurons and the number of layers to $[4096, 2048, 1024, 512, 256, 128, 64, 32, 16]$. The rest parameters remained the same.

The result was to increase the accuracy of the test set to $Testaccuracy$: 0.43759942 and $Testloss$: 0.42731276. The time was 05:13 with the same graphic card. At this moment we checked also



(a)



(b)

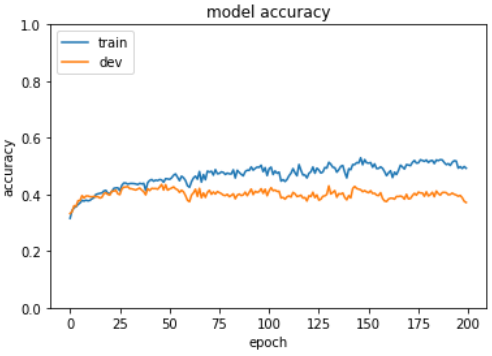
the train and validation accuracy which both remained in the same level : $Trainaccuracy$: and $Devacc$: 0.42731276

We also introduced graph plots to see the progress of the accuracy and the test loss as seen above.We can see that we are underfitting a lot but the variance is really low. That indicated that we should try to change our optimizer and maybe we could obtain better results. Also we thought that since by increasing the size a lot we could not get optimal results maybe we can try lowering the size to increase the performance.

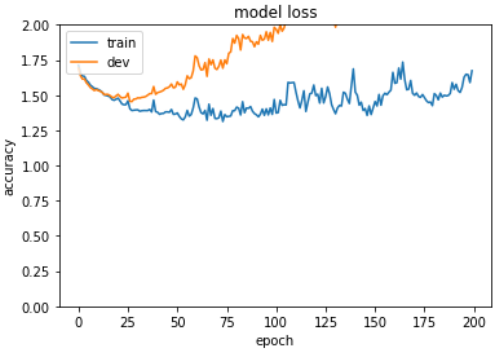
Part 3 - Changing Optimizer

Altough the NN was made bigger the loss and the accuracy are not improving both for the training and development-set. Therefore, the optimizer will be tuned. Initially we decreased the number of layers and the number of neurons to $[1024, 512, 256, 128, 64, 32, 16]$. The rest hyperparameters remained the same.

We changed the optimizer to Adam optimizer and the results were $Trainaccuracy$: 0.5115013 and $Developmentaccuracy$: 0.4082232 but we got $Testaccuracy$: 0.3962818. The time it took the model to train was 3:13 minutes.



(a)



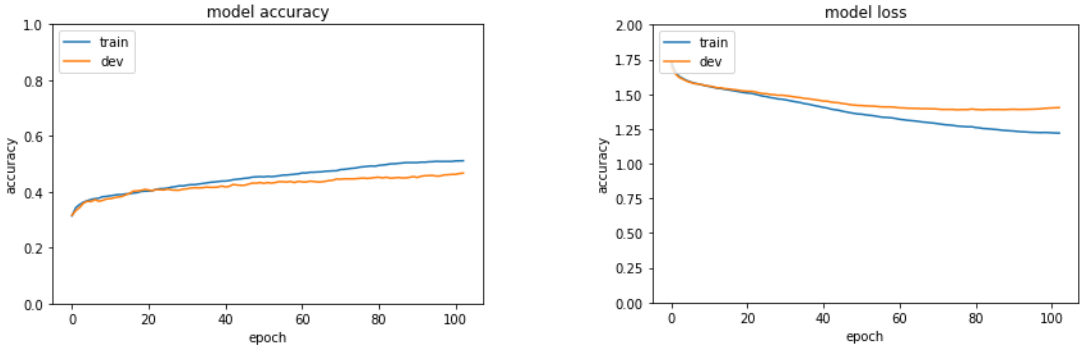
(b)

We understood that we got better training accuracy but the variance was getting higher. On the next step we tweaked some hyperparameters to achieve greater train accuracy and introduced two techniques in order to lower the variance.

Part 4 - Early stopping and decreasing learning rate

After designing the bigger network and using the new optimizer the network has begun to overfit, since there is a difference between the performance of the training set and the validation set. In the loss curves it is even possible to observe that the validation loss is increasing due to overfitting.

In this part we introduced early stopping with patience of 20 steps and decreasing learning rate to cope with high variance. We also increased the number of epochs to 10000 since the network will stop the training by itself.The batch size was increased to 64. We setting the learning rate to 0.0003 with $decreasingsteps = 1000$ and $decreasingrate = 0.96$



(a) (b)

The network would take around 2:30 hours to train but with early stopping it stopped at epoch 102 .

We got $Testaccuracy$: 0.46722114 − $Testloss$: 1.3968592, the best one yet and $Trainaccuracy$: 0.5106448 with $Developmentaccuracy$: 0.46255508. We can see that the accuracy of the model can increase but only by overfitting. In order to try to increase more the development accuracy we tried he initialization on the next part.

Part 4 - He Initialization, L2 Regularization and Dropout

In this section we wanted to enable the model to train longer but without losing the development loss. In order to do that we implemented He initialization, L2 regularization and dropout with $rate = 0.2$, techniques used to reduce the variance.

We adapted the architecture of the network to enable the process to train longer by adjusting the batch size to X and by adding X n layers and n layers as follows $[4096, 2048, 1024, 512, 256, 128, 64, 32, 16]$

Final results

Conclusion

The project was simple to understand and challenging to optimize it.It required collaborative work and ideas, clear understanding of the theory presented in the class and patience.We can conclude that we learned most of the basic deep learning techniques for feed forward neural networks in a pleasant way.

References