Lecture 16: Natural Language Generation

Instructor: Swabha Swayamdipta

USC CSCI 499 LMs in NLP

Mar 27, Spring 2024

Slides adapted from Chris Manning, Xiang Lisa Li

Logistics / Announcements

Logistics / Announcements

- HW4 due on Wed, 3rd April
- Project discussions on 8th April
- Next Monday: Quiz 5

Outline

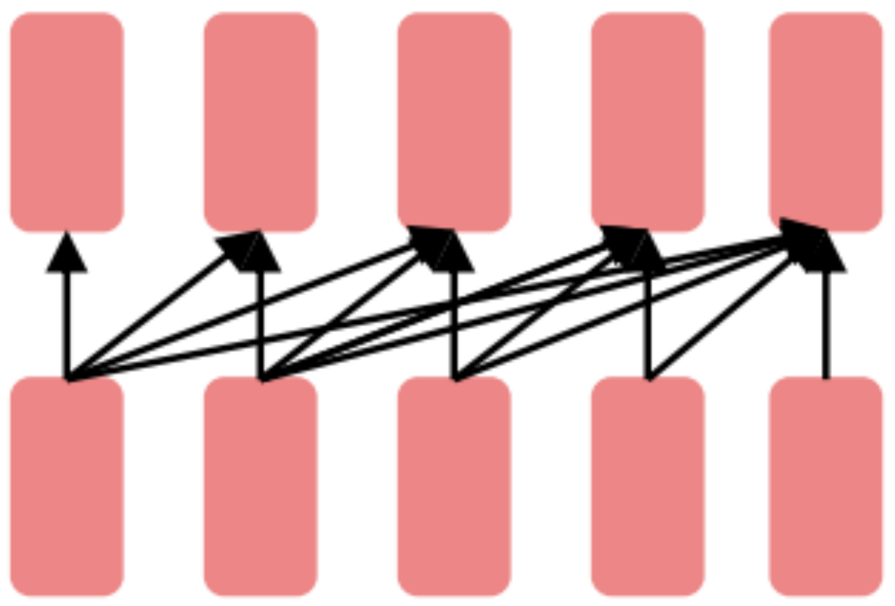
- Recap: Pretraining with Encoder-Decoder Models
- Recap: Tokenization in Transformers
- Natural Language Generation
- Classic Inference Algorithms: Greedy and Beam Search

Recap: Pretraining with Encoder-Decoder Models

Recap:
Transformer Encoder-Decoders

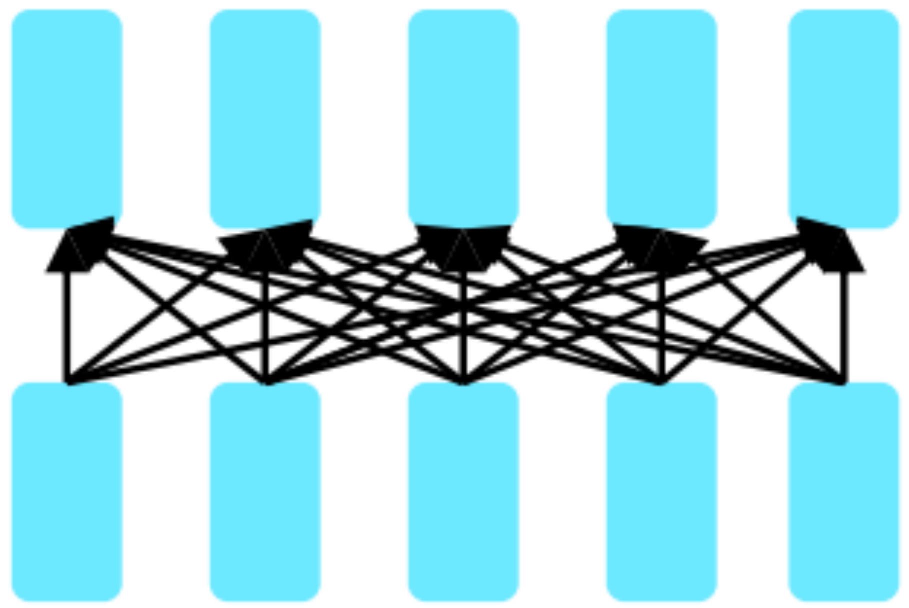
Pretraining

- Not restricted to language modeling!
  - Can be any task.
  - But most successful if the task definition is very general
  - Hence, language modeling is a great pretraining option
- Three options!



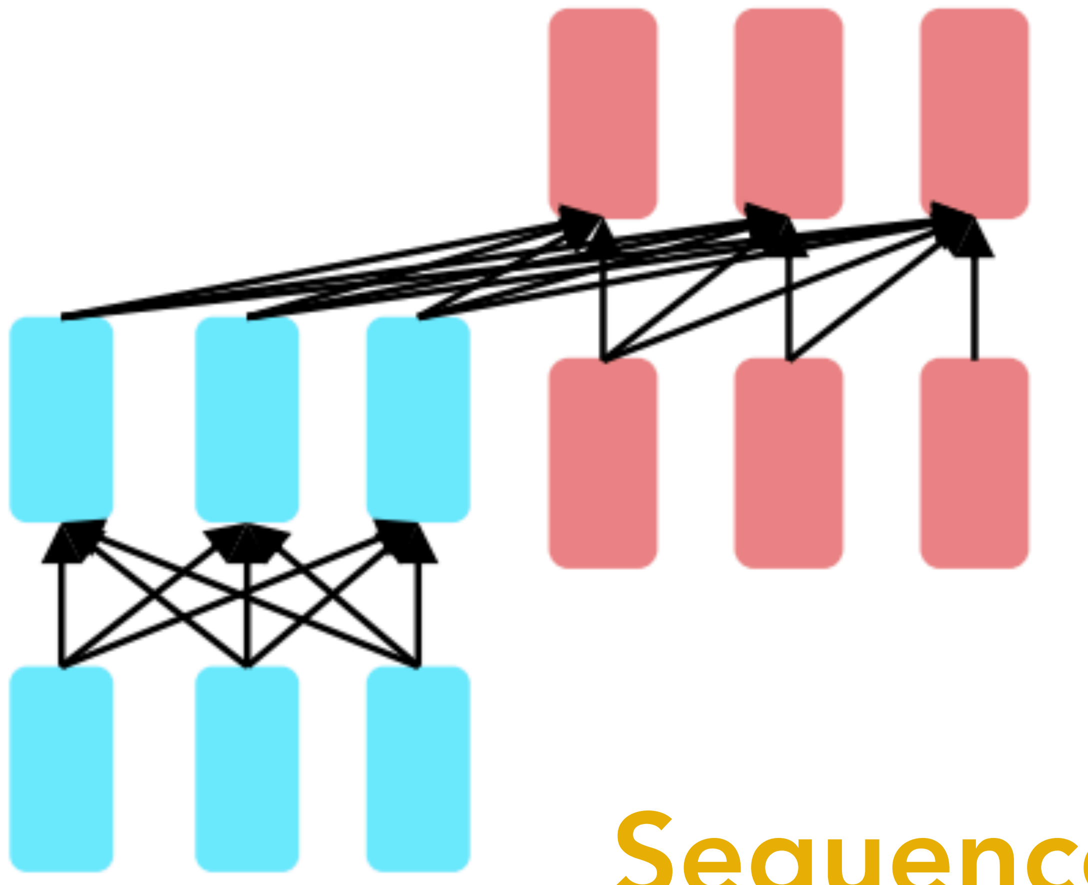
Decoders

Language Models



Encoders

Token / Sequence Classification



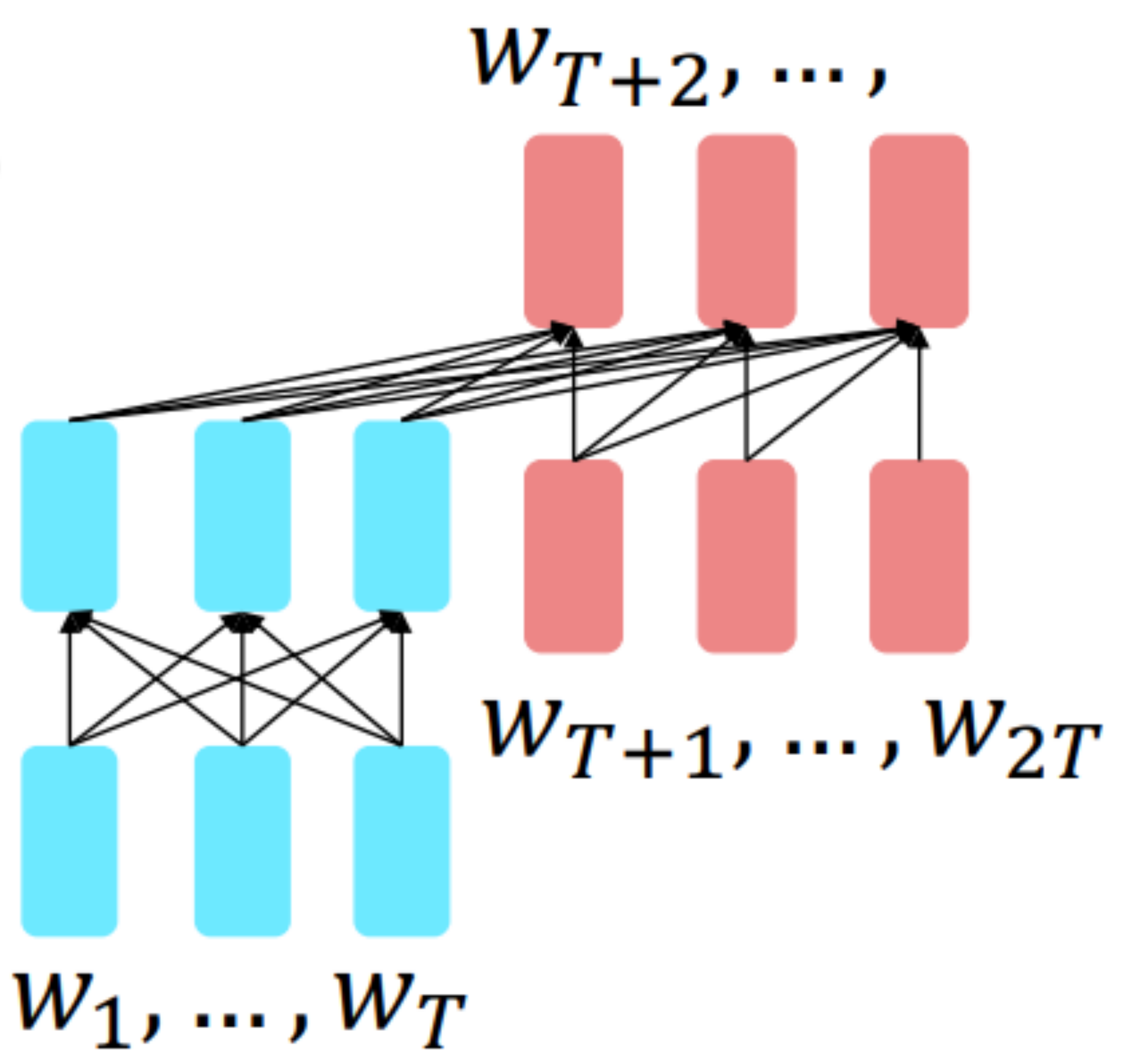
Encoder-Decoders

Sequence-to-sequence

Pretraining Encoder-Decoder Models

- For encoder-decoders, we could do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$



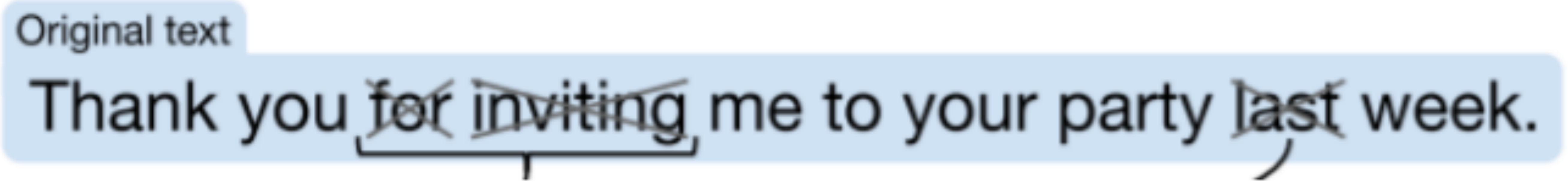
The encoder portion benefits from bidirectional context; the decoder portion is used to train the whole model through language modeling.

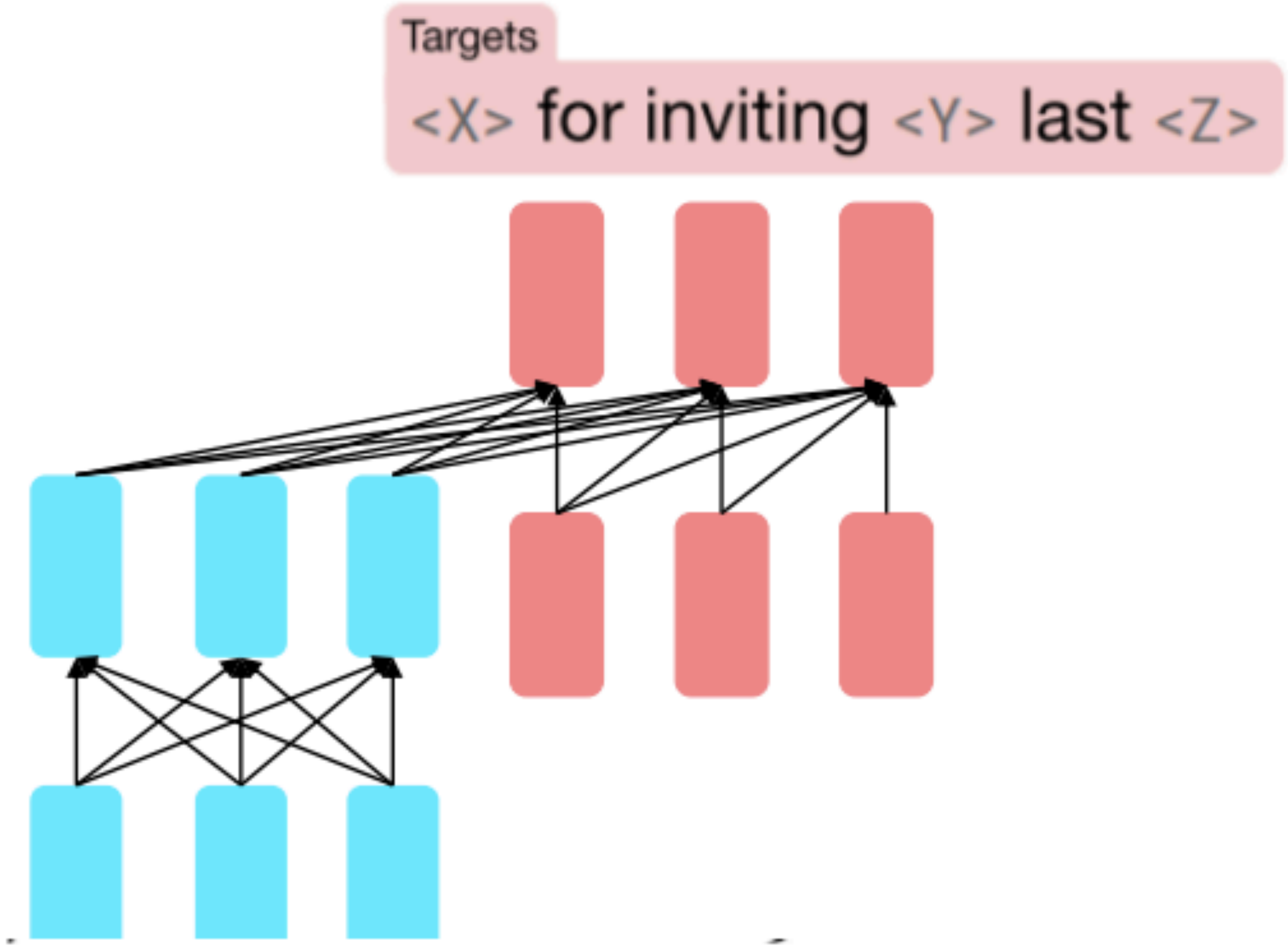
USC Viterbi

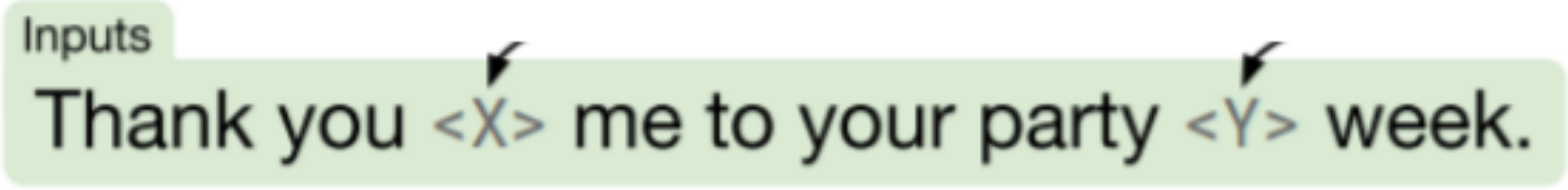
T5: A Pretrained Encoder-Decoder Model

- Raffel et al., 2018 built T5, which uses as a span corruption pretraining objective

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



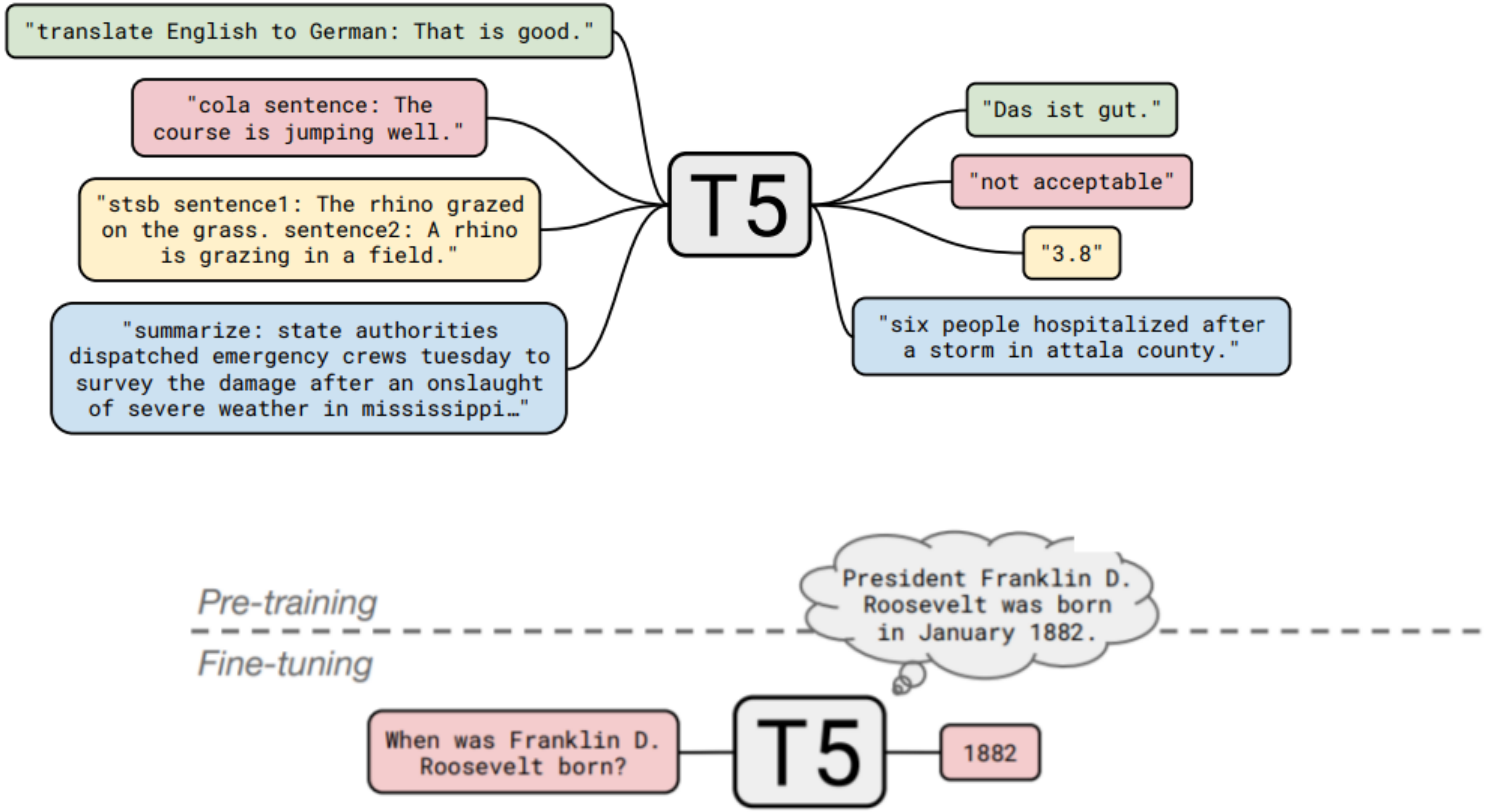




Still uses an objective that looks like language modeling at the decoder side.

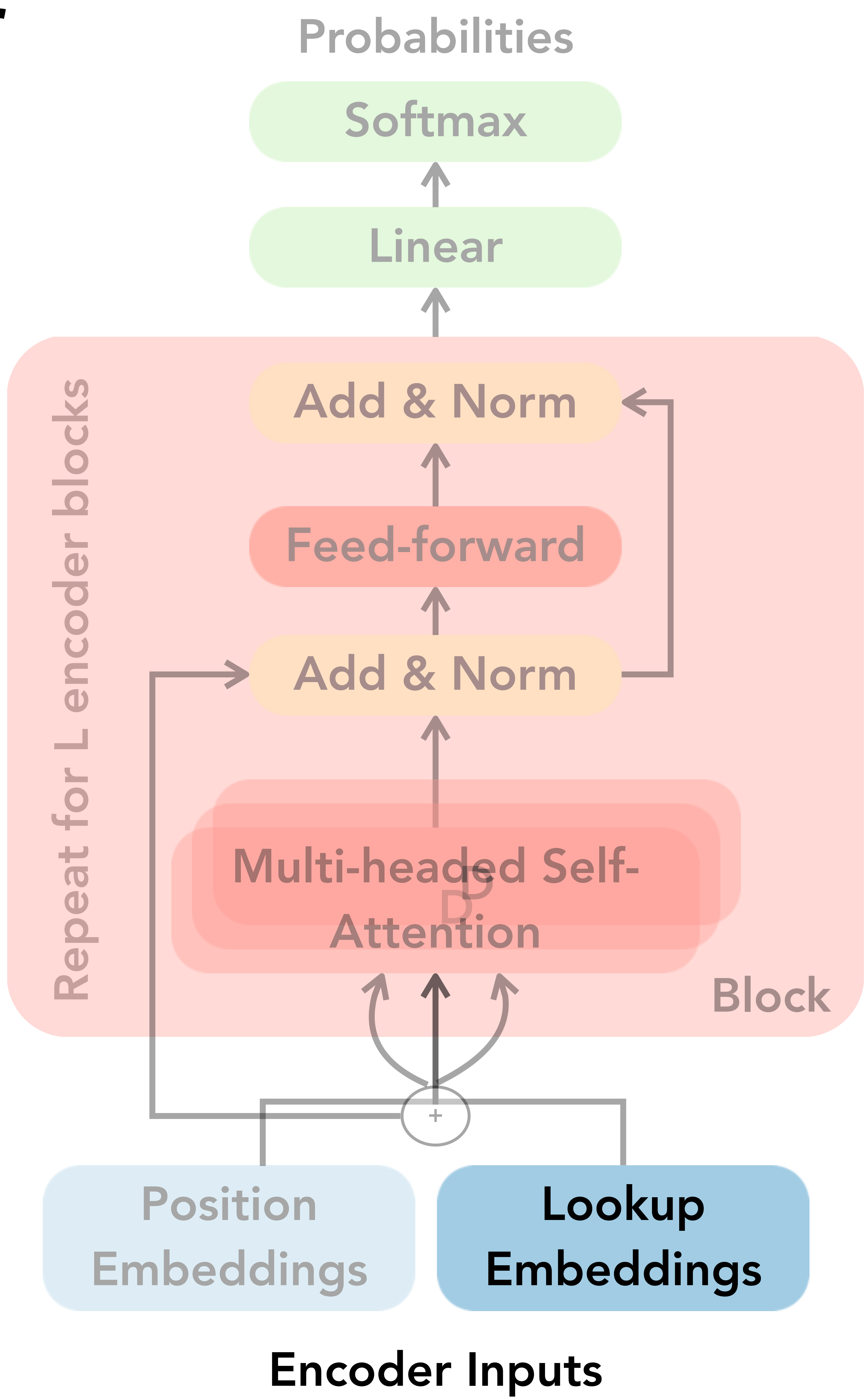
T5: Task Preparation

T5 can be finetuned to answer a wide range of tasks, where the input and output are expressed as a sequence of tokens



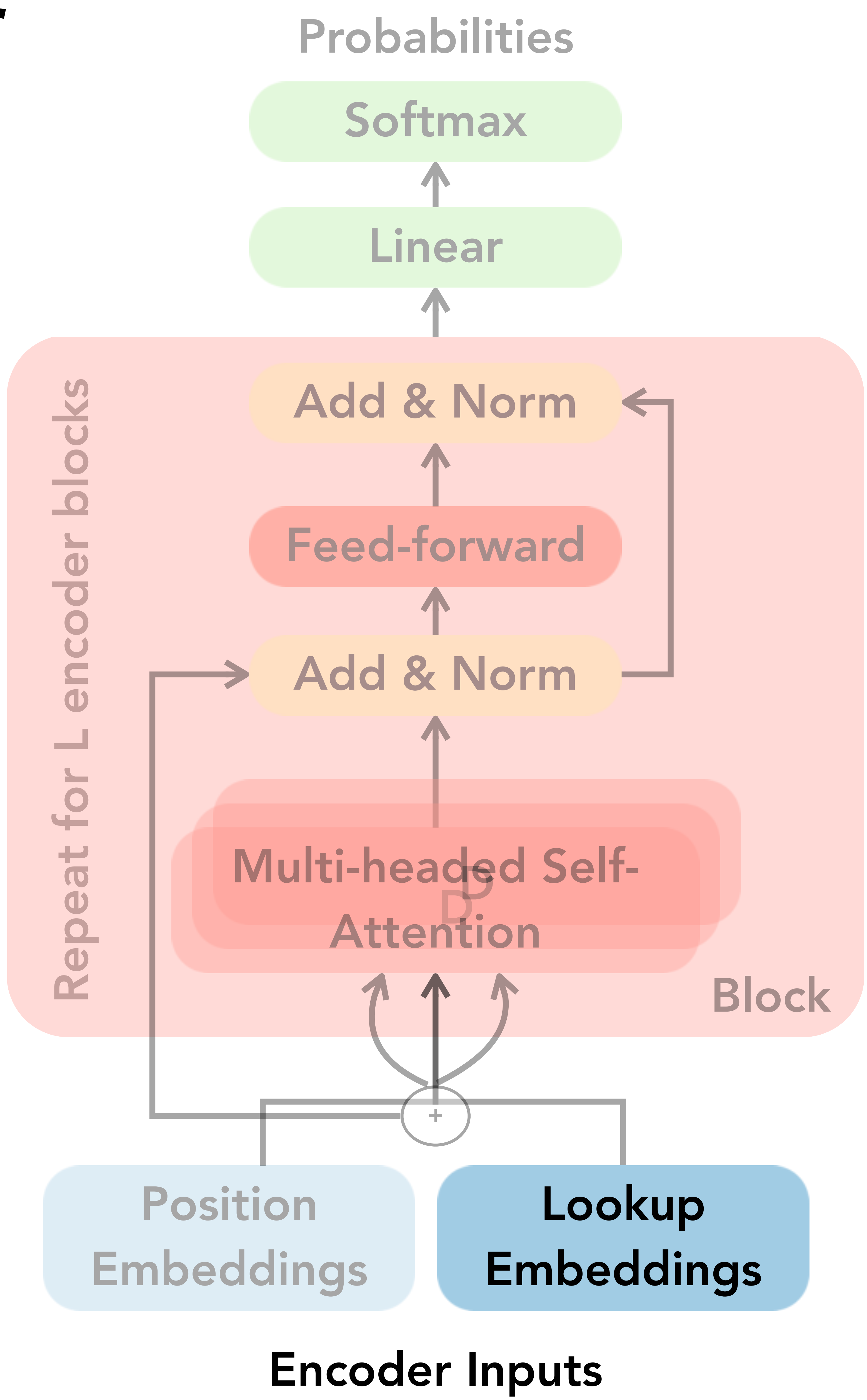
Recap: Tokenization in Transformers

The Input Layer



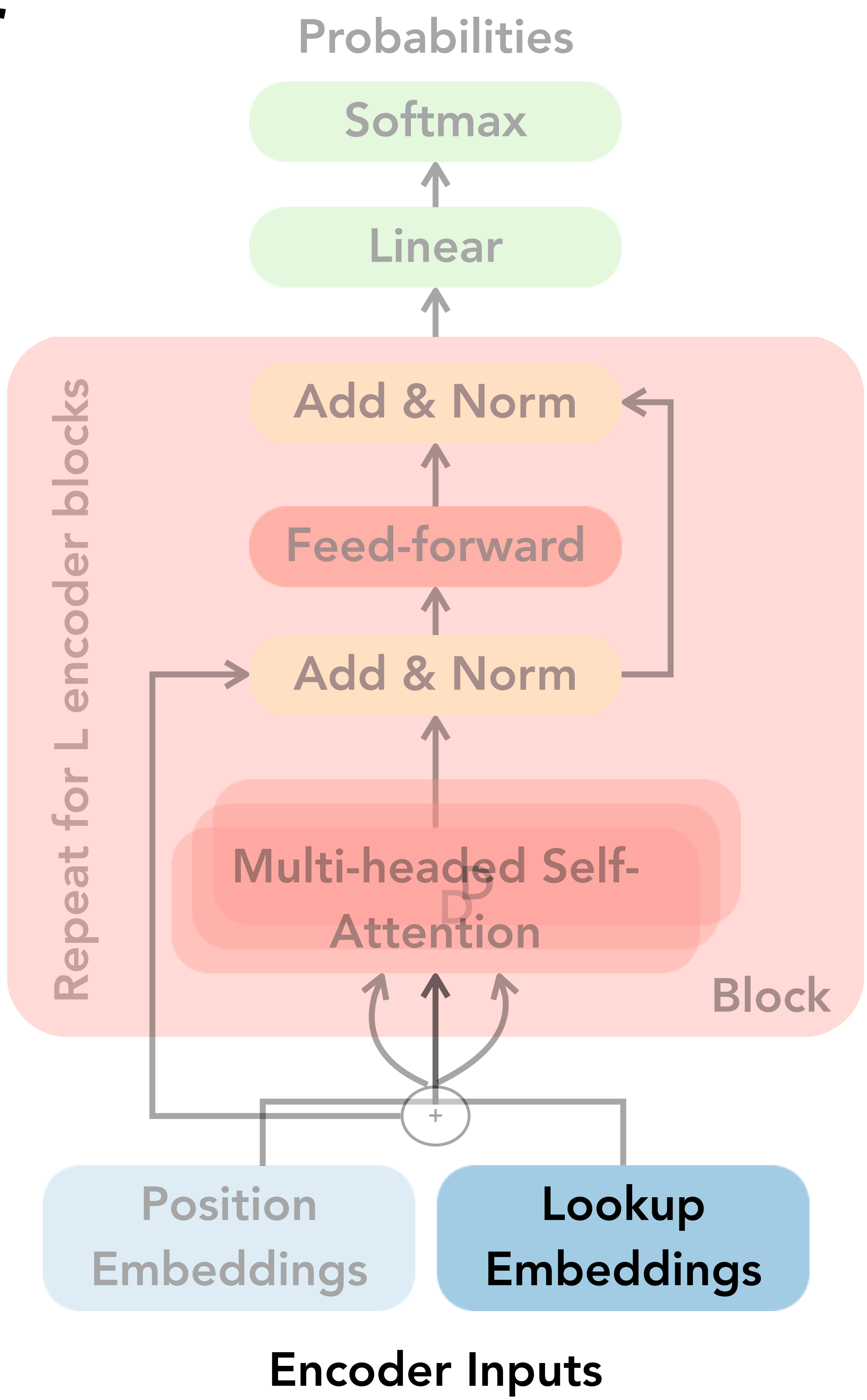
The Input Layer

- So far, we have made some assumptions about a language's vocabulary



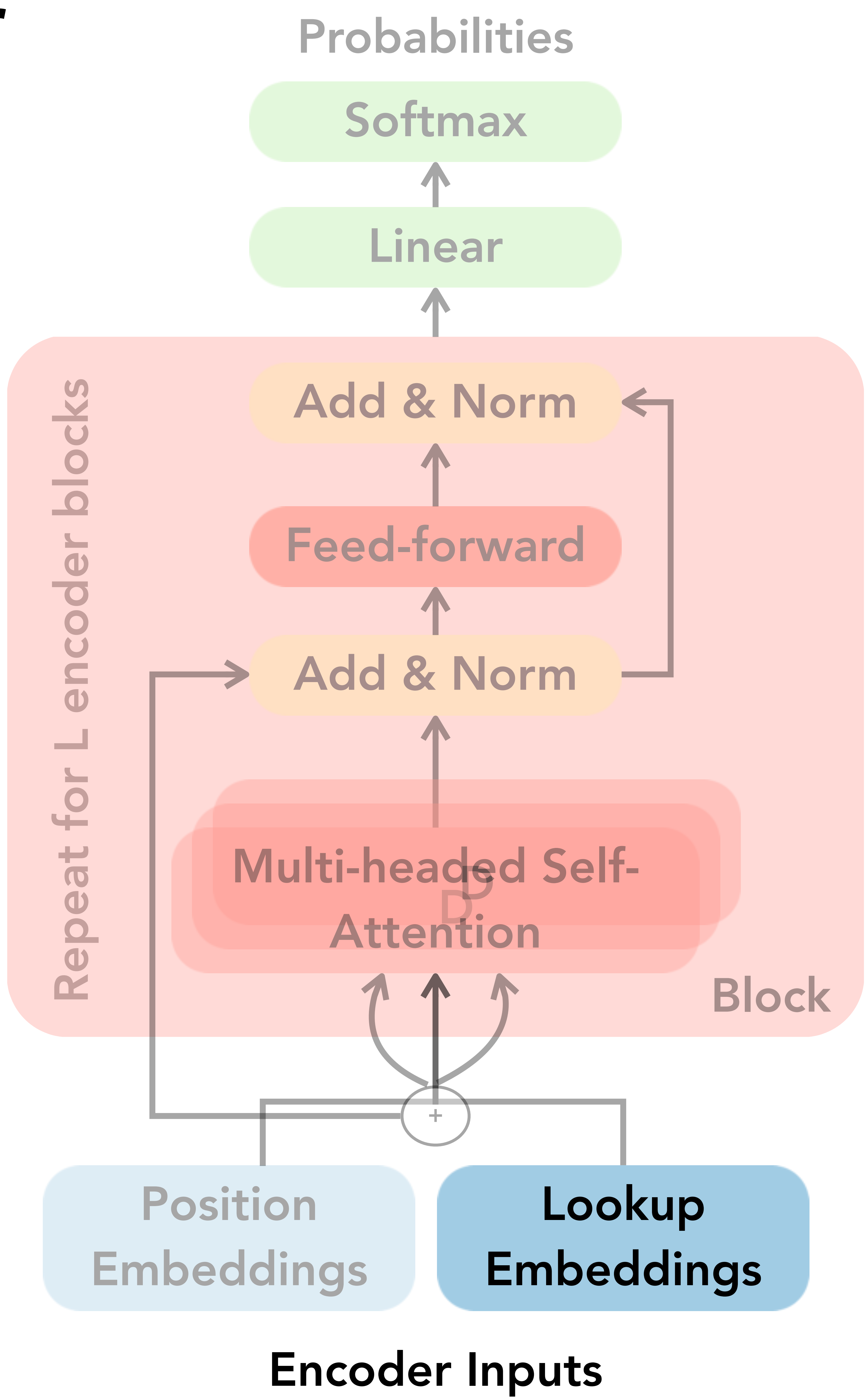
The Input Layer

- So far, we have made some assumptions about a language's vocabulary
- Our approach so far: use a known, fixed vocabulary



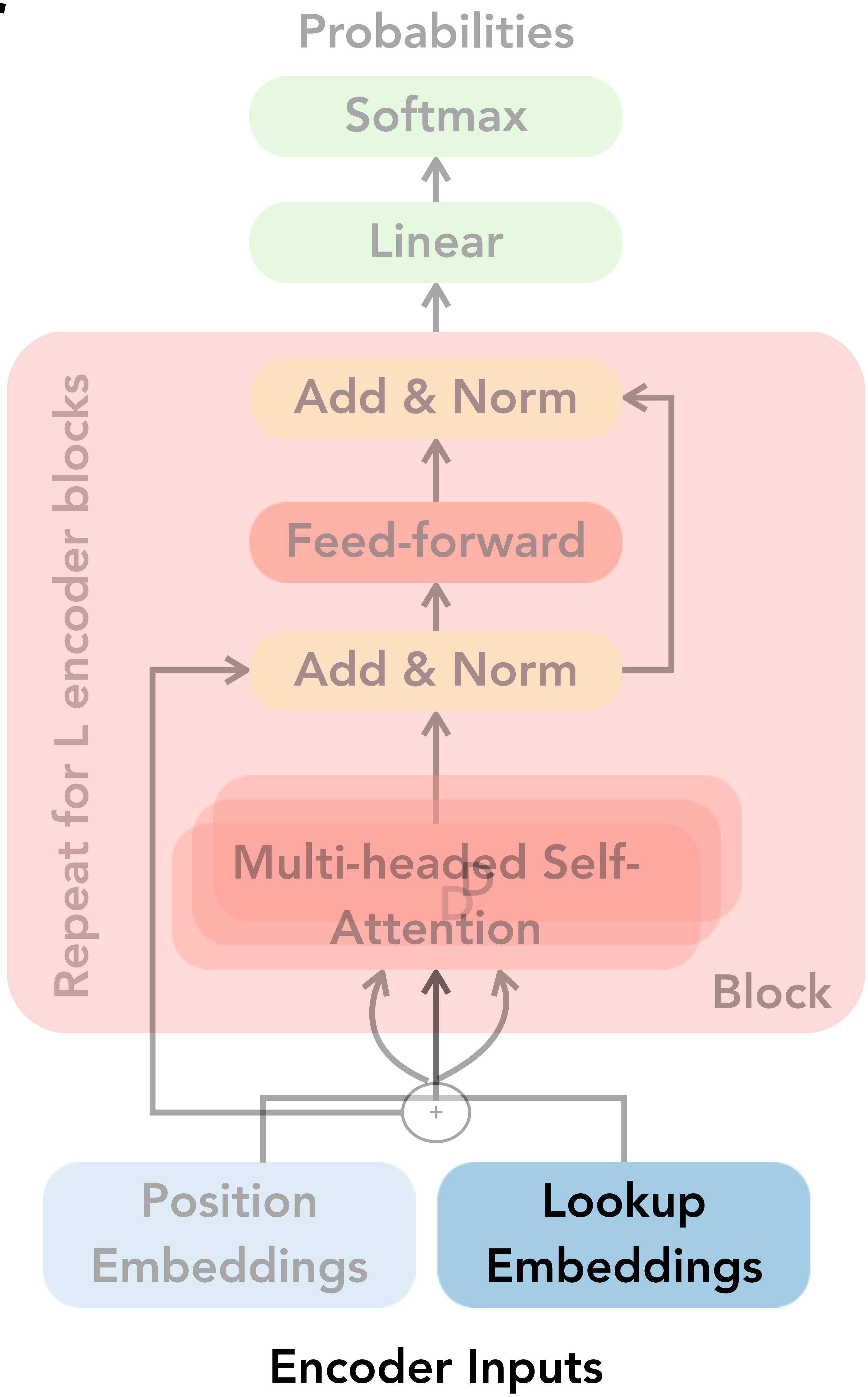
The Input Layer

- So far, we have made some assumptions about a language's vocabulary
- Our approach so far: use a known, fixed vocabulary
  - Built from training data, with tens of thousands of components



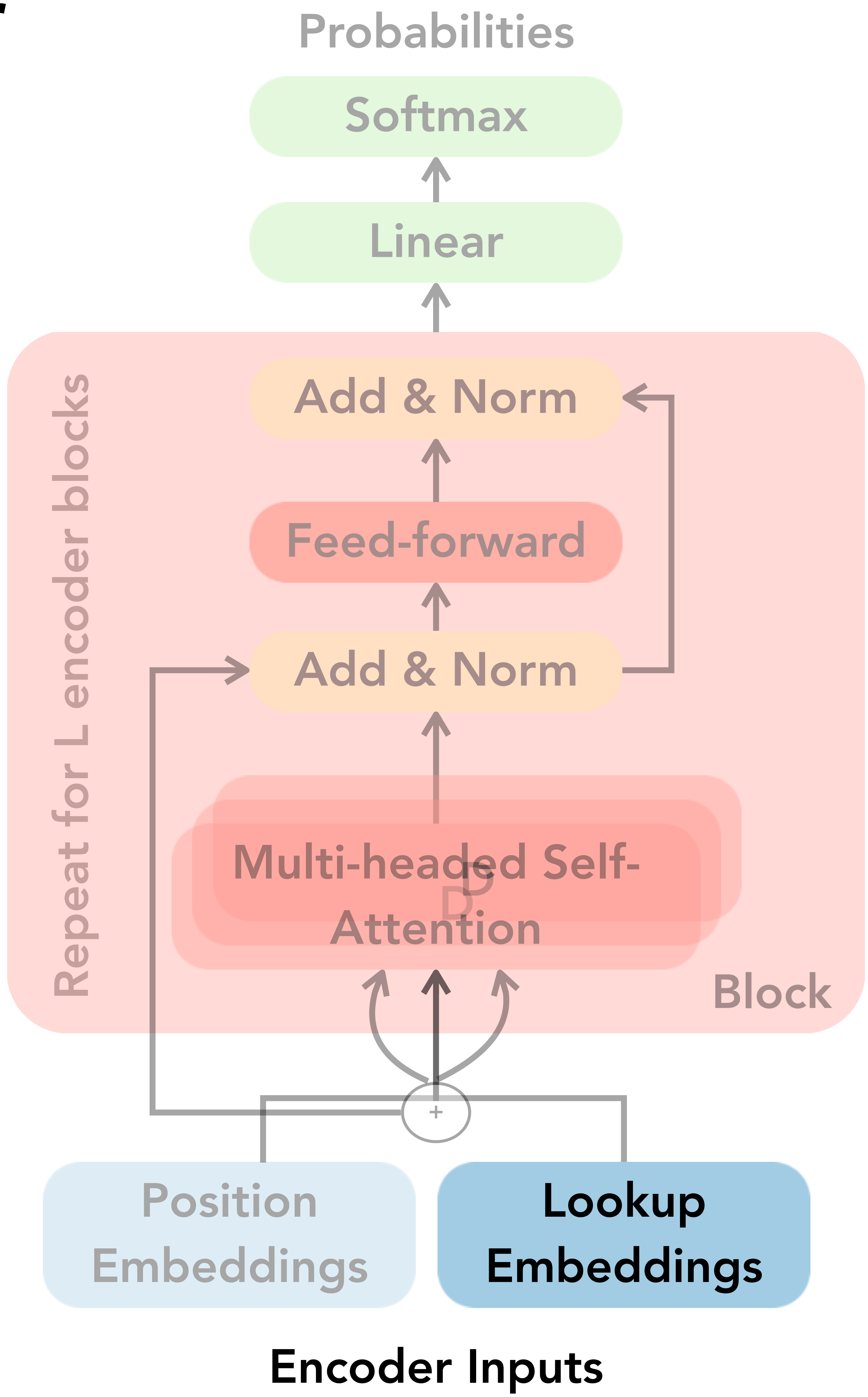
The Input Layer

- So far, we have made some assumptions about a language's vocabulary
- Our approach so far: use a known, fixed vocabulary
  - Built from training data, with tens of thousands of components
  - However, even with the largest vocabulary, we may encounter out-of-vocabulary words at test time



The Input Layer

- So far, we have made some assumptions about a language's vocabulary
- Our approach so far: use a known, fixed vocabulary
  - Built from training data, with tens of thousands of components
  - However, even with the largest vocabulary, we may encounter out-of-vocabulary words at test time
  - Our approach so far: map novel words seen at test time (OOV) to a single UNK



How to get the words?

How to get the words?

Or, more accurately, the tokens?

How to get the words?

Or, more accurately, the tokens?

- Problem: break the text into a sequence of discrete tokens

How to get the words?

Or, more accurately, the tokens?

- Problem: break the text into a sequence of discrete tokens
- For alphabetic languages such as English, deterministic scripts usually suffice to achieve accurate tokenization

How to get the words?

Or, more accurately, the tokens?

- Problem: break the text into a sequence of discrete tokens
- For alphabetic languages such as English, deterministic scripts usually suffice to achieve accurate tokenization
- However, in languages such as Chinese and Swahili, words are typically composed of a small number of characters, without intervening whitespace

Word Structure in Language

Word Structure in Language

- Finite vocabulary assumptions make even less sense in many languages.

Word Structure in Language

- Finite vocabulary assumptions make even less sense in many languages.
  - Many languages exhibit complex morphology, or word structure.

Word Structure in Language

- Finite vocabulary assumptions make even less sense in many languages.
  - Many languages exhibit complex morphology, or word structure.
  - The effect is more word types, each occurring fewer times.

 USC Viterbi

Word Structure in Language

- Finite vocabulary assumptions make even less sense in many languages.
  - Many languages exhibit complex morphology, or word structure.
  - The effect is more word types, each occurring fewer times.

-ambia = to tell

Example: Swahili verbs can have hundreds of conjugations, each encoding a wide variety of information. (Tense, mood, definiteness, negation, information about the object, ++)

Conjugation of *-ambia*

Non-finite forms		
Form	Positive	Negative
Infinitive	kuambia	kutoambia
Simple finite forms		
Positive form	Singular	Plural
Imperative	ambia	ambieni
Habitual	huambia	

Complex finite forms																		
Polarity	Persons				Persons / Classes		Classes											
	1st		2nd		3rd / M-wa		M-mi		Ma		Ki-vi		N		U	Ku	Pa	Mu
	Sg.	Pl.	Sg.	Pl.	Sg. / 1	Pl. / 2	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18
Past																		[less ▲]
Positive	niliambia naliambia	tuliambia twaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia
Present																		[less ▲]
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii
Future																		[less ▲]
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia
Subjunctive																		[less ▲]
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie
Present Conditional																		[less ▲]
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia
Negative	nisingeambia singeambia	tusingeambia hatungeambia	usingeambia hungeambia	msingeambia hamngeambia	asingeambia hangeambia	wasingeambia hawangeambia	usingeambia haungeambia	isingeambia haingeambia	lisingeambia halingeambia	yasingeambia hayangeambia	kisingeambia hakingeambia	visingeambia havingeambia	isingeambia haingeambia	zisingeambia hazingeambia	usingeambia haungeambia	kusingeambia hakungeambia	pasingeambia hapangeambia	musingeambia hamungeambia
Past Conditional																		[less ▲]
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia
Negative	nisingaliambia singaliambia	tusingaliambia hatungaliambia	usingaliambia hungaliambia	msingaliambia hamngaliambia	asingaliambia hangaliambia	wasingaliambia hawangaliambia	usingaliambia haungaliambia	isingaliambia haingaliambia	lisingaliambia halingaliambia	yasingaliambia hayangaliambia	kisingaliambia hakingaliambia	visingaliambia havingaliambia	isingaliambia haingaliambia	zisingaliambia hazingaliambia	usingaliambia haungaliambia	kusingaliambia hakungaliambia	pasingaliambia hapangaliambia	musingaliambia hamungaliambia
Conditional Contrary to Fact																		[less ▲]
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia
Gnomic																		[less ▲]
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia
Perfect																		[less ▲]

Source: Wiktionary

Subword Modeling

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

Subword Modeling

- Solution: look at subwords!

Hello how are U tday?

↓

hello how are u tday?

↓

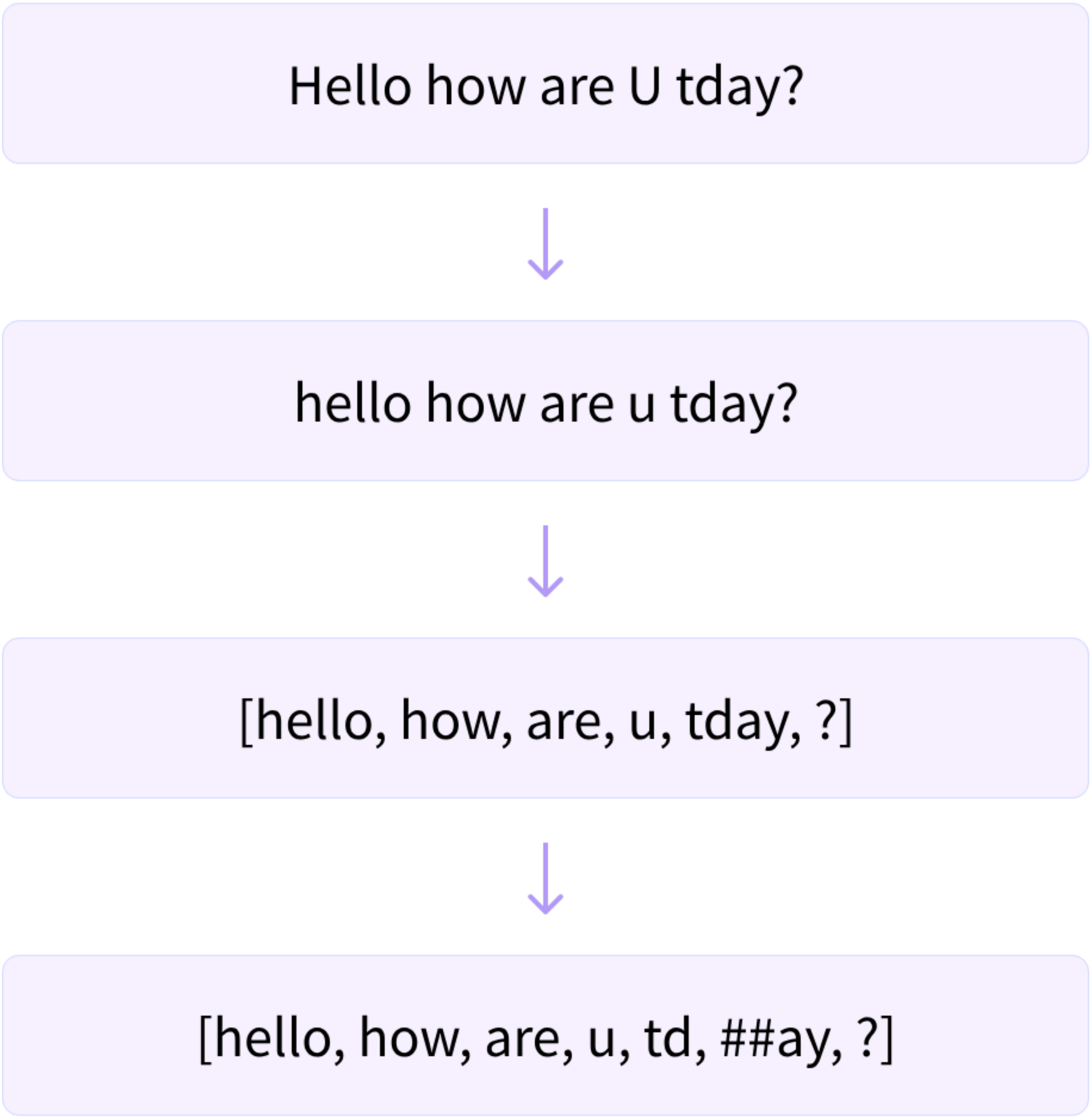
[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level



Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

USC Viterbi

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)
- At training and testing time, each word is split into a sequence of known subwords

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

USC Viterbi

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
    - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)
- At training and testing time, each word is split into a sequence of known subwords
- Different algorithms:

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

USC Viterbi

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)
- At training and testing time, each word is split into a sequence of known subwords
- Different algorithms:
  - Byte-Pair Encoding

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)
- At training and testing time, each word is split into a sequence of known subwords
- Different algorithms:
  - Byte-Pair Encoding
  - WordPiece Modeling

Hello how are U tday?

hello how are u tday?

[hello, how, are, u, tday, ?]

[hello, how, are, u, td, ##ay, ?]

Subword Modeling

- Solution: look at subwords!
- Subword modeling encompasses a wide range of methods for reasoning about structure below the word level
  - Subwords may be parts of words, characters, bytes
- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens)
- At training and testing time, each word is split into a sequence of known subwords
- Different algorithms:
  - Byte-Pair Encoding
  - WordPiece Modeling
  - Follow different strategies. Often contain prepending / appending special tokens (##, </w>)

Hello how are U tday?

↓

hello how are u tday?

↓

[hello, how, are, u, tday, ?]

↓

[hello, how, are, u, td, ##ay, ?]

USC Viterbi

Word structure and subword models

- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components.
- In the worst case, words are split into as many subwords as they have characters.

Byte-pair encoding

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an “end-of-word” symbol.

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword
     - This is a learned operation!

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword
     - This is a learned operation!
     - Only combine pairs (hence the name!)

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword
     - This is a learned operation!
     - Only combine pairs (hence the name!)
  3. Replace instances of the character pair with the new subword; repeat until desired vocabulary size.

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword
     - This is a learned operation!
     - Only combine pairs (hence the name!)
  3. Replace instances of the character pair with the new subword; repeat until desired vocabulary size.
- At test time, first split words into sequences of characters, then apply the learned operations to merge the characters into larger, known symbols

USC Viterbi

Byte-pair encoding

- Byte-pair encoding is a simple, effective strategy for defining a subword vocabulary
- Adapted for word segmentation from data compression technique (Gage, 1994)
  - Instead of merging frequent pairs of bytes, we merge characters or character sequences
- Algorithm:
  1. Start with a vocabulary containing only characters and an "end-of-word" symbol.
  2. Using a corpus of text, find *the most common adjacent characters* "a,b"; add "ab" as a subword
     - This is a learned operation!
     - Only combine pairs (hence the name!)
  3. Replace instances of the character pair with the new subword; repeat until desired vocabulary size.
- At test time, first split words into sequences of characters, then apply the learned operations to merge the characters into larger, known symbols
- Originally used in NLP for machine translation; now a similar method (WordPiece) is used in pretrained models.

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t</w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t</w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t</w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w

Frequency

d-e (3)	l-o (7)	t-</w> (8)
e-r (2)	n-e (5)	w-</w> (5)
e-s (8)	o-w (7)	w-e (7)
e-w (5)	r-</w> (2)	w-i (3)
i-d (3)	s-t (8)	

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	l o w e r </w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>
l o w </w>	w i d e s t</w>	n e w e s t </w>
l o w </w>	w i d e s t </w>	n e w e s t </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es								

Frequency

d-e (3)	l-o (7)	t-</w> (8)
e-r (2)	n-e (5)	w-</w> (5)
e-s (8)	o-w (7)	w-e (7)
e-w (5)	r-</w> (2)	w-i (3)
i-d (3)	s-t (8)	

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w es t </w>
l o w </w>	l o w e r </w>	n e w es t </w>
l o w </w>	w i d es t </w>	n e w es t </w>
l o w </w>	w i d es t</w>	n e w es t </w>
l o w </w>	w i d es t </w>	n e w es t </w>

Frequency

d-es (3)	l-o (7)	w-</w> (5)
e-r (2)	n-e (5)	w-es (5)
e-w (5)	o-w (7)	w-e (2)
es-t (8)	r-</w> (2)	w-i (3)
i-d (3)	t-</w> (8)	

Vocabulary

d	e	i	l	n	o	s	t	w
es								

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w es t </w>
l o w </w>	l o w e r </w>	n e w es t </w>
l o w </w>	w i d es t </w>	n e w es t </w>
l o w </w>	w i d es t</w>	n e w es t </w>
l o w </w>	w i d es t </w>	n e w es t </w>

Frequency

d-es (3)	l-o (7)	w-</w> (5)
e-r (2)	n-e (5)	w-es (5)
e-w (5)	o-w (7)	w-e (2)
es-t (8)	r-</w> (2)	w-i (3)
i-d (3)	t-</w> (8)	

Vocabulary

d	e	i	l	n	o	s	t	w
es	est							

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w est </w>
l o w </w>	l o w e r </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es	est							

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

USC Viterbi

BPE in action

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

Corpus

low</w>	lower</w>	newest</w>
low</w>	lower</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>
low</w>	widest</w>	newest</w>

Corpus

l o w </w>	l o w e r </w>	n e w est </w>
l o w </w>	l o w e r </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>
l o w </w>	w i d est </w>	n e w est </w>

Vocabulary

d	e	i	l	n	o	s	t	w
es	est	est</w>	lo	low	low</w>	ne	new	newest</w>

After 10 merges

Source: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/

WordPiece Modeling

WordPiece Modeling

- Algorithm from Google, similar to BPE

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
    - Each word is initially split by adding ## to all the characters inside a word

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
    - Each word is initially split by adding ## to all the characters inside a word
    - So, for instance, “word” gets split like this: w ##o ##r ##d

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
  - Each word is initially split by adding ## to all the characters inside a word
  - So, for instance, "word" gets split like this: w ##o ##r ##d
  - For this vocabulary:
    - ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

USC Viterbi

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
  - Each word is initially split by adding ## to all the characters inside a word
  - So, for instance, "word" gets split like this: w ##o ##r ##d
  - For this vocabulary:
    - ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
  - Splits may look like:
    - ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)

WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
  - Each word is initially split by adding ## to all the characters inside a word
  - So, for instance, “word” gets split like this: w ##o ##r ##d
  - For this vocabulary:
    - ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
  - Splits may look like:
    - ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
- On merging, ## ***between*** the two tokens is removed

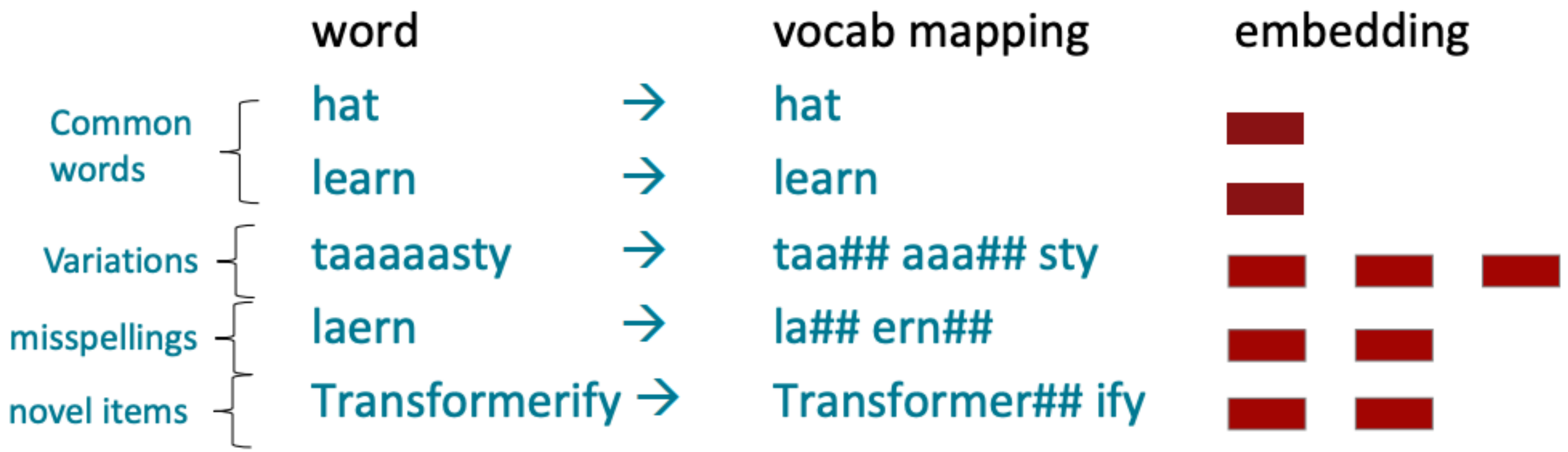
WordPiece Modeling

- Algorithm from Google, similar to BPE
- Identifies subwords by adding a prefix (##)
  - Each word is initially split by adding ## to all the characters inside a word
  - So, for instance, "word" gets split like this: w ##o ##r ##d
  - For this vocabulary:
    - ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
  - Splits may look like:
    - ("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)
- On merging, ## ***between*** the two tokens is removed
  - This explains the presence of the token "##ing"

WordPiece Modeling Outcome

- Different stopping criteria: number of merges or size of resulting vocabulary
- In the worst case, at test time, words are split into as many subwords as they have characters
- Common words end up being a part of the subword vocabulary, while rarer words are split into (sometimes intuitive, sometimes not) components

WordPiece Outcome



Tokenization: Questions

- Where does the token “##ing” come from?
  - In WordPiece tokenization, all non-starting characters are initialized as ##x.
    - Like: h, ##e, ##l, ##l, ##o.
  - Upon merging, only the first segment keeps its ##.
- How is tokenization done in Chinese?
  - Follows the same broad overall algorithm, but the initial split into characters involve language-specific rules
  - e.g. stroke-level tokenization

Source: https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt

Natural Language Generation

Natural Language Generation

Natural Language Generation

- Natural language understanding and natural language generation are two sides of the same coin

Natural Language Generation

- Natural language understanding and natural language generation are two sides of the same coin
  - In order to generate good language, you need to understand language

Natural Language Generation

- Natural language understanding and natural language generation are two sides of the same coin
  - In order to generate good language, you need to understand language
  - If you understand language, you should be able to generate it (with some effort)

Natural Language Generation

- Natural language understanding and natural language generation are two sides of the same coin
  - In order to generate good language, you need to understand language
  - If you understand language, you should be able to generate it (with some effort)
- NLG is the workhorse of many classic and novel applications
  - AI Assistants
  - Translators
  - Search summarizers

NLG Use Cases

USC Viterbi

NLG Use Cases

Simple and Effective Multi-Paragraph Reading Comprehension

Christopher Clark, Matt Gardner · Computer Science · ACL · 29 October 2017

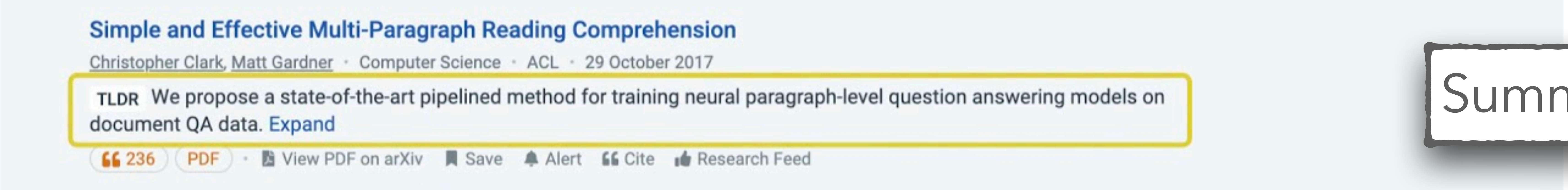
TLDR We propose a state-of-the-art pipelined method for training neural paragraph-level question answering models on document QA data. Expand

236 PDF · View PDF on arXiv Save Alert Cite Research Feed

Summarization

USC Viterbi

NLG Use Cases

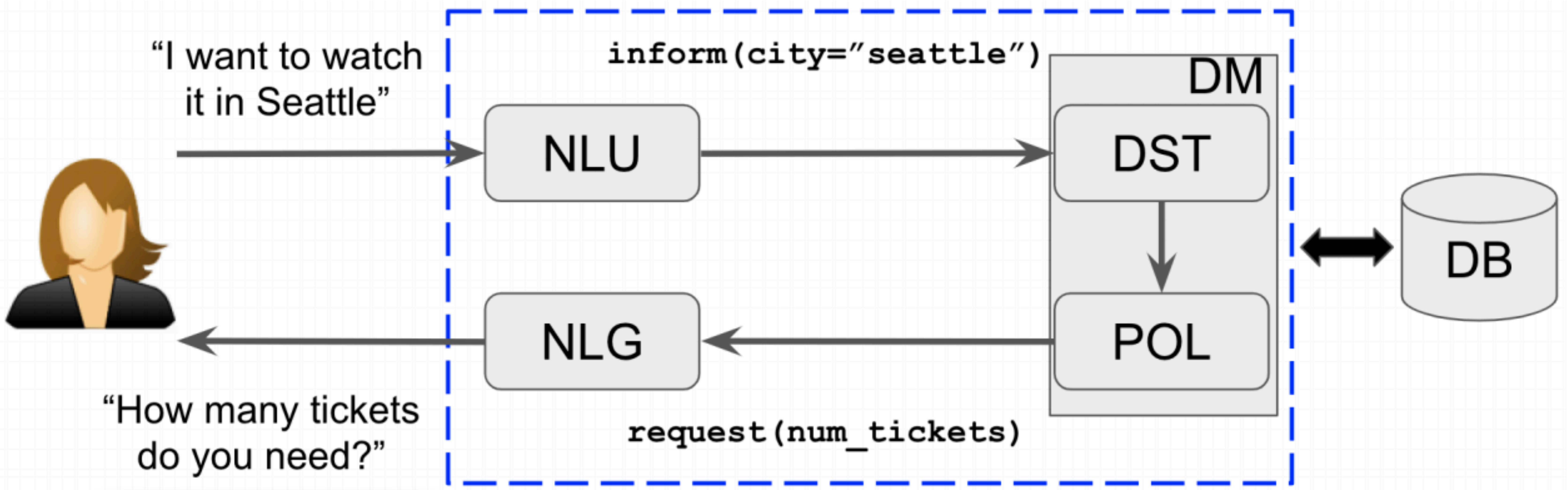

Simple and Effective Multi-Paragraph Reading Comprehension

Christopher Clark, Matt Gardner · Computer Science · ACL · 29 October 2017

TLDR We propose a state-of-the-art pipelined method for training neural paragraph-level question answering models on document QA data. Expand

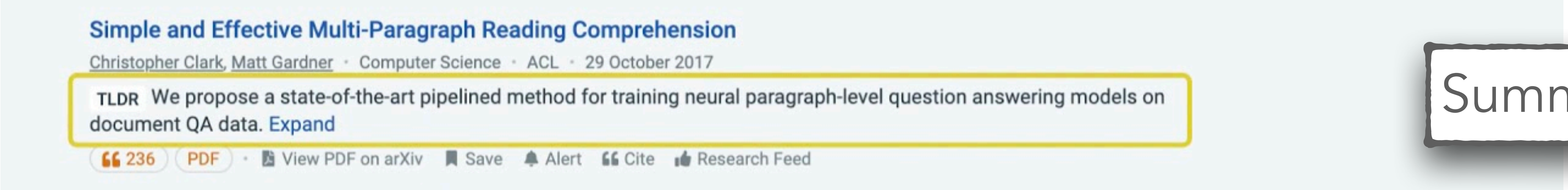
236 PDF · View PDF on arXiv Save Alert Cite Research Feed

Summarization

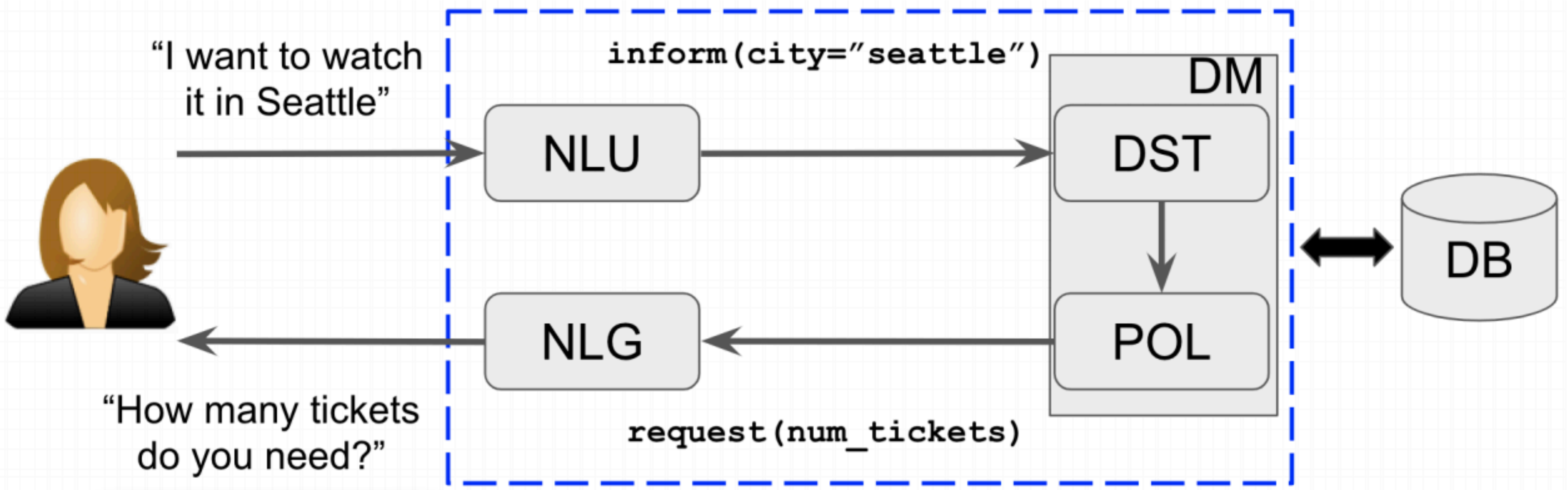


Task-driven Dialog

NLG Use Cases



Summarization



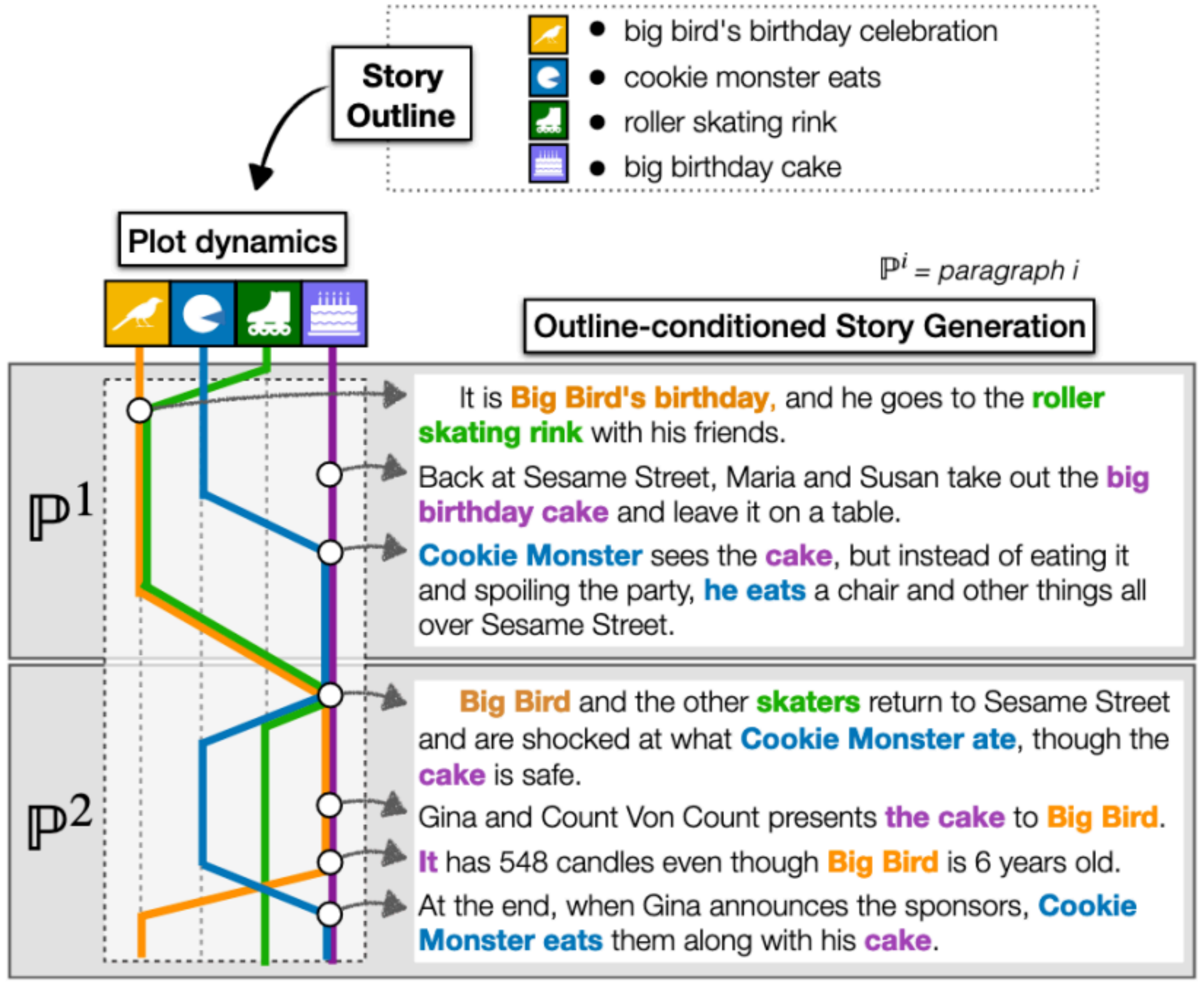
Task-driven Dialog

Chitchat Dialog

More Interesting NLG Uses

More Interesting NLG Uses

Creative stories

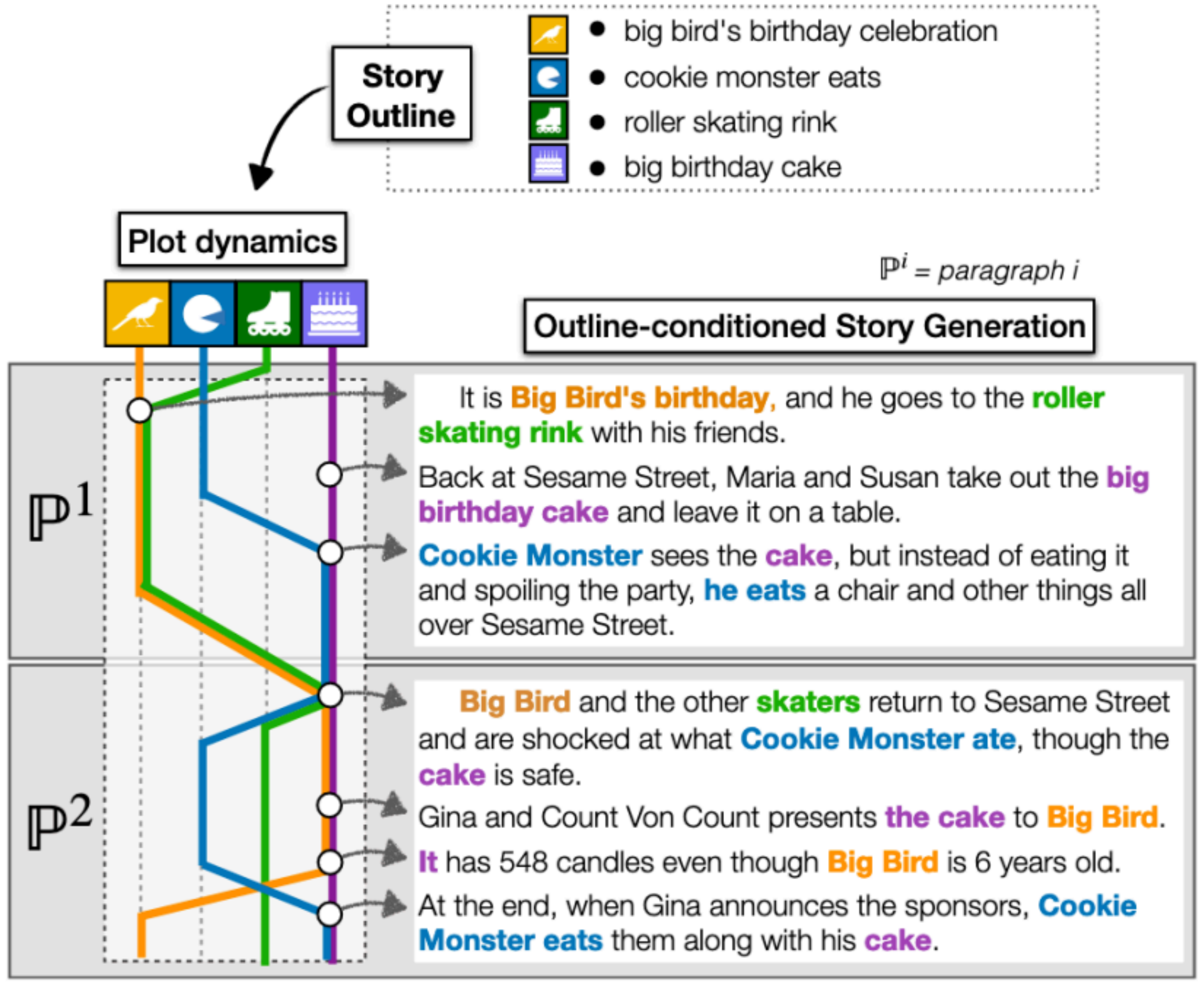


Rashkin et al., 2020

USC Viterbi

More Interesting NLG Uses

Creative stories



Rashkin et al., 2020

Data-to-text

Table Title: Robert Craig (American football)
Section Title: National Football League statistics
Table Description:None

		RUSHING					RECEIVING				
YEAR	TEAM	ATT	YDS	AVG	LNG	TD	NO.	YDS	AVG	LNG	TD
1983	SF	176	725	4.1	71	8	48	427	8.9	23	4
1984	SF	155	649	4.2	28	4	71	675	9.5	64	3
1985	SF	214	1050	4.9	62	9	92	1016	11	73	6
1986	SF	204	830	4.1	25	7	81	624	7.7	48	0
1987	SF	215	815	3.8	25	3	66	492	7.5	35	1
1988	SF	310	1502	4.8	46	9	76	534	7.0	22	1
1989	SF	271	1054	3.9	27	6	49	473	9.7	44	1
1990	SF	141	439	3.1	26	1	25	201	8.0	31	0
1991	RAI	162	590	3.6	15	1	17	136	8.0	20	0
1992	MIN	105	416	4.0	21	4	22	164	7.5	22	0
1993	MIN	38	119	3.1	11	1	19	169	8.9	31	1
Totals	**-**	**1991**	**8189**	**4.1**	**71**	**56**	**566**	**4911**	**8.7**	**73**	**17**

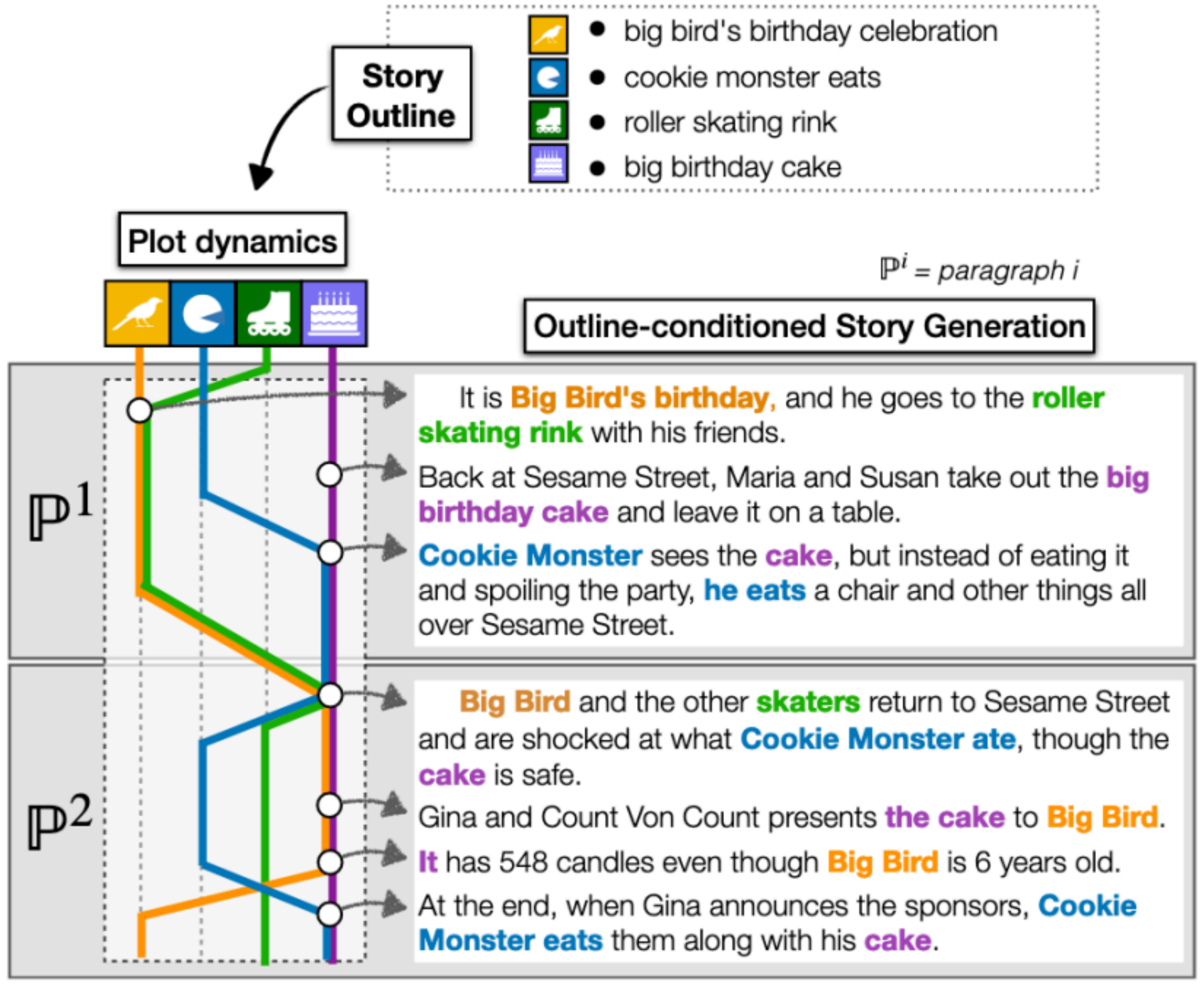
Craig finished his eleven NFL seasons with 8,189 rushing yards and 566 receptions for 4,911 receiving yards.

Parikh et al., 2020

USC Viterbi

More Interesting NLG Uses

Creative stories



Rashkin et al., 2020

Data-to-text

Table Title: Robert Craig (American football)
Section Title: National Football League statistics
Table Description:None

		RUSHING					RECEIVING				
YEAR	TEAM	ATT	YDS	AVG	LNG	TD	NO.	YDS	AVG	LNG	TD
1983	SF	176	725	4.1	71	8	48	427	8.9	23	4
1984	SF	155	649	4.2	28	4	71	675	9.5	64	3
1985	SF	214	1050	4.9	62	9	92	1016	11	73	6
1986	SF	204	830	4.1	25	7	81	624	7.7	48	0
1987	SF	215	815	3.8	25	3	66	492	7.5	35	1
1988	SF	310	1502	4.8	46	9	76	534	7.0	22	1
1989	SF	271	1054	3.9	27	6	49	473	9.7	44	1
1990	SF	141	439	3.1	26	1	25	201	8.0	31	0
1991	RAI	162	590	3.6	15	1	17	136	8.0	20	0
1992	MIN	105	416	4.0	21	4	22	164	7.5	22	0
1993	MIN	38	119	3.1	11	1	19	169	8.9	31	1
Totals	**-**	**1991**	**8189**	**4.1**	**71**	**56**	**566**	**4911**	**8.7**	**73**	**17**

Craig finished his eleven NFL seasons with 8,189 rushing yards and 566 receptions for 4,911 receiving yards.

Parikh et al., 2020

Visual description

Two children are sitting at a table in a restaurant. The children are one little girl and one little boy. The little girl is eating a pink frosted donut with white icing lines on top of it. The girl has blonde hair and is wearing a green jacket with a black long sleeve shirt underneath. The little boy is wearing a black zip up jacket and is holding his finger to his lip but is not eating. A metal napkin dispenser is in between them at the table. The wall next to them is white brick. Two adults are on the other side of the short white brick wall. The room has white circular lights on the ceiling and a large window in the front of the restaurant. It is daylight outside.

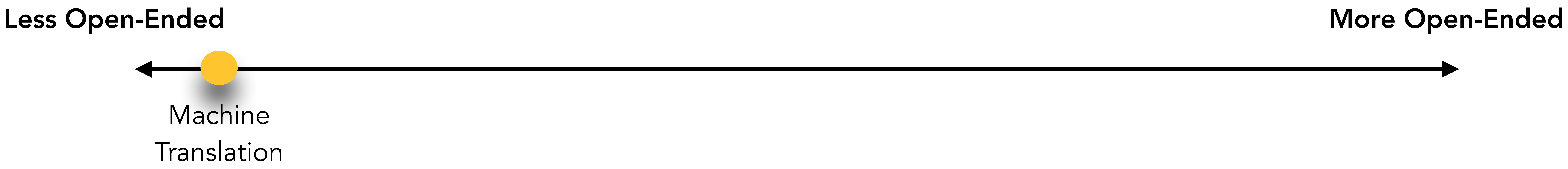
Krause et al., 2017

Broad Spectrum of NLG Tasks

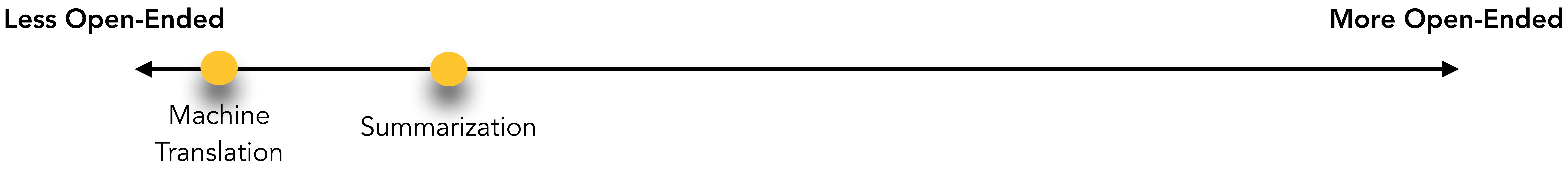
Less Open-Ended **More Open-Ended**

USC Viterbi

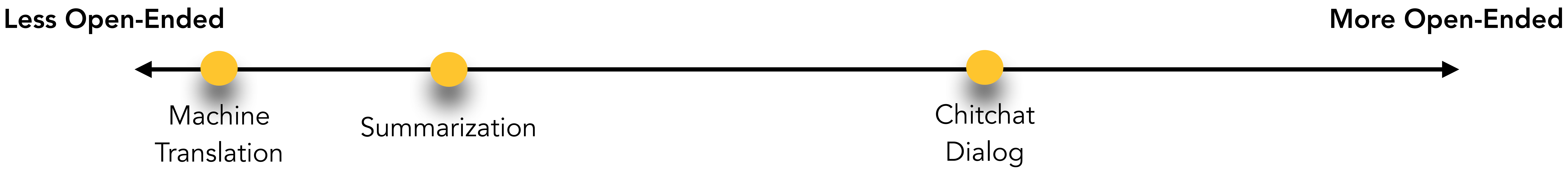
Broad Spectrum of NLG Tasks



Broad Spectrum of NLG Tasks



Broad Spectrum of NLG Tasks



Broad Spectrum of NLG Tasks

Broad Spectrum of NLG Tasks

Broad Spectrum of NLG Tasks

Open-ended generation: the output distribution still has high freedom.

Broad Spectrum of NLG Tasks

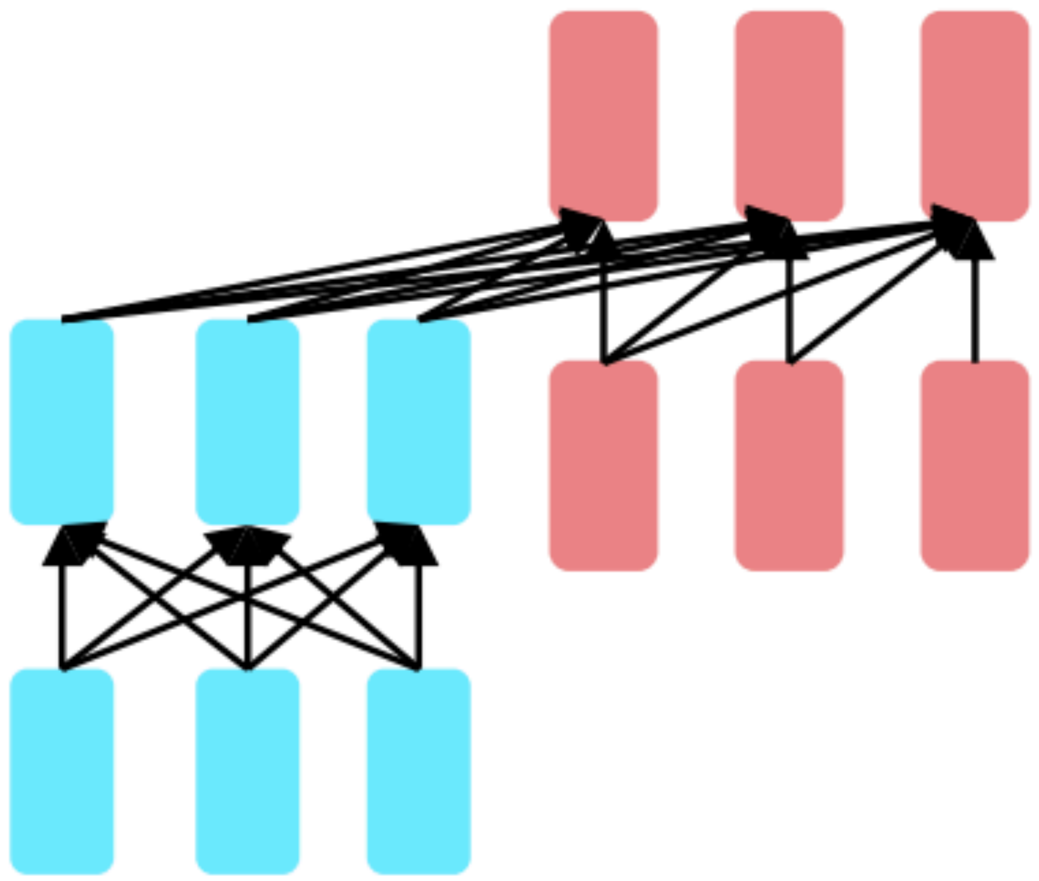
Open-ended generation: the output distribution still has high freedom.

Non-open-ended generation: the input mostly determines the output generation.

Broad Spectrum of NLG Tasks

USC Viterbi

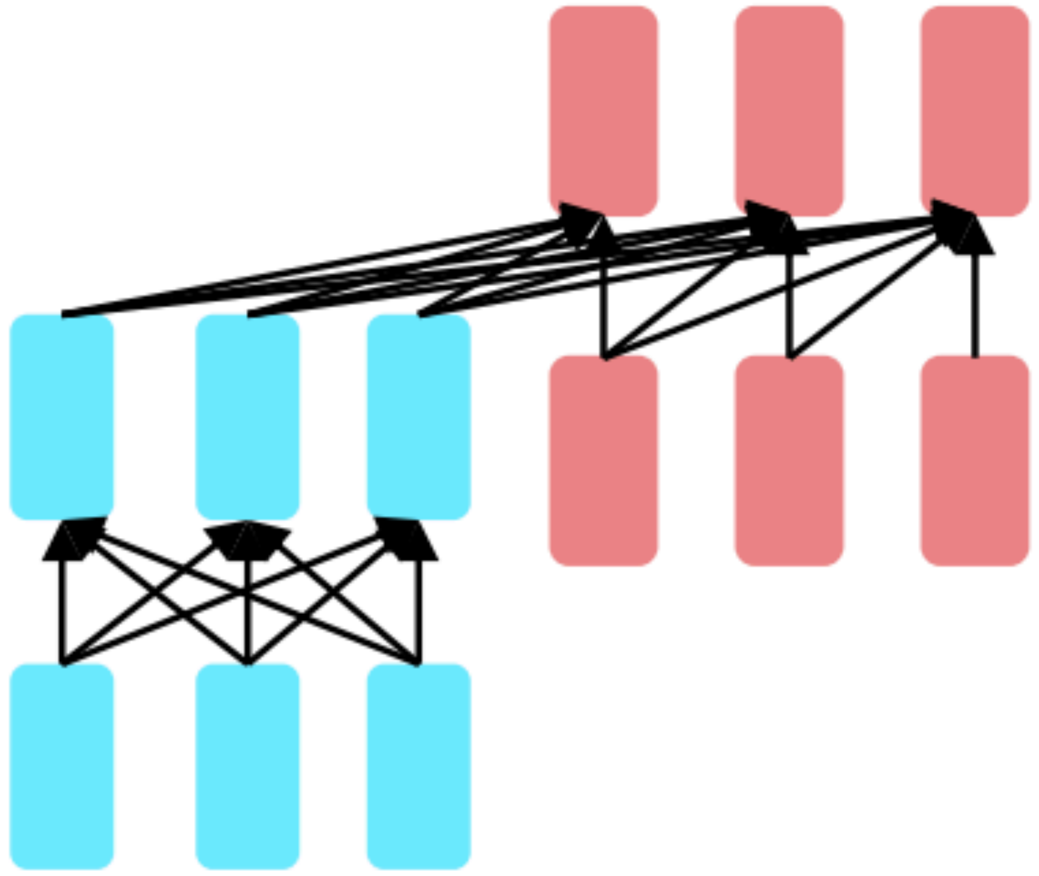
Broad Spectrum of NLG Tasks



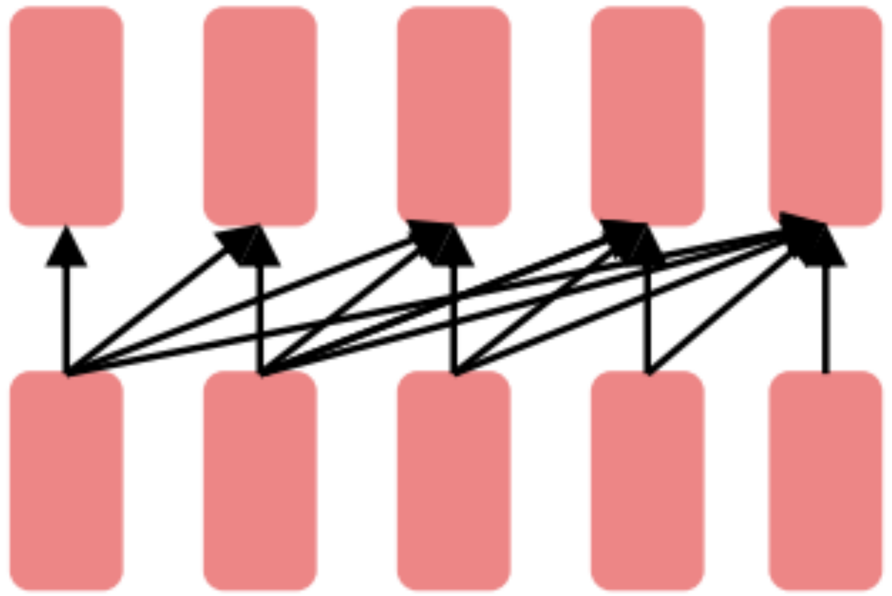
Encoder-Decoders

USC Viterbi

Broad Spectrum of NLG Tasks



Encoder-Decoders



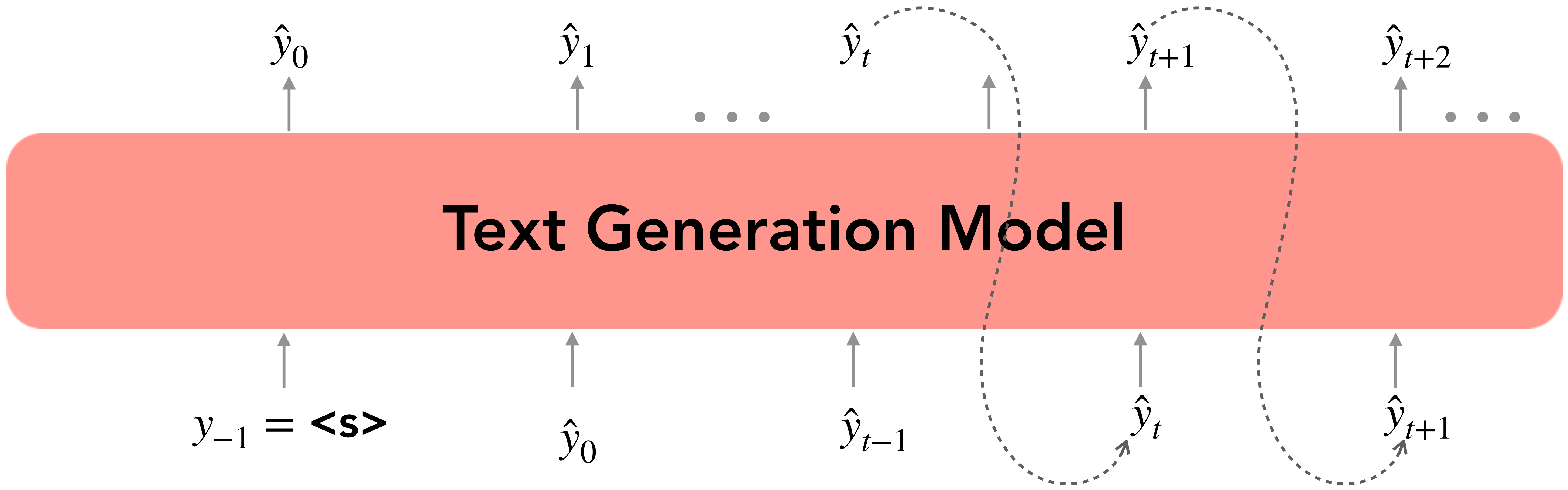
Decoders

Language Generation: Fundamentals

In autoregressive text generation models, at each time step t, our model takes in a sequence of tokens as input $S = f_\theta(y_{<t}) \in \mathbb{R}^V$ and outputs a new token, $\hat{y}_t$

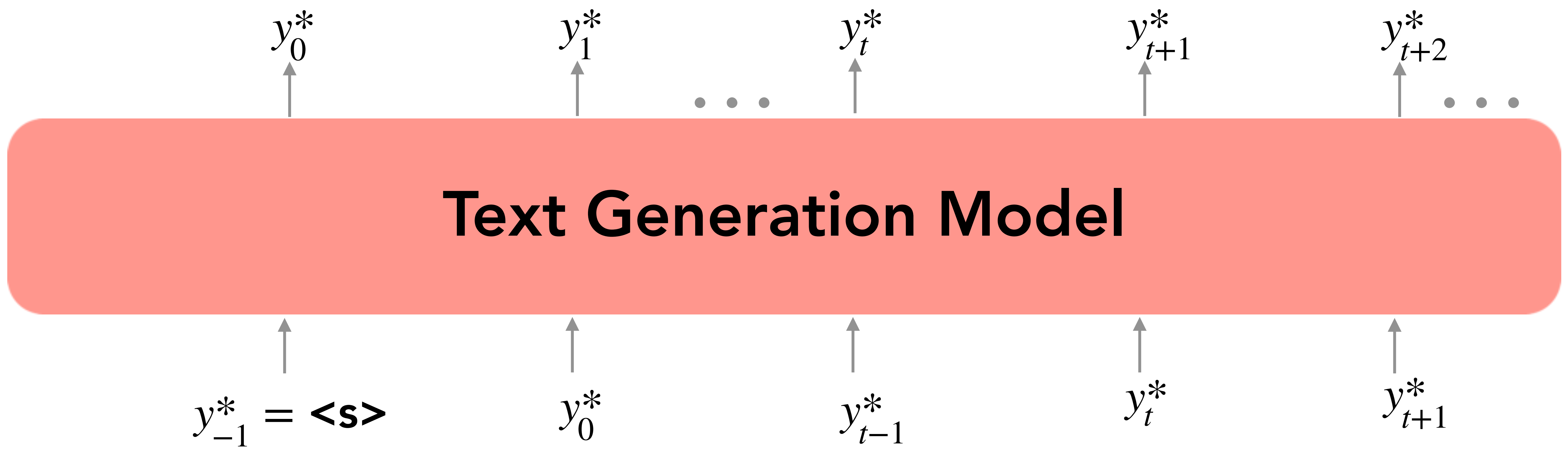
For model $f_\theta(\cdot)$ and vocabulary V, we get scores $S = f_\theta(y_{<t}) \in \mathbb{R}^V$

$$P(w \mid y_{<t}) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$



Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}^*$

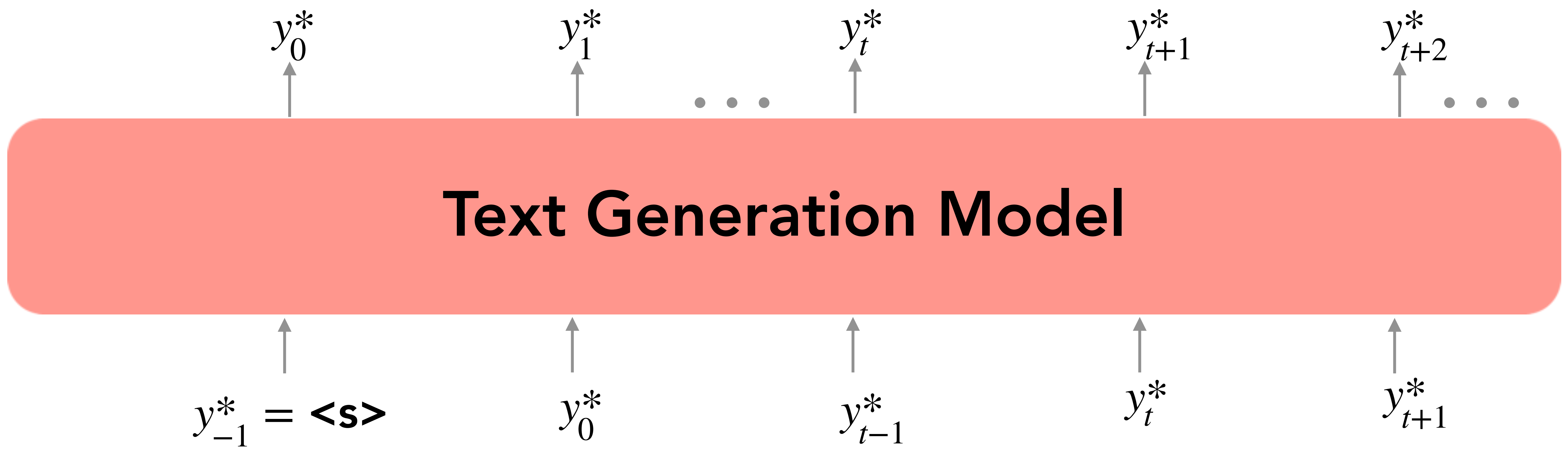


USC Viterbi

Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}^*$

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}) = -\sum_{t=1}^{T} \log \frac{\exp(S_{y_t|y_{<t}})}{\sum_{v \in V} \exp(S_{v|y_{<t}})}$$



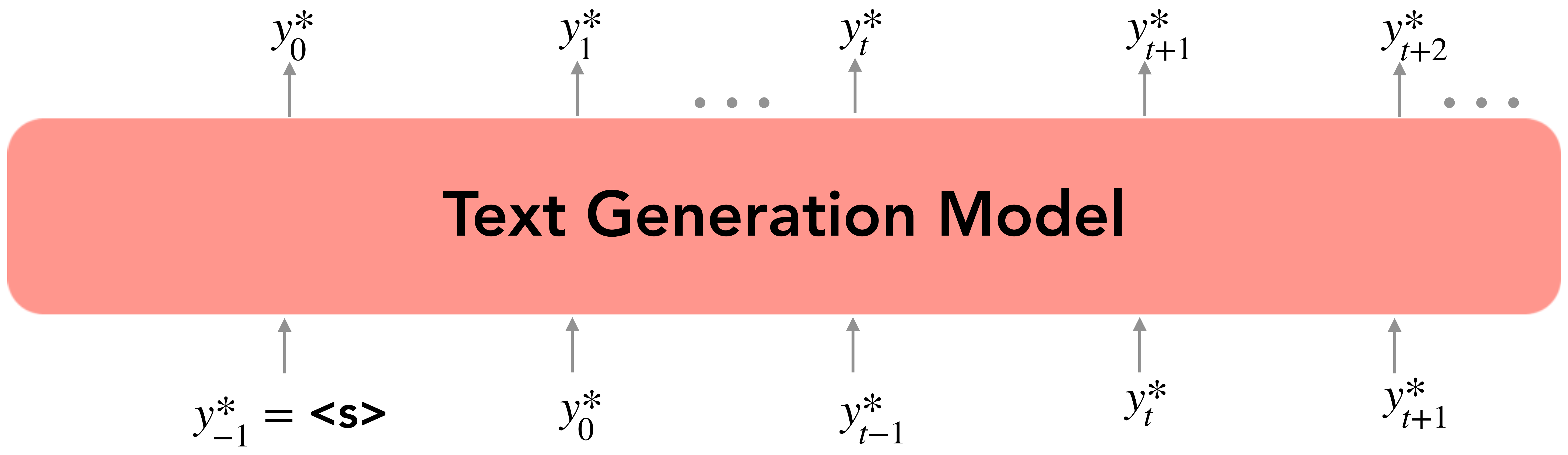
USC Viterbi

Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}^*$

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}) = -\sum_{t=1}^{T} \log \frac{\exp(S_{y_t|y_{<t}})}{\sum_{v \in V} \exp(S_{v|y_{<t}})}$$

- Classification task at each time step trying to predict the actual word y_t^* in the training data

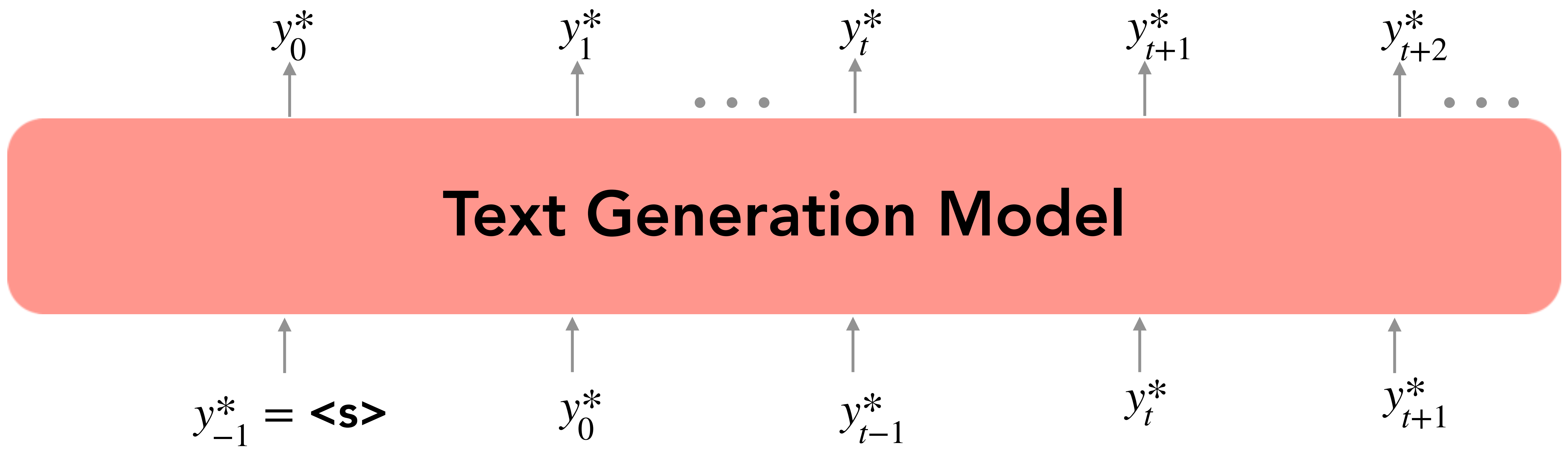


Language Generation: Training

- Trained one token at a time to maximize the probability of the next token y_t^* given preceding words $y_{<t}^*$

$$\mathscr{L} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}) = -\sum_{t=1}^{T} \log \frac{\exp(S_{y_t|y_{<t}})}{\sum_{v \in V} \exp(S_{v|y_{<t}})}$$

- Classification task at each time step trying to predict the actual word y_t^* in the training data
- "Teacher forcing" (reset at each time step to the ground truth)



Teacher Forcing

Teacher Forcing

- Strategy for **training** decoders / language models

Teacher Forcing

- Strategy for **training** decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output $\hat{y}_t$

Teacher Forcing

- Strategy for **training** decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output $\hat{y}_t$
- Runs the risk of **exposure bias**!

Teacher Forcing

- Strategy for **training** decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output $\hat{y}_t$
- Runs the risk of **exposure bias**!
  - During training, our model's inputs are gold context tokens from real, human-generated texts

Teacher Forcing

- Strategy for **training** decoders / language models
- At each time step t in decoding we force the system to use the gold target token from training as the next input x_{t+1}, rather than allowing it to rely on the (possibly erroneous) decoder output $\hat{y}_t$
- Runs the risk of **exposure bias**!
  - During training, our model's inputs are gold context tokens from real, human-generated texts
  - At generation time, our model's inputs are previously–decoded tokens

Language Generation: Inference

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

Inference / Decoding Algorithm

$$\hat{y}_t = g(P(y_t \mid y_{<t}))$$

USC Viterbi

Language Generation: Inference

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

Inference / Decoding Algorithm

$$\hat{y}_t = g(P(y_t | y_{<t}))$$

- The "obvious" decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

Language Generation: Inference

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

Inference / Decoding Algorithm

$$\hat{y}_t = g(P(y_t | y_{<t}))$$

- The "obvious" decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

$$g = \arg\max$$

Language Generation: Inference

- At inference time, our decoding algorithm defines a function to select a token from this distribution:

Inference / Decoding Algorithm

$$\hat{y}_t = g(P(y_t | y_{<t}))$$

- The "obvious" decoding algorithm is to greedily choose the highest probability next token according to the model at each time step

$$g = \arg\max$$

$$\hat{y}_t = \arg\max_{w \in V}(P(y_t = w | y_{<t}))$$

Classic Inference Algorithms: Greedy and Beam Search

Decoding

Decoding

- Generation from a language model is also called decoding

Decoding

- Generation from a language model is also called decoding
  - Think encoder-decoder

Decoding

- Generation from a language model is also called decoding
  - Think encoder-decoder
  - Also called inference

Decoding

- Generation from a language model is also called decoding
  - Think encoder-decoder
  - Also called inference
- Strategy so far: Take $\arg\max$ on each step of the decoder to produce the most probable word on each step

Decoding

- Generation from a language model is also called decoding
  - Think encoder-decoder
  - Also called inference
- Strategy so far: Take $\arg\max$ on each step of the decoder to produce the most probable word on each step
- This is called greedy decoding

Decoding

- Generation from a language model is also called decoding
  - Think encoder-decoder
  - Also called inference
- Strategy so far: Take $\arg\max$ on each step of the decoder to produce the most probable word on each step
- This is called greedy decoding
  - Greedy Strategy: we are not looking ahead, we are making the hastiest decision given all the information we have

Greedy Decoding: Issues

USC Viterbi

Greedy Decoding: Issues

- Greedy decoding has no wiggle room for errors!
  - Input: the green witch arrived
    - Output: llego
    - Output: llego la
    - Output: llego la verde

Greedy Decoding: Issues

- Greedy decoding has no wiggle room for errors!
  - Input: the green witch arrived
    - Output: llego
    - Output: llego la
    - Output: llego la verde
- How to fix this?
  - Need a lookahead strategy / longer-term planning

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$
\begin{aligned}
P(y|x) &= P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x) \\
&= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)
\end{aligned}
$$

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$
\begin{aligned}
P(y|x) &= P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x) \\
&= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)
\end{aligned}
$$

- We could try computing all possible sequences y

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences y
  - This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size

Exhaustive Search Decoding

- Ideally, we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

- We could try computing all possible sequences y
  - This means that on each step t of the decoder, we're tracking V^t possible partial translations, where V is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Beam Search Decoding

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better

USC Viterbi

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top k on each step

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution

Beam Search Decoding

- Core idea: On each step of decoder, keep track of the k most probable partial translations (which we call hypotheses)
  - k is the beam size (in practice around 5 to 10, in NMT)
- A hypothesis has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top k on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

<START>

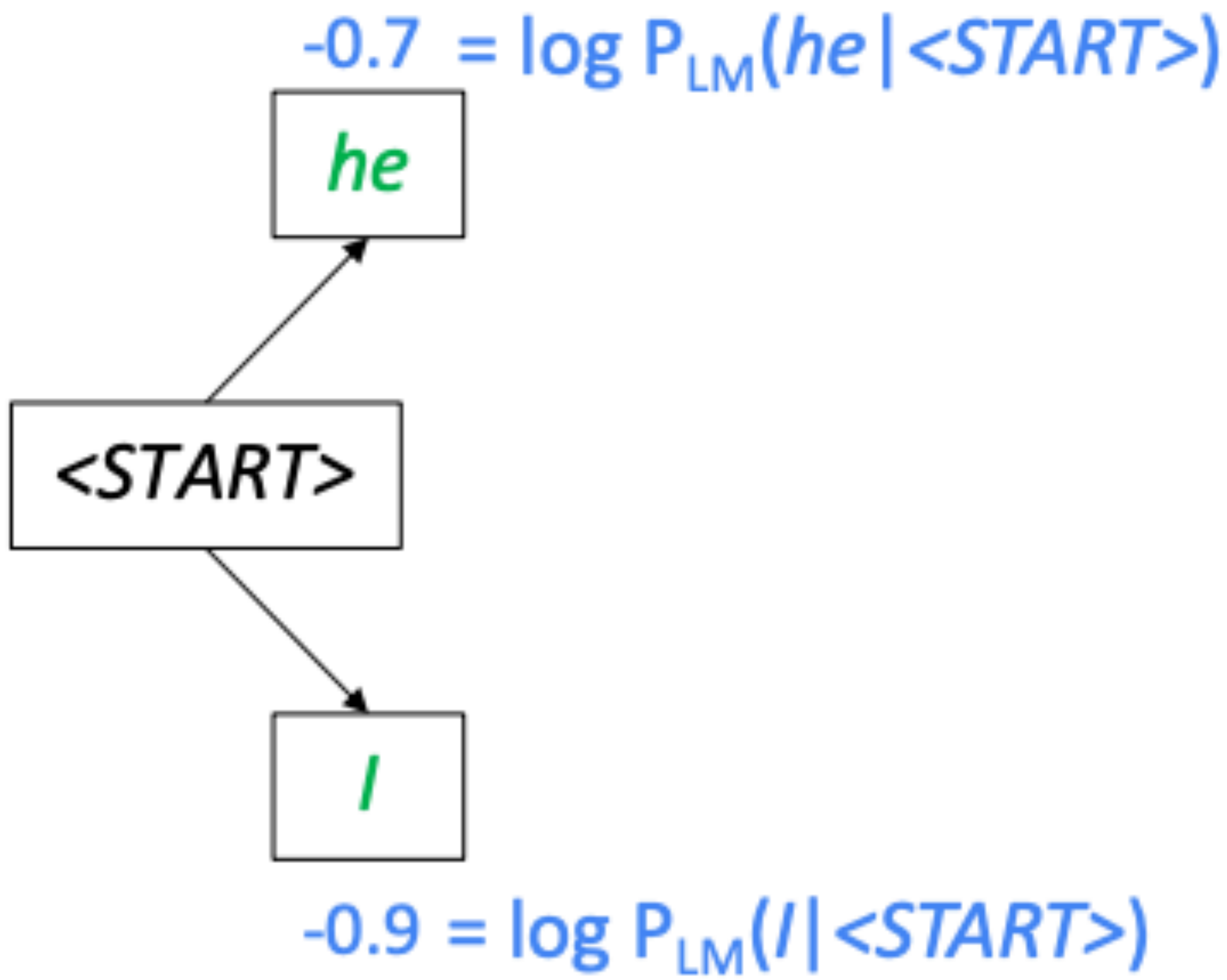
Calculate prob dist of next word

Slide credit: Chris Manning

USC Viterbi

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

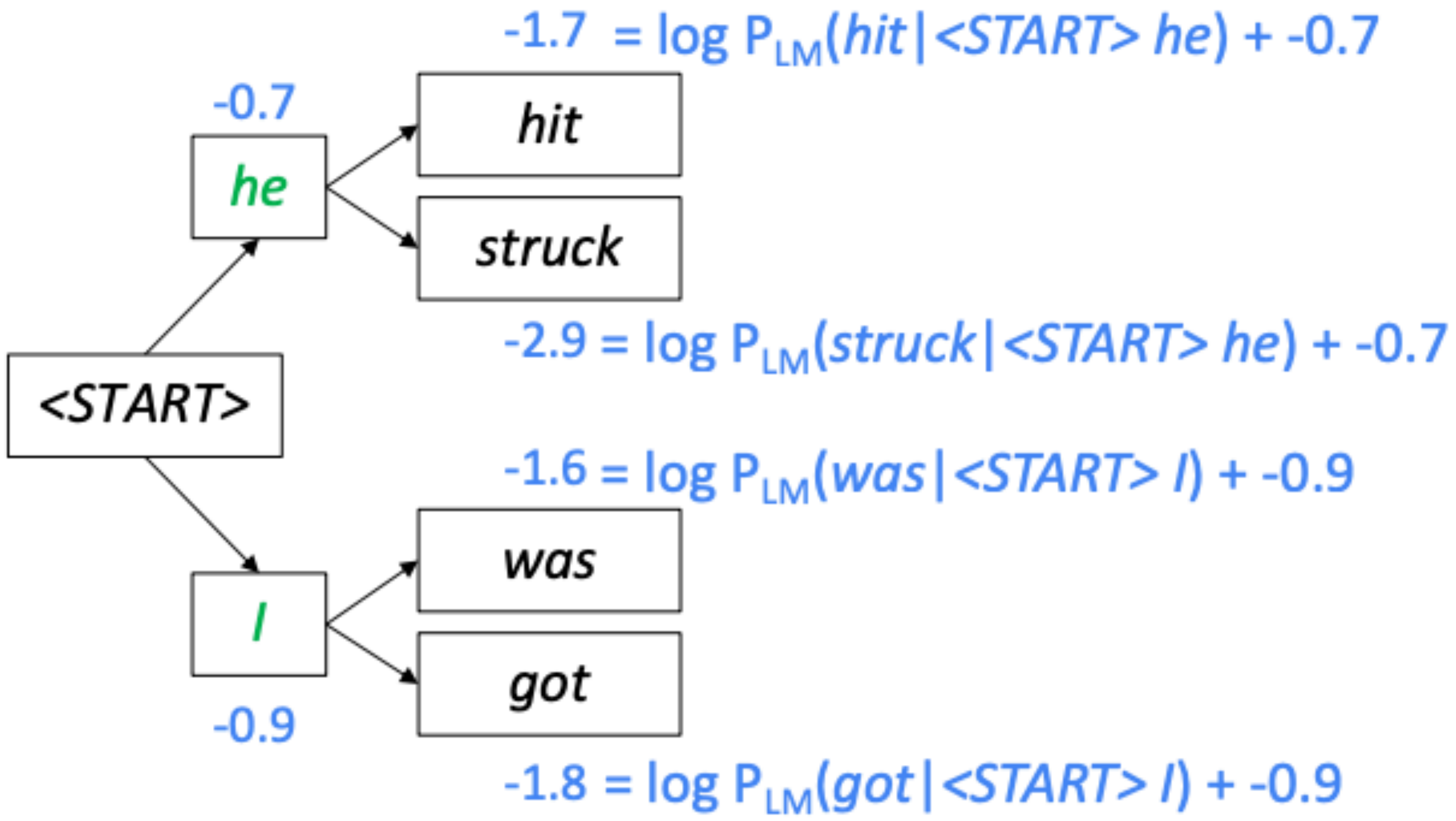


Take top *k* words and compute scores

: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

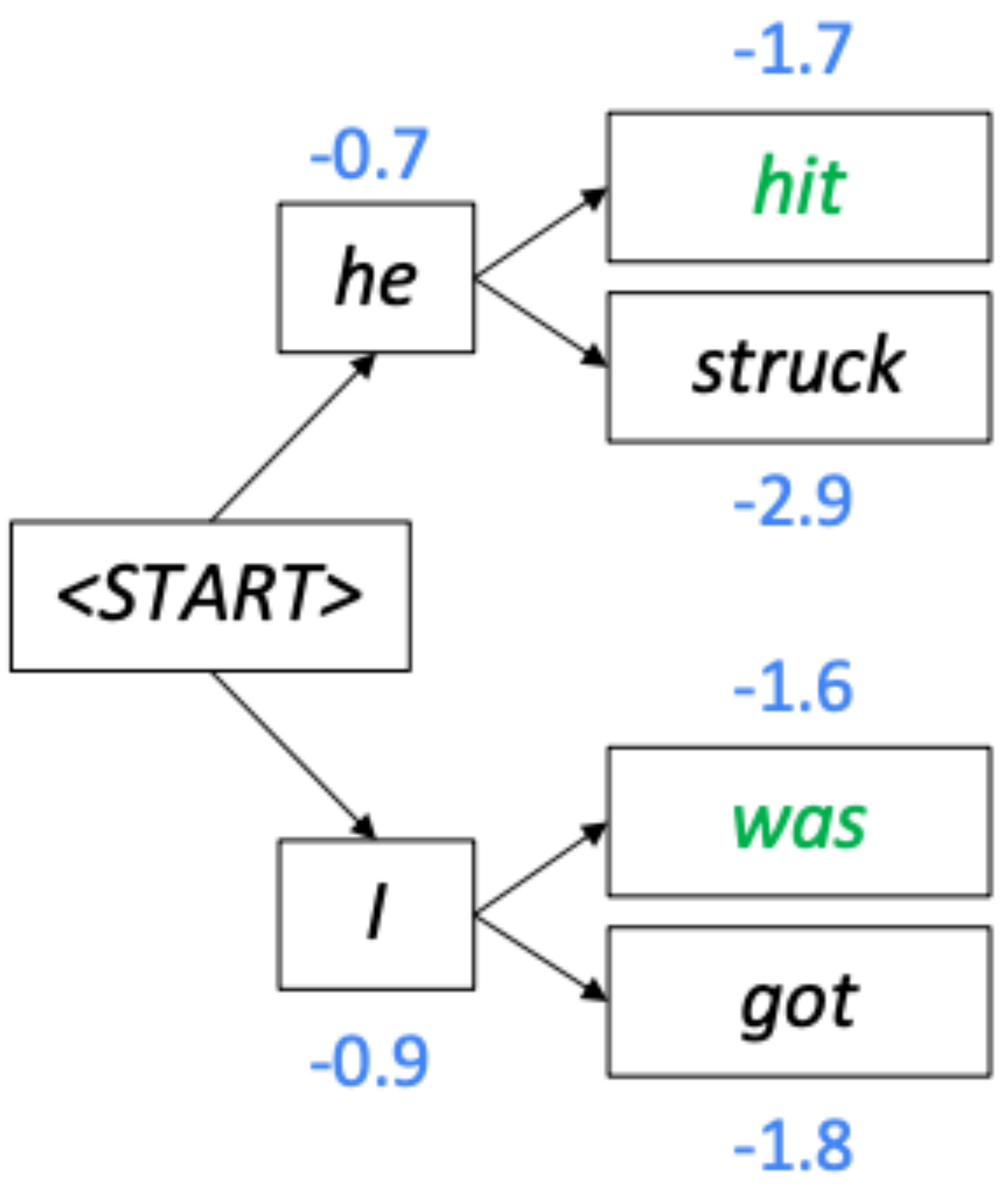


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

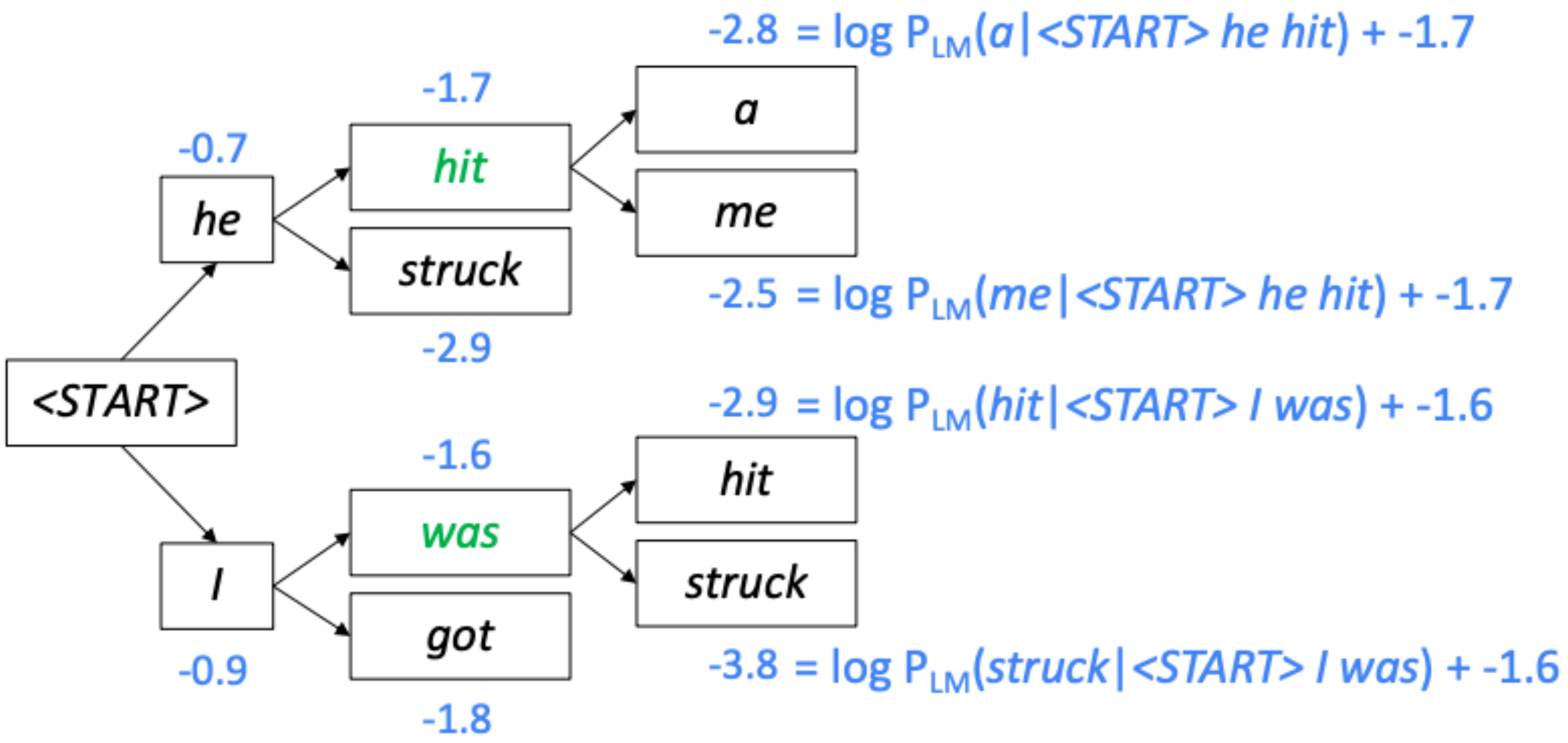


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

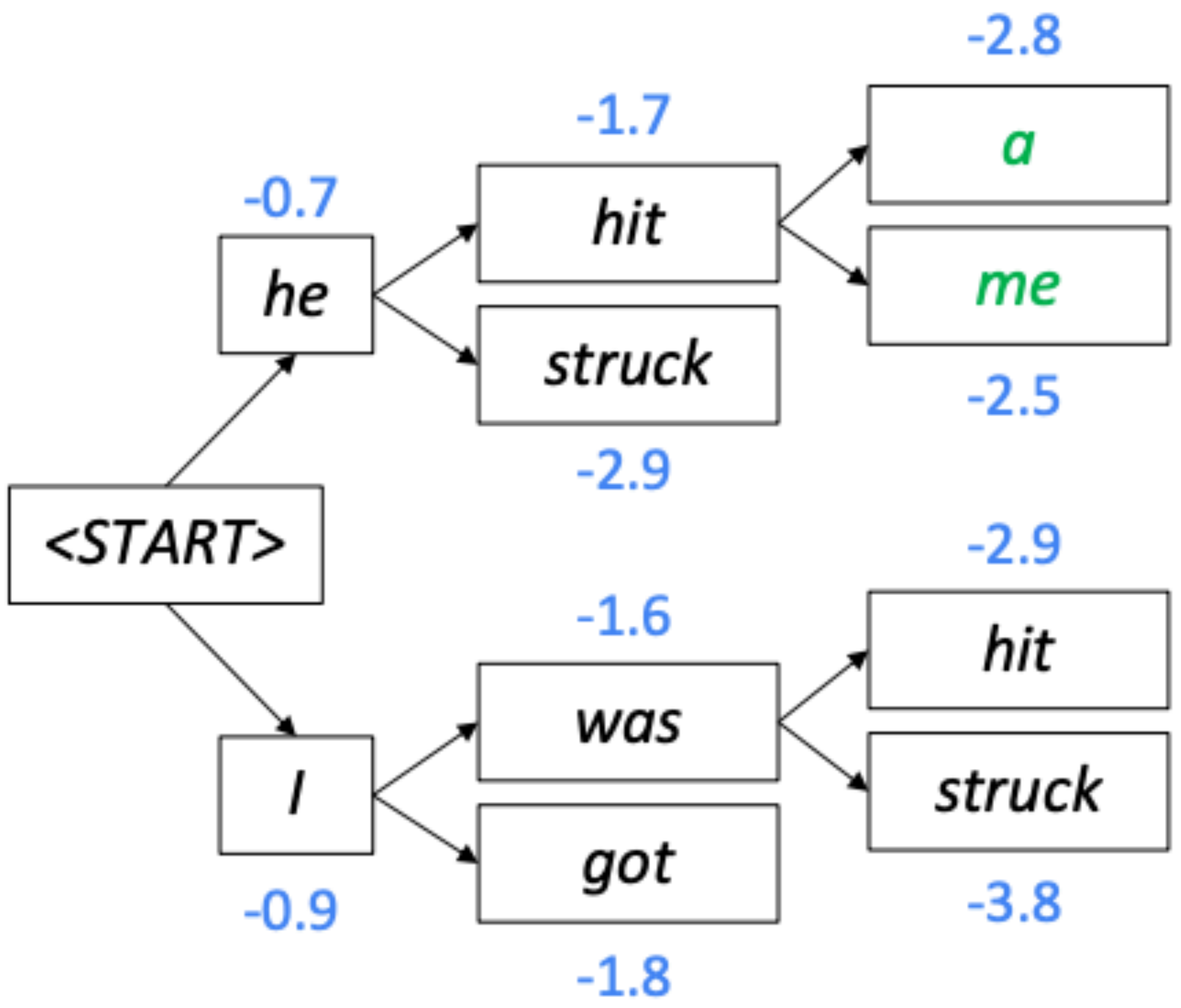


For each of the *k* hypotheses, find top *k* next words and calculate scores

de credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



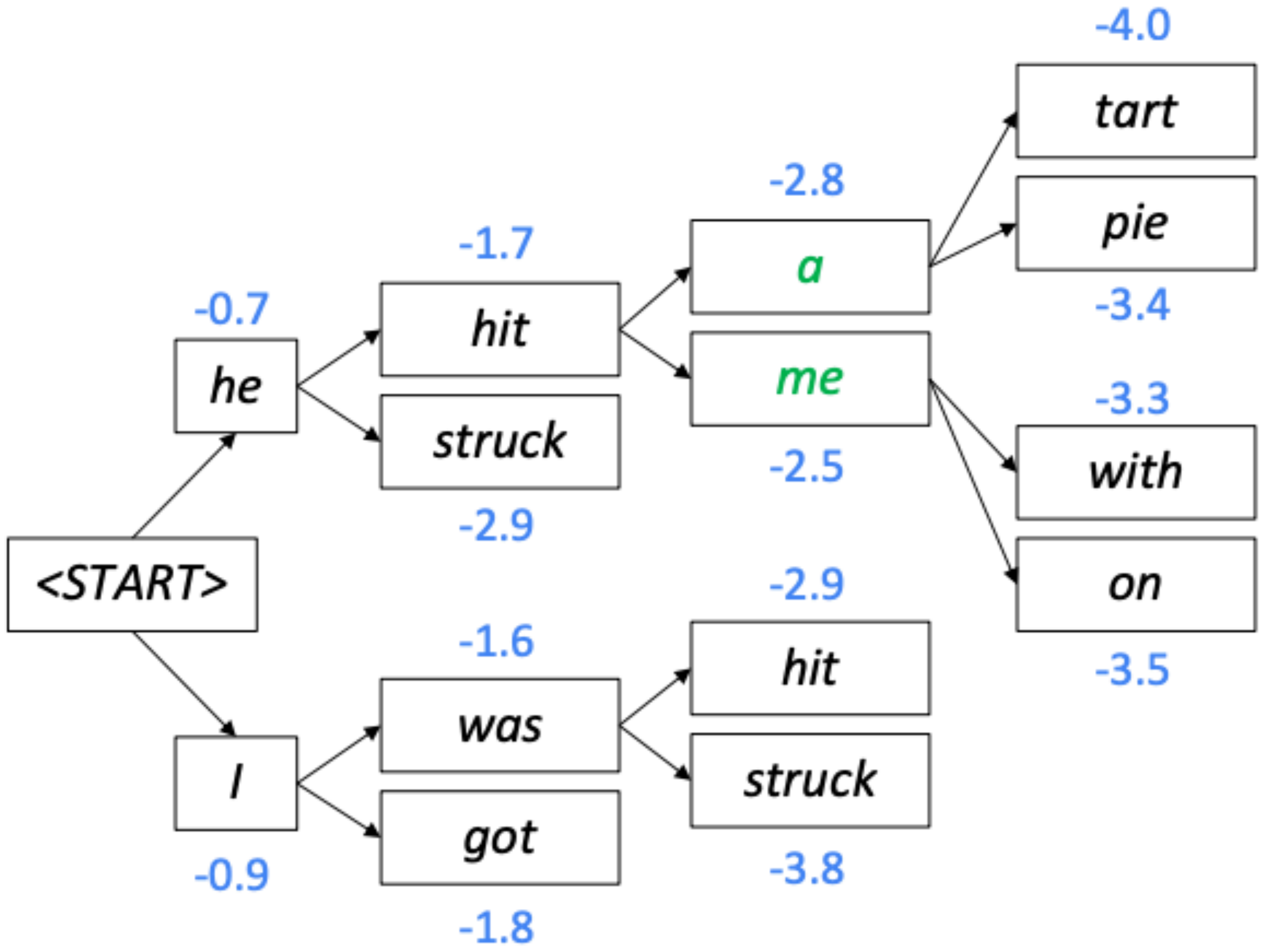
Of these k^2 hypotheses,
just keep k with highest scores

Slide credit: Chris Manning

USC Viterbi

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \dots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \dots, y_{i-1}, x)$

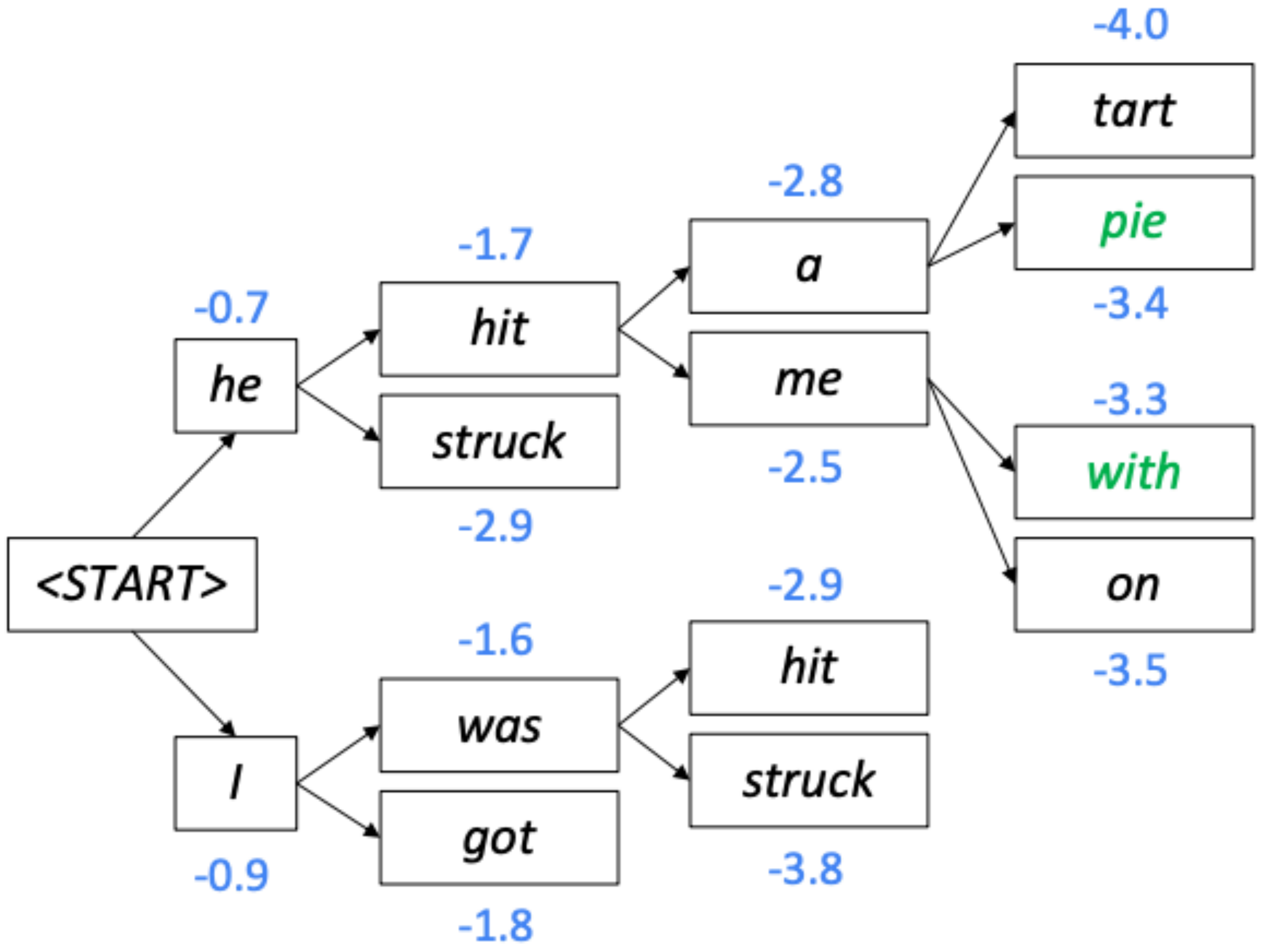


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

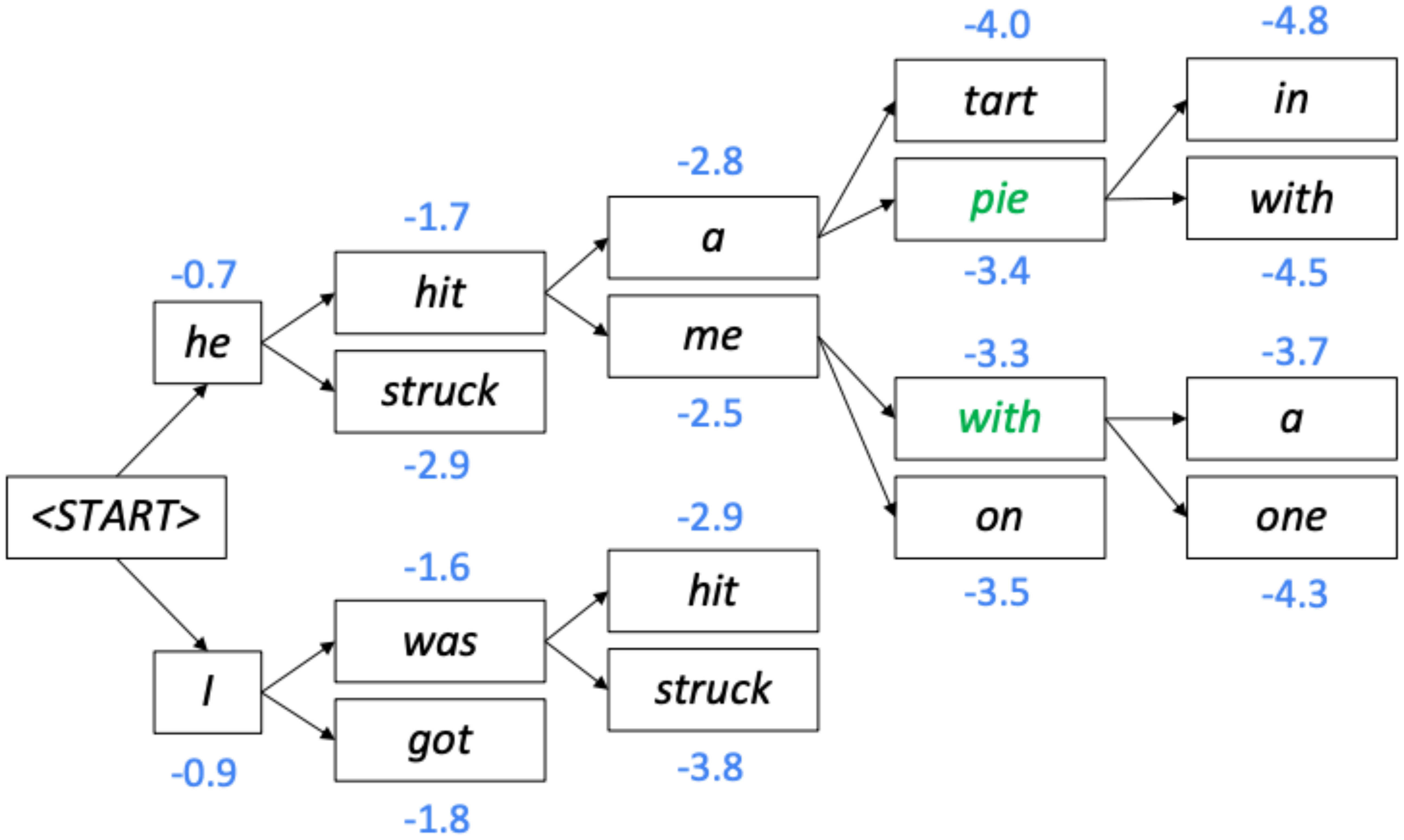


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

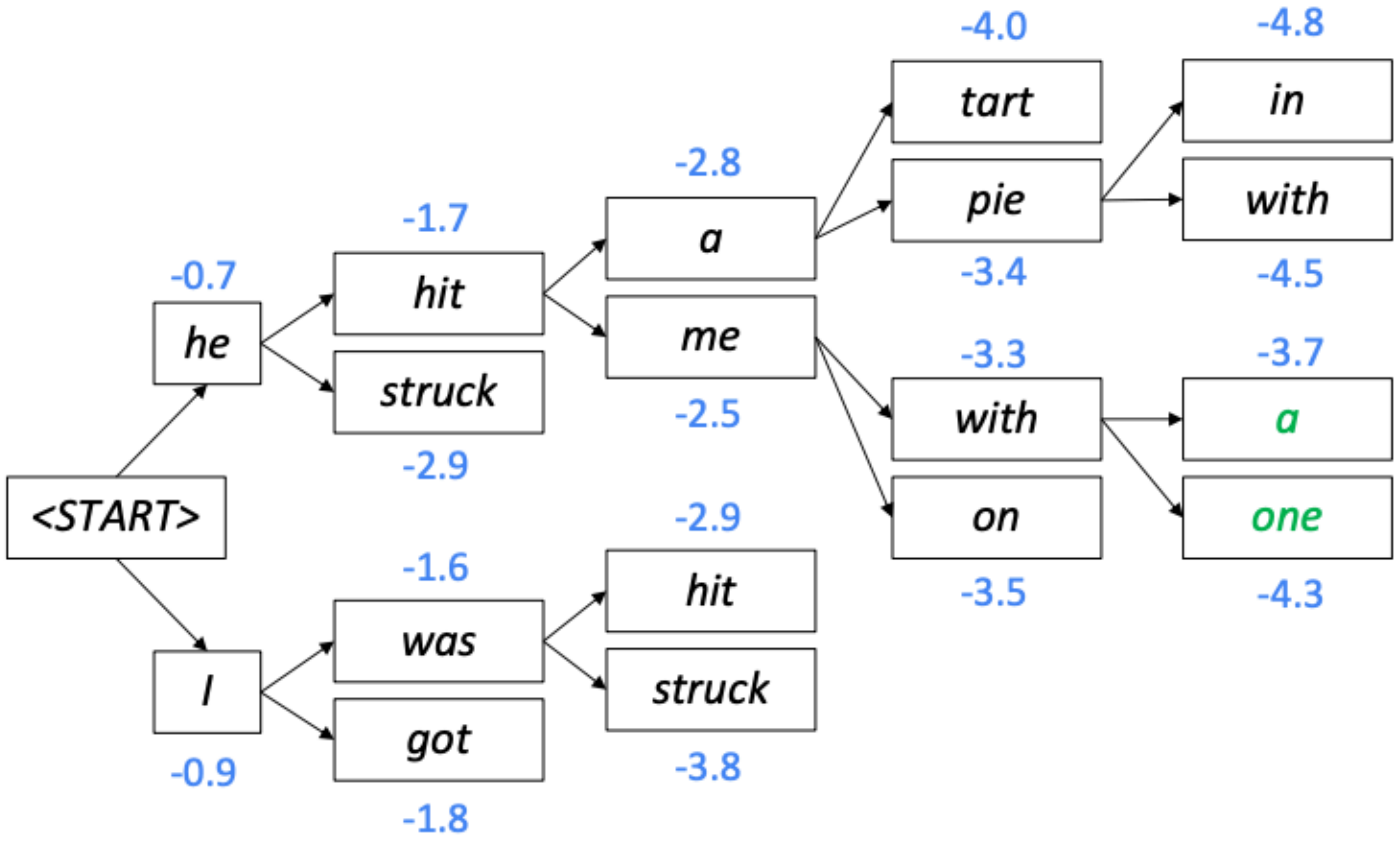


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

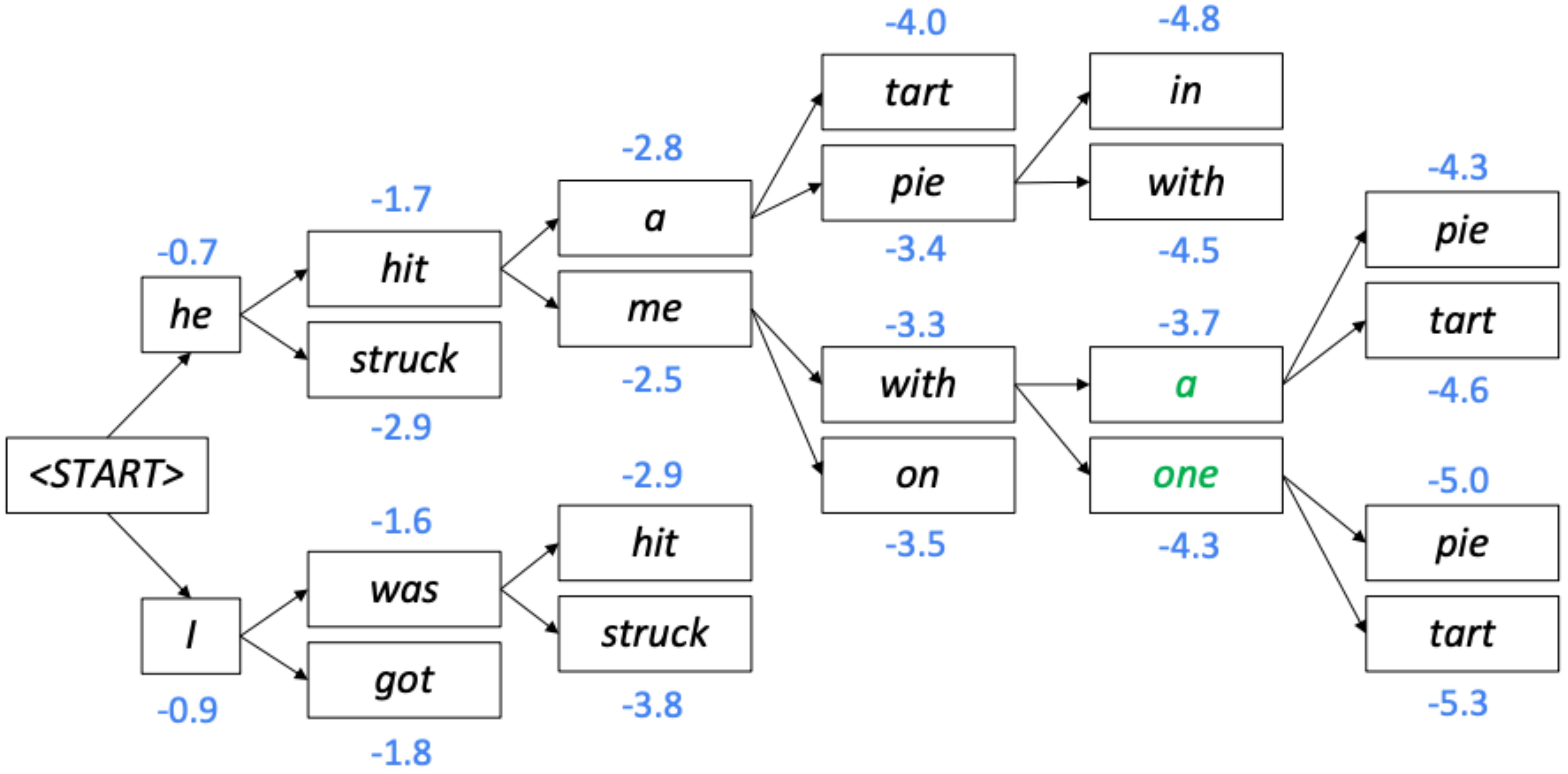


Of these k^2 hypotheses, just keep *k* with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

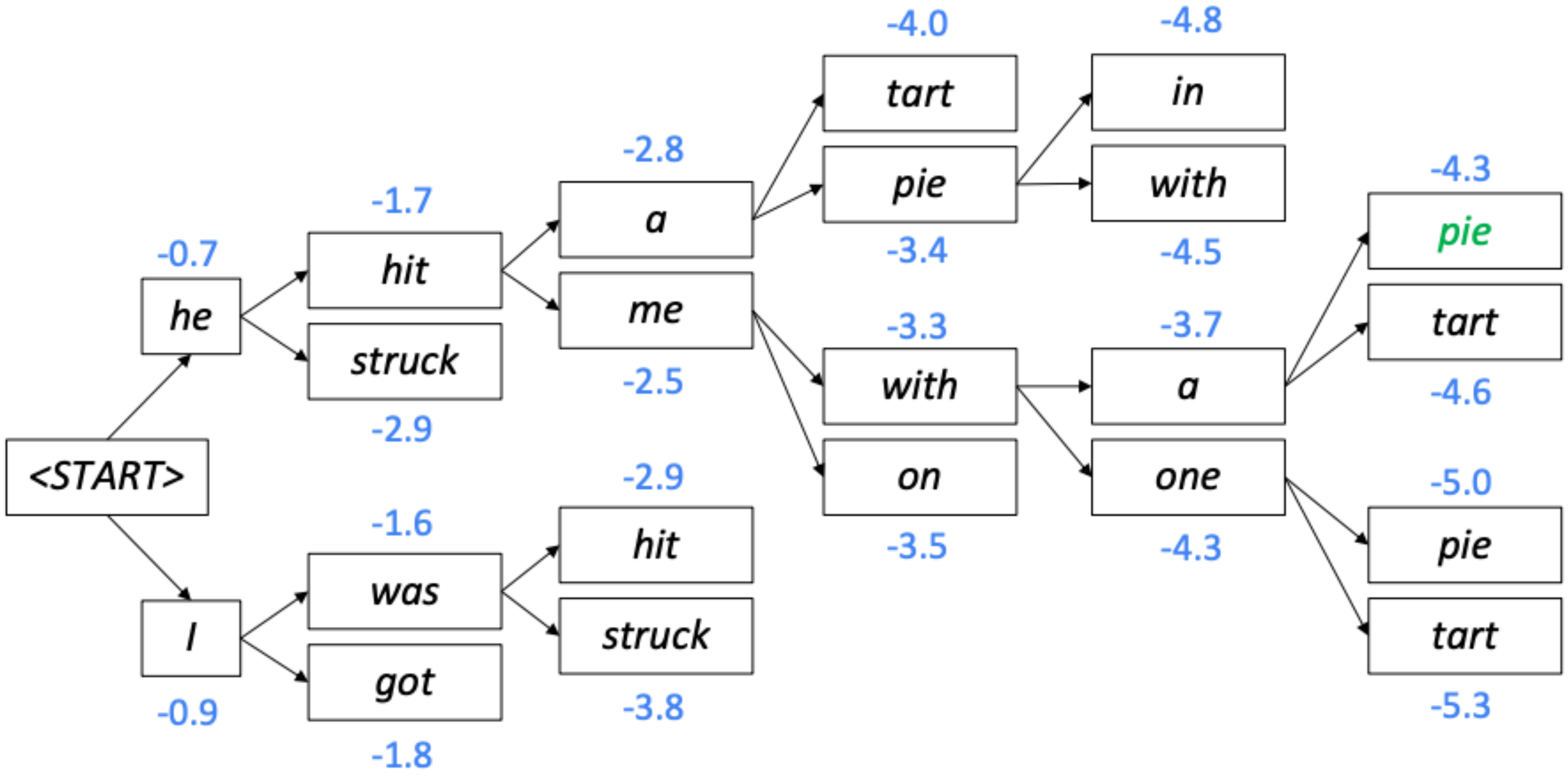


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

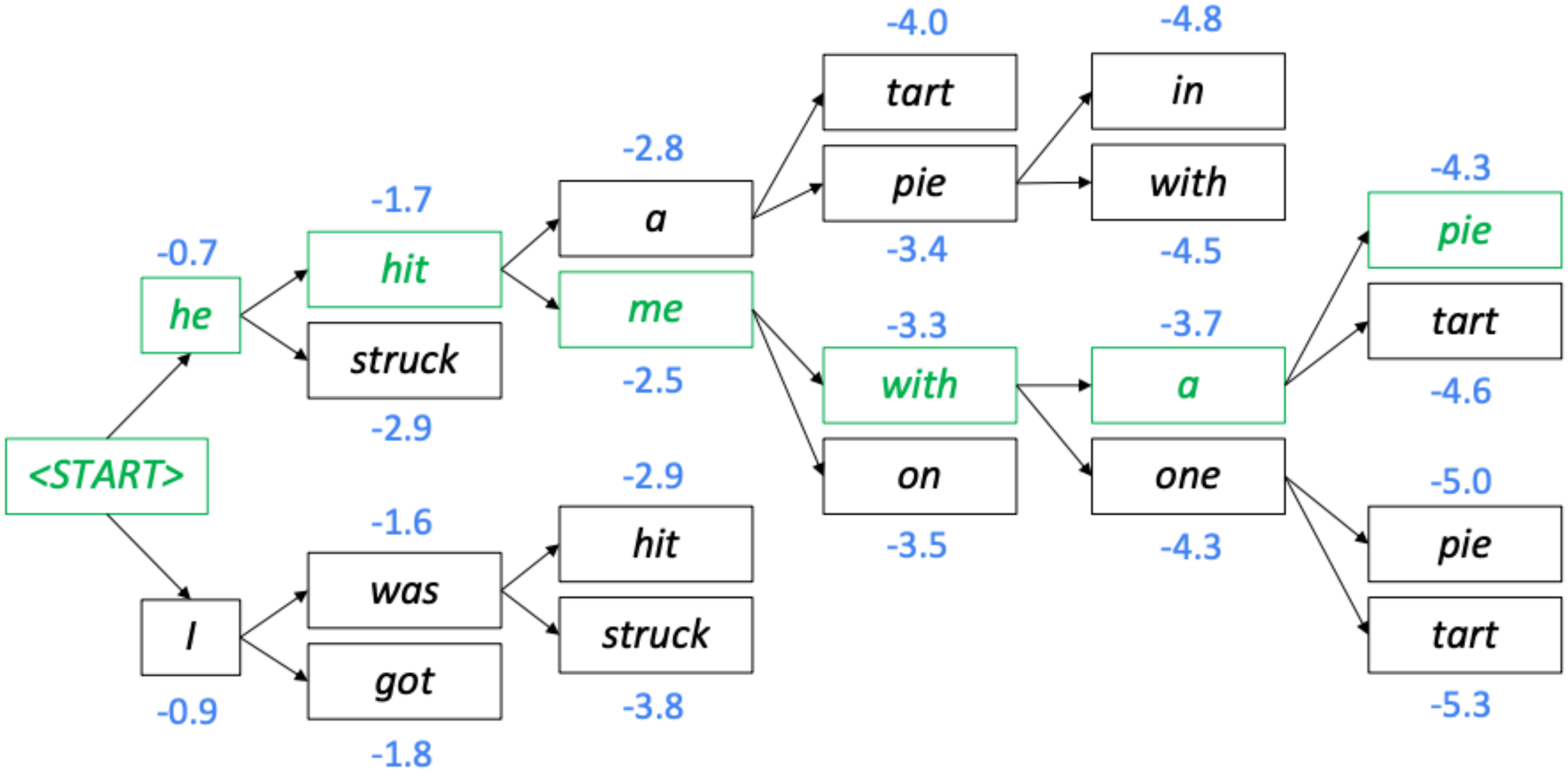


This is the top-scoring hypothesis!

Slide credit: Chris Manning

Beam Search Decoding: Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Backtrack to obtain the full hypothesis

Slide credit: Chris Manning

Beam Search Decoding: Stopping Criterion

Beam Search Decoding: Stopping Criterion

- Greedy Decoding is done until the model produces an </s> token
  - For e.g. <s> he hit me with a pie </s>

USC Viterbi

Beam Search Decoding: Stopping Criterion

- Greedy Decoding is done until the model produces an </s> token
  - For e.g. <s> he hit me with a pie </s>
- In Beam Search Decoding, different hypotheses may produce </s> tokens at different time steps
  - When a hypothesis produces </s>, that hypothesis is complete.
  - Place it aside and continue exploring other hypotheses via beam search.

Beam Search Decoding: Stopping Criterion

- Greedy Decoding is done until the model produces an </s> token
    - For e.g. <s> he hit me with a pie </s>
- In Beam Search Decoding, different hypotheses may produce </s> tokens at different time steps
    - When a hypothesis produces </s>, that hypothesis is complete.
    - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
    - We reach time step T (where T is some pre-defined cutoff), or
    - We have at least n completed hypotheses (where n is pre-defined cutoff)

Beam Search Decoding: Parting Thoughts

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score
- Fix: Normalize by length. Use this to select top one instead

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score
- Fix: Normalize by length. Use this to select top one instead

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

USC Viterbi

Beam Search Decoding: Parting Thoughts

- We have our list of completed hypotheses. Now how to select top one?
- Each hypothesis $y_1, \ldots, y_t$ on our list has a score

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

- Problem with this: longer hypotheses have lower score
- Fix: Normalize by length. Use this to select top one instead

$$\frac{1}{t} \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

But this is expensive!

Maximization Based Decoding

Maximization Based Decoding

- Either greedy or beam search

Maximization Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive

Maximization Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive
- Another key issue:

USC Viterbi

Maximization Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive
- Another key issue:

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México...**

Holtzmann et al., 2020

Maximization Based Decoding

- Either greedy or beam search
- Beam search can be more effective with large beam width, but also more expensive
- Another key issue:

Generation can be bland or repetitive (also called degenerate)

Context: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Continuation: The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the **Universidad Nacional Autónoma de México (UNAM)** and **the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México/ Universidad Nacional Autónoma de México...**

Holtzmann et al., 2020

Degenerate Outputs

Holtzmann et al., 2020

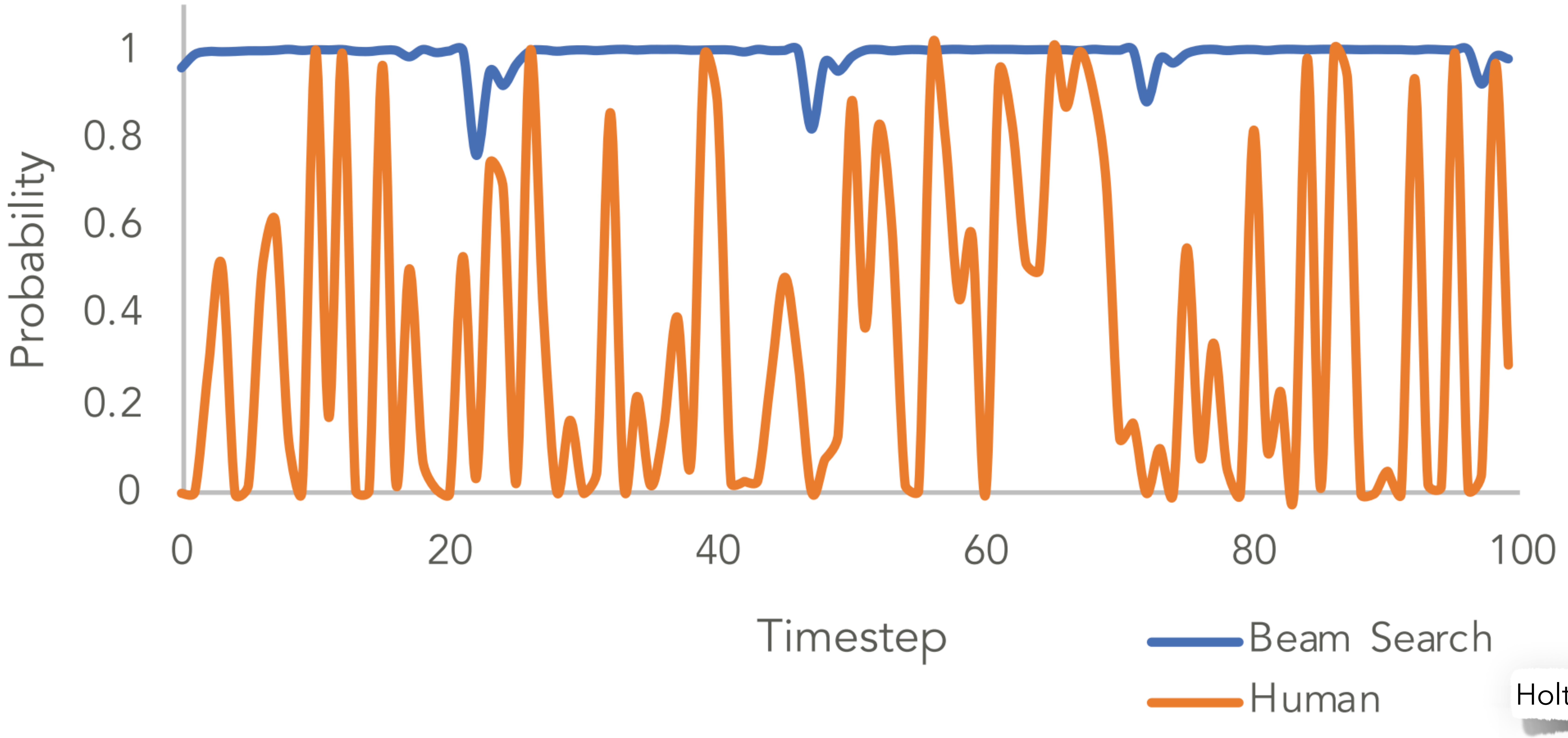
USC Viterbi

Degenerate Outputs

Holtzmann et al., 2020

Scale doesn't solve this problem: even a 175 billion parameter LM still repeats when we decode for the most likely string.

Why does repetition happen?

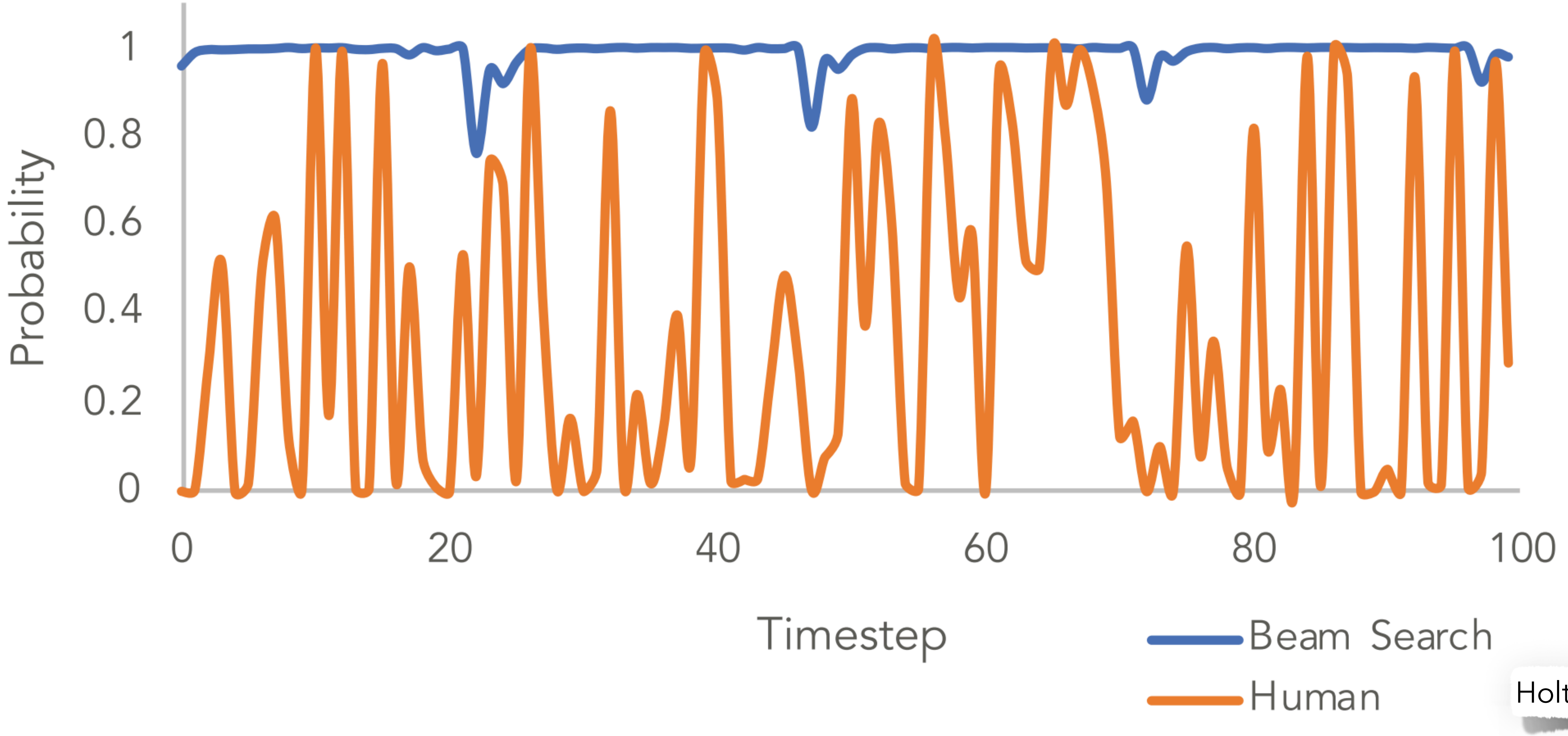


Holtzmann et al., 2020

USC Viterbi

Why does repetition happen?

- Probability amplification due to maximization based decoding

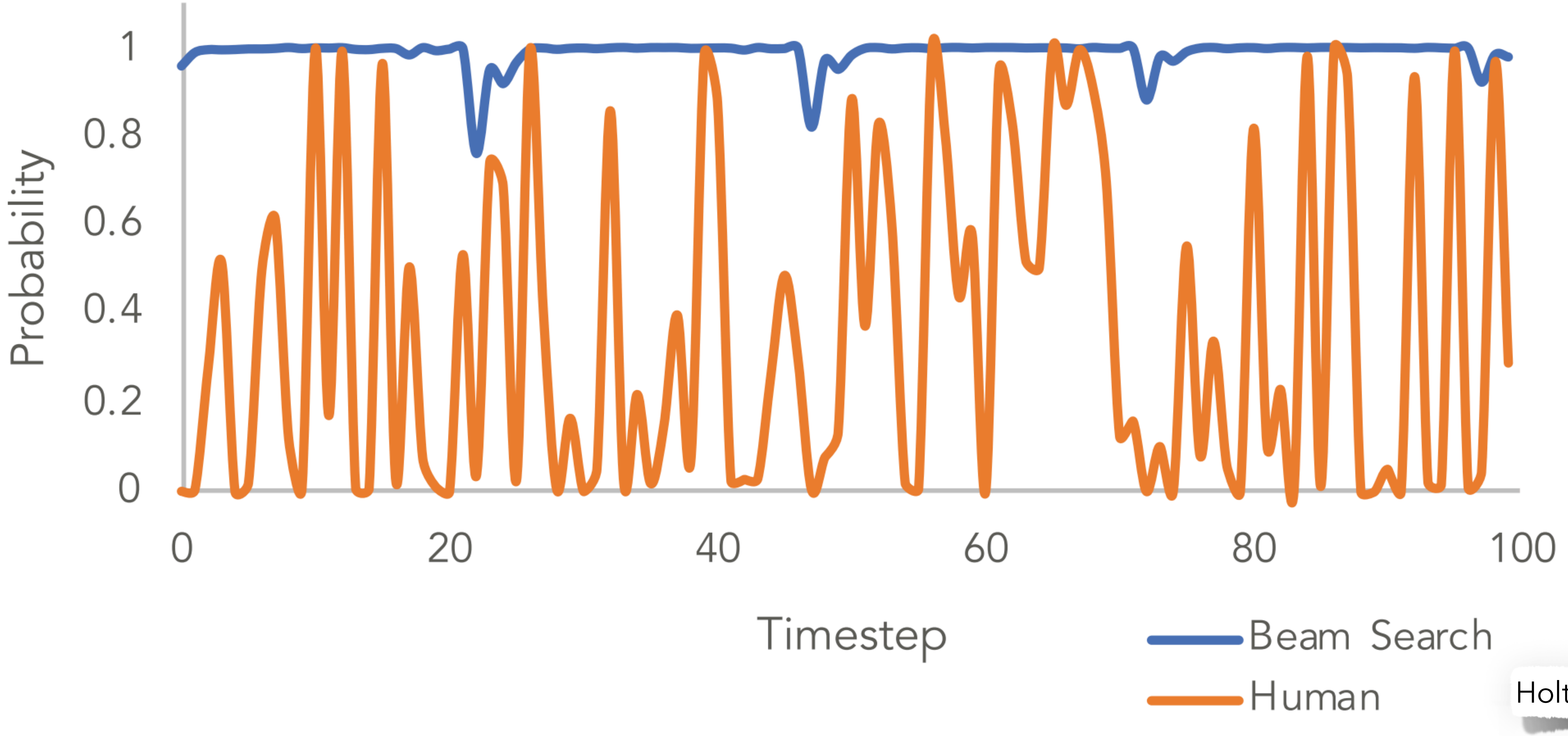


Holtzmann et al., 2020

 USC Viterbi

Why does repetition happen?

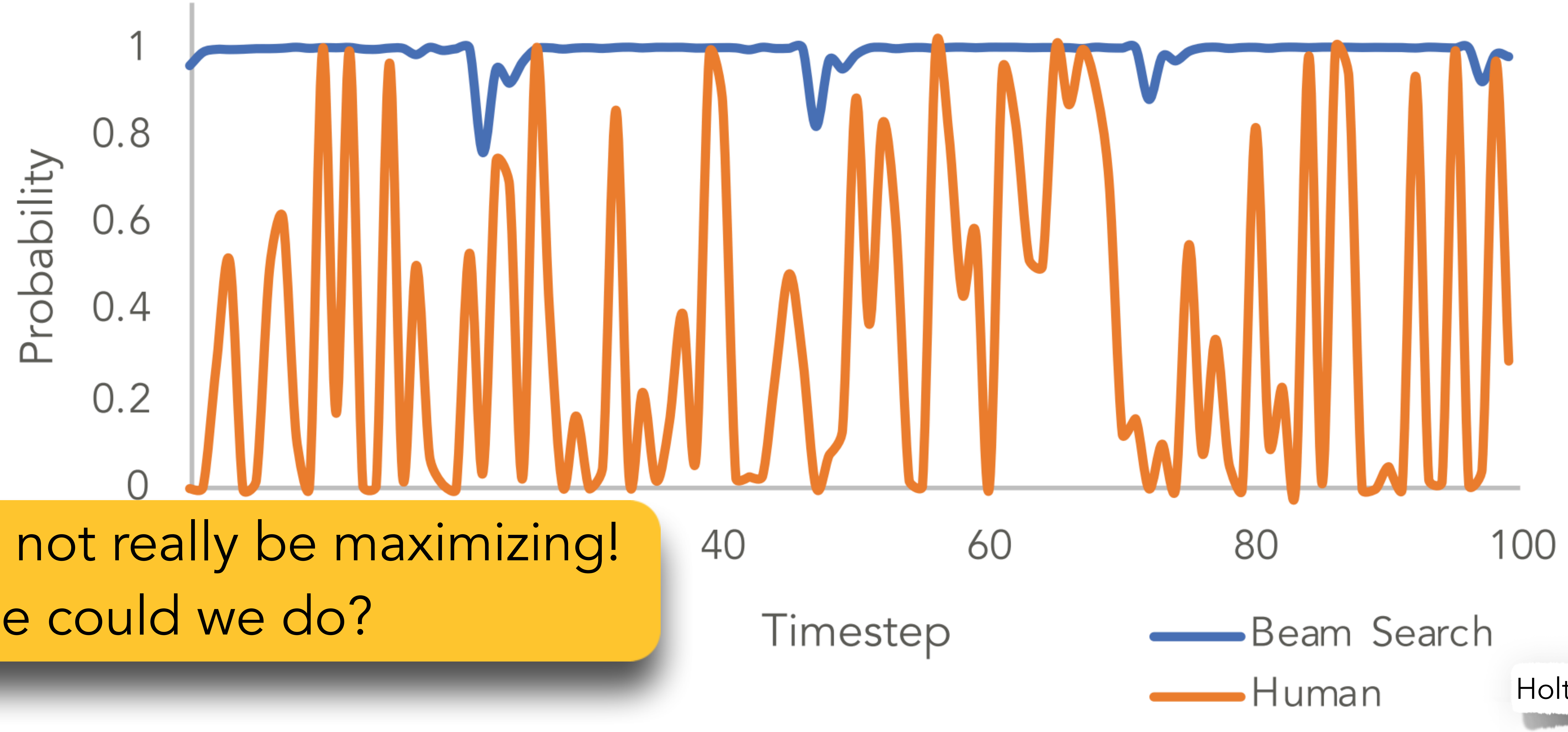
- Probability amplification due to maximization based decoding
- Generation fails to match the uncertainty distribution for human written text



Holtzmann et al., 2020

Why does repetition happen?

- Probability amplification due to maximization based decoding
- Generation fails to match the uncertainty distribution for human written text



Perhaps we should not really be maximizing!
What else could we do?

Holtzmann et al., 2020

Solution: Don't Maximize, Pick a Sample

Solution: Don't Maximize, Pick a Sample

- Sample a token from the distribution of tokens.

Solution: Don't Maximize, Pick a Sample

- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution

Solution: Don't Maximize, Pick a Sample

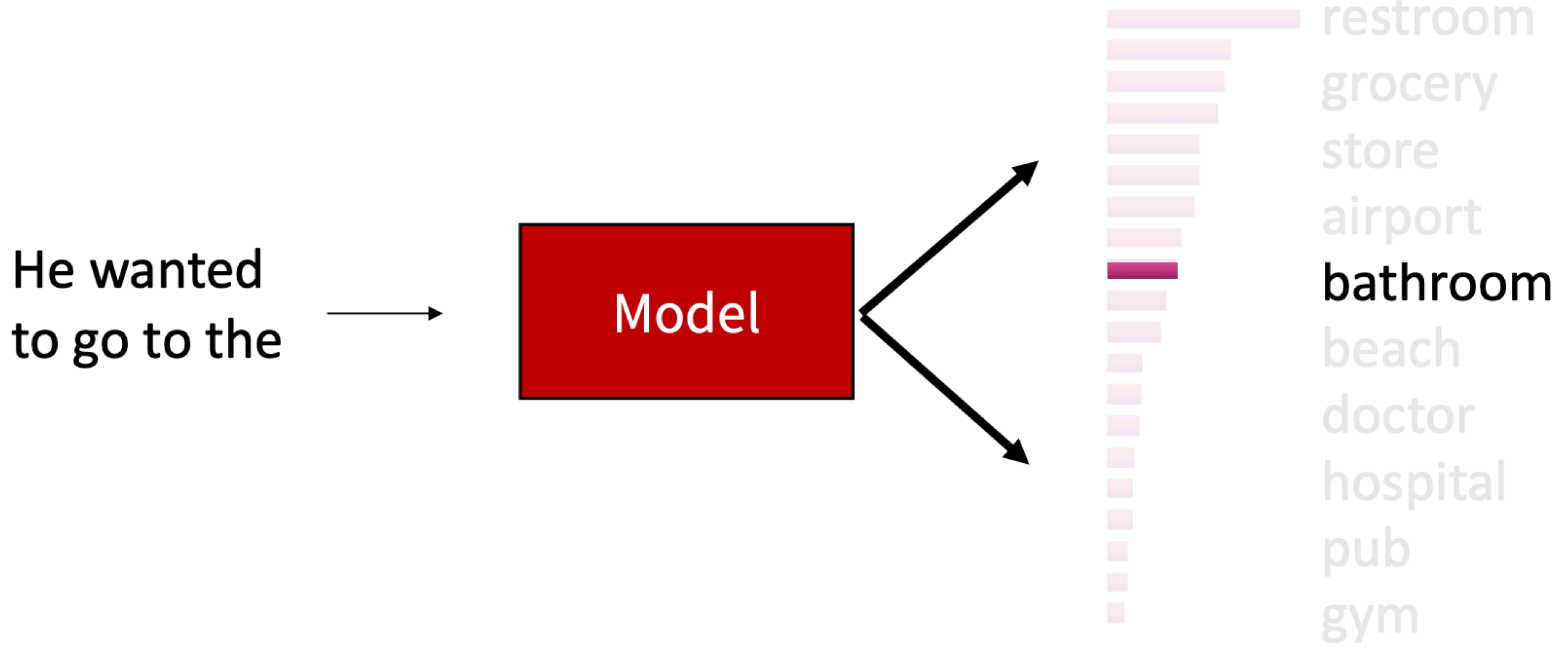
- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution
  - Respects the probabilities, without going just for the maximum probability option

Solution: Don't Maximize, Pick a Sample

- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution
    - Respects the probabilities, without going just for the maximum probability option
    - Or else, you would get something meaningless

Solution: Don't Maximize, Pick a Sample

- Sample a token from the distribution of tokens.
- But this is not a random sample, it is a sample for the learned model distribution
  - Respects the probabilities, without going just for the maximum probability option
  - Or else, you would get something meaningless
  - Many good options which are not the maximum probability!



Modern Generation: Sampling

Next Class!