Prashant Singh, LNMIIT SVL(LTRC), IIIT Hyderabad

A Report on

Speech recognition on Kaldi and Kaldi-PDNN toolkits using Deep Neural networks

Abstract :

The main objective of the project is implementation of an Automatic Speech Recognition (ASR) system using the Kaldi-ASR toolkit and then improve its performance by extracting the bottleneck features from the data and training the system using tandem neural networks over triphone models. The neural network implementation was carried out using Kaldi-PDNN . The obtained performance of the system by extracting bottleneck features was compared with other neural network based models and the HMM-based models and it was found to give a good improvement over the other methods.

1. INTRODUCTION TO KALDI-ASR TOOLKIT :

Kaldi is a freely available , open-source tool for building Automatic Speech Recognition(ASR)systems, which is written in the object oriented C++.It has modern and flexible code ,which is easy-to-understand ,modify and extend.Inspite of having many popular toolkits for speech recognition such as the HTK toolkit , the Sphinx toolkit ,etc., what makes Kaldi unique is the fact that it makes extensive use of Finite State Transducers(FST) based framework for language and acoustic modelling ,and its extensive support for the use of linear algebra libraries . These are the features of Kaldi which makes it unique :

1.Integration with the Finite state transducers :Kaldi is compiled against FSTs by using the open-source OpenFst framework , by using it as a library .

2.Extensive support for the Linear Algebra libraries : For numerical algebra libraries , we use the BLAS (Basic Linear Algebra Subroutines) package and the LAPACK (Linear Algebra PACKage) package ,which are also openly available .

3.Extensible design:

It has been attempted to provide the algorithms in as generic a form as possible. For instance, our decoders work with an interface that provides a score for a particular frame and FST input symbol. Thus the decoder could work from any suitable source of scores.

The library functionalities can be accessed through command-line tools written in C++, which are then called from a scripting language using shellscripts in Linux for building and running a speech recognizer. Each tool has its specific function or utility with a small set of command line arguments: for example, there are separate shellscripts for accumulating statistics,building language models, extracting MFCC featurres, accumulators, and updating a GMM-based acoustic model and even decoding the graphs and scoring .

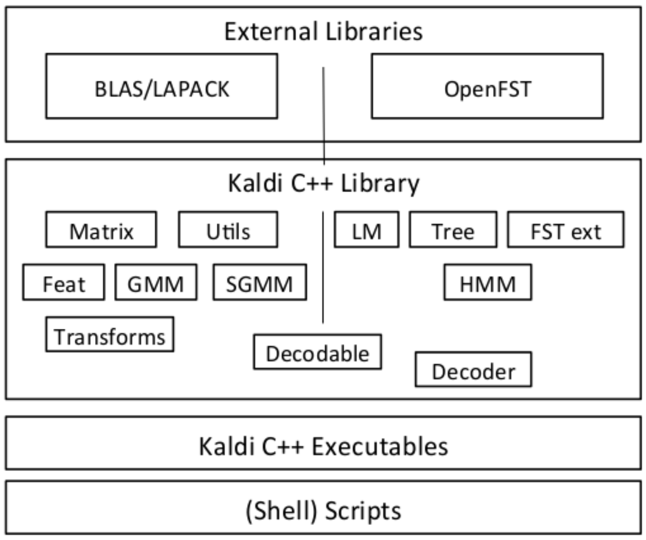


Figure -1.1 :
A simplified view of the different components of Kaldi. The library modules can be grouped into those that depend on linear algebra libraries and those that depend on OpenFst. The decodable class bridges these two halves. Modules that are lower down in the schematic depend on one or more modules that are higher up in the hierarchy as shown in the beside figure ..

2. DIRECTORY STRUCTURE OF KALDI TOOLKIT :

The top-level directories are *src, egs ,tools misc* and *windows* . The directories we will be using are *egs* and *src*.In the main sub-directories are :

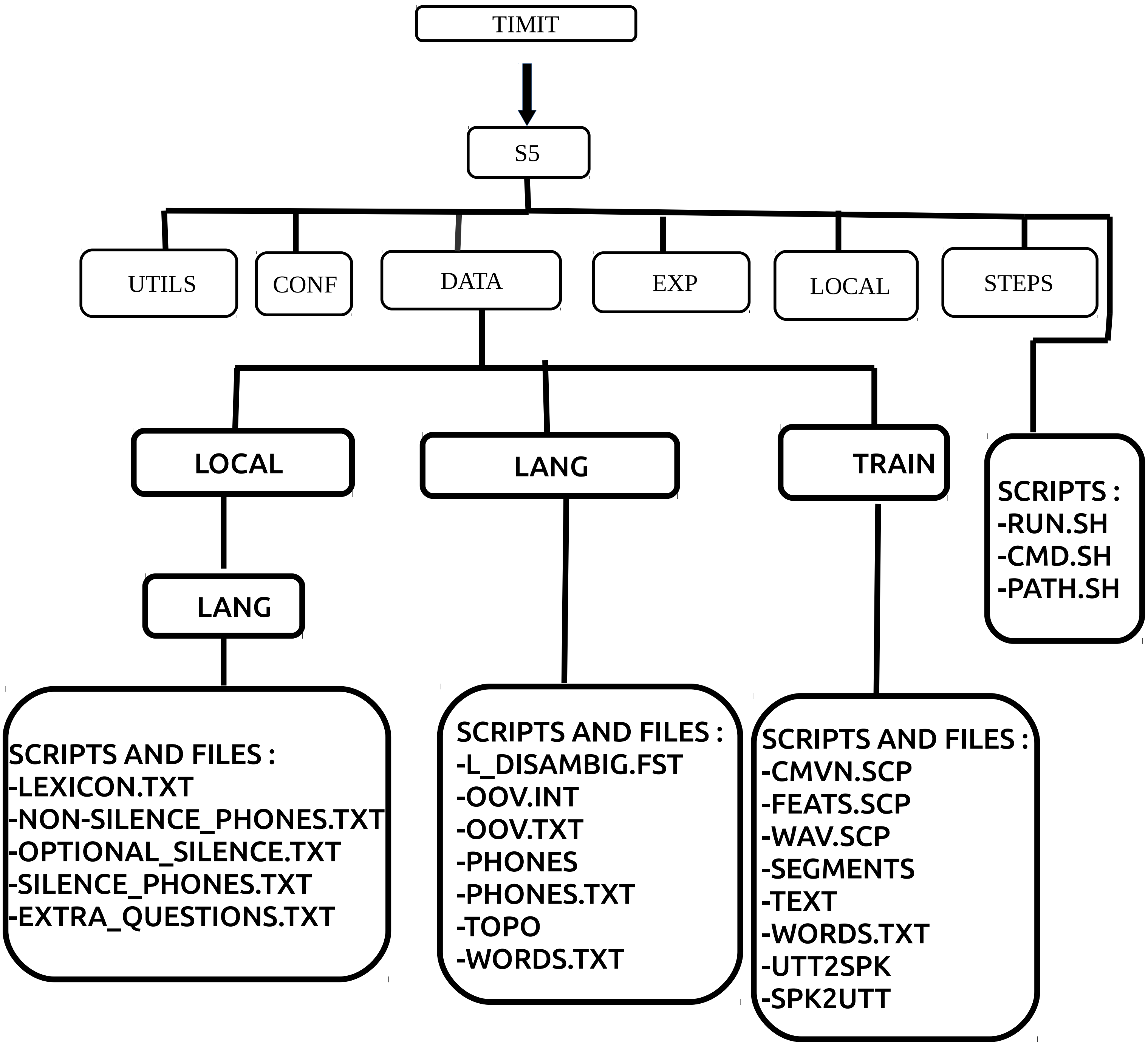
1.The *src* directory ,which means ‘source code’ and contains most of the source code for programs that the training recipes call.

2.The *tools* directory , which contains many tools for converting sphere files to Windows .wav formt , libraries such as ATLAS,CLAPACK,BLAS, IRSTLM,Openfst,SRILM toolkit,toolkit for sph2pipe conversion ,etc .

3.The *egs* directory ,in which Kaldi has many example training database recipes such as TIDIGITS , Wall Street jouranl (WSJ) ,etc.

For each training recipe directory, there is a standard sub-directory structure. Under each of these directories are usually a few different versions (s4 ,s5 , etc.) The highest number, usually , is the most current version and should be used for any new development or training. The older versions are kept for archival purposes only.The higher most directory contains the run script (*run.sh*), as well as two other required scripts (*path.sh* and *cmd.sh*). The sub-directories are *conf* (configuration),*data ,exp* (experiments), *local,steps and utils* (utilities). The directories we will primarily be using are *data* and *exp*. The *data* directory will eventually house information relevant to your own data such as transcripts, dictionaries, training and testing data etc. The *exp* directory will eventually contain the output of the training and alignment scripts, or the acoustic models.All these work is done by running the run.sh shellscript from the command-line ,by placing it in the relevant directory ,which further performs various tasks such as feature extraction , building and compiling graphs, neural network training and decoding of graphs.

Figure-2.1 : Illustration of the training recipe directory structuring :(below)



The above illustration shows the directory structure of the Kaldi recipe for “TIMIT” database which is present in *kaldi/egs* . The data sub-directory to be provided to Kaldi-ASR toolkit consist of mainly 4 folders, namely :
1.*local/lang* folder .
2.lang folder.
3.train folder.
4.test folder (contains files similar to the *train* folder).

There is an optional folder named *dev* folder which may be used ,but it is not mandatory to use the *dev* data and depends on the database recipe being used .Further during training , the training data can be split further into training and cross-validation data in some ratio to ascertain whether the training has taken place in a proper manner and that particular model which gives better accuracy on the *dev* data is used finally for decoding the test data (Note : In this project, the from the training data , 95% of data is used for training the neural network ,whereas the remaining 5% of the data is used for cross-validation .

3. DATABASE : TIMIT database

Texas Instruments/Massachusetts Institute of Technology (TIM IT)is an acoustic-phonetic corpus of read speech was designed to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems. Text corpus design was a joint effort among the Massachusetts Institute of Technology (MIT), SRI International (SRI), and Texas Instruments (TI). The speech was recorded at TI, transcribed at MIT, and has been maintained, verified, and prepared by the National Institute of Standards and Technology (NIST).One of the main reasons for choosing the TIMIT database for this task is the relatively smaller size of the database and as a result , faster processing .

TIM IT contains a total of 6300 utterances, 10 sentences spoken by each of 630 speakers from 8 major dialect regions of the United States. 70% of the speakers are male and 30% are female. The text material in the TIMIT prompts consists of 2 dialect "shibboleth" sentences designed at SRI, 450 phonemically-compact sentences designed at MIT, and 1890 phonetically-diverse sentences selected at TI. Each speaker read the 2 dialect sentences, 5 of the phonemically-compact sentences, and 3 of the phonetically-diverse sentences. The speech material in TIMIT has been subdivided into dialect-balanced portions for training and testing with complete phonemic coverage. A "core" test set contains speech data from 24 speakers, 2 male and 1 female from each dialect region, and a "complete" test set contains 134 4 utterances
spoken by 168 speakers, accounting for about 27% of the total speech

material in the corpus.

The TIMIT corpus includes several transcriptIon files associated with each utterance. These files contain an orthographic transcription, a time-aligned word transcription, and a time-aligned phonetic transcription. The orthographic transcription contains the text of the sentence the speaker said. The orthographic transcription is usually the same as the prompt, but in a few cases they disagree. Word boundaries were assigned using a dynamic programming string alignment program which aligned the word pronunciations found in the lexicon with the phonetic segments.

The lexicon found in the file "/timit/doc/timitdic.txt" contains entries for all of the words in the TIMIT prompts. There are a total of 6229 entries in the dictionary. The lexicon was derived in part from the MIT adapted version of the Merriam-Webster Pocket Dictionary and a preliminary version of a general English dictionary under development at eMU. The pronunciations in the MIT pocket lexicon have been verified and modified over the years.However some of the words present in the TIMIT database could not be found in the dictionary and were subsequently added to the lexicon .

4. Introduction to few basic concepts :

4.1 . Bayes' rule applied to speech recognition:

The following expression describes the Bayes' rule applied to speech recognition.

$$P(\text{utterance}|\text{audio}) = \frac{p(\text{audio}|\text{utterance}) \cdot P(\text{utterance})}{p(\text{audio})}$$

In the above expression , P(occurance) comes from our language model (generally a n-gram model). p(audio | utterance) is a sentence-dependent statistical model of audio production, trained from data .Given a test utterance, we pick ‘utterance’ to maximize *P*(utterance | audio) .Here *p*(audio) is a normalizer that doesn’t matter .

4.2 Markov chains and Markov models :

In simpke terms , A Markov Chain is a model of probabilities of sequences of symbols . A Markov chain (discrete-time Markov chain or DTMC), named after Andrey Markov , is a random process that undergoes transitions from one state to another on a state space . It must possess a property that is usually characterized as “memorylessness” . The probability distribution of the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property . In speech processing applications , In 1st order Markov chain, p(symbol) depends only on previous symbol. For example ,

$$p(\text{ hat }) = p(\text{ h}|?\text{ })\ p\ (\text{ a}|\text{h })\ \ p(\text{ t}|\text{a })$$

In the expression it is written as p(h|?) because we don't know the letter before “h” is used .

While the time parameter is usually discrete, the state space of a Markov chain does not have any generally agreed-on restrictions: the term may refer to a process on an arbitrary state space. However, many applications of Markov chains employ finite or countably infinite (that is, discrete) state spaces, which have a more straightforward statistical analysis. Besides time-index and state-space parameters, there are many other variations, extensions and generalisations. For simplicity, most of this article concentrates on the discrete-time, discrete state-space case, unless mentioned otherwise.

The changes of state of the system are called transitions. The probabilities associated with various state changes are called transition probabilities. The process is characterized by a state space, a transition matrix describing the probabilities of particular transitions, and an initial state (or initial distribution) across the state space. By convention, we assume all possible states and transitions have been included in the definition of the process, so there is always a next state, and the process does not terminate.

A discrete-time random process involves a system which is in a certain state at each step, with the state changing randomly between steps. The steps are often thought of as moments in time, but they can equally well refer to physical distance or any other discrete measurement. Formally, the steps are the integers or natural numbers , and the random process is a mapping of these to states. The Markov property states that the conditional probability distribution for the system at the next step (and in fact at all future steps) depends only on the current state of the system, and not additionally on the state of the system at previous steps.

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states. In simpler Markov models , the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but the output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. herefore, the sequence of

tokens generated by an HMM gives some information about the sequence of states. The adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; the model is still referred to as a 'hidden' Markov model even if these parameters are known exactly .

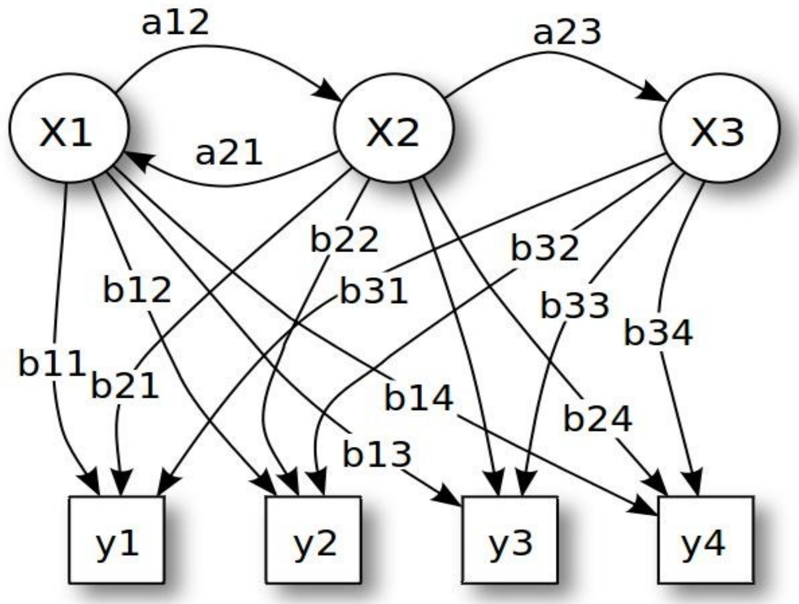


Figure-4.1 : Representation of example probabilistic parameters of a HMM :
Here the symbols denote :
X — states
y — possible observations
a — state transition probabilities
b — output probabilities

Let each spoken word be represented by a sequence of speech vectors or observations O, defined as : $O = o_1, o_2, \ldots, o_T$
where o_t is the speech vector observed at time t. The isolated word recognition problem can then be regarded as that of computing

$$\arg\max \{P(w_i|O)\}$$

where w_i is the i'th vocabulary word. This probability is not computable directly but using Bayes' Rule gives :

$$P(w_i|O) = \underline{P(O|w_i)P(w_i)}$$

P (O)

Thus, for a given set of prior probabilities $P(w_i)$, the most probable spoken word depends only on the likelihood $P(O|w_i)$. Given the dimensionality of the observation sequence O, the direct estimation of the joint conditional probability $P(o_1, o_2, \ldots |w_i)$ from examples of spoken words is not practicable. However, if a parametric model of word production such as a Markov model is assumed, then estimation from data is possible since the problem of estimating the class conditional observation densities $P(O|w_i)$ is replaced by the much simpler problem of estimating the Markov model parameters.It is possible in Kaldi to separately specify the HMM topology for each context-independent phone. The topology format allows nonemitting states, and allows the user to pre- specify tying of the p.d.f.'s in different HMM states.

4.3 Weighted Finite State Transducers(WFST) :

A finite state transducer (FST) is a finite state machine with two tapes: an input tape and an output tape. This contrasts with an ordinary finite state acceptor ,which has a single tape. A FST is a type of finite state automaton which maps between two sets of symbols. A FST is more general than a finite state acceptor (FSA). A FSA defines a formal language by defining a set of accepted strings while a FST defines relations between sets of strings.

(WFSTs) generalize WFSAs by replacing the single transition label by a pair (i, o) of an input label i and an output label o .While a weighted acceptor associates symbol sequences and weights, a WFST associates pairs of symbol sequences and weights, that is, i
t represents a weighted binary relation between symbol sequences Formally, a WFST defined as

$$T = (\Sigma, \Omega, Q, E, i, F, \lambda, \rho)$$

over the semiring K is given by an input alphabet Σ, an output alphabet Ω, a finite set of states Q , a finite set of transitions :
$E \subseteq Q \times (\Sigma \cup \{ \varrho \}) \times (\Omega \cup \{\varrho\}) \times K \times Q$, an initial state $i \in Q$, a set of final states $F \subseteq Q$, an initial weight λ and a final weight function
ρ.A transition $t = (p[t], \ell i[t], \ell o[t], w[t], n[t])$ can be represented by an arc from the source state p [t] to the destination state n [t], with the input label ℓi[t], the output label ,ℓo[t]and the weight w[t].