Speech Recognition Challenge

Federico Zanotti
Federico.zanotti@studenti.unipd.it

1. Introduction

Speech Recognition has become very famous in the recent years, but it was born a long time ago. The first experiment is Radio Rex, a toy released in 1911 [1]. It was a dog that came out of its house when the user exclaimed its name (only name Rex was compatible). It used an electromagnet sensitive to the vowel found in the word Rex. From 1911 other experiments contributed to improve speech recognition ability and importance, but nowadays the most popular applications of Speech Recognition are Google voice assistant, Alexa and Siri. The first modern virtual assistant was Siri, installed on Apple devices in 2010. The Google Voice Search Application was released one year later in 2011, while Alexa is pretty new in this type of market, because it was announced in 2015.
These three well-known devices emphasize how much speech recognition is widespread and important.

Siri, Alexa and Google can do a lot of actions with speech commands. They enable hands free technology that is useful when the hands are busy, like driving. They can create documents by speaking, faster than writing, and facilitate note taking. In businesses that provide customer services, speech recognition reduces organizational costs. Any user in fact can input information like age, name without interacting with a live agent. In a call centre this reduces the waiting time for user calls.

In this report I created a neural network using Python and Keras library [2], trained to make predictions of the words: 'down', 'go', 'left', 'off', 'on', 'right', 'stop', 'up'. The goal was to search the optimal Hyperparameter configuration that the net needs in order to reach a good accuracy value. In my model I reached an accuracy of 83,5% of the model, with a prediction accuracy of 89% for the test dataset

2. Database

The dataset is composed of 1600 train samples, 200 for each word ("down", "go",…) and a 109 validation samples. To extract the features to train, the data was processed and reshaped.

First of all, the sound is a sequence of different vibrations, and we can represent it as a Spectrogram, a graph with time as X axis and frequency as Y axis. It is created using the Fourier Transformation to transform from time to frequency domain on a little window of the audio. In my case I use a window of 1 second.
Then the Mel scale, which is a transformation of the frequency scale, is applied on Spectrogram. In this way the scale tries to capture distances from low to high frequency [3]. From the Mel spectrogram, using logarithm, it's obtained the log-Mel spectrum.
For the data in the project, it's created a bi-dimensional representation of shape [128,32], then it's resized into a [32,32] matrix and then flatten into a vector of 1024 features.

3. Method

In my project I use four different methods: Manual search, Random search, Hyperband search (like Random search but more efficient) and Bayesian optimization. I create a Keras model for every method, and I use Keras Tuner [4] for Random, Hyperband search and Bayesian optimization.

Manual search: I start from an initial standard configuration of the Hyperparameter and then I make some experiment with this order: neural network structure (layers and neurons), activation functions, weights initialization, dropout, learning rate and optimization algorithm. In every experiment I start from extreme values, in order to analyze borderline cases. For every parameter I also plot accuracy and loss (cross-entropy) with train and validation samples for the purpose of detecting the learning behavior of the network.
When the search space becomes quite small, I compute the ultimate values for a number of iterations, with 5 as default value, recording and measuring the mean of the accuracy and loss of every value. The best means becomes the best parameter and then I update the configuration with the new value. If I reach an accuracy very similar between different values, then I analyze their plot, always choosing the model that doesn't create too much overfitting.
In the case of the Learning rate and Optimization algorithm I use more methods. The first uses three decay schedules

for the learning rate, decreasing it in three different ways, while in the second one I use the class LRFinder [5] . This last procedure is opposite to the other, because instead of decreasing the learning rate, it increases it up to a customizable value.
At the beginning I thought that this type of process was too simple, but actually at the end it is a good way to make a lot of experiments and at the same time take a look at the behavior of the model, with the different plots. In the following table (Table 1) I compare the results of the predictions for all methods.

Method	Loss	Accuracy	Time
Manual	0.32	89%	3 millisec
Random	1.30	51,5%	7 min
Hyperband	0.09	98.5%	20 min
Bayesian	2.85	36%	16 min

Table 1. Results of predictions. Time is the execution time of the method

From the table the best model is obtained with Hyperband, a more efficient Random search, but the execution time is the highest. With the manual search I get a good accuracy but with a low execution time.

4. Experiments

The method with which I have done more experiments is the manual search, because I tune only one parameter every time.

The first parameter I test is the number of hidden layers and I start from extreme values.

Num hidden layers	Accuracy
1	70%
5	58%
10	11%
15	11%

Table 2. Number of hidden Layers comparison

In the table above (Table 2) we can see that with a high number of layers I have a decreasing value of accuracy, so I try to reduce the range values from 1 to 10, obtaining a good accuracy value only for 1 to 6. Then I recompute the model more times, changing value in this range, and I have obtained the best accuracy for value 1. So, the first parameter I update is the number of hidden layers.
I do the same thing with number of Neurons. Extreme values like 10240 or 20480, have similar accuracy values of 1024. This points out that if I increase the number of neurons for large values, there is no advantages, so I try with smaller multiples of 1024 obtaining a better accuracy with value 1024.
For the activation function I restrict the range of values to four elements: Elu, Selu, Relu and Softplus. They have very similar loss and accuracy value, but in the plot only one function doesn't create overfitting: Relu function.
From the figures Relu has less overfitting and less variability

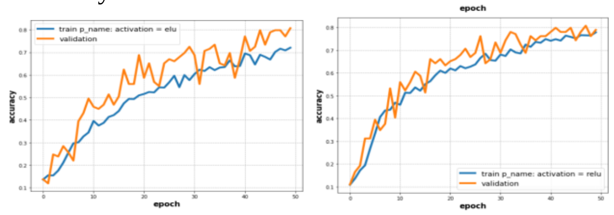

Figure 1. Elu on the left. Relu on the right

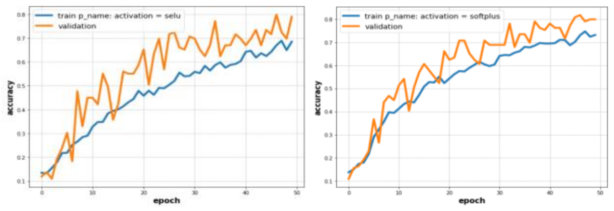

Figure 2.Selu on the left. Softplus on the right

Weights Initialization. I have similar loss and accuracy value for seven different weights initializations, but *he_uniform* [6] initialization creates less overfitting and variability.
In order to regularize my model, I add a dropout layer between every layer, and with some calculations I can exclude high values like 0.7, 0.8 and 0.9 and I find the best parameter equal to 0.1, with highest level of accuracy.

At this point I have to check for Learning rate and Optimization algorithm parameters. At the beginning I had chosen SGD (stochastic gradient descent) as optimization algorithm to find best learning rate, but after some plots I notice that SGD is very irregular, so I decide to find first the best optimization algorithm with same learning rate value (fixed value) and then I try with a variable learning rate. This exploration points out many complications with many algorithm optimizations. In Fact, almost all plots are very irregular, except for Adam and Adamax optimization.

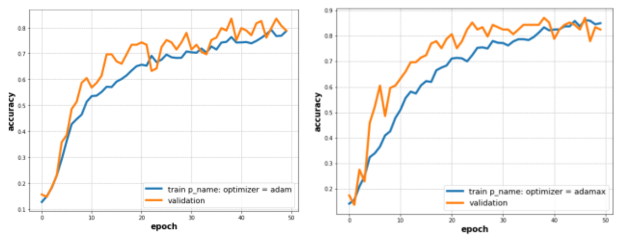

Figure 3. Adam on the left. Adamax on the right

I use these last two algorithms with three different

functions of schedule learning rate: Time based, Step based and Exponential based.
Only the time-based decay represents a good plot:

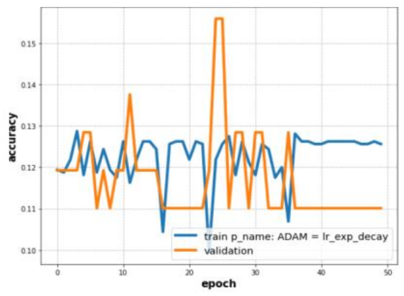

Figure 4. Exponential decay

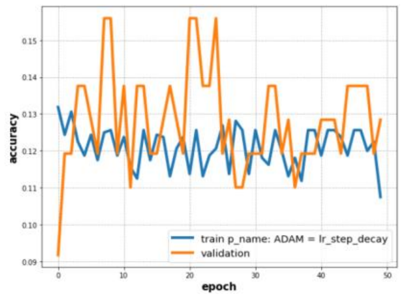

Figure 5. Step decay

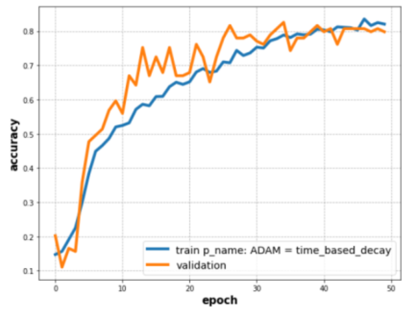

Figure 6. Time based decay

Are Adam and Adamax best Optimization algorithm for my model? Although I did a lot of experiments, I can't confirm that hypothesis, but I can plot how the different algorithms behave in a learning rate range. With LRfinder I create that range and then I plot it:

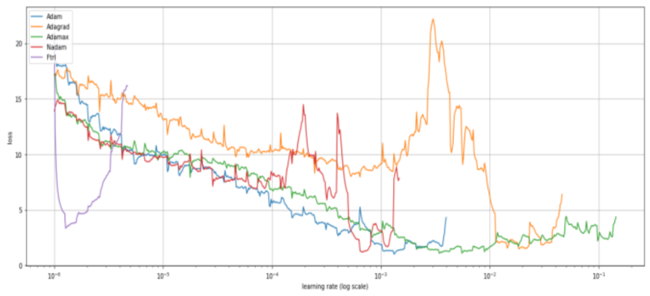

Figure 7. Yellow=Adagrad. Purple=Ftrl. Red=Nadam, Blu=Adam; Green=Adamax

In the Y axis there is the loss, and in the X axis there is the learning rate in logarithmic scale. This plot confirms my theory: Adam and Adamax are the best Optimization algorithms, because they reach a good and stable loss value. For these two algorithms I calculate the minimum loss value, in order to get the best learning rate, and then I plot two model: one with Adam and the other with Adamax:

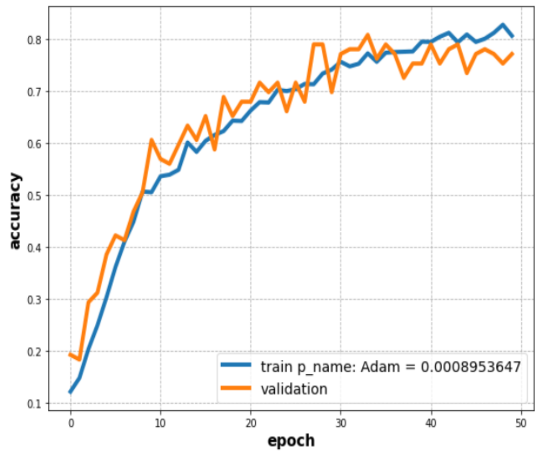

Figure 8. Adam Final Model

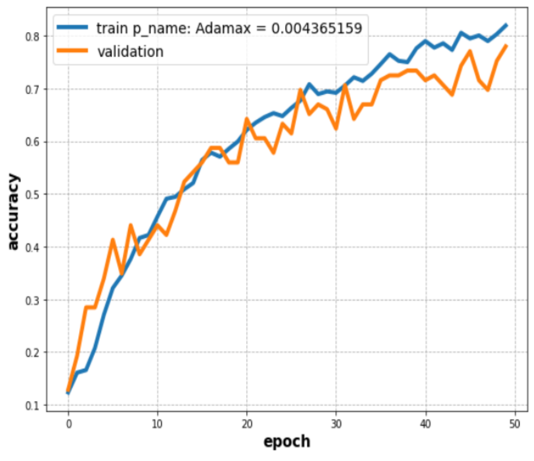

Figure 9. Adamax Final Model

At the end I have chosen Adam, because for a number of epochs equal to 50, it creates less overfitting respect to Adam.

References

[1] Virtual Assistant: https://en.wikipedia.org/wiki/Virtual_assistant.

[2] Keras Library: https://keras.io/.

[3] D. Gartzman: https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0.

[4] Keras Tuner: https://keras-team.github.io/keras-tuner/.

[5] LR Finder: https://github.com/davidtvs/pytorch-lr-finder.

[6] He Uniform: https://www.kite.com/python/docs/keras.initializers.he_uniform.