# Report for Mini-Project 1 ”Speech Commands” (Acoustic & Language Modelling)

Yonatan DELORO
yonatan.deloro@eleves.enpc.fr

February 24, 2019

The goal of this mini-project is first to build an acoustic model in order to discriminate between single voice commands (30 classes), and then to combine it with a language model in order to decode sequence of speech commands (of length around 4 to 11). The combination of best acoustic and language models found led to a word error rate of around 9% on the proposed test set of speech sequences.

## 1 Classification of single voice commands

### 1.1 Speech features extraction

I experimented with the two types of speech features proposed in the assignment, the mel-filterbanks and the MFCC features, as input data to the various classification acoustic models tried. I extracted these with the Spectral library using parameters adequate for human speech recognition. To build the mel-filterbanks features, I considered 0.97 as pre-emphasis parameter (amplify more the high frequencies than 0.6), time windows of length 25 ms (corresponds to around two times the pitch pulse period), on which are computed 512 points Fourier-Transform, and asked for 40 filter banks using [300,3400] Hz as frequency range (corresponds to the one used in telephony). In order to build the MFCC ones, I asked for the first 13 cepstral coefficients resulting from Discrete Cosine Transform, concatenated these with first and second derivatives (delta and delta-delta coefficients), and removed first cepstral coefficient and associated derivatives (first coefficient is said to capture essentially signal loudness).

In addition, the speech features in the 2d time/frequency-energy domain (with mel-filterbanks features or MFCC coefficients on the the energy axis) were flattened into a 1d-vector in the initial code. I chose instead to keep storing these features as 2d arrays in order to experiment with classification models appropriate for such structure (such as convolutional networks), especially for the mel-filterbanks features which seemed to exhibit clear patterns in this space (makes less sense for MFCCs as energies and derivatives are stacked). I modified the padding function adequately, thus ending with waveforms represented by arrays of dimensions (101,40) and (101, 36) for mel-filterbanks and MFCCs respectively.

Finally, as we aim at classifying single English words, we are theoretically not interested in features relative to the glottal source but only in the features relative to the vocal tract. Hence the MFCC features can appear a priori as a better option than the mel-filterbanks ones. However, as I will further describe, the best classification I obtained will take as input mel-filterbank features.

### 1.2 Acoustic classification models

Three type of models were tried :

**(i) Logistic Regression (LR)** A first proposal for this multi-

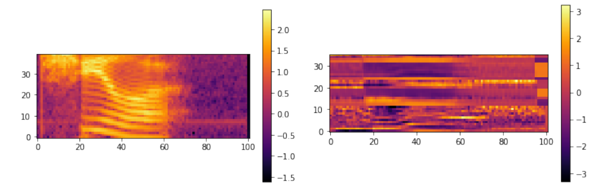

Figure 1: Mel-filterbanks (left) and MFCCs (right) features for a sample of word ”Sheila” (after classic normalization (cf.1.3))

label classification problem is to perform a logistic regression on the flattened speech features for each word class against all others. Each data is then labelled with the class obtaining the largest probability. I added a l2 regularization term to the cross-entropy loss of the LRs to allow for better generalization. In other words, the loss to be optimized for each of the 30 LR is : $\mathcal{L}(x_i, y_i) = -\sum_{i=1}^{N} y_i \log \sigma(wx_i) + (1-y_i)\log(1-\sigma(wx_i)) + \alpha||w||^2$, and I used SGD to do so for scaling purposes. I chose the $\alpha$ parameter maximizing the accuracy score obtained on the validation set.

**(ii) Fully-connected network (FCN)** I then tried a neural network with fully connected layers, taking as input the 1d-flattened speech features. The best I could get is composed of 2 hidden layers (with 256 and 64 hidden neurons respectively).

**(iii) Convolutional neural network (CNN)** Finally, I moved to a convolutional neural network taking as input the 2d-flattened speech features treated as images of depth 1. The architecture of the features extractor I ended with is composed of 3 convolutional layers (with increasing number of filters - 8,16 and 32 - and decreasing size of filters -(11, 5), (7,3) and (3,1))-, each conv layer followed by (2,2) maxpooling (or (2,1) for the last conv). Then a bottleneck with one hidden layer (64 neurons) enables to discriminate between the 30 classes. I designed the architecture especially for the Mel-filterbanks features.

For each layer (dense or convolutional) of both neural networks (FCN/CNN), ”relu” was choosen as the activation function, except for the last layer to which I applied the ”softmax” function, relevant for classification. Categorical cross-entropy loss was optimized with Adam (initial learning rate of 0.001),and training stopped whenever the validation loss did not decrease for three iterations. I also used dropout for hidden dense layers to struggle against overfitting.

### 1.3 Features normalization and data augmentation

For better generalization, I also observed the impact of normalizing the speech features and of augmenting artificially the dataset.

**(i) Features normalization** I notably compared two types of normalization : the ”classic” one, where we center and scale each

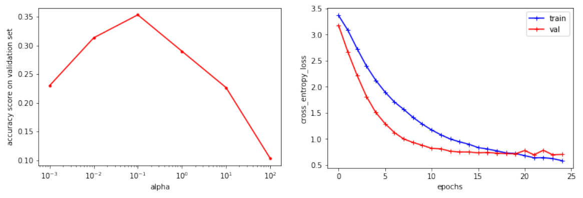

Figure 2: Left : Choice of the norm-2 regularization hyperparameter for the model based on Logistic Regressions. Right : Train and validation loss throughout the CNN training on Melfilterbanks features.

speech feature dimension computing mean and variance across the whole dataset, and a "sample-specific" one (known as the Cepstral Mean Normalization for MFCCs) which computes means and variances over time but for each specific sample. The theoretical advantage of this second normalization is that we do not assume constant channel conditions among the recordings.

**(ii) Data augmentation** I also observed the effect of augmenting the original train dataset of 300 instances per class with noisy versions of these. More precisely, the process of augmentation is the following : I select a random subset of the $300 * 30$ waveforms of the train dataset of a given size, and to each sample in the subset, I create a clone to which I add a random sequence extracted from the waveform of one of the 6 proposed background noises (artificial or real) lasting the same amount of time as the original sample waveform.

## 1.4 Results

| Features Extraction | Discriminative Model | Validation Score | Test Score |
|---|---|---|---|
| Mel-filterbanks | LR | 35.3 | 30.8 |
| Mel-filterbanks | FCN | 60.4 | 58.5 |
| Mel-filterbanks | CNN | 81.1 | **81.8** |
| MFCC | LR | 32.9 | 32.0 |
| MFCC | FCN | 62.0 | 55.7 |
| MFCC | CNN | 68.6 | 69.0 |

Table 1: Accuracy score on the validation and test set for each pipeline. 300 samples from each of the 30 classes in the train set, 30 samples per class in the validation set, as well as in the test set. "Classic" normalization was applied (cf 1.3). FCN : 1,052,894 / 949,470 parameters (256). CNN : 137,806 / 121,422 parameters (Mel-filterbanks / MFCC input).

| Features Normalization | Data Augmentation | Validation Score | Test Score |
|---|---|---|---|
| None | No | 78.3 | 70.0 |
| Sample-specific | No | 79.7 | 73.1 |
| Classic | No | 81.1 | **81.8** |
| Classic | +100% | 81.2 | **83.7** |

Table 2: Accuracy score on the validation and test set for each pipeline, using the CNN model on Mel-filterbanks features. "Classic" and "sample-specific" normalization, and artificial data augmentation process are described in 1.3.

On Table 1, we observe that, on the Logistic Regression Model, MFCCs are leading to the highest score on the test set. However,

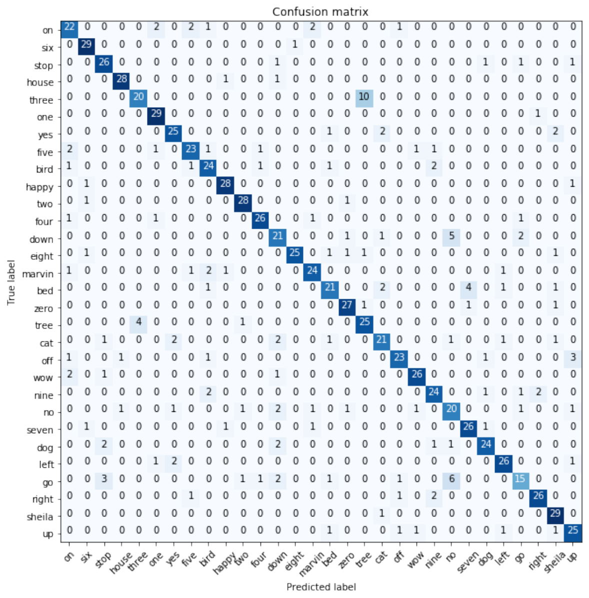

Figure 3: Confusion matrix on the test set (30 samples/class) for the CNN model on Mel-filterbanks features with classic normalization (no data augmentation).

the best model I obtained is the CNN I designed for Mel-filterbanks features. It is quite expected that the same CNN performs less well on MFCCs since the filters types I chose for Mel-filterbanks features are a priori not adequate for MFCCs stacked energies and derivatives. However, it may probably have been possible to design another network for MFCCs beating these performances (MFCCs are a priori better designed than Mel-filterbanks for English speech recognition as they characterize only the vocal tract).

On Table 2, we observe that normalizing across the whole dataset allows for better generalization than not doing it, or than doing it per sample (given the short size of the recordings, computing mean and variance over the whole dataset probably led to more accurate estimates), and that augmenting the dataset with noisy cloned samples also improves generalization.

As we see on the confusion matrix, some short words are still particularly difficult to recognize for the CNN (such as "three" often confounded with "tree", "down" and "go" often translated into "no"). The language model will probably be of help for these cases.

## 2 Classification of segmented voice commands

### 2.1 Evaluation metric and experimental setup

We will use the Word Error Rate (WER) as the metric to compare hypothesis and reference sequences. It is defined by $\frac{S+D+I}{N}$ where $S+D+I$ corresponds to the minimum total number of substitutions, deletions and insertions needed to align the two sequences, and $N$ is the length of the reference sequence.

**q1)** $S + D + I$ is non-negative by definition, and zero when the sequences are perfectly aligned. However $S + D + I$ can be larger than $N$. For instance, the hypothesis sequence can be of length $2N$ and contain no word of the reference sequence, in which case $S + D + I = 2N$ (shortest path to align the two sequences is composed of $N$ substitutions and $N$ insertions). Therefore, we cannot have WER<0 but we can have WER>100.

**q2)** The prior probability of each word was approximated to be equal to a constant as we trained the discriminator of speech commands with a balanced dataset, fixing a given number of examples per class, and using such code line :

```
elif train_labels.count(label) < nb_ex_per_class:
    train_wavs.append(waveform) ; train_labels.append(label)
```

## 2.2 Using the greedy decoder for decoding (no language model)

**q3)** The greedy decoder assumes full independence between consecutive spoken words : we label each waveform of the sentence with the word label of highest probability for the chosen discriminator acoustic model, ie. the CNN of part I.

Word error rate between "go marvin one right stop" and " go marvin one six stop" is 0.2 as the shortest path to go from one sentence to the other using the three operations (insertions, substitutions, deletions) is 1 : we substitute "right" by "left". Hence WER = 1/5 = 20%. In order to compute this edit distance, the algorithm "jiwer.wer" uses dynamic programming. If we denote $d(s[:i], s'[:j])$ the edit distance between sequence $s$ until character $i$ and sequence $s'$ until character $j$, then

$$\begin{aligned} d(s[:i], s'[:j]) =& d(s[:i-1], s'[:j-1]) \;\; if \; s[i] = s[j] \\ =& \min\{d(s[:i], s'[:j-1]), d(s[:i-1], s'[:j]), \\ & d(s[:i-1], s'[:j-1])\} \;\; otherwise \end{aligned}$$

| predicted s \| true s' | go | marvin | one | right | stop |
|---|---|---|---|---|---|
| go | **0** | 1 | 2 | 3 | 4 |
| marvin | 1 | **0** | 1 | 2 | 3 |
| one | 2 | 1 | **0** | 1 | 2 |
| six | 3 | 2 | 1 | **1** | 2 |
| stop | 4 | 3 | 2 | 2 | **1** |

Table 3: In cell (i,j), the distance between s[:i] and s'[:j]. jwer.wer initializes first row and column, then fills the cell (i,j) taking the min of the top cell (removing s[i]), left cell (inserting s'[j]), and top-left cell (substituting s[i] by s'[j]). Shortest path depicted in bold.

## 2.3 Injecting language models

### 2.3.1 Building the language model (N-gram)

**q4)** The Bigram approximation formula of the language model I first used is the following : $p(w_2|w_1) = \frac{C(w_1 w_2)}{C(w1)}$ if $C(w_1) > 0$, and $p(w_2|w_1) = \frac{C(w_2)}{S}$ otherwise, where $C(s)$ denotes the count number of the string $s$ in the corpus and $S$ is the size of the corpus (the corpus do not contain any other speech command). Note that $p(.|w_1)$ defines a probability distribution if we add, before and after each sentence, a "stop" symbol (defined as a "31-th" speech command). We will see how we can further modify this approximation formula in q10) to face sparsity.

**q5)** I did an implementation allowing to build any $N$-gram model. Hence, the $N$-gram approximation formula I first used (see q10) for smoothings), for any $N \geq 3$, is : $p(w_N|w_1, ..., w_{N-1}) = \frac{C(w_1...w_{N-1}w_N)}{C(w_1...w_{N-1})}$ if $C(w_1...w_{N-1}) > 0$, and $p(w_N|w_1, ..., w_{N-1}) = p(w_N|w_2, ..., w_{N-1})$ otherwise (and using the Bigram approximation formula of q4) to model $p(w_2|w_1)$ for $N$=2). In order to build the $N$-gram language model :

- I first extended each sentence with $N-1$ "stop" symbols (denoted "s") before the first and after the last word of the sentence. Indeed, when I will use for instance the trigram model, reading the sentence "go marvin right", I would like to model also P("go"|s,s) (probability that "go" begins the sentence), P(go,marvin|s) ("go marvin" is the first bigram), P(s|"right",s) ("right" is the last word), and P(s|"marvin", "right") ("marvin right" the last bigram).

- Then I computed the number of each unigram, bigram, ..., $N$ grams reading the texts of the training corpus. For each $k \in [1, N]$, I stored the $k$-grams counts in a $k$-dimensional array of side $V+1$ ($V = 30$ speech commands, each speech command being mapped to an index in $[0, V-1]$ and mapping the stop word "s" to index $V$).

- Finally, I computed the unigram language model $p(w_1)$, then the bigram one $p(w_2|w_1)$,... until the $N$-gram one $p(w_n|w_1, ..., w_{n-1})$ using the above approximation formula. I chose to store the counts and conditional probabilities in arrays (vs. dictionaries) as, even if we may find only some possible combinations of $k$-grams (sparse matrices), we will use smoothing, which will fill these matrices, and we will perform vector operations on these matrices for decoding (but I acknowledge such implementation may not scale in memory for larger number of speech commands than 30 and large $N$).

**q6)** If we want to model $P(w_N|w_1, ..., w_{N-1})$ with larger $N$, we may end with a model capturing longer-time dependencies between words in a sentence, but we need to face the sparsity problem arising: if the corpus is not large enough, there might be very few counts of $N$-grams so that the maximum-likelihood estimate of $p(w_n|w_1, ..., w_{n-1})$ written in q5) might become too approximate. We will see in q10) how to face sparsity for large $N$.

### 2.3.2 Decoding with the language model

**With beam search**

**q7)** In beam search algorithm, at each time $t$ of the sequence (a time corresponding to a new word), we are looking for the $K$ (beam-size parameter) most likely sequences ending at $t$ given that we kept in memory the $K$ best found sequences ending at $t-1$ (thus, the larger $K$ the less approximate the beam search is, and the longer it takes). To do so, at time $t$, (1) we compute the probability of each of the $K$ stored sequences ending at $t-1$ followed by each of the $V$ possible words ($p(w_1...w_{t-1}w_t) = p(w_1...w_{t-1})p_L(w_t|w_{t-N+1}, ...w_{t-1})p_A(w_t|s_t)$ where $p_A$ is the acoustic model, and where we used a $N$-gram language model $p_L$) ; (ii) we keep in memory the $K$ sequences ending at $t$ among the $KV$ candidates with highest probabilities. As I sorted the whole list of $KV$ candidates to select the top $K$ ones, the time complexity of my beam search algorithm is $\underline{O(TKV\log(KV))}$ where $T$ is the length of the sequence, $V$ is the number of possible speech commands and $K$ is the beam-size parameter. However, we can go up to $\underline{O(TKV)}$ theoretically using the quickselect algorithm to select the top $K$ probabilities among the $KV$ ones. Besides, the (auxiliary) space complexity of beam search is $O(KV)$.

**With Viterbi algorithm**

**q8)a)** Let us denote $\phi_j^t$ the probability to be in state $j$ at step $k$, then we have the relationship :

$$\phi_j^t = \max_{j'}\{\phi_{j'}^{t-1} \times p_L(j|j')\}p_A(j|s^t)$$

with $p_A$ the acoustic model, $p_L$ the language model, and $s^t$ the speech signal at "sequence position" $t$.

If we are given the bigram language model, $\phi_j^t$ is the probability of the most likely sequence ending with word $j$ at time $t$. For the $N$-gram model, let $\phi_{x_1,x_2,...x_{N-1}}^t$ be the probability of the most likely sequence ending with $N-1$-gram $x_1x_2...x_{N-1}$ from $t-N+1$ to $t$. Thus, we can explicit the recurrence equation below :

$$\phi_{(x_1,x_2,..x_{N-1})}^t = \max_{x_0}\{\phi_{(x_0,x_1,...x_{N-2})}^t p_L(x_{N-1}|x_0,x_1,...,x_{N-2})\} \times p_A(x_{N-1}|s^t)$$

These formulas are valid assuming that the sequence was extended with $N-1$ artificial signals before the start of the speech sequence and after its end, and having probability equal to 1 to be stopwords. ($p_A(s|.) = 1$).

**q8.b)** The Viterbi algorithm can be implemented by : (i) iterating forward in time to compute $\phi_j^t$ for each possible state $j$ for each $t \in [1,T]$ with $T$ the length of the sentence, hence we can access to the probability of the most likely sequence $\max_j \phi_j^T$ , (ii) iterating backward in time to access to the words enabling to reach this maximum probability, ie $j_T^* = \text{argmax}_j \phi_j^T$ and $j_t^* = \text{argmax}_{j'}\{\phi_{j'}^t \times p_L(j_{t+1}^*|j')\}$ for $t \in [1,T-1]$. Actually, the backward pass costs nothing as we can store during the forward pass the word $\text{argmax}_{j'}\{\phi_{j'}^t \times p_L(j|j')\}$ for each word $j$ (bigram model), and the word $\text{argmax}_{x_0}\{\phi_{(x_0,x_1,...,x_{N-2})}^t \times p_L(x_{N-1}|(x_1,x_2,...x_{N-2}))\}$ for each $(N-1)$-gram $(x_1,x_2,...x_{N-2})$ ($N-$gram model). The time complexity of the Viterbi algorithm is the one of its forward pass, equal to $O(TV^2)$ where $T$ is the length of the sequence and $V$ is the number of possible speech commands. Its (auxiliary) space complexity is $O(TV^{N-1})$ using a $N$-gram language model (we need to store $\phi$).

Notes : 1) I preferred to implement both beam search and Viterbi algorithm specifically for bigram and trigram language models in order to avoid any index error given stop words. 2) As product of probabilities can vanish towards 0 very rapidly, I computed instead the logarithms of the probability of the states at each time of the sequence.

## 2.4 Results

| Decoder | Train WER (%) | Test WER (%) |
|---|---|---|
| Greedy (no language model) | 38.9 | 33.8 |
| Beam search bigram (K=5) | 10.2 | 9.5 |
| Viterbi bigram | 9.5 | 9.3 |
| Beam search trigram (K=5) | 9.6 | **9.2** |
| Viterbi trigram | 9.6 | **9.2** |

Table 4: Word error rates for the three decoders, using bigram/trigram language model, and the CNN acoustic model (on Mel-filterbanks features normalized, without data augmentation). All take approximately the same computation time (37-40s)

Using a language model definitely improves performances. Though approximate, beam search with K=5 almost equalizes performances of Viterbi decoder (with our naive implementation of beam search, it is however not faster than Viterbi). Trigram model brings a tiny benefit but does not take more time.

**q9)** Our best language model found (trigram) leads however to systematic errors : for instance, reference 'go marvin zero/two' seems to be often predicted as 'go sheila zero/two", "happy cat" as "happy bird". There are also common errors for numbers followed by directions (such as "one right") (as all possible combinations are not present in the corpus).

**q10)** We do not want our language model to assign a zero probability to rare seen words or sequences, especially if we want to use higher $N$-language model (where many sequences can be rare in the corpus). A way to shift probability mass from more frequent sequences to rarely/never seen ones is to use smoothing. For this purpose, I tried : (i) Laplace smoothing which consists in adding 1 or 2 to all counts before normalizing into probabilities (ii) Good Turing discounting which re-estimates the mass to be shifted by looking at the number of N-grams with higher counts (revised count of count $c$ is $c^* = (c+1)N_{c+1}/N_c$ with $N_c$ the number of $N$-grams occurring $c$ times) As we see on Table 5, my implementation of these strategies did not improve performances (though I can suspect that my estimation of Good-Turing discounting is not accurate for all counts).

With more time, I would have implemented Katz smoothing which consists in backing off to the $(N-1)$-gram whenever the $N$-gram needed has zero counts. Thus, $p_{BO}(w_N|w_1,...,w_{N-1}) = d_C C/C(w_1...w_{N-1})$ if $C > 0$ and $p_{BO}(w_N|w_1,...,w_{N-1}) = \alpha(w_1,...,w_{N-1})p_{BO}(w_N|w_2,...,w_{N-1})$ otherwise, where $C$ is a short writing of $C(w_1...w_N)$, $d_C = \frac{(C+1)N_{C+1}}{CN_C}$ and $\alpha(w_1,...,w_{N-1}) = \frac{1-\sum_{C>0} d_C C/C(w_1...w_{N-1})}{\sum_{C=0} p_{BO}(w_N|w_2,...,w_{N-1})}$. A last strategy to face rare words such as "seven","eight","dog" which could have led better results, would have consisted in grouping words into clusters (all numbers/animals together) and building high $N-$grams models but with the classes instead of the speech commands (eg. modelling P(animal | surname,number) and P("dog" | animal)).

| Decoder | Train WER (%) | Test WER (%) |
|---|---|---|
| Viterbi trigram | 9.6 | 9.2 |
| + Laplace-1 smoothing | 8.2 | 10.0 |
| + Laplace-2 smoothing | 10.7 | 10.6 |
| + Good-Turing discounting | 8.9 | 9.2 |

Table 5: Smoothing strategies.

**q11)** In order to optimize jointly the language and the acoustic model, we could use Recurrent Neural Networks, with inputs corresponding to the speech signal, and outputs to the probability of transitions, which would be used to annotate the arcs of the language model (Weight-Finite State Transducer) [3].

# References

[1] Jurafsky, Martin, Speech and Language Processing (part 3.4) [2] Gales, Young, The application of HMM in Speech Recognition (part 2.3 and 2.4) [3] Kubo and al., Speech Recognition Based on Unified Model of Acoustic and Language Aspects of Speech [4] http://www.cs.cornell.edu/courses/cs474/2006fa/handouts/smoothing+backoff%20REV.pdf