RPTU

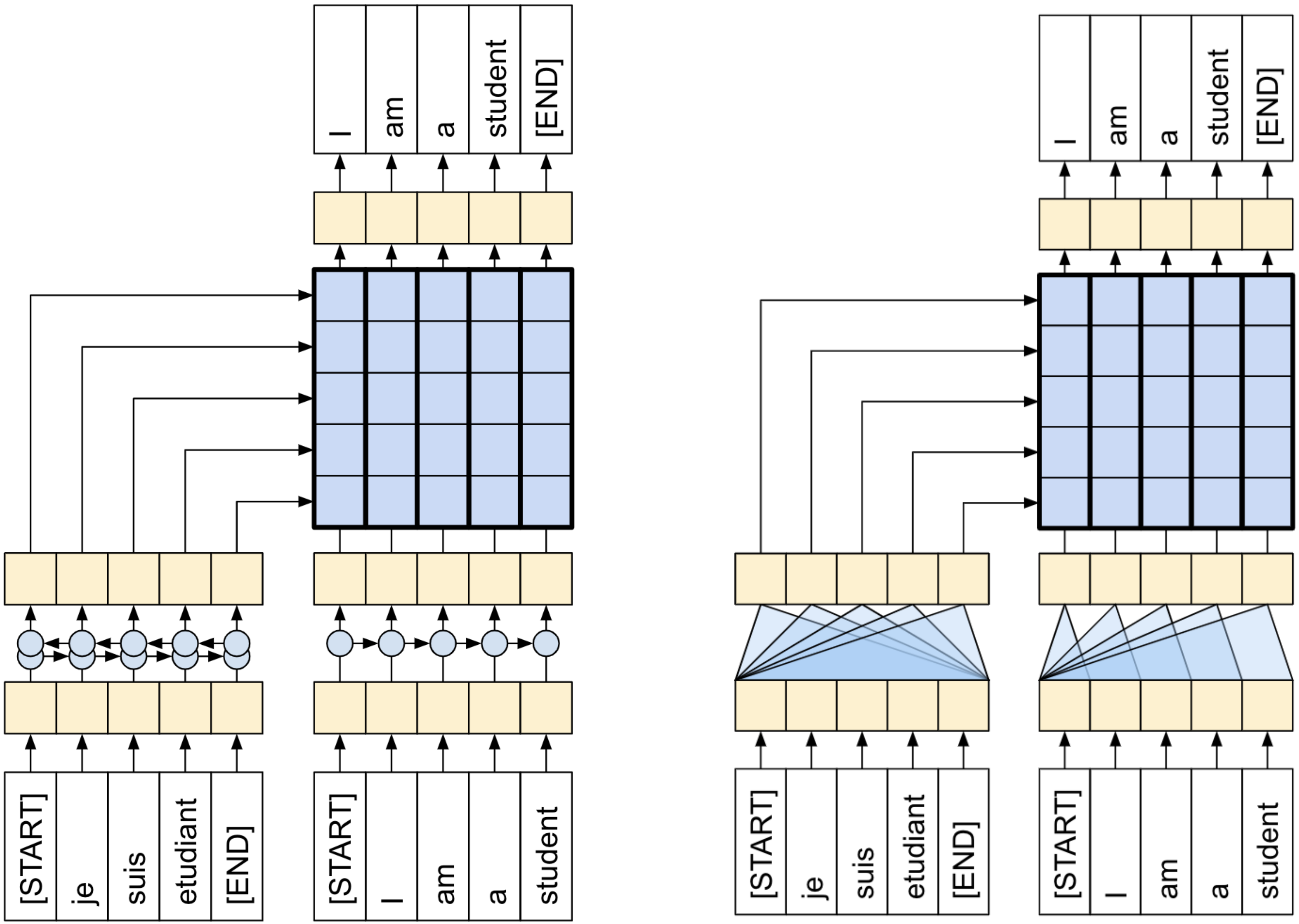
Neural Networks for Natural Language Processing

Lecture 7 – Language Models – Generating Text from Language Models

09.12.2024

Jun.-Prof. Sophie Fellenz

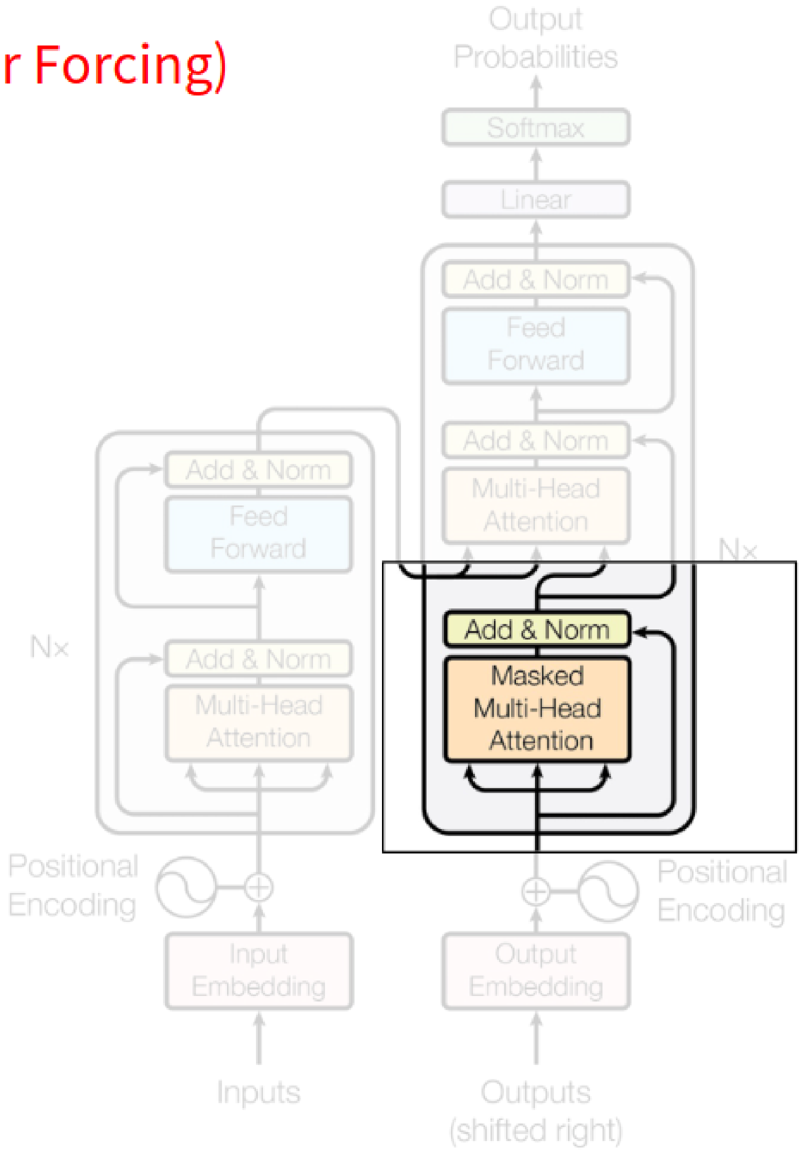
Image Source: https://www.tensorflow.org/text/tutorials/transformer



RPTU

Decoding step by step (using Teacher Forcing)

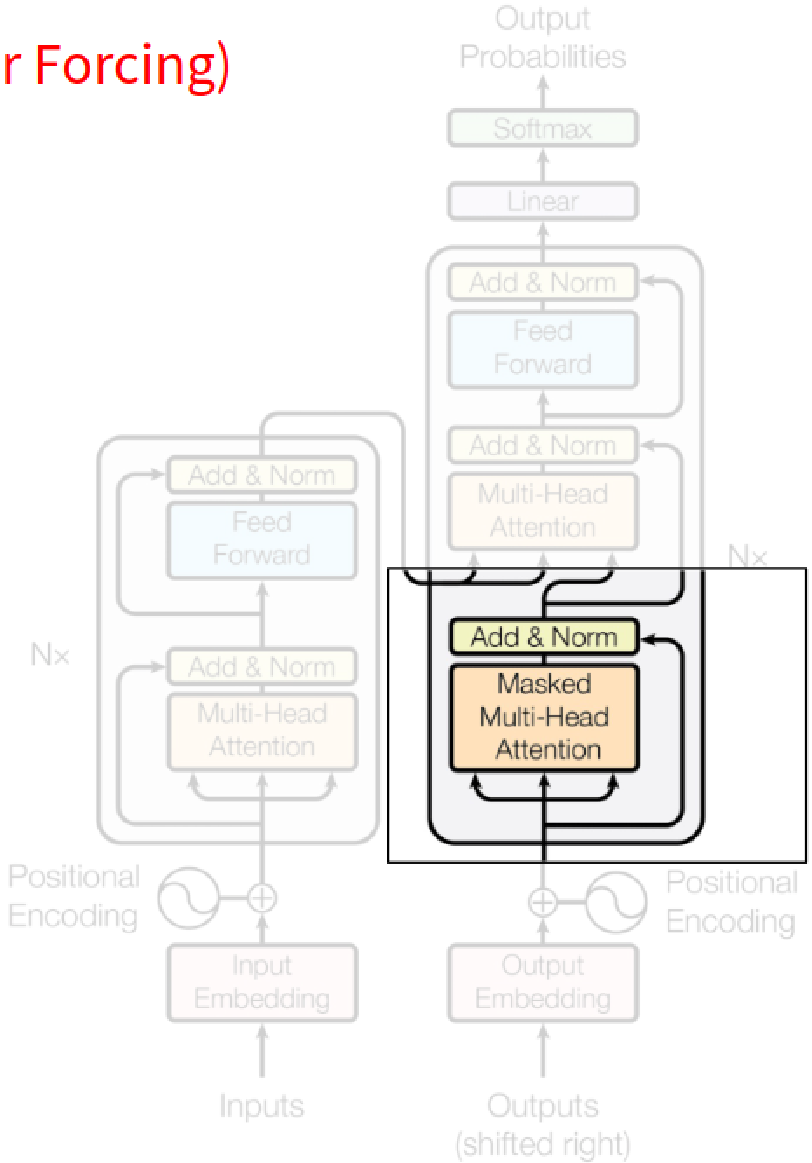
1	<sos>	I	have	eaten	an	apple	<eos>
2	<sos>	I	have	eaten	an	apple	<eos>
3	<sos>	I	have	eaten	an	apple	<eos>
4	<sos>	I	have	eaten	an	apple	<eos>
5	<sos>	I	have	eaten	an	apple	<eos>
6	<sos>	I	have	eaten	an	apple	<eos>
7	<sos>	I	have	eaten	an	apple	<eos>



Mask the available attention values ?

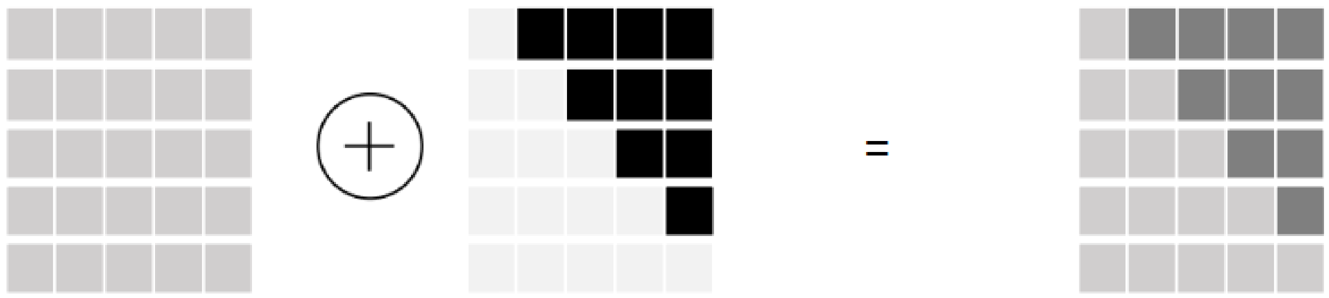
Decoding step by step (using Teacher Forcing)

1	<sos>	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
2	<sos>	I	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
3	<sos>	I	have	$-\infty$	$-\infty$	$-\infty$	$-\infty$
4	<sos>	I	have	eaten	$-\infty$	$-\infty$	$-\infty$
5	<sos>	I	have	eaten	an	$-\infty$	$-\infty$
6	<sos>	I	have	eaten	an	apple	$-\infty$
7	<sos>	I	have	eaten	an	apple	<eos>



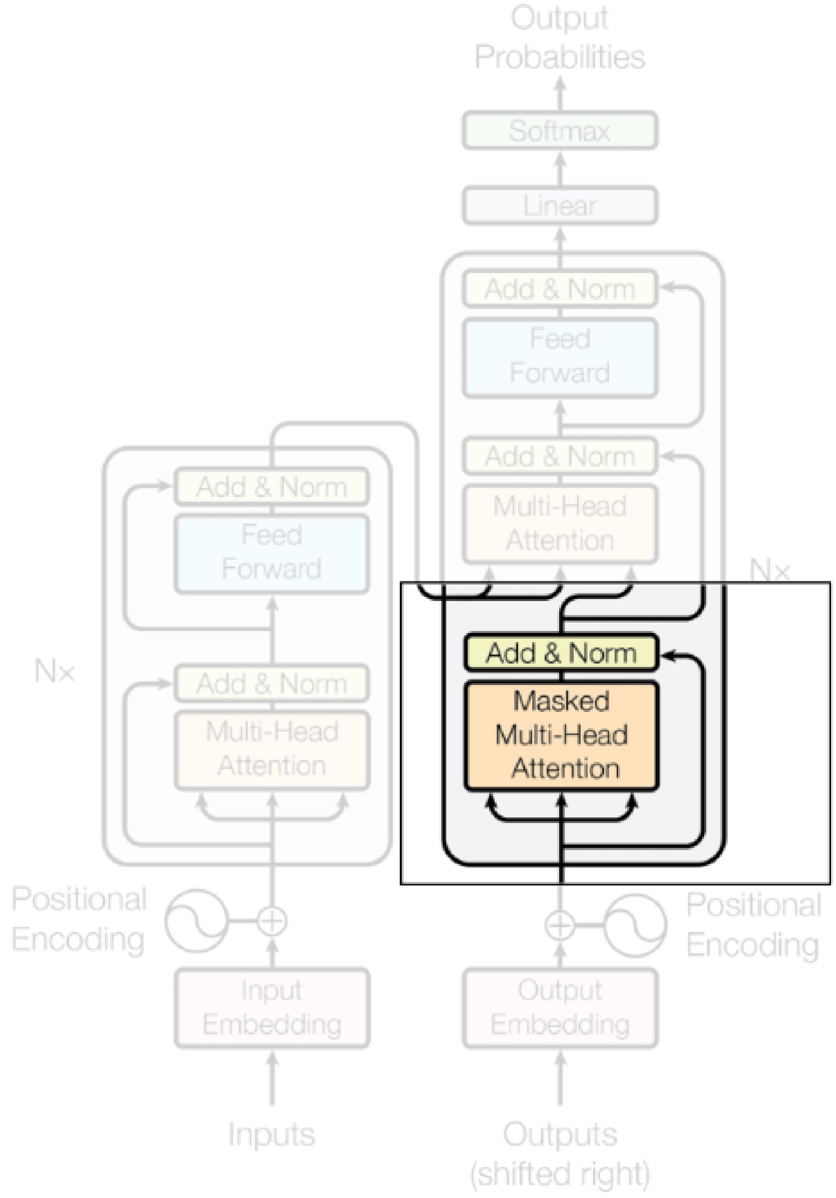
Masked Multi Head Attention

$R^{T\times T}$ $\quad$ $R^{T\times T}$



QK^T $\oplus$ Attention Mask: M = Masked Attention

Masked Multi Head Attention : Z'



Beam search

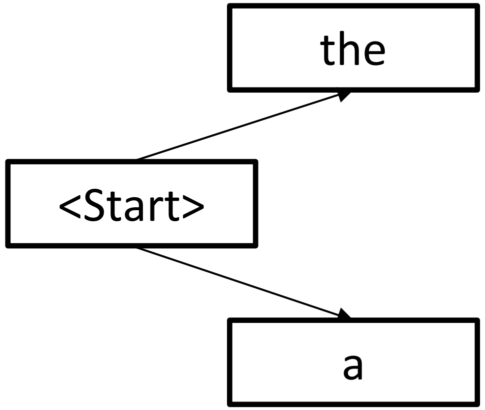
Beam search: Keep track of the k most likely partial translations

k is the beam size

Beam search

RPTU

Beam size: 2

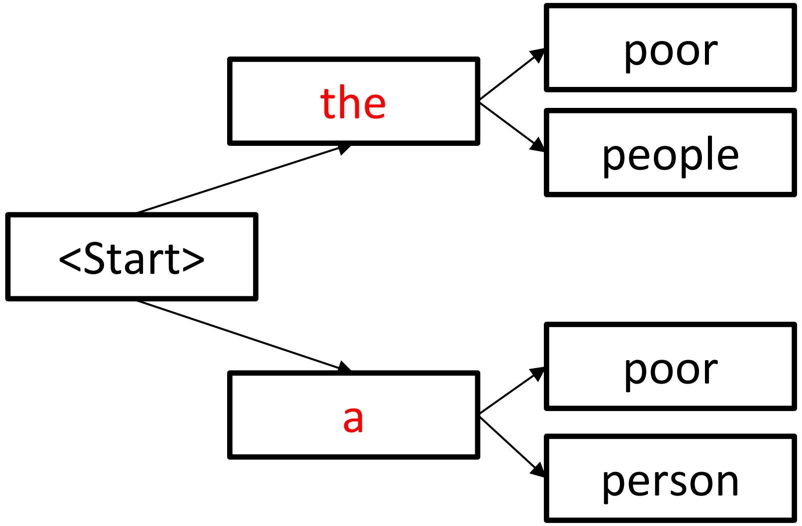


Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

RPTU

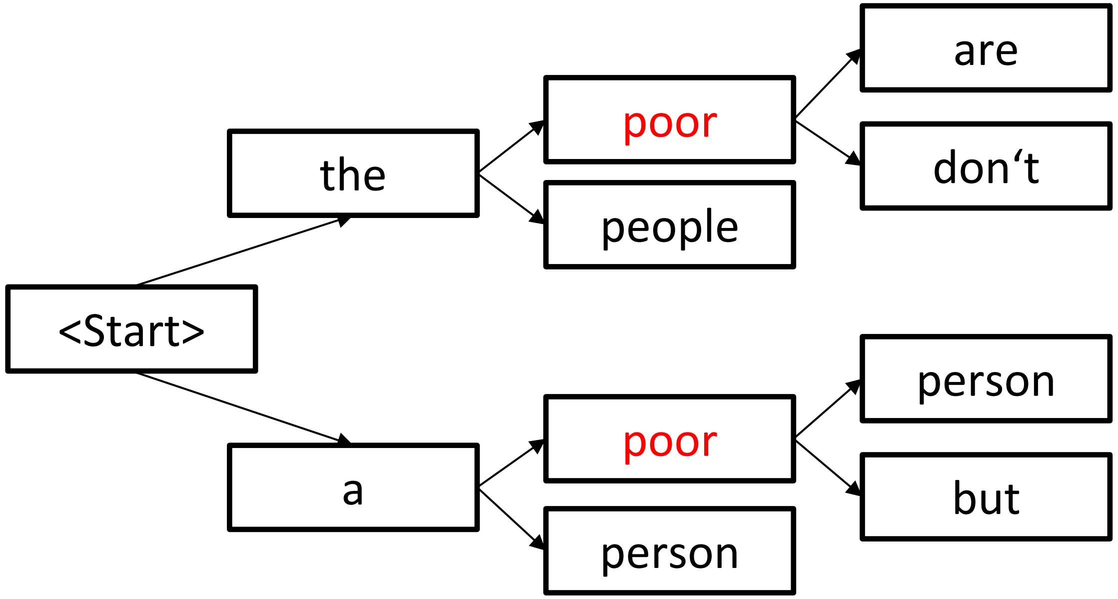
Beam size: 2



Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

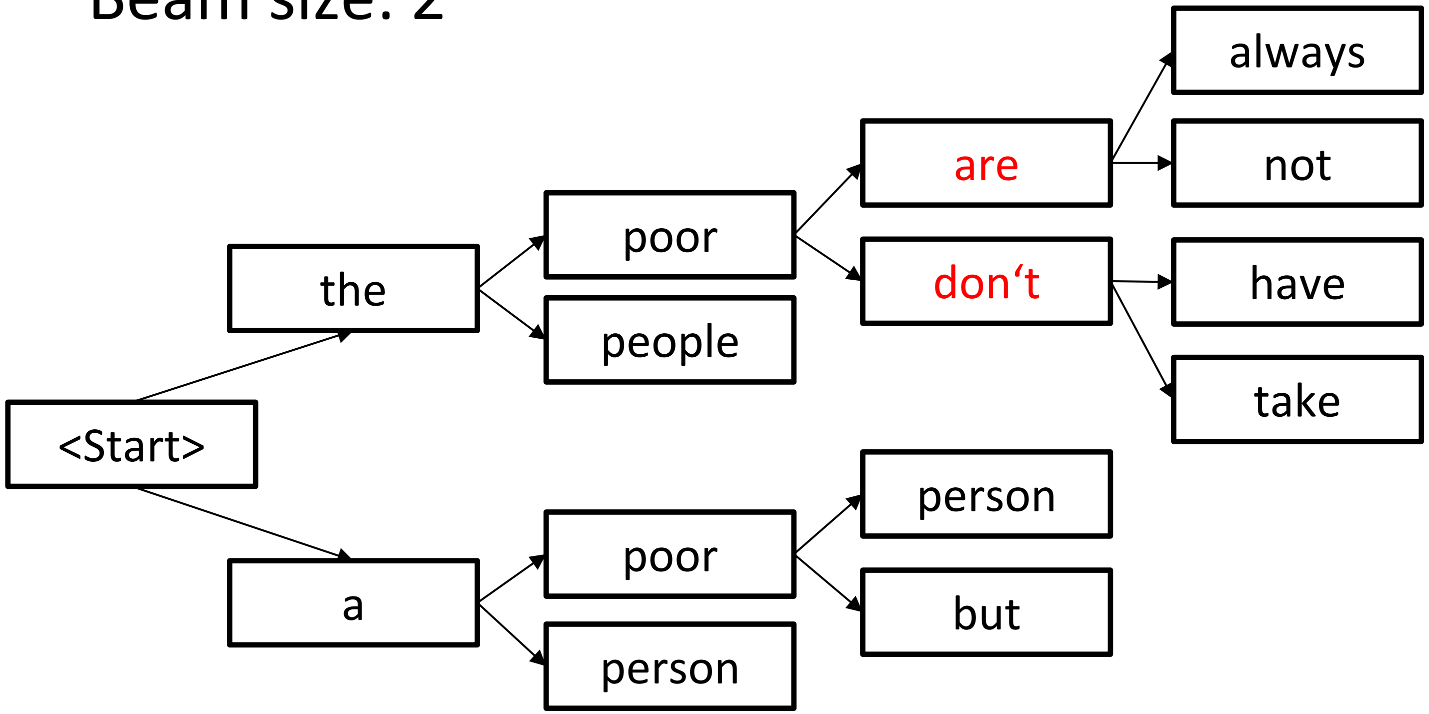
Beam size: 2



Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

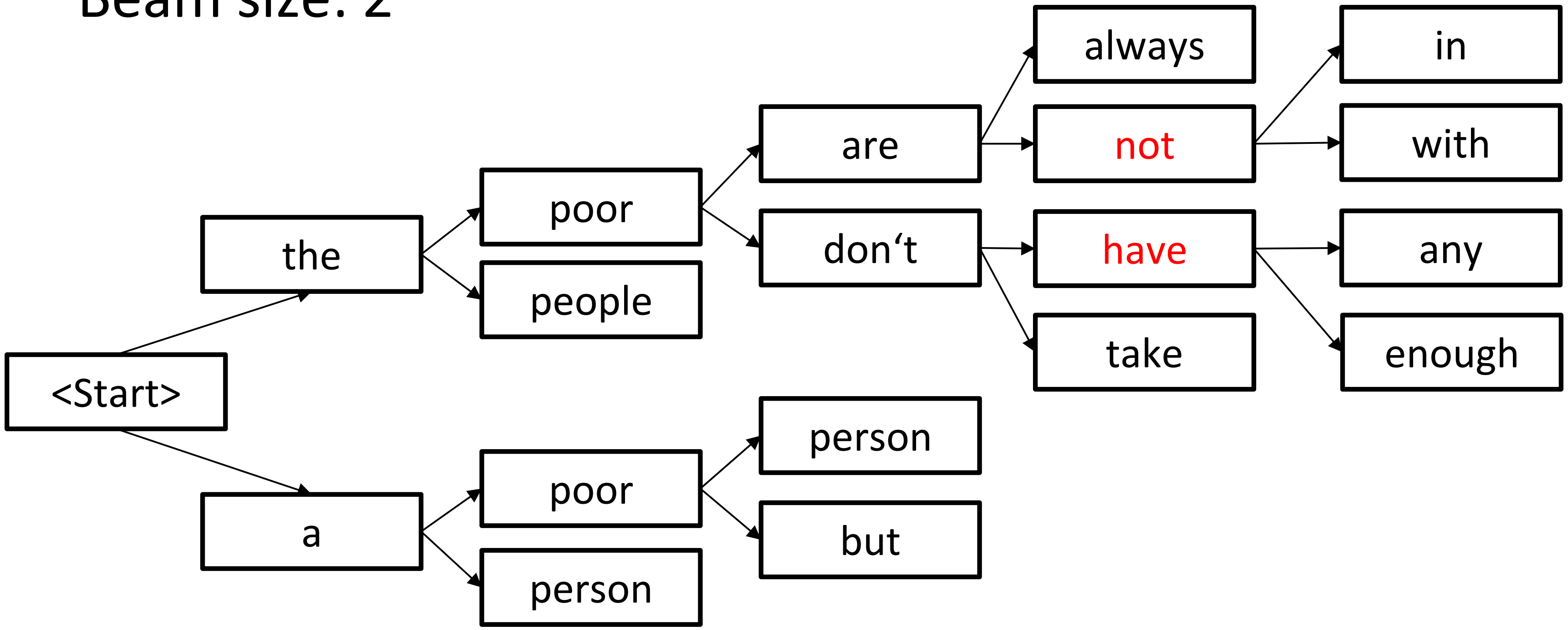
Beam size: 2



Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

Beam size: 2

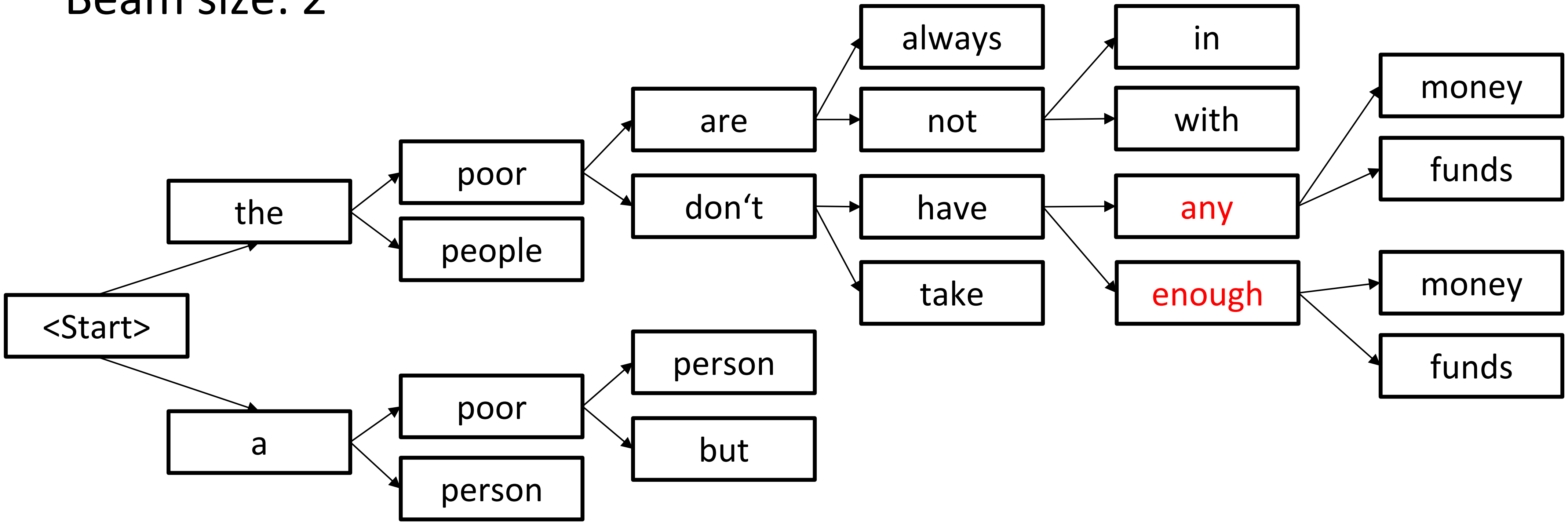


Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

RPTU

Beam size: 2

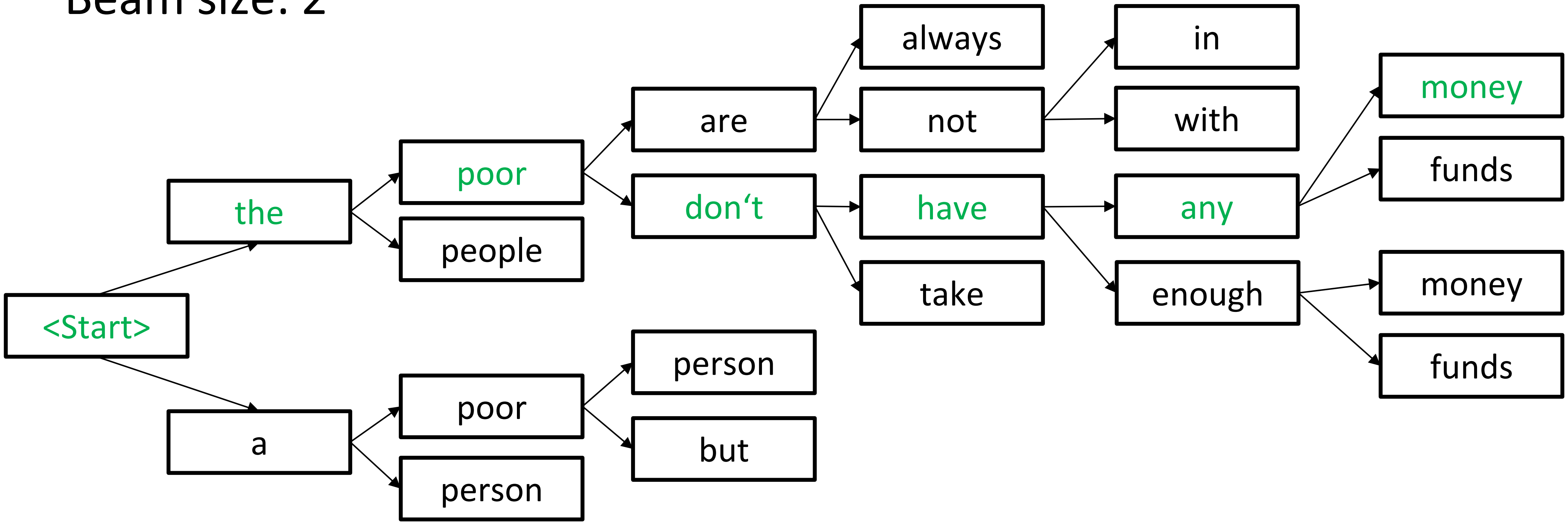


Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Beam search

RPTU

Beam size: 2



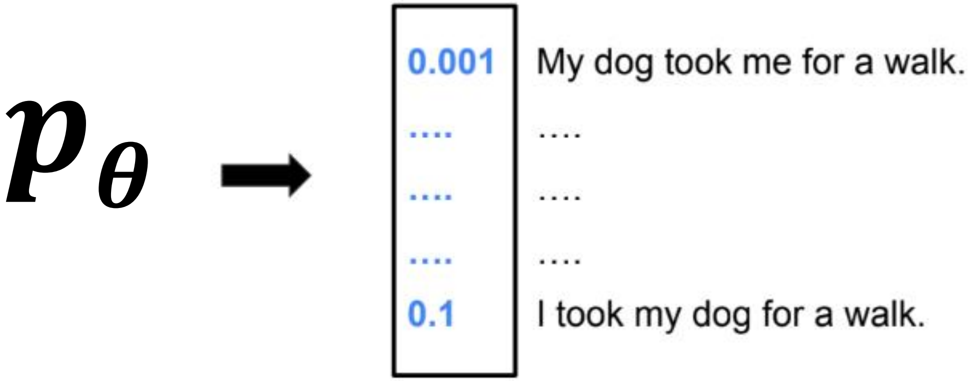
Source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture10.pdf

Agenda

- Recap: language models
- Generating text: Intro
- Scoring functions
- Prompting

Language Models: Definition

- A language model (LM) p_θ is a **probability distribution over strings**
  - It should tell us how likely a piece of text is to occur in a particular natural language domain
  - (i.e., the domain of the training corpus)



Language Models: Language Generators?

- Intuitively, are Language Models naturally suited to be language generators?

- Questions with non-obvious answers:
  - Do we want the highest-probability string, or high-probability strings in general?
  - Can we “re-orient” these distributions towards text with specific attributes without retraining/fine-tuning them?
  - If so, how can we quantify the qualitative attributes of text that we want?

Language Models: Language Generators?

- Language models might not be perfectly-suited for out-of-the-box language generation.
- Several popular approaches to turning language models into better language generators involve fine-tuning with alternative objectives

Language Models: Language Generators?

- Language models might not be perfectly-suited for out-of-the-box language generation.
- Several popular approaches to turning language models into better language generators involve fine-tuning with alternative objectives
  - Such approaches can be resource-intensive and have high variance.
- We will discuss methods developed for **standard language models.**

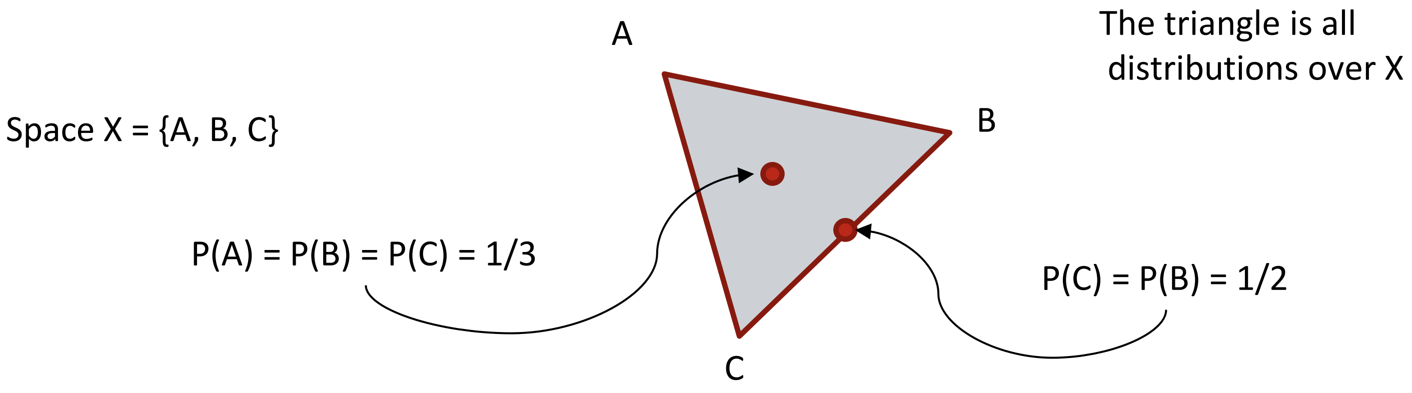
Language Models: Language Generators?

- Language models might not be perfectly-suited for out-of-the-box language generation.
- Several popular approaches to turning language models into better language generators involve fine-tuning with alternative objectives
  - Such approaches can be resource-intensive and have high variance.
- We will discuss methods developed for **standard language models.**
  - (Understanding the key characteristics/properties of language models will help in developing and choosing methods for steering out-of-the-box models towards generating the type of text that we want without resource intensive methods.)

Fundamentals of discrete distributions

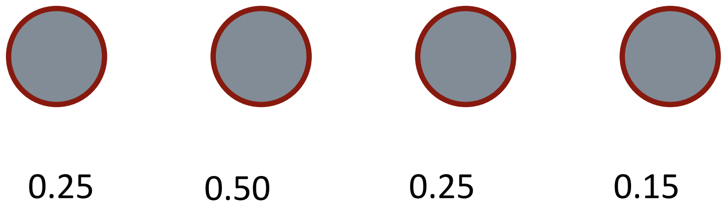
Discrete probability distributions

- A discrete probability distribution p(x) is defined over a countable space X, and
- follows the constraints:
  - **Non-negativity**: For all x in X, p(x) ≥ 0
  - **Normalization**: The sum over all x of p(x) is 1.



“Small” discrete sets

Easy to “just write out all the probabilities”

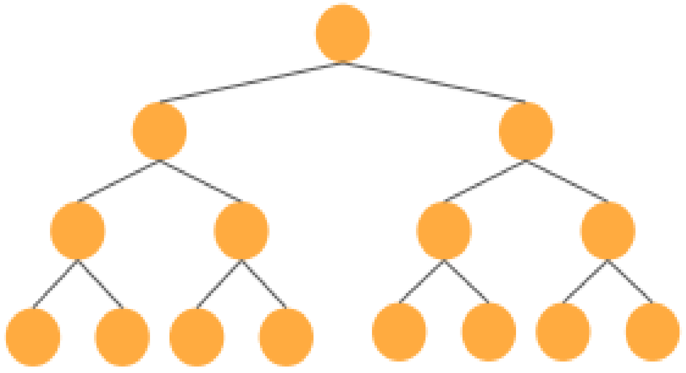


e.g., a vocabulary: **5,000** to **250,000** elements

No underlying structure in the set!

“Large” discrete sets

Very hard to
“just write out”
all the probabilities



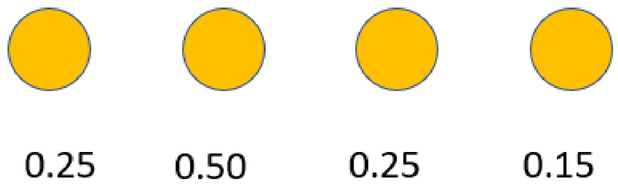
e.g., sequences of up to length m
defined over the vocabulary.

Underlying structure:
Elements of the set
are strings over a
common vocabulary!

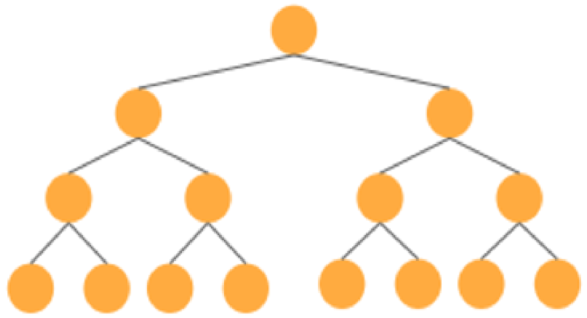
This means $|\text{vocab}|^m$ strings!

Discrete probability distributions

- A discrete probability distribution *p(x)* is defined over a countable space X, and
- follows the constraints:
  - **Non-negativity**: For all x in X, p(x) ≥ 0
  - **Normalization**: The sum over all x of p(x) is 1.



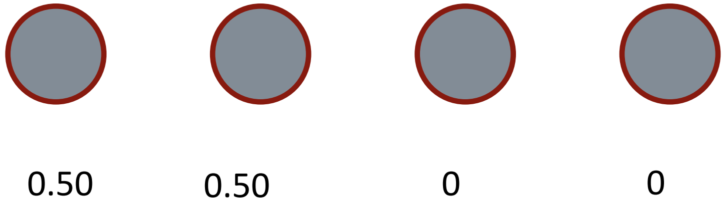
e.g., a vocabulary: **5000** to **250,000** elements



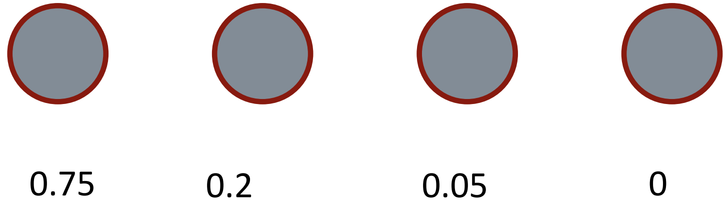
e.g., sequences of up to length m defined over the vocabulary.

The entropy of a distribution measures uncertainty

- Entropy is arguably the single most useful property of a distribution when trying to understand it qualitatively.
- It is a quantification of how random, or how uncertain the value of a random variable drawn from the distribution is.



This distribution has 1 bit of entropy, meaning there is as much uncertainty as two uniformly possible options



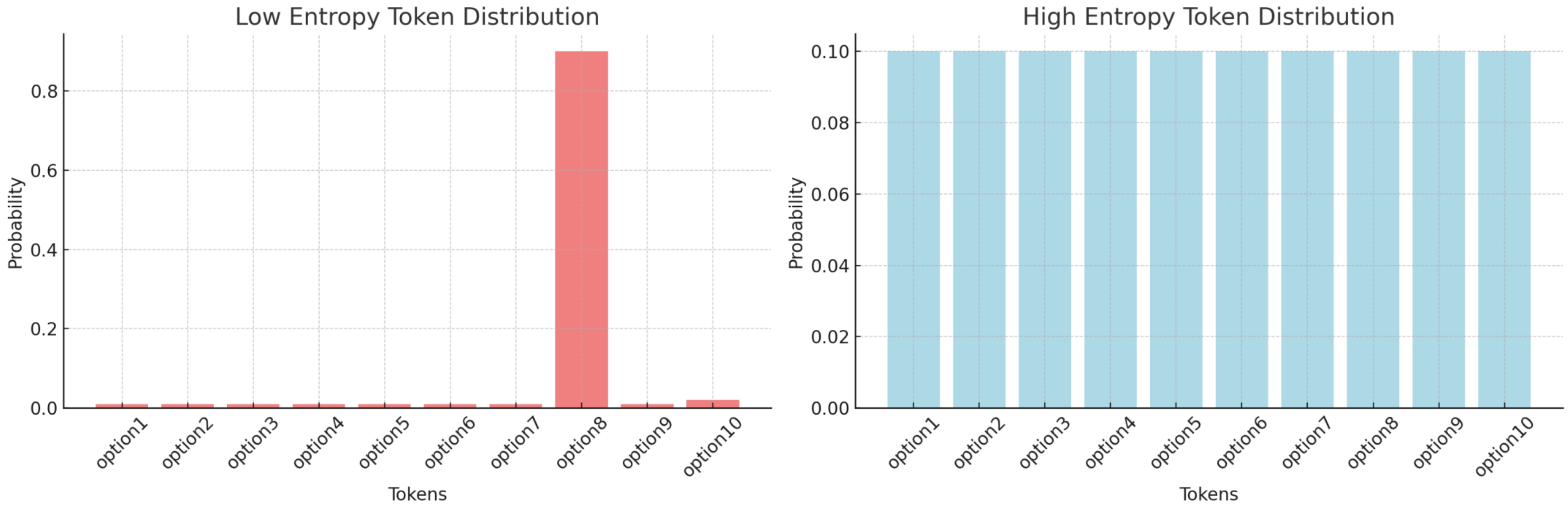
This distribution also has ~1 bit of entropy.

Intuition of Entropy – low entropy and high entropy

Imagine you have a bag of colored balls with different colors representing different outcomes.
If the bag contains balls of only one color, there is no surprise in picking a ball this situation has low entropy.

If the bag contains balls of many colors in equal proportions, picking a ball is highly unpredictable, corresponding to high entropy.

Extreme Example of
High-entropy and Low-entropy token decisions

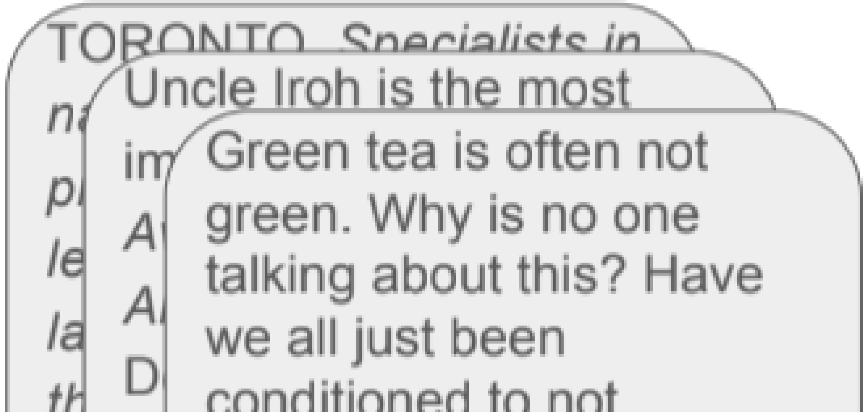


Intuitions of entropy in natural language generation tasks

In natural language generation, we often describe tasks to be on a spectrum from "open-ended" to "not open-ended." Instead you can think about it in terms of entropy:

Distribution: web documents
Entropy: high
Good generations: many

Distribution: sentence translations
Entropy: low
Good generations: few



J'aime vim → I like vim

The definition of entropy

We have a discrete set χ and a distribution $p(x)$

The entropy of the distribution is

$$\sum_{x \in \chi} p(x)(-\log(p(x)))$$

Sum over all elements

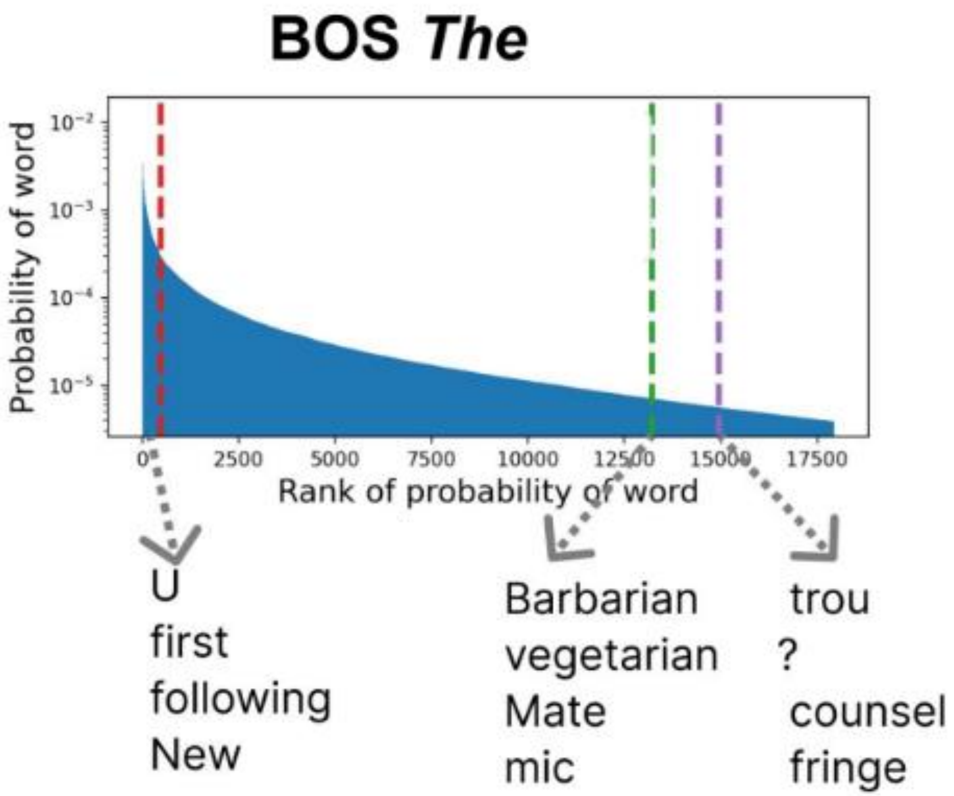
Probability of the element

Negative logarithm of the probability (lower probability -> larger negative log!)

High-entropy token decisions

Sometimes, the true next token distribution has many plausible options.

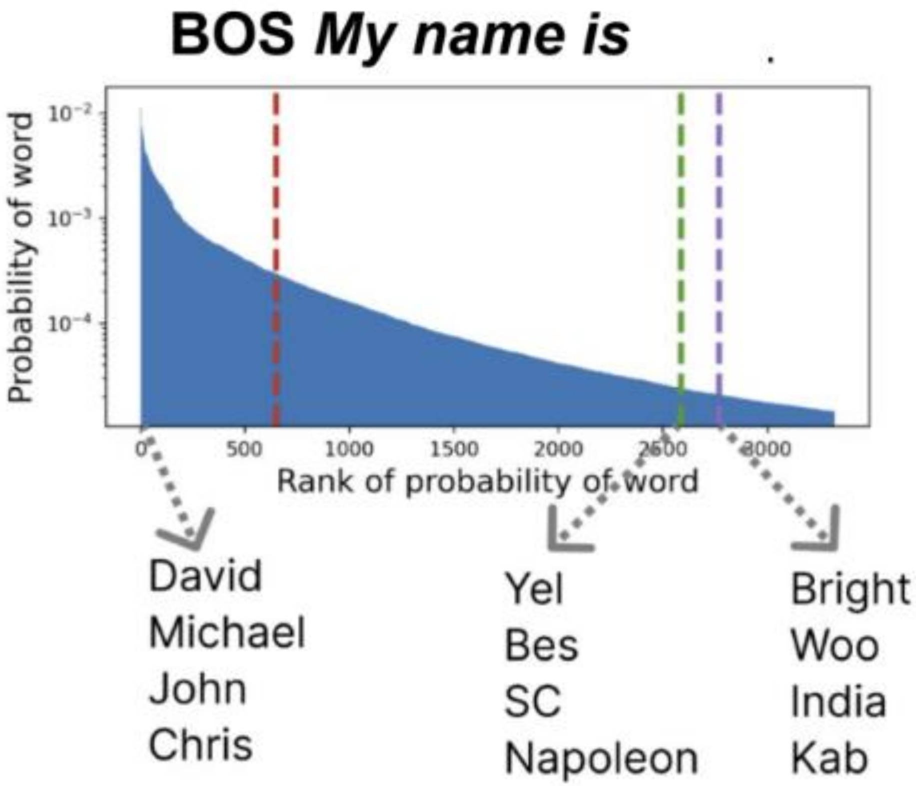
(BOS) - Beginning of the Sentence



BOS-> Beginning of Sentence (first token)

High-entropy token decisions

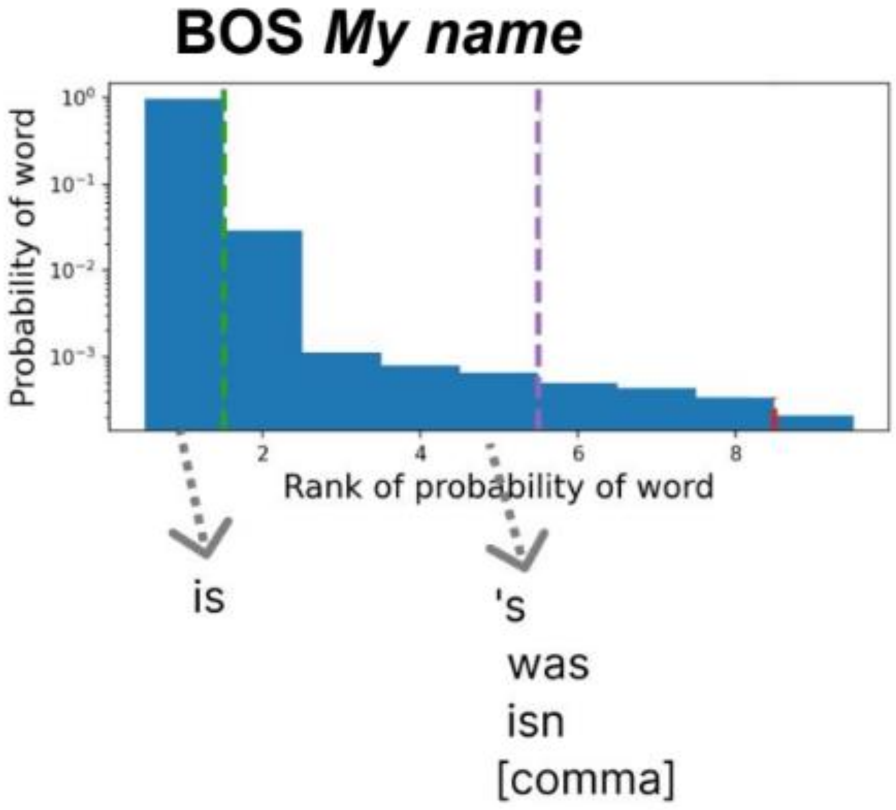
Sometimes, the true next token distribution has many plausible options.



(BOS) - Beginning of the Sentence

Low-entropy token decisions

Sometimes, the true next token distribution has few plausible options.



(BOS) - Beginning of the Sentence

Context lowers entropy

My Car

is
drives
was
has
runs

High entropy:
many plausible
options

It was so, so cold
this morning. My car

battery

(didn’t
start)

Fewer options:
Some are much
more likely than
before

Adding Context Reduces Entropy (Important for the concept of Prompting)

```python
prefix_no_context = 'They need to go to the'
```

```python
probabilities = get_next_word_probs(prefix_no_context)
top_token_probs, top_token_vals = torch.topk(probabilities, 10)

for token, prob in zip(top_token_vals, top_token_probs):
  print("%.3f" % prob.item(), tokenizer.decode(token))
```

```
0.020  police
0.019  people
0.017  top
0.016  next
0.011  same
0.010  hospital
0.010  polls
0.009  court
0.008  doctor
0.008  source
```

```python
entropy = torch.distributions.Categorical(probs = probabilities).entropy()
entropy.item()
```

```
7.403604507446289
```

Prefix with no context – High Entropy

```python
prefix_with_context = "They drank a lot of water. As a result, they need to go to the"
```

```python
probabilities = get_next_word_probs(prefix_with_context)
top_token_probs, top_token_vals = torch.topk(probabilities, 10)

for token, prob in zip(top_token_vals, top_token_probs):
  print("%.3f" % prob.item(), tokenizer.decode(token))
```

```
0.474  bathroom
0.129  hospital
0.118  doctor
0.059  toilet
0.023  gym
0.015  restroom
0.015  clinic
0.010  emergency
0.007  doctors
0.007  dentist
```

```python
entropy = torch.distributions.Categorical(probs = probabilities).entropy()
entropy.item()
```

```
2.640349864959717
```

Prefix with context – Low Entropy

Mode of a discrete distribution

We have a discrete set χ and a distribution $p(x)$

The mode of the distribution is

$$\underset{x \in \chi}{\arg max}\, p(x)$$

The element that achieves

the max over all elements

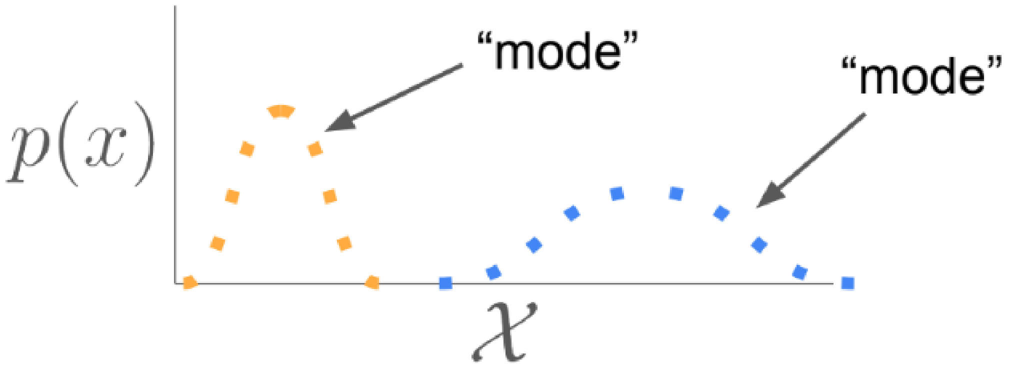
of the probability of the element

“Modes” of a distribution

We have a discrete set χ and a distribution $p(x)$, and a notion of similarity

Informally, a “mode” of a distribution is a set of similar examples which together compose a meaningful amount of probability. Similarity might be defined semantically, e.g., according to some embedding space.

(intuition only;
X-axis represents a semantic notion of similarity)

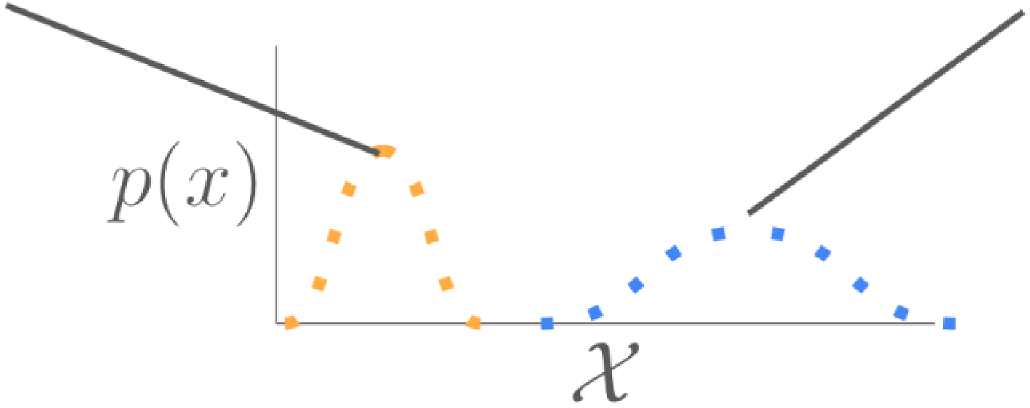


“Modes” of language

Here are some examples of semantically similar examples that could compose a mode

Lindy hop is a type of dance…
West coast swing was developed in the …
Waltz is a traditional dance …

Take two pounds of potatoes, …
Once the water has come to boil, …
First, generously season the …



Wanting the argmax: a time-honored tradition in NLP

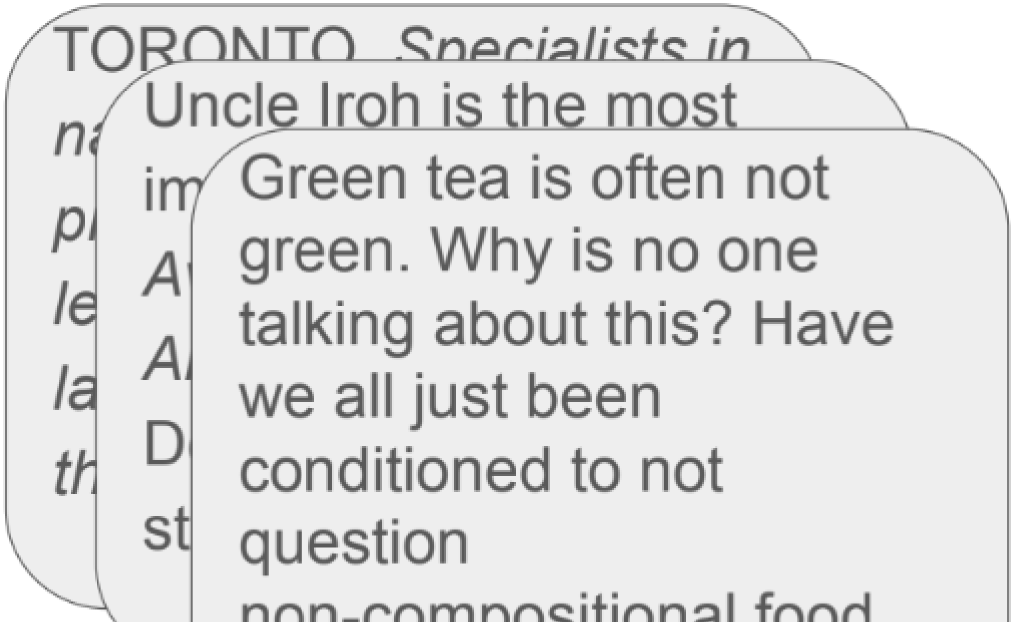
From parsing to machine translation, there's a long history of **wanting the most likely element of a set because it's the "best" (most probable) under the model.**

J'aime vim	I like vim	0.94
	I love vim	0.03

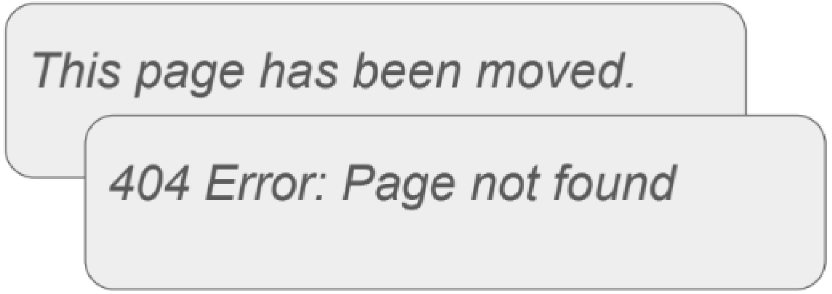
Entropy and the pitfalls of searching for the argmax

In a high entropy distribution, most probability tends to be on elements that don't necessarily look like the **mode (argmax).** Typical samples are **lower-probability**.

Most web documents



Hypothetical argmax-like documents



Failures of LM Estimation

- Often, researchers/developers are not working with super large LMs (availability or monetary restrictions). Smaller models exhibit some common shortcomings:
  - Tails of distribution are poorly estimated
  - Models seem to put a lot of probability mass on repetitions
  - Models are disproportionately biased towards predicting more lexically-frequent tokens*

- While these problems might go away with larger models and/or models trained on more data, we don't always have access to the means for creating such models. Thus, model quality needs to be taken into account when choosing how to generate text!

*Jason Wei, Dan Garrette, Tal Linzen, Ellie Pavlick (EMNLP 2021)

Decoding as a choice of Algorithm + Scoring Function

A Taxonomy of Decoding Strategies

- Given a distribution p_θ, how can we characterize decoding algorithms?

A Taxonomy of Decoding Strategies

- Given a distribution p_θ, how can we characterize decoding algorithms?
  - **Choice #1: Scoring Function**
    - How do we rank/score tokens (or entire strings) given some prefix?

$$s(y_{<t}, \mathcal{Y})$$

Scoring functions give us a score for each token in our vocab as a possible continuation of the current (set of) prefix(es)

A Taxonomy of Decoding Strategies

- Given a distribution p_θ, how can we characterize decoding algorithms?
  - **Choice #1: Scoring Function**
    - How do we rank/score tokens (or entire strings) given some prefix?
  - **Choice #2: Algorithm**
    - Given this score, how do we choose the next token in our string(s)?

```
def decoding_alg(scores) -> int:
    ...
    return token_id
```

The algorithm then provides the set of decision rules for choosing the next token given scores from $\mathcal{S}$

A Taxonomy of Decoding Strategies

- Given a distribution p_θ, how can we characterize decoding algorithms?
  - **Choice #1: Scoring Function**
    - How do we rank/score tokens (or entire strings) given some prefix?
  - **Choice #2: Algorithm**
    - Given this score, how do we choose the next token in our string(s)?

- Breaking it down like this allows us to better understand the different components of our design decisions, and potentially, how the desirable aspects from different decoding strategies can be combined

Common Choices of Scoring Functions

RPTU

Why do we need alternative scoring functions?

If current language generators were already flawless, couldn't we simply sample directly from them and obtain texts as fluent as the ones on which the generator was trained?

$$y_{perfect} \sim p_\theta(\cdot)?$$

Why do we need alternative scoring functions?

$$\cancel{y_{perfect} \sim p_\theta(\cdot)?}$$

Unfortunately, this is not the case

Different scoring functions were developed heuristically to fix some of the observed shortcomings

Why do we need alternative scoring functions?

The mean-seeking nature of the MLE **training objective**

exposure bias: training models only on ground-truth sequences

The **softmax function** forcing non-zero probabilities across the entire vocabulary

Smaller LMs (i.e., with fewer parameters or trained on less data) tend to have an **unreliable tail** and can't recover easily after sampling a low-probability token

Scoring functions with multinomial sampling

- In this section we'll choose **multinomial sampling** as the algorithm and explore different choices for the scoring function
- A scoring function is applied to a distribution over a vocabulary to obtain a modified distribution that satisfies some favourable property

$$s(y_{<t}, \mathcal{Y}) \ : \mathbb{R}^{\overline{\mathcal{V}}} \to \mathbb{R}^{\overline{\mathcal{V}}}$$
$$\tilde{p}(\cdot \mid y_{<t}) \propto s(p(\cdot \mid y_{<t}))$$

Two simple examples

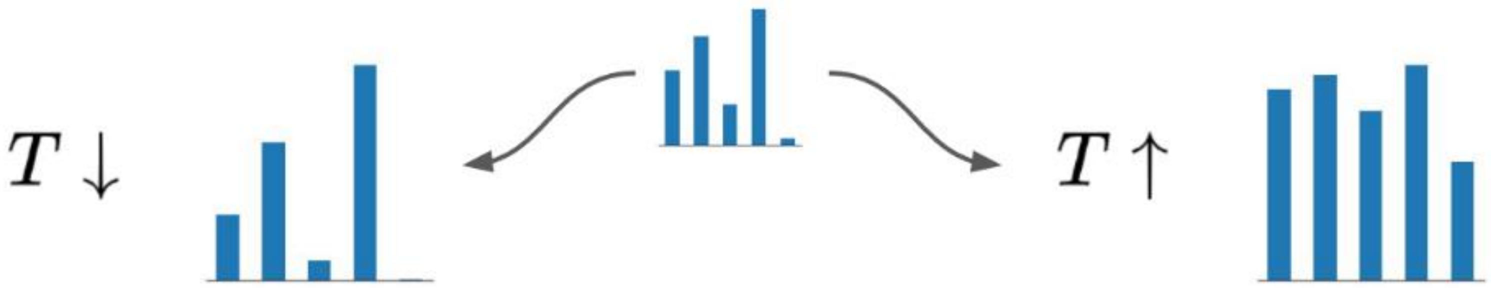
- **Ancestral sampling** is obtained when the scoring function is simply the identity function

$$s(y_{<t}, y) = p(y \mid y_{<t})$$

- **Temperature sampling** can be recovered as

$$s(y_{<t}, y, \tau) = p(y \mid y_{<t})^{\frac{1}{\tau}}$$

Consequence of temperature sampling



One day a cat decided to climb a tree but was caught by a dog. The cat was taken to the vet and the vet said the cat had a broken leg. The cat was then taken to the vet and the vet said the cat had a broken leg and was in a lot of pain. The cat was then .....

One day a cat decided to climb a tree but did not properly rock climb, Zoo workers put her (feet down), disastrously Raphael Staonymusensht October 28, 2014 at 12:38 PM Hot Cat written by .....

Peaky distributions lead to repetitive text

Flat distributions lead to incoherent text

Temperature Sampling code examples – Higher Temperature -> Higher Entropy

```
model_generation_config.temperature = 3.0

out = model.generate(
    input_ids,
    generation_config=model_generation_config,
)
tokenizer.decode(out.sequences[0])
```

I want to go to London to look, I thought he spoke badly— my opinion probably came way behind everyone else," Spouse commented from behind Tyllin seat in Lelouch seats across Monsegur seats before adding:"She has absolutely impeccable judgement..." Mr Levy

```
avg_token_entropy(out.scores)
```

```
4.434394788742066
```

Temperature Sampling code examples – Lower Temperature -> Lower Entropy

```
model_generation_config.temperature = 1.0

out = model.generate(
    input_ids,
    generation_config=model_generation_config,
)
tokenizer.decode(out.sequences[0])
```

```
avg_token_entropy(out.scores)
```

```
2.5590301859378815
```

I want to go to my parents house, but I want to go to my grandma's house. So... I'd like to go to my grandma's house, but it's still too far.

The Secret to Your Success: In my opinion, you must actually

Truncation-based scoring functions

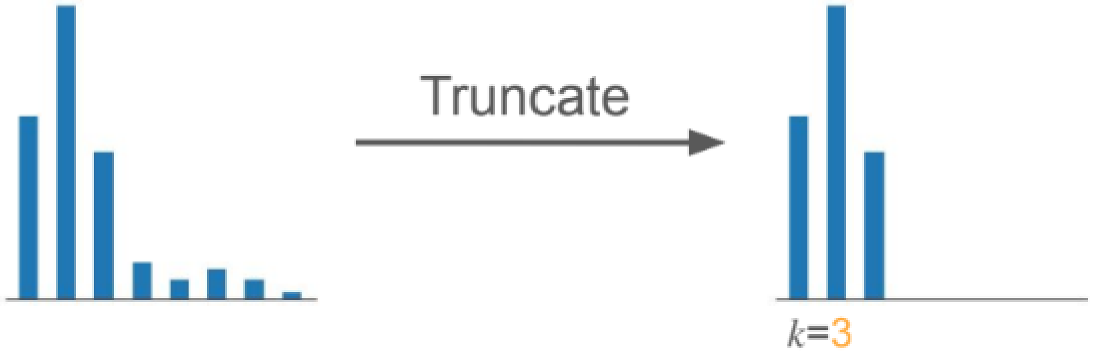
- The presence of an unreliable tail suggests the use of a scoring function that cuts off the tail and redistributes its mass to the head tokens
- We refer to scoring functions of this kind as **Truncation-based functions**

$$s(y_{<t}, \mathcal{y}) = \begin{cases} p\,(\mathcal{y}\,|y_{<t}) & if\ \mathcal{y} \in \mathcal{C}(p(\cdot|\,y_{<t})) \\ 0 & otherwise \end{cases}$$

$\boxed{\mathcal{C} : \mathbb{R}^{|\overline{\mathcal{V}}|} \to \mathcal{P}(\overline{\mathcal{V}})}$ is a function that selects the tokens to be kept

Top-k sampling

- Simply truncate the tail by selecting the k tokens with the largest probability



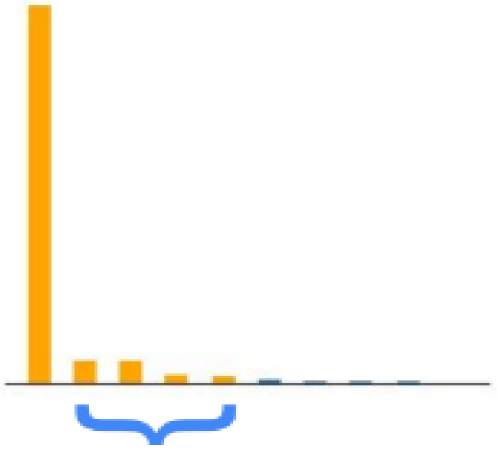
$$\mathcal{C}(p(.\mid y_t)) = argmax_{\mathcal{V}' \subseteq \overline{\mathcal{V}}} \sum_{y \in \mathcal{V}'} p(y \mid y_{<t}) \quad s.t\ |\mathcal{V}'| = k$$

- However, the use of a fixed k can become problematic across distributions with different entropies

Two failure modes of top-k sampling

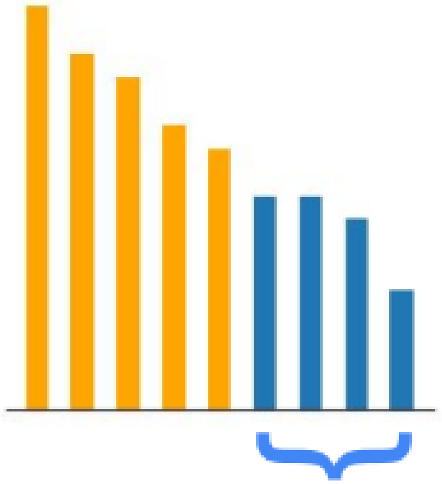
Suppose we fix k=5

p(. | hi, my name)



If the entropy of the distribution is low, we unnecessarily select tokens from its tail

p(. | I want to)



If the entropy of the distribution is high, we truncate tokens that are not in the tail

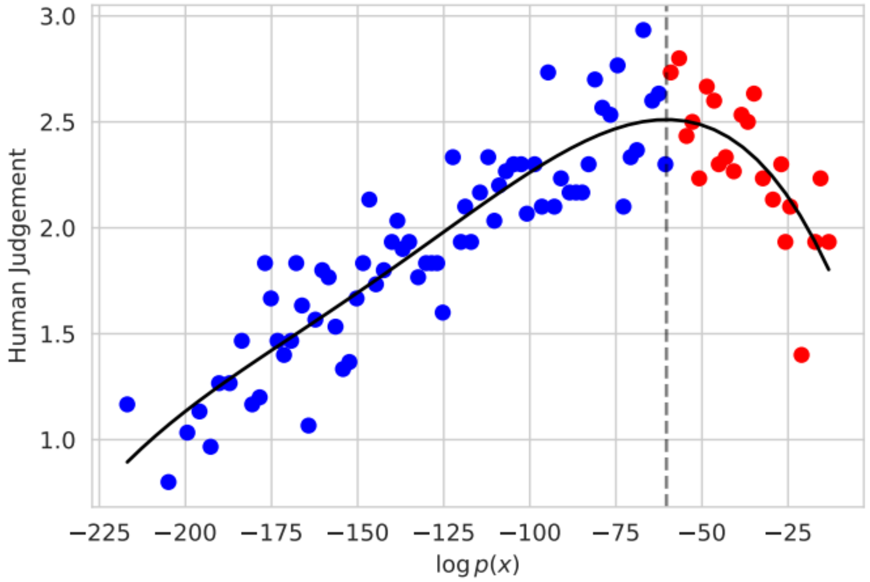
More modern truncation-based approaches try to set a value of k **dynamically**

Perplexity and generation quality

- Some scoring functions used the concept of surprisal (information content) to develop a sampling rule

$$\boxed{-\log(p(y_t \mid y_{<t}))}$$

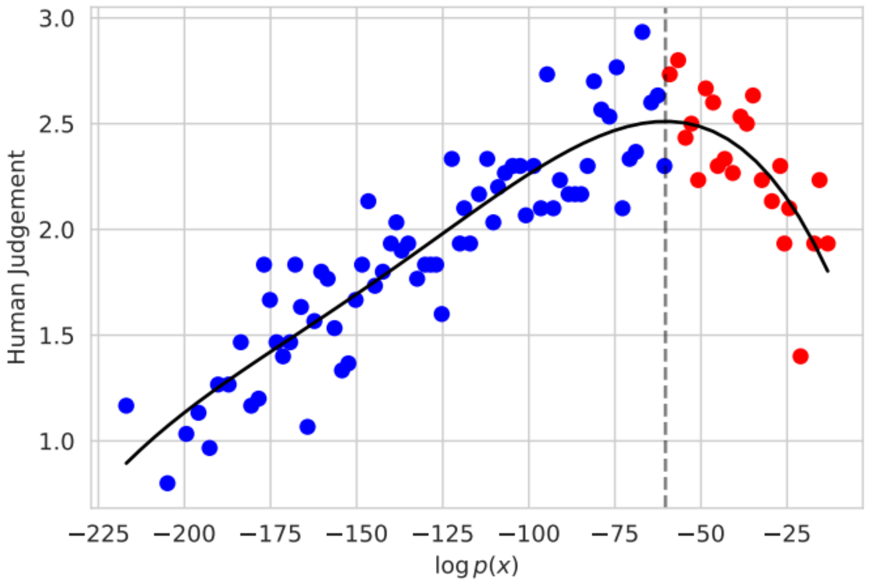
- High-quality text has been shown* to exhibit surprisal values that fall within a narrow range



*Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan (ACL 2021)

Perplexity and generation quality

- High-quality text has been shown* to exhibit surprisal values that fall within a narrow range



Method idea:

- Tune k so that the surprisal of the generated text is close to a target value T

*Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan (ACL 2021)

Mirostat

At each generation step

- Choose the top-k set to sample a token with a desirable surprisal

$$k = |\{ y \in \overline{\mathcal{V}} \ : \ -\log(p(y \mid y_{<t}) \leq 2\tau\}|$$

- Update τ using the surprisal of the sampled token y^*

$$\tau = \tau - \eta\,(-\log(p(y^* \mid y_{<t}) - \tau)$$

Sourya Basu, Govardana Sachitanandam Ramachandran, and Nitish Shirish Keskar (ICLR 2021)

Mirostat

Tune the size of the top-k set to sample a token with a desirable surprisal

$$k = |\{ y \in \overline{\mathcal{V}} \ : -\log(p(y \mid y_{<t}) \ \leq 2\tau \}|$$

Typical of low values of k

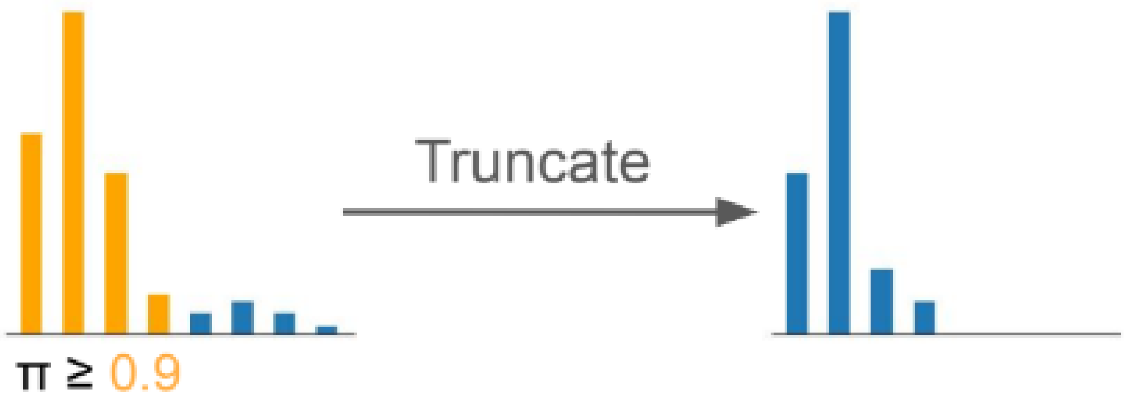
The dynamic value of k helps avoiding repetitions and incoherent text

Typical of large values of k

Sourya Basu, Govardana Sachitanandam Ramachandran, and Nitish Shirish Keskar (ICLR 2021)

Nucleus sampling

- Select tail size dynamically by only considering the **nucleus** of the distribution



- The nucleus contains the highest probability tokens with cumulative mass over π. Its size changes depending on the shape of the distribution

$$\mathcal{C}(p(\cdot \mid y_{<t})) = argmin_{\mathcal{V}' \subseteq \bar{\mathcal{V}}} |\mathcal{V}'| \quad s.t. \sum_{y \in \mathcal{V}'} p(y \mid y_{<t}) \geq \pi$$

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi (ICLR 2020)

RPTU

Decreasing the pi in Nucleus Sampling decreases the entropy

```python
model_generation_config.top_k = tokenizer.vocab_size

model_generation_config.top_p = 0.8

out = model.generate(
    input_ids,
    generation_config=model_generation_config,
)
tokenizer.decode(out.sequences[0])
```

Higher pi value (top_p) value higher entropy

```
'I want to go to Damascus to try and fight the regime. But you don\'t really want to go there, do you? And you don\'t really want to stay there because you\'re afraid of the regime or are you afraid of the regime?\n\n"And then'
```

```python
avg_token_entropy(out.scores)
```

```
2.0000397825241087
```

```python
model_generation_config.top_k = tokenizer.vocab_size

model_generation_config.top_p = 0.5

out = model.generate(
    input_ids,
    generation_config=model_generation_config,
)
tokenizer.decode(out.sequences[0])
```

Lower pi value (top_p) value lower entropy

```
'I want to go to the top of my climb. I want to climb down. I want to go up. I want to go up. I want to go up. I want to go up. I want to go up. I want to go up. I want'
```

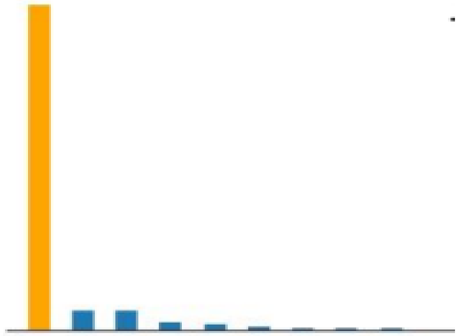
```python
avg_token_entropy(out.scores)
```

```
0.43632920145988463
```

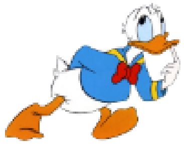
A failure mode of nucleus sampling

Suppose we fix π=0.8. In extreme cases, one of the failure modes of top-k sampling can also appear with top-π sampling

$$p(\cdot|Donald)$$



If the distribution is overly peaky, the nucleus can exclude perfectly valid continuations

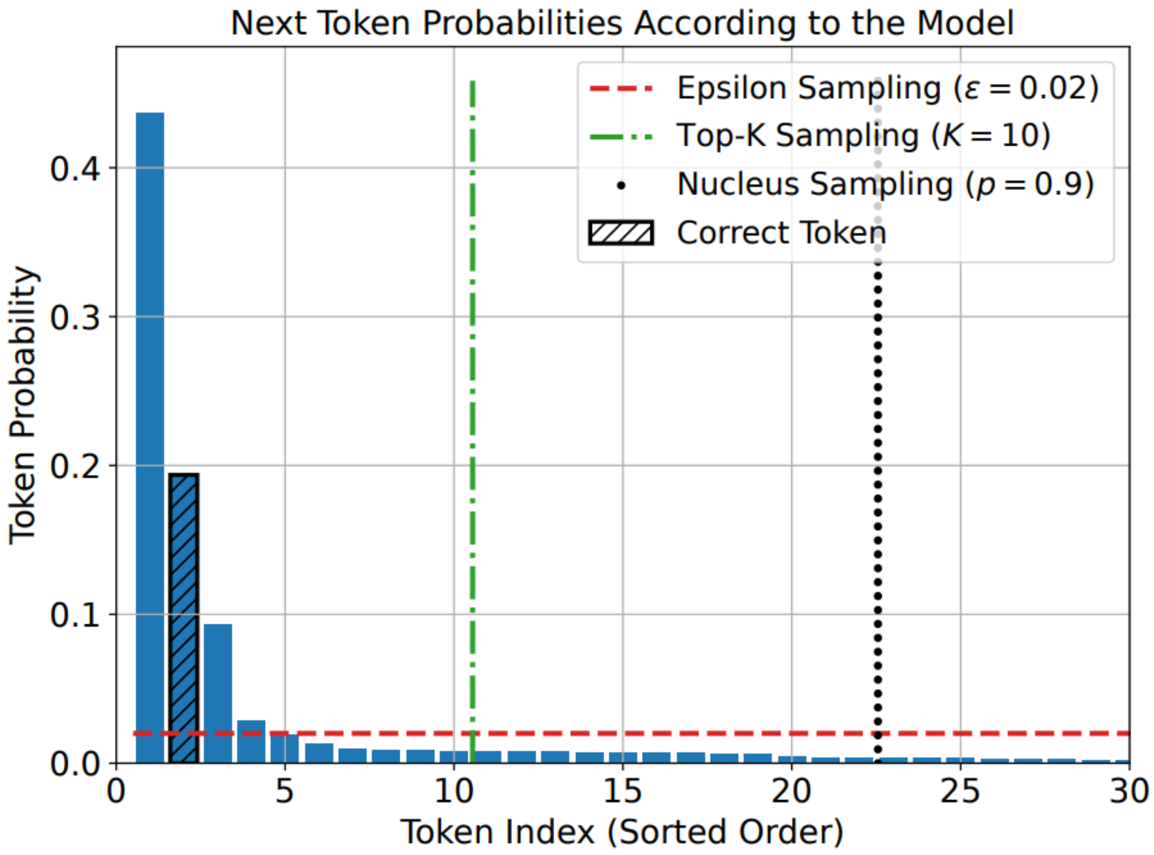
Why forget about  ?

Epsilon Sampling

- The absolute probability principle — that words outside the support of the true distribution have low probability — suggests a simple truncation algorithm:
- for some hyperparameter threshold ϵ allow any word with greater than ϵ probability.

- The scoring function shouldn’t truncate high-probability tokens

Epsilon Sampling vs Top-K Sampling vs Nucleus Sampling



Epsilon Sampling Rocks: Investigating Sampling Strategies for Minimum Bayes Risk Decoding for Machine Translation

Prompting: The Precursor to Controlled Generation

What is prompting?

- So far, we've looked at how to make text more fluent/coherent. But what if we want our model to
  - do a specific task: translation, summarization, question answering...
  - talk about a specific subject
  - produce text with a certain tone or style

- Prompting (a.k.a. in-context learning) is perhaps the simplest way of trying to steer our model in a certain direction: can broadly be described as the **augmentation of context with some sort of instructions or template** for the language model to follow when making predictions
  - We hope that the model follows these instructions or picks up on the patterns present

- We do not apply hard constraints in this setting: rather, we're kind of asking the model *nicely* to cooperate with our wishes!

Prompting in our framework

- Prompting can be viewed as a change to the scoring function used during decoding

$$s(y_{<t}, y) = p_\theta(y \mid \boldsymbol{y}_{<t})$$

$$\Downarrow$$

$$s(y_{<t}, y) = p_\theta(y \mid \boldsymbol{y}_{<t}, \boldsymbol{x})$$

Prompt

Prompting in our framework

- Prompting can be viewed as a change to the scoring function used during decoding

$$p_\theta(\cdot | The\ capital\ of\ Canada\ is\ )$$

$$p_\theta\left(\cdot \,\middle|\, \begin{matrix} \underbrace{The\ capital\ of\ Germany\ is\ Berlin.}_{\text{Prompt}} \\ The\ capital\ of\ Canada\ is \end{matrix}\right)$$

Why does prompting work?

Recall: Context (often) lowers entropy

My Car

is
drives
was
has
runs

High entropy:
many plausible
options

It was so, so cold
this morning. My car

battery

(didn't start)

Fewer options:
Some are much
more likely than
before

Why does prompting work?

Intuition: providing more context to the model can **narrow down the set of viable options** when it is predicting continuations

Prompting often lowers entropy for crucial token decisions

The capital of Canada is → Ottawa / my / Located / voting / in

High entropy: many plausible options

The capital of Germany is Berlin. The capital of Canada is → Ottawa

Fewer options: Some are much more likely than before

Prompting to solve lower entropy tasks

French: J’aime vim

English:

Translation

Toronto police reported ...

TL;DR:

Summarization

Types of Prompting: Demonstrations

For a task with input X and desired output Y, where we have examples of **x, y** pairs

- Concatenate together **x, y** pairs and give them to the model as input

```
Translate English to French:

sea otter => loutre de mer

peppermint => menthe poivrée

plush girafe => girafe peluche

cheese =>
```

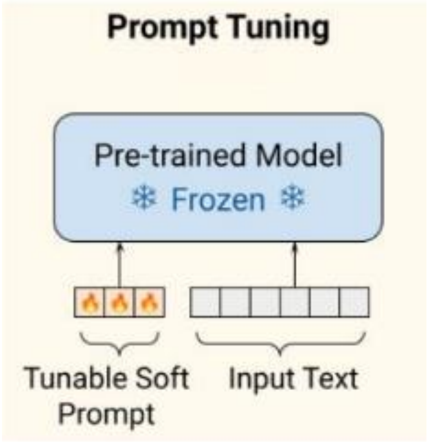
- **Caveats**: Many seemingly arbitrary choices can have a large effect* on model performance: Ordering, Language or symbols use to connect **x, y** examples

* Tony Z. Zhao, Eric Wallace, Shi Feng, Dan Klein, Sameer Singh (ICML 2021)

Types of Prompting: Learning Prompts

For a task with input X and desired output Y, where we have examples of **x**, **y** pairs

- Prompts can also be learned, akin to very lightweight fine-tuning.

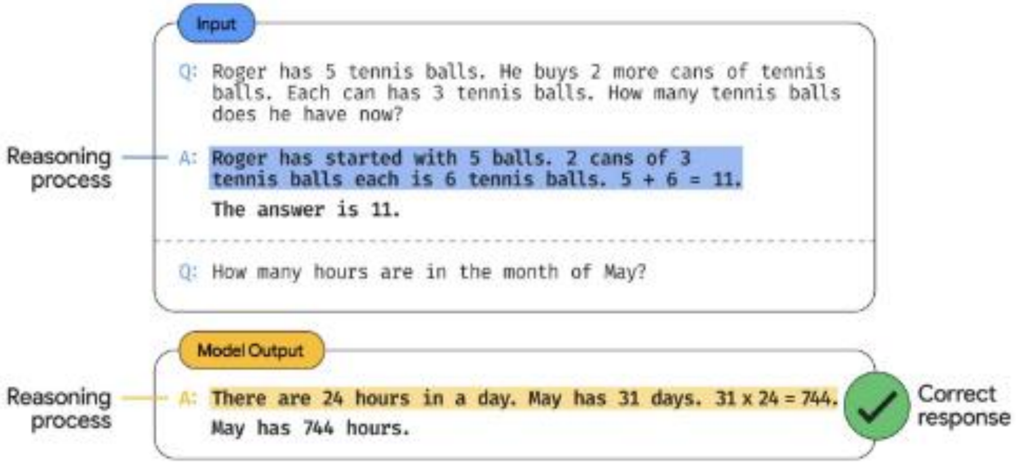


- **Caveats**: Less interpretable since the context that is being tuned does not necessarily correspond to a natural language string

Types of Prompting: Chain-of-Thought

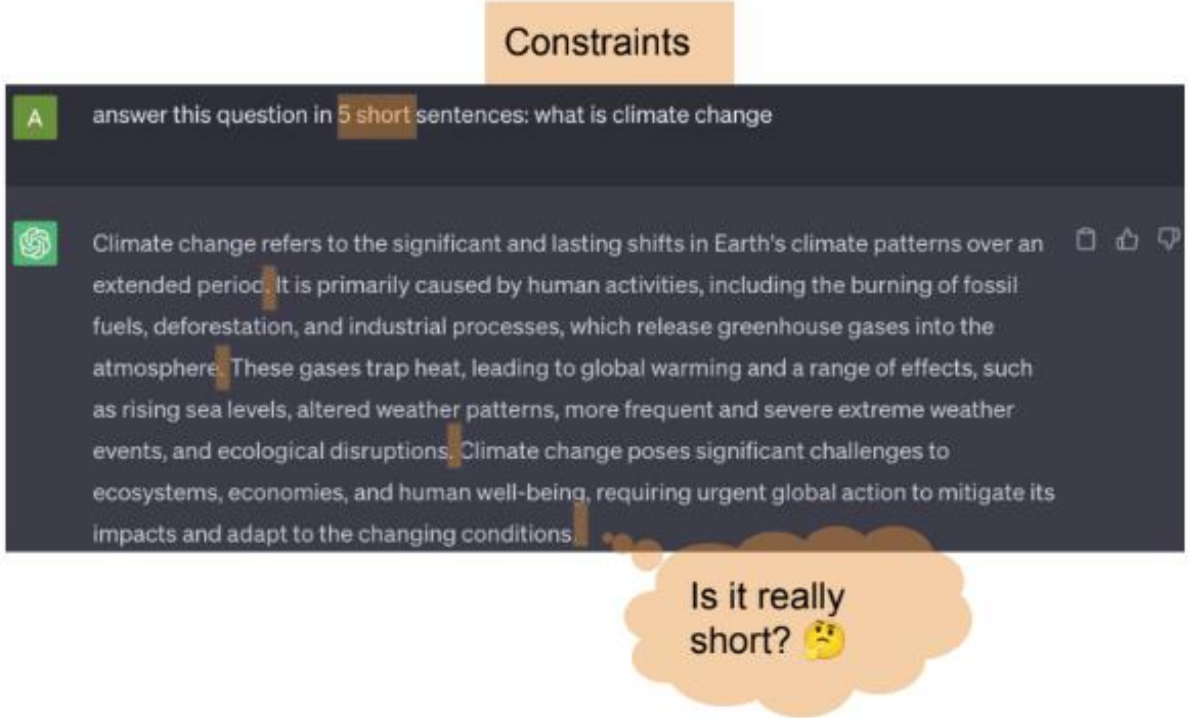
For a task with input X and desired output Y, where we have examples of **x**, **y** pairs

- When the task is more complicated/algorithmic, asking the model to "reason through" an answer can yield better results



- **Caveats**: Coming up with **x**, **y** pairs can be resource intensive, since each answer must include numerous reasoning steps

Is prompting the definite solution?



Current Trends in Language Generation: Larger Models and Fine-Tuning

Implications of Larger Models

- Many of the methods discussed today were developed/tested on smaller models, e.g., the sampling methods meant to alleviate undesirable behavior.
- Larger (and better) language models exhibit lots of behaviors that aren’t seen for their smaller counterparts.
  - Emergent behaviors
  - Much stronger in-context learning abilities
- Thus, for larger language models, it’s not clear that all of these methods will still work as intended. Without access to/resources for running experiments with the best models, this remains an open question!

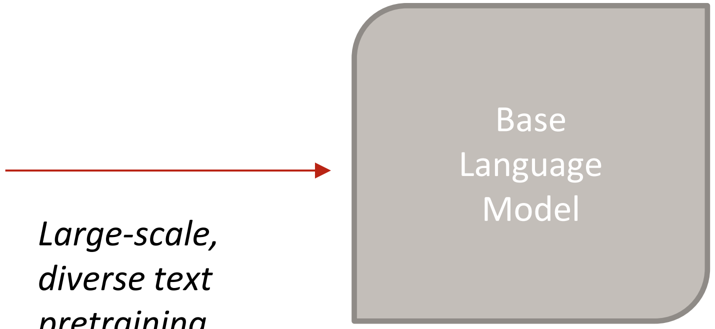
Fine-Tuning Approaches

- If we're willing to retrain a language model, we can try to explicitly impart certain characteristics on the model
- Still encounter the problem that quantifying the (largely qualitative) characteristics that we want is difficult
- **Instruction-tuning** and **reinforcement learning from human feedback** (RLHF) are two increasingly popular methods

Idea: Reorient the model using input, output pairs that we deem desirable

Instruction-Tuned and RLHF Language Models

Base pretraining on trillions of tokens from the web leads to a high-entropy LM.

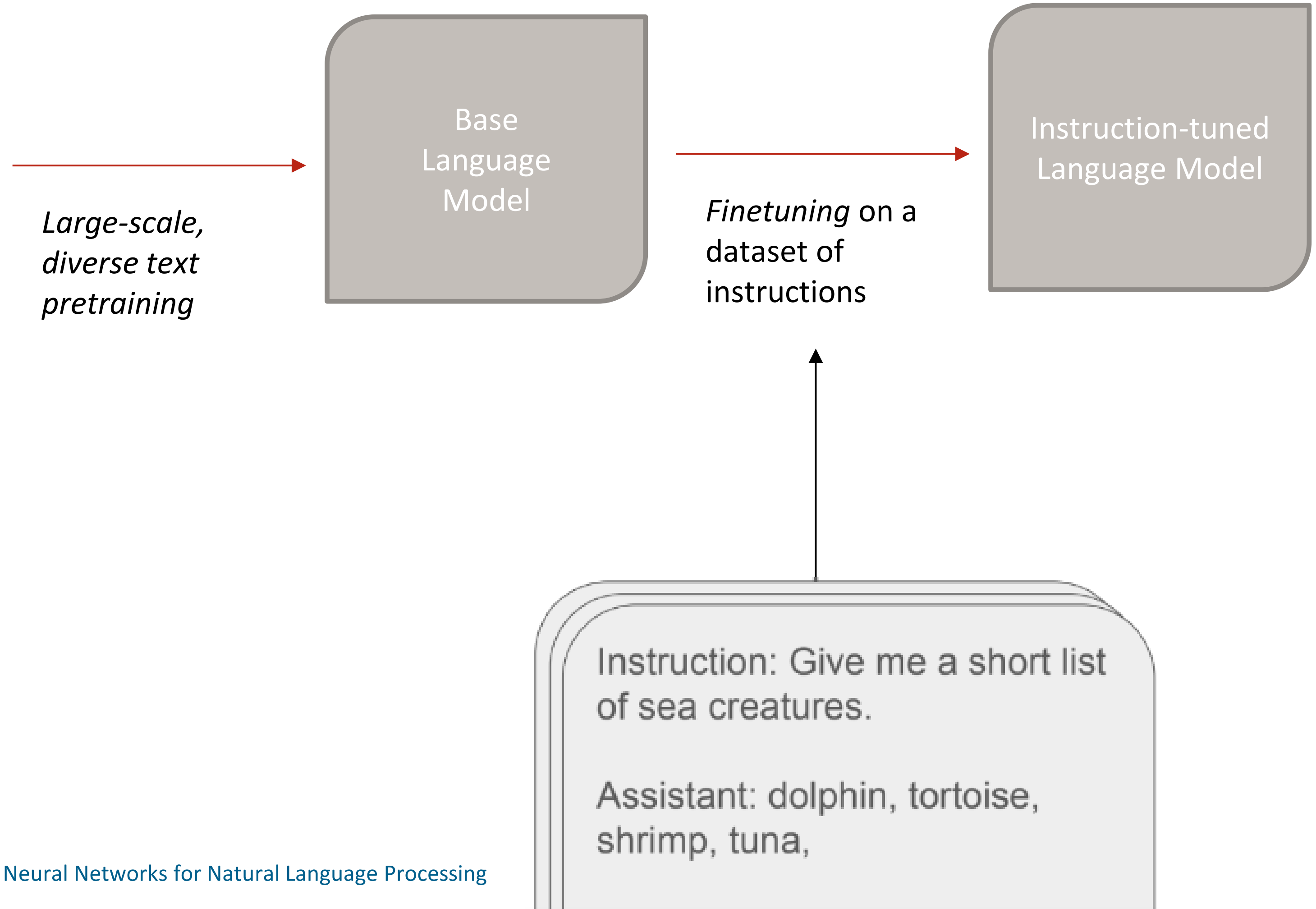


Large-scale, diverse text pretraining

The base language model pretraining process is done on ~trillions of tokens of text from a **diverse, high-entropy distribution.**

Instruction-Tuned and RLHF Language Models

Instruction-tuning reduces entropy and specifies the instruction-answer format.



Instruction tuning is just supervised learning on instruction-answer data. This is a lower-entropy distribution than the pretraining distribution.

After instruction-tuning, models don't need to be prompted to know they should be generating answers instead of general continuations.

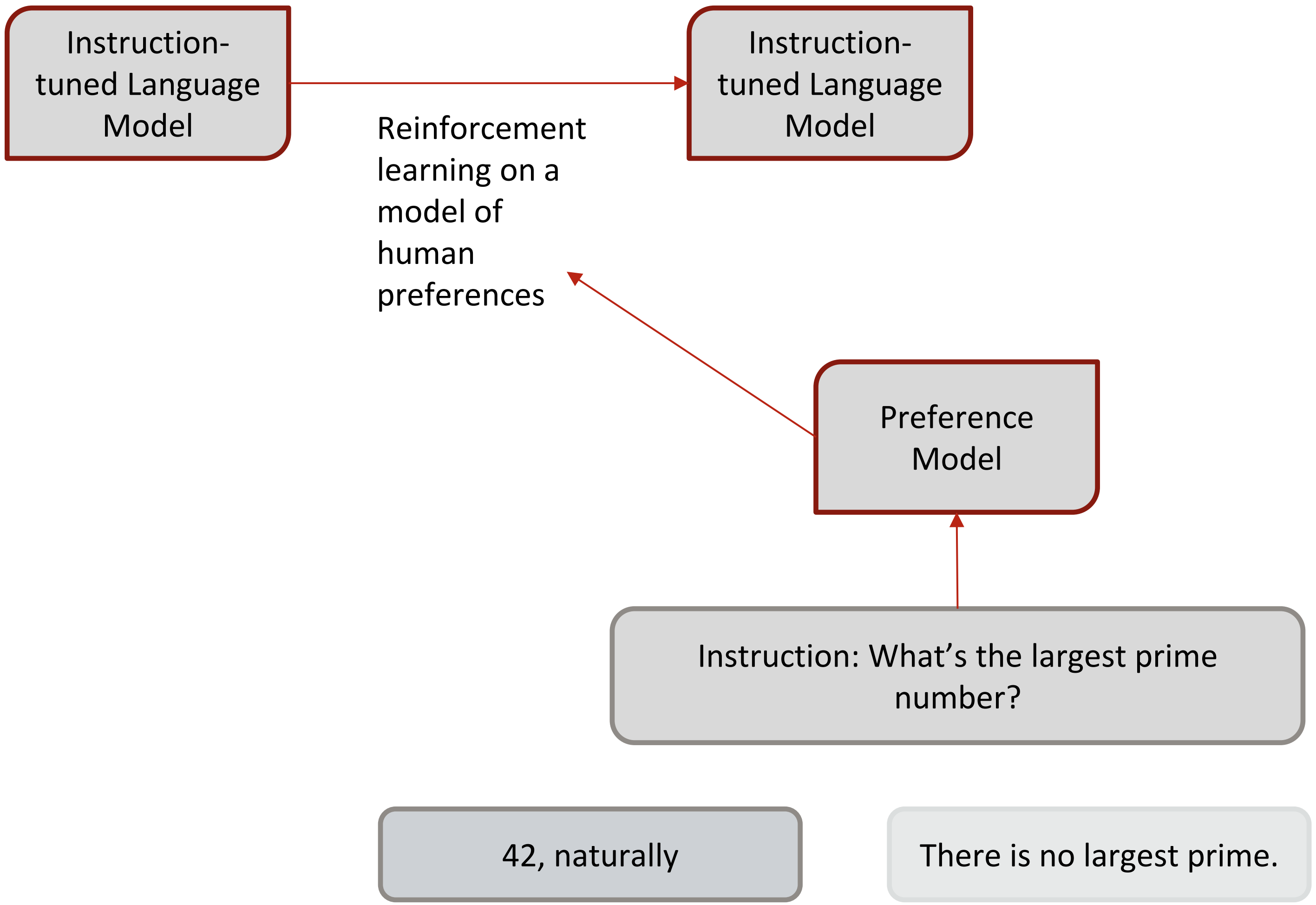
Instruction-Tuned and RLHF Language Models

RLHF *likely* also reduces entropy; makes “preferable” generations more likely

RLHF, or reinforcement learning from human feedback, is an exemplar of methods that learn from preference data to mitigate bad behaviour and emphasize preferred behaviour.

We speculate that RLHF reduces entropy.

Most (all) successful methods **regularize back to the instruction-tuned language model** to avoid mode collapse.



Conclusion

- High-entropy vs low-entropy generation
- Two components for decoding: score and algorithm
- Prompting and fine-tuning are ways of reducing entropy

Liu et al., 2023. Lost in the Middle: How Language Models Use Long Contexts

Next time: Reinforcement Learning for NLP

Liu et al., 2023. Lost in the Middle: How Language Models Use Long Contexts

References

Probability distributions over strings
Formal Aspects of Language Modeling (Cotterell et al., 2022)
Calibration, Entropy Rates, and Memory in Language Models (Braverman et al., 2019)
Elements of Information Theory (Cover and Thomas, 2006)

Sampling Methods
Hierarchical Neural Story Generation (Fan et al., 2018)
Mirostat: A Neural Text Decoding Algorithm that Directly Controls Perplexity (Basu et al., 2021)
Locally Typical Sampling (Meister et al., 2023)
Truncation Sampling as Language Model Desmoothing (Hewitt et al., 2022)

Controlled Generation
Self-Diagnosis and Self-Debiasing: A Proposal for Reducing Corpus-Based Bias in NLP (Schick et al., 2021)
FUDGE: Controlled Text Generation With Future Discriminators (Yang et al., 2021)

Measuring the quality of language generators and its challenge
Bleu: a Method for Automatic Evaluation of Machine Translation (Papineni et al., 2018)
BERTScore: Evaluating Text Generation with BERT (Zhang et al., 2020)

Acknowledgements

- “Generating Text from Language Models”: https://rycolab.io/classes/acl-2023-tutorial/
- Prathap Kashyap
- Mayank Chetan Ahuja