GROUP 2212

Selection of neural network hyper-parameters for a binary classification task

Eugenio Fella, Matteo Pedrazzi, Gaetano Ricucci, and Carlo Sgorlon Gaiatto
(Dated: March 19, 2022)

Modeling a neural network to solve a targeted task can be an extremely difficult challenge. Performance depends on both the architecture and the algorithm used for training. Properly setting up all the hyper-parameters that characterize these two features can not be done by following an instruction booklet. Every problem is different and often it is not possible to use the same recipes. In this work we present a method for modeling a feed forward neural network for binary classification. The key to this work is to exploit the a priori knowledge of the problem to fix some hyper-parameters and restrict the search for the others to a small and reasonable set of values, through the grid search technique.

INTRODUCTION

The main purpose of this work is to model and train an artificial feed-forward neural network for a simple binary classification task. The standard dataset consists of 4000 2-dimensional points $x = (x_1, x_2)$, each component extracted from a uniform distribution over the interval $[-50, 50]$. These points are labelled accordingly to the following function:

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

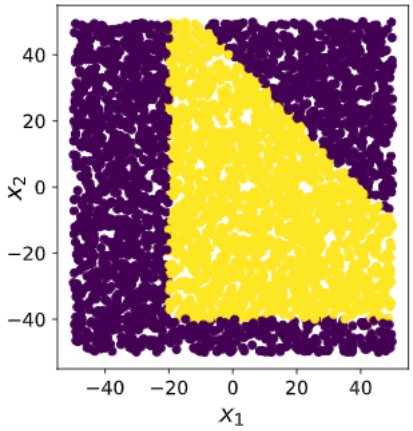

FIG. 1. The standard dataset made of 4000 random distributed points labeled according to Function 1. The yellow and purple color correspond to labels 1 and 0 respectively.

We aim to define a reproducible pipeline for selecting network hyper-parameters using the Keras interface [1]. The naive approach would be to use a grid search over all the hyper-parameters and zooming in the correct region with a finer grid at each iteration. However, this strategy can become very time consuming due to both the high number of possible combinations in the hyper-parameter landscape and the amount of data required to learn the problem. Therefore, our approach is to exploit prior information about the task to fix as many hyper-parameters as possible and perform a standard grid search on some values of the remaining ones. Finally, we provide some details on how Keras effectively implements modeling and training of neural networks, hence making this work a useful overview for anyone who will need to take steps in this field.

METHODS

The design of a neural network requires some non-deterministic routines, for example the initialization of the network weights. In order to make our procedure reproducible we fix the seeds of the random number generators of all libraries in our code.
The architecture of the neural network and the training algorithm are implemented by the `Sequential` class of Keras. It provides three important methods for managing the hyper-parameters. The `add` method is used to select the number of layers and for each of them the number of neurons, the nonlinear activation function and the initialization of the weights. It also permits the introduction of regularization through the dropout technique. The `compile` method configures the learning process by specifying the loss function, the optimizer, and the metrics to be tracked during the process. The `fit` method trains the model by allowing the choice of batch size and number of epochs. We briefly describe the hyper-parameters and for each of them we fix its value if we have sufficient prior information, otherwise we identify a possible set of values.

Hyper-parameters of the Architecture

- **Hidden layers** The depth of the neural network is related to the complexity of the task; in our case, the figure drawn by the labeling function (1) can be considered as a conjunction of three half-spaces (Figure 1) suggesting that even a shallow neural network can correctly learn the problem [2]. Therefore, we try a neural network made of one, two or three hidden layers.

- **Number of neurons** Regarding the number of neurons in the hidden layer of the network, there is no straightforward way to constrain its value from the given problem. In particular, we look for identical hidden layers of 4, 8, 16 neurons each. With equal performance between two different architectures, we choose the one with the least number of neurons and hidden layers to prevent overfitting and save computation time due to the reduced number of edges.

- **Activation function** In a simple binary classification problem the output layer consists of a single neuron. Since our label domain is $\{0, 1\}$ the sigmoid is a suitable choice for the activation function of the last neuron. Considering the activation function of hidden layers, we try sigmoid, ReLU, and ELU. In particular, the last two are commonly used to avoid the vanishing gradient problem.

- **Weight initialization** The performance of the network in the training process is strongly influenced by the initialization of the weights. Setting the weights all to the same value, for example zero or one, does not work because the symmetry of the neurons in the same layer is not broken. Viable alternatives are the Glorot and He initialization methods, which help prevent vanishing and exploding gradients. In particular, Glorot initialization is optimal for sigmoidal activations while He is commonly used for ReLU and its variants [3],[4]. Thus, for the weights in the output layer we choose the uniform Glorot method while for those in the hidden layers we try both the uniform Glorot and normal He.

- **Dropout** To prevent the output from depending too much on a single neuron, at each step it is possible to train the network by randomly selecting only a subset of the neurons. This regularization technique is called dropout [5]. In our case the tested rates, namely the percentage of neurons dropped out in each hidden layer at each iteration, are 0 and 0.25. We can not test too high rate values due to the small size of our network.

Hyper-parameters of the Training Algorithm

- **Loss Function** The label $y \in \{0, 1\}$ of our binary classification task and the prediction $\hat{y}$ of the sigmoid function of the last neuron suggest cross-entropy (2) as an appropriate choice for the loss function.

$$loss = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y}) \tag{2}$$

- **Optimizer** The optimizers are classes of methods used to minimize the loss by iteratively adjusting the weights and biases of the neural network. The simplest is the Stochastic Gradient Descent (SGD) that updates the weights making steps in the opposite direction to the gradient of the loss, with amplitude controlled by the learning rate parameter η. It is possible to add the momentum parameter γ which can be interpreted as the memory of the direction in which we are moving in the weight space. In addition, we also try Nesterov's approach whose idea is to evaluate the gradient at the expected value of the weights taking into account the present momentum. There are several generalizations of the gradient descent method. The RMSprop algorithm updates the weights by considering the second moment of the gradient, while the ADAM optimizer also includes the first moment of the gradient [6]. Since we are not able to choose in advance which is the best optimizer, we try all the ones we listed above. We choose the learning rate by searching on three logarithmic points, $\eta = [0.1, 0.01, 0.001]$, while maintaining the default value provided by Keras for all the other hyper-parameters in order to limit the computation time of the grid search. Regarding the SGD optimizer, we test three values for the momentum, $\gamma = [0, 0.8, 0.9]$, combining them with and without the Nesterov method. Note that for equal performance, we select the computationally less demanding optimizer.

- **Batch Size** The mini-batch technique consists of dividing the dataset into a number of subsets called mini-batches, whose size is controlled by the batch size hyper-parameter. In Keras the idea is to update the weights by averaging the individual updates evaluated on each sample of the batch starting from the same weights resulting from the previous step. We try two values for the batch size, 32 and 128. The smaller the value of the batch size, the more frequent the updates of the weights and the greater the amount of stochasticity introduced in the training process, which acts as a regularization factor. On the other hand, a larger value for the batch size can increase the number of operations per second.

- **Epochs** The iteration over the entire training set is called an epoch. We consider two values, 64 and 16. The former is used to check the performance of the models after a sufficient number of iterations. The latter is used to give more emphasis to models that produce good results even with fewer epochs.

The dataset is divided into training, validation, and test sets, which represent 0.5, 0.1, and 0.4 of the labelled samples, respectively. The data in each set are preprocessed by removing the mean and rescaling to unit variance.
The grid search over the hyper-parameters previosuly described is performerd using the `GridSearchCV` class from the Scikit-Learn library [7]. This class requires three fundamental arguments: the estimator, representing the neural network to be trained, which we provide using the `KerasClassifier` method that wraps the Keras Sequential model with the Scikit-Learn interface; a dictionary, containing the names and values of the hyper-parameters; and the number of folds, used by the K-Fold Cross-Validation procedure, which we set to 3. It is also possible to make the local machine use all available processors by setting the argument `n_jobs` to -1.
After selecting the best hyper-parameters, we train the final model using the entire training set and we increase the maximum number of epochs to 128. As an additional stopping criteria, we use the Keras `EarlyStopping` class setting the validation accuracy as the monitored metric. In particular we set to 16 the patience, namely the number of epochs in which the validation accuracy can get worse before the training stops [8].

In order to evaluate the impact of the number of data points on the performance of our neural network, we train the final model again by considering three different datasets: a reduced dataset, with 1000 samples; an increased dataset, with 8000 samples; and an augmented one, obtained from the reduced training set by generating as many artificial samples as to match the size of the standard training set and keeping the same original labels. More precisely, the augmentation process is defined as follows:

$$x = (x_1, x_2) \to (x_1 + dx_1, x_2 + dx_2) \tag{3}$$

where each (dx_1, dx_2) is a small random shift extracted from a uniform distribution over an interval of $1/100$ th of the original box size. Note that both the reduced and the increased sets are divided into training, validation, and test sets with the same percentage used with the standard dataset.

RESULTS

The grid search on the hyper-parameters leads to several optimal results, namely there are more than one combination with a score - the model mean cross-validated accuracy - greater than 0.99. Among these we select as final model the one whose hyper-parameters are shown in Table I. The choice is made considering several factors. The learning rate set to 0.01 coupled with the Adam optimizer ensure a higher stability of the algorithm, as can be inferred from a lower oscillation of the training and validation accuracy. The selection of the other parameters is driven by considerations about the computational complexity of the model. The ReLU activation function provides a fast evaluation of the gradient in the Back-propagation algorithm. Moreover, the neural network architecture we choose is the one with the minimal number of edges, shown in Figure 2. In this way we manage to find a good compromise between the computational time and the performances achieved by the model.

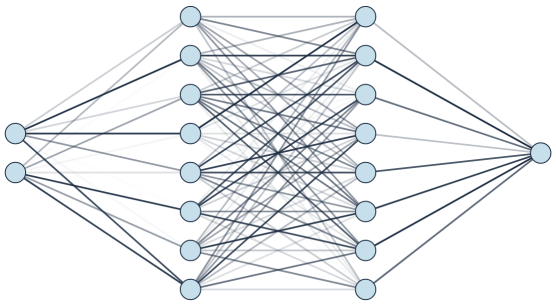

FIG. 2. The architecture of the final neural network consists of 2 input neurons, 2 hidden layers of 8 neurons each, and a single output. The biases are not shown. The illustration is created using the open source software NNSVG [10].

The learning curves of both accuracy and loss over epochs obtained from training the final model over the entire training set are shown in Figure 3. The early stopping criteria stops the training procedure after 35 epochs.

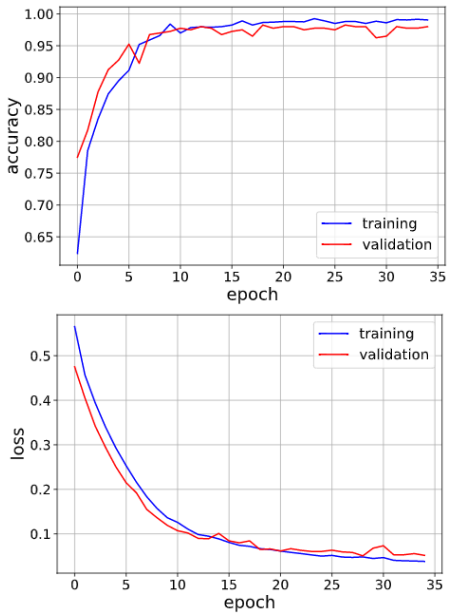

FIG. 3. Learning curves of accuracy (a) and loss (b) on training and validation set. Training stops after 35 epochs, achieving an accuracy of 0.99 and 0.98 in the training and validation sets, respectively.

score	units	activation	initialization	dropout	optimizer	learning rate	batch size	epochs
0.99 (0.01)	[8, 8]	ReLU	He normal	0	Adam	0.01	128	64

TABLE I. Score, standard deviation and all the hyper-parameters of the selected algorithm.

The accuracy of the trained model on the test set is 0.99. In Figure 4 we show the normalized confusion matrix of the true and predicted labels while in Figure 5 it is possible to visualize the predictions of the final model on the test set.

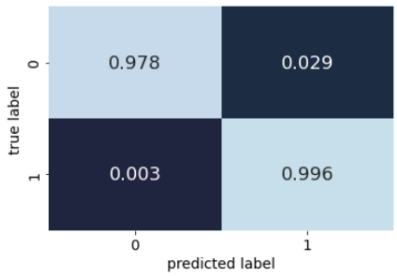

FIG. 4. Confusion matrix with normalized entries: dark boxes show that the model achieves impressive precision on test set.

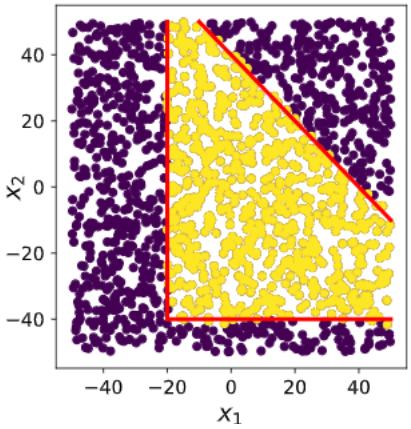

FIG. 5. Model predictions on the test set. The red boundaries outline the geometric figure representing the binary calssification task.

Table II presents the test accuracies obtained on the reduced, augmented, and increased dataset. As can be seen, the accuracy on the reduced dataset is worse than the one obtained on the standard dataset while the accuracy on the increased one is even better, highlighting the importance of the amount of data needed to train a neural network. On the other hand, the performance on both the standard and augmented datasets is similar. This means that the augmentation process is an effective method to use when there is no plenty of data.

dataset	test accuracy
reduced	0.95
augmented	0.97
increased	0.99

TABLE II. The test accuracy of the final model obtained with different datasets.

CONCLUSIONS

We have proposed a pipeline to set the hyper-parameters of a feed forward neural network for a binary classification task, exploiting our prior information about the problem itself. Our approach was to limit the number of hyper-parameter values to be explored through the grid search technique by inspecting the properties of each of them. This procedure was applied to our case study obtaining a model capable of learning the problem with high accuracy. We also analyzed the impact of dataset size on model performance and found that performance increases as the number of samples increases, while in the case when there is no plenty of data the augmentation procedure is an effective strategy.

[1] F. Chollet et al., Keras, https://keras.io (2015).
[2] S. Shalev-Shwartz and S. Ben-David, Understanding machine learning: from theory to algorithms, Cambridge University Press (2014).
[3] X. Glorot and Y. Bengio, Journal of Machine Learning Research - Proceedings Track, **9**, 249–256 (2010).
[4] K. He and X. Zhang and S. Ren and J. Sun, IEEE International Conference on Computer Vision (ICCV 2015), **1502**, 1026–1034 (2015).
[5] N. Srivastava and G. Hinton and A. Krizhevsky and I. Sutskever and R. Salakhutdinov, Journal of Machine Learning Research, **15**(56), 1929–1958 (2014).
[6] P. Metha et al., Physics Reports, **810**, 1–124 (2019).
[7] F. Pedregosa et al., Journal of Machine Learning Research, **12**, 2825–2830 (2011).
[8] L. Prechelt, Neural Networks: Tricks of the trade, 55–69 (2012).
[9] Y. Bengio, Neural Networks: Tricks of the Trade, 437–478 (2012).
[10] A. LeNail, Journal of Open Source Software, **4**(33), 747 (2019).