HOUSING LOCATION

Practical Project

Building a Deep Neural Network

Authors:

Al Bahri Yassir (yassir.albahri@alumnos.upm.es)
Alderisi Serena (s.alderisi@alumnos.upm.es)
Mateos Vicente Elisa (elisa.mateos.vicente@alumnos.upm.es)

Course:

Deep Learning
Master Degree in Data Science

April 15, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The following study is an implementation of a deep neural network for solving a classification problem. This consists in estimating the approximate location of housing blocks in California.

The neural network classifies an input vector in one of the 4 classes that describe the approximate locations of a house: Near Bay, Less than 1 Hour from the Ocean, Inland and Near the Ocean. These categorical values are converted using one hot encoding as can be seen in Table 1.1.

Table 1.1: Original variables.

Ocean Proximity	**One Hot Encoding**
Near Bay	0 0 0 1
<1h Ocean	0 0 1 0
Inland	0 1 0 0
Near Ocean	1 0 0 0

The First dataset *OceanProximityPreparedCleanAttributes.csv* contains the already cleaned and prepared data of 20428 houses in California with 9 attributes: longitude, latitude, median age, total rooms, total bedrooms, population, households, median income, and median house value.

The second dataset *OceanProximityOneHotEncodedClasses.csv* contains the one hot encoding converted values for each of the 20428 houses. The 9 attributes are the Predictor Variables, while Ocean Proximity is the Target Variable. The output of the network is the predicted class that describes the house location.

California Housing Prices dataset is available at StatLib repository [1]. It is based on data from the 1990 California census. The original dataset appeared in R. Kelley Pace and Ronald Barry, *Sparse Spatial Autoregressions* [3].

Chapter 2

Design Process

The data is ready for the training but before that, the dataset needs to be partitioned. Considering the amount of information present, the partition is done according to the general guidelines for datasets this size (Figure 2.1).

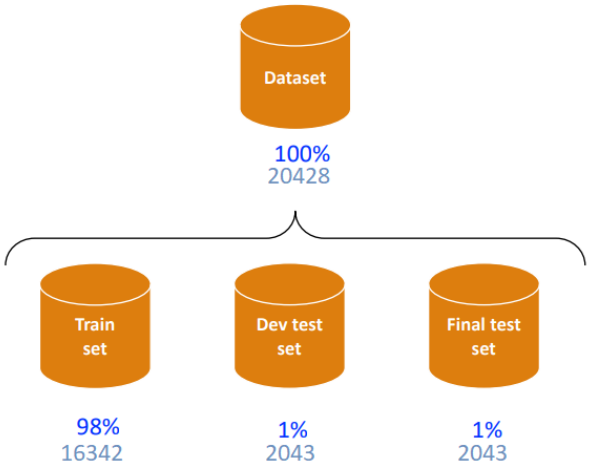

Figure 2.1: Dataset partitions.

2.1 Previous designs

The architecture can be set now that the dataset is partitioned. Starting from the deep neural model with ReLU in Keras for the Median House Value studied in class, different attempts have been executed. The accuracy values can be seen in Table 2.1.

- The **First** model consists of the same hyperparameters and network (600 epochs, a learning rate of 0.1 and a batch size of 25), adding the Stochastic Gradient Descent as the optimizer. This optimization algorithm helps to reduce high bias. This architecture will allow a decent accuracy in the development test but it will need a high performance time (Figure 2.2).

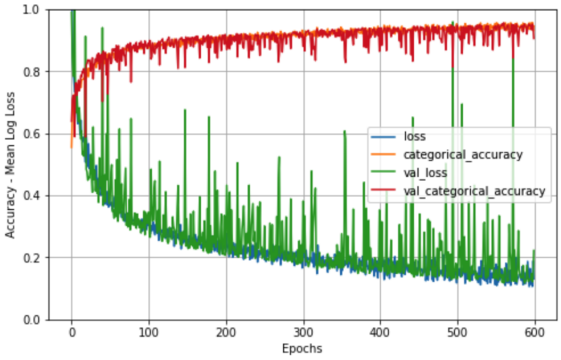

Figure 2.2: First model.

- In the **Second** model the hyperparameters have been changed. The number of epochs has been reduced to 400 and the batch size upgraded to 50. The learning rate and the net remained the same as in the previous attempt. These changes are meant to reduce slightly the time, overfitting and by reducing the epochs and the batch size. A slight improvement is also expected (Figure 2.3).

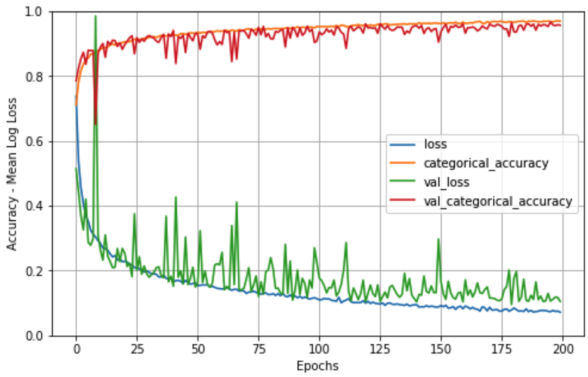

Figure 2.3: Second model.

- In the **Third** model some hyperparameters have been reduced even more. The numbers of epochs is now 200 and the batch size is 25. The learning rate and the net remained unchanged. Similar to the previous model but more time efficient (Figure 2.4).

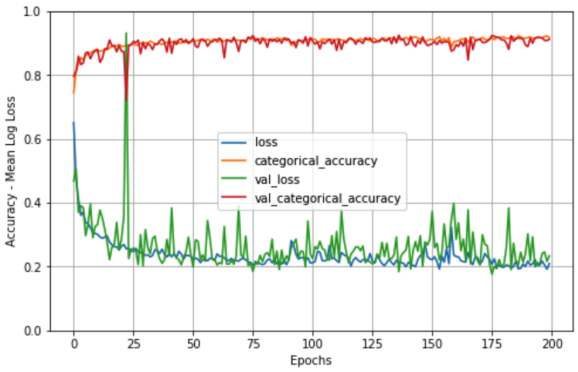

Figure 2.4: Third model.

- In the **Fourth** model the activation functions in the hidden layers were reset to ReLUs. The number of epochs rosen to 500 and the batch size to 35. The learning rate is always set to 0.1. This model is similar to the third one but with more brute force (and time cost). The best result in terms of accuracy so far. In Figure 2.5 the 2 curves related to the train set and the validation set become gradually, at each epoch, more and more distant between each other. This is a signal that even if this model produced a good accuracy it does not mean it's a satisfactory model, indeed it's affected by overfitting.

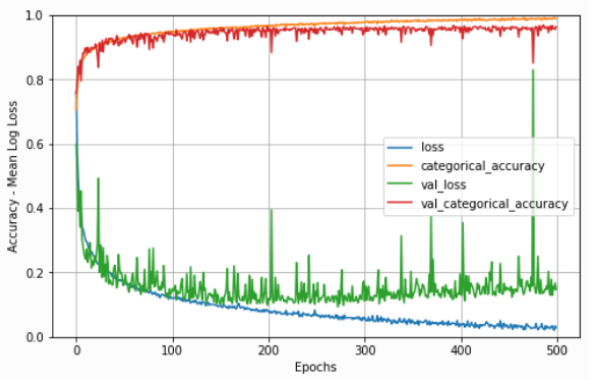

Figure 2.5: Fourth model.

- In the **Fifth** model the focus is on reducing the overfitting and the performance time. All the previous models took hours to complete. The main changes concern the optimizer. The use of Adam is expected to reduce the time by a very high margin. Additionally dropout (0.2) is added to solve the problem of overfitting as can be seen in Figure 2.6, reaching similar accuracy values that the more powerfuls SGD in the previous models while taking way less time to compute. This models consist of 200 epochs and a batch size of 25, while mainting 3 hidden layers of 10000 neurons. The Adam parameters are:
  - learning rate: 0.001
  - β_1: 0.9
  - β_2: 0.999
  - amsgrad: False

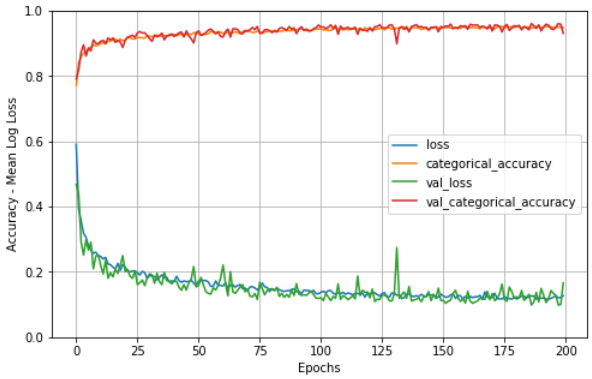

Figure 2.6: Sixth model.

Table 2.1: Various layers and neurons design and the results obtained.

Model	Train Accuracy	Dev Accuracy
1	0.94609	0.90602
2	0.97461	0.94371
3	0.96934	0.95693
4	0.98947	0.96476
5	0.95501	0.96080

2.2 Main model

The efforts are refocused in trying to improve the previous. For this, some changes are done. Remainder that the dropout is kept at 0.2 in all of the hidden layers.

2.2.1 Layers and neurons

The next models focus on the changes in the structure of the net used in the **Fifth** model. Different configurations have been tried, increasing the number of hidden layers and changing the distribution of neurons per layer. Since the accuracy in these models does not suffer much changes, we are not going to discuss in detail each attempt. Table 2.2 presents a summary of all the changes performed and the results.

Table 2.2: Various layers and neurons design and the results obtained.

Structure	Accuracy	Train Error	Test Error	Time (s)
[10000, 10000, 10000]	**0.9680**	**4.71%**	**4.77%**	**1563**
[10000, 10000, 10000, 10000]	0.9501	5.33%	4.99%	1635
[10000, 10000, 10000, 10000, 10000]	0.9486	6.25%	5.14%	1661
[5000, 5000, 5000, 5000, 5000]	0.9462	5.72%	5.38%	584
[5000, 5000, 5000, 5000]	0.9486	5.27%	5.14%	564
[5000, 5000, 5000]	0.9555	4.71%	4.45%	558
[10000, 5000, 2500, 750, 250]	0.9530	5.73%	5.38%	580
[5000, 250, 750, 250]	0.9471	5.74%	5.29%	557

2.2.2 Batch normalization

Normalization is implemented in the form of Batch normalization. To study the different effects it has on the network, the normalization layer will be implemented before and after the activation function.

Before the activation function

As it was somehow expected, the results worsen (0.9429 validation accuracy). This is probably due to the Batch normalization layer nullifying part of the gradient each layer. This means that only part of the ReLUs are firing, thus getting vanishing gradient.

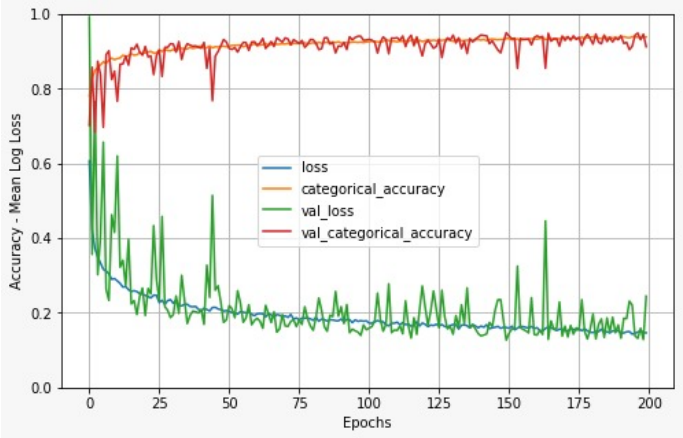

Figure 2.7: Batch normalization applied before activation.

After the activation function

In this case, the results are slightly worse than using the normalization layer before the ReLU. It was expected from this design to perfom worse than the previous one but the outcome show a different picture.

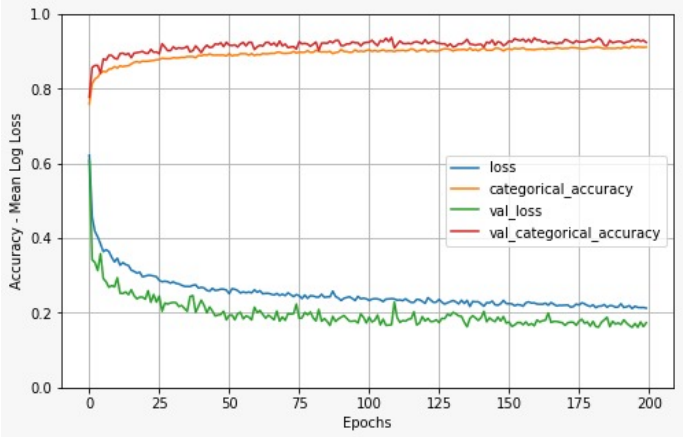

Figure 2.8: Batch normalization applied after activation.

2.2.3 Initializers

An implementation of the random uniform initializer is tried. In this case the bias is set to zero. As can be seen in Figure 2.9, the accuracy doesn't improve but it is interesting to notice how from the very first epoch, the accuracy and the loss function stabilize.

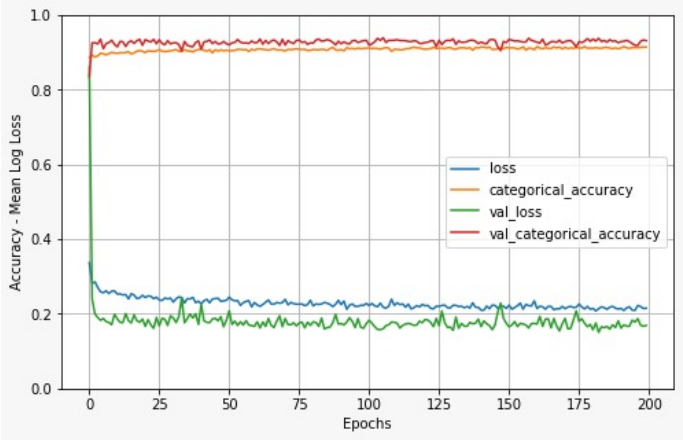

Figure 2.9: Random uniform with bias set to zero.

2.2.4 Regularizers L1/L2

For some reason using the L1 and L2 method of regularization with the main setup seems to give worse results (0.8925 validation accuracy). This is probably a sign of over-regularization in the model as it is penalized so severely that it under-fits the data (considering that dropout is also implemented).

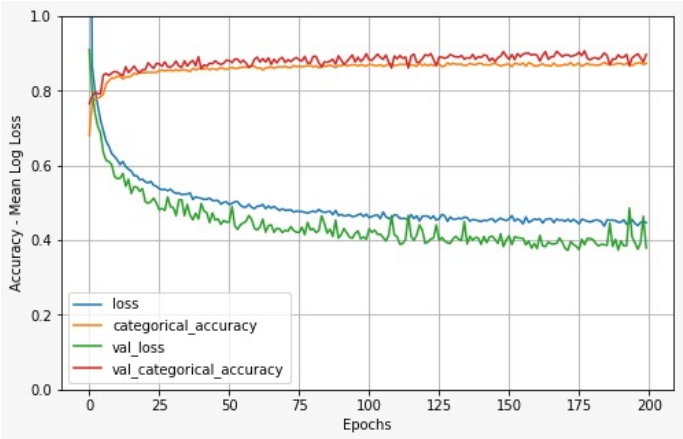

Figure 2.10: Random uniform with bias set to zero.

Chapter 3

Results

At the end, the final model that proved to perform best and most efficiently had the following architecture (Figure 3.1). The optimizer used for this architecture is Adam.

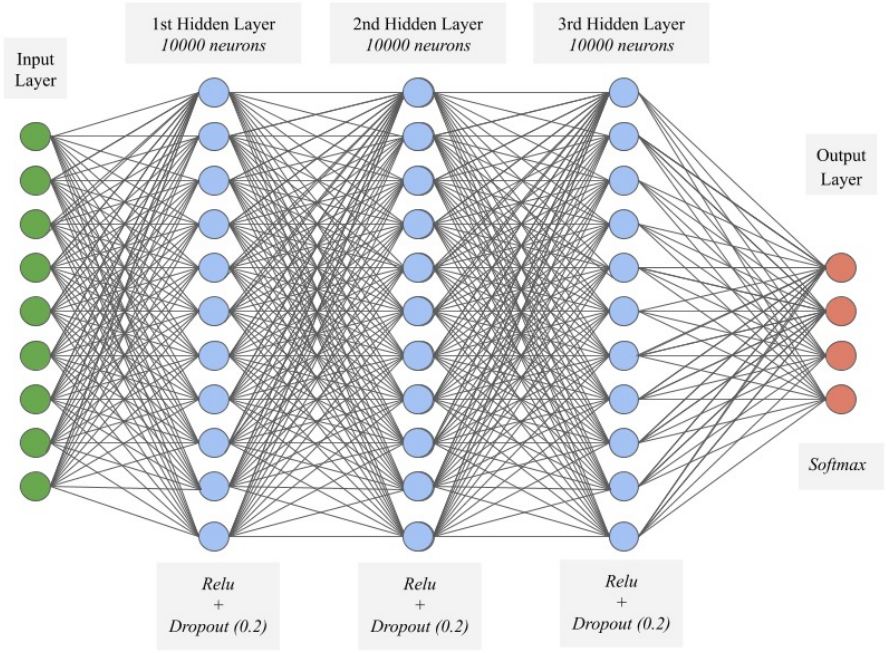

Figure 3.1: Architecture of the definitive Deep Neural Network.

Table 3.1 showcase most metrics of the network for the analysis.

Table 3.1: Results of the definitive network.

Train Loss	**Train Acc.**	**Dev Loss**	**Dev Acc.**	**Train Error**	**Test Error**	**Time (s)**
0.1185	0.9521	0.1235	0.9608	4.71%	4.77%	1563

3.1 Model Analysis

The final network training progression is seen in Figure 3.2. In this plot there is a noticeable and progressive progression in both, accuracy and loss. Overfitting is non-existent as both training and test curves overlap almost perfectly.

It is to notice the presence of some spikes in the accuracy but mostly in the loss function curves. This is due to the mini-batch method used. If would the batch size be smaller, these spikes would be more frequent and noticeable. The training and test error are almost the same and both are lower than 5%. This allow to assert that both, bias and variance are low.

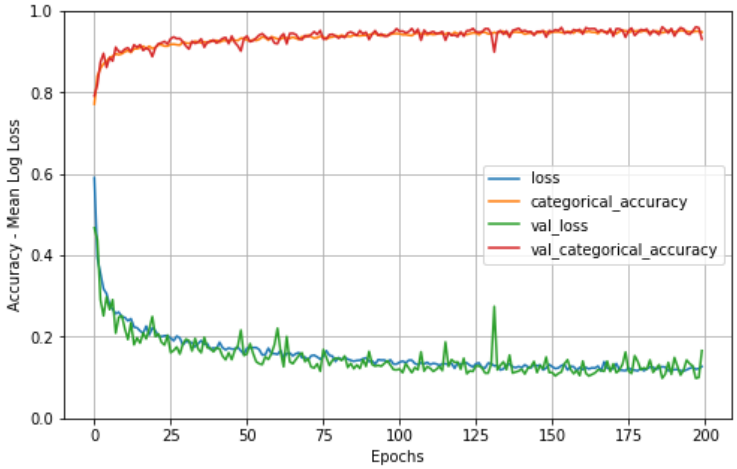

Figure 3.2: Sixth model.

Chapter 4

Conclusion

Accuracy is usually the deciding factor in a Neural Network but efficiency, regarding time cost, is also of high importance. This practical work approach had both factors (accuracy and time) in mind. In our opinion we could have focused in getting the highest accuracy as possible by using brute and raw strength, thus increasing the number of epochs and neurons/layers as much as possible. But we consider that the time constraint gets out of hand very quickly thus, training the network for multiple hours. Our purpose overall was to improve the accuracy as much as possible but keeping the time cost in line.

Bibliography

[1] StatLib, *California Housing Prices*, `http://lib.stat.cmu.edu/datasets/`.

[2] R. Kelley Pace and Ronald Barry, *Statistics & Probability Letters*, Sparse Spatial Autoregressions, Letter 33, no. 3, (291-297), 1997.

[3] Daniel Manrique and Martin Molina, *Artificial Neural Networks and Deep Learning*, Deep Learning Course, MSc. in Data Science, UPM, 2020.