Classical and Modern formulations of ASR

EECS 183/283a: Natural Language Processing

Speech recognition

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$

End-to-end speech recognition

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$

O W

"I want to go to campus"

How to obtain the posterior $p(W|O)$

- We just replace it with a neural network-based function $f^{\mathrm{nn}}(\cdot)$

$$\underset{W}{\operatorname{argmax}}\, p(W|O) = \underset{W}{\operatorname{argmax}}\, f^{\mathrm{nn}}(W|O)$$

  - Easy and simple, no math (in this level), however
- W is a **sequence**! $W = (w_n \in \mathcal{V} | n = 1, \ldots, N)$
  - Very difficult to deal with it
  - Say $N = 10, \left|\mathcal{V}\right| = 100,$ we have to deal with 100^{10} possible sequences
  - Also, the length N is variable
  - We have to use a **special neural network** (e.g., attention, CTC, and RNN-transducer)

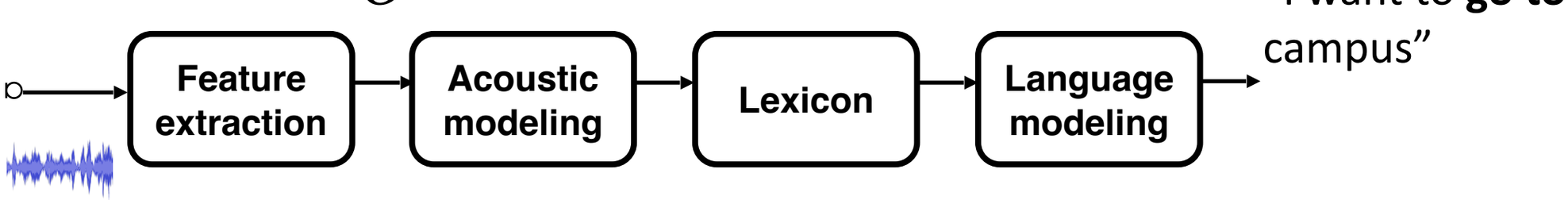
- Classical speech recognition
  - Pipeline

Speech recognition

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$

Classical (non-end-to-end) speech recognition

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$



Speech → Text

Speech o:

ASR

Text W: I want to go to the campus

Speech → Phoneme → Text

Speech O:

ASR

Phoneme L: AY W AA N T T UW G OW T UW K AE M P AH S

ASR

Text W: I want to go to campus

How to obtain the posterior $p(W|O)$

- Factorize the model with **phoneme**
  - Let $L = (l_i \in \{/\mathrm{AA}/, /\mathrm{AE}/, \cdots\} | i = 1, \cdots, J)$ be a phoneme sequence

$$
\begin{aligned}
\arg\max_W p(W|O) &= \arg\max_W \sum_L p(W, L|O) && \text{Sum rule} \\
&= \arg\max_W \sum_L \frac{p(O|W, L)p(L|W)p(W)}{p(O)} && \text{Bayes+ Product rule} \\
&= \arg\max_W \sum_L p(O|W, L)p(L|W)p(W) && \text{Ignore } p(O) \text{ as it does not depend on } W \\
&= \arg\max_W \sum_L p(O|L)p(L|W)p(W) && \text{Conditional independence assumption}
\end{aligned}
$$

Note: the right hand side is not the probability as it lacks a sum to one constraint

Noisy channel model

$$\underset{\mathrm{W}}{\operatorname{argmax}} p(W \mid O) = \underset{W}{\operatorname{argmax}} p(O \mid W) p(W) \approx \underset{W}{\operatorname{argmax}} \sum_{L} p(O \mid L) p(L \mid W) p(W)$$

- **Speech recognition**
  - $p(O \mid L)$: Acoustic model (Hidden Markov model)
  - $p(L \mid W)$: Lexicon
  - $p(W)$: Language model (n-gram)

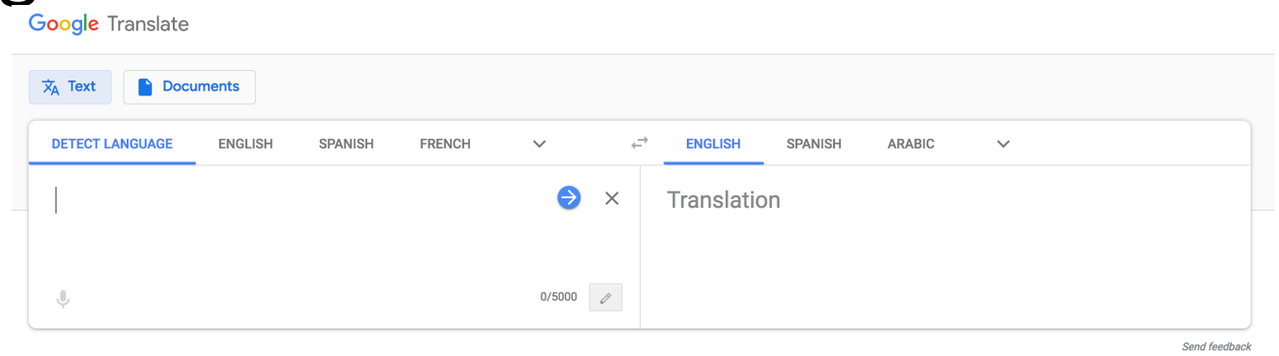
Noisy channel model

$$\underset{W}{\operatorname{argmax}} p(W \mid Y) = \underset{W}{\operatorname{argmax}} p(Y \mid W) p(W)$$

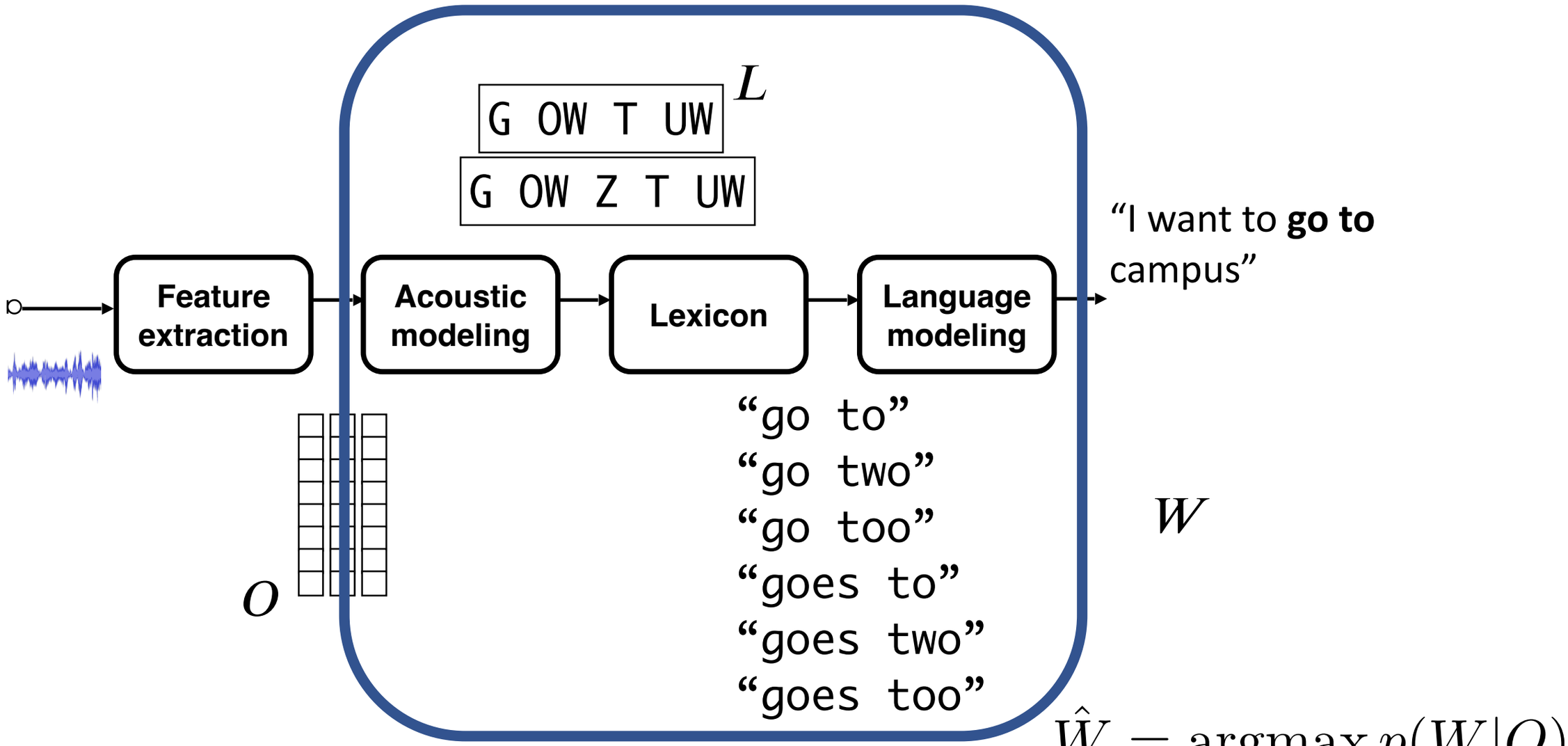
W: Target language text

Y: Source language text

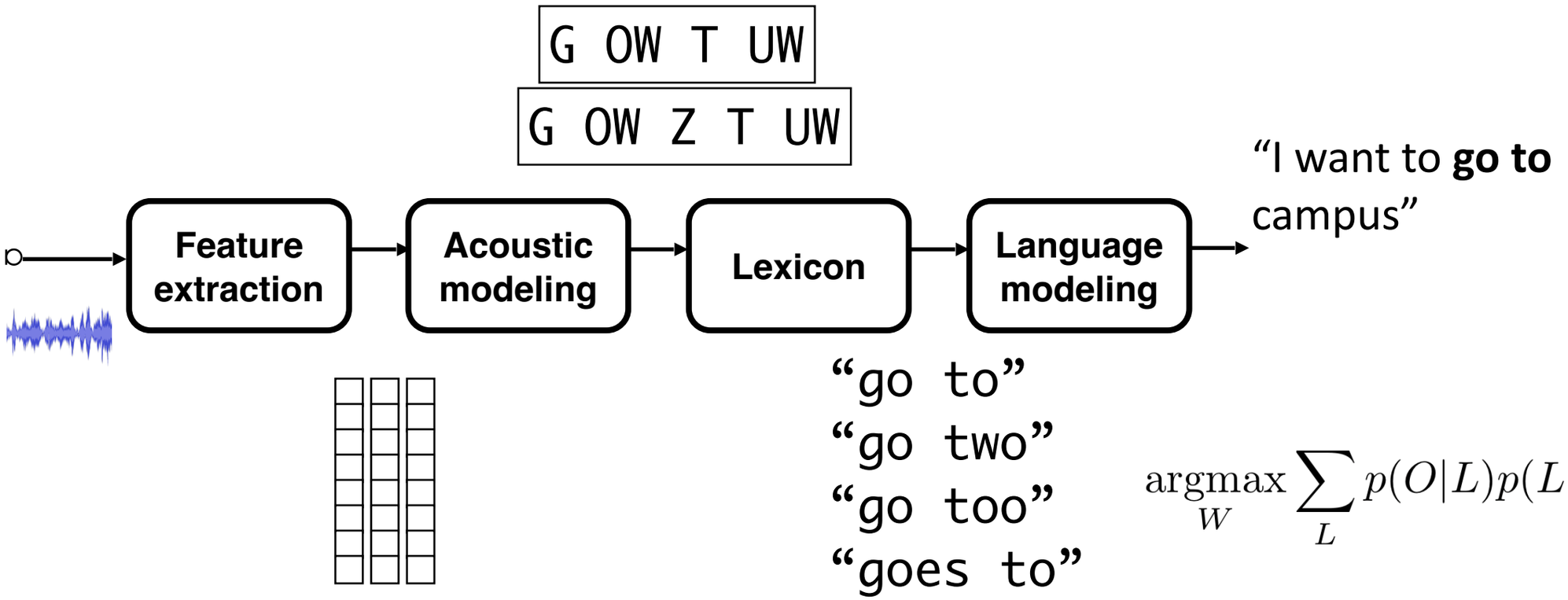
- **Machine translation**
  - $p(Y \mid W)$: Translation model
  - $p(W)$: Language model



Speech recognition pipeline



Speech recognition pipeline



$$\operatorname*{argmax}_{W} \sum_{L} p(O|L)p(L|W)p(W)$$

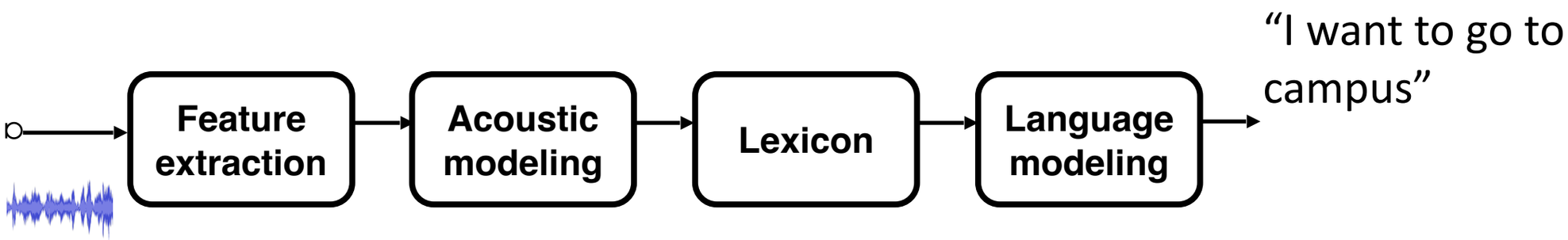
$p(O|L)$ $p(L|W)$ $p(W)$

Please remember the noisy channel model

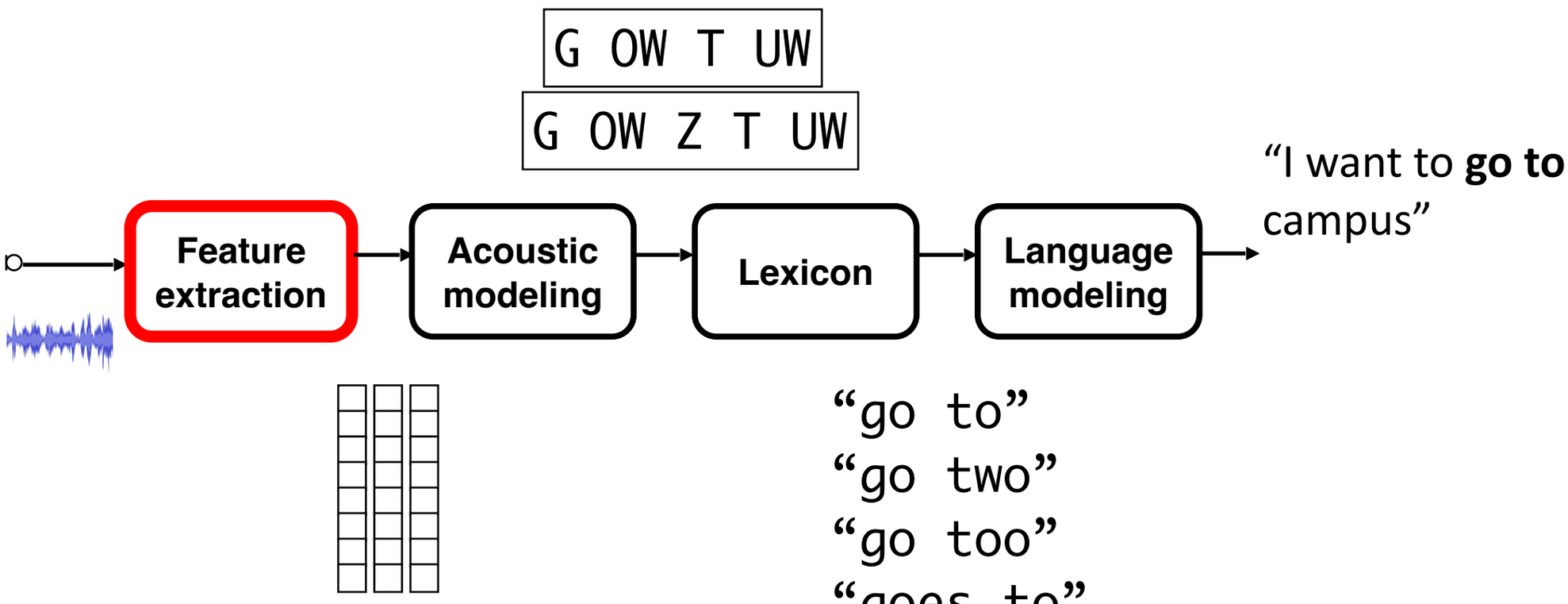
- **Factorization**
- **Conditional independence (Markov) assumptions**

We can elegantly factorize the speech recognition problem with a tractable subproblem

Main blocks of Classical ASR

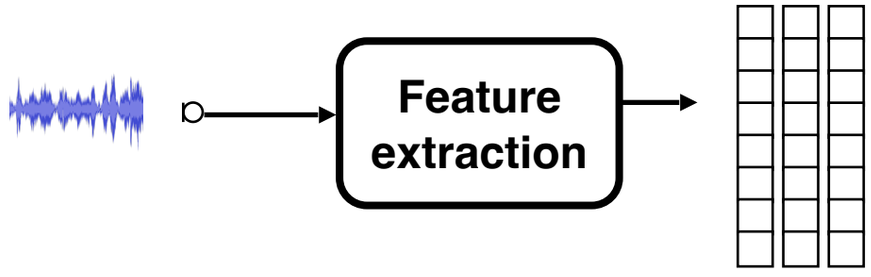


Speech recognition pipeline



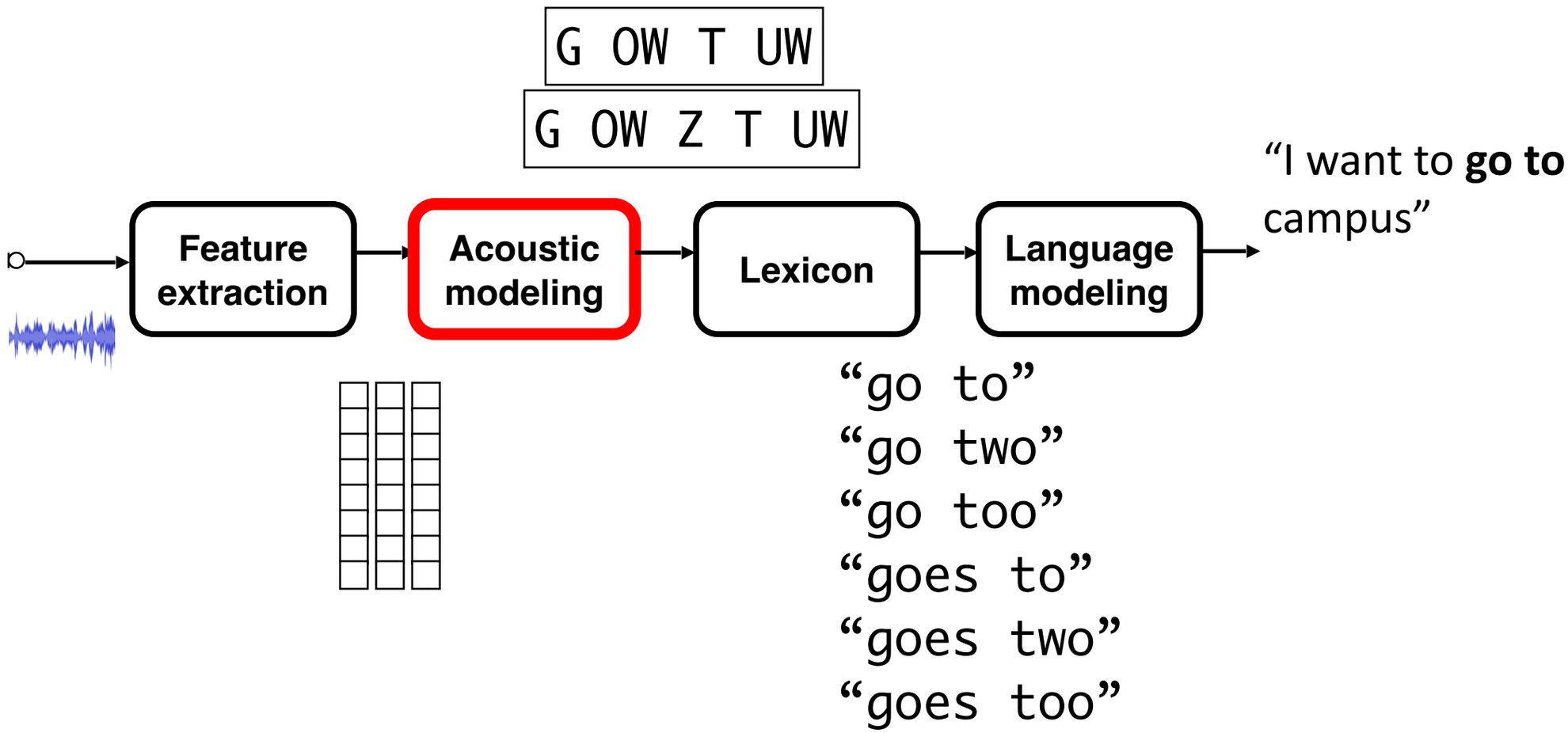
$$p(O|L) \quad p(L|W) \quad p(W)$$

Waveform to speech feature

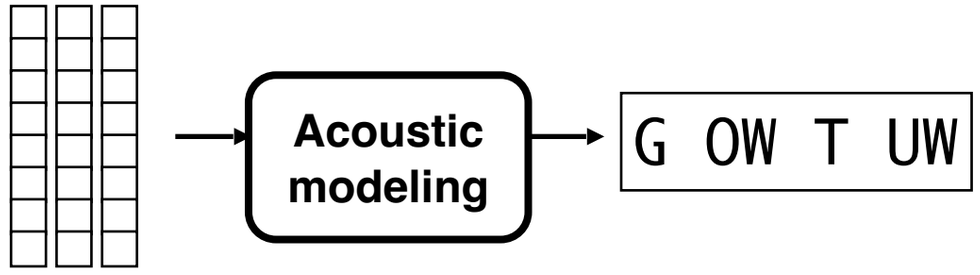


- Performed by so-called **feature extraction** module
  - Mel-frequency cepstral coefficient (MFCC), Perceptual Linear Prediction (PLP) used for **Gaussian mixture model (GMM)**
  - Log Mel filterbank used for **deep neural network (DNN)**
- Time scale
  - 0.0625 milliseconds (16kHz) to 10 milliseconds
- Type of values
  - Scalar (or discrete) to 12—40 dimensional vector

Speech recognition pipeline



Speech feature to phoneme

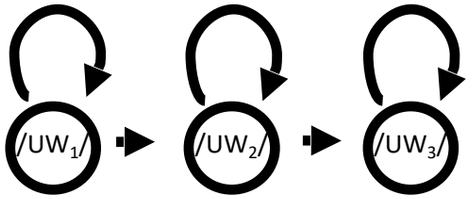


- Performed by so-called **acoustic modeling** module
  - Hidden Markov model (HMM) with **GMM** as an emission probability function
  - Hidden Markov model (HMM) with **DNN** as an emission probability function
- Time scale
  - 10 milliseconds to ~100 milliseconds (depending on a phoneme)
- Type of values
  - 12-dimensional continuous vector to 50 categorical value (~6bit)
- The most critical component to get the ASR performance
- It can be a probability of possible phoneme sequences, e.g.,

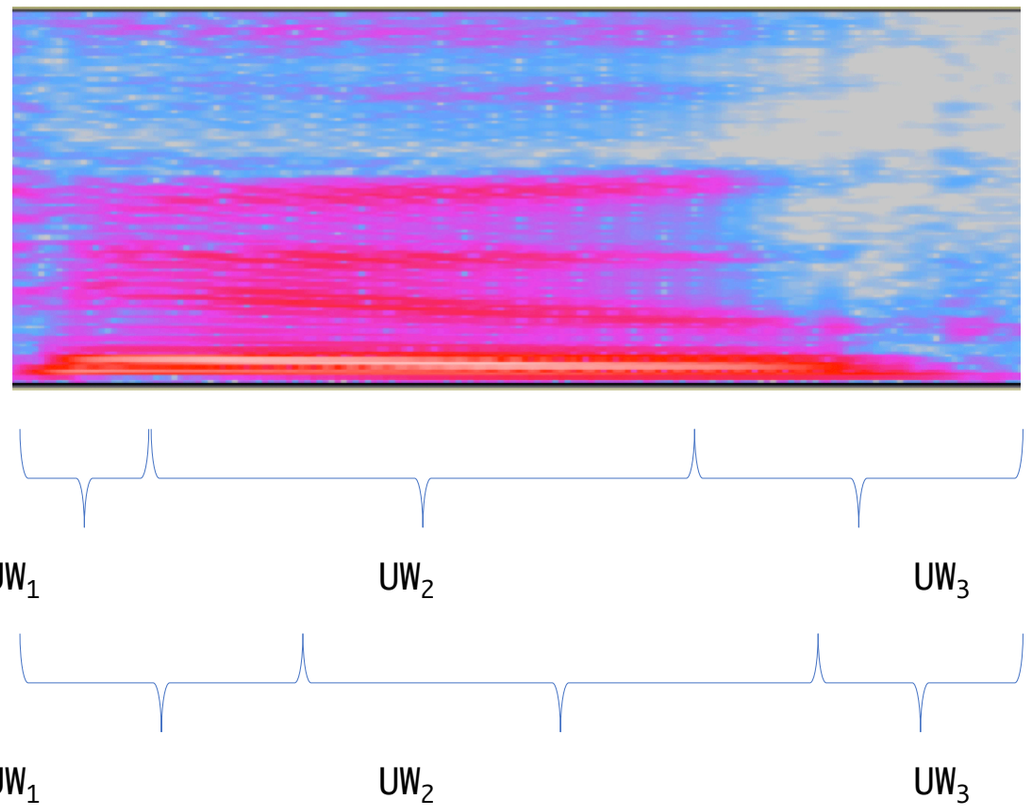
  `G OW T UW` or `G OW Z T UW` with some scores

Acoustic model $p(O \mid L)$

- O and L are **different lengths**
- **Align** speech features and phoneme sequences by using HMM



- Provide $p(O \mid L)$ based on this alignment and model
- The most important problem in speech recognition

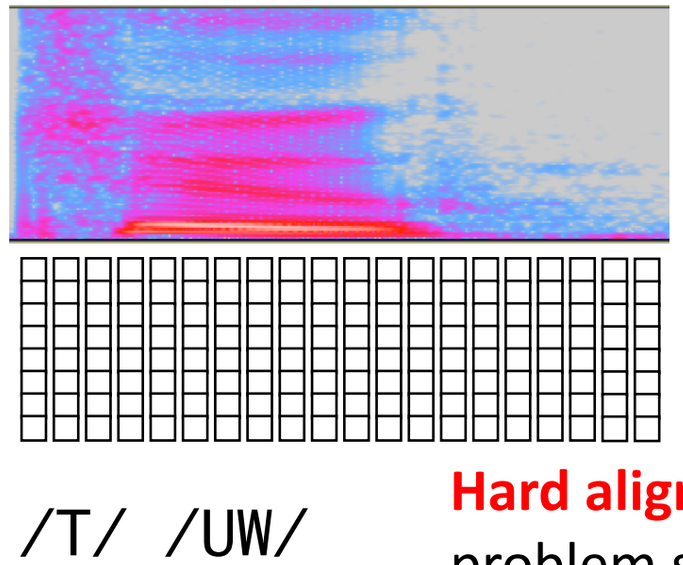


How to formulate an acoustic model

- Acoustic model:
  - Again, O and L are **different lengths**

$$O = (\mathbf{o}_t \in \mathbb{R}^D | t = 1, \ldots, T)$$

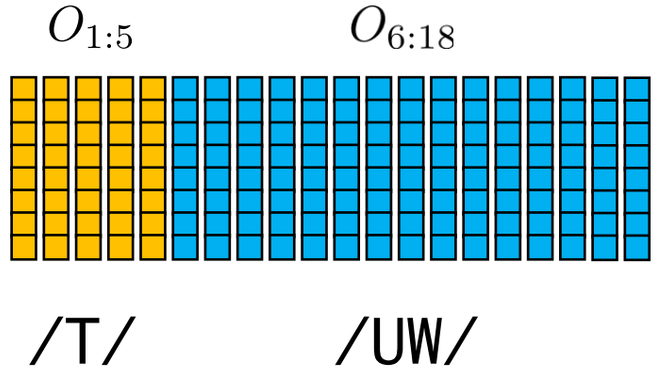
$$L = (l_i \in \{/\text{AA}/, /\text{AE}/, \cdots\} | i = 1, \cdots, J)$$



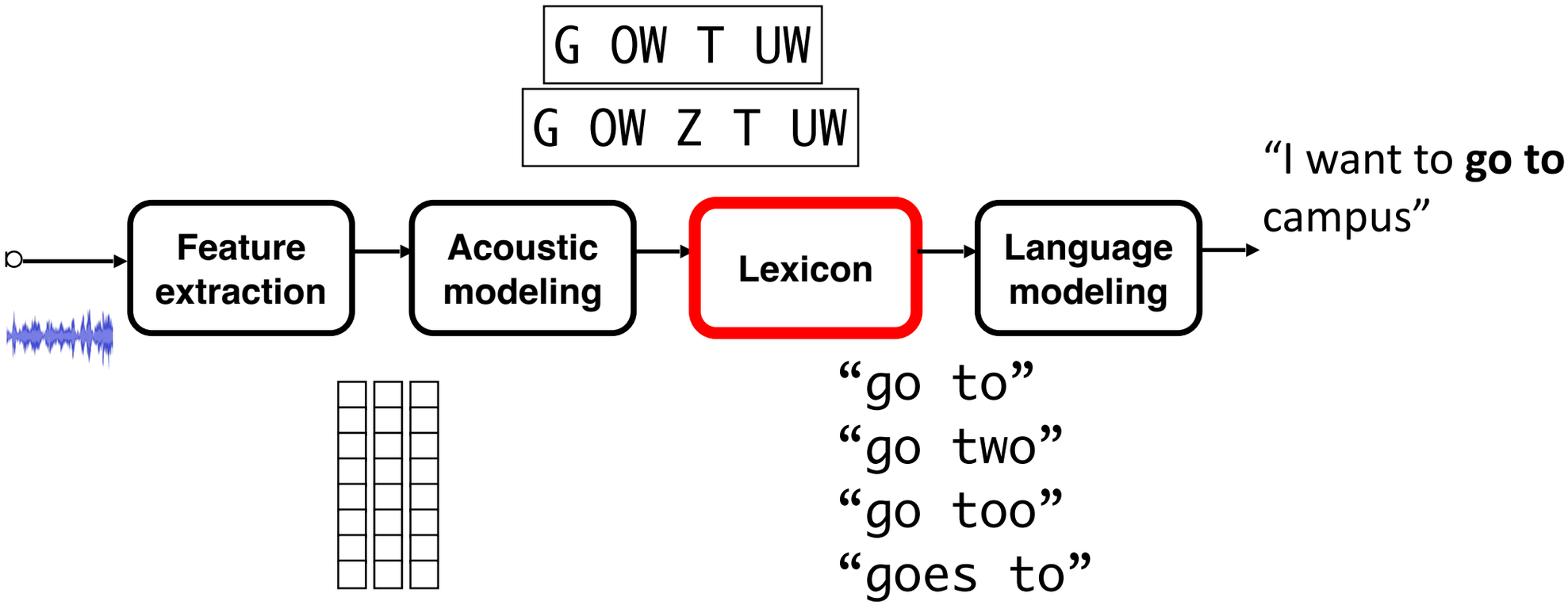
Hard alignment
problem similar to CTC and RNN-T

- **If we assume that the alignment information is given, the problem becomes easy**

$$p(O|L) = p(O_{1:T_1}, O_{T_1+1:T_2}, \cdots | l_1, l_2, \cdots)$$



Speech recognition pipeline



Phoneme to word

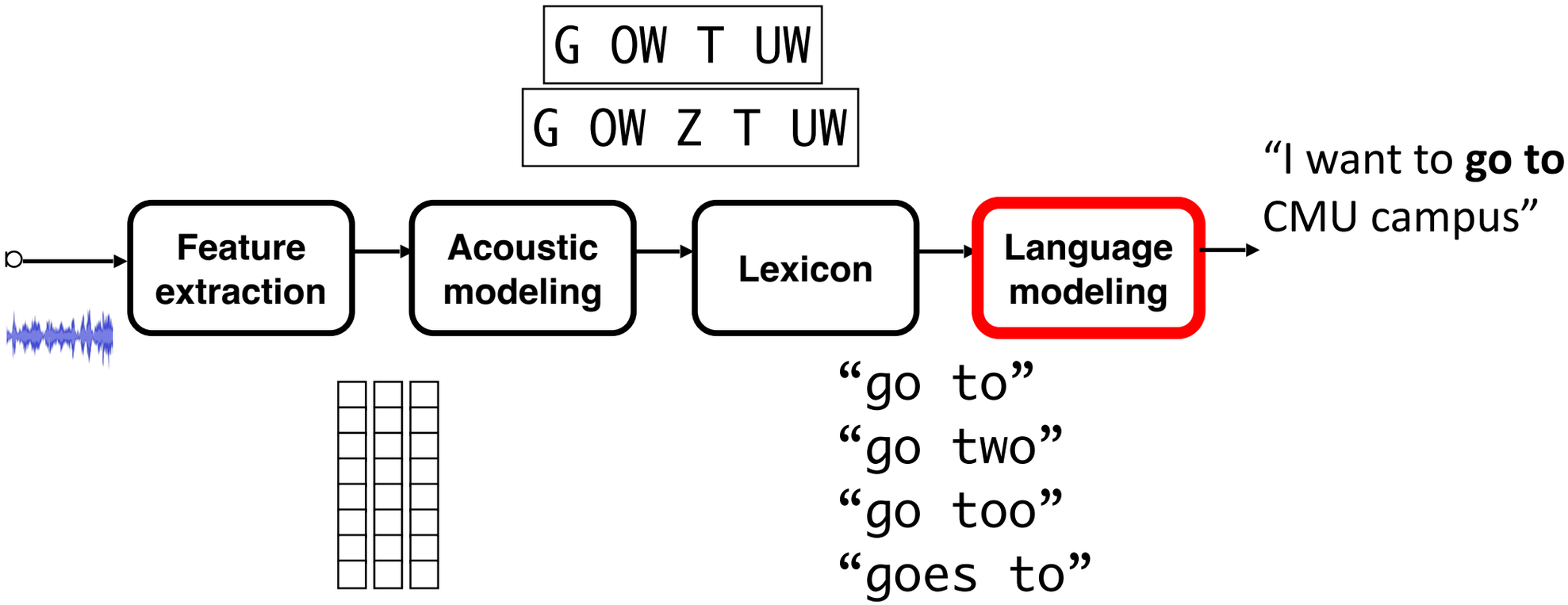
- Performed by **lexicon** module
  - American English: CMU dictionary
- Time scale
  - 100 milliseconds (depending on a phoneme) to 1 second (depending on a word and also language)
- Type of values
  - 50 categorical value (~6bit) to 100K categorical value (~2Byte)
- **We need a pronunciation dictionary**
- It can be **multiple** word sequences (one to many)

Lexicon $p(L \mid W)$

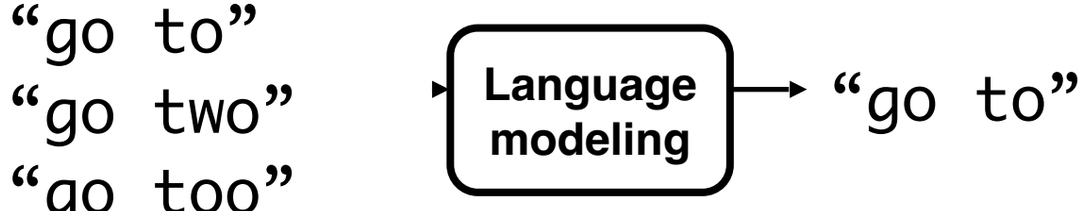
- Basically use a pronunciation dictionary, and map a word to the corresponding phoneme sequence
  - with the probability = 1.0 when single pronunciation
  - with the probability = $1.0/J$ when multiple (J) pronunciations

$$p(L|W) = p(/\mathrm{T}/, /\mathrm{OW}/|\text{"two"}) = 1.0$$

Speech recognition pipeline



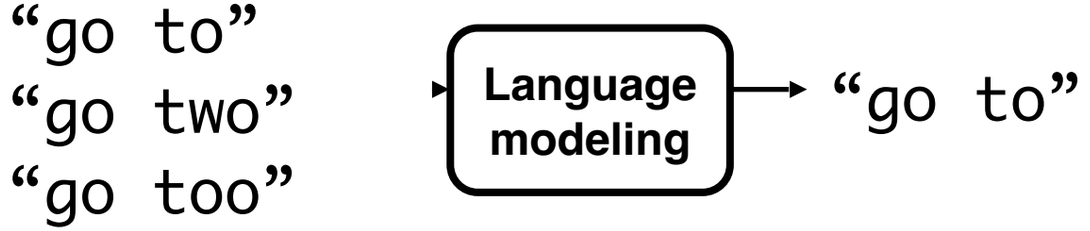
Word to text



- Performed by **language modeling** module $p(W)$
  - N-gram
  - Neural language model (recurrent neural network or transformer)
- From training data, we can basically find how possibly ”to”, “two”, and “too” will be appeared after “go”
  - Part of WSJ training data, 37,416 utterances
    - “go to”: **51** times
    - “go two”:
    - “go too”:

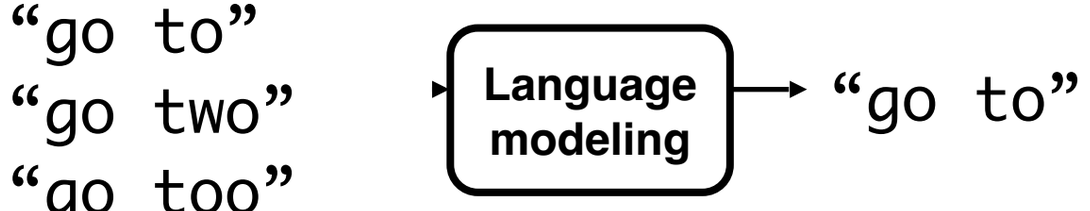
THE WALL STREET JOURNAL.

Word to text



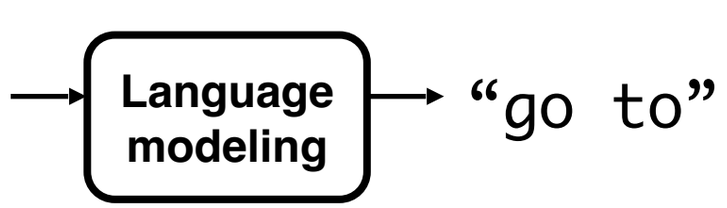
- Performed by **language modeling** module $p(W)$
  - N-gram
  - Neural language model (recurrent neural network or transformer)
- From training data, we can basically find how possibly ”to”, “two”, and “too” will be appeared after “go”
  - Part of WSJ training data, 37,416 utterances
    - “go to”: **51** times
    - “go two”: **0** times
    - “go too”: **0** times

Word to text



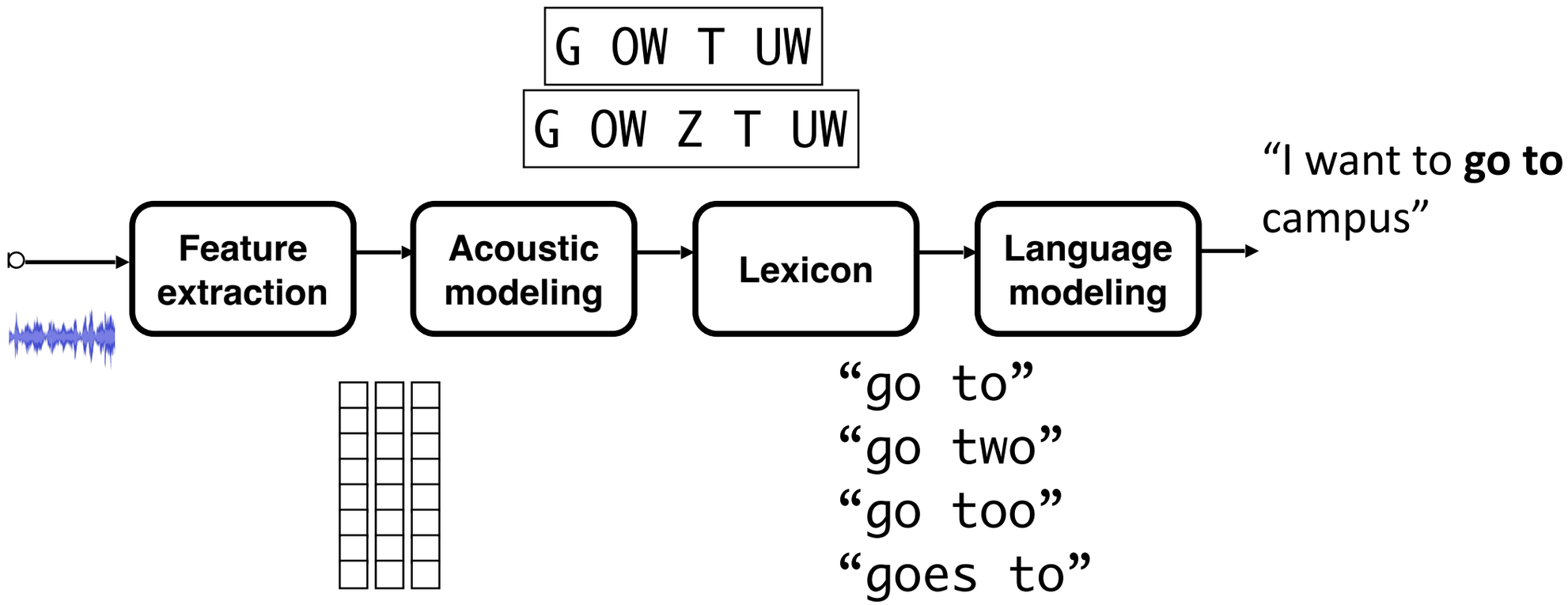
- Performed by **language modeling** module $p(W)$
  - N-gram
  - Neural language model (recurrent neural network or transformer)
- From training data, we can basically find how possibly ”to”, “two”, and “too” will be appeared after “go”
  - WSJ all text data, 6,375,622 sentences
    - “go to”: **2710** times
    - “go two”:
    - “go too”:

Word to text



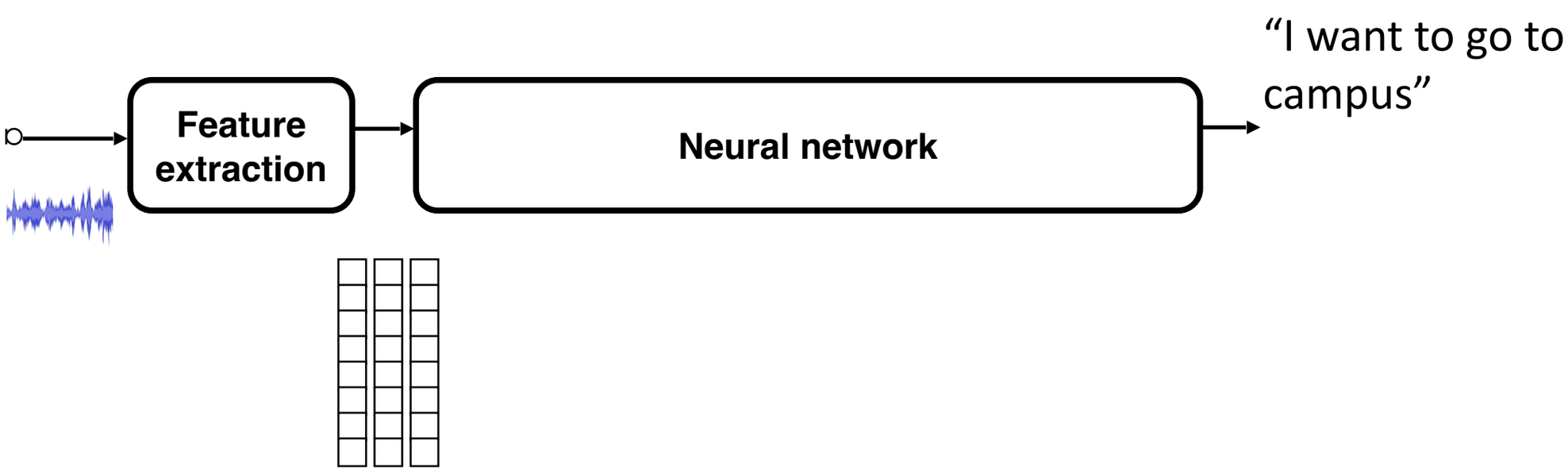
- Performed by **language modeling** mod
  - N-gram
  - Neural language model (recurrent neural network or transformer)
- From training data, we can basically find how possibly ”to”, “two”, and “too” will be appeared after “go”
  - WSJ all text data, 6,375,622 sentences
    - “go to”: **2710** times
    - “go two”: **2** times, e.g., “those serving shore plants often go two hundred miles or more”
    - “go too”:

Building speech recognition was really difficult…



- We need to develop all components
- Each component requires a lot of background knowledge
- We need to tune hyper-parameters in each module

Next: End-to-end speech recognition

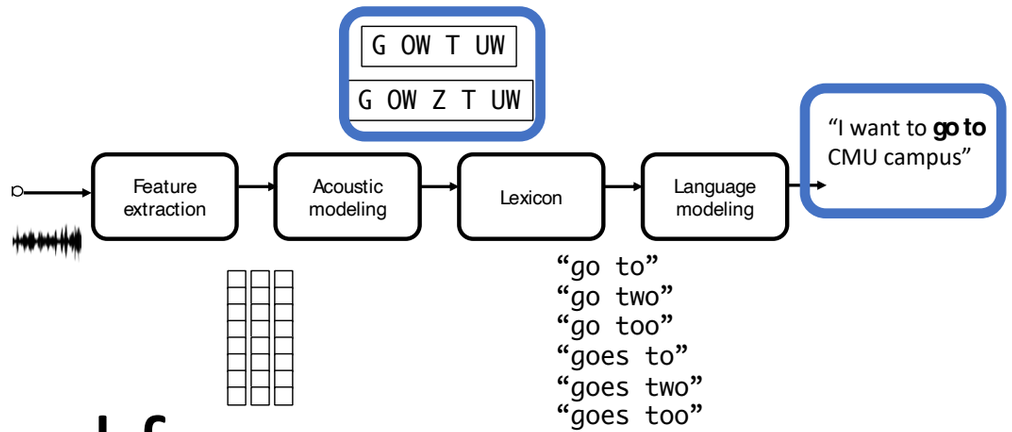


- We can simply the complicated models
- Optimize all components by using back propagation
- We still need some formulations to make a problem tractable

Today’s agenda

- Output
- Alignments

Output unit



- Our final goal: output a text
- Text can be represented by several forms
  - Word, Character, phoneme, etc.
  - We will discuss the characteristics of each output unit

Which unit?
End-to-End ASR case

How to describe the phrase “go to”?

- **Word**
  - “go” and “to”
  - More **semantic/syntactic**
  - Very large vocabulary size, e.g., $|\mathcal{V}|$ would be 100K
  - **Out of vocabulary issue** ☹
  - The length ($N = 2$) is very short. **Less computational cost**, but **larger mismatch between the input and output lengths**
- **Character**
  - “g” “o” “_“ “t” and “o” (“_” means the space)
  - The vocabulary size is not large in general. ~30 in the Roman script, ~10K in the Chinese script
  - **No out of vocabulary issue** (rarely happens) ☺
  - The length ($N = 5$) becomes longer. **More computational cost**, but **relaxing the mismatch between the input and output lengths**
- **BPE: Byte Pair Encoding** (**sentence piece**)
  - “go to” → “▁g” “o” “_to” ($N = 3$)
  - **Something between**, we can also control the vocabulary size

In general, we do not specify which unit we use in our lecture since this is one of the model configurations.

Which unit?
HMM-based (classical) system

How to describe the phrase “go to”?

- **Phoneme**
  - “go to” → “G OW T UW”
  - More **acoustic**
  - The phoneme vocabulary size is not very large in general
  - The length ($N = 5$) becomes long (like the discussion in the (Roman) character).
  - **We need a dictionary**
  - If we use a phone, we can make this part language-independent
- **State** (hidden Markov model state)
  - We further decompose a phoneme into several states
  - Furthermore **acoustic**
  - Classically, we use this representation a lot (e.g., 3-state HMM)
  - It makes the unit further longer and **the mismatch between the input and output lengths is further relaxed**
  - It will be introduced later in HMM

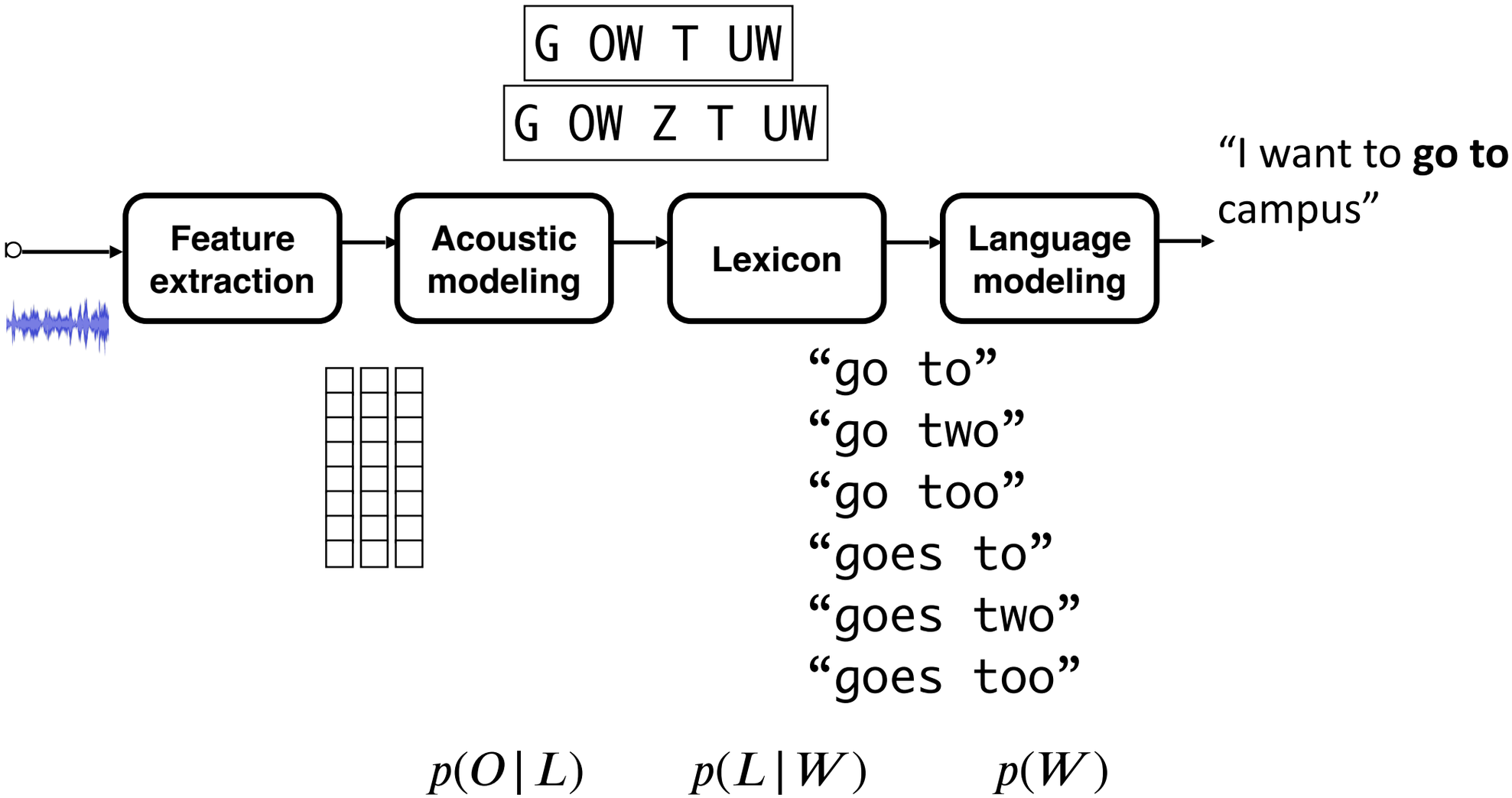
Which unit practically?

- Mostly, we use BPE (or sentence piece)
  - We need to set the maximum number depending on the training data
- Character for the low-resource case or Chinese and Japanese
  - Chinese/Japanese has ~10,000 characters
- Some languages do not have scripts
  - Phoneme, phone
  - Translation to the other languages (not ASR but speech translation)

Today’s agenda

- Output
- Alignments

HMM-based (classical) speech recognition pipeline



Speech recognition

$$\hat{W} = \underset{W \in \mathcal{W}}{\operatorname{argmax}}\, p(W|O)$$

End-to-End ASR

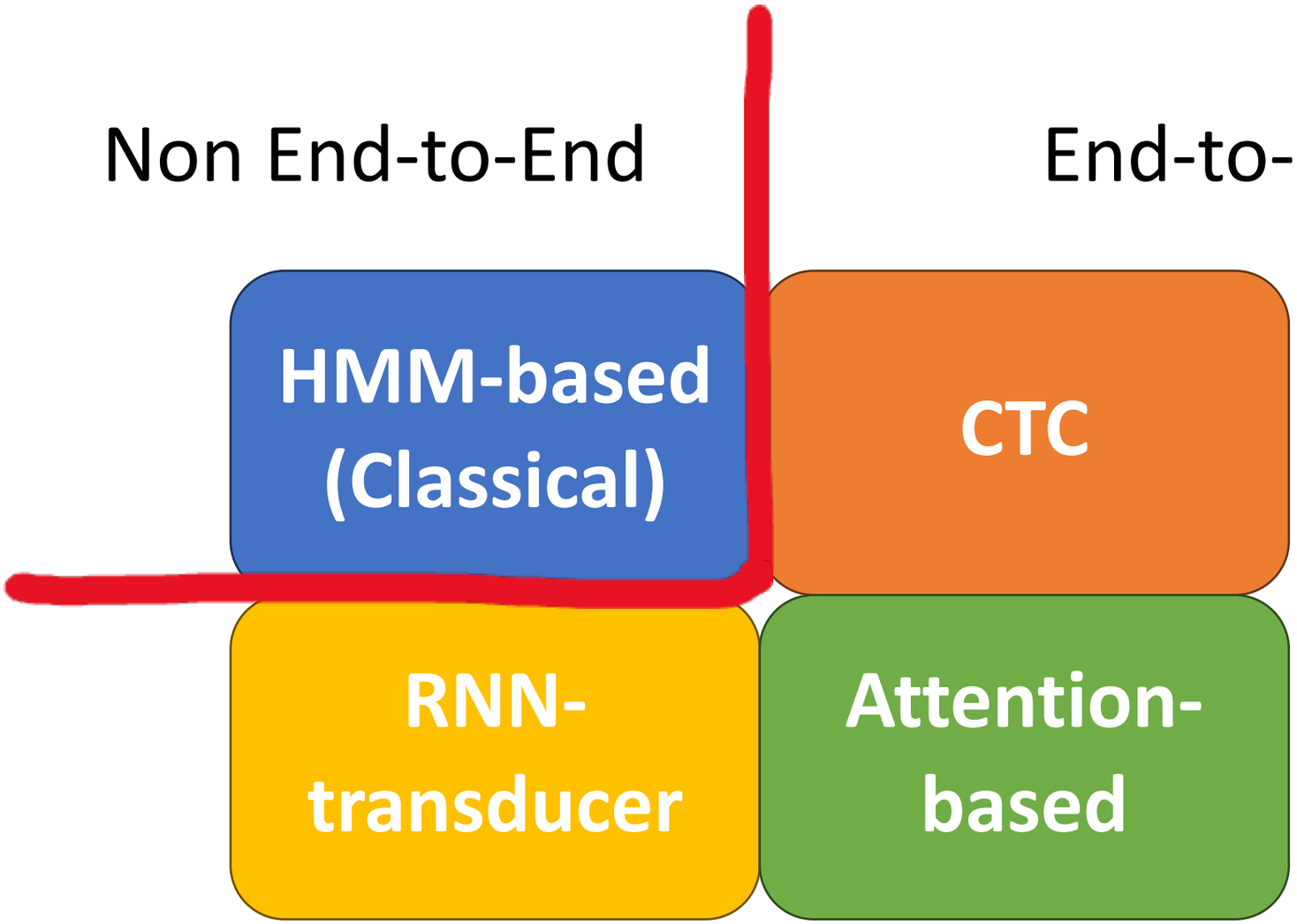
- Direct mapping function from speech feature sequence O to text W
- Usually, it does **not** deal with the phoneme-based intermediate representation
- Mainly three architectures
  - Attention-based
    - Decoder-only is also included here (not fully end-to-end)
  - Connectionist temporal classification (CTC)
  - Recurrent neural network transducer (RNN-T)

HMM-based (Classical)

CTC

RNN-transducer

Attention-based


Non End-to-End
End-to-End
HMM-based (Classical)
CTC
RNN-transducer
Attention-based

Attention-based ASR

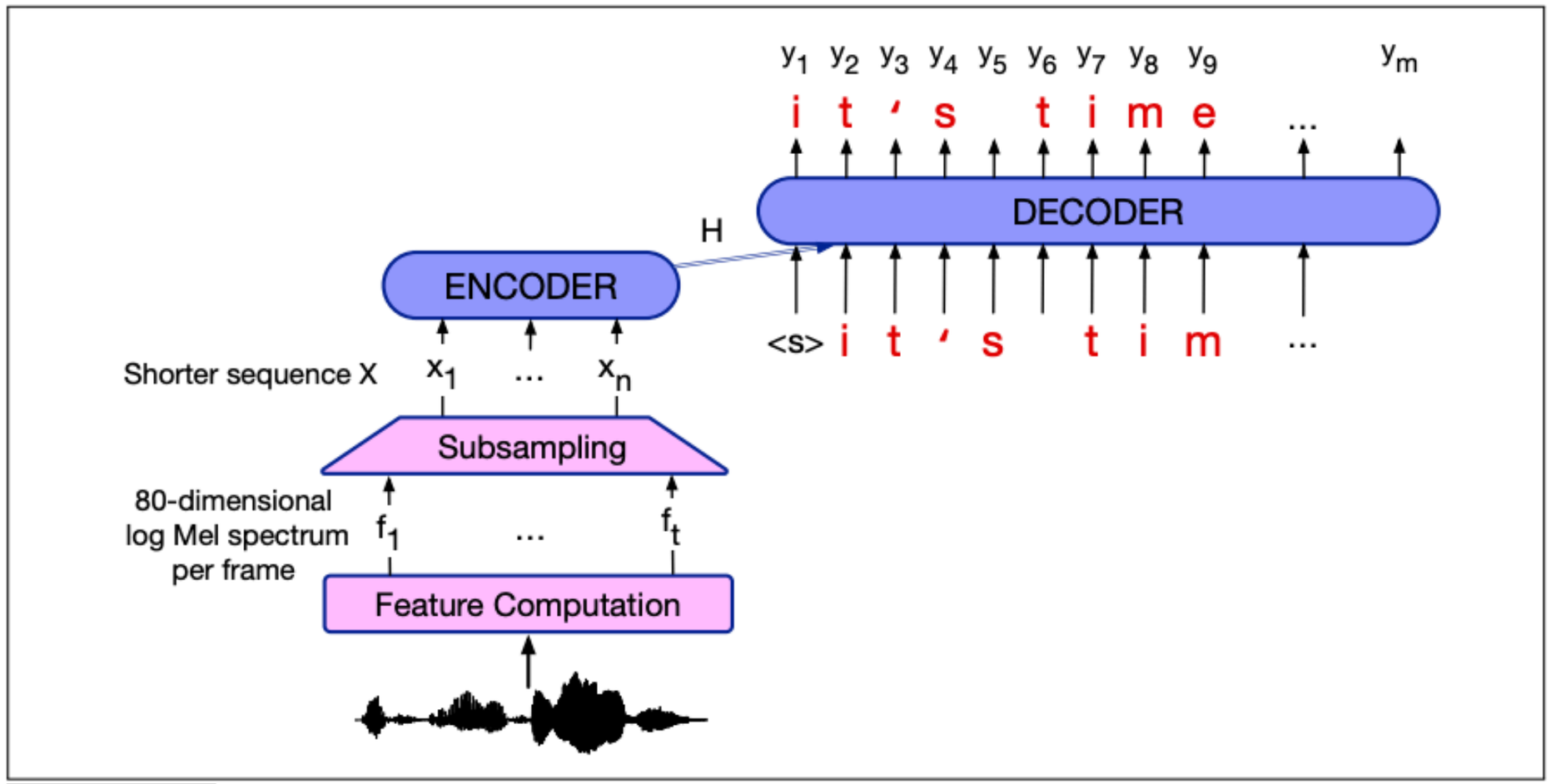


Figure 15.5 Schematic architecture for an encoder-decoder speech recognizer.

Listen Attend & Spell (2016)

Attention-based ASR

- Our staring point $p(W|O)$
  - Input: $O = (\mathbf{o}_t | t = 1,\ldots,T)$
  - It is difficult to deal with $W = \left(w_i \in \mathcal{V} \middle| i = 1,\ldots,N\right)$
  - $T \neq N$
- Instead, we **factorize** $p(W|O)$ as follows based on a **probabilistic chain rule**

$$p(W|O) = \prod_{i=1}^{N} p(w_i|w_{1:i-1}, O)$$

Attention-based ASR

- Our staring point $p(W|O)$
  - Input: $O = (\mathbf{o}_t | t = 1,\ldots,T)$
  - It is difficult to deal with $W = \left(w_i \in \mathscr{V} \middle| i = 1,\ldots,N\right)$
  - $T \neq N$
- Instead, we **factorize** $p(W|O)$ as follows based on a **probabilistic chain rule**

$$p(W|O) = \prod_{i=1}^{N} p(w_i|w_{1:i-1}, O)$$

- This neural network is handled by an **attention-based method** to align the input and output **(soft alignments)**
- We usually do not use the phoneme

Whisper (OpenAI)

$$p(W|O) = \prod_{i=1}^{N} p(w_i|w_{1:i-1}, O)$$

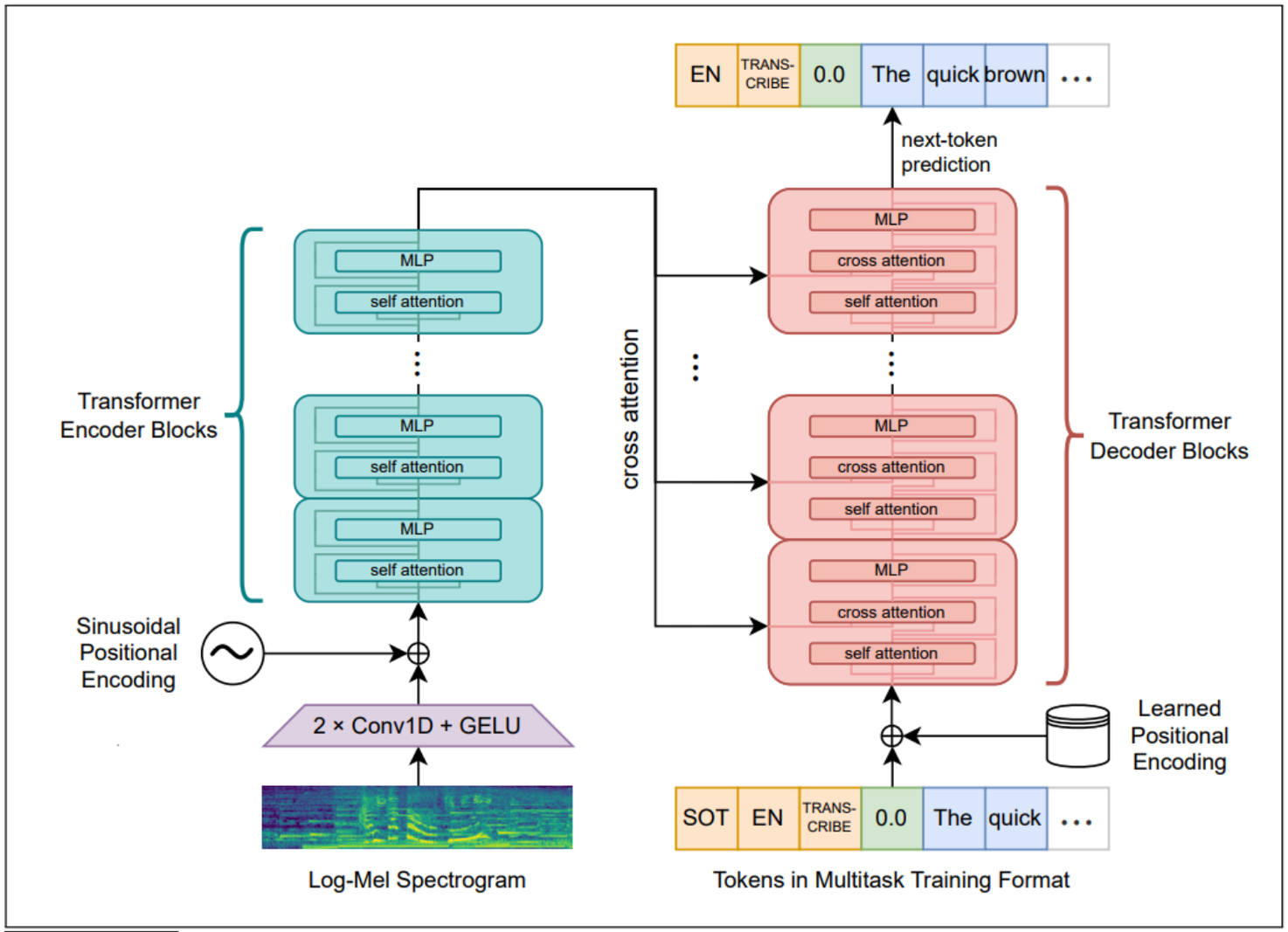
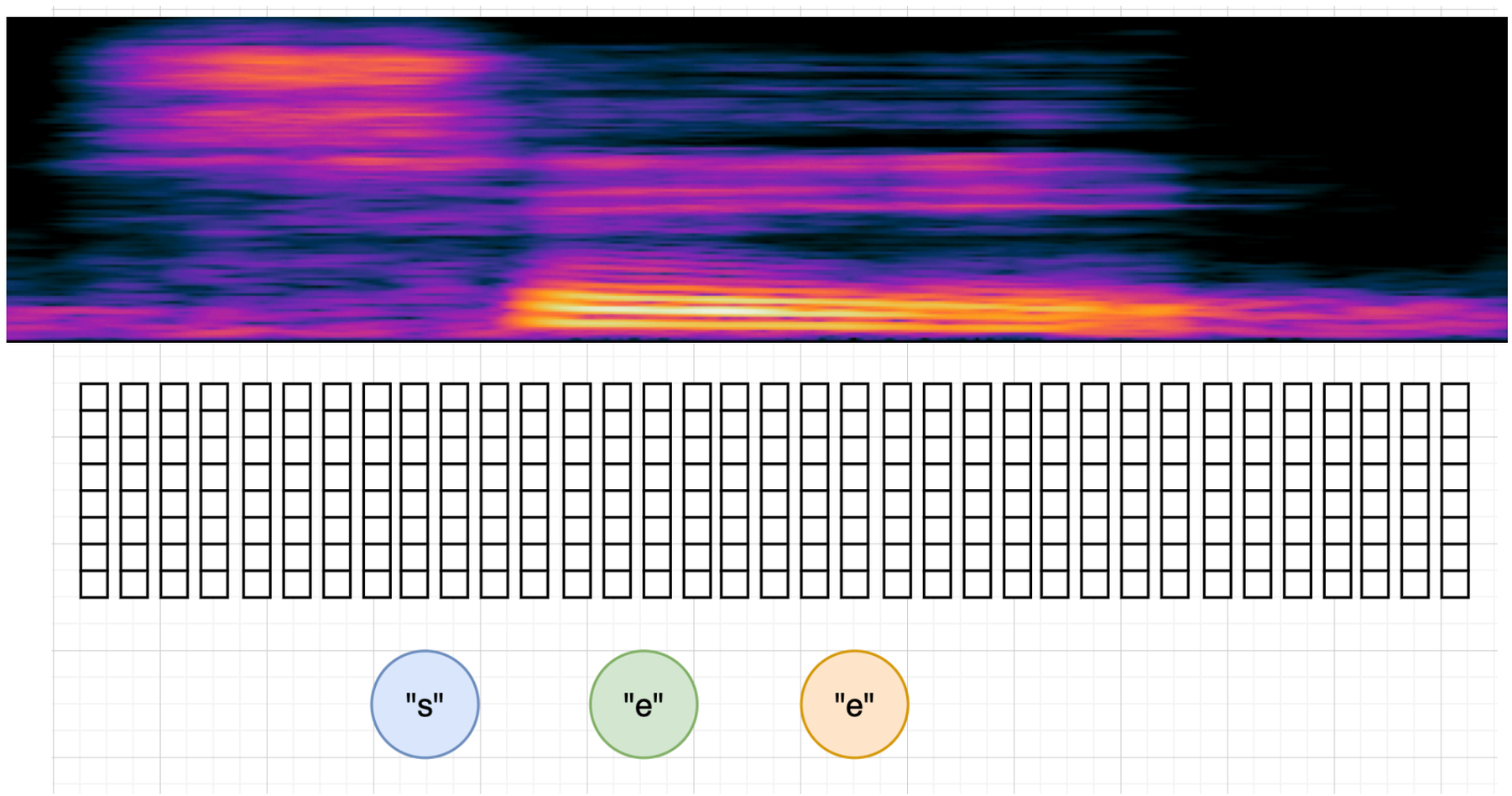


Figure 15.6 A sketch of the Whisper architecture from Radford et al. (2023). Because Whisper is a multitask system that also does translation and diarization (we'll discuss these non-ASR tasks in the following chapter), Whisper's transcription format has a Start of Transcript (SOT) token, a language tag, and then an instruction token for whether to transcribe or translate.

Alignments



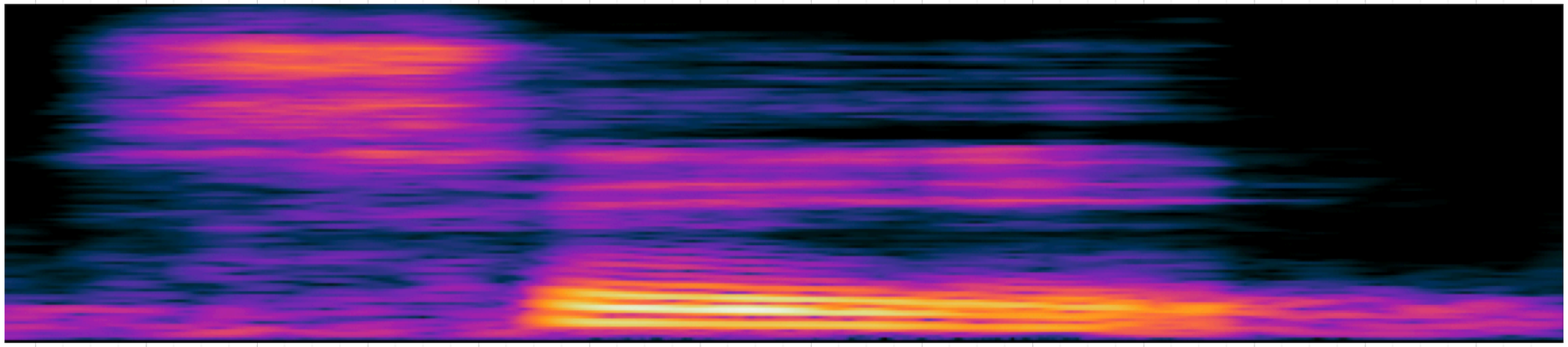
Speech features

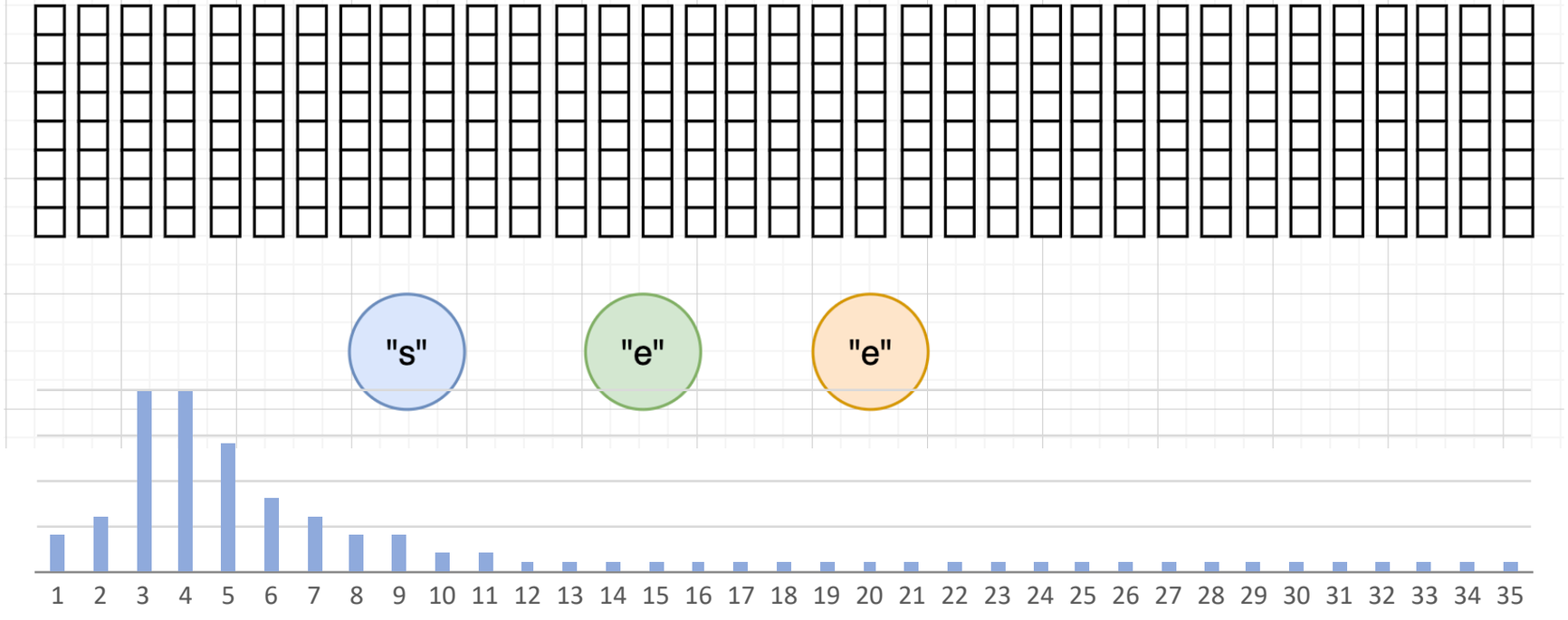
$O\ (|O| = 35)$

Token

$W(|W| = 3)$

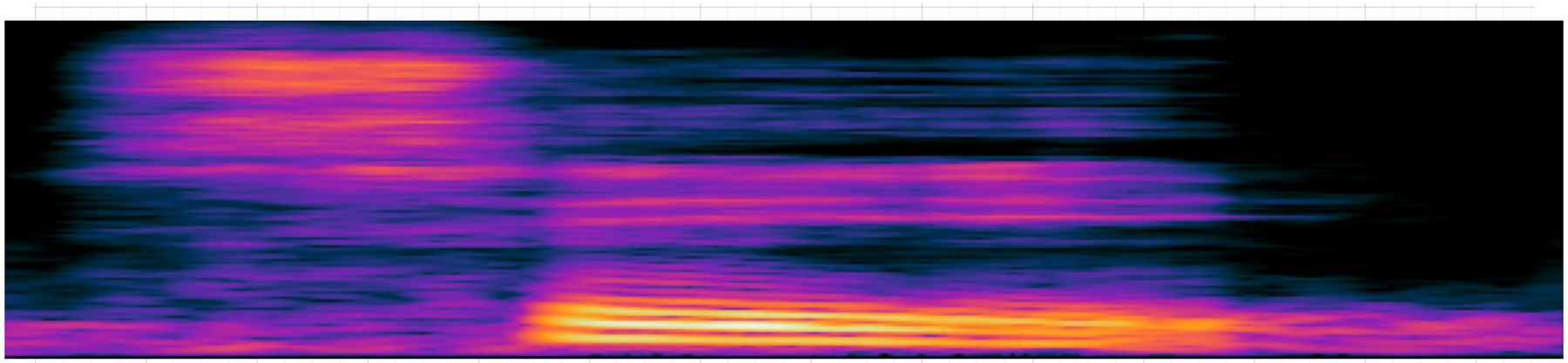
Soft alignments





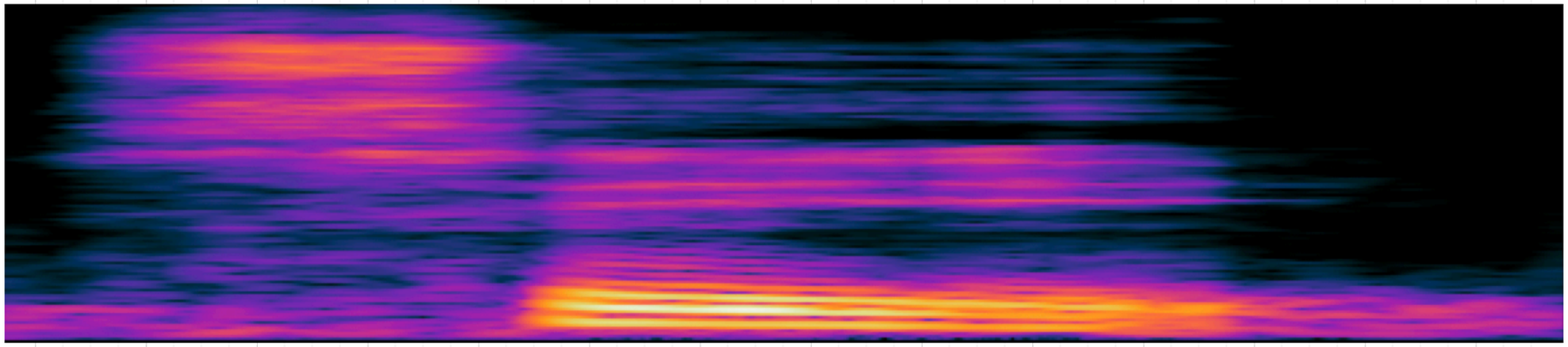
How many %
“s” is aligned?

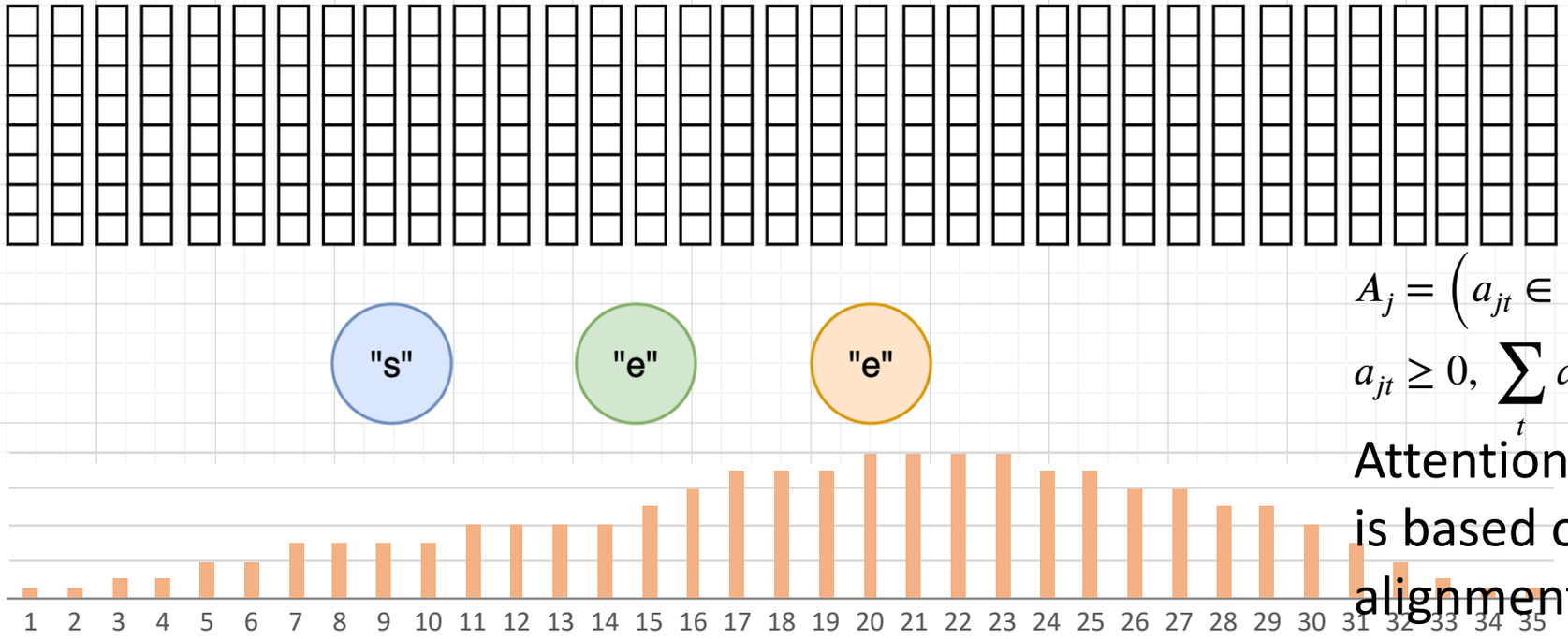
Soft alignments



How many %
“e” is aligned?

Soft alignments





How many % “e” is aligned?

$$A_j = \left(a_{jt} \in [0,1] \,\middle|\, t = 1,\ldots,T \right)$$

$$a_{jt} \geq 0, \; \sum_t a_{jt} = 1$$

Attention-based approach is based on the **soft** alignment

Connectionist Temporal Classification

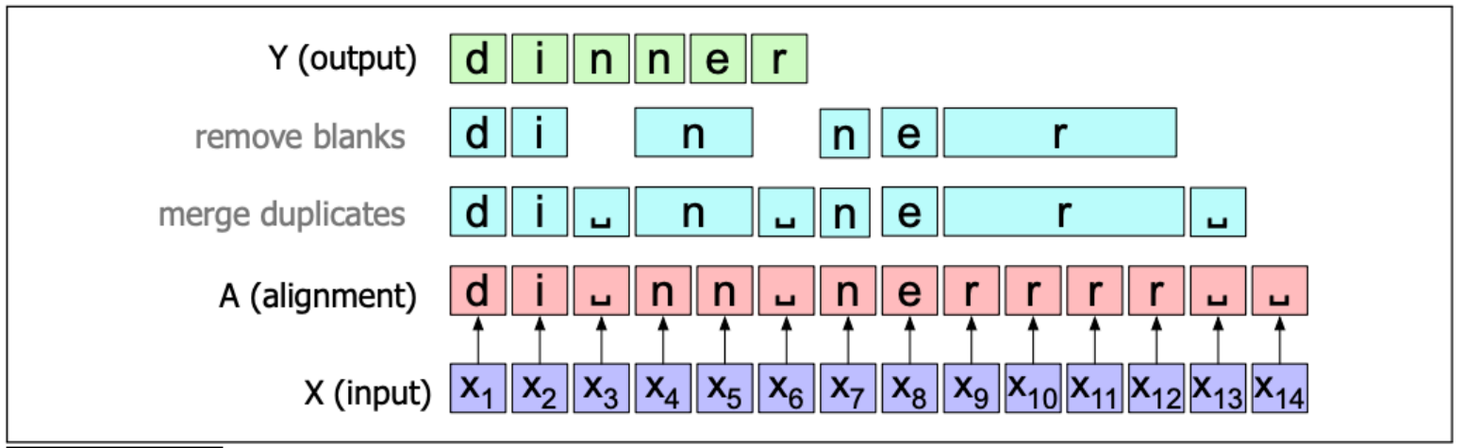


Figure 15.13 The CTC collapsing function B, showing the space blank character ␣; repeated (consecutive) characters in an alignment A are removed to form the output Y.

The CTC collapsing function is many-to-one; lots of different alignments map to the same output string. For example, the alignment shown in Fig. 15.13 is not the only alignment that results in the string *dinner*. Fig. 15.14 shows some other alignments that would produce the same output.

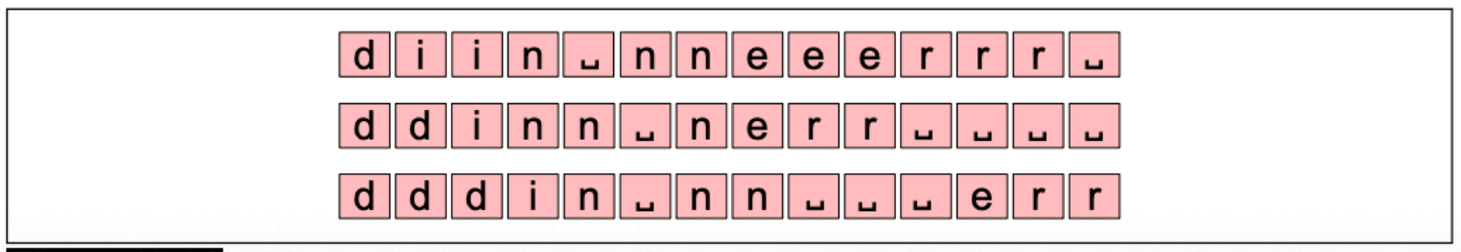


Figure 15.14 Three other legitimate alignments producing the transcript *dinner*.

RNN-Transducer

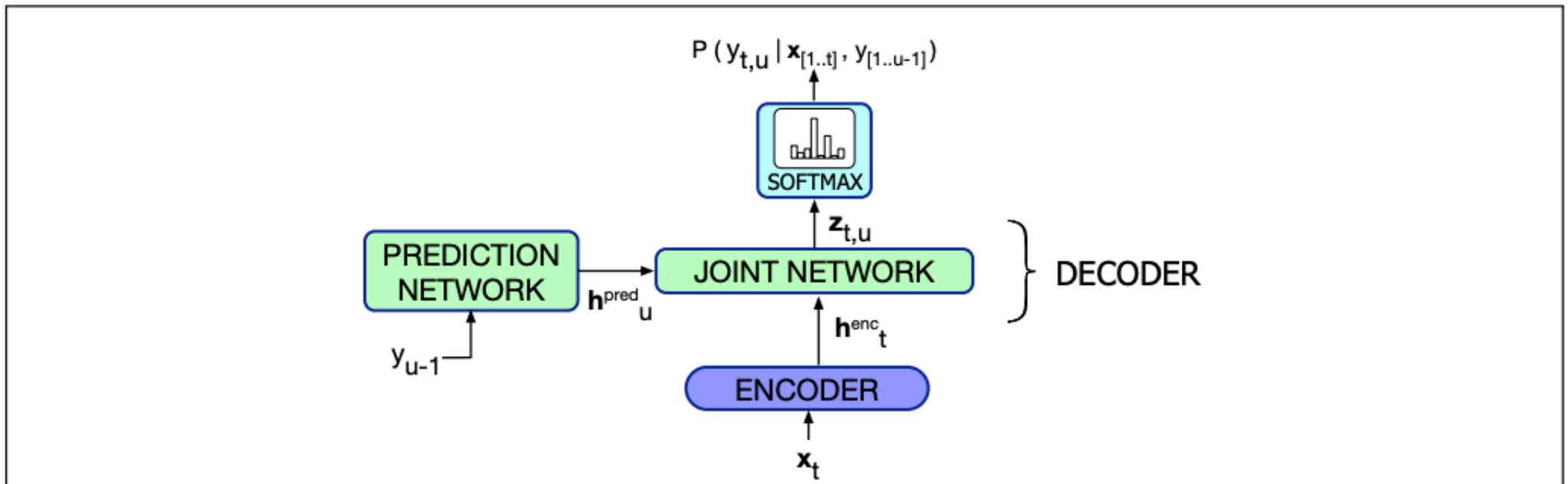


Figure 15.17 The RNN-T model computing the output token distribution at time *t* by integrating the output of a CTC acoustic encoder and a separate 'predictor' language model.

CTC or RNN transducer cases

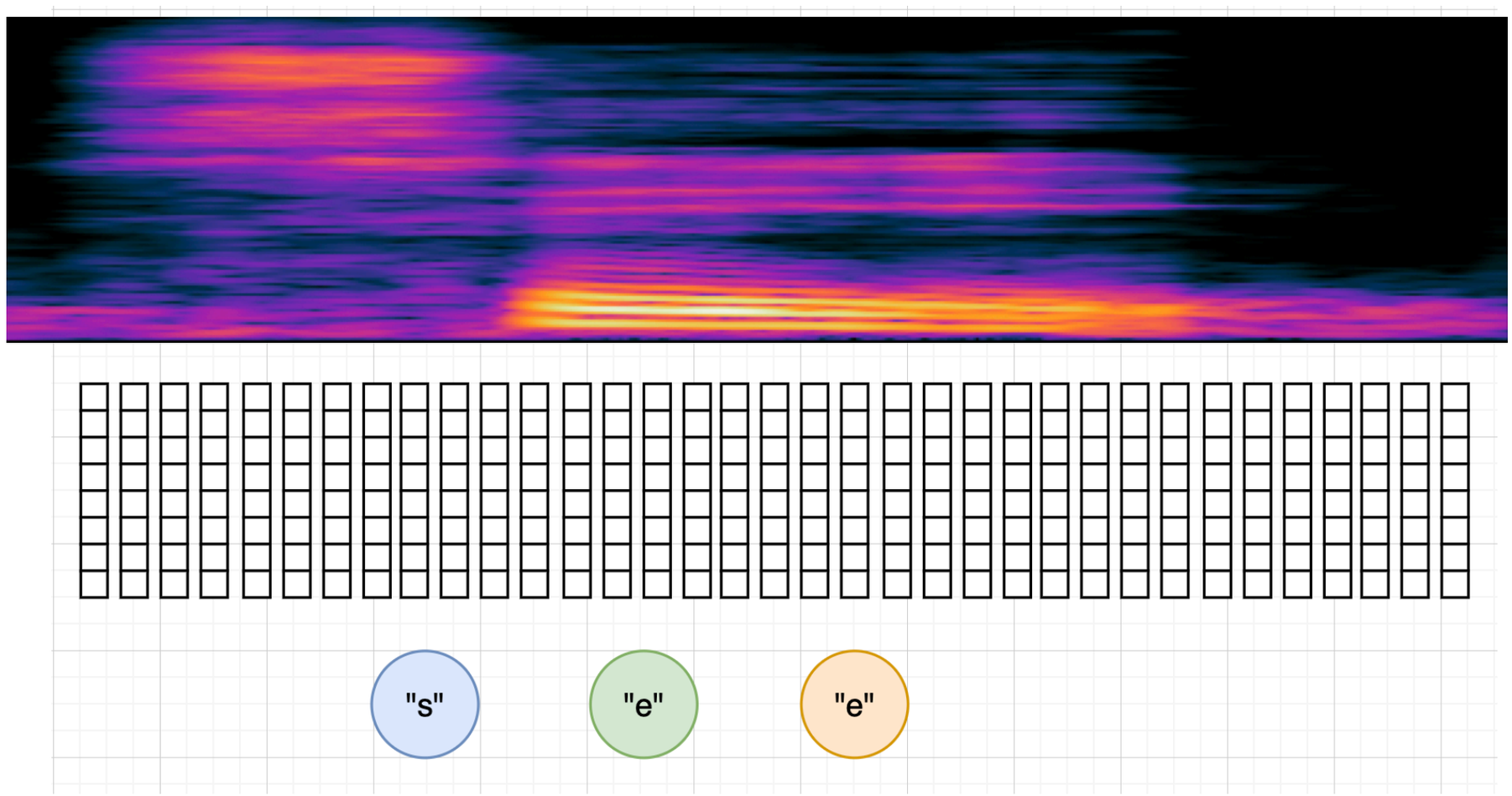
- Again, our staring point $p(W \mid O)$
- We introduce a **hard alignment path** as a random variable:

$$Z = (z_t \in \mathscr{V}' \mid t = 1, \ldots, T)$$

- We consider the following equations

$$\begin{aligned} p(W|O) &= \sum_Z p(Z, W|O) \\ &= \sum_Z p(Z|O) p(W|Z, O) \end{aligned}$$

Alignments



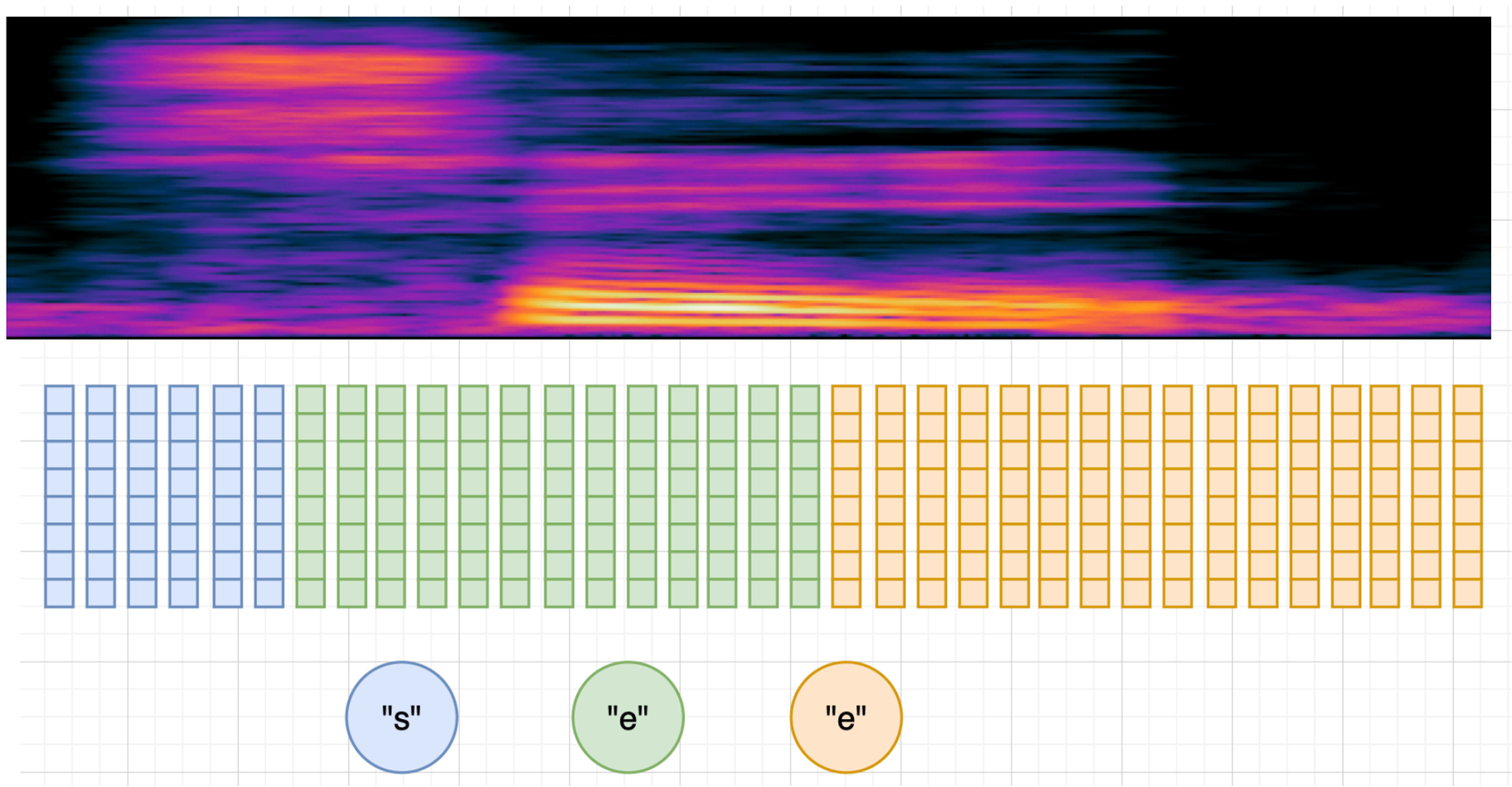
Speech features

$O\ (|O| = 35)$

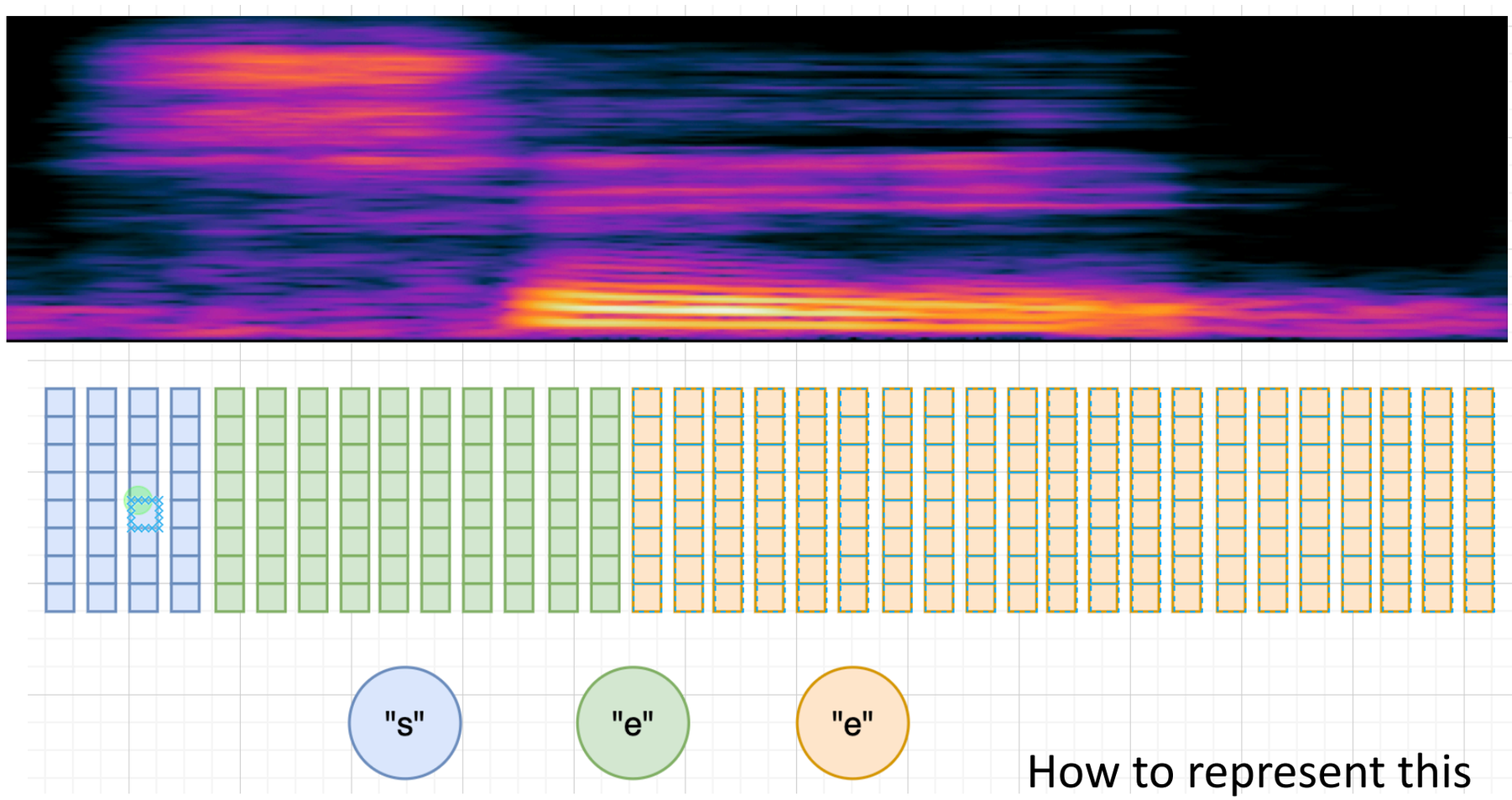
Token

$W(|W| = 3)$

Hard Alignments

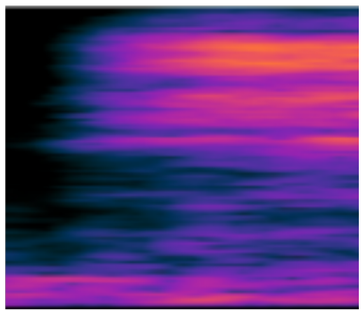


Hard Alignments



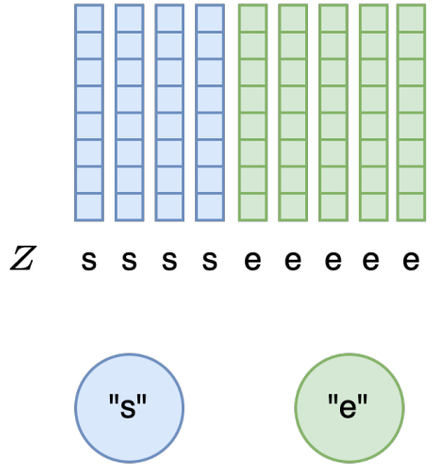
How to represent this information mathematically?

Alignment variable

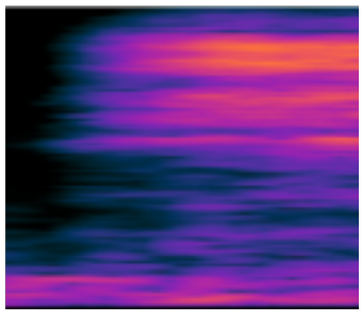


- z_1 = “s”, z_2 = “s”, .., z_5 = “e”, z_6 = “e”,

One possible realization of alignments

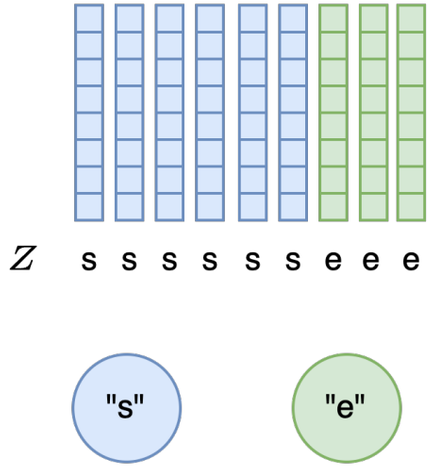


Alignment variable

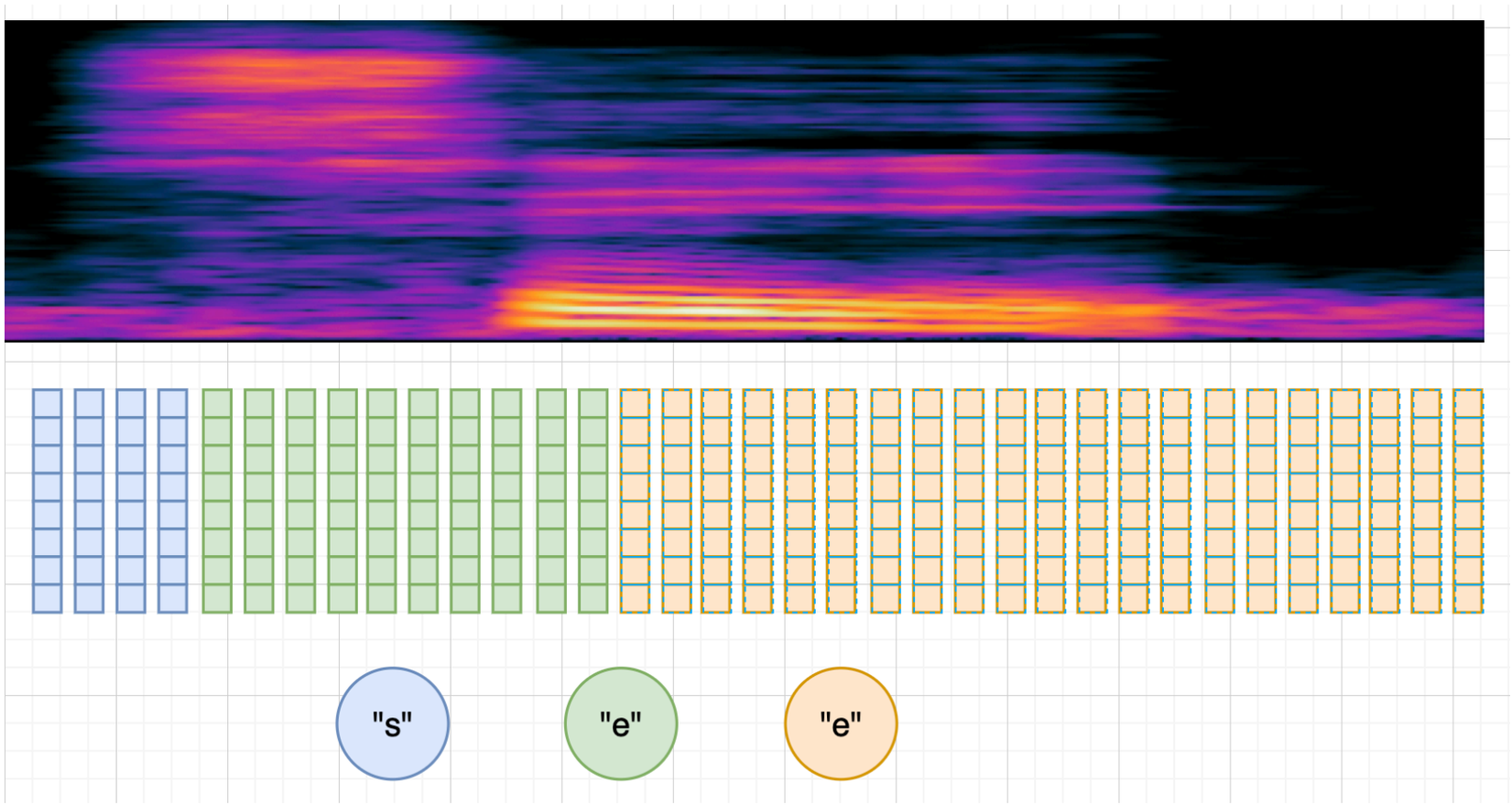


- z_1 = “s”, z_2 = “s”, .., z_7 = “e”, z_8 = “e”,

The other possibility



Hard alignments



We can represent a path as a sequence variable
This is called a hard alignment

$z_1 =$ “s”, $z_2 =$ “s”, .., $z_5 =$ “e”, …, $z_T =$ “e”

$Z = (z_t \in \{\text{“}s\text{”}, \text{“}e\text{”}, \ldots\} \mid t = 1, \ldots, T)$

HMM, CTC, and RNN-Transducer are based on the **hard** alignment

CTC case

- Again, our staring point $p(W|O)$
- We introduce a **hard alignment path** as a random variable: $Z = (z_t \in \mathscr{V}' | t = 1,\ldots,T)$
- We consider a set of all possible alignment paths representing W. We call this set as $\mathscr{Z}(W)$
- We incorporate this random variable to $p(W|O)$ as follows:

$$p(W|O) = \sum_{Z \in \mathscr{Z}(W)} p(Z|O)$$

  - Z is a sequence and it is difficult to deal with. Similar to the attention case, we use a chain rule to further factorize this equation:

$$p(W|O) = \sum_{Z \in \mathscr{Z}(W)} \prod_{t=1}^{T} p(z_t | z_{1:t}, O)$$

- This is not very difficult compared with an attention-based method, since Z **and** O **are based on the same length** T
- **RNN transducer can also be formulated in a similar manner**

Difference between E2E and HMM-based

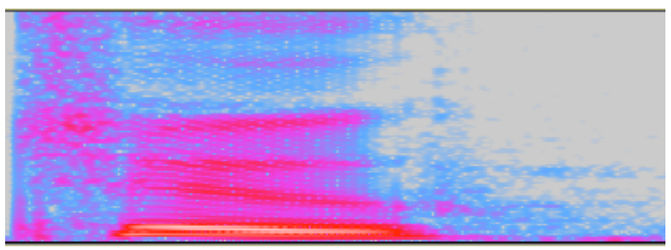
- $p(W|O)$ (end-to-end)

- $p(O|L)$ (HMM based)

- In $p(W|O)$ and $p(O|L)$, there are two differences

Simple case

- We know that phoneme /T/ is aligned with $O_{1:5}$, and phoneme /UW/ is aligned with $O_{6:18}$

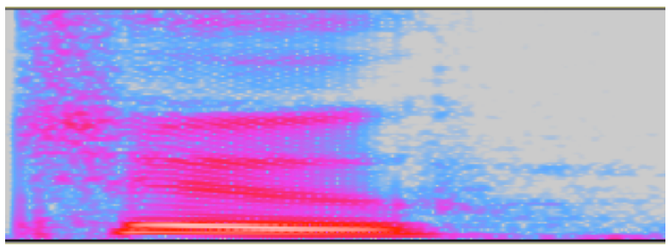


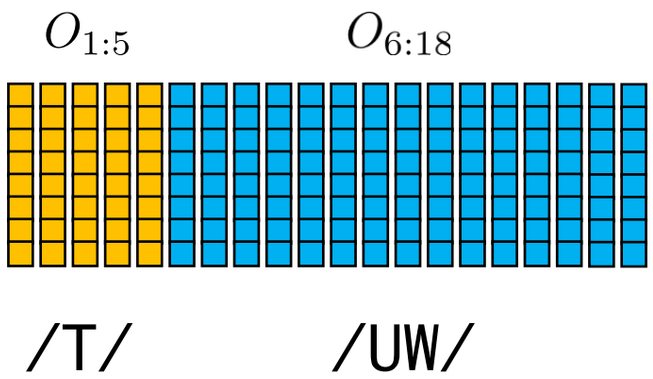
$$p(O_{1:18}|/\mathrm{T}/, /\mathrm{UW}/)$$

$$= p(O_{1:5}|/\mathrm{T}/)p(O_{6:18}|/\mathrm{UW}/)$$

This factorization is very natural if they are aligned

Simple case





- We know that phoneme /T/ is aligned with $O_{1:5}$, and phoneme /UW/ is aligned with $O_{6:18}$

$$
\begin{aligned}
p(O_{1:18}|/\mathrm{T}/,/\mathrm{UW}/) &= p(O_{1:5}, O_{6:18}|/\mathrm{T}/,/\mathrm{UW}/) \\
&= p(O_{1:5}|\cancel{O_{6:18}}, /\mathrm{T}/, \cancel{/\mathrm{UW}/})p(O_{6:18}|\cancel{/\mathrm{T}/}, /\mathrm{UW}/) \\
&= p(O_{1:5}|/\mathrm{T}/)p(O_{6:18}|/\mathrm{UW}/)
\end{aligned}
$$

Conditional independence assumptions

- Note that the conditional independence assumptions are “approximation or modeling”
- Not based on the derivation.

General case

- Factorize for each phoneme:

$$
\begin{aligned}
p(O|L) &= p(O_{1:T_1}, O_{T_1+1:T_2}, \cdots | l_1, l_2, \cdots) \\
&= p(O_{1:T_1} | O_{T_1+1:T_2}, \cdots, l_1, l_2, \cdots) p(O_{T_1+1:T_2}, \cdots, | l_1, l_2, \cdots) \\
&= p(O_{1:T_1} | l_1) p(O_{T_1+1:T_2}, \cdots, | l_1, l_2, \cdots) \\
&\vdots \\
&= p(O_{1:T_1} | l_1) p(O_{T_1+1:T_2} | l_2) \cdots \\
&= \prod_{i=1}^{J} p(O_{T_{j-1}+1:T_j} | l_j)
\end{aligned}
$$

- If we know the alignment information, we can factorize the acoustic model probability for each phoneme
- We use conditional independence assumptions very aggressively

General case

- Factorize for each phoneme:

$$
\begin{aligned}
p(O|L) &= p(O_{1:T_1}, O_{T_1+1:T_2}, \cdots | l_1, l_2, \cdots) \\
&= p(O_{1:T_1} | O_{T_1+1:T_2}, \cdots, l_1, l_2, \cdots) p(O_{T_1+1:T_2}, \cdots, | l_1, l_2, \cdots) \\
&= p(O_{1:T_1} | l_1) p(O_{T_1+1:T_2}, \cdots, | l_1, l_2, \cdots) \\
&\vdots \\
&= p(O_{1:T_1} | l_1) p(O_{T_1+1:T_2} | l_2) \cdots \\
&= \prod_{i=1}^{J} p(O_{T_{j-1}+1:T_j} | l_j)
\end{aligned}
$$

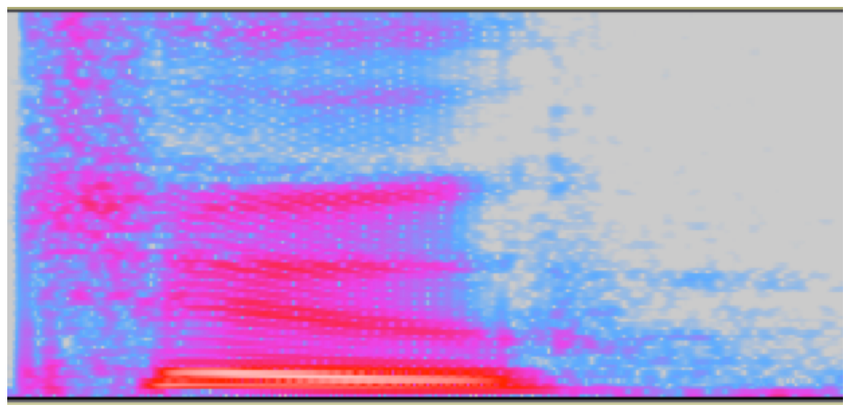
Product rule (chain rule)

Conditional independence assumption

- If we know the alignment information, we can factorize the acoustic model probability for each phoneme
- We use conditional independence assumptions very aggressively

Word -> Phoneme -> State

- We need a **more precise unit** to present an acoustic dynamics



“two”

Introduce phoneme representation

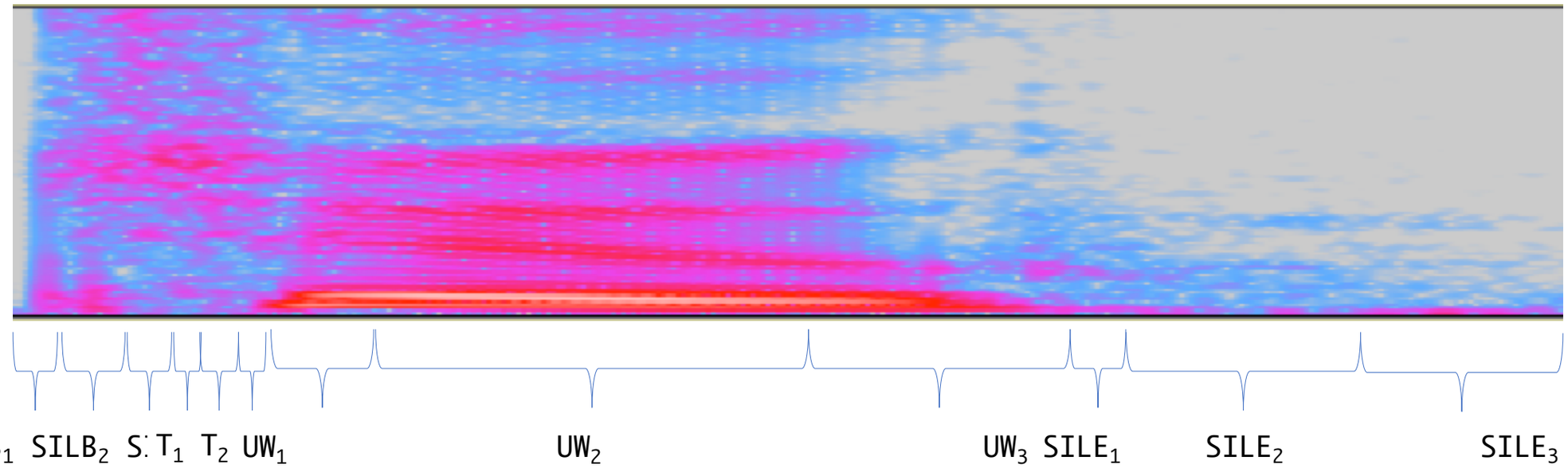
/SILB/ /T/ /UW/ /SILE/

$SILB_1$ $SILB_2$ $SILB_3$ T_1 T_2 T_3 UW_1 UW_2 UW_3 $SILE_1$ $SILE_2$ $SILE_3$

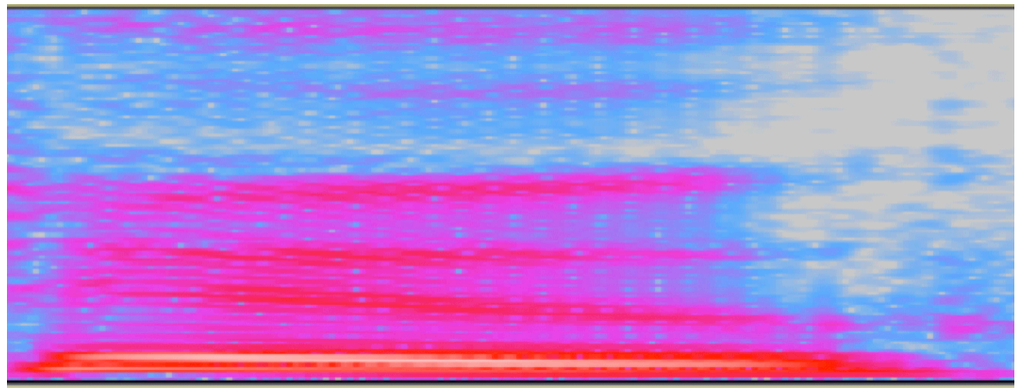
Introduce further precise representation (we call it **state**)

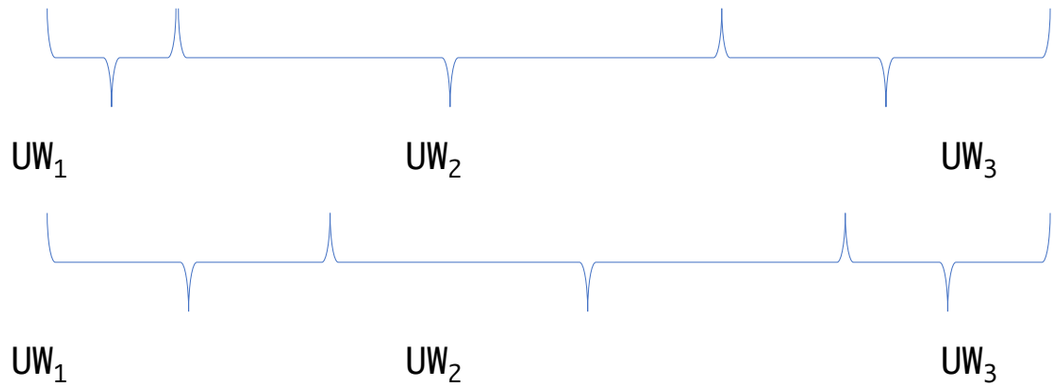
How to align states to features?

- Alignment problem

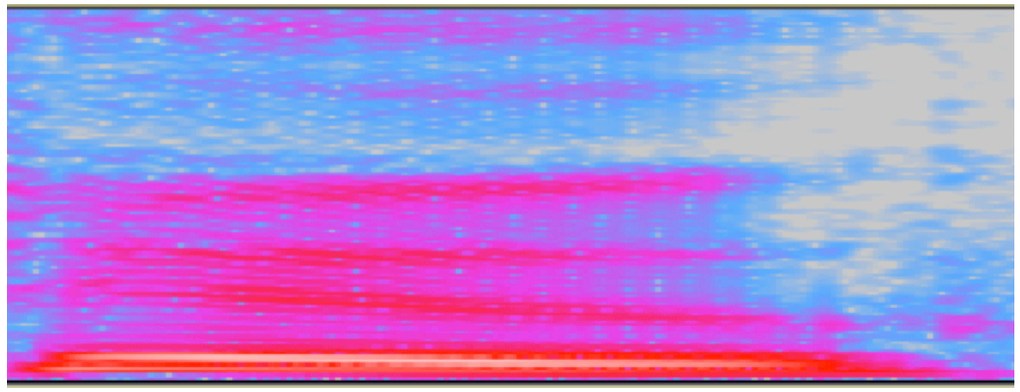


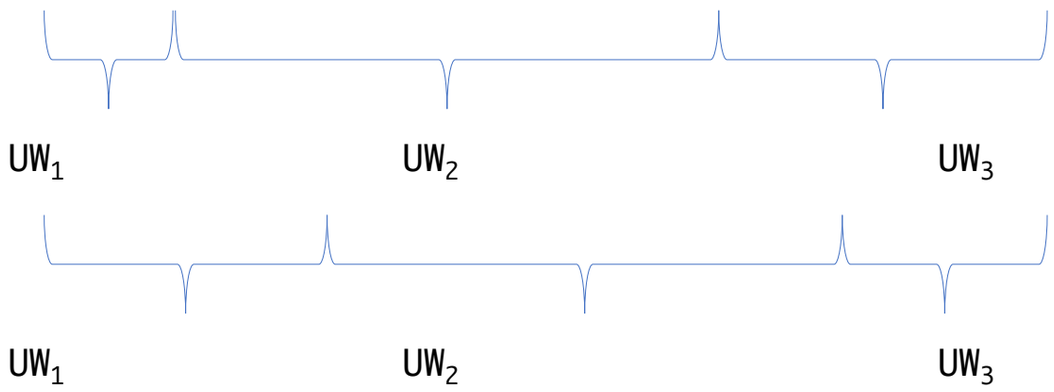
We don’t know the alignment information…

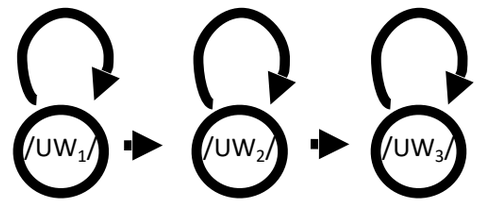




We don’t know the alignment information…





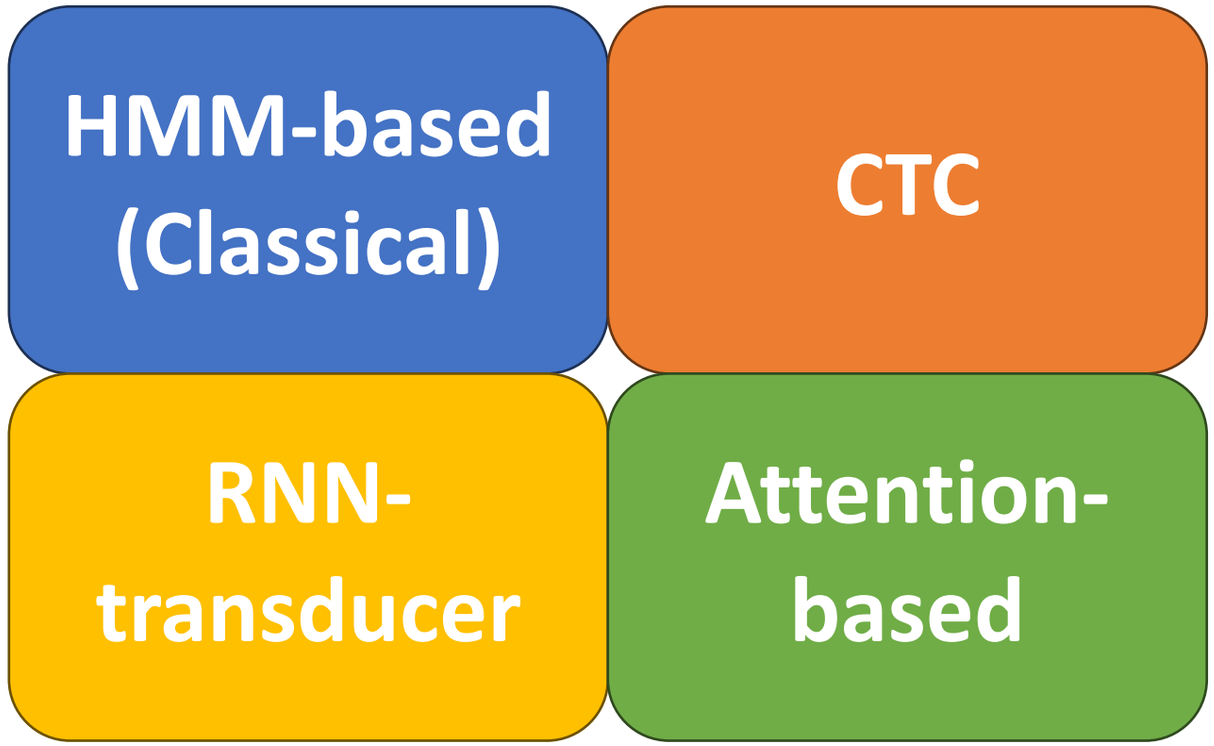


We use HMM!

We will use the similar formulation (introducing **alignment variable z**) to CTC/RNN-T

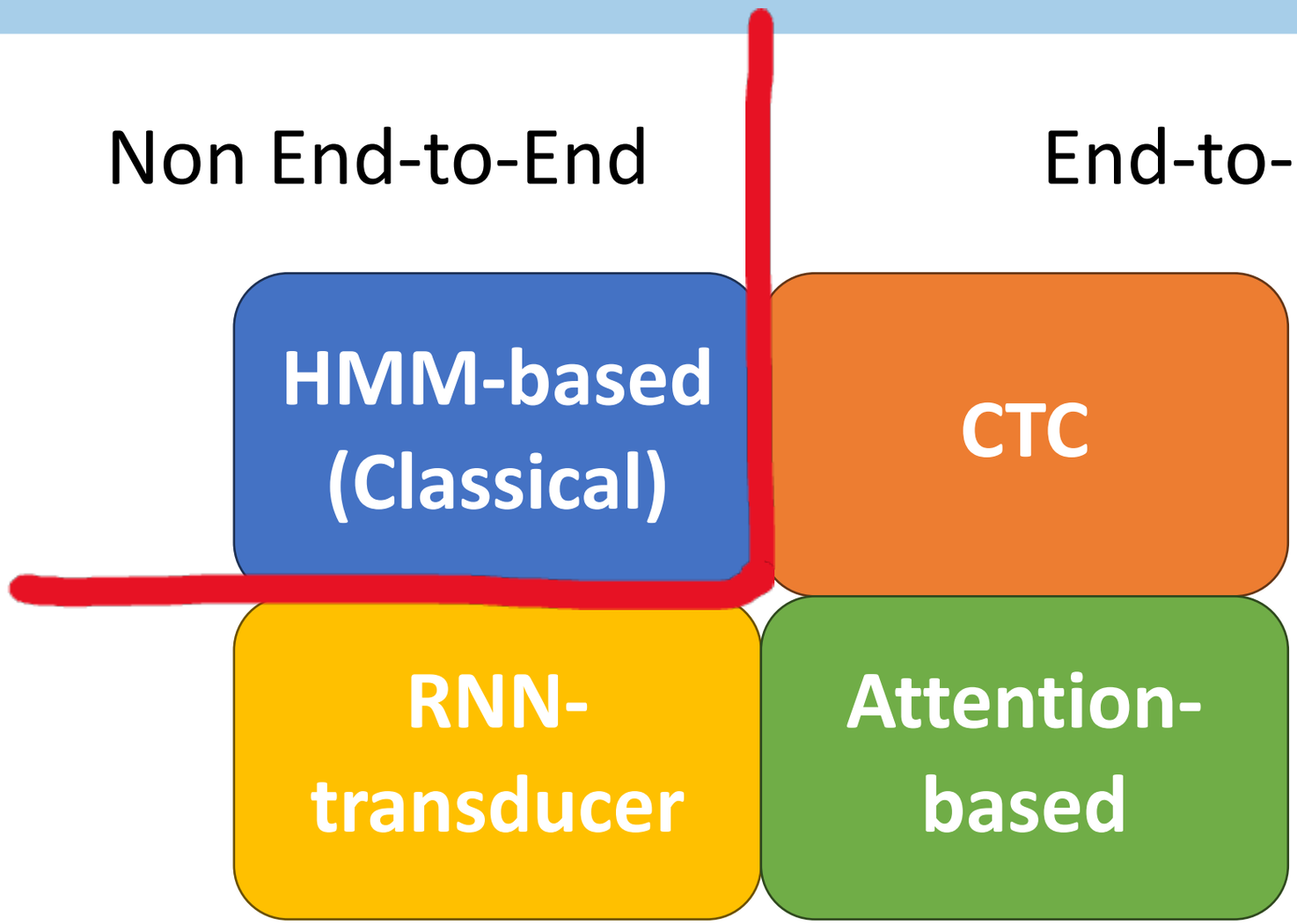
Interim Summary

- The most difficult issue in speech recognition: the input and output lengths are different
  - We need some alignments
- The soft alignment case
  - We use an attention-based neural network (this will be introduced later).
- The hard alignment cases
  - We explicitly introduce an alignment path z.
  - We can use it for CTC, RNN-transducer, and HMM-based approach

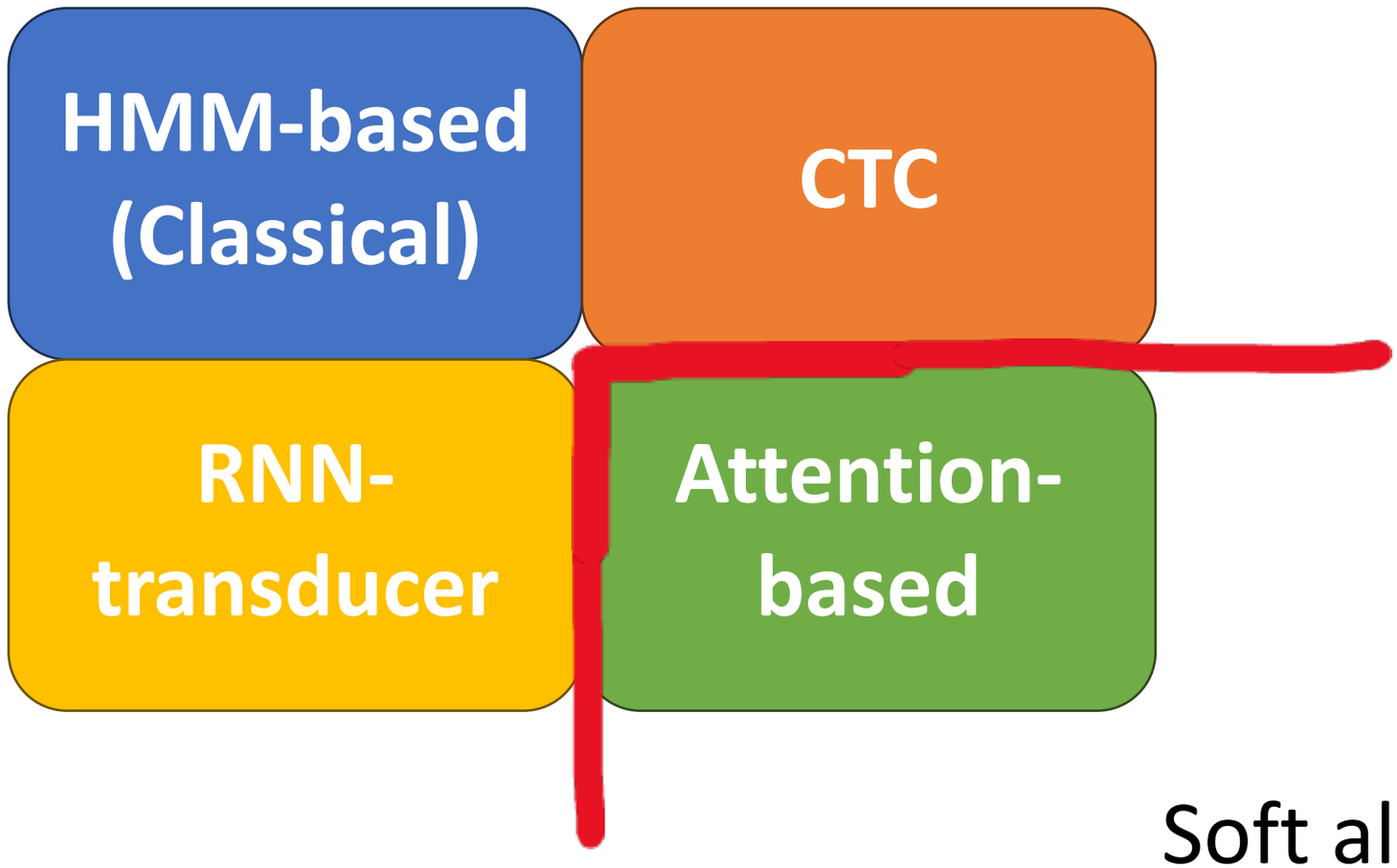

HMM-based (Classical)
CTC
RNN-transducer
Attention-based

Non End-to-End

End-to-End



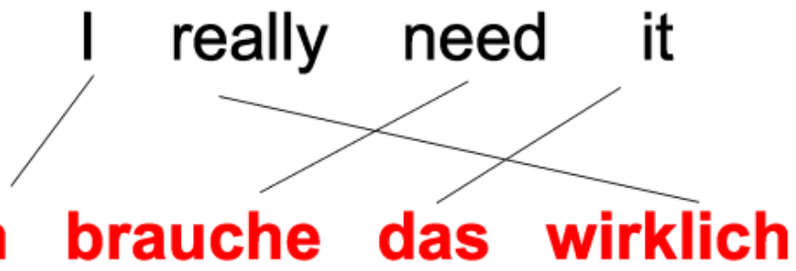
Hard alignment



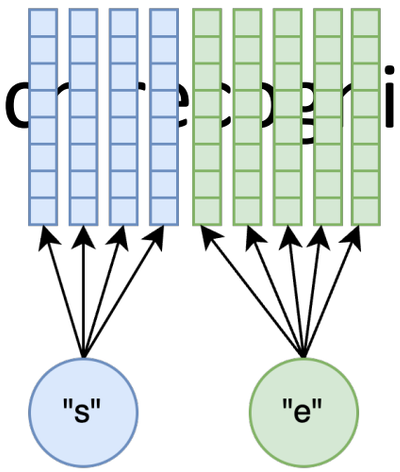
Soft alignment

Monotonic vs. non-monotonic

- Machine translation case (**non monotonic**)



- Speech recognition case (**monotonic**)

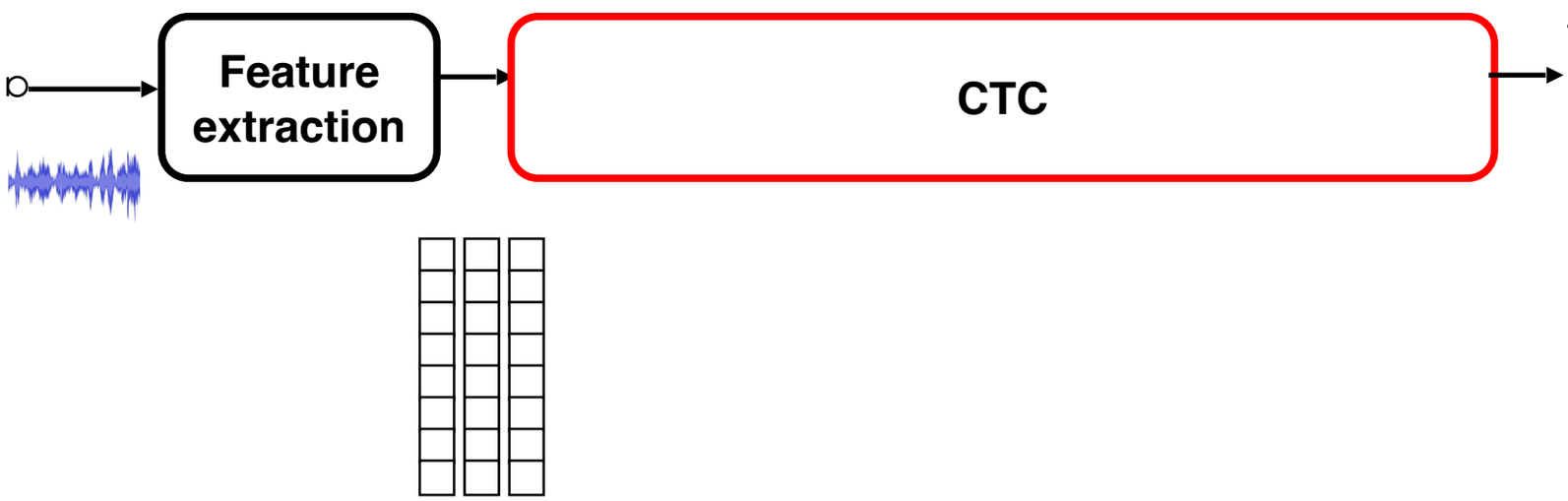


Agenda

- Alignment paths
  - CTC
  - RNN-Transducer
  - HMM (3-state left-to-right)

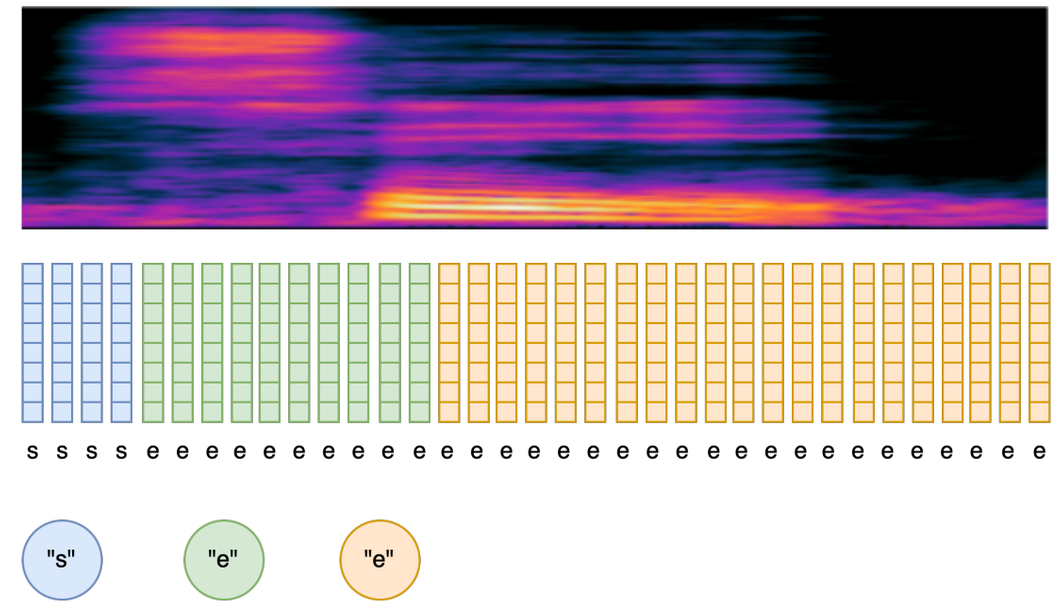
Speech recognition pipeline

$$p(O \mid W)$$



"I want to **go to** the CMU campus"

Hard alignments for repeated tokens



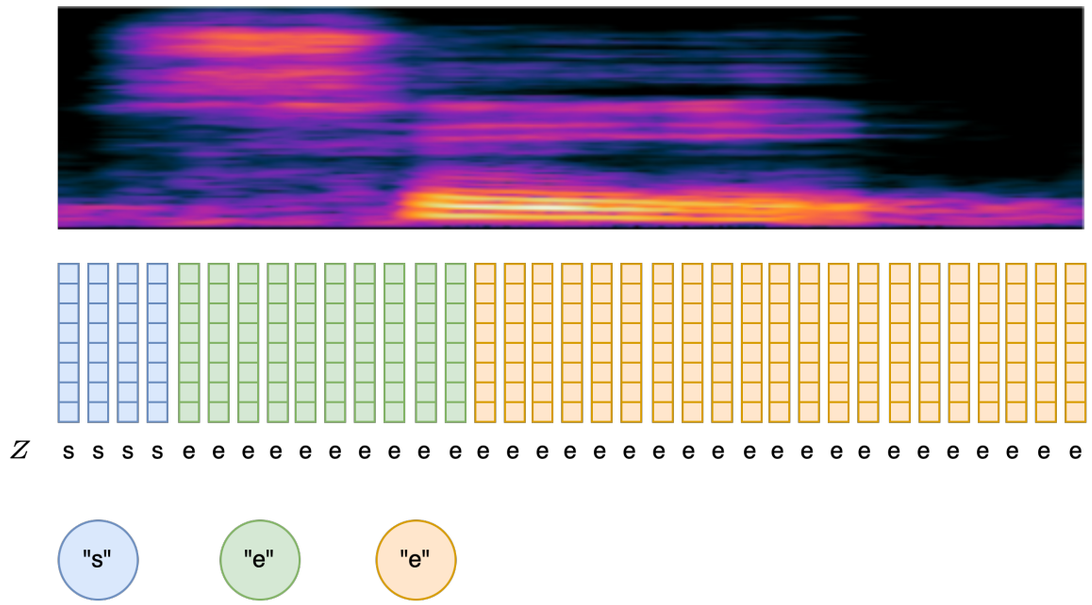
- Hard alignment examples
- How to distinguish them?
  - (“s”, “e”, “e”)

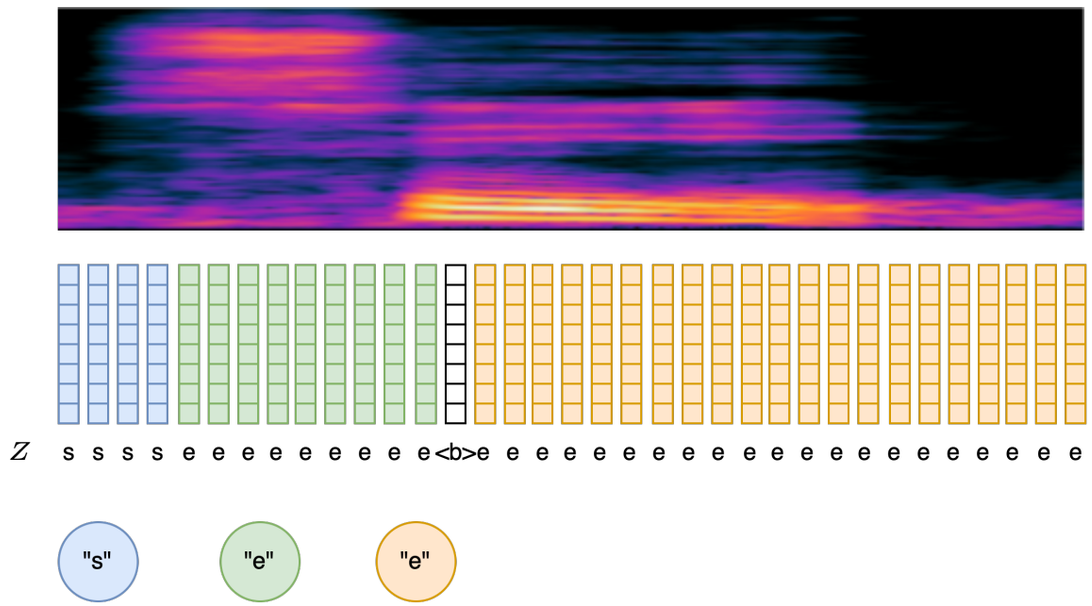
    vs.
  - (“s”, “e”)
  - Both are written as
  - $Z = (sssseeeeeeeeeeeeeeeee\ldots)$

Hard alignments for repeated tokens

- Hard alignment examples
- How to distinguish them?
  - (“s”, “e”, “e”)

    vs.
  - (“s”, “e”)
  - Both are written as
  - $Z = (sssseeeeeeeeeeeeeeee\ldots)$
- We introduce the blank symbol!
  - $Z = (sssseeeeeeeee < b >\ eee\ldots)$



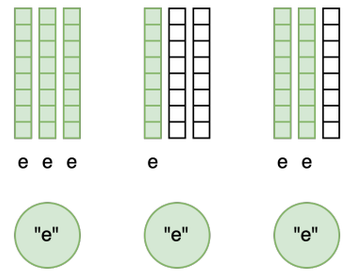


Introduction of blank symbol <b>

- First, we insert <b> to the character sequence ”see”

  → W = (“s”, “e”, “e”), where $|W| = J$

- Then, **expand W' to the frame length T** to form Z
  - Assuming that $T > J$ in general: We cannot use CTC if it is not satisfied
- All tokens and <b> can be repeated to adjust the length
  - For example, if “e” is repeated three times
  - W = “e” → Z = (”e”, “e”, “e”)
  - W = “e” → Z = (“e”, “e”, “<b>”)
  - W = “e” → Z = (“e”, “<b>”, “<b>”)
- <b> must be inserted between repeated character
  - W = (“e”, “e”), then Z = (…“e”, ”<b>”, ”e”…): we **cannot** skip <b>
  - W = (“s”, “e”), then, Z = (…“s”, “e”…): we can skip <b>



Example of Z

- $W = (\text{“s”}, \text{“e”}, \text{“e”}),\ T = 5$

Then

$Z = (\text{“<b>”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”}),$ or

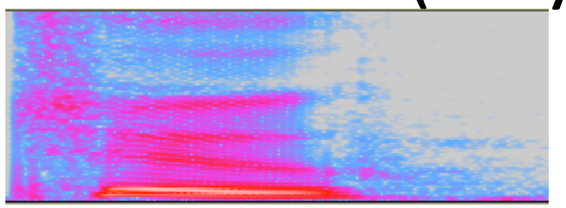
$Z = (\text{“s”}, \text{“<b>”}, \text{“e”}, \text{“<b>”}, \text{“e”}),$ or

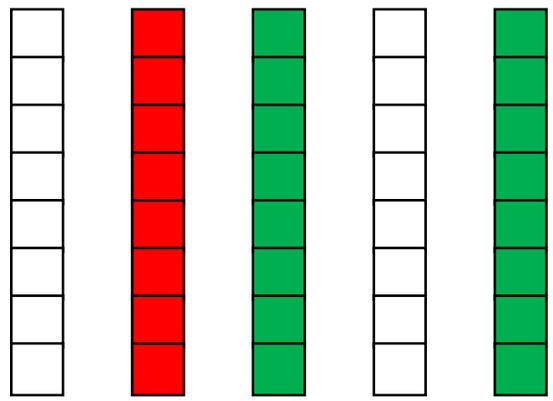
$Z = (\text{“s”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”}),$ or

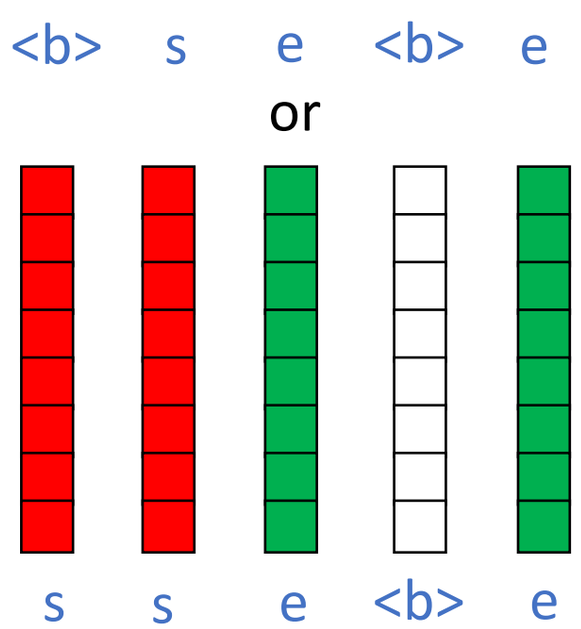
….

This is an alignment problem

“see” (only 5 frames)







Example of Z

“see” (only 5 frames)

- $W = (\text{“s”}, \text{“e”}, \text{“e”}),\ T = 5$

Then

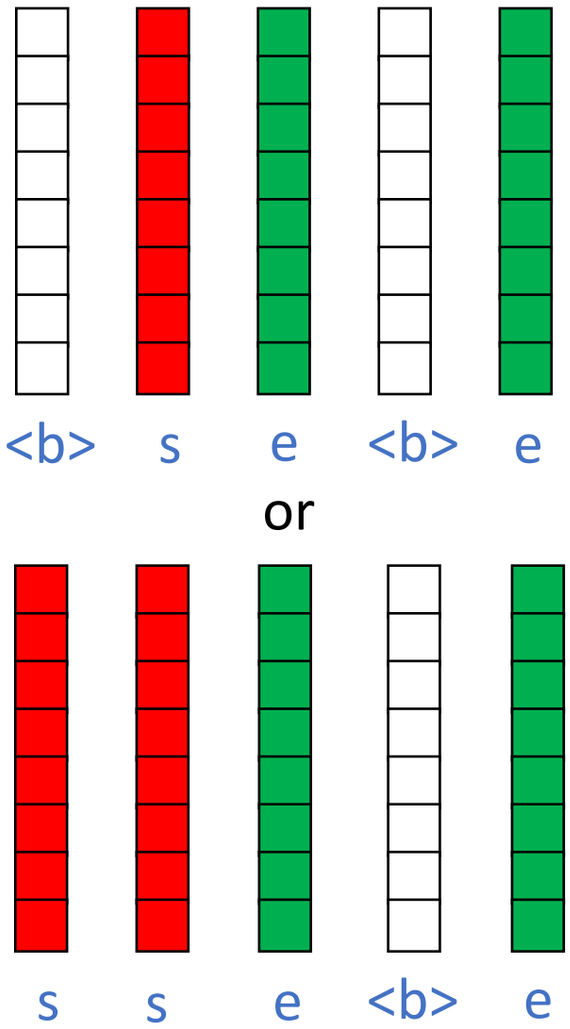
$Z = (\text{“<b>”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”}) \rightarrow W = (???)$

$Z = (\text{“s”}, \text{“<b>”}, \text{“e”}, \text{“<b>”}, \text{“e”}) \rightarrow W = (???)$

$Z = (\text{“s”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”}) \rightarrow W = (???)$

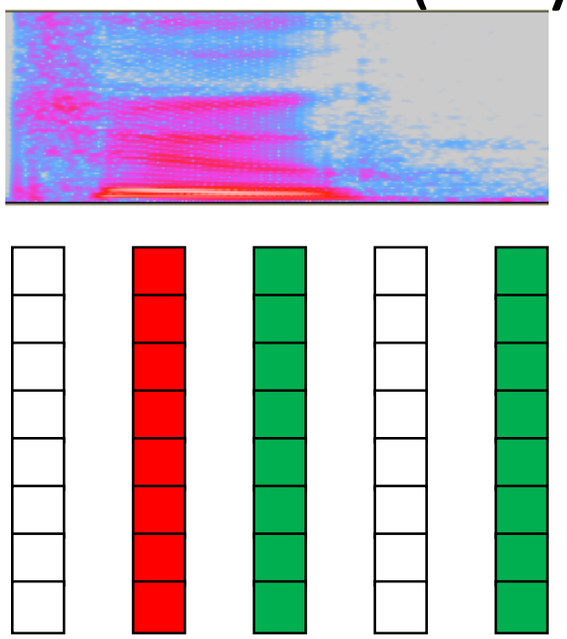
….

- $f: Z \rightarrow W$: many to one mapping
  1) Remove repeated tokens
  2) Remove the blank token <b>



Example of Z

“see” (only 5 frames)



- $W =$ (“s”, “e”, “e”), $T = 5$

Then

$Z =$ (“<b>”, “s”, “e”, “<b>”, “e”), or

$Z =$ (“s”, “<b>”, “e”, “<b>”, “e”), or

$Z =$ (“s”, “s”, “e”, “<b>”, “e”), or

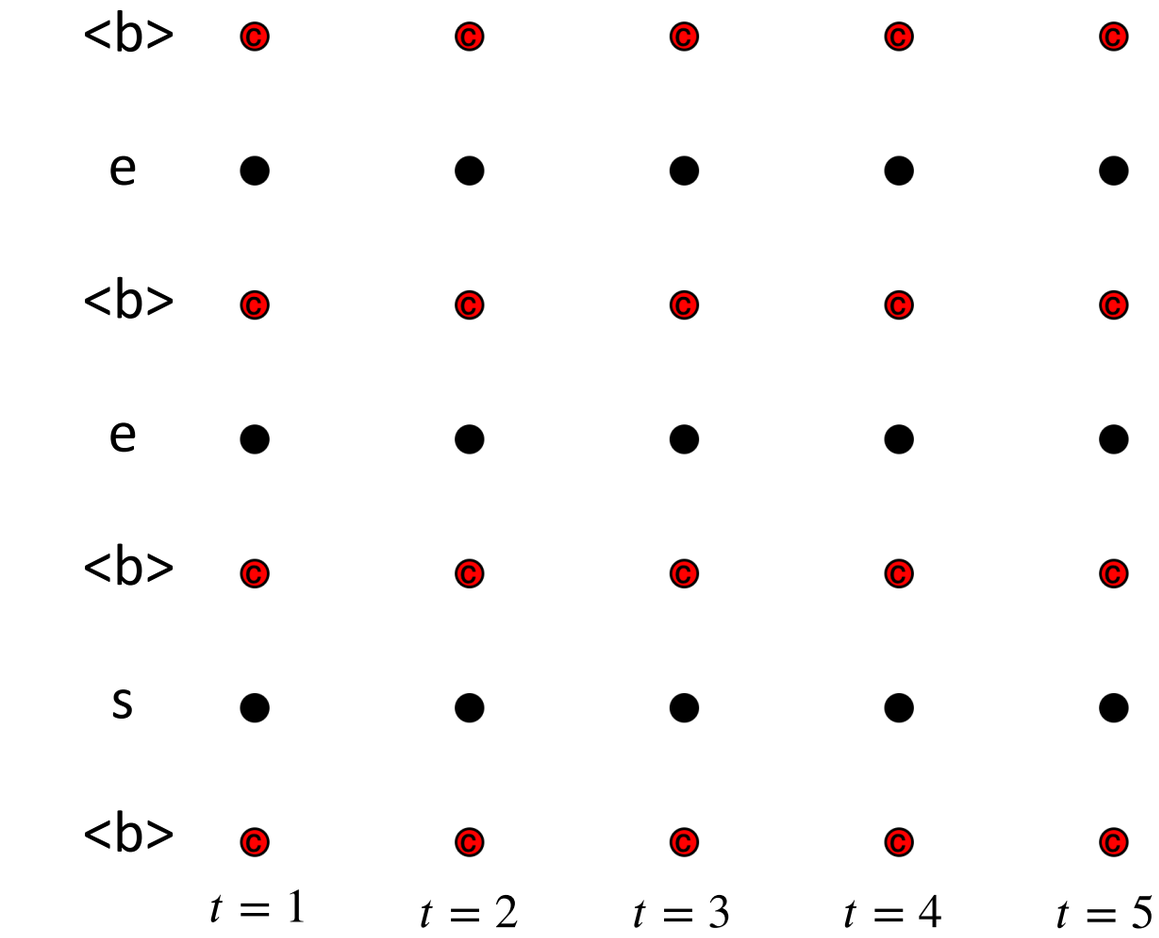
….

This is an alignment problem

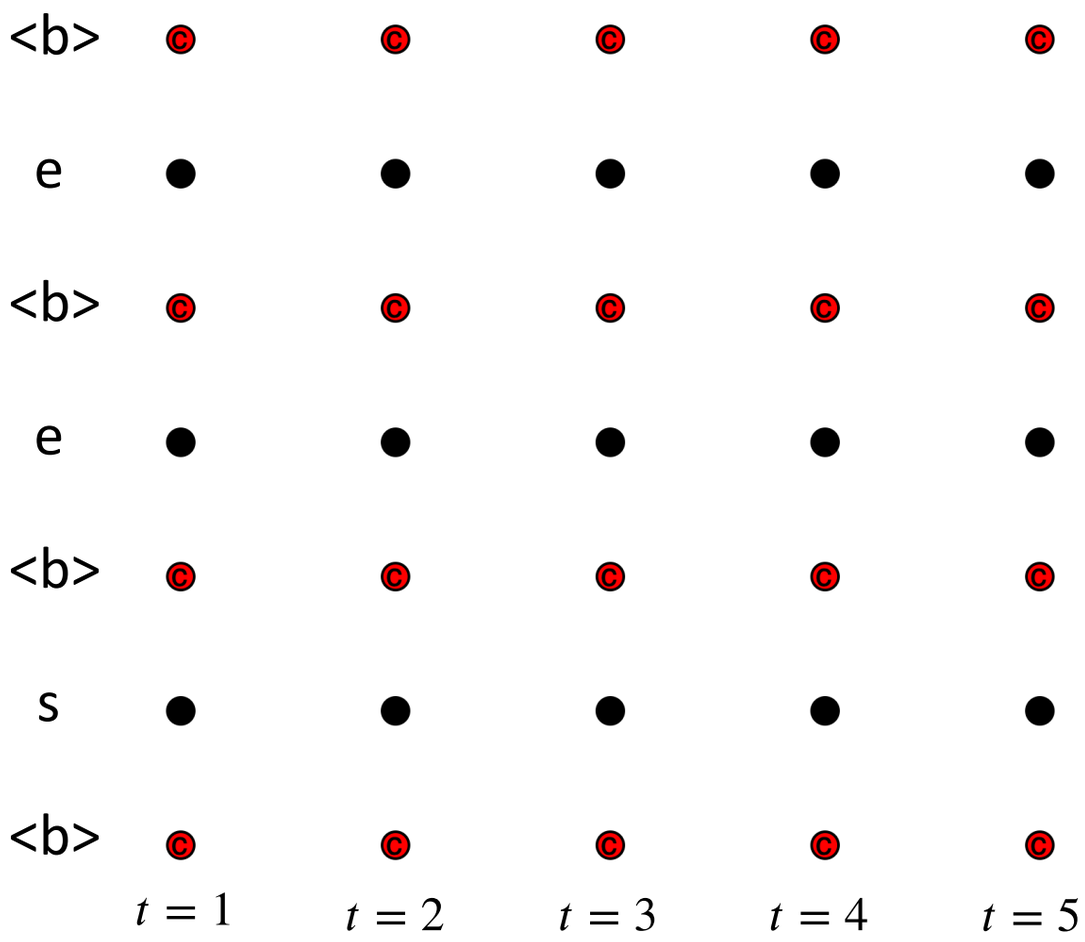
Note that

- $f: Z \rightarrow W$: many to one mapping
  1) Remove repeated tokens
  2) Remove the blank token <b>
- $f^{-1}: W \rightarrow Z$: one to **many** mapping
  - How to efficiently represent it? We use the trellis representation

Trellis of Z



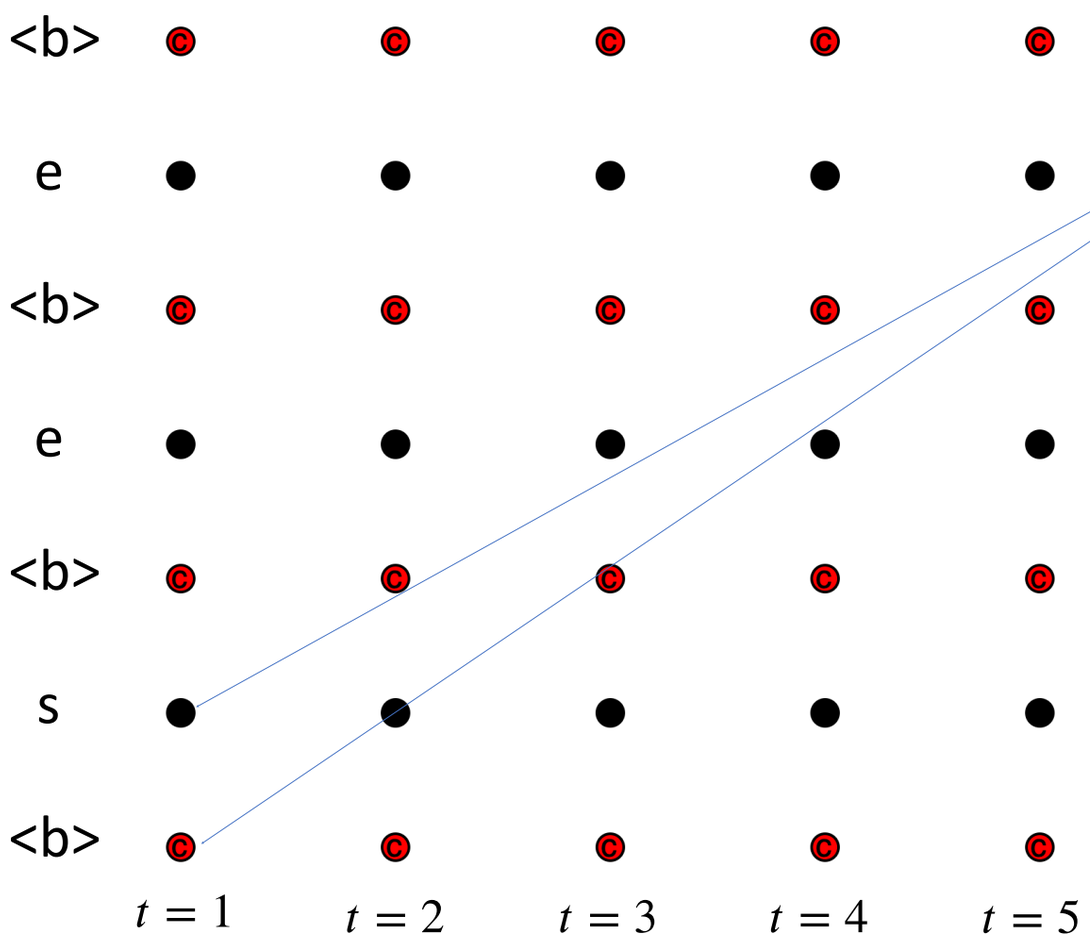
Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

- Note that all arcs consume

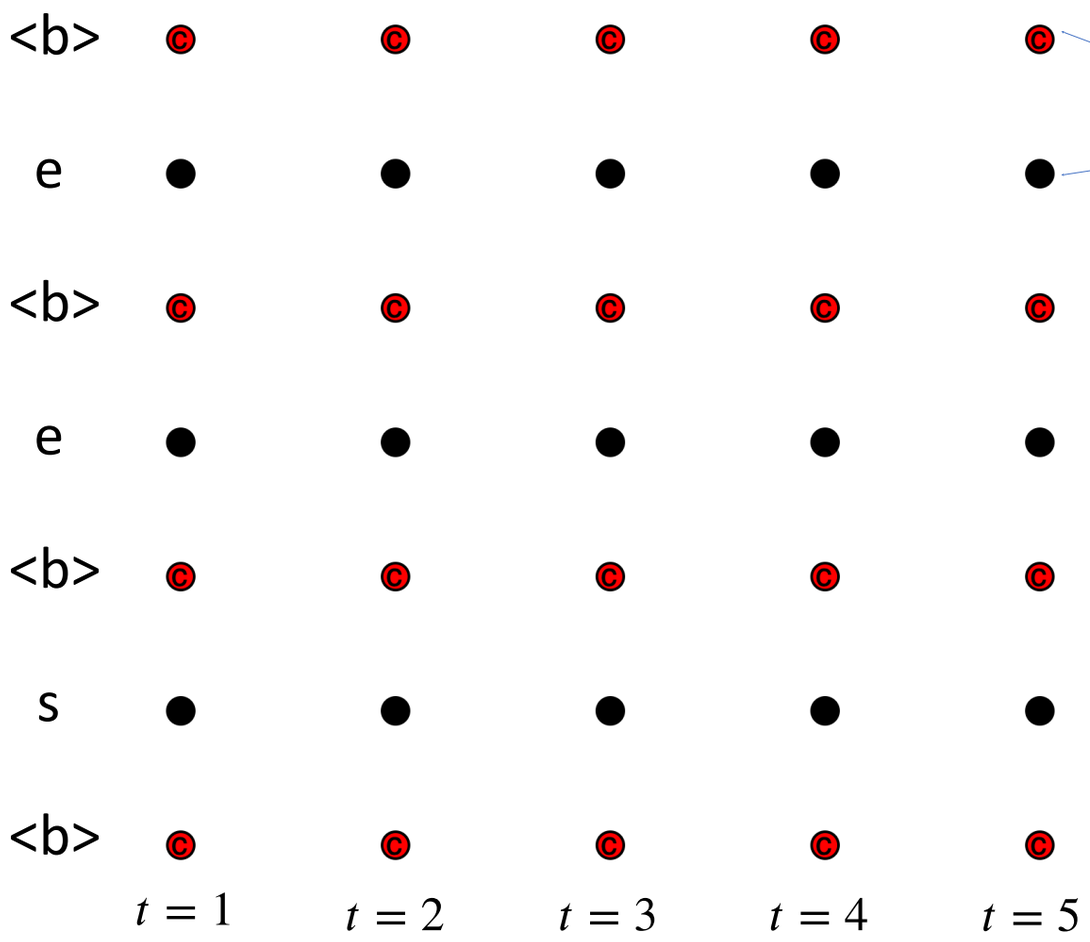
Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

- Note that all arcs consume

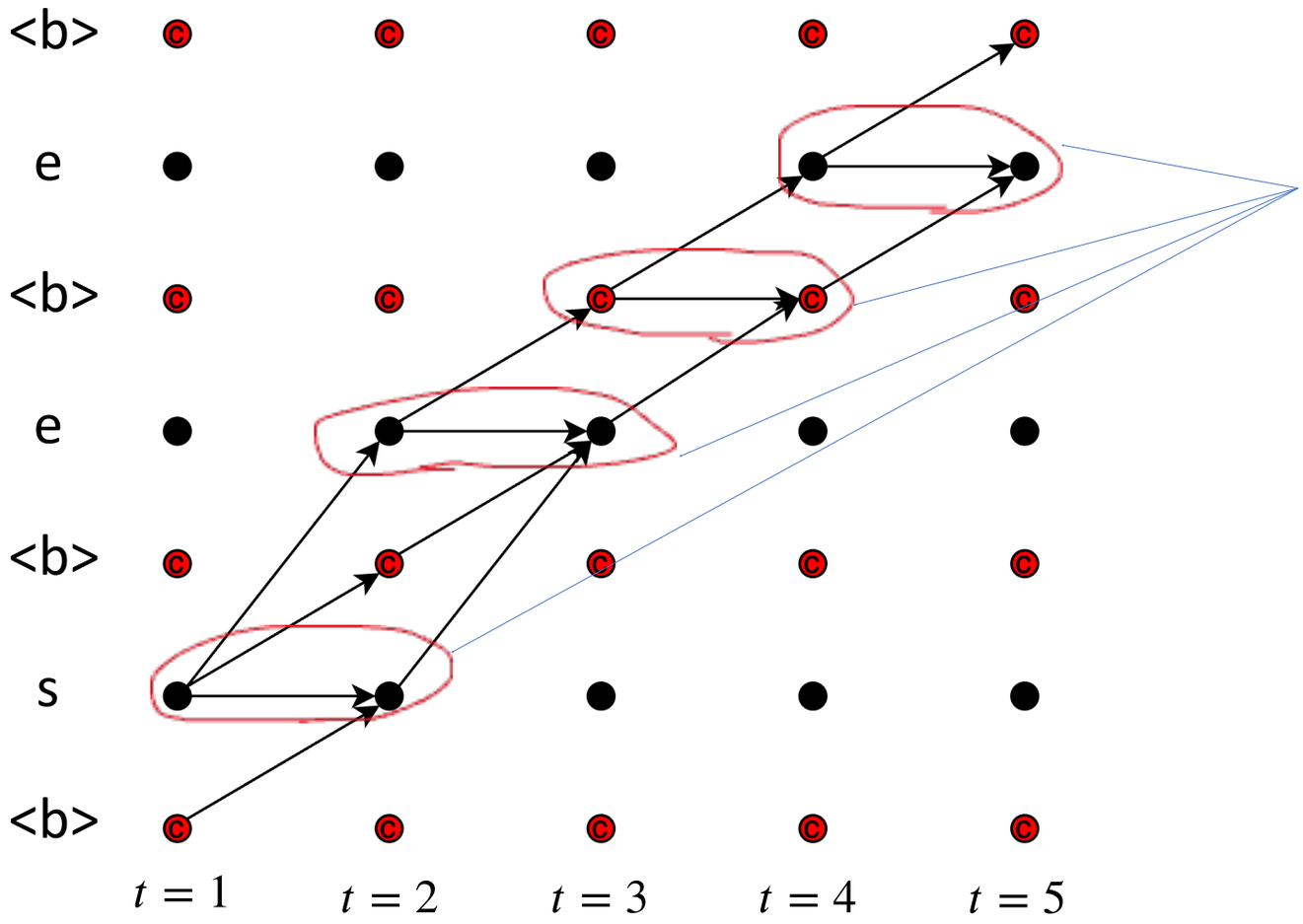
Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - “s”, ”<b>”, ”e”: we can skip <b>
   - “e”, ”<b>”, ”e”: we **cannot** skip <b>

- Note that all arcs consume

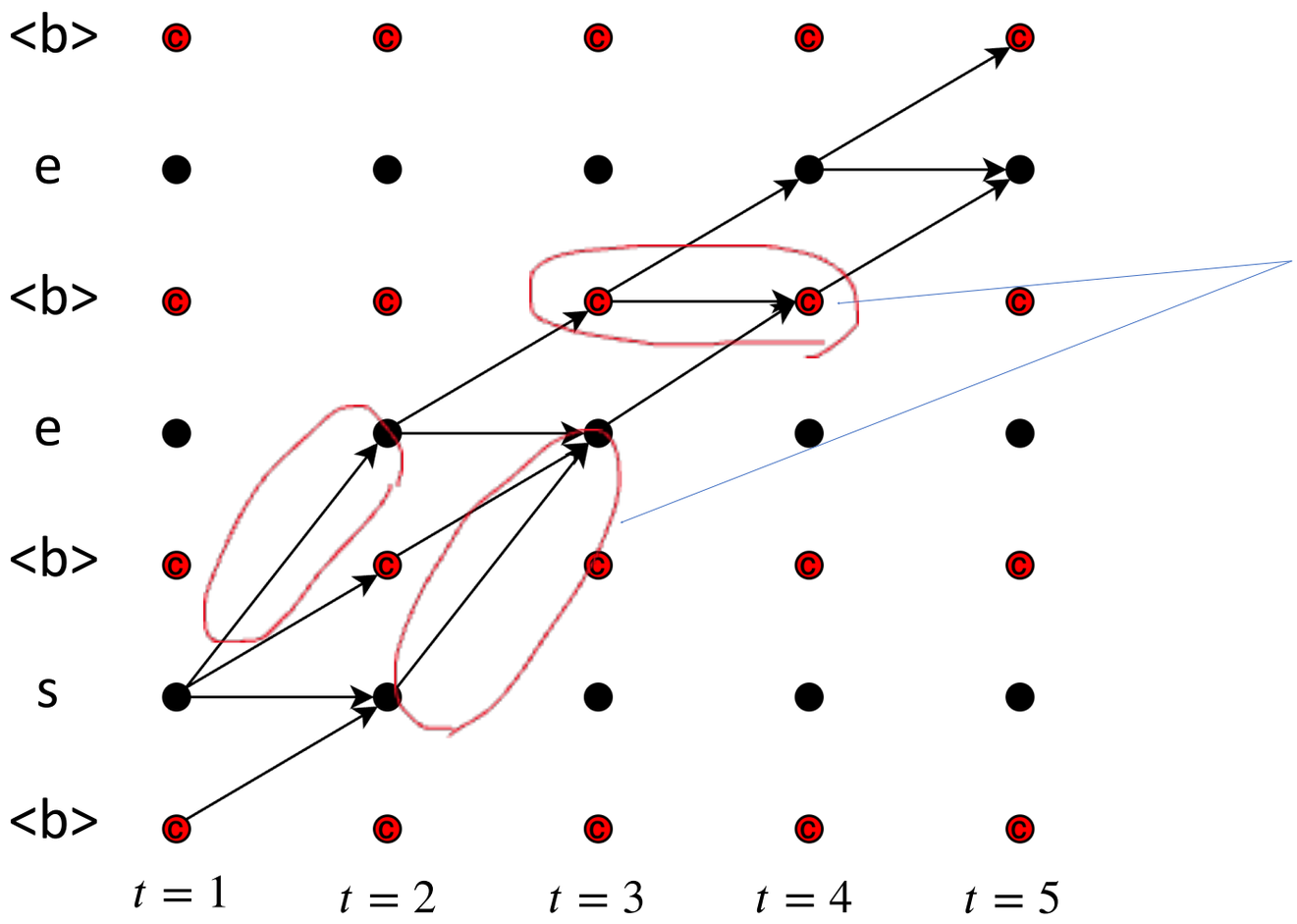
Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

- Note that all arcs consume

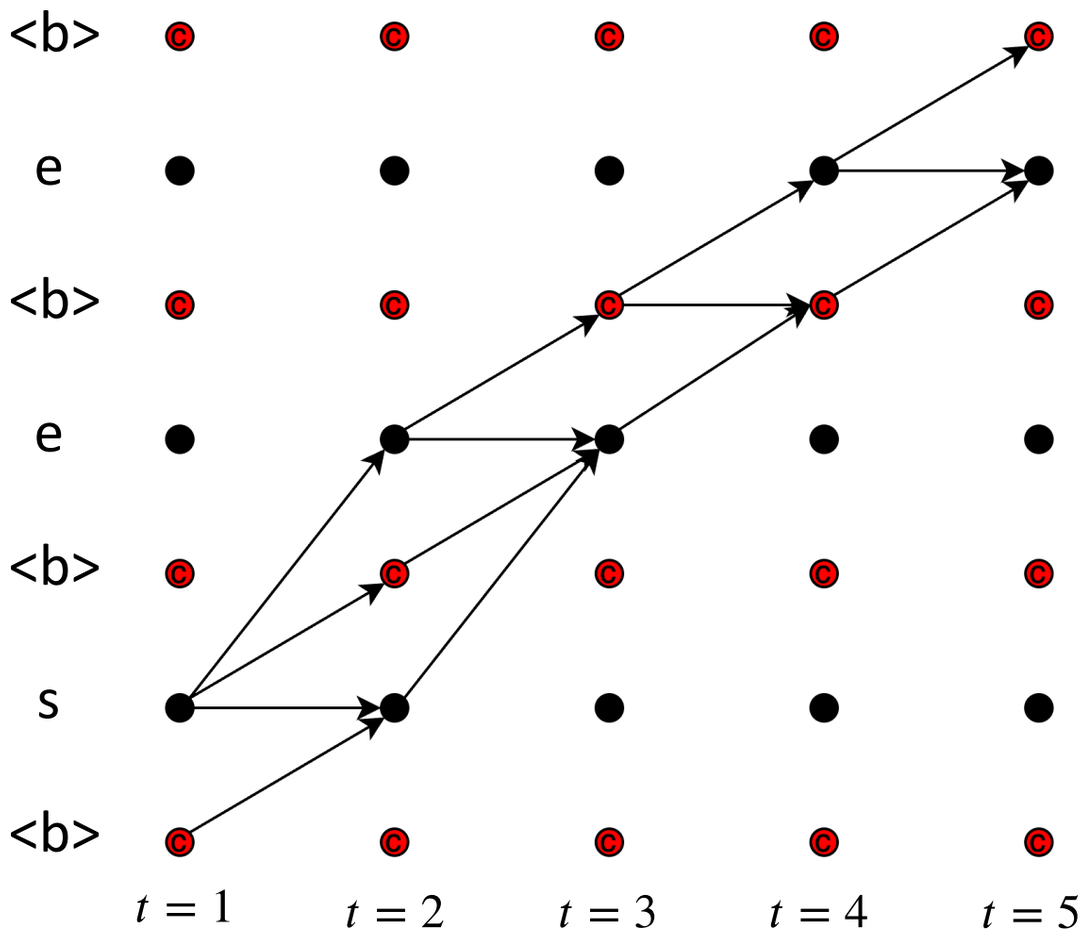
Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

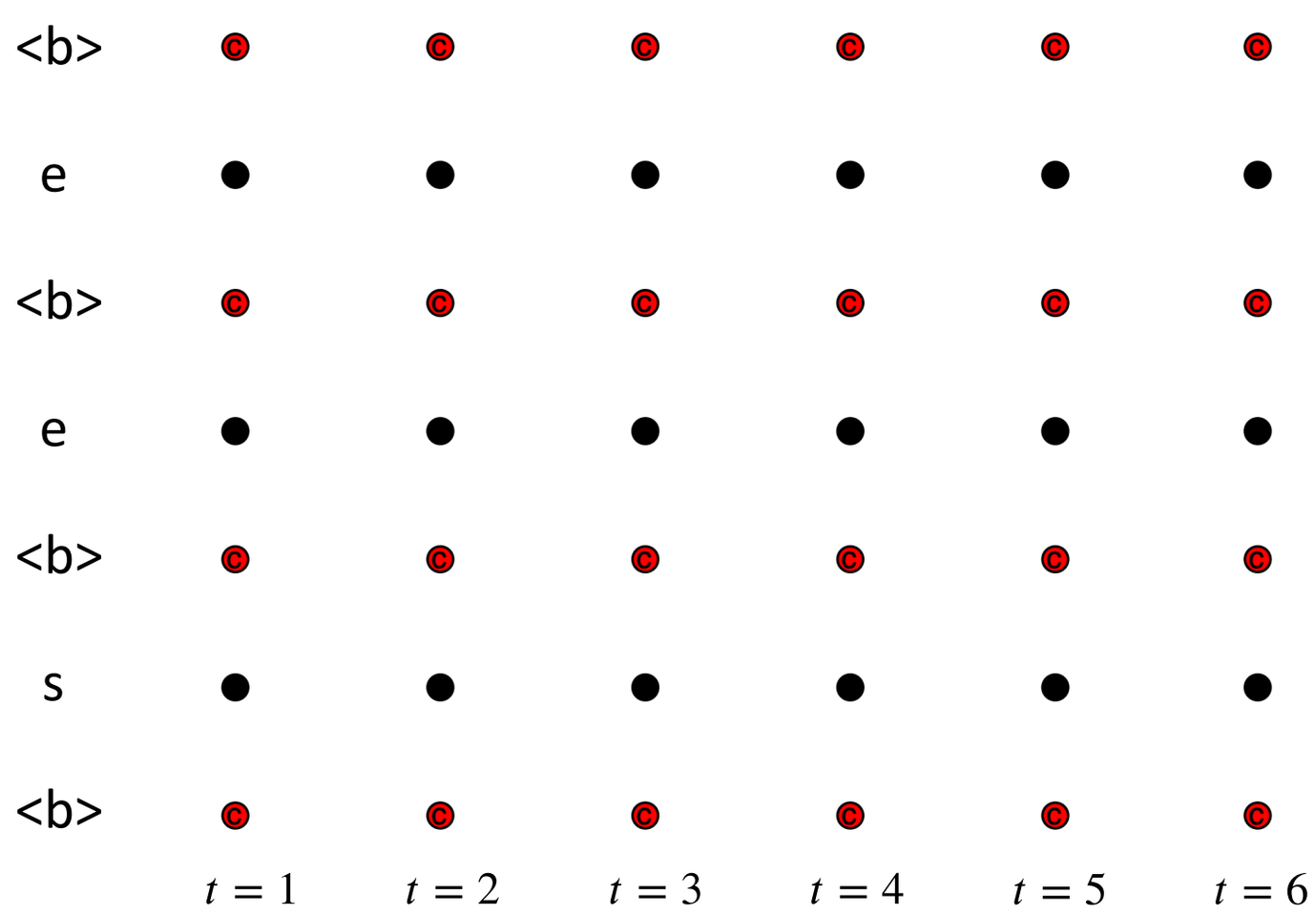
- Note that all arcs consume

Trellis of z



1. Must start at the first <b> or s
2. Must end at the last <b> or e
3. All characters can be repeated
4. <b> can be skipped except when it is inserted between repeated character
   - "s", "<b>", "e": we can skip <b>
   - "e", "<b>", "e": we **cannot** skip <b>

- Note that all arcs consume


<b>
e
<b>
e
s
<b>
$t = 1$
$t = 2$
$t = 3$
$t = 4$
$t = 5$
$t = 6$

How to represent it as an actual function?

- We define an alignment variable as $Z = (z_t \in \mathcal{V} \cup < \text{b} > | t = 1, \ldots, T)$
  - $|Z| = |\text{O}| = \text{T}$
  - The vocabulary set is augmented by the blank symbol <b>
- We incorporate this random variable to $p(W|O)$ as follows:

$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|\cancel{z_{1:t-1}}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|O)
\end{aligned}
$$

- $p(z_t|O)$ is represented by a neural network (Bidirectional LSTM or self-attention)

- $f^{-1}(W)$: **all possible Z representing W**
- Use a chain rule to further factorize Z
- Perform conditional independence assumptions and ignore the dependency of previous alignments $z_{1:t-1}$ (this is an approximation)
- We need to solve the **summuzation over all possible paths** $\sum_{Z}$

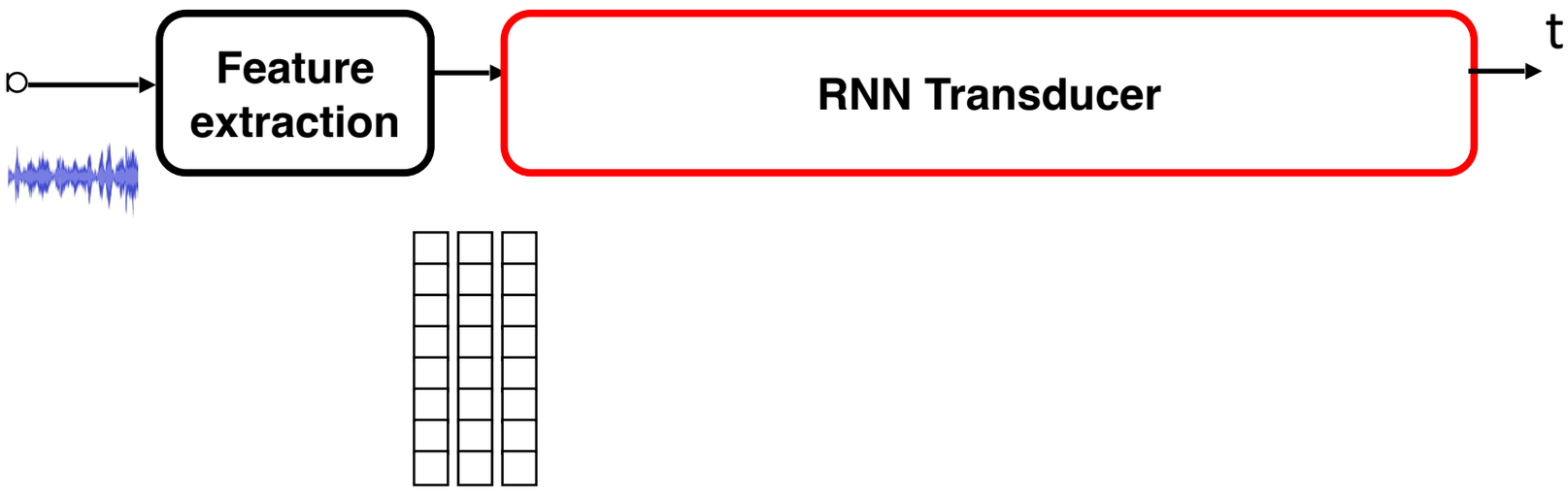
←Exponential

Agenda

- Alignment paths
  - CTC
  - RNN-Transducer
  - HMM (3-state left-to-right)

Speech recognition pipeline

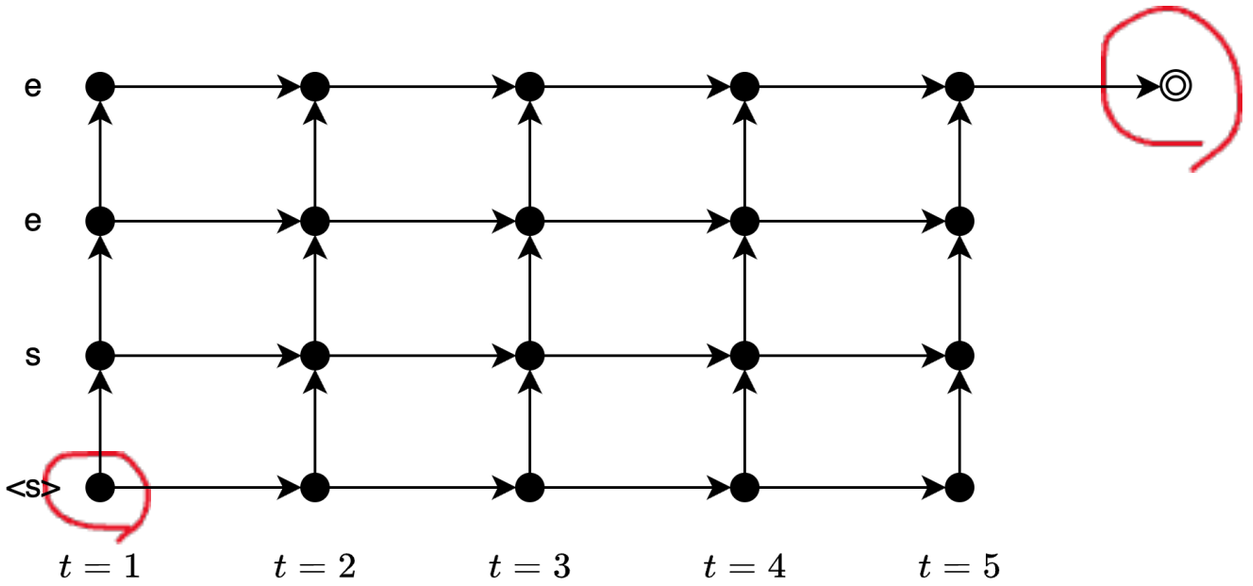
$$p(O \mid W)$$



"I want to **go to** the campus"

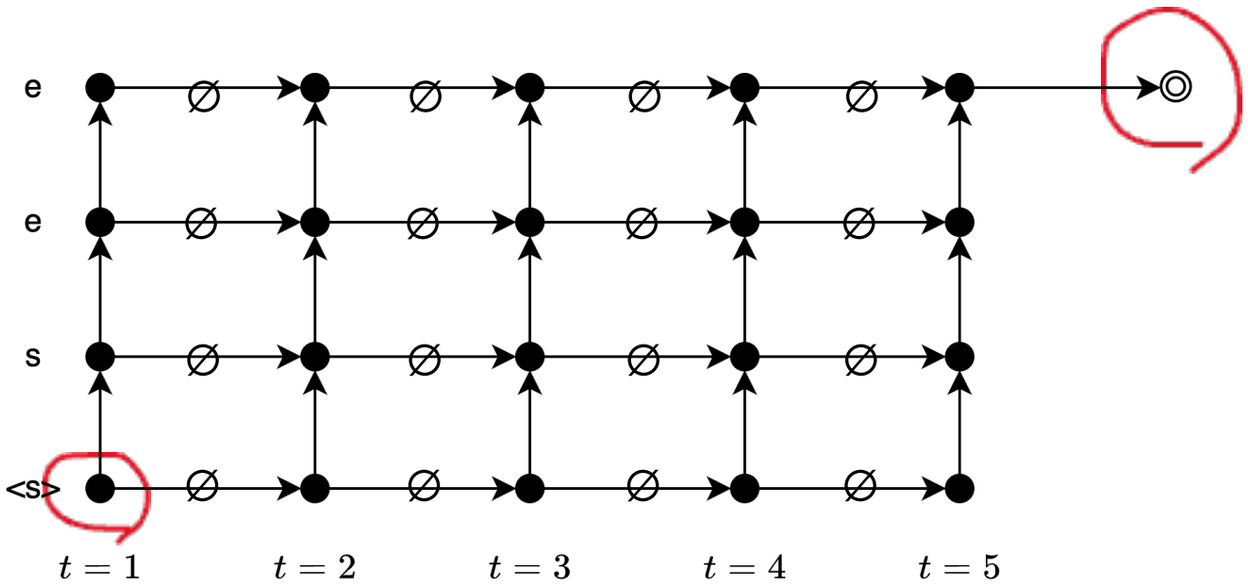
Alignment path in the RNN-transducer

- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (o_1, \ldots, o_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: **left bottom corner**, i.e., <s> (start of sentence) at $t = 1$
  - End: **top right corner**
  - ↑: output token (it does not consume the input frame, unlike CTC)!



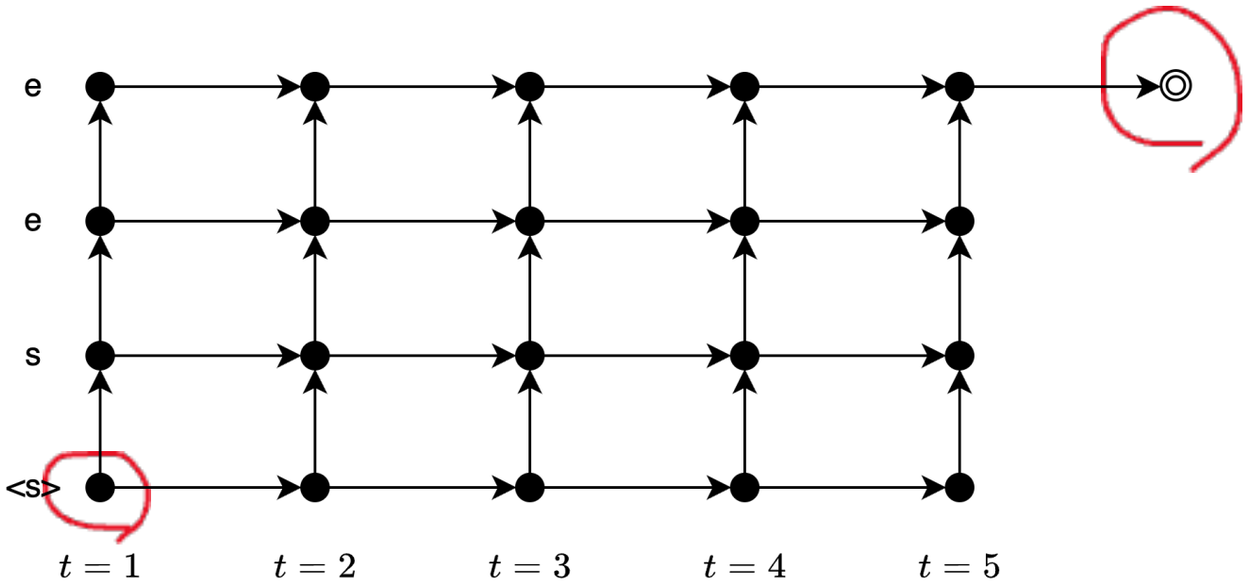
Alignment path in the RNN-transducer

- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (o_1, \ldots, o_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: **left bottom corner**, i.e., <s> (start of sentence) at $t = 1$
  - End: **top right corner**
  - ↑: output token (it does not consume the input frame, unlike CTC)!
  - →: no output (∅)



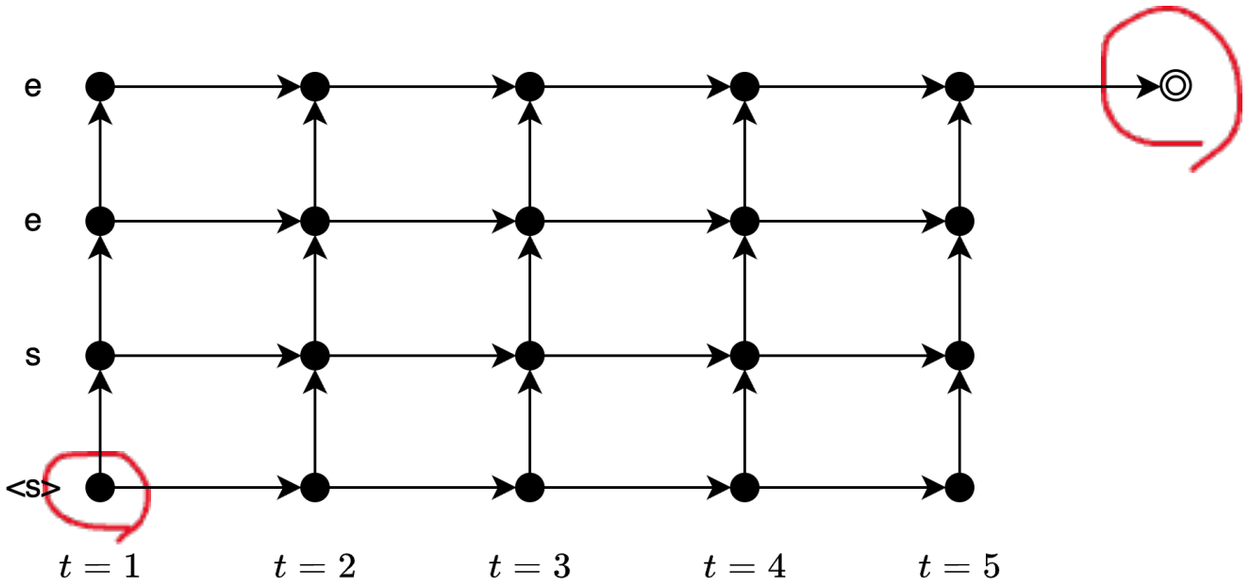
Alignment path in the RNN-transducer

- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (o_1, \ldots, o_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: **left bottom corner**, i.e., <s> (start of sentence) at $t = 1$
  - End: **top right corner**
  - ↑: output token (it does not consume the input frame, unlike CTC)!
  - →: no output (∅)
  - f^{-1}: W=('s', 'e', 'e') → Z=(∅, 's', ∅, ∅, 'e', 'e', ∅, ∅), ('s', ∅, 'e', ∅, ∅, ∅, 'e', ∅) etc. if $T = 5$ and $J = 3$
  - $f: Z \rightarrow W$ is a many to one mapping (we can just remove ∅
- Consider all possible paths including the case that it allows the output of the token W



Alignment path in the RNN-transducer

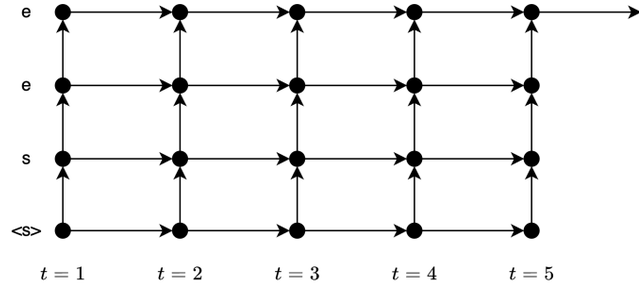
- Alternative alignment paths
  - $W = (w_1, \ldots, w_J)$: J-length label sequence
  - $O = (o_1, \ldots, o_T)$: T-length input sequence
- Similar to CTC, consider all possible paths in the right-side trellis
  - Start: **left bottom corner**, i.e., <s> (start of sentence) at $t = 1$
  - End: **top right corner**
  - ↑: output token (it does not consume the input frame, unlike CTC)!
  - →: no output (∅)
  - f^{-1}: W=('s', 'e', 'e') → Z=(∅, 's', ∅, ∅, 'e', 'e', ∅, ∅), ('s', ∅, 'e', ∅, ∅, ∅, 'e', ∅) etc. if $T = 5$ and $J = 3$
  - $f: Z \rightarrow W$ is a many to one mapping (we can just remove ∅
- Consider all possible paths including the case that it allows the output of the token W



How to represent it as an actual function?

- We define an alignment variable as $Z = (z_k \in \mathscr{V} \cup \varnothing \,|\, k = 1,\ldots,T+J)$
  - Vocabulary set is augmented by the blank symbol $\varnothing$
  - Note that $|Z| = T+J$, not T
- We can introduce the following equations

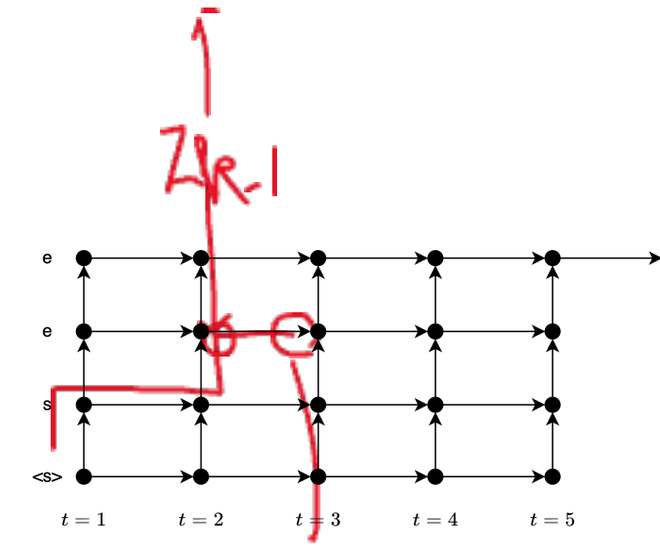
$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k | z_{1:k-1}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k | f(z_{1:k-1}), O)
\end{aligned}
$$



- $f^{-1}(W)$: all possible Z representing W
- Use a chain rule to further factorize Z
- $f(z_{1:k-1})$ represents the partial token sequence up to $k-1$

How to represent it as an actual function?

$f(z_{1:k-1}) =$ s, e,



- We define an alignment variable as $Z = (z_k \in \mathcal{V} \cup \varnothing \,|\, k = 1,\ldots,T+J)$
  - Vocabulary set is augmented by the blank symbol $\varnothing$
  - Note that $|Z| = T+J$, not T
- We can introduce the following equations

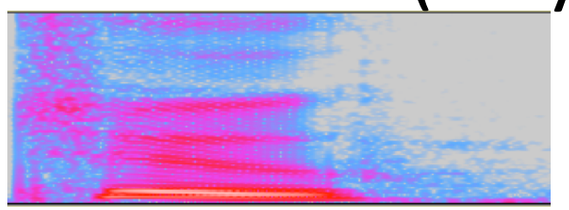
$$
\begin{aligned}
p(W|O) &= \sum_{Z \in f^{-1}(W)} p(Z|O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|z_{1:k-1}, O) \\
&= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|f(z_{1:k-1}), O)
\end{aligned}
$$

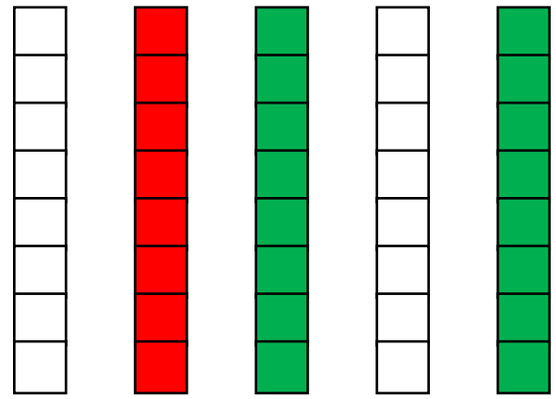
- $f^{-1}(W)$: all possible Z representing W
- Use a chain rule to further factorize Z
- $f(z_{1:k-1})$ represents the partial token sequence up to $k-1$
- We need to solve the **summuzation over all possible paths** $\sum_{Z}$ ← Exponential

- $p(z_k \,|\, f(z_{1:k-1}), O)$ is represented by a special neural network that has the information of both t and i

Example of Z in CTC

“see” (only 5 frames)





- $W = (\text{“s”}, \text{“e”}, \text{“e”}),\ T = 5$

Then

$Z = (\text{“<b>”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”}),$ or

$Z = (\text{“s”}, \text{“<b>”}, \text{“e”}, \text{“<b>”}, \text{“e”})$, or

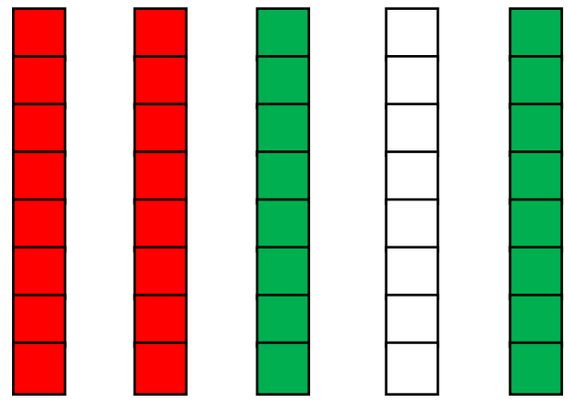
$Z = (\text{“s”}, \text{“s”}, \text{“e”}, \text{“<b>”}, \text{“e”})$, or

….

This is an alignment problem

<b> s e <b> e

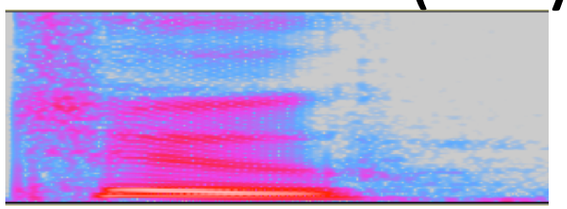
or

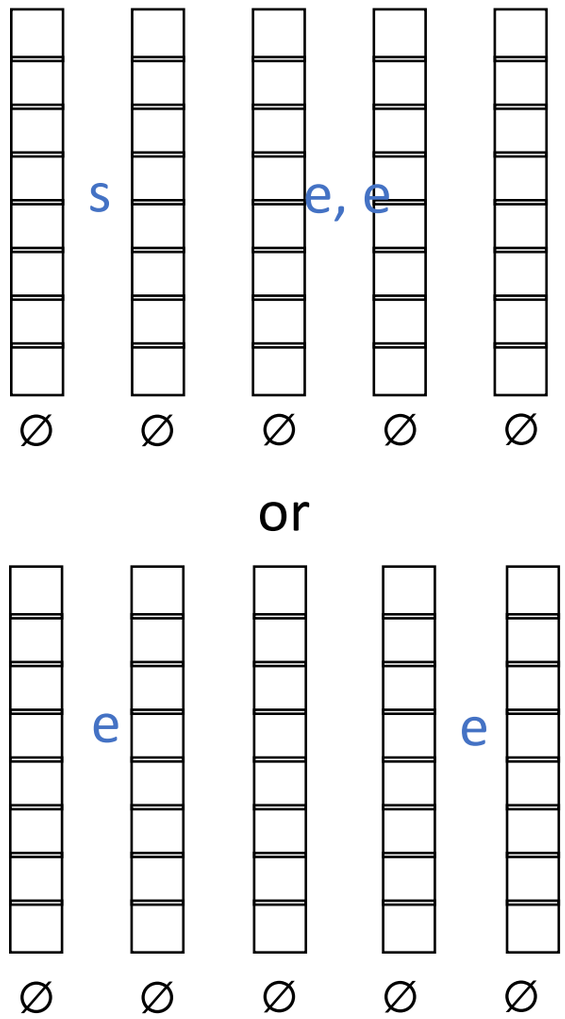


s s e <b> e

Example of Z in RNN-trans.

“see” (only 5 frames)





- $W = (\text{“s”}, \text{“e”}, \text{“e”}),\ T = 5$

Then

$Z = (\varnothing, \text{‘s’}, \varnothing, \varnothing, \text{‘e’}, \text{‘e’}, \varnothing, \varnothing),$ or

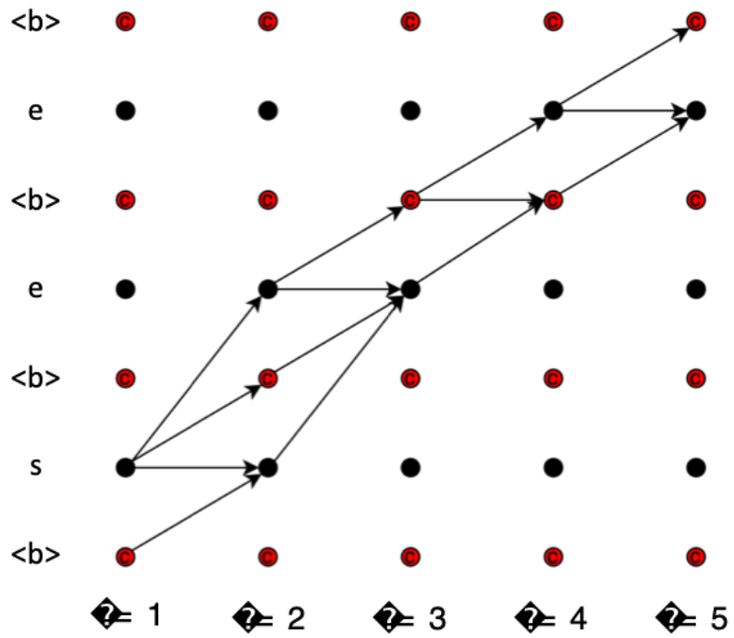
$Z = (\text{‘s’}, \varnothing, \text{‘e’}, \varnothing, \varnothing, \varnothing, \text{‘e’}, \varnothing),$or

….

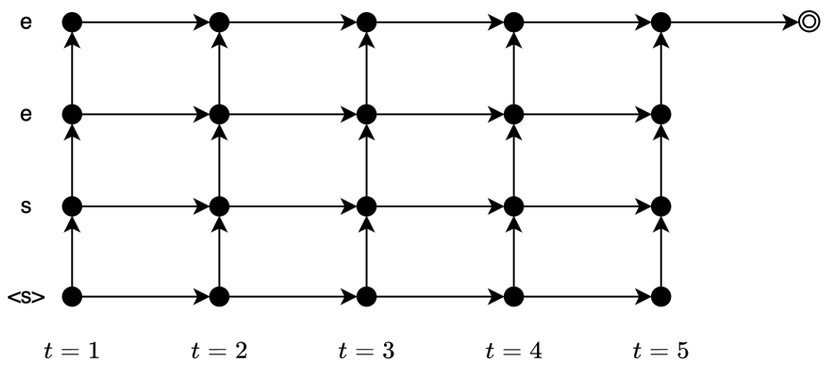
This is an alignment problem

CTC vs. RNN-T

- CTC



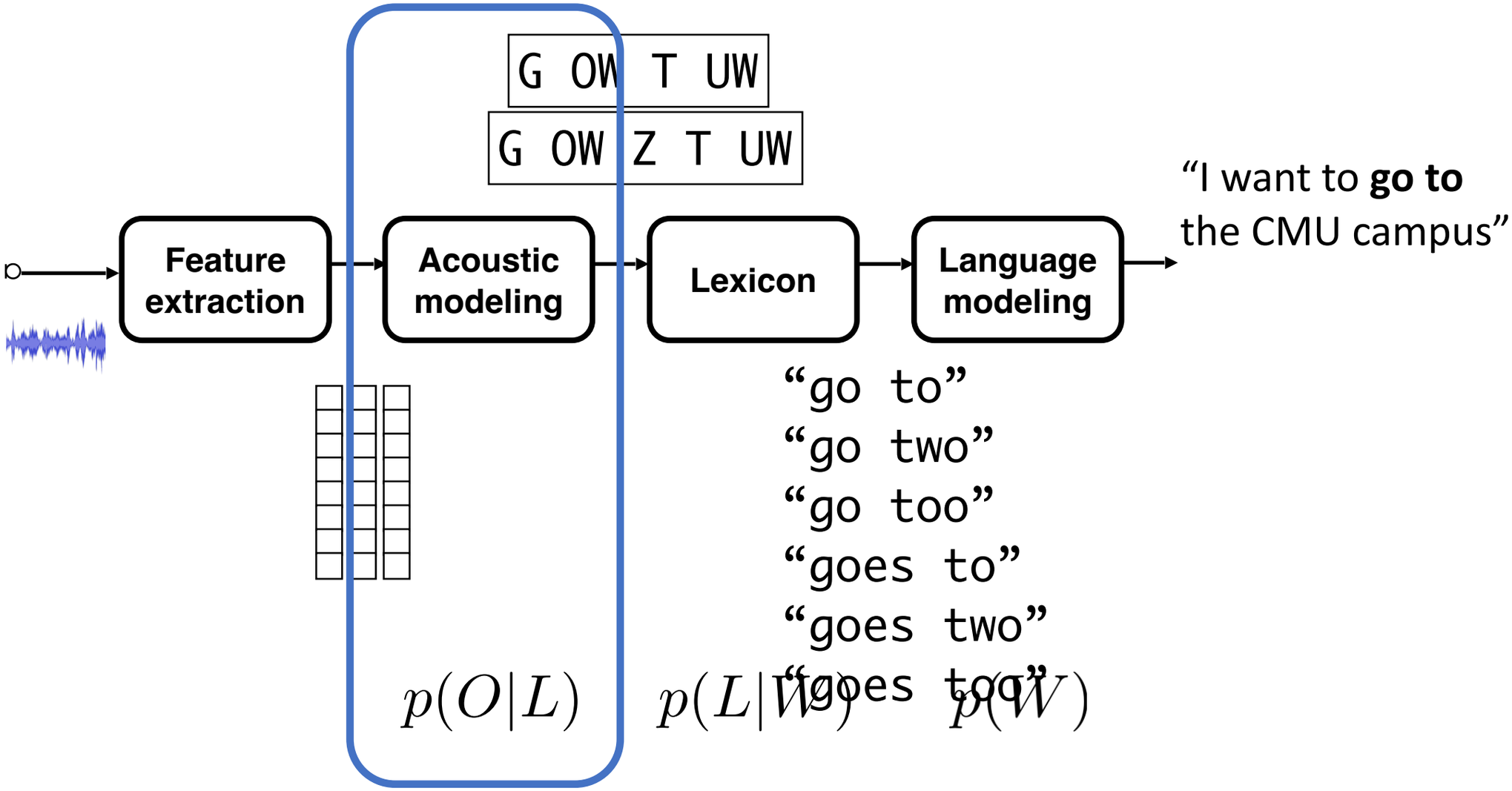
- RNN-transducer



Agenda

- Alignment paths
  - CTC
  - RNN-Transducer
  - HMM (3-state left-to-right)

Speech recognition pipeline



Introduction of HMM states

- Two differences
  - Phoneme sequence L instead of word sequence W
  - $p(O \mid L)$ instead of $p(L \mid O)$ or $p(W \mid O)$ due to the Bayes theorem

1. Decompose the word sequence to the phoneme sequence by using a pronunciation dictionary

   “one,” “two” → /W/, /AH/, /N/, /T/, /UW/

2. Introduce the silence related symbols
   - Silence begin (/SilB/): placed in the beginning of the sentence
   - Silence end (/SilE/): placed in the end of the sentence
   - Short pause (/SP/): placed between the words (can be skipped)

     /W/, /AH/, /N/, /T/, /UW/ → /SilB/, /W/, /AH/, /N/, /SP/, /T/, /UW/, /SilE/

3. Expand each phoneme with **three** states

   /SilB/, /W/, /AH/, /N/, /SP/, /T/, /UW/, /SilE/

   → $/\text{SilB}/_1$, $/\text{SilB}/_2$, $/\text{SilB}/_3$, …, $/\text{T}/_1$, $/\text{T}/_2$, $/\text{T}/_3$, $/\text{UW}/_1$, $/\text{UW}/_2$, $/\text{UW}/_3$, …

   - This structure is widely used, but there are some variants (e.g., short pause can be one state)

HMM-based (Classical)

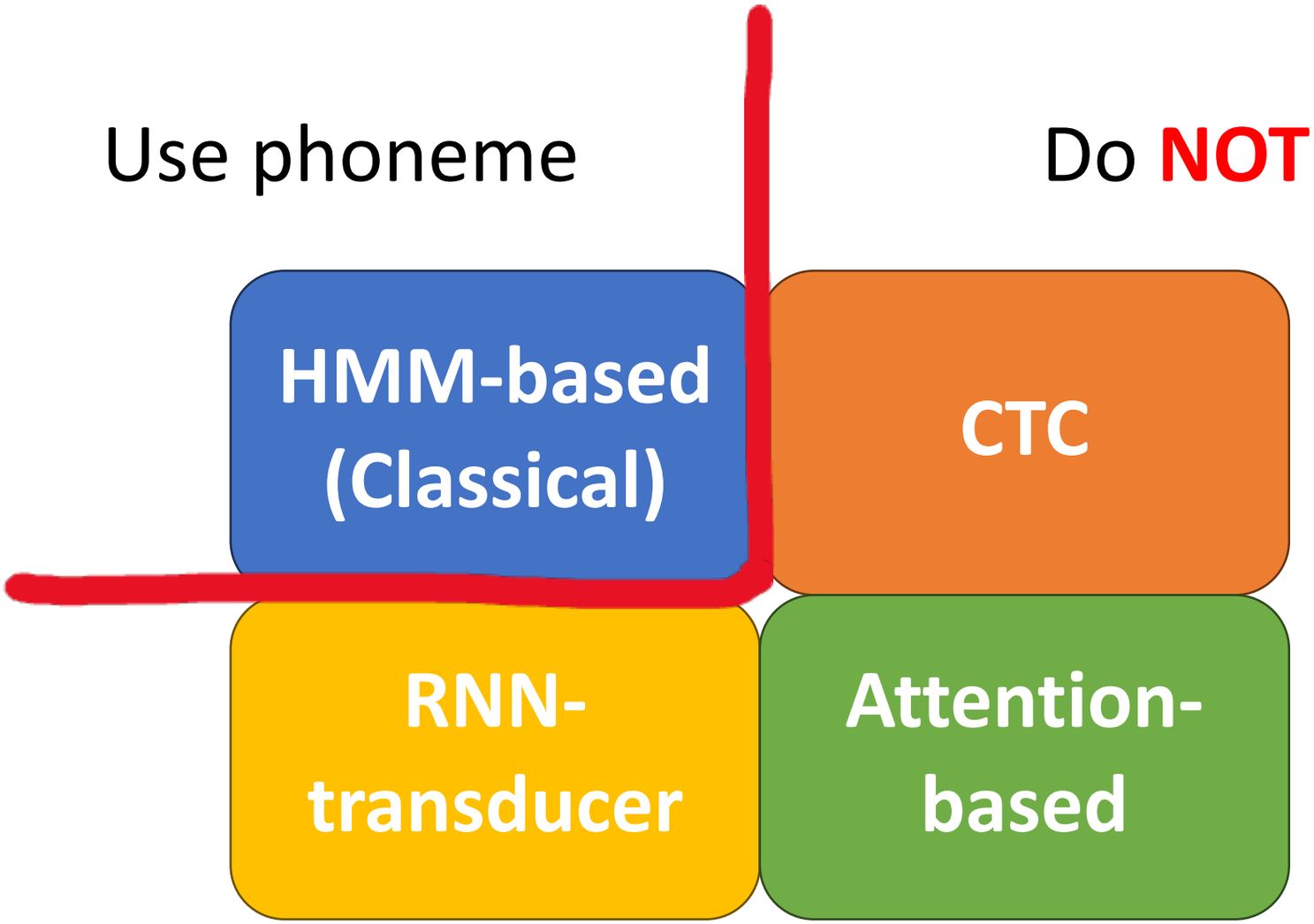
CTC

RNN-transducer

Attention-based

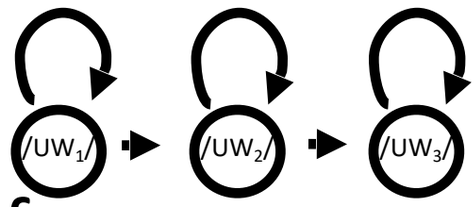
Use phoneme

Do **NOT** use phoneme

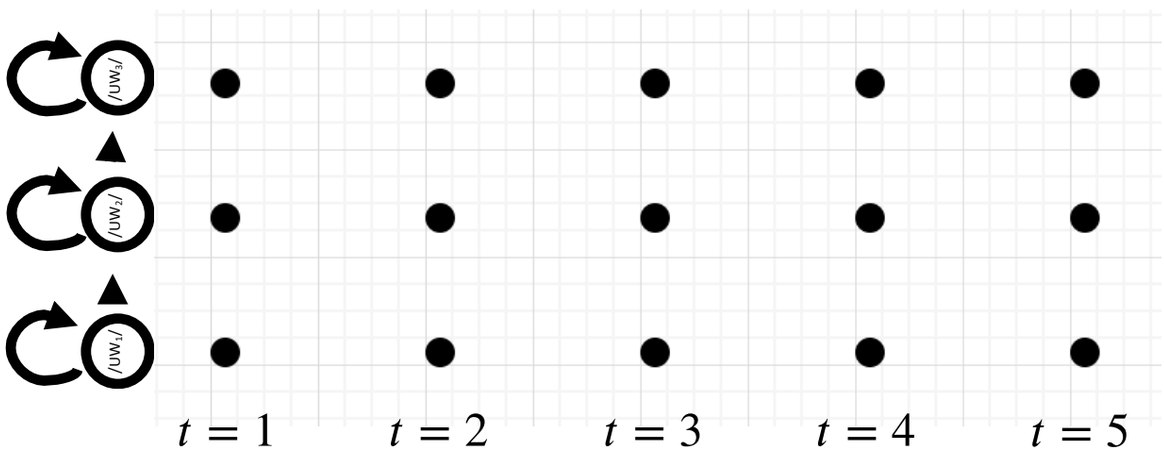


Trellis to find all possible paths

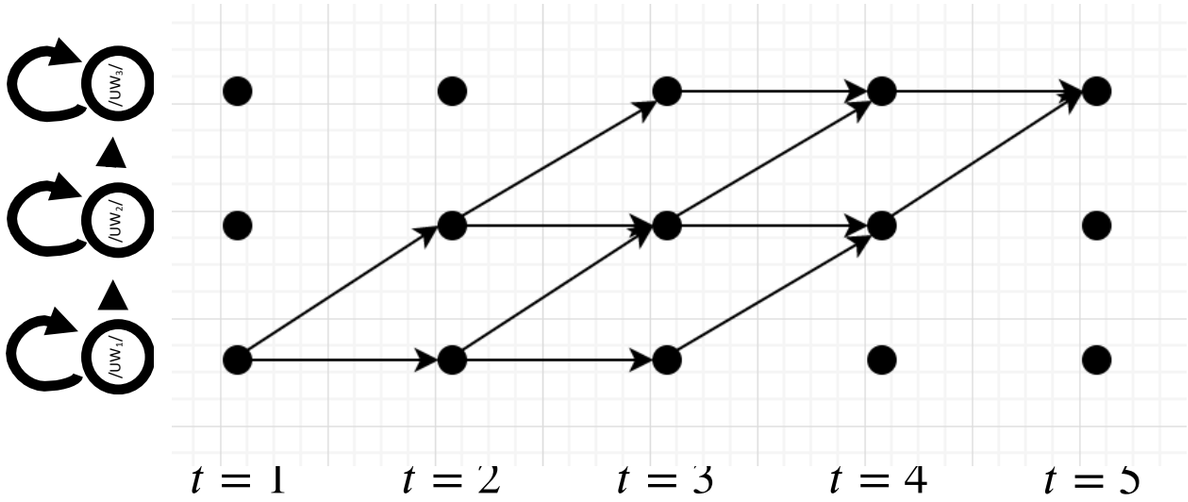
- Left-to-right HMM without skip



  - Each state must have at least one frame
- Example, 3 state left-to-right HMM (/UW_1/, /UW_2/, /UW_3/) $S = (s_t \in \{/\mathrm{UW}_1/, /\mathrm{UW}_2/, /\mathrm{UW}_3/\} | t = 1, 2, 3, 4, 5)$ quences



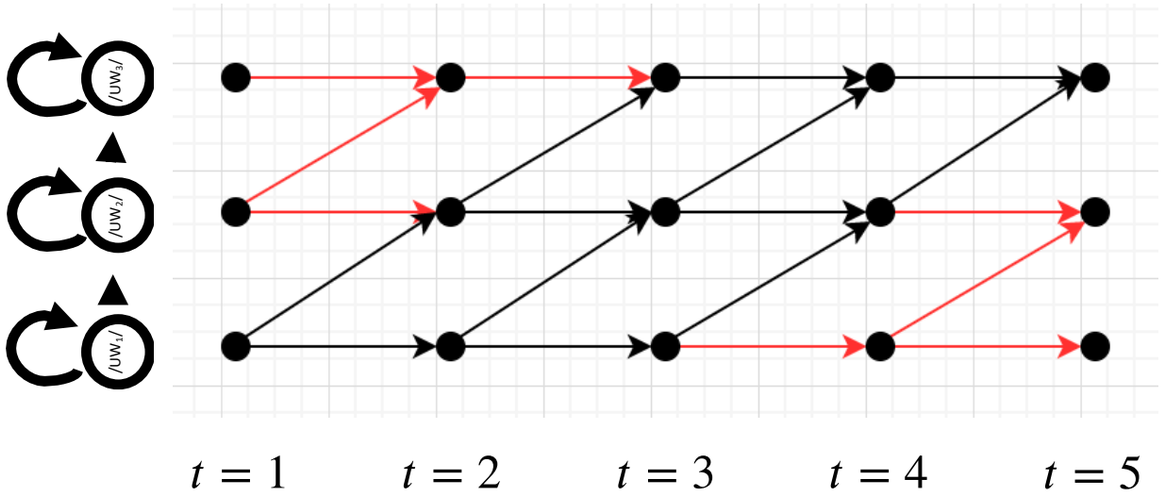
Trellis to find all possible paths



- It must start at the initial state (/UW_1/)
- It must end at the final state (/UW_3/)

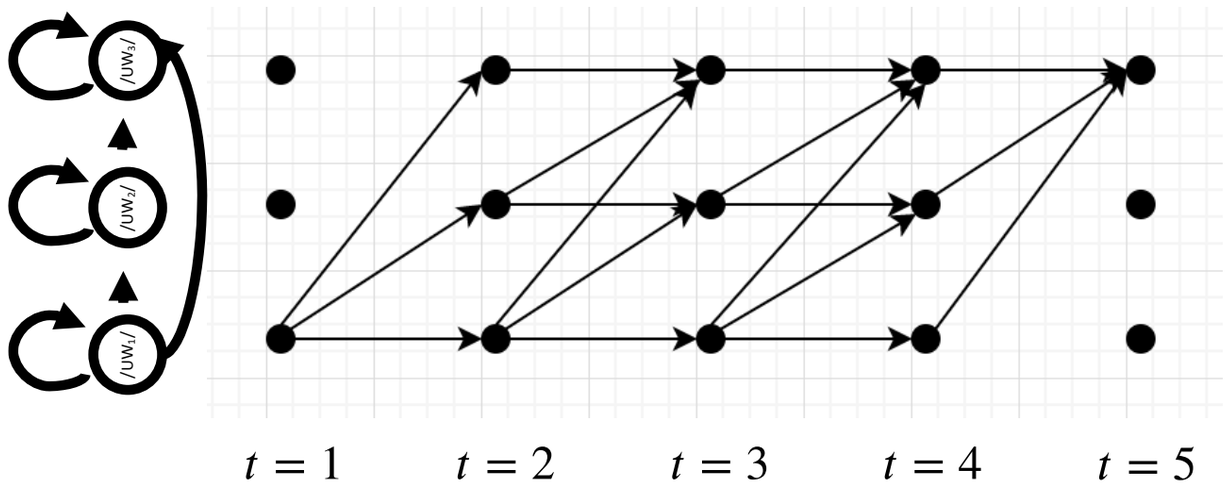
Trellis to find all possible paths

- Red arrows cannot reach the final state



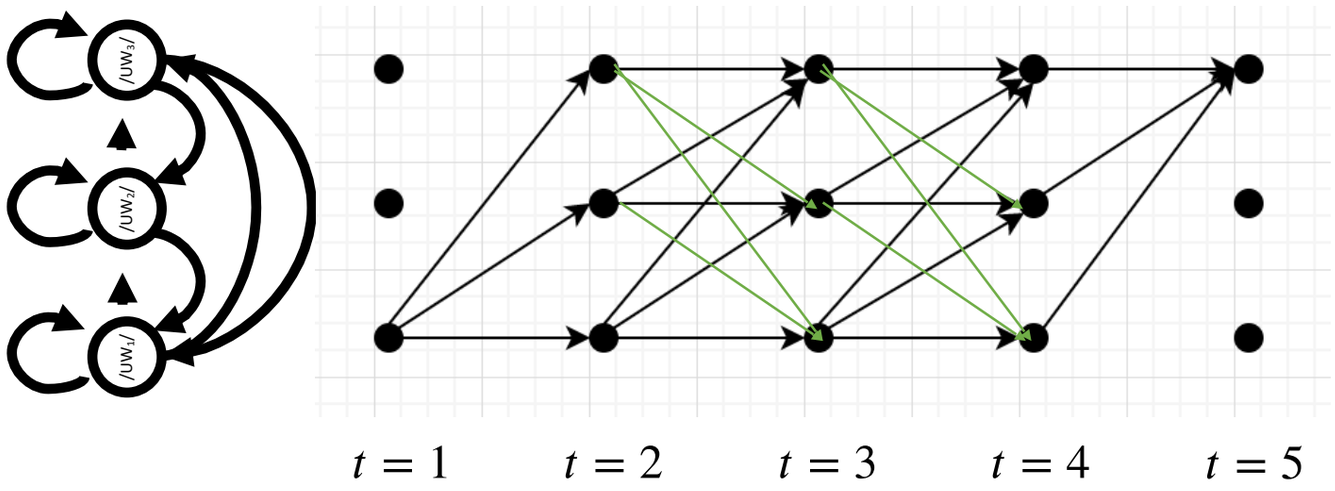
Trellis to find all possible paths

- If we have some skips…



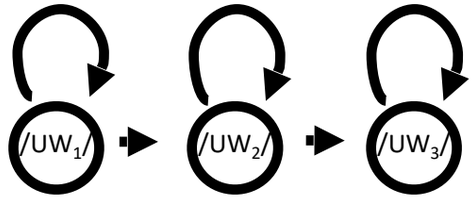
Trellis to find all possible paths

- If we have some skips…

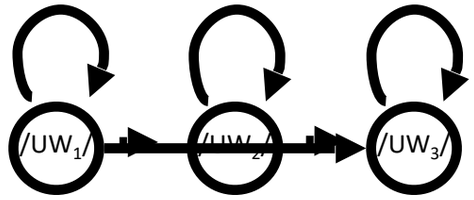


Variant of HMMs

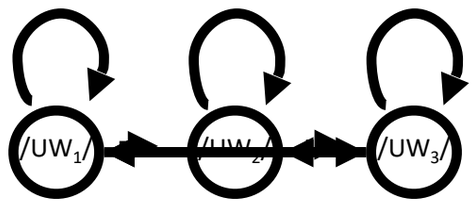
- Left to right
  - We mainly use this



- Left to right with skip
  - 2nd order in this example

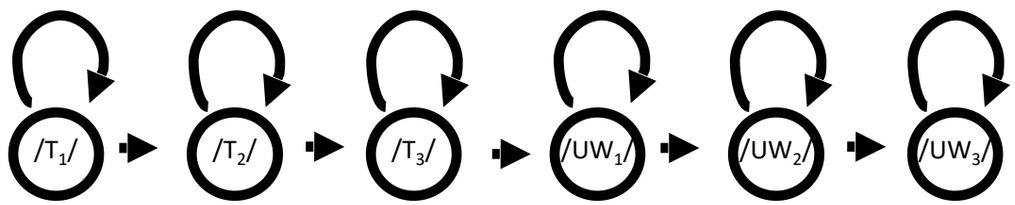


- Ergodic (fully connected)
  - No specific order anymore…



Phoneme sequence

- We just connect the HMMs for each phoneme



Let’s check three distributions

- CTC

$$p(W|O) = \sum_{Z \in f^{-1}(W)} p(Z|O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|\cancel{z_{1:t-1}}, O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|O)$$

- RNN transducer

$$p(W|O) = \sum_{Z \in f^{-1}(W)} p(Z|O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|z_{1:k-1}, O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|f(z_{1:k-1}), O)$$

- HMM

$$p(W|O) \propto \sum_{L} p(O|L)p(L|W)p(W)$$
$$= \sum_{L} \sum_{S} p(O, S|L)p(L|W)p(W)$$
$$= \sum_{L} \sum_{S} p(\mathbf{o}_1|s_1, L)p(s_1|L) \prod_{t=2}^{T} (\mathbf{o}_t|s_t, L)p(s_t|s_{t-1}, L)p(L|W)p(W)$$

Which part is similar?

Let’s check three distributions

- CTC

$$p(W|O) = \sum_{Z \in f^{-1}(W)} p(Z|O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|\cancel{z_{1:t-1}}, O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{t=1}^{T} p(z_t|O)$$

- RNN transducer

$$p(W|O) = \sum_{Z \in f^{-1}(W)} p(Z|O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|z_{1:k-1}, O)$$
$$= \sum_{Z \in f^{-1}(W)} \prod_{k=1}^{T+J} p(z_k|f(z_{1:k-1}), O)$$

- HMM

$$p(W|O) \propto \sum_{L} p(O|L)p(L|W)p(W)$$
$$= \sum_{L} \sum_{S} p(O, S|L)p(L|W)p(W)$$
$$= \sum_{L} \sum_{S} p(\mathbf{o}_1|s_1, L)p(s_1|L) \prod_{t=2}^{T} (\mathbf{o}_t|s_t, L)p(s_t|s_{t-1}, L)p(L|W)p(W)$$

Which part is similar?

→ All are based on $\sum_{Z \text{ or } S} \prod_{t} f(t)$

→ We can compute them based on the dynamic programming

Summary

- CTC, RNN-transducer, and HMM are represented by a similar function form, i.e., $\sum_{Z \text{ or } S} \prod_{t} f(t)$
- All possible alignments are constrained by each method
- The most difficult issue in speech recognition: the input and output lengths are different
  - We need some alignments
- The soft alignment case
  - We use an attention-based neural network (this will be introduced later).
- The hard alignment cases
  - We explicitly introduce an alignment path z.
  - We can use it for CTC, RNN-transducer, and HMM-based approach