Neural networks - Kaggle

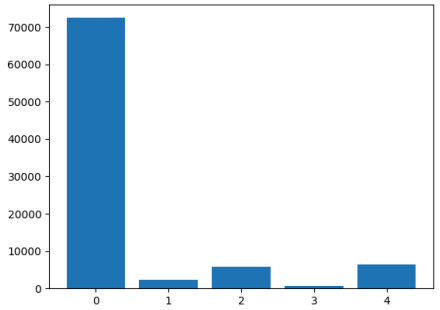
Olga Bonachera del Pozo - Daniel Kwapien - Alejandro Sánchez

1. Dataset

For this Kaggle competition we were given a dataset of almost 90.000 of time series sequences with a length of 187, although not all the instances were the same length, most of them were padded with 0's.

One of the issues is that the data had some bursts, so we needed a method to fill them. In order to address this problem we used panda's interpolation method. Specifically the spline method uses a SciPy wrapper in order to fill the NA's. There are also other options that we tried such as linear interpolation. But for us the spline interpolation worked the best

Then, the main problem is that the data had a huge imbalance within its classes. As we can see in the following figure, class 0 is 113 times more frequent than class 3. So we should definitely treat this issue.



We tried different approaches to solve it, but what best worked for us was to create a Weighted Random Sampler. This sampler works in such a way that it is given weights assigned to each class, calculated based on the frequencies of these classes, so when we sample to create the batches the sampler takes into account this weights and samples the instances according to it, this way we have balanced batches and the neural network sees each class for each sampler and we are able to compute the loss taking into account all the classes.

Finally, in order to create our batches and apply the sampler we create a Torch dataloader and create our training, validation and testing sets with a batch size of 256. Note that we only apply the sampler to the training set.

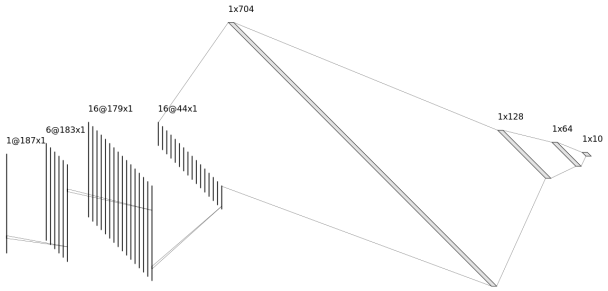
2. Training Process

Our first approach was to use a Recurrent Neural Network with LSTM cells, but this did not really perform well. We also tried implementing attention but it did not provide any major improvements. We were achieving around 85% accuracy and 68% macro F1-Score, far from our goal.

In the following iteration we came along this post, which introduced us to the idea of trying using a 1-dimension CNN or a hybrid approach using first a 1-dimensional CNN and then a RNN.

At first we went with the hybrid approach, trying with LSTM and GRU cells, but it did not really perform well, and the training time was huge compared with CNN. So we decided to go with 1-dimensional CNN completely.

After a few iterations, we found an architecture that worked well for us, it was composed of two convolutional layers, one pooling layer and three classification layers; we wanted to keep it simple in order to avoid overfitting. We also try with different drop out rates, between 0.6 and 0.8. We also introduced weight decay, equivalent to L2 regularization with a lambda between 0.001 and 0.0001.



.

Although we noticed the network was performing well, with a high accuracy of 95% in the test set and a macro F1-score of 85%, we noticed that it was not capturing the minority classes. It seems that distinguishing between class 3 and 0, and class 1 and 0 was quite difficult, surely because of the low number of training samples. So we decided to introduce a final resource, a Focal Loss[1], basically it consists in putting more weight on the hard examples so the network is able to focus more on them controlled by a parameter gamma.

So with this final approach, iterating between different architectures and parameters, we ended up having a classifier with 97% accuracy in the test set and a macro F1-Score of 89%. We think that there is surely much room for improvement trying to increase the accuracy in the minority classes.

3. Conclusions

This was a fun challenge. Creating our network and making it achieve a good enough performance taught us about the iterations and the process behind creating a network for a given task. As we said before, we think that there is room for improvement, but it would require more iterations and hyperparameter tuning. From what we have seen, it seems that the data does not really have strong time dependencies, but it does have patterns that the convolutional network is able to find, even though it is a difficult task with such an imbalanced dataset.

But after all, we are able to achieve a 75% accuracy on the class 3 and 80% accuracy in the class 1, which is good enough for that little number of samples.

---

[1]Lin et al., “Focal Loss for Dense Object Detection.”