Understanding the Effect of Diverse Data Preprocessing and Hyper-parameter Optimization Techniques over Deep Neural Networks

Fatıma Rabia Yapicioglu, Asal Rangraziasi, Giulia Bellentani, and Pu Yin

(Dated: March 19, 2022)

In recent years, deep neural networks have gained remarkable achievements in many research areas. The increasing success of deep neural networks depends on several aspects, such as the development of a wide variety of data preprocessing techniques, activation functions, and "learning rate optimization" algorithms designed specifically to train deep neural networks. In this assignment, we compare the different performances of simple DNN models when the data samples are reduced, increased, or augmented. Then, for optimization purposes, we exploit "Grid search", a tuning technique that tries to find the best combination of hyper-parameters: we search through all possible combinations of various activation functions, minimization algorithms, number of units in the inner layers, and dropout rates to improve the performance of DNN on our data. Finally, the value of validation accuracy for the DNN designed with our best combination of hyper-parameters is roughly (91.75%) over 250 epochs.

Introduction

Artificial Neural Networks are interconnected groups of nodes, inspired by the simplification of interconnected neurons inside a brain[1]. They represent a step further in the implementation of machines that reason in a way that resembles more the human way, finding patterns through the repeated exposition to the same kind of data, exploiting the power of induction. To make sense of the data provided to them, Artificial Neural Networks (ANN) employ many layers of mathematical computation: an ANN typically consists of hundreds to millions of artificial neurons (called units) stacked in layers. The input layer (the first one) gets data from the outside world in many formats: this is the information that the network intends to process and learn. The data is routed through one or more hidden units after leaving the input unit. The job of the hidden layer is to convert the input into something that the output unit can exploit. Generally, the decision mechanism of the output unit will actualize the requested prediction task, that could be, for example, classification or regression. An ANN that has numerous layers between the input and output layers is called deep neural network (DNN)[2].
When training a DNN the ultimate goal is to fine-tune the hyper-parameters[3] in order to generalize better the model that we have just found screening the given data, avoiding overfitting or capturing the noise. A good model is a model that can be successfully applied to similar data that it has never seen before. Training a DNN that can generalize well to new data is challenging. A model with too little capacity cannot learn the problem, whereas a model with too much capacity can end in being too dependent on the given training dataset (overfitting). Both cases result in a model that does not generalize well. Nevertheless, model complexity is not the only point that should be taken into consideration, but also the size and quality of the dataset fed to the model[4].

Thus, the purposes of this assignment are the following three:

- comparing the results obtained with different preprocessings of the training data (reducing, increasing, or augmenting the number of training samples)
- observing the effects on the DNN performance of hyper-parameter optimization techniques such as "grid search"
- analyzing metrics such as validation accuracy, loss, precision, recall, f1-score, and confusion matrix for the model that performed best among the others during "grid search"

Data augmentation is a method of artificially creating fresh training data from existing data[5]. This is accomplished by using domain-specific approaches to transform samples from a previously given training dataset into new and unique training samples[6]. It enables us to extend the dataset size and provide variety without actually gathering new data, so it is usually exploited when the labelling function used during the original data generation step is unknown to the current developers.
"Grid search" is essentially an optimization algorithm which tries to select the best combination of hyper-parameters from a provided list of options. It is worth mentioning that, during the analysis, the number of hyper-parameters rises exponentially. In fact, one of the limitations of grid search is that it suffers from dimensionality.

To summarize, applying the correct preprocessing to the provided data and effective hyper-parameter tuning will result in a more generalizable and predictive model.

Methods

A. Dataset Description

Regarding the original dataset, the initial 4000 two-dimensional random samples were drawn randomly from

a uniform distribution.
We exploited a two-dimensional nonlinear function to label each sample. The right "0/1" label for each 2D sample was chosen depending on its position on the coordinate system: "1" if it laid inside of the triangle defined by the aforementioned function, otherwise "0".

B. The base model

Initially, using "Keras" library, we built a base DNN model that has these characteristics: 2 nodes for the input layer, 20 nodes for both inner layers, and 1 node for the output layer. For the output layer the activation function is "sigmoid", anywhere else is "rectified linear". Immediately after the two inner layers, we also added a dropout layer with rate "0.2".
Starting from this base model, in this paper we will analyze the results obtained by various models trained with different amounts of data and different combinations of hyper-parameters, as explained in the Introduction section.

C. Data generation

We defined a method called "generate and split data" that contains all that is needed to generate a dataset with the aforementioned characteristics of the samples and the labels. It is flexible regarding the dataset size, receiving this number as an argument: if the default "4000 samples" size is changed, this clearly leads to an increment or a decrement of the dataset size. In addition to returning the whole generated dataset, this method also returns the dataset already split in training set (80%) and validation set (20%).
Therefore, if we managed to deal with incrementing and decrementing the dataset size already inside the dataset generation function, this was not possible for the data augmentation. It is a procedure that can be done only on samples from the training dataset and it needs a dedicated method.
The core of the "augment data" method consists in the small random shifting of both 2D coordinates of a sample: they are separately shifted by two values drawn from a uniform distribution. We chose the parameters of this distribution limiting that small values in a range that is centered in zero and equal to two times the maximum coordinate value in the training set divided by the current number of samples of the whole dataset.
In principle, when one uses "data augmentation" he is forced to simply paste the previous label of the unshifted sample to the shifted one, because this procedure is useful precisely when the once used labelling function is unknown to current developers. We indeed paired the augmented data with the labels in the just explained way, but, given the fact that actually we did know the labelling function, we also checked how many shifted values ended up having the wrong label after the augmentation. This had the aim to confirm that the chosen range for the uniform distribution was "small enough". Producing each time 1000 augmented samples, the mean over 3000 augmentation procedures is of 0.75 wrongly labelled samples. We considered this acceptable.
Then we added the 1000 augmented samples to the original 3200 training samples, creating a "4200 samples" training set. Another reason to choose only 1000 augmented samples was to keep a reasonable ratio between training set size and validation set size, avoiding to end up with a validation set size too small in relation to the training set size.

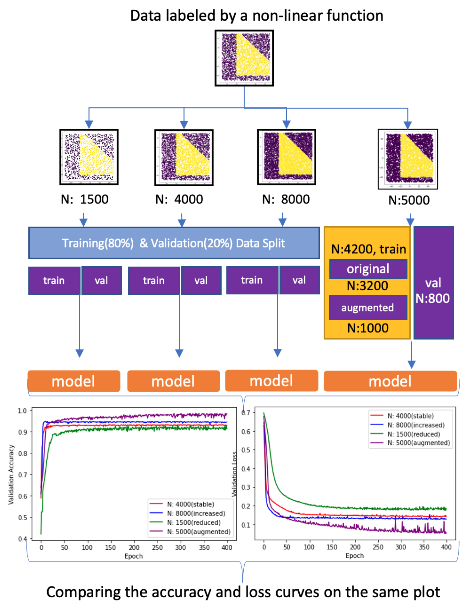


FIG. 1. General scheme of the two first steps: "Generating data" and "Training". In the first two scatter plots, the yellow dots correspond to points with the label "1" and the violet ones to the label "0". The two last graphs show the validation loss curves and the validation accuracy curves for each of the four numbers of samples against the number of epochs

D. Training

We defined a method called "train" that takes as an input a training set and a validation set. During our experiment, we used four different pairs of sets: "default" (N=4000), "decremented" (N=1500), "incremented" (N=8000), "augmented" (N=5000) -where N is the total number of samples in the whole dataset-.
First of all, the "train" method rescales the samples of both sets, dividing them all by the maximum value contained in their own set. Then, after setting the random seed, it creates the base model described in the previous dedicated subsection and performs a fit, using the aforementioned training set and validation set. Lastly, it returns both the model and the fit.
To show the fit results in an immediate way, we extracted the validation losses and accuracies from the "fit history"

and we plotted them against the number of epochs.
The general scheme of "Generating data" step and "Training" step is shown in Figure 1.

E. Grid-Search and Hyper-parameter Optimization

"Grid search" is a "model hyperparameter optimization method" that attempts to discover the optimal combination of parameters from a list of options. The "GridSearchCV" class of the "Scikit-learn" library implements this strategy. After that, the "GridSearchCV" algorithm will create and analyze one model for each set of hyper-parameters.
First of all, we wrote the function "train grid" that contains the rescaling of the input data, the definition of the "build classifier" method, and the implementation of the "KerasClassifier". The "build classifier" method creates and returns the Keras sequential model that is then passed as an argument to the "KerasClassifier". In order to find the best combination of hyper-parameters, lists of different optimizers, activation functions, dropout rates, and number of units in the hidden layers have been provided to the model as shown in Figure 2. Due to computational complexity and the consequent run time issues, the experiment has been performed with a restricted list of parameters for each class of hyper-parameters:

- Optimizers: SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax
- Activation functions: Softmax, Relu, Tanh, Sigmoid, Linear
- Dropout rates: 0.0, 0.1, 0.3, 0.5
- Number of units in hidden layers: 20, 25, 30

The "Grid search" algorithm implements an exhaustive research of all possible combinations of parameters and trains the final model with the one which provides the best mean test score.

Group Work Distribution

The whole experiment has been performed on Google Colab as a joint work by utilizing GPU environment. Group tasks have been divided as follows:

- Fatıma Rabia Yapicioglu: Coding the skeleton and modularizing methods for data generation and training; coding "Grid search" optimization procedure; creating the schemes in Figure 2 and Figure 3; computing Precision, Recall, F1-Score, Computing Matrix results for the model; contributing to the report in all of its parts
- Asal Rangraziasi: Plotting results of "Grid search" optimization procedure; writing abstract and part of introduction; joining the collective code discussion

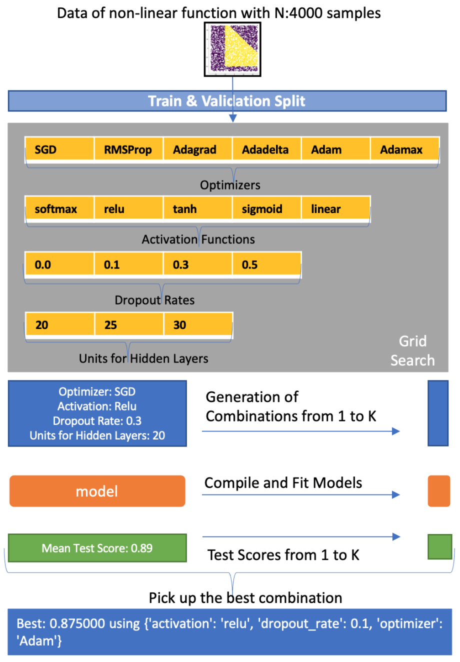


FIG. 2. Scheme of "Grid search" optimization procedure

- Giulia Bellentani: Modularizing methods for data generation and training; coding "data augmentation" procedure; writing especially subsections from A to D of the "Methods" section; revising the report in all of its parts
- Pu Yin: Writing abstract; searching and editing references; joining the collective code discussion

Results

Data preprocessing results

Firstly, the base model structure was trained with "Adam" as the optimizer and "binary cross entropy" as the loss function. We performed the training for 400 epochs. During each epoch, we took 500 samples from each batch. Four copies of this model have been trained with the four aforementioned different datasets: "default" (N=4000), "decreased" (N=1500), increased (N=8000), augmented (N=5000) -where N is the total number of samples-. The results are displayed in Figure 3, superposing the accuracy curves and the loss curves for comparison purposes.
According to Figure 3, the model trained on the augmented dataset reached the local minimum faster than the other models. Moreover, the model trained on the reduced dataset reached the local minimum slower than the other models.
Furthermore, according to Figure 4, the two models trained on the default and the augmented datasets classified correctly a higher number of samples than the other models, thus producing a more neat triangular shape. Thanks to the "augmentation labelling check"

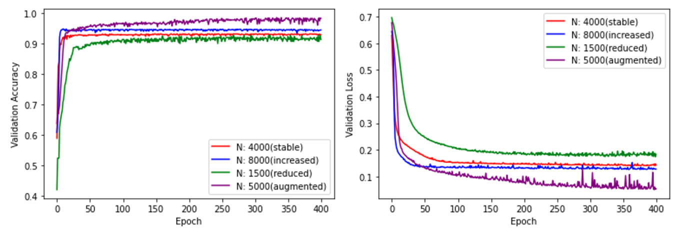
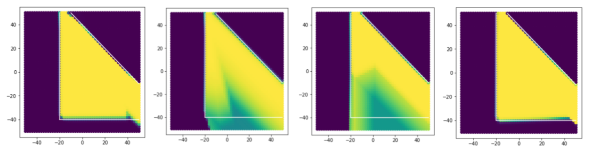


FIG. 3. Accuracy and loss curves related to the different data preprocessing techniques



(a). Stable, N: 4000 (b). Reduced Data Samples, N: 1500 (c). Increased Data Samples, N: 8000 (d). Augmented Data Samples, N: 5000

FIG. 4. Predictions of models trained with different number of samples. Yellow corresponds to label "1", violet to label "0"

explained in the dedicated section, we could suppose that the better performances of the augmented dataset are not linked to the augmentation procedure per se, but simply to a number of samples high enough to learn the model in a good way and small enough not to lead to overfitting. Searching for the best number of samples could have been a step further in this direction.

"Grid search" results

The results of the "Grid search" procedure are summed up in Table 1. We used the default dataset (4000 samples).

Rank	Optimizer	Activation	Dropout	Units	Score
1	Adam	Relu	0.1	25	0.883
2	Adam	Relu	0.3	25	0.880
3	Adam	Relu	0.5	20	0.876
4	Adam	Relu	0.3	20	0.869
5	RMSprop	Relu	0.1	20	0.865
6	RMSprop	Relu	0.3	20	0.858
7	Adam	Relu	0.0	25	0.857
8	RMSprop	Tanh	0.0	20	0.851
9	Adam	Relu	0.1	20	0.848
10	Adam	Relu	0.0	20	0.843

TABLE I. "Top 10 ranking" of the results of "Grid search" hyper-parameter combinations (Optimizer, Activation Function, Dropout Rate, Hidden Layer Units)

It is worth mentioning that the models used during "Grid search" optimization procedure have been trained with only 10 epochs, because of computational complexity and consequent "run time saving" necessity.
The model that performed best regarding "mean test score" was the one trained with Adam optimizer, Relu activation function, dropout rate "0.1", and 25 units in each one of its hidden layers. It should be highlighted that this "winning" model is related to an optimization ranking that relied on only 10 epochs, so this ranking could change using a higher number of epochs: this is one of the main reasons why we desplayed all the "top 10" hyperparameter combinations.
To further inspect the performance of a model initialized with our best hyper-parameter combination, we trained it over 250 epochs: the evaluation metric results can be analyzed in Table 2.

Class	Precision	Recall	F1-Score	Support
0	0.63	0.90	0.74	460
1	0.68	0.28	0.40	340
Average	0.65	0.64	0.60	800

TABLE II. Precision, Recall, and F1-Score of the final model

Moreover, in Figure 5 we show the confusion matrix that belongs to this model. A step further in the improvement of this model performance could be adding the number of hidden layers to the categories of hyperparameters involved in the "Grid search" optimization algorithm.

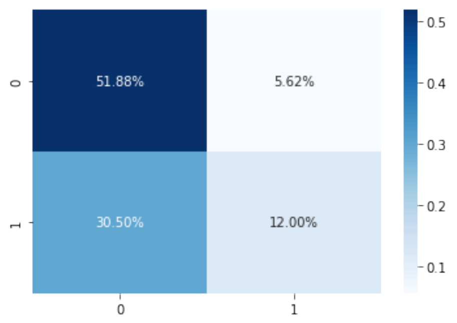


FIG. 5. Confusion Matrix for the model which is trained with best hyper-parameter combination according to Table 1.

Conclusions

The first goal of this paper was to build a deep neural network and analyze its performance in relation to different data preprocessing (change in the amount of data, data augmentation) and different combinations of hyperparameters. We have seen that both these aspects play an important role in the optimization of the model. According to our results, using a number of samples around 5000 and applying the aforementioned best hyperparameter combination found with "Grid search" could lead to a model that increases its performance in terms of generalization and "overfitting/underfitting" avoidance.

[1] Sonali, B. Maind and Priyanka Wankar. "Research Paper on Basic of Artificial Neural Network." (2014).

[2] Liu, Weibo Wang, Zidong Liu, Xiaohui Zeng, Nianyin Liu, Yurong Alsaadi, Fuad. (2016). A survey of deep neural network architectures and their applications. Neurocomputing. 234. 10.1016/j.neucom.2016.12.038.

[3] Baydilli, Yusuf Atila, Umit. (2018). Understanding effects of hyper-parameters on learning: A comparative analysis.

[4] Fouki, Mohammed Aknin, Noura el kadiri, kamal eddine. (2019). Multidimensional Approach Based on Deep Learning to Improve the Prediction Performance of DNN Models. International Journal of Emerging Technologies in Learning (iJET). 14. 30. 10.3991/ijet.v14i02.8873.

[5] S. C. Wong, A. Gatt, V. Stamatescu and M. D. McDonnell, "Understanding Data Augmentation for Classification: When to Warp?," 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA). (2016)

[6] Yan et al.Improved relation classification by deep recurrent neural networks with data augmentation. (2016)