Fundamentals of an ASR system

What is the task of Automatic Speech Recognition (ASR)?

Create an automatic transcription (in any given token format) of speech given an audio file containing (among other things) said speech.
or to put it a bit more Shannon
Decrypt the original **text** (in any given token format) that has been **encrypted as an audio recording**.

This task does **NOT** involve any of the following:

- Understanding or making use of the *meaning* of words
- Enforcing or even assuming a grammatical structure in the speech of an assumed language (e.g. English)
- Creating well-formed text in any natural language
- Inserting words/tokens that have not been said (e.g. punctuation)
- Censoring anything that has been said.

This task **may** involve the following:

- Providing more than just one textual representation of the speech contained in the audio
    - n-best list (multiple options for a given file)
    - lattice (a graph representation of the complete search result of the system)
- Providing additional information regarding the actual pronunciation of a token
- Providing para-linguistic information
    - cough, sneeze, humming, noise or other markings
- Adaption to
    - individual speakers
    - acoustic conditions
- Detection of voice activity and speech endpointing
- Re-scoring / re-ranking outputs based on external context information / world knowledge

This task **does** involve the following:

- Dealing with different types of speech
    - Command and Control versus Free From versus Conversational
- Dealing with different domains of speech
    - Sports commentary versus Home Automation
- Dealing with different accents and even dialects of speech
    - Scottish versus Queens English versus Texan
    - Austrian German versus High German
- Dealing with different speech patterns
    - Women versus Man versus Children (think of the pitch of the voice)

    - Different vocal tracts shapes
- Dealing with conversational speech artifacts
    - False starts
    - Self corrections
    - Hesitations
- Dealing with any kind of background speech or noise patterns <each of them might require a purpose build model to deal with the noise patterns>
    - Cocktail party
    - New York Times Square
    - Honking trucks in India
    - ...

Which parts constitute an ASR system?

We start with the ASR problem

$P(X,W) = P(X|W)P(W)$

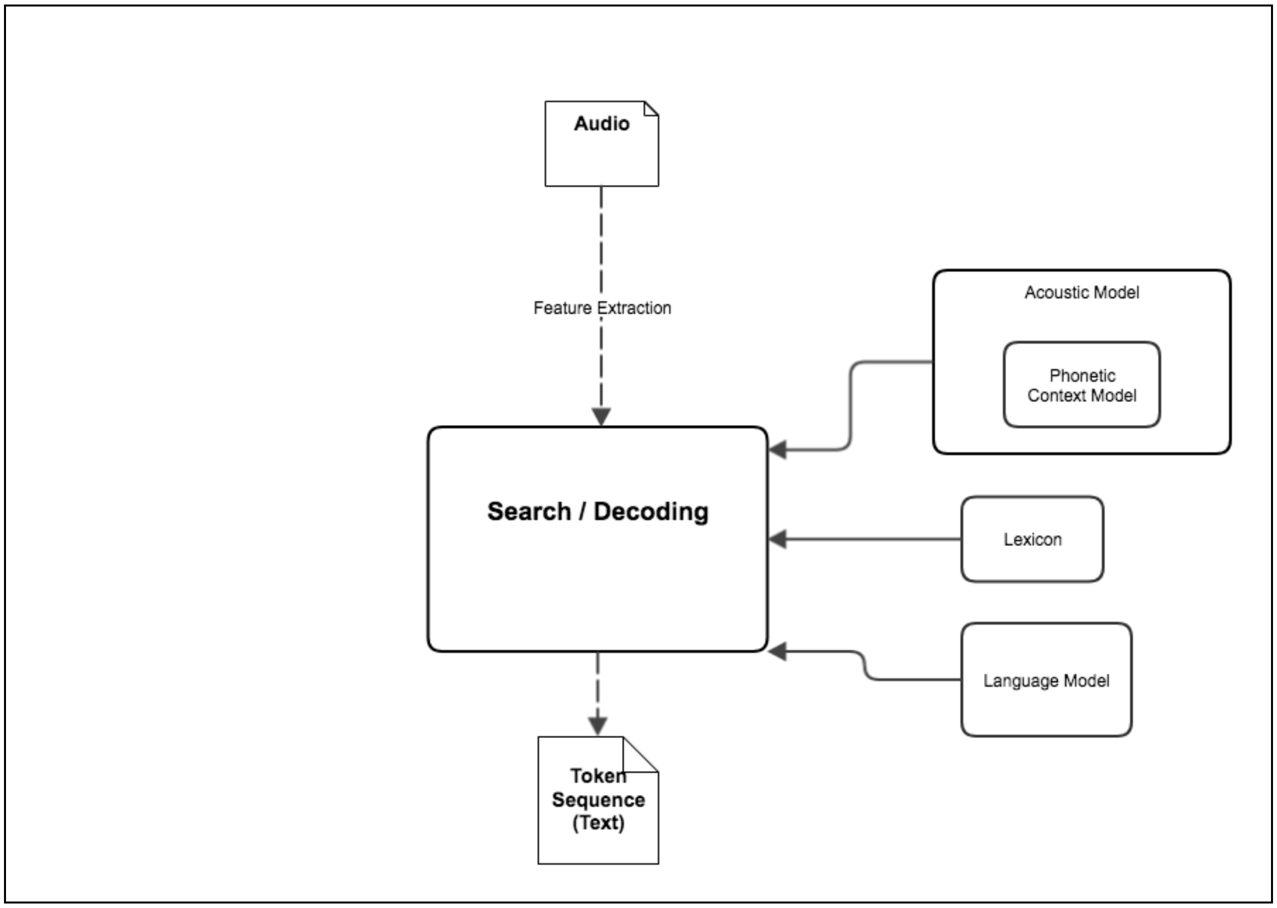
$P(X,W) = P(X|S)P(S|H)P(H|L)P(L|W)P(W)$

In the classical, factorized, approach, an ASR system is split into 5 individual parts:

- Acoustic Model [X->S]
- Phonetic Context Model (state tying/ tri-phones, quint-phones etc) [S->H->L]
- Pronunciation Lexicon (Lexicon for short) [L->W]
- Language Model [W]
- Search / Decoding (take your pick on the name)

The first four together form the ASR system while the fifth one enables us to actually recognize the tokens in an audio file by exploring the search space spanned by the model and the audio file.

We refer to the search part also as decoding (see Shannon) and consequently to the speech recognition engine as decoder.



Acoustic Model

- Models "how", on average, a certain word in our token language "sounds".
- Problem:
  - To train this we would need to "see" every word in a language sufficiently often (several hundred occurrences, by multiple speakers)
  - English alone has >500k words
  - Added bonus: The pronunciation depends on the context of the word e.g. and in "Mac and Cheese" sounds differently from "somehow and somewhere"
- Observation:
  - Any language has a limited set of sounds it uses to form words
  - Those sounds (phones) are shared across words
  - The pronunciation is not set in stone but follows a certain distribution → phonemes
  - Example:
    - US English words: "dance" and "can" use the same "a" sound;
    - US English words: "fish" and "enough" share the "f" phoneme in the starting "f" of fish and the "gh" in enough

    - The characteristic pronunciation of a phoneme depends on the context → Model phonemes in context
        - Example: Let the acoustic model learn a representation for e_n_ou instead of just n
- In terms of math the acoustic model represents the probability: P(x[1:T]|w[1:N]) where x[1:T] is a sequence of features for T time steps and w[1:N] is the sequence of words/tokens encoded in the features sequence, N <= T and unknown.

We need to solve 2 problems in the acoustic model to train/decode with it:

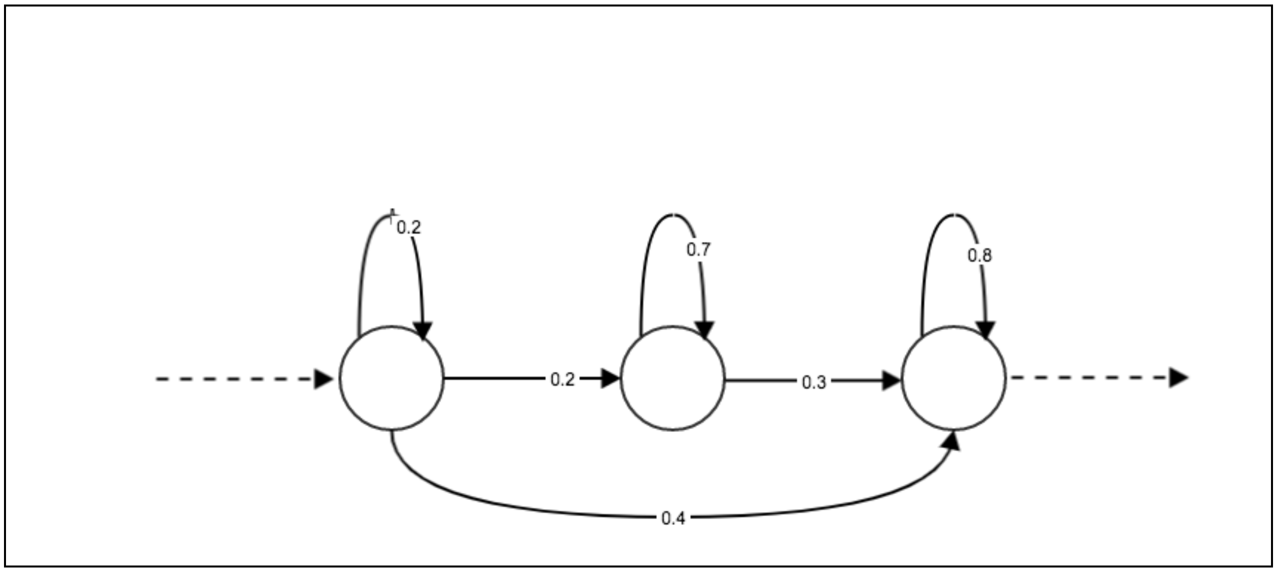
1. How an individual phoneme in context sounds
2. How to deal with the sequence of phonemes and the fact that people speak in different speeds

Issue 1):

- Create a forced alignment on training data and keep the association of features to phonemes in context fix
- Use a GMM or Neural Network to discriminate distribution

Issue 2):

- Use a Hidden Markov Model (**HMM**) to deal with difference in speaking rate
    - Finite state automaton per phoneme, usually 3 states with learned transition probabilities on the arcs. This is an acceptor
- Use the posteriors from the GMM or DNN as state emission probabilities in the HMM



Math of the acoustic model as HMM (minus how to convert the LSTM posteriors and plus first order HMM assumptions)

$$p(x_1^T|w_1^N) \approx \max_{[s_1^T]} \sum_{t=1}^{T} \left\{ p(x_t|s_t, w_1^N) \cdot p(s_t|s_{t-1}, w_1^N) \right\}$$

Phonetic Context Model / State Tying

- Automatically learn a model that "ties"/ clusters phones into tri-phones depending on data
- Share distributions across tri-phones that are similar to keep the number of posterior models in check
- Achieved via training a CART (Classification And Regression Tree) on forced alignment data
- Provides a mapping from "tri-phone" to monophone/phoneme

Lexicon

- The link between tokens (words) and their pronunciation

Example (beware French):

```
parios  p_B a R i j o_E
parios  p_B a R j o s_E
parios  p_B a R j o_E
pariot  p_B a R i j O tS_E
pariot  p_B a R i j O t_E
pariot  p_B a R i j o_E
pariot  p_B a R j o_E
pariota p_B a R i j O t a_E
pariota p_B a R j o t a_E
pariou  p_B E R j u_E
pariou  p_B a R i j u_E
pariou  p_B a R j u_E
paripenné       p_B E R i p E n e_E
paripenné       p_B a R i p E n e_E
paripenné       p_B a r\ i p E n e_E
paris   p_B a R i s_E
paris   p_B a R i_E
```

- Each word/token can have multiple pronunciations
- The **recognition** lexicon (lexicon.txt in the final package) is composed of:
- g2p:
    - grapheme:
        - "smallest textual representation of a particular sound in a particular language"
        - e.g: "sch" in school

- The g2p system
  - learned sequence to sequence model which has words/tokens as input and phonemes as output

Language Model

- Purely text based
- Models the probability P(w[n]| h = w[1:n-1]) of a word w[n] appearing after the sequence of words w[1:N-1]
  - e.g. P("Earl"| "name", "is", "my")
- Typically build and evaluated in isolation from the other components (Acoustic model etc.)
- Approaches:
  - Classical, well established and in production: Token frequency based n-gram models
    - ~discrete state space
  - Neural: Train a neural network to predict words given history
    - ~continuous state space
- **Not a human written grammar for any given language!**
  - ASR does not care one bit about what is proper speech and grammar. We are all about how people speak and write not about how the should be doing it.

Search / Decoding

All of the above models are fine and dandy but useless if we can't decode a given audio file.

To get to the best recognition result in a "Viterbi" manner (just the best for now), we need to solve:

$$x_1^T \mapsto \hat{w}(x_1^T) \approx arg \max_{w_1^N} \left\{ p(w_1^N) \cdot \max_{[s_1^T]} \sum_{t=1}^{T} \left\{ p(x_t|s_t, w_1^N) \cdot p(s_t|s_{t-1}, w_1^N) \right\} \right\}$$

So what is hard about this?

- Max over “s” implies finding the best path out of all possible paths -> huge number of paths
- N is unknown as are t[n] the time points where w[n] ends and w[n+1] starts
  - i.e. we are not just checking for which words are in the sequence but also where they are in the time span
- Huge potential search space, getting worse and worse with the length of the audio
- Languages have real prefixes: e.g. "childproof" → "child" is a word of its own

Observations:

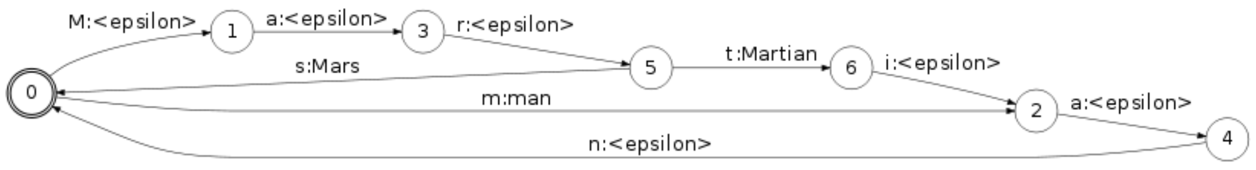
- The correct path in the search space will become dominant at some point
- It is highly unlikely that paths at the fringes will become the dominant one
    - Pruning of the search space by
        - distance to the best partial path at the current time index
        - absolute number of active states in the search space
- In the classical formulation can be solved using Dynamic Programming and back tracking of the best path
    - Efficient but gets complicated with the language model context (i.e. 3-gram requires to condition on the previous 2 words)

As a result:

- Decoder is the single most optimized component of any ASR system.
- Venture into that field only if you have stomach for advanced math, short cuts, assembler code and other tricks.

Representation as Finite State Transducer

- In the classical formulation all components can be expressed as an FST (Finite State Transducer)
- An fst is an finite automation (not necessarily deterministic), that transduces its input language to its output language



- Building blocks:
    - States (numbered)
    - Arcs between states
    - Input label on arc
    - Output label on arc
    - Weight of an arc (in our case scores or probabilities)
- In ASR we typically refer to HCL or HCLG
    - H = fst representing the **H**idden Markov Model
        - Input: acoustic features | Output: Phonemes in Context
    - C = phonemes in **C**ontext
        - Input: Phonemes in context labels | Output: phoneme symbols
    - L = **L**exicon
        - Input phoneme sequence | Output: Words
    - G = Language Model (originally referred to a **G**rammar)
        - Input words | Output: Word sequence
        - Giving context to words and removing unlikely sequences
- These fsts are "composed" to form the final model

    - Composing of 2 fsts A and B:
        - Output of fst A must match input of fst B
        - Replace all occurances of A output by output of B
        - Add the scores and prune all parts of A or B that can't be linked
- Original concept goes back to:
    - `Mohri, M., Pereira, F., & Riley, M. (2002). Weighted finite-state transducers in speech recognition. Computer Speech & Language, 16(1), 69-88.`
    - Check the paper for details and algorithm explanations
- Example fst sniplet in plain text:

```
0    100    arrêtez    arrêtez    7.16699219
0    28    installe    installe    6.65625
0    99    écoutez    écoutez    7.16699219
0    29    as    as    6.87988281
0    99    activez    activez    7.57226562
0    99    recherche    recherche    7.57226562
1    30    [#app_name]    [#app_name]    0.220703125
1    31    la    la    3.99414062
1    32    le    le    2.87207031
1    33    un    un    2.86328125
1    99    les    les    4.21191406
1    100    [#service_name]    [#service_name]    4.30664062
1    99    toc    toc    4.57226562
1    34    l'    l'    4.27441406
```

- Benefits:
    - Search graph pre-computed and can be loaded into main memory → speed

Evaluation Criteria

Overall Evaluation:

- WER = Word Error Rate (also known as minimal Edit distance or Levenshtein distance as percentage)
    - Given a groundtruth transcription A
    - Given a recognition result B
    - Calculate the minimal number of insertion, deletion, and substitution operations to transform B into A
    - Normalized by the length of A
    - $ WER(A,B) = 100 \cdot \min \frac{\#sub + \#del + \#ins}{|A|} $
    - Can be > 100% !!
    - Example:
    - `A = All your base belong to us`
    - `B = None base belong to you and me`
    -
    - `Transform B into A requires:`

```
o  All   your base belong to us
o  None  __   base belong to you and me
o  -----------------------------------
o   S     I                   S   D   D
o
o  -> 2 Substitutions, 1 Insertion, 2 Deletions = 5 Errors
o  |A| = 6 words
   WER = 100 * 5/6 = 83.34%
```

- CER = Character Error Rate (same as WER but on character level. Used for JP)
- SER = Sentence Error Rate (same as WER but binary)
    - Either 0 or 1 for a sentence
    - One insertion, deletion or substitution counts as hard failure

WER, SER and CER for a whole testset is calculated by averaging the results for each utterance/sentence in the test set

Language Model evaluation:

- The language model is evaluated using log Perplexity (PPL)

$$PPL = -\frac{1}{M} \sum_{[hw] \in T} \{M(h,w) \cdot \log p(w|h)\}$$

        - M(h,w) = count of all occurrences of the word w with history h in test corpus T
        - M number of words in the test corpus T
    - PPL is related to the coding **entropy** of a probability distribution on words (discrete distribution)
        - So you can read it as, how many words on average, the model is considering for each position in a text
        - The lower the better
    - PPL is only comparable across corpora if:
        - The model is not altered
        - The vocabulary of the model is not altered
        - There are no unknowns in the test corpus
    - Caveat: PPL assumes that the test corpus is drawn from the same distribution as the one the model was trained on
        - i.e. train a model on sports news and than evalute on Harry Potter gives you a number but that does not mean much
    - Caveat:
        - Special care must be taken to deal with out-of-vocabulary words in the test corpus
        - Either remove all sentences with OOV or add special handling in the PPL calculation
    - There is a relation between WER and PPL for PPL ranges of ~100 to 300