MaFalDDa – Convolutional NN for analyzing time series

Delfina Iriarte, Davide Maniscalco, Mattia Pujatti, and Fabio Semenzato
(Dated: April 14, 2020)

The purpose of this work is studying several aspects of the training with convolutional neural networks, used in this case to analyze time series. Data are generated sampling an exponential distribution, and then a pattern in the data is introduced using a sine function. Some convolutional neural networks are built and their performances tested, with the constraint of having less than 600 parameters. Finally, also the impact of the regularization is studied, together with new different data, generated with a different signal-to-noise ratio.

INTRODUCTION

Convolutional Neural Networks (CNN) gained considerable interest during the past years both in the field of speech and image analysis. One of the main features is that the model complexity and the number of weights is efficiently reduced by *weight sharing* and restriction to local connections. Moreover, a convolutional neural network can be described as a translationally invariant neural network that respects locality of the input data [1].

Within a convolutional architecture, there are several possibilities for combining different kind of layers and learning rules. A typical architecture of a CNN is shown in Figure 1, on the right, and it is formed by Convolutional layers, Pooling layers, and Fully-Connected Layers.

The purpose of this paper is to study the several hyperparameters of a CNN, using as a dataset a diffusive time series, and to compare the performances of different models proposed.

METHODS

Data Generation

The data consist of time-series samples. Each sample is composed of 60 time steps and each time step is linked to the one before as $y_i = y_{i-1} + dy + b$, where dy is a random noise, which amplitude is controlled and independent from the value of y, and b is a constant bias, which causes the sample to have an overall positive slope; the first time step refers to the last time step of the previous sample. Then, to each of these samples thus generated, we added one of these 3 signals, each one with probability 1/3:

- *Zero Signal*: the sample has been left unchanged; we will assign them to Label 0;
- *Positive Signal*: we added, at a random position inside the sample, the positive phase of a sinusoid, with constant period and amplitude; we will assign them to Label 1;
- *Negative Signal*: we added, at a random position inside the sample, the negative phase of a sinusoid, with the same period and amplitude of the positive one; we will assign them to Label 2.

An example of data analyzed is given in figure 1 left (the data shown are already prepared as explained in the section *Results*). Recall that this is a supervised learning problem.

Data Preparation

Since there is a constant additive bias at each time step, the magnitude of the values keep increasing from sample to sample. This would make it almost impossible for the neural network to detect the patterns, since the high variance of the inputs would make the optimization really unstable. Therefore, we had to subtract the mean out of each sample. Moreover, a trend inside the sample would slower the training as well, thus, we subtracted from the n-th time step the mean of the n-th time steps of all the samples.

Search for the best model

In the search for the best model, various points have been taken into account:

- **Restrictions**: for this project, we have been asked to provide the best architecture with a number of trainable parameters less than 600.
- **Initializer**: a random normal initializer was used in the beginning, however we noticed that switching to the He normal initializer improved the training speed.
- **Data Augmentation**: beside the restriction on the number of parameters of our model, another limiting one has been the small number of training sample. In order to solve this, we opted for a Gaussian-Noise Layer as the initial layer, which would act on the input before passing it to the convolutional layer.

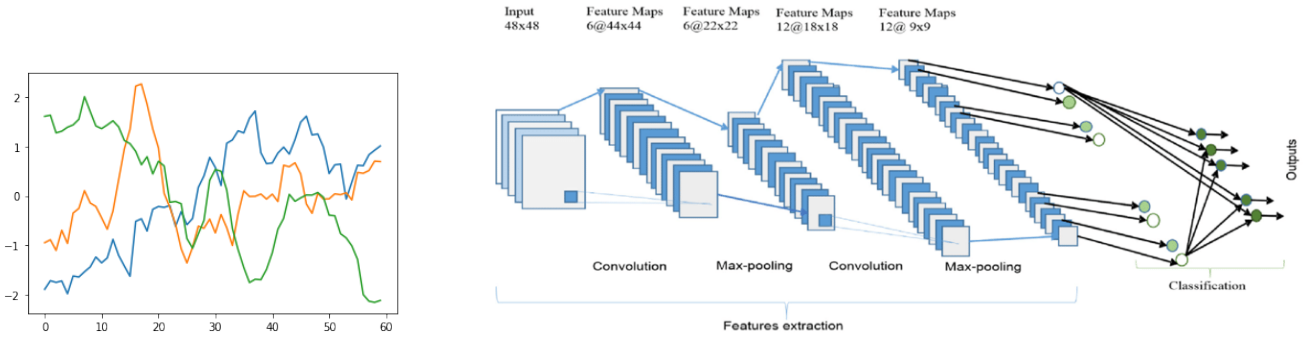

FIG. 1. *Left*: in orange, green and blue are represented three data samples. *Right*: example of architecture of a convolutional neural networks with their typical layers.

- **Convolutional Layers**: the kernel size of the first convolutional layer should be large enough to include the half-period of the sinusoid. Moreover, the number of filters should be at least 2, in order to detect the positive and negative phases of the sinusoid (the Zero Signal could be detected by a low activation of these two filters).
- **Pooling Layers**: the number of pooling layers, as well as their pooling size, highly influence the number of parameters of the convolutional part, as well as the number of connections with the fully connected part. Thus, they have been useful in order to control the number of total parameter and to restrict it to be less than 600. As regarding the type of pooling layer, both the Average Pooling and the Max Pooling have been tried.
- **Activation Function**: the activation functions we tried are: ELU, ReLU and PReLU. The problem with the PReLU is that, by default, it trains a different slope for every neuron of the layer, thus increasing a lot the number of trainable parameters. However, it is possible to share a single slope among various neurons, in order to reduce the number of additive trainable parameters.
- **Regularization**: we needed to introduce some regularization in order to make the training more stable and to avoid overfitting. Thus, we added dropout layers between the fully-connected ones and a weight regularizer that we chose between l1, l2 and a mixture of these two.

RESULTS

Best Model Found

After an extensive search of the optimal architecture and a fine tuning of the hyperparameters, the best model found is the following:

Gaussian Noise Layer, with standard deviation 0.07

Convolutional Layer, with 7 filters, kernel size of 15 and a $l2$ regularization with $\lambda = 0.01$

PReLU Activation Function, with a different trainable slope for each convolutional filter

Max Pooling, with pooling size 6 and a padding copying the first time step to the left and last time step to the right

Convolutional Layer, with 6 filters, kernel size of 3 and a $l2$ regularization with $\lambda = 8 \cdot 10^{-5}$

PReLU Activation Function, with a different trainable slope for each convolutional filter

Dense Layer, with 8 neurons and a $l2$ regularization with $\lambda = 8 \cdot 10^{-5}$

Dropout Layer, with probability 0.16

Dense Layer, with 3 neurons and softmax activation function, which output corresponds to the probability of belonging to each of the 3 labels

Layer (type)	Trainable Parameters
Gaussian Noise	0
Convolutional	112
PReLU	7
Max Pooling	0
Convolutional	132
PReLU	6
Flatten	0
Dense	296
Dropout	0
PReLU	1
Dense	27
Total	581

TABLE I. Summary of the best model along with the trainable parameters.

In Table I we report a brief summary of the best model along with the trainable parameters of each layer.

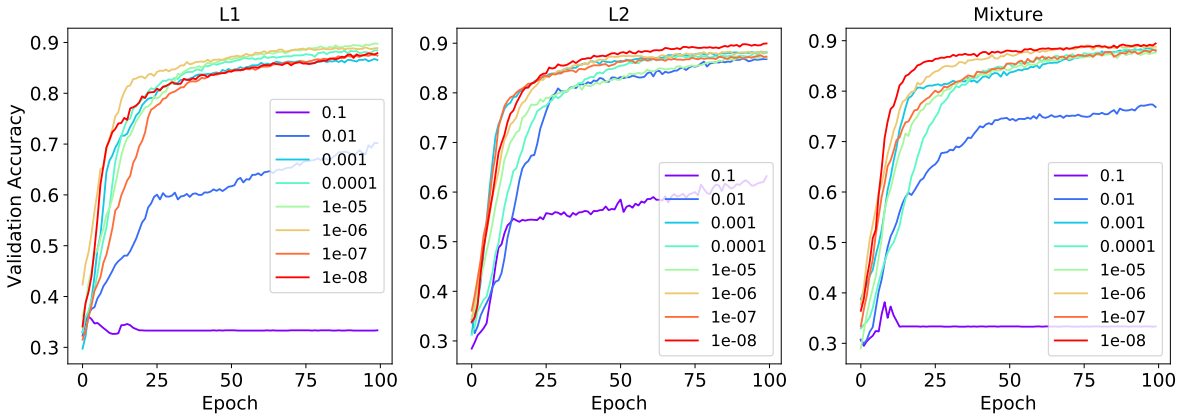

FIG. 2. Accuracy over the validation set for the regularization (a)*l1*, (b) *l2*, (c) *l1_l2*; the values reported in the plots represent the weight of the regularization in the loss function.

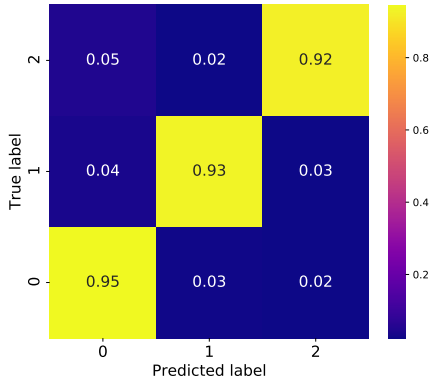

FIG. 3. Normalized confusion matrix on the validation set for the best model.

Confusion Matrix

We report in Figure 3 the confusion matrix of the best model found. We can see that all the 3 classes are well separated and there are no strong cases of confusion between two classes. As predictable, the class which has the highest TPR (True Positive Rate) is the *Zero Signal* one, since it does not require a structured convolutional filter to be detected.

Analysis of regularization

In Figure 2 we report the training evolution of the validation accuracy, using various regularizations for our model. As explained in Section *Methods*, we tried three different regularization losses: *l1*, called *Lasso Regression*, *l2*, called *Ridge Regression*, and a symmetric mixture of these two. The values reported in the plots represent the weight of the regularization in the loss function. We can notice that the model seems to perform better with smaller regularizations; moreover, with high regularizations the training starts to fail. It is possible to see that the *l1* method seems to have a slower convergence, while the *l1_l2* seems to convergence faster but is more unstable. Thus, we chose as optimal method the *l2* one.

Test on different data

Finally, we wanted to study the performance of our model on data with different signal-to-noise ratio, meaning in modifying the amplitude of the sinusoid for the Positive and Negative Signal classes. We report our results in Figure 4. It is possible to notice a systematic decreasing of the accuracy as the amplitude decreases; the training starts to fail for amplitudes that are half the original one, while the accuracy tends to 1 quickly as we increase the original amplitude.

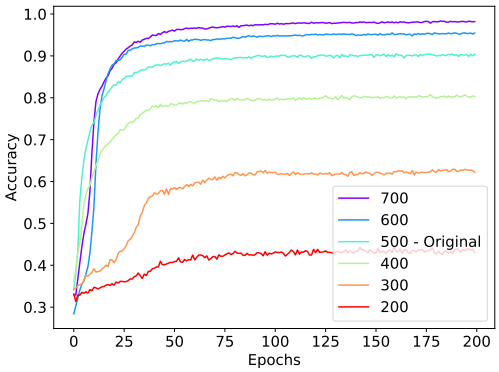

FIG. 4. Accuracy over the validation set, varying the amplitude of the signal.

CONCLUSIONS

This project highlighted some key points that are needed to be taken into account when facing particular tasks, which require the features extraction power of the Convolutional Neural Networks. We went through the most important layers needed to build this kind of architecture, as well as the main hyperparameters that are needed to be tuned in order to reach a better and faster convergence.

Moreover, we quickly discussed how to deal with a restriction on the total number of trainable parameters. This could be useful in real-world applications when deeper and wider Convolutional Neural Networks are needed to analyze more complex samples: lowering the number of trainable parameters improves the training speed and decreases the risk of overfitting. We showed as well three different regularization methods, useful to further decrease this risk.

[1] P. Metha et al., "A high-bias, low-variance introduction to Machine Learning for physicists", Physics reports **810**, 61-63 (2019)