ClusterPro – Deep Neural Networks with Keras

Barzon Giacomo, Kabbur Hanumanthappa Manjunatha Karan, and Wolfgang Rugel
(Dated: May 7, 2020)

We create a supervised machine learning model using Deep Neural Network which learns to detect secret-keys in a dataset similar to secret signals of "steal" or "no steal" conveyed by players in Baseball game. The network thus predicts the labels of "steal" hence able to crack the code of the opponent team. We do the grid search to find the best model among many. Along with this we gain a better understanding of the role of various parameters and compare and contrast the hyper-parameters performance relevant to our model as well as its correlation among them. We record top 10 performing models with accuracies up to 93.5%.

Keywords: Deep Neural Network, grid-search, baseball secret-keys, learning rate, optimizer, activation function.

INTRODUCTION

Our work has been inspired mainly from the Baseball secretkeys Youtube video by Mark Rober [7]. The leading idea is to mimic the secret-signals that players convey to each other if there is a 'steal' or 'no steal' in baseball game. The generated data are apparently random and some of them include the so-called secret-keys representing the secret-signals using a simple algorithm. The dataset consists of N data points, given by $\mathbf{x}$ which are strings of L=6 digits and each digit is from 1-9. The labels $\mathbf{y}$ is also vector of length N of integers either 0 or 1. This dataset is in such a way that with certain probability we have the secret key inside the string and the remaining probability we do not. There are also additional secret-keys embedded for some of the strings. Our model consisting of Deep Neural Networks from Keras package [1] are employed to learn the data and predict the labels which is same as predicting whether there is 'steal' or 'no steal' from data with secret keys hidden in them.

METHODS

Augmenting of the dataset – We have assumed the invariance of the results with respect to shift of the digits in a data sample (e.g. 1234567 and 7123456 gives the same result y=0 or y=1). So as to improve the accuracy of the model over the validation data set, we have "augmented" the data. For our dataset where each string is of length L=6, we have 5 more equivalent ones which are obtained by a simple algorithm to cyclically rotate the digits for a given string.

One Hot encoding of the dataset – One hot encoding comes under the pre-processing of data before being given as an input to the Machine learning model. Each digit in a string is 1-9 and so we have 1-9 categories.
If we were to use our dataset directly, which is an integer based encoding, allowing the model to assume a natural ordering between categories may result in poor performance. In this case, a one-hot encoding can be applied to the integer representation. This is where the integer encoded variable is removed and a new binary variable is added for each unique integer value [4, ch.9.2].

Rescaling – Often the network's performance is improved by rescaling the input to an interval $[-1, 1]$. This avoids large values in the activation function, for which its gradients vanish. It ensures that the weights of the DNN are of a similar order of magnitude.
As seen in Figure 1, for our data rescaling does not show strong effects. Here, all data has a similar and low order of magnitude, so flattening effects are negligible and the method is not expected to be impactful.

FIG. 1: Training and Test accuracy with and without data rescaling.

Network architecture – The network considered in this work is a deep neural network made of four fully-connected layers. It consists of an input layer with the same size as the samples, two hidden layers with half the size of the previous one and a final output layer consisting of a single unit, since the labels are a single binary value (Table I). Each unit in the network, called *neuron*, pro-

cesses the weighted inputs coming from the previous layer with an activation function. With the given network architecture we compare the effects of the hyperparameters on its predictive abilities.

Layer type	Input shape	Output shape	# Params
dense	63	63	4832
dense	63	31	1984
dense	31	15	480
dense	15	1	16

TABLE I: Network architecture.
Total number of parameters: 6512

Training procedure – In the training, weights of the network are adjusted such that its predictive and generalizing power are improved. In practice one minimizes the cost function with gradient descent methods [4, ch.4].
The binary nature of the labels suggests the use of the *binary cross-entropy* as the cost function together with the sigmoid as activation function of the output layer: in this way the network maps each sample to a real value between 0 and 1, which represents the predicted probability of having a secret-keys in the sample [2]. A L2-regularization term is added to the cost function in order to penalize more complex networks (i.e. bigger weights) and try to improve its generalization abilities [4, ch.6], so the cost function becomes:

$$C(\mathbf{w}) = -\sum_{i=1}^{N} y_i \log \hat{y}_i(\mathbf{w}) + (1-y_i)\log[1-\hat{y}_i(\mathbf{w})] + \lambda\|\mathbf{w}\|^2,$$

where $\mathbf{x}$, $\mathbf{y}$ are samples and the true labels, $\mathbf{w}$ the weight matrix and $\hat{\mathbf{y}}(\mathbf{w})$ the predicted labels. λ is the regularization parameter, which determines how much to penalizes the norm of the weights $\mathbf{w}$.
At each training step, called *epoch*, the weights of the network are updated by estimating the gradient of the cost function on a subset of the training samples of size m called *minibatch* to speed up the training procedure. The initial values of the weights are drown from some probability distribution known as *weight initializers* [3]. Some techniques are introduced to prevent from overfitting. A common method is the dropout, which consists of excluding neurons with a fixed dropout rate randomly in each training step, so the network tends to learn less specific characteristics of the training data [8]. Further we use early stopping to interrupt the learning process whenever the test loss has stopped improving (bounded to a maximum number of epochs of 200) [6].

Grid-search – A grid-search is carried out among the hyperparameters reported in Table II, which are common choices for the fine tuning. This leads to a comparison of $3^7 = 2187$ models in order to find the best parameter values.
The activation functions and optimizers are defined in [4, ch.4,9], while the weight initializers in [3]. Instead the following hyperparameters are fixed, otherwise the total number of analyzed models would become too large: momentum and weight decay for SGD: 0.7, 10^{-6}; ρ for RMSprop: 0.9; β_1 and β_2 for Adam: 0.9, 0.999.

optimizer	SGD	RMSprop	Adam
learning rate	10^{-1}	10^{-2}	10^{-3}
activation function	ReLU	sigmoid	tanh
weight initializer	orthogonal	Glorot uniform	He normal
regularization	10^{-3}	10^{-4}	10^{-5}
dropout	0.0	0.2	0.4

TABLE II: Values of the hyperparameters in the grid-search.

RESULTS

All the training and validation curves and the best weights are available on Github, together with the dataset and the code used for the training of networks.

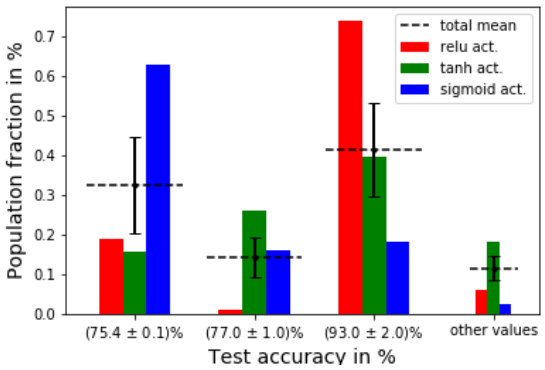
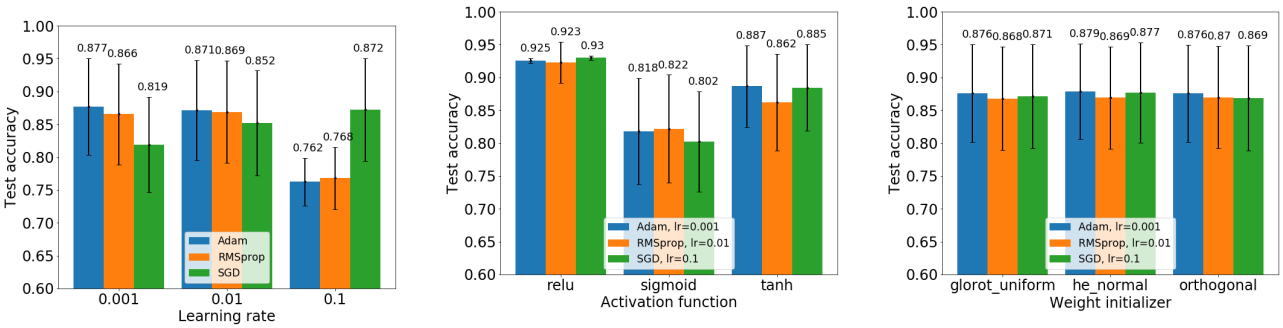


FIG. 2: Histogram showing the fraction of models with test accuracies equal to 75%, 77% or 93%. "other values" represent the models with different test results.

General results – As all models perform well with test accuracies within 75% and 93%, we consider a relevant subset of the hyperparameter space.
Remarkably only three accuracy values describe the performance of 88% of the models, as one can see from the dashed lines in Figure 2. Since the number of relevant test results agrees with the amount of tested values per hyperparameter, one could assume that a decisive hyperparameter causes that behavior. However, models constrained in one parameter do not have different test



(a) Learning rate vs. Optimizer

(b) Activation function vs. Optimizer with best learning rate

(c) Weight initializer vs. Optimizer with best learning rate

FIG. 3: Histograms with the average test accuracy values for coupled hyperparameters. We consider the models with the selected values of hyperparameters, marginalizing over the remaining hyperparameters.

results, therefore we reject the hypothesis. The strongest deviation from this three-valued pattern is shown in Figure 2.
The discrete set of achieved accuracies could instead represent the well defined steps in the understanding of the secret keys. As the network learns new parts of a key, one expects the observed stepwise increase of the test accuracy.

Optimizer and learning rates – Optimizer and learning rates are the two parameters which are highly related and so were chosen to be analyzed together. By keeping different optimizers and different learning rate fixed for all the various combinations of models, an average over training and test accuracy as well as standard deviations were found. The various mean test accuracies along with the corresponding standard deviation so obtained are visualized in Figure 3a. From the plot it is evident that Adam and RMSprop do not seem to perform better with higher learning rates of 0.1 whereas SGD performs better with higher learning rates than lower ones. Adam and RMSprop have differences in accuracies from 3rd decimal place showing us their performance is quite similar. Comparing the bar plots, we see Adam optimizer with lower learning rates is the better performer which is giving us higher accuracies (both train and test) relative to RMSprop and SGD.

Activation function – In order to analyze the effect of activation functions and weight initializers, we consider only the models with the best learning rate for each optimizer shown in Figure 3a. As can be clearly noticed from the Figure 3b, the test accuracies are independent of the chosen optimizer, while there are significant differences between the various activation functions: in fact the ReLU (test accuracy of 92%) exceeds the models trained with tanh-activation, for which the average accuracy is less than 90%.

In addition, most of the models trained with the sigmoid do not seem to converge to appreciable performance. A possible explanation could be the so-called *vanishing gradient problem* [5]: the derivative of e.g. the sigmoid becomes close to zero as the input becomes larger or smaller, meaning that the weights are not updated effectively and they get stuck in those regions. The ReLU activation instead does not saturate and avoids the problem.

Weight initializer – The network performance appears to be independent of weight initializers (cf. Figure 3c), although significant effects have been shown in [3] due to different initializers. This might come from the simplicity of the network and the dataset.

Regularization parameter – The optimal value for the regularization parameter λ is hardly predictable. However one observes that the network is sensitive to this parameter. By choosing a certain λ and considering the average test accuracy of all models with that regularization parameter, the mean performance increases remarkably with lowering λ, from $(81.0 \pm 0.5)\%$ for $\lambda = 0.01$ to $(87.0 \pm 0.5)\%$ for $\lambda = 10^{-5}$. For even lower λ, one might find higher accuracy values.

Dropout – An indicator for overfitting is a large positive difference between train and test accuracy. This value is in fact reduced from $(0.1 \pm 0.05)\%$ to $(0.04 \pm 0.02)\%$ as one introduces a dropout $p \in \{0.2, 0.4\}$. We also observe a decay as we increase the dropping rate, from $(86.0 \pm 0.1)\%$ ($p = 0$) to $(83 \pm 0.1)\%$ ($p = 0.4$). A possible explanation is the absence of dropout neurons in the training procedure. Since skipping neurons during a training step effectively reduces the number of training cycles for each neuron, higher dropout rates could come with the price of less accuracy.

Optimizer	L.Rate	Activation	W.Initializer	Regularization	Dropout	B.Size	TrainAcc	TestAcc
RMSprop	10^{-1}	sigmoid	He normal	10^{-5}	0.2	64	0.9342	0.9352
SGD	10^{-1}	tanh	Glorot uniform	10^{-5}	0.2	32	0.9357	0.9350
RMSprop	10^{-2}	sigmoid	He normal	10^{-5}	0.2	32	0.9351	0.9348
RMSprop	10^{-3}	relu	Glorot uniform	10^{-5}	0.2	16	0.9352	0.9348
RMSprop	10^{-2}	sigmoid	He normal	10^{-5}	0.2	16	0.9352	0.9348
RMSprop	10^{-2}	sigmoid	orthogonal	10^{-5}	0.2	16	0.9342	0.9348
RMSprop	10^{-2}	sigmoid	orthogonal	10^{-5}	0.4	32	0.9347	0.9348
RMSprop	10^{-2}	sigmoid	Glorot uniform	10^{-5}	0.4	32	0.9349	0.9348
Adam	10^{-2}	ReLU	Glorot uniform	10^{-4}	0.0	32	0.9353	0.9348
Adam	10^{-2}	ReLU	Glorot uniform	10^{-4}	0.0	16	0.9344	0.9348

TABLE III: Top-10 networks with higher test accuracy.

CONCLUSIONS

In this work we show the effectiveness of a Deep Neural Network in predicting the secret-signals in a baseball game. Precisely, the network is able to predict almost correctly (up to 95%) the presence of a hidden key for some choices of the hyperparameters.
In addition, some considerations emerge from the tuning of the hyperparameters. The ReLU outperforms the other activation functions, while smaller regularization parameters seems to improve the generalization capabilities of the network. Adam optimizer leads to slightly better accuracy. Introducing a dropout reduces overfitting as well as training performance such that in rare cases the test exceeds the training results. Although the network was invariant for weight initializers, we find some correlations between optimizers, learning rates and activation functions.
Even though these results are true on average, they do not reflect the best performances (Table III).

[1] François Chollet et al. 2015. Keras. https://keras.io

[2] Daniel Godoy. 2018. "Understanding binary cross-entropy/log loss: a visual explanation." Towards Data Science. https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

[3] Daniel Godoy. 2018. "Hyper-parameters in Action! Part II - Weight Initializers." Towards Data Science. https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404

[4] Mehta, Pankaj et al. 2019. "A High-Bias, Low-Variance Introduction to Machine Learning for Physicists." Physics Reports 810 (2019): 1–124.

[5] Michael Nielsen. 2019. "Why are deep neural networks hard to train?", in "Neural networks and deep learning." http://neuralnetworksanddeeplearning.com/chap5.html

[6] Piotr Skalski . 2018. "Preventing Deep Neural Network from Overfitting." Towards Data Science. https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800a

[7] Mark Rober. 2019. "Stealing baseball signs with a Phone (Machine Learning)" https://www.youtube.com/watch?v=PmlRbfSavbI

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. "Dropout: a simple way to prevent neural networks from overfitting." J. Mach. Learn. Res. 15, 1 (January 2014), 1929–1958.