DEEP LEARNING 2022

SPEECH COMMANDS CLASSIFICATION WITH RECURRENT NEURAL NETWORKS PROJECT NO. 2 REPORT

May 9, 2022

Elżbieta Jowik (298821), Agata Makarewicz (298827)

1 Introduction

The objective of this document is to describe the methodology and summarise the results of the applied approach to the problem of speech commands classification with recurrent neural networks. The report consists of two main parts. The first one is focused on giving a theoretical background of the project and adopted assumptions most of which concern the experiments' setup. The other one concerns the practical application of the networks under consideration and presents the performance of custom models. While the full code is included in the attached files, this paper points out only the most important facts like an overview of the design, performance of different models and interpretations of the results.

2 Data

A dataset used for this project is *Speech Commands* [1] dataset, which is a set of 64,727 one-second `.wav` audio files, each containing a single spoken English word. These words are from a small set of commands and are spoken by a variety of different speakers. The audio files are organized into folders based on the word they contain. The task realized in this project belongs to the class of multi-classification problems - the cardinality of the set of unique values of the target variable is equal to 11 or 12, depending on the adopted approach (described in detail in the following paragraph).

The labels we will need to predict are `yes`, `no`, `up`, `down`, `left`, `right`, `on`, `off`, `stop`, `go`. There are other types of commands available in the datasets, however, everything else should be considered either `unknown` or `silence`. Among these other commands, there is a collection of realistic or artificially generated sounds provided within the dataset - the `_background_noise_` folder contains a set of longer audio clips that are either recordings or mathematical simulations of background noise. We adopted two different approaches regarding this part of the data. The first one included treating them as an individual class - `silence`. The second one was randomly adding these recordings to the training dataset in order to improve the learning process and help the networks cope with noisy environments.

As for the data preprocessing itself, we found it inevitable to perform target variable categorical encoding. After renaming the classes according to the selected approach, the desired dummy form of the target variable was obtained by applying the `One-Hot Encoding` method. Additionally, we ensured that all the recordings are exactly one second long - the identified shorter or longer recordings were cropped or artificially prolonged (padded). We also divided the records containing noise into sets of one-second-long recordings to match the length of the rest of the data. In order to perform modelling, we converted provided audio files into processable data - spectrograms - using the `scipy.signal` Python package for audio analysis and information retrieval.

3 Architectures

In this project, we proposed and compared three different architectures of the neural network model. The difference between them was the type of the recurrent layer used. Three widely-used algorithms were used - SimpleRNN, LSTM and GRU (all from `tensorflow` Python package). Additionally, as a first layer, the convolutional layer was used. Since the `ReLu` activation function is not associated with the vanishing gradient problem, it was applied in each layer of the network (except for the last one, in which softmax was used). To reduce the risk of an overfitting problem

[1] Warden P. Speech Commands: A public dataset for single-word speech recognition, 2017. Available from `http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz` .

occurrence we added dropouts whereas to reduce the number of epochs (relevant for estimating the most accurate parameters) batch normalization and early stopping were applied.

Model 1

- Conv1D
- BatchNormalization
- Activation
- Dropout
- SimpleRNN
- Dense
- Dropout
- Dense

Model 2

- Conv1D
- BatchNormalization
- Activation
- Dropout
- LSTM
- Dense
- Dropout
- Dense

Model 3

- Conv1D
- BatchNormalization
- Activation
- Dropout
- GRU
- Dense
- Dropout
- Dense

4 Experiments & evaluation

The first stage of the experimental part of the project aimed at investigating the influence of parameter change on the obtained results and in the end finding the optimal configuration for each model. Based on the results of this part of the project we determined the optimal parameterization of further experiments for each of the models under consideration. The grids of hyperparameters considered like `batch size` or `learning rate` were determined empirically through conducted experiments and based on available existing sources. The evaluation measures against which the performance of the models was assessed and compared were classification accuracy scores and metrics resistant to the problem of class imbalance, such as confusion matrices heatmaps and visualizations illustrating the course of the loss function in the learning process. The core objective of all conducted experiments was to investigate the real model behaviour, regardless of any randomness. For this purpose, we fit each of the models, 10 times, and evaluate it on the test data set. Based on that evaluations we measure simple statistics of the model's results - minimum, maximum, mean and standard deviation value of the accuracy metrics. We also investigate the mean number of epochs it took for the model to learn because due to the usage of early stopping, the number of iterations might differ in each run of the model.

4.1 Hyperparameters tuning

The initial stage of the experimental part of the project involved searching a grid of selected hyperparameters, i.e. `batch size` and `learning rate`. As for the experiment set-up, it was estimated that the learning process will require up to 50 epochs, so the maximum number of epochs in the learning process was set according to this assumption. To reduce the number of unnecessary steps (if needed) we applied early stopping. It's `patience` was set to 10% of the total, assumed number of epochs while the `min_delta` parameter value was set to $1e-3$. Therefore the fitting should be stopped if a monitored metric i.e. accuracy has not improved by 0.001 for 5 epochs on the test set. As an optimizer, the ADAM method was used which, according to the literature, is currently perceived as the most powerful one.
The accuracy of predictions of models for individual configurations of hyperparameters has been aggregated (averaged over the iterations) and presented on the heatmap1. Due to slight differences in performances, for some settings, an additional table with more detailed results 1 was prepared.

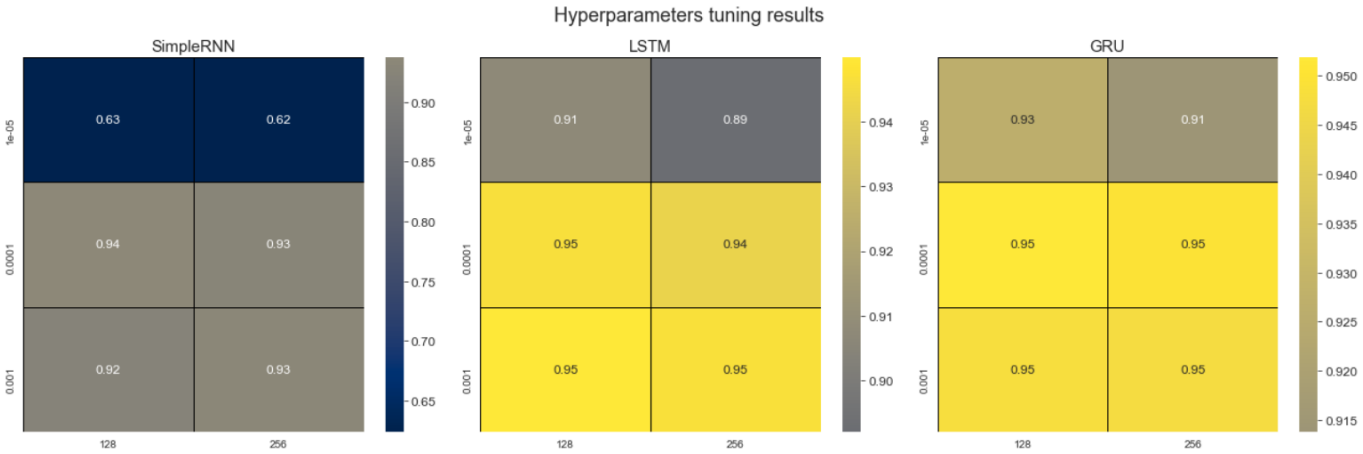

Figure 1: Models performance evaluation (accuracy) for individual combinations of `learning rate` and `batch size` parameters

SimpleRNN			
batch size\learning rate	**0.00001**	**0.0001**	**0.001**
128	0.625165	0.937235	0.921873
256	0.624287	0.926262	0.932114
LSTM			
batch size\learning rate	**0.00001**	**0.0001**	**0.001**
128	0.907388	0.946452	0.949817
256	0.892173	0.94177	0.946745
GRU			
batch size\learning rate	**0.00001**	**0.0001**	**0.001**
128	0.926554	0.951865	0.947769
256	0.913826	0.949525	0.947184

Table 1: Models performance evaluation (accuracy) for individual combinations of learning rate and batch size parameters

Based on the results, it could be concluded that the learning rate parameter has a significantly greater impact on the learning process than the batch size. Therefore, in addition to analyzing the results per pair of parameters, we decided to average the results over batch size parameter value and analyze the variability of accuracy concerning the change of the learning rate parameter. The results of the analysis are presented in the scatter plot2.
The scatterplot of accuracy values shown in Figure 2 confirms that 2 out of 3 models (SimpleRNN & GRU) achieve their best results for the `learning rate = 1e-4` whereas the third (LSTM) model performs slightly better for the `learning rate = 1e-3`. However, for each model the differences in accuracy observed for `learning rate` equal to `1e-3` and `1e-4` are so small that they might be perceived as unrelatable in the long run.

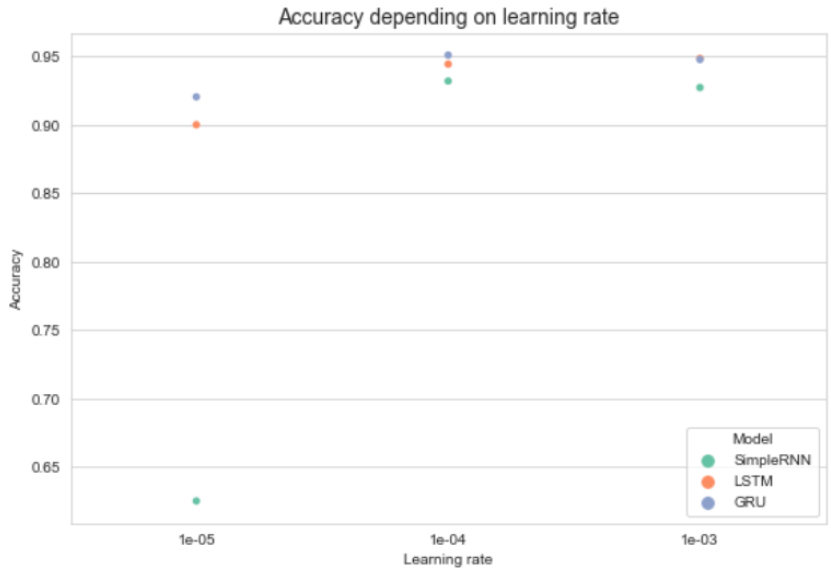

Figure 2: Accuracy of each model (basic version) measured for different values of the learning rate parameter.

Nevertheless, based on the conducted experiments, it was indicated that the optimal pairs of parameters for each of the models were, respectively:

- SimpleRNN: `learning_rate = 1e-4`, `batch_size = 128`
- LSTM: `learning_rate = 1e-3`, `batch_size = 128`
- GRU: `learning_rate = 1e-4`, `batch_size = 128`

After selecting the optimal configurations of the hyperparameters, the learning process of all models was performed again. As a result, it was possible to examine the convergence of the learning process and assess the predictive power of each of the algorithms.

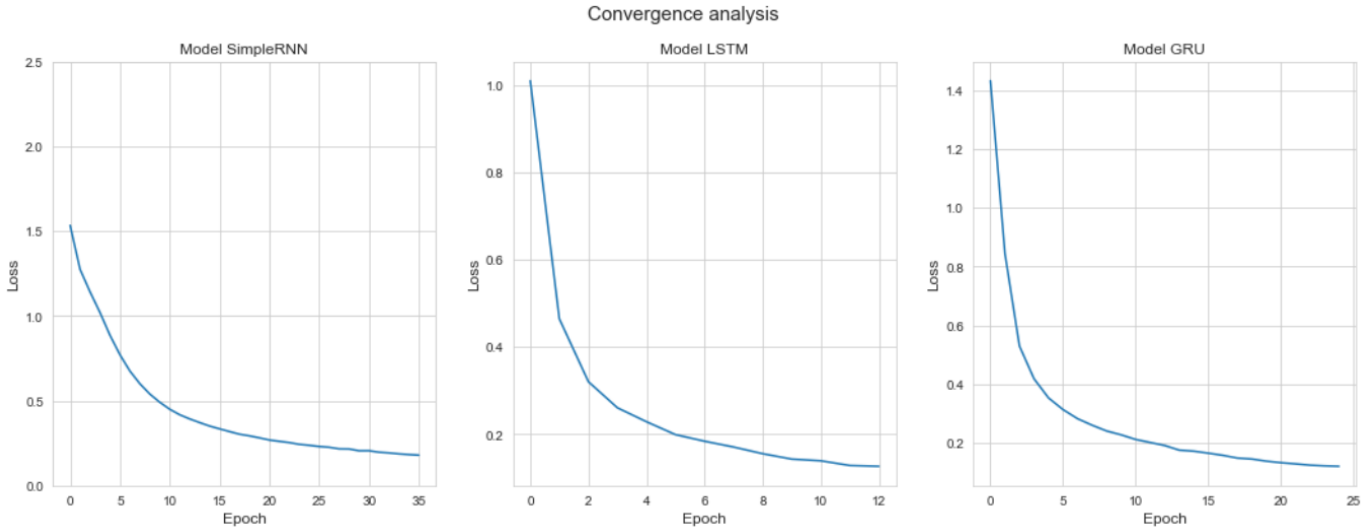

Figure 3: Learning curves of each model (basic version) measured on the train set, for different values of the learning rate parameter.

We can observe that the LSTM model has the fastest convergence rate. Plots differ for the X-axis range since we use early stopping to cease the learning before the model starts to overfit, so the

speed of the fitting is directly associated with the length of the process.

The last step in the first stage of the experimental part was the analysis of the confusion matrices for all models (Figure 4), which aimed to verify how well each of the classes is recognized. In general, models perform very well regardless of the chosen architecture and there are no classes that can be unambiguously indicated as extremely problematic

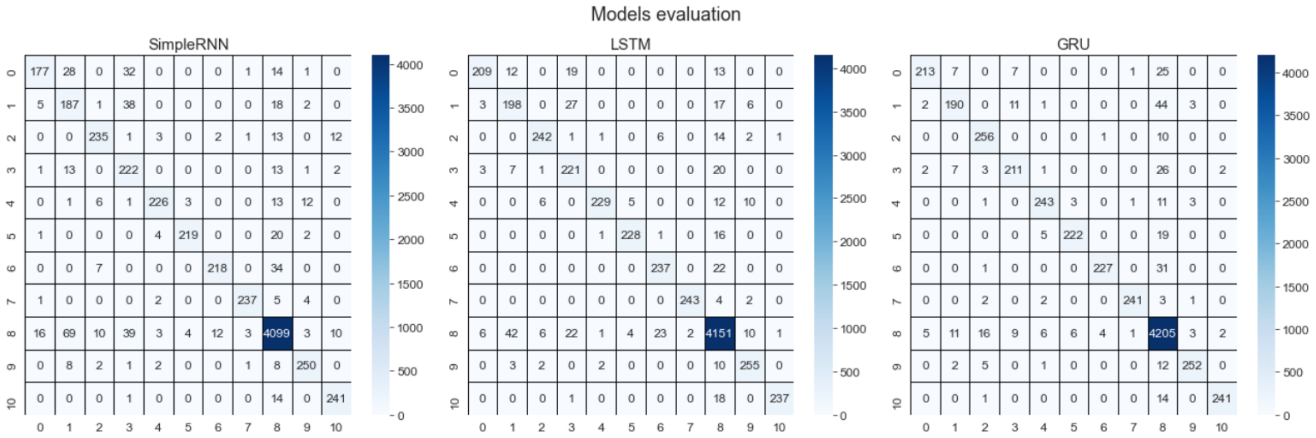

Figure 4: Confusion matrices of all models trained for optimal parameters configurations

4.2 Noise data handling

According to the project requirements we were to pay special attention on `silence` and `unknown` classes. We decided to implement two different concepts. One of them was taking `background_noise` observations as separate silence class, while the other one consisted in using these observations to noise the training data. In the second approach, the noise was added in the amount of 10% of the total set count to all target classes with probabilities proportional to their cardinalities4. The detailed results obtained as an outcome of individual experiments are summarized in the Tables 2, 3 in the following subsections.

Models performance statistics			
model\metric	**mean**	**std**	**epochs**
SimpleRNN	0.921798	0.009139	31.0
LSTM	0.949898	0.003697	21.0
GRU	0.952004	0.001540	30.5

Table 2: Models performance summary for the first approach of noisy data handling i.e. taking background noise observations as separate `silence` class

Models performance statistics			
model\metric	**mean**	**std**	**epochs**
SimpleRNN	0.937527	0.004345	46.5
LSTM	0.951426	0.003517	22.5
GRU	0.951280	0.002483	27.5

Table 3: Models performance summary for the second approach of noisy data handling i.e. noising the training subsets with the background noise class observations

In general, the predictive power of the models for both approaches is similar. However, adding noise to training data for 2 out of 3 models, led to an increase in the number of epochs of learning processes.

5 Appendix

Class	Observations Count	Noise Count	Noise Ratio
one	1892	204	0.107822
stop	1885	185	0.098143
seven	1875	173	0.092267
nine	1875	170	0.090667
two	1873	167	0.089162
zero	1866	166	0.088960
on	1864	183	0.098176
six	1863	181	0.097155
go	1861	163	0.087587
yes	1860	191	0.102688
no	1853	177	0.095521
eight	1852	180	0.097192
right	1852	201	0.108531
five	1844	183	0.099241
up	1843	175	0.094954
down	1842	189	0.102606
three	1841	173	0.093971
left	1839	174	0.094617
four	1839	213	0.115824
off	1839	207	0.112561
house	1427	127	0.088998
marvin	1424	149	0.104635
wow	1414	138	0.097595
bird	1411	154	0.109142
cat	1399	150	0.107219
dog	1396	140	0.100287
tree	1374	158	0.114993
happy	1373	133	0.096868
sheila	1372	153	0.111516
bed	1340	151	0.112687

Table 4: The number and percentage of noisy observations in individual target classes concerning the number of these classes.