Automatic Speech Recognition  
Kaldi Toolkit

Ref: [Probability and Statistics](https://docs.google.com/document/d/1gMh2JRQpoNt3K35CJShe5lhTdBMSOBiFRboIECzC1SI/edit?usp=sharing)

horizontal line

# [Overview](https://towardsdatascience.com/how-to-start-with-kaldi-and-speech-recognition-a9b7670ffff6)

## Preprocessing and Feature Extraction

* Models that deal with audio data work with some pixel-based representation of that data which identifies the sound of human speech discarding any noise.
* **MFCC** and **CMVN** are used for representing the content of each audio utterance.
* **I-Vectors** are used for representing the style of each audio utterance or speaker.

## The Model

* The matrix math behind Kaldi is implemented in either BLAS, LAPACK and CUDA.
* Kaldi’s model can be divided into two main components:

### Acoustic Model

* Model will transcribe the audio features that we created into some sequence of context-dependent phonemes.
* **A phoneme** is a unit of sound that can distinguish one word from another. In Kaldi, we call them “**pdf-ids**” and represent them by numbers.  
  **Ex**: /pʊʃ/ represents a sequence of three phonemes, /p/, /ʊ/, /ʃ/ for word “**push**”.
* There are two types of acoustic models:
  + **GMM** - Gaussian Mixture Model
  + **DNN** - Deep Neural Networks

### Decoding Graph

* Turns phonemes into **lattices** (representation of alternative likely word sequences).
* It takes into account the grammar and probability distribution of contiguous specific words (**n-grams**).
* It is essentially a **WFST** (Weighted Finite-State Transducer)
* The composition of different WFSTs is named in the Kaldi project — “**HCLG.fst** file” and it’s based on the **open-fst** framework.

## Training Process

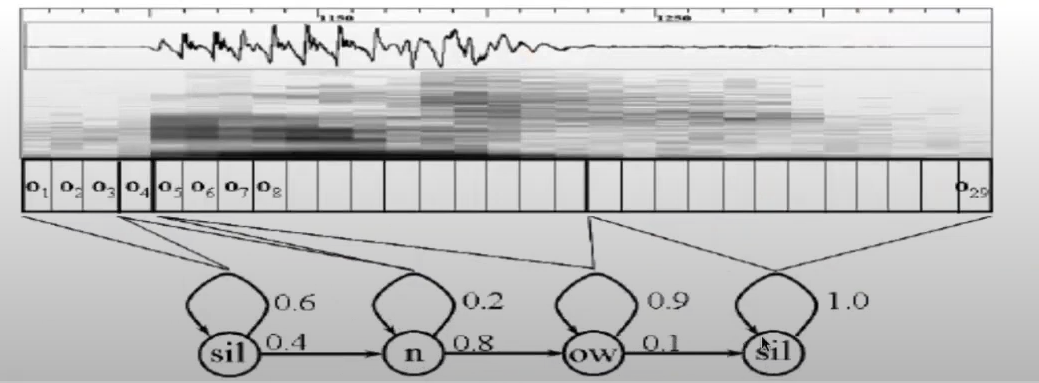
1. Order the transcribed audio data in a really specific order according to the documentation.
2. Create a phoneme dictionary that determines the output of the acoustic model.  
   **Ex:** eight -> ey t  
    five -> f ay v  
    four -> f ao r  
    nine -> n ay n
3. Start training the model using one of the Kaldi recipes. Ex: **WSJ** Recipe
4. In most of the recipes, we are starting with aligning the phonemes into the audio sound with GMM. This basic step (named “alignment”) helps us to determine what is the sequence that we want our DNN to spit out later.
5. After the alignment, we will create the DNN that will form the Acoustic Model, and we will train it to match the alignment output. After creating the acoustic model we can train the WFST to transform the DNN output into the desired lattices.

# [MUCS Conference:](https://www.youtube.com/watch?v=XB7EWu0awSM)

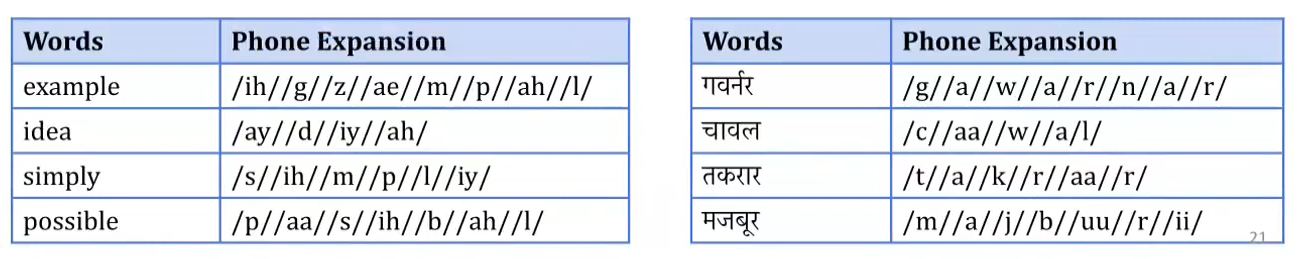
## ASR using models from GMM-HMM to Transformers to wav2vec 2.0

### GMM - HMM

* Every language has a limited number of sounds that make up the pronunciation. Each of these sounds has a unique frequency envelope, which is the preliminary step in every ASR system, also known as Feature Recognition.
* However, for the same sound, there is still a lot of variation in the frequency envelope between individuals. **GMM** is one way to do this. (An arbitrary p.d.f can be approximated by summing N weighted Gaussians)
* The audio waveform series is separated into 10ms intervals, and acoustic vectors (MFCC or Mel Feature Bank Vectors) are computed for each interval. Now we need to figure out what the hidden/underlying phoneme sequence is based on the observation vectors.
* During training, given the phoneme sequence, we calculate the probabilities of the phoneme sequence for the corresponding feature vectors.

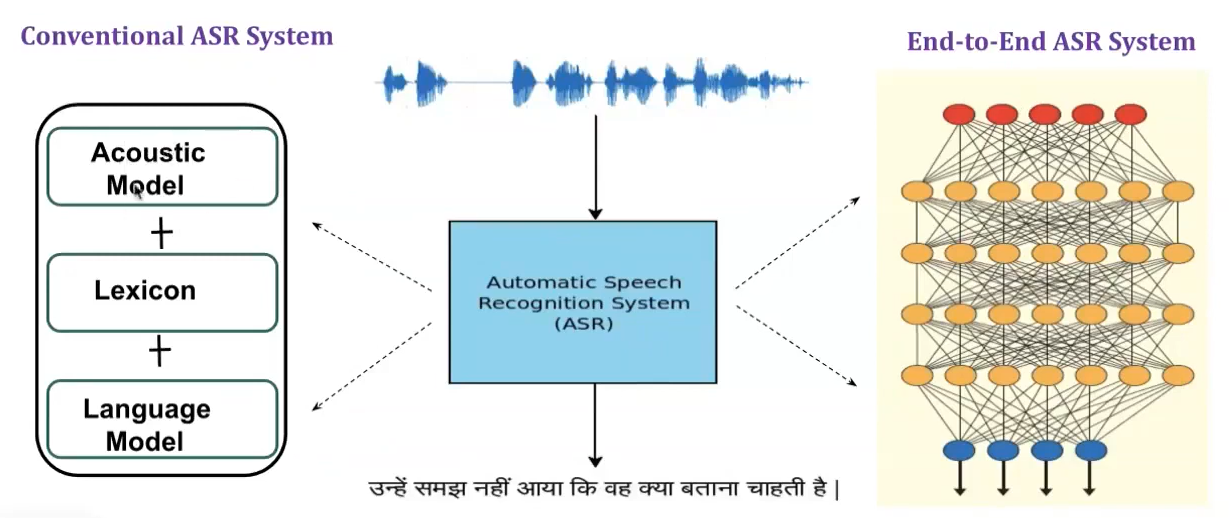


* Now, given the sequence of phonemes, we have to convert them into words using the lexicon look-up dictionary.

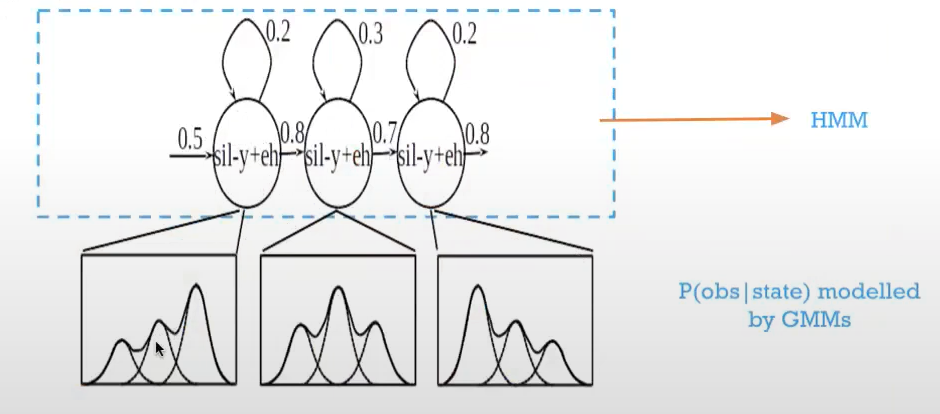


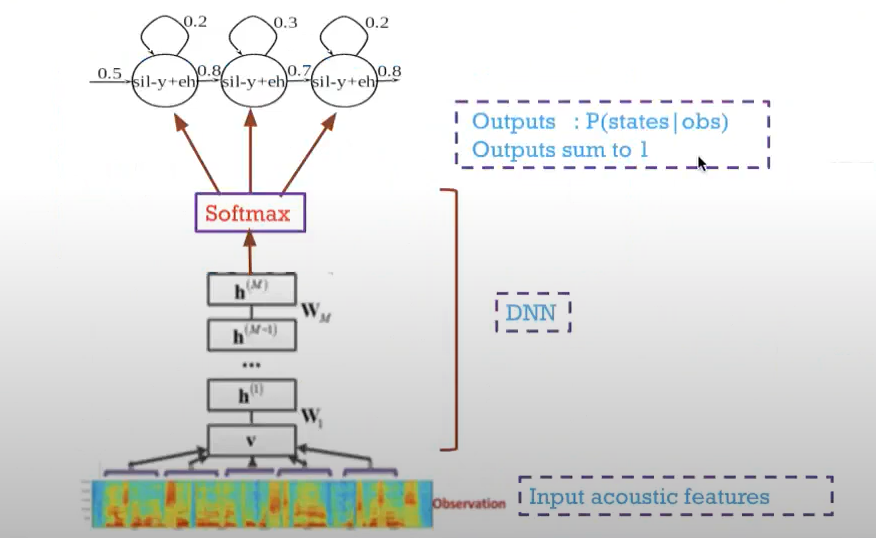
* Once the words are generated, we use **language modelling** to figure out the most meaningful sentence that can be generated using them.
* Finally, the performance is calculated using metrics like **W**ord **E**rror **R**ates (WER)

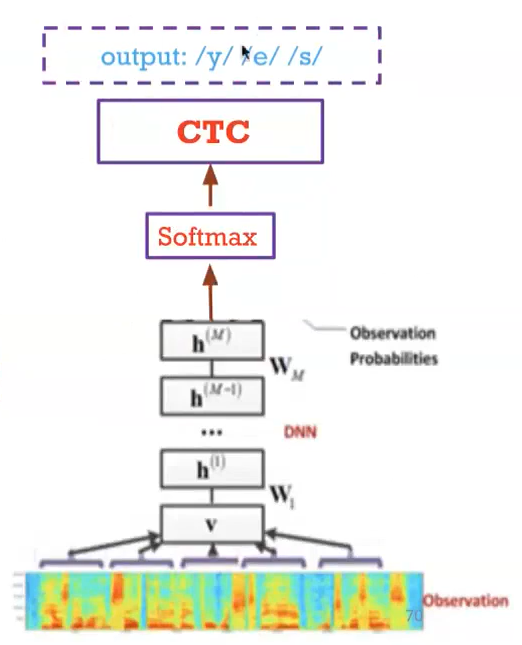
### Deep Learning Models



**Types of DNN models**

**GMM - HMM**

**DNN - HMM**

**End-to-End CTC**

* RNN is basically an extended version of old N-gram language models.