Adrien SALLÉ - April 9, 2023

MLF-MPA Project

Introduction

The goal of this project is to develop a Machine Learning Model to classify wireless transmitters. In this report I will present the data I was given before explaining how I choosed to build my model. To conclude I will try to understand the results I obtained.

Contents

1 Processed data

1.1 Presentation

As explained before the goal of the project is to classify wireless transmitters. To do so, we were given two trainning datasets and one testing dataset. I decided to use only the dataset given as test (so I didn't separate the datasets into training and test data). For the classifications, we were given dataset with 10 features corresponding to the main radio frequency impairments. The given features for classification were :

- cfo_demod
- gain_imb
- or_of
- quadr_err
- m_power
- ph_err
- mag_err
- evm
- Tosc
- Tmix

1.2 Preprocessing

First, I removed the m_power, Tosc and Tmix features as they were not considered as relevent for the classification in the subject. I also removed gain_imb and quadr_err as they were not presenting much variations unlike other features.
I also added a scaling method, in order to improve my system efficiency, I used the *standardScale* object provided by the *sklearn* library. By standardizing the data I was able to increase the efficiency of my model.
Finally I also needed to encode the training data into *one-hot*. Indeed, when working with classes, it is needed to convert labels into integers, in order for the data to be understandable by the system. In our case, the categories need to be 1 to 6.

2 Model building

2.1 Layers

First, I defined my model, I decided to use a system of three layers, the first one with an input of 6, corresponding to the number of features used to perform the classification. This first layer, as for the second one, uses the *relu* function as the activation function, as it is advised in the keras documentation for this kind of classification. I started with layers of two neurons at first. The output layer uses the *softmax* activation function, which gives as an output a set of probability corresponding to each classes, which means I also had to set 8 neurons in this layer, one for each class.
Here is the plot of my final model :

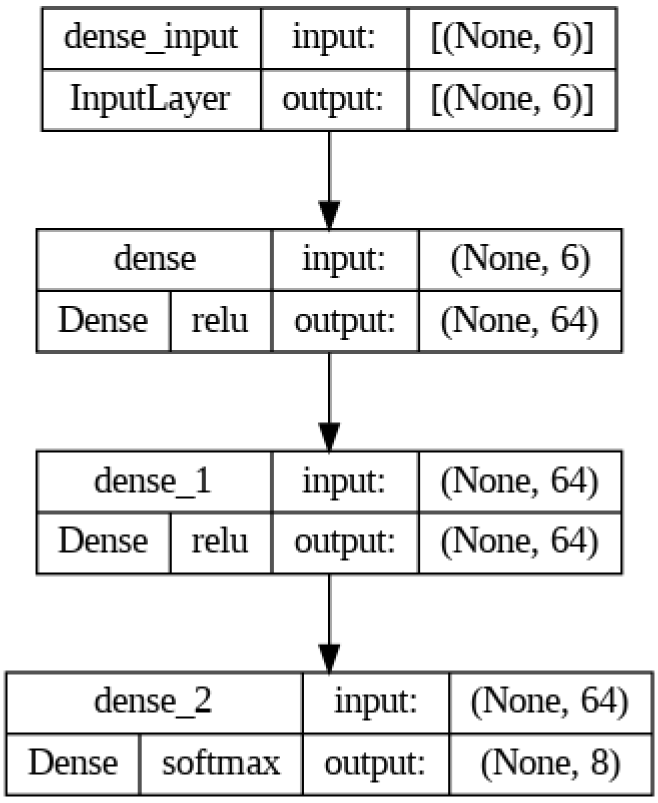

Figure 1: Plotted MLA model

2.2 Compiling the model

Next step is to compile the model. I chosed to use the *adam* optimizer, as it is the one with the less drawbacks according to the keras documentation. I started with a learning rate of 0.001 as it is a standard value. I also choosed the *categoricalcrossentropy* loss function, as it is the one used for this kind of classification.

2.3 Export to dataset

To obtain my set of predictions, I used the *model.predict* method. This method, when given a a fitting dataset, wil return a array corresponding for the prediction for each category given the input features. The next step is to use the *np.argmax* method, allowing to "compress" each prediction in a single integer corresponding to the category with the highest probability. Finally, I convert the array into a dataframe, to end up exporting it as a csv file to submit it on *kaggle*.

3 Results and tunning

3.1 Hyperparameters tunning

Once my model built, I started attempt to train it and test it. I choosed and modify several hyperparameters to increase the score obtained on kaggle. First I noticed that I obtained better results setting the neurons to 64 for my hidden layers. Here is a summary of the final system :

```
    Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                448

 dense_1 (Dense)             (None, 64)                4160

 dense_2 (Dense)             (None, 8)                 520

=================================================================
Total params: 5,128
Trainable params: 5,128
Non-trainable params: 0
_________________________________________________________________
```

I also set the learning rate of the optimizers to 0.025 and choosed to set the epoch number to 128 and the batch number to 120. (Theses aren't the hyperparameter with which I reach the highest score on *Kaggle*)

3.2 Results

After adding the scaling process to my system I gained a significant increase of score on *Kaggle*, however, my system started to behave weirdly as the acccuracy were decreasing through the epoch and the loss were negative. I didn't achieve to fix nor understand this problem exactly.

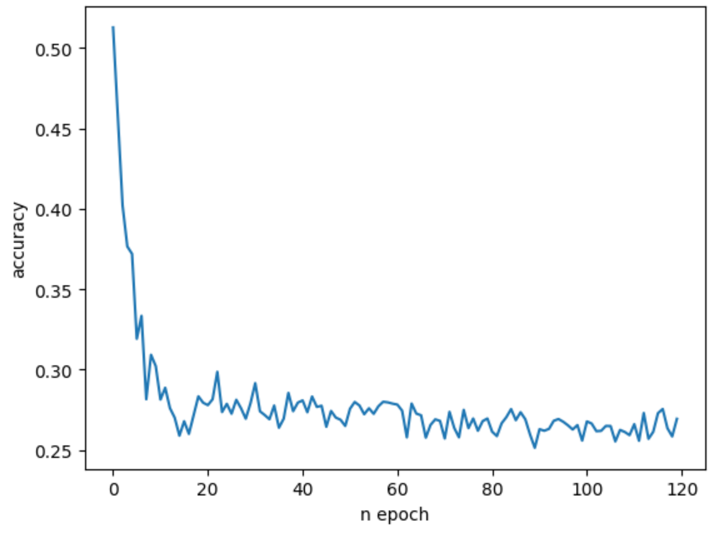

Figure 2: Accuracy in function of epoch

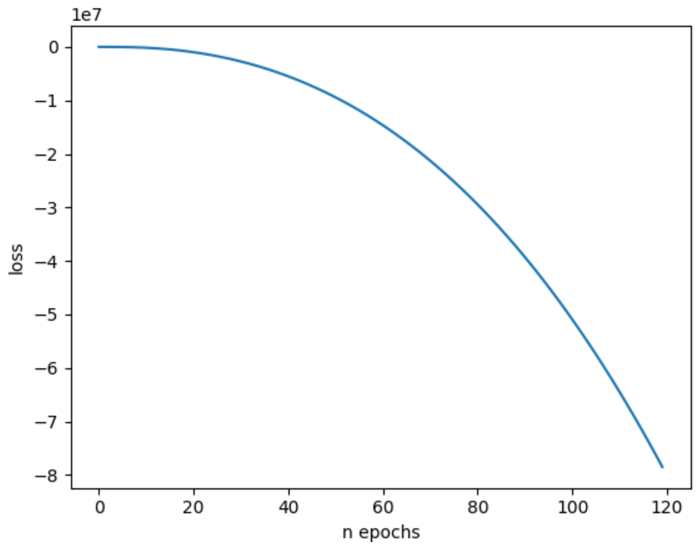

Figure 3: Loss in function of epoch

4 Conclusion

I haven't been able to reach more than an 43% score on kaggle by modifying the parameters. When looking at the evolution of the loss and the accuracy of my model through the epochs,it look like my model might be overfitting. Some modification could have been done in the model building by adding regularization or dropout layers. I might also have used some more preprocessing or tried others optimizer to improve my model.

List of Figures