Speech commands
Machine Learning

Alice Daldossi

Abstract

In this laboratory activity a problem of speech recognition is solved using a multi-layer perceptron (MLP). In particular, I will try to recognize the sound of 35 commands from a training set of 84291 audio clips uttered by many different speakers. I will compare the results obtained from the MLP built with different size of the batch of the gradient descent, different number of hidden layers and of neurons. Moreover, I will study the best feature normalization.

1 Data and visualization

Script: `Training.py`

The collected data are 105829 and they are already divided in training, test and validation set. Moreover, the feature extraction has already been performed: the features are spectrograms extracted from the recorded waveforms and then reshaped as vectors of 1600 components. I reported the visualization of the mean spectrogram in figure 1. The range of the frequencies is very wide and this leads the method to

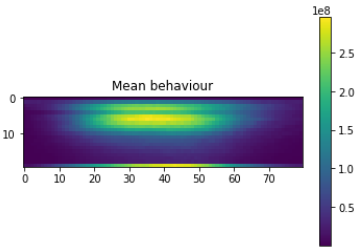

Figure 1: The most influential part is colored in yellow and it happens to be concentrated in the middle. The actual range of frequencies is larger than 20 (the highest value in the figure), but here the values greater than 20 are collapsed in the last raw, causing another yellow area.

consider just the more heavier values, ignoring even the middle frequencies. This is a problem because the cancellation may concern the effectively important sounds, as the highest frequencies may be caused by sneezes, coughs or other background noise. A possible solution consists in applying the feature normalization (see section 4).

I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.

2 Training

Script: `Training.py`

I imported the `pvml` library and used its MLP class to build my neural network (NN). In the training process I used the gradient descent (GD) method, the mean-var normalization of the features, no hidden layers (so it is a linear perceptron) and a loop of 50 epochs, while the evaluation is done every 5. In order to have a more efficient code, I used a compromise between batch GD and stochastic GD: I took minibatches. I tried different sizes of them and the results are written in the table 1, together with the accuracy of the batch gradient descent. It appears that the minibatch size of 30 and of 60 lead to very similar validation accuracies, but the largest one is better.

	Regular	BS = 1	BS = 30	BS = 60
Training accuracy	3.2 %	20.7 %	28.9 %	29.6 %
Test accuracy	3.1 %	14.1 %	18.1 %	18.1 %
Validation accuracy	3.2 %	13.8 %	17.7 %	**18.0 %**

Table 1: Comparison of the accuracies of the MLP with different values of the hyperparameter *batch size* (BS), but always the mean-var normalization, 50 epochs and no hidden layers.

3 Network architecture

Script: `Training.py`

The linear perceptron is not so accurate, in fact it gives too low accuracies. As improvement, I added some hidden layers of different width and studied the resulting accuracies. First I used 50 epochs and, as written in the table 2, the best choice between the tried ones seems to be the network with three hidden layers with respectively $900, 600$ and 300 neurons. In general, the validation accuracies are quite low, even if I increase the number of epochs to 100, as written in the table 3. In particular, it seems that there is convergence around a number of epochs equal to 50.

epochs = 50	0	600	600, 300	600, 300, 150	900, 600, 300
Training accuracy	29.4 %	81.4 %	81.8 %	90.0 %	92.1 %
Test accuracy	18.2 %	47.9 %	47.6 %	49.5 %	50.2 %
Validation accuracy	18.0 %	47.9 %	48.3 %	49.4 %	**51.0 %**

Table 2: Comparison of the accuracies of the MLP with different hidden layers using the mean-var scaling, minibatches of size 60 and 50 epochs.

epochs = 100	0	600	600, 300, 150
Training accuracy	31.0 %	87.5 %	94.5 %
Test accuracy	18.5 %	49.4 %	50.8 %
Validation accuracy	18.0 %	49.8 %	**51.3 %**

Table 3: Comparison of the accuracies of the MLP with different hidden layers using the mean-var scaling, minibatches of size 60 and 100 epochs.

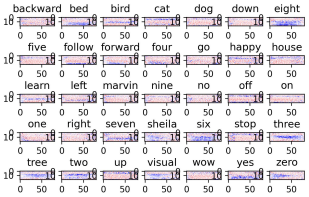

Figure 2: These are the spectrograms of the weights of each class. The different colors indicate the different intensity of the influence: blue is for positive, red negative.

4 Feature normalization

Script: `Training.py`

As mentioned in section 1, it is convenient to normalize the data before implementing the MLP. I compared different methods of feature normalization (min-max scaling, mean-var scaling, max abs scaling, whitening transform, instance normalization) and reported the results in the table 4. According to the validation accuracy, the instance normalization with the L_2 norm is the best one. Instead, the highest training accuracy corresponds to the whitening, but this normalization has also a validation accuracy that is lower than the instance's; this suggests a problem of overfit.

	Nothing	Min-max	Mean-var	Max abs	Whitening	Instance
Training acc.	20.7 %	9.7 %	29.4 %	9.7 %	29.6 %	20.5 %
Test acc.	14.8 %	9.6 %	18.2 %	9.5 %	15.7 %	19.1 %
Validation acc.	14.7 %	8.8 %	18.0 %	8.8 %	15.6 %	**20.2 %**

Table 4: These values are calculated using an MLP with no hidden layers and performing 50 epochs; the instance normalization is done with the most commonly used L_2 norm.

5 Analysis and visualization

Script: `Analysis.py`

I wanted to summarize the behavior of the network, so, using the saved values of the MLP with no hidden layers, 50 epochs, mean-var scaling and minibatches of size 60, I built a 35×35 confusion matrix C such that the element C_{ij} represents the percentage of the times the class i has been recognized as a sample of class j. Looking at this matrix I concluded that the class *six* is the most likely to be correct, because the corresponding value on the diagonal is the highest over the entire matrix C ($C_{25,25} = 65.7\%$). But the word *eight* is misclassified as *six* with a percentage of $C_{6,25} = 47\%$ and the opposite misclassification has a value of $C_{25,6} = 9.4\%$. I suppose that this phenomenon is caused by the fact that, looking at the spectrograms of the weights (figure 2) of these two words, the blue parts are in similar areas, so they need almost the same positive influence. Instead, *dog* is the most difficult to be correctly classified, it has a percentage of just $C_{4,4} = 2.1$; in particular, it has the most misclassification in *wow*. Once again the spectrograms of these two words *dog* and *wow* agree with the confusion matrix.

Moreover, the misclassification could be caused by a bad source data set. In fact, listening to the original audio clips corresponding to some misclassification errors and looking at their spectrograms, I presumed that background noise or faulty recordings may divert the MLP.