Language Modeling III, Tokenization

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-fall-2025

THE OHIO STATE UNIVERSITY

Logistics

- Hw2 due next week.

- Project proposal samples shared on teams – will keep updating with more.

Recap

- Recurrent Neural Networks
  - $h_t = f(h_{t-1}, x_t; \theta)$
  - Recurrent models are slow (cannot be parallelized with GPUs), h_t is expected to memorize everything that came before it – impractical

- RNNs with attention solve issue 2 to an extent but still slow to train.

- Transformers: attention is all you need (remove RNNs).

q: query (to match others)

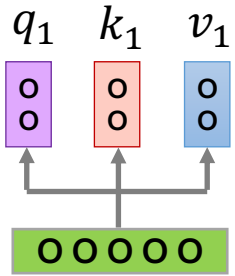
$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

$$v_t = W^v x_t$$



The

q: query (to match others)

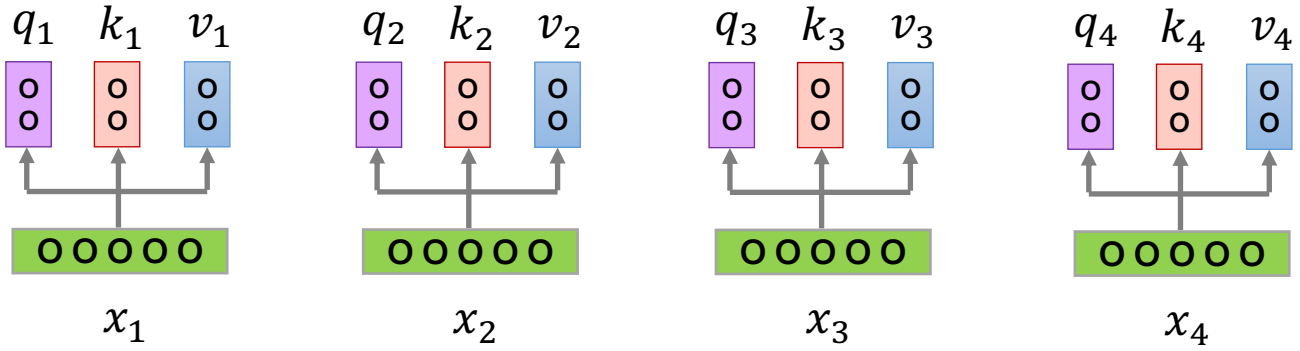
$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

$$v_t = W^v x_t$$



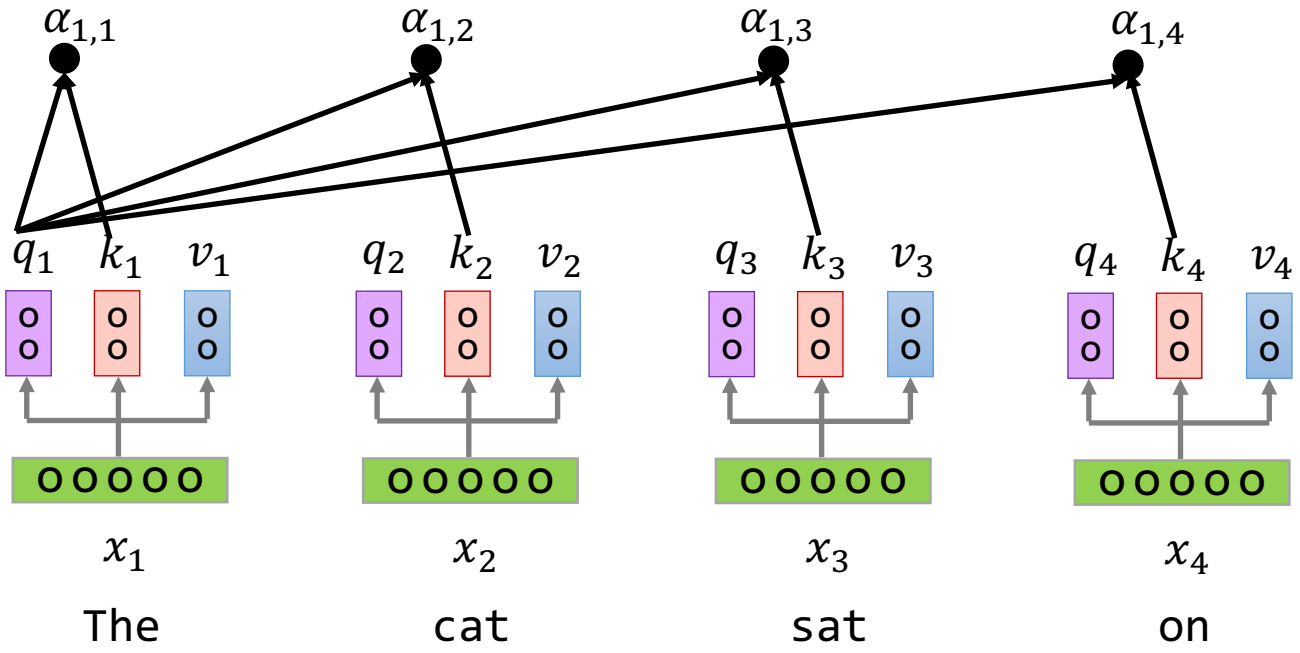
$$\alpha_{1,t} = \underbrace{q^1 \cdot k^t / \alpha}_{\text{Scaled dot product}}$$

q: query (to match others)

k: key (to be matched)

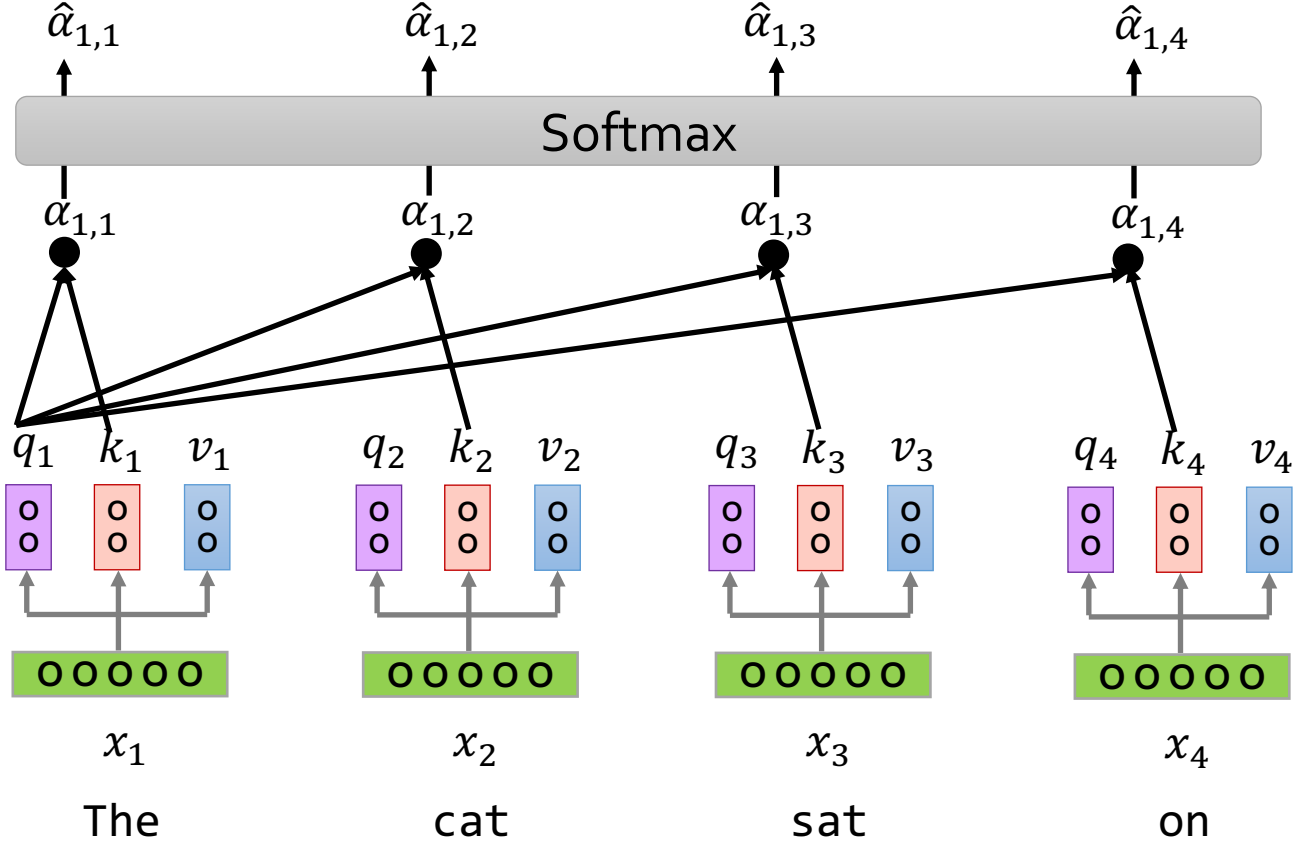
v: value (information to be extracted)

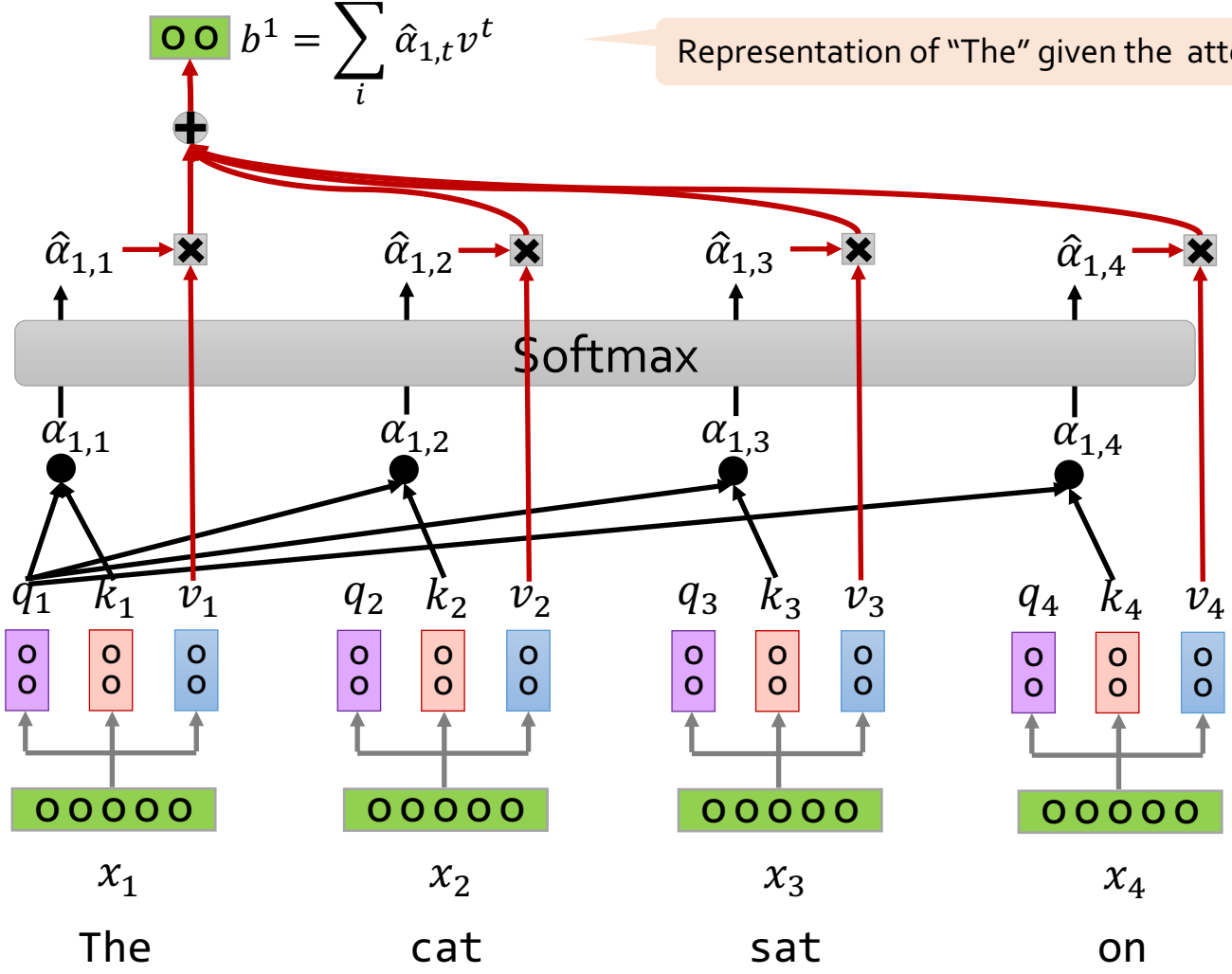
How much should "The" attend to other positions?



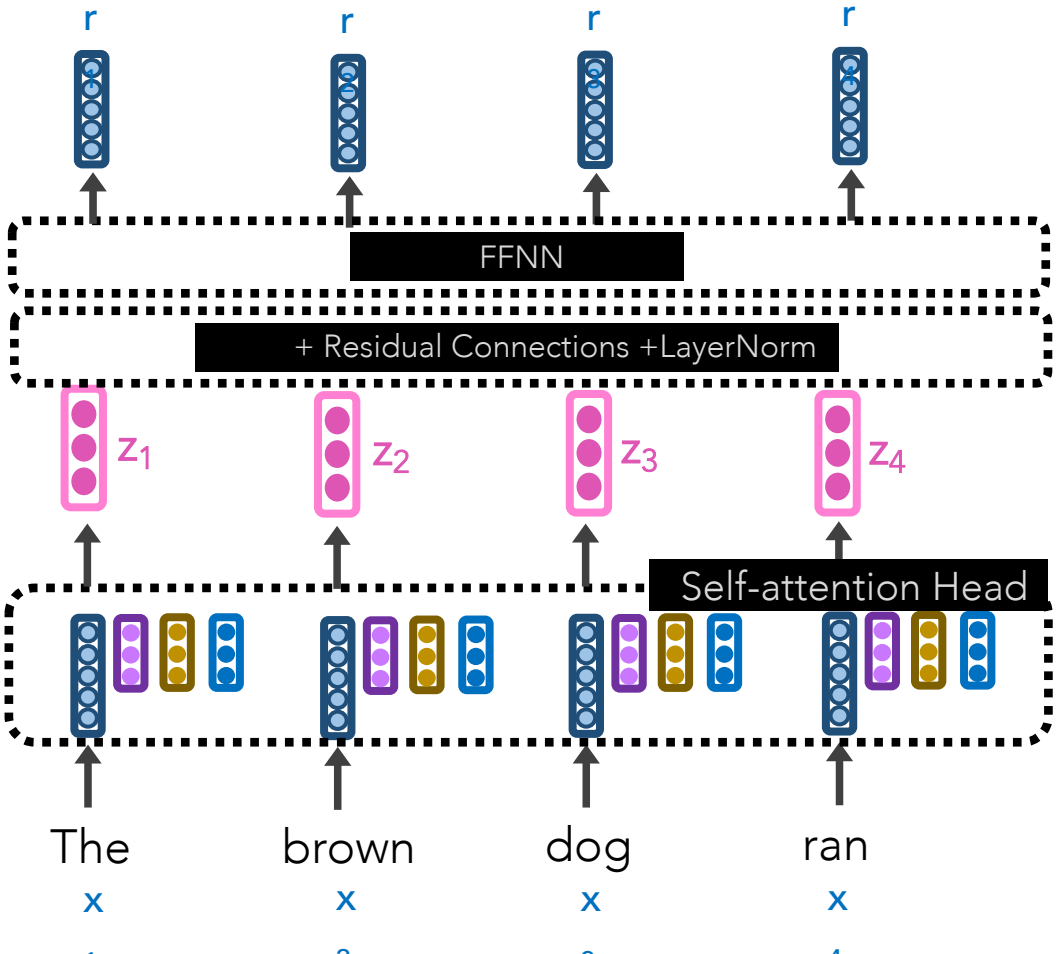
$$\sigma(z)_t = \frac{exp(z_t)}{\sum_j exp(z_j)}$$

How much should "The" attend to other positions?




$b^1 = \sum_i \hat{\alpha}_{1,t} v^t$
Representation of "The" given the attention weights
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
x_1
x_2
x_3
x_4
The
cat
sat
on

Self-Attention + FFNN + Residual Connections



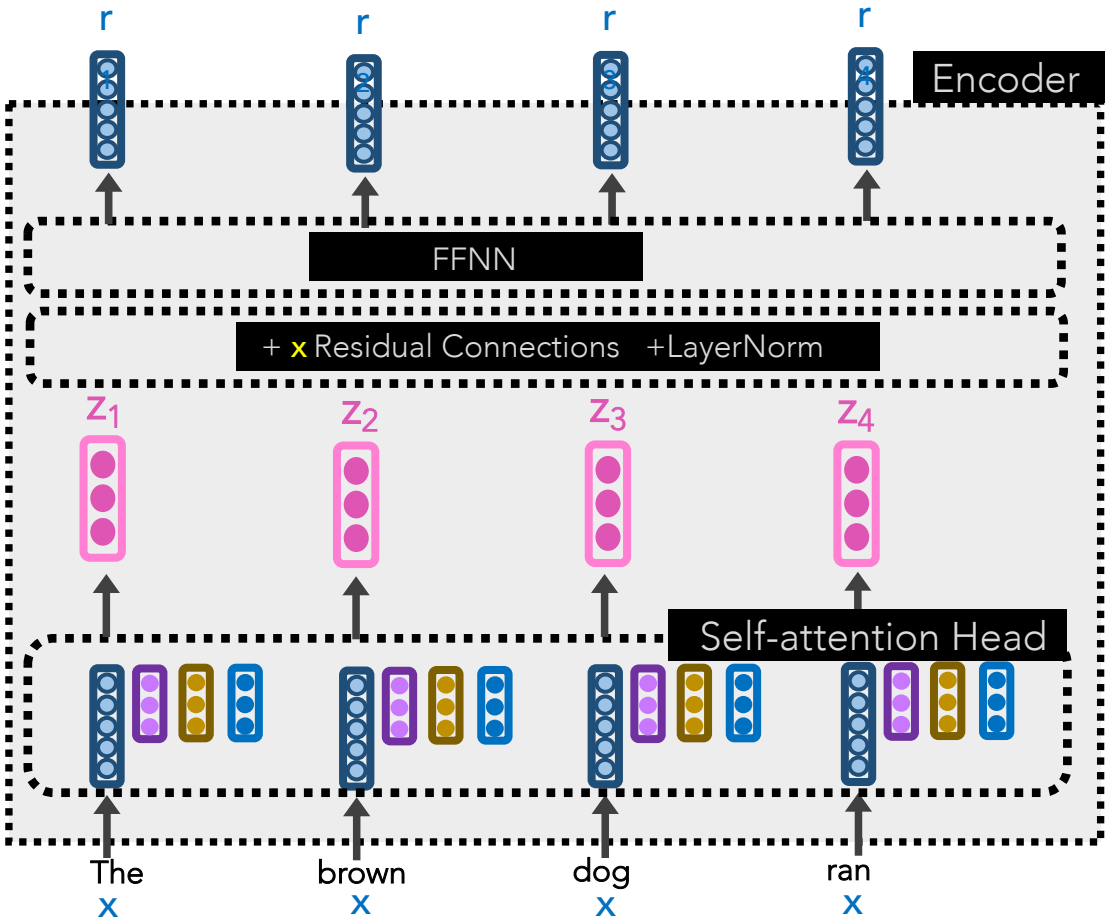
Let's further pass each z_i through a FFNN

We add residual connection to help ensure relevant info is getting forward passed.

$$v = z + x$$

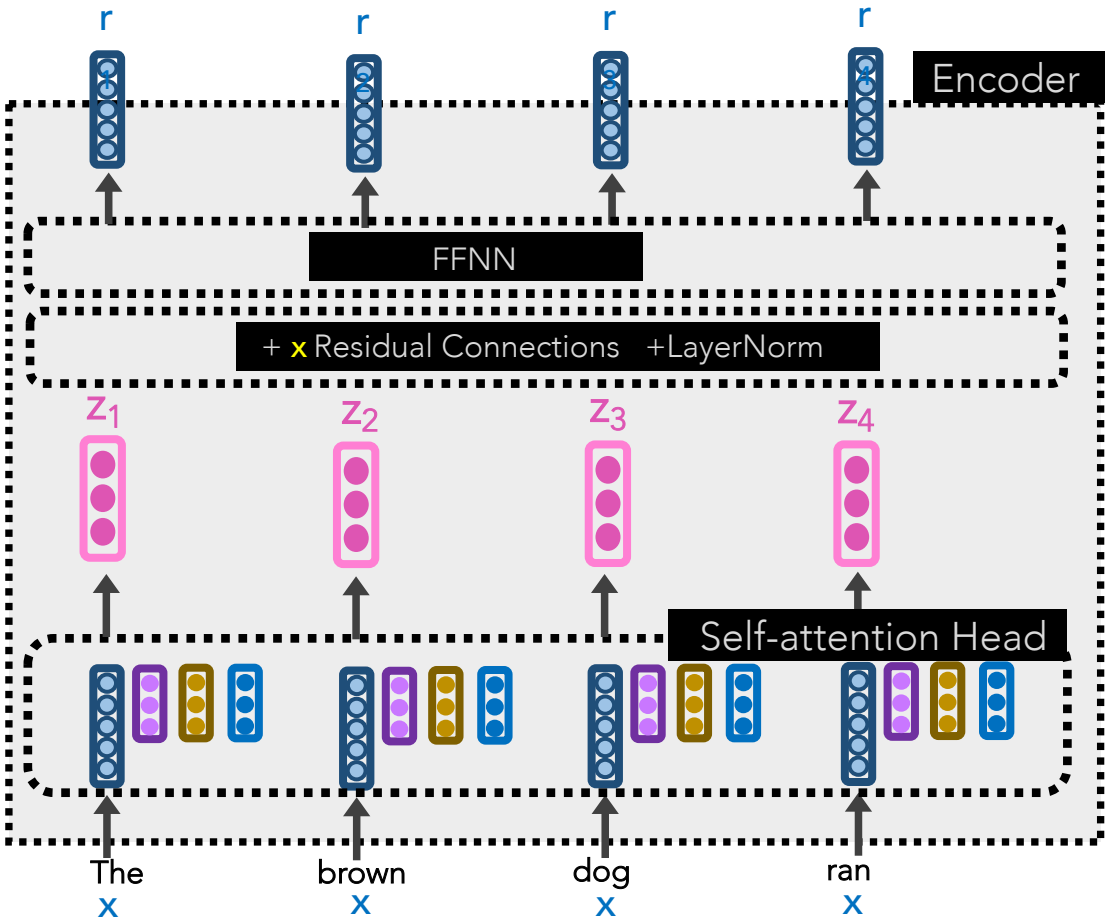
We perform LayerNorm to stabilize the network and allow for proper gradient flow. You should do this after the FFNN, too.

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

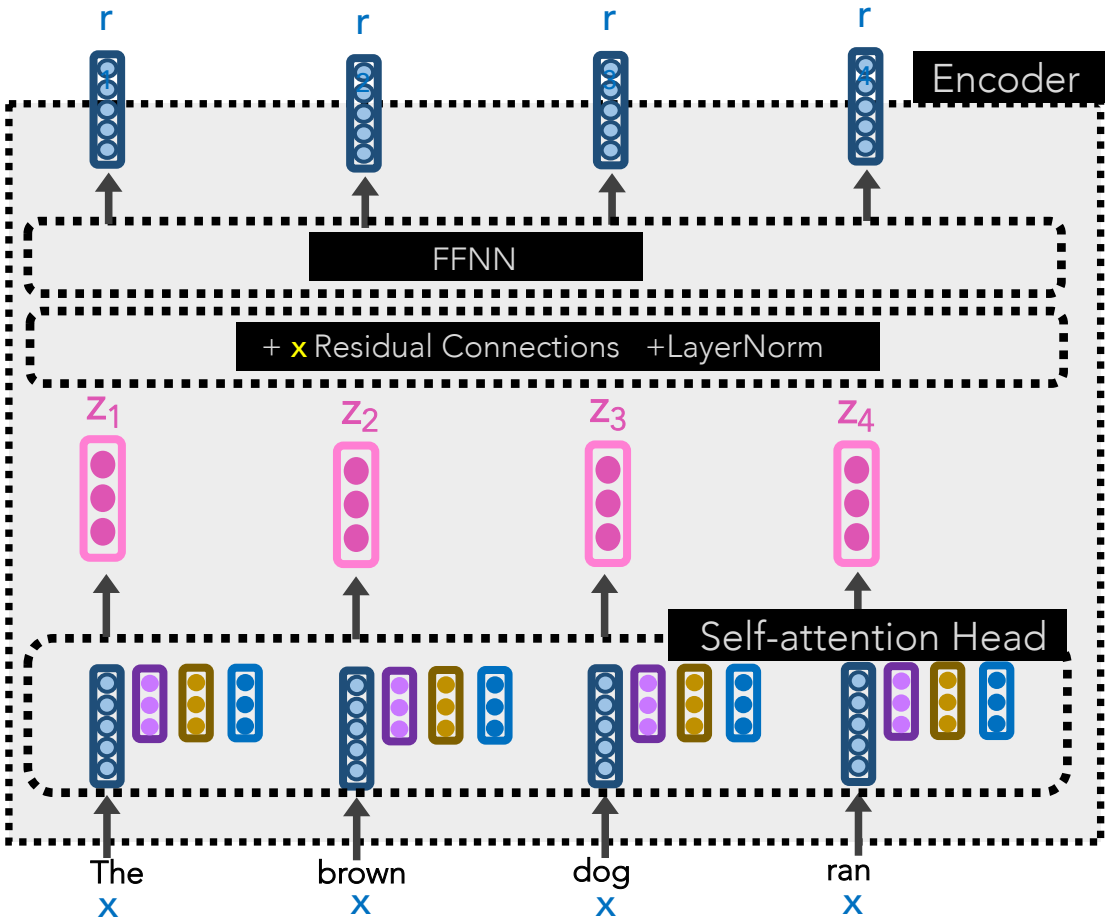
Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Transformer Encoder



Yay! Our r_i vectors are our new representations, and this entire process is called a **Transformer Encoder**

Problem: there is no concept of positionality. Words are weighted as if a "bag of words"

Solution: add to each input word x_i a positional encoding

Input to the model is now $x_i + pos_i$

Properties of a good positional embedding

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
  - The cat sat on the mat
  - The happy cat sat on the mat
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.

Absolute position embeddings

- Define a maximum context length you model can encode: say 1000 tokens.
  - Create a separate embedding table for each position.
  - Each index 1, 2, 3, … gets an embedding.
  - Learn the embeddings with the model.

- Issues with Learned positions embeddings:
  - Maximum length that can be presented is limited (what if I get a 2000 token input)
  - Difficult to encode relative positions
    - The cat sat on the mat
    - The happy cat sat on the mat

Functional (and fixed) position embeddings

Sinusoidal embeddings

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}$$

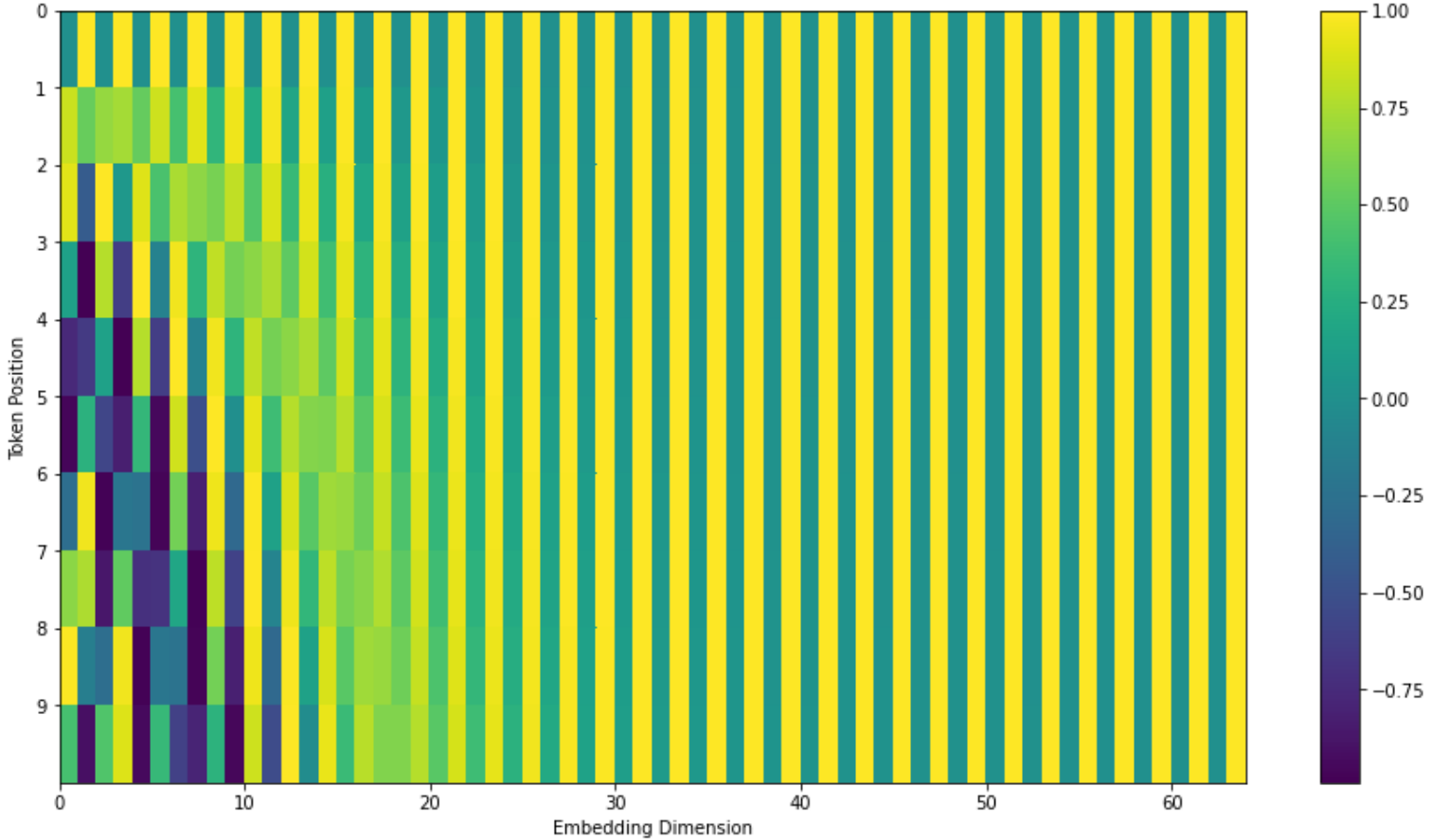
The frequencies are decreasing along the vector dimension. It forms a geometric progression on the wavelengths.

Sinusoidal Embeddings: Intuition

0 :	0	0	0	0	
1 :	0	0	0	1	
2 :	0	0	1	0	
3 :	0	0	1	1	
4 :	0	1	0	0	
5 :	0	1	0	1	
6 :	0	1	1	0	
7 :	0	1	1	1	
8 :	1	0	0	0	
9 :	1	0	0	1	
10 :	1	0	1	0	
11 :	1	0	1	1	
12 :	1	1	0	0	
13 :	1	1	0	1	
14 :	1	1	1	0	
15 :	1	1	1	1	

Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog

Position Encodings



https://jalammar.github.io/illustrated-transformer/

Variants of Positional Embeddings

- Rotary Positional Embeddings (RoPE): [2104.09864] RoFormer: Enhanced Transformer with Rotary Position Embedding (arxiv.org)

- AliBi: [2108.12409] Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (arxiv.org)

- No embeddings(!?): [2203.16634] Transformer Language Models without Positional Encodings Still Learn Positional Information (arxiv.org)

Summary So Far

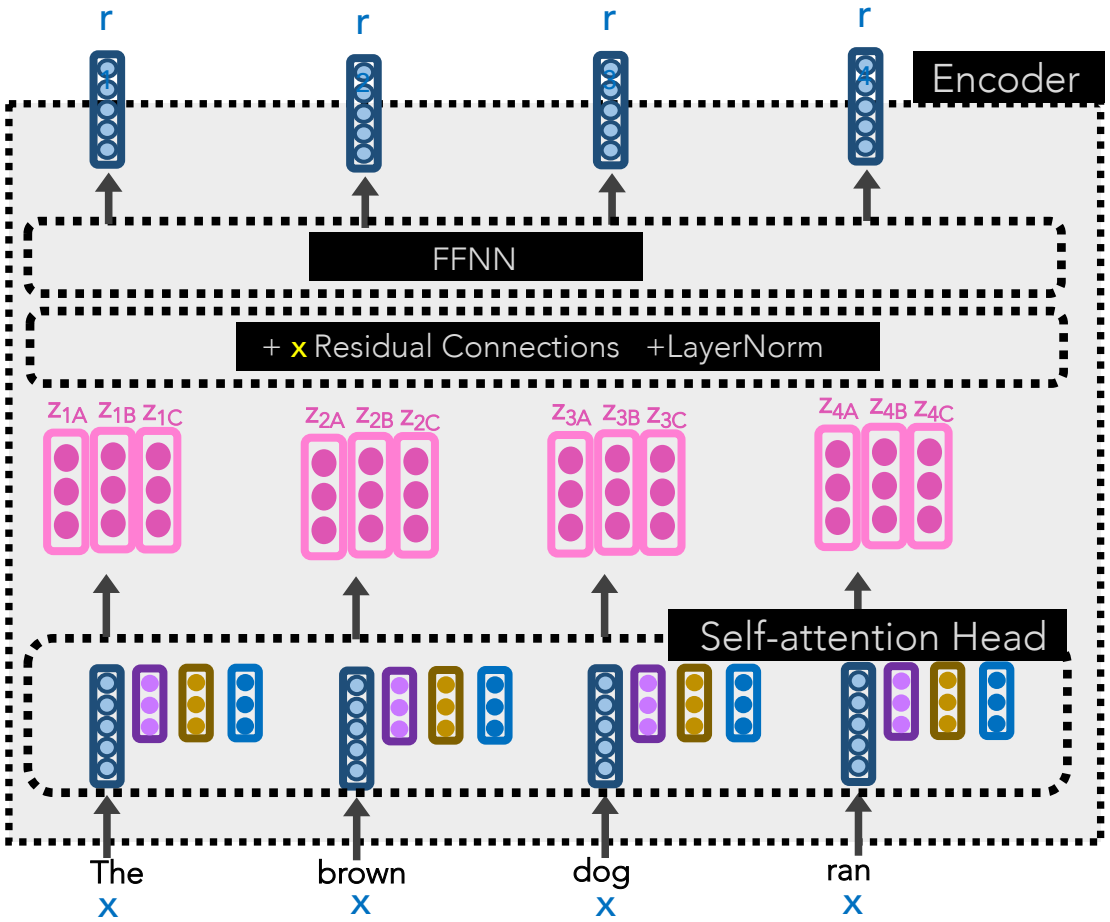
- Self Attention (Query, Key, Value)
- Residual Connections, Layernorm, FFN layers in between
- Positional Embeddings
- Combine all that and we get a transformer encoder.

A Self-Attention Head has just one set of query/key/value weight matrices w_q, w_k, w_v

Words can relate in many ways, so it's restrictive to rely on just one Self-Attention Head in the system.

Let's create Multi-headed Self-Attention

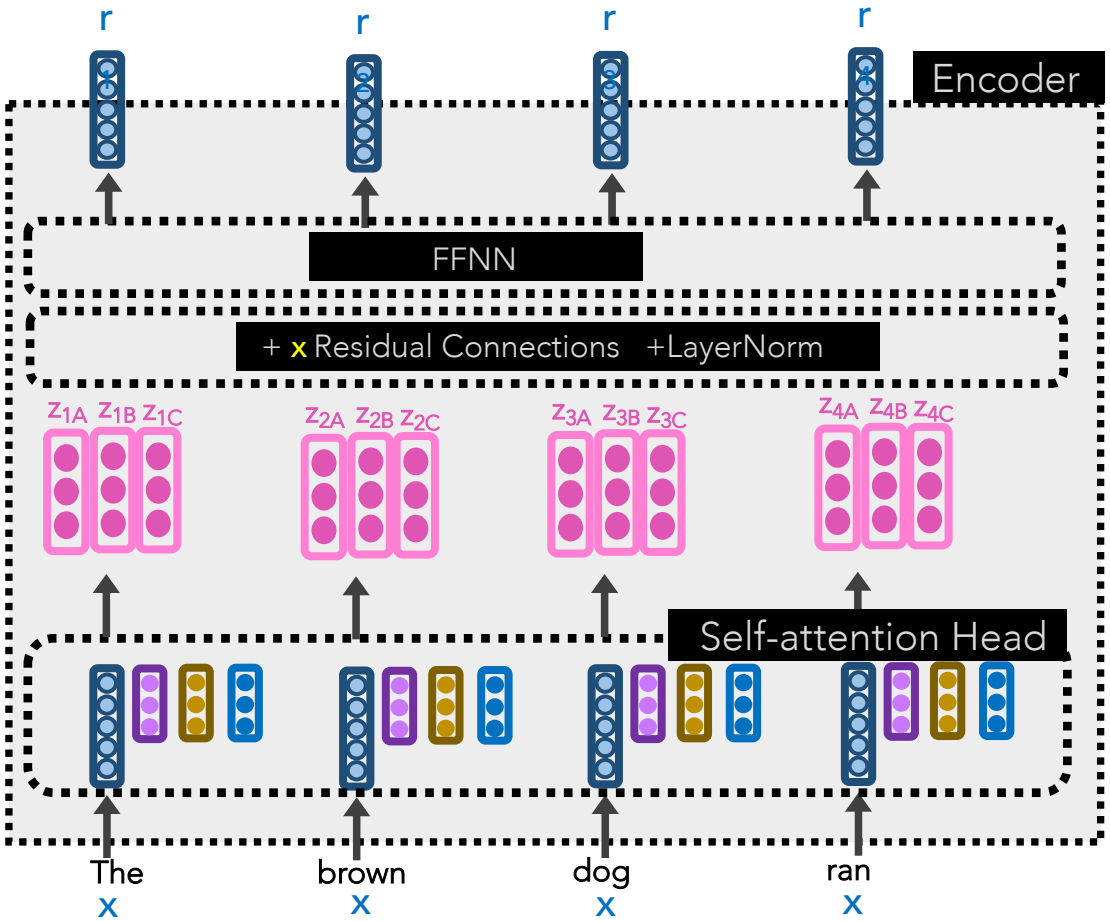
Multi-head Attention



Each Self-Attention Head produces a z_i vector using query, key, and value vectors

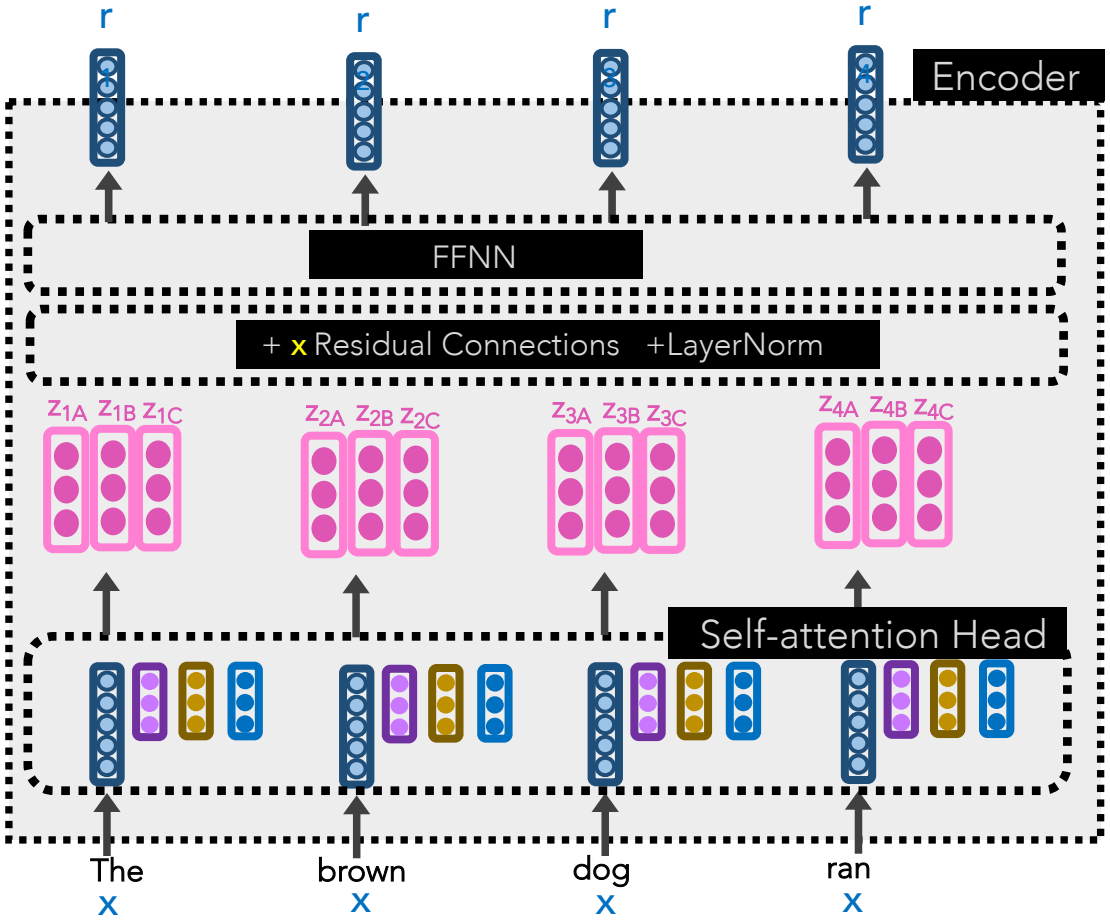
We can, in parallel, use multiple heads and concat the z_i's. For each input create multiple query, key, and value vectors

Transformer Encoder

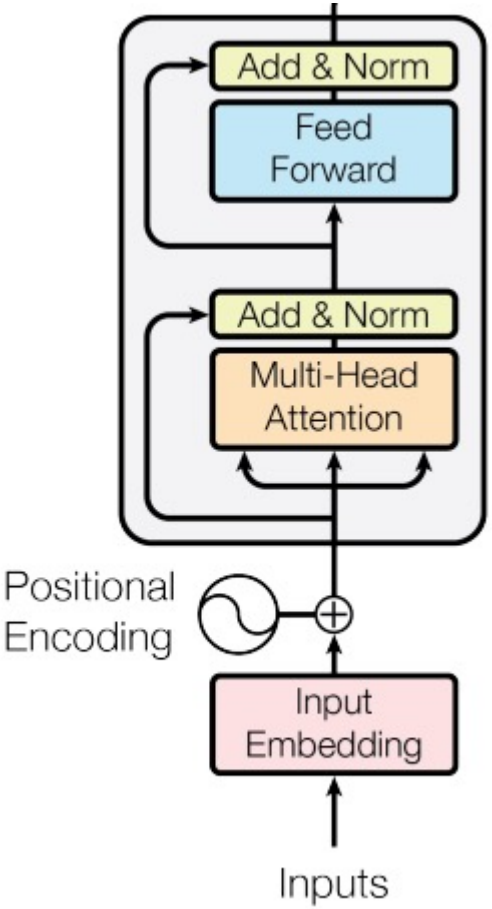


To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

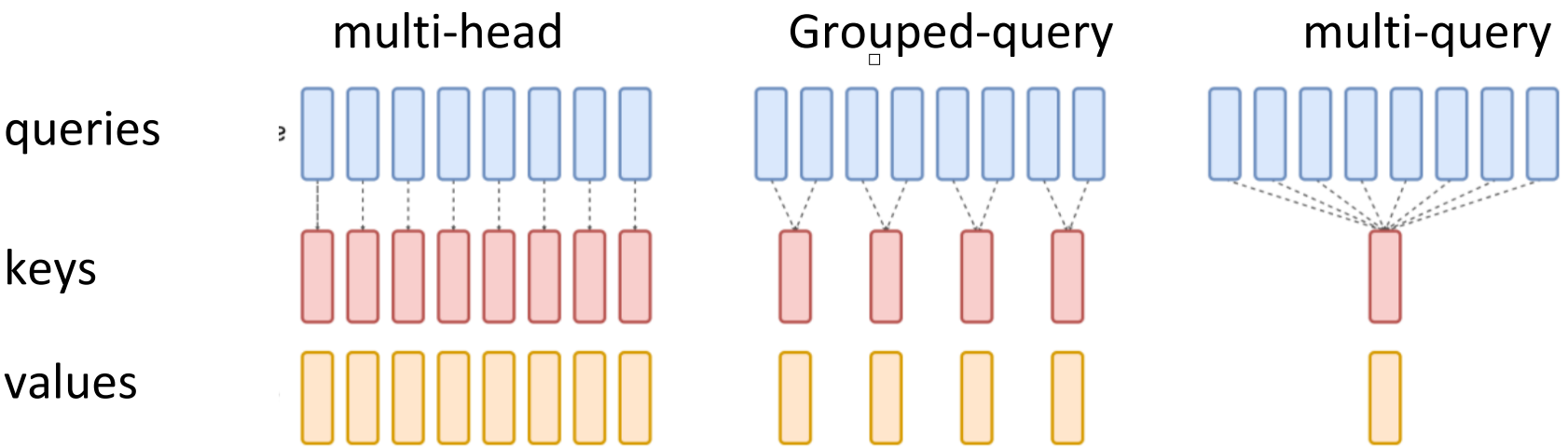
Transformer Encoder



=

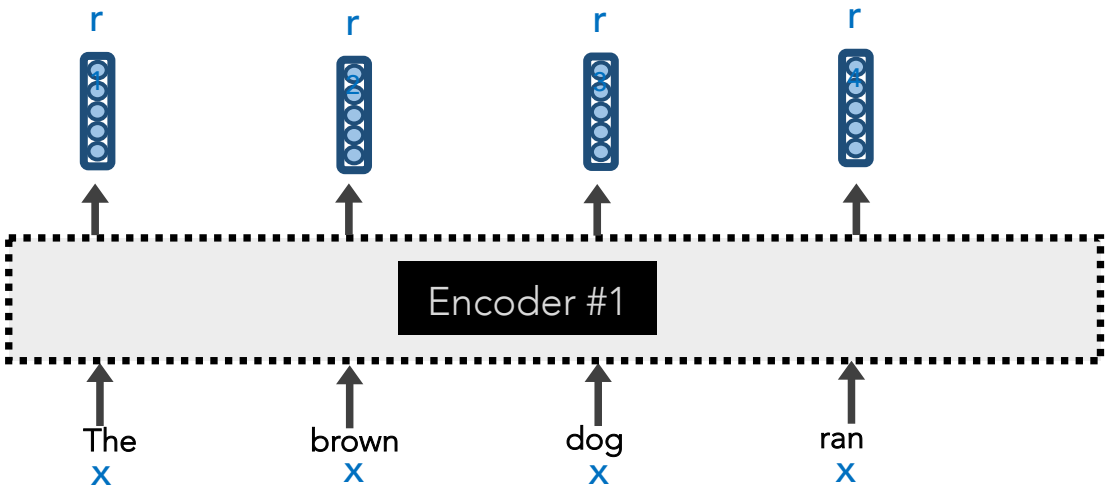


Variants of multi-head attention attention



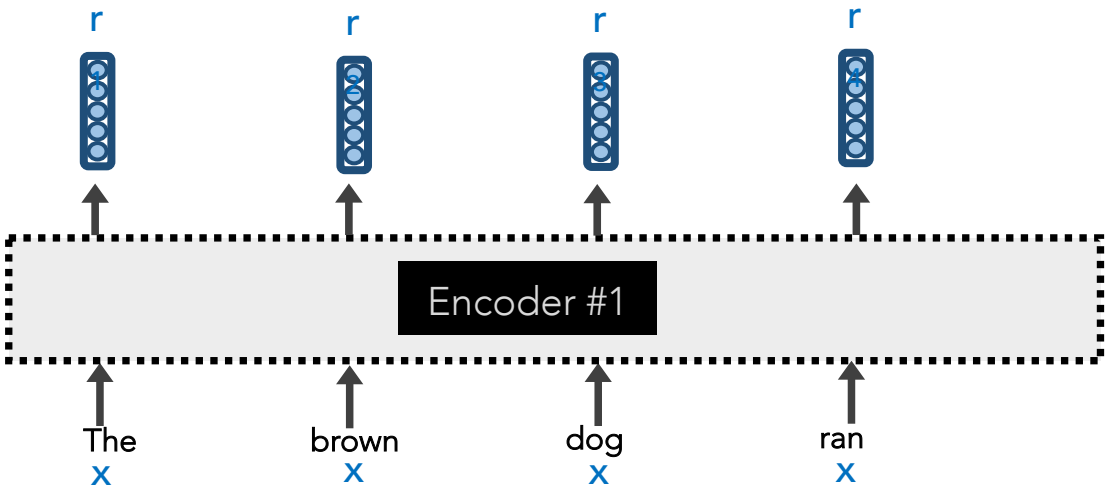
GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints (Ainslie et al., 2023)

Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

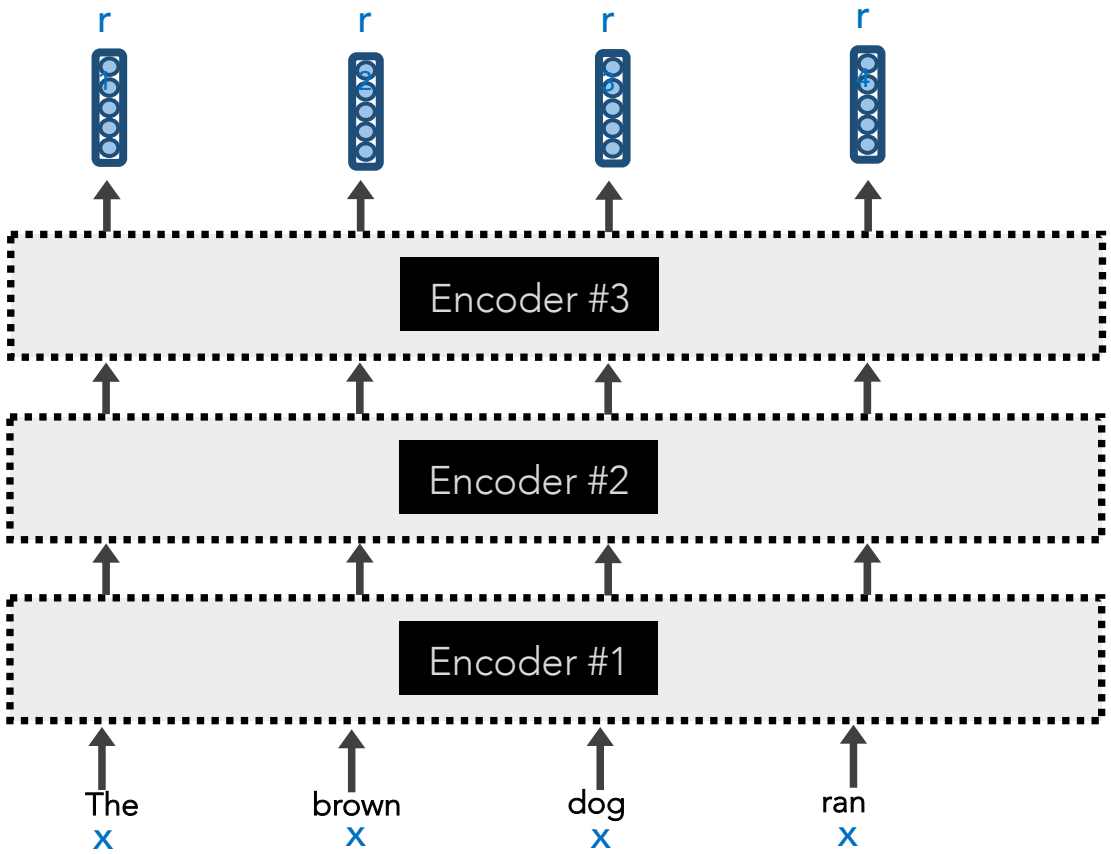
Transformer Encoder



To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

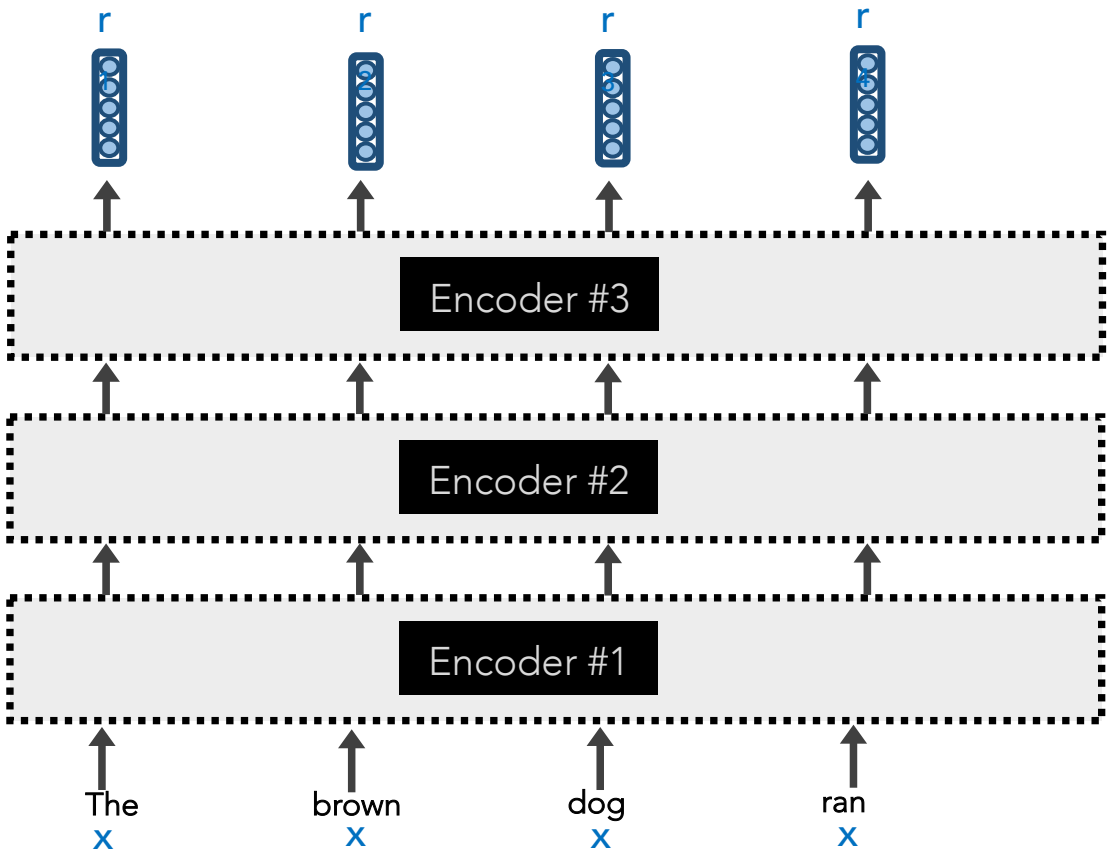
Transformer Encoder



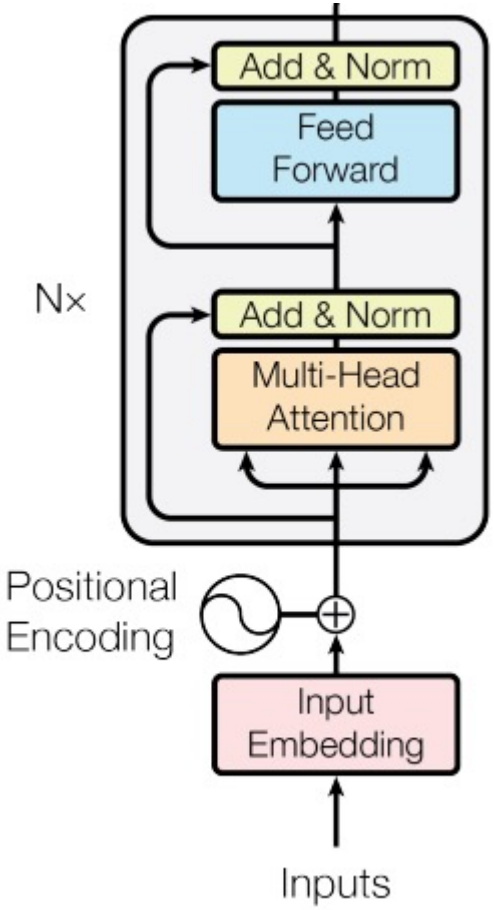
To recap: all of this looks fancy, but ultimately it's just producing a very good **contextualized embedding** r_i of each word x_i

Why stop with just 1 Transformer Encoder? We could stack several!

Transformer Encoder



=



The original Transformer model was intended for Machine Translation, so it had Decoders, too

Outline

Self-Attention

Transformer Encoder

Transformer Decoder

Language Modeling With Transformers

Outline

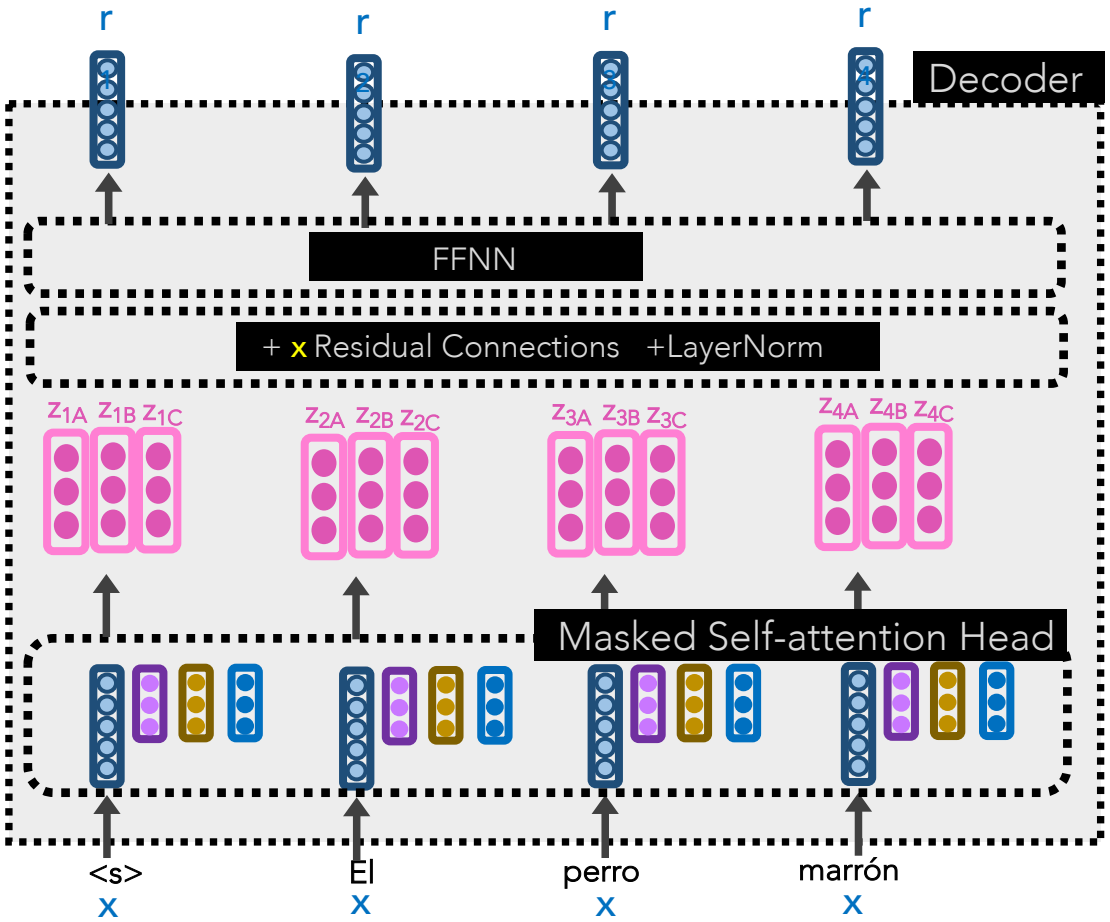
Self-Attention

Transformer Encoder

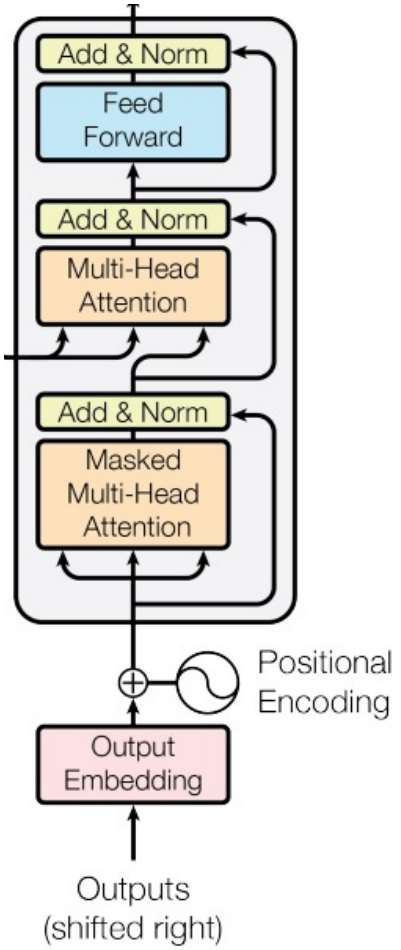
Transformer Decoder

Language Modeling With Transformers

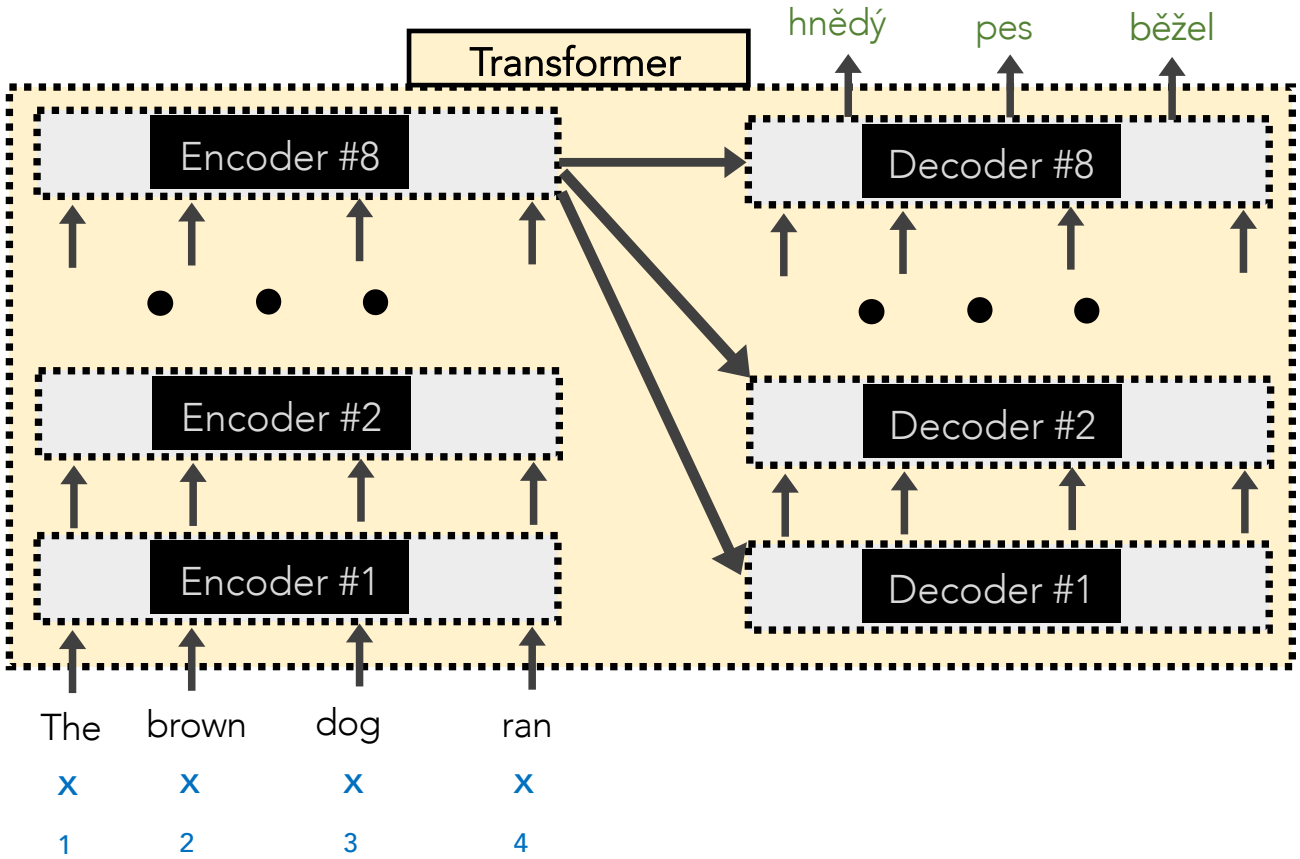
Transformer Decoder



=



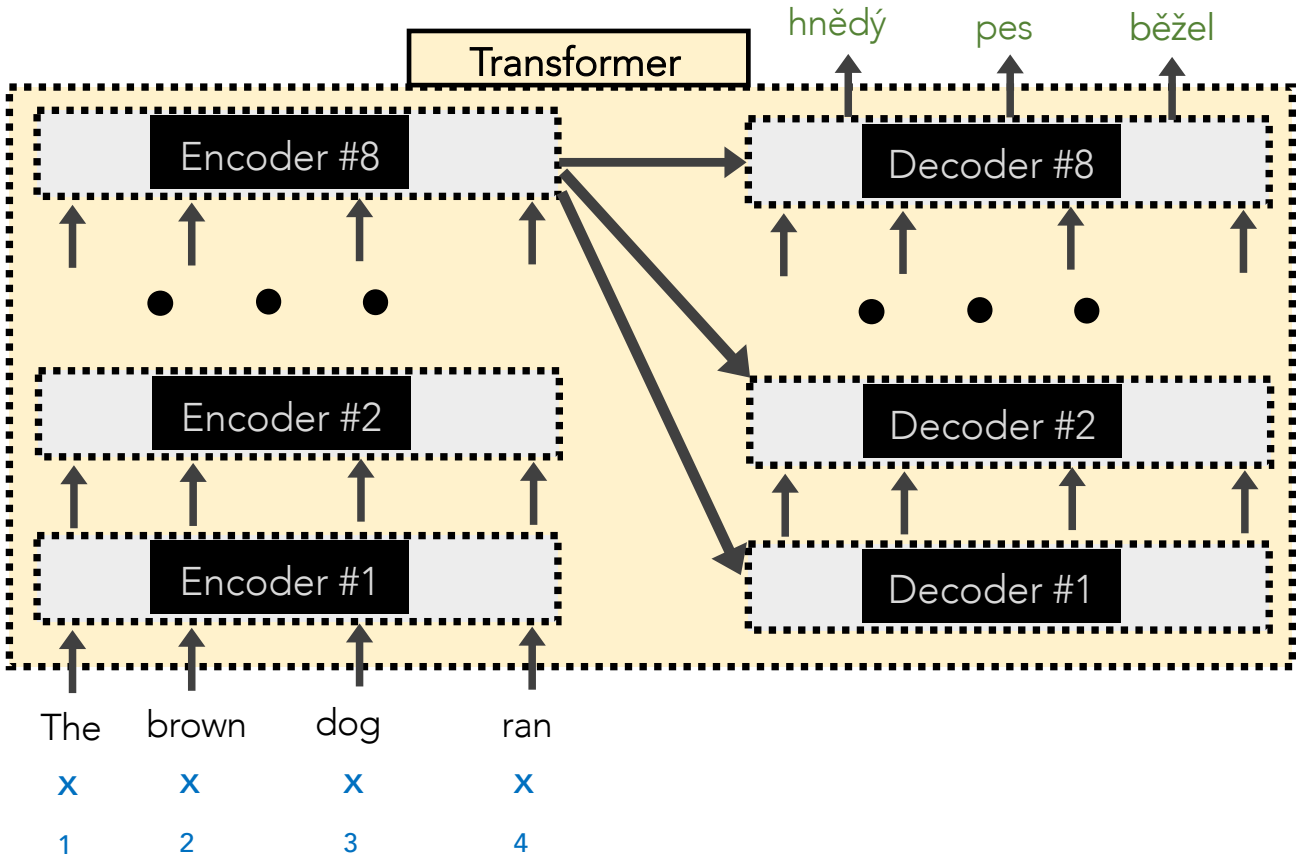
Transformer Encoders and Decoders



Transformer Encoders produce **contextualized embeddings** of each word

Transformer Decoders generate new sequences of text

Transformer Encoders and Decoders

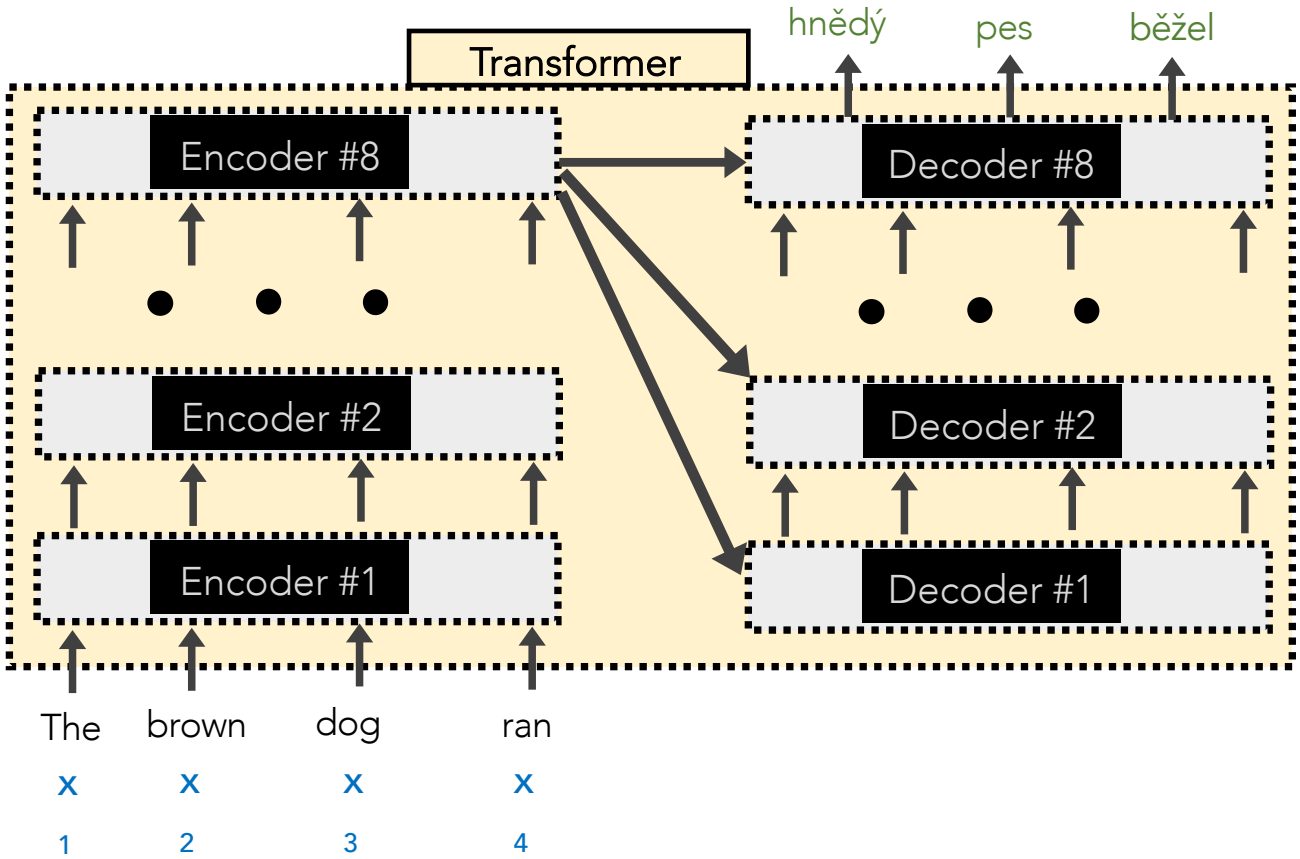


NOTE

Transformer Decoders are identical to the Encoders, except they have an additional Attention Head in between the Self-Attention and FFNN layers.

This additional Attention Head **focuses on parts of the encoder's representations.**

Transformer Encoders and Decoders

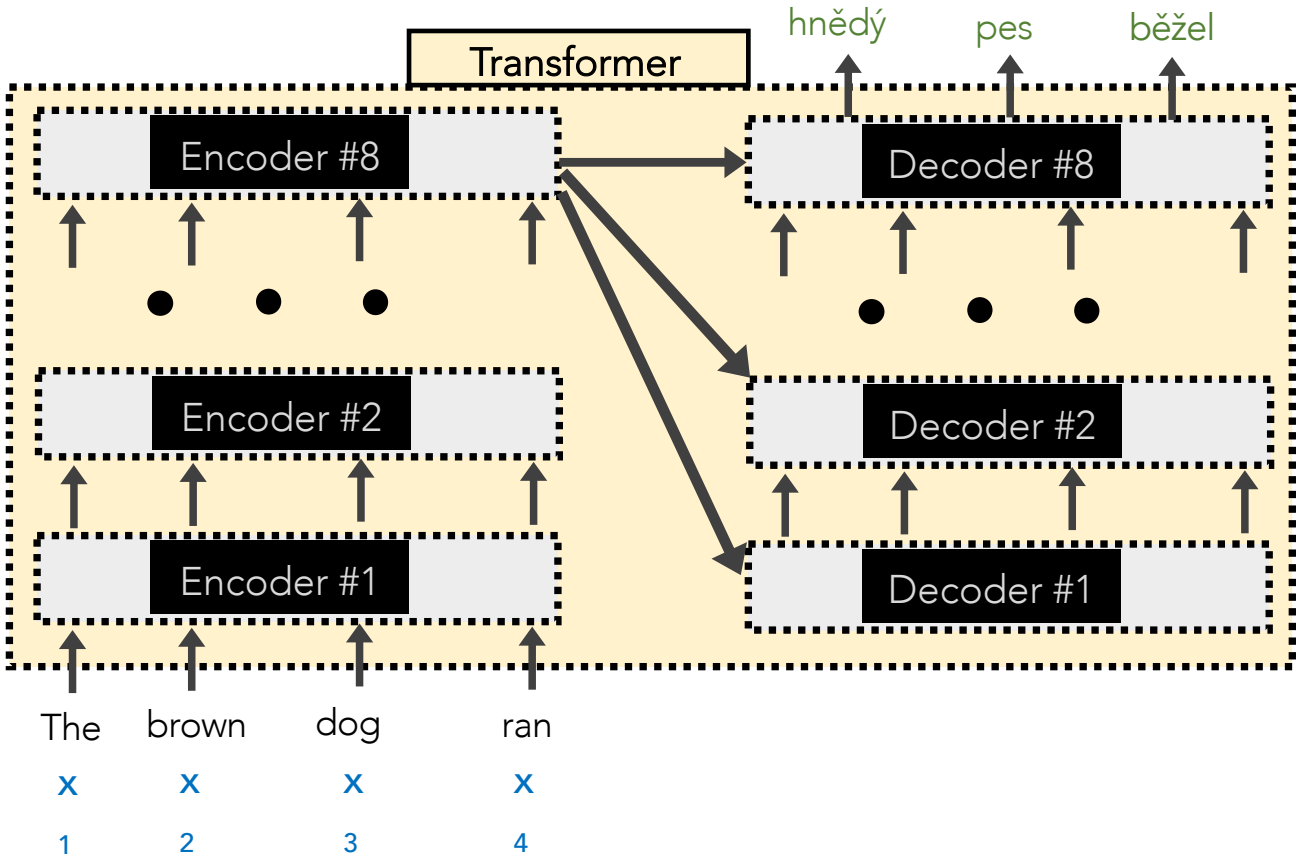


NOTE

The query vector for a Transformer Decoder's Attention Head (not Self-Attention Head) is from the output of the previous decoder layer.

However, the key and value vectors are from the Transformer Encoders' outputs.

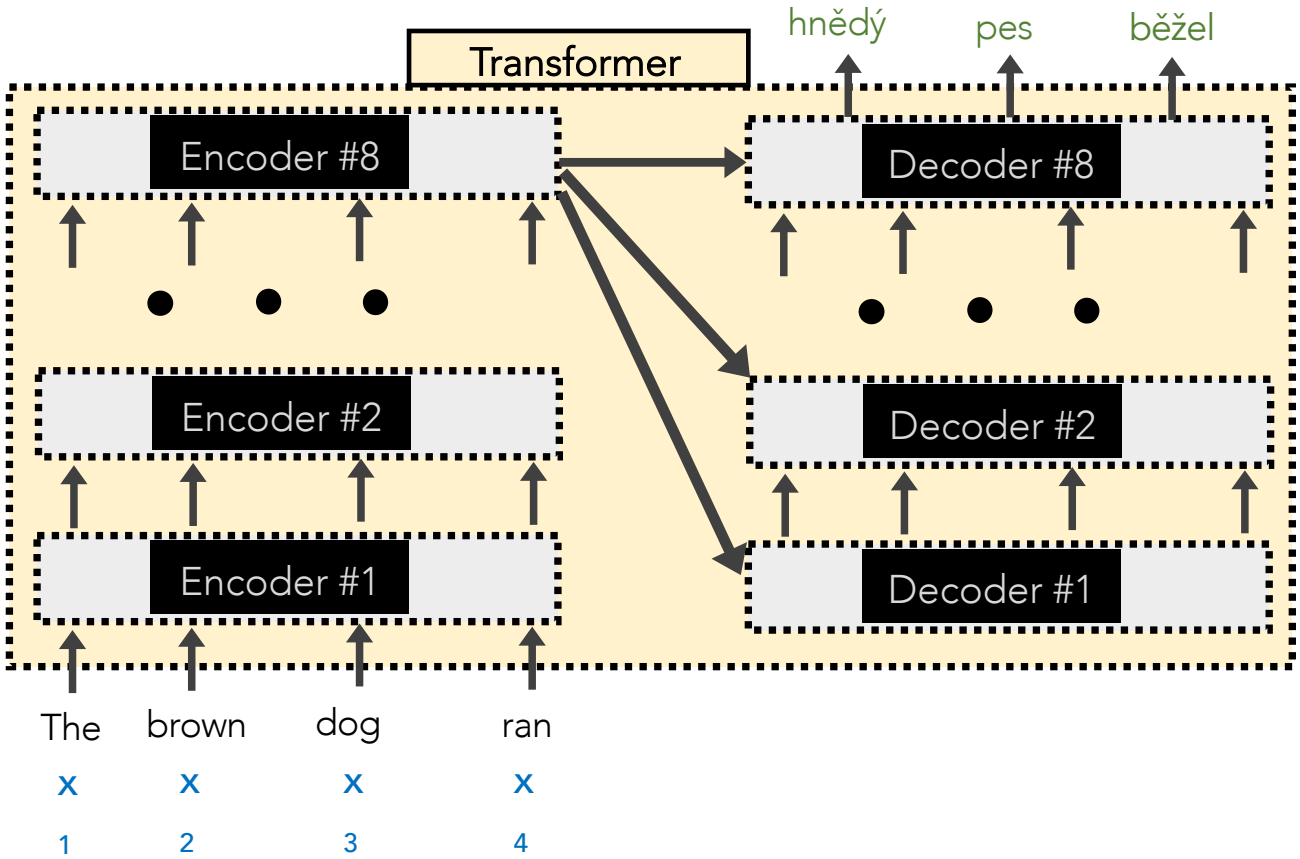
Transformer Encoders and Decoders



NOTE

The query, key, and value vectors for a Transformer Decoder's Self-Attention Head (not Attention Head) are all from the output of the previous decoder layer.

Transformer Encoders and Decoders



IMPORTANT

The Transformer Decoders have positional embeddings, too, just like the Encoders.

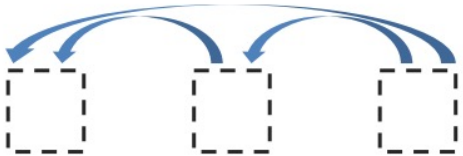
Critically, each position is only allowed to attend to the previous indices. This *masked* Attention preserves it as being an auto-regressive LM.

Transformer [Vaswani et al. 2017]

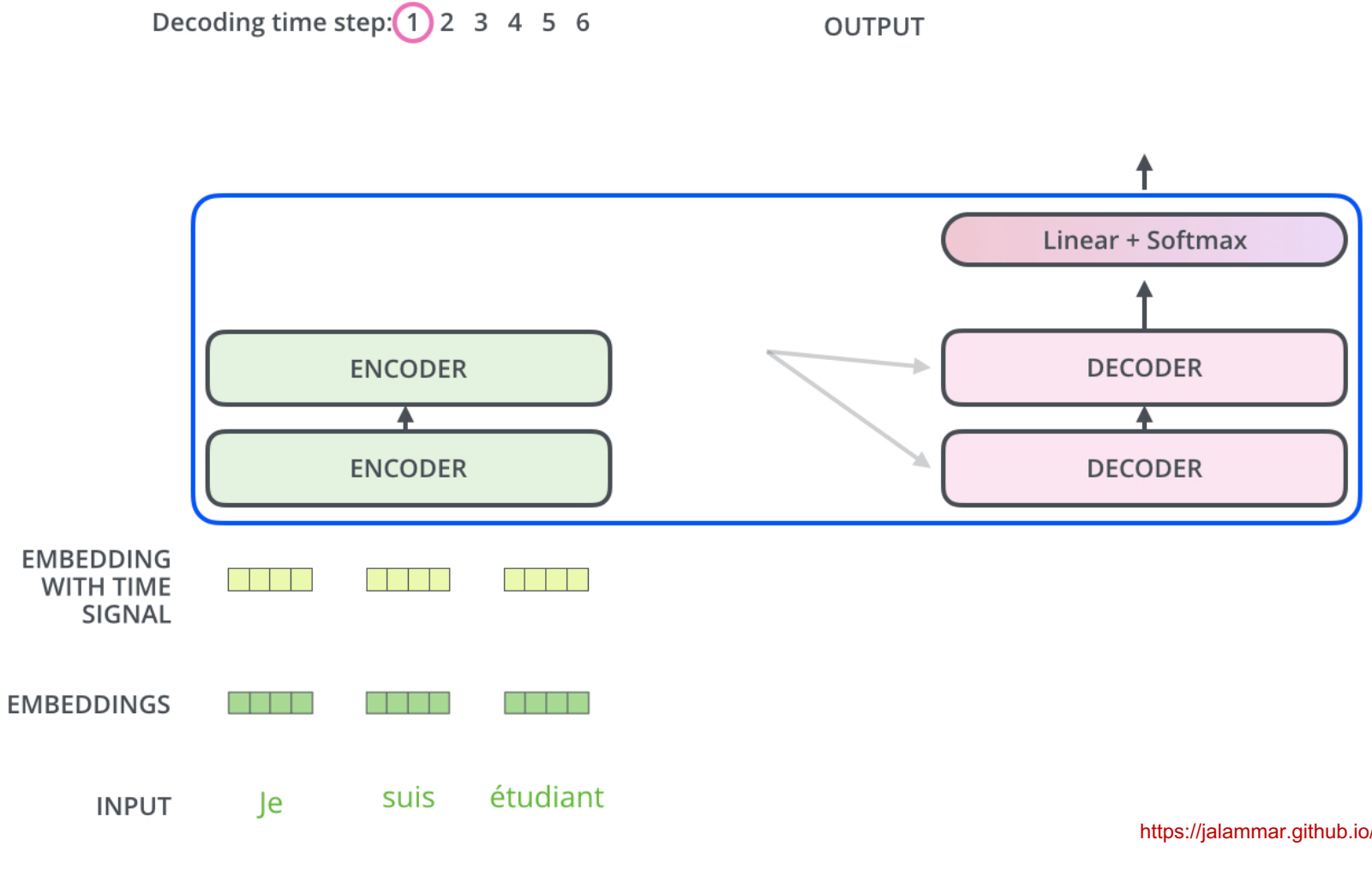
- An encoder-decoder architecture built with attention modules.
- 3 forms of attention

Encoder-Decoder Attention

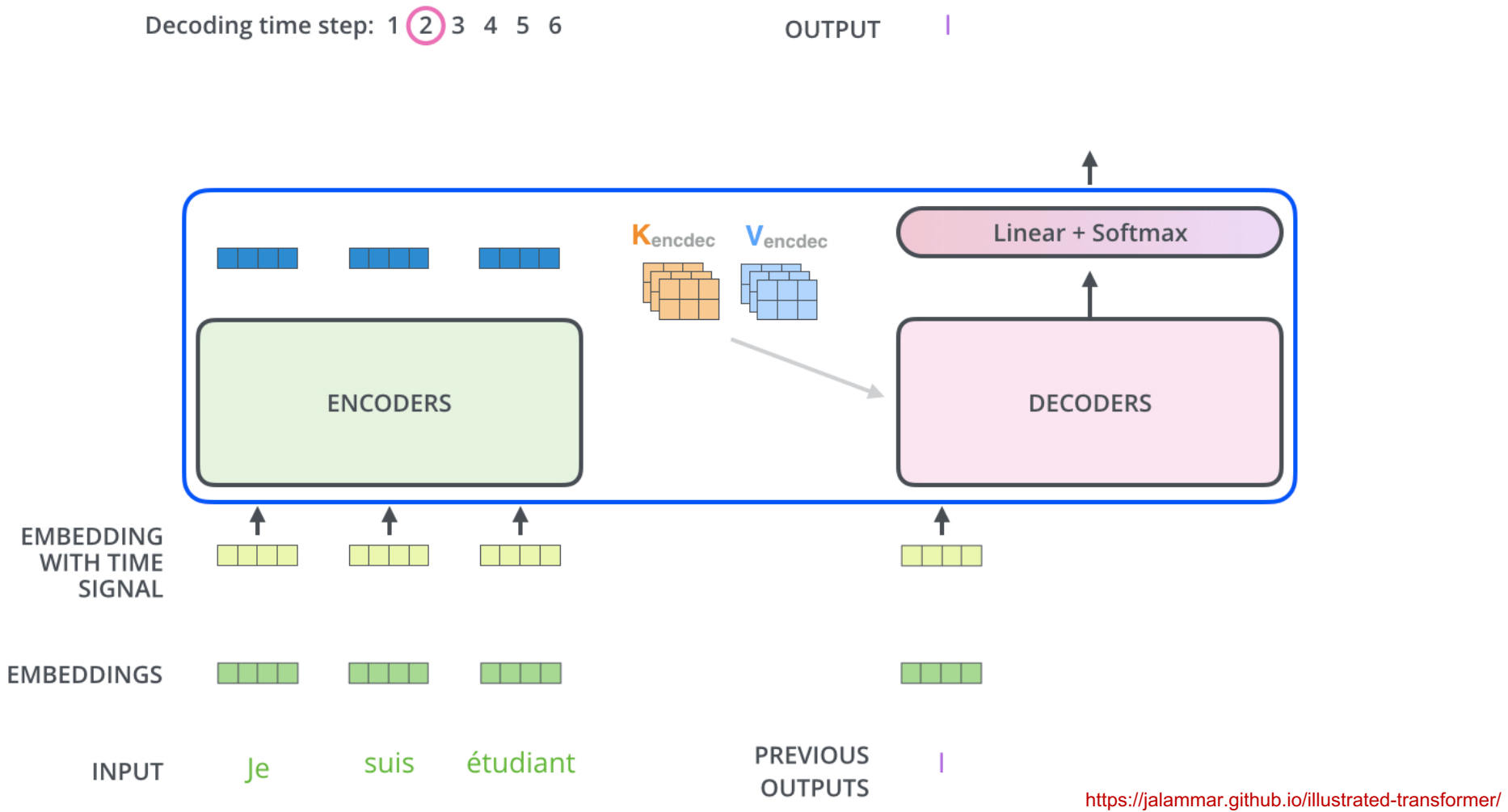
Encoder Self-Attention



MaskedDecoder Self-Attention



https://jalammar.github.io/illustrated-transformer/



https://jalammar.github.io/illustrated-transformer/

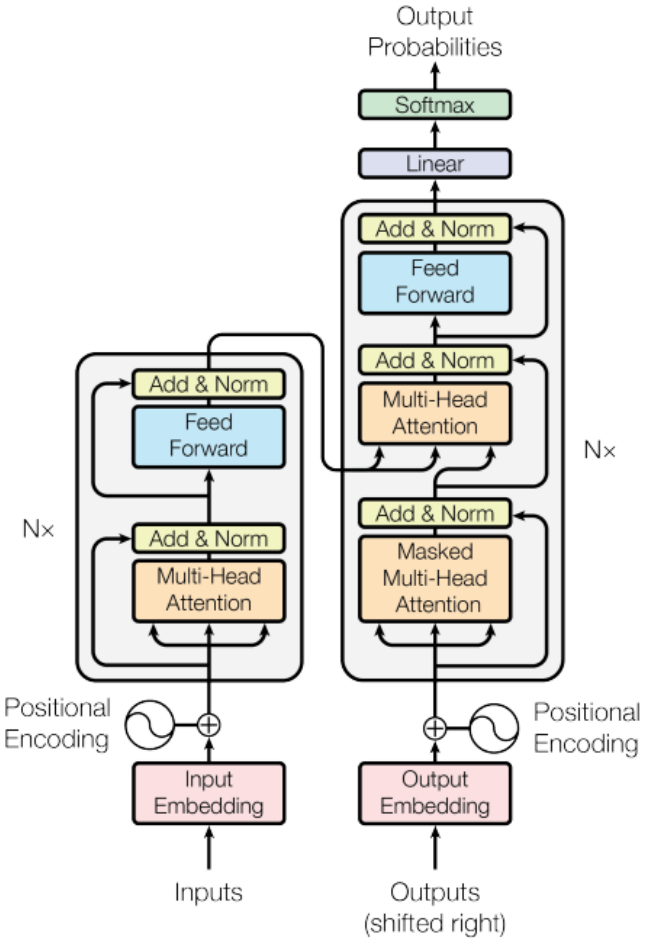
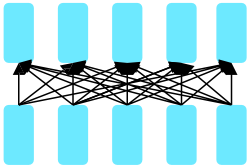


Figure 1: The Transformer - model architecture.

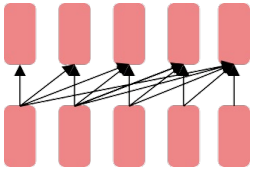
Impact of Transformers

- A building block for a variety of LMs



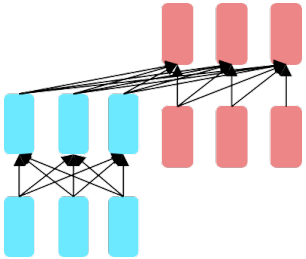
Encoders

- **Examples:** BERT, RoBERTa, SciBERT.
- Captures bidirectional context. How do we pretrain them?



Decoders

- **Examples:** GPT-2, GPT-3, Llama models, and many many more
- Other name: **causal or auto-regressive language model**
- Nice to generate from; can't condition on future words



Encoder-Decoders

- **Examples:** Transformer, T5, BART
- What's the best way to pretrain them?

Transformers - Summary

- Self-attention + positional embedding + others = NLP go brr
- Much faster to train than any previous architectures, much easier to scale
- Perform on par or better than previous RNN based models
  - Ease of scaling allows to extract much better performance

Language Modeling with Transformers

Testing: Measuring Probability

- Language model is trained using a decoder-only transformer – goal is to model the probability of each token given the prefix
  - Masked self-attention ensures we only condition on the prefix

- Computing probability of a sequence: (x1, x2, …, xn, </s>)
  - Input to the transformer: (<s>, x1, x2, …, xn)
  - The output is a sequence of probability distributions over the vocabulary via softmax. At input position t, the output represents the probability of the "next token".
  - Compute the probability of the sequence (x1, x2, … </s>) by indexing from the vocabulary distributions. -> this returns a vector
  - Multiply to get probability of the sequence

Language Modeling with Transformers

Training

- Goal: maximize the likelihood of a corpus.
- At training time, we know the "correct" next token.
  - Using previous slide, compute the probability or likelihood of the training corpus.
  - Define the loss as the "negative log likelihood" – same as classification (but now classes = size of the vocabulary)
  - The loss is also called the cross-entropy loss.

Language Modeling with Transformers

Testing: Generating Text

- The input:
  - Can just be <s>
  - Can be a "prompt" – a partial input which the model should continue such as: "<s> The cat".
- Input the prompt to the model
- At the last token, compute a probability distribution via softmax (over the vocabulary)
- "Sample" from this distribution to generate the next token – same as a throwing a "V" sided dice (with each side having a different probability).
- The sampled index is your next token. Let's say we sampled "ran"
- Now input to the transformer "<s> The cat ran". Repeat the process
- Stop when "</s>" is generated or a desired number of tokens are generated.

Agenda: Tokenization

I. Word-level tokenization
II. Character / byte level tokenization
III. Subword Tokenization
    I. BPE (primary topic)
    II. WordPiece
    III. Unigram (depending on time)

What's tokenization

Recall: A language model is a probability distribution over a sequence of "tokens"; each token is from a "vocabulary".

What is a token: "basic unit that need not be decomposed for further processing" [Webster and Kit, 1992]

Tokenization: Splitting or segmenting a "string" of text into a sequence of *tokens*

"I love watching the television"

$$\bar{X} = \langle \text{I, love, watching, the, television} \rangle$$

Why do we need to tokenize?

- Tokenization is not a typical preprocessing step in most machine learning domains
- Neural networks work with real-valued numbers. Most machine learning deals with continuous data but text is discrete
- Text needs to be converted to a form that a model can consume/generate.

Set of all tokens form a *vocabulary*

- Given a tokenization algorithm
  - Tokenize a corpus of text
  - Collect all unique tokens (aka types) → vocabulary

- The role of vocabulary in a language model
  - The vocabulary and its size is part of the model architecture
  - It defines the size of the input embedding table and final output layer
    - large vocabulary = more parameters

A simple tokenizer: Split by whitespace?

"I love watching the television"

$$\bar{x} = \langle \text{I, love, watching, the, television} \rangle$$

- Tokenization is not always this simple
- Such as, what will we do with the following strings:
  - "amazing!", "state-of-the-art", "unthinkable", "prize-winning", "aren't", "O'Neill"
  - Some languages don't even use spaces to mark word boundaries!

私は日本語を勉強しています

I am studying Japanese.

Çekoslovakyalılaştıramadıklarımızdanmışsınız

You are one of those whom we could not turn into a Czechoslovakian.

What about a word level tokenizer?

- Define each token as a word or a punctuation:
  - What is a word: smallest unit of language that carries meaning and can stand alone or combine with other units to create more complex meanings

- How to split into words?
  - For languages like English:
    - Could be simple regexes: split on all spaces and punctuation
      - What about "The value of pi = 3.14", "the IP address is 0.0.0.0"
      - What about "He got cold feet" → should cold feet be one or two words?
  - For languages like Chinese:
    - 我爱自然语言处理 (I love Natural Language Processing)
    - Need specialized tools (e.g. jieba)

Some related terminology

- A **morpheme** is the smallest meaning-bearing unit of a language
  - "unlikeliest" has the morphemes {un-, likely, -est}
- **Morphology** is the study of the way words are built up from morphemes
- **Word forms** are the variations of a word that express different grammatical categories
  - **Tense** (when something happened; past, present, future)
  - **Case** (inflecting nouns/pronoun and their modifiers to express their relationship with other words; English has largely lost its inflected case system)
  - **Number** (singular/plural)
  - **Gender** (masculine, feminine, neuter; not extensively used in English)
- and thus help convey the specific meaning and function of the word in a sentence

Word level tokenizer – How to define a vocabulary?

- Given a tokenization algorithm
  - Tokenize a corpus of text
  - Collect all unique tokens → vocabulary

- For a English corpus, a large corpus can have 1M+ unique words
  - Vocabulary becomes too large

- A popular solution: Cut this list to include only K tokens.
  - How to chose K – based on frequency
  - What to do with the rest? – replace with an UNK "unknown" token
  - Word level tokenizers lead to "closed" vocabulary models.

Handling Unknown Words

- What happens when we encounter a word that we have never seen in our training data?
  - Not much we can do
  - Except assigning to it a special <UNK> token
  - Why this is bad?

Limitations of <UNK>

- Generally, we lose most of the information the word conveys. UNKs don't give features for novel words that are useful anchors of meaning
- What if you want to generate a word which is not in the vocabulary? Imagine ChatGPT generating UNK.
- Especially hurts in [productive] languages with many rare words/entities
    - *The chapel is sometimes referred to as "Hen Gapel Lligwy" ("hen" being the Welsh word for "old" and "capel" meaning "chapel").*
    - *The chapel is sometimes referred to as " Hen <unk> <unk> " (" hen " being the Welsh word for " old " and " <unk> " meaning " chapel ").*

Word Level Tokenization

Other Limitations

- Word-level tokenization treats different forms of the same root as completely separate (e.g., “open”, “opened”, “opens”, “opening”, etc)
- This means separate features or embeddings.

But deciding what counts as a word in Chinese is complex. For example, consider the following sentence:

(2.4) 姚明进入总决赛
“Yao Ming reaches the finals”

As Chen et al. (2017) point out, this could be treated as 3 words (‘Chinese Treebank’ segmentation):

(2.5) 姚明 进入 总决赛
YaoMing reaches finals

or as 5 words (‘Peking University’ segmentation):

(2.6) 姚 明 进入 总 决赛
Yao Ming reaches overall finals

Finally, it is possible in Chinese simply to ignore words altogether and use characters as the basic elements, treating the sentence as a series of 7 characters:

(2.7) 姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

In fact, for most Chinese NLP tasks it turns out to work better to take characters rather than words as input, since characters are at a reasonable semantic level for most applications, and since most word standards, by contrast, result in a huge vocabulary with large numbers of very rare words (Li et al., 2019).

A simpl*er* tokenizer: Character-level

I love watching the television

<I _ l o v e _ w a t c h i n g _ t h e _ t e l e v i s i o n>

Issues

- Make sequences very long
- Word-level models provide an inductive bias to models. BUT character-level models must learn to compose characters into words.
    - Need deeper models

[Jurafsky and Martin Sec 2.4 p17]

Current standard: *Subword* Tokenization

- "Word"-level: issues with unknown words and information sharing, and gets complex fast
  - Also, fits poorly to some languages
- Character-level: long sequences, the model needs to do a lot of heavy lifting in representing that is encoded in plain-sight
- **Let's find a middle ground!**
- Subword tokenization first developed for machine translation
  - Based on byte pair encoding (Gage, 1994)
- Now, used everywhere

Neural Machine Translation of Rare Words with Subword Units

Rico Sennrich and **Barry Haddow** and **Alexandra Birch**
School of Informatics, University of Edinburgh
{rico.sennrich,a.birch}@ed.ac.uk,
bhaddow@inf.ed.ac.uk

The main motivation behind this paper is that the translation of some words is transparent in that they are translatable by a competent translator even if they are novel to him or her, based on a translation of known subword units such as morphemes or phonemes.

A redefinition of the notion of tokenization

Due to:

- Scientific results: The impact of sub-word segmentation on machine translation performance in 2016
- Technical requirements: A fixed-size vocabulary for neural language models & a reasonable vocabulary size

…in current NLP, the notion of *token* and *tokenization* changed

"Tokenization" is now the task of segmenting a sentence into non-typographically (& non-linguistically) motivated units, which are often smaller than classical tokens, and therefore often called **sub-words**

Typographic units (the "old" tokens) are now often called **"pre-tokens",** and what used to be called "tokenization" is therefore called nowadays **"pre-tokenization"**

- https://github.com/huggingface/tokenizers/tree/main/tokenizers/src/pre_tokenizers

Unseen word can be represented by some sequence of known subwords; "lower"="low"+ "er"

[Mielke et al., 2021]

Byte-Pair-Encoding (BPE)

[coined by Gage et al., 1994; adapted to the task of word segmentation by Sennrich et al., 2016; see Gallé (2019) for more]

Main idea: Use our data to automatically tell us what the tokens should be

Token learner:

Raw train corpus ⇒ Vocabulary (a set of tokens)

Token segmenter:

Raw sentences ⇒ Tokens in the vocabulary

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – **Token learner**

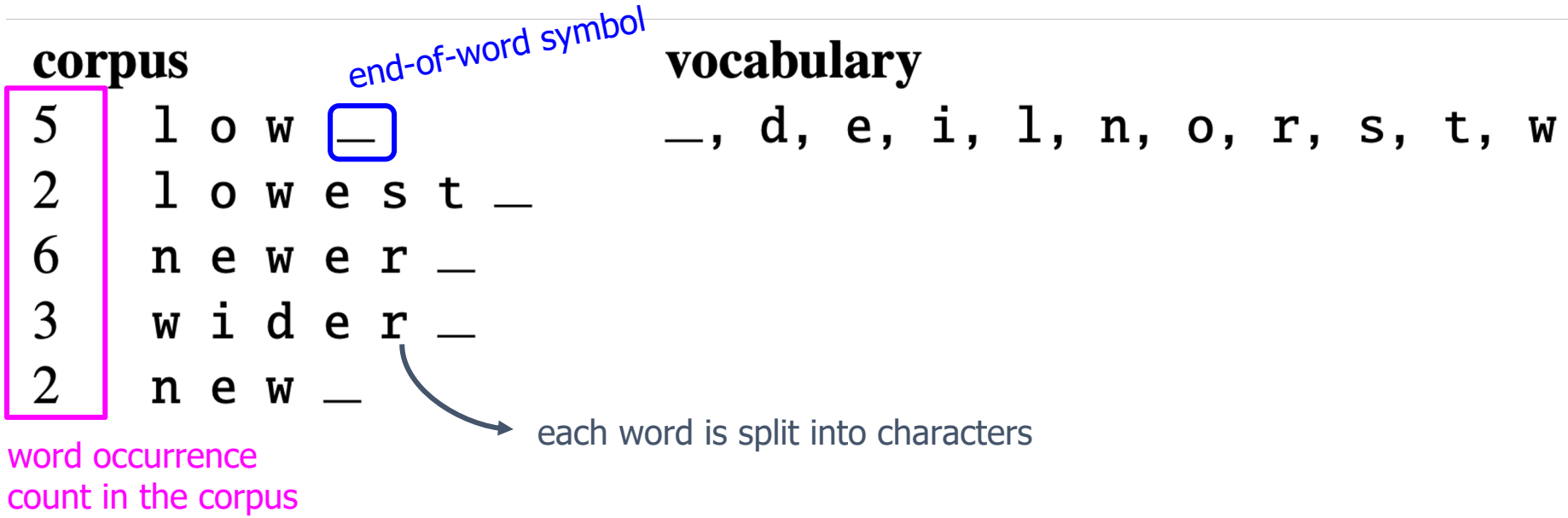
[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

Raw train corpus ⇒ Vocabulary (a set of tokens)

- Pre-tokenize the corpus in words & append a special end-of-word symbol _ to each word
- Initialize vocabulary with the set of all individual characters
- Choose 2 tokens that are most frequently adjacent ("A", "B")
  - Respect word boundaries: Run the algorithm inside words
- Add a new merged symbol ("AB") to the vocabulary
- Change the occurrence of the 2 selected tokens with the new merged token in the corpus
- Continues doing this until k merges are done

[Jurafsky & Martin (2023)]

Byte-Pair-Encoding (BPE) – Token learner *Example*



[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `e r` [a total of 9 occurrences]
- Merge these symbols, treating `er` as one symbol, & add the new symbol to the vocabulary

corpus

```
5   l o w _
2   l o w e s t _
6   n e w er _
3   w i d er _
2   n e w _
```

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair `er _`
- Merge these symbols, treating `er_` as one symbol, & add the new symbol to the vocabulary

corpus

```
5   l o w _
2   l o w e s t _
6   n e w er_
3   w i d er_
2   n e w _
```

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

- Counts all pairs of adjacent symbols
- The most frequent is the pair n e
- Merge these symbols, treating ne as one symbol, & add the new symbol to the vocabulary

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – Token learner *Example*

merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Final vocabulary

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – **Token segmenter**

[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

- The token segmenter is used to tokenize a test sentence
  - Goal: Apply the merges we've learned from the training data, greedily, in the order we learned them

- First, we segment each test sentence word into characters

- Then, we apply the first merge rule
  - E.g., replace every instance of e r in the test corpus with er

- Then the second merge rule
  - E.g., replace every instance of er _ in the test corpus with er_

- And so on

[Example from Jurafsky & Martin (2023), pages 18–19]

Byte-Pair-Encoding (BPE) – **Token segmenter**

[coined by Gage et al., 1994; adapted to the task of word segmentation by **Sennrich et al., 2016**; see Gallé (2019) for more]

- Test example: slow_ → s l o w_ → s lo w_ -> s low_

- Test example: now → n o w

BPE can tokenize a word never seen at training time.

Leads to an open vocabulary model (well, almost)

Can often learn morphological segmentations

Deescalation → De escalat ion

[Example from Jurafsky & Martin (2023), pages 18–19]

A variant of BPE: WordPiece

used in BERT and some follow ups

- Training algorithm: Same as BPE
- Segmentation algorithm: greedily pick the longest prefix that exists in the vocabulary (and repeat)
    - Slow_ → s low_ [stop]

merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

BPE/Wordpiece summary

- Start with a character vocabulary
- Merge frequent bigrams
- Repeat until a desired vocab size/merge size is reached.

Unigram LM Tokenizer

1. Start with a **large base vocabulary**, **remove tokens** until a desired size is reached.

1. **How to construct a base vocabulary**: all substrings of pre-tokenized words OR start with a large BPE vocabulary.

1. **How to remove tokens:**
   a. Compute **the unigram LM loss** over the corpus (more details later)
   b. Removing tokens increases this loss.
   c. Select and remove tokens that increase it the least.
   d. Repeat

Base Vocabulary

- The corpus:

  ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- Initial Vocabulary (all strict substrings)

  ("h", 15) ("u", 36) ("g", 20) ("hu", 15) ("ug", 20) ("p", 17) ("pu", 17) ("n", 16)

  ("un", 16) ("b", 4) ("bu", 4) ("s", 5) ("hug", 15) ("gs", 5) ("ugs", 5)

Unigram LM loss

- Unigram LM loss = negative log **probability of the corpus**.

- **Probability of a corpus** = product of marginal **probability of individual words**

$$p(\text{pug hugs bugs}) = p(\text{pug}) \times p(\text{hugs}) \times p(\text{bugs})$$

Probability of a word

- product of marginal probability of its subwords (based on frequency)

  p(pug) = p(“p”) x p(“u”) x p(“g”)

  Or, p(pug) = p(“pu”) x p(“g”)

  Or, p(pug) = p(“p”) x p(“ug”)

  Choose highest of all possible splits (why?)

- How to this efficiently: Dynamic programming (Viterbi algorithm)

Unigram Tokenization Algorithm

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

3. For every token, w, in the vocabulary

VERY SLOW!

   a. Remove w from the vocabulary and recompute the loss, L'(w)
   b. Define score(w) = L'(w) - L

4. Compute w* = min_w score(w).
   a. Remove w* from the vocabulary.
   b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

2. Compute the unigram loss, L, over the corpus

3. For every token, w, in the vocabulary
    a. Remove w from the vocabulary and recompute the loss, L'(w)
    b. Define score(w) = L'(w) - L

4. Compute **W** = x% of tokens with the lowest score.
    a. Remove w* from the vocabulary.
    b. Got to step 2. Repeat until a desired vocabulary size is reached.

Unigram Tokenization Algorithm (Slightly Faster)

1. Start with a base vocabulary

1. Compute the unigram loss, L

1. For every token, w, in the vocabulary
   a. Remove w from the vocabulary and recompute the loss, L'(w)
   b. Define score(w) = L'(w) - L

1. Compute **W** = x% of tokens with the lowest score.
   a. Remove **W** from the vocabulary.
   b. Go to step 2.

How to tokenize once the vocabulary is decided

- Tokenization which maximizes the unigram probability of the word
- (or find top *k* tokenizations)
- "Unhug" (For each position, the subwords with the best scores ending in that position:)

  Character 0 (u): "u" (score 0.171429)

  Character 1 (n): "un" (score 0.076191)

  Character 2 (h): "un" "h" (score 0.005442)

  Character 3 (u): "un" "hu" (score 0.005442)

  Character 4 (g): "un" "hug" (score 0.005442) **[final tokenization]**

Unigram vs BPE

1. Why unigram over BPE and WordPiece?
    a. Unigram finds optimal coding length of a sequence (according to Shannon's entropy).
    b. Unigram allows sampling multiple tokenizations for every word – subword regularization – robustness

1. Why is Unigram tokenization not more popular?
    a. In many papers simply referred to as SentencePiece, which is actually not a tokenizer but a library wrapping several tokenizers
    b. Does the actual subword tokenizer matter as we scale up models?

Subword methods are not TRULY open-vocabulary

- What if you encounter a character not seen at training time?

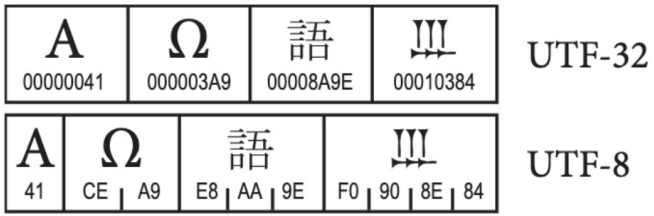
- For example, a BPE model trained on only English encounters a Chinese character at test time --- it gets assigned UNK

- How to solve this issue?
  - Train on a mix of all characters? Accounting for all languages, they are millions of characters – vocabulary size would be too large

Solution: Byte-level subword Models (BBPE)

- Every written language is represented in Unicode

Unicode is a text encoding standard designed to support the use of text in all of the world's writing systems that can be digitized. It has multiple versions like UTF-8, UTF-16, UTF-32. UTF-8 is the most common one.



- Each character is a sequence of “bytes”.
  - Each byte is 8-bit. Total 256 unique byte values
  - Each character requires between 1 to 4 bytes.

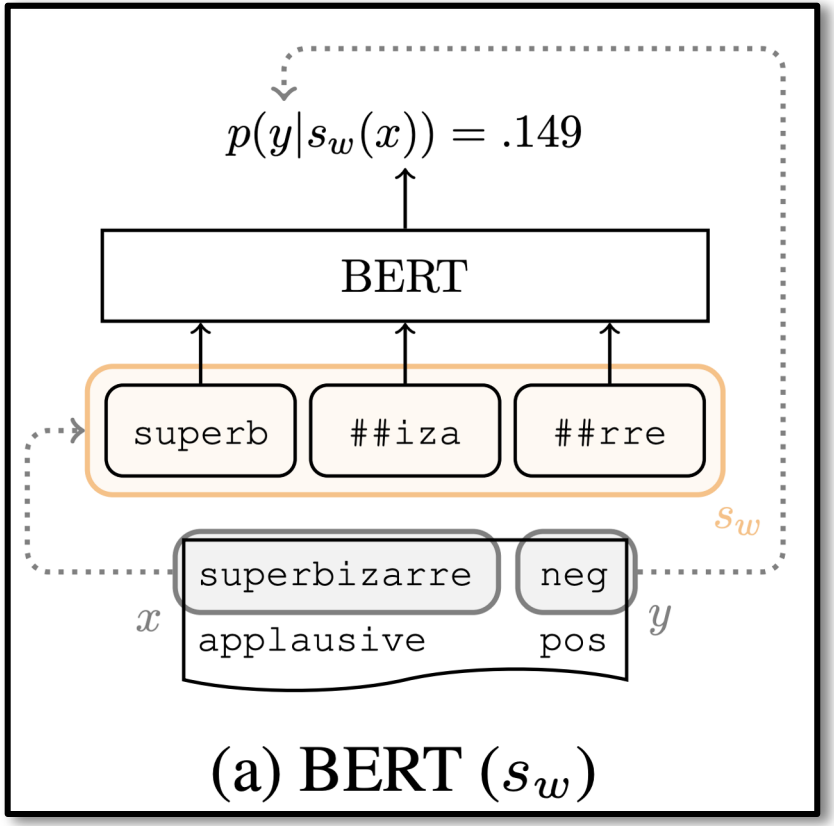
Solution: Treat a corpus as a sequence of byte. Start the vocabulary with all bytes (256) and train a BPE model. TRULY Open Vocabulary. Works for all characters!*

Subword Models -- Summary

- Split text in tokens learned statistically from the training corpus
- Makes the model open vocabulary
- Subword methods prove to be an effective method of “compressing text”

Issues with subword models

BERT thinks the sentiment of "superbizarre" is positive because its tokenization contains the token "superb"



[Hofmann et al., 2021]

Non-concatenative Languages

كتب	k-t-b	"write" (root form)
كَتَبَ	**k**a**t**a**b**a	"he wrote"
كَتَّبَ	**k**a**tt**a**b**a	"he made (someone) write"
اِكْتَتَبَ	i**k**ta**t**a**b**a	"he signed up"

Table 1: Non-concatenative morphology in Arabic.[4] The root contains only consonants; when conjugating, vowels, and sometimes consonants, are interleaved with the root. The root is not separable from its inflection via any contiguous split.

Subword Tokenization and “noise”

- He fell and broke his **coccix** (vs coccyx)
- Neighbor = 1 token, neighbour = two tokens
- John Smith = 2 tokens, Srinivas Ramanujam = ??

Subword Tokenization and “numbers”

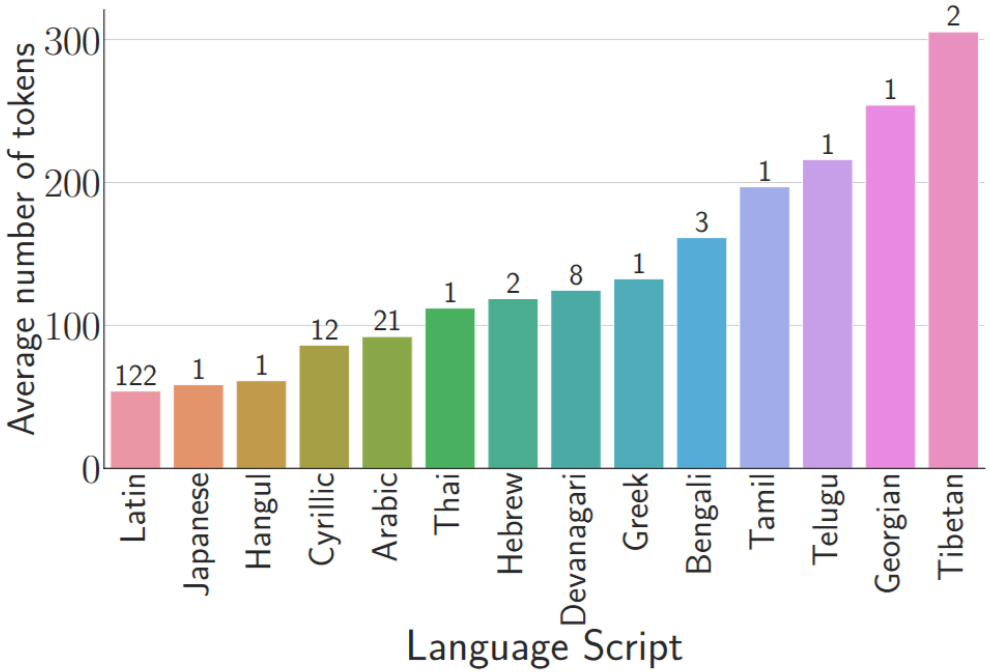
- Why are LMs bad at basic arithmetic?
  - 3.14 is tokenized as 3.14
  - 3.15 is tokenized as 3 . 15

Natural phenomena like diacritics or a little easily human-readable noise lead to unexpected BPE sequences and catastrophic translation failure

Arabic–English	
src	أنا كندية، وأنا أصغر أخواني السبعة
diacritics 1.0	أَنا كَنَدِيَّةٌ ، وَأَنا أَصْغَرِ إِخْوانِي السَبْعَةِ
ref	I'm Canadian, and I'm the youngest of seven kids.
in_{vis}	
out_{vis}	I'm a Canadian, and I'm the youngest of my seven sisters.
COMET	0.764
in_{text}	_أَ ان _ك نَ دَ ي ّ َة ِ_, _و َ_أ َان _أ َص ْغ َر ِ_إ ِخ ْاو ن ِي _سلا َب ْع َ_ة .
out_{text}	We grew up as a teacher, and we gave me a hug.
COMET	-1.387

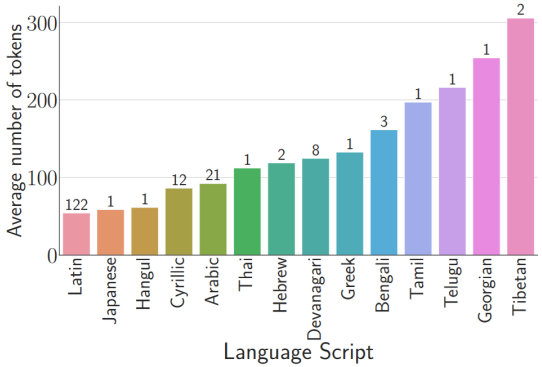
[Salesky et al., 2021]

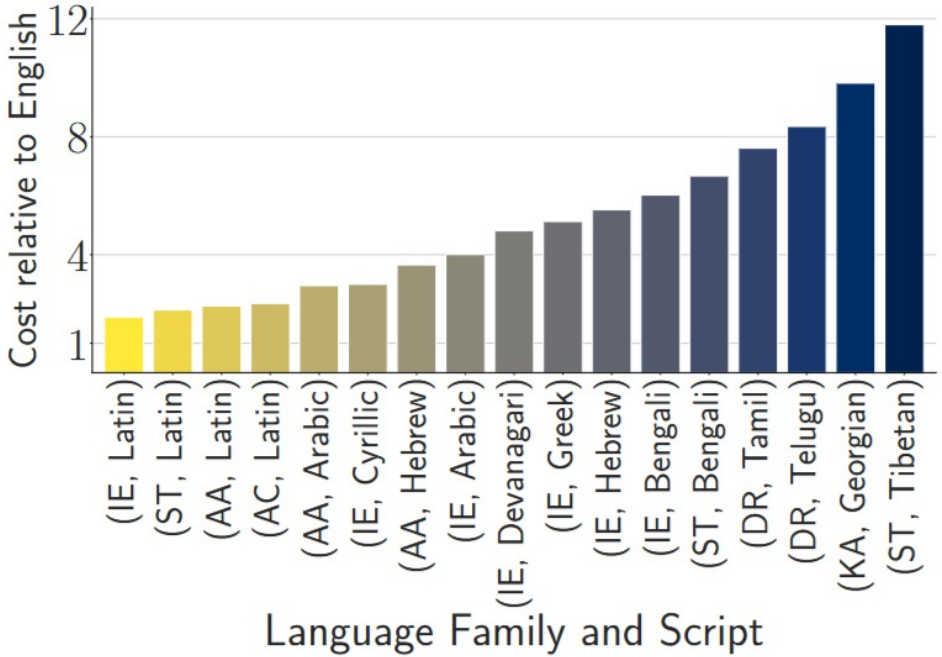
Sequence Lengths, Costs, and Performance



[2305.13707] Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models

Sequence Lengths, Costs, and Performance

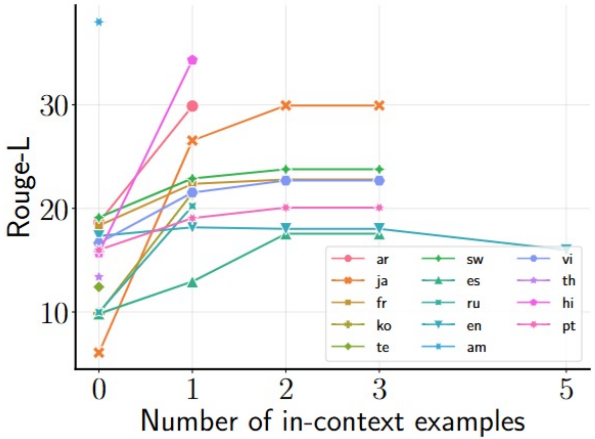




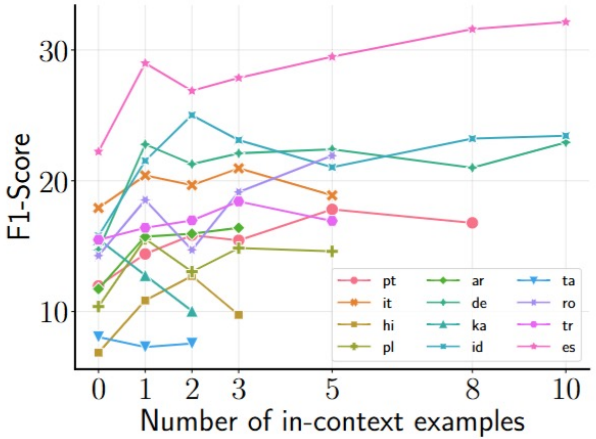
LLM APIs charge per token

[2305.13707] Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models

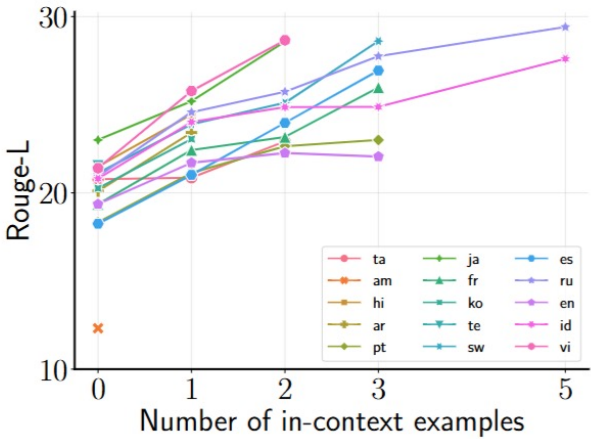
Sequence Lengths, Costs, and Performance



(a) XLSUM



(b) XFACT



(c) CROSS-SUM

[2305.13707] Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models

Tokenization

Tokenization is at the heart of much weirdness of LLMs. Do not brush it off.

- 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 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**.

[`Let's build the GPT Tokenizer' by Andrej Karpathy]


Cody Blakeney
@code_star
Do you guys ever think about tokenizers?


Prithviraj (Raj) Ammanabrolu @rajammanabrolu · Sep 26, 2023
If there's one thing standing in the way of AGI, it's tokenizers


Luca Soldaini @soldni · Aug 21
0
DAYS WITHOUT
TOKENIZATION
ACCIDENTS
0
imgflip.com

Recent tweaks to subword tokenizers

- Tokenize each digit separately (i.e. no merge on digits)

- Add special tokens to deal with everything else:
  - E.g. special tokens for keywords from programming languages

- Train the tokenizer on a more balanced dataset.
  - Apply tricks like “alpha-sampling” – up sample lower resource languages and scripts to up their frequency.
  - Does not solve all issues but helps.