ASR and Dynamic Programming

Automatic speech recognition

- Given an utterance, recorded as a waveform…

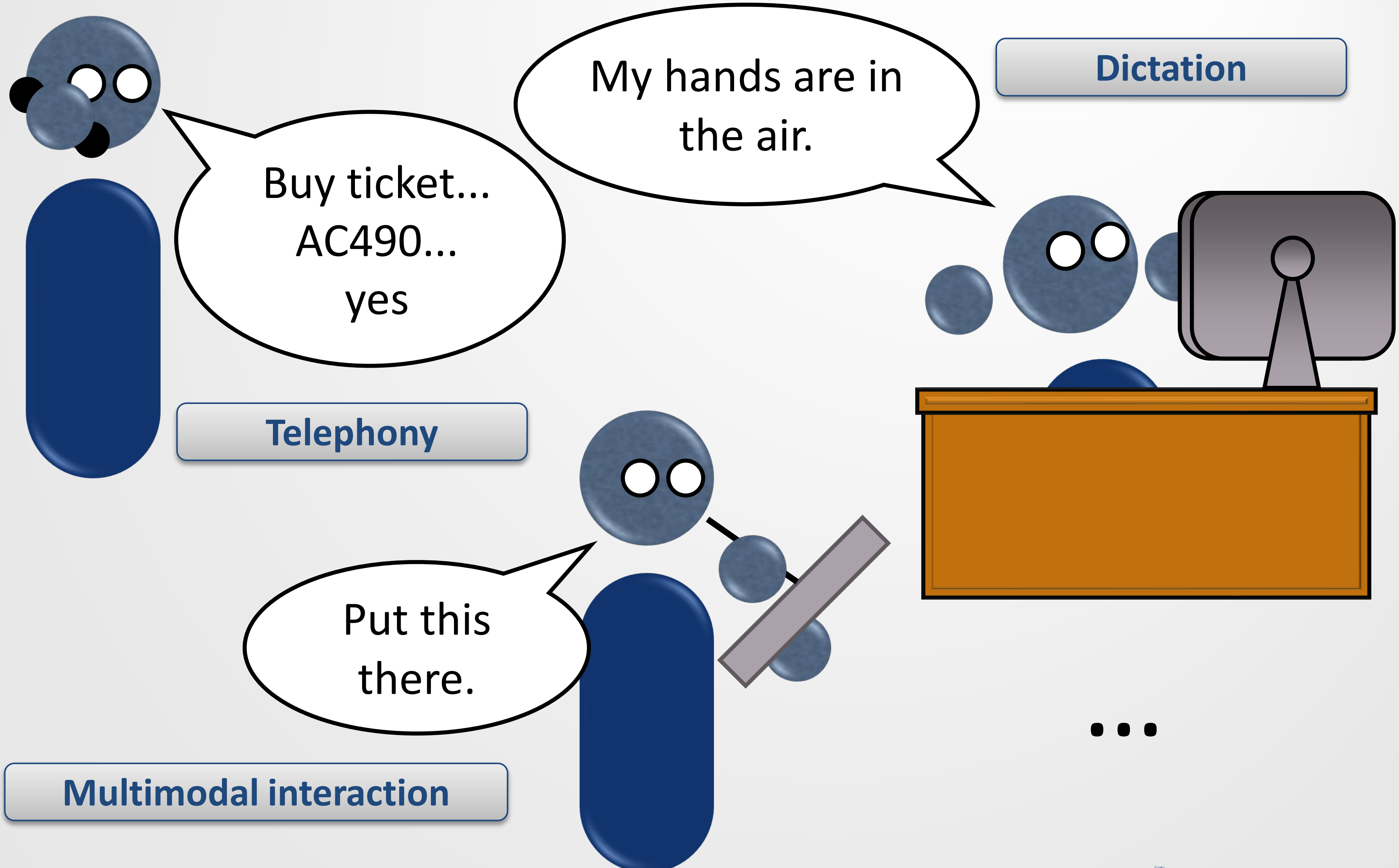
- Automatic Speech Recognition (ASR), *a.k.a.* speech-to-text (STT), transcribes it as a sequence of tokens, usually words

```
a dog
```

- Though increasingly as sub-words, like
  - Phones: `AH0 D AO1 G`
  - Character-by-character: `a _ d o g`
  - Spans of characters: `a_ do g`

UNIVERSITY OF TORONTO

Consider what we want ASR to do



Aspects of ASR systems in the world

- **Speaking style**: **Read** speech vs. **spontaneous** speech; the latter contains many **dysfluencies** (e.g., stuttering, *uh*, *like*, …)
- **Accent, dialect**: Mass-deployed or highly localized?
- **Vocabulary**: **Small** (<20 words) or **large** (>50,000 words). Words, phones, characters, sub-words? Technical? Conversational?
- **Channel**: Cell phone? Noise-cancelling microphone? Teleconference microphone?

Speech features

- Waveform inputs are very, very long
  - Usually: 1 second = 16,000 samples
- Dilated convolutional neural networks (CNNs) can learn & process the waveform directly
  - We will see an example of this in the TTS lecture
- Speech embeddings/representations can be learned with unsupervised objectives
  - The topic of CSC2518 this term
- The **f-bank** remains a convenient, fast, and widely used pre-processing step
- The result is always a sequence of speech feature vectors spaced 10s of milliseconds apart in time

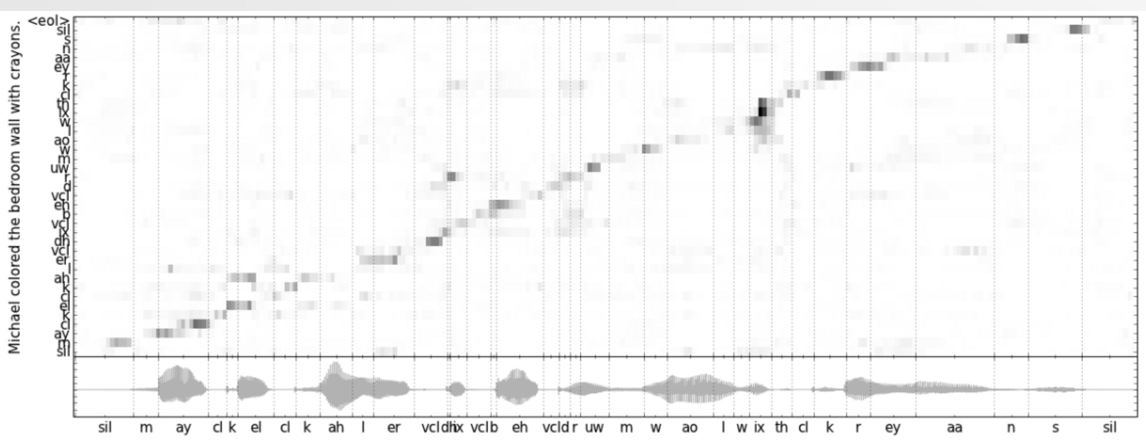
A neural approach

- We are given a sequence speech features
$$x = x_1, x_2, \dots, x_T, \qquad x_t \in \mathbb{R}^D$$
- We want a sequence of tokens (a transcription)
$$y = y_1, y_2, \dots, y_U, \qquad y_u \in \{1,2,\dots,V\}$$
  - y_u could be a character, word, phone, *etc.*
- $T \neq U$
- **Sound familiar?**
- We can do encoder/decoder NMT!
  - Source sequence (F) embeddings are now features (x)
  - Target sequences (E) are now transcriptions (y)

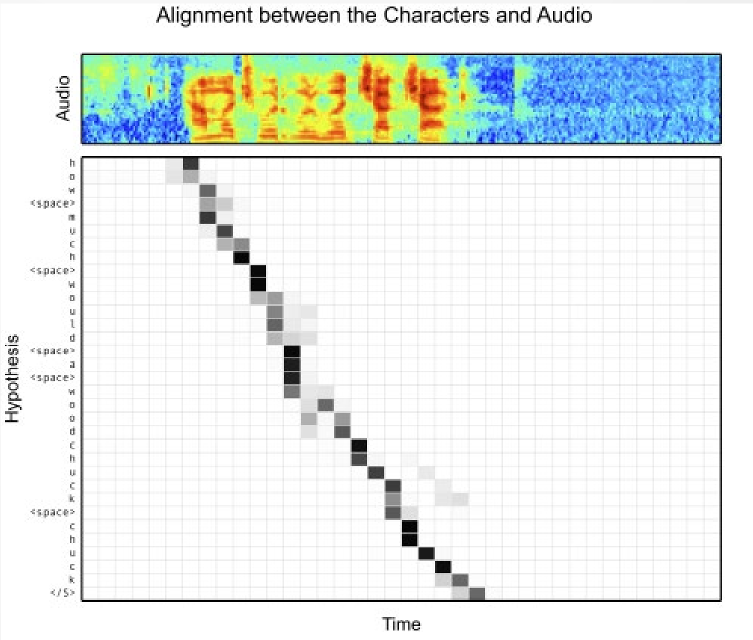
Encoder/decoder ASR

- Networks such as the Attention-based Encoder Decoder or Listen, Attend, and Spell are RNN-based encoder-decoders trained with teacher forcing (ML)

$$\mathcal{L} = -\sum_{u=1}^{U} \log P_\theta(y_u | y_{<u}, x)$$



From Chorowski *et al*. (2014) “End-to-end continuous speech recognition using attention-based recurrent NN: First results”



From Chan *et al*. (2016) “Listen, attend, and spell”

Decoding

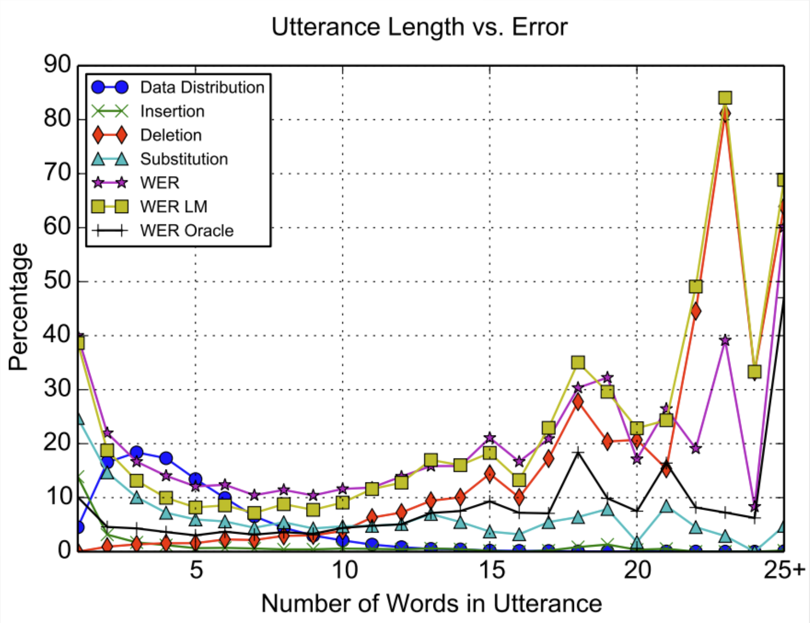
- Like in NMT, we approximate the hypothesis transcription
$$y^* = \mathrm{argmax}_y P_\theta(y|x)$$
with the beam search algorithm
- For best performance, an external, auto-regressive language model may be incorporated into each time step via shallow fusion:

$$\underbrace{\log P'_\theta(y_u|y_{<u},x)}_{\text{Total score}} \approx \underbrace{\log P_\theta(y_u|y_{<u},x)}_{\text{Encoder-decoder score}} + \lambda \underbrace{\log P_\xi(y_u|y_{<u})}_{\text{External LM score}}$$

- The impact of the external LM grows with λ

Pros and cons

- Encoder/decoder ASR with Transformers are often state-of-the-art on ASR benchmarks
- They do have some drawbacks
    - They are unsuited to streaming (real-time transcription)
    - Performance suffers on long utterances

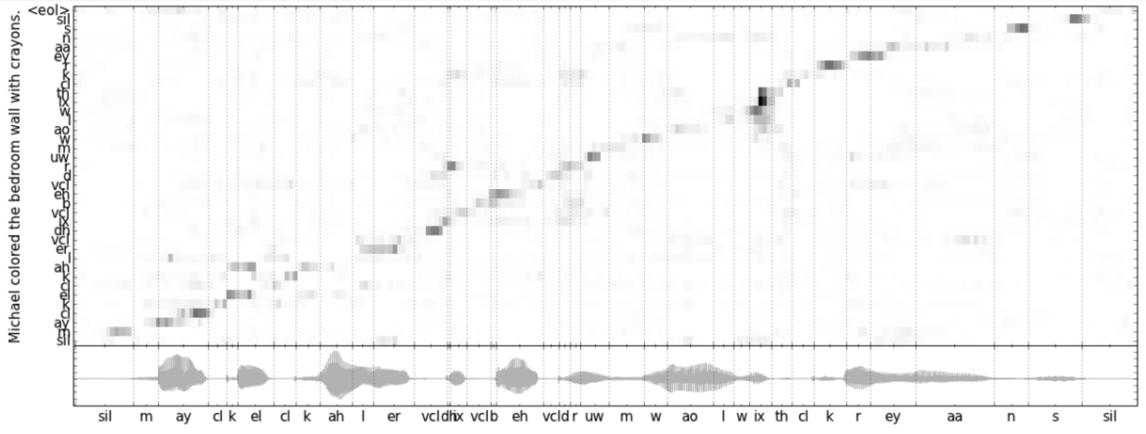


From Chan *et al*. (preprint) “Listen, attend, and spell”

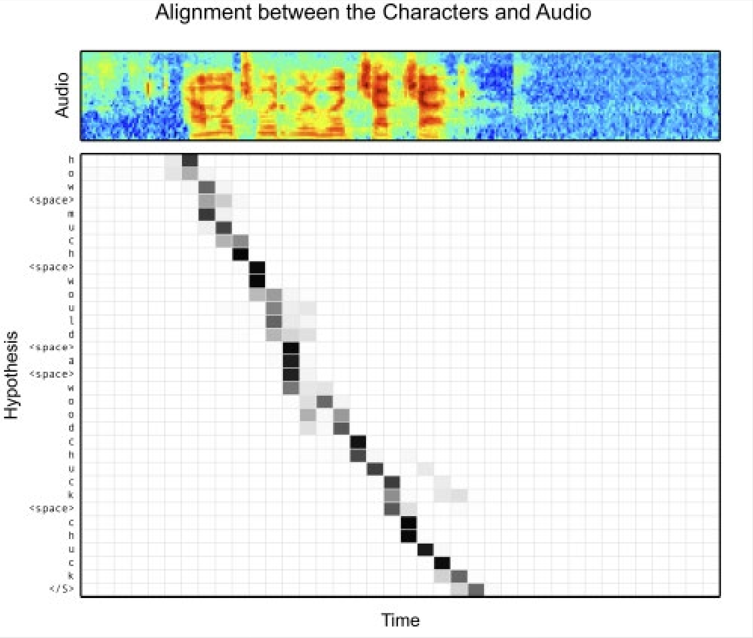
UNIVERSITY OF TORONTO

An alternative

- A decoder can attend to any hidden state $h_1, \ldots, h_T$
- This is useful for NMT: tokens or phrases can be re-ordered, added, or removed
- But it is excessive in ASR: tokens are transcribed in the **same order** that they are uttered
- **Can we exploit this monotonicity?**



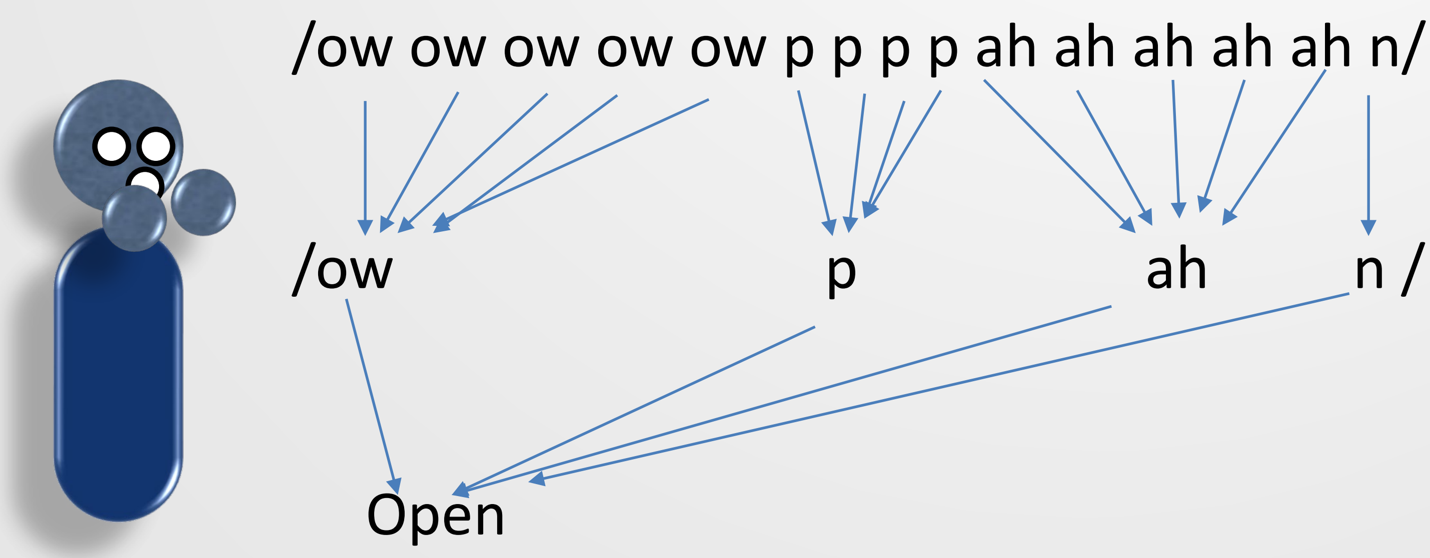
From Chorowski *et al*. (2014) "End-to-end continuous speech recognition using attention-based recurrent NN: First results"



From Chan *et al*. (2016) "Listen, attend, and spell"

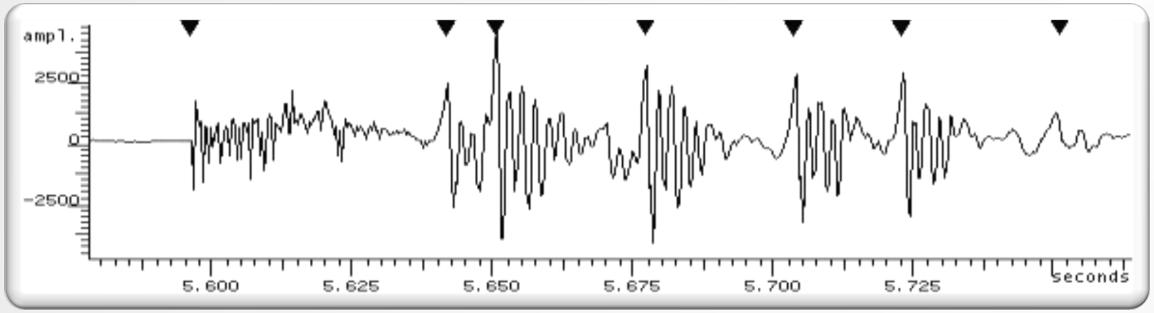
Recognizing ~~speakers~~ phones

- A first idea: since GMM can be used to recognize speakers, it can be used to recognize phones.
  - For each frame, a GMM (or DNN) classifies a phone.
  - Then we can look up to convert them into words!
    pronunciations



UNIVERSITY OF TORONTO

Some issues



- Speech **changes** over time
  - Per-frame decisions do not encode label order
  - This is valuable predictive context!
- During training, we don't know each frame's phone label!
  - We have "Open", not /ow ow ow ow ow p p p .../
  - How do we maximize the likelihood of "Open"?

Learning alignments

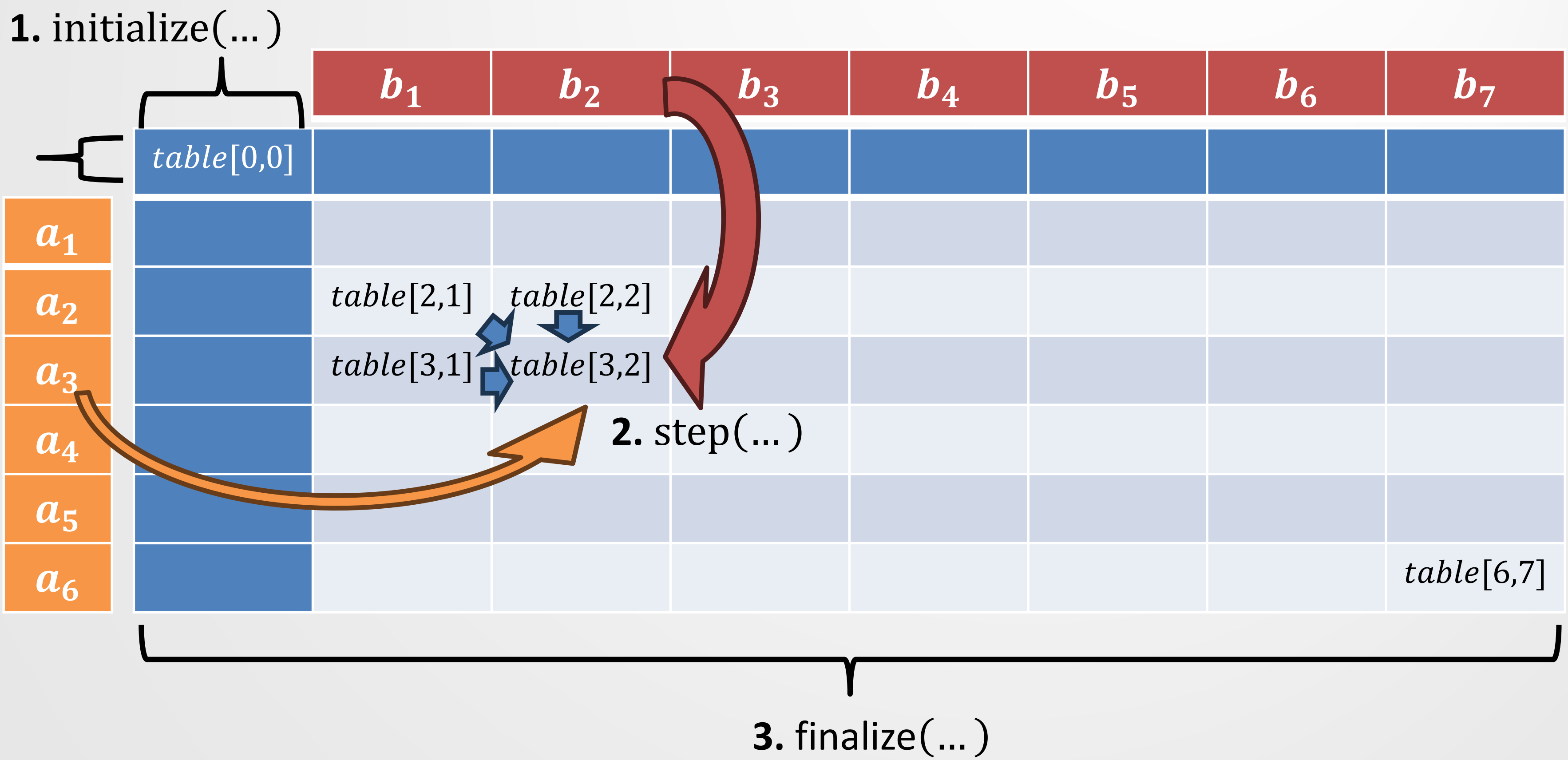
- We can solve both problems by learning monotonic alignments between sequences of frames and tokens
- To do so, both classic and end-to-end neural ASR rely on Dynamic Programming
    - Classic: Dynamic Time Warping (DTW), HMMs
    - E2E: Connectionist Temporal Classification (CTC), RNN-Transducer (RNN-T)
- DP can be used to align arbitrary sequences a and b
    - Error rates, bitext alignment, phrase-based SMT…
- The forward algorithm applies to all of these

A monotonic forward algorithm

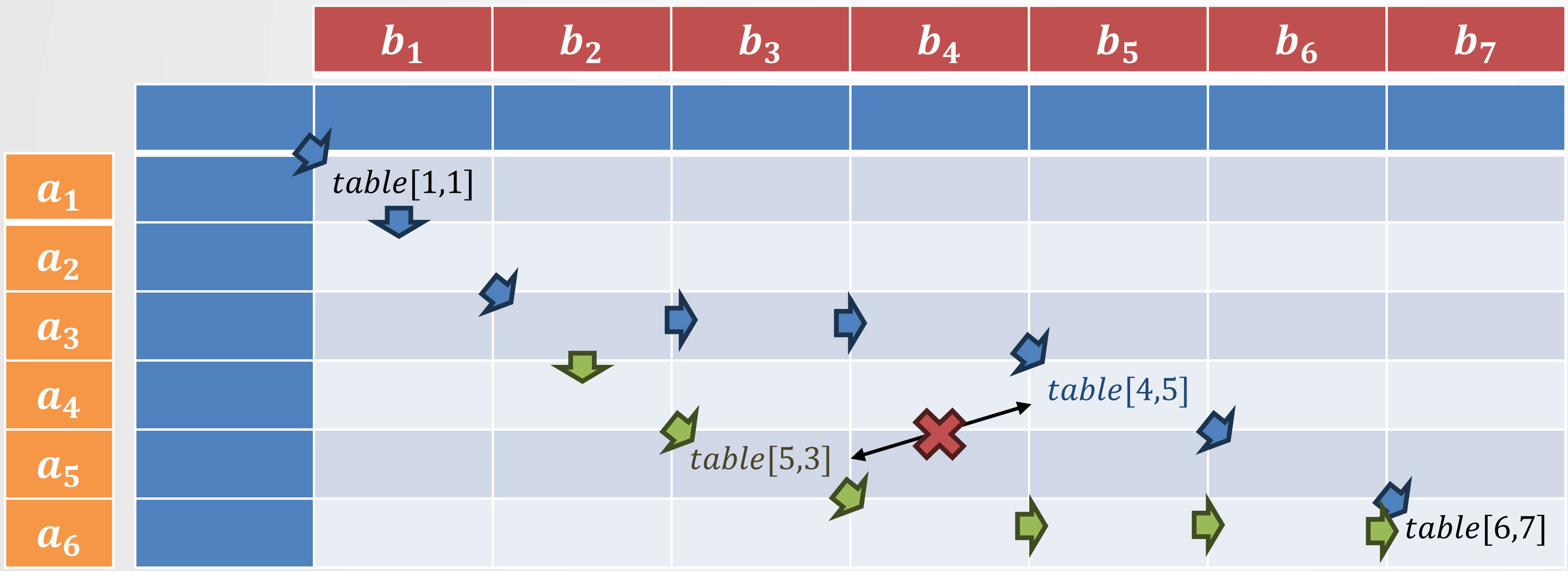
Function monotonic_forward
Inputs $a = a_1, a_2, \ldots, a_U$ **and** $b = b_1, b_2, \ldots, b_T$
1: **Define** $table[0 \ldots U, 0 \ldots T]$
2: initialize($table[0 \ldots U, 0], table[0,1 \ldots T]$)
3: **For each** u **in** $1 \ldots U$:
4:     **For each** t **in** $1 \ldots T$:
5:         $table[u, t] =$
            $\text{step}(a_u, b_t, table[u-1, t-1], table[u-1, t], table[u, t-1])$
6: **Return** finalize($table[0 \ldots U, 0 \ldots T]$)

1: Define $(U + 1) \times (T + 1)$ table $table$ to store partial results
2: Initialize first row and column of $table$
3+4: Iterate over columns and rows with t and u
5: Use a_u, b_t, and left, up, and diagonal cells to compute $table[u, t]$
6: Use $table$ to compute results

A graphical representation



Order of operations

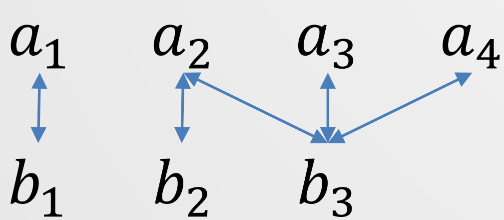


- Cells are populated in **non-decreasing** order of indices
  - $table[u,t]$ depends on all $table[u',t']$ where $u' \le u$ and $t' \le t$
  - **Not** when either $u' > u$ and/or $t' > t$

Which problems?

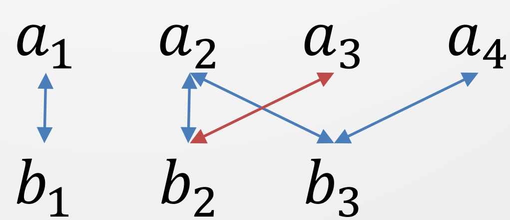
- We use monotonic_forward when solutions involve aligning each element of a to one from b and vice versa
- **All** elements of a and b must be **aligned without crossing**
- The solution may involve one or more alignments

A valid alignment



$\{a_1, b_1\} \leq \{a_2, b_2\} \leq \{a_2, b_3\}$…
…$\leq \{a_3, b_3\} \leq \{a_4, b_3\}$

An invalid alignment



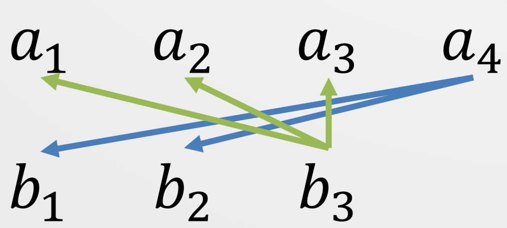
$\{a_1, b_1\} \leq \{a_2, b_2\} \leq \{a_2, b_3\}$…
…??? $\{a_3, b_2\} \leq \{a_4, b_3\}$

How?

- In DP, a problem is broken up into sub-problems which share computations
- For $\mathrm{monotonic\_forward}$, those sub-problems handle alignments of prefixes of a and b
- $table[u, t]$ stores the solution for $\{a_1, \dots, a_u\}, \{b_1, \dots, b_t\}$
- All elements of the prefixes must **also** be aligned without crossing
- This guarantees that $\{a_u, b_t\}$ is part of **all** the alignments considered in $table[u, t]$
- Therefore, $table[u, t]$ need only consider how to **correctly** extend a prefix with $\{a_u, b_t\}$

Aligning the last element

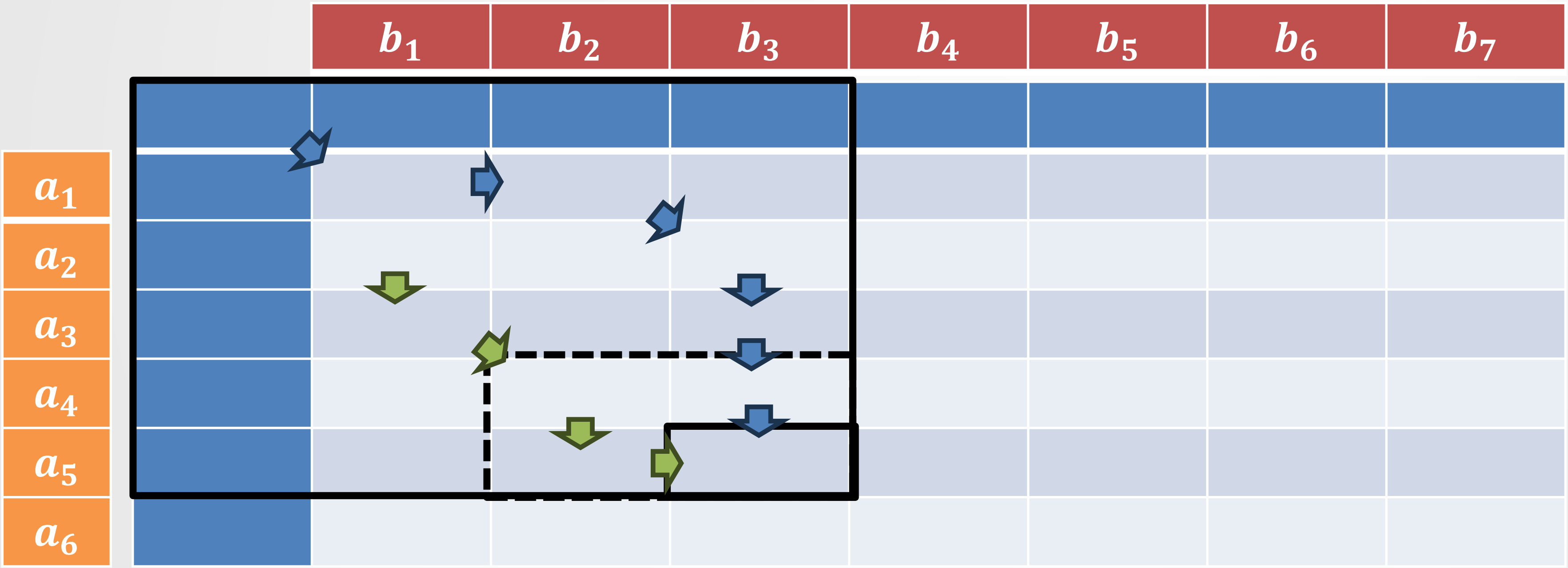
- We prove that $\{a_u, b_t\}$ must be part of $table[u, t]$
  - Suppose an alignment doesn't contain $\{a_u, b_t\}$
  - Recall: every element of $a_1, \dots, a_u$ must align with some $b_1, \dots, b_t$ without crossing (and vice versa)
  - Then $\{a_{u'}, b_t\}$ and $\{a_u, b_{t'}\}$ are in the alignment for some $u' < u$ and $t' < t$
  - But these alignments cross!



(choose one blue line and one green; they always cross)

UNIVERSITY OF TORONTO

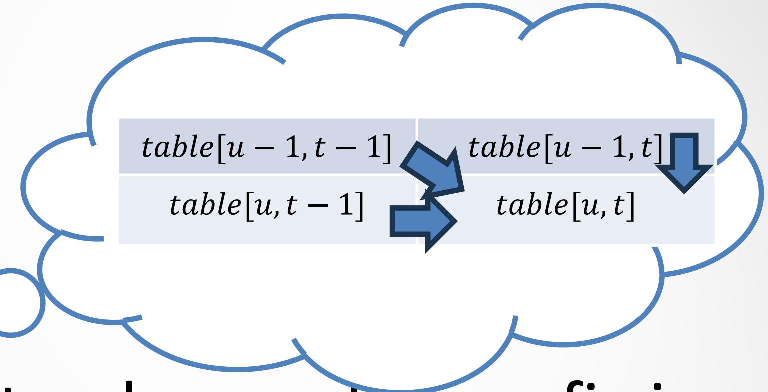
Which prefixes to extend?



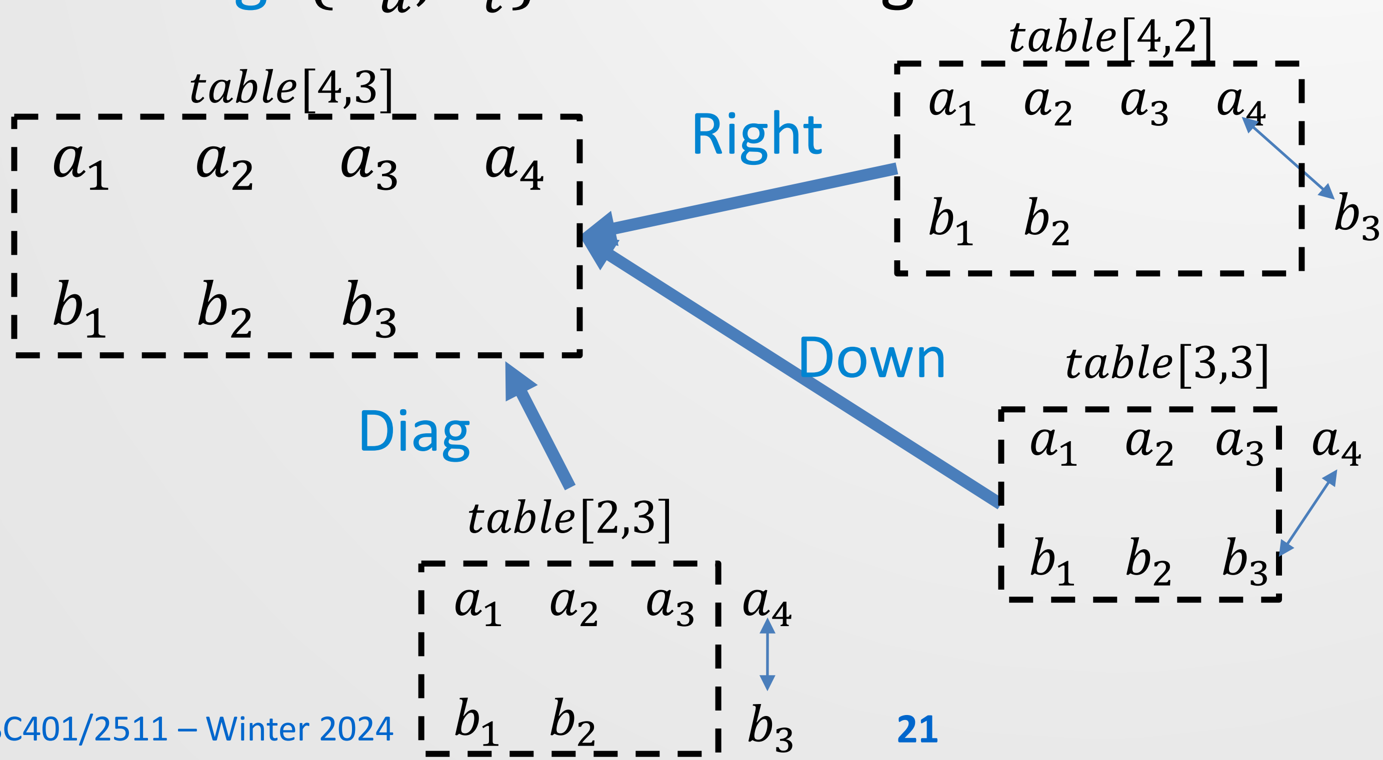
- Any cell $table[u, t]$ must include one of $\{u-1, t-1\}$, $\{u-1, t\}$, or $\{u, t-1\}$
- $table[u-1, t-1]$, $table[u-1, t]$, and $table[u, t-1]$ include these
- We **assume** a correct solution can be derived by extending these cells

UNIVERSITY OF TORONTO

Extending prefixes



- The alignments in $table[u,t]$ can extend an earlier prefix in one of three ways:
  - Right: b_t extends alignments in $table[u,t-1]$
  - Down: a_u extends alignments in $table[u-1,t]$
  - Diag: $\{a_u, b_t\}$ extends alignments in $table[u-1,t-1]$



UNIVERSITY OF TORONTO

General to specific

- To make our algorithm specific, we need answers to the following questions:
  1. What are a_u and b_t?
  2. What is a solution to a prefix?
  3. How do we build the initial prefix(es) (base case)?
  4. How do we extend a prefix correctly (recursive step)?
  5. How is the result computed from $table$?

- Let's look at some examples