Lecture 13:
Natural Language Generation Algorithms

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Oct 22, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements + Logistics

- HW3 Released
- Next Week: HW2 grades out
- Next Wed: Project Progress Report Due
- Today: Quiz 3

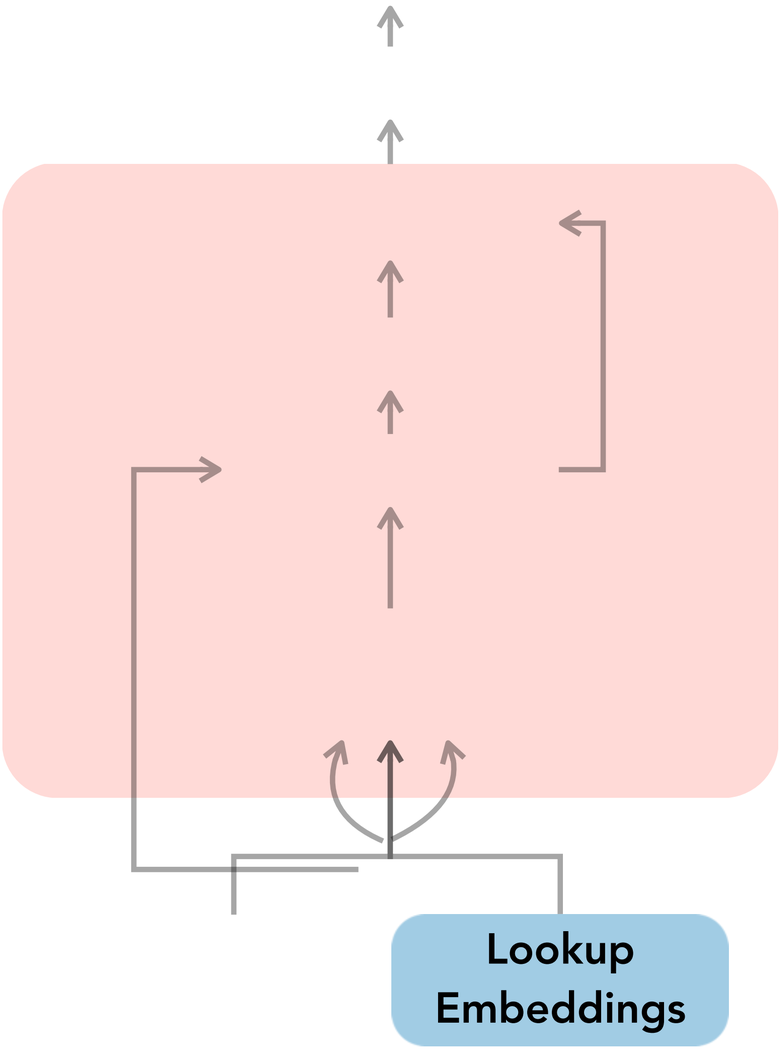
Lecture Outline

- Recap: Tokenization
- Natural Language Generation
  - Classic Inference Algorithms: Greedy and Beam Search
- Modern Generation: Sampling

Recap: Tokenization in Modern LMs

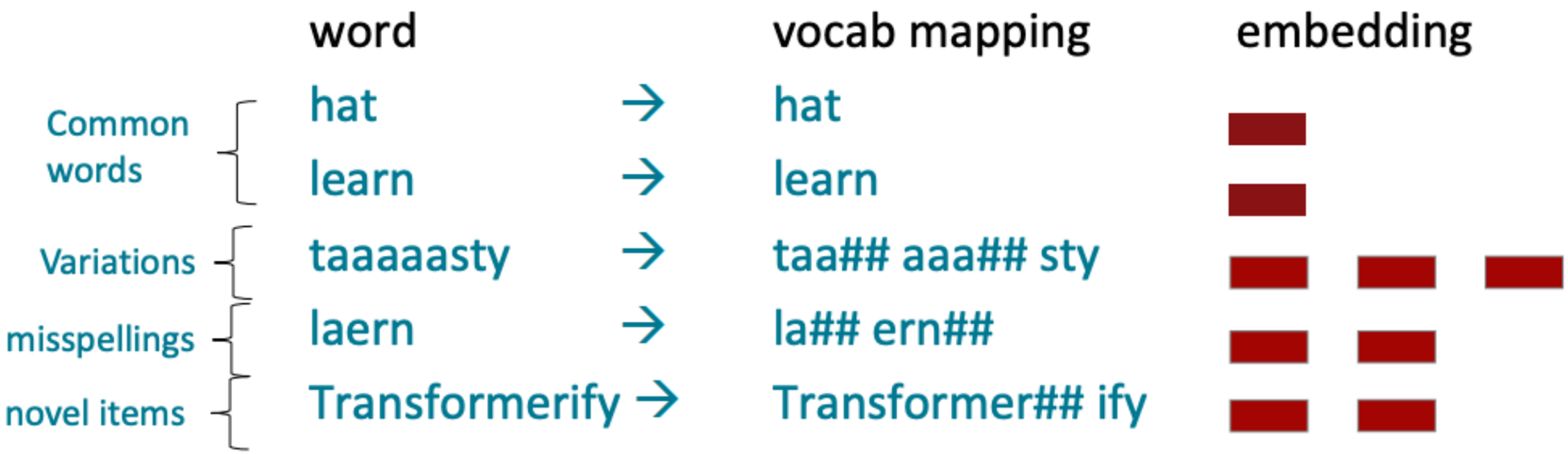
The Input Layer

- 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



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.
  - Llama 3 uses a tokenizer with a vocabulary of 128K tokens



Byte-pair encoding

- Byte-pair encoding merges 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! However, not a parametric function
        - 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
- **WordPiece Modeling:** Similar to BPE
    - Each word is initially split by adding a prefix (##) to all the characters inside a word

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>

Vocabulary

d	e	i	l	n	o	s	t	w
es	est							

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)	

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 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/

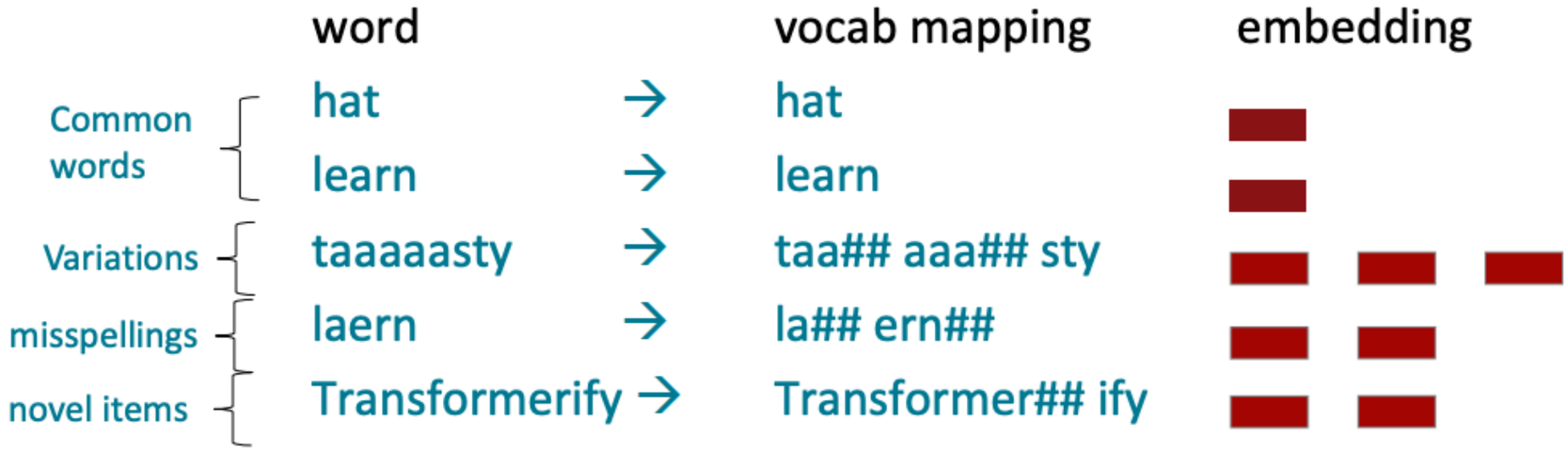
WordPiece Modeling

- 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



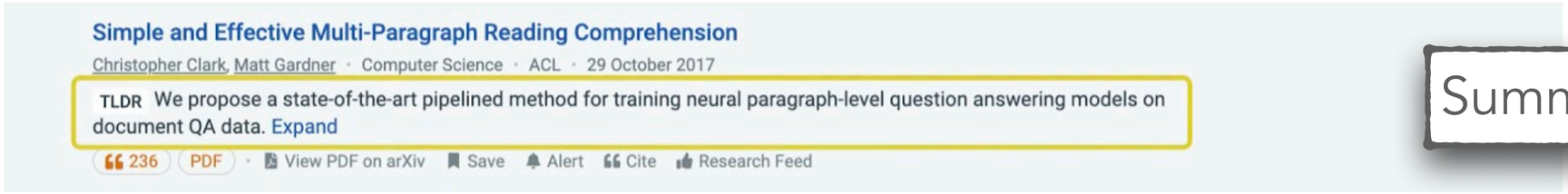
Quiz 3
Password: attention

Natural Language Generation

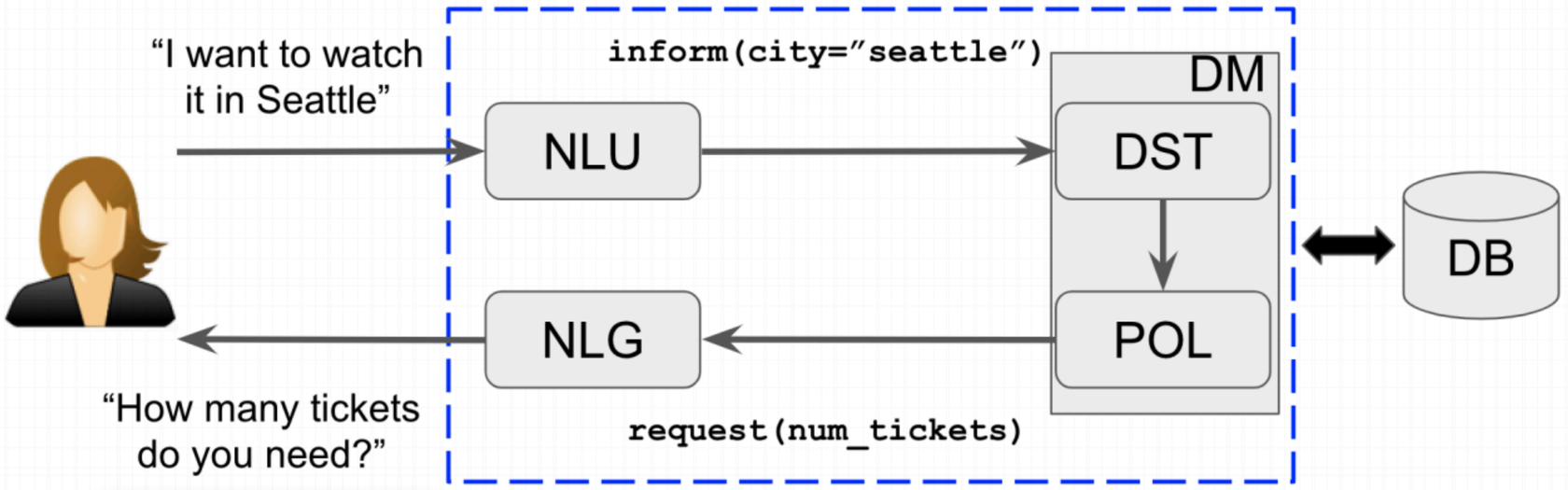
Natural Language Generation

- Natural Language Generation (NLG) is the workhorse of many classic and novel applications
  - AI Assistants, like ChatGPT
  - Translators and Summarizers
- NLG and Natural language understanding (NLU) are two sides of the same coin
  - Natural language understanding: Learning representations that perform well on downstream tasks
  - To generate good language, one needs to understand language
  - If you understand language, you should be able to generate it (with some effort)

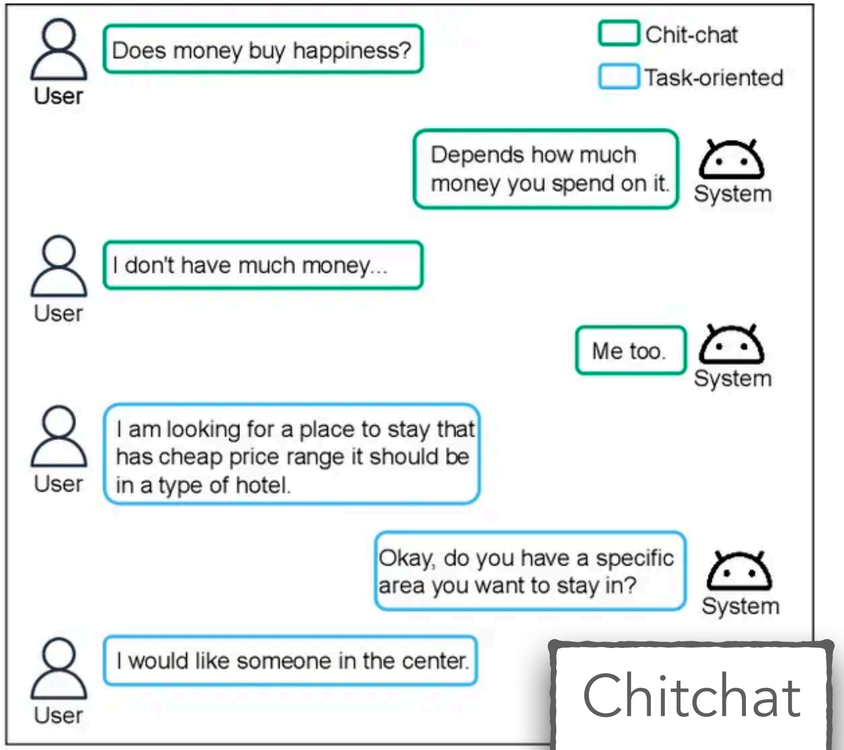
NLG Use Cases



Summarization



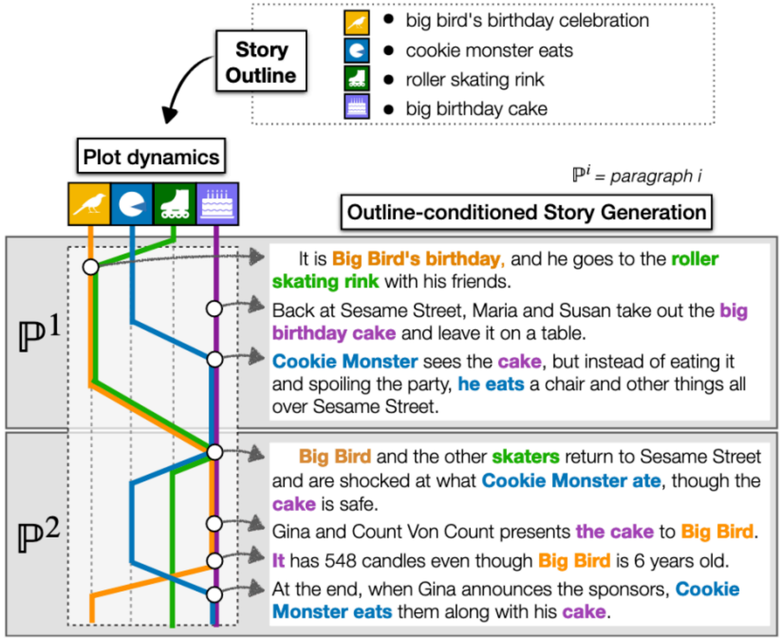
Task-driven Dialog



Chitchat Dialog

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

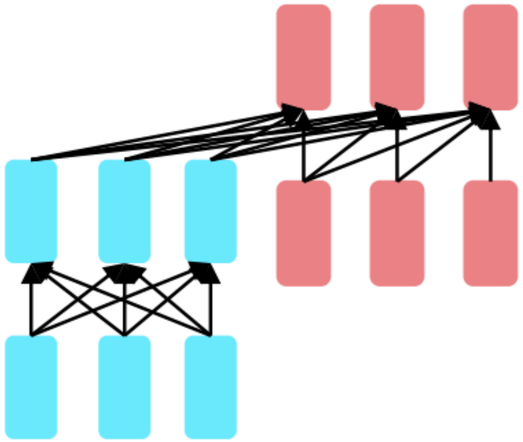
Open-ended generation: the output distribution still has high freedom.

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

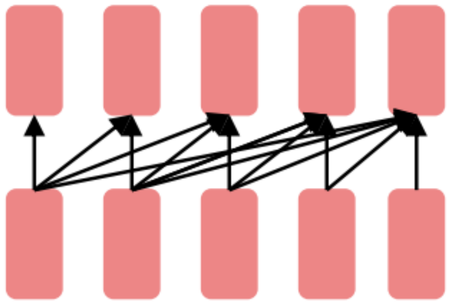
Broad Spectrum of NLG Tasks

Less Open-Ended — **More Open-Ended**

Machine Translation — Summarization — Task-driven Dialog — Chitchat Dialog — Story Generation



Encoder-Decoders



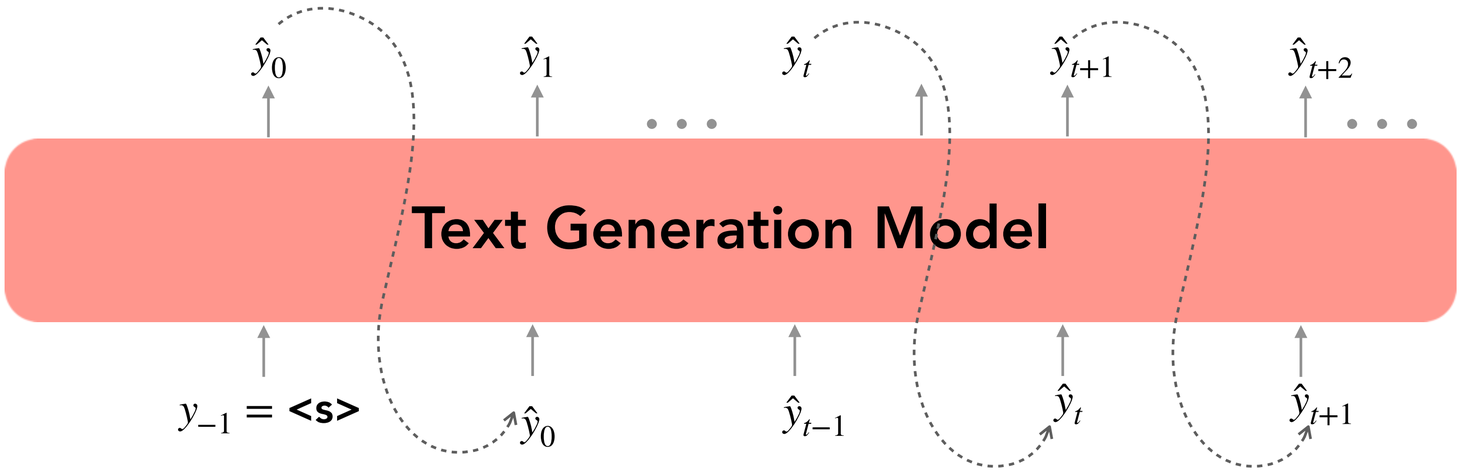
Decoders

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 $S = f_\theta(y_{<t}) \in \mathbb{R}^V$

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

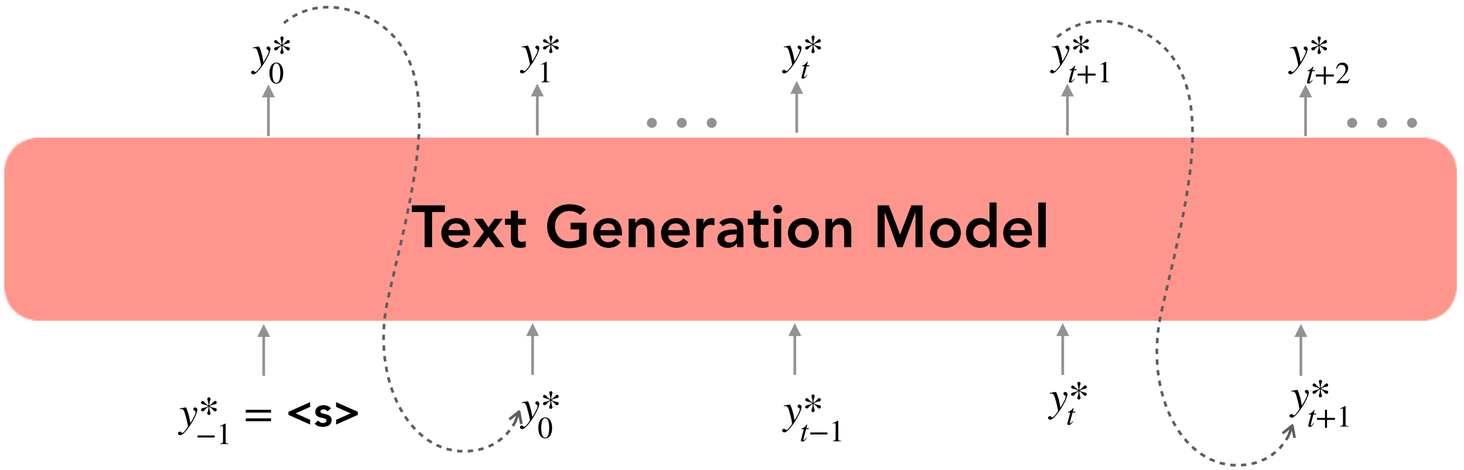


Language Generation: Training

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

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

- 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

- 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

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

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

$$g = \arg\max$$

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

Classic Inference Algorithms: Greedy and Beam Search

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

- 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

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

- We could try computing all possible sequences y
  - This means that on each step t of the decoder, we will be tracking V^t possible partial translations, where V is the vocab size
  - This $O(V^T)$ complexity is far too expensive!

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

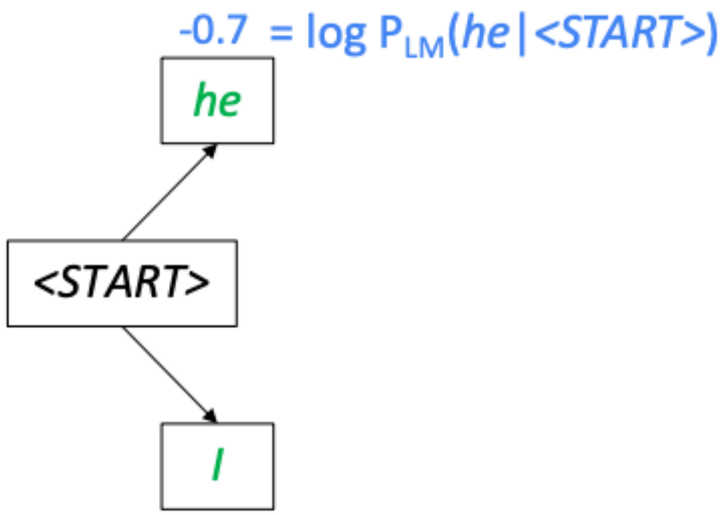
<START>

Calculate prob dist of next word

Slide credit: Chris Manning

Beam Search Decoding: Example

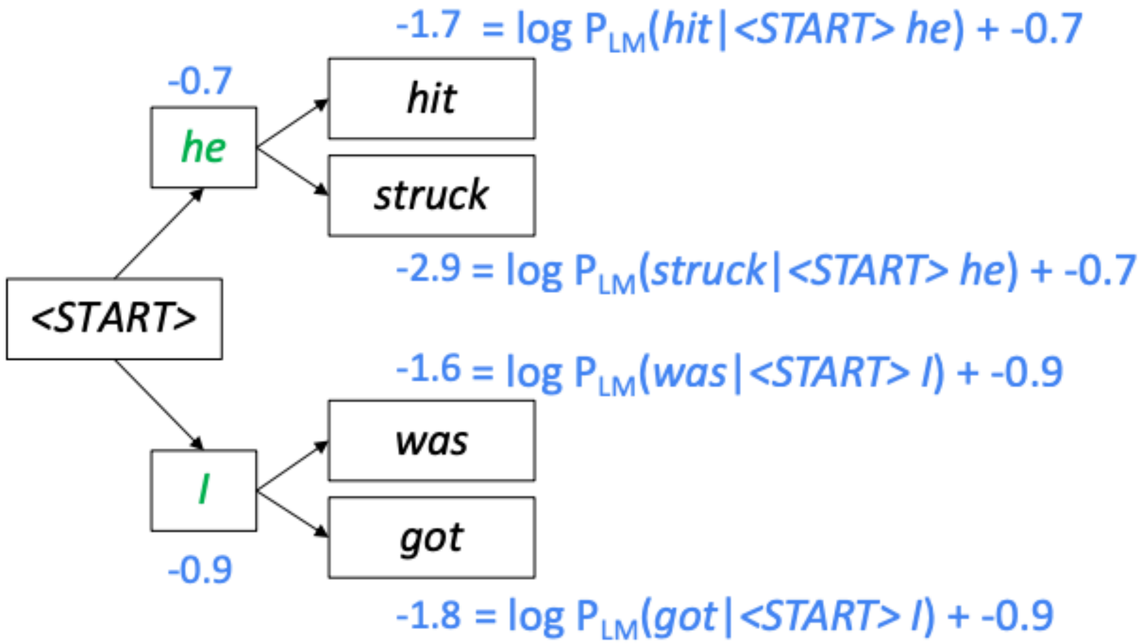
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam Search Decoding: Example

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

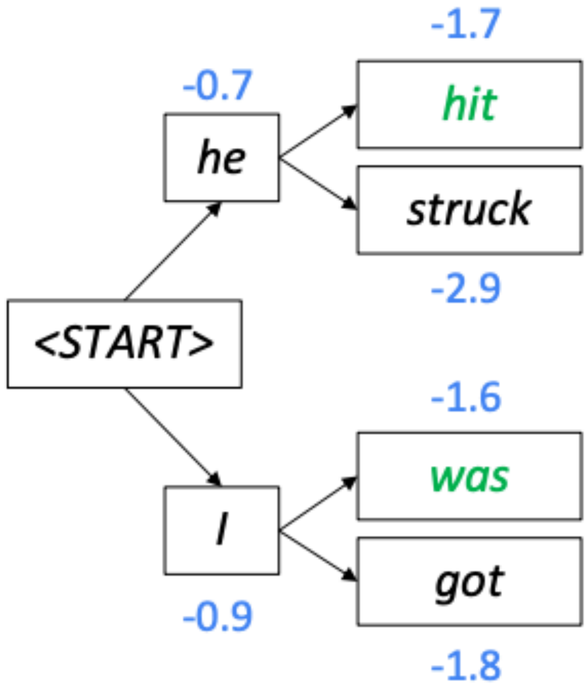


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

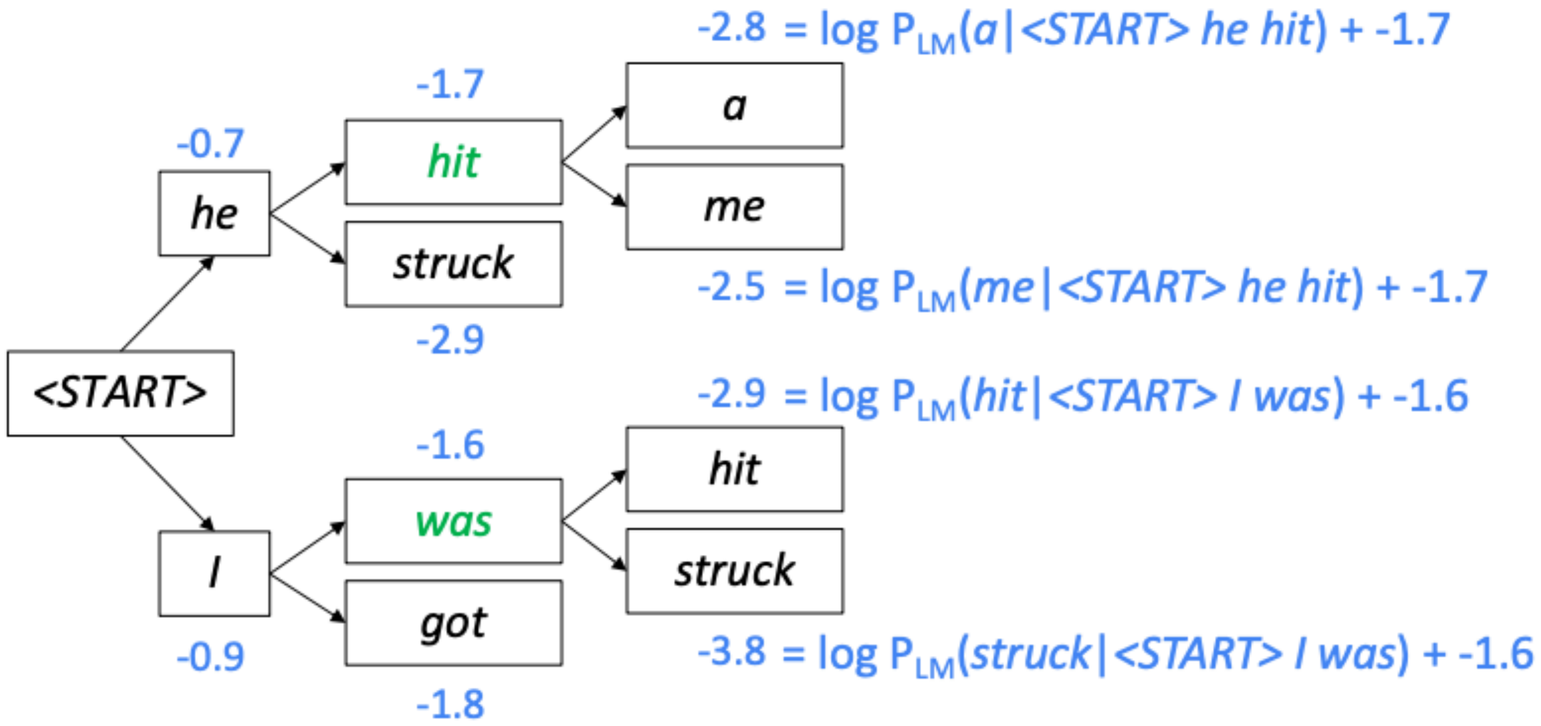


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

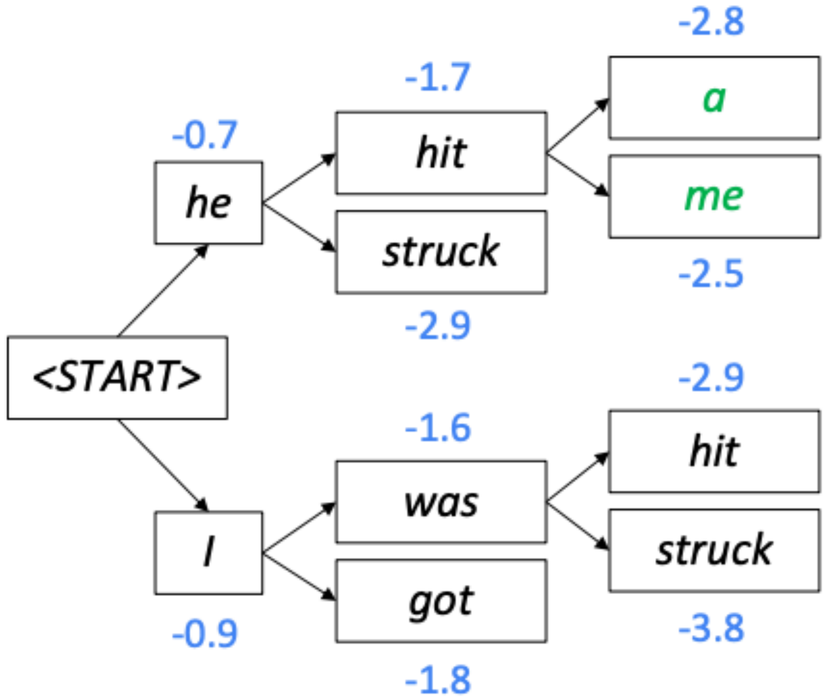


For each of the *k* hypotheses, find top *k* next words and calculate scores

de credit: Chris Manning

Beam Search Decoding: Example

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

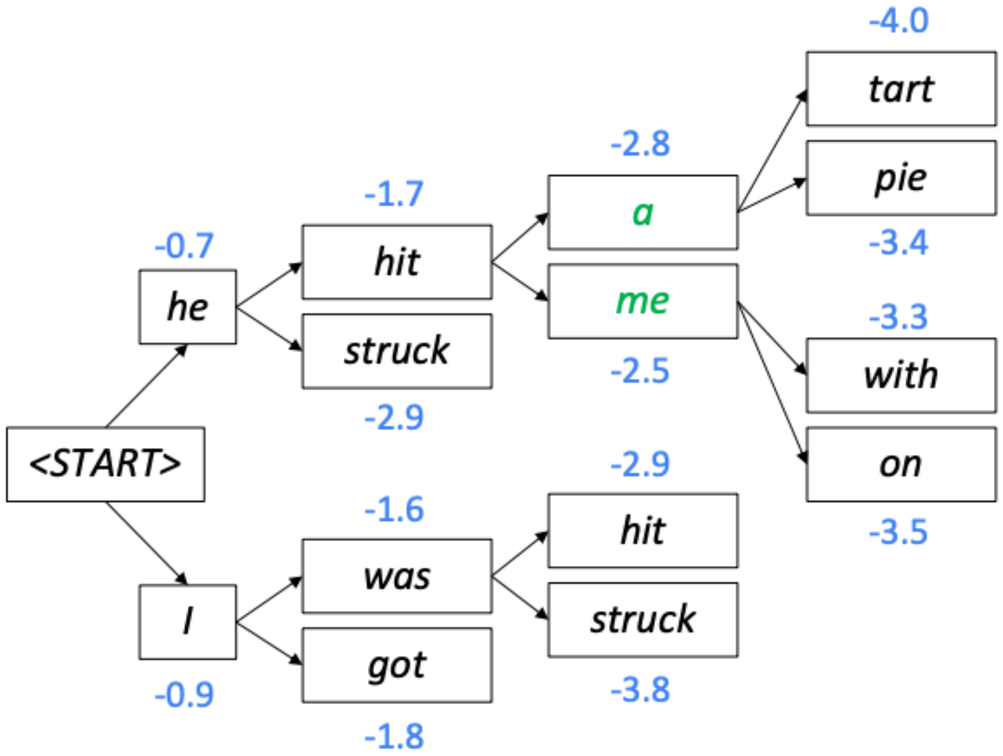


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

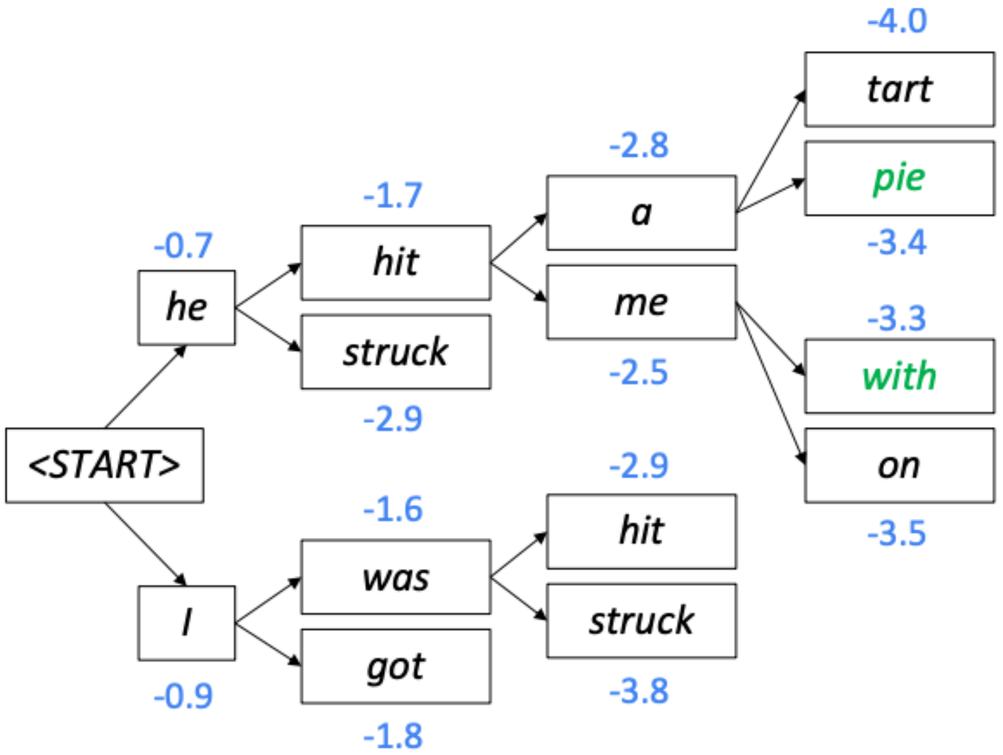


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

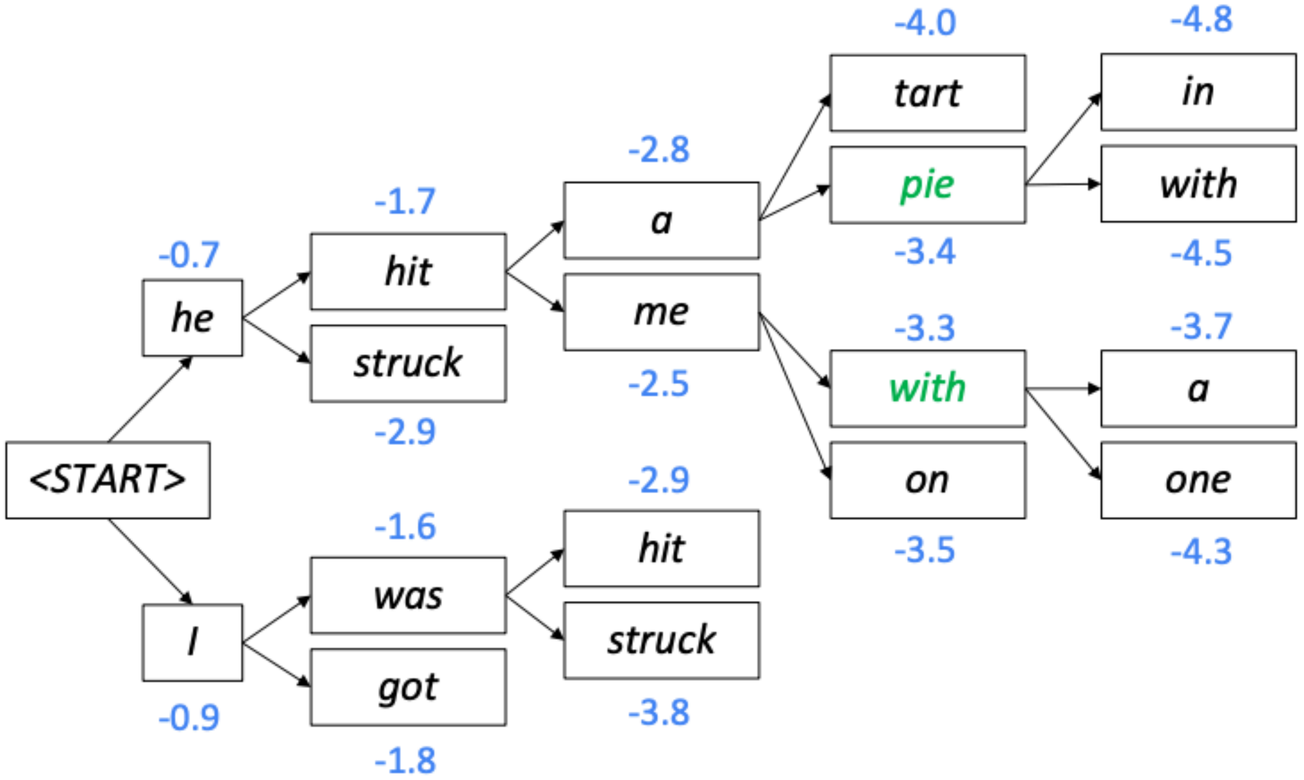


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

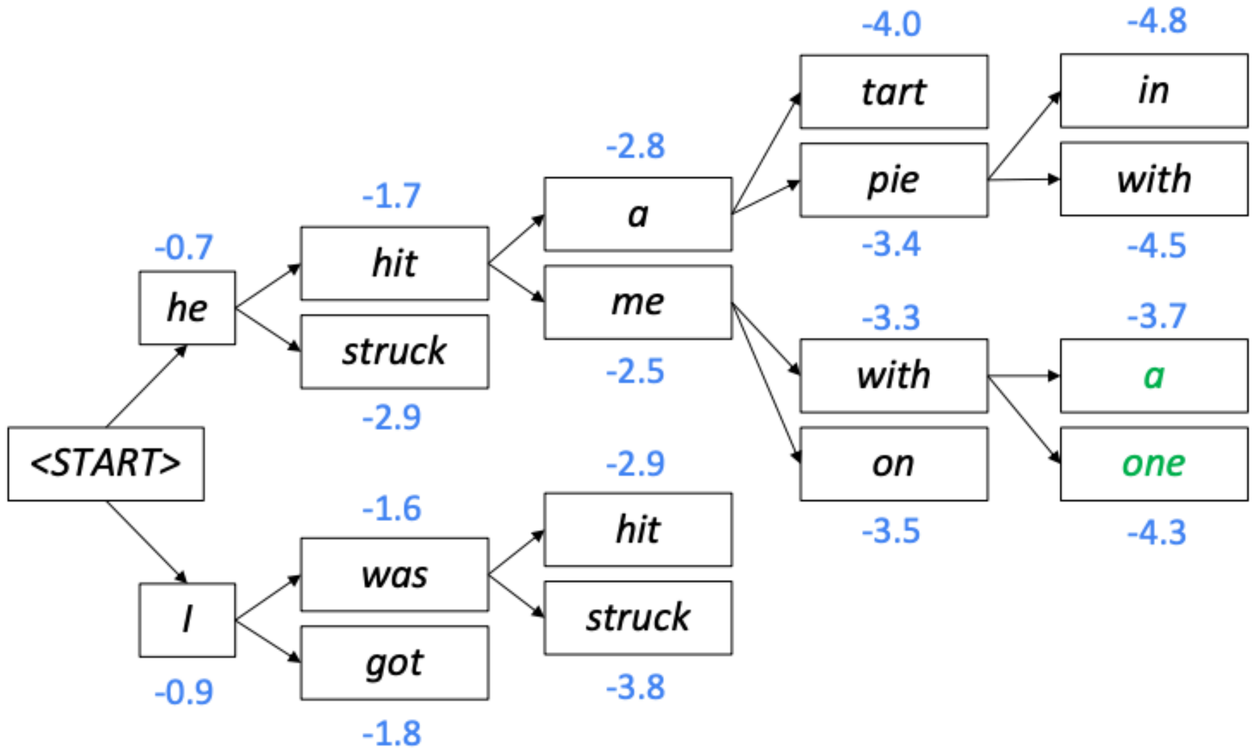


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

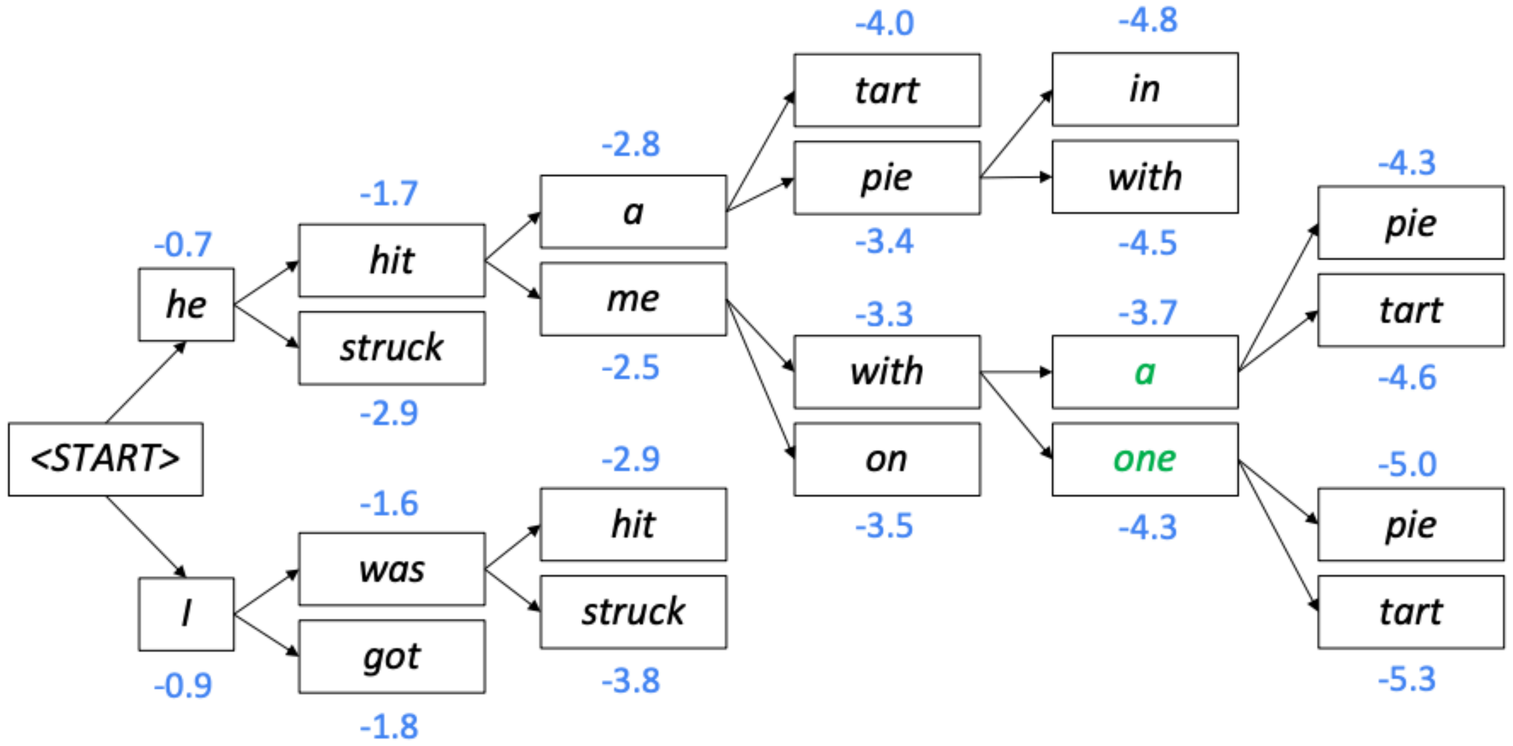


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

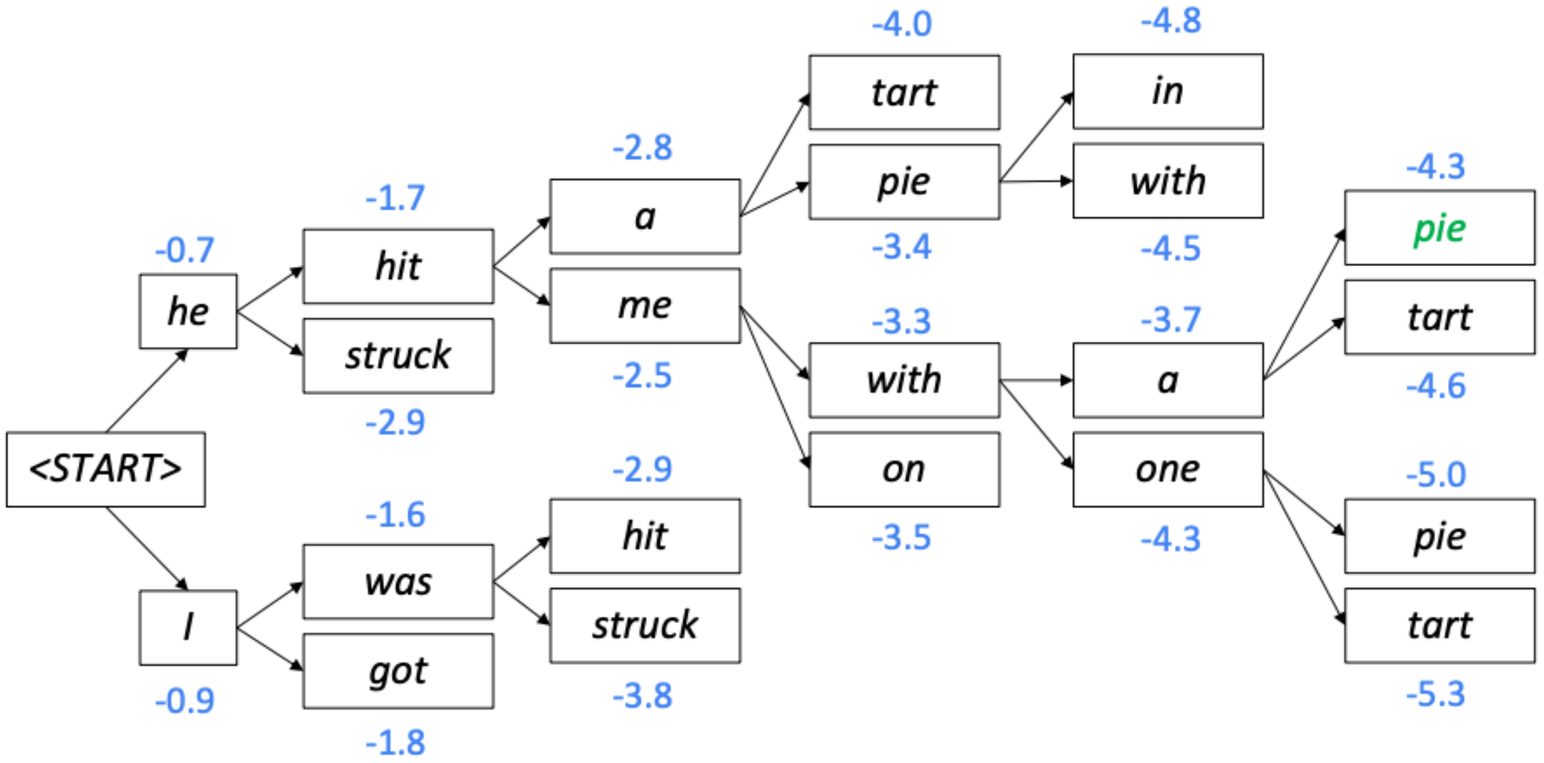


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

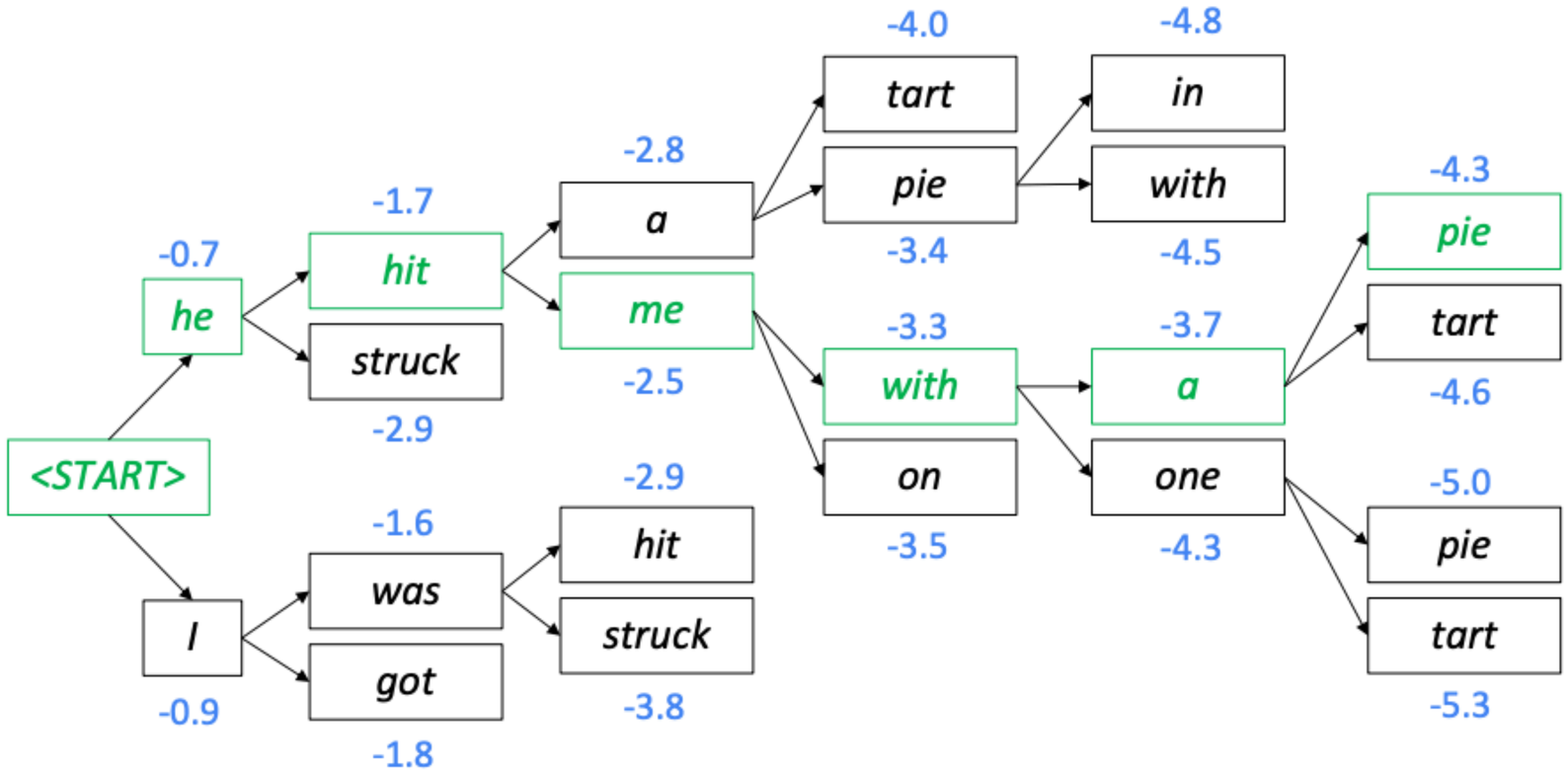


This is the top-scoring hypothesis!

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

- 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

- 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 scores
- 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

- 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