Lecture 14:
Natural Language Generation (contd.)

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Oct 27, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements + Logistics

- Quiz 3 grades out. HW2 grades this week, apologies for the delay
  - Once again, you all deserve a TA, please let USC know!
- Wed: Project Progress Report Due
- Starting Next Week: **Paper Discussions** [10% of grade]
  - 3 weeks of paper discussions following lectures on advanced topics
  - Before class, read 3 modern classic papers per week (see website)
  - In class, discuss each paper with (randomly assigned) group of 4-5 students
    - 20 mins per paper within group
    - 15 mins per paper with whole class
      - I will randomly pick a student to summarize a group discussion
  - After class, turn in your review of the 3 papers, ranking them

Lecture Outline

- Quiz 3 Answers
- Recap: Classic Inference Algorithms: Greedy and Beam Search
- Modern Generation: Sampling
- Generative Evaluation Metrics / Methods

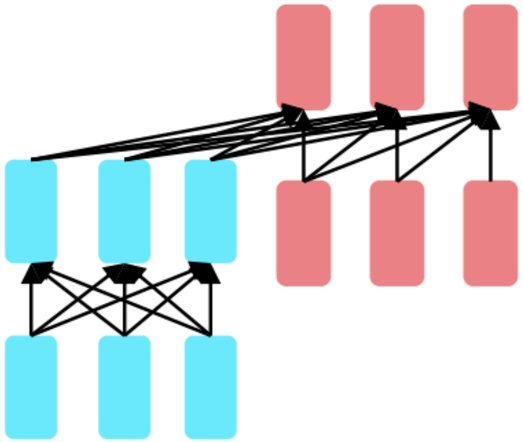
Quiz 3
Answers (Redacted)

Recap: Natural Language Generation

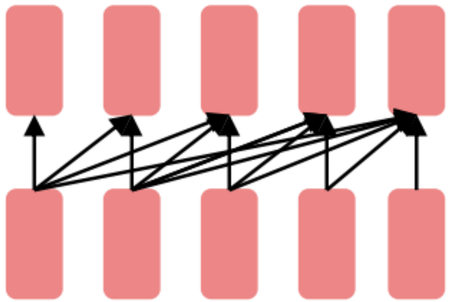
Broad Spectrum of NLG Tasks

Less Open-Ended — **More Open-Ended**

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



Encoder-Decoders



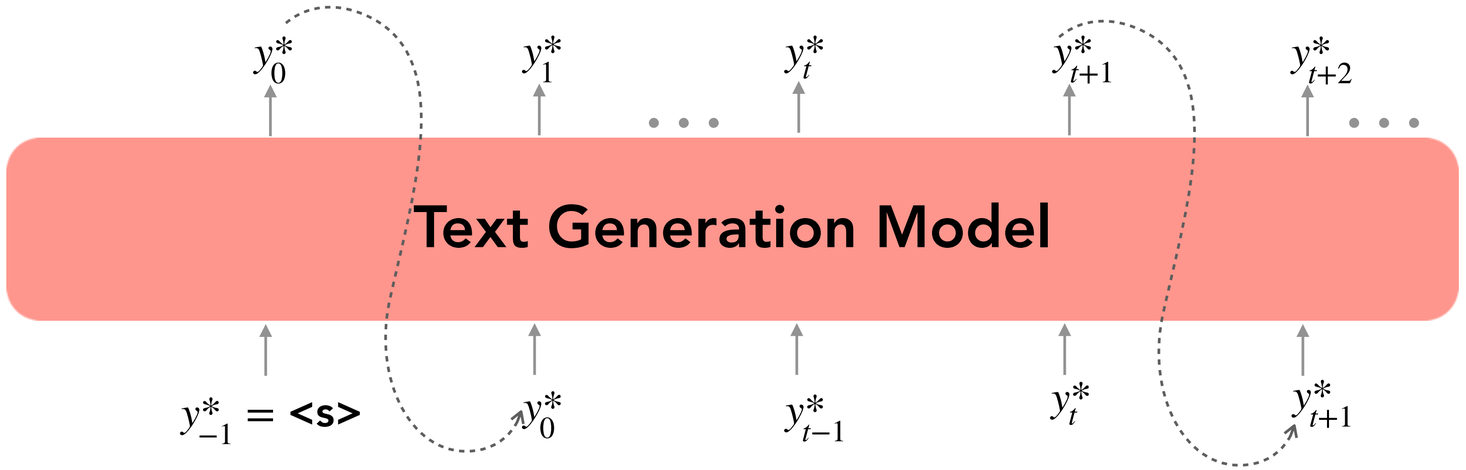
Decoders

Language Generation: 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.
  - But… greedy decoding has no wiggle room for errors!

$$g = \arg\max$$

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

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

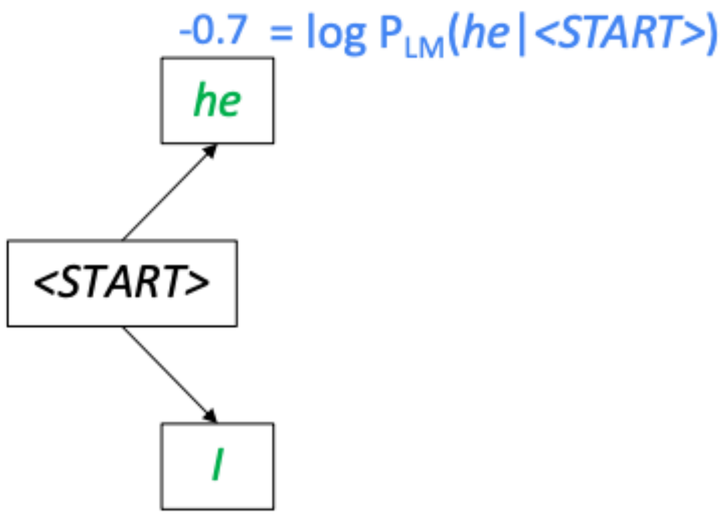
<START>

Calculate prob dist of next word

Slide credit: Chris Manning

Beam Search Decoding: Example

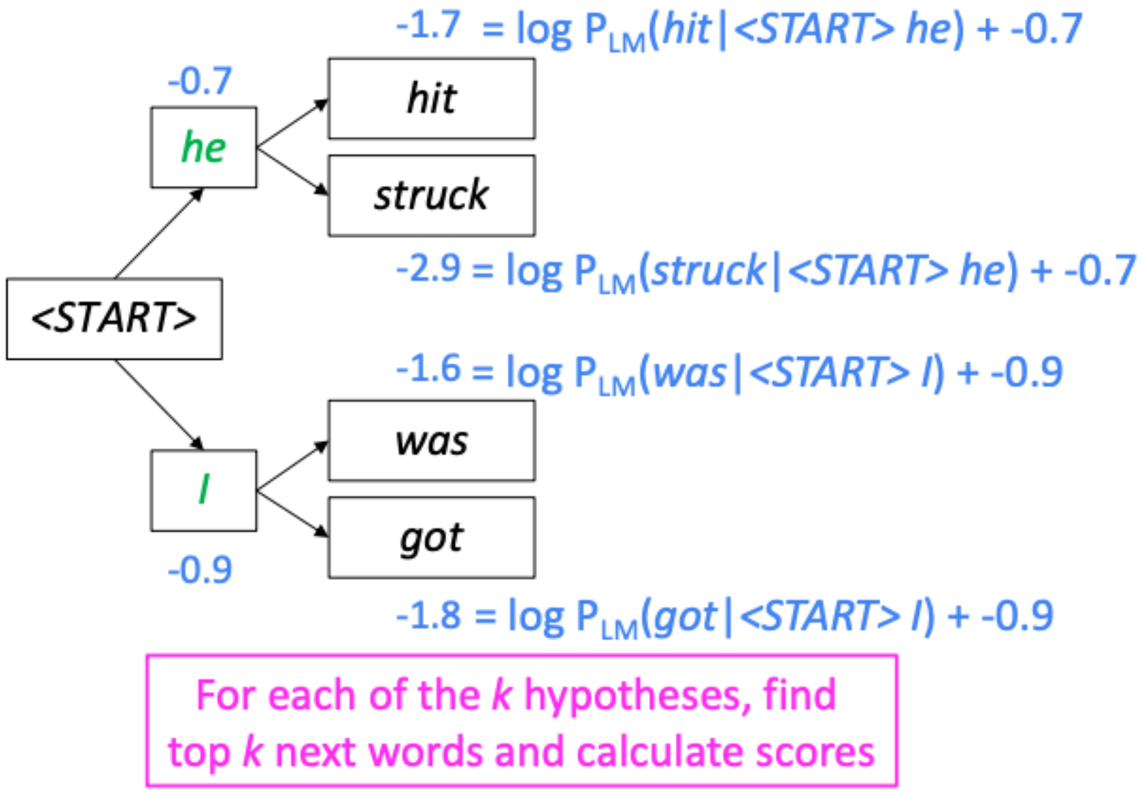
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Take top *k* words and compute scores

Beam Search Decoding: Example

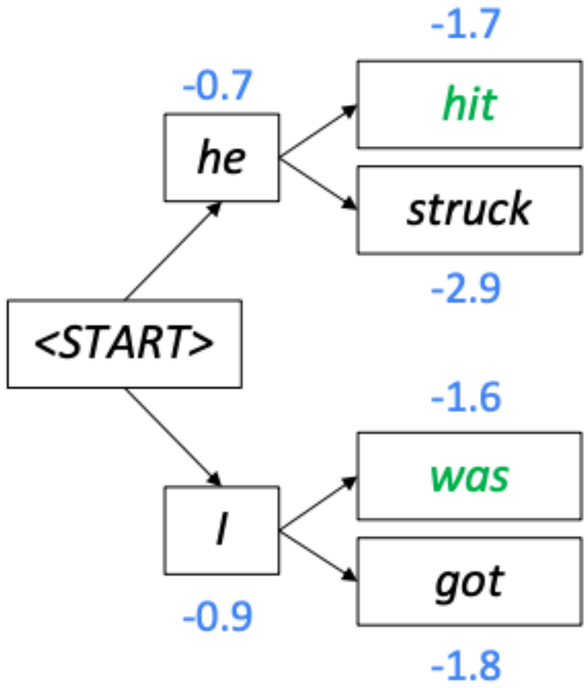
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Slide credit: Chris Manning

Beam Search Decoding: Example

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

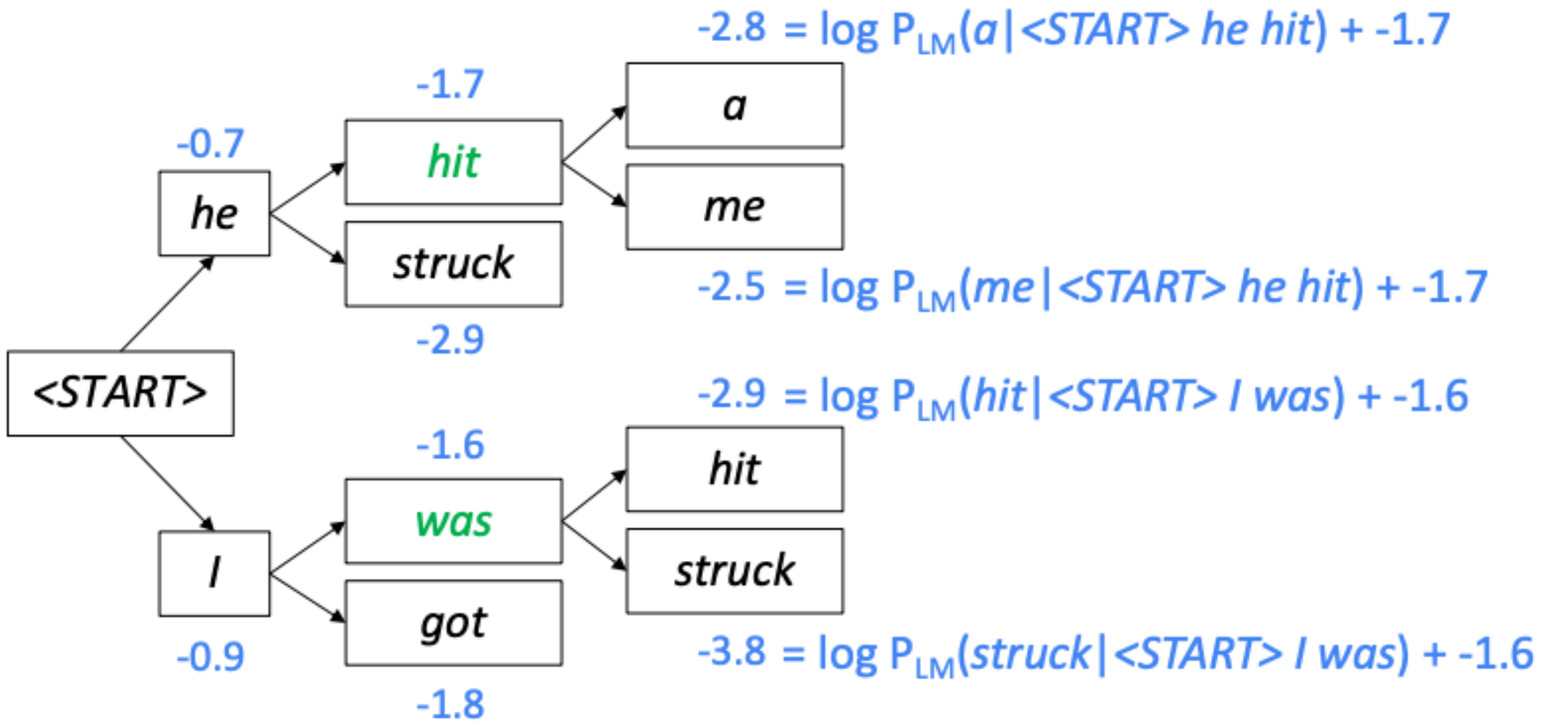


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

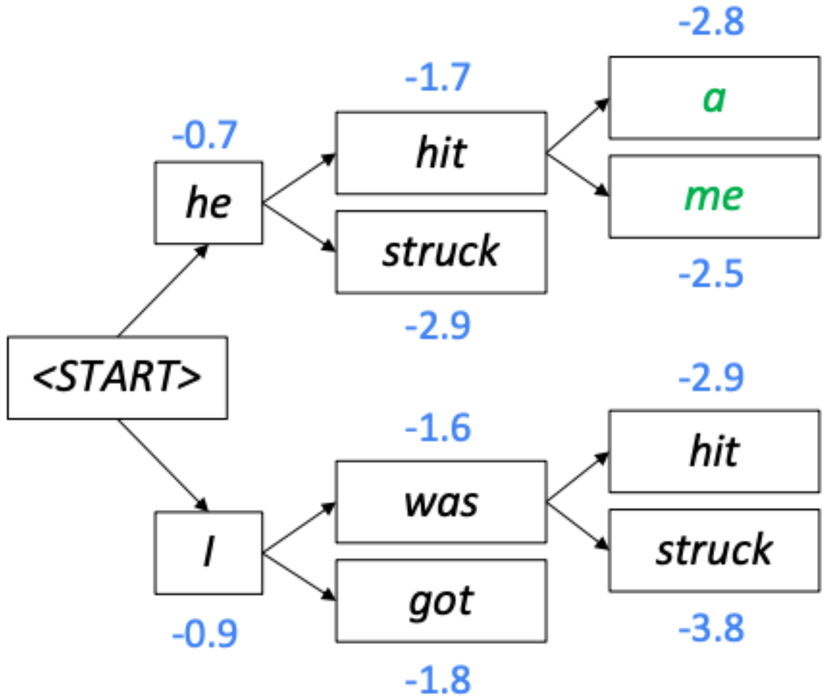


For each of the *k* hypotheses, find top *k* next words and calculate scores

de credit: Chris Manning

Beam Search Decoding: Example

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

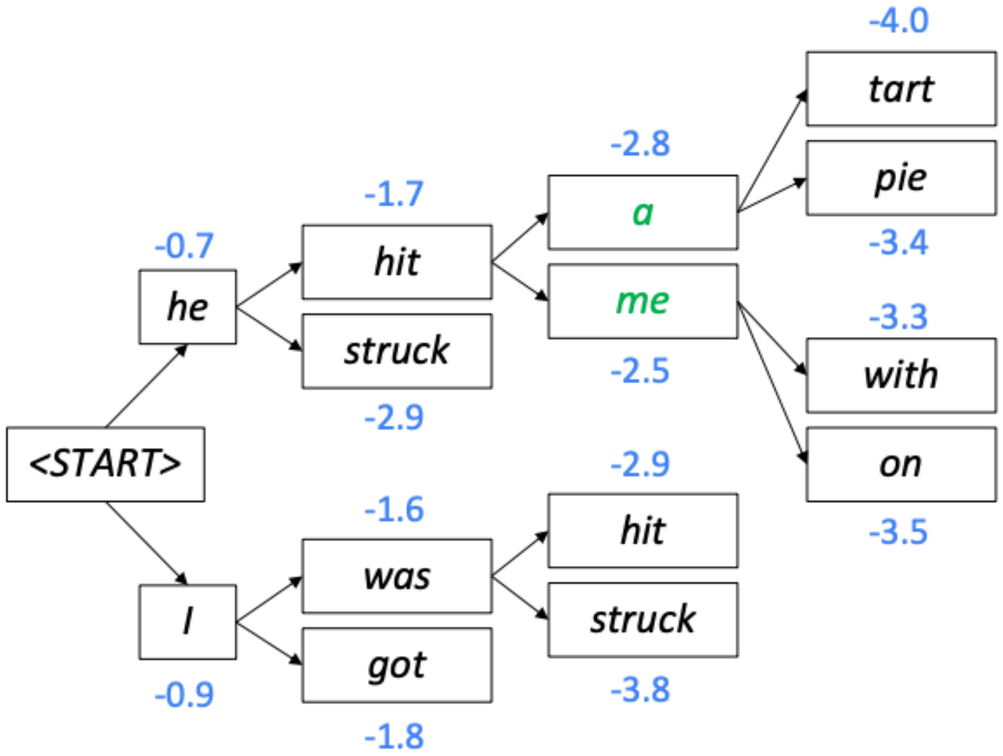


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

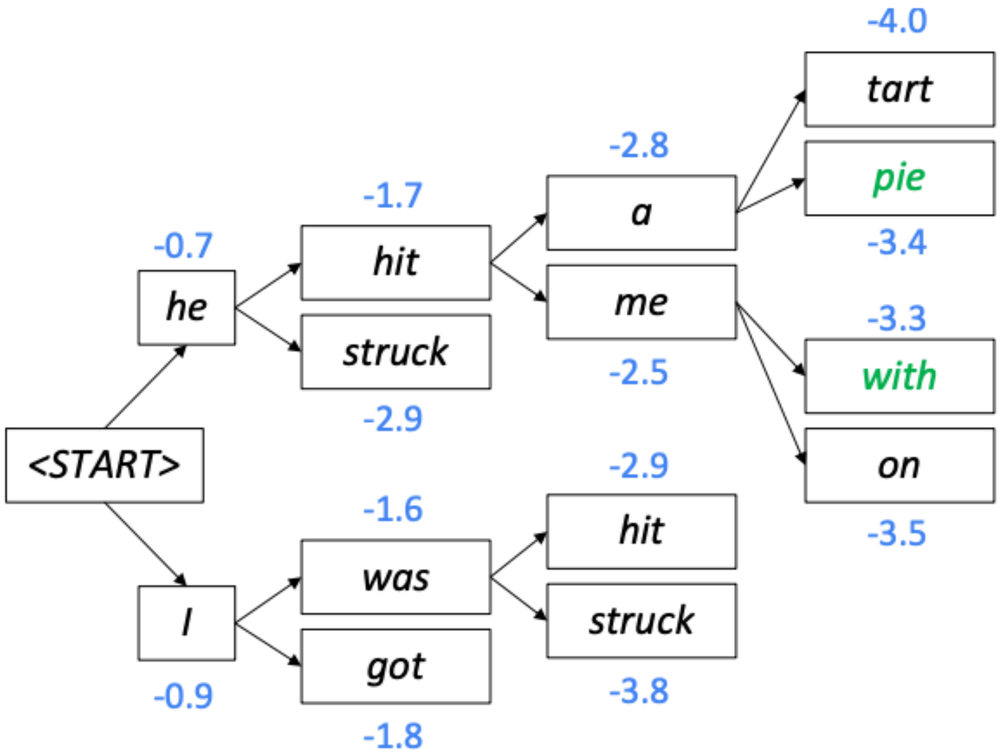


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

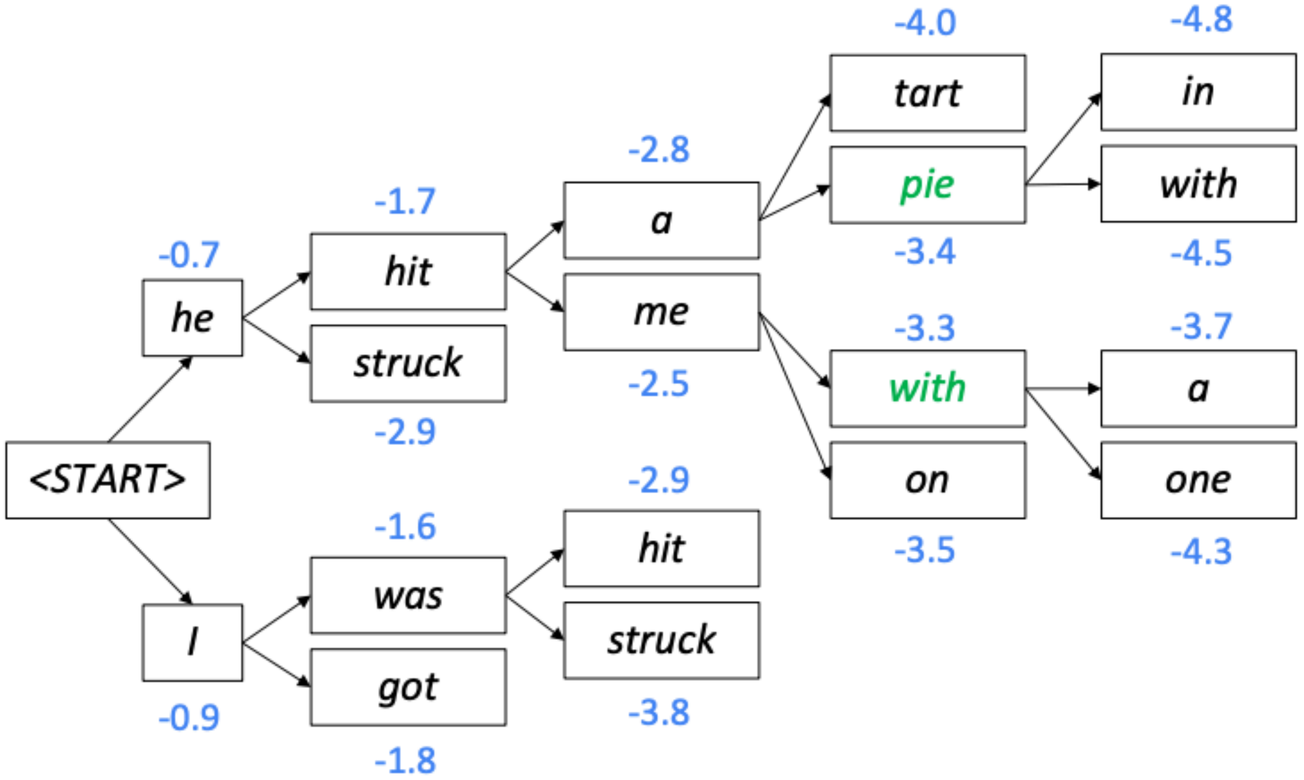


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

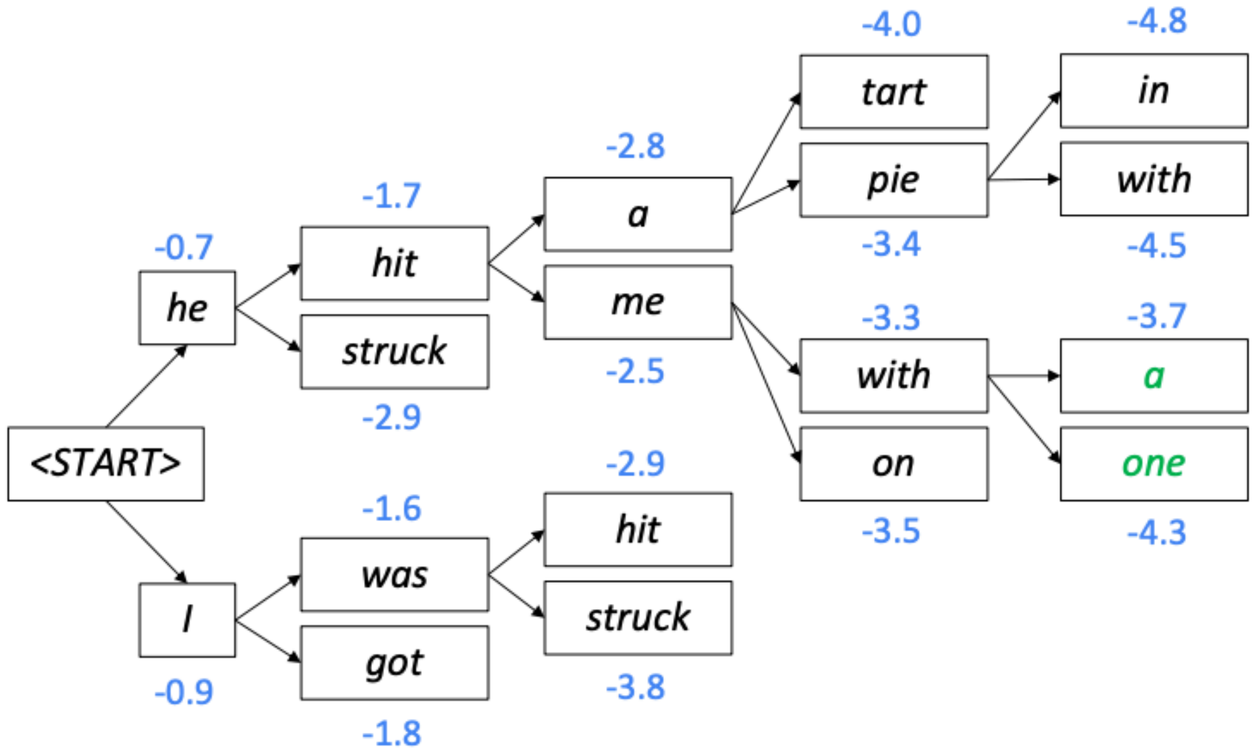


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

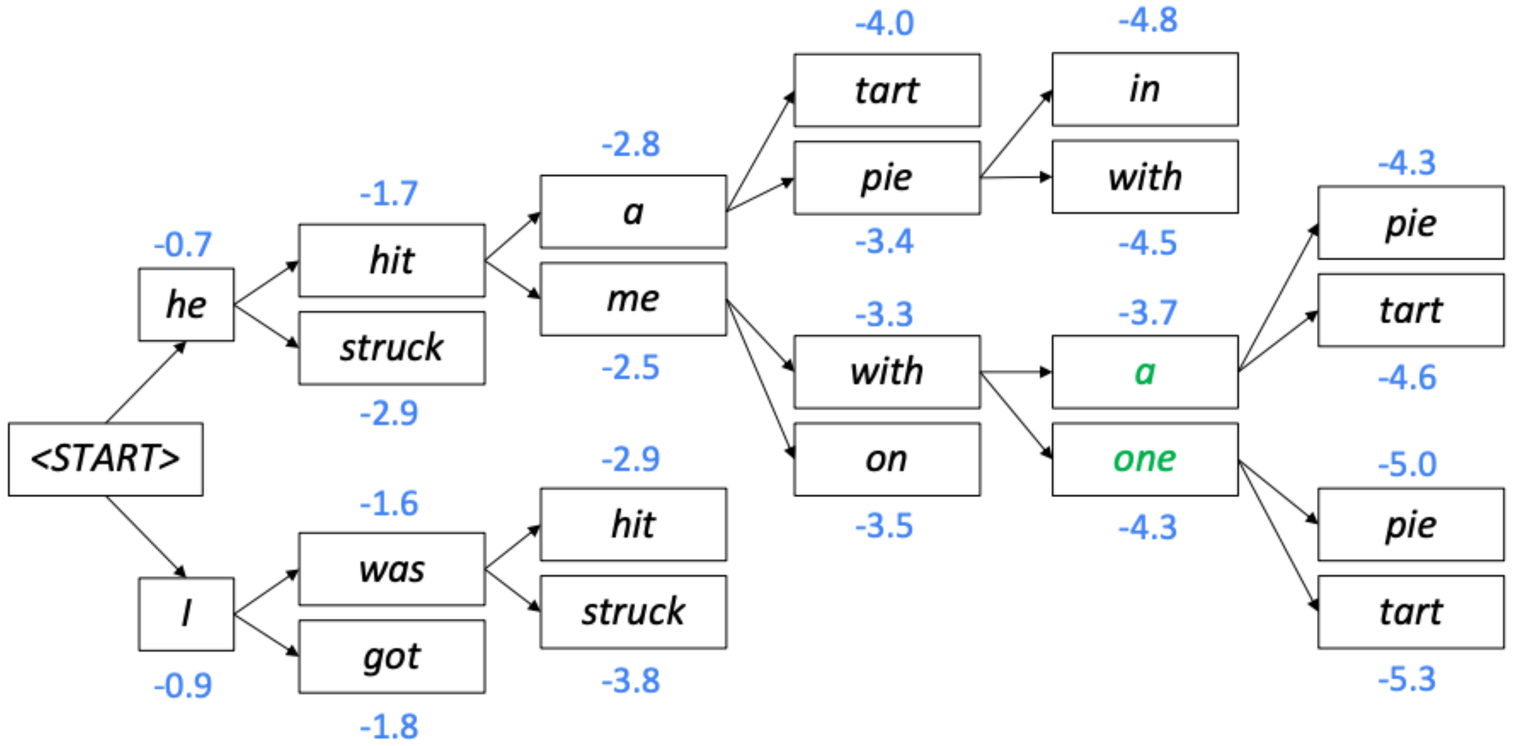


Of these k^2 hypotheses, just keep k with highest scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

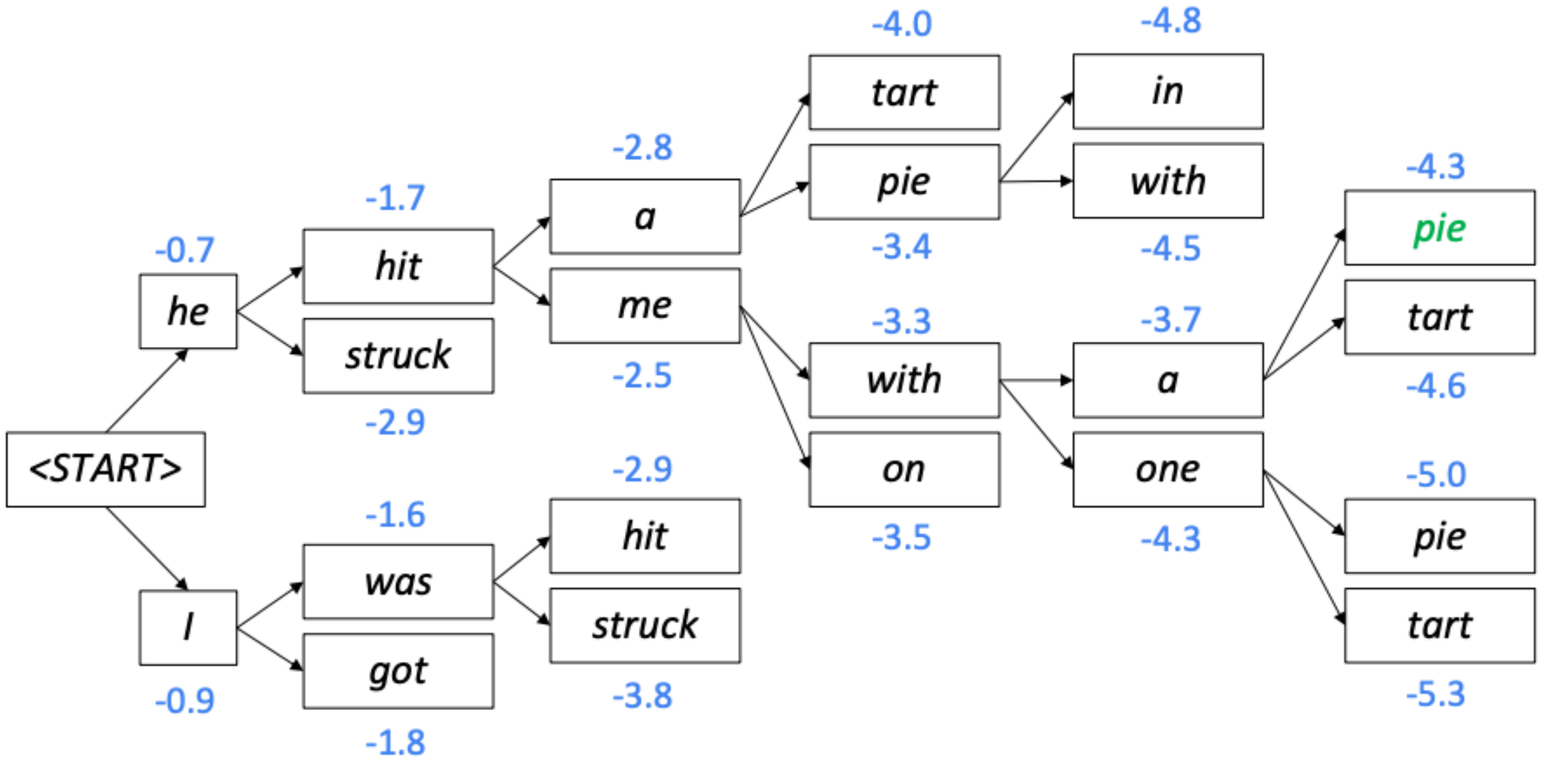


For each of the *k* hypotheses, find top *k* next words and calculate scores

Slide credit: Chris Manning

Beam Search Decoding: Example

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

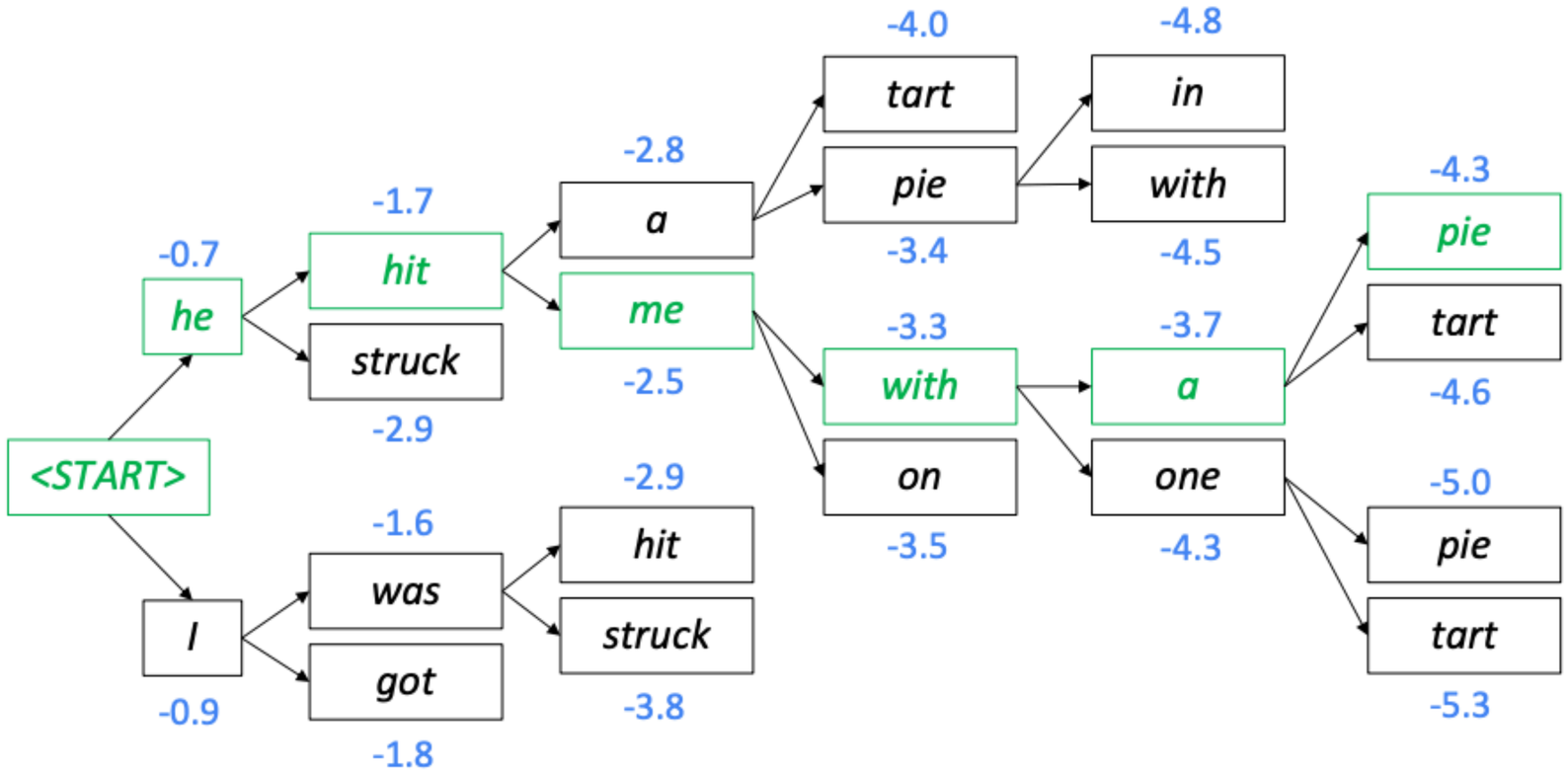


This is the top-scoring hypothesis!

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:

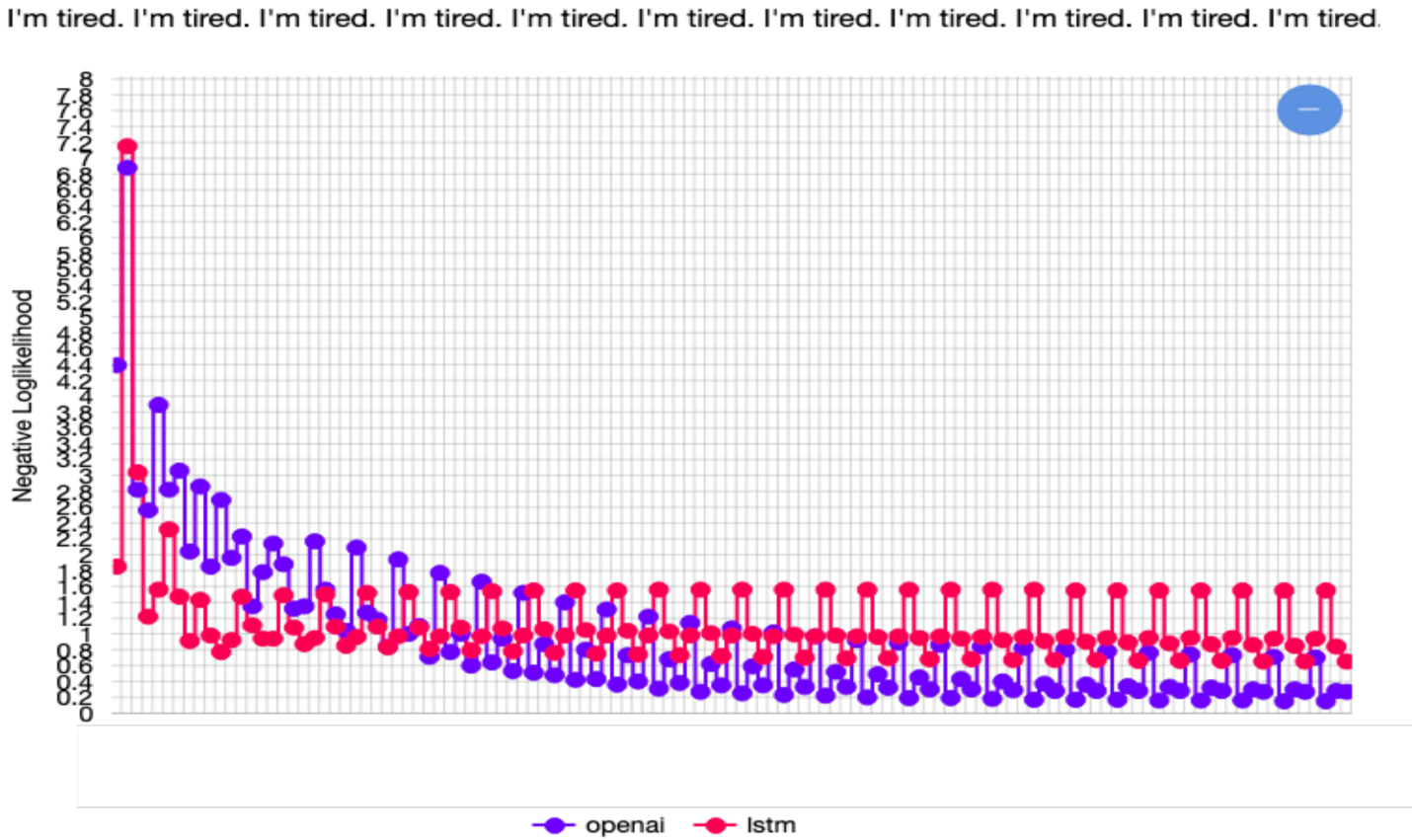
Generation can be bland or repetitive (also called degenerate)

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

Degenerate Outputs

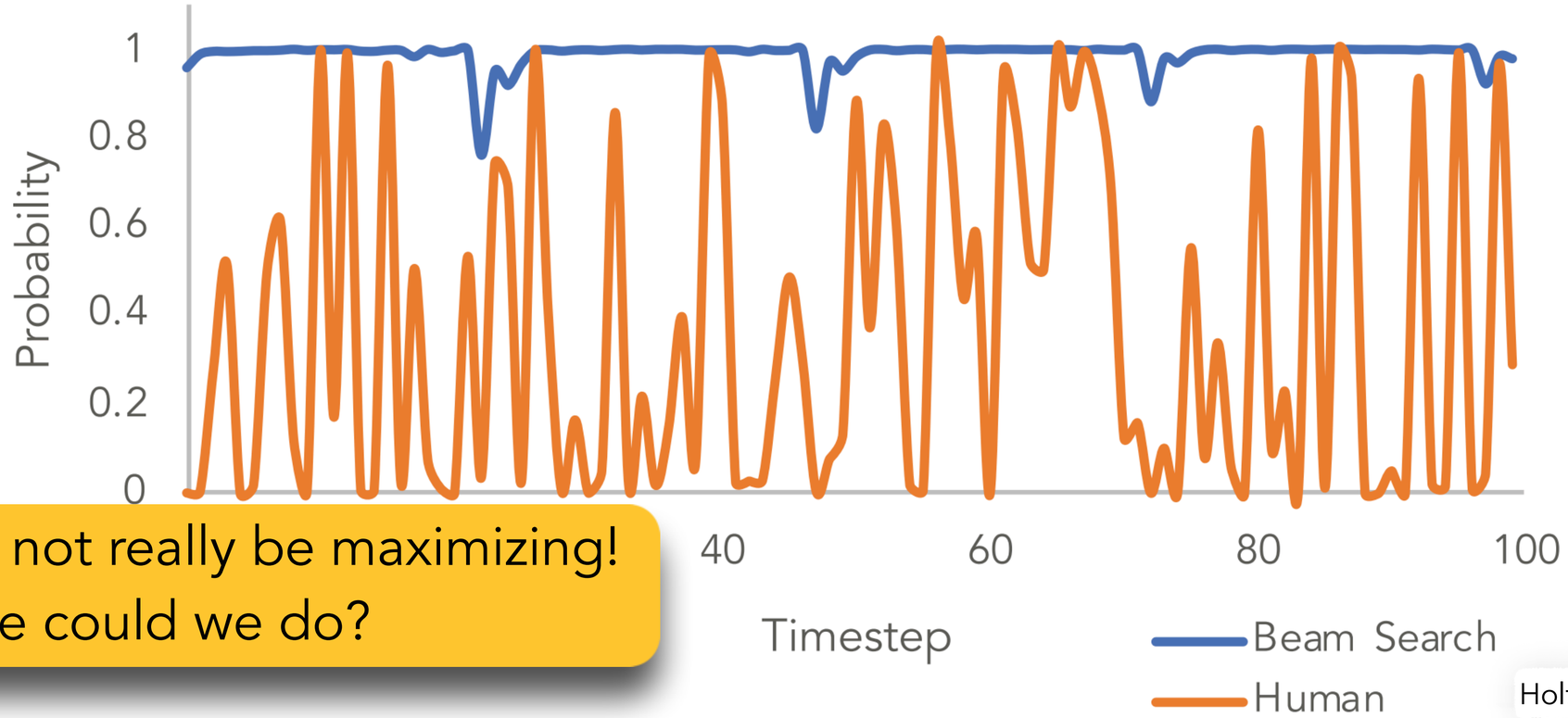


Holtzmann et al., 2020

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

Why does repetition happen?

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

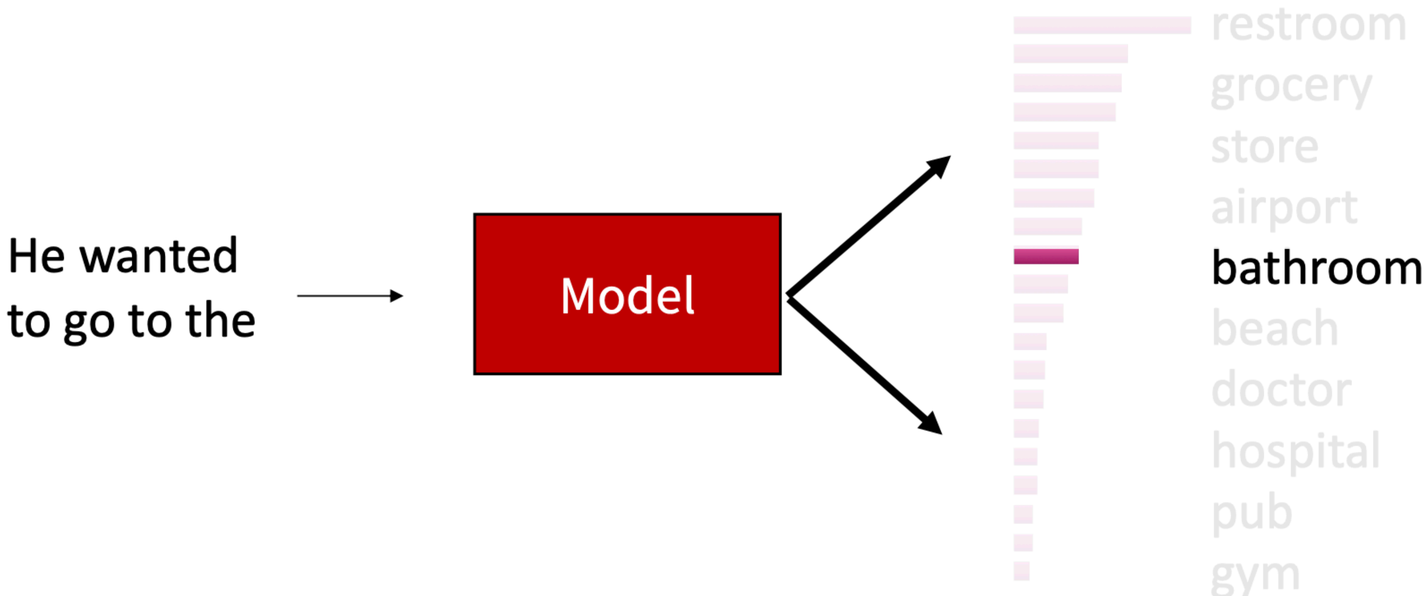


Perhaps we should not really be maximizing!
What else could we do?

Holtzmann et al., 2020

Solution: Don't Maximize, Pick a Sample

- 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!

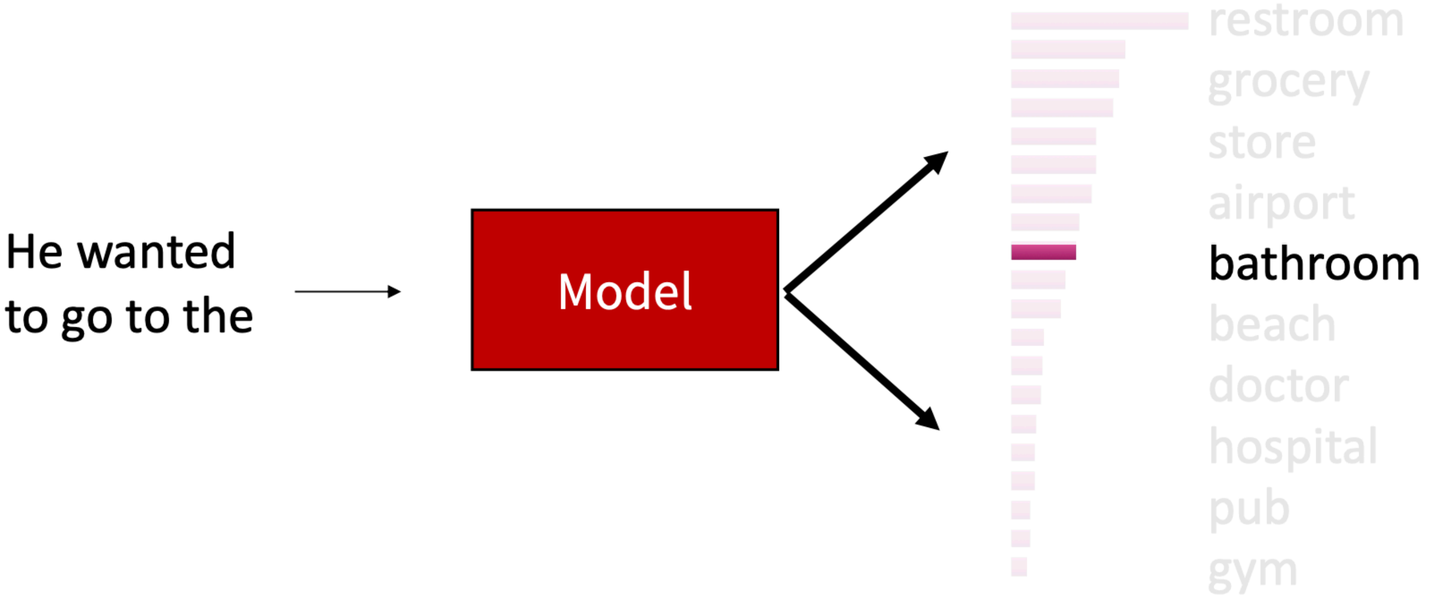


Modern Generation: Sampling

Pure / Ancestral Sampling

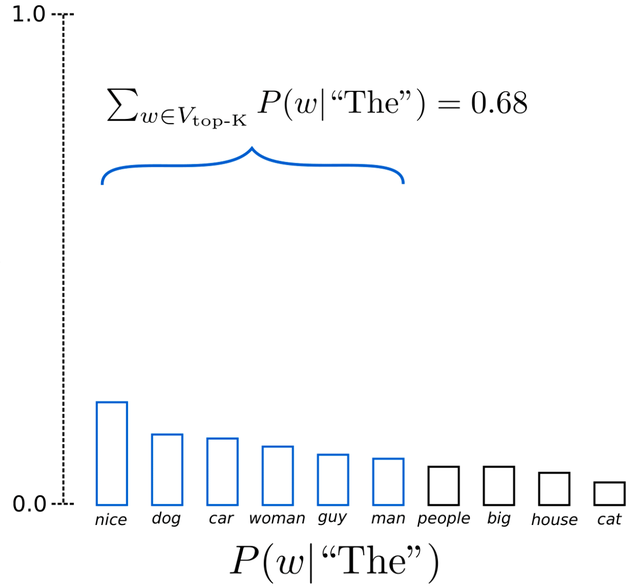
- Sample directly from P_t
  - Similar to sampling from an n -gram language model
- Still has access to the entire vocabulary
- But if the model distributions are of low quality, generations will be of low quality as well
- Often results in ill-formed generations
  - No guarantee of fluency

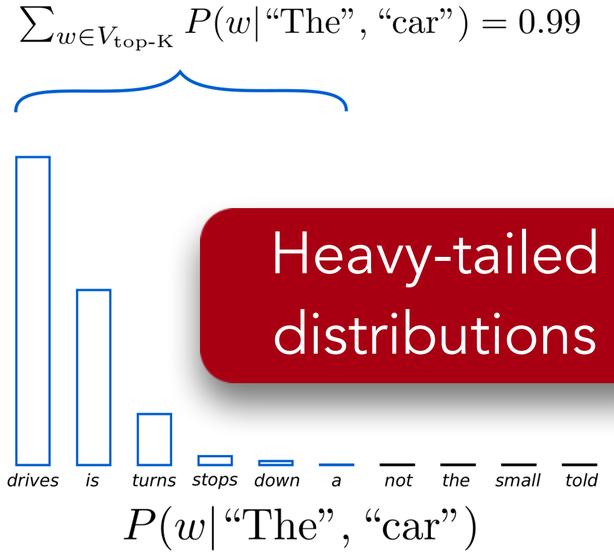
$$y_t \sim P_t(w) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$$



Top-K Sampling

- Problem: Ancestral sampling makes every token in the vocabulary an option
  - Even if most of the probability mass in the distribution is over a limited set of options, the tail of the distribution could be very long and in aggregate have considerable mass
  - Many tokens are probably really wrong in the current context. Yet, we give them individually a tiny chance to be selected.
  - But because there are many of them, we still give them as a group a high chance to be selected.
- Solution: Top-K sampling
  - Only sample from the top K tokens in the probability distribution





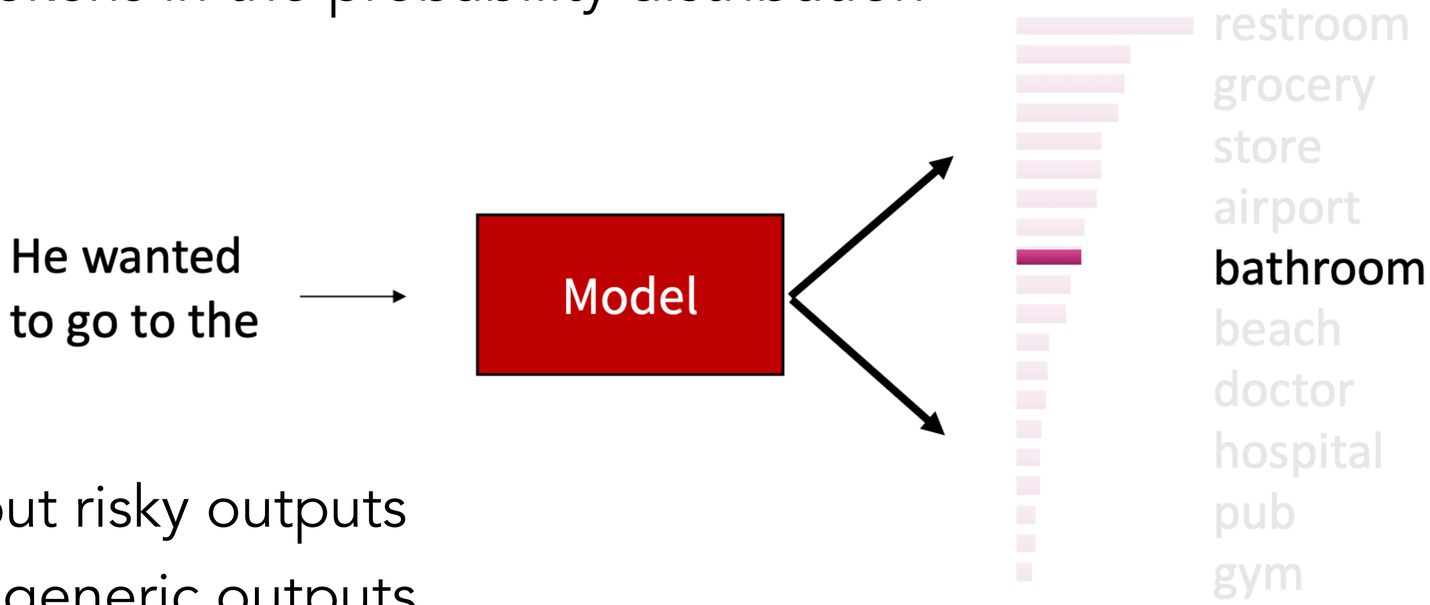
Heavy-tailed distributions

Fan et al., ACL 2018; Holtzman et al., ACL 2018

Image Source: Huggingface

Top-K Sampling: Value of K

- Solution: Top-K sampling
  - Only sample from the top K tokens in the probability distribution
  - Common values are K = 50



- Increase K yields more diverse, but risky outputs
- Decrease K yields more safe but generic outputs

Top-K Sampling: Issues

Top-K sampling can cut off too quickly

Top-K sampling can also cut off too slowly!

We can do better than having one-size-fits-all: a fixed K for all contexts

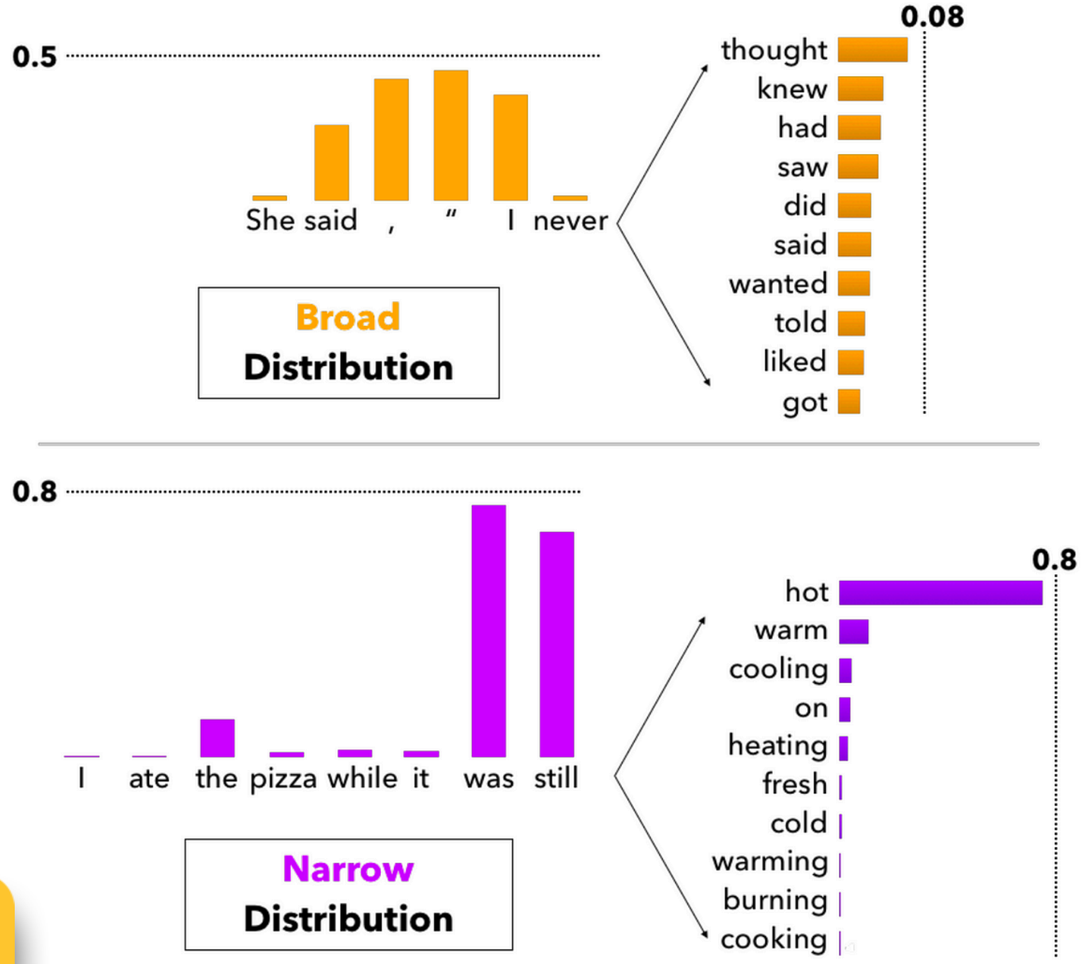


Image Source: Holtzmann et al., 2019

Modern Decoding: Nucleus Sampling

- Problem: The probability distributions we sample from are dynamic
  - When the distribution P_t is flatter, a limited K removes many viable options
  - When the distribution P_t is peakier, a high K allows for too many options to have a chance of being selected

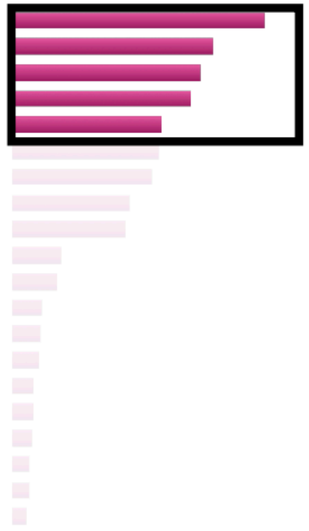
- Solution: Nucleus Sampling / Top-P sampling
  - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
  - Varies K depending on the uniformity of P_t

Holtzman et al., ICLR 2020

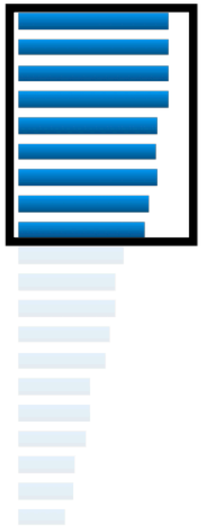
Nucleus (Top-P) Sampling

- Solution: Top-P sampling
  - Sample from all tokens in the top P cumulative probability mass (i.e., where mass is concentrated)
  - Varies K depending on the uniformity of P_t

$P_t^1(y_t = w \mid \{y\}_{<t})$



$P_t^2(y_t = w \mid \{y\}_{<t})$



$P_t^3(y_t = w \mid \{y\}_{<t})$

Holtzman et al., ICLR 2020

Temperature Scaling

Originally, $P(y_t = w) = \frac{\exp(S_w)}{\sum_{v \in V} \exp(S_v)}$

- Recall: On timestep t, the model computes a prob distribution P_t by applying the softmax function to a vector of scores $s \in \mathbb{R}^{|V|}$
- We can apply a temperature hyperparameter τ to the softmax to rebalance P_t

$$P(y_t = w) = \frac{\exp(S_w/\tau)}{\sum_{v \in V} \exp(S_v/\tau)}$$

- Let's say initial scores, S_w: (remember these are real-valued)
  - 0.1912, 0.7492, 0.5966, 0.5528, 0.8324, **0.9409**
- After softmax, p:
  - 0.1031, 0.1802, 0.1547, 0.1480, 0.1958, **0.2182**
- S_w/τ when $\tau = 0.01$:
  - 19.12, 74.92, 59.66, 55.28, 83.24, **94.09**
- After softmax, p
  - 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, **1.0000**

Temperature is a hyperparameter for decoding: It can be tuned for both beam search and sampling.

Sampling after Temperature Scaling

- Raise the temperature $\tau > 1$: P_t becomes more uniform
  - More diverse output (probability is spread around vocab)
- Lower the temperature $\tau < 1$: P_t becomes more spiky
  - Less diverse output (probability is concentrated on top words)