Group 2210 – A study on Deep Neural Networks using Keras

George Prodan, Lorenzo Ausilio, Elaheh Ahmadieslamloo, and Farshad Jafarpour
(Dated: March 19, 2022)

Deep learning tools are very useful when working with huge amounts of data. The choice of the model that is implemented for particular tasks depends on the nature of data. In this work we generate artificial datasets through different methods in order to explore the performances of a simple deep neural network (DNN) with two hidden layers. We observe various behaviours of the DNN when working with datasets generated in different ways. Also, we apply grid searching to find better parameters which increase the DNN performance.

Contributions

- **All members of the group contributed equally**

INTRODUCTION

Deep learning (DL) field is one of the most popular research directions nowadays. Many DL applications are developed in order to cope with high amounts of data being more effective than the usual machine learning tools in any domain (e.g. cybersecurity, nuclear physics, astrophysics, bioinformatics and so on) [1]. Among many others, some of the most popular deep neural networks (DNNs) are the convolutional neural network (CNN), deep feed-forward neural networks, recurrent neural networks (RNN), autoencoders, or long short-term memory networks (LSTM) [2]. One can distinguish three main DL approaches: unsupervised (mostly generative networks [3]), semi-superivsed (e.g. RNNs [4], GANs, autoencoders [5]) and supervised (e.g. RNNs, CNNs [6], DNNs).
In this study we focus on the use of simple DNNs for binary classification of two-dimensional data [7]. We randomly generate data and establish a classification of it based on a non-linear function. In the following sections we describe the methods used in creating the datasets, the data augmentation and the DNN implementation. The results section shows how the DNN performance varies when changing several parameters. Also, we explore by grid searching different configurations of the DNN. In the last section, we provide an overall description of the results.

METHODS

The first step is to define the functions and classes, for that we use the NumPy library [8] of Python [9] to generate data and Keras library [10] to implement our DL model. Finally for plotting our results we use matplotlib [11]. The procedure is described by the following steps:

1. **Creating the dataset.** We generate data by using NumPy library and nonlinear functions to have random numbers as a dataset. Ee create a dataset containing 4000 samples. Each sample contains a pair of points. For those samples we use a nonlinear function according to which the labels either get a value of 0, or 1. In the next step we divided our data in two parts, training and validation set. Data normalization is done after

2. **Defining Keras Model**. Models in Keras are defined as a sequence of layers. We create a Sequential model and add layers. We use a fully-connected network structure with four layers. Fully connected layers are defined using the Dense class. We specify the number of neurons or nodes in the layer as the first argument and specify the activation function using the activation argument. We use 20 neurons for each hidden layer and also use *relu* as an activation function in preliminary experiments. However, the final function for output layer is the sigmoid function.

3. **Compiling the Keras model.** After defining the DNN, we should compile our model. We must specify the loss function to use to evaluate a set of weights, the optimizer is used to search through different weights for the network and any optional metrics we would like to collect and report during training. In this case, we use cross entropy as the loss argument. This loss is for a binary classification problems and is defined in Keras as *binary_crossentropy*.

4. **Fitting and Evaluating the Keras Model.** After compiling we should apply our DNN on some data, so we use it for fitting on our data by using *fit()* function and we calculate different parameters which are : training loss, training accuracy, validation loss and validation accuracy.

For generating data we use three non-linear functions f. The first function classify the points based their position with respect to a triangle, namely

$$y = f_1(x) = \begin{cases} 1, \text{if } x_0 > -20, x_1 > -40, x_0 + x_1 < 40 \\ 0, \text{otherwise} \end{cases}$$

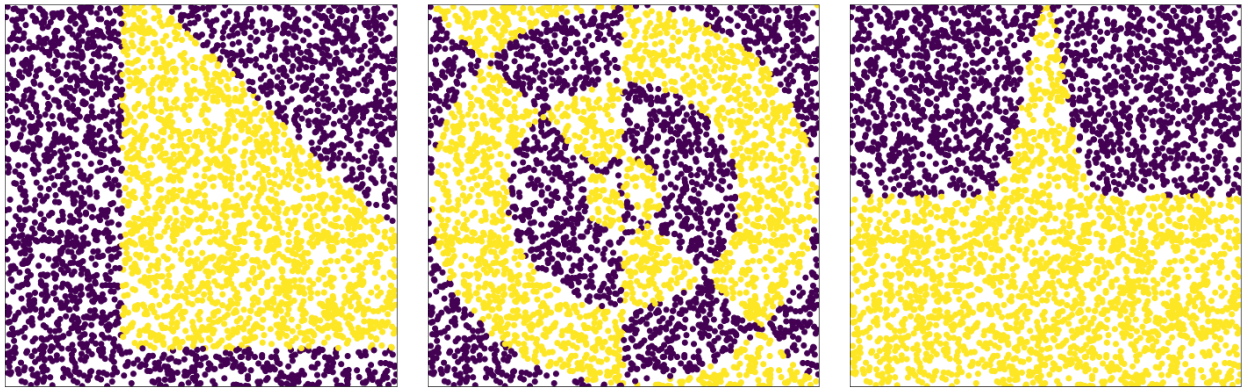


FIG. 1. Data distribution for each non-linear function we use in generating the labels (purple - label 0, yellow - label 1). From left to right: f_1, f_2 and f_3.

The second function splits the data in several clusters using a formula based on sinusoidal functions

$$y = f_2(x) = \begin{cases} 1, \text{if } sgn(\sum x_i) \cdot sgn(x_0) \cdot \cos(\frac{\|x\|}{2\pi})) > 0 \\ 0, \text{otherwise} \end{cases}$$

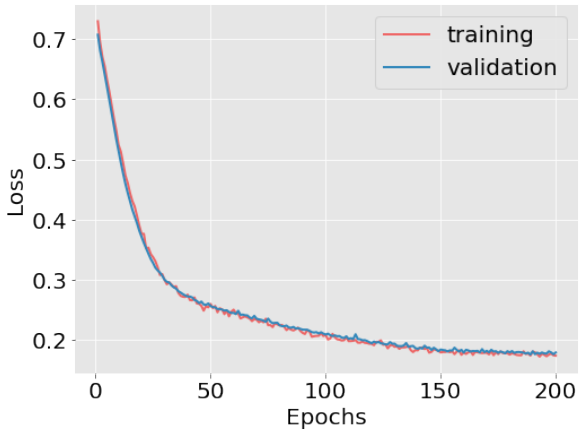
The last function is based on a gaussian function,

$$y = f_3(x) = \begin{cases} 1, \text{if } x_1 < 50e^{-\frac{x_0^2}{50}} \\ 0, \text{otherwise} \end{cases}$$

In Figure 3 we show how data is labeled based on those three functions. We study the behaviour of DL model for each one of them. The results are discussed in the next section.

We implement a sequential DNN with two hidden layers of 20 neurons each and the same activation function. However, for the output layer we consider an activation function which can be different. Also, we add a Dropout layer.

FIG. 2. The training and validation losses with respect to the number of epochs elapsed ($N = 2000$, label function f_1).



Furthermore we also analyse the performance of our model when we did some data augmentation where we used some Gaussian noise. In fact it has been shown that adding a small amount of random noise to our data can aid generalization and fault tolerance of our model [12] and training with noise is equivalent to Tikhonov regularization [13]. Since our data samples are two dimensional, we add small shifts s1 and s2 which are drawn from a Gaussian distribution with mean zero and a fixed standard deviation. We use the first activation function in this case.

We focus on two scenarios : firstly we augment the data by 10% and secondly by 20%. In both scenarios we also changed the standard deviation of our Gaussian noise.

RESULTS

Number of generated samples

We want to study the performance of the DNN when the number of samples is reduced or increased. An example of loss and accuracy variation for $N = 2000$ and $f = f_1$ is provided in Figure 2. We change the number of samples and monitor the two parameters, minimum loss and maximum accuracy over the validation set. It is obvious from the Figure 3 that validation accuracy goes down and validation loss becomes worse by reducing the number of samples. This makes sense since by reducing the sample size, our DNN fails to learn the necessary parameters well and struggles to make accurate predictions for new data. Moreover, we observe that for the third function the algorithm learns by using less samples and we notice a decrease in accuracy when we overfeed the DNN.

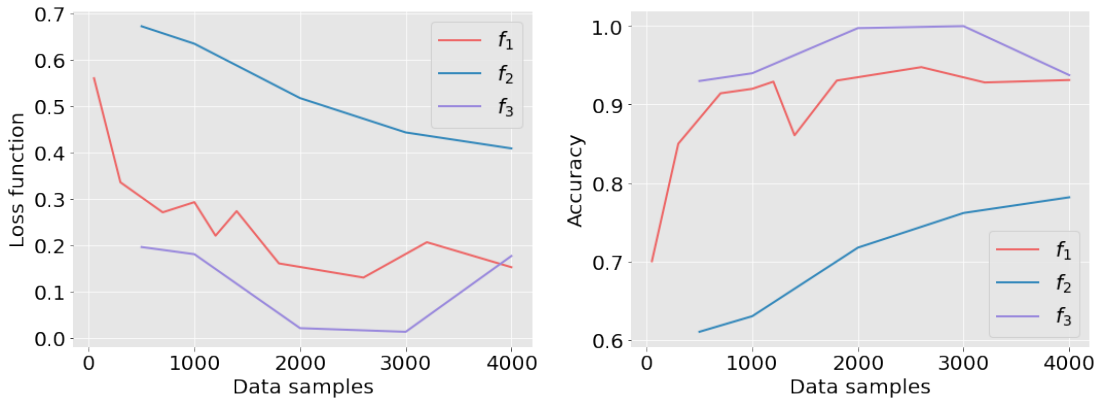


FIG. 3. Plots of loss function (a) and accuracy (b) with respect to the number of samples for each function used in generating the labels.

TABLE I. Minimum loss and maximum accuracy during training for augmented datasets. The noise is described by the standard deviation parameter σ.

% of aug. data	10 %			20 %		
σ	1	2	3	1	2	3
loss	0.055	0.087	0.040	0.090	0.127	0.062
acc (%)	98.75	97.12	99.12	97.62	94.62	98.37

Data augmentation

In this part we study the performance of our Neural Network model when augmenting the training data. In all cases we fix the seed so as to always reproduce the same results and also so that we could more fairly compare the model in all cases. In each case the number of epochs, batch size and all other parameters of our network stayed the same.

When analysing both the 10% case and the 20% case (I) we can see that a pattern starts to emerge. Both the smallest validation loss and the highest accuracy on the validation set is achieved when the standard deviation is 3. Then a standard deviation of 1 yields the second-best results and the worst results are achieved for a standard deviation of 2. As such we see that adding more noise doesn't always equate to a model that can generalize better.

By comparing the 10% case with the 20% one, we can also see that in the 10% case our model learns better each time, irregardless of the amount of noise added. This shows that increasing the amount of training data by adding some noisy data to it doesn't always mean we will get better and better results; there is a limit to how much noisy data you can add that will benefit the model. The more such data that you feed to the model, the more difficult it will be to tune the necessary parameters so that it can generalize well on novel data.

TABLE II. Grid search results for the three function used in binary classification. The parameters shown correspond the the best accuracy obtained in the grid searching process, acc (%). Weigths are initialized to zeros.

f	optimizer	activation function	activation function 2	layers	drop-out rate	acc (%)
1	SGD	sigmoid	sigmoid	5	0.3	52.31
2	Adam	relu	sigmoid	15	0.3	50.62
3	SGD	sigmoid	sigmoid	5	0.3	57.75

TABLE III. DNN performance when using different weight initializers. Loss and accuracy are provided.

weight initializer	Zeros	Ones	Random Uniform	**Random Normal**	Truncated Normal	Orthogonal
loss	0.6914	0.6913	0.2921	**0.1405**	0.2534	0.1751
acc %	52.99	55.25	91.00	**94.74**	91.25	91.50

Grid search

To find the parameters of the most efficient DNN we apply grid search [14]. We take into account the optimizer (SGD, RMSprop, Adagrad, Adam, or Nadam), the activation functions (sigmoid, tanh, relu or elu), the number of layers for the hidden layers (5, 10, 15, 20 or 25), and the drop-out rate (between 0.1 and 0.5 with the step of 0.1). We make use of the grid search tool already implemented in scikit-learn package [15].

We perform grid searching for different datasets given by the three functions used in binary classification. The results, when weights are initialized to zeros, are shown in II. We observe that the SGD optimizer and a smaller

number of layers lead to the best accuracy when data is separated only in two clusters. The best drop-out rate seems to be around 0.3 in any case possible.

Weight initialization

We train the DNN using several weight initializers and measure the minimum loss function and maximum accuracy. The results are presented in Table III. We observe that initializing the weigths randomly by a normal distribution leads to the best results.

CONCLUSIONS

We observe that when we increase the amount of data, our model learns better and can generalize better on new, unseen data. The opposite happens when we reduce the size of our data set. By doing data augmentation the model can learn better, but one has to carefully choose the amount of noisy data to add, and also the amount of noise, because it could lead to a decrease in performance.

Regarding the architecture of DNN, the best choice of the optimizer, number of neurons, activation functions, and drop-out rate depends on the nature of data we process. We observe that particular labeling functions which split the data in two clusters require the same optimizer (SGD) and similar activation functions. Another important aspect is the weight initialization. In the most of the experiments we initialized the weights randomly, whereas for grid search was made by intializing the weights to zero in order to avoid fluctuations of the results.

[1] Z. J. H. A. e. a. Alzubaidi, L., J Big Data **8, 53** (2021), 10.1186/s40537-021-00444-8.

[2] F. e. a. Emmert-Streib, Frontiers in Artificial Intelligence **3** (2020), 10.3389/frai.2020.00004.

[3] L. Ruthotto and E. Haber, arXiv e-prints , arXiv:2103.05180 (2021), arXiv:2103.05180 [cs.LG].

[4] A. M. Dai and Q. V. Le, arXiv e-prints , arXiv:1511.01432 (2015), arXiv:1511.01432 [cs.LG].

[5] Y. Ouali, C. Hudelot, and M. Tami, arXiv e-prints , arXiv:2006.05278 (2020), arXiv:2006.05278 [cs.LG].

[6] A. Ajit, K. Acharya, and A. Samanta, in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)* (2020) pp. 1–5.

[7] M. Baiesi, "First assignment of laboratory of computational physics, mod. b - dnn," (2022).

[8] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, Nature **585**, 357 (2020).

[9] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual* (CreateSpace, Scotts Valley, CA, 2009).

[10] F. Chollet *et al.*, "Keras," (2015).

[11] J. D. Hunter, Computing in Science & Engineering **9**, 90 (2007).

[12] R. Reed and R. J. MarksII, *Neural smithing: supervised learning in feedforward artificial neural networks* (Mit Press, 1999).

[13] C. M. Bishop, Neural Computation **7**, 108 (1995), https://direct.mit.edu/neco/article-pdf/7/1/108/812990/neco.1995.7.1.108.pdf.

[14] C.-w. Hsu, C.-c. Chang, and C.-J. Lin, (2003).

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).