GROUP 2213

–

Performance Improvements of Deep Neural Networks for Non-Linear Classification Tasks

Amico Tommaso, Lazzari Andrea, Zinesi Paolo, and Zomer Nicola

(Dated: March 19, 2022)

Deep Neural Networks offer a great advantage in terms of classification performances with respect to other algorithms available nowadays. However, a careful tuning of the hyper-parameters is necessary to fully exploit this advantage. — This short paper presents a general strategy for improving the performances of a DNN and shows the results obtained with two different non-linear functions. Concepts such as data augmentation, grid search, fine tuning of both network architecture and minimization algorithm, and initialization strategies are explored. Our study reveals that in the considered task the network architecture plays a main role with respect to the other hyper-parameters, which can still be strategically investigated in more complex classification problems.

INTRODUCTION

The classification task consists in the ability of correctly assigning a label to each of the data given as an input. The domain to which input data belongs and the possible labels that can be assigned depend on the considered problem. In our work we focus on the simple example of classifying two-dimensional points randomly generated inside a square and labeled according to a non-linear function. As a result, the possible outcomes are two: 0 and 1.

In order to accomplish the classification task we make use of a fully-connected feed-forward Deep Neural Network (DNN). The output of each neuron is computed as the weighted sum of the outputs of the neurons in the previous layer, followed by the application of a non-linear activation function. The classification goal is to find suitable weights that allow the DNN to correctly classify the input data.

We build the Deep Neural Network with *Keras* and *TensorFlow* libraries, but we also use tools from *Scikit-learn* for example to perform model selection.

The generated data are split into training and test data (see Sect. Methods for details). We make use of this two disjoint sets, respectively, to train our network by minimizing a loss function and to evaluate the model performances on a set not used during training. The test procedure takes place at the end of training and is used to obtain a more accurate estimation of the true error.

The search for an optimal model proceeds in a similar way for both non-linear functions. At first we define a starting model ("base model") with a given architecture, optimization algorithm and characterizing parameters. All those starting choices come from both informed guesses (when available) and random guesses. Then, the base model is improved. Possible ways to improve the model are the augmentation of training data, the fine tuning of hyper-parameters of the model via a grid search procedure, and the exploration of different initializations of weights.

Regarding the selection of the hyper-parameters we use the *Scikit-learn* routine *GridSearchCV*. This routine divides the training data into K independent folds (in the default case $K = 4$), chooses one fold at a time to be the validation set while it trains the algorithm on the union of the others. The final error is computed as the average of the K errors and the best combination of hyper-parameters is the one that minimizes it.

This paper is organized as following. In Sect. Methods we define all the tools used in the analysis, such as the two non-linear functions, the data augmentation algorithm, the error metrics, the parameters to be optimized in the grid search, and the possible weights initializations. Then, in Sect. Results we show the results of the different models and strategies, supported by graphs and tables. We then conclude by summarizing our main achievements and by suggesting some possible outlooks to improve the model (see Conclusions).

METHODS

The training sets chosen for the analysis are obtained by drawing random data uniformly distributed in the square of coordinates $(x_1, x_2) \in [-50, 50] \times [-50, 50]$ using the *numpy.random.random* routine. The default number of 2D samples is $N = 4000$, divided $80\% - 20\%$ into training and test samples. In Sect. Results the consequences of having different number of samples are shown.

The first non-linear function used to assign labels to the training data is

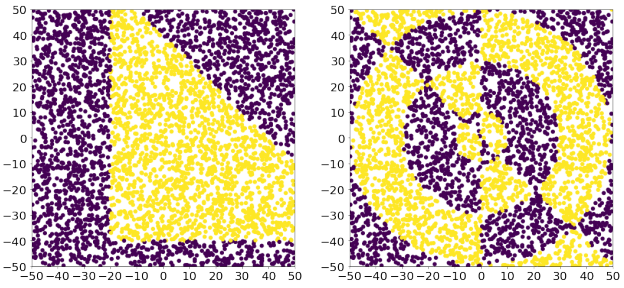
$$f(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 > -20 \wedge x_2 > -40 \wedge x_1 + x_2 < 40 \\ 0 & \text{otherwise} \end{cases}$$

The results of the labelling can be seen on Fig. 1(a). The

second non-linear function used to label training data is

$$g(x_1,\, x_2) = \begin{cases} & \text{if } \operatorname{sign}(x_1 + x_2) \cdot \operatorname{sign}(x_1) \\ 1 & \cdot \cos\left(\frac{\sqrt{x_1^2+x_2^2}}{2\pi}\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Similarly, the results of this second labelling function can be seen on Fig. 1(b).



(a) (b)

FIG. 1: Graphical Representation of the two analysed non-linear functions

After having obtained the numerical labels (0 and 1) of the input data from the two non-linear functions, those labels are one-hot encoded into the canonical base $(1, 0), (0, 1)$.

A possible strategy to overcome the scarcity of training data is to apply data augmentation on the available samples. The augmentation we perform consists in shifting some of the training samples along random directions and to append the shifted points to the original training set. The magnitude of each shift is sampled from a Gaussian with mean $\mu = 0.1$ and variance $\sigma^2 = 0.01$. The classification labels of the *artificial* new data remains the same of their corresponding original ones. This procedure is not equivalent to generate new training data, as the augmented data are strongly correlated to the original ones. In Sect. Results we discuss the impact of data augmentation to the training procedure.

Once the training data are generated, we need to define a metric that evaluates the performances of the model on the training and test data. In this short paper we consider only the cross-entropy and accuracy metrics. Cross-entropy (also called log-loss) is calculated from the probability that a sample belongs to each of the possible classes. As the probability that the model classifies correctly a given sample tends to 0, the computed value of the cross-entropy tends to infinity and the learning algorithm will move away from that model. Instead, accuracy is calculated as the ratio of correct classifications over the total number of classifications performed. While the learning algorithm takes care of minimizing the cross-entropy loss (renamed only loss from now on), even the accuracy turns out to have a value close to its minimum. We use the accuracy metrics to show clearly the percentage of misclassified samples for each trained model.

The definition of the loss and of the accuracy metrics, together with a first guess of the model parameters, allow to obtain a preliminary classification model. In order to improve the model, a grid of (hyper-)parameters is defined and using the grid search combined with K-fold cross-validation we can find the combination of parameters that minimizes the loss.

Parameters are divided into architecture parameters and optimizer parameters. Architecture parameters define the structure of the DNN and strongly depend on the problem to tackle. We considered the following architecture parameters.

- **Layers size**. Number of (hidden) layers and the number of neurons on each of them.
- **Activation function**. Non-linear function that for each neuron transforms the weighted sums of the input neurons into the output.
- **Weight constraint**. Constraint on the maximum value a weight can assume, to avoid the strong dependence of the model on a single weight.
- **Dropout rate**. Fraction of neurons to be randomly dropped during each training step, to avoid the strong dependence of the net to a single neuron.

Optimizer parameters, instead, select the best optimizer in order to minimize the loss. These parameters depend weakly on the considered problem but they are fundamental to reach convergence in the training procedure. We considered the following optimizer parameters.

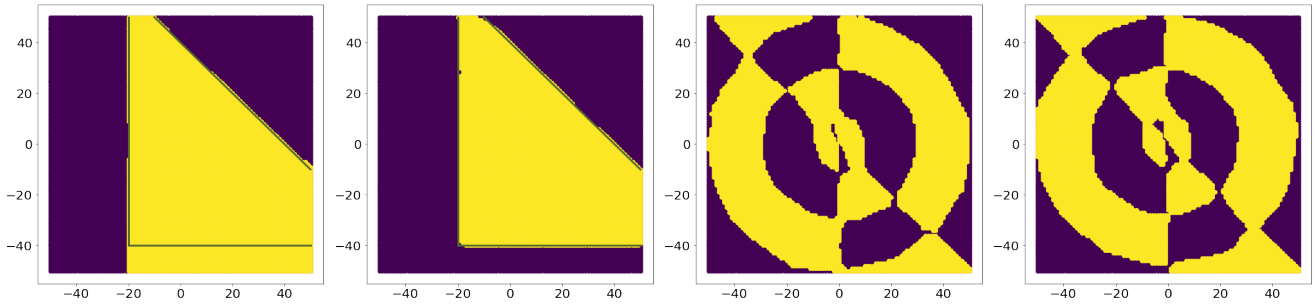
- **Optimizer**. Minimization algorithm used to find the loss minimum.
- **Learning rate**. Step size of updates during the iteration of the minimization algorithm.
- **Momentum**. Parameter in the update rule that accelerates gradient descent in the relevant direction and dampens oscillations.

We finally test as possible weights initializations all-zeros, random, LeCun, Glorot and He strategies, according to [2].

RESULTS

First non-linear function

We start the analysis of the first non-linear function by defining a model using an architecture with hidden layers' structure $(2-20-20)$, sigmoid activation function, dropout $= 0.1$. This base model is not able to perform a satisfying classification, as it can be seen from Fig. 2(a).



(a) (b) (c) (d)

FIG. 2: Effect of the grid search in the improvement of a model

This poor result might be due to the presence of only 2 neurons in the first hidden layer, which implies the model is able to intersect at most 2 hyperplanes. To overcome this issue we define a different model by simply adding a third neuron to the first layer. With this small change our metrics greatly improved, as Fig. 2(b) and Table I show.

	Train Acc.	Train Loss	Test Acc.	Test Loss
$(2-20-20)$	0.9275	0.1511	0.9312	0.1385
$(3-20-20)$	0.9909	0.0217	0.9937	0.0153

TABLE I: Metrics of the first two models

This result bring us to the conclusion that an extensive grid search is not needed, so the cross validation is used to tune only a limited set of parameters. We try Adam and RMSprop as optimizers, sigmoid and relu as activation functions and $(3-20-20)$, $(10-10-5)$, $(50-50)$ as architectures. While other parameters such as the weight constraint, the learning rate and the momentum are kept fixed.

The best model found by the grid search has layer size $(50-50)$, optimizer RMSprop, activation function relu. The metrics obtained by compiling the model fitted with the optimal set of parameters are shown in Table II.

	Train Acc.	Train Loss	Test Acc.	Test Loss
(50-50)	0.9919	0.0142	0.9950	0.0112

TABLE II: Performance of the best model found

We can see that the two models, before and after cross validation, have very similar performances. In fact, both networks are optimal in terms of accuracy and loss, also due to the goodness of the initial one. Finally, as we can see in Fig. 3, the base model after grid search improves for both the accuracy and the loss.

In Fig. 4 are shown the values of test accuracy and test loss, each one identified by a different size N of the generated dataset. As expected, increasing N both the test accuracy and the test loss improve, while the opposite

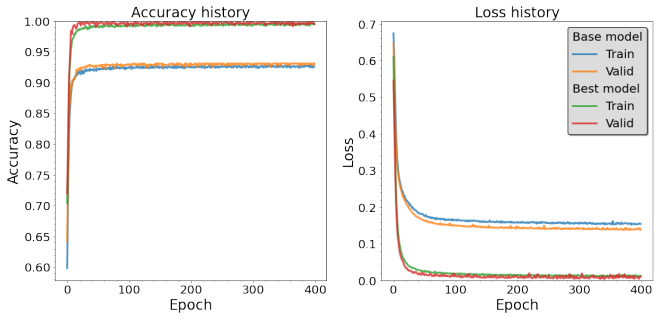


FIG. 3: Accuracy and loss of the base and best models as a function of the epochs

behavior occurs by decreasing it. The result of applying data augmentation depends on the initial value of N: if small it helps a bit, if large there is a concrete risk of reducing the performances, as some generated points are incorrectly classified during their generation and this introduces an error in the training set.

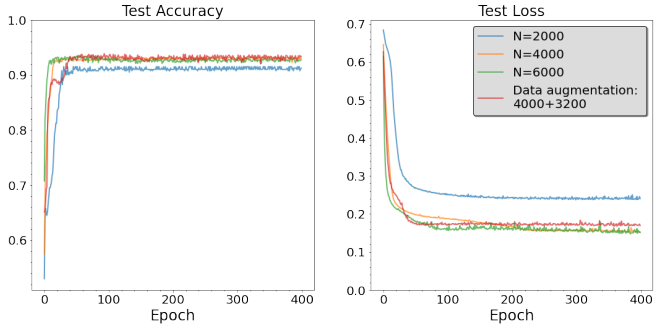


FIG. 4: Test accuracy and test loss of the base model for different values of N

Second non-linear function

Similarly, for the second non-linear function we choose a base model with initial architecture $(20-20-20)$, Adam as optimizer, dropout $= 0.1$ and sigmoid activation function. The complexity of this function may require

a more in-depth analysis, that we divide into two grid searches to reduce the computational cost and have a better understanding of what is going on.

The first grid search is similar to the one presented for the first non-linear function, except that we include the weight constraint, increase the set of possible layers' structures, add Nesterov to the optimizers and elu and tanh to the activation functions.

In the second grid, after fixing the best network architecture, we fine tune the specific optimizer together with the learning rate and the momentum. Generally, it is a good idea to also include the number of epochs as there is a inter-dependency between the learning rate, the batch size and the number of epochs.

The first grid search finds the best parameters to be $(30-30-30)$ for the layer size, dropout $= 0.05$, weight constraint $= 1$, RMSprop optimizer and relu activation function, while the fine-tuning grid search finds a learning rate of 0.001, momentum $= 0$ and 400 epochs. The metrics of the base model and of the fine-tuned model are shown in Table III. The improvements achieved tuning the hyper-parameters are shown in the differences between the graphical representations of the networks predictions in Fig. 2(c)-2(d).

	Train Acc.	Train Loss	Test Acc.	Test Loss
(20-20-20)	0.9800	0.0602	0.9688	0.1180
(30-30-30)	0.9650	0.0638	0.9787	0.1504

TABLE III: Metrics of the neural network before and after the Grid Search with Fine Tuning

As a last point of the analysis we investigate if a different method of initialization of the network weights can correspond to better performances. Regarding the possible choices about the initialization techniques, we consider the benefits and the drawbacks illustrated in [2]. The results are shown in the following table:

	Train Acc.	Train Loss	Test Acc.	Test Loss
All-zeros	0.5138	0.6928	0.5150	0.6927
Random	0.9150	0.2221	0.9062	0.2221
LeCun	0.9322	0.1406	0.9300	0.1776
Glorot	0.9287	0.1671	0.9150	0.2210
He	0.9541	0.1284	0.9500	0.1769

TABLE IV: Metrics for different initializations of the weights

CONCLUSIONS

From the analysis of the classification of the data labelled according to the first non-linear function, we can say that by solving the problem related to the number of neurons of the first hidden layer, we obtain a model with satisfying performances. The improvements achieved with cross validation do not justify the process itself and a great predictor can be obtained just with a careful choice of the architecture, without spending too much time in tuning the other parameters of the network.

Regarding the second non-linear function, its complexity invites us to perform a more in-depth research of the optimal hyper-parameters for the model. However, the performance of the model obtained with an in-depth cross validation are far to be a significant improvement of the base model, which has an architecture chosen thanks to the lesson learned on the first function analysis.

We can say that, in this kind of task, the choice of the architecture is fundamental for the well being of the model. Regarding the other parameters, an informed guess on their initial value is enough and the computational time required by the grid search is not justified.

As a final result we present the two confusion matrices (Fig. 5) of the best model found for the two non-linear functions: the amount of correctly labelled points are shown in the diagonal whereas the misclassified samples are present in the off-diagonal cells.

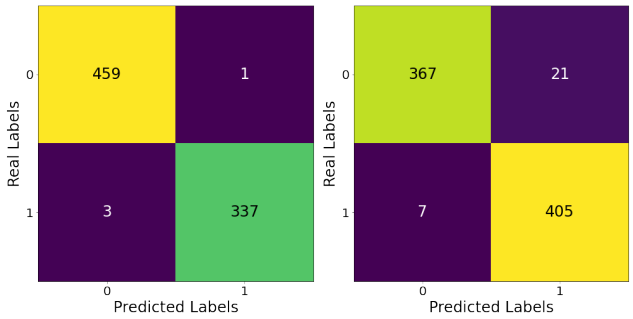


FIG. 5: Confusion Matrices: best model of the first non-linear function (on the left) and of the second non-linear function (on the right)

As a possible extension of our work we could look into the generalization properties of our models in the classification of other non-linear functions or, more in general, of similar tasks.

REFERENCES

[1] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," (2019).

[2] W. Boulila, M. Driss, M. Al-Sarem, F. Saeed, and M. Krichen, "Weight initialization techniques for deep learning algorithms in remote sensing: Recent trends and future perspectives," (2021).

[3] Tensorflow documentation, https://www.tensorflow.org/api_docs/python/tf.

[4] Keras documentation, https://keras.io/api/.

[5] Scikit-learn documentation, https://scikit-learn.org/stable/.