2203 – Neural Network classification: a cut triangle shaped problem

Eleonora Bergamin, Filippo Boni, Stefano Campagnola, and Luca Santagata

(Dated: March 19, 2022)

The aim of this project is the analysis of different architectures of a neural network created to solve a simple binary classification task; then to determine the values of the hyperparameters so that the network has a better performance. In order to improve the network performance, different dataset sizes are studied and data augmentation techniques are considered. The tuning of the hyperparameters is made using two different methods: the classical k-fold cross-validation and the random search technique. Finally, the network is tested on a test set in order to investigate the quality of the analysis. The final results on it seem to indicate a well-performed tuning.

INTRODUCTION

The binary classification problem that will be discussed is the following. The dataset contains N points characterized by two spatial coordinates. Both the coordinates are sampled from a uniform distribution in the interval [-50, 50]. The two possible labels (0 and 1) are assigned to the points in order to satisfy the following rule:

$$\begin{cases} \text{if } x > -20 \text{ and } y > -40 \text{ and } x + y < 40 \ : f(x, y) = 1 \\ \text{else } : f(x, y) = 0 \end{cases}$$

The task of the neural network is to predict the correct label for each point and thus to understand if that point is inside this triangle shaped domain. In order to solve this classification problem a neural network with the following characteristics is considered:

optimizer	**batch size**	**input layer**
Adam	50	(2, ReLU)
hidden layer 1	**hidden layer 2**	**output layer**
(20, ReLU)	(20, ReLU)	(1, Sigmoid)

TABLE I. Initial network

Then dropout method is applied to the last hidden layer.
The steps followed in the report are the following:

1. Analysis of the performance of the network as a result of the variation of the training (and validation) set size.

2. Analysis of the performance of the network with augmented data added to the original training set.

3. Tuning of the main hyperparameters of the network: optimizer, batch size, number of layers and neurons for each layer, activation function, dropout rate, weights initialization.

4. Tuning the hyperparameters using the random search method.

MATERIALS AND METHODS

Data pre-processing Every dataset is rescaled in order to deal with a unitary box [-1, 1]x[-1, 1]. Standardization, instead, would have led to a distortion in the positional information of the samples.

The dataset dedicated to the analysis is always divided into training and validation set with an $80 - 20$ proportion. Its dimension N=$N_{val} + N_{train}$ is set in the following manner: $N = 4000$ for the augmentation discussion and $N = 10000$ for the hyperparameters tuning. From the results of the analysis of the network performance with respect to the number of samples, the number of epochs is fixed at $N_{epoch} = 400$ for every training. A test set of size $N_{test} = 4000$ is used at the very end of the analysis to verify the quality of it.

Cost function and accuracy The cost function used for the training of the network is the *binary cross entropy*; The *accuracy* is the metric used to better understand the network performance.

Number of samples In order to study how the network behaves varying the number of samples, different sizes of the dataset are analyzed: N = [2000, 3000, 4000, 6000, 8000, 10000, 15000].
Then the aim is to study the response of the network to an augmented training set, since data augmentation is known to be useful for avoiding overfitting and for improving the generalization properties of the network. Since the dataset consists of points, which are classified on the basis of their spatial coordinates, the only transformation that a datum can undergo is a translation. The procedure followed, then, is to apply a small random shift to each sample of the training set: the augmented dataset then will be twice the size of the original one. The random displacements are different along each coordinate of each sample. Two different probability distributions are considered to generate the shifts: a uniform distribution in the interval $[-1, 1]$, and a normal distribution with $\mu = 0$ and $\sigma^2 = 1$.

Cross-validation For the hyperparameters tuning, the technique of the k-fold cross-validation is used and applied separately to each of them. The choice of making independent cross-validations allows studying more options for each hyperparameter, without having a huge computational power. The tuning order is presented in the introduction. It is to note that, once a hyperparameter is chosen with tuning, it is fixed for the subsequent analysis.

Optimizers First, the optimizer is selected; the ones considered are *SGD*, *RMSprop*, *Adagrad*, *Adadelta*, *Adam*, *Adamax*, *Nadam*. In the experiments, their default parameters are not modified. It is to note that, as stated in [1], picking an optimizer using its default parameters (such as its learning rate) corresponds to tuning the hyperparameters of a specific optimizer. It should be noted that the performance of optimizers varies by changing the task, therefore the choice is closely linked to the task to be carried out.

Batch size To help speed up network convergence and to ensure sufficient stability, each update in the learning process takes place on a small number of samples (batch) and not on the whole training set. The tested batch sizes are [10, 25, 50, 100, 250, 500].

Architecture The basic network to work on is again the initial network already described. When the function describing the data is not very complex (as in this specific case), it is known that models with less capacity can perform just as well as models with more parameters. Moreover, it is not desired to excessively increase the complexity of the model. As a consequence, the maximum number of hidden layers and neurons per layer is respectively four and 40.

Activation function There exist two main families of activation functions, that of the sigmoid and that of the ReLU. Sigmoid type functions can interpret the firing of a neuron, but can lead to the gradient vanishing during the training, hindering the learning of the network; ReLU type functions, instead, prevent the gradient from vanishing. During the tuning, the activation function is fixed for the input and output layer respectively to ReLU and sigmoid, and the same activation function (*Sigmoid*, *Tanh*, *Softmax*, *ReLU*, *Leaky ReLU*, *ELU*, *Swish* or *SELU*) is applied to all the hidden layers.

Dropout It is applied only to the last hidden layer; the fractions of units to drop (dropout rate) tested are [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]. This means that for all the considered choices each neuron in the layer is retained with a probability equal to or greater than 0.5.

Weights initialization The techniques consideredinvolve the usage of weights chosen from a properly scaled uniform or normal distribution. The method applied depends on the activation function type. Xavier (also called Glorot) initialization [5] is used if the function belongs to the sigmoid family, as it helps prevent gradient vanishing. For the ReLU type functions, the best method of initializing weights is He [4]. *Glorot uniform*, *Glorot normal*, *He uniform* and *He normal* are experimented. The initialization method is chosen based on the result obtained by the tuning of the activation function.

Grid and random search There are several techniques to optimize hyperparameters. When tuning each parameter at a time the interaction between the different parameters is neglected, which instead can also play an important role [2]. To take this effect into account, a grid search can be used. This is an exhaustive search as all combinations of hyperparameters are tested, so one of its weaknesses is dimensionality. A greater number of hyperparameters or their values substantially increases the number of combinations of hyperparameters, and therefore also the training time. To overcome this problem, random search [6] was introduced. In this case not all combinations of hyperparameters are considered, but only a randomly chosen subselection. Due to randomness, the entire parameter space is likely to be sampled, and therefore a combination that results in near-optimal performance is likely to be selected. In previous studies, this type of search has been shown to return better results than grid search and to have a lower computational cost. It is therefore the research model detailed here. To carry out the optimization we consider a network with a variable number of layers between 1 and 4 each with the same number 1-40 of neurons, activation function in the hidden layers of the ReLU family, dropout rate in range $0.1 - 0.5$, and batch size and optimizers same as before.

RESULTS

Performance and dataset dimension The results of the variation of $N{=}N_{val} + N_{train}$ are shown in figure 1(a). It is evident that the network gets better results when $N \geq 6000$ is considered, indeed the accuracy reaches values close to 1. If this condition is not met, the network stops improving after about 150 epochs, and both the loss and the accuracy stabilize around a fixed value (figure 1(a)). This means that the data provided are not enough for the network to learn. It is interesting to notice that before reaching 200-250 epochs the results are similar for every size, and only after that number of epochs it is possible to notice an accuracy improvement when $N \geq 6000$. The only exception is $N = 10000$ where the improvement with respect to the others starts around 150 epochs. Comparing for various dataset sizes the loss value obtained during training with the loss value on the validation set, it can be seen that they follow the same decreasing trend and do not reach zero, therefore the training performed does not result in overfitting (figure 1(b)). This may be due to the fact that the dataset is large enough in all cases, and the number of samples is much greater than the number of network parameters. For these reasons, therefore, as stated in the methods, the chosen number of epochs is 400, and the dimension of N is $N = 10000$ ($N_{train} = 8000$ and $N_{val} = 2000$), with the exception of the data augmentation.

Data augmentation The aim is now to investigate whether augmentation is as effective as using new data. In this case the augmentation does not lead to an improvement in the network performance. Indeed the methods used return an accuracy of approximately 0.92 and

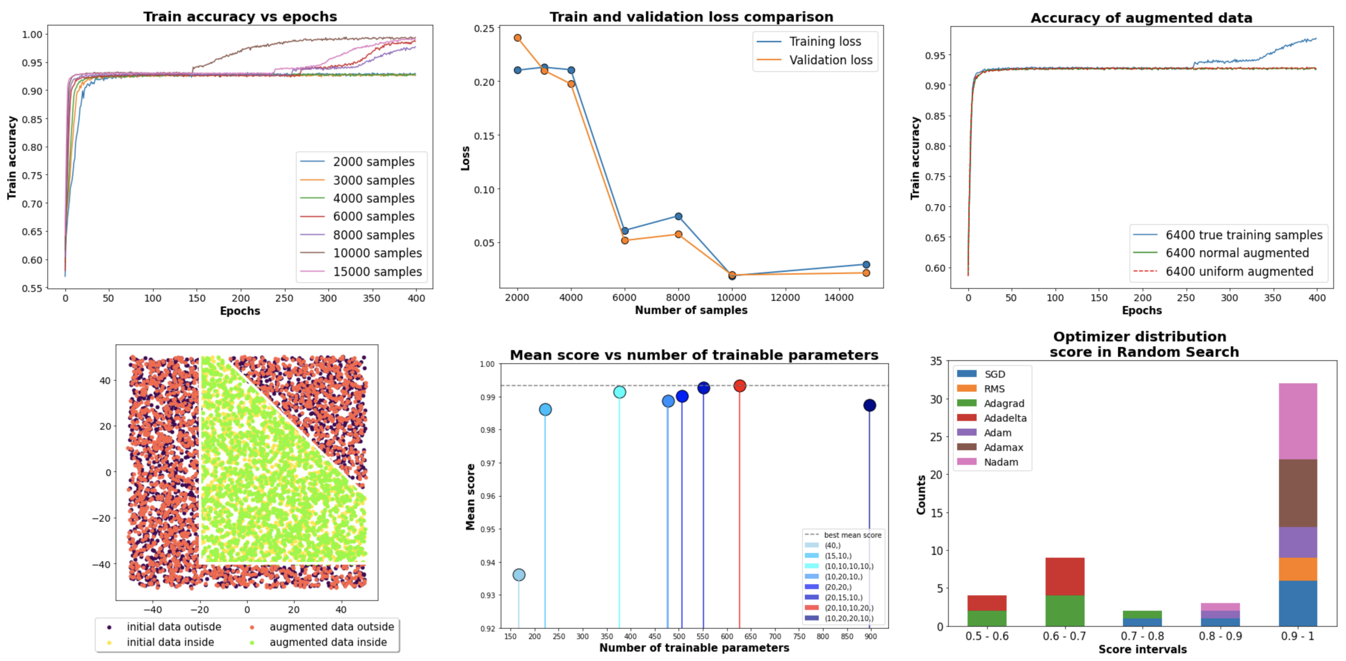

FIG. 1. (a) Evolution of the network training accuracy as a function of the epochs for datasets of different size. (b) Final value of training and validation loss for different dataset dimensions. (c) Training accuracy of data augmented with two different methods and of a training set of similar size. (d) Original and augmented data with uniform distrivution method in their domain. (e) Comparison of the scores from the cross-validation on different network architectures. (f) Optimizer distribution score obtained with random search.

0.93, which are values close to the ones obtained with the non augmented dataset of size $N_{train} = 3200$. In figure 1(c) are shown the accuracy related to the two augmentation methods and the one obtained using a set of the same dimension of the augmented one of newly generated data points. It can be observed how the accuracy on the set of original data continues to increase as the number of epochs increases, while that relating to the augmented data stabilizes around a certain value. This might be due to a correlation of the augmented data with the original ones.

Hyperparameters selection The result of the tuning performed are shown in the table:

optimizer	**batch size**	**architecture**
RMSprop	25	(10, 10, 10, 10)
activation	**dropout rate**	**weights initialization**
ReLU	0.05	He uniform, Glorot uniform

TABLE II. Tuning results

For the optimizer it is seen that the best is RMSprop, however it is noted that it does not blatantly outperform the others; in fact, it can be noticed that Adam and Nadam have comparable performances, confirming Adam as a good optimizer [1][3], and this is not a surprise since Adam is an update of the RMSprop optimizer. Regarding the batch size, it is observed that small batch sizes are preferred. This may indicate that the loss function profile has different local minima: the increased stochasticity due to a reduced number of samples helps the algorithm to get out of the local minima.

The results are also interesting with regard to the activation function. The best performance is obtained using the ReLU and Leaky ReLU functions. However, it is observed that all the other functions tested result in the same performances, including functions belonging to the family of the sigmoid. This might be due to the fact that the network considered is not very deep, and therefore the vanishing of the gradient problem is not substantial.

Considering the network architecture, for the different configurations chosen, figure 1(e) shows the different score values obtained, as a function of the number of learnable parameters. It is noted that models with a large number of parameters, as already hypothesized, have similar performances to architectures corresponding to fewer parameters. Therefore, a model able to reconcile a small number of parameters with a good score is chosen: four layers each consisting of 10 neurons.

As for the dropout rate, the selected value is 0.05, but all the experimented dropout rates give a score value higher than 0.95. This may indicate that the network does not overfit and that there is no over-dependence on a single neuron as the performance is similar with both high and low dropout rates.

Moving on to the initialization of the weights, it is observed that the best distribution for both ReLU and sigmoid functions is uniform. In any case, as long as an appropriate initialization (He vs Xavier) is chosen for the family of activation functions, the score is optimal regardless of the choice of the distribution from which the weights are drawn.

Test set results After the choice of the hyperparameters, the performance of the network is evaluated on a test set. It can be noticed how the network correctly classifies the majority of the points (figure 2): indeed the accuracy of the network on the test set is 0.993. While this is a good result, the network is not actually able to correctly label all the points closer to the edges of the triangle.

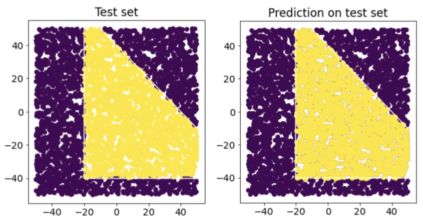

FIG. 2. Predictions on test set

Considerations about the random seed During the analysis of the network a random seed is fixed in order to make results reproducible. However a dependence of some results, such as the hyperparameters, on the value of the random seed is noticed. The results using different random seeds (rs) are shown in the table below:

	rs1	**rs2**	**rs3**
optimizer	Nadam	Nadam	RMSprop
batch size	250	100	10
architecture	(10,20,10)	(10,10,10,10)	(20,15,10)
activation	ReLU	ReLU	Leaky ReLU
dropout	0.4	0.2	0.05
weight init	Glorot norm He norm	Glorot unif He norm	Glorot norm He unif

TABLE III. Random Seed variation results

It is also noticed that sometimes some data augmentation methods greatly increase the network performance, which results in an accuracy of 0.98-0.99. To summarize, to obtain reproducibility there is no other choice than to fix the random seed; however, this analysis highlights that the results and behavior of the network may also be dependent on the seed.

Random search The result of the random search is shown in the table IV.

The results found are consistent with what has already been found by tuning the individual parameters. As can be seen from figure 1(f), in addition to the optimizer that determined the best performances, Adam and Nadam also confirm themselves as good optimizers. It is also observed that the best results are obtained by architectures with a greater number of layers (three or four), and a large number of neurons per layer ($\geq$ 15). It is noted that the batch size is very variable in the combinations that have determined the best results, oscillating between 10 and 100. The dropout rate is also variable in the tested architectures indicating its low influence on the network performance. Therefore, apart from the case of the model architecture, consistent results are not observed in the choice of hyperparameters: this may mean that these quantities have no particular influence on the performance of the network. It should be noted that by carrying out a random search, the effect of the mutual interaction of the network hyperparameters is taken into account; this may be another factor affecting the results obtained.

After tuning, the new network is tested: the accuracy is 0.996, comparable with what has already been found. However, from the experiments performed it is found that this method has a faster convergence.

optimizer	**batch size**	**architecture**
Adamax	50	(24, 24, 24, 24)
activation	**dropout rate**	
SELU	0.3	

TABLE IV. Random Search results

CONCLUSIONS

This paper analyzes how the neural network considered behaves when the amount of data available varies, and how it is possible to perform a tuning of the hyperparameters of the network.

The analysis shows that having more data available for training positively influences the performance of the network.

The creation of new data with the augmentation technique does not lead to a significant betterment of performances, especially if compared with the case using newly generated training data.

Two different methods for the tuning of hyperparameters are applied. Both of them result in a very good performance when tested with the test set. However it is importance to notice that with the random search the final accuracy on the test set is slightly better than the one obtained with the first method. In addition random search allows to consider the interaction between the hyperparameters without excessive computational time. In light of this analysis, therefore, this type of research may be preferable to the tuning of individual parameters.

Finally it is important to remember that the random seed chosen can influence the final outputs.

[1] Robin M. Schmidt, Frank Schneider, Philipp Hennig, Descending through a Crowded Valley —Benchmarking Deep Learning Optimizers, https://arxiv.org/abs/2007.01547 (2021)

[2] Klaus Greff, Rupesh K. Srivastava, Jan Koutn´ık, Bas R. Steunebrink, Jurgen Schmidhuber , LSTM: A Search Space Odyssey, *Transactions on Neural Network and Learning Systems*, http://arxiv.org/abs/1503.04069 (2017)

[3] Diederik P. Kingma, Jimmy Lei Ba , Adam: a method for Stochastic Optimization, https://arxiv.org/abs/1412.6980 (2017)

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, http://arxiv.org/abs/1502.01852 (2015)

[5] Xavier Glorot, Yoshua Bengio , Understanding the difficulty of training deep feedforward neural networks, *Université de Montréal*, *Montréal*, *Québec*, http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf (2010)

[6] James Bergstra, Yoshua Bengio, Random Search for Hyper-Parameter Optimization, *Journal of Machine Learning Research*, https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf (2012)