DNN/HMM speech recognition systems

Monday 20th June, 2022 - 21:07

Rafidison Santatra Rakotondrasoa
University of Luxembourg
Email: rafidison.rakotondrasoa.001@student.uni.lu

Vladimir Despotovic
University of Luxembourg
Email: vladimir.despotovic@uni.lu

***Abstract*—This document [1] presents the Bachelor Semester Project made by Rafidison Santatra Rakotonsrasoa under the direction of his Project Academic Tutor Vladimir Despotovic. In this project, we are interested in the implementation of a speech recognition system. The idea of this project is to develop a Deep Neural Net speech recognition system. The scientific aspect concerns the methods, the processes and the theoretical aspects of our speech recognition system development using machine learning and neural networks. Furthermore, the technical aspect concerns especially on the programming aspect of our project using a specific speech recognition toolkit called Kaldi.**

1. Introduction

Communication among humans is dominated by spoken language, therefore it is natural for people to expect speech interfaces with the computer which can speak and recognize speech in the native language. In recent years, Artificial Intelligence (AI), has received increased attention and have changed due to technological evolution. Speech recognition also improved with it to a very advanced performance and became widely used. Industry investments in AI are rapidly increasing, and governments are trying to understand what the technology could mean for their citizens. Applications based on AI are already visible in many fields such as education, targeted treatments, healthcare diagnostics, transportation, games, marketing and more. In speech recognition, there are intelligent personal assistant applications like Siri by Apple, Google assistant by Google, Alexa by Amazon, Cortana by Microsoft and Bixby by Samsung which are the most used in the term of vocal assistant nowadays.

Those ASR technologies improved with Deep learning which contributed to overcoming issues with previous speech recognizers based on Gaussian Mixture Models (GMMs). Others factors led the improvement of the field such as several speech-related projects and challenges. Among the other factors, the development of open-source software plays also an important role. One of that software is the Kaldi toolkit which helped a lot of popularising ASR applications. There exist other open-source toolkits such as Hidden Markov Model Tool Kit (HTK) [2] or Julius [3].

Kaldi [4] provides sophisticated facilities for speech analysis, Hidden Markov Model (HMM) training, testing and result analysis. The software can be used to build complex HMM systems. This toolkit is composed of modern and flexible code, easy to understand, modify and extend. Also, some other features and libraries are included in Kaldi. Instead of building our system from scratch, we will use this software to develop and work on speech recognition and deep learning for our project. After the speech feature extraction the HMM-GMM system generates labelled frames, and in the other hand, a Deep Neural Network (DNN) is trained from the labelled frames which were generated by an HMM-GMM system. Labelled frames are the alignments between phonemes and audio. From those audio features, we build an acoustic DNN model that take those features as inputs and train to get better predictions. We compare what We iterate over all of our training frames to improve the weights of our network in order to get better predictions.

2. Project description

2.1. Domains

The aim of this project is to explore how a DNN/HMM speech recognition system can be created and managed using Kaldi toolkit. This Bachelor Semester Project is done in different domains in Computer Science such as Machine Learning, Neural Networks, Speech recognition.

2.1.1. Scientific. Speech recognition, also known as Automatic Speech Recognition (ASR), is the process of transcribing speech to text. The speech recognizer creates a waveform from a speech, which is then broken into phonemes. For instance, English has 44 phonemes, which are further used to build words and sentences. One approach to speech recognition is to create statistical models for each word in the vocabulary and use this statistical pattern classification to recognize speech. The system uses statistical probability analysis to deduce words and sentences.

Machine learning is a subset of Artificial Intelligence that studies the computer algorithms that improve automatically through experience. An algorithm is a sequence of

instructions developed by programmers which are used to solve a problem. In Machine learning, learning algorithms not the programmers create the rules. This approach gives the computer instructions that allow it to learn from data instead of programming step-by-step instructions. Machine learning gives the computer the ability to learn by itself. Based on data training, the computer should be able to improve itself to give better results like a speech recognition system for instance.

Neural networks are a set of machine learning techniques, inspired by the biological neural networks that constitute human brains. It has an important role in Machine Learning and in speech recognition. Neural Nets contain statistical models which are used to generate outputs or results from the model. Deep Neural Networks are more complex models of Neural Networks. A deep neural network is simply a feedforward network with many hidden layers.

2.1.2. Technical. As parts of an Automatic Speech Recognition (ASR), there are different models such as the acoustic model, language model and pronunciation model. The idea of an ASR is to find the most likely word sequence according to the acoustic model and observations, the language model and the pronunciation model. First, the acoustic model represents the likelihood of an observed acoustic signal given a word sequence. The language model represents the likelihood of an observed word sequence. The pronunciation model contains all possible sequences of phonemes that compose all words in the corpus.

Kaldi is an open-source speech recognition toolkit. It is one of the best Speech Recognition toolkit to build our speech recognition system. It provides a set of libraries for efficiently implementing state-of-the-art speech recognition systems and includes a large set of recipes that cover some popular speech corpora. Besides Kaldi, technical aspects can be data preparation, feature extraction, training and decoding.

Data preparation is the first step in order to use Kaldi. For instance, all audio files should be re-sample as expected by the toolkit. Besides audio recordings, we need to prepare meta-data for the acoustic and the language model. The tasks are done manually or we can program scripts for the large iterative task. In fact, the speech recognition training starts with a corpus containing a collection of transcripted speech recordings. Some files should be created such as *spk2gender, wav.scp, text, utt2spk, corpus.txt,* files. The acoustic data includes gender information on the speakers (spk2gender), the identifier and the audio data for each utterance (wav.scp), the transcripts for each utterance (text), the mapping between utterances and speakers (utt2spk) and the corpus's transcript (corpus.text). The language data includes the lexicon, and the non-silence and silence phone information.

Next, Mel Frequency Cepstral Coefficients (MFCCs) are extracted from a speech signal. A speech signal is represented as a sequence of phonemes and each phoneme is represented by three-state Hidden Markov Model (HMM) with each state being represented by the Gaussian model. A single mixture Gaussian model does not represent the distribution of feature vectors in a better way, then the Gaussian Mixture Model (GMM) is used. Monophone and triphone models are generated by the HMM-GMM system.

After, we perform the training with the training samples. We train the network to get better results each time by minimizing the cost or the error.

Finally, we perform the decoding. Given an utterance, we want to find the most likely sequence of words. We create the decoding graph for inference. Afterwards, we can decode the testing data with and we get the results.

2.2. Targeted Deliverables

2.2.1. Scientific deliverables. The scientific deliverables are divided into two parts. The first deliverable is a scientific representation of an Automatic Speech recognition. In this field, we will explain speech recognition in general. We will encounter language modelling and speech signal processing. The second deliverable is deep learning and building neural networks for a speech recognition system. In fact, Deep Neural Networks model (DNN) and the Hidden Markov Model (HMM) are used to build our speech recognition system.

2.2.2. Technical deliverables. There are two deliverables in this section which allow us to answer the question of how to use the Kaldi software to build our DNN/HMM speech recognition system. Kaldi makes our tasks much easier instead of programming from scratch. We will see all the steps to create a speech recognition system including acoustic and language modelling. In our project, we will use a dataset of speech sample to train and test our model. The second deliverable consists of Deep Neural Net training and testing on our dataset in order to get the best results.

3. Pre-requisites

3.1. Scientific pre-requisites

In a speech recognition system, domains such as machine learning and neural networks are used. In machine learning, background knowledge in statistics is required. Indeed, data analysis is also helpful, as well as linear algebra, or other college-level math. Deep Learning fundamentals will be very helpful in this project which requires basic knowledge in neural networks. This field is a rapidly growing field in Data Science.

3.2. Technical pre-requisites

In order to build an Automatic Speech recognition in this project background in Unix shell scripts and some knowledge about speech recognition and ASR systems, in general, are helpful. Unix provides different flavors of shells

such as Bourne shell (sh), C shell (csh), TC shell (tcsh), Korn shell (ksh) and Bourne Again shell (bash). Over time, people developed many different varieties of shell by adding extra features for the command line user. In our case, we use bash scripts even if Kaldi can work with any type of shell, but the example scripts are done this way as many people are familiar with this shell. It is also recommended to use Linux although it is possible to use Kaldi on Windows. Being familiar with Matlab or SoX [5] is helpful to be able to re-sample audios because the audio data should be correctly sampled. In fact, some toolkit expects a 16 kHz sampling rate of audio files.

4. A Scientific Deliverable 1

4.1. Requirements

All speech recognizers have an initial front end that converts speech signal into feature vectors. The first step in any automatic speech recognition system is to extract features and identify the components of the audio signal that are good for identifying the linguistic content. We use the Mel Frequency Cepstral Coefficients (MFCCs) method for feature extraction. MFCCs are widely used in automatic speech and speaker recognition. MFCCs catch important features of the signal and compress those pieces of information into a small number of parameters.

Pattern processing is used to capture the phonetically important characteristics of speech. It determines the category of each pattern by comparing with stored patterns to recognise speech and transcribe it to text.

Hidden Markov Model (HMM [6]) is a stochastic approach which provides the most robust way of quantifying and classifying speech patterns. Speech is described by parameterized statistical models and the speech recognition is performed by considering the most likely model which produces the given sequence of observations. We aim to develop this continuous speech recognition using HMM. This model has several states and gives the probabilities for going from one state to another state. Usually, we consider that one phone or phoneme represents three or several frames. Frames are sub-phone units.

In DNN training, we train the labelled frames which were generated by an HMM-GMM system. Neural networks with many hidden layers and a large output layer are used to account for the large number of HMM states used for modelling phones. We start from the input layer, go through the hidden layers to the output layer. In the output layer, we make a prediction, compute the cost or the error and update weights in order to minimize the error. The DNN outputs the posterior probabilities of HMM states with given acoustic observations. These probabilities are divided by the prior probability of each state during the decoding, that is used instead of the state emission probabilities in HMM.

4.2. Design

Our ASR objective is to find the most likely word sequence $\hat{W}$ according to the acoustic observations, the pronunciation lexicon and the language model. Pronunciation dictionary or pronunciation lexicon should be composed of all possible pronunciations of all words in our data.

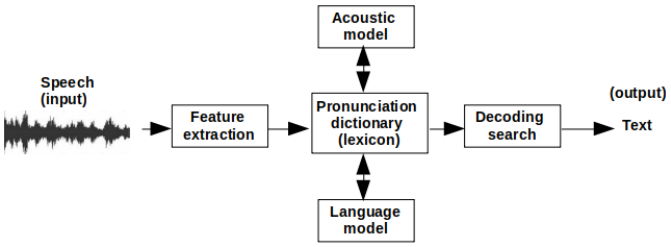

Figure 1. ASR system

The statistical model of an ASR represents a priori probability of word sequence obtained from a model of the language $P(W)$ and the probability $P(X|W)$ which is the probability of X computed on the probability distribution function of the word sequence W. Hidden Markov Models represent probability distributions of word sequences. We want to know the probability that a signal belongs to a particular class in our model, it is the probability of word W knowing its acoustic observations X, $P(W|X)$.

The input waveform is compressed or extracted into fixed-size acoustic observations, $X = x_1, x_2, x_3, ..., x_T$.

After the feature extraction, the decoder attempts to find the sequence of words $W = w_1, ..., w_L$ which is most likely to have generated X.

The idea is to compute $P(W|X)$ called posterior probability, using the estimation $\hat{W} = argmax\ P(W|X) = argmax\ P(X|W) \times P(W)$.

$$\hat{W} = argmax\ P(X|W) \times P(W)$$

where $P(X|W)$ is the likelihood of an observed acoustic signal X given a corresponding sequence of words W which represent the acoustic model and $P(W)$ is the likelihood of an observed words derived from the language model which represent the language model.

Feature extraction

In speech recognition, the main goal of the feature extraction step is to compute a parsimonious sequence of feature vectors providing a compact representation of the given input signal. Feature extraction is the first part of speech recognition and it plays an important role to separate one speech from others. Features are used to emphasize individual speech characteristics embedded in the utterances. These characteristics can be extracted from a wide range of feature extraction techniques proposed and successfully exploited for the speech recognition task. This forms a sequence of acoustic observations $X = x_1, x_2, x_3, ..., x_{39}$

with the typical length of 25 milliseconds, as shown in Figure 2.

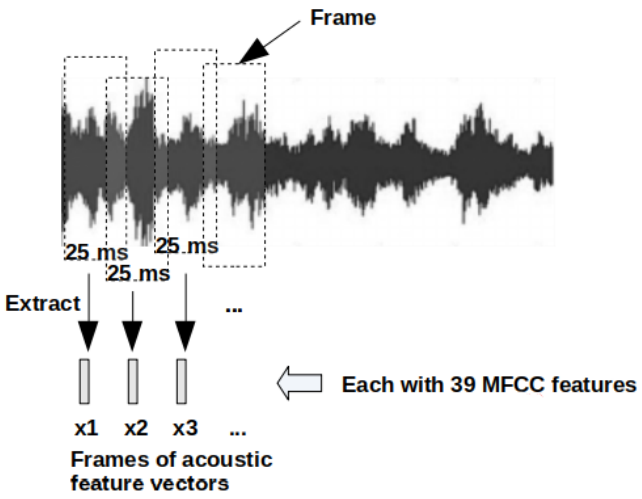

Figure 2. MFCCs feature extraction

The acoustic model is a statistical model for phonemes that represents all possible values of all the possible signals represented by MFCCs. Each audio frame will contain 39 MFCC features.

Acoustic model

The acoustic model is the statistical distribution for X corresponding to word W is created to compute $P(X|W)$. Each word or each phone of the word is represented by a statistical model called Hidden Markov Model or HMM and modelled by a pronunciation lexicon. In our case, we have in our dictionary 20 different phonemes that represent 20 classes in our model. We take a signal and compare it to all signal classes. From the model, we have 20 probabilities and choose the largest probability to match a target phonemes. A state is the representation of frames of a phone, the combination of the state create a model of phones. Each state is modelled by a mixture of Gaussian distributions or Gaussian Mixture Model (GMM) and the Hidden Markov model gives the probability of going from one state to another state or one frame to another frame in order to get the model of all phonemes. Single mixture gaussian component does not fit the distributions of feature vectors. Hence mixture splitting is performed successively in stages to multiple mixture components. Usually, we define the number of mixtures. For instance, we can see in figure 2 the model of a phone (HMM) and a Gaussian Mixture Model (GMM) of 3 distributions (3-component GMM). Each distribution is represented by 2 parameters, the mean and the variance.

Triphones and monophones are important concepts for the HMM acoustic model. We talk about monophone when each phone does not depend on the others and triphones represent the context-dependent phones.

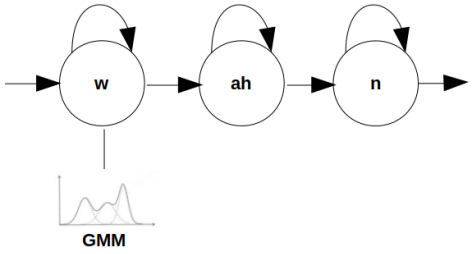

Figure 3. Acoustic model $P(X|"one")$

Language model

To improve the accuracy of ASR, a language model is used to compute the transition probabilities between words and to restrict word search. We can extend the language model into *n-gram*. The *n-gram* language model assumes that the probability of a phone is dependent only on the past *n-1* phones. For example, a *2-gram* or *bigram* language model assumes that the probability of a phone is dependent only on the previous phone.

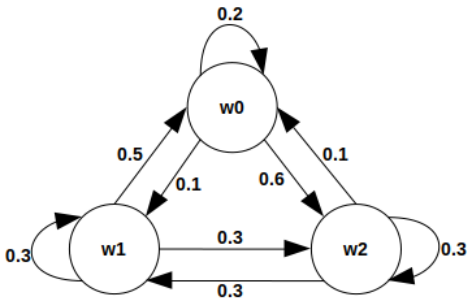

Figure 4. Language model representation in bigram

Figure 4 represents the state diagram and the transitions for the bigram model. In this model, we simplify the language model into a bigram. The states in the model are represented by phones. Instead of words, our model is build with the sequence of phones.

We will see in the technical part on this paper a language model toolkit that builds an internal N-gram count set by reading counts from a file. Then it generates and manipulates N-gram counts, and estimates N-gram language models from them.

Deep Neural Network

There are two basic tasks in machine learning: classification and prediction or regression. In our case, we want to predict a class in which we put the value of the signal from different signals. Each phoneme in our data is represented as a class. In the other hand, the second task is classification in which we want to check in which pattern is a speech or a word.

To recognize speech, we split it at phones then try to recognize what's being said in each phone. We take the

best matching combination of phones that match the target word. Phone can be related to each other. In monophone, each phone is independent of the others but in triphone, there is a context-dependent between phones. Each phone is represented by sub-phonetic units known as frames. For the computational purpose, it is helpful to detect parts phones. Based on the features of each frame in phones, a recognition pattern is created by training audio samples.

This pattern represents statistical models that catch similarities of speech. The model should be correct enough to recognize word and robust enough to catch the variability of different speakers for instance. The goal is to obtain the high probability that whoever speaks a word, it should be able to recognise the word.

A mapping from words to phones is also defined by a pronunciation dictionary or pronunciation lexicon which contains pronunciations for each word in our data. However, this is not the only method for mapping words to phones, some complex function learned with machine learning can be used.

There are many sort of classification with multi-layer neural networks in machine learning such as a multi-layer perceptron. The task of our neural networks is to find the optimal weights $W^{(h)}$ and $W^{(out)}$. Our training data are used to learn our model parameters. First, we choose the cost function and minimize it the cost or the error. By iteration, the model tries to find the best weights by the Gradient Descent. In every iteration the weights are updated in order to minimize the error.

Figure 5 represents a Multi-layer perceptron with one hidden layer and one output layer. However, a deep neural network has more than one hidden layer. There are m feature

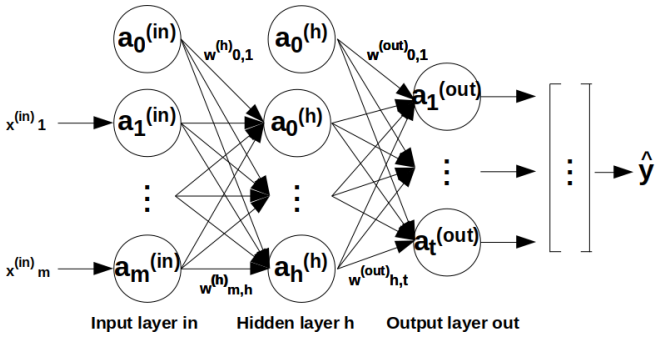

Figure 5. Multi-layer networks

vectors for each training instances. The number of training instances is n which is equal to 1500 training files. $A^{(in)}$ is the input of the hidden layer with nxm elements and $W^{(h)}$ the weights with $m \times h$ elements such that $Z^{(h)} = A^{(in)}W^{(h)}$ is the input matrix of hidden layer and $A^{(h)} = f(Z^{(h)})$ where f is the activation function. A the same time, $A^{(h)}$ is the output of hidden layer and the input of the output layer with $n \times h$ elements and $W^{(out)}$ the weights with hxt elements such that $Z^{(out)} = A^{(h)}W^{(out)}$ is the input of each of the output neurons. A the end, we apply the activation function to the outputs $Z^{(out)}$ of the output layer and get the predictive value. Once we have the prediction, we calculate the cost or the error.

In the beginning, random values are set for all the weights in the Neural Net.

The aim of the activation function is to introduce the non-linearity into the model. This function is applied for the output of hidden layer $Z^{(h)}$ and the output of the output layer $Z^{(out)}$. We can see bellow some examples of activation functions:

- *Sigmoid function*: sigmoid activation function is one common activation function. This function bounds values between 0 and 1 which are often interpreted as probabilities.
$$A(z) = 1/(1 + e^{-z})$$
- *Tanh or hyperbolic tangent function*: it re-scales the sigmoid function, such that its outputs have values between -1 and 1.
$$A(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$$
- *Softmax function*: softmax activation function bounds values between 0 and 1. It is a normalized version of the sigmoid function used for the output layer in multiclass classification.
$$A(z_j) = e^{z_j} / \sum_{k} e^{z_k}$$
- *ReLu function*: ReLu function is a simple activation function used basically for large deep networks applications.
$$A(z) = max(0, z)$$

We control the values of parameters for our deep learning network such as the number of iteration by minimizing the error, the number of the hidden layers, the number of inputs, the number of iteration for each hidden layer. The idea is to find the optimal parameters for our deep learning network in order to have the best performance.

4.3. Production

Data preparation

We start to prepare directories in Kaldi folder (cd kaldi/egs/digits) and we create all the sub-folders in the folder such as train which contains all information for the training samples and test which contains all information for the test samples. We have to define the audio data (cd digits/train and cd digits/test), the acoustic data containing files like spk2gender, wav.scp, text, utt2spk, corpus.txt in both folders train and test. Those acoustic data are generated using a bash script (acoustic.sh).

All of the audio files are not correctly named to be more useful with Kaldi. For sorting purpose, all the data are renamed in the correct way (speaker_digits_iteration.wav).

The data files are sorted by the speaker to be more useful than sorting by digits. The modifications are made by a bash script that swaps the first element in the audio name with the second element. Following this renaming process, we have to sort the audio files into speaker folders (jackson, nicolas, theo, yweweler) such that one speaker (jackson) is used for testing (cd digits_audio/test) and the others for training (cd digits_audio/test).

In our case, the language data is created manually with the lexicon file, the non silence_phones file, the silence phones file and the optional silence phones.

Feature extraction

We extract MFCC features and compute the cepstral mean and variance normalization (CMVN) stats for both training and testing data samples.

```
# Making feats.scp files
mfccdir=mfcc
utils/validate_data_dir.sh data/train
utils/fix_data_dir.sh data/train
steps/make_mfcc.sh --nj $nj --cmd "$train
_cmd" data/train exp/make_mfcc/train $mfcc
dir
steps/make_mfcc.sh --nj $nj --cmd "$train
_cmd" data/test exp/make_mfcc/test $mfcc
dir
# Making cmvn.scp files
steps/compute_cmvn_stats.sh data/train exp
/make_mfcc/train $mfccdir
steps/compute_cmvn_stats.sh data/test exp/
make_mfcc/test $mfccdir
```

where –nj option is the number of parallel jobs. We set the value of *nj* to 1 because we have a small dataset.

Language model

First, we make the language model lm.arpa file using SRILM [7]. SRILM is a language modelling tool. The function ngram-count in SRILM builds an internal N-gram count set by reading counts from a file. Then it generates and manipulates N-gram counts, and estimates N-gram language models from them.

```
local=data/local
mkdir $local/tmp
ngram-count -order $lm_order -write-vocab
$local/tmp/vocab-full.txt -wbdiscount
-text $local/corpus.txt -lm $local/tmp/
lm.arpa
```

We get the lm.arpa file which contains the priori probability $P(W)$ of all words in our corpus.

Finite state transducer (FST)

A decoding-graph of the HCLG format is created, the grammar or language model G, the lexicon L, the context-dependency C, and the HMM-topology H and form a single transducer H C L G. Kaldi uses OpenFst [8] for constructing and searching weighted finite-state transducers (WFSTs).

The decoding task is equivalent to finding the shortest path through the transducer. The acoustic GMM is the probabilistic model used to represent an acoustic output.

The G.fst file is created. The G transducer is called a factor transducer. It allows any substring of the original transcript to be detected.

```
lang=data/lang
arpa2fst --disambig-symbol=#0 --read-
symbol-table=$lang/words.txt $local/tmp/
lm.arpa $lang/G.fst
```

Next, the pronunciation lexicon L for each word is defined.

```
run.pl JOB=1:$nj $dir/log/compile_graphs
.JOB.log \
compile-train-graphs --read-disambig-
syms=$lang/phones/disambig.int $dir/
tree $dir/0.mdl $lang/L.fst \
"ark:sym2int.pl --map-oov $oov_sym -f 2-
$lang/words.txt < $sdata/JOB/text|" \
"ark:|gzip -c >$dir/fsts.JOB.gz" || exit
1;
```

Given an utterance, we want to find the most likely sequence of phones. Thus, we need to create the decoding graph for inference using the script $utils/mkgraph.sh$.

The transducer LG is composed.

```
fsttablecompose $lang/L_disambig.fst
 $lang/G.fst | fstdeterminizestar
 --use-log=true | \
 fstminimizeencoded | fstpushspecial >
 $lang/tmp/LG.fst.$$ || exit 1;
 mv $lang/tmp/LG.fst.$$ $lang/tmp/LG.fst
 fstisstochastic $lang/tmp/LG.fst || echo
 "[info]: LG not stochastic."
```

Next, we want to get a transducer whose inputs are context-dependent phones. Kaldi prepares an FST called CLG which is C o L o G.

```
fstcomposecontext $nonterm_opt --context
-size=$N --central-position=$P \
 --read-disambig-syms=$lang/phones/disa
 mbig.int \
 --write-disambig-syms=$lang/tmp/disamb
```

```
ig_ilabels_${N}_${P}.int \
$ilabels_tmp $lang/tmp/LG.fst |\
fstarcsort --sort_type=ilabel > $clg_tmp
mv $clg_tmp $clg
mv $ilabels_tmp $ilabels
fstisstochastic $clg || echo "[info]:
␣CLG␣not␣stochastic."
```

Afterwards, H transducer file (Ha.fst) using the HMM topology, the decision tree, and the transition model is created.

```
make-h-transducer $nonterm_opt --disambig
-syms-out=$dir/disambig_tid.int \
 --transition-scale=$tscale $lang/tmp/ila
 bels_${N}_${P} $tree $model \
 > $dir/Ha.fst.$$ || exit 1;
 mv $dir/Ha.fst.$$ $dir/Ha.fst
```

Now, we compose H o C o L o G.

```
fsttablecompose $dir/Ha.fst "$clg" | fst
determinizestar --use-log=true \
 | fstrmsymbols $dir/disambig_tid.int |
 fstrmepslocal | \
 fstminimizeencoded > $dir/HCLGa.fst.$$
 || exit 1;
```

At the end, we add self-loops to HCLG.

```
add-self-loops --self-loop-scale=$loopscale
--reorder=true $model $dir/HCLGa.fst | \
  $prepare_grammar_command | \
  fstconvert --fst_type=const >
  $dir/HCLG.fst.$$ || exit 1;
```

When the decoding-graph is created, we decode the testing data using the script $steps/decode.sh$.

4.4. Assessment

The aim in this section is to present our scientific deliverable such as feature extraction, pronunciation dictionary, acoustic model, language model, Hidden Markov Model, Gaussian Mixture Model. We know $P(W)$ is the priori probability of the word W that can be estimated from language model, $P(X|W)$ is the observation likelihood called the acoustic model. We introduced the concept of machine learning, deep learning and neural networks for a speech recognition system.

We can see some results using deep learning after a number of iteration with a number of the hidden layer, and data training. We apply some non-linear function known as the activation function for each neuron in our neural networks. We can find some results here.

5. A Technical Deliverable 1

5.1. Requirements

This section will cover the steps that are encountered to build the speech recognition system. The question in this part of our project is how to create a DNN/HMM speech recognition system in Kaldi toolkit using a set of data. To use Kaldi, you can find some information on how to install Kaldi and all software that we need here. The first step is to prepare data that is needed for building and testing our speech recognition system. It is a collection of grammar, corresponding pronunciation dictionary, recording wave files. A grammar collection corresponds to words to be recognized and a pronunciation dictionary or pronunciation lexicon that contains phonemes which constitute a word. The next part is the training phase and testing phase and the last but not the least is the result analysis. Finally, the results will be analysed.

5.2. Design

Dataset

We want to set our ASR system, using the audio files from the free-spoken digit dataset Github repository that we downloaded here[1]. This audio data used for the realization of this application is a set of isolated records. The corpus contains 2000 audio files with four speakers, 1500 used for training and 500 used for the test. The name of the file is the utterance ID that will be used in the program. There are 4 different speakers, each speaker uttering 500 words (digits from zero to nine). There is one word per audio file and there are 50 audio files per digit per speaker. The audio files are in wav format.

Each filename is changed using the following notation to be incorporated into the Kaldi system: *speaker_word_iteration.wav*, for example, *jackson_0_0.wav* or *jackson_0_50.wav*.

Sampling is an act of capturing a waveform over time and sample rate is the number of samples taken per second. The standard sampling frequency to sample the entire audio range is 44.1 kHz because 20 kHz is the maximum frequency component. However, the speech signal has frequency components up to 8 kHz, therefore 16 kHz is the optimal sampling rate. We used Matlab [9] to resample the audio files to the desired frequency of 16 kHz.

Data preparation

Phones are not homogeneous and change their characteristics across time. Every word is composed of phones. For instance, the word *one* can be pronounced differently like [*w ah n*] or [*w oa n*]. Variation in pronunciation of the same word are caused by regional and social differences but also differences in age and style of speaking (e.g. rapid or slow

1. https://github.com/Jakobovski/free-spoken-digit-dataset

speech) and so one. Hence, we may have different sequence of phones for a word but they should be transcribed the same way. Pronunciation dictionary or pronunciation lexicon should be composed of all possible pronunciations of all words in our data.

The acoustic data that we need is composed of the following files:

- *spk2gender* file[6]: for the test and train folders, containing the name and the gender of the speaker.
- *wav.scp* file[7]: for the test and train folders, matching utterance IDs to full paths in the directory.
- *text* file[8]: for the test and train folders, matching utterance ID to a transcription.
- *utt2spk* file[9]: for the test and train folders, matching utterance ID to speaker.
- *corpus.txt* file[10]: for the test and train folders, containing all possible text transcriptions.

The language data that we need is composed of the following files:

- *lexicon.txt* file: the lexicon file contains the phonetic transcription of all words in the corpus.

  !SIL sil
  UNK spn
  zero z ih r ow
  zero z iy r ow
  one hh w ah n
  one w ah n
  two t uw
  three th r iy
  four f ao r
  five f ay v
  six s ih k s
  seven s eh v ah n
  eight ey t
  nine n ay n

  From the lexicon we have the non-silence phones and the silence phones.
- *nonsilence_phones.txt* file: the non-silence phones are *ah, ao, ay ,eh, ey, f, hh, ih, iy, k, n, ow, r, s, t, th, uw, w, v, z*.
- *silence_phones.txt* file: this file is used to represent silence or unknown sounds such as *sil, spn*.
- *optional_silence.txt* file: this file is used to represent some optional silence sounds such as *sil*.

To build our language model we used SRI Language Model or SRILM toolkit. This toolkit is used to create statistical language models. The audio should be re-sampled to be manipulated by SRILM. There are a total of 1500 training utterances and 500 test utterances which have to be sampled at the rate of 16 kHz.

Training

For the training phase, we perform the monophone and triphone training. Since we cannot represent a frame or sub-phones units by just a single gaussian distribution, we apply a GMM or Gaussian Mixture Model for all frames. The model can go from one state to another state. Each state is modelled by GMM. GMM is the model of the sub-phones and HMM is the model for phones. The multi-gaussian monophone models so generated do not capture all the variations of a phone with respect to its context. It is observed that recognition accuracy increases as the number of mixture components are increased and it works well for tied-state triphone based HMMs for large vocabulary. Isolated digits speech corpus is used for implementation.

Monophone training

Once all the information is ready from the previous step, we can perform the monophone training.

In monophone training, each phoneme is independent and is represented by a pronunciation lexicon. A word is represented by independent phones with the highest probability of all phones of our stored signals. There is a transitive probability between those states from one phone to another.

```
steps/train_mono.sh --nj $nj --cmd
"$train_cmd" data/train data/lang
exp/mono || exit 1
```

Triphone training

The first step is to align data using the monophone system. Then we train the triphone system followed by the decoding-graph creation.

In triphone training, a phone depends on its surrounding phones. The probability of having a phone is assumed to be dependent on the surrounding phones. In this case, there are more models that represent all possible combinations of phones. This number increases exponentially in comparison to the number in monophone training.

As discussed before, to avoid the local optima, we train an ASR with multiple passes. We start training a monophone system. Then we move gradually to a triphone system seeded with information prepared in the previous training step. In most of the cases, we obtain better recognition results using triphone compared to monophone.

```
steps/train_deltas.sh --cmd
"$train_cmd" 2000 11000 data/train
data/lang exp/mono_ali exp/tri1
|| exit 1
```

5.3. Production

The aim is to build an isolated digit recognition system by exploiting deep learning acoustic modelling with Kaldi. We take the alignment of the previous model and perform a Deep Neural Network. In order to train our data samples, our Neural Net needs to be set up correctly.

In the running scripts for the neural network training and testing (*run_nnet2_simple.sh*), we can manage the values of parameters of our neural network. For the training phase, we tune the parameters in order to find the optimal set of parameters that maximise the performance.

A training dataset can be divided into one or more batches. We are talking about batch training when we take all the samples at once (batch size = size of the training set), stochastic batch training when we take sample one by one (batch size = 1) and minibatch training when we take a subset of sample for each iteration. The *minibatch-size* is the size of the subset of the training dataset samples that are used to update the weights. The batch size controls the number of training samples to work through before the weights are updated. In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples.

Learning rates define the amount of update made for the weights. The value of the learning rate starts with the initial rate and converge to the value of the final learning rate.

The number of epochs is the number of complete passes through the training dataset. We define also the number of iterations for each epoch. After the certain number of epochs, the predictions are compared to the expected output variables and an error is calculated. The number of epochs is traditionally large, usually, hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized.

The hidden layer dimension defines the number of neurons in each hidden layer. We can also set the number of hidden layers.

In our model, we start with only one hidden layer and additional hidden layer are added after a certain number of epochs. For instance, if we set the value of the added-layers-period parameter to 3, then we start with one hidden layer and after every 3 epochs, one layer is added.

The linear discriminant analysis dimension or LDA dimension is used to set the size of the feature vector. Each of the training data samples will be represented by this size of coefficient vectors. In our case, we set the number of vectors to m.

The neural network running file (run_nnet2_simple.sh) contains those parameters that have an important role for our DNN training process.

5.4. Assessment

Some results provided from the speech recognition system are shown in this section. Word Error Rate (WER) is the performance measure used to measure the accuracy of the ASR system. WER is the number of errors divided by the total words where the number of error is the sum of insertion, deletion and substitution. Substitution occurs when a word gets replaced (for example, "one" is transcribed as "two"), an insertion occurs when a word is added that was not said (for example, "one" becomes "one two") and deletion happens when a word is left out of the transcript (for example, "one" becomes "").

$$WER = (I + D + S) \div N$$

where I is the number of inserted words, D is the number of deleted words and S represent the number of substituted words.

Lower WER often indicates that the ASR software is more accurate in recognizing speech. A higher WER, then, often indicates lower ASR accuracy.

The results in table 1 are obtained from a monophone training using only the HMM/GMM model.

WER	Percent	Ratio
WER_7	8.40	42/500
WER_8	8.40	42/500
WER_9	8.20	41/500
WER_10	8.20	41/500
WER_11	8.20	41/500
WER_12	8.40	42/500
WER_13	8.40	42/500
WER_14	8.60	43/500
WER_15	8.60	43/500
WER_16	8.60	43/500
WER_17	8.60	43/500
Average	8.42	42/500

Table 1. Monophone training

The results in table 2 are obtained from a triphone training using only the HMM/GMM model.

WER	Percent	Ratio
WER_7	16.20	81/500
WER_8	15.80	79/500
WER_9	15.60	78/500
WER_10	15.60	78/500
WER_11	16.00	80/500
WER_12	16.00	80/500
WER_13	16.40	82/500
WER_14	16.60	83/500
WER_15	16.80	84/500
WER_16	17.40	87/500
WER_17	17.40	87/500
Average	16.35	82/500

Table 2. Triphone training

The results in table 3 are obtained using DNN acoustic modeling.

WER	Percent	Ratio
WER_9	8.20	41/500
WER_10	7.00	35/500
WER_11	6.40	32/500
Average	7.20	36/500

Table 3. DNN training

We can see in table 1 and table 2 that monophone training have better results than triphone training, the ASR is more accurate using monophone training than triphone

training. Triphone training does not match very well with our models.

We tested DNN training with different parameters. We deduced that making the model very complex with a large number of epochs, a large number of hidden layer and big dimension for each hidden layer does not mean necessarily that at the end we get better predictions. The model may become too well adapted to the training data and then cannot do good predictions for other testing data.

However, with the acceptable parameter, we can see in table 3 that DNN training has the best results.

Acknowledgment

I would like to thank the BiCS management and education team and the BiCS course director Mr Nicolas GUELFI.

I would also like to thank my Project Academic Tutor Mr Vladimir DESPOTOVIC, for this project, for his support, for his time spent helping me throughout the project and for the advice concerning the missions mentioned in this report.

6. Conclusion

In conclusion, the aim of this project is to build a DNN/HMM speech recognition system. This paper cover all important concepts such as data preparation, audio sampling, feature extraction, pronunciation lexicon, acoustic model, language model, Hidden Markov Model, Gaussian Mixture Model, finite state transducer, Deep Neural Network model, training and decoding. Different training such as monophone training, triphone training and DNN training was performed. The implementation the ASR using Kaldi was explained step by step.

References

[1] Nicolas Guelfi. Bics bachelor semester project report template, 2017. University of Luxembourg, BiCS - Bachelor in Computer Science, https://github.com/nicolasguelfi/lu.uni.course.bics.global.

[2] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D.Povey, V. Valtchev, P. Woodland, The HTK Book, Cambridge university, 2006.

[3] A. Lee and T. Kawahara, Recent Development of Open-source Speech Recognition Engine Julius, Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2009.

[4] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwartz, J. Silovsky, G. Stemmer, K. Vesely, IEEE 2011 Workshop on Automatic Speech Recognition and Understanding (ASRU),2011.

[5] SoX, 2012. Sound eXchange, 2015. PmWiki.

[6] D. Jurafsky, J.H. Martin, Speech and language processing, 2nd edition, Prentice Hall, 2008.

[7] SRILM. The SRI Language Modeling Toolkit, 2011. SRI International.

[8] OpenFst. library for constructing, combining, optimizing, and searching weighted finite-state transducers (FSTs) http://www.openfst.org/twiki/bin/view/FST/WebHome.

[9] Matlab. https://nl.mathworks.com/products/matlab.html?s_tid=hp_products_matlab.

7. Appendix

```
nicolas m
theo m
yweweler m
```

Figure 6. spk2gender

```
nicolas_0_0 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_0.wav
nicolas_0_1 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_1.wav
nicolas_0_10 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_10.wav
nicolas_0_11 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_11.wav
nicolas_0_12 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_12.wav
nicolas_0_13 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_13.wav
nicolas_0_14 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_14.wav
nicolas_0_15 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_15.wav
nicolas_0_16 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_16.wav
nicolas_0_17 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_17.wav
nicolas_0_18 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_18.wav
nicolas_0_19 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_19.wav
nicolas_0_2 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_2.wav
nicolas_0_20 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_20.wav
nicolas_0_21 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_21.wav
nicolas_0_22 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_22.wav
nicolas_0_23 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_23.wav
nicolas_0_24 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_24.wav
nicolas_0_25 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_25.wav
nicolas_0_26 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_26.wav
nicolas_0_27 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_27.wav
nicolas_0_28 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_28.wav
nicolas_0_29 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_29.wav
nicolas_0_3 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_3.wav
nicolas_0_30 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_30.wav
nicolas_0_31 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_31.wav
nicolas_0_32 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_32.wav
nicolas_0_33 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_33.wav
nicolas_0_34 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_34.wav
nicolas_0_35 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_35.wav
nicolas_0_36 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_36.wav
nicolas_0_37 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_37.wav
nicolas_0_38 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_38.wav
nicolas_0_39 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_39.wav
nicolas_0_4 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_4.wav
nicolas_0_40 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_40.wav
nicolas_0_41 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_41.wav
nicolas_0_42 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_42.wav
nicolas_0_43 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_43.wav
nicolas_0_44 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_44.wav
nicolas_0_45 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_45.wav
nicolas_0_46 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_46.wav
nicolas_0_47 /home/santatra/Documents/kaldi/egs/digits/digits_audio/train/nicolas/nicolas_0_47.wav
```

Figure 7. wav.scp

```
nicolas_0_43 zero
nicolas_0_44 zero
nicolas_0_45 zero
nicolas_0_46 zero
nicolas_0_47 zero
nicolas_0_48 zero
nicolas_0_49 zero
nicolas_0_5 zero
nicolas_0_6 zero
nicolas_0_7 zero
nicolas_0_8 zero
nicolas_0_9 zero
nicolas_1_0 one
nicolas_1_1 one
nicolas_1_10 one
nicolas_1_11 one
nicolas_1_12 one
nicolas_1_13 one
```

Figure 8. text

```
nicolas_1_29 nicolas
nicolas_1_3 nicolas
nicolas_1_30 nicolas
nicolas_1_31 nicolas
nicolas_1_32 nicolas
nicolas_1_33 nicolas
nicolas_1_34 nicolas
nicolas_1_35 nicolas
nicolas_1_36 nicolas
nicolas_1_37 nicolas
nicolas_1_38 nicolas
nicolas_1_39 nicolas
nicolas_1_4 nicolas
nicolas_1_40 nicolas
nicolas_1_41 nicolas
nicolas_1_42 nicolas
nicolas_1_43 nicolas
nicolas_1_44 nicolas
nicolas_1_45 nicolas
nicolas_1_46 nicolas
```

Figure 9. utt2spk

```
one
two
three
four
five
six
seven
eight
nine
zero
```

Figure 10. corpus.txt