REPORT Homework 2

PART 1 -- PREPROCESSING

>Dataset first inspection : Given the training dataset we have identified two “variables”: x_train and y_train. The first, x_train is composed of 2k matrices 36x6 each with its respective label in y_train.

Without information about the dataset, except from that we are making time series classification, we derived some hypothesis of our own:

a) 2k elements, 36 features captured at t = [0,..5];

b) 2k elements, 6 features captured at t = [0,..35];

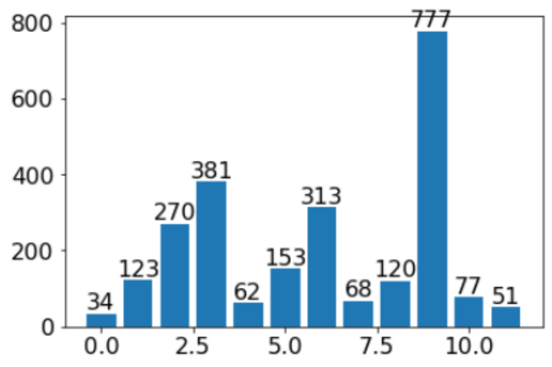
>Split into train and test set: We have decided to split the dataset into two sets, one used for training and one used for testing on our machines. The split happened setting the following parameters : 10% test size and stratified splitting on y_train, this last decision was made to ensure that we get the same percentage of target classes in both sets.

>Normalization: By looking at plots of a single subject and average plot we highlighted different dimensionality. Thus, we tried to standardize every matrix by column and by row via minmaxscaler by setting two different feature_range : [0,1] and [-1,1]. We obtained the best result by normalizing every matrix column-wise and in the feature range [-1,1].

>Balancing:

-balanced inspection:

After inspecting the dataset, we proved that it was imbalanced due to class 9, as shown in the figure below. So we tried several approaches to deal with this problem.



1. class weights: Our first attempt to assess this issue was to use class weights that train more on less present data. Results improved on original data for Vanilla lstm and Bidirectional lstm standard models (see part 2), while improved only on Vanilla lstm architecture when applying scaled/standardized data.

see notebook: ts_balance_inspection

2. GANs study : Our second attempt to deal with the imbalanced dataset was made using a generative adversarial network. GANs do not rely on prior assumptions about the data distribution and can generate data with high fidelity. Firstly, starting from the original dataset, we tried to generate new data using non conditional GAN. This affected the overall performance of the Vanilla lstm model(see part 2), which gave us worse results than before, (when we simply used the standardized dataset); so we tried to resort to conditional GAN, which applied to the same architecture gave us better results. This last choice was made given the advantages in faster convergence and the possibility to control the output of the generator at test time by giving it the label you want to generate.

PART 2 – RNN NETWORKS

>First attempts – Vanilla lstm, Bidirectional lstm and 1DCNN

As previously anticipated in part 1, in a first attempt of implementation, we used theVanilla lstm and Bidirectional lstm with standard parameters: the feature extractor layer with 128 units with a dropout layer (0.5), one dense ReLu classifier with 128 units and the output layer being Dense with a Softmax activation function. We decided to leave the current combination of hyperparameters as it was due to the simplicity of the problem (more complex networks were discouraged by the literature we found). As regards the 1DCNN, we built our feature extractor layer with a 1D convolution with 128 and ReLu, followed by a Maxpooling and another 1D convolution coupled with a GAP and Dropout layers.

We tried to feed the minmax standardized dataset to the above models and obtained fair results on all of the architectures, especially on the Bilstm.

In order to improve accuracy on other models, we tried to feed the original dataset directly into Vanilla lstm, Bidirectional lstm and 1DCNN, given the matching of the dimensionality structure with the input of lstm while applying class weights. Results were the same on both Vanilla and Bilstm but were significantly worse on the 1D-CNN.

Then we tried GANs, which gave an improvement only on Vanilla lstm. (see part 1)

>Further Extensions – gru and transformers

We also tried to feed directly the data structure into gru and transformers. We decided to implement gru because it is similar but newer than lstm since they address the vanishing gradient problem and are not only more efficient but also better on small datasets. [cite paper]

Then, we have also implemented transformers because of their attention mechanism that leaves out of the implementation any RNN; this makes them not only less complex but also computationally more efficient and therefore helpful when dealing with massive datasets but this is not always the case, so we decided to apply them here. (see: https://www.borealisai.com/publications/optimizing-deeper-transformers-small-datasets/ this paper says that a common believe is that transformers need big datasets to work properly but here they say that it can not be the case if properly tuned). Referring to the literature that we found we tried the transformer approach but it resulted in results that were neither so good nor better than the ones obtained through the previous approaches, thus we drew the conclusion that the network probably needed better fine tuning as cited in the above paper.

PART 3 – CNN NETWORKS

The use of convolutional neural networks to extract features and help in the classification of time series is well documented in literature. Given it is a valid alternative, we tried to use some CNN architectures to see if a convolutional approach gave us better results.
We tried to exploit a simple architecture made by us, and later we replicated some famous networks coming from image classification competitions, like DenseNet, Inception and ResNet.
The best result was achieved with a three-block ResNet network, reaching a 0.644 accuracy on the test set. The network was able to produce this result after we fed to it a balanced dataset augmented with the previously programmed GAN, setting all classes to a total of 200 samples, to limit the synthetical aspect of the virtual data generated. Tests with higher class numerosity yielded worse results.

Another step forward was obtained thanks to the addition of a bidirectional LSTM module at the end of the three-block ResNet, reaching a 0.68 accuracy on the test set. The decision to place the recurrent block at the end of the ResNet network was done due to some examples found in the literature (see: H. Choi, S. Ryu and H. Kim, "Short-Term Load Forecasting based on ResNet and LSTM," 2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm), 2018, pp. 1-6, doi: 10.1109/SmartGridComm.2018.8587554). Training data was still GAN-augmented at 200 samples per class.
Further augmentation with GAN to avoid undersampling for some classes was detrimental, as the training would not fit well, displaying strong underfitting behavior, thus probably displaying inability in capturing relevant patterns in the training data, because, we think, of the synthetic features introduced by the data generator, especially given the strong number of virtual samples we had to produce.

RESULTS

The data displayed a more encouraging reaction to a convolutional approach, which was still integrated with a recurrent module in our final model.
Data augmentation with GAN provided, in general, a good response in terms of balancing, as the initial tests delivered a strong difficulty from the models to discern any class which was not class number 9, the one whose numerosity was higher in the original data. After several tuning steps, we decided for a 200-per-class distribution, given that a bigger threshold resulted in underfitting.
Our model also displayed a lot of fluctuation, especially in the validation loss, which we were unable to reduce even by adjusting callbacks by tuning the patience and minimal learning rate parameters.
Further techniques, like ensemble, cross-validation and a better integration of recurrent modules in the ResNet architecture are all possible ways to try and improve our results.