Artificial Neural Networks and Deep Learning : Homework 1
Politecnico di Milano
fall '22

Galimberti Claudio, Mileto Alessandro, Staiano Enrico

1 Problem statement

We were asked to develop a Deep Learning model to classify time series. We were provided with a dataset made up of series belonging to 12 different classes. The problem turned out to be an imbalanced learning one as dataset composition is not uniform. Moreover, we observed data in a feature wise fashion and we found out that certain samples exhibit spikes in the magnitude of the values of some features (see dataVisualization). In practice, this is a symptom of irregularity in the character of the series and it may constitute an additional element of difficulty in this classification problem.

2 Our work

2.1 Preprocessing

We tried to scale the data using the class MinMax caler provided by the scikit-learn library. This turned out to be a suboptimal choice probably because of the spikes above mentioned, so we moved onto other strategies and, after trying out StandardScaler and RobustScaler (scikit-learn implementations), we settled to StandardScaler as this seemed to be a good enough choice.

2.2 Dealing with imbalance

2.2.1 Resampling

To deal with the unbalancing of the dataset, we tried to use different resampling strategies and also class weights in order to make the model pay more attention to the samples belonging to under represented classes. We found out that the latter choice did not help much in this particular instance of the problem, so we focused on resampling. We opted out for using the Imbalanced Learn Python library. After trying several variants of SMOTE resampling and of Random Over and Under Sampling we mainly stuck to plain SMOTE.

2.2.2 Gaussian Noise : regularization and online augmentation

After several trials, we introduced in our best model a GaussianNoise layer provided by the Tensorflow library. The benefits it provides are several, we used it mostly for regularization (and augmentation) purposes. We used a very small standard deviation in accordance to the scale of the features.

2.3 Window Size: interpolation

On suggestion of Prof. Lomurno, we did some experiments on the window size of the series samples. We tried to increase the length of the window to find out if a bigger window would bring better performance. To increase the length of the window, we used different interpolation strategies (linear, quadratic, cubic

and so on) and, in fact, interpolation in general yielded good results. We stuck to cubic interpolation moving the size from 36 to 117 and we observed a remarkable performance boost. However, we did not include interpolation in every experiment.

2.4 Different Network Architectures

2.4.1 The beginning: LSTM

Initially, we tried out Long Short Term Memory layers in order to exploit their memory capabilities. We built a very simple model stacking several LSTM layers together, starting from unidirectional and then moving to bidirectional ones, on top of a quasi SVM classifier. We thought that using such layers with a robust classifier was a good design choice to tackle the problem effectively but the findings quite disappointed us as we expected more. (see *LSTM_SVM*)

2.4.2 Hamletic doubt: LSTM or CNN?

Since previous experiments on LSTM (with different configurations and regularization strategies) did not go as expected we started wondering that maybe for this particular instance of time series classification problem we needed something different. So, we tried to compare a 1DCNN based architecture with the previously mentioned LSTM model. We found out that the convolutional model yielded from the very beginning better performance with respect to the LSTM counterpart and with way less parameters. As always, there is no "free lunch".(see *comparison_LSTM_CNN*) We also wondered whether or not putting together CNN and LSTM in the same architecture was beneficial to our scope, so we stacked those layers in different manners (CNN layers before, LSTM layers after and viceversa) but performance did not improve so we stopped doing experiments in this direction.

2.4.3 Peek-a-boo! CNN is the way: the rise of Staianet

As we found out, 1DCNN was interesting and worth exploring so we stacked three 1D convolutional layers followed by two fully connected ones and a softmax output. We named this model Staianet and, after submitting it, we got back a 72% accuracy over the remote test set in phase 1. This was encouraging and so we kept exploiting this architecture. (see *Staianet_CNN_72*).

2.4.4 I need something more: Residual Learning

Starting from Staianet, we tried to improve performance using residual learning implemented by means of skip connections. They, tipically, work on very deep networks but this clearly not the case as our architecture was pretty shallow. Anyways, we implemented them and the validation accuracy on the local set improved considerably. We submitted the model but we did not beat our vanilla Staianet model. Nevertheless, we kept skip connections into consideration for further experiments (see *skipconnections*).

2.4.5 Know what? Attention is (almost) all you need

At some point, we included into the Staianet model with Residual Learning a special layer provided by the Tensorflow library named AttentionLayer. We put it after the stack of convolutional layers which Staianet is made up of. Moreover, we introduced for the first time into the experiments cubic interpolation as described above. These changes helped us bring up performance with respect to vanilla Staianet as the new network achieved 72.5% of accuracy over the remote set in phase 1 (see *smote_interp_attention_Staianet*).

2.4.6 Come together : Ensemble

In order to improve the confidence at prediction time of our best model (Staianet + Skip Connections + Attention + Interpolation) we trained it twice with different initialization and dataset splits and then

we put the two models together, averaging their predictions. This trick helped us achieve nearly 74% of accuracy on the remote test set in phase 1, so we decided that model ensembling was a reasonable next step to take in order to improve what we already had.

2.4.7 The more the better : Model Averaging

We did several experiments with model ensembling, mainly focusing on model averaging. The first time, we put together 10 vanilla Staianet (very first architecture, neither residual learning and attention nor resampling) models trained on 10 different splits of the dataset and it reached 74% on remote set (see *model_averaging_Staianet*) during the final phase of the competition.

2.4.8 Endgame : Ubernet

In the end, we tried to build a more powerful model: we put together two stacks of convolutional layers with a skip connection and an attention layer each, ending with a batch normalization layer. They were followed by two fully connected layers and a softmax output. Moreover, dropout layers were used for regularization along with Gaussian Noise and ridge regression. We named this model Ubernet. It was very promising and so we tried to submit it during phase 1 but the accuracy over the test set was only slightly better with respect to the previous best result at the time. Nevertheless, we went back to this model in phase 2, and we took inspiration from our previous work. We basically did averaging over this model the same way we did for Staianet (see previous section) and we used SMOTE resampling to try to balance our dataset. We got a remarkable result as our model scored nearly 75% over phase 2 remote test set (see *avg_resampling_attention_noise*).

2.5 Other experiments

On suggestion of professor Lomurno, we checked the paper Attention Is All You Need and we thought of reusing the Transformers Architecture for classification. This very naive attempt ended up in a failure as the network turned out to be not suitable for sequential data classification (see *transformers*). However, exploring the Attention layer machinery was useful for subsequent experiments and for building our best model . We tried to tune the parameters of the Staianet model using the Keras Tuner toolchain. We used two search algorithms : Random search and Bayesian search with different numbers and types of hyperparameters to tune (regularization parameters, number of fully connected layers and so on). We did not manage to obtain good results out of the toolchain and, in fact, we ended up abandoning it (see *kerastunerbayesian* and *kerastunerrandom*). One last experiment series regarded data augmentation. We tried to use the tsaug Python library (https://tsaug.readthedocs.io/en/stable/) in order to augment our dataset using common transformations over time series such as convolution, time warping along with interpolation and noise addition as previously described. As far as this problem instance is concerned, we believe that only interpolation and noise addition make sense and are helpful as transformations. In fact, we tried out several combination of warping, dropout, time drift and other operations (see library docs) but performance did not go up as expected. It may be that those transformations are more useful in other use case scenarios (audio samples classification, for instance), but not in this one or, at least, not in the way we implemented them (see *tsaug* notebook).