POLITECNICO
MILANO 1863

Artificial Neural Network and Deep Learning
Academic Year 2022 - 2023

Homework 2 Report

Team: KitKat

Christian Grasso 10652464
Filippo Lazzati 10629918
Matteo Nunziante 10670132

Professor: Matteo Matteucci
Tutor: Eugenio Lomurno

Contents

1 Introduction

In this project, we had to deal with a **time series classification problem**. The dataset is made of 2429 samples, and each of them is described by 6 features with 36 timestamps each. The total number of classes is 12 and they are very unbalanced since some classes have less than 100 samples, such as classes 0, 4, 7, 10 and 11.

2 Development

2.1 Training, Validation and Testing

The dataset has been divided into a *training set*, a *validation set*, and a *test set*, with respectively the 80%, the 10%, and the 10% of the total dataset. During the splitting of the dataset, we made sure to divide it keeping the distribution of the classes fixed. Even though the dataset is unbalanced, we did not use weights to give more importance to some classes in the loss function, because some of them (such as class 4) are very difficult to recognize, thus the overall performance was worse than when using unitary weights.

2.2 Normalization

We have tried several approaches for normalization of the dataset, mainly min-max normalization and Z-Score.

Min-max normalization simply consists in rescaling the original data to a specific range, such as $[0, 1]$ or $[-1, 1]$. With this method we obtained performance at best equal to the unscaled data, even after tweaking hyper-parameters, so we did not use it in the final models.

Z-Score normalization consists in making sure the mean of all values in the dataset is 0, and the standard deviation is 1. We tried applying both on a feature-by-feature basis and on the whole set. It was useful mainly when using the convolutional architecture (outlined below), while it did not bring significant improvements for the BiLSTM network.

2.3 Dataset augmentation

During the development, we tried different types of data augmentation:

- **Increasing the length of the sequences**: we tried adding new synthetic timesteps, for example by calculating the average between two consecutive timesteps or by repeating the same value multiple times.
- **Data generation**: we tried adding new samples in the dataset by applying some augmentations to a random sample such as noise (jittering), scaling, or random flipping.
- **Data generation**: we tried an approach based on SMOTE (Synthetic Minority Oversampling Technique), which consists in selecting 2 random samples and then creating a new sample by choosing a random position along the ”imaginary line” that connects the 2 samples in a 6-dimensional space.
- **Autoencoder**: for each of the 12 data classes, we trained an autoencoder with the original training data, and then used it to reconstruct the original samples. The reconstructed samples were then appended to the training

set for our network. We also tried using VAEs (Variational Autoencoders) and GANs to generate new samples from noise.

All the augmentations we tried did not significantly affect performance.

2.4 Features selection and new features

For what concerns **feature selection**, we tried to find a subset of the 6 features available that could bring better performances. Below is an overview of the algorithm we used:

1. Start from an empty list of features.
2. For each missing feature, try to add it and check the performance. Then select the most discriminative feature.
3. For each feature in the current optimal set, remove it and check the performance. If you get better performance without a feature, remove it.
4. Repeat from step 2 until all the features have been added or the performance did not improve in the last iteration.

The result was that with just 3 or 4 features (they depend on the architecture), we were able to reach almost the same result of using all of them, and the difference was less than 5%.

Also, we tried to **generate new features**. In particular, we used an approach based on PCA (Principal Component Analysis). In this way, we projected the original features in a new space, but, in the end, the model trained with the PCA features and the model trained on the original features were equally powerful, they made almost the same predictions.

2.5 Architectures

To solve the classification problem we used 2 different approaches: one based on Bidirectional LSTM and one based on Convolutional layers.

2.5.1 Bidirectional Long Short Term Memory

LSTM is a particular type of Recurrent NN, which means it bases the prediction on pattern recognition. We used a bidirectional LSTM because we have the complete sequence available from the beginning, so we analyzed it from the start to the end and vice-versa. This improved the results than just using LSTM layers.

The architecture we used was made of 2 bidirectional layers of 128 units each, a dense layer of 128 units, and the output layer with 12 units.

With this architecture, we also obtained some improvements by reducing the window size and combining results from different slices of the original samples. Given a window size $w < 36$, we used it as a sliding window over

the time dimension to obtain $s = 36 - w + 1$ slices of length w for each sample. We then trained s models and averaged their predictions. After tuning, the optimal size of the window resulted to be between around $w = 30$, thus we trained $s = 7$ different models. This approach led to an improvement of the test set accuracy of around 4%, for a total of 68%.

2.5.2 Convolutional Neural Network

The other approach is based on a classical CNN to classify the time series. A 1D convolutional layer is able to extract relevant features from a short series and, unlike a RNN, it does not look for patterns in the data, but only looks for features and whether they are present or not.

The architecture we used was made of 5 convolutional 1D layers with 128, 256, 1024, 256, and 128 units respectively. They are separated by max pooling and dropout layers. Then there is a dense layer of 32 units and, finally, the output layer with 12 units.

In order to get good performances, we had to apply Z-score normalization. The resulting network, trained on the normalized dataset, yielded an accuracy in the test set of 70%.

2.5.3 Transformers

Another approach we have tried is that of Transformers. The idea was that of using the attention mechanism to make the network focus on the relevant parts of the input series. Based on this Keras tutorial, we have implemented Transformers for the multivariate timeseries classification task. However, even though we have tried various implementations, changing the various hyperparameters like the number and the size of the heads of the multi-attention layers and the type and shape of networks (merely dense but also convolutional) of the encoders and decoders that made up the transformer, nothing has changed: we have not been able to overcome the 60% of accuracy in the test set. Therefore, we have avoided to insert this model in the ensemble.

2.6 Ensemble model

Given the 2 models described above (BiLSTM and CNN), we combined them by averaging the prediction probabilities and, in such a way, we increased the overall performances up to 72% in the test set.

3 Conclusion

During the development of the project, we spent a lot of time trying to apply augmentation to the data or to create new consistent samples. We think that, as in almost all the deep learning tasks, augmentation is the key to reaching high performances, but given that some classes are very similar to each other and given the unbalance of the dataset, it resulted to be very difficult.

Despite this, we have been able to test our knowledge and we trained 2 different types of models that work quite well, reaching an accuracy of 72% on the test set combining them.