Machine Learning for Natural Language Processing

Generating Text from a Language Model

UNIVERSITY OF
GOTHENBURG

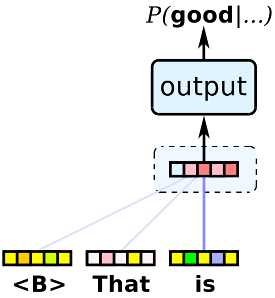
CHALMERS

Richard Johansson

richajo@chalmers.se

generating text from a language model

- assuming we have $P(X)$, how do we **generate** or "**decode**"?



- we will discuss the most common algorithms for **autoregressive** LMs
- there are several algorithms: this is an active research area

use cases: generating from a prompt

- given a prefix or "prompt", how do we find

$$\text{text}^* = \arg\max_{\text{text}} P(\text{text}|\text{prompt})$$

```
generated_ids = generate("NLP stands for natural", greedy_strategy, stopping_criterion=partial(has_n_sentences, n=2),

NLP stands for natural language processing. It is a method of processing language by using a computer to learn the mea
----------------------------------------------------------------
```

use cases: sampling

- if we have $P(X)$, how can we generate random texts?
- again, we might want to use a prompt

$$\text{text} \sim P(\text{text}|\text{prompt})$$

```
generated_ids = generate("Meatballs", random_strategy, stopping_criterion=partial(has_n_sentences, n=2))
```

```
Meatballs are a well-loved family recipe that typically serve as the centerpiece of spring gatherings in
-------------------------------------------------------------
```

```
generated_ids = generate("Meatballs", random_strategy, stopping_criterion=partial(has_n_sentences, n=2))
```

```
Meatballs. I made a weekly ritual of "drilling" out new ones, starting with how-to guides and deciding or
-------------------------------------------------------------
```

```
generated_ids = generate("Meatballs", random_strategy, stopping_criterion=partial(has_n_sentences, n=2))
```

```
Meatballs. Go for junk food and chips, no way you can show some athletic skill with a mound of food.
-------------------------------------------------------------
```

```
generated_ids = generate("Meatballs", random_strategy, stopping_criterion=partial(has_n_sentences, n=2))
```

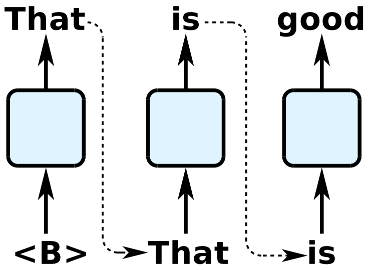
```
Meatballs, fried crickets and soggy greens. Another favourite – a crab-stuffed baked potato and chutney p
-------------------------------------------------------------
```

unsupported use cases

- it is difficult to solve “fill-in-the-blank” tasks with autoregressive LMs
- what is the most likely missing text?

NLP stands for ___ . It is a method…

first idea: greedy decoding

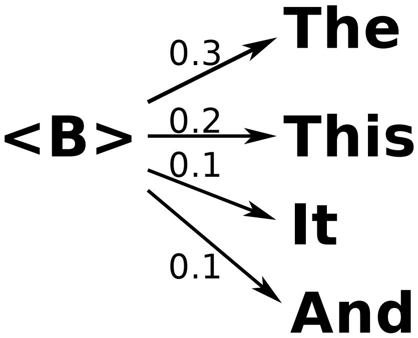


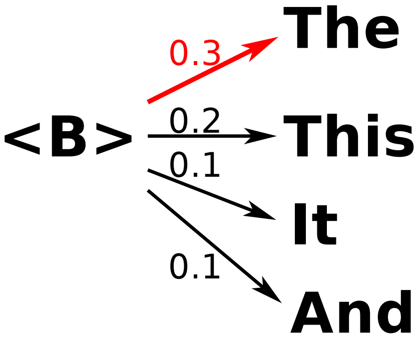
- select the **highest-scoring** alternative at each step

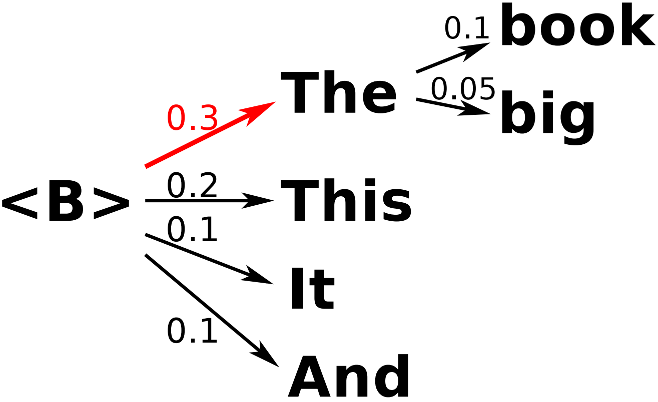
greedy decoding: pseudocode

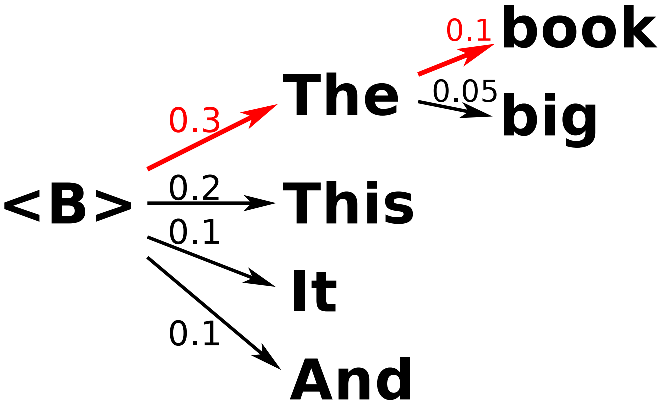
initialize $X = x_1, \ldots, x_m$ to some token sequence
for $i = m + 1, \ldots$ until some stopping criterion met
    $x_i \leftarrow \arg\max_x P(x|X)$
    append x_i to X
return X

<B>


The
0.3
<B>
0.2
This
0.1
It
0.1
And


<B>
0.3
The
0.2
This
0.1
It
0.1
And


<B>
0.3
The
0.2
This
0.1
It
0.1
And
0.1
book
0.05
big


<B>
0.3
0.2
0.1
0.1
The
This
It
And
0.1
0.05
book
big

pros and cons of greedy decoding

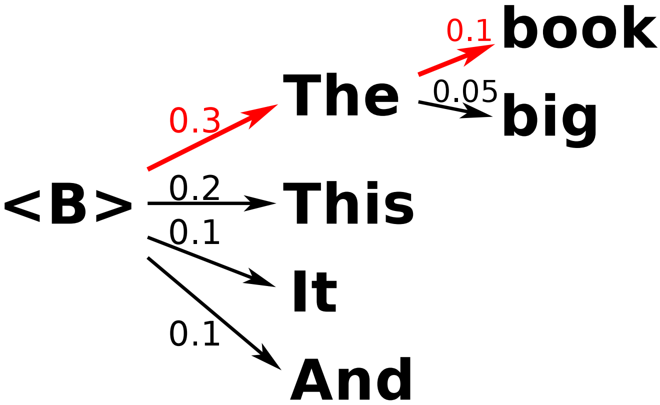
fast and easy to implement

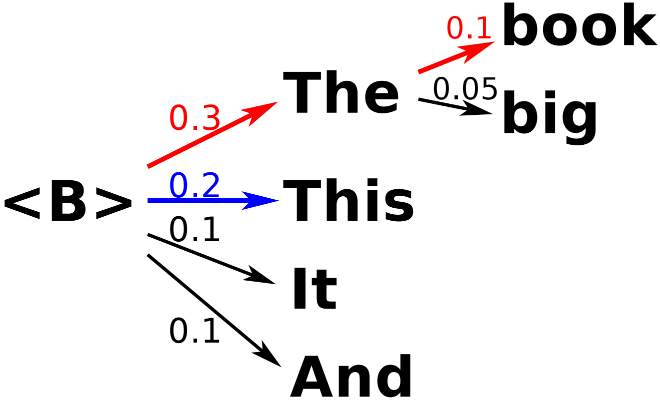
pros and cons of greedy decoding

fast and easy to implement

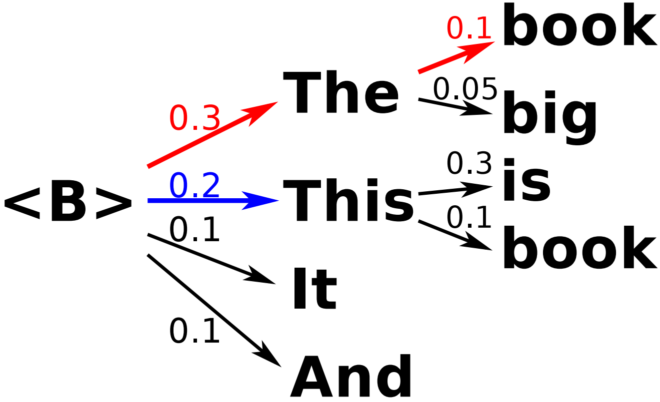
BUT:

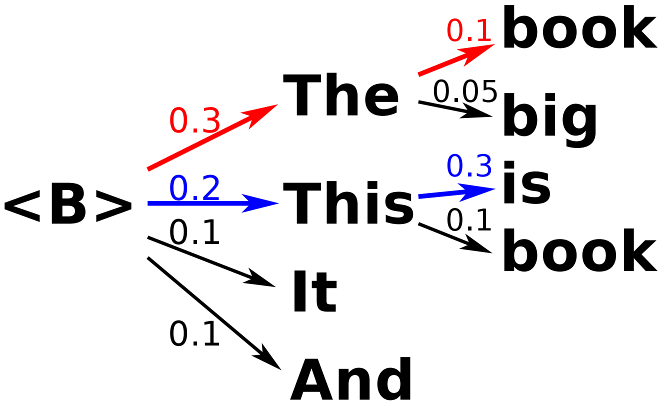
does not find the highest-scoring sequence


<B>
0.3
0.2
0.1
0.1
The
This
It
And
0.1
0.05
book
big



<B>
0.3
The
0.2
This
0.1
It
0.1
And
0.1
book
0.05
big

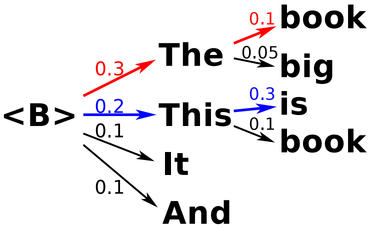

<B>
0.3
0.2
0.1
0.1
The
This
It
And
0.1
0.05
0.3
0.1
book
big
is
book



<B>
0.3
0.2
0.1
0.1
The
This
It
And
0.1
0.05
0.3
0.1
book
big
is
book

beam search decoding

problem: we can't consider all possible sequences

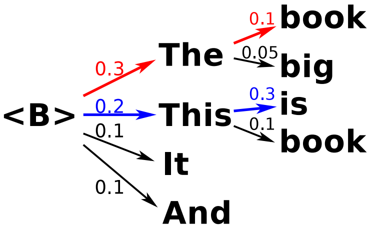


beam search decoding

problem: we can't consider all possible sequences

as an approximation, let's keep k **candidates** at each step

this idea is called **beam search**



beam search decoding: pseudocode

set the beam width k
initialize $X = x_1, \ldots, x_m$ to some token sequence
$B \leftarrow [X]$
for $i = m + 1, \ldots$ until some stopping criterion met
    $C \leftarrow [\,]$
    for each b **in** B
        compute $P(x|b)$
        add $b + [x]$ to C for all x in the vocabulary
    $B \leftarrow$ select k top-scoring candidates from C
return top-scoring beam from B

drawbacks of greedy and beam search decoding

- generated texts can be **bland and uninformative**
- the generation often has problems with **repetition**

drawbacks of greedy and beam search decoding

- generated texts can be **bland and uninformative**
- the generation often has problems with **repetition**

```
input_ids = tokenizer('Hello', return_tensors='pt').input_ids
outputs = model.generate(input_ids, num_beams=8, max_length=50, pad_token_id=0)
print(tokenizer.decode(outputs[0]))
```

```
Hello

I've been working on this project for a while now. I've been working on this project for a while now. I've
```

- some research describing these problems: (Holtzman et al., 2020), (Kulikov, 2022)
- repetition poorly understood theoretically (Fu et al., 2021)

THE CURIOUS CASE OF
NEURAL TEXT *De*GENERATION

Ari Holtzman[†‡] **Jan Buys**[§†] **Li Du**[†] **Maxwell Forbes**[†‡] **Yejin Choi**[†‡]
[†]Paul G. Allen School of Computer Science & Engineering, University of Washington
[‡]Allen Institute for Artificial Intelligence
[§]Department of Computer Science, University of Cape Town
{ahai,dul2,mbforbes,yejin}@cs.washington.edu, jbuys@cs.uct.ac.za

ABSTRACT

Despite considerable advances in neural language modeling, it remains an open question what the best *decoding strategy* is for text generation from a language model (e.g. to generate a story). The counter-intuitive empirical observation is that even though the use of likelihood as training objective leads to high quality models for a broad range of language understanding tasks, maximization-based decoding methods such as beam search lead to *degeneration* — output text that is bland, incoherent, or gets stuck in repetitive loops.

sampling

initialize $X = x_1, \ldots, x_m$ to some token sequence
for $i = m + 1, \ldots$ until some stopping criterion met
    $x_i \sim P(x|X)$
    append x_i to X
return X

drawbacks of sampling

```
input_ids = tokenizer('Hello', return_tensors='pt').input_ids
outputs = model.generate(input_ids, do_sample=True, max_length=50, pad_token_id=0)
print(tokenizer.decode(outputs[0]))
```

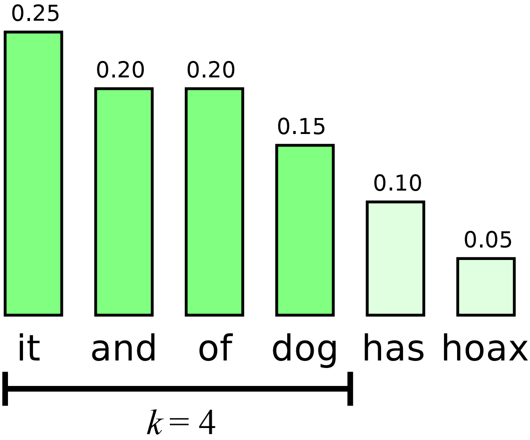
```
Hello - this is a big win for everyone of you who support C++11!

The winner of this week's poll will be taken here in June 2016.

If you enjoyed this article you will like my Facebook Page.<|endoftext|>
```

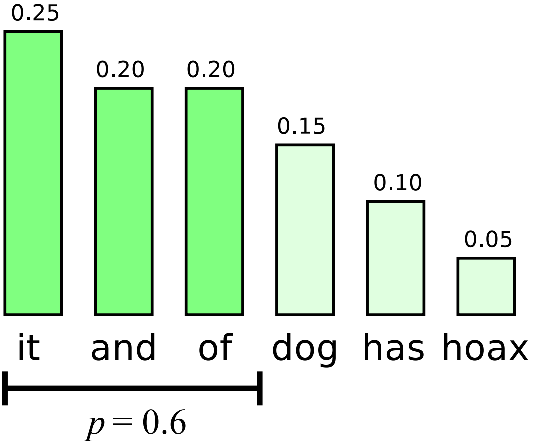
improving sampling: truncating the distribution (1)

in **top-k** sampling, we only include the k most probable words when sampling:



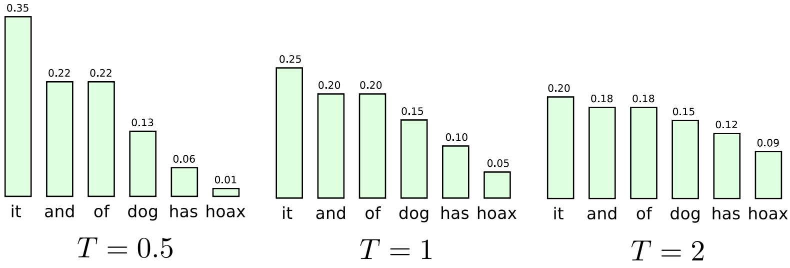
improving sampling: truncating the distribution (2)

in **top-p** or **nucleus** sampling (Holtzman et al., 2020), we select the most probable tokens with a probability mass of at least p:

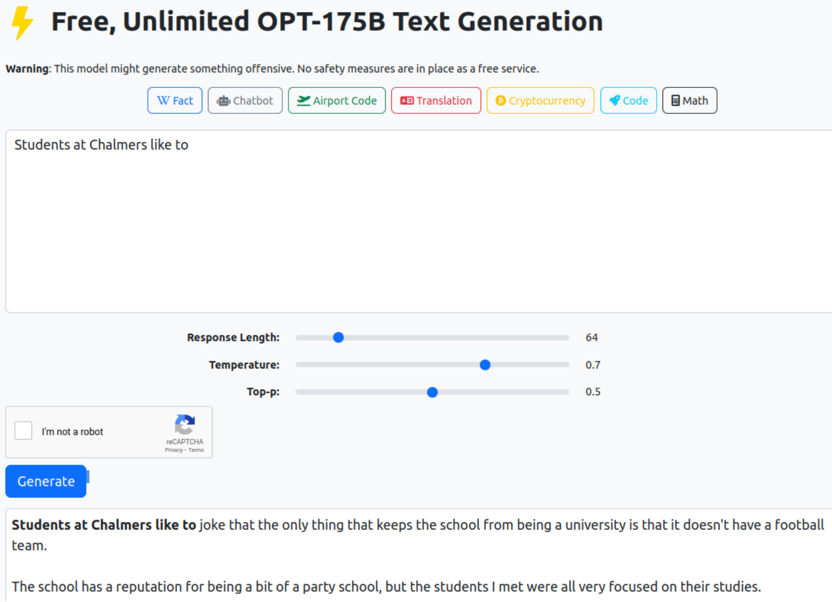


improving sampling: temperature

temperature T: divide the logits by T before applying the softmax



conclusion


Free, Unlimited OPT-175B Text Generation
Warning: This model might generate something offensive. No safety measures are in place as a free service.
Fact
Chatbot
Airport Code
Translation
Cryptocurrency
Code
Math
Students at Chalmers like to
Response Length:
64
Temperature:
0.7
Top-p:
0.5
I'm not a robot
reCAPTCHA
Privacy - Terms
Generate
Students at Chalmers like to joke that the only thing that keeps the school from being a university is that it doesn't have a football team.
The school has a reputation for being a bit of a party school, but the students I met were all very focused on their studies.

references

Z. Fu, W. Lam, A. M.-C. So, and B. Shi. 2021. A theoretical analysis of the repetition problem in text generation. In *AAAI*.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. 2020. The curious case of neural text degeneration. In *ICLR*.

I. Kulikov. 2022. *Characterizing and Resolving Degeneracies in Neural Autoregressive Text Generation*. Ph.D. thesis, New York University.