Transformers – Part 2

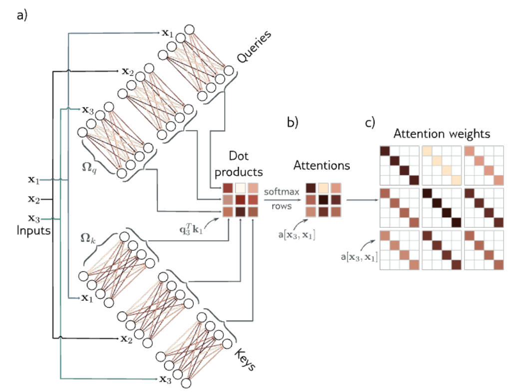
DL4DS – Spring 2025

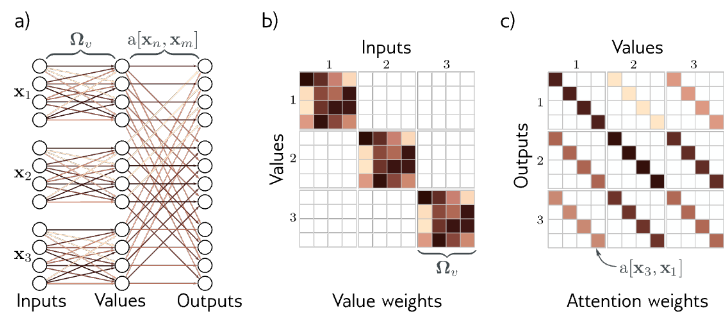
Today

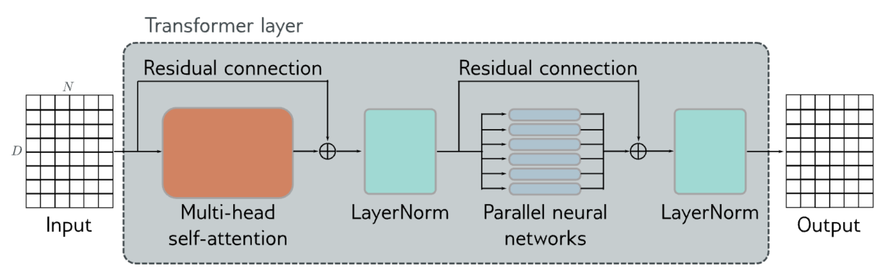
- Recap of Transformers Part 1
- Next token selection
- Transformers for Long Sequences
- Tokenization and Word Embedding

Recap From Part 1

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
  - Encoder
  - Decoder
  - Encoder-Decoder







Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models

Transformers

- Motivation
- Dot-product self-attention
- Applying Self-Attention
- The Transformer Architecture
- Three Types of NLP Transformer Models
    - Encoder
    - Decoder
    - Encoder-Decoder

slido

Which model flavor do you use for Named Entity Recognition?

ⓘ Start presenting to display the poll results on this slide.

slido

Which model flavor do you use for language translation?

ⓘ Start presenting to display the poll results on this slide.

slido

Which model flavor do you use for generating text, question answering, AI assistant?

ⓘ Start presenting to display the poll results on this slide.

3 Types of Transformer Models

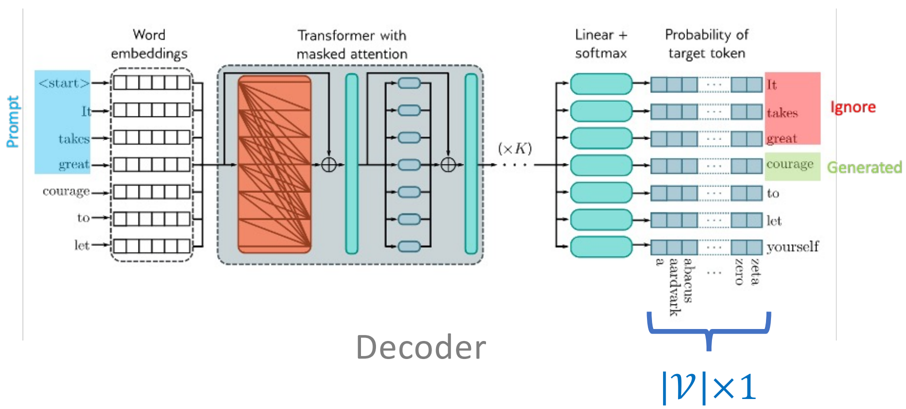
1. *Encoder* – transforms text embeddings into representations that support variety of tasks (e.g. sentiment analysis, classification)
   - ❖ Model Example: BERT

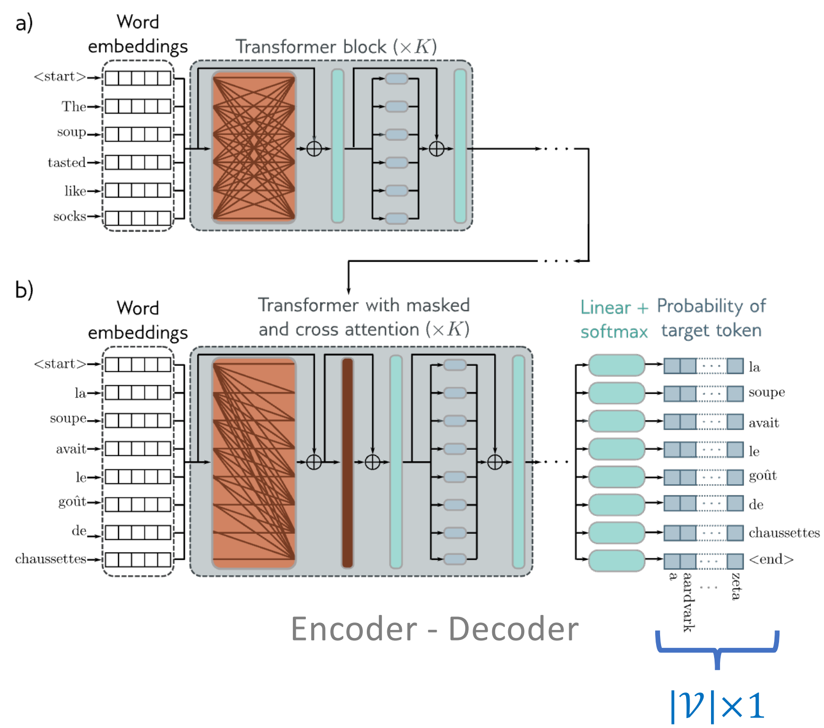
2. *Decoder* – predicts the next token to continue the input text (e.g. ChatGPT, AI assistants)
   - ❖ Model Example: GPT4, GPT4

3. *Encoder-Decoder* – used in sequence-to-sequence tasks, where one text string is converted to another (e.g. machine translation)

Next Token Selection

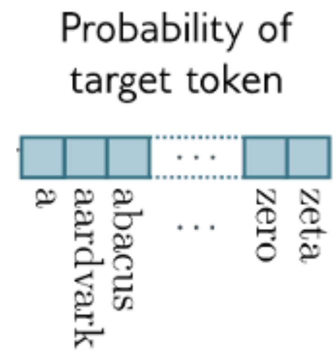
Next Token Selection





- Recall: output is a $|\mathcal{V}|\times 1$ vector of probabilities
- How should we pick the next token?
- Trade off between accuracy and diversity

Next Token Selection

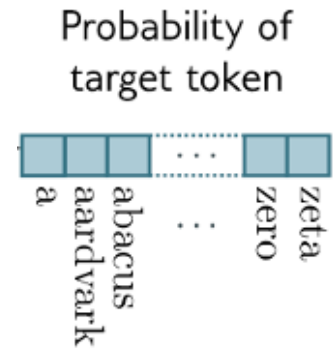


Recall: output is a $|\mathcal{V}| \times 1$ vector of probabilities

Selectin methods:

- Greedy selection
- Top-K
- Nucleus
- Beam search

Next Token Selection – Greedy



Pick most likely token (greedy)

Simple to implement. Just take the max().

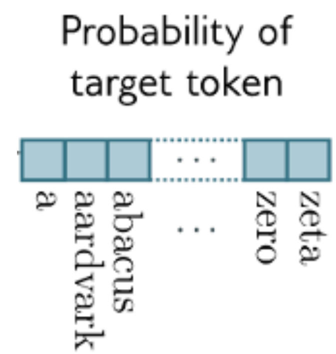
$$\hat{y}_t = \underset{w \in \mathcal{V}}{\operatorname{argmax}} \left[Pr(y_t = w | \hat{\mathbf{y}}_{<t}, \mathbf{x}, \boldsymbol{\phi}) \right]$$

```
# in PyTorch
outputs = model(inputs)
value, index = outputs.max(1)
```

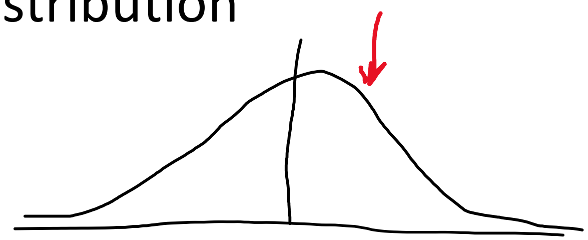
Might pick first token y_0, but then there is no y_1 where $\Pr(y_1 | y_0)$ is high.

Result is generic and predictable. Same output for a given input context.

Next Token Selection -- Sampling



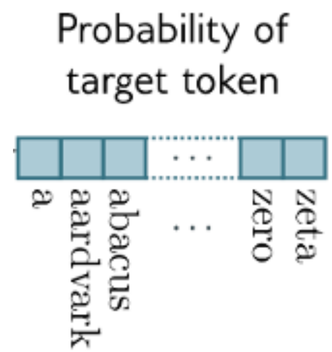
Sample from the probability distribution



Get a bit more diversity in the output

Will occasionally sample from the long tail of the distribution, producing some unlikely word combinations

Next Token Selection – Top *K* Sampling

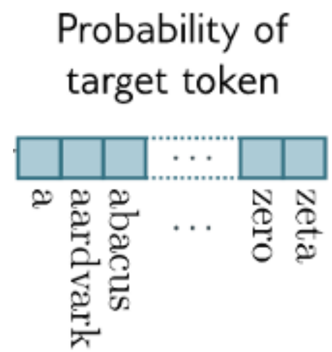


1. Generate the probability vector as usual
2. Sort tokens by likelihood
3. Discard all but top *k* most probable words
4. Renormalize the probabilities to be valid probability distribution (e.g. sum to 1)
5. Sample from the new distribution

Diversifies word selection

Depends on the distribution. Could be low variance, reducing diversity

Next Token Selection – Nucleus Sampling



Instead of keeping top-*k*, keep the top *p* percent of the probability mass.

Choose from the smallest set from the vocabulary such that

$$\sum_{w \in V^{(p)}} P(w|\mathbf{w}_{<t}) \geq p.$$

Diversifies word selection with less dependence on nature of distribution.

Depends on the distribution. Could be low variance, reducing diversity

Next Token Selection – Beam Search

Commonly used in *machine translation*

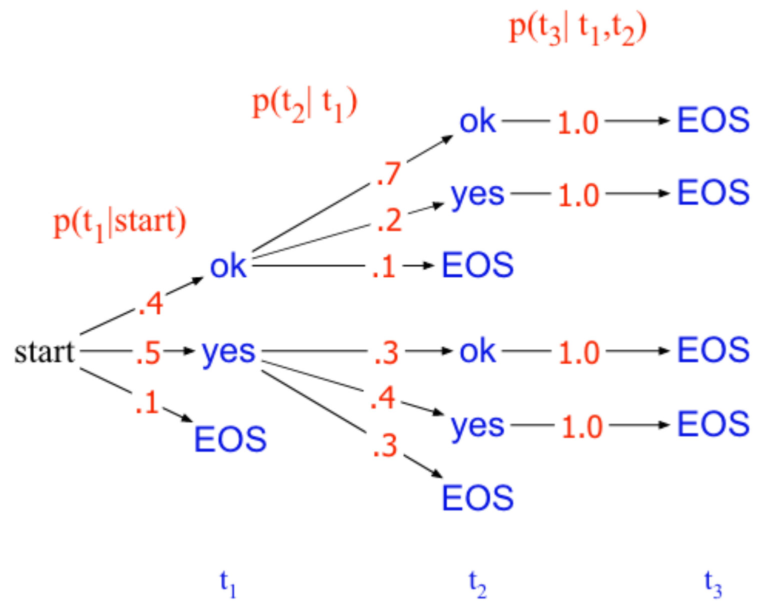
Maintain multiple output choices and then choose best combinations later via tree search

V = {yes, ok, <eos>}

We want to maximize $p(t_1, t_2, t_3)$.

Greedy: $0.5 \times 0.4 \times 1.0 = 0.20$

Optimal: $0.4 \times 0.7 \times 1.0 = 0.28$



D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search

But we can't exhaustively search the entire vocabulary

Keep k tokens (beam width) at each step

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

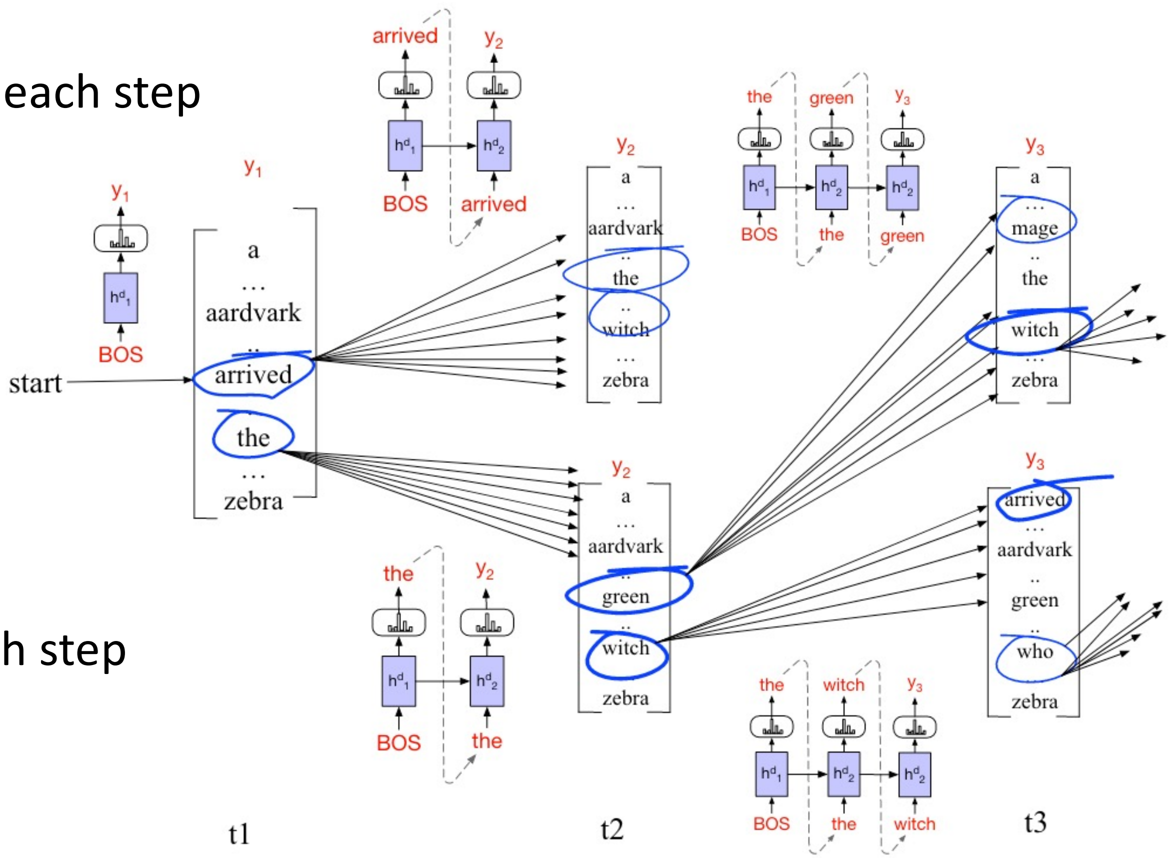
Next Token Selection – Beam Search

BOS: Beginning of Sentence token

Keep k tokens at each step

E.g. k = 2

Prune to k at each step

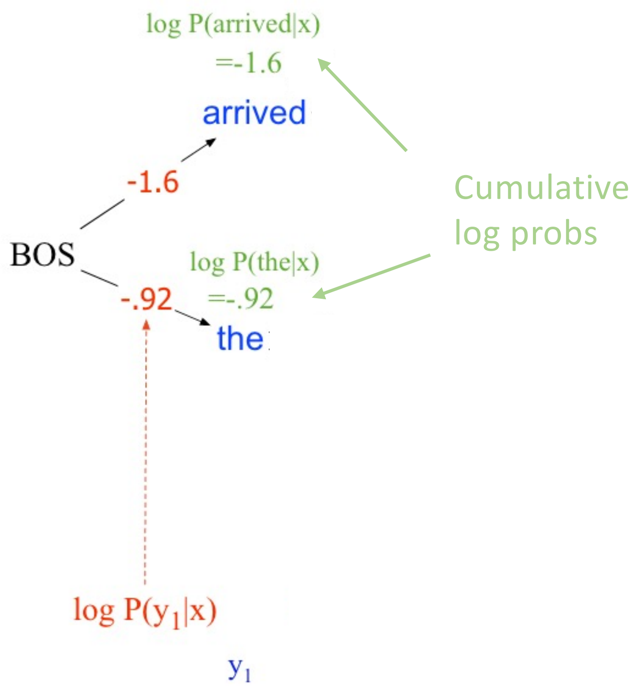


D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)

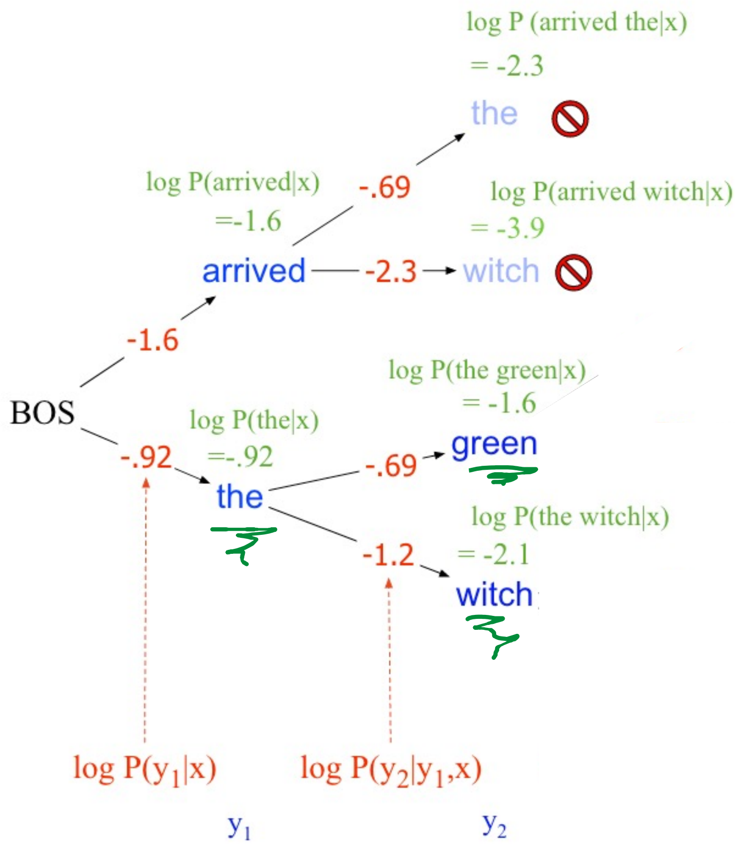
Calculated with *log probabilities* and add

Pick the top 2 tokens.



D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)



Then pick the next 2 from each of the first 2 tokens.

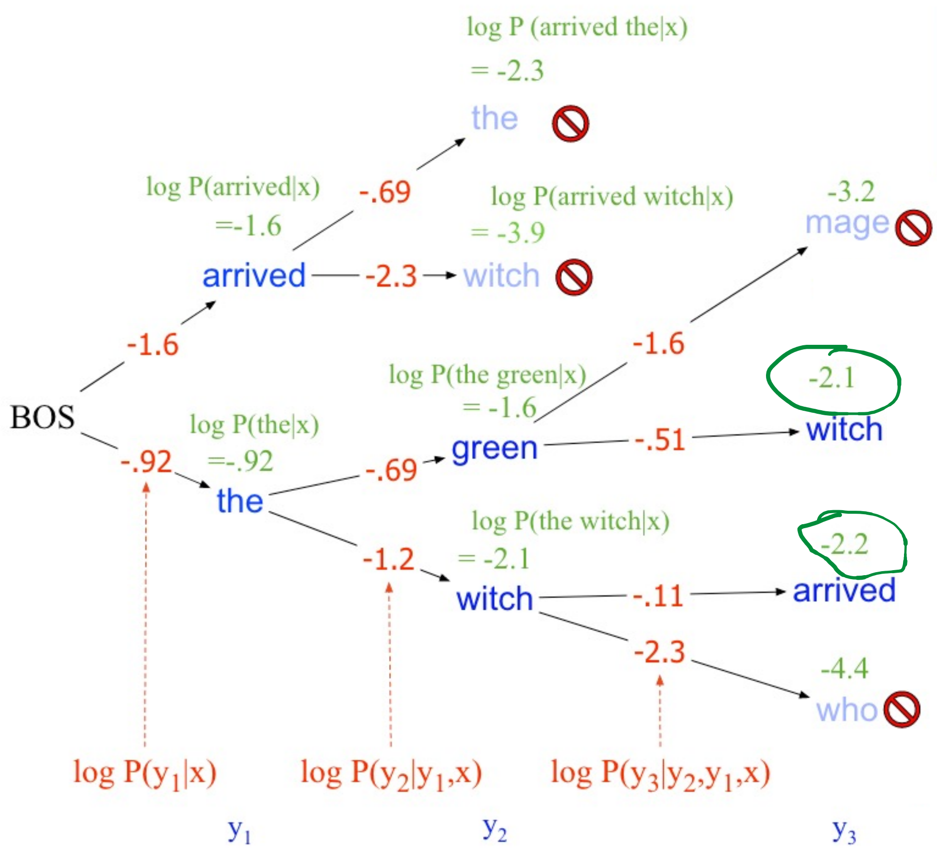
Calc cumulative log probs:
-1.6 -.69 = -2.3
-1.6 -2.3 = -3.9

-.92 -.69 = -1.6
-.92 -1.2 = -2.1

Pick the 1st token with highest log probability.

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)



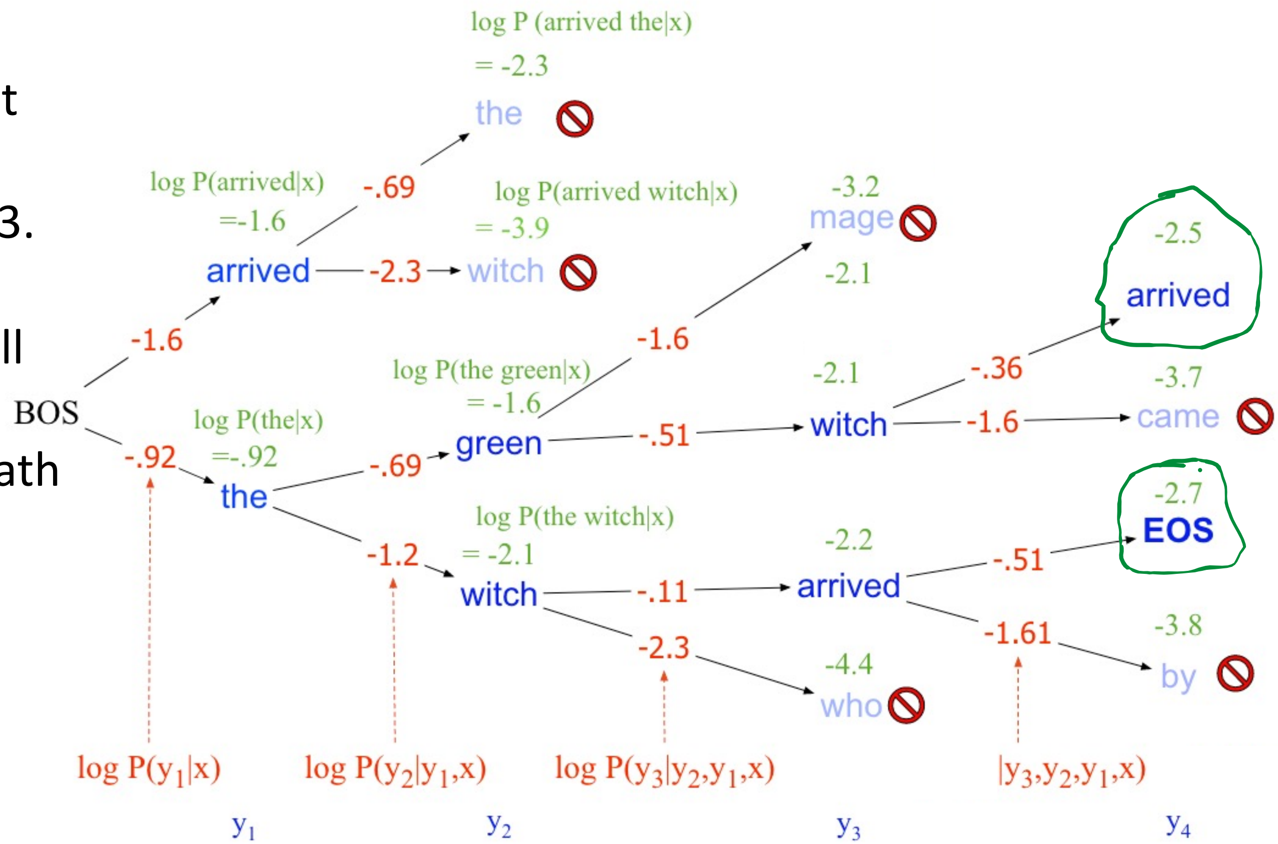
Then generate the next 2 tokens from each of the y2 and pick the 2 highest log probability paths.

D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search (k=2)

Continue to predict the next 2 highest from each of the y3.

We hit EOS, but still have a lower cumulative prob path

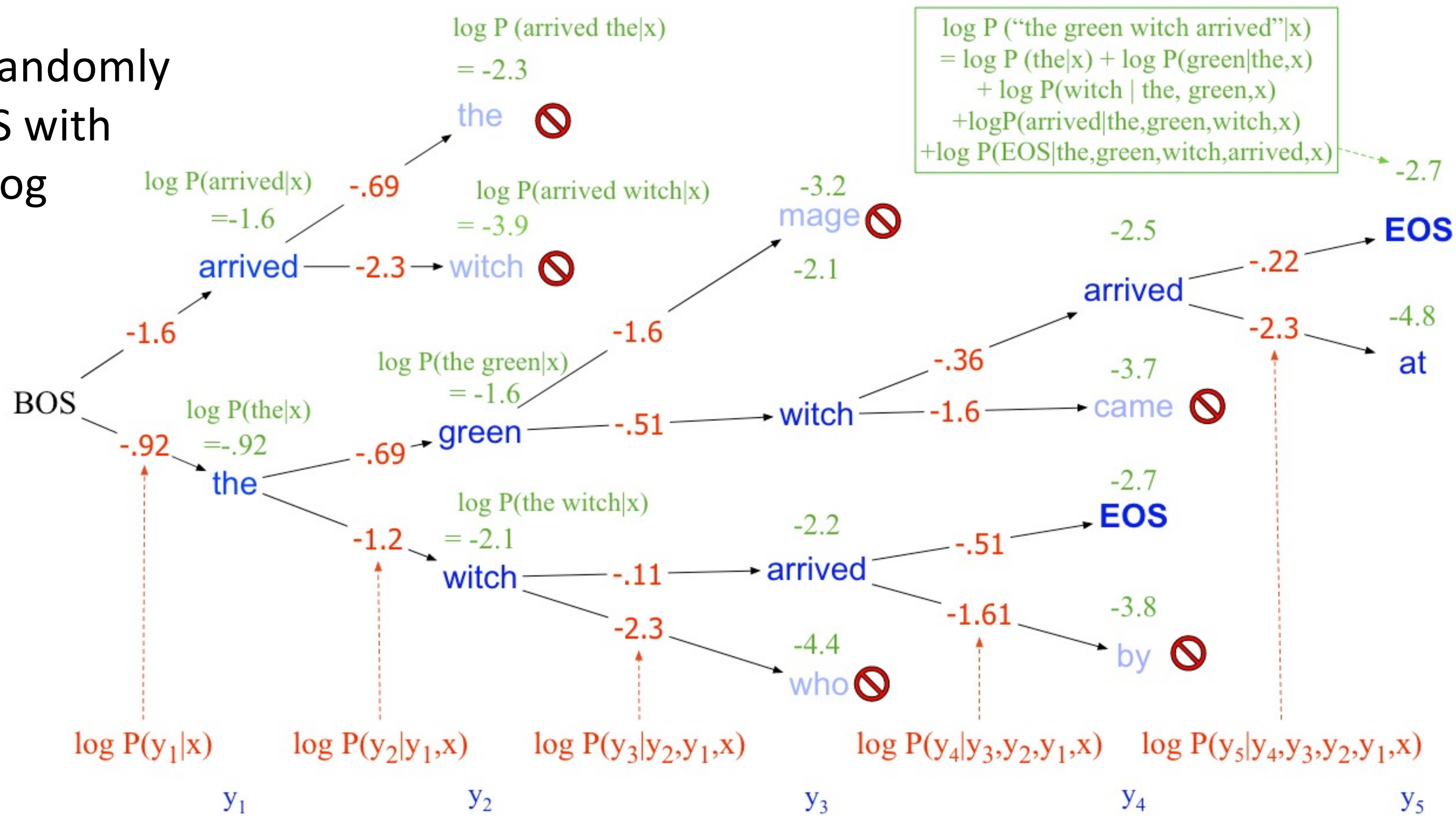


D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

Next Token Selection – Beam Search

We have 2 paths randomly terminating at EOS with same cumulative log probabilities.

Randomly pick 1.



D. Jurafsky and J. H. Martin, *Speech and Language Processing*. 2024. https://web.stanford.edu/~jurafsky/slpdraft/

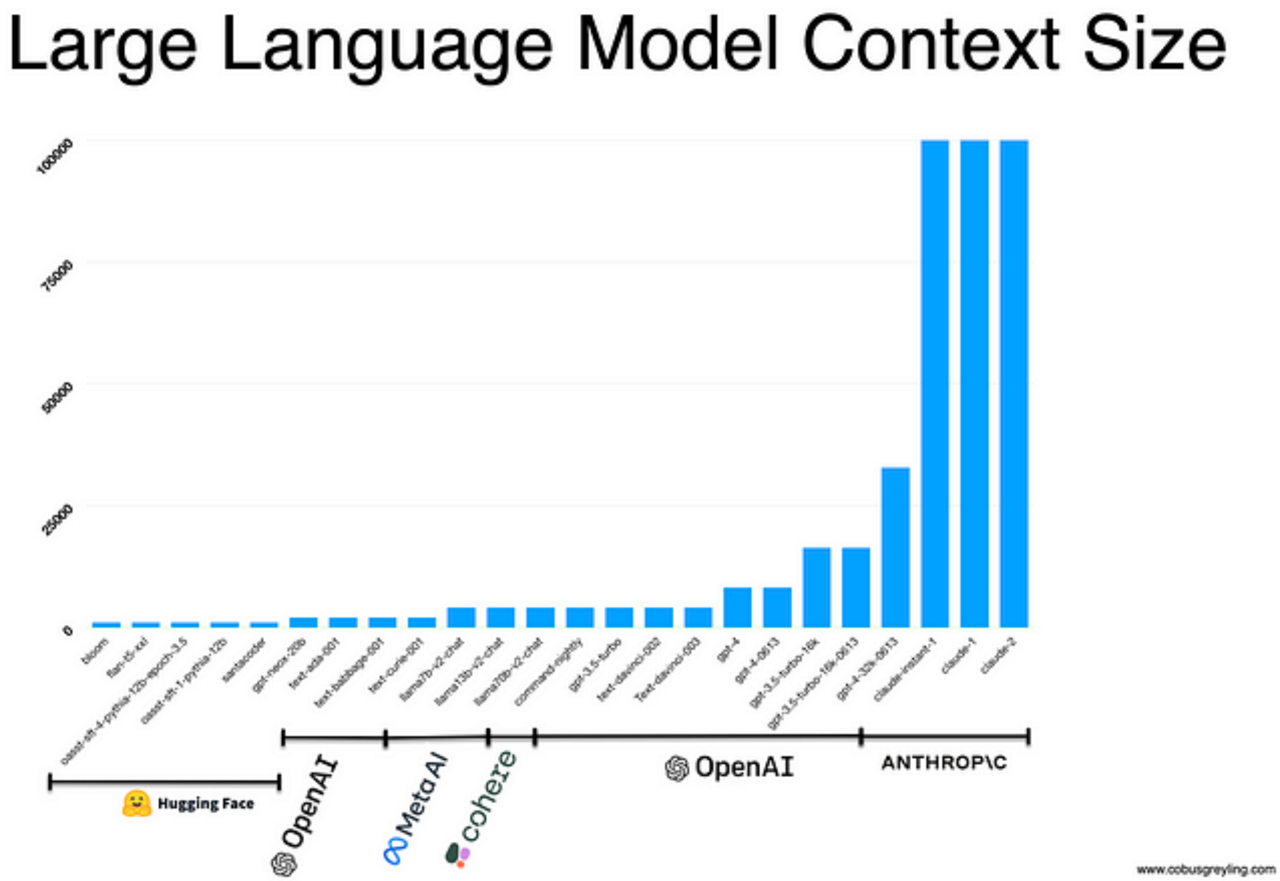
Next Token Selection

- Greedy selection
- Top-K
- Nucleus
- Beam search

Transformers for Long Sequences

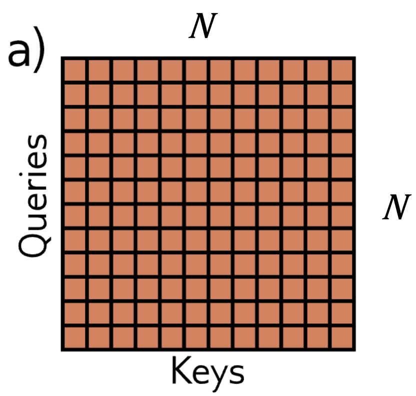
Context Length of LLMs

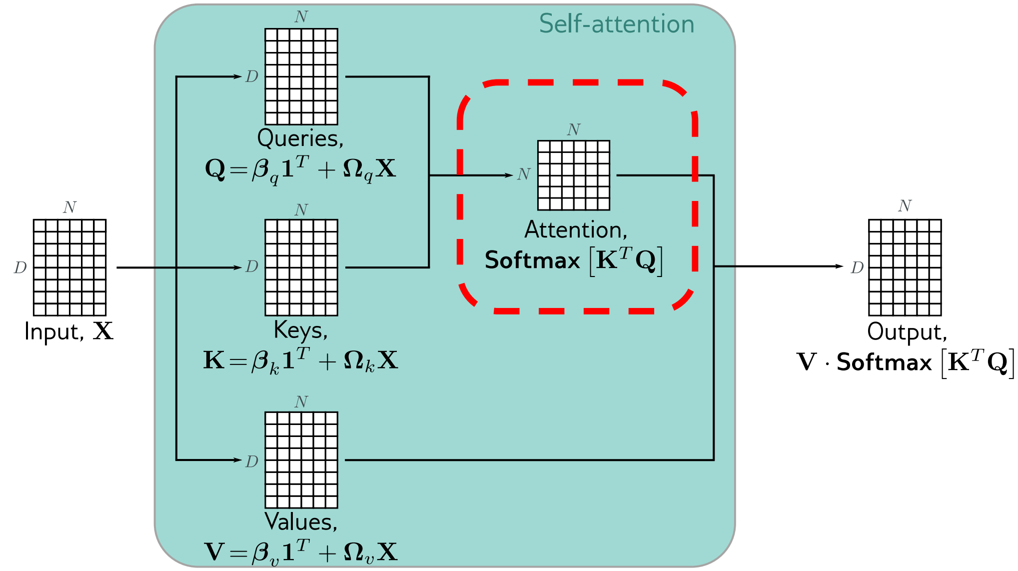
Model	Context Length
Llama 2	32K
GPT4	32K
GPT-4 Turbo, Llama 3.1	128K
Claude 3.5 Sonnet	200K
Google Gemini 1.5 Pro	Millions



https://cobusgreyling.medium.com/rag-llm-context-size-6728a2f44beb

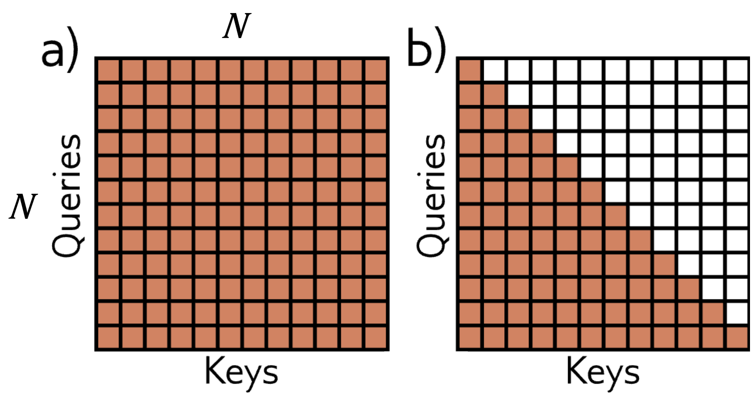
Attention Matrix

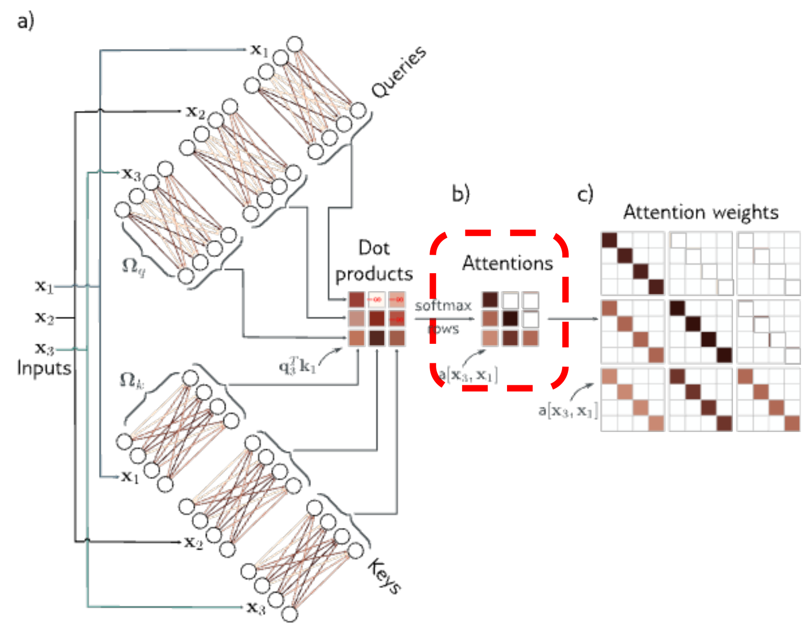




Scales quadratically with sequence length N, e.g. N^2.

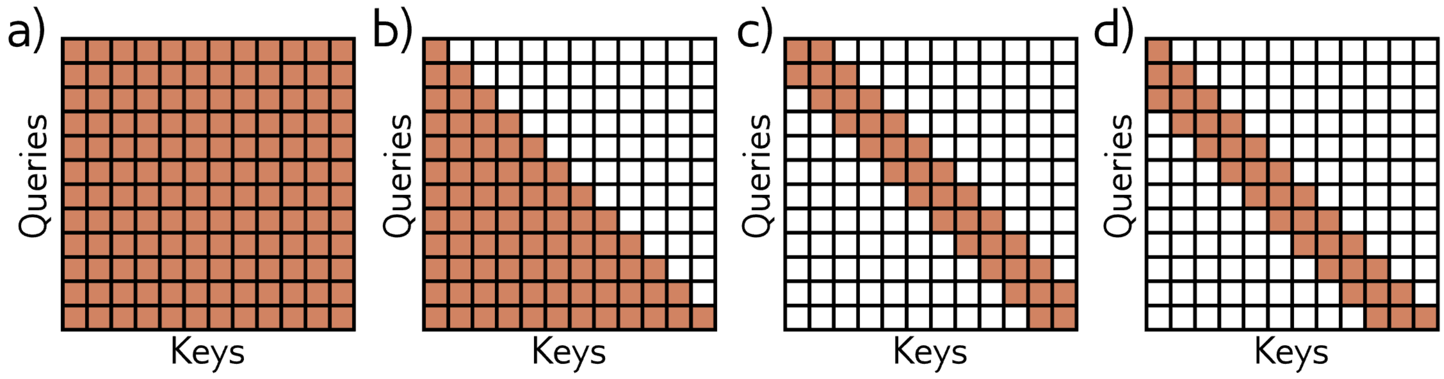
Masked Attention





~1/2 the interactions but
still scales quadratically

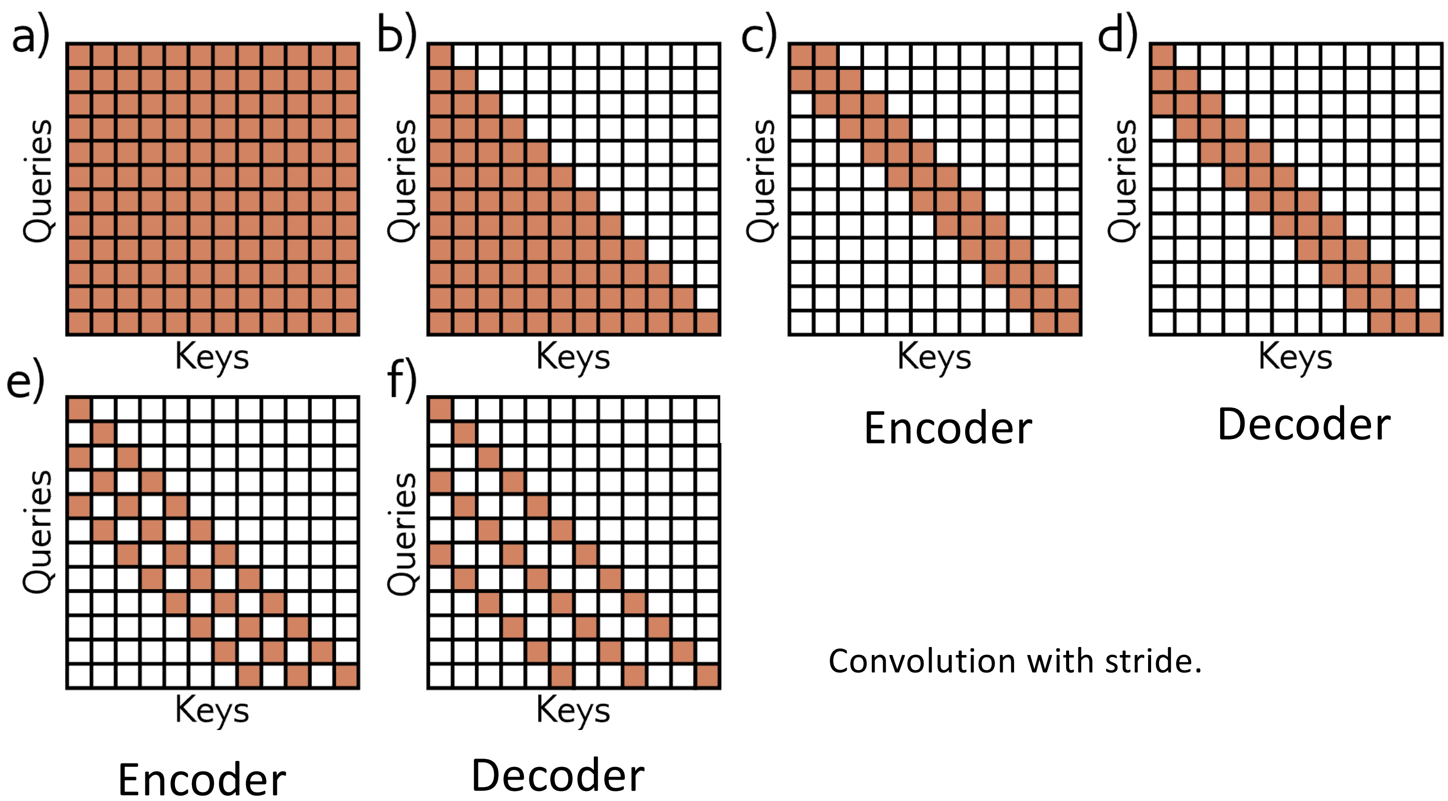
Use Convolutional Structure in Attention



Encoder
(non-causal)

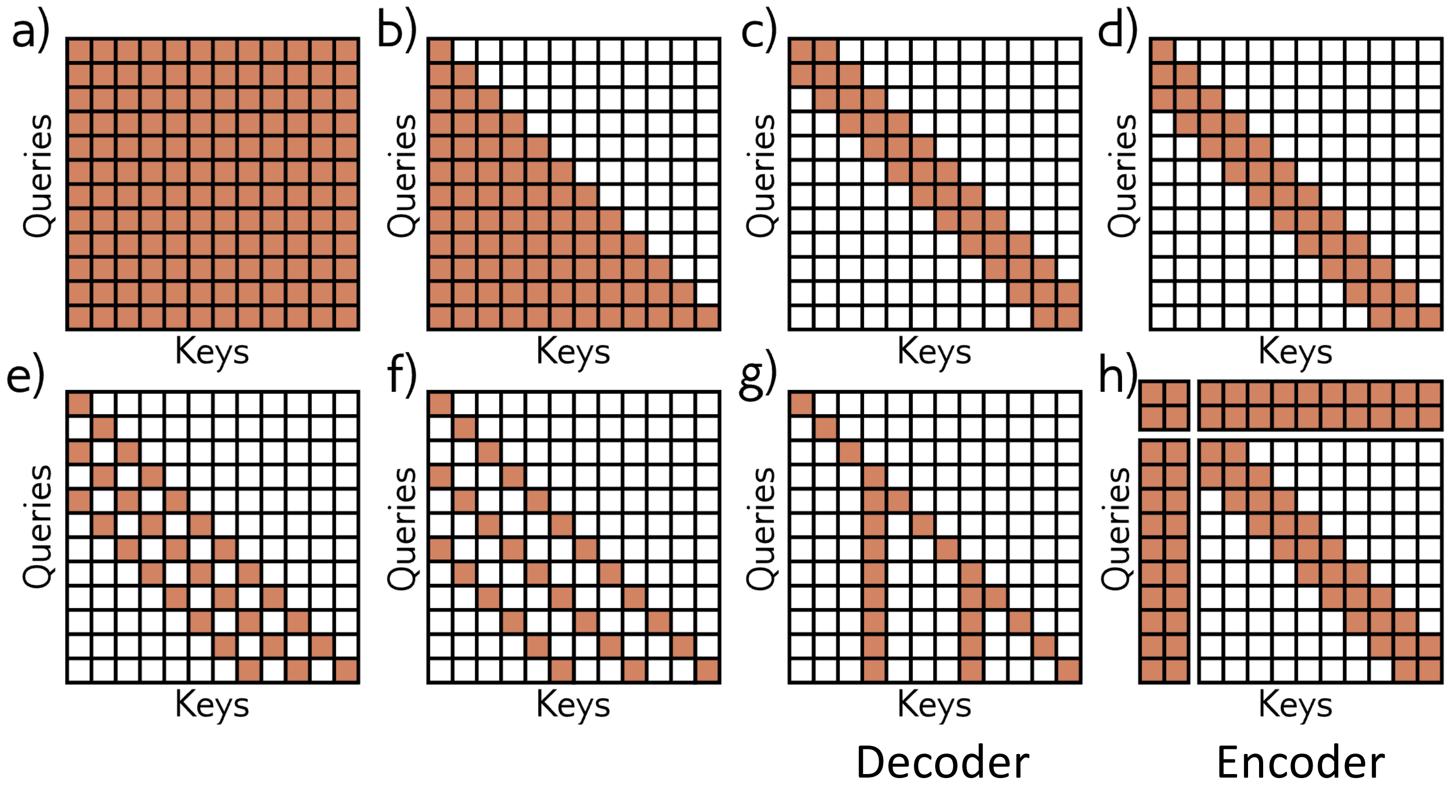
Decoder
(causal)

Dilated Convolutional Structures



Convolution with stride.

Have some tokens interact globally

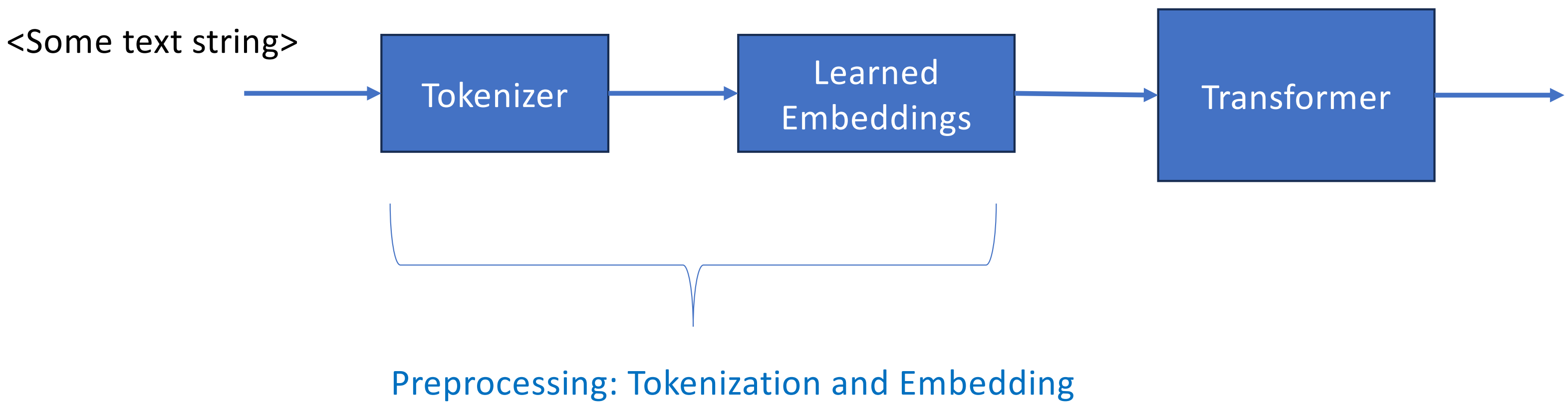


Tokenization and Word Embedding

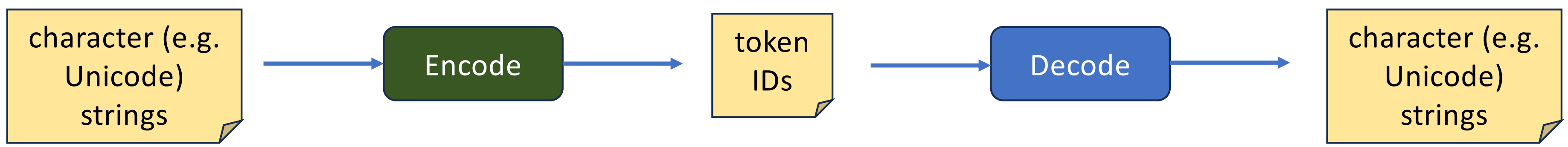
NLP Preprocessing Pipeline

Transformers don’t work on character string directly, but rather on vectors.

The character strings must be converted to vectors



Tokenizer



Tokenizer chooses input "units", e.g. words, sub-words, characters via *tokenizer training*

In tokenizer training, commonly occurring substrings are greedily merged based on their frequency, starting with character pairs

Tokenization Issues

“A lot of the issues that may look like issues with the neural network architecture actually trace back to tokenization. Here are just a few examples” – Andrej Karpathy

- Why can't LLM spell words? Tokenization.
- Why can't LLM do super simple string processing tasks like reversing a string? Tokenization.
- Why is LLM worse at non-English languages (e.g. Japanese)? Tokenization.
- Why is LLM bad at simple arithmetic? Tokenization.
- Why did GPT-2 have more than necessary trouble coding in Python? Tokenization.
- Why did my LLM abruptly halt when it sees the string "<|endoftext|>"? Tokenization.
- What is this weird warning I get about a "trailing whitespace"? Tokenization.
- Why did the LLM break if I ask it about "SolidGoldMagikarp"? Tokenization.
- Why should I prefer to use YAML over JSON with LLMs? Tokenization.
- Why is LLM not actually end-to-end language modeling? Tokenization.
- What is the real root of suffering? Tokenization.

https://github.com/karpathy/minbpe/blob/master/lecture.md

Unicode Standard and UTF-8

- Unicode – *variable length* character encoding standard. currently defines 149,813 characters and 161 scripts, including emoji, symbols, etc.
- Unicode Codepoint – can represent up to $17 \times 2^{16} = 1{,}114{,}112$ entries. e.g. U+0000 – U+10FFFF in hexadecimal
- Unicode Transformation Standard (e.g. UTF-8) – is a *variable length encoding* using one to four bytes
  - First 128 chars same as ASCII

Code point ↔ UTF-8 conversion

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4	
U+0000	U+007F	0xxxxxxx				Covers ASCII
U+0080	U+07FF	110xxxxx	10xxxxxx			Covers remainder of almost all Latin-script alphabets
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx		Basic Multilingual Plane including Chinese, Japanese and Korean characters
U+010000	[b]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx	Emoji, historic scripts, math symbols

https://en.wikipedia.org/wiki/Unicode
https://en.wikipedia.org/wiki/UTF-8

Tokenizer

Two common tokenizers:

- Byte Pair Encoding (BPE) – Used by OpenAI GPT2, GPT4, etc.
  - The BPE algorithm is "byte-level" because it runs on UTF-8 encoded strings.
  - This algorithm was popularized for LLMs by the GPT-2 paper and the associated GPT-2 code release from OpenAI. Sennrich et al. 2015 is cited as the original reference for the use of BPE in NLP applications. Today, all modern LLMs (e.g. GPT, Llama, Mistral) use this algorithm to train their tokenizers.*
- sentencepiece
  - (e.g. Llama, Mistral) use sentencepiece instead. Primary difference being that sentencepiece runs BPE directly on Unicode code points instead of on UTF-8 encoded bytes.

* https://github.com/karpathy/minbpe/tree/master

BPE Pseudocode

```
Initialize vocabulary with individual characters in
the text and their frequencies
While desired vocabulary size not reached:
    Identify the most frequent pair of adjacent
    tokens/characters in the vocabulary
    Merge this pair to form a new token
    Update the vocabulary with this new token
    Recalculate frequencies of all tokens including
    the new token
Return the final vocabulary
```

Enforce a Token Split Pattern

```
GPT2_SPLIT_PATTERN = r"""'(?:[sdmt]|ll|ve|re)| ?\p{L}+| ?\p{N}+|
?[^\s\p{L}\p{N}]+|\s+(?!\S)|\s+"""

GPT4_SPLIT_PATTERN = r"""'(?i:[sdmt]|ll|ve|re)|[^\r\n\p{L}\p{N}]?+\p{L}+|\p{N}{1,3}|
?[^\s\p{L}\p{N}]++[\r\n]*|\s*[\r\n]|\s+(?!\S)|\s+"""
```

- Do not allow tokens to merge across certain characters or patterns
- Common contraction endings: ‘ll, ‘ve, ‘re
- Match words with a leading space
- Match numeric sequences
- carriage returns, new lines

GPT4 Tokenizer

cl100k_base is the GPT4 tokenizer

Tiktokenizer

cl100k_base

```
a sailor went to sea sea sea
to see what he could see see see
but all that he could see see see
was the bottom of the deep blue sea sea sea
```

Token count

36

```
a·sailor·went·to·sea·sea·sea\n
to·see·what·he·could·see·see·see\n
but·all·that·he·could·see·see·see\n
was·the·bottom·of·the·deep·blue·sea·sea·sea
```

```
[64, 93637, 4024, 311, 9581, 9581, 9581, 198, 99
8, 1518, 1148, 568, 1436, 1518, 1518, 1518, 198,
8248, 682, 430, 568, 1436, 1518, 1518, 1518, 198,
16514, 279, 5740, 315, 279, 5655, 6437, 9581, 958
1, 9581]
```

☑ Show whitespace

https://tiktokenizer.vercel.app/

GPT2 Tokenizer

Tiktokenizer

gpt2

```
class Tokenizer:
    """Base class for Tokenizers"""

    def __init__(self):
        # default: vocab size of 256 (all bytes), no merges,
no patterns
        self.merges = {} # (int, int) -> int
        self.pattern = "" # str
        self.special_tokens = {} # str -> int, e.g.
{'<|endoftext|>': 100257}
        self.vocab = self._build_vocab() # int -> bytes
```

Token count
146

```
class·Tokenizer:\n
····"""Base·class·for·Tokenizers"""\n
\n
····def·__init__(self):\n
········#·default:·vocab·size·of·256·(all·bytes),·no·m
erges,·no·patterns\n
········self.merges·=·{}·#·(int,·int)·->·int\n
········self.pattern·=·""·#·str\n
········self.special_tokens·=·{}·#·str·->·int,·e.g.·
{'<|endoftext|>':·100257}\n
········self.vocab·=·self._build_vocab()·#·int·->·byte
s
```

```
[4871, 29130, 7509, 25, 198, 220, 220, 220, 37227, 148
81, 1398, 329, 29130, 11341, 37811, 628, 220, 220, 22
0, 825, 11593, 15003, 834, 7, 944, 2599, 198, 220, 22
0, 220, 220, 220, 220, 220, 1303, 4277, 25, 12776, 39
7, 2546, 286, 17759, 357, 439, 9881, 828, 645, 4017, 3
212, 11, 645, 7572, 198, 220, 220, 220, 220, 220, 220,
220, 2116, 13, 647, 3212, 796, 23884, 1303, 357, 600,
11, 493, 8, 4613, 493, 198, 220, 220, 220, 220, 220, 2
20, 220, 2116, 13, 33279, 796, 13538, 1303, 965, 198,
220, 220, 220, 220, 220, 220, 220, 2116, 13, 20887, 6
2, 83, 482, 641, 796, 23884, 1303, 965, 4613, 493, 11,
304, 13, 70, 13, 1391, 6, 50256, 10354, 1802, 28676, 9
2, 198, 220, 220, 220, 220, 220, 220, 220, 2116, 13, 1
8893, 397, 796, 2116, 13557, 11249, 62, 18893, 397, 34
19, 1303, 493, 4613, 9881]
```

Show whitespace

You can see some issues with the GPT2 tokenizer with respect to python code

https://tiktokenizer.vercel.app/

GPT4 Tokenizer

Tiktokenizer

cl100k_base

```
class Tokenizer:
    """Base class for Tokenizers"""

    def __init__(self):
        # default: vocab size of 256 (all bytes), no merges,
no patterns
        self.merges = {} # (int, int) -> int
        self.pattern = "" # str
        self.special_tokens = {} # str -> int, e.g.
{'<|endoftext|>': 100257}
        self.vocab = self._build_vocab() # int -> bytes
```

Token count
96

```
class·Tokenizer:\n
···"""Base·class·for·Tokenizers"""\n
\n
···def·__init__(self):\n
·······#·default:·vocab·size·of·256·(all·bytes),·no·m
erges,·no·patterns\n
·······self.merges·=·{}·#·(int,·int)·->·int\n
·······self.pattern·=·""·#·str\n
·······self.special_tokens·=·{}·#·str·->·int,·e.g.·
{'<|endoftext|>':·100257}\n
·······self.vocab·=·self._build_vocab()·#·int·->·byte
s
```

```
[1058, 9857, 3213, 512, 262, 4304, 4066, 538, 369, 985
7, 12509, 15425, 262, 711, 1328, 2381, 3889, 726, 997,
286, 674, 1670, 25, 24757, 1404, 315, 220, 4146, 320,
543, 5943, 705, 912, 82053, 11, 912, 12912, 198, 286,
659, 749, 2431, 288, 284, 4792, 674, 320, 396, 11, 52
8, 8, 1492, 528, 198, 286, 659, 40209, 284, 1621, 674,
610, 198, 286, 659, 64308, 29938, 284, 4792, 674, 610,
1492, 528, 11, 384, 1326, 13, 5473, 100257, 1232, 220,
1041, 15574, 534, 286, 659, 78557, 284, 659, 1462, 595
7, 53923, 368, 674, 528, 1492, 5943]
```

✓ Show whitespace

Issues are improved with GPT4 tokenizer

https://tiktokenizer.vercel.app/

Byte Pair Encoding (BPE) Example

Minimal starting vocabulary of subset of lower case latin alphabet and space `_`.

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

Byte Pair Encoding (BPE) Example

Find the most frequent pair of adjacent tokens, `se`, in this case and form new token.

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

Next most frequent pair of tokens is `e_`

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

Continue until you hit your vocabulary size limit.

d)

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

Byte Pair Encoding (BPE) Example

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

d)

see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

⋮ ⋮ ⋮

e)

see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

a)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	s	a	t	o	h	l	u	b	d	w	c	f	i	m	n	p	r
33	28	15	12	11	8	6	6	4	3	3	3	2	1	1	1	1	1	1

b)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	e	se	a	t	o	h	l	u	b	d	w	c	s	f	i	m	n	p	r
33	15	13	12	11	8	6	6	4	3	3	3	2	2	1	1	1	1	1	1

c)

a_sailor_went_to_sea_sea_sea_
to_see_what_he_could_see_see_see_
but_all_that_he_could_see_see_see_
was_the_bottom_of_the_deep_blue_sea_sea_sea_

_	se	a	e_	t	o	h	l	u	b	d	e	w	c	s	f	i	m	n	p	r
21	13	12	12	11	8	6	6	4	3	3	3	3	2	2	1	1	1	1	1	1

⋮ ⋮

d)

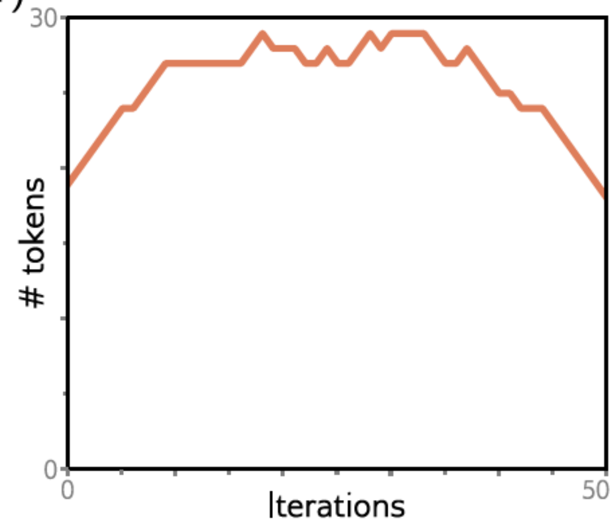
see_	sea_	e	b	l	w	a	could_	hat_	he_	o	t	t_	the_	to_	u	a_	d	f	m	n	p	s	sailor_	to
7	6	4	3	3	3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1

⋮ ⋮ ⋮

e)

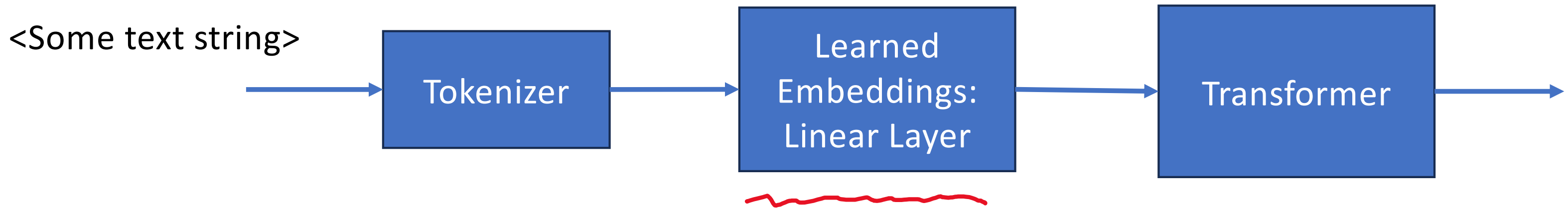
see_	sea_	could_	he_	the_	a_	all_	blue_	bottom_	but_	deep_	of_	sailor_	that_	to_	was_	went_	what_
7	6	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1

f)



Generally # of tokens increases and then starts decreasing after continuing to merge tokens

Learned Embeddings



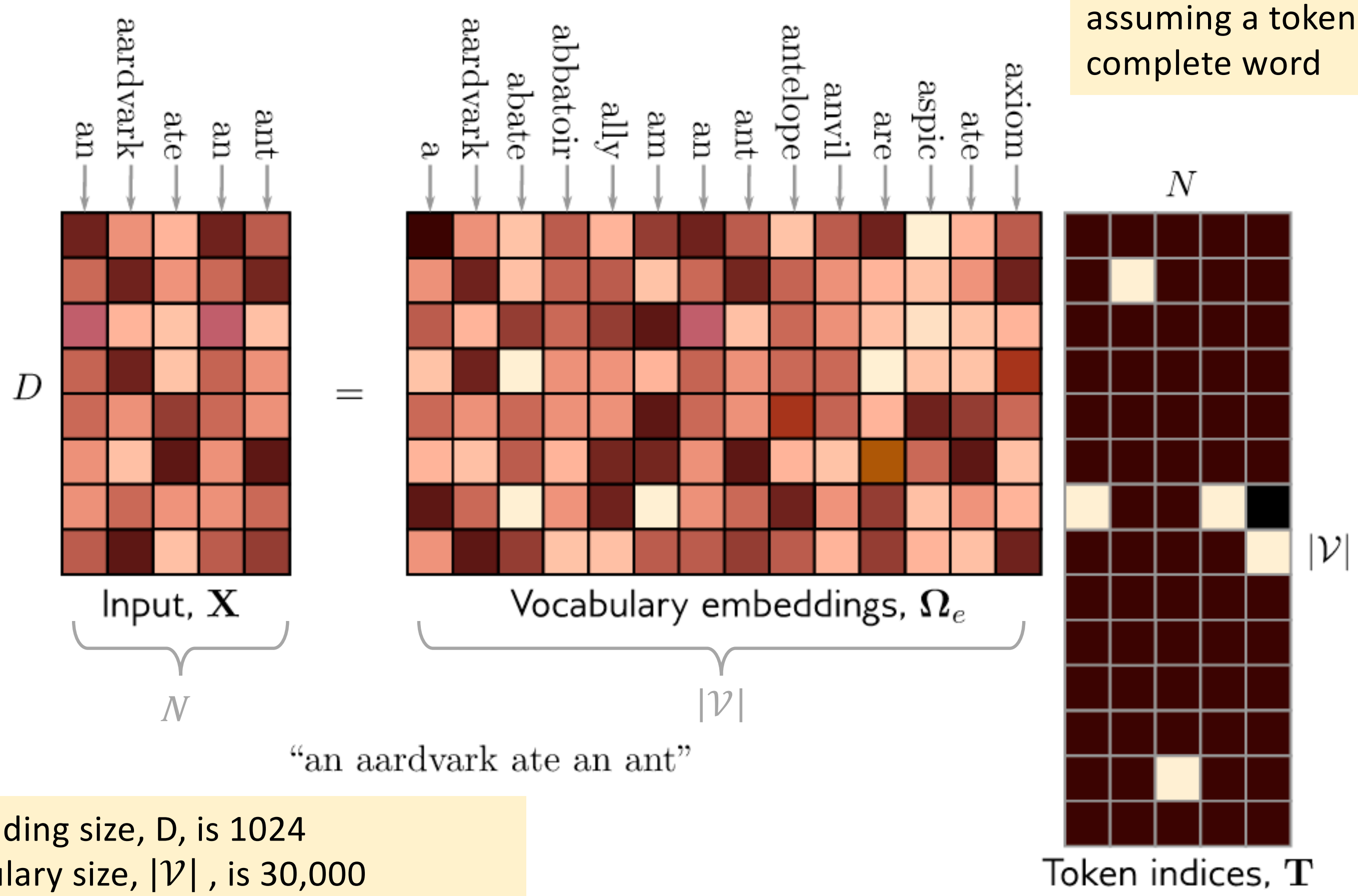
- After the tokenizer, you have an updated ”vocabulary” indexed by token ID
- Next step is to translate the token into an embedding vector
- Translation is done via a linear layer which is typically learned with the rest of the transformer model

```
self.embedding = nn.Embedding(vocab_size, embedding_dim)
```

- Special layer definition, likely to exploit sparsity of input

Embeddings Output

In this example, we are assuming a token is simply a complete word



"an aardvark ate an ant"

"One hot encoding"

- Typical embedding size, D, is 1024
- Typical vocabulary size, $|\mathcal{V}|$, is 30,000
- So 30M parameters just for this matrix!

Next Jupyter Notebook assignment

- will release shortly

- ➢ self-attention
- ➢ multi-head self-attention

Next

- Image Transformers
- Multimodal Transformers
- …

Feedback

Link