Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Máster Universitario en Inteligencia Artificial
Curso 2020-2021

Artificial Neural Networks and Deep Learning

Building an artificial neural network to estimate the quality of a wine

Alejandro Muñoz Navarro and Irene Múgica Aramberri
{alejandro.mnavarro,i.mugica}@alumnos.upm.es

July 22, 2021

Contents

1 Introduction

The objective of this project is to conduct a deep artificial neural network construction process to solve a classification problem.

The classification task aims to estimate the wine quality based on physicochemical analysis. We will base on a dataset named *"WineQualityRawDataset.csv"* provided by the teacher where there are given diferent attribute values of different wines and their quality classification. This dataset of Wine Quality has been grabbed from the UCI repository `https://archive.ics.uci.edu/ml/datasets/wine+quality`.

For the end of the project we will be able to classify a wine into one of three categories: Poor, Regular or Excellent wine from a set of attributes resultant of objective tests with a quite satisfying accuracy.

2 Preprocess of data

This section aims to describe the preprocess that the data has suffered before going through the construction of the neural network. We will describe the initial structure of the database and it will be shown how the two .csv necessary for the neural network have been created.

Initially, the dataset contains 4.898 instances with 11 attributes, each of them related to objective tests: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulfates, and alcohol. The label quality (output) has 11 classes based on sensory data (median of at least three wine experts' evaluations). Each expert graded the wine quality between 0 (horrible) and 10 (excellent). However, the data is unbalanced. There are many more normal or intermediate wines than excellent or poor ones. Therefore, it is a good idea to group wines into less than 11 quality classes. So that, we have made an agrupation of Poor, Regular and Excellent wines.

The PreparingWineQualityDataSet code provided by the practice has been used for data processing. No modifications were made other than some misspelled names.

This code, receives the initial dataset finally creating two new datasets. To do this, the following steps have been followed:

- The first step in preparing the data is to find out whether or not there are missing values.
- The second step is to review outliers and scale data. In this step there are several actions to consider with respect to outliers:
  - Scale each attribute based on its mean and standard deviation (normalization). This approach can produce values greater than 1 or lower than -1.
  - Remove rows containing outliers; at least some of them. The disadvantage is that we don't have as many examples in the data set.
  - Cure the data by modifying the outliers.
- The third step is to check how many instances per tag there are and group them if necessary.
- The fourth step is to suffle the data set, which has been done three times.
- The fifth step is split the dataset vertically into attributes $\boldsymbol{x}$ and label $\boldsymbol{t}$ (target) for supervised learning. In our exercise, there are 11 attributes and one label (quality) with three different values or classes.

- The sixth Step is one-hot encode the classes for a classification problem.
- The seventh step is min-max scale of the input dataset (attributes) within the range [-1,1] for each feature independently.
- Finally, the last step is to save both the x-array and the target t-values in csv files:
  - x: WineQualityPreparedCleanAttributes.csv
  - t: WineQualityOneHotEncodedClasses.csv"

3 Design process

In this section we will be showing the intermediate networks we have constructed in order to reach our final neural network. We have followed an order of parameters over which we have made modifications. These modifications have been made in a direction toward best performance of the model. Each subsection is directed to fix one parameter. The value of that parameter that has been finally chosen will be written in bold.

In order to quantify this performance, for each network configuration, we got the training and testing accuracy, as well as the time needed for each model to learn. Based on this, we change the values of the network. Additionally, and when required, a graph visualizing the evolution of these metrics will be attached.

We will depart from a default configuration of the network:

n_epoch	1000	**Initializer**	-
learning_rate	0.1	**Batch normalization**	-
n_neurons_per_layer	[500, 250, 75, 25]	**Regularization**	-
batch_size	512	**Optimizer**	SGD(lr=learning_rate)
Activation function	relu		

We will try to get the highest testing accuracy. However, we need to bear in mind that high testing accuracies could be due to to overfitting. If the training accuracy is extremely high, it is a sign of overfitting, and therefore, the resulting high testing accuracy is a "fake" good performance as the network has learnt literally the data and will not be able to well classify data that has not prevously seen. Therefore, as well as analyzing the high value of the testing accuracy, we will check if the difference between training accuracy and testing accuracy is small. If this is the case, then we will have a satisfying network, else, we will have overfitting.

3.1 Changing basic hyperparameters

Batch-size

First, we will modify the batch size and see which one fits best our data. As we will do for every parameter, we will show the training and testing accuracy as well as the time needed for the network to learn.

Batch size	train_acc	test_acc	Time
16.342	63.3%	63.3%	19
8192	62.5%	61.2%	19
4096	62.2%	62.7%	19
2048	**68%**	**63.3%**	**21**
1024	74.2%	55.1%	26
512	97.2%	71.6%	35
256	99.8%	69.6%	54
128	99.8%	70%	90
64	100%	70.4%	166

Table 1: Batch size selection.

As it can be seen in table 1, from batch size 512 to 64 we are suffering from overfitting. If we were just looking for high test accuracy, one of these will be chosen, however, the difference between the two accuracies is really high. We have very high training accuracies, resulting in overfitting. Therefore, we will focus our attention in higher batch sizes.

On the one hand, we can appreciate that batch size 2048 has a relatively high test accuracy (63.3%), and, moreover, the training accuracy is not far away from it (68%). Therefore, it will be a quite good candidate. On the other hand, with batch size 4096, we have a similar testing accuracy (62.7%) with a training accuracy nearly identical (62.2%). This ensures us that we are not absolutelly having overfitting, and that is a good point to take into account. However, between these two high testing accuracies we ended up choosing batch size 2048 and the reason why we have gone with it is that, although we have a higher gap between testing and training accuracies than with batch size 4096, it is still small and we benefit from a bit of higher precision.

Structure

In this section we will change the structure of the layers. We will both modify the number of layers and the number of neurons per layer. We will proceed as in with the batch size.

Structure	train_acc	test_acc	Time
$[1000, 500, 250, 75, 25]$	66.1%	62.2%	22
$[500, 250, 75, 25]$	68%	63.3%	21
$[\mathbf{250}, \mathbf{75}, \mathbf{25}]$	**63%**	**64.5%**	**21**
$[75, 25]$	62.3%	63.1%	21
$[25]$	59.5%	61%	20

Table 2: Structure selection.

As we can see in table 2, we have tried with 5, 4, 3, 2 and 1 layers, modifying in each of them the number of neurons. The results got from this analysis make us choose the

third structure $[250, 75, 25]$ as it is the one returning the highest testing accuracy and it has nearly the same training accuracy. Therefore, we will go with it. Actually, all the structures we have tried return really satisfying results, however, in order to follow efficiency, we will try to go with the lower number of layers and neurons possible.

3.2 Changing activation function

Activation function	train_acc	test_acc	Time
relu	**63%**	**64.5%**	**21**
elu	60.1%	63.3%	21
tanh	61.1%	61.4%	21
linear	57%	59%	21
sigmoid	47.1%	45.3%	21
softsign	59.5%	61%	22
softplus	56.8%	60.4%	21
hard_sigmoid	45%	44.1%	21
exponential	33.4%	33.9%	21
softmax	45%	44.1%	21
selu	63.7%	62.2%	20

Table 3: Activation function selection.

For the same reason as in the previous sections, we will go with the network returning the highest testing accuracy bearing in mind it is not far away from the training accuracy. In this case, the network returning the most satisfying values is the one with activation function *relu*.

3.3 Changing initializers

Now, we will go through the same procedure but changing instead the initializers.

Initializer	train_acc	test_acc	Time
None	63%	64.5%	21
Zeros	45%	44.1%	20
RandomUniform(-0.1, 0.1)	61.9%	62.9%	20
RandomUniform(-0.3, 0.3)	68.2%	62.7%	20
RandomUniform(-0.05, 0.05)	60.8%	64.5%	21
RandomNormal(0.0, 0.05)	59.4%	60.6%	21
he_uniform	**69.2%**	**65.5%**	**21**
he_normal	67.5%	61%	21

Table 4: Initializer selection.

After trying with different initializers, we move from not using any initializer to use the *He (Uniform)*. It upgrades the testing accuracy from 64.5% to 65.5%, and not differing that much from the training accuracy (69.2%).

3.4 Using batch normalization

Batch normalization	train_acc	test_acc	Time
Without	**69.2%**	**65.5%**	**21**
After	99.9%	68.8%	26
Before	99.5%	68.6%	26

Table 5: Batch normalization selection.

Until now, no batch normalization was used, and once we have tried to introduce it either after of before the activation function, we get overfitting. The training accuracies go very high, and although the testing accuracies get higher than without batch normalization, they are result of overfitting. We will stay without batch normalization. Moreover, although it is not very significant, we save in learning time.

3.5 Using regularization

In this section we will try with different regularizers. This will be done with both batch size 2048 and 4096. This is done due to the good results that we got in section 3.1 with these two batch sizes.

Batch size: 2048

Regularizer	train_acc	test_acc	Time
Without regularization	65.8%	58.6%	21
Regularizer L2 (lambda=0.001)	66.9%	62%	23
Regularizer L2 (lambda=0.0001)	71.4%	63.7%	23
Regularizer L2 (lambda=0.00005)	70.8%	63.3%	23
Regularizer L2 (lambda=0.00001)	69.2%	62.7%	23
Regularizer L1 (lambda=0.0001)	65.1%	57.8%	23
Dropout (rate=0.2)	**59.4%**	**64.7%**	**22**
Dropout (rate=0.1)	64.3%	63.5%	23
Dropout (rate=0.01)	67.6%	60.4%	22
Dropout (rate=0.001)	71.3%	62.4%	22

Table 6: Regularizer selection with batch size 2048.

Batch size: 4096

Regularizer	train_acc	test_acc	Time
Without regularization	65.2%	61.4%	21
Regularizer L2 (lambda=0.001)	65.1%	61.4%	20
Regularizer L2 (lambda=0.0001)	65.9%	64.3%	20
Regularizer L2 (lambda=0.00005)	65.7%	63.1%	20
Regularizer L2 (lambda=0.00001)	65.3%	64.1%	20
Regularizer L1 (lambda=0.0001)	66.1%	63.7%	21
Dropout (rate=0.2)	**58.5%**	**64.5%**	**20**
Dropout (rate=0.1)	59.6%	63.5%	20
Dropout (rate=0.01)	64.1%	62.4%	20
Dropout (rate=0.001)	65.6%	61.4%	20

Table 7: Regularizer selection with batch size 4096.

As it can be appreciated in both tables 6 and 7, the highest testing accuracy is got with a *Dropout* with a rate of 0.2. We run the previous model once again and got a lower value without regularization. Therefore, we consider introducing a Dropout. Between the two batch sizes, we will go with batch size 2048 as we ensure a higher precision and it has quite small difference with the training accuracy.

3.6 Changing learning rate and epoch

Learning rate

In this section we will change the learning rate in order to go with the network with the best performance.

Learning rate	train_acc	test_acc	Time
1	72.5%	64.1%	22
0.7	66%	60.6%	22
0.5	**68.5%**	**64.7%**	**23**
0.1	59.6%	63.5%	22
0.05	58.4%	63.1%	22
0.01	55.4%	60.8%	23
0.001	47.7%	51.4%	22

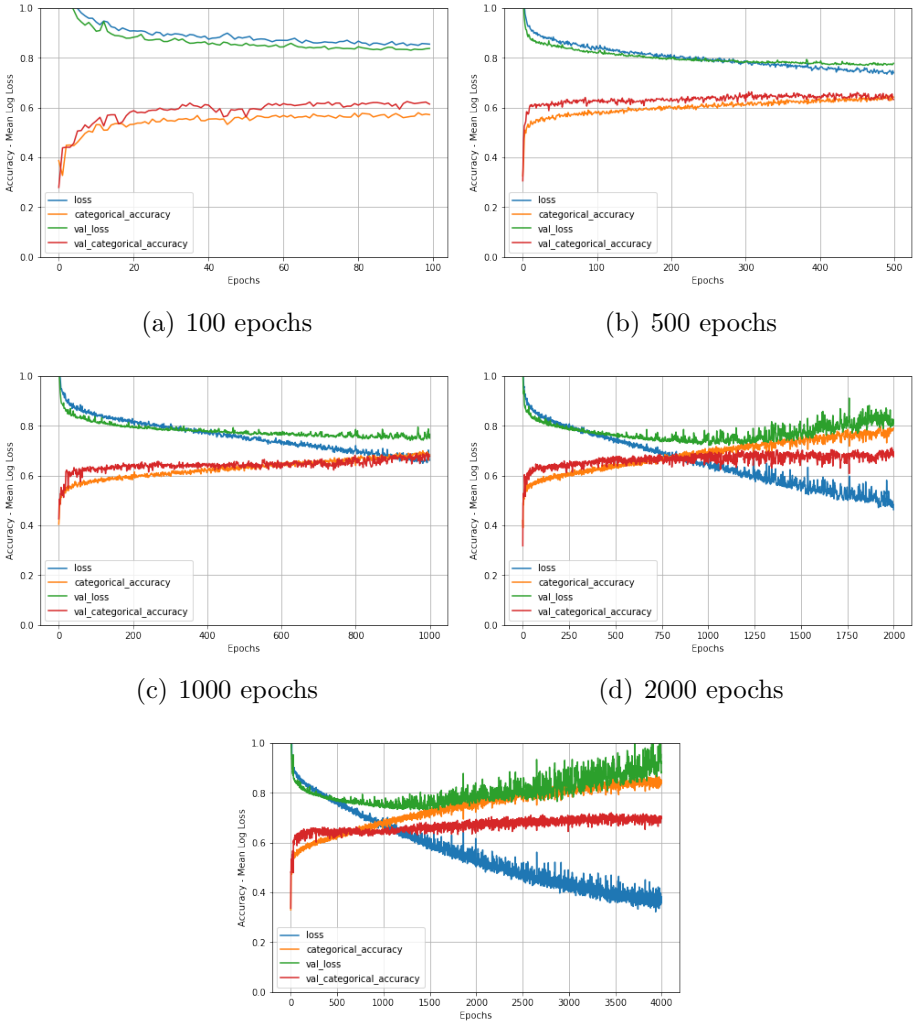
Table 8: Learning rate

As we can see in table 8, with a learning rate of 0.5 is when we get the highest testing accuracy at the same time that we are not going far away with the training accuracy. However, 0.5 it is a really high value for learning rate that probably will be changed in the future.

Epochs

Here, we will try to fix the number of epochs and to help on this decision, we will add the graphs returned by each of the networks.

Epochs	train_acc	test_acc	Time
100	57.2%	61.4%	3
500	64.2%	63.3%	11
1000	**68.4%**	**67.8%**	**20**
2000	79.1%	69.4%	40
4000	84%	69.8%	81



(a) 100 epochs

(b) 500 epochs

(c) 1000 epochs

(d) 2000 epochs

(e) 4000 epochs

Figure 1: Performance metrics evolution.

The chosen number of epochs is 1000. The performance of the network with this number of epochs could be appreciated in Figure 1 (c). This is due to the fact that, apart

from getting good testing accuracy which is at the same time near the training accuracy, it is the point where the testing loss is getting a slightly upgrading tendency, which is a sign of deterioration. Therefore, we are cutting there. Moreover, we see that if we let more epochs, the training loss will be getting very low values, sign of overfitting.

3.7 Changing optimizers

In this section we have changed the optimizer and as a result, that can be appreciated in table 9, we are going to stay with SGD and with a learning rate of 0.5.

Optimizer	**learnin rate**	**train_acc**	**test_acc**	**Time**
SGD	**0.5**	**68.4%**	**67.8%**	**20**
Momentum	0.5	83.5%	69.8%	23
Nesterov	0.5	86.5%	68.4%	22
RSMprop	0.5	45%	44.1%	23
RSMprop	0.05	73.8%	63.9%	22
RSMprop	0.005	89.3%	71.2%	23
RSMprop	0.0005	72.2%	65.5%	22
Adam	0.005	89.7%	70.6%	21
Adam	0.0005	71.6%	65.1%	21

Table 9: Optimizer.

3.8 Multiple changes

In this section we will try to modify different parameters in order to reach the highest performance of our network. These modifications will be done following a logic that will be further explained in the next lines.

The first step will be to modify the number of epochs. Since we don't know if it will be better to increase it or to reduce it, we will do both. As can be seen in the table 10, these changes have not improved our original neural network. This is why we will keep the number of epochs in the following cases until it is necessary to modify this parameter again.

The next step has been to modify the dropout. It has been done in small quantities, and like the previous one, it has been increased and decreased.

As can be seen, no better results were obtained either, so the original dropout was continued.

Since none of the previous parameters were able to improve our network, we modified the size of the batch along with the learning rate. We decided to decrease it and we saw that this time we obtained a better result than the original. We also decided to improve it by returning to the previous parameters. Observing the corresponding graph, we decided

to increase in a very small amount the number of epochs, since it seemed that still there could be no overfitting. Finally we obtained a better network with a higher accuracy than the original.

L. Rate	**Epochs**	**Initializer**	**Activation**	**Optimizer**	**Dropout**	**Batch_size**	***Train_acc***	***Test_acc***	***Time***
0.5	1000	He uniform	Relu	SGD	0.2	2048	68.4%	67.8%	20
0.5	**1050**	He uniform	Relu	SGD	0.2	2048	70%	65.5%	22
0.5	**950**	He uniform	Relu	SGD	0.2	2048	69%	64.1%	20
0.5	1000	He uniform	Relu	SGD	**0.23**	2048	66.7%	64.9%	21
0.5	1000	He uniform	Relu	SGD	**0.19**	2048	69.7%	64.9%	21
0.3	1000	He uniform	Relu	SGD	0.2	**1048**	71.5%	67%	25
0.3	**1005**	He uniform	Relu	SGD	0.2	**1048**	71.7%	69%	26

Table 10: Multiple changes.

4 Final results

As a final result, after making the previous modifications to the original neural network, a neural network with an accuracy of 71.7% in training and 69% in development has been obtained. The resulting parameters of our final network are the following:

n_epoch	1005	**Initializer**	He uniform
learning_rate	0.3	**Batch normalization**	Without
n_neurons_per_layer	[250, 75, 25]	**Regularization**	Dropout (rate=0.2)
batch_size	1048	**Optimizer**	SGD(lr=learning_rate)
Activation function	relu		

Observing the graph 2, , we can appreciate how in val_loss, it begins to take a growing direction. That means that the neural network cannot learn more. Also, if we compare the accuracy lines, they are not very far from each other, which means that we have no overfitting.

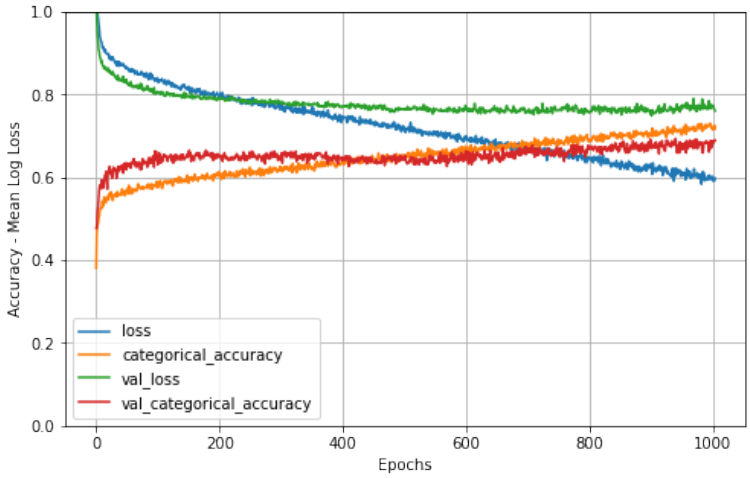


Figure 2: Accuracy - Mean Log Loss vs Epochs

Finally, to test the effectiveness of the neural network, 10% of the dataset that was reserved at the beginning of the practice for testing has been used. As it has been possible to verify, from this dataset an accuracy of 64.1% has been obtained, which is less than in development. This may be due to the fact that the neural network has learned quite well, but since it does not have a large number of data, this result together with the development result are not very high numbers. In addition, the development accuracy is more similar to the training accuracy because the data are more similar to those used to train the neural network and this causes the accuracy of the final test to be lower.

5 Conclusions

To conclude, the result obtained has reached almost 70% in almost all the tests. We would have liked to achieve a neural network capable of reaching a higher accuracy to classify the data but after the modification of the parameters, we have not been able to achieve a better classifier.

We believe that this is because we do not have the necessary amount of data to be able to construct a classifier with a higher accuracy. So it could be improved with a larger dataset.