Artificial Neural Networks and Deep Learning 2022

Challenge 2

Prof. Matteo Matteucci

Lorenzo Trevisan (Personal Code: 10687901)
Giuseppe Urso (Personal Code: 10628602)
Matteo Venturelli (Personal Code: 10629913)

POLITECNICO
MILANO 1863

1 Problem Definition

The goal of this challenge is solving a time series classification problem. Given a training set of 2429 samples, the team was supposed to predict the correct class label of the samples contained in an unknown test set. Each sample is a (36×6) vector which describes the temporal evolution of 6 features along 36 time instants.

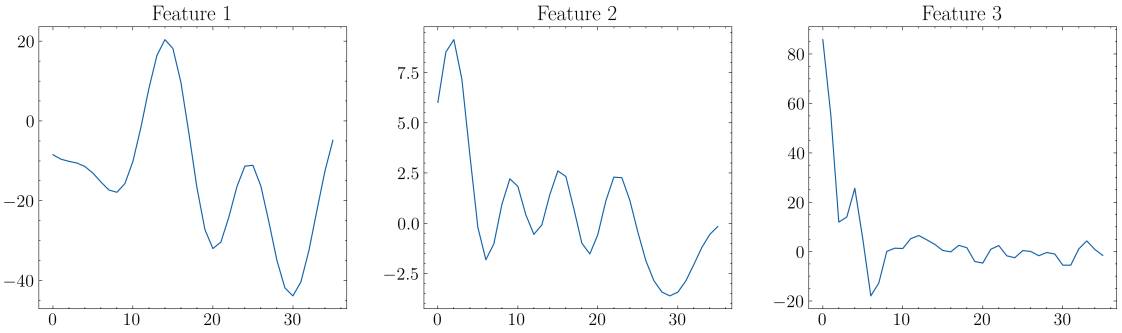

Figure 1: Example of some of the features that form the samples contained in the training set.

2 Proposed Solution

Time series classification is an important and challenging problem in data mining. Several deep learning architectures can be used to solve this task, and each one has its strengths and weaknesses. For this reason there's not a general model that can be applied to any scenarios; different architectures have to be tested in order to select the one that works better for the specific case study.

2.1 First Model

First of all, the 2429 samples dataset was split into training and validation subsets (with proportions 0.85 and 0.15 respectively), then the samples were organized in batches with `batch_size = 128`. In this first model no preprocessing was implemented. The actual model consisted in a basic LSTM network:

- input layer (`input_shape = (36, 6)`);
- first LSTM layer (`units = 128`, `return_sequences=True`);
- second LSTM layer (`units = 128`, `return_sequences=False`);
- dense layer (`units = 128`, `ReLU` activation function);
- output dense layer (`units = 12`, `softmax` activation function).

The output layer had a number of neurons equal to the number of possible classes and `softmax` activation function in order to be able to interpret the output values as probabilities.

We set `SparseCategoricalCrossentropy` loss function and `Adam` optimizer with learning rate equal to 10^{-3}. The number of epochs was set to 200 as maximum value, but in practice was ruled

by `EarlyStopping` callback with `patience = 25` on the validation accuracy. `ReduceLROnPlateau` callback with `patience = 10` on the validation accuracy was also implemented.

The training dataset presented high class imbalance. The least represented class contained only 34 samples, whereas the most represented one contained 777 samples. To avoid having the network perform much worse on some classes with respect to the others, we trained the model with `class_weight` argument, which allows to "pay more attention" to examples from an under-represented class, improving their accuracy.

The best results were obtained at epoch 25 and showed clear signs of overfitting: the training accuracy reached 0.9679 whereas the validation accuracy reached the much lower value of 0.6297.

2.2 Further Improvements

Differences in the scales across input variables may increase the difficulty of the problem being modeled: the model may suffer from poor performance during learning and sensitivity to input values resulting in higher generalization error. Input data inspection revealed that the mean of each feature was of the order of 10^2 and that the standard deviation of each feature was of the order of 10^3, which are not ideal for network training. To obtain a more suitable input dataset, the team decided to implement some form of scaling. Different approaches were tried out (both feature-wise and on the whole data):

- `MinMaxScaler()` was the first preprocessing method applied. Because of the high variance of the data, the team expected this type of scaling to shrink the majority of the data to a very shallow range of values, causing learning incapability. As a matter of fact, very bad results were achieved in terms of training and validation accuracy with this scaling;
- Input whitening (zero mean, unit variance) by means of `StandardScaler()` method (from scikitlearn library) showed minor improvements (approximately 1% on validation accuracy);
- `RobustScaler()` method (also from scikitlearn library) yielded the best performance since it improved validation accuracy by almost 3%. This scaler uses statistics that are robust to outliers: it removes the median and scales the data according to the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

In order to be able to apply the same scaling on the test data, the scaler parameters were imported into the submission model by means of `pickle` library.

The validation accuracy after training the model was still much lower than the training accuracy, so different combinations of dropout layers were tested in order to reduce overfitting. The best results were achieved with a single dropout layer with `rate = 0.5` introduced between the LSTM layers and the dense layers. This update increased the generalization capabilities of the model, and a test accuracy of 66% was achieved on the external test set.

Even though the `model_weight` attribute was set in order to reduce the effects of class imbalance, the team still noticed that the test accuracy of some classes were extremely lower than others. For this reason oversampling for the least represented classes was implemented. Different multiplication factors for each class were tested, but the learning did not improve: the network seemed to overfit the data related to the oversampled classes, increasing the training accuracy but reducing the validation accuracy at the same time.

With the goal of improving the generalization capabilities of the network, data augmentation was introduced. The augmented training dataset consisted in two copies of the original dataset, one augmented and one not, whereas the validation dataset was not modified. Transformations were carried out by means of `tsaug` library, which provides useful methods to augment time series

data. Multiple configurations of `Dropout`, `AddNoise` (zero mean, different values for variance) and `TimeWarp` were tested, but there was no improvement in terms of validation accuracy.

The last attempt to heavily modify the input dataset consisted in fragmenting the samples and removing features. The former was implemented by feeding the network with shorter time series with respect to the original ones (`window_size = 9` and `window_size = 18` were tested), whereas the latter was introduced to verify the significance of each feature. Neither of those approaches yielded better performance.

The final step was to test different architectures. The team tried out Bidirectional LSTM layers instead of regular LSTM ones, tuning the number of neurons and the other hyperparameters in order to improve performance. This model yielded slightly more than 1% higher accuracy on the local validation set. Since transformer models have shown state of the art performance in a number of time series forecasting and classification problems, the team decided to also test this type of model. Unfortunately no more than 67% on validation accuracy was obtained with transformers. Finally, one-dimensional convolutional neural networks have been tested. With this architecture the team obtained outstanding results with respect to the other models. As a matter of fact, using a basic 1D-CNN with almost no hyperparameter tuning the team obtained an accuracy of 0.7062 on the external test set. Further tweaks (discussed in detail in the next section) allowed to increase this value by roughly 2.5%.

2.3 Final Model

The input data was preprocessed by means of `RobustScaler()` acting on all data (not feature-wise). Data was not augmented nor oversampled. The data was organized in batches with `batch_size = 32`. The final model consisted in a 1D-CNN with the following architecture:

- input layer (`input_shape = (36, 6)`);
- 3 identical sequential blocks constituted by a 1D convolutional layer (128 filters with `kernel_size = 3`, `padding = 'same'` and `ReLU` activation function) and a dropout layer (`rate = 0.2`)
- global average pooling layer;
- first droput layer (`rate = 0.5`);
- first dense layer (`units = 512`, `ReLU` activation function);
- second dropout layer (`rate = 0.5`);
- second dense layer (`units = 64`, `ReLU` activation function);
- output dense layer (`units = 12`, `softmax` activation function).

`Adam` optimizer with learning rate equal to 10^{-3} and `SparseCategoricalCrossentropy` loss function were used. `EarlyStopping` and `ReduceLROnPlateau` callbacks were also used. `class_weight` parameter was set to reduce the effects of class imbalance. On the 143^{rd} epoch, `accuracy = 0.8422`, `val_accuracy = 0.7568` and `test_accuracy = 0.7301` (on the external test set during phase two) were achieved.