An Introduction to Automatic Speech Recognition (ASR)

Sumit Dugar

Hello, my name is Sumit and I have been working with e-bot7 for the last couple of months as a data scientist. Most of my background in machine learning has been in image processing and NLP. But Since last few weeks I have been studying about speech recognition. The goal of this presentation is to give a short introduction about automatic speech recognition to get you started quickly..

e-bot7

Outline

- Introduction
- Classical ASR Pipeline
  - Input Audio Signal
  - Signal Processing and Feature Extraction
  - Acoustic Model
  - Hypothesis Search
  - Language Model
- Evaluation Metric
- End to End Approach
  - LAS - Listen Attend Spell

This is the outline for today's presentation. We are mainly going to focus on the classical ASR pipeline and in this we will be talking about what audio signals are? We will discuss a common feature extraction technique that is used with audio data and we will also talk about other components of the pipeline like Acoustic Model, Language Model and Hypothesis Search. Towards the end we will see an evaluation metric that is used to compare the performance of various ASR models and then we will shortly discuss how an end to end approach works.

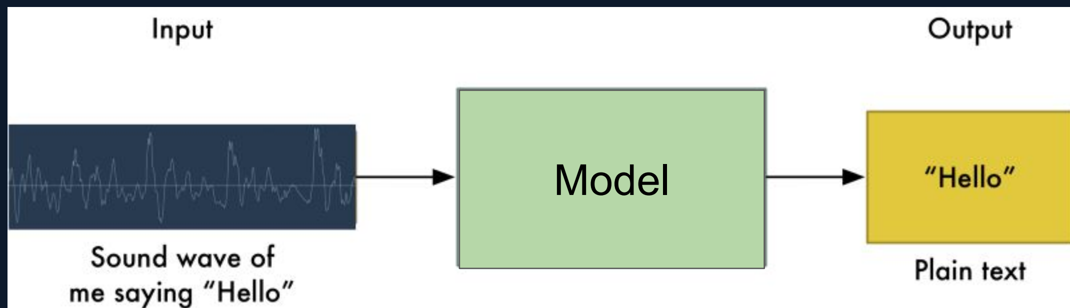
Ok so lets start.

e-bot7

Introduction

What is ASR?

It is the task of detecting spoken words from an audio signal.



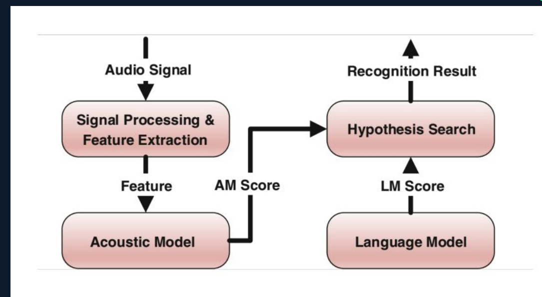
Source - https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

Automatic speech recognition is the task of detecting spoken words from an audio signal. The input to the ASR pipeline or model is a sequence of features extracted from audio files and the output is simple plain text.

e-bot7

Classical ASR Pipeline

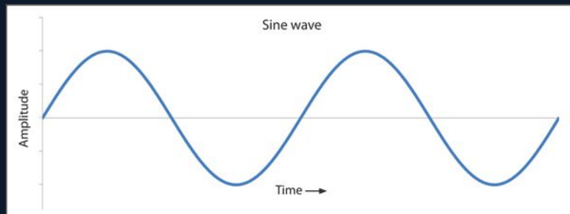
1. **Signal processing and feature extraction** - cleans the audio signal and extracts feature vectors.
2. **Acoustic model (AM)** - estimates the probability of a hypothesized audio sequence.
3. **Language model (LM)** - estimates the probability of a hypothesized word/character sequence.
4. **Hypothesis search** - combines AM and LM scores and outputs the word sequence with the highest score.



Source - http://125.234.102.146:8080/dspace/bitstream/DNULIB_52011/6853/1/automatic_speech_recognition_a_deep_learning_approach.pdf

For now lets assume that the neural network is not a single end-to-end network but instead a pipeline consisting of these 4 components.

What is Sound?



Source - https://towardsdatascience.com/ok-google-how-to-do-speech-recognition-f77b5d7cbe0b
Source - https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

Sound is a longitudinal pressure wave formed of compressions and rarefactions of air molecules in a direction parallel to that of the application of energy. Compressions are zones where air molecules have been forced by the application of energy into a tighter-than-usual configuration, and rarefactions are zones where air molecules are less tightly packed.

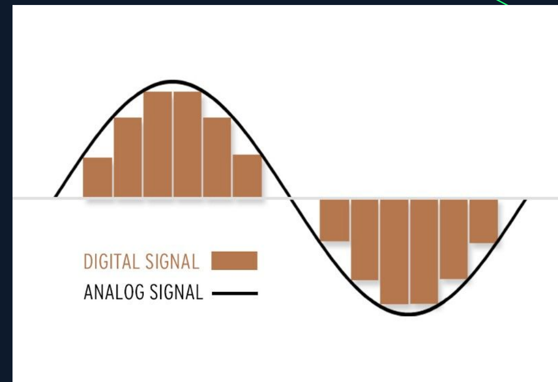
Sound is represented as waves. Generally, these waves have 2 axes. Time is represented on the x-axis and Amplitude on the y-axis. So at every instant of time t, we have a value for amplitude.

The sounds we hear generally are mixture of multiple sound waves of different frequencies. See the second fig.

Analog to Digital Signal

e-bot7

- Audio is a continuous analog signal.
- **Sampling Rate** - number of samples collected per second.
- Some common sampling rates are:
  - 44.1KHz
  - 16KHz
- Audio files such as .wav files contains discrete digital signals.
- Use python package — librosa for reading .wav files.



Source - https://towardsdatascience.com/ok-google-how-to-do-speech-recognition-f77b5d7cbe0b

Audio is a continuous analog signal. What this means is that it has some amplitude value at every instant of time aka it has some value for every nanosecond, or maybe every picosecond. We don't want to store that amount of information so we convert it into discrete form. To convert it into discrete form, we record samples (aka the amplitude values) at every time step. So for a 5-second audio, we can record samples at every 1 second. That's just 5 values (samples)! This is called the Sampling Rate.

For speech recognition, a sampling rate of 16khz is enough to cover the frequency range of human speech.

Raw audio files such as .wav files contains this discrete digital signals. A python package like librosa can be used for reading these .wav files.

Signal Processing and Feature Extraction

e-bot7

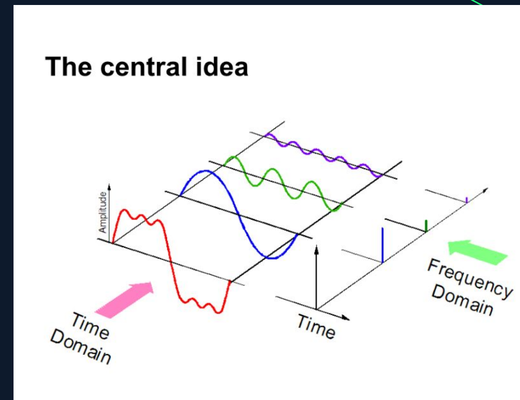
- **Chunking** - Grouping our sampled audio into 20-millisecond-long chunks.
- So now the audio is a sequence of 20-millisecond-long chunks or audio frames.
- **Fourier Series** - Is a way to represent a function as the sum of simple sine/cosine waves.

We now have an array of numbers with each number representing the sound wave's amplitude at 1/16,000th of a second intervals. We could feed these numbers right into a neural network. But trying to recognize speech patterns by processing these samples directly is difficult and computationally expensive. Instead, we can make the problem easier by doing some pre-processing on the audio data.

Let's start by grouping our sampled audio into 20-millisecond-long chunks. So now the audio is a sequence of 20ms chunks. But even this short recording is a complex mixture of different frequencies of sound. To make this data easier for a neural network to process, we are going to break apart this complex sound wave into its component frequency parts.

Fourier Transform

- **Fourier Series** - Is a way to represent a function as the sum of simple sine/cosine waves.
- **Fourier Transform** - It uses Fourier Series to convert our time-vs-amplitude signal to time-vs-frequency signal.
  - Requires less storage space.
  - Sort of a frequency fingerprint for audio snippet.
- **Spectrogram** - If we repeat this process on every 20 millisecond chunk of audio, we end up with a spectrogram.



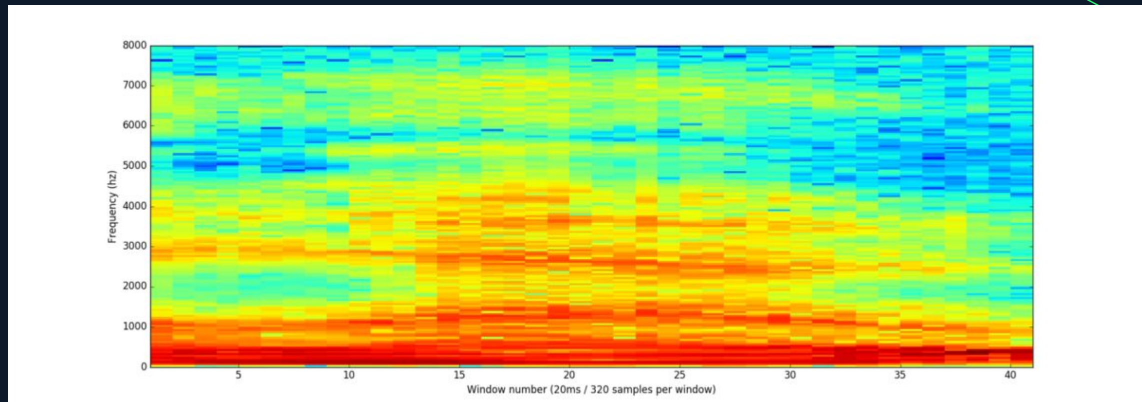
Source - https://towardsdatascience.com/ok-google-how-to-do-speech-recognition-f77b5d7cbe0b

We do this using a mathematical operation called a Fourier transform. It breaks apart the complex sound wave into the simple sound waves that make it up. Once we have those individual sound waves, we add up how much energy is contained in each one. We create a fingerprint of sorts for this audio snippet.

If we repeat this process on every 20 millisecond chunk of audio, we end up with a spectrogram.

e-bot7

Spectrogram

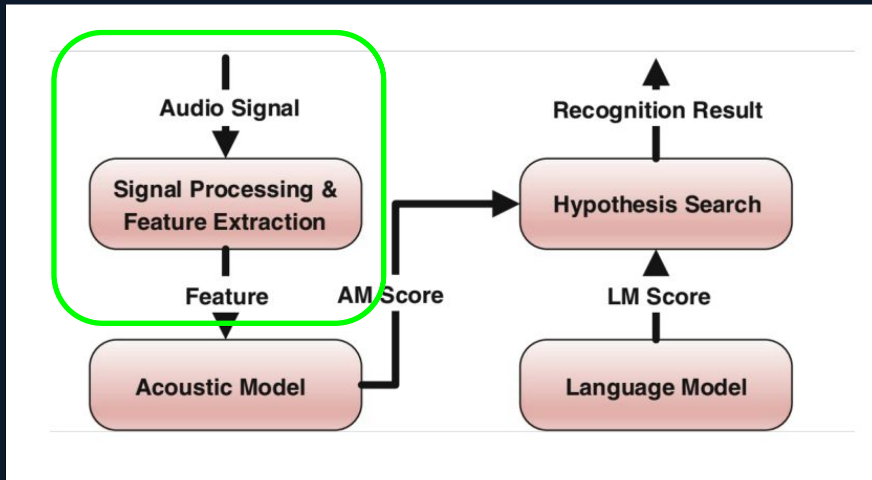


Source - https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

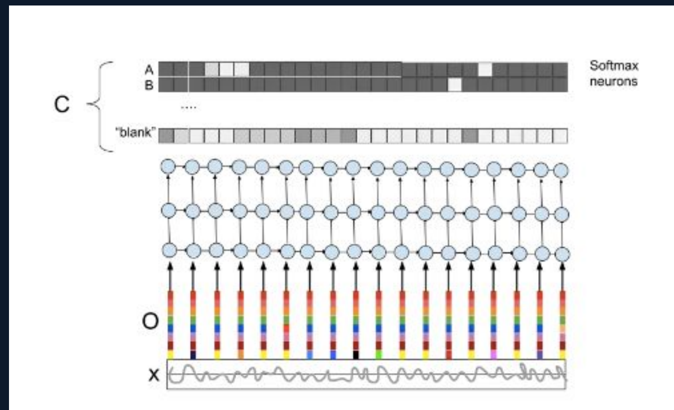
On the x-axis are time chunks or windows and on the y-axis is frequency. Each column is a feature vector of a 20ms chunk. The amplitude of the spectrum at a particular frequency for a particular chunk of time is represented as the color intensity at that point. Red denotes higher intensities and blue denotes lower intensities. Each element represents sort of the strengths of each frequency in this little window. You can see that this sound snippet has a lot of low-frequency energy and not much intensity in the higher frequencies. That’s typical of “male” voices.

A spectrogram is cool because you can actually see musical notes and other patterns in audio data. A neural network can find patterns in this kind of data more easily than raw sound waves. So this is the data representation we’ll actually feed into our neural network.

Recap



Acoustic Model



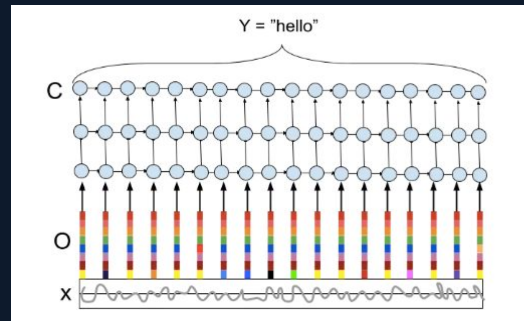
Source - https://androidkt.com////recognition-encoding/

Now that we have our audio in a format that's easy to process, we will feed it into a deep neural network. The input to the neural network will be a sequence of 20 millisecond audio chunks. For each little audio slice, it will try to figure out the letter that corresponds the sound currently being spoken.

Acoustic Model - Challenges

e-bot7

- Length of the input sequence is not the same as the length of the transcription sequence.
  - Use a dataset where labels are chunkwise. Each chunk is mapped either to valid character from the transcript or to an empty character.
    - Labelling is time consuming (boring) and error prone.
    - A single character spanning multiple chunks.
  - Connectionist Temporal Classification (CTC).



Source - https://androidkt.com////recognition-encoding/

One fundamental problem is that the length of the input is not the same as the length of the transcription. If I say “hello” very slowly then I can have a very long audio signal even though I didn’t change the length of the transcription or if I say hello very quickly then I have a very short transcription or a very short piece of audio.

One way to overcome this is to have a dataset that is patiently labelled chunk wise. For chunks which do not correspond to any alphabet we can label them with some empty character. One big problem with this solution is that this kind of labelling is very time consuming, boring and error prone. Another issue with this approach is that a single character can span multiple chunks e.g. we could get “ ttooo” because the “o” is spoken for a longer duration.

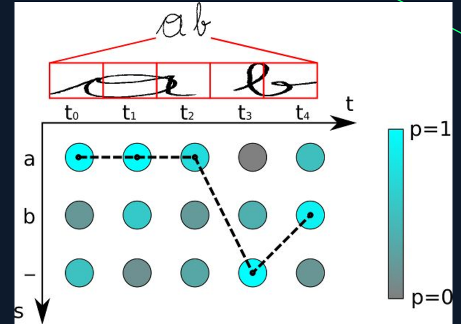
Connectionist Temporal Classification solves both of these problems for us.

Connectionist Temporal Classification (CTC)

e-bot7

It is a loss that solves both problems for us. With CTC

- we only have to tell the CTC loss function the transcribed ground truth that occurs in the audio. Therefore we ignore both the position and length of the audio and transcription.
- No need to worry about single character spanning over multiple windows as CTC performs a squeezing operation.
    - AAA-B = AB



Source - https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

So what is connectionist temporal classification? It is a kind of loss. It requires the transcribed ground truth that occurs in the audio sequence and no one to one mapping between audio frames and transcribed characters is required. So we don't need to worry about the length of audio and transcription and their alignment.

Another thing that CTC does is it uses a squeezing operation while computing transcription paths. The aim of this squeezing operation is to remove duplicate characters and unnecessary blanks. Because of this we don't need to worry about single character spanning multiple audio frames.

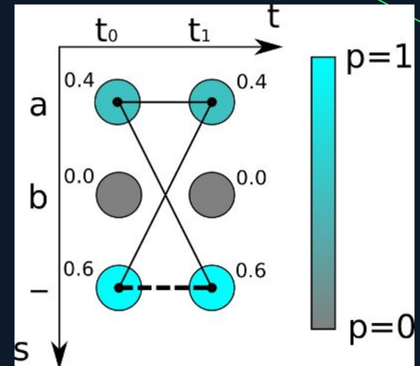
If you see the image on the right the character A is spanning 3 audio frames and then there's a blank and then B. After this squeezing operation all the duplicate As will be removed and then blanks will be removed and we will be left with only AB.

e-bot7

Connectionist Temporal Classification (CTC)

How CTC works?

1. Compute all the possible paths to the ground truth transcription.
   a. AA = A
   b. A- = A
   c. -A = A
2. Compute the scores for these paths by multiplying the corresponding character scores together.
   a. AA = 0.4*0.4 = 0.16
   b. A- = 0.4*0.6 = 0.24
   c. -A = 0.6*0.4 = 0.24



Source - https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

Now lets try and understand how this CTC loss is computed? And How does it know where each character occurs in the audio frame? Well, it does not know. Instead, it computes all possible paths/alignments of the ground truth transcription that can be reached from the AM output matrix and takes the sum scores of these paths. In this way, the models learns to predict those characters that will increase the overall score of a ground truth transcriptions.

Lets try to understand this with a simple example. This figure shows the softmax output of an AM model for an input sequence of 2 audio frames. The model outputs a probability distribution over 3 alphabets - a,b,empty. Let the ground truth transcription be a.

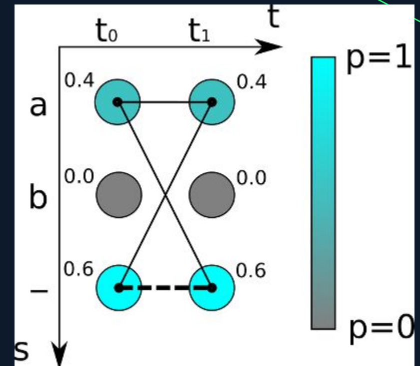
So the first step is to compute all the possible path to the ground truth transcription. In this example we have 3 possible paths. Next step is compute the scores of each of these paths. For this just multiply the probability of each character being predicted at that time step.

Connectionist Temporal Classification (CTC)

e-bot7

How CTC works?

3. Calculate the score for a given ground truth transcription by summing over the scores of all paths corresponding to this transcription.
   a. A = 0.16 + 0.24 + 0.24 = 0.64
   b. This is the likelihood probability of the ground truth.
4. To compute the loss just take the negative log of the likelihood probability computed in the previous step.



Tensorflow provides the implementation of CTC loss.

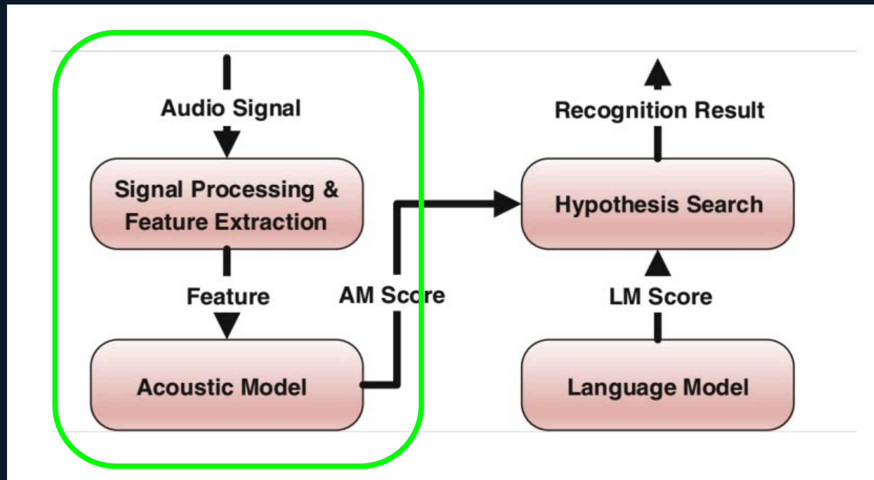
Source - https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c

Next step is to compute the overall score of the ground truth transcription. For this just add the individual scores that you computed in the previous step. This score denotes the likelihood probability of the ground truth. Now to compute the loss just take the negative log of this likelihood.

Almost all the deep learning frameworks provide off the shelf implementation of CTC loss.

So using this loss we can comfortably train our acoustic model without being worried about the alignment of input audio and ground truth transcription.

Recap

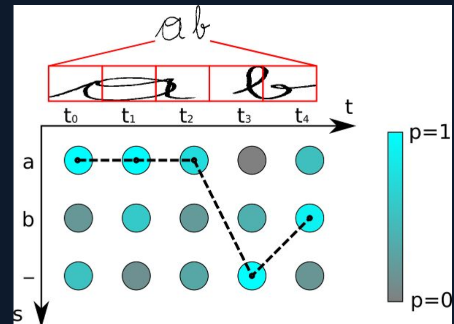


Inference & Hypothesis Search

How to use the Acoustic Model for Inference?

A simple and very fast algorithm is **max decoding** which consists of two steps:

1. It calculates the best path by taking the most likely character per time-step.
2. It then performs a squeeze operation on the encoding by first removing duplicate characters and then removing all blanks from the path. What remains represents the recognized transcription.



Now assuming that the acoustic model is trained how can we generate transcription from this?

So far we know using CTC we can calculate the scores or paths to a given ground truth transcription. But during inference no ground truth transcriptions are available. So how can we generate the transcriptions? One way would be to compute all the path and then just choose the one with the best score but it is combinatorially very expensive.

A simple and very fast algorithm is max decoding which consists of two steps.

It calculates the best path by taking the most likely character per time-step. That is for each time step simply choose the character which has the highest probability. Once you have done this for each time step then simply use the squeeze operation.
It undoes the encoding by first removing duplicate characters and then removing all blanks from the path. What remains represents the recognized transcription.

It is only an approximate algorithm and it might always not give the best results. Actually just using this often gives very terrible results.

There are no efficient solution in general. People usually use generic search solution like - Prefix beam search instead of max decoding. We are not going to go into the details of beam search instead we will focus on another issue that arises even if you use beam search or any other hypothesis search techniques.

Language Model

e-bot7

RNN output	Decoded Transcription
what is the weather like in bostin right now	what is the weather like in boston right now
prime miniter nerenr modi	prime minister narendra modi
arther n tickets for the game	are there any tickets for the game

Table 1: Examples of transcriptions directly from the RNN (left) with errors that are fixed by addition of a language model (right).

- Even with better decoding, CTC models tend to make spelling + linguistic errors.
  - Independent assumption.
  - Not enough audio data to learn complicated spelling and grammatical structure.
  - Very small vocabulary.

Source - https://arxiv.org/pdf/1412.5567.pdf

So the problem is that even with better decoding CTC models make spelling and grammar mistakes. The left table shows the transcriptions from a CTC model. You can see that their are some spelling mistakes for example the spelling of boston is wrong. Some reasons why CTC has these issues are
It assumes that each time step is independent of each other. Then there's not enough audio data to learn complicated spellings or names of persons or places and due to this the effective vocabulary of the acoustic model is not big.

The right table shows the results that were refined using a language model.

Language Model

A possible solution is to train a language model on massive text corpora.

- Learn spelling + grammar
- Greatly expanded vocabulary
- Contextual bias

$$Q(c) = \log(\mathbb{P}(c|x)) + \alpha \log(\mathbb{P}_{\text{lm}}(c)) + \beta \,\text{word_count}(c)$$

The aim here is to find a sequence c that maximizes the combined objective Q(c).

Source - https://arxiv.org/pdf/1412.5567.pdf

Language models can be really useful in this regard. Since textual data is comparatively easily available and there are a lot of unsupervised algorithms for training language models so it is easy to train a language model. A Language model is capable of learning spelling and grammar rules and because of huge textual data it has a big vocabulary too.

Because of all these reasons it makes a lot of sense to use a language model to refine the results of acoustic model. With help of this equation we combine the scores of acoustic and language model to get the final prediction score for a character. Alpha and beta are regularizations parameters to control the influence of language model and sequence length.

Evaluation Metric - Word Error Rate (WER)

e-bot7

$$WER = \frac{S + D + I}{N}$$

Reference Audio : What a bright day

- **S (Substitutions)** : Number of words replaced.
  - What a light day
- **I (Insertions)** : Number of words that got inserted but were not said.
  - What a beautiful bright day
- **D (Deletions)** : Number of words that got omitted but were said.
  - What a day
- **N** : Total number of spoken words in the reference audio.

To measure the performance of ASR systems we use word error rate as the evaluation metric. It is defined as

Cons of Pipeline Approach

e-bot7

Cons of pipeline approach for ASR

- A lot of manual effort is required in designing these pipelines.
  - Audio and transcription alignment with CTC
  - Hypothesis search using prefix beam search
- Each component of the pipeline has to be individually trained.
  - Acoustic Model
  - Language Model

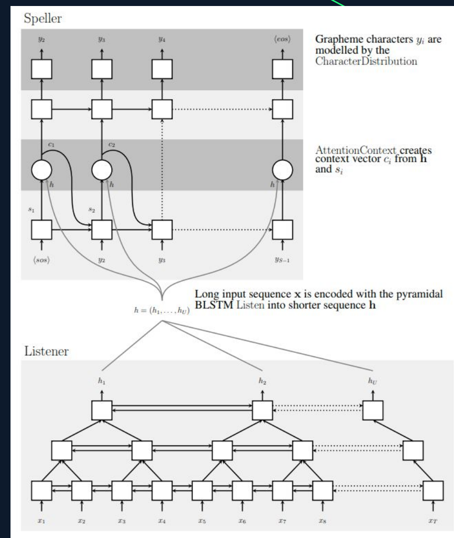
So far we saw a pipeline based approach for automatic speech recognition. There are some pain points or bottlenecks of these pipeline based approaches. A lot of manual effort goes in designing these pipeline and making them robust specially for edge cases. For instance we saw that we needed CTC algorithm for aligning audio and ground truth transcriptions. Then for decoding or inference we need to use some generic search algorithms like prefix beam search and even after this the results were not perfect so we refined them with the help of language model.

So there are so many components in this pipeline and each of these components has to be individually trained.

One way to avoid all this is to use End-to-End approaches. In recent times a lot of focus has moved towards the end-to-end approaches. Performance gap between pipeline and end-to-end approaches has become very narrow now and many state of the art methods are trained end-to-end.

Listen, Attend and Spell (LAS)

- The model uses a Seq-Seq Encoder-Decoder approach.
- Encoder (Listener) is a Pyramidal BiLSTM network.
- Decoder (Speller) is an attention based LSTM network.
- 40000 hours of data.
- A month of training.
- WER of 14% (without language model)
- WER of 10% (with language model)



Source - https://arxiv.org/pdf/1508.01211.pdf

Listen, Attend Spell is an end-to-end model that learns to transcribe speech utterances to characters. It was published in 2015 by google. It follows a sequence to sequence architecture and uses an encoder-decoder approach. Unlike pipeline based approach this model doesn't need to worry about the alignment of audio and ground truth transcriptions and it doesnt need to worry about single character spanning multiple audio frames. All this is inherently handled by the encoder-decoder architecture.

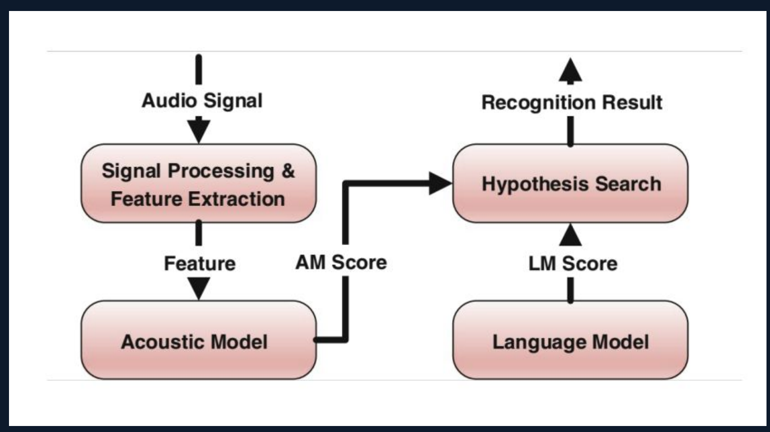
The input is a sequence of feature vectors and the output is also a sequence of characters. These sequence need not be of similar length. The first part if the encoder part which is also called the listener. It is a pyramidal BiLSTM network. The goal of the listener is to encode the sequence of input features into a smaller sequence of condensed representations.

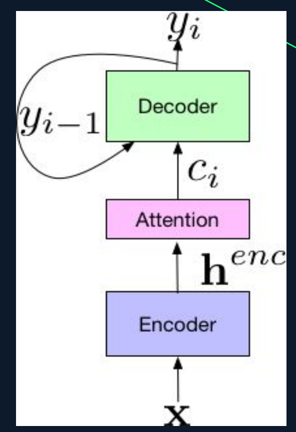
These condensed representations are then used by the Decoder for generating the transcriptions. The decoder is an attention based LSTM network. For each timestep it takes the previous LSTM state as input along with the previous predicted characters and also the context from the attention layer. This way end-to-end models do not make an independent assumption about the predicted characters which the CTC model was assuming.

These models are huge and takes a lot of data, computing power and time to train. This particular model was trained on 40000 hours of data. It took almost 1 month for this model to converge.

It achieved an WER rate of 10% with language model refinement and an WER of 14% without any language model refinement.

Summary





e-bot7

Thank You!

Questions?