Speech Processing: TTS Waveform Generation

Module 6
Catherine Lai
28 Oct 2021
Assignment 1: Deadline Extension

The assignment is now due on

Tuesday 9 November 2021, 12 noon

Submit on Learn
Previously: Text-to-Speech

Two main components:

- **Front-end**: Analyze text, generate a linguistic specification of what to actually generate
- **Back-end**: Waveform generation from the linguistic specification

This week we’ll look at waveform generation
Front-end summary

Text normalization

- Tokenization
- Non-standard words, e.g. numbers, acronyms
- Word sense disambiguation, e.g. POS tagging

Pronunciation

- Pronunciation dictionary, e.g. Unilex
- Letter to Sound rules, e.g. decision tree

Prosody

- Prominences: Pitch accents
- Boundaries: Breaks, boundary tones

We want to develop rules or statistical mappings so we can generate new words and utterances robustly e.g. pronunciations of words that aren’t in the dictionary
TTS Front-end - Festival

**Front-end:**

festival> (set! myutt (Utterance Text "Put your own text here."))
festival> (Initialize myutt)
festival> (Text myutt)
festival> (Token_POS myutt)
festival> (Token myutt)
festival> (POS myutt)
festival> (Phrasify myutt)
festival> (Word myutt)
festival> (Pauses myutt)
festival> (PostLex myutt)

**Back-end:**

festival> (Wave_Synth myutt)
Front-end output: Festival

In festival, the linguistic specification generated by the front-end is a set of relations, e.g.:

- Token
- Word
- Phrase
- Syllable
- SylStructure
- Segment

These all hold different bits of information, e.g. words, POS tags, break positions, syllable boundaries....
**Festival: Front-end Modules and Relations**

<table>
<thead>
<tr>
<th>Festival module</th>
<th>Added relation</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Token</td>
<td>Whitespace tokenization</td>
</tr>
<tr>
<td>Token_POS</td>
<td>Word</td>
<td>basic token identification/homograph disambiguation</td>
</tr>
<tr>
<td>Token</td>
<td></td>
<td>Token to word rules building the Word relation</td>
</tr>
<tr>
<td>POS</td>
<td></td>
<td>Part of Speech tagger (e.g. HMM)</td>
</tr>
<tr>
<td>Phrasify</td>
<td>Phrase</td>
<td>Predict phrase breaks (decision tree)</td>
</tr>
<tr>
<td>Word</td>
<td>Syllable, Segment, SylStructure</td>
<td>Lexical look up/LTS rules</td>
</tr>
<tr>
<td>Pauses</td>
<td></td>
<td>Prediction of pauses (e.g. decision tree)</td>
</tr>
<tr>
<td>PostLex</td>
<td></td>
<td>Post lexicon rules: modify segments based on their context. E.g. vowel reduction, contractions, etc.</td>
</tr>
</tbody>
</table>

[http://festvox.org/docs/manual-2.4.0/festival.html](http://festvox.org/docs/manual-2.4.0/festival.html)
[http://festvox.org/bsv/](http://festvox.org/bsv/)
## Festival: Front-end Modules and Relations

<table>
<thead>
<tr>
<th>Festival Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens</td>
<td>Token properties</td>
</tr>
<tr>
<td>Word</td>
<td>Word properties</td>
</tr>
<tr>
<td>Phrase</td>
<td>Groups of words that form phrases</td>
</tr>
<tr>
<td>Syntax</td>
<td>Relates on words via tree provided by parser</td>
</tr>
<tr>
<td>SylStructure</td>
<td>Relates words to syllables and segments</td>
</tr>
<tr>
<td>Syllable</td>
<td>Groups of phones that form syllables</td>
</tr>
<tr>
<td>Segment</td>
<td>Phones</td>
</tr>
<tr>
<td>IntEvent</td>
<td>Accents and boundary labels</td>
</tr>
<tr>
<td>Intonation</td>
<td>Relates syllables to IntEvents</td>
</tr>
</tbody>
</table>
## Festival: Front-end Modules and Relations

<table>
<thead>
<tr>
<th>Festival Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens</td>
<td>Token properties</td>
</tr>
<tr>
<td>Word</td>
<td>Word properties</td>
</tr>
<tr>
<td>Phrase</td>
<td>Groups of words that form phrases</td>
</tr>
<tr>
<td>Syntax</td>
<td>Relates on words via tree provided by parser</td>
</tr>
<tr>
<td>SylStructure</td>
<td>Relates words to syllables and segments</td>
</tr>
<tr>
<td>Syllable</td>
<td>Groups of phones that form syllables</td>
</tr>
<tr>
<td>Segment</td>
<td>Phones</td>
</tr>
<tr>
<td>IntEvent</td>
<td>Accents and boundary labels</td>
</tr>
<tr>
<td>Intonation</td>
<td>Relates syllables to IntEvents</td>
</tr>
</tbody>
</table>

Easier just to inspect the Festival output.

Not all relations are generate in the voice you are analyzing for the assignment!
I work in Edinburgh

festival> (set! myutt (SayText "I work in Edinburgh."))

festival> (utt.relation.print myutt 'Word)
()
id _5 ; name I ; pos nn ; pos_index 0 ; pos_index_score 0 ; pbreak NB ;
id _6 ; name work ; pos_index 0 ; pos_index_score 0 ; pos nn ; pbreak NB ;
id _7 ; name in ; pos_index 4 ; pos_index_score 0 ; pos in ; pbreak NB ;
id _8 ; name Edinburgh ; pos_index 2 ; pos_index_score 0 ; pos nnp ; pbreak BB ;
Nil

festival> (utt.relation.print myutt 'SylStructure)
()
id _5 ; name I ; pos nn ; pos_index 0 ; pos_index_score 0 ; pbreak NB ;
id _6 ; name work ; pos_index 0 ; pos_index_score 0 ; pos nn ; pbreak NB ;
id _7 ; name in ; pos_index 4 ; pos_index_score 0 ; pos in ; pbreak NB ;
id _8 ; name Edinburgh ; pos_index 2 ; pos_index_score 0 ; pos nnp ; pbreak BB ;
id _9 ; name . ; pos_index 1 ; pos_index_score 0 ; pos punc ; pbreak BB ;
nil

Word and SylStructure relation both tell us something about words, but looking at this view doesn't tell us the difference….
festival> (utt.relation_tree myutt 'SylStructure)

((("I"
    ((id "_.5")
      (name "I")
      (pos "nn")
      (pos_index 0)
      (pos_index_score 0)
      (pbreak "NB")))))

("syl" ((id ".11") (name "syl") (stress 1))
  ("ae" ((id ".12") (name "ae") (end 0.23012498))))))

("work"
 ((id "_.6")
   (name "work")
   (pos_index 0)
   (pos_index_score 0)
   (pos "nn")
   (pbreak "NB")))

("syl" ((id ".13") (name "syl") (stress 1))
  ("w" ((id ".14") (name "w") (end 0.35974997))))
  ("@" ((id ".15") (name "@") (end 0.40056247))))
  ("r" ((id ".16") (name "r") (end 0.46806258))))
  ("k" ((id ".17") (name "k") (end 0.53968757))))))

("in"
 ((id "_.7")
   (name "in")
   (pos_index 4)
   ....etc

Words are made have word attributes: Id, name, pos,...,pbreak

Each words is made up of syllables

Each syllable is made up of segments
Pronunciations

Question: Why not just store everything in a dictionary?

Answer: We want to be able to generalize to new words

e.g. productive morphology

- Edinburghish
- Netflixify
TTS Backend: Waveform Generation

Once we have a linguistic specification, we want to use it to generate a waveform that has the described properties.

Some options:

- **Concatenative synthesis**: Join units of real speech together
- **Statistical parametric speech synthesis**: Learn a mapping between linguistic features and a discrete set of states which have the ‘right’ acoustic features
- **Neural TTS**: Learn a mapping between linguistic features and spectral features, and spectral features to waveform using neural networks

In this course, we’ll only be talking about concatenative synthesis (other methods in sem 2)
Concatenative Synthesis

Learning a probabilistic mapping between linguistic and acoustic features is hard! For a long time, parametric models just didn’t sound very natural

- Need a lot of data to learn models
- Need of lot of compute to train and run models
- Mathematical models were too simplistic
Concatenative Synthesis

BUT joining together existing speech can sound very natural, if you can guarantee:

- You can select appropriate units of speech to concatenate, such that
- The joins are smooth
- You can select the right units to convey the right meaning

You need a lot of high quality data from speakers
Concatenative Synthesis

Three main components

- Database of appropriate units
- Join cost function
- Target cost function
The Units: Diphones

Concatenative synthesis generally uses diphones: a diphone starts at the middle of one phone and ends at the middle of the other.
The Units: Diphones

Middles of phones are more stable in their spectral properties than the edges, because of coarticulation. So, concatenating diphones should lead to smoother joins.
Diphone database

Requirements

- Clean, clear recordings of a single speaker
- Recordings of every possible diphone in the language
- Phone segmentation (timings) to calculate where diphones start and end
Diphone synthesis vs Unit selection

Diphone synthesis:

- One recording of every diphone (small database)
- Use signal processing methods to change F0, duration, and smooth joins to match linguistic specification
  - e.g. TD-PSOLA (later)

Unit selection:

- Record a large naturalistic database
- Select diphone units based on closeness to the linguistic specification
- If the database has enough variation, don’t worry about signal processing!
Diphone synthesis vs Unit selection

Diphone synthesis:
- Only captures coarticulation/contextual variation for the one recording

Unit selection:
- Potentially captures more global effects, e.g. prosodic, emotional variation if the database is big enough
- If the application vocabulary is small enough, you may end up with very few joins.

The voice you will analyze in the assignment is a unit selection voice!
Unit selection

Goal:

● Given a database of speech
● Select the best sequence of diphone units that corresponds to the linguistic specification (the target)
Unit Selection

The “best” sequence

- Minimizes the difference between the linguistic specification of the units and the selected units (target cost)
- Minimizes the perception of discontinuities at where the units are joined together (join cost)
Unit selection: Target Cost

- **Target cost**: How well the target linguistic specification matches the potential unit

  e.g.

<table>
<thead>
<tr>
<th>Target: m-b</th>
<th>Candidate: i-b</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>i</td>
</tr>
<tr>
<td>bilabial</td>
<td>High-front</td>
</tr>
<tr>
<td>nasal</td>
<td>vowel</td>
</tr>
<tr>
<td>voiced</td>
<td>voiced</td>
</tr>
<tr>
<td>Syl end</td>
<td>Syl end</td>
</tr>
<tr>
<td>stress:0</td>
<td>stress:1</td>
</tr>
<tr>
<td>stress:0</td>
<td>stress:0</td>
</tr>
</tbody>
</table>

**Not a very good match!**
Unit selection: Target Cost

- Target cost: How well the target linguistic specification matches the potential unit

  e.g.

      Target: m-b
      | m    | b   |
      | bilabial | bilabial |
      | nasal  | stop |
      | voiced | voice |
      | Syl end | Styl start |
      | stress:0 | stress:0 |

      Candidate: n-p
      | n    | p    |
      | alveolar | bilabial |
      | nasal  | stop |
      | voiced | unvoiced |
      | Syl end | Styl start |
      | stress:1 | stress:0 |

A better match, though still not perfect

You would usually consider many more features!
Target cost

\[ T(s_t, u_j) = \sum_{p=1}^{P} w_p \Delta p \]

- Target specification
- Candidate unit
- \( P \) different linguistic features, e.g. place, manner, stress, POS, etc.
- Weighting for feature \( p \), e.g.
  - Matching vowel vs consonant may be more important than matching position in phrase

Weighted sum the differences across all features of interest

Bigger sum -> larger cost -> worse match
Unit selection: Join Cost

Join cost: How well potential unit joins with its neighbours, i.e. how well they match at the edges in terms of acoustic characteristics: spectrum, F0, energy, etc.

\[ J(u_t, u_{t+1}) = \sum_{p=1}^{P} w_p J_p(u_t[p], u_{t+1}[p]) \]

- Potentially adjacent candidate units
- Weighted sum of the differences across all P features of interest
- Weighting for feature p
- A measure of how smoothly candidates would join
Unit selection: objective function

Find the sequence of units that minimizes the overall target and join costs

We want a sequence of units that will represent our utterance

\[ \hat{U} = \arg\min_U \sum_{t=1}^{T} T(s_t, u_t) + \sum_{t=1}^{T} J(u_t, u_{t+1}) \]

From all possible sequences of T units, find the one that minimizes...

The sum of the target costs for all the units making up utterance

The sum of the join costs between each unit in the utterance
Unit selection:

Given a database of diphones
And a target linguistic specification
Find the sequence of diphone units that
- Best match the target specification
- Minimizes join discontinuity

Actual optimization is done using the Viterbi algorithm. We will cover this later in the course, but you don’t need to explain what this is for assignment 1. For now just assume such an algorithm exists.
Common database and target problems

- Incorrect phone boundaries
- Incorrect phonetic transcription
- Missing diphones
- Diphones don’t match all aspects of the target specification
Common join problems

- Mismatch in spectral features,
  - e.g. due to (di)phone alignment
- Abrupt change in acoustic features
  - Energy
  - F0
- Amplitude discontinuity -> clicks
Links to database recordings

http://festvox.org/cmu_arctic/cmu_arctic/cmu_us_awb_arctic/

On the remote desktop (no wave files):
/Volumes/Network/courses/ss/festival/lib.incomplete/voices-multisyn/english/cstr_edi_awb_arctic_multisyn/awb

List of utterances:
/Volumes/Network/courses/ss/festival/lib.incomplete/voices-multisyn/english/cstr_edi_awb_arctic_multisyn/awb/utts.data

Phone label files (*.lab):
/Volumes/Network/courses/ss/festival/lib.incomplete/voices-multisyn/english/cstr_edi_awb_arctic_multisyn/utt
Diphone synthesis vs Unit Selection

Concatenative Synthesis

- Once units are selected, we need to concatenate their waveforms.
- How? Just stick them together?
- What is if the endpoints do match?
Concatenative Synthesis

- Once units are selected, we need to concatenate their waveforms
- How? Just stick them together?
Concatenating waveforms

Just stick them together? Discontinuities at the edges lead to ‘pops’
Concatenating waveforms

We can reduce discontinuities by joining at ‘zero crossings’: where the waveform has amplitude 0
Concatenating waveforms

We can reduce discontinuities by joining at ‘zero crossings’: where the waveform has amplitude 0
Concatenating waveforms

- Naively concatenating may result in discontinuities → clicks
- Joining at zero crossings reduces edge discontinuities → less obvious joins
- Best if we can join ‘pitch synchronously’:
  - Use tapered windows based on pitch period start points
  - Overlap and add to smooth out discontinuities
Epoch detection

Large peaks can be used to mark the start of new pitch periods.

Use autocorrelation methods to detect the starts of periods.
We can extract epoch windows as approximately 1 pitch period either side of the largest peaks in each cycle.

Apply a tapered window to reduce the edges to zero.
Time Domain Pitch Synchronous Overlap and Add (TD-PSOLA)

- Extract overlapped windowed epochs in the time domain
- Change the amount of overlap between windows
- Add back together to create a new waveform with changed F0 and duration
- Epochs further apart → lower F0
- Epochs closer together → higher F0
- Remove epochs → Shorter duration
- Add epochs → longer duration
Time Domain Pitch Synchronous Overlap and Add (TD-PSOLA)

Extract overlapping tapered windows covering 2 pitch periods

Overlap and add the epochs in the time domain to create a new waveform → “cross fade”

Bring epochs closer together → shortening the period between impulses in the source-filter model → higher F0

Figure 14.2 Timing manipulation with PSOLA. Here the original pitch is kept but the section of speech is made longer by the duplication of frames.
Spreading out epochs → lengthening the period between impulses in the source-filter model → lower F0
Simultaneous pitch and duration modification

Figure 14.4 Simultaneous pitch and timing modification, whereby a new set of synthesis epochs and a single mapping that performs pitch and timing modification are used.
Summary: Waveform generation

For concatenative synthesis, you need

- A database of units: usually diphones to cover coarticulation

For diphone synthesis:

- 1 recording of each diphone
- Use signal processing methods, e.g. TD-PSOLA to change the waveform’s duration and F0 to fit the linguistic specification and to smooth joins
Summary: Waveform generation

For unit selection:

- Large diphone database, covering many contexts
- Select units to minimize both target and join costs

The assignment voice is a unit selection voice:

- No signal processing used to smooth joins, just rely on minimize the target and join costs
Festival examples: Post-lexical rules

Word pronunciations may change depending on the surrounding words:

For example, you may pronounce the possessive ‘s with a schwa if the last phone is a fricative

- Joe’s suitcase
- The suitcase’s zip

Let’s see...
Festival examples: Bad joins

E.g. ‘I live in Edinburgh’

- Incorrect pronunciation of ‘live’
- Weird joins - let’s look in praat
Festival examples: Database issues

e.g. ‘I work in Edinburgh’

- Generated speech doesn’t seem to match the phonetic transcription - why?
- Maybe a database issue?
- Look at where the selected diphones came from using the Unit relation
Writing Advice - Background

You don’t have to write a lot about human speech production, just enough to briefly explain how concatenative TTS differs from human speech production. Mostly, you want to motivate why this might lead to errors. For example,

- What in the speech signal differentiates different phones?
- Why do we need to capture coarticulation between phones?
- Why don’t we get joins artifacts in real human speech?
Writing Advice - Background

For each Festival module (e.g. token, token_pos), try to answer:

- Why is it necessary?
- What is the theoretical goal? What should this module do?
- What does Festival actually do?

Festival modules may not directly mirror the idealized/theoretical TTS pipelines - try to highlight any differences you find
Writing advice - Background

- The background section is a good place to put a diagram, e.g. a flow chart, to illustrate the TTS pipeline
- Use this to highlight how Festival matches or doesn’t match the theoretical TTS pipeline (e.g. as discussed in the lecture/videos/readings)

Your diagrams need to be your own - definitely don’t just copy the ones from the textbook!
Try to answer the following questions for each mistake you identify

- **What is the error?**
  - Give the text input, what the output should have been and identify the part of the linguistic specification/waveform/spectrogram that shows the error
  - e.g. for pronunciation errors, give the ideal and actual phonetic transcription
  - e.g. For waveform generation errors, show the spectrogram and annotate where the error occurred

- **Where did it originate?**
  - Provide evidence for this, but don’t just copy and past the terminal output!
  - Describe the process that lead to the error occurring

- **How severe is the error? Is it likely to affect many utterances?**

- **How might you fix the error?**
  - But don’t just say “use neural networks”!
Writing Advice: Discussion and Conclusion

This doesn’t have to be very long, but you should be able to make some general comments and identify implications based on your analysis.

You could think about, in general:

- What errors make it sound the most unnatural?
- What types errors could be easily fixed?
- What types of errors would be difficult to fix?
- What makes more of a difference: front-end errors or back-end errors?
- Given the analysis you’ve done, what is Festival generally good at? What is it generally bad at?
Writing Advice

- You won’t get penalized for typos or small grammatical errors (as long as we understand what you are trying to say), but do try to give your report a proofread before submission.
- It’s ok to discuss the project with classmates and give each other feedback, but the final submission must be your own words.
Assignment 1: Deadline Extension

The assignment is now due on

Tuesday 9 November 2021, 12 noon

Submit on Learn