PRACTICAL PROJECT: BUILDING A DEEP NEURAL NETWORK

Deep Learning

Stefano Baggetto *(stefanobaggetto@gmail.com)*
Angel Igareta *(angel@igareta.com)*
Giorgio Segalla *(giorgiosgl1997@gmail.com)*

April 15, 2020

1 Introduction

The aim of this report is to illustrate the creation of a neural network to solve the classification problem of predicting the approximate location of housing blocks given its information. The goal of a classification problem is to identify to which of a set of categories a new observation belong, based on a training set of data containing observations whose category membership is known. In this case our output category will be one of this four classes: *NEAR BAY, <1H OCEAN, INLAND* and NEAR OCEAN

1.1 Structure

After the introduction and the presentation of the dataset, the document will present the approach adopted for the classification of the target variable ocean_proximity using a ffNN along with the multiple tests done. Different network architectures have been designed in order to accomplish the classification test, reaching desirable performance in terms of time and accuracy.

1.2 Dataset

The California Housing Price dataset pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. The data is partitioned in 2 files: *OceanProximityPreparedCleanAttributes.csv* and *OceanProximityOneHotEncodedClasses.csv.* The attribute files contains the clean and prepared data for the nine attributes, or predictors, ready to feed the neural network. It consists of 20428 rows, plus the header, and nine columns labeled as longitude, latitude, median age, total rooms, total bedrooms, population, households, median income and median house value.

longitude	**latitude**	**housing median age**	**total rooms**	**total bedrooms**	**population**	**households**	**median income**	**median house value**
0.013944	-0.630181	0.568627	-0.906760	-0.903786	-0.961882	-0.899030	-0.393415	0.415256
-0.555777	0.086079	-0.019608	-0.991861	-0.991930	-0.995347	-0.992107	-0.741369	-0.494843
0.099602	-0.419766	0.254902	-0.939570	-0.932961	-0.959976	-0.935866	-0.706170	-0.726595
0.302789	-0.759830	-0.725490	-0.893382	-0.894165	-0.949270	-0.892123	0.001228	0.352163
-0.599602	0.060574	0.019608	-0.830256	-0.851024	-0.924269	-0.849696	-0.060951	0.854017

Table 1: Ocean-Proximity Attributes

<1H OCEAN	**INLAND**	**NEAR BAY**	**NEAR OCEAN**
0.0	0.0	0.0	1.0
0.0	0.0	1.0	0.0
0.0	1.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0

Table 2: Ocean-Proximity One-Hot Encoded Classes

1.3 Preprocessing

The data provided is split into input features and labels. The instances corresponding to the value "ISLAND" for the ocean_proximity variable have been removed because there is not enough data for the learning process, as well as instances with missing data.

Regarding the transformations within the data, the input features features have been normalized, which means they have similar orders of magnitude, specifically in the range $[-1,\ 1]$. This can be done with the method MinMaxScaler in the package *preprocessing* of sklearn. Besides, the labels recieved have been *one-hot encoded*, so no further preprocessing was required.

Finally, the data was shuffled and divided manually according to the following percentages:

- Train set: 80%, being a total of 16342 rows.
- Validation set: 10%, being a total of 2044 rows.
- Test set: 10%, being a total of 2042 rows.

1.4 Model Structure

Our ffNN structure for the dataset is based on three dense layers with decreasing number of nodes and dropout between each layer. In order to choose the number of nodes, dropout parameters, learning rates and optimizers we made use of hyperparameter tuning. Here are some of the tested components:

- Dense layers sizes: As we previously commented, we experimented with different sizes per layer, in the range of [32, 64, 128, 256, 512, 700].
- Dropout: It was used for reduce overfitting in our neural network by preventing complex co-adpation on training data. The values we tested with were: [0.05, 0.1, 0.15, 0.2, 0.25].
- Optimizers: Adam, Adamax, Nadam and SGD.
- Activation Functions: Relu and LeakyRelu.
- Learning Rate: $1e^{-03}$, $1e^{-04}$.

1.5 Hyperparameter Tuning

In the next table are presented the **top 20** combinations of hyperparameters we tested for the California Housing Prices dataset 3 with a total number of combinations tested of 480. Note that *Units* column refers to the number of units in the first layer, being the number of units in the following ones half of that size.

Besides, we added the train, validation and test accuracy and loss to have a fairer comparison between the hyperparameter combinations (indicating the overfitting and generalization) and to compare the test time to take into account not only the size of the neural network but the time they take into testing our test set.

Units	Optimizer	Dropout	Train Acc	Train Loss	Val Acc	Val Loss	Test Acc	Test Loss	Test Time
1024	adamax	0.05	93.92%	0.1423	93.88%	0.1415	94.86%	0.1295	0.2590
256	nadam	0.05	92.48%	0.1822	93.49%	0.1527	94.52%	0.1379	0.1162
512	adamax	0.05	93.56%	0.1534	93.20%	0.1513	93.88%	0.1465	0.1362
1024	adamax	0.15	93.04%	0.1668	92.32%	0.1581	93.63%	0.1491	0.2419
256	adam	0.1	92.06%	0.1879	93.05%	0.1653	93.58%	0.1567	0.1327
1024	adamax	0.15	93.01%	0.1658	94.23%	0.1376	93.54%	0.1558	0.2166
256	adamax	0.15	91.49%	0.2	92.71%	0.1825	93.49%	0.1709	0.1249
512	nadam	0.05	93.34%	0.1565	92.81%	0.1636	93.44%	0.1538	0.1382
256	nadam	0.15	91.73%	0.1965	92.27%	0.1761	93.29%	0.1696	0.1174
512	nadam	0.1	92.55%	0.1783	93.79%	0.1449	93.24%	0.1505	0.1335
512	nadam	0.1	92.43%	0.1769	93.40%	0.1608	93.24%	0.1502	0.1335
1024	nadam	0.1	92.67%	0.1839	93.69%	0.1542	93.19%	0.1538	0.2413
1024	adamax	0.1	93.53%	0.1511	93.64%	0.1527	93.10%	0.1761	0.2058
512	adamax	0.15	92.61%	0.1786	92.27%	0.1668	93.05%	0.1569	0.1449
128	adam	0.1	91.40%	0.2045	93.10%	0.1668	92.95%	0.1685	0.1068
1024	nadam	0.2	91.08%	0.2156	93.69%	0.1625	92.95%	0.1697	0.2074
1024	adamax	0.1	93.50%	0.1544	92.56%	0.1802	92.90%	0.1757	0.2292
256	adamax	0.1	92.20%	0.1844	92.66%	0.1745	92.85%	0.1595	0.1346
256	adamax	0.25	90.88%	0.2212	91.59%	0.197	92.85%	0.1884	0.1183
1024	adamax	0.2	92.60%	0.1778	94.32%	0.1502	92.85%	0.1745	0.2116

Table 3: Top 20 Hyperparemeters Combinations

1.6 Final configuration

As we can see, both first and second combinations in the list achieved a very similar test accuracy and difference between test and train accuracy, however, if we take into account the time, the second configuration was 2.2 times faster than the second one.

Hence, the configuration which we obtained better results in balance with elapsed time was:

- Dense layers sizes: As it can be appreciated in the figure 1, the best result was to add three dense layers, the first one with 256 nodes and the second and third ones with half of the size, 128.
- Dropout: 0.05.

- Activation Function: LeakyReLU.
- Optimizers: Nadam.
- Learning Rate: $1e^{-03}$.

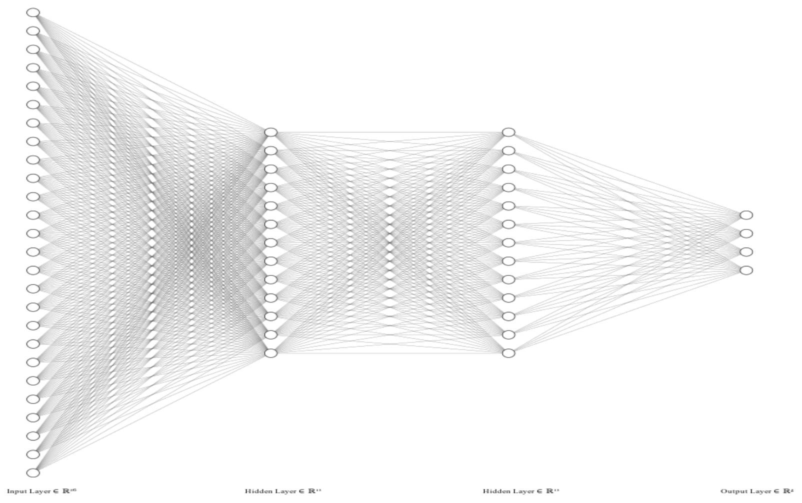

Input Layer $\in \mathbb{R}^{26}$
Hidden Layer $\in \mathbb{R}^{13}$
Hidden Layer $\in \mathbb{R}^{13}$
Output Layer $\in \mathbb{R}^{4}$

Figure 1: ffNN Model for California Housing Prices dataset

Once our best configuration was selected, we decided to test with different number of epochs to improve the test accuracy, always avoiding overfitting the model. The results of those tests are:

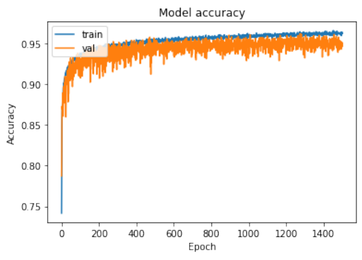
Units	**Optimizer**	**Dropout**	**Epochs**	**Train Acc**	**Train Loss**	**Val Acc**	**Val Loss**	**Test Acc**	**Test Loss**	**Test Time**
128	nadam	0.1	500	0.9514	0.114	0.9496	0.1215	0.952	0.1079	0.1161
128	nadam	0.1	800	0.9593	0.0964	0.9472	0.1204	0.9554	0.1011	0.1333
128	nadam	0.1	1500	0.9632	0.0921	0.9501	0.1288	0.9608	0.1018	0.1161
128	nadam	0.1	2400	0.9673	0.0844	0.9584	0.11	0.9711	0.0836	0.1172

Table 4: Epochs tuning

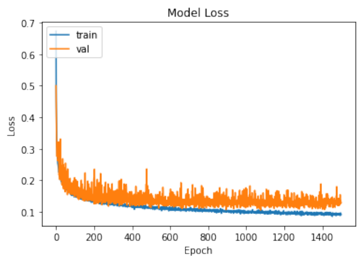
Hence, the best results obtained with this configuration where:

- Train Accuracy and Loss: 96.73% and 0.0844.
- Validation Accuracy and Loss: 95.84% and 0.11.
- Test Accuracy and Loss: 97.11% and 0.0836. Elapsed time: 0.1172s

As we can observe in the figure 2 there is a little of overfitting in the end stage of the epochs but we stopped the training when the difference from train and test metrics was significantly increasing, hence obtaining a good and robust model, with a good bias-variance tradeoff.



(a)



(b)

Figure 2

1.7 Conclusion

Carrying out this classification task we have been able to put in practice the knowledge acquired during the previous classes. First we understood the problem and what we had to do in order to find a solution. We proceeded testing different model structures to see which one gave the best early results. After that, we tuned the hyperparameters improving our accuracy from 70% to 95%. Finally, we tested different epochs number in order to finally reach our best configuration with a 97% of test accuracy.

As a conclusion, this assignment has been a good exercise in order to follow an organized methodology to solve this type of deep learning problems.