Artificial Neural Networks and Deep Learning - Homework 2

M. Galliani, A. Preti, A. Solinas - Bongcloud Team

December 23, 2022

1 Introduction

The objective of this homework is time series classification. In particular, we have data about 2429 samples composed of 36 timestamps and 6 features, divided into 12 different classes.

We proceded to split the data in training, that consists in 80% of the samples and validation, composed of the remaining 20% of them. We opted for this split in order to have more data in the underrepresented classes. Furthermore we tried different rates of split but we eventually stuck to the original decision due to a lack of significant change in performance.

2 Preprocessing and Data Augmentation

Concerning our preprocessing choices, we decided to scale feature-by-feature instead of timestep-by-timestep since we supposed that features contained related information. We decided to use RobustScaler from the 'sklearn preprocessing' module because we could see the presence of extreme values in some of our samples, as can be seen in Figure 2

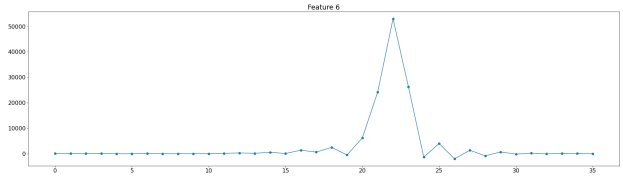

Figure 1: Feature 6 of a time series with extreme values

This scaler removes the median and scales the data according to the quantile range (usually the 1st quartile and the 3rd quartile) so that is less influenced by the presence of outliers with respect to common scalers (as min-max or standardization).

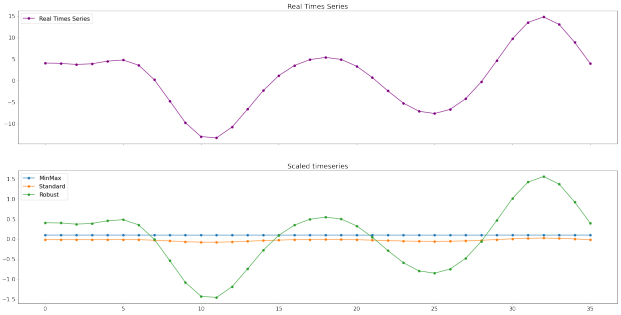

Figure 2: Examples of different scaling methods. As we can see, due to outliers, the values of the time series scaled with Standard and MinMax scalers are very small.

In addition to scaling, we resorted to data augmentation by sampling a random fraction of our data and feeding them to an augmenter using the 'tsaug' module. The augmentation operations which led us to higher performance were: adding noise to the series, dropping a random fraction of timestamps, reversing the series with a chosen probability, convolving with a kernel window and reducing the temporal resolution without changing the length. Using tsaug we created 1943 new data points that added to the original training set gave us 3886 samples.

3 Class Imbalance

From a preliminary analysis we could see a strong class imbalance, having the class 'Sorrow' with 777 samples (31% of the data) as the most represented, and the class 'Wish' with only 34 samples the most underrepresented. A visual representation of class imbalance can be seen in Figure 3

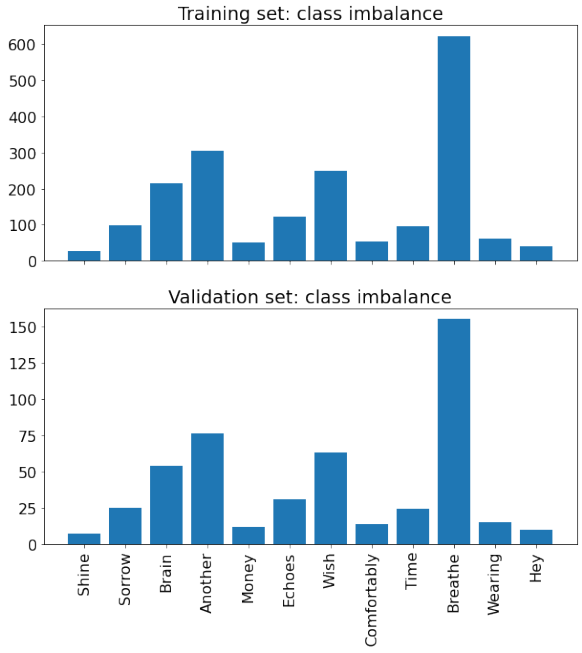

Figure 3: We split the data to have the same distributions between training set and validation set

We experimented with different ways to deal with class imbalance, like computing weights for each class and penalizing errors in prediction of underrepresented classes and applying oversampling to get a more balanced training set. These techniques lead to more balanced predictions between classe but they could not improve our overall performances **??**. For this reason, we resorted not using any method to deal with class imbalance.

4 Models

We tuned four models of increasing complexity to deal with the times series classification problem.

4.1 Conv1D

We started from the simple convolutional network seen in class and modified it by changing the architecture and tuning the hyperparameters with the Keras tuner. In particular, for each convolutional block, we stacked three one-dimensional convolutional layers (similarly to VGG architectures) with kernel sizes equal to 3. At the end of each block, we put a max pooling layer to reduce dimensions. To top our architecture, we added a global average pooling layer followed by two dense layers to classify the time series. We relied only on L2-regularization to reduce overfitting, as we tested batch normalization layers that gave low performances. Improvements may be achieved with dropout layers, which we widely use in the following architectures.

4.2 Conv1DResNetStyle

We adapted an implementation of a Resnet-style one-dimensional CNN [1]. The network had approximately 500k parameters and consisted of three main blocks composed of several 1D-convolutions, batch normalization layers, activation and a connection to skip each block respectively. After the blocks, a GAP layer was applied, followed by a dense output layer.

The main adjustment we performed was to replace the batch normalization layers used for regularization with dropout layers, as we learned from the simple conv1D architecture previously shown. In addition, we added a classifier dense layer in-between the GAP and the outputs and experimented with various hyperparameters configurations through the Keras tuner. In the end, we settled on a 0.3 dropout rate for the convolutional blocks, 0.5 for the classifier ones and used L2-penalization on each layer for some extra regularization.

4.3 Transformer

Another trial was made starting from an implementation of a transformer model found on the Keras website[2]. However, we had to modify the feature extractor architecture to increase performance. In particular, in the transformer encoder, after the multiheaded attention layer, we stacked three convolutional layers with 64 filters 3x3. To have coherence between different branches in the output shapes, we had also to add a convolutional layer with 64 filters in the other branch of the encoder. In the final architecture, we stacked 3 transformer blocks (the Keras tuner found the number of transformer blocks to be stacked). After the transformer blocks we added two more convolutional layers, each followed by a max pooling layer to reduce the dimension of the time series. At the top of this network we put a global average pooling layer followed by two dense layers.

[1] https://github.com/hfawaz/dl-4-tsc/blob/master/classifiers/resnet.py

[2] https://keras.io/examples/timeseries/timeseries_classification_transformer/

To properly set the hyperparameters introduced by the multiheaded attention layers, we based ourselves on the paper "Attention Is All You Need"[3] and we found that while the number of heads is arbitrary, the head size has to satisfy the following:

$$\text{head size} = \frac{\text{time series dimension}}{\text{number of heads}}$$

We used both L2 regularization and dropout in-between each layer to avoid overfitting.

4.4 SNN

To further explore new architectures, we implemented a so-called 'siamese neural network', a class of neural network architecture containing two or more identical subnetworks, which should help with the class imbalance issue. Therefore, we constructed a network with 2,3 or 4 identical blocks composed of two bidirectional LSTM layers and two convolutional layers. We then concatenated these blocks and fed them to a final dense layer. With this type of architecture we could achieve 73.5% of validation accuracy and 71.1% on the development test set. The peculiarity of these networks was that we could achieve more 'balanced' predictions and not 'simply' predict more the overrepresented classes. On the other hand, we could not improve the accuracy as much as we expected.

5 Ensemble methods

After tuning all the possible models cited above, we tried ensembling to aim for a performance increase. We took the best three models from the different architectures: Conv1D, ResNet style Conv1D and the transformer. The ensemble with those three networks led to a remarkable increase in performance on the test set. Our score went from 0.718 on the development test set to 0.7451 on the final test set.

6 Conclusions

We were able to significantly improve performances compared to the starting models. We believe that the most significant steps are:

- Preprocessing our data relying on RobustScaler
- Implementing data augmentation to expand our training set
- Avoiding batch normalization layers in favor of dropout layers and L2-regularization to reduce overfitting
- Building deeper convolutional blocks in order to increse the models' complexity
- Ensembling different models to maximize performances

Coming to possible future developments, we think there could be significant room for improvement in models like LSTM and SNN that we could not tune due to time issues. Concatenating them to our best three models is also worth exploring. Moreover, a possible development direction could be expanding the data augmentation methods since there is evidence that this positively affects models' performances. Another possible improvement could be experimenting with other techniques to deal with class imbalance available in the imbalanced learn library. For instance, a combination of oversampling and undersampling, such as SMOTEENN[4] may yield good results.

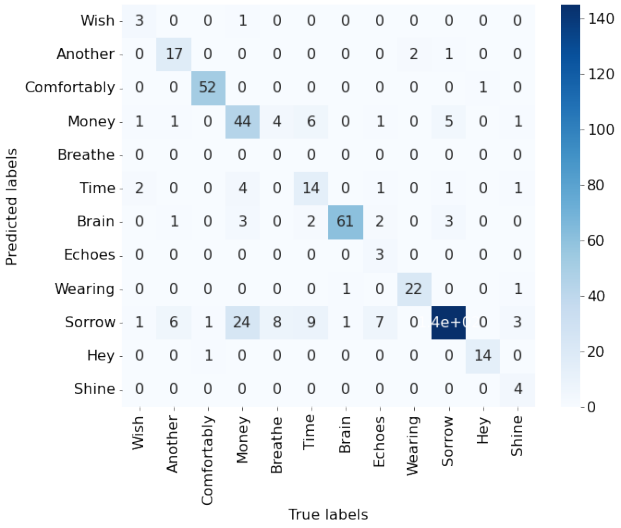

Figure 4: Performances of our best model on validation set (Accuracy: 0.7798). We can see that predictions are unbalanced, as we decided not to deal with class imbalance3.

[3] https://arxiv.org/pdf/1706.03762.pdf

[4] https://imbalanced-learn.org/stable/references/generated/imblearn.combine.SMOTEENN.html