AUGUST 29, 2021

Total words count: 3818

Machine Learning- COIY065H7

COURSEWORK REPORT

STUDENT NAME: ORLANDO TADDEO

MSC DATA SCIENCE (PT)

Email: otadde01 @mail.bbk.ac.uk

Academic Declaration

I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

1 Introduction

The aim of this coursework is to use real-world data to investigate the structure and the functioning of Artificial Neural Networks (ANNs), as well as the basics of the Deep Learning algorithms based on such networks. We will investigate the potential of a particular training algorithm developed by Alan Mosca and George Magoulas (Mosca & Magoulas, 2017). The Authors devised a new way of updating the weights of a Convolutional Neural Network (CNN), based on the well-known gradient-descent approach, called "**Weigh-wise adaptive learning rates with moving Average Estimator**" (or **WAME**, in short). We will assess its performances, evaluating how they are affected by the values of the hyperparameters on which it depends, as well as its limitations, by comparing the results to other, more common approaches usually adopted in training ANNs.

Deep Learning can be applied to a wide variety of fields, ranging from image recognition, natural language processing and speech recognition to simpler classification and regression tasks (Goodfellow, et al., 2016), (Chollet, 2018). In this work, we will limit our attention to a classification problem.

The dataset used is the one introduced in (Cortez, et al., 2009). It contains data about *Vinho Verde* wine, a product from the Minho region, in Portugal. The dataset contains some specific physicochemical characteristics for each wine sample, as well as a value for its quality. Here, the quality of the wine sample is a variable that can take values from 0 to 10, where 0 is "*very bad*" and 10 is "*excellent*". The evaluation of the wine quality was carried out via blind tastes. Details about the data will be provided in the following sections.

The Authors in (Cortez, et al., 2009) approached the problem using different methods in order to predict the wine quality. In particular, they used:

- An Artificial Neural Network with a variable number of hidden layers H, the value of which is chosen via training and evaluating a different NN for each of the values of H in the search space, defined a priori. Each network performance is evaluated using a generalisation estimate (for example, using a validation sample)
- A Support Vector Machine, with the use of kernelization. We will not focus on this approach here since this coursework regards the use of neural Networks.

It is important to note that the approach used in (Cortez, et al., 2009) is the regression one, not the classification one. This choice derives from the fact that the data in the output variable is an interval-scaled feature (Theodoridis & Koutroumbas, 2009). This means that the values

are numerical and their difference is meaningful, but their ratio is not. Putting it simply, if the actual response is, for example, equal to 4, a model that predicts 3 is better than a model that predicts 7. It is not possible to capture this in a classification setting, where when an input is misclassified, it is counted as an error, and all the errors have the same weight in the model evaluation. The solution adopted by the Authors in (Cortez, et al., 2009) is to use a regression together with a tolerance parameter $T = \{0.25, 0.5, 1.0\}$ that controls the rounding of the model's output. In this work, we used the same regression approach, simplified by the absence of the tolerance parameter T. This preserves the nature of the `quality` feature of being interval-scaled, but allows us to use directly the evaluating process implemented in keras' model.fit() method. In fact, passing the validation data to its 'validation_data' parameter, the MAE is automatically calculated for each sample and for each epoch, from the model output. If we had used the tolerance T, we would have needed to post-process the output by rounding the results, and then calculate the MAE. However, this would have prevented us from using the direct, quick and easy-to-use built-in capabilities of keras. Yet, this choice did not affect the validity of our study, since the same approach was chosen in training the NN using both RMSprop and WAME. Further details about the procedure adopted are given in the following sections.

In the paper, a detailed comparison between the results obtained using both the NN and the SVM is presented, with the SVM regression performing better.

2 Methodology, design, technical contribution

2.1 Dataset description and analysis

The dataset used here, as mentioned in the introduction, contains some specific physicochemical characteristics of wine samples, as well as a value for each sample quality. The quality of the wine sample is a variable that can take values from 0 to 10, where 0 is "*very bad*" and 10 is "*excellent*". The main dataset is actually split in two sub-datasets, one containing data about white wine only, and the other about red wine. We will focus, for our study, on the white wine data. In fact, it is large enough to provide useful results for our study about WAME. The dataset that will be used contains 4898 samples, 11 features and the target variable 'quality', described above. Also, no missing data is present.

In the following figure, the head of the dataset is shown, where we can see all the variables involved.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Figure 1

All the 11 features contained in the dataset are physicochemical characteristics of the wine, which values are real numbers. Therefore, we do not have categorical features in our dataset. The detailed description of the meaning of each physicochemical characteristic is available in (Cortez, et al., 2009). Here, we will not focus on the physical meaning of each feature, but rather on its predictive power.

If we take a closer look at the data, we can see the main statistics about it:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267	5.877909
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621	0.885639
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000	5.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000	6.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000	6.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000	9.000000

Figure 2

As it can be seen from the figure above, each variable belongs to a different range, and numerical values of one variable differ significantly from the others. This is problematic in a regression context, where variables with higher values can have a higher influence on the model predictions, without being more strongly correlated to the target variable. This issue can be solved via normalisation. One of the most common and effective data normalisation techniques is standardization, performed by subtracting from each feature vector component the respective feature's mean and diving by the respective feature's standard deviation. By doing so, all the features will have zero mean and unit standard deviation, therefore their numerical values cannot bias the regression results. We did not perform this normalisation immediately. In fact, we want to use a k-fold cross validation technique to assess the generalisation capabilities of the model we will create. To do so, the test data must be completely independent from the training data. Performing the normalisation at this step would mean that the mean would be calculated on the entire dataset, and so would happen for the standard deviation. Therefore, the normalisation needs to happen at the same time the folds are created, and separately for the test and training data. In our case, this happened later, using the KFold cross validator available in the scikit-learn package (Pedregosa, et al., 2011). In the following figures, we show the frequency histograms related to each of the 11 features, to have an understanding of their distributions.

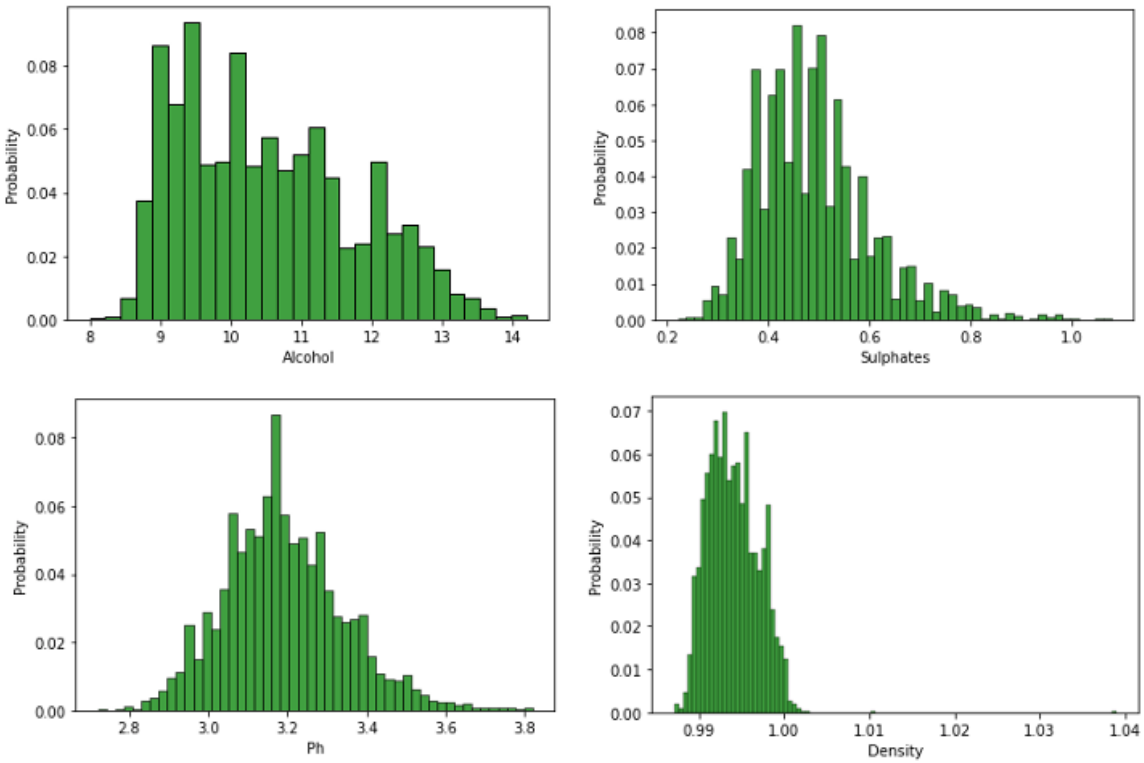

Figure 3

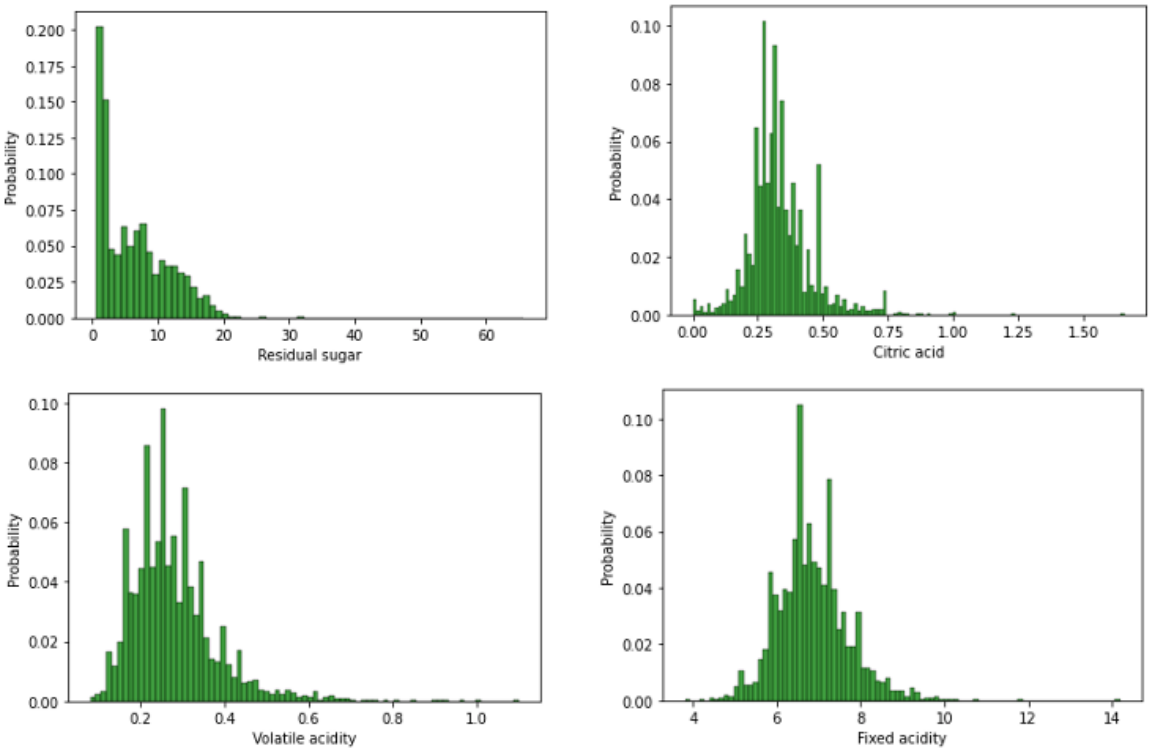

Figure 4

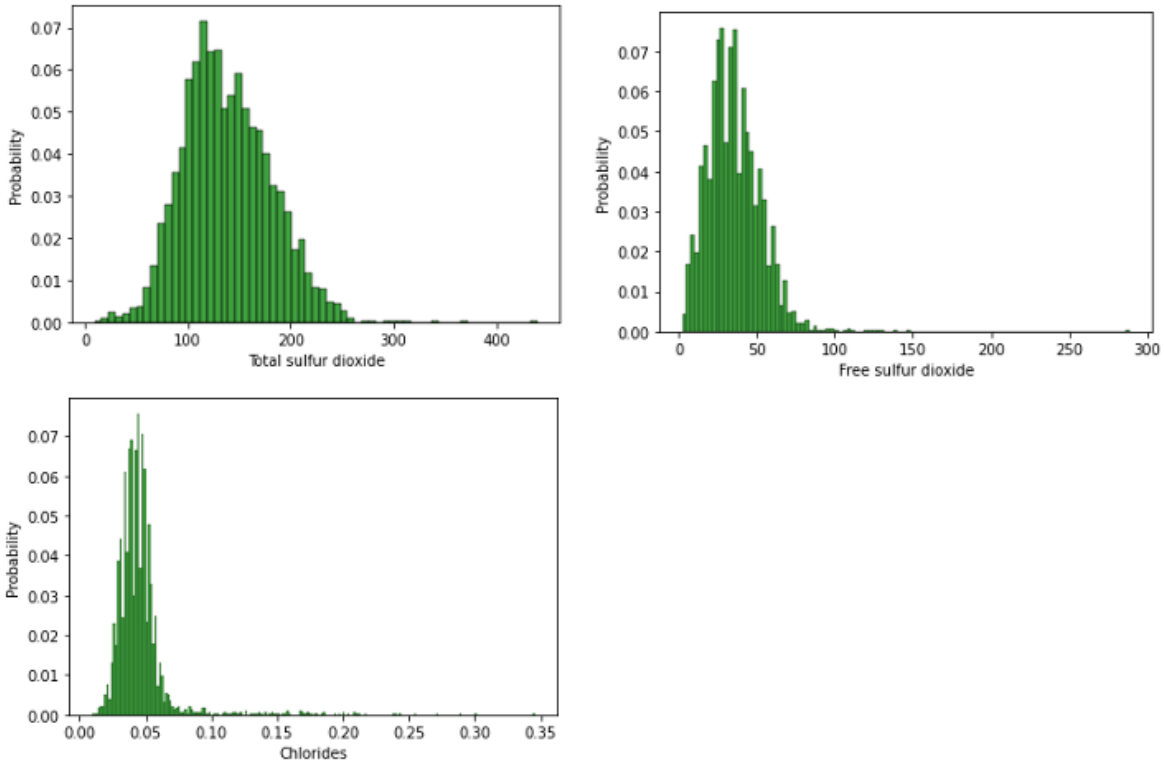

Figure 5

As it can be seen, the distributions are not Gaussian, but presents peaks with long tails. Only the `ph` feature seems to have an approximately mound-shaped distribution, whilst the `alcohol` one does not present a significant skewness, however its peak does not look to be approximately at the centre of the distribution. In our work, however, we do not take into consideration the actual shape of the feature distributions, but we limit ourselves to a visual exploratory analysis.

The last frequency histogram is the one related to the target variable, `quality`.

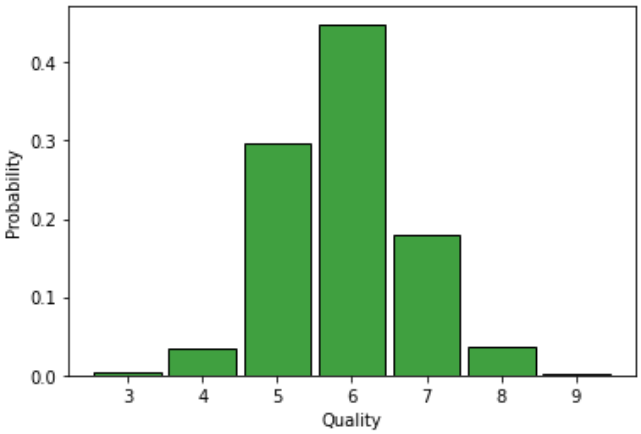

Figure 6

As shown in the figure above, not all the values in the set $\{0, 1, 2, \dots, 10\}$ are present. The distribution of the quality values look mound-shaped, with the majority of the values concentrate in the range $[5,7]$.

2.2 Optimizer implementation

The optimiser we needed to implement and test is the **Weigh-wise adaptive learning rates with moving Average Estimator**" (or **WAME**, in short). WAME is described in (Mosca & Magoulas, 2017). This algorithm is based on the gradient descent approach, like other popular optimisers available in keras (Keras special interest group, 2021), the Python package we are using to build a Deep Learning model and analyse the wine dataset. The major difference between WAME and the other available solutions is, according to the Authors, in its gradient-based update rule: a per-weight acceleration factor is used. This means that a different, adaptive learning rate is used for each of the weights in the network. At each iteration, the weight-wise learning rate is updated considering the sign of the product of the gradient at the current and the preceding step. The pseudo-code for the algorithm is shown below.

Adjusted WAME algorithm	
1:	**procedure** WAME$(\alpha, \eta_+, \eta_-, \zeta_{min}, \zeta_{max}, \lambda)$:
2:	$\theta_{ij}(0) = 0;\ Z_{ij}(0) = 0;\ \zeta_{ij}(0) = 1\ \forall i, j$
3:	**for all** $t \in [1, \dots T]$ **do**:
4:	**if** $\frac{\partial E(t)}{\partial w_{ij}} \cdot \frac{\partial E(t-1)}{\partial w_{ij}} > 0$:
5:	$\zeta_{ij}(t) = min\{\zeta_{ij}(t-1) \cdot \eta_+, \zeta_{max}\}$
6:	**else if** $\frac{\partial E(t)}{\partial w_{ij}} \cdot \frac{\partial E(t-1)}{\partial w_{ij}} < 0$:
7:	$\zeta_{ij}(t) = min\{\zeta_{ij}(t-1) \cdot \eta_-, \zeta_{min}\}$
8:	**end if**
9:	$Z_{ij}(t) = \alpha \cdot Z_{ij}(t-1) + (1-\alpha) \cdot Z_{ij}(t)$
10:	$\theta_{ij}(t) = \alpha \cdot \theta_{ij}(t-1) + (1-\alpha) \cdot \left(\frac{\partial E(t)}{\partial w_{ij}}\right)^2$
11:	$\Delta w_{ij}(t) = -\frac{\lambda}{Z_{ij}(t)} \cdot \frac{\partial E(t)}{\partial w_{ij}} \cdot \frac{1}{\theta_{ij}(t)}$
12:	$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$
13:	**end for**

As it can be seen from the pseudo code above, the WAME parameters are as follows:

- α is an exponential decay parameter, used to calculate the exponentially weighted moving average of ζ_{ij}
- ζ_{min} and ζ_{max} are clipping values for the adaptive per-weight acceleration factors, needed to avoid runway effects
- λ is the learning rate
- η_+ and η_- are further parameters of the optimiser

According to the Authors, the optimal values of these parameters have been found experimentally equal to:

$$\alpha = 0.9$$
$$\eta_+ = 1.2$$
$$\eta_- = 0.1$$
$$\zeta_{min} = 0.01$$
$$\zeta_{max} = 100$$

For the learning rate λ, instead, there is no suggested value.

For this coursework, we did not implement from scratch the WAME optimiser. Indeed, several good implementations can be found online. We chose to use the one provided by (Hill, 2020), freely accessible on GitHub. This implementation is developed by taking advantage of the TensorFlow OptimizerV2 class (The TensorFlow Authors, 2020). A WAME class is built by inheriting from OptimizerV2. In this way, this optimizer is fully integrated in the keras API, and it can be used simply by assigning an instance of the WAME class to the parameter 'optimizer' when compiling the neural network model in keras.

In the implementation considered, when a new instance of the WAME class is created, all the values of the hyperparameters mentioned above are set to what has been suggested in (Mosca & Magoulas, 2017). The value chosen for the learning rate is $\lambda = 0.0001$. However, all these parameters can be tweaked by assigning their desired value to the class' parameters.

2.3 Performance assessment

In order to understand how the WAME performs in the regression setting considered, we will:

1. Build a relatively simple neural network that features a well-known and widely used optimiser, that will act as a benchmark.
2. Use the same neural network to perform the regression task, but this time choosing WAME as the optimizer
3. Make the WAME parameters vary in order to understand how they affect the results
4. Compare the results obtained using a common optimiser to those obtained by using the WAME optimiser.

In the following Section 3, we will detail the procedure described above.

3 Experiments, findings and discussion

3.1 Baseline model

In order to understand how the WAME optimiser performs in our regression setting, we will need a model to act as a reference and a benchmark. We decide to build, using keras, a simple neural network with the following features:

- An input layer with 11 nodes;
- Two hidden layers with 32 nodes each, with ReLu activation function;
- An output layer with 1 node, with linear activation function;
- The RMSprop optimizer;
- The Mean Absolute Error (MAE) as an accuracy measure. It is defined as:

$$MAE = \frac{1}{n} \cdot \sum_{i=1}^{n} |y_i - y_{i,d}|$$

Where:

n is the number of samples presented to the network in a certain epoch;

y_i is the model output for the i-th training sample;

$y_{i,d}$ is the desired output for the i-th training sample.

It is known that the behaviour and the performance of a NN depends heavily on parameters like the number of hidden layers, the number of nodes belonging to such layers, and the activation functions used. The choices described above derive, essentially, from the necessity to find an equilibrium between the number of parameters to be investigated and the available computational power. In fact, the computations related to this coursework were carried out using Google Colab (Gogle, Inc., 2020), an online service that allows any user to run Jupyter notebooks containing Python code on fast remote GPUs. However, even using this service, training a NN as the one described above for 100 epochs and applying a typical k-fold cross validation with 10 folds, would require more than two hours. Also, if all the experiments are not carried out in a reasonably short amount of time (e.g., a few hours), the Google Colab runtime is disconnected, and all the saved objects are lost. Yet, the absence of a sufficiently powerful physical machine made the choice of such an online service inevitable. Therefore, in order to keep the computing time acceptable, a sensible choice for the NN architecture was needed, because it was found to be practically infeasible to try different architectures. The architecture described above was chosen based on the suggestions found in the technical literature, with reference to regression problems of the same kind (e.g., (Chollet, 2018)). This

allowed to investigate the behaviour of the WAME optimizer as its parameters changed, without incurring in unsustainable running times.

The process of training the NN described above was carried out using popular cross-validation techniques in order to optimize its generalisation performances. The scikit-learn KFold class was used to split the data in training and test sets, with k = 10. As pointed out above, the training and test data was normalised separately when creating each fold, in order to avoid the use of training data in calculating the mean and the standard deviation for the test data.

Being the architecture of the NN fixed, the only parameter for which it was possible to conduct the cross-validation was the number of epochs. It was initially set to a high value (i.e., 100 epochs), and the results obtained are shown in the plot below.

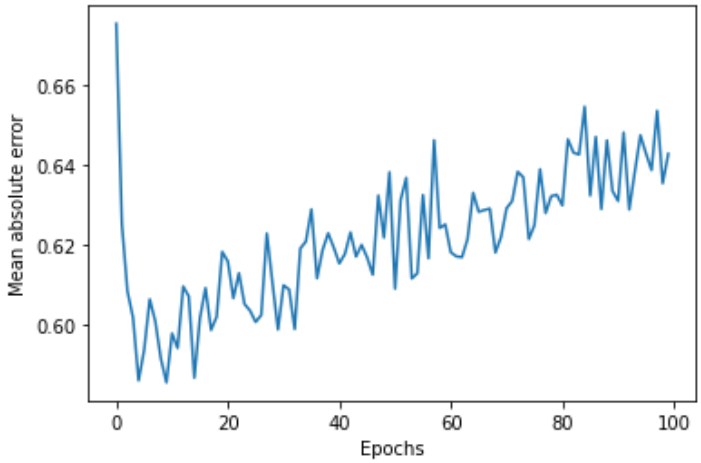

Figure 7

The Mean Absolute Error, as a function of the epoch, is shown. However, each value of the MAE on the plot is calculated by averaging the 10 values deriving by the 10 folds used in the cross-validation process. It is clear that the overfitting issue starts around 20 epochs, even though the plot is very wiggly. From a simple observation of the plot, it was decided to fix the optimal number of epochs for this model at 20. The results are shown below.

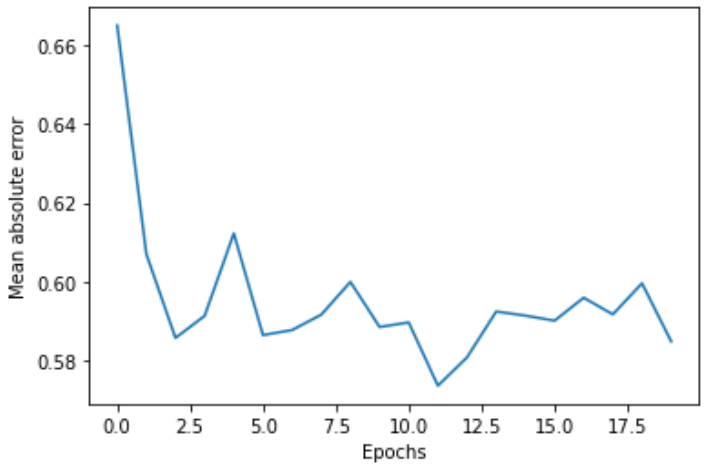

Figure 8

It can be seen that the cross-validated MAE is around 0.58. this means that, when predicting the wine quality value, the model is off by 0.58 points, on average. We consider this an acceptable result for the aims of this coursework.

It is worth highlighting again that the MAE does not take into account the rounding factor K described in Section 1 and used in (Cortez, et al., 2009). A meaningful way of using it would have been to calculate the error *after* the rounding procedure, and not directly on the output of the model, as the keras API allows us to do, via the validation_data argument of the model.fit() method.

3.2 WAME model

As mentioned above, the WAME optimiser was used on the same NN (i.e., the same architecture). We used the same cross-validation procedure as above to find an optimal value for the number of epochs to be used. We started with a high value of 200 epochs, and obtained the plot shown below.

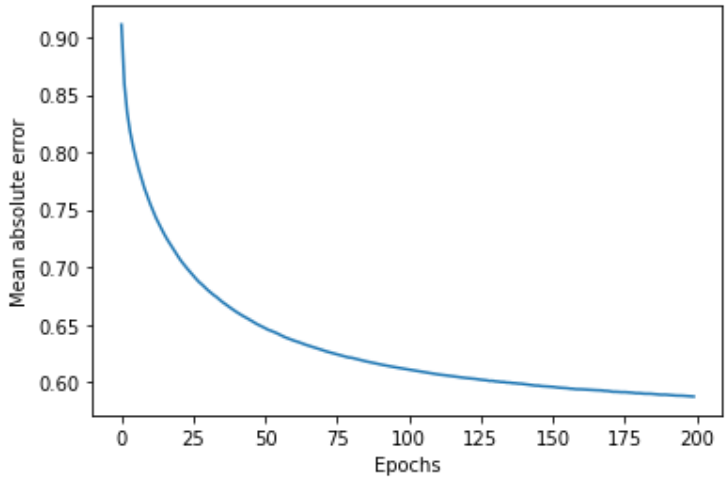

Figure 9

We can make the following observations:

- The error observed by training the network using WAME is higher than RMSprop in the first epochs, but it steadily decreases, instead of having a wiggly shape as we observed with RMSprop. Indeed, the curve is very smooth.
- The model takes a longer training time to reach its best performance, compared to the RMSprop case. For example, here we trained the network for 200 epochs, and even if it is clear that the validation error is approaching a minimum, such a minimum does not appear on the plot.
- The minimum value of the validation error attained is similar to the one obtained via RMSprop, but it may be further improved by running the training for a longer time.
- The WAME optimiser is capable of outperforming RMSprop, given the shape of the curve in its last part, and it makes also easier to identify the correct number of epochs to be used, due to the curve smoothness.

The reason why we did not increase the number of epochs is the computing time needed. In the Google Colab setting, with the GPU acceleration enabled, this task took almost three hours. Therefore, we deem a sensible choice to reduce the number of epochs to 75 in the following experiments. This would mean saving a significant computing time, without losing too much accuracy.

3.3 WAME model tuning

As mentioned above, computing power limitations do not allow us to carry out experiments including neither all the parameters of the model, nor all the parameters of the network architecture. Therefore, we choose to leave the NN architecture the same as before, and to make vary only one of the hyperparameters of the model. In particular, we chose to make the learning rate vary, since it is the only parameter for which the Authors didn't suggest a specific value, determined empirically. The default value of the learning rate int the WAMEprop class (Hill, 2020) is $\lambda = 0.0001$. We will make λ vary in a small search space, in order to keep the computing time reasonable. We will use the following values:

$$\lambda = \{0.0001, 0.0002, 0.0005, 0.001\}$$

As it can be seen, the space searched encompasses values extremely dissimilar from each other, covering one order of magnitude. Once again, a k-fold cross-validation technique with k=10 is used. The results of this experiment are summarised in the following plot.

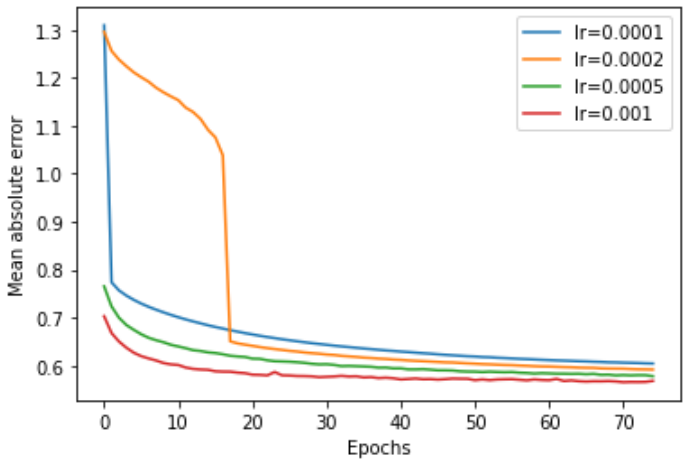

Figure 10

From the figure above, we can see the comparison among the different values of the training rate used. For all the curves, there seems to be space for improvement of the mean absolute error, even though they are clearly approaching a plateau. Although the final value attained by each curve is very similar, a different behaviour can be observed in the initial part of the training, especially for $\lambda = 0.0001$ and $\lambda = 0.0002$.

- In the case $\lambda = 0.0001$, we have an extremely rapid decline in the MAE after just the first epoch. The error decreases smoothly after this, following the behaviour observed for the other cases.
- In the case $\lambda = 0.0002$, the MAE takes much longer to reach values comparable to those obtained using the other learning rates considered (about 20 epochs). However, a sharp decline is observed even in this case, and the final result is even better than the case $\lambda = 0.0001$.
- In the remaining two cases $\lambda = 0.0005$ and $\lambda = 0.001$, the initial MAE is much lower than the other options considered, and it declines smoothly as the epochs number increases, offering the best performances among all the learning rates used.

From the figure below, we can also compare the performance of the WAME optimiser to that of the one used before, RMSprop.

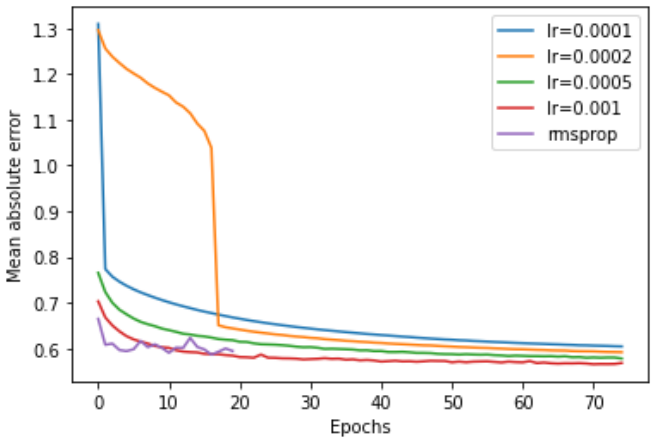

Figure 11

We can observe that the best performance of RMSprop is worse than that of WAME, but comparable indeed. On the other hand, however, performance improvements offered by WAME come at the price of a much higher number of epochs. Finally, the behaviour of the model equipped with RMSprop is less stable, meaning that there are significant oscillations in the MAE values registered, whilst WAME provides always very smooth curves.

4 Conclusions

In this work, we explored the structure and the performance of the WAME optimizer originally proposed to train convolutional neural networks by (Mosca & Magoulas, 2017). We used an implementation of it provided by (Hill, 2020), publicly available on GitHub. In order to understand how this optimiser performs in a real setting, we used a dataset including information about Portuguese *Vihno Verde* white wine. Such a dataset included several physicochemical characteristics of samples of such wine, together with a quality measure. The task was to predict the wine quality based on its characteristics, using a regression approach. In order to compare the WAME performances to a well-known and widespread optimiser, we built a model using the RMSprop optimiser, and trained it using a 10-fold cross validation approach. The same procedure was used for the same network, but this time using the WAME optimiser. For the latter we included a sensitivity analysis, investigating the effects on its performance of the only parameter for which no specific value was suggested the original article by (Mosca & Magoulas, 2017): the learning rate. Even in this case, a 10-fold cross validation was used. Finally, we compared the various results obtained by such a sensitivity analysis to each other and to the RMSprop performance. In all the cases, the mean absolute error (MAE) was used as a metric. We concluded that the performances of the two methods

were similar, even though WAME could get to a lower MAE, but at the expense of a longer training time.

All the code used to produce the results shown here is provided in a separate Jupyter notebook, submitted together with this report.

5 Bibliography

Chollet, F., 2018. *Deep Learnign with Pyhton.* s.l.:Manning.

Cortez, P. et al., 2009. Modeling wine preferences by data mining from physicochemical properties. *Decision support systems,* 47(4), pp. 547-533.

Gogle, Inc., 2020. *Gooogle Colaboratory.* [Online]

Available at: https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index

[Accessed 26 August 2021].

Goodfellow, I., Benjo, Y. & Courville, A., 2016. *Deep Learning.* s.l.:MIT Press.

Hill, R., 2020. *WAME Optimiser for convolutional neural Networks - GitHub.* [Online]

Available at: https://github.com/rlh1994/cw_wame_optimiser_cnn

[Accessed 27 August 2021].

Keras special interest group, 2021. *Keras: the Python Deep Learning API.* [Online]

Available at: https://keras.io

[Accessed 26 August 2021].

Mosca, A. & Magoulas, G. D., 2017. *Training convolutional neural networks with weigh-wise adaptive learning rates.* Bruges, s.n.

Pedregosa, F. et al., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* 12(85), p. 2825–2830.

The TensorFlow Authors, 2020. *TensorFlow Optimizer_V2.* [Online]

Available at: https://github.com/tensorflow/tensorflow/blob/v2.3.0/tensorflow/python/keras/optimizer_v2/optimizer_v2.py

[Accessed 26 August 2021].

Theodoridis, S. & Koutroumbas, K., 2009. *Pattern recognition.* s.l.:Academic Press.