Group 02 - Tuning a simple Deep Neural Network

Francesco Pio Barone, Gianmarco Nagaro Quiroz, Daniele Ninni, and Lorenzo Valentini
(Dated: March 22, 2022)

In this report we discuss the tuning of a simple Deep Neural Network model for the classification of 2D data patterns. Through several steps, we select a model capable of achieving good performances (classification error) whilst reducing its overall complexity. Eventually, we test such model on increasingly more elaborated data patterns to prove that its effectiveness holds. We also show that the theoretical bound on the expressive power of a neural network, about the minimum number of layers and neurons, is an helpful suggestion in order to reduce the number of epochs necessary to achieve an optimal target accuracy.

INTRODUCTION

The increasing computational power of modern calculators has greatly contributed to the spread of more complex machine learning methods. For instance, artificial neural networks can be implemented on many hidden layers. This branch of neural network models is usually referred as Deep Neural Networks. With respect to the case of a shallow neural network (just one hidden layer), the number of parameters to be learned is significantly higher. Although such models have proved to be effective and expressively powerful [1], there is a certain freedom in the configuration of the model.

In this report we discuss the tuning of a simple Deep Neural Network (DNN) model. This is indeed an hyperparameter optimization problem: we wish to select a model capable of achieving good results (classification error) whilst reducing the overall complexity of the network (layers, number of neurons), taking into account various optimizer and activation functions. Moreover, the training data also accounts for variations in the performance of the model. All these optimization aspects are crucial for the effective applications of DNNs in real-world data.

METHODS

To probe the performance of the DNN model in various scenarios we start from a default model [2]. We choose a feed-forward fully-connected neural network, with three hidden layers, one of which is a dropout layer. The model is built in Keras, on the full specifications listed in Table I. The performance of the model is evaluated on the binary classifier accuracy

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{number of samples}}$$

and the loss function is the commonly used `binary_crossentropy`.

This default configuration is exploited all through the next section of this report. In section II we will discuss the impact of optimizer and activation functions, as well as the layer configuration. The section III accounts for a brief analysis about the role of data rescaling and weight initialization methods. Ultimately, in section IV we will use an optimal model, derived using the information presented in the previous sections, to test the accuracy on more complex data patterns.

layer	type	outputs	parameters	activation
input	Dense	2	6	ReLU
dense_1	Dense	20	60	ReLU
dense_2	Dense	20	420	ReLU
dropout	Dropout	20	0	-
output	Dense	1	21	sigmoid
dropout rate			0.2	
optimizer			adam	

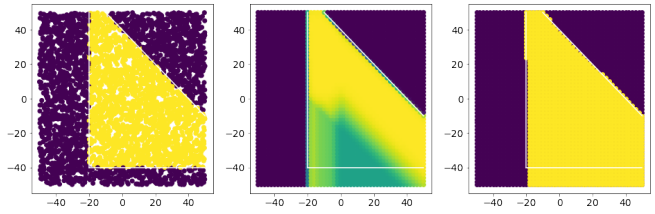
TABLE I: Layout of the default model for our DNN. This configuration corresponds to a total of 507 trainable parameters.

A. The reference training dataset

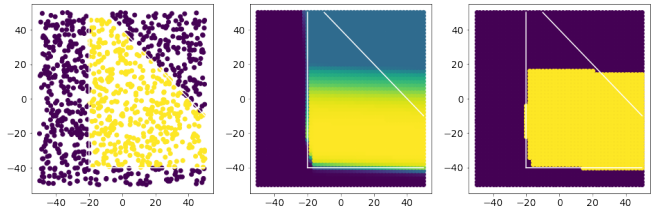
The datasets we exploit consist in several thousands of 2-dimensional points $\vec{x}$, labelled according to a pattern function $y = f(\vec{x}) \in \{0, 1\}$. At first, we create reference dataset generating 4000 random points in the domain $[-50, 50]^2$, applying a pattern function that visually resembles a trimmed *triangle*, as shown in Figures 1, on the left. We choose to use 80% of the dataset as the training set. For instance, the 4000 points of the reference dataset will be splitted in 3200 samples for training, and 800 for validation.

I. IMPACT OF THE DATASET SIZE ON THE DNN PERFORMANCE

As a first attempt, one can train the reference model of Table I with the reference dataset of 4000 (3200) samples. The resulting DNN is typically accurate, correctly classifying more than 90% of the samples. However, if we look at the prediction capabilities of such model (Figure 1a, central and right pictures), we notice that it naively fails on the lower region of the domain, in which the 'triangle' boundary is closer to the domain limit. A possible



(a) Training using the reference dataset (3200 + 800 points).



(b) Training using the reduced dataset (800 + 200 points).

FIG. 1: Prediction grids of the default DNN trained, respectively, on the standard and reduced datasets in 400 epochs.

interpretation of the issue is that the undermost region of the figure gains more importance in the loss function only when all the other regions (as they are more extended, thus they contain more points) are correctly classified, especially when we train for a restricted number of epochs (400).

To characterize the model accuracy dependence on the train dataset size, let us show how it changes if we use a reduced dataset, an increased dataset, or an augmented dataset. The difference between increased and augmented dataset is that the *increased* dataset is prior generated with a larger number of samples (20k); the *augmented* dataset is made of a large number of samples which are computed starting from a smaller dataset, without knowing the labelling function.

A. The reduced dataset

At first, the DNN is trained on a reduced dataset, taking a sub-sample of 1000 points from the reference dataset. Since the dataset has to be split for training and validation, the training set ultimately counts only 800 points. This obviously impacts the training of the DNN: the small amount of data makes the training very unstable and the results are discrete (see Figure 1b), leading to (validation) accuracy typically $< 90\%$. The validation loss is on average greater than the train loss, which prompts a situation of overfitting.

B. The increased dataset

On the other hand, we can use an increased dataset of $20k$ samples. Considering the 80% splitting, the training set counts $16k$ points, exactly 5 times the reference training dataset size.

In this scenario the model performs better (Figure 2) and the final result is more stable. This result suggests that the size of the dataset is relevant for achieving a good accuracy, at least within the current DNN layout and data pattern. We achieve an accuracy in the order of $\sim 99\%$, while the validation and train loss are comparable (therefore, there are no overfitting issues).

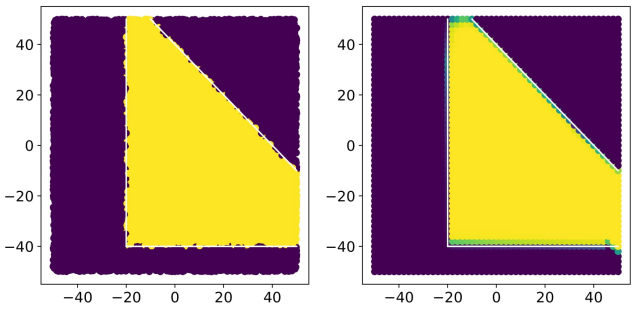


FIG. 2: On the left, the increased dataset (16k + 4k points). On the right, the prediction grid of the default DNN trained for 400 epochs.

C. The augmented dataset

The advantages of training with more data samples can also be obtained by applying on the reference dataset a data augmentation technique, which we have implemented from scratch to work on n-dimensional datasets.

Originally, we performed the data augmentation considering each training sample $(\vec{x}, y) = (x_1, x_2, y)$, and generating a fixed number of multiple artificial samples $(\vec{x'}, y)_i = (x_1 + s_1, x_2 + s_2, y)_i$, where s_1 and s_2 are random displacements drawn from a uniform distribution in $[-\sigma, +\sigma]$. However, we observed that this procedure would have ignored the spatial distribution of the samples' labels. Indeed, there would have been no difference in the displacements for a training sample at the boundary of the triangular region with respect to a sample located far from any boundaries. This would have caused the boundaries to be too shattered. Otherwise, reducing the amplitude of the uniform distribution σ would have caused the presence of clusters.

To fix this issue and take into account that our dataset presents boundaries between regions of points, we choose the amplitude of the displacement uniform distribution to vary depending on how close is the point to differently labeled points.

For 2D datasets, the procedure is the following:

- at each iteration, a random sample C_j is drawn from the training dataset
- we select all the samples $\vec{x}_i$ enclosed in a rectangle centered on C_j (each side is chosen to be $1/20^{th}$ of

the corresponding domain range)
- we compute the relative frequency f_y of the most common label among the points $\vec{x}_i$ and we set the amplitude σ of the displacement uniform distribution to be $\sigma = 3 \cdot (f_y)^3$
- each point $\vec{x}_i$ is used to compute an identically labeled artificial sample $\vec{x'}_i = (x_1 + s_1, x_2 + s_2)_i$

This procedure is repeated according to a parameter, which we call *augmentation factor* E. Eventually, the number of samples is increased approximately to E times the size of the initial dataset.

Augmentation factor	Train accuracy	Validation accuracy
(no augmentation)	0.9278	0.9253
2	0.9857	0.9306
3	0.9826	0.9869
10	0.9749	0.9912
20	0.9740	0.9937
30	0.9768	0.9962

TABLE II: Accuracy of the DNN when different augmented datasets are prompted.

We observe that a greater set size, even if artificially augmented, leads to a saturation of the model accuracy (Table II). This is reasonable, as it is not possible to indefinitely draw more and more constructive information from the same sample of training data. Furthermore, the additional samples act as stabilizers when we check the accuracy over independent training iterations.

II. GRID-SEARCH ON DNN PARAMETERS

a. ***Grid-search on optimizer, activation functions and dropout probabilities -*** We have picked several candidates for these three parameters in order to test their combinations using a huge initial grid-search. We tested 200 epochs for each combination, using a 4-fold cross validation.

From this grid-search we got many combinations which achieved an almost perfect accuracy ($\sim 99\%$), while there exists some combinations which lead to very bad accuracy ($\sim 50\%$, i.e. a random guess). In the poll of the best classifiers (see the output in the attached Jupyter notebook), we see that *Nadam*, *Adam* and *RMSprop* optimizers, combined with *softplus*, *softsign* and *tanh* activation functions typically lead to better results. Also, lower dropout values are very common among the best classifiers. From now on we restrict the DNN model to the previously listed parameters, as well as the small dropout values from 0.0 (no dropout) up to 0.3.

b. ***Grid-search on layer configuration -*** We want to probe the impact on the accuracy due to the number of neurons in the layers, as well as to the number of layers itself. Therefore, in this second grid-search we will test DNNs with $q = [1, 2, 3, 4, 5, 6]$ layers in grid-combination with $s = [10, 20, 30, 40, 50]$ neurons per layer. The general rule is that each DNN will have a certain amount q of layers, all of same size s. Explicitly, a first DNN might have two layers of 30 neurons each; another one might have three layers of 50 neurons each. All the possible DNNs are also tested by varying the hyper-parameters selected in the first grid search.

Instead of looking at the spurious results of this grid-search, it is interesting to notice how the accuracy declines when we reduce the number of layers. Indeed, it is trivial that deeper DNN will more likely achieve better results. Figure 3 shows that 1-layer DNNs are more likely to achieve an accuracy $\sim 90\%$, while 2-layer DNNs do slightly better $\sim 95\%$. The DNNs with 3 and more layers are more likely to saturate the accuracy $\sim 99\%$.

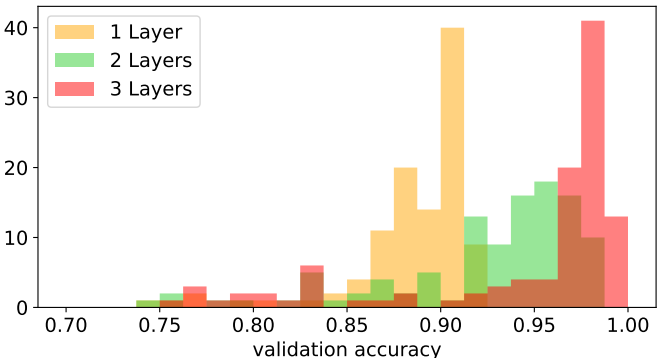


FIG. 3: The accuracy distribution based on the layer configuration grid-search.

III. IMPACT OF DATA RESCALING AND WEIGHT INITIALIZATION

a. ***Data rescaling -*** We wish to analyze whether the rescaling of the features impacts on the final DNN accuracy. As usual, we selected a poll of rescaling functions [3] and performed multiple training iterations to pick the average validation accuracy for each configuration.

From Table III we can observe that the best results are obtained using standardization with stabilization factor $\epsilon = 100$ or the *min-max* normalization in the range $[0, 1]$. These two methods lead to an additional 7% accuracy comparing to the ones obtained without feature scaling.

b. ***Weights initialization -*** The same method of inquiry can be applied to several weights initialization. As a result (see attached Jupyter notebook), we see that an accuracy $> 98\%$ is achieved by the Keras built-in modules `random_normal`, `truncated_normal`, `glorot_normal`, `orthogonal` and `variance_scaling`. We also notice that there exists trivial initialize modules (`ones`, `zeros`, `constant`) which ultimately lead to bad DNN models (accuracy $\sim 55\%$).

Method	Train acc.	Validation acc.
no rescaling	0.9231	0.9287
domain rescaling (x/50)	0.9337	0.9413
standardization	0.9262	0.9262
std. with stabilizer $\epsilon = 0.01$	0.9219	0.9237
std. with stabilizer $\epsilon = 0.1$	0.9241	0.9187
std. with stabilizer $\epsilon = 1$	0.9578	0.9588
std. with stabilizer $\epsilon = 10$	0.9825	0.9712
std. with stabilizer $\epsilon = 100$	0.9919	0.9950
std. with stabilizer $\epsilon = 1000$	0.9266	0.9250
min-max in $[0, 1]$	0.9909	0.9962
min-max in $[-1, 1]$	0.9266	0.9275

TABLE III: Average accuracy of the DNN when the data is rescaled with different methods.

IV. TRAINING ON MORE COMPLEX DATA PATTERNS

The observations presented in the previous sections can be used to select an 'overall better' model.

layer	type	outputs	parameters	activation
input	Dense	2	6	softplus
dense_1	Dense	20	60	softplus
dense_2	Dense	20	420	softplus
output	Dense	1	21	sigmoid
dropout			no	
initialization			glorot normal	
optimizer			Nadam	

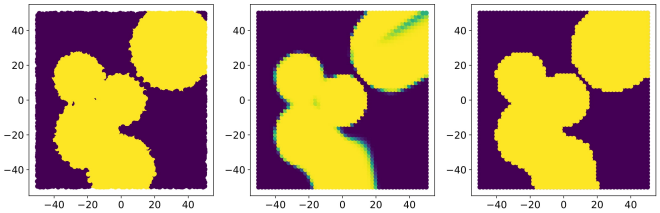
TABLE IV: Layout of the best DNN we have tweaked using the results of section II and III.

Finally, we have tested the application of such model to learn more complex patterns in 2D data. We have selected a set of more complex labeling functions in order to find some edge cases. The full list of functions is shown in the attached Jupyter notebook, whereas the Figure 4 shows some of the most interesting results.

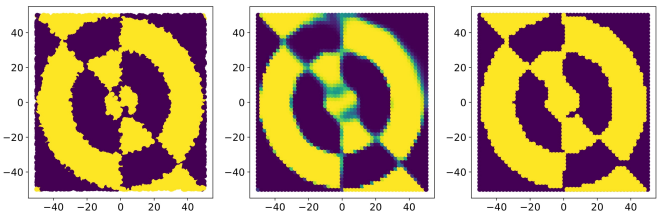
V. CONCLUSIONS

The tweaked model of Table IV proves to be effective when trained on more complex data patterns, but only if it is trained for a sufficient number of epochs (~ 1200). On separate trials, we also observe that the convergence speed of the DNN crucially improves if we add an additional layer of 20 neurons, for a total of 3 hidden layers. In this scenario, the number of epochs required to achieve a target accuracy of 95% is drastically reduced to ~ 300 epochs.

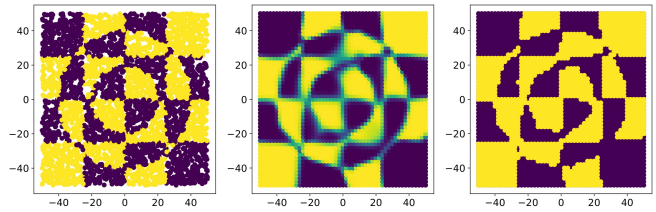
This result confirms what we expect from the theory on the *expressive power of neural networks* [1]: in order to express two or more convex polygons (2D) with $k-1$ faces, a minimum of 3 layers is required, with at least k neurons per layer. For this reason, if we use less than three layers, the training is more prone to optimization issues and will require more epochs to maximize the accuracy. On the other hand, DNN models with more than 3 layers do not show a significant accuracy boost. Moreover, reducing the number of neurons would reduce the model ability of learning curved shapes.



(a) Multi-sphere.



(b) Alternative function suggested by Prof. Baiesi.



(c) Mixed chessboard. In particular, this model has been trained using the data augmentation technique presented in section I C.

FIG. 4: Prediction grids of the best DNN configuration (IV) (2-layers) trained for > 800 epochs on various data patterns (of size $20k$). The first figures on the left show the original dataset, while the two other figures show the prediction grids of the DNN model. DNNs with 3 hidden layers are able to achieve the same results but in much less epochs.

[1] S. B.-D. Shai Shalev-Shwartz, *Understanding Machine Learning: From Theory To Algorithms* (CUP, 2014).

[2] pankajmehtabu, "Keras dnn notebooks," GitHub.

[3] Wikipedia, "Feature scaling," .