Artificial Neural Networks and Deep Learning 2022
Homework 2

Team: All Is Well
Fatma Hamila - 10913201, Kodai Takigawa - 10808008, Zheng Maria Yu - 10596129

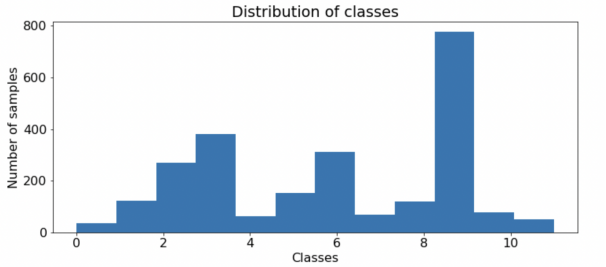
1. Introduction

The goal of this homework is to classify multivariate time series sequences. It is a supervised task given that all the sequences are labeled with the corresponding class. In the present report, we will describe how we approached this problem, the difficulties we faced and our observations.

2. Data preprocessing & splitting

The given data are in NumPy array format. The extracted dataset has a shape of 2429x36x6: there are 2429 time series sequences, each one of length 36 steps and 6 channels. The total number of classes is 12, which we proceeded to convert into one-hot encoders.

At the beginning we used the default sample shapes for training the models, and we divided the available data randomly into a training set (80%) and a validation set (20%) thus preserving the distribution of classes. Nevertheless, this approach did not give good results.



Some data analysis and plots are made on the dataset. As the figure shows, the dataset is highly imbalanced and some classes have very few samples. This posed a challenge we wanted to overcome all along the project by seeking different approaches.

The data scarcity issue led us to concatenate all the sequences belonging to the same class together, and to treat them as 12 long time series. The first 80% of the sequences was used for training, while the last 20% was kept apart for validation purposes. A sliding window is implemented to “cut” the sequences in order to obtain more samples: the window size was set at 36 in accordance with the original dimension, and several stride values within the range 1 - 24 were tested. In the end we chose to set the stride to 6.

Different attempts of data preprocessing were made, such as applying min-max normalization (brings each feature to a 0-1 value range), or importing the StandardScaler to standardize the distribution to mean 0 and variance 1, or even trying to normalize the data according to the global maximum (transformed into 1) and minimum (transformed into 0) among all features to preserve the differences in the variations of each feature. However, none of them brought significant improvements to our models.

3. Data augmentation

Due to the data scarcity and the dataset imbalance, we also used data augmentation methods to further increase the number of samples. The augmented samples were generated through the codes reported in the repository https://github.com/uchidalab/time_series_augmentation. The main methods we considered were jitter, scaling, permutation, magnitude_warp, time_warp, rotation, window_slice, and window_warp.

The transformations were first applied on the original values of the sequences directly, but the model seemed to perform worse. Since we noticed that those methods were designed to work on values in range [-1, 1], we scaled the data to [-1, 1] and brought it back on the original scale after applying the transformations. This time the operation went well, it successfully helped us limit overfitting and to improve the model's validation accuracy by about 5%.

4. Model structure

We used several models with different structures to solve the classification problem, and the main implementations are described below.

4.1. Vanilla LSTM & BiLSTM

The first model we built was the vanilla LSTM neural network seen during the exercise session. The feature extractor part consisted of two LSTM layers, while the classifier was composed of a Dense layer with the ReLU activation function and an output layer. We also implemented variants by changing units in LSTM layers and the number of LSTM layers. However, even stacking more LSTM layers brought a greater amount of parameters, we noticed that the models with 2 or 3 LSTM layers performed better.

We trained the proposed BiLSTM networks too. In this kind of models, the LSTM layers are enveloped with the Bidirectional wrapper to combine the forward and the backward outputs. Similar to the vanilla LSTM case, we changed the model structure to try to improve the results, but the default models with 2 or 3 BiLSTM layers still had better performance.

4.2. 1D Convolutional Neural Networks

We built this model by stacking Conv1D and BatchNormalization layers for feature extraction on scaled input. This type of model seemed to capture the patterns of the training test getting good accuracy scores, but it overfitted on the validation set and was stuck on the same scores even variating the number of layers, number of units, and the activations.

4.3. Encoder-decoder

An encoder-decoder structure was used for the feature extraction too. The encoder consisted in a LSTM layer, while the decoder was also a LSTM whose initial state was based on the encoder's hidden state output and the cell state.

4.4. Model with skip connections

One other approach was to build models with some skip connections between the layers. Our best performing model had the following structure:

- The feature extraction part was composed by An initial BiLSTM, followed by two stacks of two BiLSTM layers each, with their outputs concatenated.
- A GlobalMaxPooling1D layer.
- The classifier was composed of a Dense layer with the ReLU activation function, a Dropout layer and the output layer of 12 units with the Softmax activation function.

Unfortunately, the integration of an Attention layer in this model did not seem to improve its accuracy. The reason could be the relatively small window size of the samples.

5. Model Training

We kept adjusting the training settings and tuning the hyperparameters with respect to the monitoring indicators obtained in previous training sessions.

- Batch size: A batch size of 128 was chosen.
- Optimizer & learning rate: Generally we chose Adam as optimizer with a learning rate of 1e-4.
- Loss function: Categorical cross entropy was used during training. Later on, we decided to fine-tune the best models with categorical focal loss to compensate for the class imbalance. Categorical focal loss allowed us to get higher accuracies for the classes with few quantities of training samples, but it lowered the score of the other classes.
- Early Stopping: Once the validation accuracy got stuck, early stopping allowed us to interrupt the training to prevent overfitting.
- Scheduler: Cosine Annealing Scheduler was used in training some models. It determined periodical variations of the learning rate and it aimed to find better minima points.

6. Model Evaluation

In order to achieve better results, we selected some models to apply ensemble learning techniques, and we implemented the test time augmentation too.

6.1. Ensemble Learning

- Ensembling two different architectures: Some well-performing models with different architectures were ensembled: their predictions were summed up to output the class with the highest probability. In our case, this did not determine great variations in the results, maybe due to the fact that some classes were hard to learn for all the models.

- Ensembling models obtained from K-Fold cross validation: Splitting the dataset into 5 folds, we trained 5 different models based on the K-Fold cross validation technique. Their predictions were averaged later on. This time it gave us the best result too.

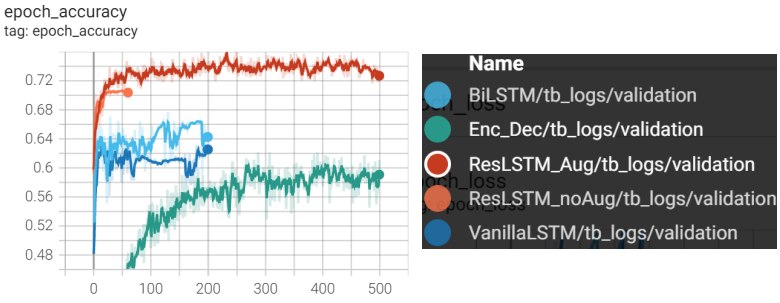
6.2. Test Time Augmentation

An attempt of test time augmentation was realized in the same way as the data augmentation technique described in the previous part, but it did not seem to work with those time series.

7. Results & Conclusion

7.1. Validation accuracy graph

Our main models' training tendency comparison is plotted here using Tensorboard.



7.2. Conclusion

By trying out different approaches to tackle the data scarcity and imbalance challenges, our models reached satisfactory testing accuracies in both phases on CodaLab platform: the best model achieved 0.7582 in Phase 1 and 0.7512 in Phase 2.

We would like to highlight the importance of the dataset quality in training deep learning models: in fact data augmentations greatly improved the models' performance in this case. In addition, we notice that the past steps' information is much relevant when dealing with time series, and suitable models should be considered for this kind of problem.