Speech commands classification with recurrent neural networks

Deep Learning

Szymon Hryniewski and Mateusz Kierznowski

May 25, 2022

Contents

1 Introduction

1.1 Theoretical introduction

A recurrent neural network (RNN) is a type of artificial neural network, most often used for sequential data. RNNs are commonly used for ordinal or temporal problems, such as language translation, natural language processing (NLP) or speech recognition; they are incorporated into popular applications such as Siri and Google Translate. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depends on the prior elements within the sequence. For detailed explanation of RNN we reccomend reading [1] or [3].

Recurrent Neural Networks suffer from short-term memory. While sequence of inputs is long enough it can lose historic information due to influence of previous observation. During back propagation, RNN suffer from the vanishing gradient problem. For this reason LSTM block has been created. It provides a solution by providing gates which develop Long Memory. Structure of a LSTM block is shown on the following figure.

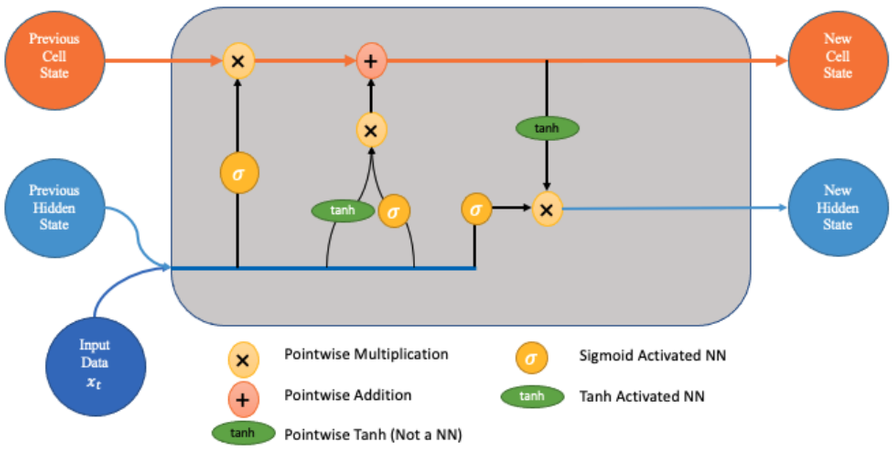

Figure 1: schema of an LSTM block
source: https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9

1.2 Data set and task description

In this project we test and compare performance of different network architectures in speech commands classification problem. Speech Commands data set, on which the experiments are performed on, consists of 64,727 audio files in the training set and 158538 files in test set. In the training data set recordings of the following words can be found:

- yes,
- no,
- up,
- down,
- left,
- right,
- on,
- off,
- stop,
- go,
- zero,
- one,
- two,
- three,
- four,
- five,
- six,
- seven,
- eight,
- nine,
- bed,
- bird
- cat,
- dog,
- happy,
- house,
- Marvin,
- Sheila,
- tree,
- wow.

Each recording of these words is approximately 1 second long. Whats more in the folder _background_noise_ 2 longer recordings are included. These recordings can be broken up and used as examples of silence class. All recordings in training data set are labeled, recordings in test data set are unlabeled.

The model is trained to distinguish 12 classes:

- yes,
- no,
- up,
- down,
- left,
- right,
- on,
- off,
- stop,
- go,
- unknown,
- silence.

Unknown label should be assigned to commands that aren't one of first 10 labels or that aren't silence. Similarly silence should be assigned to files on which no words are spoken.

1.3 Feature extraction and preprocessing

In our solution we have decided not to use raw .wav files as an input for our models. Instead of that each file is extended to 2 seconds and Mel frequency cepstral coefficients are derived. Detailed explanation of mfcc's is provided in [5]. The coefficients are later stored in the separate file, allowing us to load them easily instead of calculating them every time we use them. The script used for this purpose can be found: Colab Notebook: export_features

2 Architectures

2.1 Single CNN

2.1.1 Overview

This approach comes strictly within data shape. MFCC transformation represent voice as a matrix instead of vector. Thus CNN provides great understanding of this type of data. Representation of MFCC transformation is shown on the following fig:

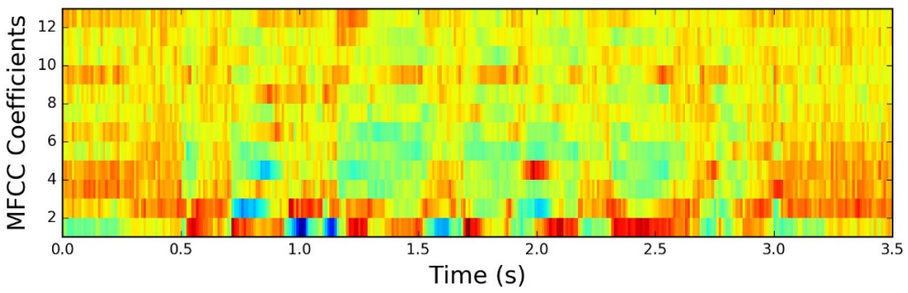

Figure 2: MFCC tranformation
source: https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd

Implemented architecture follows four 2d Convolutional layer and Dense Layers.

2.1.2 Experiments

We want to select the best hiperparameters for CNN architecture. To obtain them we used grid search method on following parameters:

- Kernel size
- Dropout
- Learning Rate
- No filters

	Accuracy	Learning Rate	Units	Dropout	Kernels
0	0.9531	0.001	512, 256, 128	0.2	5
1	0.9505	0.0001	512, 256, 128	0.2	5
2	0.9409	0.01	512, 256, 128	0.3	7
3	0.9365	0.0001	512, 256, 128	0.3	5
4	0.9361	0.001	512, 256, 128	0.2	3
5	0.9350	0.001	512, 256, 128	0.3	7
6	0.9321	0.01	128, 128, 128	0.2	7
7	0.9310	0.001	128, 128, 128	0.2	3
8	0.9165	0.001	128, 128, 128	0.4	3
9	0.8236	0.0001	128, 128, 128	0.4	7

Table 1: Parameters Search

2.1.3 Results

Final model can be found under link Colab CNN link
The best results are: 96.6% on validation dataset. Submission on kaggle returns 79.002%.
In addition we would like to present Confusion Matrix on train dataset which present as follows:

	down	go	left	no	off	on	right	silence	stop	unknown	up	yes
down	551	1	0	0	0	0	0	0	0	1	0	0
go	8	564	0	5	1	0	0	0	0	16	0	0
left	0	0	564	0	0	0	0	0	0	2	0	2
no	18	9	0	574	0	0	0	0	0	7	0	0
off	0	1	1	0	573	14	0	0	1	39	24	0
on	0	0	0	0	0	545	0	0	0	5	0	0
right	0	0	0	0	0	0	576	0	0	5	0	0
silence	0	0	0	0	0	0	0	100	0	0	0	0
stop	0	0	0	0	0	0	0	0	584	9	1	0
unknown	13	18	23	14	13	33	16	0	10	10173	11	15
up	0	0	0	0	2	0	0	0	0	3	558	0
yes	0	0	0	0	0	0	0	0	0	0	0	577

Table 2: confusion matrix

2.2 Double CNN

2.2.1 Overview

Additionally we want to create two different CNN architectures which divide main problem into two smaller subproblems.

1. Classify audio that are familiar to model. This subproblem separate classes into 2 groups: Classes like stop, go etc. which are our target and other words which label is set to be "unknown". Overall accuracy on this subproblem was 97.10% on validation data.
2. Second subproblem was to find correct label for words previously considered as a familiar word. It also received 96.76% accuracy on validation data.

However final results on kaggle is vastly lower than single CNN thus we have to abandom this approach. It's hence that binary classification has to be almost ideal to increase model accuracy.

2.3 LSTM

2.3.1 Experiments

We have decided to test model consisting of 3 LSTM layers, 2 dense layers and an output layer. To prevent over-fitting, dropout has been added after each layer, except for the output one. To test the influence of the hyper-parameters change on the models performance, we have decided to test all combinations of the following hyper-parameters:

- dropout value in LSTM layers (values $0, 0.1, 0.2, 0.3, 0.4$ were tested),
- learning rate of the model (values $10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$ were tested),
- number of units in LSTM layers and the shape. We have tested following structures:
  (a) $128 \to 128 \to 128$,
  (b) $256 \to 128 \to 64$,
  (c) $64 \to 64 \to 64$,
  (d) $512 \to 256 \to 128$.

Number of units in the first dense layer in each case has been set to be the same as the size of the output in the last LSTM layer.

2.3.2 Results

The plots of loss and accuracy plots, for the best performing model, for a given hyper-parameter can be found below.

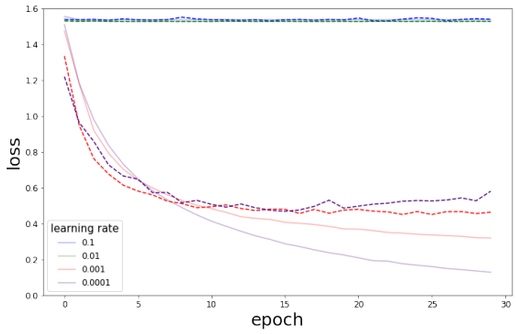

Figure 3: Loss value over time for various learning rates

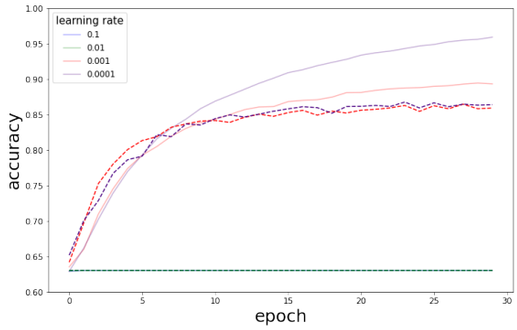

Figure 4: Accuracy over time for various learning rates

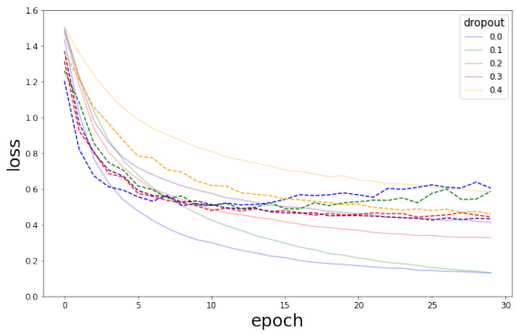

Figure 5: Loss value over time for various dropout values

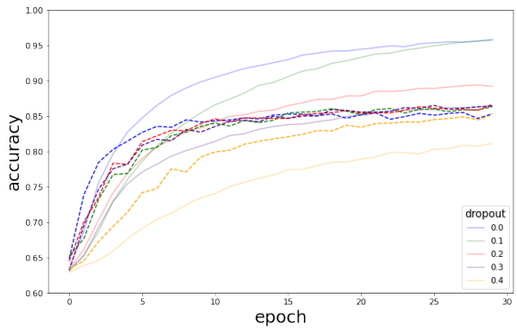

Figure 6: Accuracy over time for various dropout values

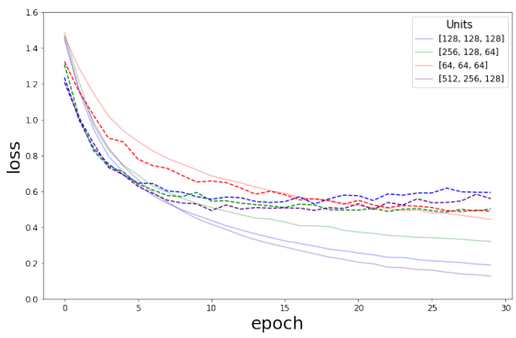

Figure 7: Loss value over time for various structures and sizes

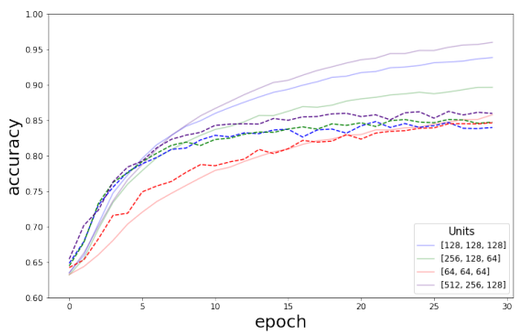

Figure 8: Accuracy over time for various structures and sizes

After studying them and the results of all tested models we have came up to the conclusion that a change in one hyper-parameter doesn't significantly change the models performance most of the time. However in general:

- the best learning rate parameter for this problem seem to be 10^{-4}. 10^{-3} gives only slightly worse results, remaining values are too big and model remains in a situation where all files are assigned to unknown class.
- as expected dropout value doesn't influence the accuracy of the model but convergence time and prevents over-fitting. For limited number of epochs, like it was in our case. Values between 0.1 and 0.3 seem to be the best compromise.
- experiments shown also that models structure doesn't influence the models performance that significantly. However structures with decreasing size tend to have a slight edge over the constant versions. Similarly bigger models tend to perform slightly better/faster.

The 3 best performing models achieve results ranging from 0.58771 to 0.59594 in Kaggle public score (a Colab notebook used to submit models is available here).

The confusion matrix of the best performing model on the 25% of the training set left out for validation purposes looks as follows:

	down	go	left	no	off	on	right	silence	stop	unknown	up	yes
down	351	26	2	15	1	3	0	1	2	185	3	1
go	25	309	2	95	0	1	0	1	0	156	3	1
left	1	1	453	8	4	0	16	1	1	91	0	12
no	11	72	1	386	0	0	1	1	3	115	4	0
off	0	1	16	0	409	4	0	1	4	121	32	1
on	1	0	1	1	13	378	0	1	2	191	4	0
right	0	0	19	0	1	0	437	1	0	130	3	1
silence	0	0	0	0	0	0	0	95	0	5	0	0
stop	6	7	1	1	2	2	0	2	491	75	8	0
unknown	63	60	60	43	38	72	102	13	63	9678	53	15
up	0	8	7	1	40	15	2	2	14	143	360	1
yes	1	0	29	6	3	0	2	1	0	57	0	495

Table 3: confusion matrix

From the matrix, we can see that unknown class is the most challenging one. Except for this the most commonly mistaken classes are "go" and "on" what after giving it some thought isn't very surprising. Colab notebook with all the experiments is available here.

3 Conclusions

Even though audio files could be treated as ordinary sequential data, it turns out that with mfcc transformation Convolutional Networks obtain a satisfactory results. What is more, CNNs performance can be even better than RNNs for the transformed data. The experiments performed on architectures based on CNNs show that:

- CNN performance is satisfactory on MFCCs transormed audio data. It comes that observation that MFCCs transformed data plots looks very simmilar to images.
- To maximize performance of double CNN, we should prepare voting method among models. Separate models does not full-fill the task.

Whereas experiments performed on RNN (in particular LSTM) based architectures have led us to following conclusions:

- LSTM networks tend to prefer very small learning rates,
- such networks tend to learn significantly faster than CNN ones,
- structures with decreasing number of blocks in following layers tend to achieve better results than the ones with constant size.

Bibliography

[1] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow : concepts, tools, and techniques to build intelligent systems*. eng. Second edition. Sixth release. Beijing ; Boston ; Farnham ; Sebastopol ; Tokyo: O'Reilly, 2020. ISBN: 9781492032649.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[3] Chris Nicholson. *A beginner's guide to LSTMs and Recurrent Neural Networks*. URL: `https://wiki.pathmind.com/lstm`.

[4] *TensorFlow documentation*. URL: `https://www.tensorflow.org/api_docs/python/tf`.

[5] Valerio Velardo. *Mel-Frequency Cepstral Coefficients Explained Easily*. URL: `https://www.youtube.com/watch?v=4_SH2nfbQZ8`.