Transformers and Pretraining

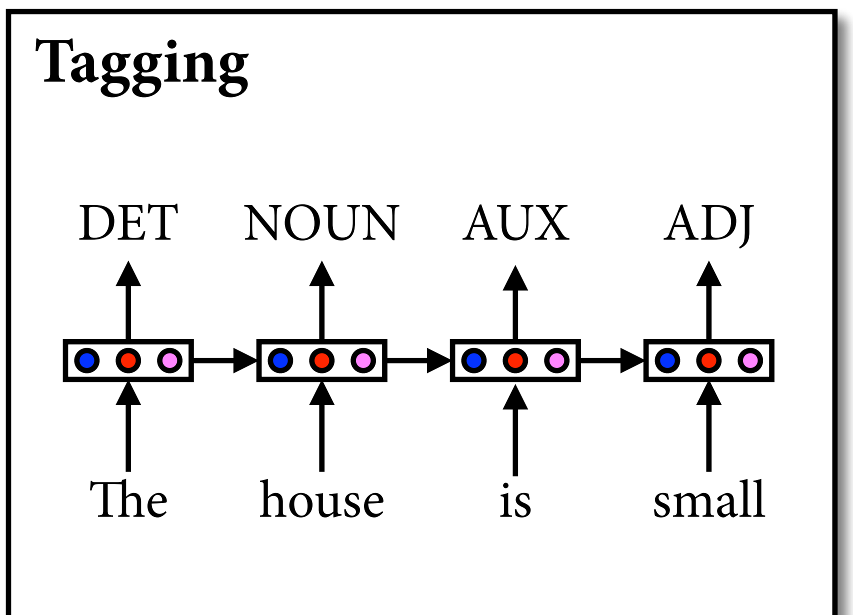
Computational Linguistics

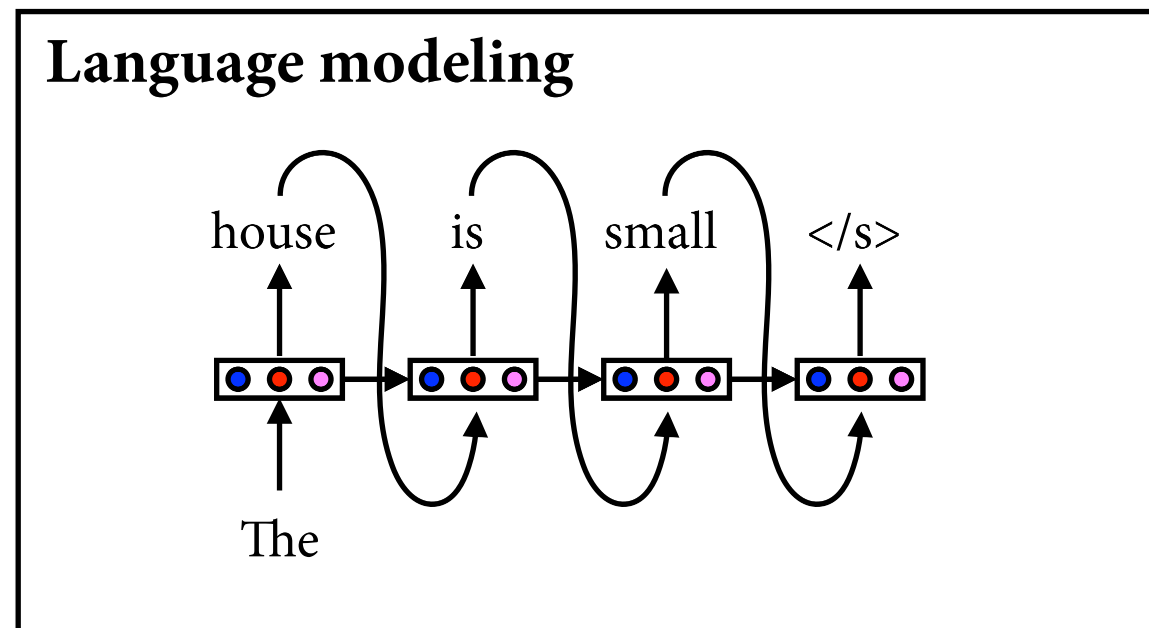
Alexander Koller

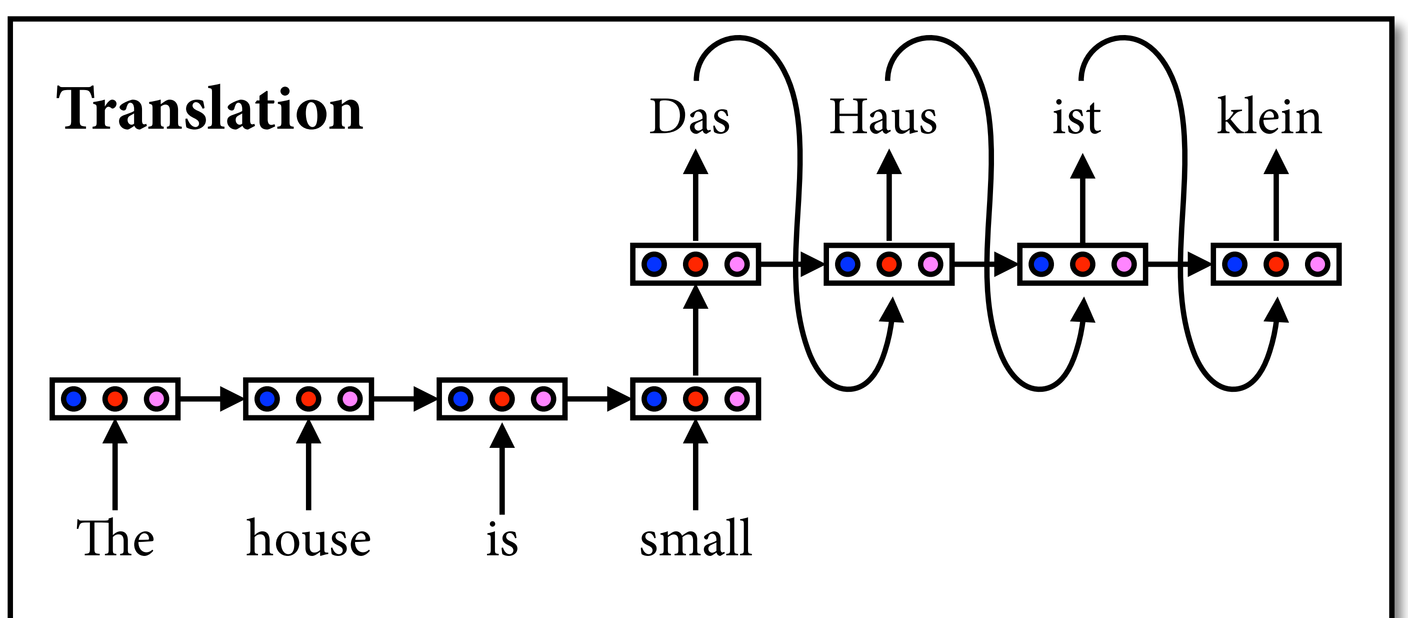
08 December 2023

Recurrent neural networks

- When we process language, we usually don't have inputs of fixed length (sentences can be arbitrarily long).

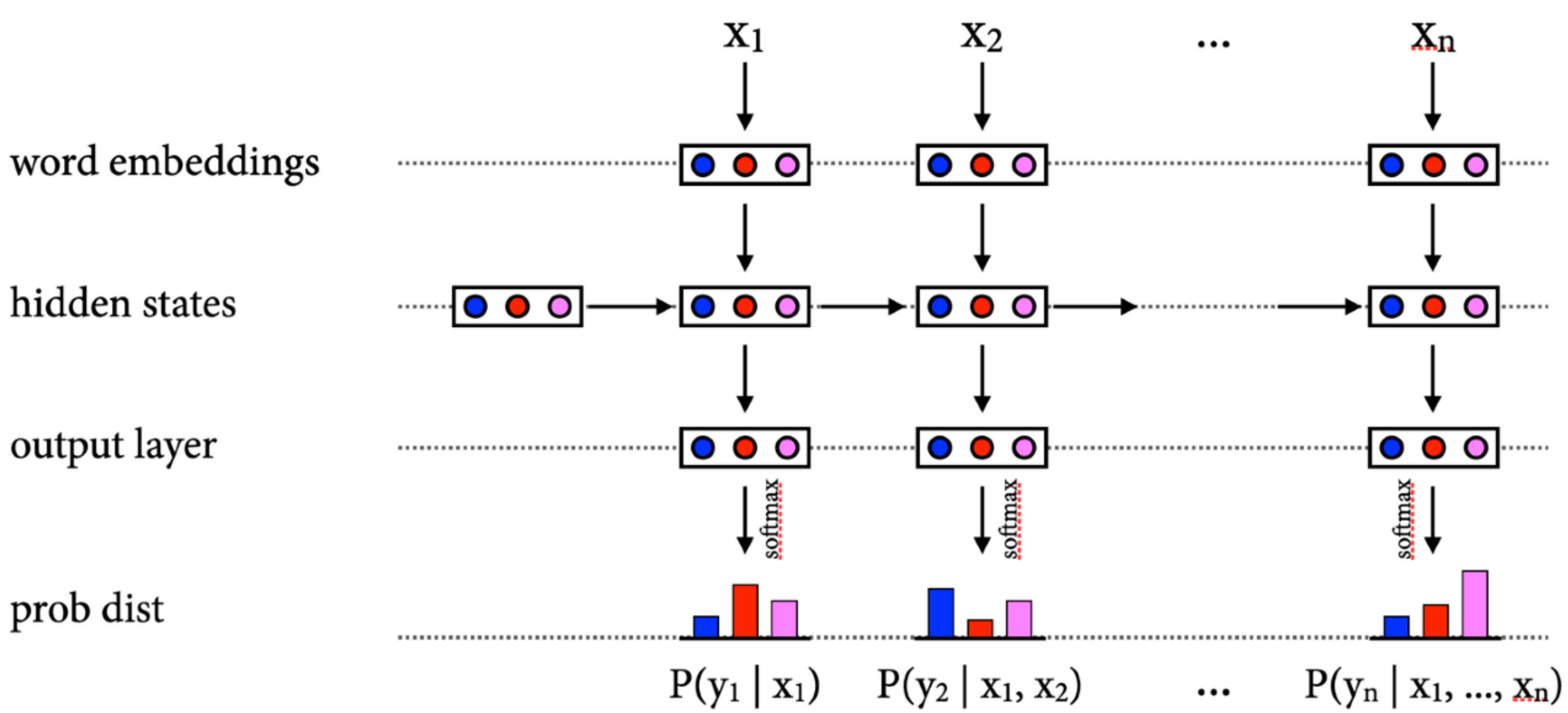






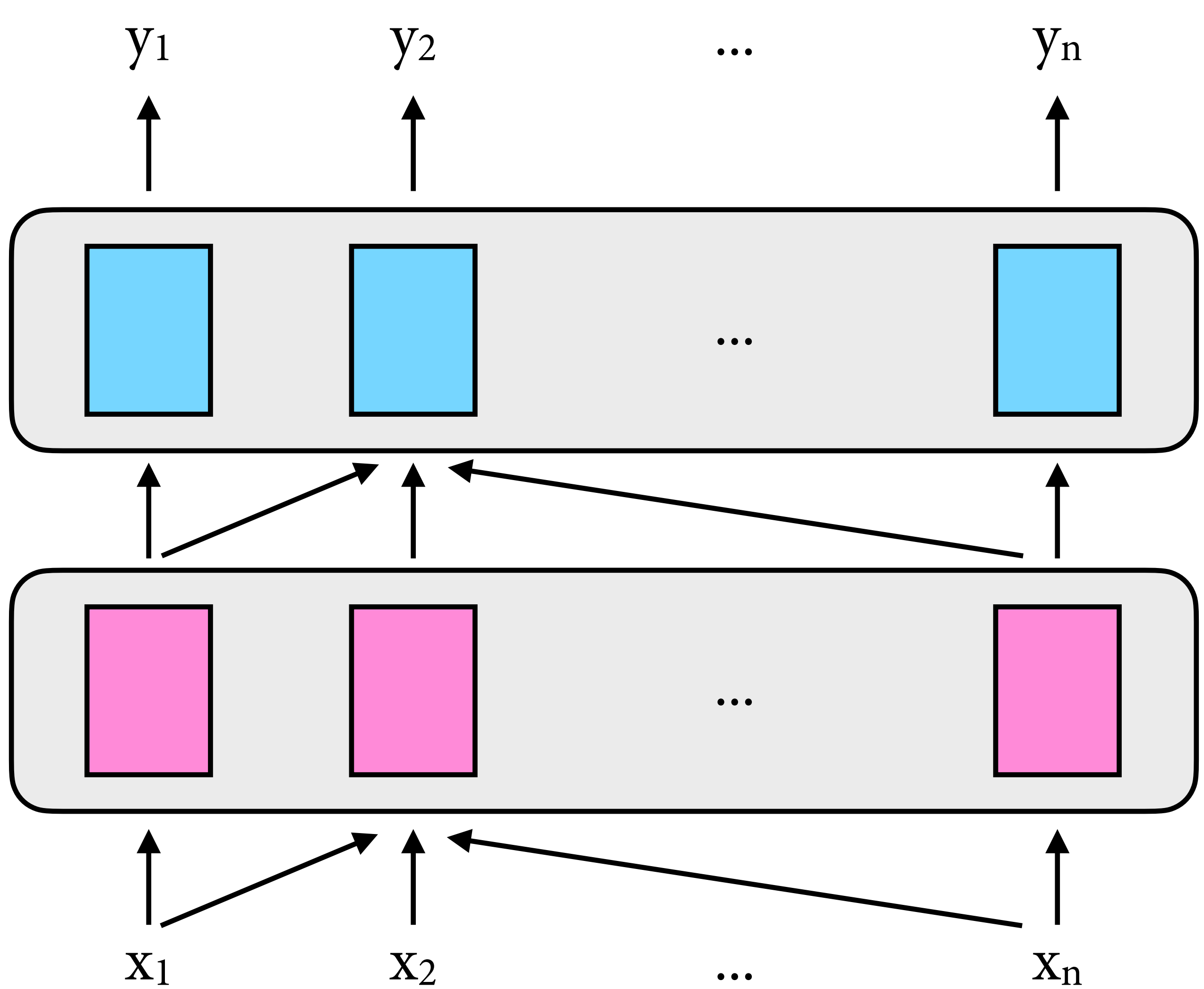
Limitations of RNNs

- Long computation paths make backprop hard.
- Computing loss at timestep t requires doing all the computations at times 1, ..., t-1.
  - Computations for different timesteps are different.
  - Can't parallelize these computations.

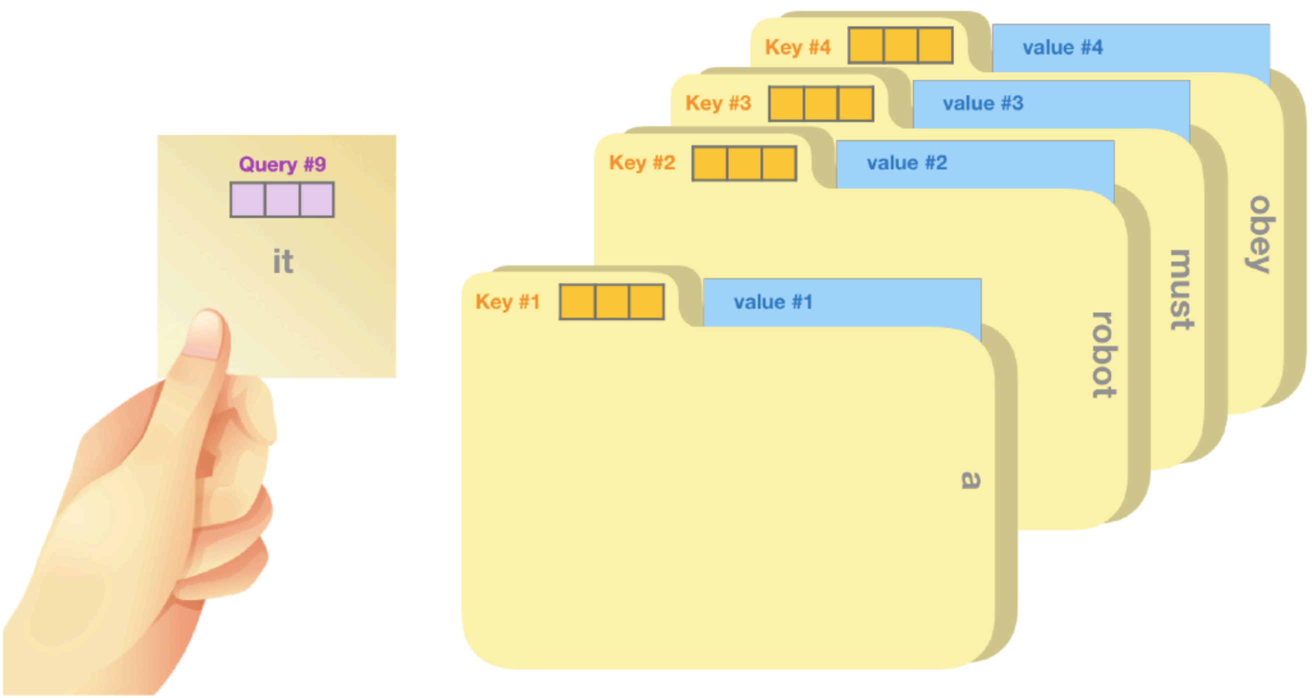


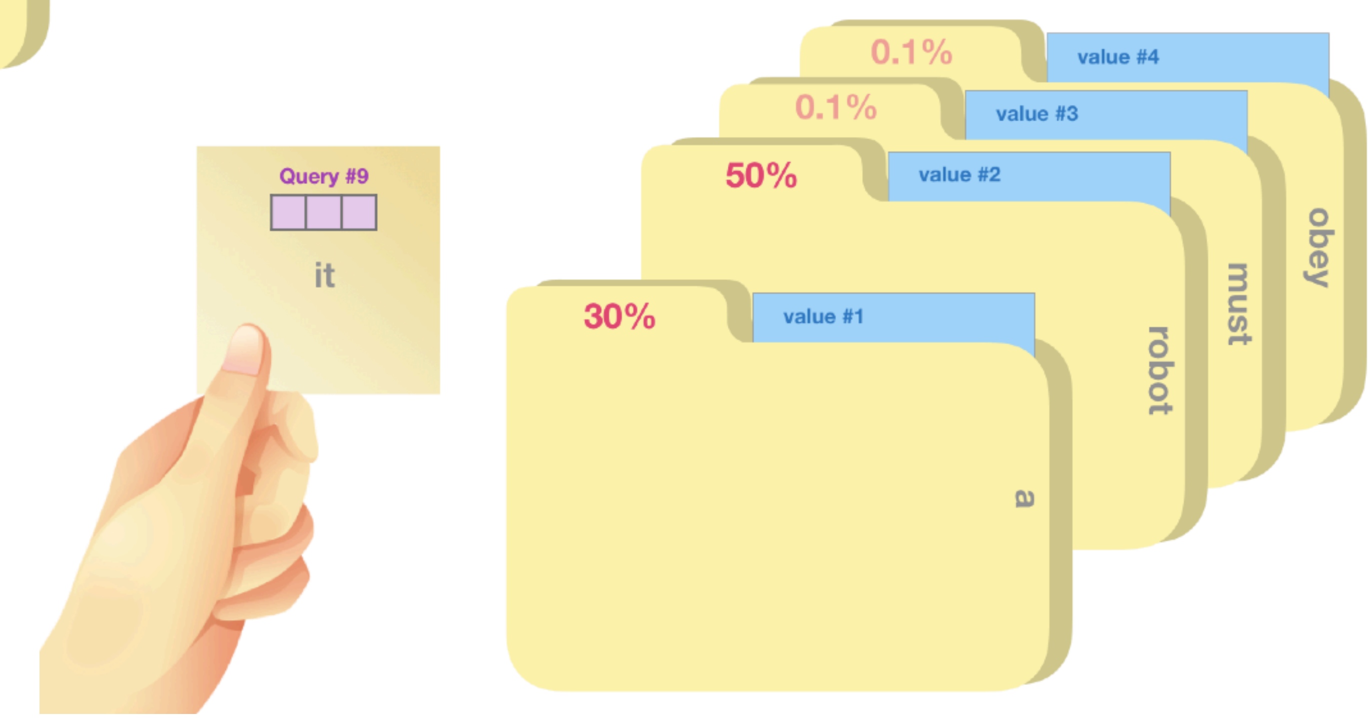
Transformers: Big picture

Encode sequence of inputs in parallel!



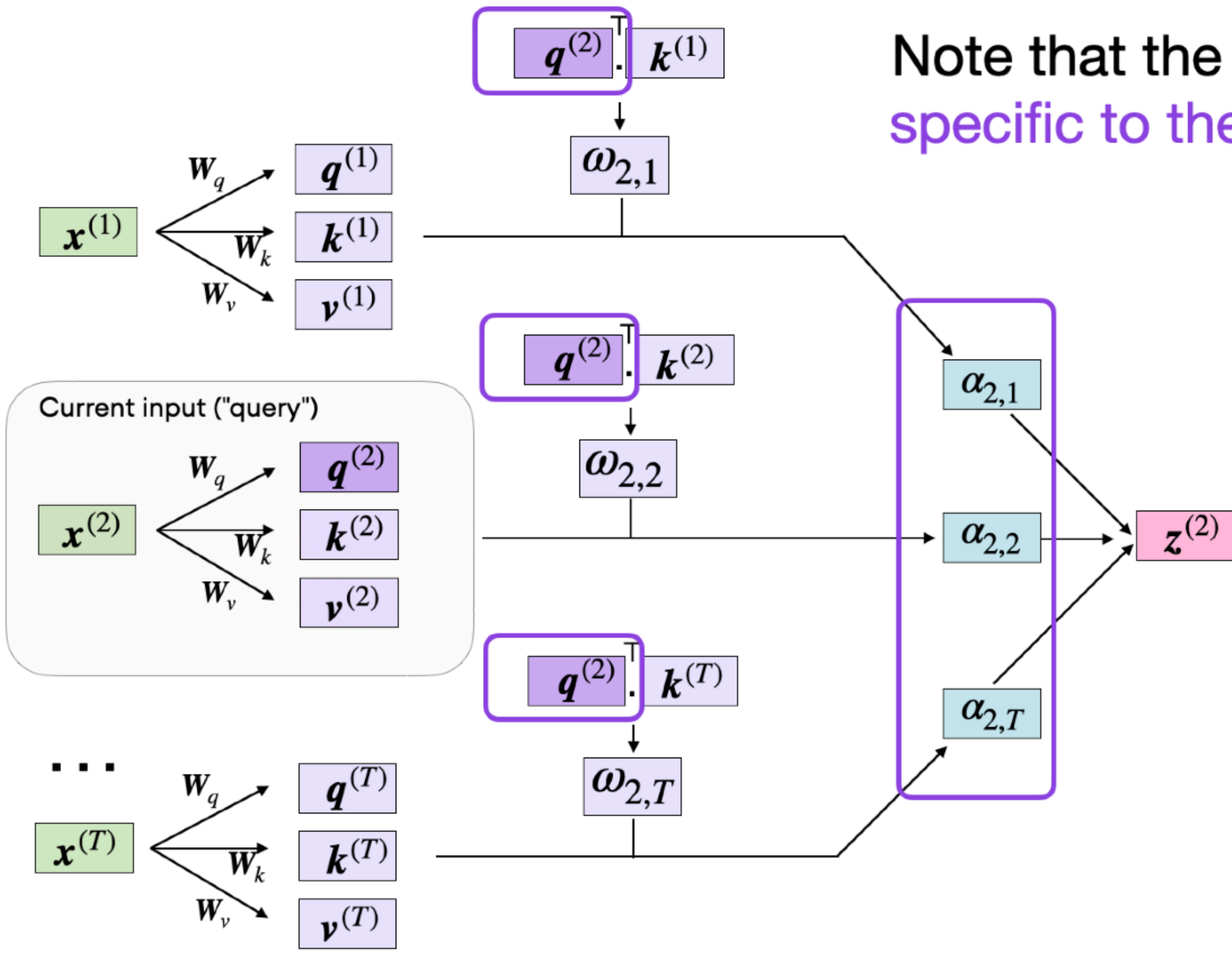
Attention: Big picture





https://jalammar.github.io/illustrated-gpt2/

Attention: Details

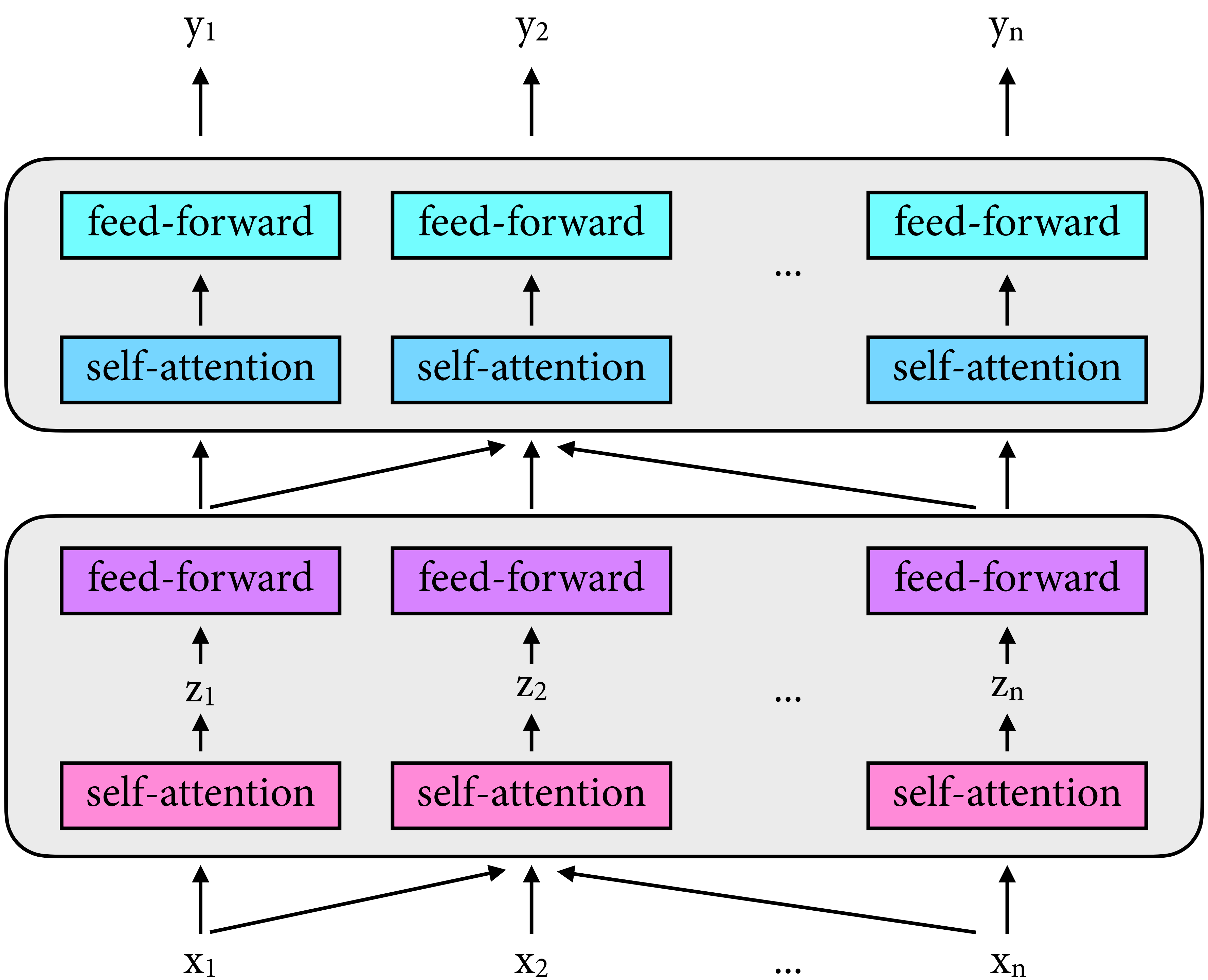


Note that the attention scores are specific to the current input token

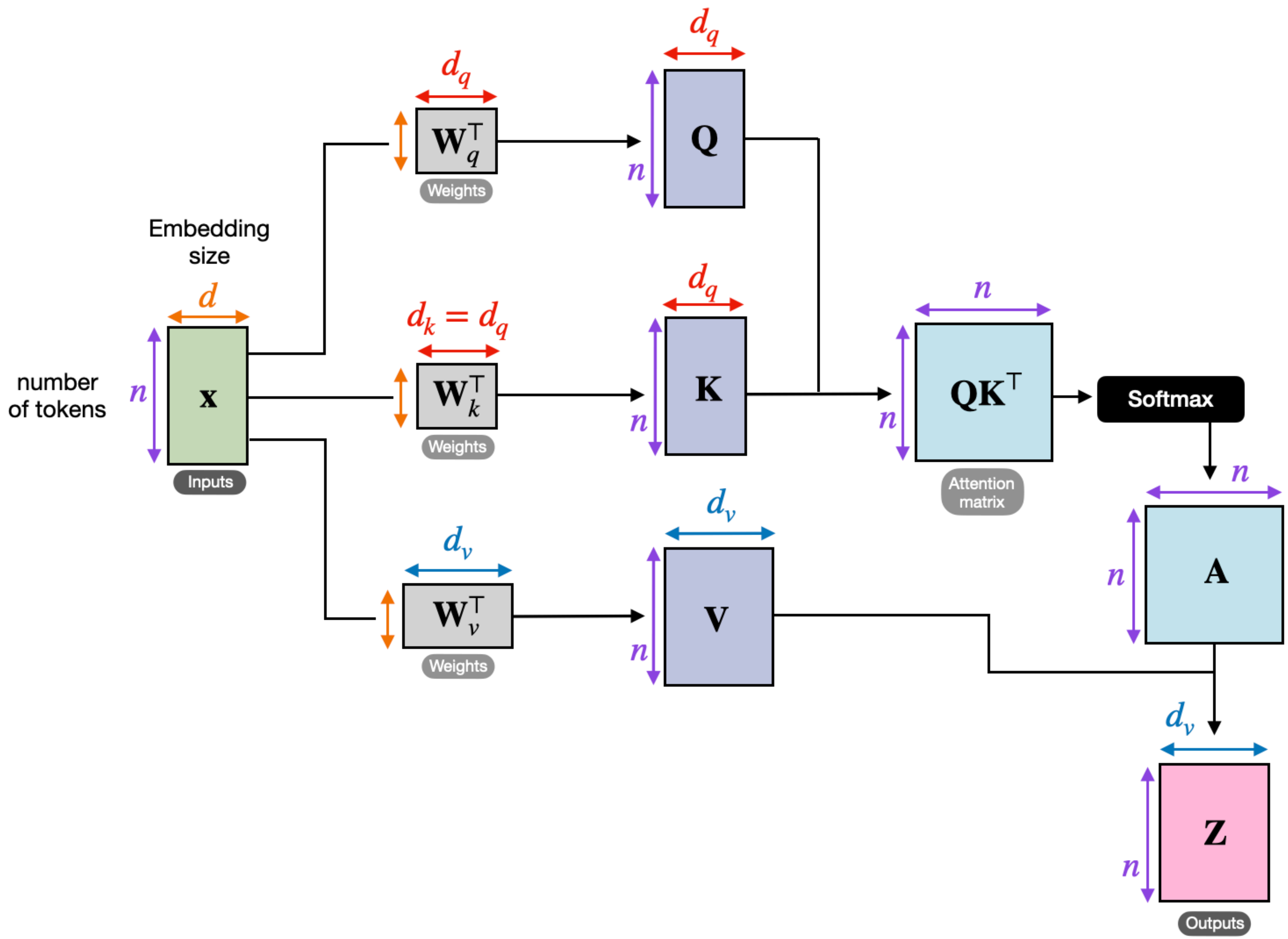
where $z^{(2)} = \sum_{j=1}^{T} \boxed{\alpha_{2,j}} v^{(j)}$

https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

Transformer Encoder



Self-attention dimensions



https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

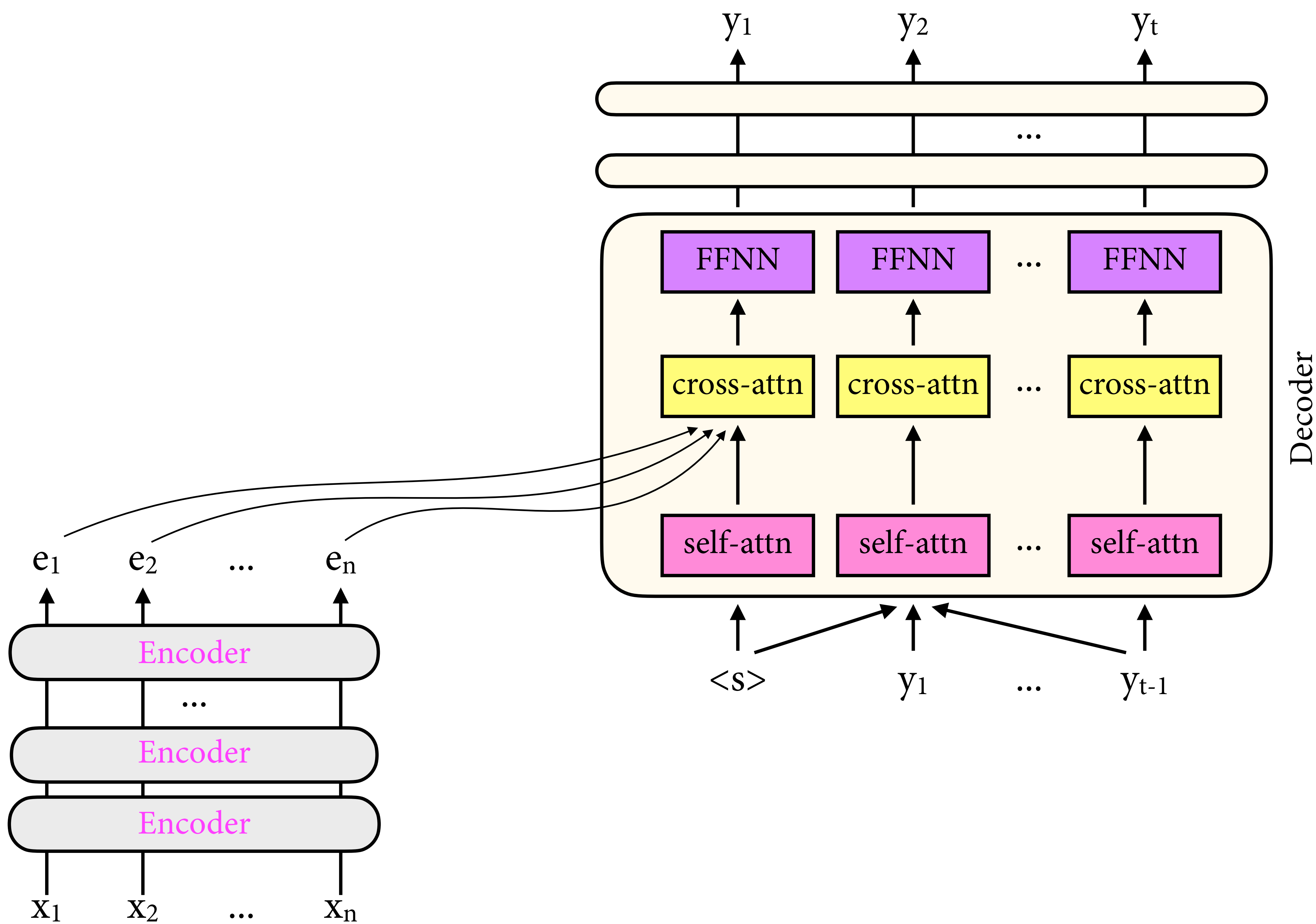
Some extra details

- Need to add *positional encodings* to each input, so the transformer can keep string positions apart.

- Typical transformers have *multiple attention heads* rather than just one. Can pay independent attention, outputs are concatenated.

- I skipped over some technical details that are important in practice; see original paper.

Transformer Decoders



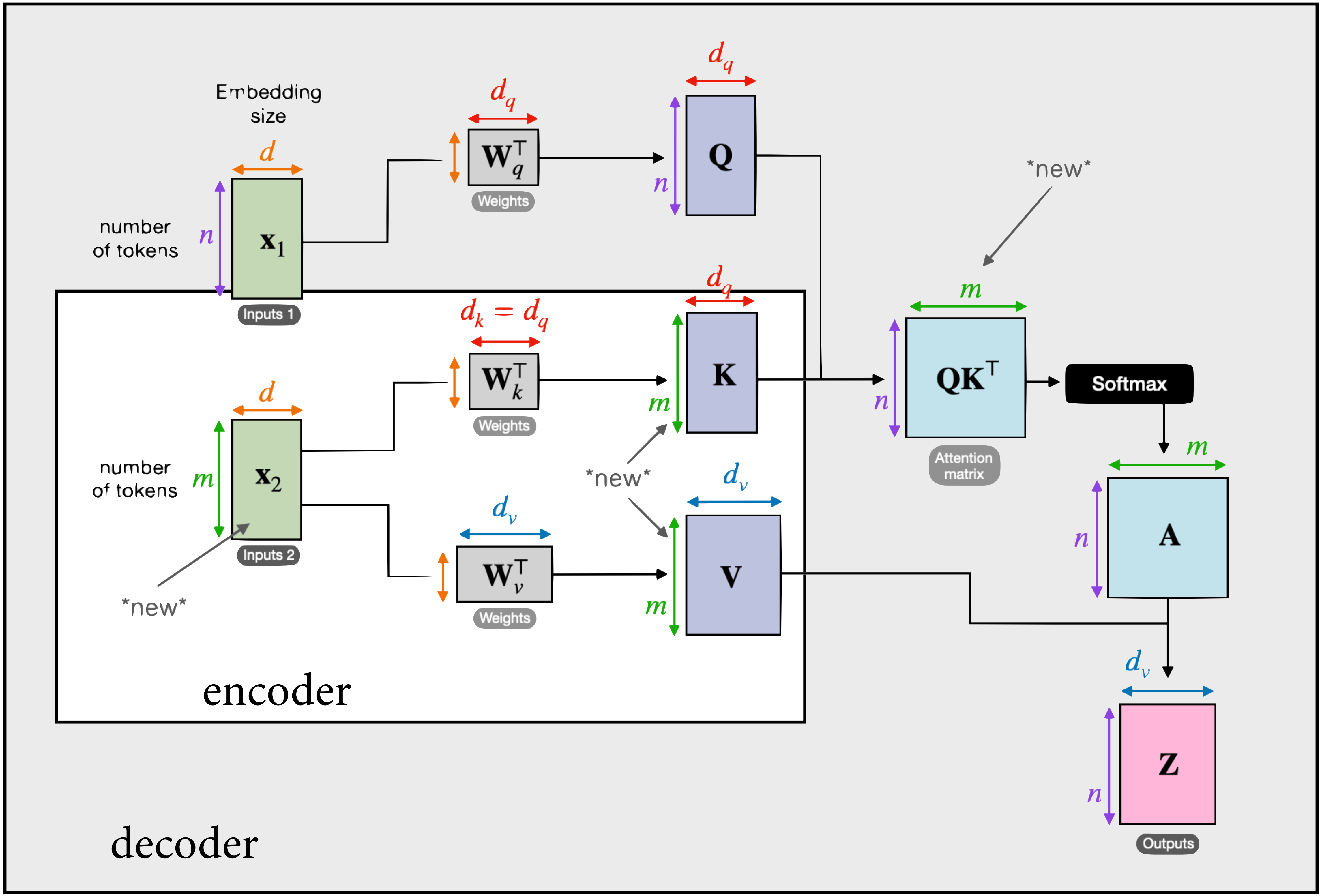
Decoder differences

- Decoder produces outputs token by token.
  Each token is appended to the decoder input.

- Decoder can have self-attention to its own inputs - but only to inputs to the left of the current token.

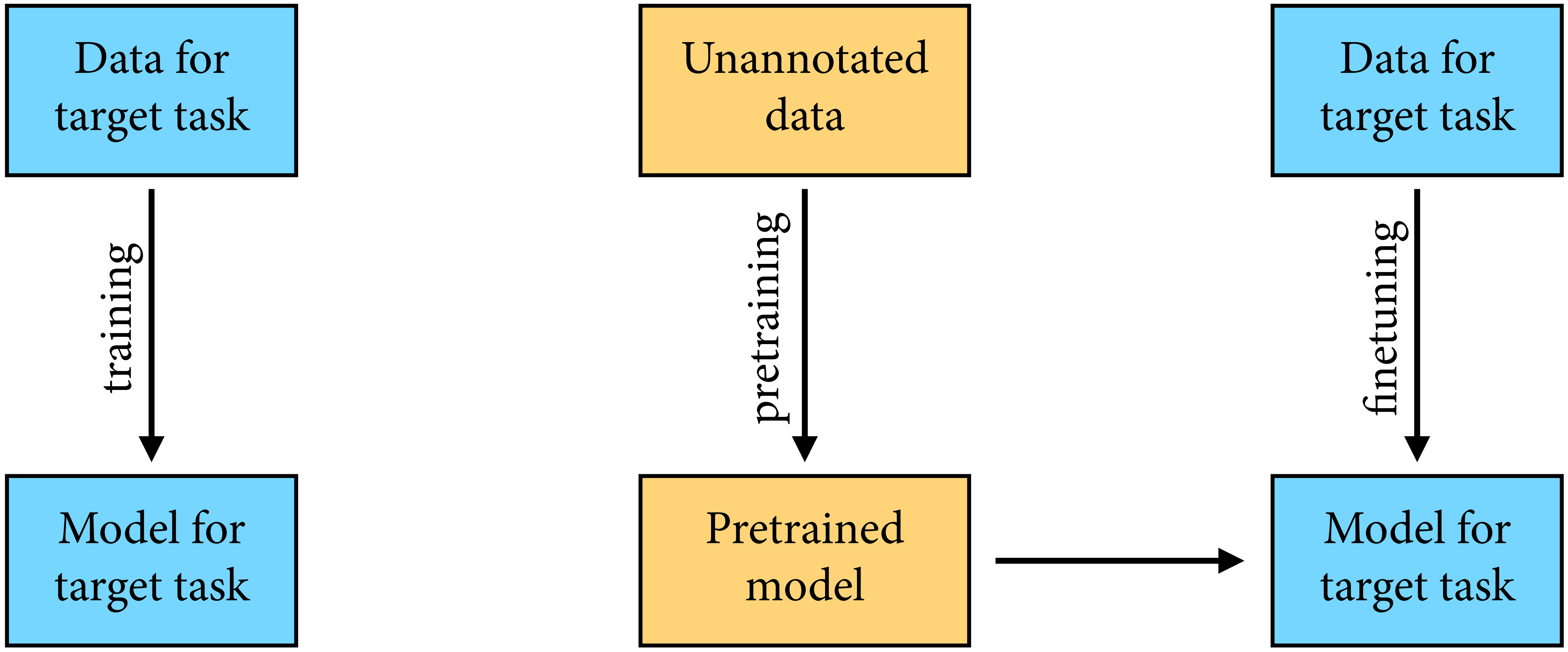
- Decoder also has cross-attention ("encoder-decoder attention") to read encoder outputs.

Cross-attention dimensions



https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

Pretraining



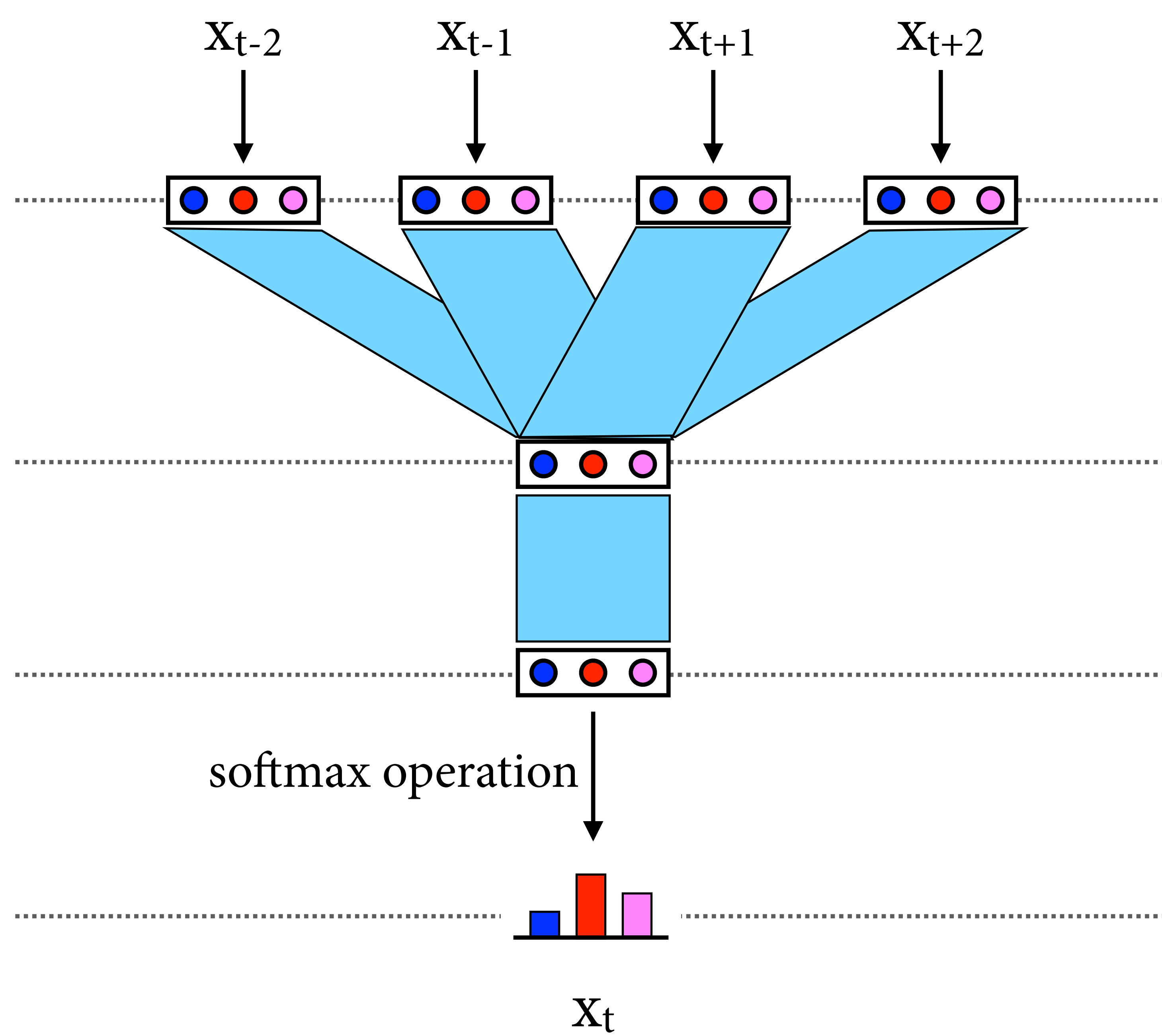
Pretrained word embeddings

word embeddings:
vector representations
for individual tokens

hidden representations:
vectors for internal use

output layer:
scores for all possible outputs

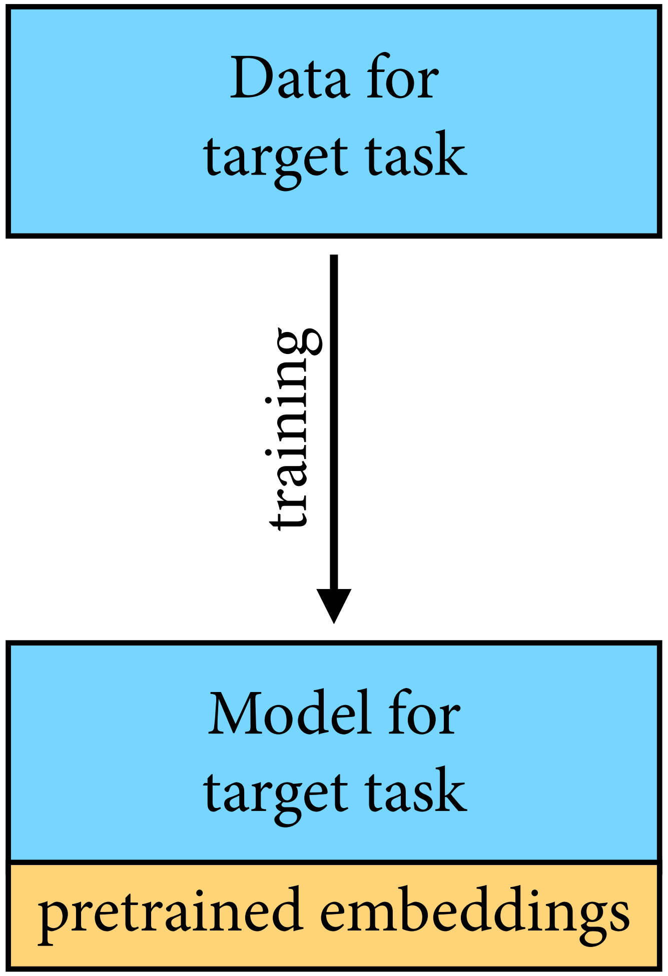
prob dist over outputs
$P(x_t \mid x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2})$



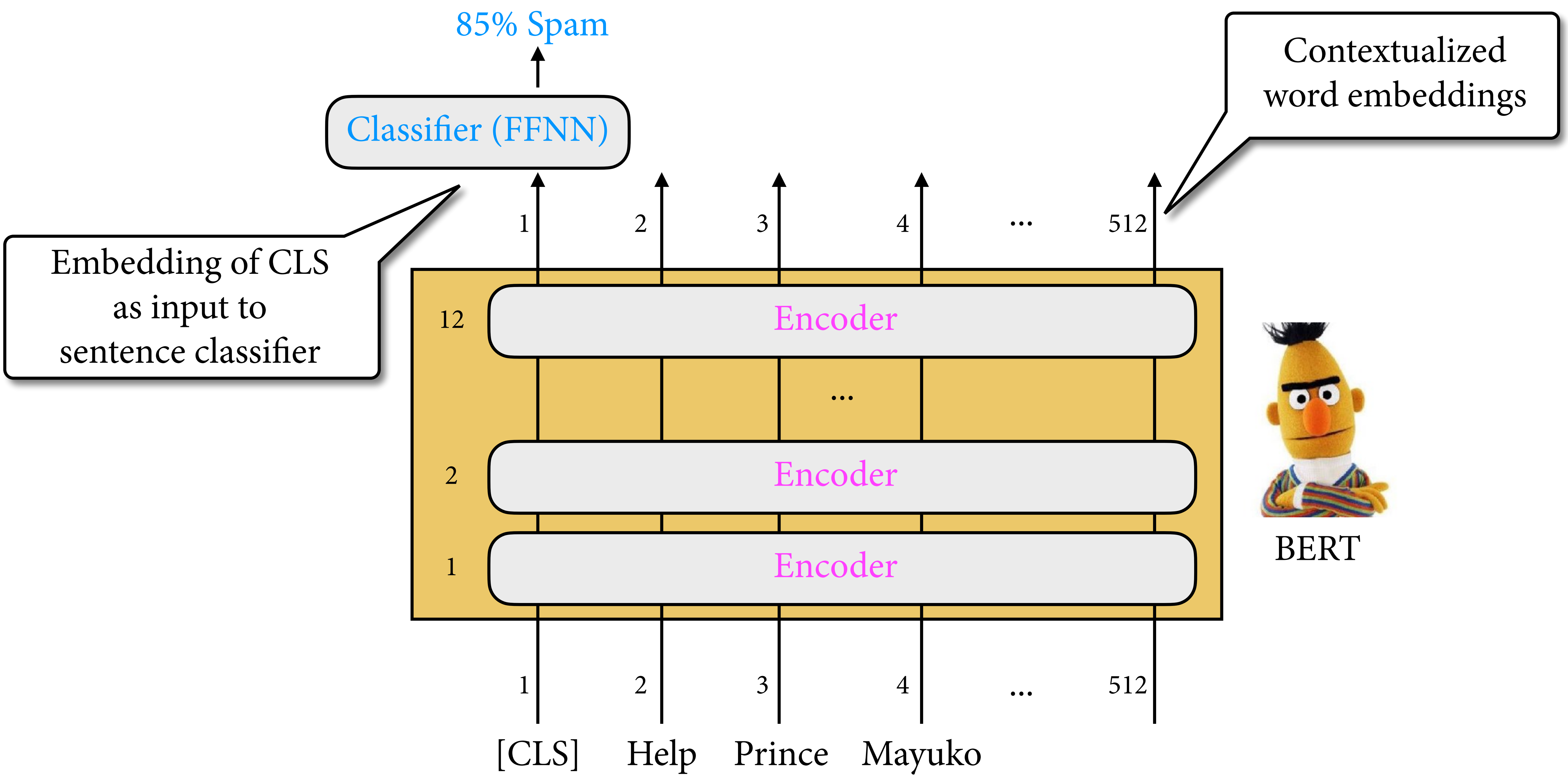
Simplified version of CBOW model (Mikolov et al. 13). Vectors at different layers have different lengths.

Pretrained word embeddings

- Classical word embeddings:
  - word2vec (2013)
  - GloVe (2014)
  - Fasttext (2016)
- All of these map words (or subword tokens) into vector representations.
- Representation depends only on word, not context.



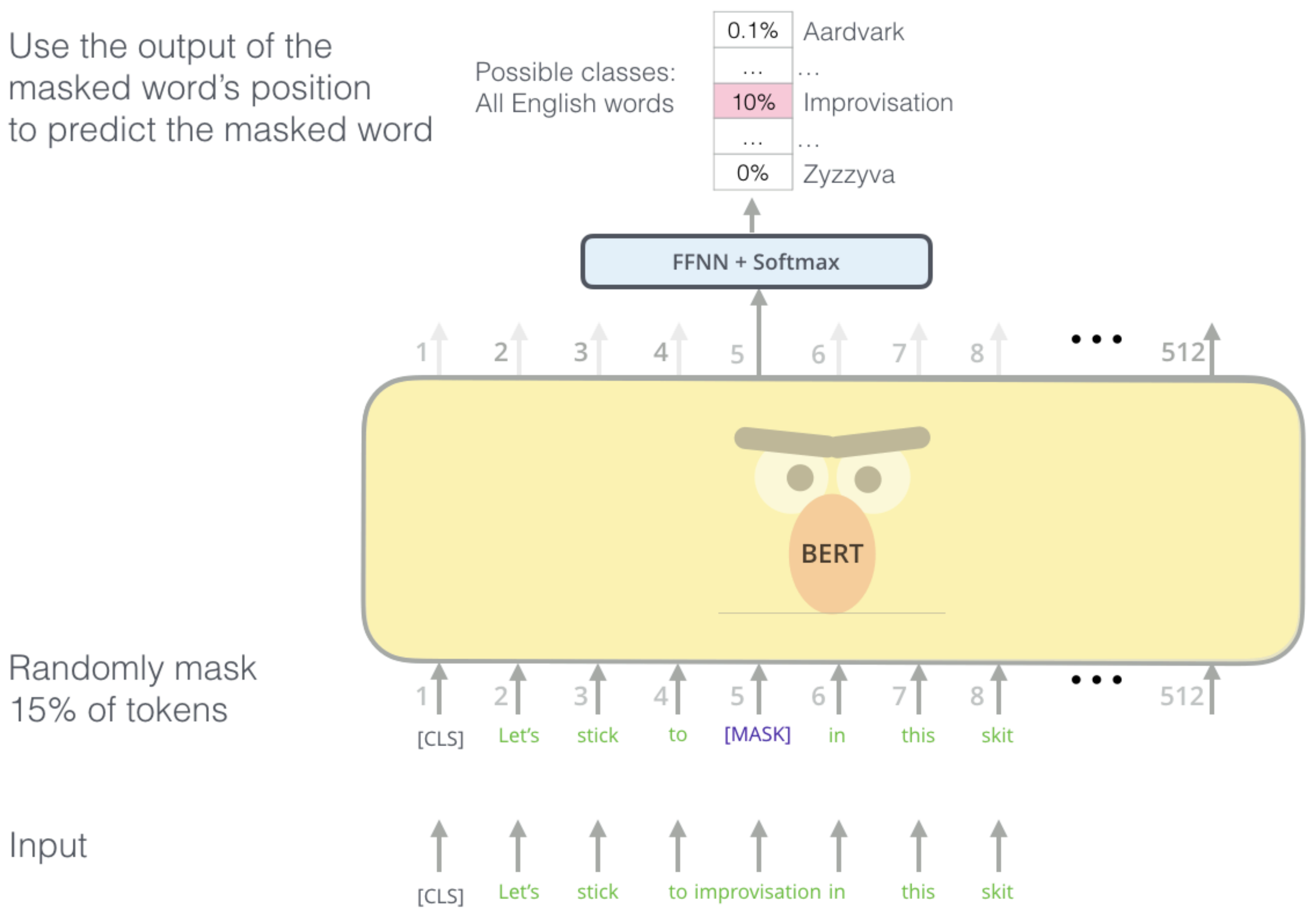
Pretrained transformers



(Devlin et al. 2019)

BERT pretraining task

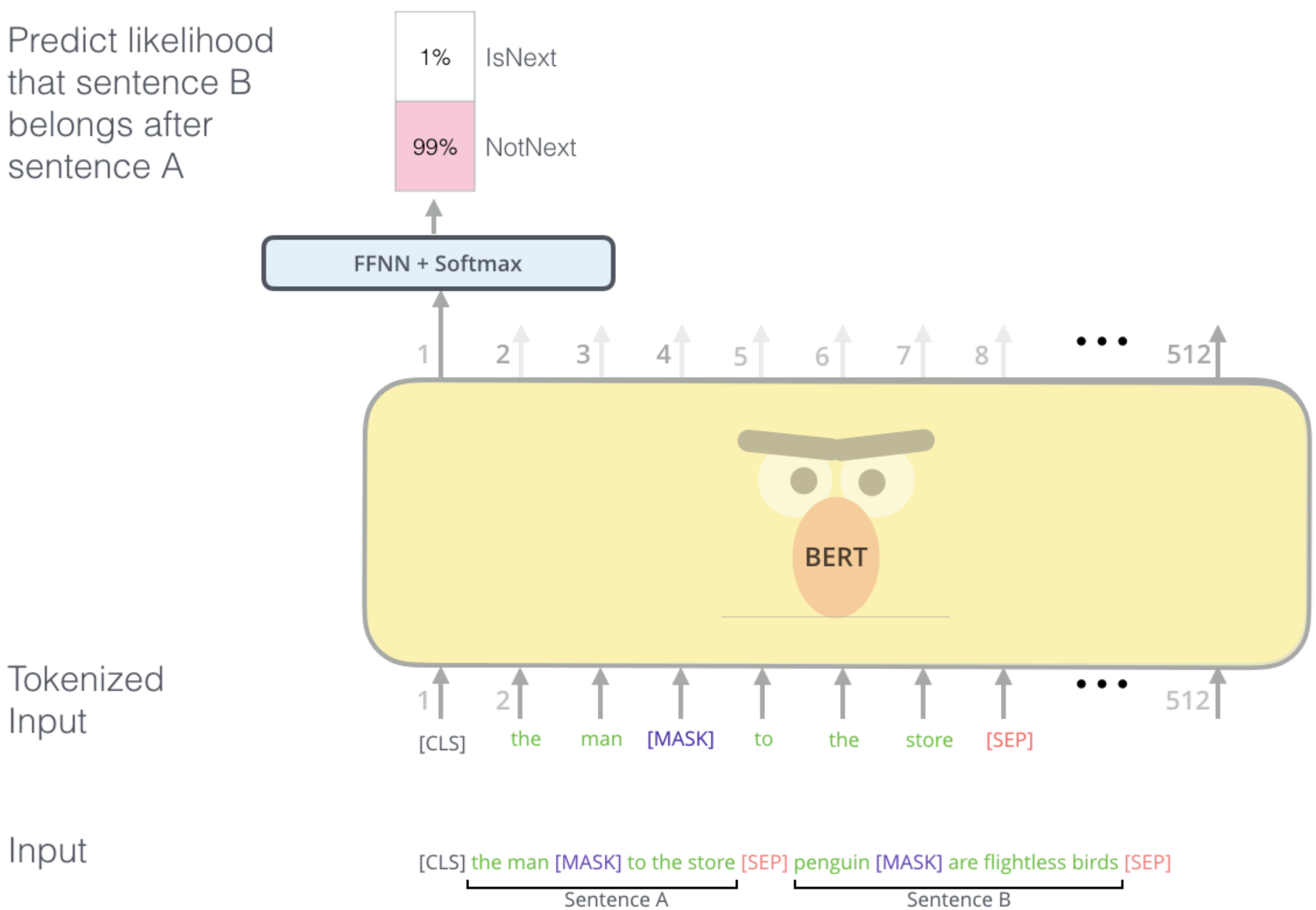
(1) Masked language modeling



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

BERT pretraining task

(2) Next-sentence prediction



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

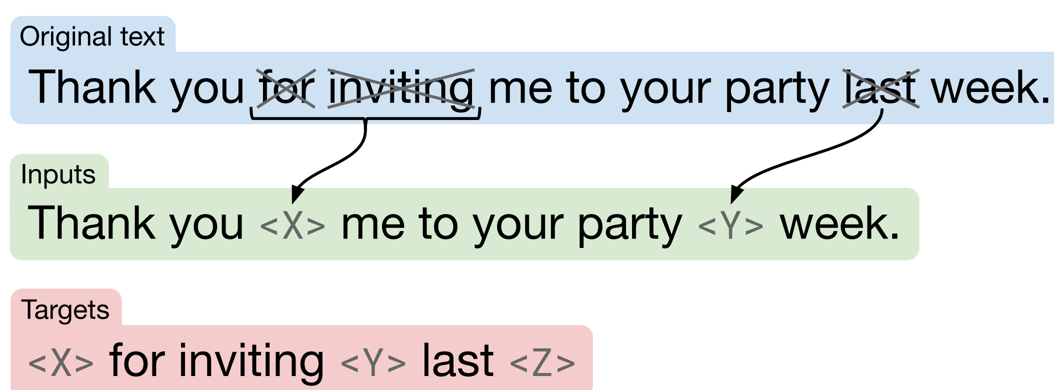
Usefulness of BERT

	DM		PAS		PSD		EDS		AMR 2015	AMR 2017
	id F	ood F	id F	ood F	id F	ood F	Smatch F	EDM	Smatch F	Smatch F
Groschwitz et al. (2018)	-	-	-	-	-	-	-	-	70.2	71.0
Lyu and Titov (2018)	-	-	-	-	-	-	-	[illegible]	73.7	74.4 ±0.16
Zhang et al. (2019)	-	-	-	-	-	-	-	-	-	**76.3** ±0.1
Peng et al. (2017) Basic	89.4	84.5	92.2	88.3	77.6	75.3	[illegible]	-	-	-
Dozat and Manning (2018)	93.7	88.9	94.0	90.8	81.0	79.4	[illegible]	-	-	-
Buys and Blunsom (2017)	-	-	-	-	-	-	85.5	85.9	60.1	-
Chen et al. (2018)	-	-	-	-	-	-	**90.9**[1,2]	**90.4**[1]	-	-
This paper (GloVe)	90.4 ±0.2	84.3 ±0.2	91.4 ±0.1	86.6 ±0.1	78.1 ±0.2	74.5 ±0.2	87.6 ±0.1	82.5 ±0.1	69.2 ±0.4	70.7 ±0.2
This paper (BERT)	**93.9** ±0.1	**90.3** ±0.1	**94.5** ±0.1	**92.5** ±0.1	**82.0** ±0.1	**81.5** ±0.3	90.1 ±0.1	84.9 ±0.1	**74.3** ±0.2	75.3 ±0.2
Peng et al. (2017) Freda1	90.0	84.9	92.3	88.3	78.1	75.8	-	-	-	-
Peng et al. (2017) Freda3	90.4	85.3	92.7	89.0	78.5	76.4	-	-	-	-
This paper, MTL (GloVe)	91.2 ±0.1	85.7 ±0.0	92.2 ±0.2	88.0 ±0.3	78.9 ±0.3	76.2 ±0.4	88.2 ±0.1	83.3 ±0.1	(70.4)[3] ±0.2	71.2 ±0.2
This paper, MTL (BERT)	**94.1** ±0.1	**90.5** ±0.1	**94.7** ±0.1	**92.8** ±0.1	**82.1** ±0.2	**81.6** ±0.1	90.4 ±0.1	85.2 ±0.1	(74.5)[3] ±0.1	75.3 ±0.1

(Lindemann et al. 2019)

Pretrained enc-dec models

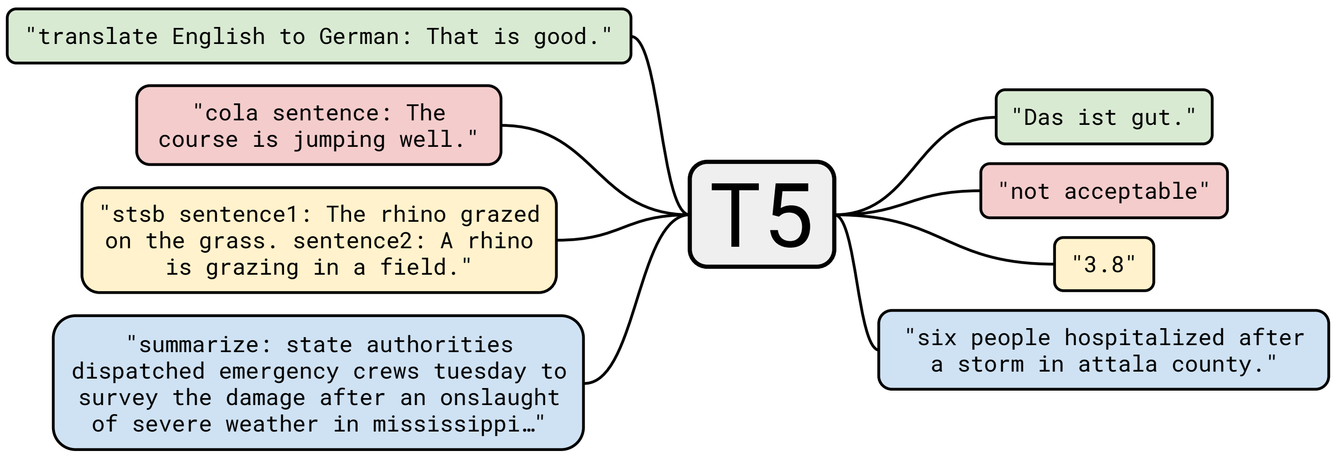
- T5: Standard encoder-decoder transformer.
- Unsupervised pretraining on C4 corpus: cleaned-up version of the Common Crawl.
- Training objective: Denoising.



(Raffel et al. 2019)

Finetuning T5

- Map wide variety of NLP tasks into seq2seq task:

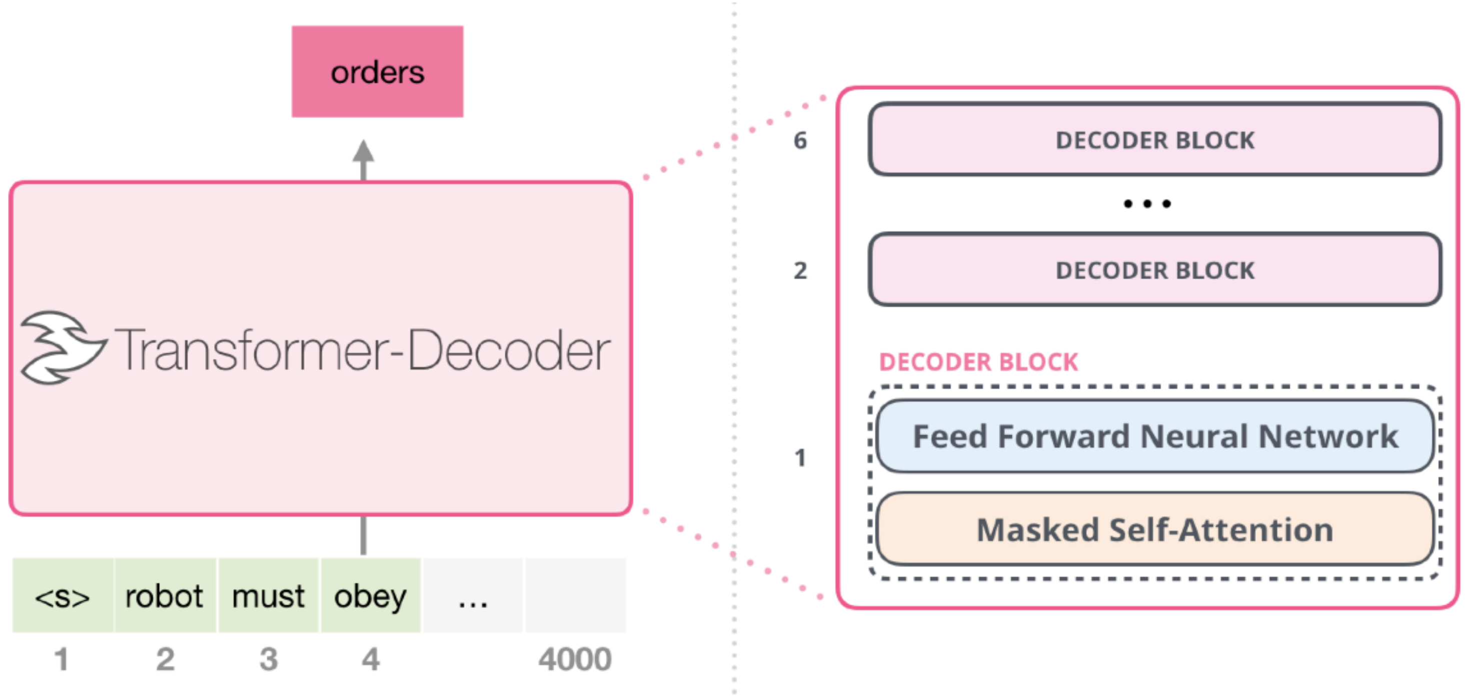


- Apply T5 to new task by finetuning all of its parameters:

Fine-tuning method	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ All parameters	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Adapter layers, $d = 32$	80.52	15.08	79.32	60.40	13.84	17.88	15.54
Adapter layers, $d = 128$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Adapter layers, $d = 512$	81.54	17.78	79.18	64.30	23.45	33.98	25.81
Adapter layers, $d = 2048$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Gradual unfreezing	82.50	18.95	79.17	**70.79**	26.71	39.02	26.93

Decoder-only models

- GPT models: Use only a stack of decoders (GPT-2 XL has 48 of them).



- No cross-attention because there is no encoder.

- *Masked* self-attention: can only attend to the left.

https://jalammar.github.io/illustrated-gpt2/

In-context learning

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese => ..........                ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese => ..........                ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => ..........                ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer          ← example #1
```

↓

gradient update

↓

```
peppermint => menthe poivrée        ← example #2
```

↓

gradient update

↓ • • • ↓

```
plush giraffe => girafe peluche     ← example #N
```

gradient update

```
cheese => ..........                ← prompt
```

(Brown et al. 2020: GPT-3 as "few-shot learner")

Tokenization

- Challenge in broad-coverage neural LMs:
  How do you keep vocabulary size under control?

- Standard solution nowadays is to use
  subword tokenizers, which break words up into reusable pieces.
  - Byte-pair encoding (BPE, Sennrich et al. 2016)
  - WordPiece (introduced for BERT, Devlin et al. 2019)
  - SentencePiece is a popular implementation
  - Huggingface models such as XML-RoBERTa come
    with their own subword tokenizers already implemented.

Byte-pair encoding

Original corpus:	hug x 10	pug x 5	pun x 12	bun x 4	hugs x 5
Split into characters:	h u g x 10	p u g x 5	p u n x 12	b u n x 4	h u g s x 5
Merge most frequent token pair:	h **ug** x 10	p **ug** x 5	p u n x 12	b u n x 4	h **ug** s x 5
(repeatedly)	h ug x 10	p ug x 5	p **un** x 12	b **un** x 4	h ug s x 5
Stop at intended vocabulary size:	**hug** x 10	p ug x 5	p un x 12	b un x 4	**hug** s x 5

example from https://huggingface.co/learn/nlp-course/chapter6/5

Summary

- Transformers: sequence models that can be trained in parallel.

- Pretraining is extremely effective method, especially for very large transformers.
  - finetuning vs. in-context learning

- Tokenization is a challenge which is often adressed with subword tokenization.