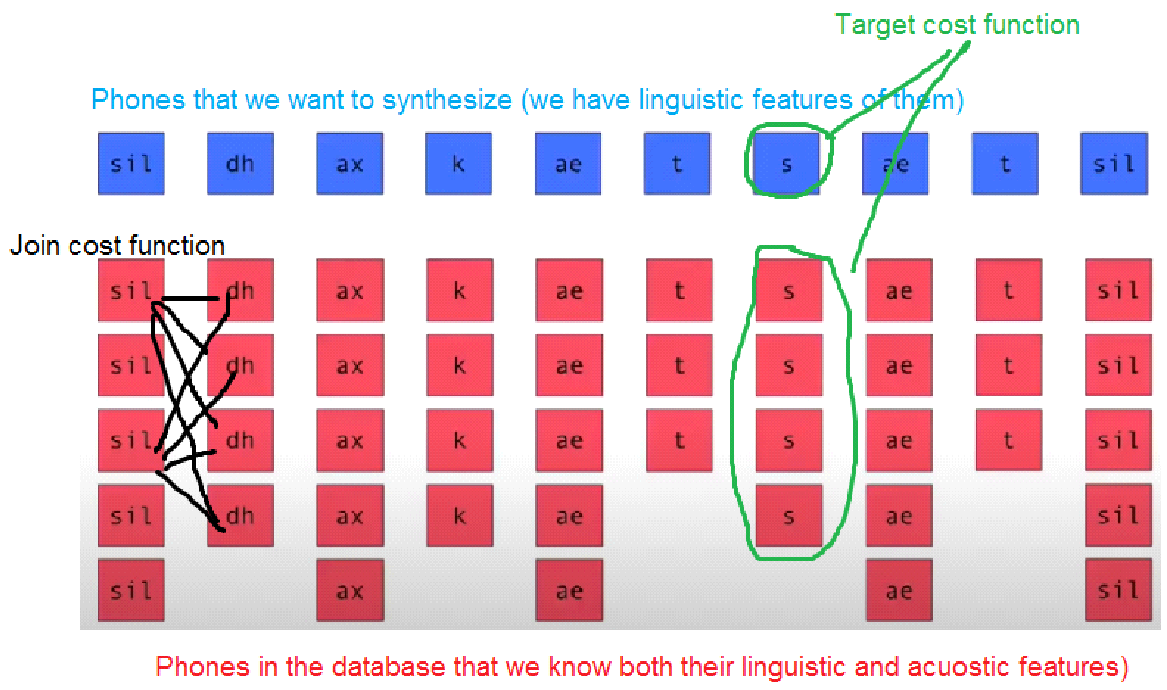
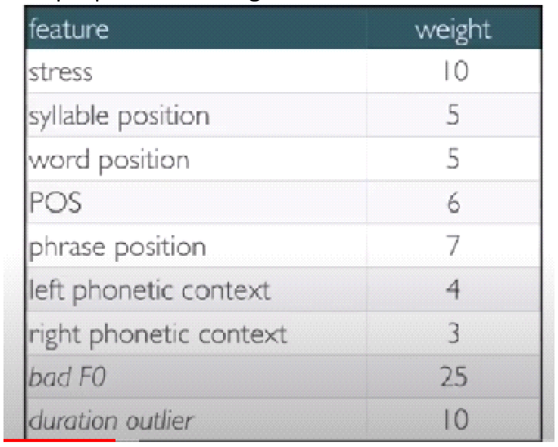
TTS Session 2

٢٣ يناير، ٢٠٢٢ ٠٩:٤٩ م

- **Session 2**
  - Basic Processes of the TTS using unit selection synthesis
    - **Text analysis**: going from written text, possibly with some mark-up, to a set of words and their relationships in an internal representation, called an utterance structure.
      - The first task is **Utterance chunking**: to chunk the text into more reasonably sized utterances. An utterance structure is used to hold the information for what might most simply be described as a sentence
        - Separating a text into utterances is important, as it allows synthesis to work bit by bit, allowing the waveform of the first utterance to be available more quickly than if the whole files was processed as one
        - As a first approximation, utterances can be separated after full stops (periods), question marks, or exclamation points
      - Inside every utterance we do **tokenization**, For many languages, tokens are white-space separated
      - There is also **text normalization**, in this step we convert tokens that can't be looked up directly of their pronounciation to one or more words, such as: (Numbers ("23" => "Twenty Three"), dates ("May 5 1996" => "Fifth of May nineteen ninety six"), abbreviations ("DR." => "doctor"), money ("$12" => "twelve dollars" , time, addresses, ...)
    - **Linguistic analysis**:
      - Finding **pronunciations** of the words (for every word we find sequence of phones for it using either lexicon or letter-to-sound rules)
      - **Extracting Linguistic features** (syllables, stress, previous and next phones, ...) to them: phrasing, intonation (mainly F0 contours), durations and energy. we predict these features using statistically trained models.
    - **Waveform generation**: From a fully specified form (pronunciation and linguistic features) generate a waveform.
      - Speech Synthesis - Module 2 unit selection - 4 Target cost and join cost
        - Consider that we want to say the word "The cats" and we got its pronuonciation as sil, dh, ax, k, ae, t, s, ae, t, sil
          - We now want to choose for each unit (phone for our example but in realty it will be diphone) one the units of same type in our database


Target cost function
Phones that we want to synthesize (we have linguistic features of them)
sil dh ax k ae t s ae t sil
Join cost function
sil dh ax k ae t s ae t sil
Phones in the database that we know both their linguistic and acuostic features)

- We need a target cost function at each phone to choose one of its candidates (for example choose one t from the 3 t's in our database)
  - The target cost function will take in its account the context of the phone that we want to synthesize and the context of recorded phone in pur database at the sentence from which it is recorded
    - Target cost function gets cost of a unit to be selected (with its context) and its candidates (with their respective contexts) **across columns**
    - Target cost function
      - target cost function evaluates linguistic properties between candidates in the database (**across rows**)
    - In festival we use the following weighted linguistic properties for target cost function



feature	weight
stress	10
syllable position	5
word position	5
POS	6
phrase position	7
left phonetic context	4
right phonetic context	3
bad F0	25
duration outlier	10

- We also need another metric which is the join cost function which is a measure of how well the selected phone from the database "fits-well" (concatenation quality) with the other selected phones from the database
  - Note that un-naturalness is observed most at joins
  - So we need to evaluate this join cost function with every pair of units in our database (dynamic programming approaches is useful here to reduce

                - time complexity)
                - Join cost function takes into its accounts acoustic properties (Fundamental frequency, spectral envelope, energy) of the 2 phones we wish to compute their join function
                - This is done in Festival through the use of `multisyn` function
            - We combine the 2 cost functions to reach at a sequence of phones from the database that minimizes the overall cost
- Festival notes
    - Basic use
        - Festival is just the engine that we will use in the process of building voices, both as a run-time engine for the voices we build and as a tool in the building process itself
        - The Festival system consists of a set of C++ objects and core methods suitable for doing synthesis tasks. These objects include synthesis specific objects like, waveforms, tracks and utterances as well as more general objects like feature sets, n-grams, and decision trees.
        - In order to give parameters and specify flow of control Festival offers a scripting language based on the Scheme programming language. Having a scripting language is one of the key factors that makes Festival a useful system. Most of the techniques in this book for building new voices within Festival can be done without any changes to the core C++ objects.
            - This makes development of new voices not only more accessible to a larger population or users, as C++ knowledge nor a C++ compiler is necessary, it also makes the distribution of voices built by these techniques easy as users do not require any recompilation to use newly created voices.
            - Festival's use of Scheme is (in general) limited to simple functions
            - Speech application developers typically does most of their customizing in Scheme
            - It is primarily only the synthesis research community that has to deal with the C++ end of the system, though C/C++ interfaces to the systems as a library are also provided
        - Basic commands
            - *festival --tts example.txt* //This will speak the text in example.txt using the default voice.
            - You may select other voices for synthesis by calling the appropriate function
                - (voice_cmu_sls_diphone)
            - `(Parameter.set 'Duration_Stretch 1.5)` //Will make all durations longer for the current voice (making the voice speak slower), changing the voice will reset this value
            - the tts funciton takes two arguments, a filename and a mode, mode can be used to allow special processing of text, such as respecting markup or particular styles of text like email etc. In simple case the mode will be nil which denotes the basic raw text (such as text in .txt file)
                - `(tts "WarandPeace.txt" nil)`
            - Commands can also be stored in files, which is normal when a number of function definitions and parameter settings are required. These scheme files can be loaded by the function load
                - `(load "commands.scm")`
                - Arguments to Festival at startup time will normally be

treated as command files and loaded
        - *festival commands.scm* //is equivalent to starting festival and calling (load "commands.scm ")
    - However if the argument starts with a left parenthesis (the argument is interpreted directly as a Scheme command.
        - *festival '(SayText "a short example.")'* //is equivalent to starting festival and calling (SayText "a short example.")
    - If the -b (batch) option is specified Festival does not go into interactive mode (displaying propmpt and waiting for commands) and exits after processing all of the given arguments.
        - *festival -b mynewvoicedefs.scm '(SayText "a short example.")'*
            - //Will load mynewvoicedefs.scm in festival and then says the text a short example and then return to cmd
    -

- Utterance structure
    - The utterance structure consists of a set of **relations** over a set of **items**
        - Each item represents a object such as a word, segment (phone), syllable, etc. while relations relate these items together. An item may appear in multiple relations, such as a segment will be in a Segment relation and also in the SylStructure relation.
            - Items may contain a number of features.
        - Relations define an ordered structure over the items within them, in general these may be arbitrary graphs but in practice so far we have only used lists and trees
            - There are no built-in relations in Festival and the names and use of them is controlled by the particular modules used to do synthesis. Language, voice and module specific relations can easy be created and manipulated.
    - For our basic English voices the relations used are as follows
        - Text: Contains a single item which contains a feature with the input character string that is being synthesized
        - Token: A list of trees where each root of each tree is the white space separated tokenized object from the input character string. Punctuation and whitespace has been stripped and placed on features on these token items.
        - Word: The words in the utterance (something that can be given a pronunciation from a lexicon or letter-to-sound rules). these words appear also as leaves in the following relations: Token, Phrase and appear as roots for SylStructure relation.
        - Phrase: A simple list of trees representing the prosodic phrasing on the utterance. The tree roots are labeled with the phrase type and the leaves of these trees are words that are in the Word relation.
        - Syllable: A simple list of syllable items.
        - Segment: A simple list of segment (phone) items.
        - SylStructure: A list of tree structures where each tree root is a word that has daughters of syllable items and each syllable item has daughters of segments (phones) items
        - IntEvent: A simple list of intonation events (accents and boundaries)
        - Intonation: A list of trees whose roots are syllable items, and daughters are intonation events.

    - Target: A list of trees whose roots are segments and daughters are F0 target points.
    - Wave: A relation consisting of a single item that has a feature with the synthesized waveform.
- Modules
  - The basic synthesis process in Festival is viewed as applying a set of modules to an utterance. Each module will access various relations and items and potentially generate new features, items and relations. Thus as the modules are applied the utterance structure is filled in with more and more relations until ultimately the waveform is generated (and thus the relaion Wave is filled).
  - Modules may be written in C++ or Scheme. Which modules are executed are defined in terms of the utterance type, a simple feature on the utterance itself. For most text-to-speech cases this is defined to be of type Tokens. The function `utt.synth` simply looks up an utterance's type and then looks up the definition of the defined synthesis process for that type and applies the named modules.
  - Synthesis types maybe defined using the function defUttType. For example definition for utterances of type Tokens is

    -
      ```
      (defUttType Tokens
        (Token_POS utt)
        (Token utt)
        (POS utt)
        (Phrasify utt)
        (Word utt)
        (Pauses utt)
        (Intonation utt)
        (PostLex utt)
        (Duration utt)
        (Int_Targets utt)
        (Wave_Synth utt)
        )
      ```
  - In general the modules named in the type definitions are general and actually allow further selection of more specific modules within them. For example the Duration module respects the global parameter Duration_Method and will call then desired duration module depending on this value.
  - Basic modules have the following basic functions
    - `Token_POS`: basic token identification, used for homograph disambiguation
    - `Token`: Apply the token to word rules building the Word relation
    - `POS`: A standard part of speech tagger (if desired)
    - `Phrasify`: Build the Phrase relation using the specified method. Various are offered, from statistically trained models to simple CART trees.
    - `Word`: Lexical look up building the Syllable, Segment and SylStructure relations.
    - `Pauses`: Prediction of pauses, inserting silence into the Segment relation, again through a choice of different prediction mechanisms
    - `Intonation`: Prediction of accents and boundaries, building the IntEvent and the Intonation relations
    - `PostLex`: Post lexicon rules that can modify segments based on their context. This is used for things like vowel reduction, contractions, etc.
    - `Duration`: Prediction of durations of segments.
    - `Int_Targets`: The second part of intonation. This creates the Target relation representing the desired F0 contour.

- `Wave_Synth`: A rather general function that in turn calls the appropriate method to actually generate the waveform