WASC – Deep Neural Network

Walter Zigliotto, Andrea Nicolai, Sandeep Kumar Shekhar, and Camilla Quaglia

(Dated: May 7, 2020)

This work aims to develop a two layers Deep Neural Network (DNN) algorithm able to predict whether a specific sequence contains secret keys. For such purpose, using a given neural network architecture, data processing techniques are used to manage the initial dataset. Furthermore, few DNN hyperparameters, such as activation function, regularization coefficient, dropout rate and optimizer, are varied in order to improve the results. Finally, the outcomes from two different versions of Tensorflow are compared.

Keywords: Data processing, Deep Neural Network, Hyperparameters, Tensorflow.

INTRODUCTION

Deep Neural Network in principle is a supervised machine learning technique. DNN is a discriminative model, which utilizes the concept of "weights" for data classification. The human brain is composed of tiny functional units called neurons. A similar analogy is adopted for building the DNN architecture. The DNN architecture classifies data set using different layers. The main structure is based on the following layers:

- **Input layer:** Receives initial or preprocessed data.
- **Hidden layers:** Intermediate layer, the black box of the system, where the weights are set and computation is achieved.
- **Output layer:** Returns the predicted label of the classification for a given input.

Each neuron receives an input, which is a linear combination of neurons from the previous layer, weighted by the edge weights. Further the result is transmitted to a non-linear activation function and is obtained as the output. Mathematically, this process can be described by the following equation:

$$o_{t+1,j}(x) = \sigma\left(\sum_{r:(v_{t,r},v_{t+1,j})} w(v_{t,r},v_{t+1,j})o_{t,r}(x)\right) \quad (1)$$

Where $o_{t+1,j}(x)$ is the output of the neuron, $w(v_{t,r},v_{t+1,j})$ is the weight of each neuron connected with $o_{t,r}(x)$(i.e., the outcome of the previous neuron layer) and σ is the activation function [1]. Several possible activation functions (e.g., sigmoid function, Rectified Linear Unit (ReLU)) can be chosen and this choice affects the action potential firing of the neuron.

Once the structure is defined, DNN has to be trained in order to improve its performance. This procedure is done through a gradient method and a backpropagation algorithm. There are several gradient techniques, called optimizers. Optimizers are used to minimize the cost/loss function and to find the optimal weights and biases. Some of them are Stochastic Gradient Descent

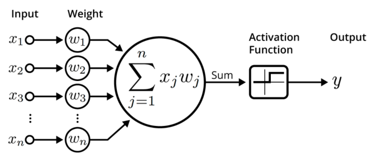

FIG. 1: Schematic representation of a single neuron behaviour.

(SGD), Adam and Nadam. Alongside optimizer, backpropagation algorithm is deployed, which is a clever procedure that rearranges the weights of the layered structure of Neural Network (NN) in order to improve the results [2]. So to optimize the outcome, several algorithms are employed.

The generalization of DNN structure to a new dataset might lead to an overfitting problem. To tackle this issue, regularization techniques become handy. Primarily, Regularized Loss Minimization (RLM) methods are introduced to minimize the sum of the empirical risk and a regularization function. This regularization function acts as a stabilizer for the learning algorithm, so that the slew rate is unaffected. Basically, the idea is to balance the complexity of hypothesis (e.g., if the empirical risk is low, but the complexity is high, probably leading to overfitting). For this purpose $L2$ norm function is utilized. Another tool used is Dropout. The idea of 'Dropout' is to prevent overfitting by reducing spurious correlations between neurons within the network by introducing a randomization. It consists of randomly "dropped out" neurons of the neural network with some probability giving rise to a thinned network.

METHODS

The NN implemented is trained with the aid of Tensorflow 2.1.0, which already contains Keras API implemented [3]. In addition to this, a different configuration (Keras API v. 2.2.4 using Tensorflow v. 1.14.0 backend) [4] is implemented and different results are obtained, despite the use of the same set of best parameters.

The dataset constitutes a sequence of 7 digits, that might contain a couple of subsequences of two digits each. If so, then the correct label is 1, thus becoming a problem of binary classification. Each possible digit of the sequence is indeed in the set $D \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$: therefore it fixes some constraints to the shape of the first layer, since each number in D could be present in any of the seven positions of the given sequence. These arguments give rise to the condition that the input layer needs to be composed of exactly $D \cdot L = 63$ nodes, whose value can be either 0 or 1. Moreover, following the instructions that are provided, we are forced to obey the constraints that the first and the second layers must be formed at most respectively by $(DL/2, DL/4)$ vertices.

The problem we face is regarding the binary classification, so the activation function in the final layer of our NN is chosen to be a sigmoid and the loss is the binary cross-entropy. In order to find the best set of parameters, we consider to implement a CV grid search, using 80% of the whole dataset as a training set, thus splitting it into 10 folds, and using a batch size of 50. The training lasts 40 epochs and we reshuffle the set after every iteration. The seed set for the complete process is 1234. Finally, after having obtained the best set of parameters, we try to predict the labels of the remaining 20% dataset, namely the test set.

The parameters to be tuned are only four in number. Therefore, we conclude that a good compromise is by using just a couple of CV Grids Search: the first one allows us to find the best Optimizer and Ridge Regularization coefficient parameters together, while the others are the activation function and Drop out rate. The listed out parameters are to be set equal for all the layers, except for the penultimate output layer whose nodes are “burnt out” according to the Drop out rate.

We fancy to apply different transformations to our dataset, to figure out if it leads to the improvement in the performance of our trained NN. We expect to see different set of parameters, corresponding to different transformations. At first we use the raw dataset consisting of 3000 entries. We expect the accuracy not to be very high, and found our results accordingly.

The next step is to shift and rescale the data. By rescaling we mean that, for each of the $D \cdot L = 63$ mentioned earlier, we subtract its mean and set its standard deviation to unity by normalization. Once this transformation is applied, we use Grid Search CV.

Finally, we notice in our dataset there is an invariance of the label with respect to shift of the digits. This is translated to the fact that if a certain sequence (e.g. 3456**7**89) contains the key, then the $L-1$ equivalent sequences also hold it (45**67**893, 9345**67**8 and so forth). In order to augment our dataset, we define the following function that accepts input as the data and its labels and returns the augmented dataset:

```
def expand_augment(data, label):
  S = data

  x_temp = [0] * LD  #initialize empty
  ↪  vectors
  y_temp = label

  p = 10**(L-1)
  j = 0
#in this way we obtain the first digit (MSD)
  while (j < L):
    q = int(S/p)

    x_temp[j*D + (q-1)] = 1
    j += 1
    S = S - q*p
    p = int(p/10)

  x_aug  = [0] * LD * L
  x_aug  = np.reshape(x_aug, (L, LD))
  y_aug  = np.array([0]*L)
#for each combination possible by shifting
  for combination in range(L):
  #for each possible number
    for index in range(LD):
      x_aug[combination][(index +
      ↪  combination*D)%LD] = x_temp[index]
      y_aug[combination] = label

  return x_aug, y_aug
```

In this way we are able to obtain a larger dataset by a factor of $L-1=6$. As previously done, we train our NN using a Cross Validation Grid Search in order to find the set of parameters that best predicts the labels of the test set.

RESULTS

Tensorflow version: 2.1.0

The following results are obtained using the version 2.1.0 of Tensorflow.

From the CV Grid Search performed over the small dataset (3000 entries), we find the best parameters as shown in the first column of Table I.
The model created with the best parameters above is trained over all the epochs using the training set. The accuracy reached and the loss function are shown in Fig

2 and 3 respectively, where the blue line and the orange line represents the training and validation set.

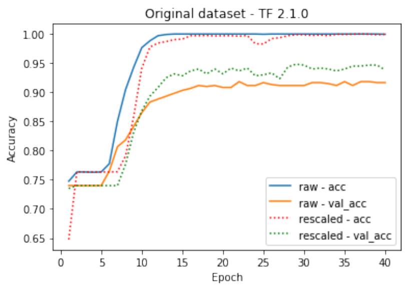

FIG. 2: Accuracy vs Epoch for the small dataset.

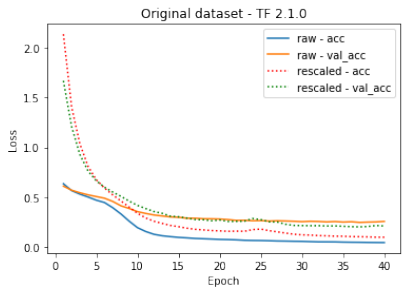

FIG. 3: Loss at different epochs for the small dataset.

From the Fig 2, we can see that the training set reaches an accuracy of 100%. The latter remains constant after about 12 epochs, for both training and validation set, so there is no gain in training the network after the 12th epoch. Moreover, we come across a variance between the training and the validation set (orange and blue lines). This relation between the sets is undesirable and is dealt later in the following paragraphs. The green and the red dashed lines from Fig 2 and 3 represent the rescaled data. Also, we can notice the improvement in the performance of the network after data rescaling. The best parameters found by the Grid Search procedure for this dataset are shown in the second column of Table I. The rescaled data is obtained mathematically and is given as:

$$x_i = \frac{x_i - E(x_i)}{\sqrt{var(x_i)}} \qquad i = 1....n \tag{2}$$

Where n is the total number of samples and $E(x_i)$ is the mean of the 63 entries. Even though we see a performance enhancement, it is still not the ideal result required. The variance between training and the validation set is solved through the 'data augmentation' procedure. This procedure allows us to obtain a larger dataset, (21000 entries), which is divided into training (80%) and validation (20%) sets, as done previously. As seen in the methods section we point that there is an invariance by shifting digits to left or right. The grid search procedure carried out over this bigger dataset gives the best parameters. The best parameters for the augmented dataset are shown in the third column of the Table I and for the augmented rescaled dataset in the successive column.

Similarly, two plots (Fig 4 and 5 respectively) indicate the accuracy of the methods and the cost function over the epochs. We deduce that the network performs at its best for the augmented data and the maximum accuracy (100%) is reached, for both the training and the validation set.

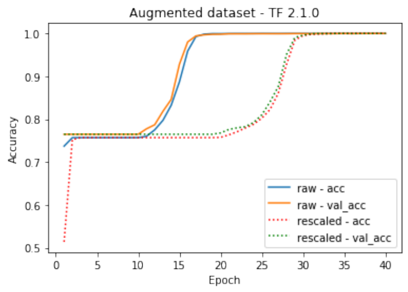

FIG. 4: The variation in accuracy for the augmented dataset.

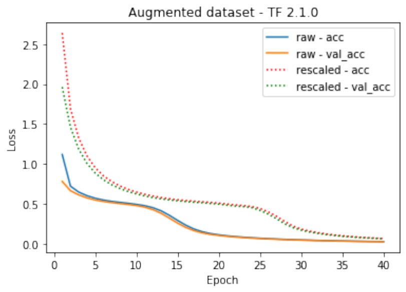

FIG. 5: Loss depicted for the augmented dataset.

TABLE I: Best parameters

Dataset:	**Small**	**Small**	**Augmented**	**Augmented**
Optimizer	Adam	Adam	Nadam	Nadam
Regularizer	0.01	0.05	0.05	0.1
Activation function	Relu	Relu	Sigmoid	Sigmoid
Dropout	0.25	0.2	0.15	0.15
		rescaled		**rescaled**

Tensorflow version: 1.14.0

A similar procedure is implemented, with the same set of parameters in Table I (even the seed), and different results on the same dataset are obtained, by using an older version of Tensorflow (1.14.0).

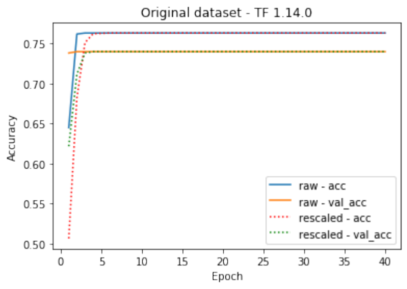

FIG. 6: Accuracy vs Epoch for the small dataset.

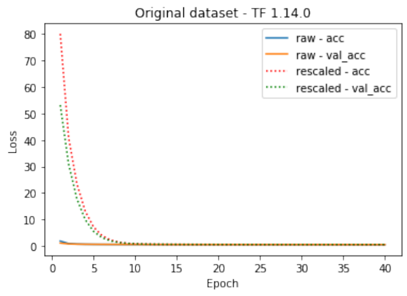

FIG. 7: Loss for the small dataset.

The accuracy remains flat for all the epochs after an initial increment (Fig 6), for both raw and rescaled data. As in Fig 2, we notice that the variance between the training and the validation set is not negligible. For this case, rescaling is not beneficial, since the raw data (continuous lines) and the rescaled data (dashed lines) superimpose on one another.

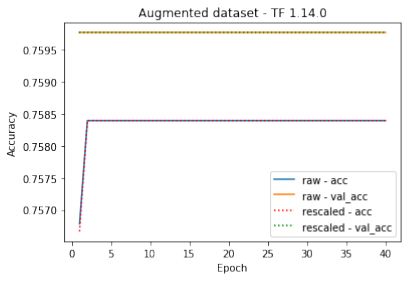

FIG. 8: Accuracy vs Epoch for the augmented dataset.

The behaviour of the accuracy in Fig 8 is almost the same than the one in Fig 6. A perfect superimposition is found between the raw and the rescaled data. There is no considerable improvement with respect to the small dataset. The code and the dataset are available at the following link [5].

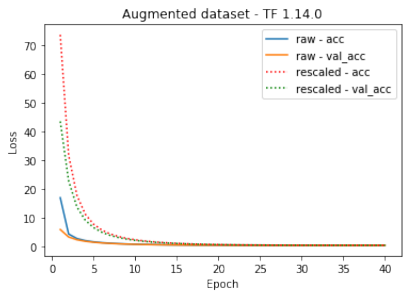

FIG. 9: Loss for the augmented dataset.

CONCLUSIONS

This research stresses out once again that in Machine learning the amount of data we have is fundamental, even when dealing with simple problems: we are encouraged to find if there are symmetries in our dataset and exploit them, in order to have better performances. Another important point is that the version of Tensorflow affects the results, which are generally better for the newer one. Of course we can not dwell deeper into the changelog between the two versions. Therefore, this encourages us to use the latest release.

REFERENCES

[1] S. Shalev-Shwartz and S. Ben-David. Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, 2014.
[2] Mehta, P. et al. A high-bias, low-variance introduction to Machine Learning for physicists. Phys. Rep. 810, 1–124 (2019).
[3] https://keras.io/
[4] https://www.tensorflow.org/api docs
[5] https://github.com/walterzigliotto/DNN