ASR and Dynamic Programming

Automatic speech recognition

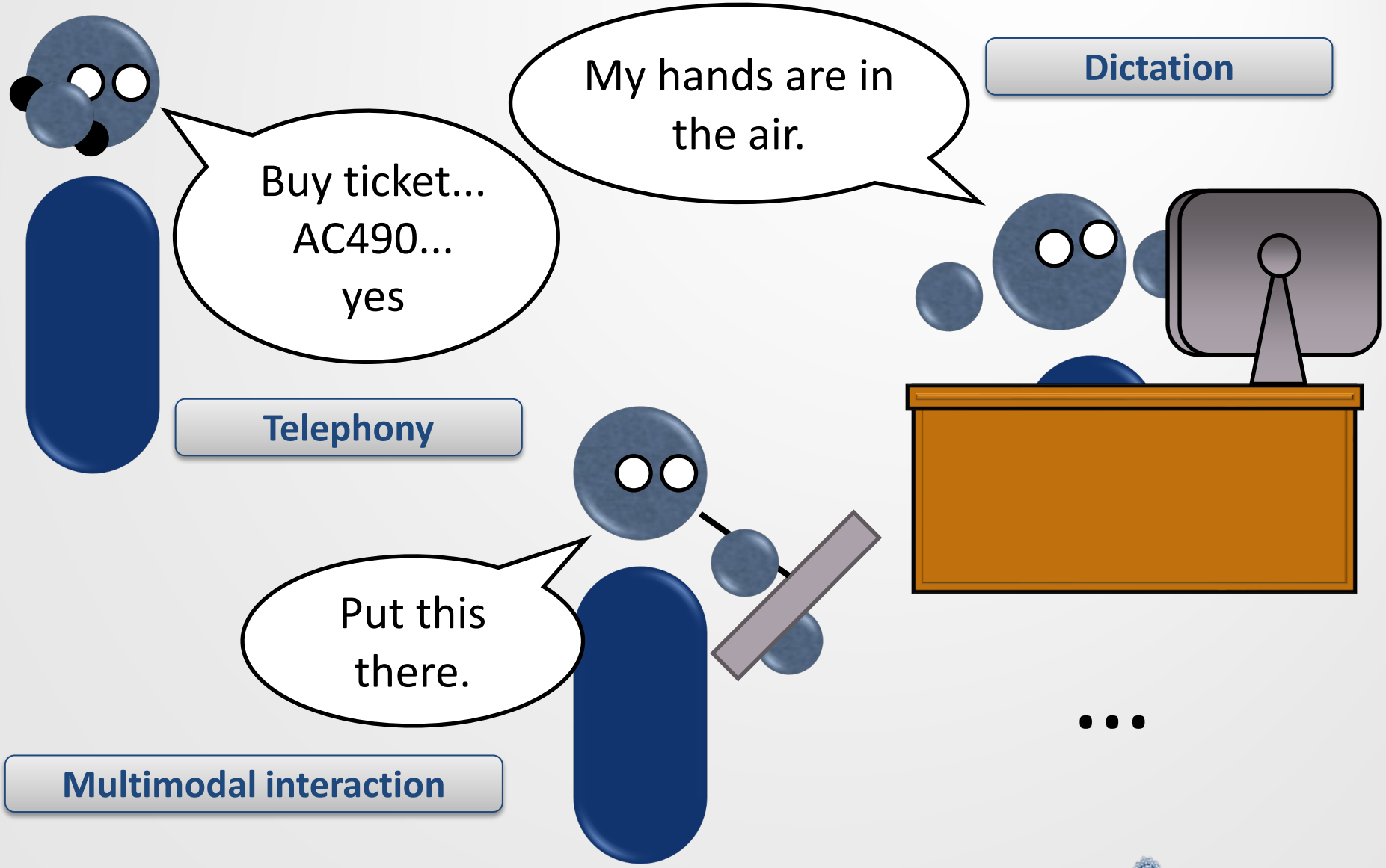
- Given an utterance, recorded as a waveform…

- 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`

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

$$x = x_1, x_2, \ldots, x_T, \qquad x_t \in \mathbb{R}^D$$

- We want a sequence of tokens (a transcription)

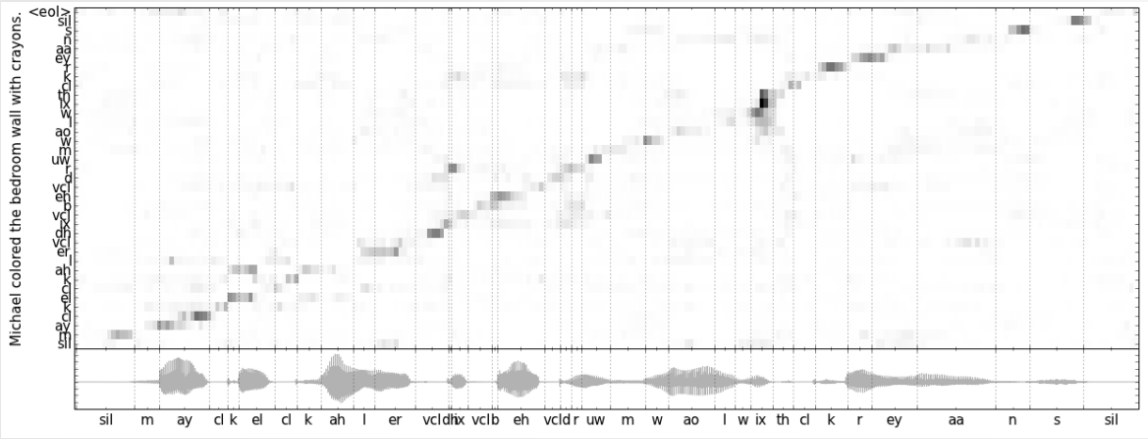
$$y = y_1, y_2, \ldots, y_U, \qquad y_u \in \{1,2, \ldots, V\}$$

  - 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)

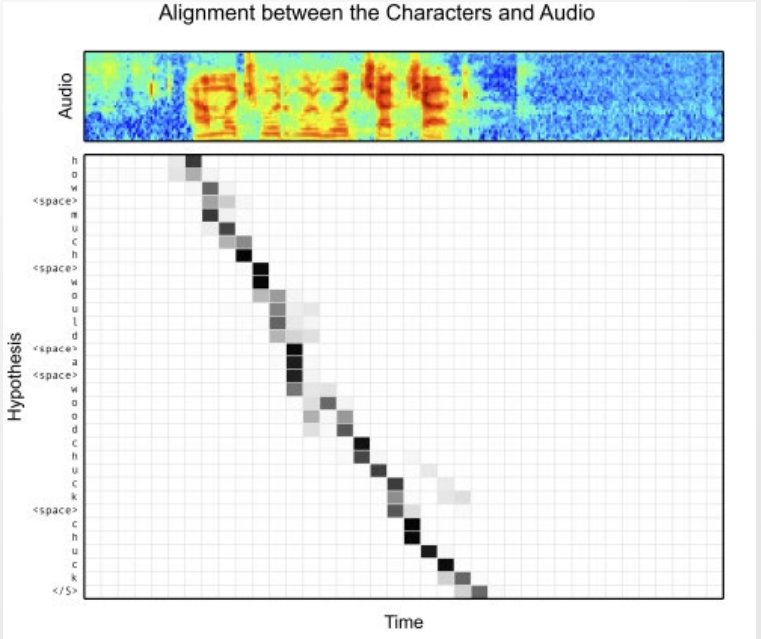
Encoder/decoder ASR

- 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

$$y^* = \text{argmax}_y P_\theta(y|x)$$

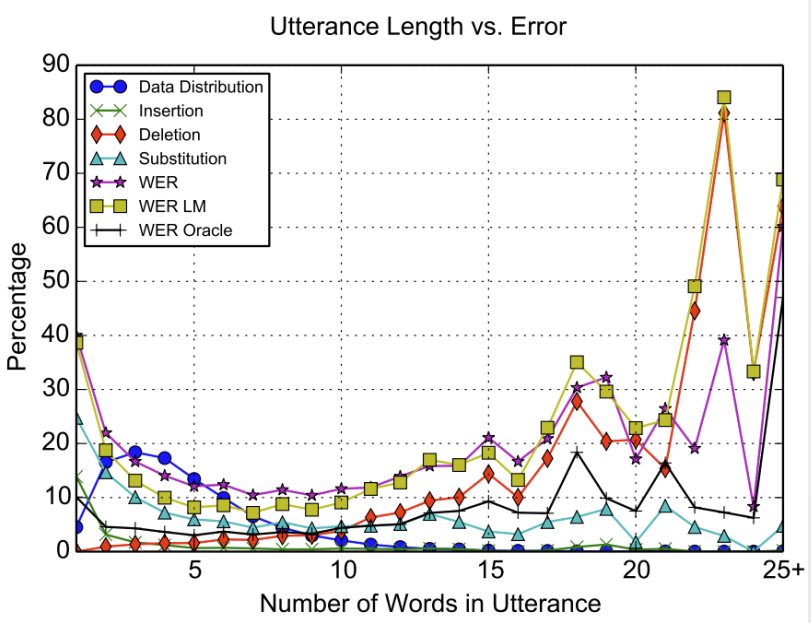
  with the beam search algorithm
- For best performance, an external, auto-regressive language model may be incorporated into each time step via shallow fusion:

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

- The impact of the external LM grows with λ

Pros and cons

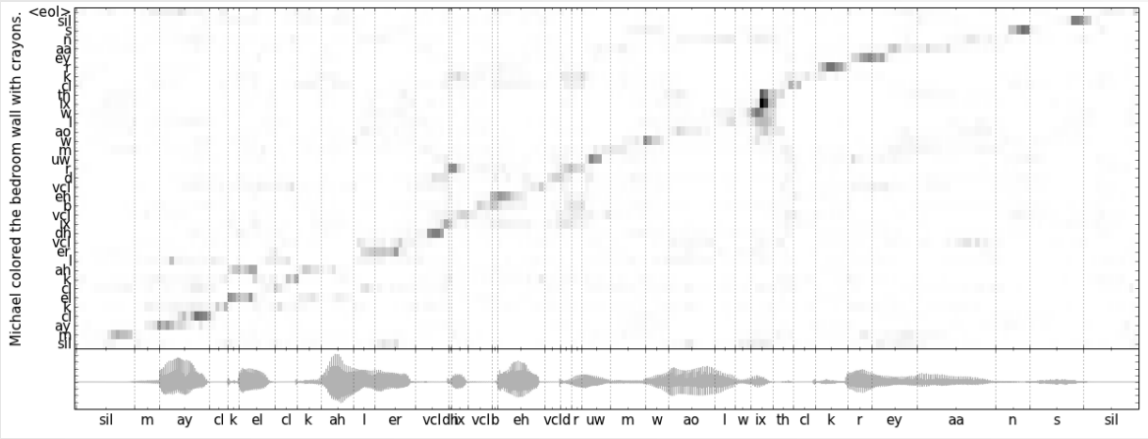
- Encoder/decoder ASR with Transformers are often state-of-the-art on ASR benchmarks
- They do have some drawbacks
  - They are unsuited to streaming (real-time transcription)
  - Performance suffers on long utterances



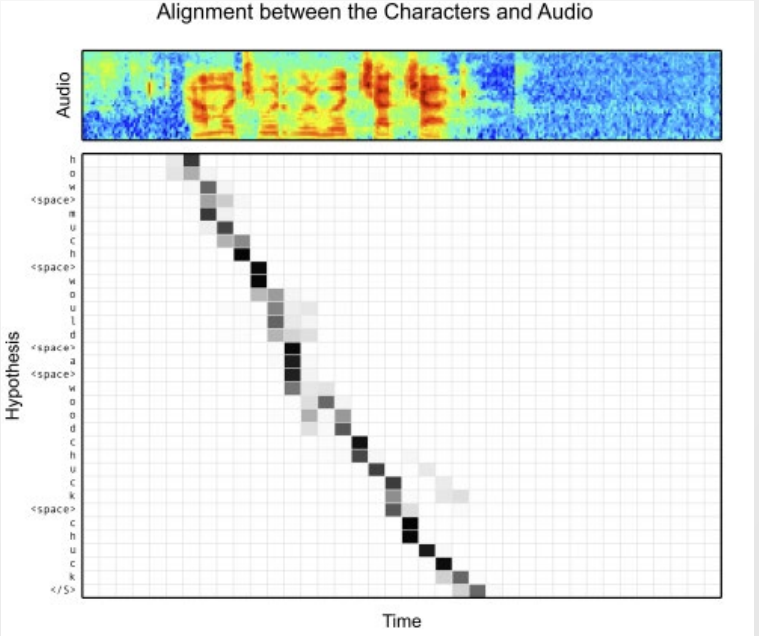
From Chan *et al*. (preprint) "Listen, attend, and spell"

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?**



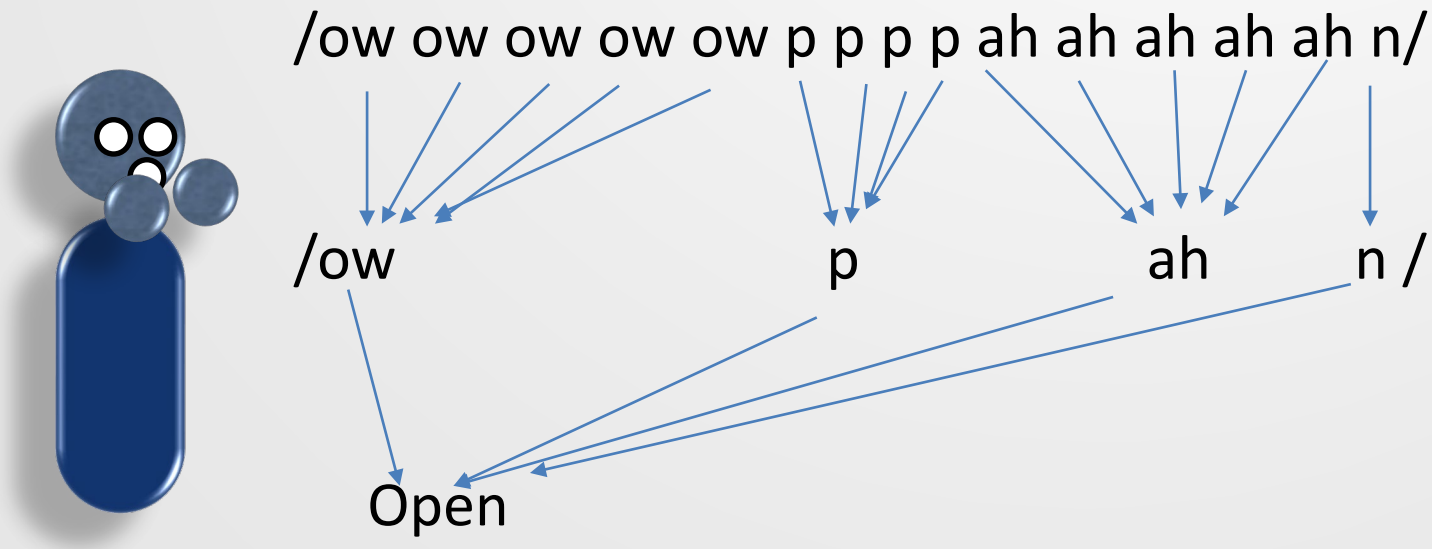
From Chorowski *et al*. (2014) "End-to-end continuous speech recognition using attention-based recurrent NN: First results"



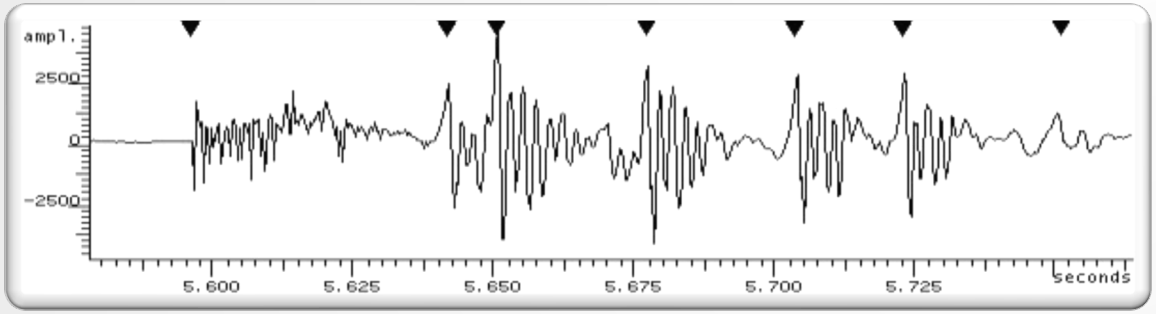
From Chan *et al*. (2016) "Listen, attend, and spell"

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



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

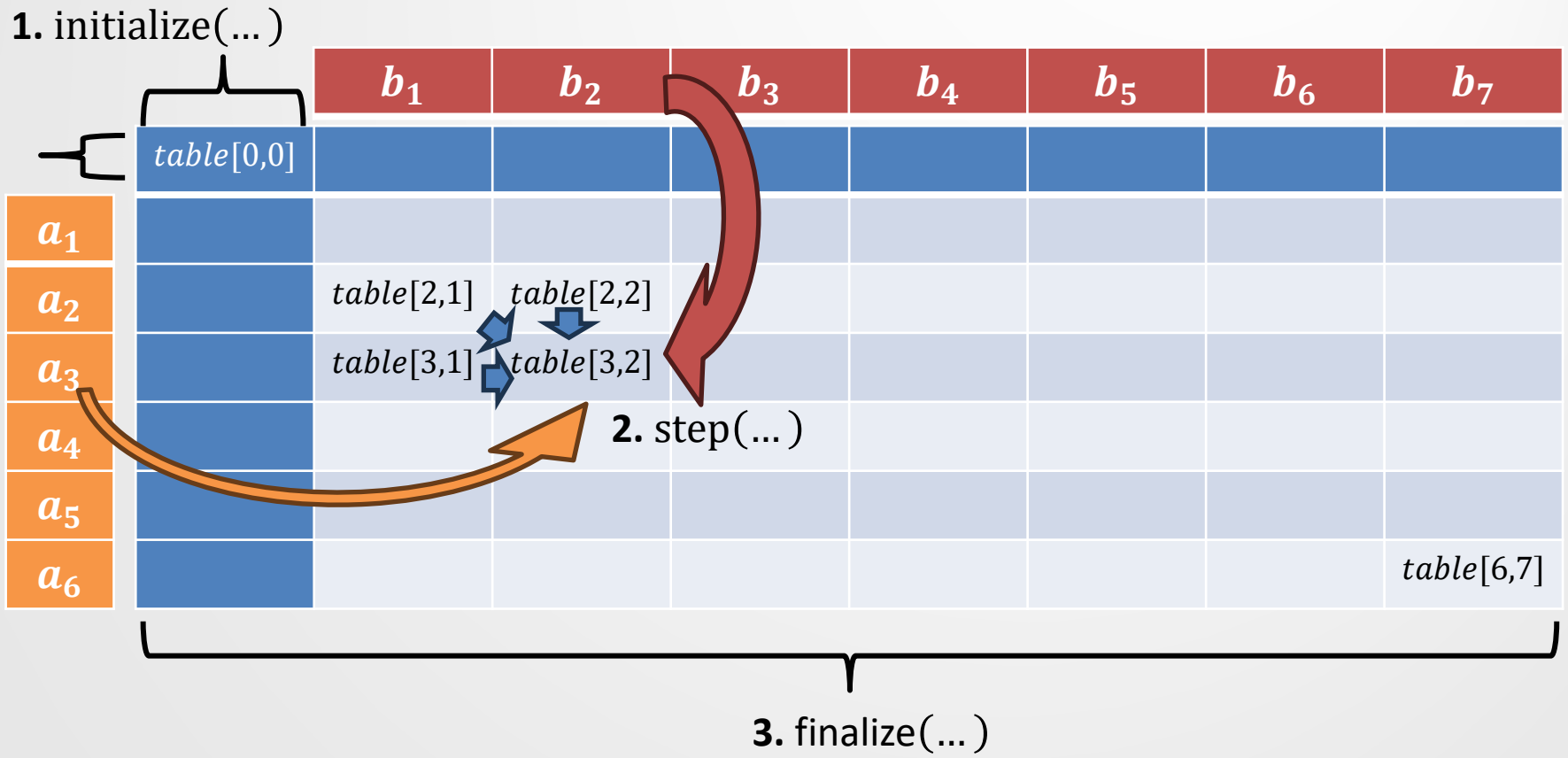
- We can solve both problems by learning monotonic alignments between sequences of frames and tokens
- To do so, both classic and end-to-end neural ASR rely on Dynamic Programming
  - Classic: Dynamic Time Warping (DTW), HMMs
  - E2E: Connectionist Temporal Classification (CTC), RNN-Transducer (RNN-T)
- 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

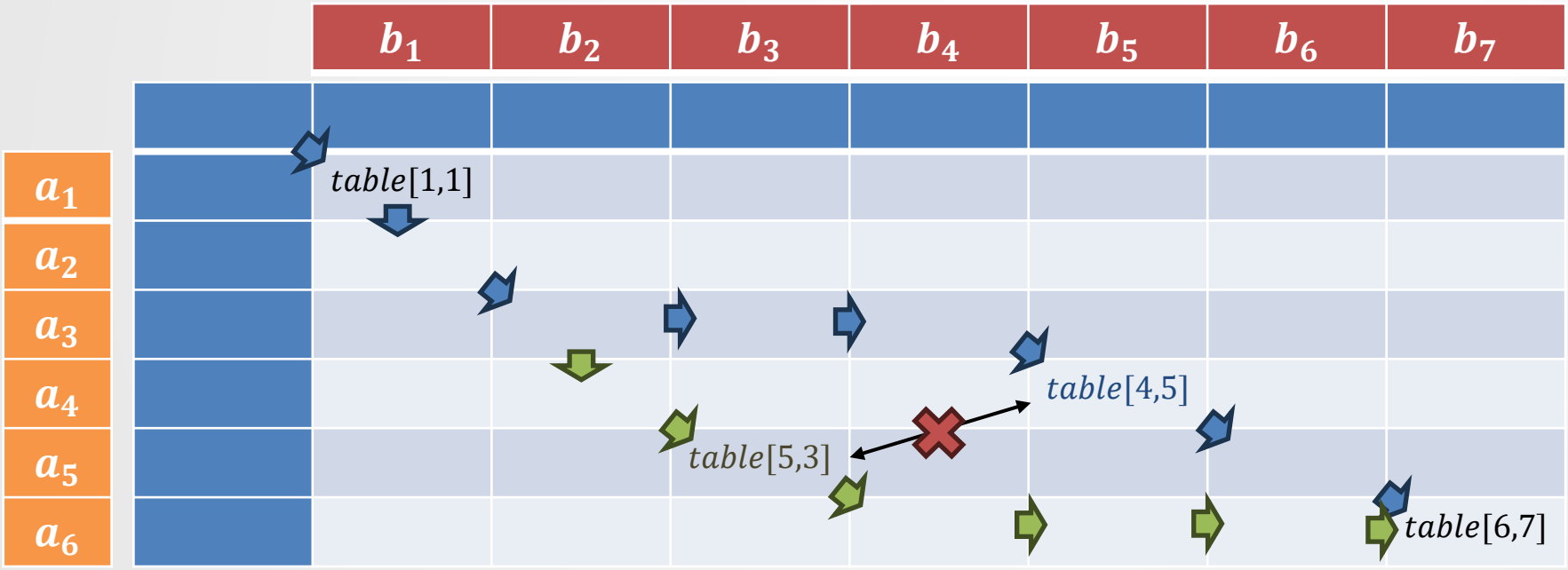
Function monotonic_forward
Inputs $a = a_1, a_2, \ldots, a_U$ **and** $b = b_1, b_2, \ldots, b_T$
1: **Define** $table[0 \ldots U, 0 \ldots T]$
2: initialize$(table[0 \ldots U, 0], table[0,1 \ldots T])$
3: **For each** u **in** $1 \ldots U$:
4: **For each** t **in** $1 \ldots T$:
5: $table[u,t] = \text{step}(a_u,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

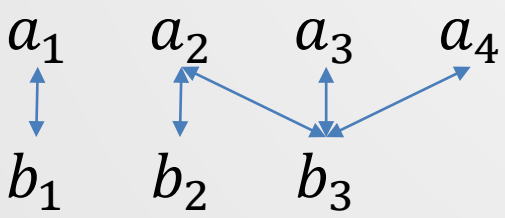


- Cells are populated in **non-decreasing** order of indices
  - $table[u,t]$ depends on all $table[u',t']$ where $u' \leq u$ and $t' \leq t$
  - **Not** when either $u' > u$ and/or $t' > t$

Which problems?

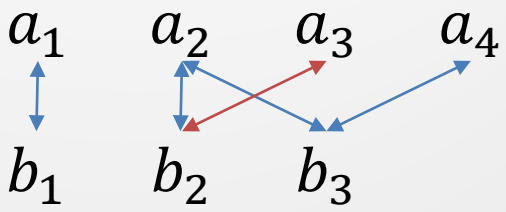
- We use monotonic_forward when solutions involve **aligning** each element of a to one from b and vice versa
- **All** elements of a and b must be **aligned without crossing**
- The solution may involve one or more alignments

A valid alignment



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

An invalid alignment



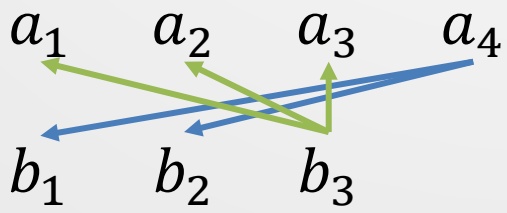
$\{a_1, b_1\} \leq \{a_2, b_2\} \leq \{a_2, b_3\} \ldots$
$\ldots ??? \{a_3, b_2\} \leq \{a_4, b_3\}$

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 $\{a_1, \ldots, a_u\}, \{b_1, \ldots, b_t\}$
- 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 $\{a_u, b_t\}$

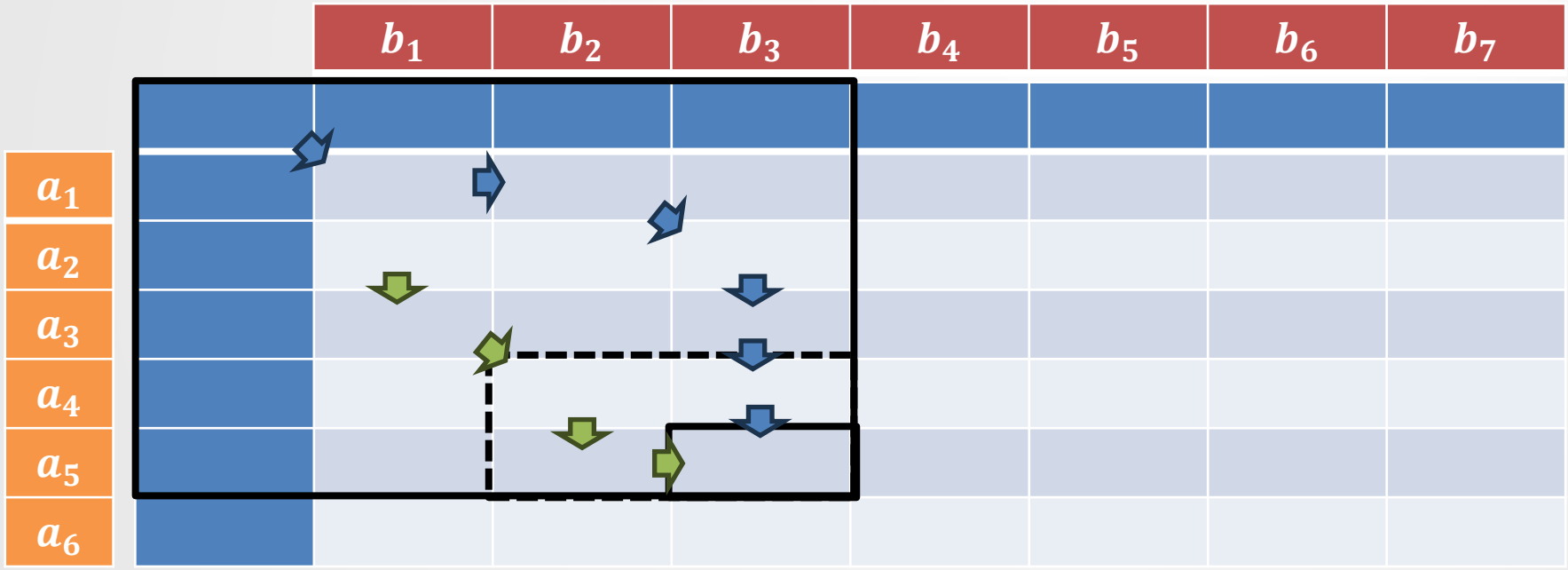
Aligning the last element

- 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 $a_1, \ldots, a_u$ must align with some $b_1, \ldots, b_t$ 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!



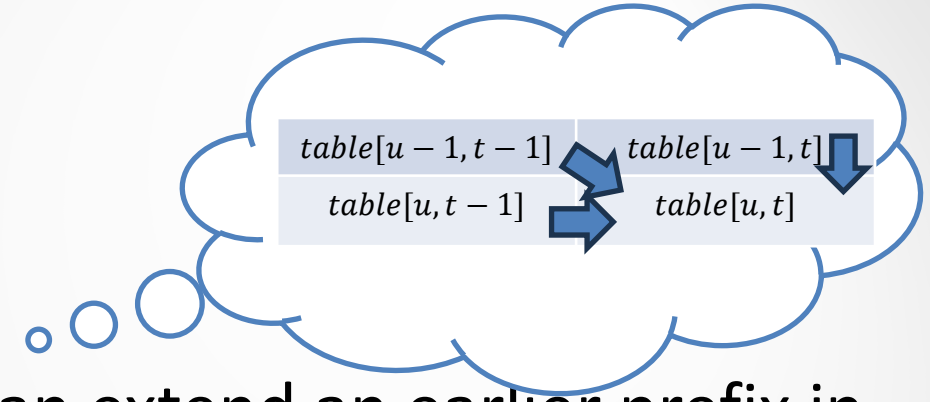
(choose one blue line and one green; they always cross)

Which prefixes to extend?

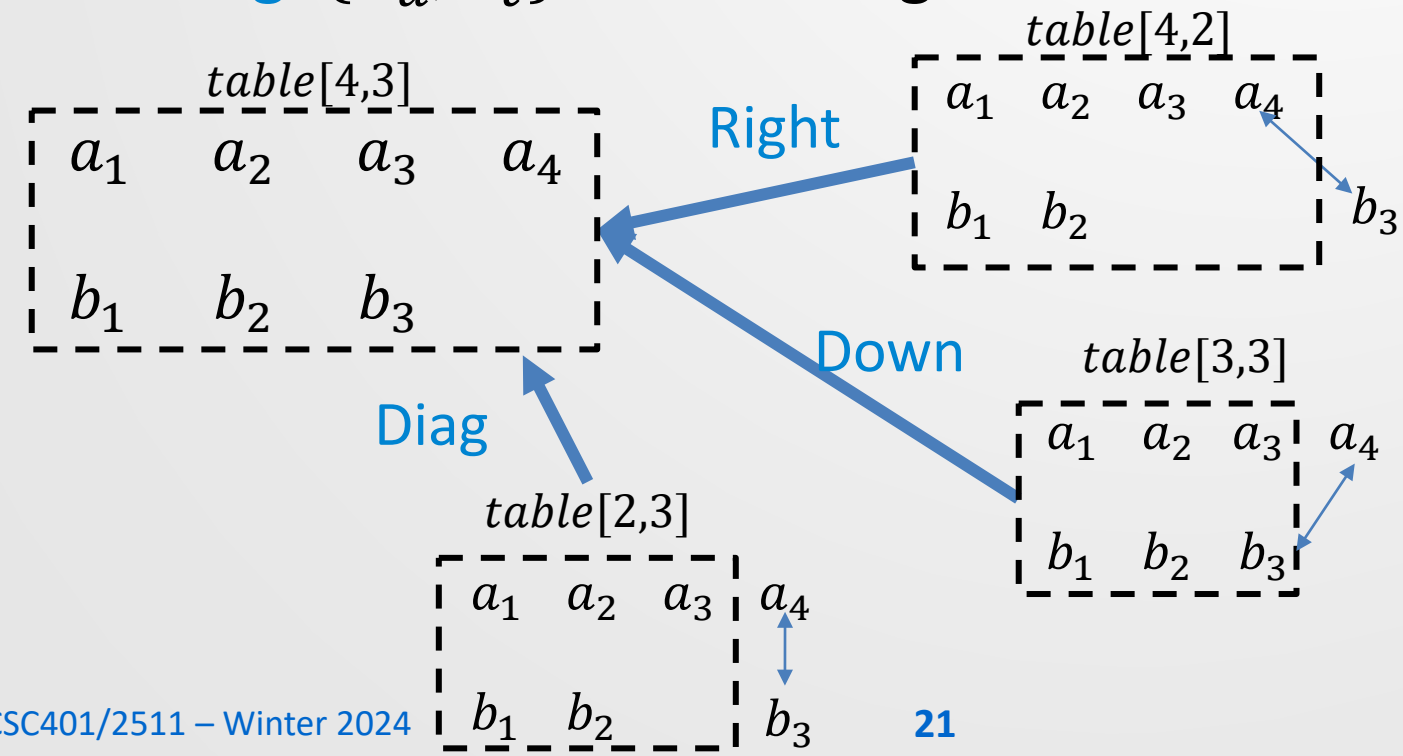


- Any cell $table[u, t]$ must include one of $\{u-1, t-1\}, \{u-1, t\}$, or $\{u, t-1\}$
- $table[u-1, t-1], table[u-1, t]$, and $table[u, t-1]$ include these
- We **assume** a correct solution can be derived by extending these cells

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 $table[u-1,t-1]$



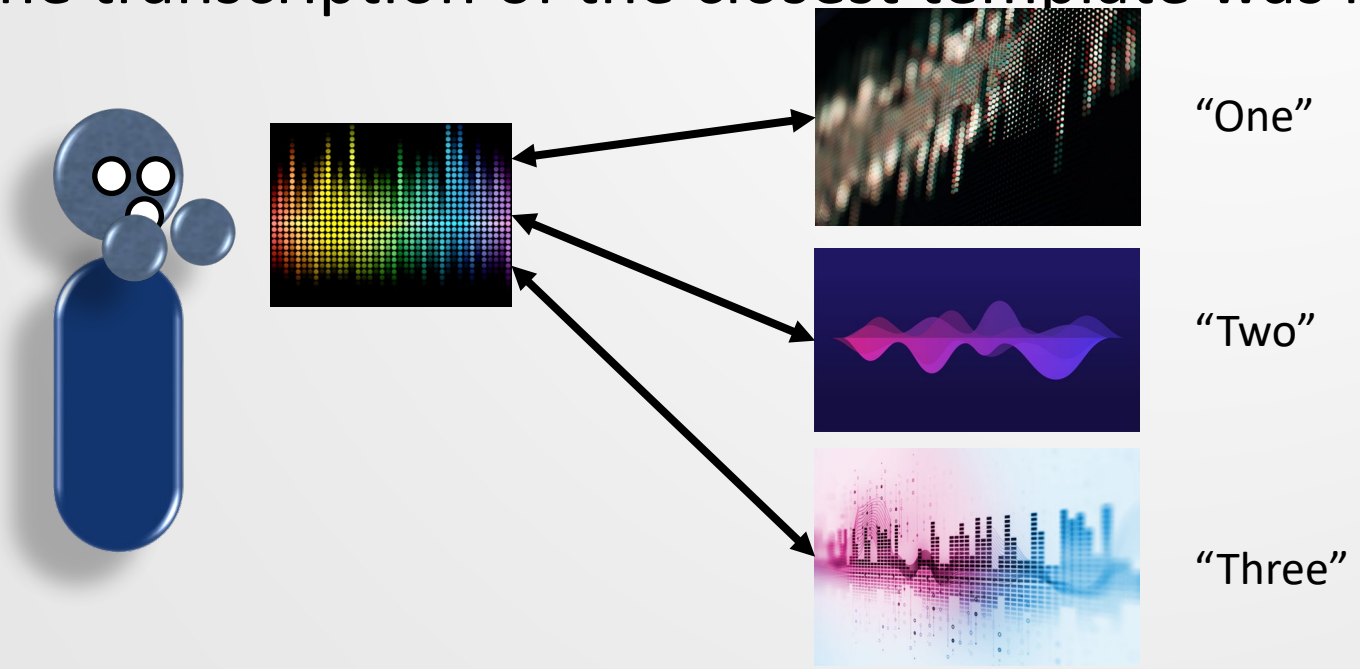
General to specific

- 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

Template matching in ASR

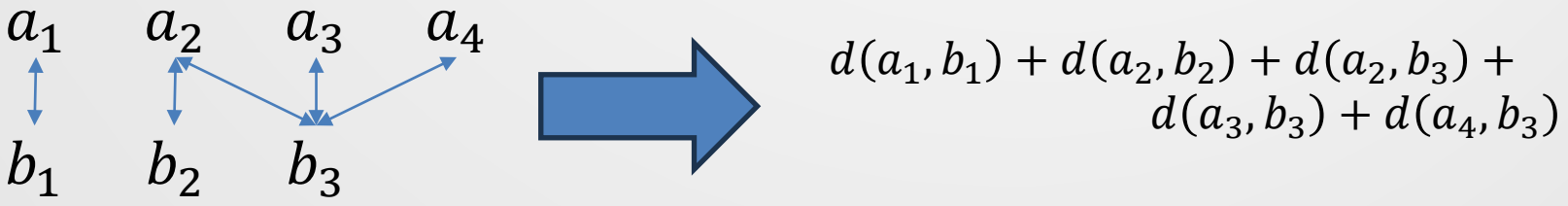
- Early ASR systems were based on template matching
- Input utterances were compared against stored templates
- The transcription of the closest template was returned



- How to deal with stretching and shrinking in time?

Aligning features

- Let $a \in \mathbb{R}^{U \times D}$ and $b \in \mathbb{R}^{T \times D}$ be seqs. of speech features
  - MFCCs, f-bank feats, *etc*.
- Choose some frame-wise (vector) distance function
  - E.g. $d(a_u, b_t) = \sum_{d=1}^{D} \left| a_{u,d} - b_{t,d} \right|$
- We can sum those frame-wise distances in a monotonic alignment to get a "distance" between utterances!

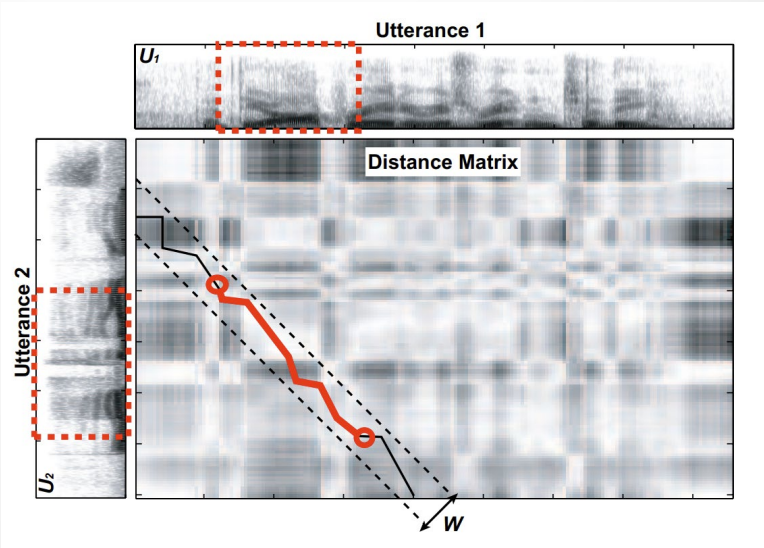


Dynamic time warping

- There are many possible ways to align a and b (call them $\mathcal{A}$)
- In **Dynamic Time Warping** (DTW), *the* "distance" between a and b is that with the minimal frame-wise sum

$$\mathcal{D}_d(a,b) = \min_{\alpha \in \mathcal{A}} \sum_{\{u,t\} \in \alpha} d(a_u, b_t)$$

- $\mathcal{D}_d(a,b)$ can be computed with monotonic_forward



From Park and Glass (2006) "Towards unsupervised pattern discovery in speech"

Specifying DTW

1. What are a_u and b_t? → Feature vectors of utts a and b
2. What is a solution to a prefix?
   $\rightarrow table[u,t] = \mathcal{D}_d(a_{1,\ldots,u}, b_{1,\ldots,t})$
3. How do we build the initial prefix(es)?
   $\rightarrow table[1,1] = d(a_1, b_1)$
4. How do we extend a prefix correctly?
   $$\rightarrow table[u,t] = d(a_u, b_t) + \min \begin{cases} table[u-1,t] \\ table[u-1,t-1] \\ table[u,t-1] \end{cases}$$
5. How is the result computed from $table$?
   $\rightarrow \mathcal{D}_d(a,b) = table[U,T]$

Adapting monotonic_forward

Function monotonic_forward
Inputs $a = a_1, a_2, \dots, a_U$ **and** $b = b_1, b_2, \dots, b_T$
1: **Define** $table[0 \dots U, 0 \dots T]$
2: initialize($table[0 \dots U, 0]$, $table[0,1 \dots T]$)
3: **For each** u **in** $1 \dots U$:
4: **For each** t **in** $1 \dots T$:
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($table[0 \dots U, 0 \dots T]$)

initialize: set $table[0,0] = 0$, $table[1 \dots U, 0] = table[0,1 \dots T] = \infty$

$$\text{step: } table[u,t] = d(a_u, b_t) + \min \begin{cases} table[u-1,t] \\ table[u-1,t-1] \\ table[u,t-1] \end{cases}$$

finalize: $table[U,T]$

A DTW example

		b_1	b_2	b_3
a_1		??+0	??+3	??+1
a_2		??+1	??+2	??+5
a_3		??+1	??+2	??+4
a_4		??+1	??+0	??+1

Since $d(a_u, b_t)$ is a fixed cost in $table[u, t]$, we denote it as "+ x"

A DTW example

		b_1	b_2	b_3
	0	∞	∞	∞
a_1	∞	??+0	??+3	??+1
a_2	∞	??+1	??+2	??+5
a_3	∞	??+1	??+2	??+4
a_4	∞	??+1	??+0	??+1

Initialize row 0 and column 0

A DTW example

		b_1	b_2	b_3
	0	∞	∞	∞
a_1	∞	$0+0$	$0+3$	$3+1$
a_2	∞	$??+1$	$??+2$	$??+5$
a_3	∞	$??+1$	$??+2$	$??+4$
a_4	∞	$??+1$	$??+0$	$??+1$

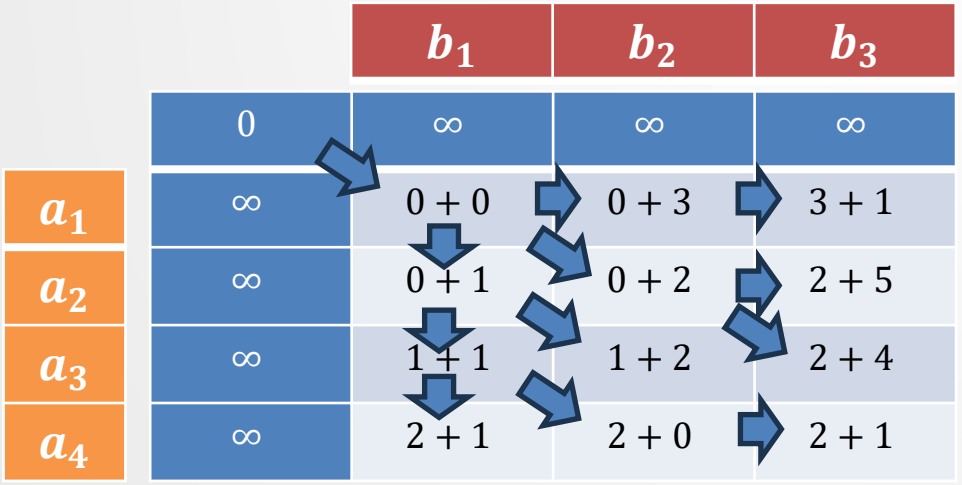
Fill row 1

A DTW example

		b_1	b_2	b_3
	0	∞	∞	∞
a_1	∞	$0+0$	$0+3$	$3+1$
a_2	∞	$0+1$	$0+2$	$2+5$
a_3	∞	$??+1$	$??+2$	$??+4$
a_4	∞	$??+1$	$??+0$	$??+1$

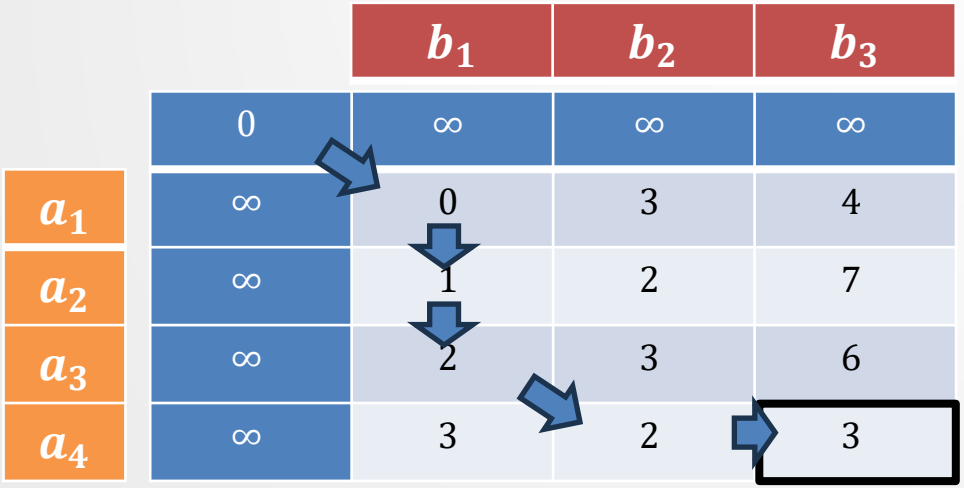
Fill row 2

A DTW example



… and so forth

A DTW example



Return $table[4,3] = 3$

ASR evaluation

- ASR systems often generate **hypothesis** transcripts which don't match the gold-standard **reference** transcript
- But some are closer than others:
  - Reference: `errors are common here`
  - Hypothesis 1: `his errors are commas here`
  - Hypothesis 2: `here are are`
- We may describe the errors in terms of transformations:
  - Hypothesis 1 **inserted** `his` and **substituted** `commas` for `common`
  - Hypothesis 2 **substituted** `here` for `errors`, `are` for `common`, and **deleted** `here`
    - The `here`s cannot match without re-ordering

Word-error rate (WER)

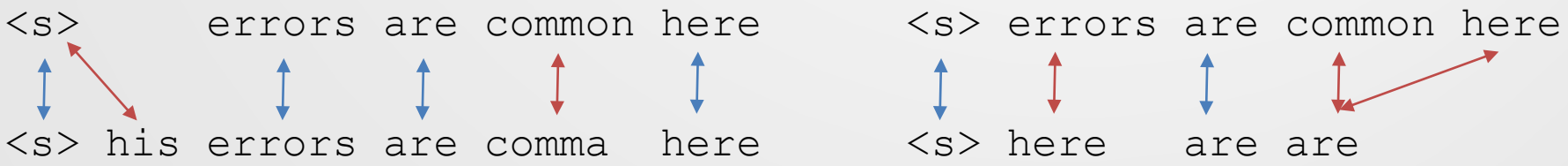
- ASR enthusiasts are often concerned with **word-error rate (WER)**, which counts the **minimum** number of 3 types of errors a hypothesis makes given a reference
  - **Substitution error**: One word being mistook for another
    e.g., hyp: **`shift`**, ref: `ship`
  - **Deletion error**: An input word that is 'skipped'
    e.g., hyp: `I Torgo`, ref: `I` **`am`** `Torgo`
  - **Insertion error**: A 'hallucinated' word not in the input.
    e.g., hyp: **`steamed`** `hams`, ref: `hams`
- Given S substitutions, D deletions, I insertions, and N reference tokens:

$$WER = \frac{S + D + I}{N} \times 100\%$$

(this is not a valid percentage)

Aligning strings

- We can count errors by building up prefixes of alignments between reference (a) and hypothesis (b)
  - Insertion: add b_t to hypothesis
  - Deletion: add a_u to reference
  - Match/substitution: add b_t, a_u
- Prepend ref + hyp with a special token <s> to allow for insertions/deletions at the beginning of the reference/hypothesis



Specifying WER

1. What are a_u and b_t? → Reference and hypothesis tokens
2. What is a solution to a prefix?

   $\rightarrow table[u,t] = \min \#errors(<\text{s}>a_{1,\dots,u}, <\text{s}>b_{1,\dots,t})$
3. How do we build the initial prefix(es)?

   $\rightarrow table[u,0] = u,\ table[0,t] = t$
4. How do we extend a prefix correctly?

$$\rightarrow table[u,t] = \min \begin{cases} table[u-1,t]+1 \\ table[u-1,t-1] + \begin{cases} 0 & a_u = b_t \\ 1 & a_u \neq b_t \end{cases} \\ table[u,t-1]+1 \end{cases}$$

5. How is the result computed from $table$?

$$\rightarrow WER(a,b) = \frac{table[U,T]}{U} \times 100\%$$

Adapting monotonic_forward

Function monotonic_forward
Inputs $a = a_1, a_2, \dots, a_U$ **and** $b = b_1, b_2, \dots, b_T$
1: **Define** $table[0 \dots U, 0 \dots T]$
2: initialize($table[0 \dots U, 0]$, $table[0,1 \dots T]$)
3: **For each** u **in** $1 \dots U$:
4: **For each** t **in** $1 \dots T$:
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($table[0 \dots U, 0 \dots T]$)

initialize: set $table[u,0] = u$, $table[0,t] = t$

$$\text{step: } table[u,t] = \min \begin{cases} table[u-1,t] + 1 \\ table[u-1,t-1] + \begin{cases} 0 & a_u = b_t \\ 1 & a_u \neq b_t \end{cases} \\ table[u,t-1] + 1 \end{cases}$$

$$\text{finalize: } \frac{table[U,T]}{U} \times 100\%$$

A WER example

	<s>	his	errors	are	comma	here
<s>						
errors						
are						
common						
here						

Rows (a_u) are reference tokens, columns (b_t) are hypothesis tokens

A WER example

	<s>	his	errors	are	comma	here
<s>	0	1	2	3	4	5
errors	1					
are	2					
common	3					
here	4					

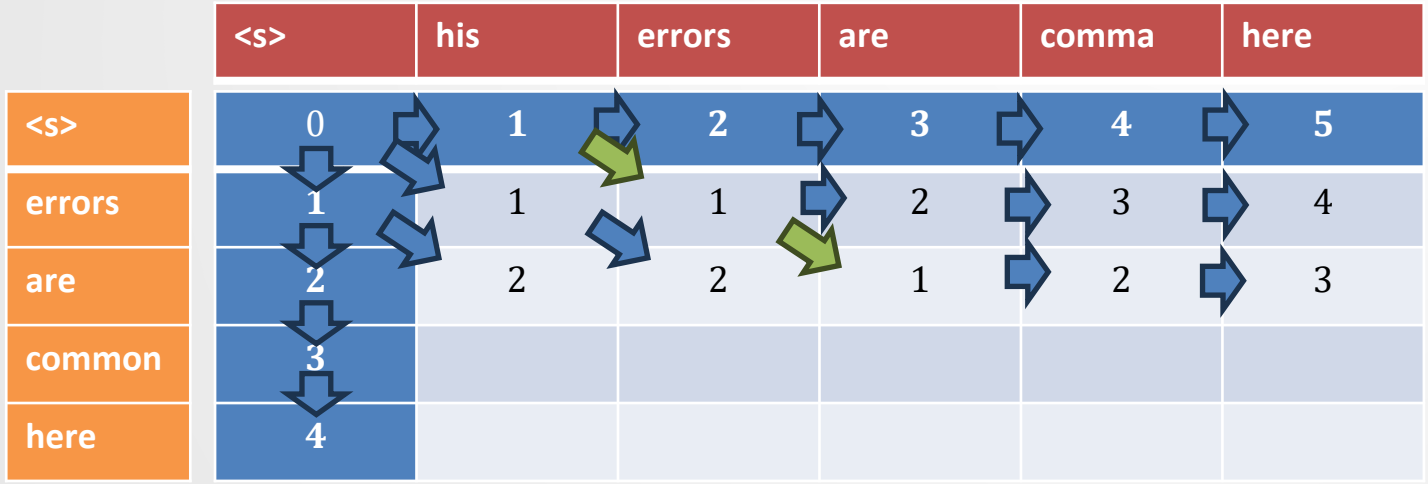
- Initialize row 0 and column 0
  - Row 0 inserts hypothesis tokens
  - Column 0 deletes references tokens

A WER example

	<s>	his	errors	are	comma	here
<s>	0	1	2	3	4	5
errors	1	1	1	2	3	4
are	2					
common	3					
here	4					

- Fill row 1
  - Note: `errors` match

A WER example



	<s>	his	errors	are	comma	here
<s>	0	1	2	3	4	5
errors	1	1	1	2	3	4
are	2	2	2	1	2	3
common	3					
here	4					

- Fill row 2
  - Note: `are` match
  - Note: this priority goes: match > sub > ins > del

A WER example

	<s>	his	errors	are	comma	here
<s>	0	1	2	3	4	5
errors	1	1	1	2	3	4
are	2	2	2	1	2	3
common	3	3	3	2	2	3
here	4	4	4	3	3	2

- …and so on

A WER example

	<s>	his	errors	are	comma	here
<s>	0	1	2	3	4	5
errors	1	1	1	2	3	4
are	2	2	2	1	2	3
common	3	3	3	2	2	3
here	4	4	4	3	3	2

- Return $\frac{table[4,5]}{4} \times 100\% = 50\%$

Returning to ASR

- Recall: each frame gets a label by repeating transcript tokens

/ow ow ow ow ow p p p p ah ah ah ah ah n/

/ow p ah n /

- Let $a_{1,\ldots,U}$ be the **reference transcript**, $b_{1,\ldots,T}$ be **a** frame-wise transcript with repetitions of a
- It is easy to train an RNN or Transformer to maximize the likelihood of b

$$\mathcal{L} = -\log P_\theta(b)$$

- But there are many choices of b!

Marginalization

- Solution: maximize the likelihood of all such b!
- Let $\mathcal{B}$ be a function which **removes sequential repetitions** from b: $\mathcal{B}(\mathrm{A\ B\ B\ A}) = \mathrm{A\ B\ A}$
- Let $\mathcal{B}^{-1}(a; T)$ be all b of length T which reduce to a

$$\mathcal{B}^{-1}(a; T) = \left\{b_{1,\ldots,T} : \mathcal{B}(b) = a\right\}$$

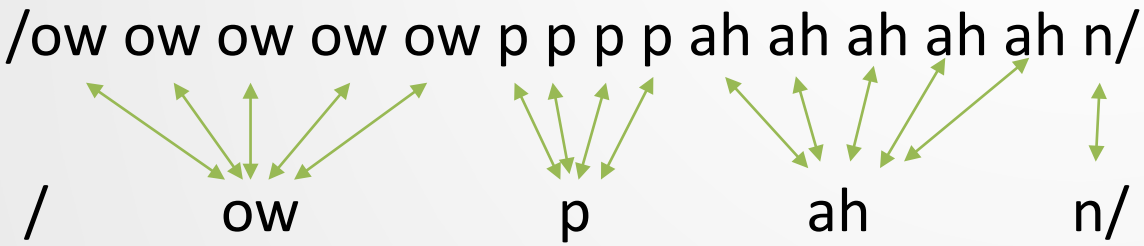
- Then we minimize

$$\mathcal{L} = -\log P_\theta(a) = -\log \sum_{b \in \mathcal{B}^{-1}(a;T)} P_\theta(b)$$

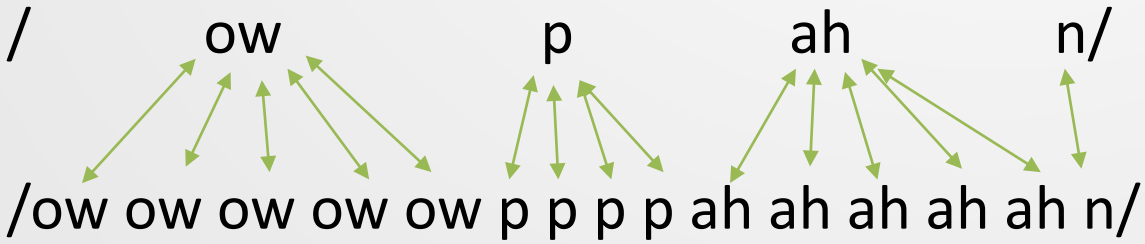
- However, there are $\binom{T-1}{U-1}$ paths in $\mathcal{B}^{-1}(a; T)$
- Can we use monotonic_forward?

Monotonic alignments

- If $\mathcal{B}(b) = a$, a monotonic alignment exists between (a, b)



- But **not all** monotonic alignments (a, b) imply $\mathcal{B}(b) = a$



- We can fix this by disallowing "down" (adding a_u to a prefix)

Partial solutions

- For DTW and WER, $table[u, t]$ keeps track of **one** optimal alignment per prefix pair $a_{1,\dots,u}, b_{1,\dots,t}$

$$table[u, t] = \min \#\text{errors}\left(a_{1,\dots,u}, b_{1,\dots,t}\right)$$

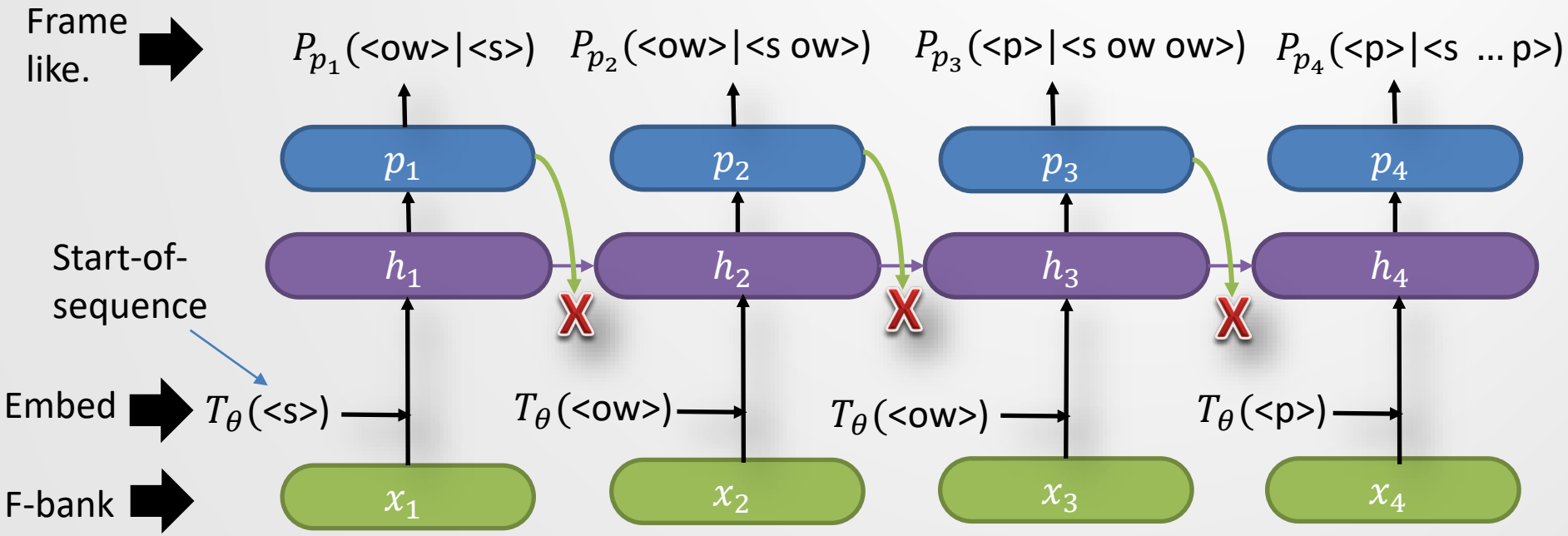
- For ASR, $table[u, t]$ keeps track of **all** alignments per prefix pair $a_{1,\dots,u}, b_{1,\dots,t}$

$$table[u, t] = \sum_{b_{1,\dots,t} \in \mathcal{B}^{-1}(a_{1,\dots,u}; t)} P_\theta\left(b_{1,\dots,t} | x\right)$$

- Then $\mathcal{L} = -\log table[U, T]$

An auto-regressive approach?

- Suppose we use an auto-regressive RNN to generate frame-level predictions $P_\theta(b_t | b_{1,\ldots,t-1}, x)$

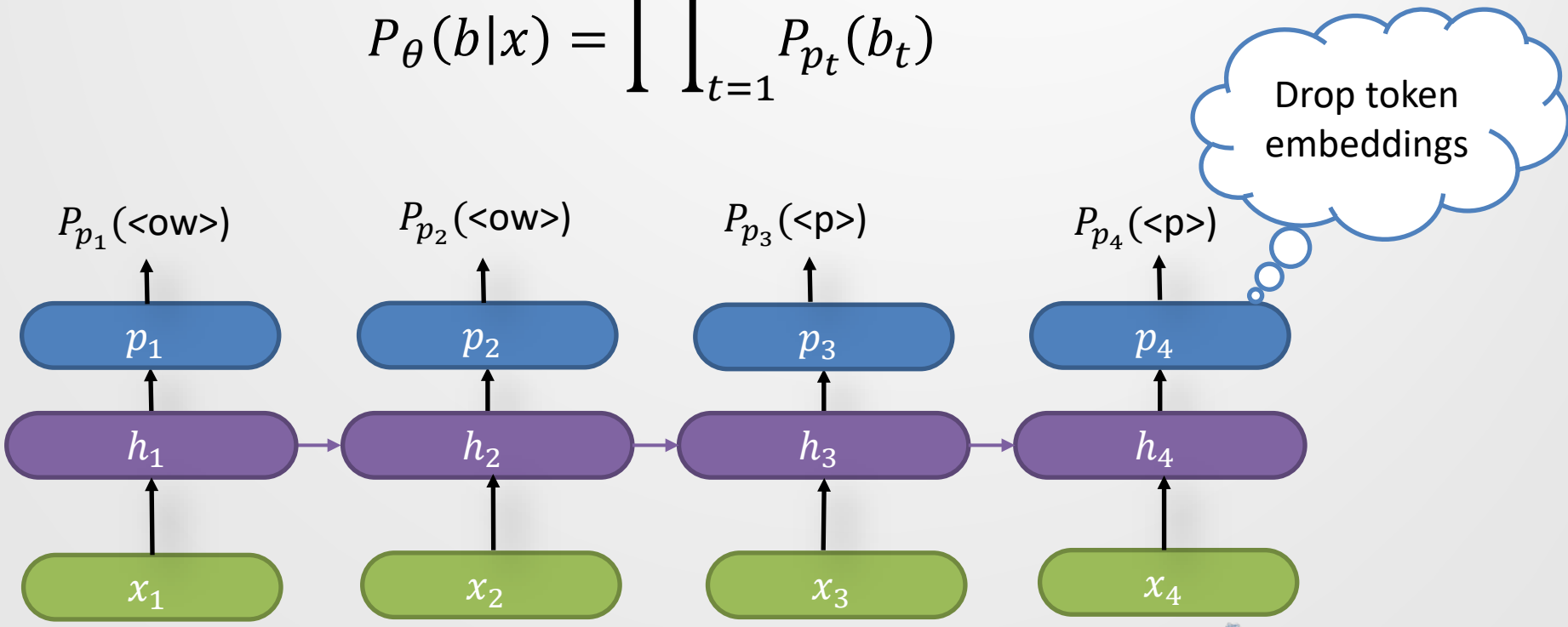


- Can we use this with monotonic_forward?

Conditional independence

- **No.** The frame-wise likelihoods depend on a specific prefix
  - $P_\theta(\text{<p>}|\text{<ow ow>}) \neq P_\theta(\text{<p>}|\text{<ow p>})$
  - We cannot share computations over prefixes
- We require elements of b to be **conditionally independent** given x:

$$P_\theta(b|x) = \prod_{t=1}^{T} P_{p_t}(b_t)$$



Extending prefixes

- For $b_{1,\dots,t} \in \mathcal{B}^{-1}(a_{1,\dots,u}; t)$, let
  - $b_{1,\dots,t}^{u-1}$ denote a prefix where a_u is first aligned to b_t
    - a_u aligns to b_t **and** a_{u-1} aligns to b_{t-1}
  - $b_{1,\dots,t}^{u}$ denote a prefix where a_u has already been aligned
    - a_u aligns to b_t **and** a_u aligns to b_{t-1}
- $\therefore a_u$ aligns to b_t **and** b_{t-1} aligns to a_u **or** a_{u-1}
- Using conditional independence, we have

$$table[u,t] = \sum_{b_{1,\dots,t} \in \mathcal{B}^{-1}(a_{1,\dots,u};\, t)} P_\theta(b_{1,\dots,t}|x)$$

$$= P_{p_t}(b_t = a_u) \sum_{b_{1,\dots,t} \in \mathcal{B}^{-1}(a_{1,\dots,u};\, t)} P_\theta(b_{1,\dots,t-1}|x)$$

$$= P_{p_t}(b_t = a_u) \left(\sum_{b_{1,\dots,t}^{u-1}} P_\theta(\underbrace{b_{1,\dots,t-1}^{u-1}}_{\in \mathcal{B}^{-1}(a_{1,\dots,u-1};\, t-1)}|x) + \sum_{b_{1,\dots,t}^{u}} P_\theta(\underbrace{b_{1,\dots,t-1}^{u}}_{\in \mathcal{B}^{-1}(a_{1,\dots,u};\, t-1)}|x) \right)$$

$$= P_{p_t}(b_t = a_u)(table[u-1, t-1] + table[u, t-1])$$

Specifying the ASR loss

1. What are a_u and b_t? → Reference and frame-level tokens
2. What is a solution to a prefix?

   $\rightarrow table[u,t] = \sum_{b_{1,\dots,t} \in \mathcal{B}^{-1}(a_{1,\dots,u};\, t)} P_\theta\left(b_{1,\dots,t} | x\right)$
3. How do we build the initial prefix(es)?

   $\rightarrow table[0,0] = 1,\ table[0,1 \dots T] = table[1 \dots U, 0] = 0$
4. How do we extend a prefix correctly?

   $\rightarrow table[u,t] = P_{p_t}(b_t = a_u)(table[u-1, t-1] + table[u, t-1])$
5. How is the result computed from $table$?

   $\rightarrow \mathcal{L} = -\log table[U,T]$

Adapting monotonic_forward

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($table[0 \ldots U, 0 \ldots T]$)

initialize: set $table[0,0] = 1$, $table[0,1 \ldots T] = table[1 \ldots U, 0] = 0$

step: $P_{p_t}(b_t = a_u)(table[u-1, t-1] + table[u, t-1])$

finalize: $-\log table[U, T]$

An ASR example

		b_1	b_2	b_3	b_4
a_1		?? × 0.1	?? × 0.3	?? × 0.1	?? × 0.1
a_2		?? × 0.1	?? × 0.2	?? × 0.5	?? × 0.1
a_3		?? × 0.1	?? × 0.2	?? × 0.4	?? × 0.1

Since $P_{p_t}(b_t = a_u)$is a fixed cost in $table[u, t]$, we denote it as “x x”

An ASR example

		b_1	b_2	b_3	b_4
	1	0	0	0	0
a_1	0	?? × 0.1	?? × 0.3	?? × 0.1	?? × 0.1
a_2	0	?? × 0.1	?? × 0.2	?? × 0.5	?? × 0.1
a_3	0	?? × 0.1	?? × 0.2	?? × 0.4	?? × 0.1

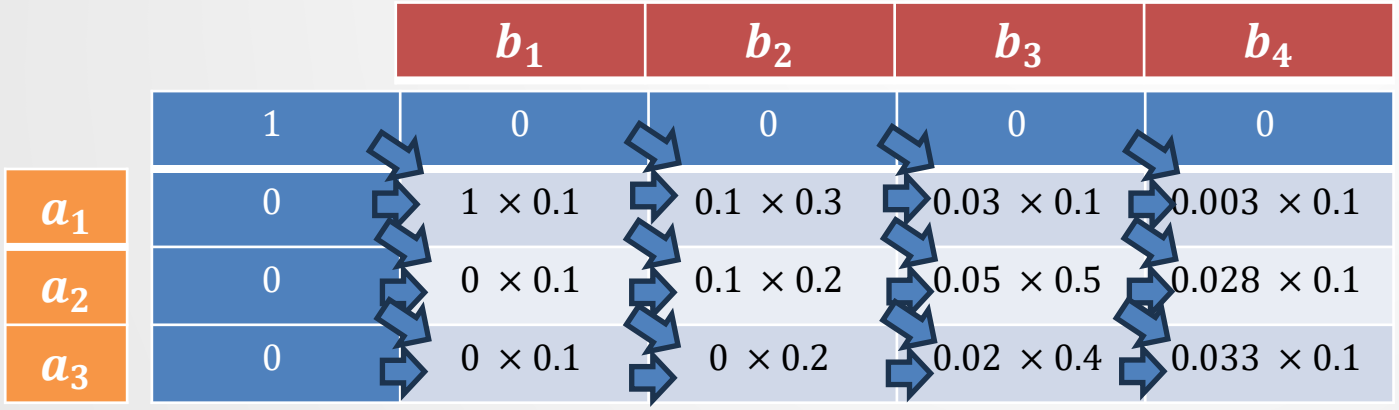
Initialize the first row and column

An ASR example

	b_1	b_2	b_3	b_4
	1	0	0	0	0
a_1	0	1×0.1	0.1×0.3	0.03×0.1	0.003×0.1
a_2	0	$?? \times 0.1$	$?? \times 0.2$	$?? \times 0.5$	$?? \times 0.1$
a_3	0	$?? \times 0.1$	$?? \times 0.2$	$?? \times 0.4$	$?? \times 0.1$

Fill row 1

An ASR example



… and so on

An ASR example

		b_1	b_2	b_3	b_4
	1	0	0	0	0
a_1	0	0.1	0.03	0.003	0.0003
a_2	0	0	0.02	0.025	0.0028
a_3	0	0	0	0.008	0.0033

Return $-\log table[U,T] = -\log 0.0033 \approx 5.7$

Inference

- We can compute an error signal for $\mathcal{L} = -\log P_\theta(a|x)$

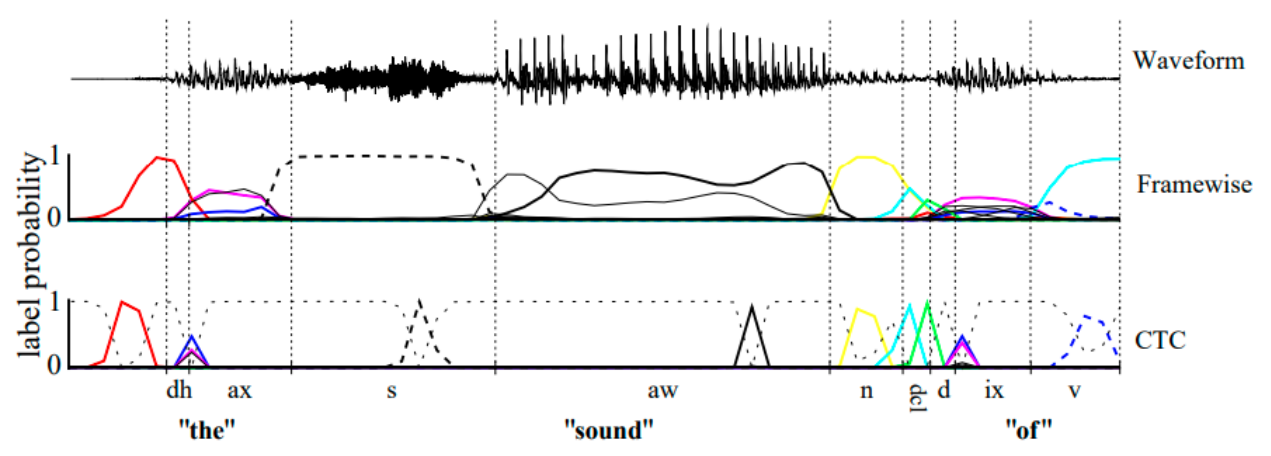
- At test time, how do we generate transcriptions a?
- Computing $b^* = \text{argmax}_b P_\theta(b|x)$ is **easy**
  - $b^* = \text{argmax}_b \prod_{t=1}^{T} P_{p_t}(b_t) \Rightarrow b_t^* = \text{argmax}_{b_t} P_{p_t}(b_t)$
- Computing $a^* = \text{argmax}_a P_\theta(a|x)$ is **hard**
  - We use DP to compute $P_\theta(a|x)$ **once**
  - For vocab size V, there are $\frac{V(V^T-1)}{V-1}$ possible a
  - Vanilla beam search on b will just return b^*

Prefix search (sketch)

- We use a modified beam search called **prefix search**
- Keeps track of K prefixes of a, **not** b
- For each frame t
  - Extend each prefix $a^{(k)}$ with each element of the vocabulary v and frame-level likelihoods $P_{p_t}(v)$: $a^{(k,v)}$
  - NEW **Collapse** any repeats in extensions using $\mathcal{B}$, summing the likelihoods of matching prefixes: $\mathcal{B}\left(a^{(k,v)}\right) = \mathcal{B}\left(a^{(k',v')}\right)$
  - Pick the top K most likely extensions as new $a^{(k)}$

Dealing with doubles

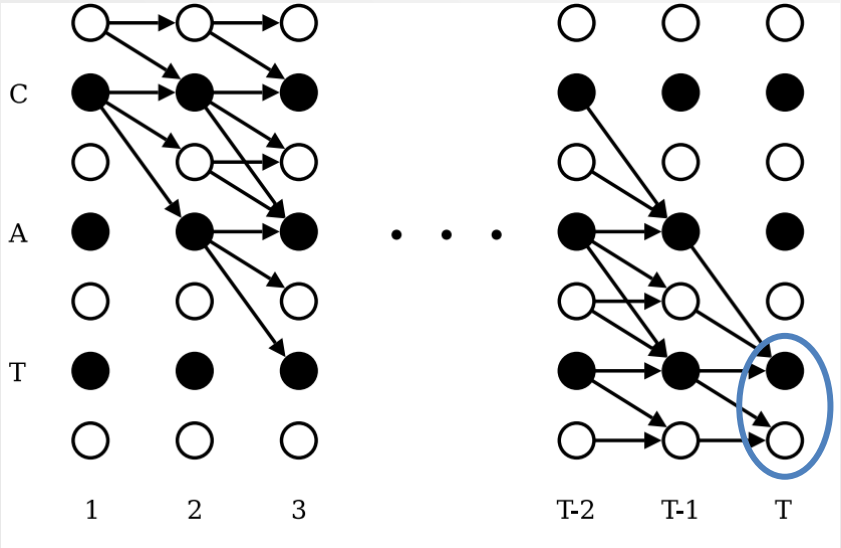
- What happens when a has natural repetitions?
  - E.g. hesitation <I- I am...> = [ey ey ae m...]
  - $\mathcal{B}(a)$ gives us <I am> = [ey ae m]!
- **Connectionist Temporal Classification** (CTC) fixes this by introducing a **blank token** ε to the vocabulary
- $\mathcal{B}$ removes ε **after** removing duplicates
  - $\mathcal{B}([\text{ey ey } \varepsilon\ \varepsilon \text{ ey } \varepsilon \text{ ae } \varepsilon\ \varepsilon\ \varepsilon \text{ m}]) = [\text{ey ey ae m}]$



From Graves *et al.* (2006) "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks"

CTC and monotonic_forward (sketch)

- monotonic_forward cannot handle blanks **as-is**
  - There are optional ε between each reference token
- Three step solution:
  1. Add blanks between reference tokens
     - $a = \langle C\ A\ T \rangle \mapsto a' = \langle \varepsilon\ C\ \varepsilon\ A\ \varepsilon\ T\ \varepsilon \rangle$
  2. Add extra diagonal dependency to skip blanks
  3. $\mathcal{L} = -\log\left(table[U-1,T] + table[U,T]\right)$



From Graves *et al*. (2006) "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks"

Aside: related topics and extensions

- The **Recurrent Neural Network Transducer** (RNN-T) extends CTC by allowing dependencies between b_t and $a_{1,\dots,u-1}$
  - $table[u,t] = P(b_t = a_u | a_{1,\dots,u-1}, x)(table[u-1,t-1] + table[u,t-1])$
  - Compute p_u with auto-regressive RNN, then combine with p_t
- Partial derivatives w.r.t. the loss $\mathcal{L}$ can be efficiently calculated with the forward_backward algorithm
  - monotonic_backward builds suffixes instead of prefixes
  - Multiplies $\frac{\delta P_{p_t}(b_t = a_u)}{\delta p_t}$ with prefix and suffix likelihoods
- Setting $b = x$ and a as a state sequence, forward_backward can be used to train a **GMM-HMM**
  - $P_u(b_t)\left(a_{u-1,u} table[u-1,t-1] + a_{u,u} table[u,t-1]\right)$
- More details can be found in the appendices

Everything past here is an ASIDE

(not on your exam)

Backpropagating the loss 1

- To learn $\mathcal{L} = -\log P_\theta(a)$ end-to-end, we need to **backprop**
- The parameters for frame t are p_t, and

$$\frac{\partial \mathcal{L}}{\partial p_t} = -\frac{1}{P_\theta(a)} \frac{\partial P_\theta(a)}{\partial p_t} = -\frac{1}{P_\theta(a)} \sum_{b \in \mathcal{B}^{-1}(a;T)} \frac{\partial P_\theta(b|x)}{\partial p_t}$$

- For a single alignment b we have

$$\frac{\partial P_\theta(b|x)}{\partial p_t} = \frac{\partial \prod_{t'=1}^{T} P_{p_{t'}}(b_{t'})}{\partial p_t}$$

$$= \left(\prod_{t'=1}^{t-1} P_{p_{t'}}(b_{t'})\right)\left(\prod_{t'=t+1}^{T} P_{p_{t'}}(b_{t'})\right) \frac{\partial P_{p_t}(b_t)}{\partial p_t}$$

$$= P_\theta(b_{1,\ldots,t-1}|x) P_\theta(b_{t+1,\ldots,T}|x) \frac{\partial P_{p_t}(b_t)}{\partial p_t}$$

Backpropagating the loss 2

- Considering **all** paths $\mathcal{B}(b) = a$, we have

$$\frac{\partial \mathcal{L}}{\partial p_t} = -\frac{1}{P_\theta(a)} \sum_{b \in \mathcal{B}^{-1}(a;T)} \frac{\partial P_\theta(b|x)}{\partial p_t}$$

$$= -\frac{1}{P_\theta(a)} \sum_{b \in \mathcal{B}^{-1}(a;T)} P_\theta\left(b_{1,\dots,t-1}|x\right) P_\theta\left(b_{t+1,\dots,T}|x\right) \frac{\partial P_{p_t}(b_t)}{\partial p_t}$$

$$= -\frac{1}{P_\theta(a)} \sum_{u=1}^{U} \frac{\partial P_{p_t}(b_t = a_u)}{\partial p_t} \left(\sum_{b_{1,\dots,t-1} \in \mathcal{B}^{-1}(a_{1,\dots,u-1};t-1)} P_\theta\left(b_{1,\dots,t-1}|x\right) \right) \quad \text{①}$$

$$\left(\sum_{b_{t+1,\dots,T} \in \mathcal{B}^{-1}(a_{u+1,\dots,U};T-t-1)} P_\theta\left(b_{t+1,\dots,T}|x\right) \right) \quad \text{②}$$

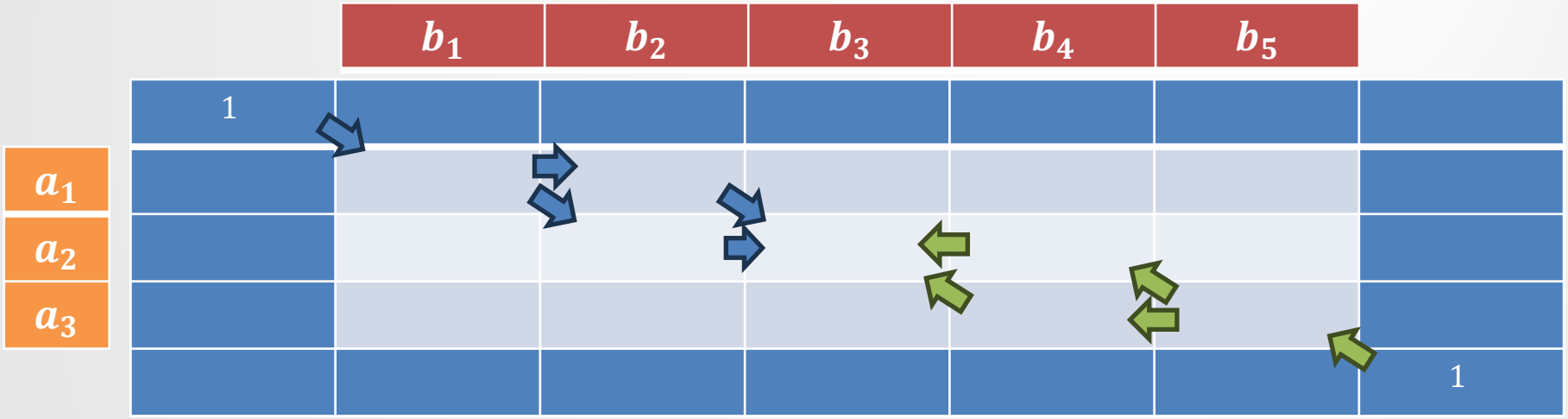
- ① is $table[u-1, t-1]$ in monotonic_forward
- ② is $table[u+1, t+1]$ in monotonic_backward

The monotonic_backward algorithm

Function monotonic_backward
Inputs $a = a_1, a_2, \ldots, a_U$ **and** $b = b_1, b_2, \ldots, b_T$
1: **Define** $table[1 \ldots U+1, 1 \ldots T+1]$
2: initialize($table[1 \ldots U+1, T+1], table[U+1, 1 \ldots T+1]$)
3: **For each** u **in** $U \ldots 1$:
4: **For each** t **in** $T \ldots 1$:
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($table[1 \ldots U+1, 1 \ldots T+1]$)

Same as monotonic_forward, but with partial solutions over **suffixes**

The forward_backward algorithm



- Prefixes are computed with the monotonic_forward algorithm (⇨)
- Suffixes are computed with the monotonic_backward algorithm (⇦)
- $\frac{\partial}{\partial p_t}$ are summed over columns (a_u)

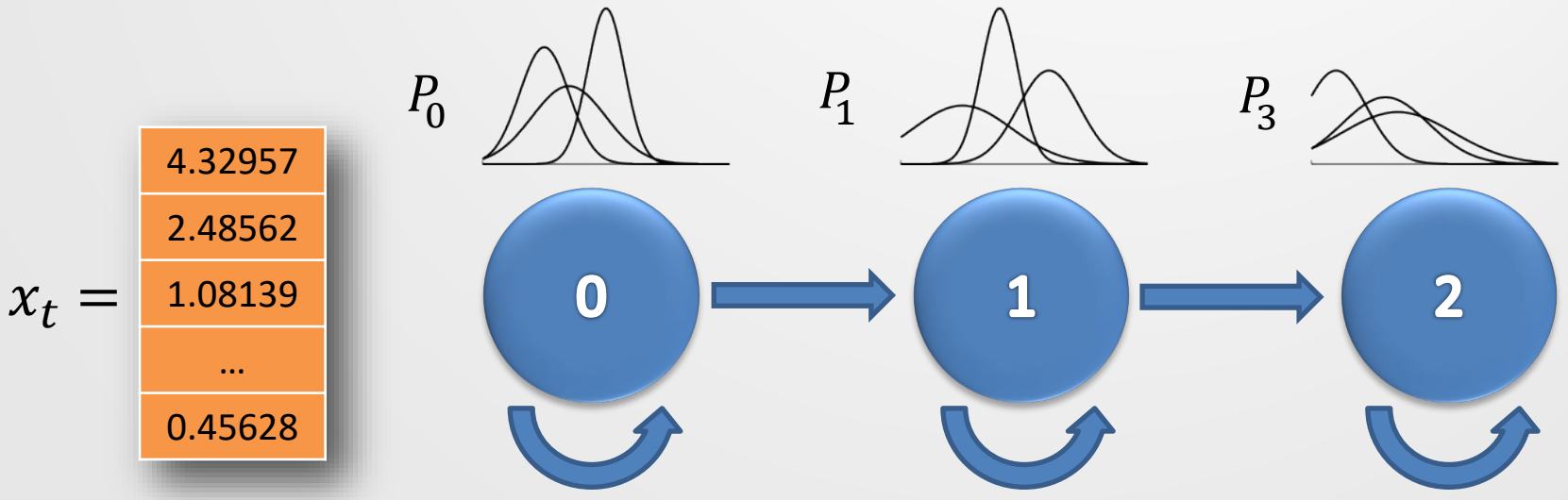
HMM-based ASR

- In GMM-based phone classification, $P(x_t|q_t)$ depends on which phone $q_t = s$ we assume x_t was drawn from
- We can model temporal dependencies across frames by specifying $P(q)$
  - $q^* = \text{argmax}_q P(q)P(x|q)$ is a frame-level transcript
- Using a **Hidden Markov Model** (HMM) over GMM-based observation likelihoods, we have a GMM-HMM ASR system

$$P(q,x) = \underbrace{P(q_o)}_{\text{Prior state probability}} \prod_{t=1}^{T} \underbrace{P(q_t|q_{t-1})}_{\text{Transition probability}} \underbrace{P(x_t|q_t)}_{\text{Observation probability (GMM)}}$$

Continuous HMMs (CHMM)

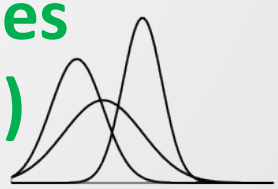
- A **continuous HMM** has observations that are distributed over continuous variables.
  - Observation probabilities, P_s, are also continuous.
  - E.g., here $b_s(x_t)$ tells us the probability of seeing the (multivariate) continuous observation x_t while in state s.



GMM-HMM as Continuous HMM

- Continuous HMMs are very similar to discrete HMMs.
  - $S = \{s_1, \ldots, s_V\}$: set of states (e.g., phonemes)
  - $X = \mathbb{R}^d$: **continuous observation space**

θ
- $\Pi = \{\pi_1, \ldots, \pi_V\}$: initial state probabilities
- $A = \{a_{ij}\}, i, j \in S$: state transition probabilities
- $B = P_v(\vec{x}), i \in S, \vec{x} \in X$: **state output probabilities (i.e., Gaussian mixtures)**



yielding

- $Q = \{q_1, \ldots, q_T\}, q_t \in S$: state sequence
- $\mathcal{O} = \{o_1, \ldots, o_T\}, o_t \in X$: observation sequence

Adapting monotonic_forward

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$(table[0 \ldots U, 0 \ldots T])$

a is the state sequence, $b = x$ the speech features
initialize: set $table[0,0] = \pi_{q_0}$, $table[0,1 \ldots T] = table[1 \ldots U, 0] = 0$
step: $P_u(b_t)\left(a_{u-1,u} table[u-1,t-1] + a_{u,u} table[u,t-1]\right)$
finalize: $-\log table[U,T]$
(Though training usually involves EM, not backprop)