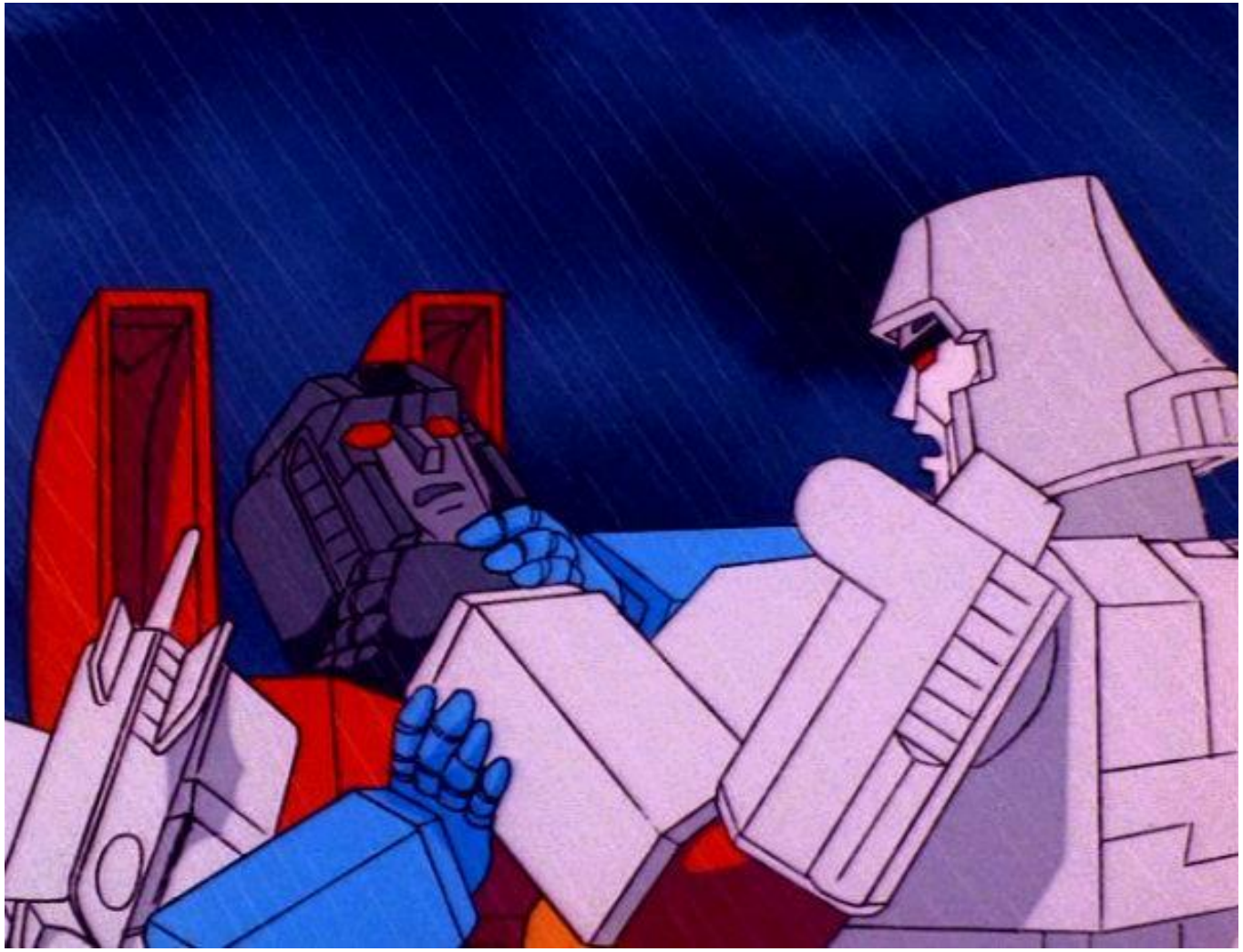
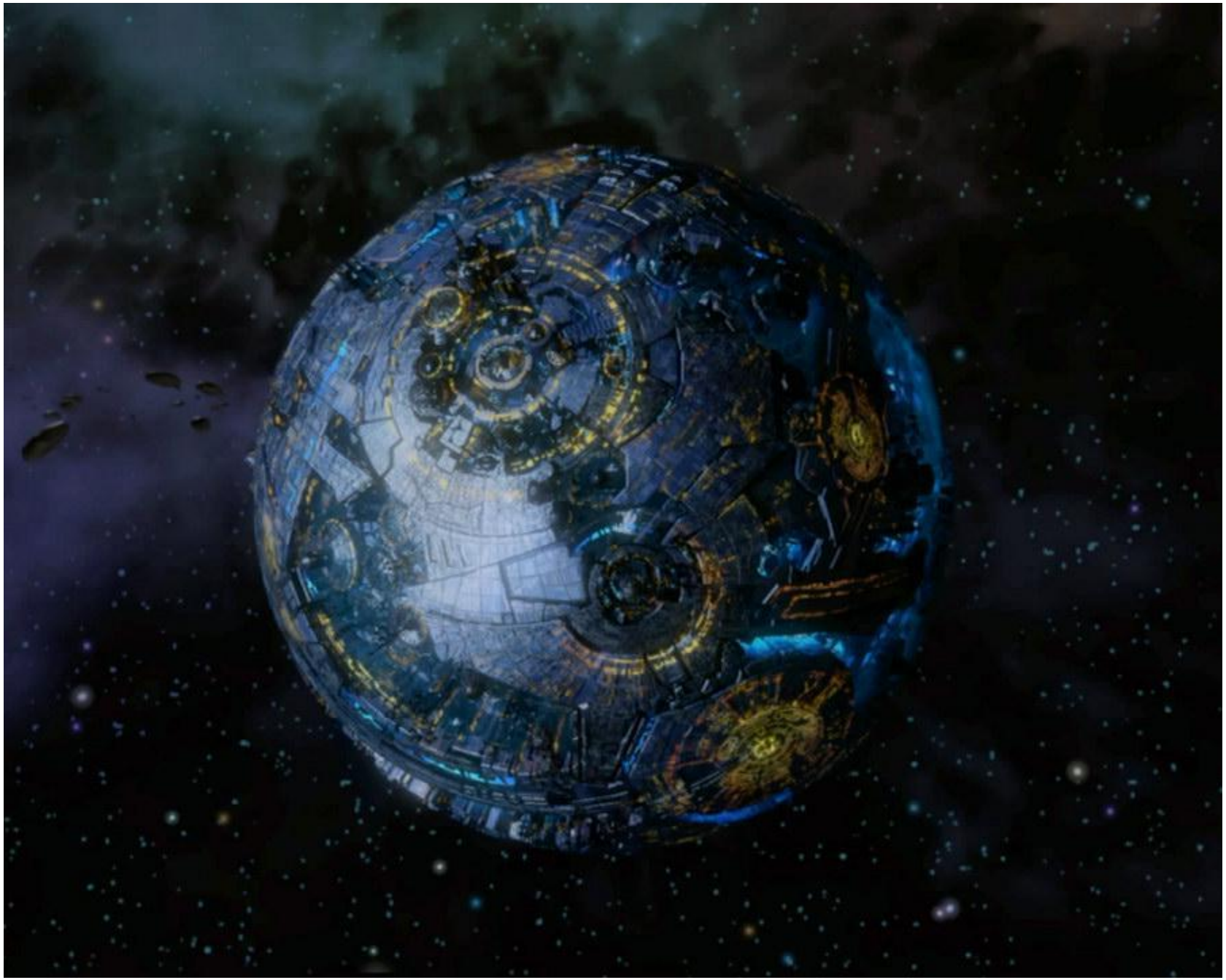
Transformers

Lecture 6





Language Modelling Task

- Final goal: predict/estimate the probability of a sequence

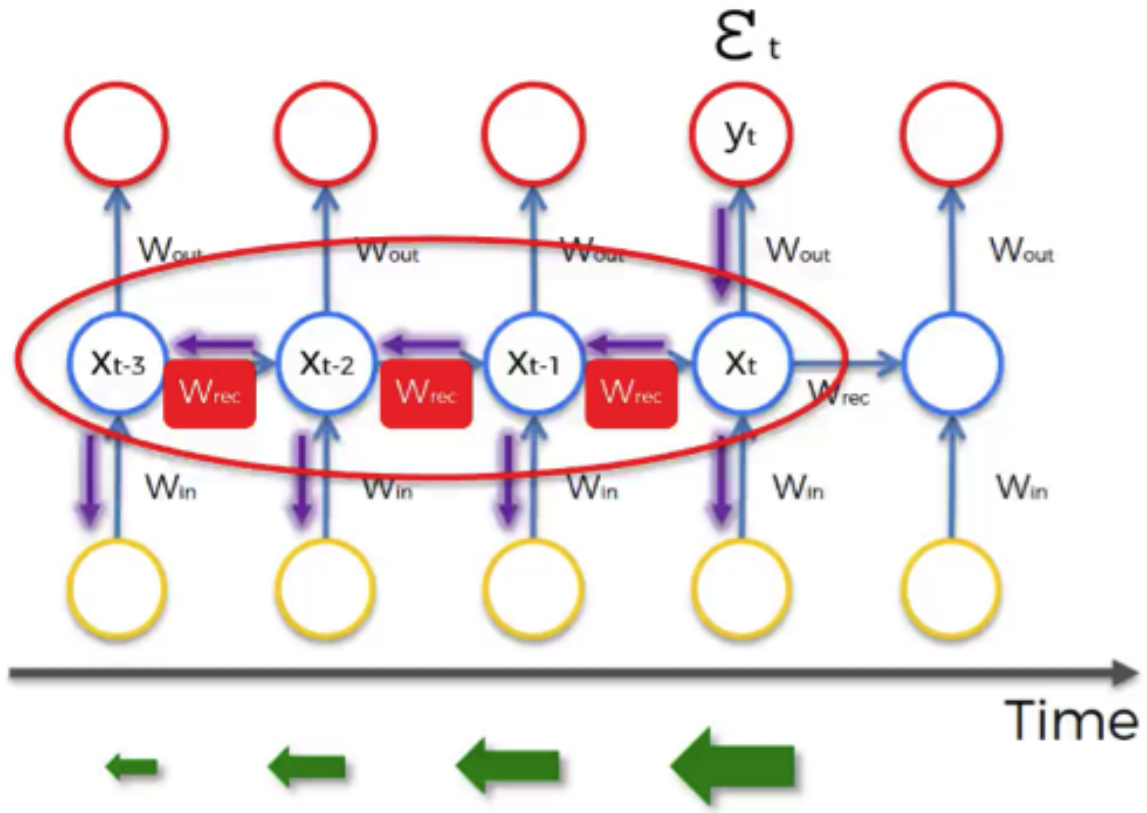
Probability(*Some sentence over here.*)

- Actual task:
  - Predict the next word
  - MLM

- In a perfect world:
  - The RNN hidden states should be able capture all contextual information
  - Right?

Reality

The Vanishing Gradient Problem



$$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \le t \le T} \frac{\partial \mathcal{E}_t}{\partial \theta} \quad (3)$$

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \le k \le t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \quad (4)$$

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \ge i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1})) \quad (5)$$

W_{rec} ~ small ⇨ Vanishing

W_{rec} ~ large ⇨ Exploding

Formula Source: Razvan Pascanu et al. (2013)

Again, we need more! What of larger semantic units?

- How can we know when larger units are similar in meaning?
  - *CTV News*: Poilievre-led attempt to bring down Trudeau minority over carbon tax fails.
  - *CBC News*: Liberals survive non-confidence vote on carbon tax with Bloc, NDP backing.
  - The Beaverton: Co-worker that everyone hates surprised he can't get colleagues to do what he wants.

NATIONAL - 2 WEEKS AGO

Co-worker that everyone hates surprised he can't get colleagues to do what he wants

OTTAWA – Local man Pierre Poilievre, an employee at an Ottawa small business named the House of Commons, was surprised that none of the colleagues who despise him were willing to support hi...

SHARE

RNN & next word prediction: Not good compositional representation

- Next word prediction:

$$P(t_i|t_1, t_2, \ldots t_{i-1})$$

- The hidden state i is encoding information of everything from the beginning (index 0) to the very end (index i).
- We want some bigger semantic units
  - Poilievre-led attempt to **bring down Trudeau minority over carbon tax** fails.
- Some hacks may work, but not really

Again, we need more! What of larger semantic units?

- How can we know when larger units are similar in meaning?
  - *CTV News*: Poilievre-led attempt to bring down Trudeau minority over carbon tax fails.
  - *CBC News*: Liberals survive non-confidence vote on carbon tax with Bloc, NDP backing.
  - The Beaverton: Co-worker that everyone hates surprised he can't get colleagues to do what he wants.

People interpret the meaning of larger text units – entities, descriptive terms, facts, arguments, stories – by **semantic composition** of smaller elements.

Beyond Word2Vec and RNN-LMs

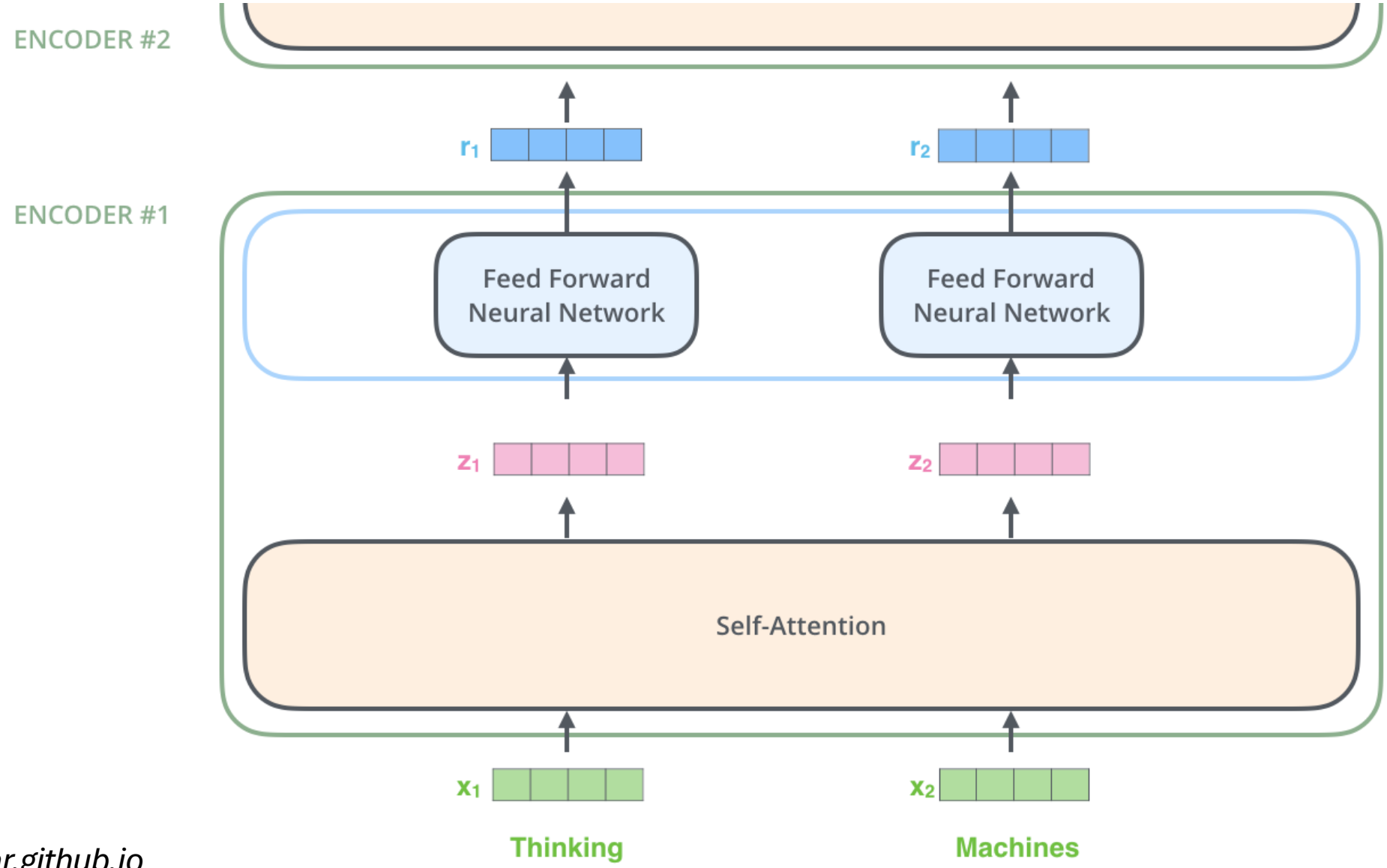
- Mitigation #1:
  - Long short-term memory (LSTM).
  - Won’t cover in this class, but covered DL4NLP.

- Mitigation #2:
  - Attention Mechanism -> Transformer

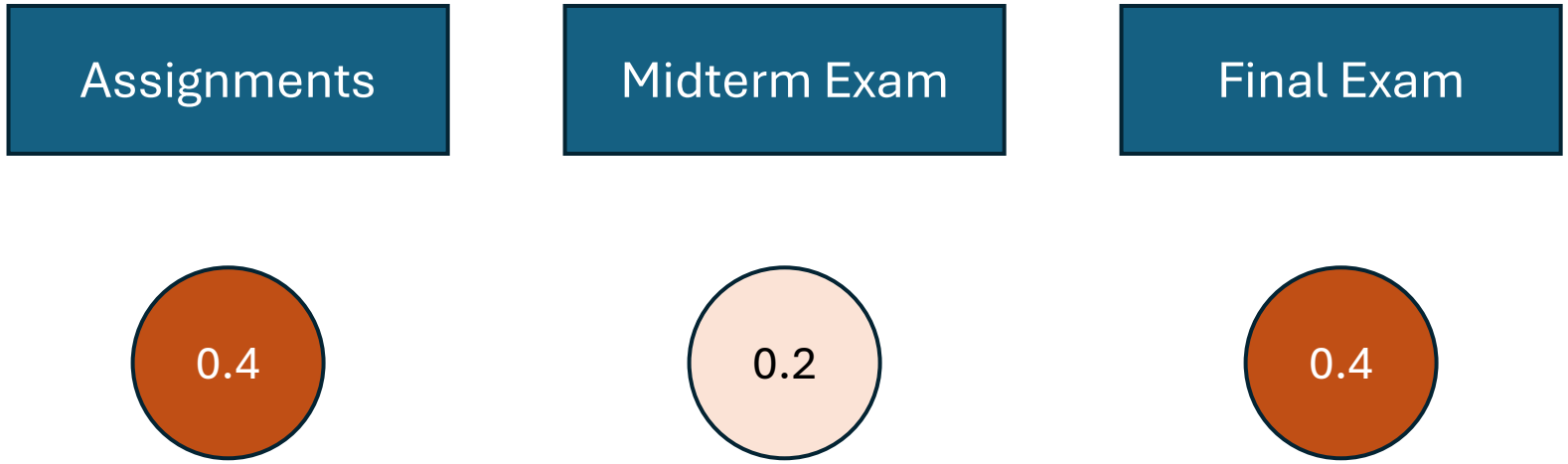
LSTM

- (Won’t appear on your exam)
- Basic idea:
  - Reset the hidden state once a while.
  - When to know how to reset? Train a separate model (actually, a part of the model) to do it.

Transformers

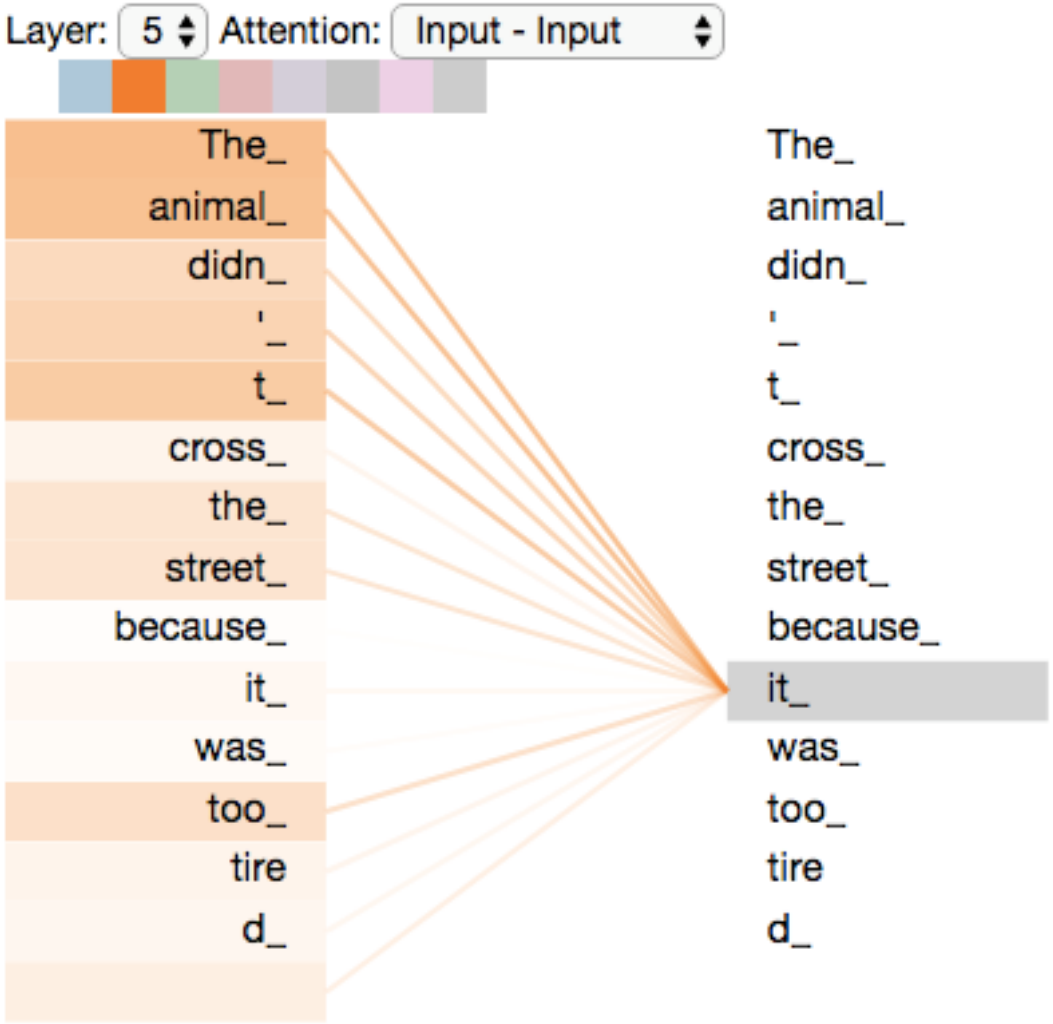


Attention Mechanism

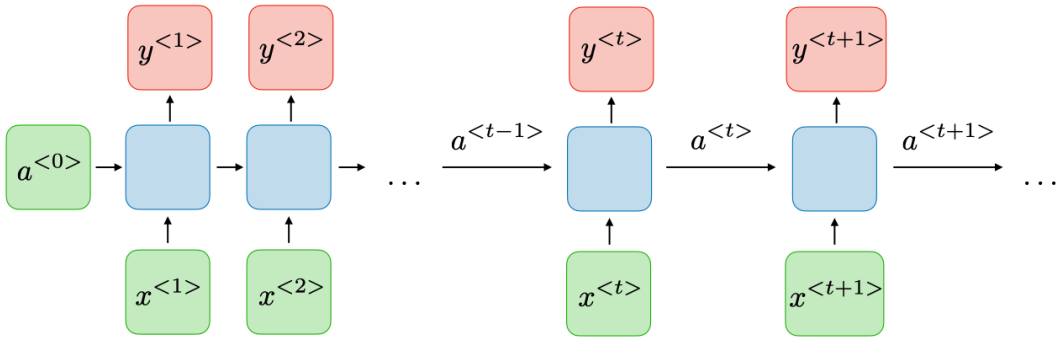


$$\text{Total} = 0.75 * A + 0.1 * Q + 0.15 * E$$

Attention Mechanism

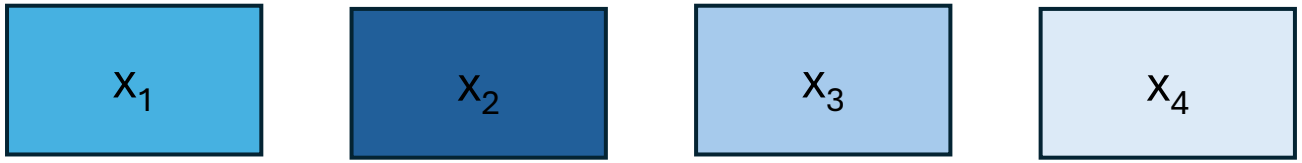


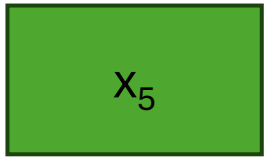
RNN vs. Attention



Recurrent neural network.
Almost “recursive.”

$$h_i = f(h_{i-1} + x_i)$$





$= 0.3\, x_1 + 0.6\, x_2 + 0.07\, x_3 + 0.03\, x_4$

$$x_i = \sum_j a_j x_j$$

????

$$h_i = f(x_i)$$

Recall GloVe

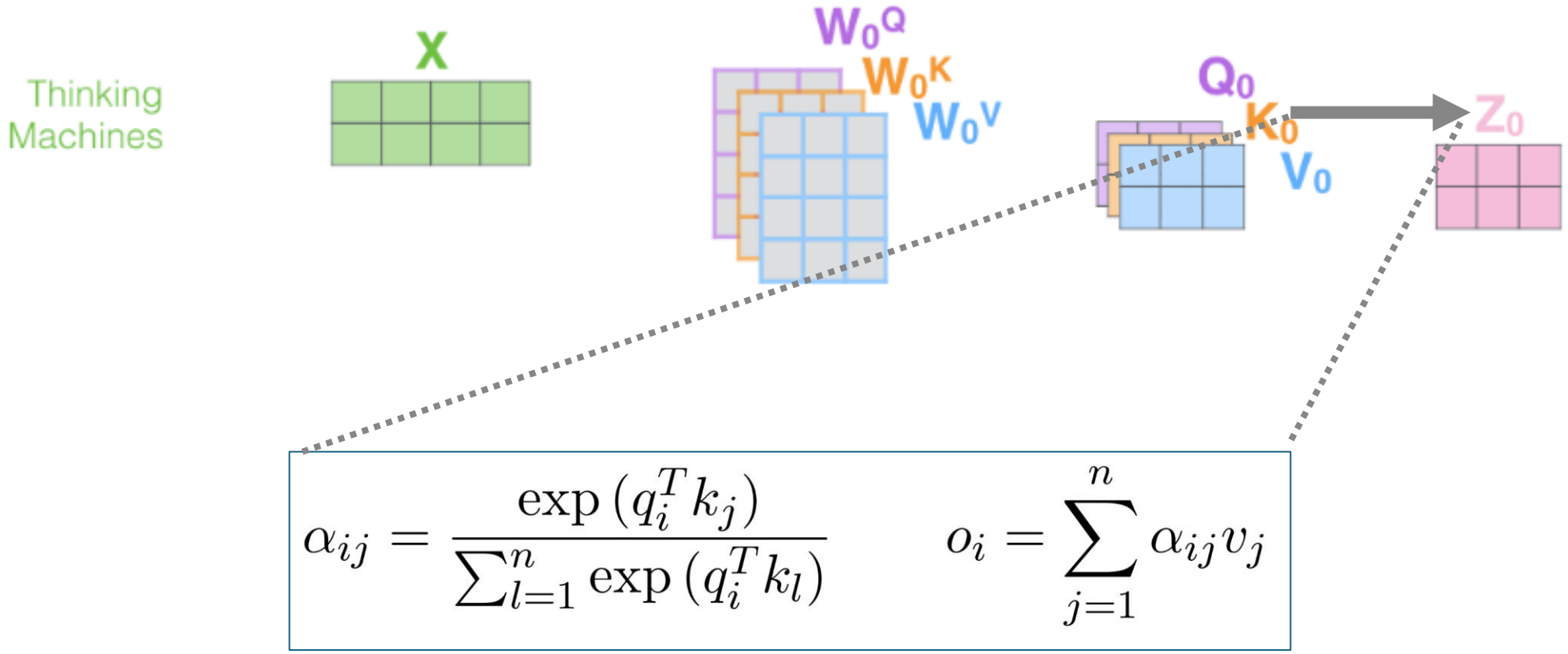
Encoding meaning in vector

- How can we capture ratios of co-occurrence proba components in a word vector space?
- Solution:
  - Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$
  - with vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

Pennington et al. (2014)

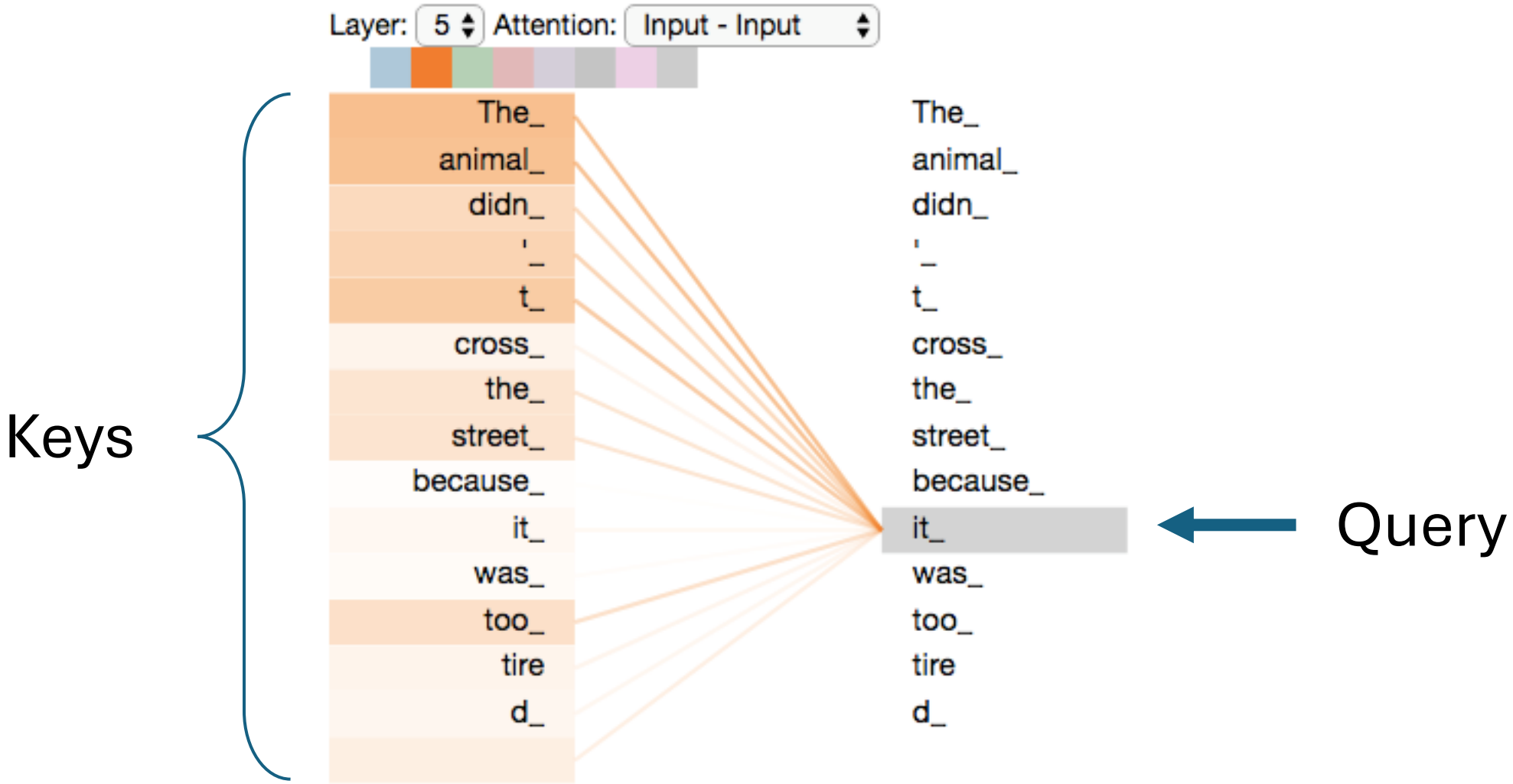
Use the same idea to compute attention (token importance)!

Self Attention

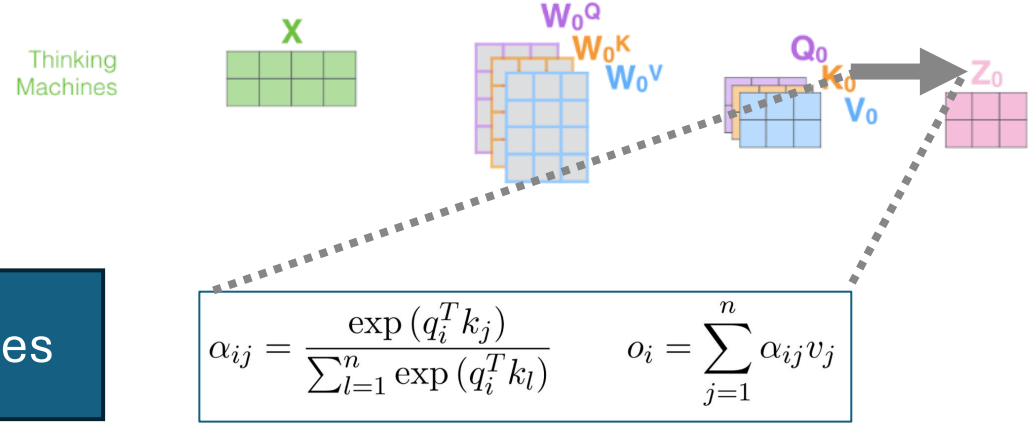


$$\alpha_{ij} = \frac{\exp\left(q_i^T k_j\right)}{\sum_{l=1}^{n} \exp\left(q_i^T k_l\right)} \qquad o_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

Attention Mechanism



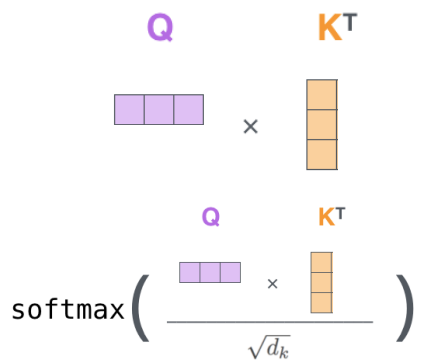
Self Attention



I	like	eating	apples

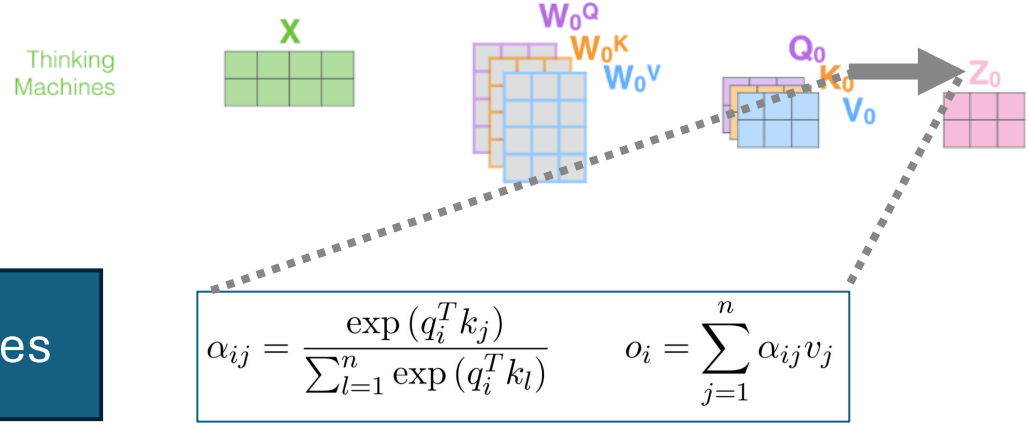
What is o(like)? I.e., the attention score from like to the sentence.

query		Q2 “like”		
key	K1 “I”	K2 “like”	K3 “eating”	K4 “apples”
attention weights*	14.2	18.1	10.3	7.9
attention scores* (o)	0.3	0.6	0.07	0.03



*: made-up numbers, not real.

Self Attention



I | like | eating | apples

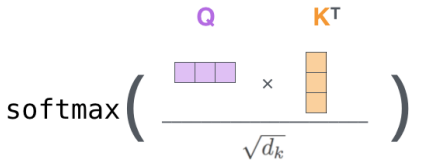
What is z(like)?

attention scores* (o): 0.3 | 0.6 | 0.07 | 0.03

values: V1 "I" | V2 "like" | V3 "eating" | V4 "apples"

Weighted sum

Z2 "like"



*: made-up numbers, not real.

Multi-Head Self Attention

1) This is our input sentence*

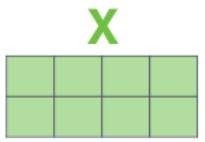
2) We embed each word*

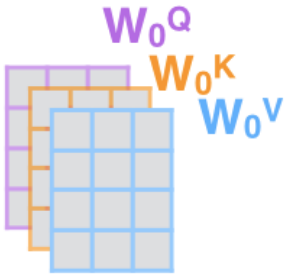
3) Split into 8 heads. We multiply X or R with weight matrices

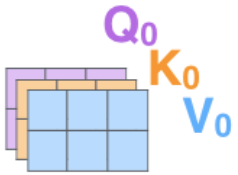
4) Calculate attention using the resulting Q/K/V matrices

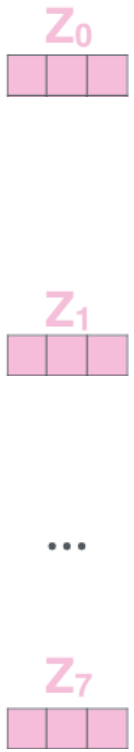
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

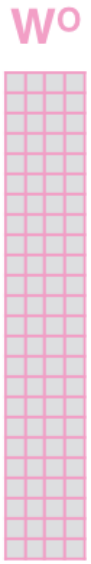
Thinking Machines

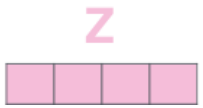




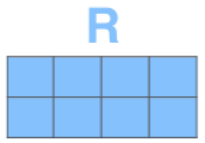




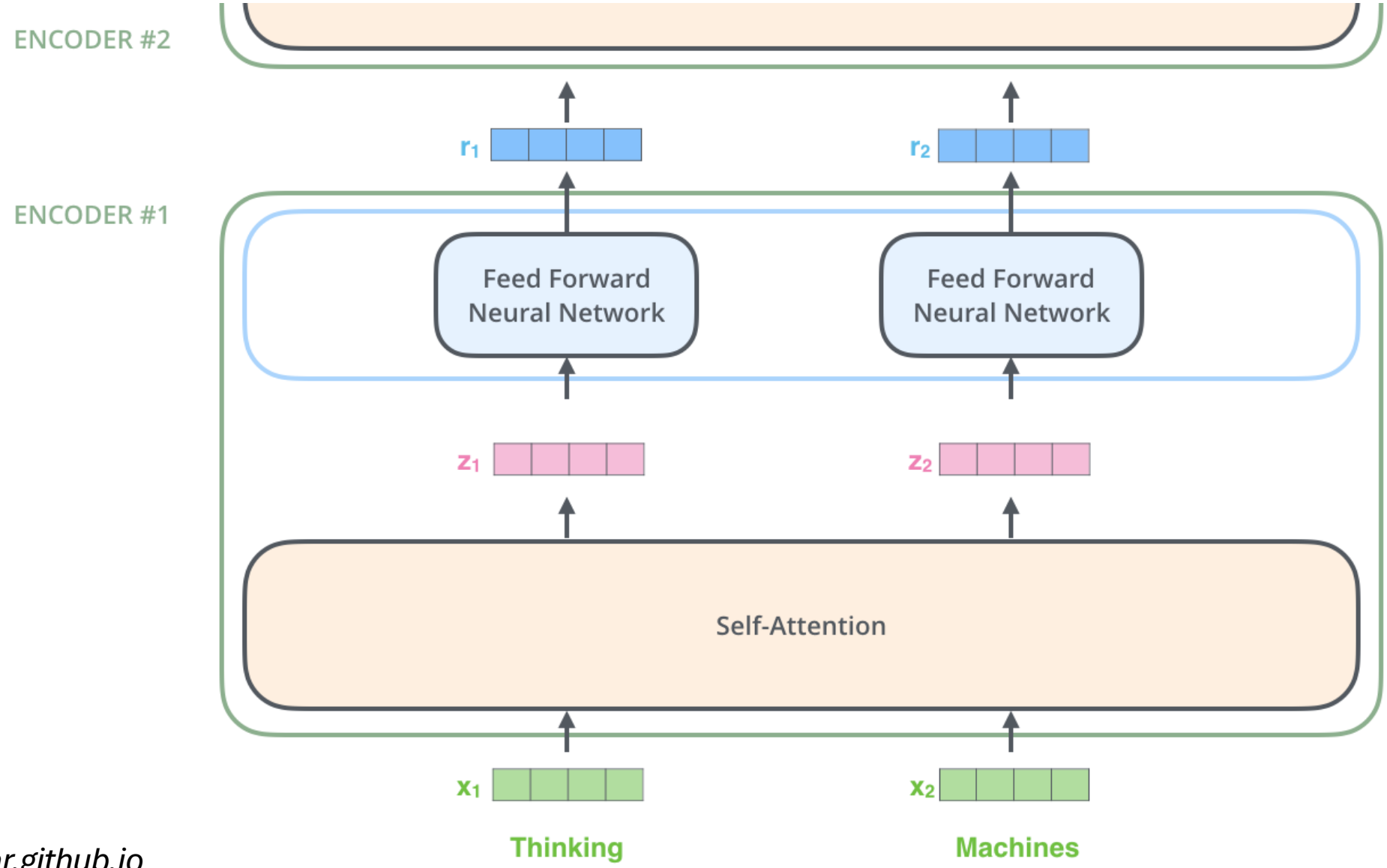




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



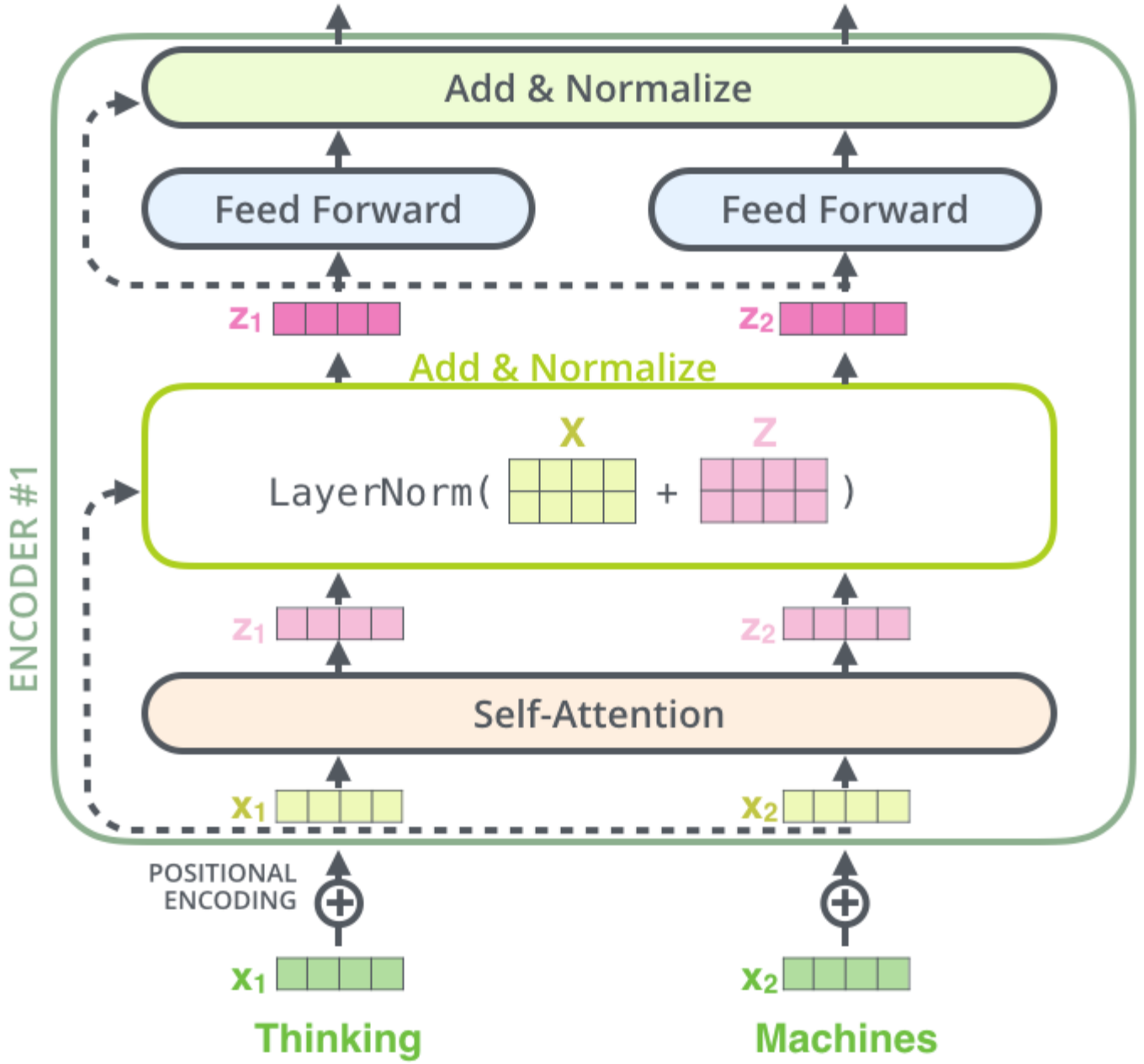
Transformers



Typically
2-layer MLP

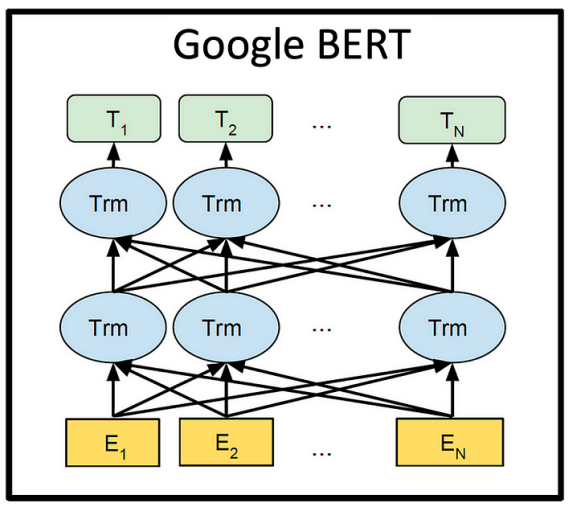
mlp_out + residual_mid

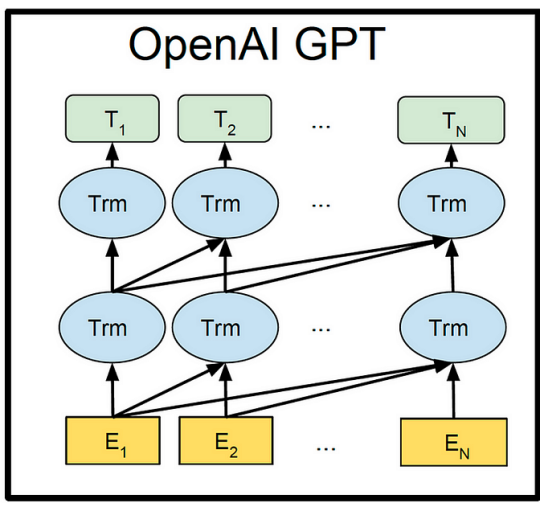
residual_mid

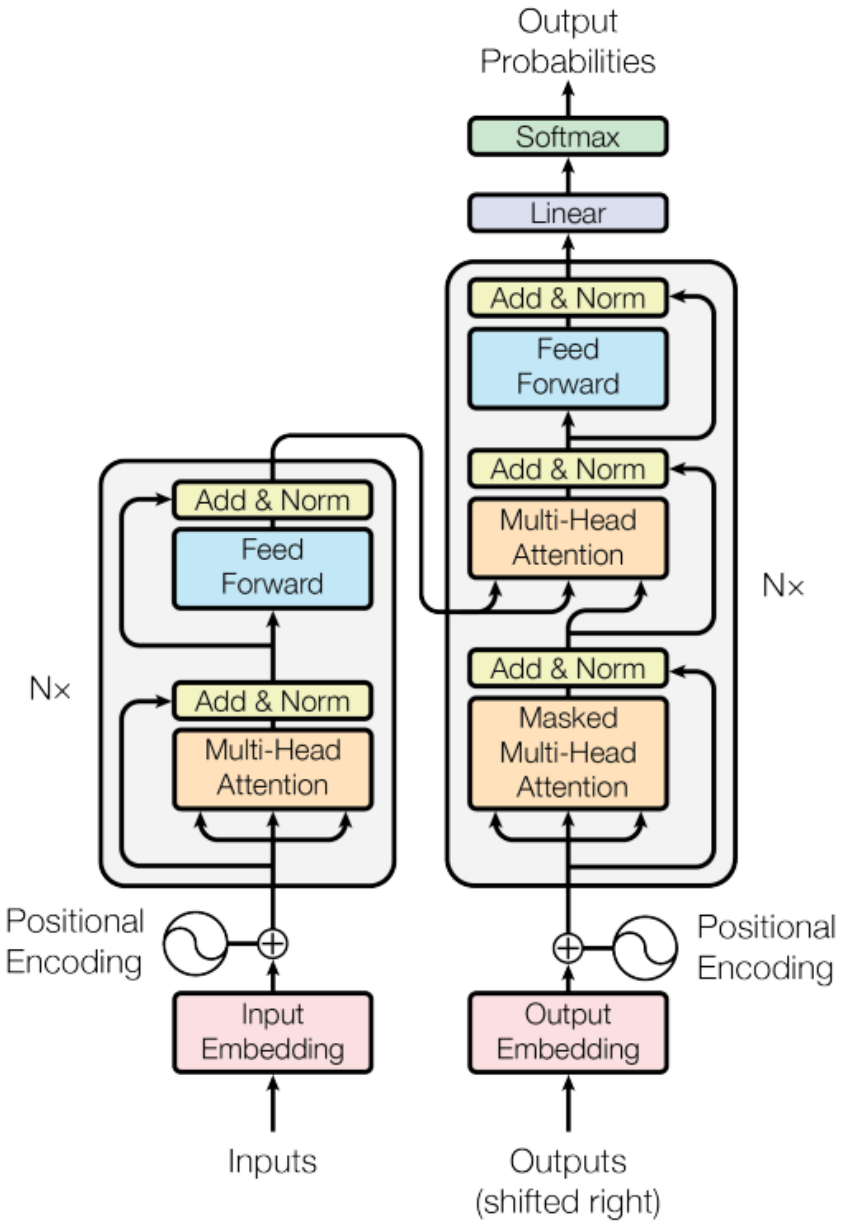


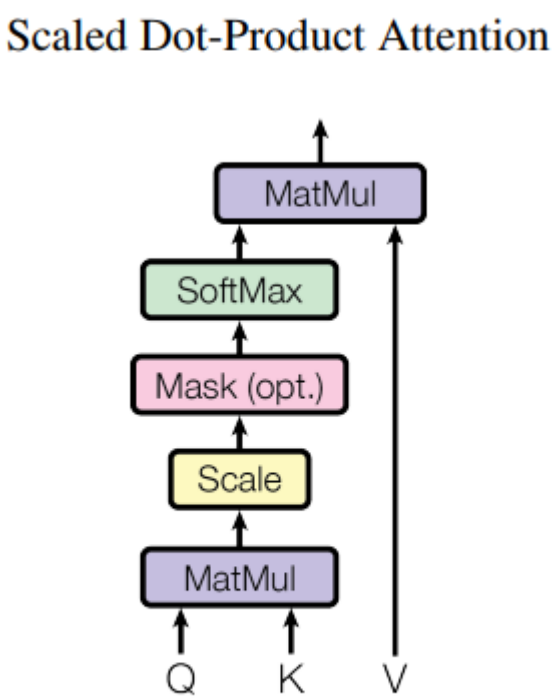
Residuals

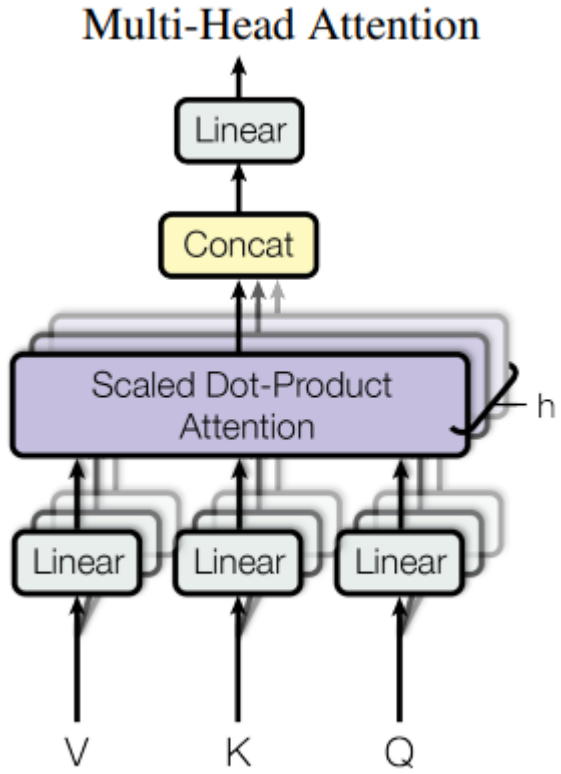

“Ogres are like onions ....
they have layers”
- Shrek


Google BERT
T1
T2
...
TN
Trm
Trm
...
Trm
Trm
Trm
...
Trm
E1
E2
...
EN


OpenAI GPT
T1
T2
...
TN
Trm
Trm
...
Trm
Trm
Trm
...
Trm
E1
E2
...
EN

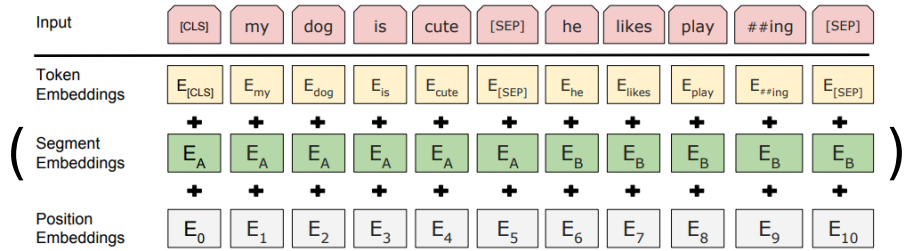






Vaswani et al. (2017). Attention Is All You Need.

Position Encoding



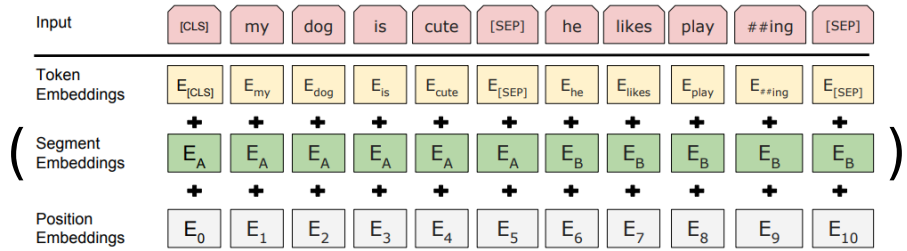
$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

where

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

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

Position Encoding



- Encodings of any two distinct positions are distinct
- Each position maps to only one encoding
- Test sentences may be longer than training
- For any fixed offset *k*, PE(*t*+*k*) can be represented as a linear function of PE(*t*).

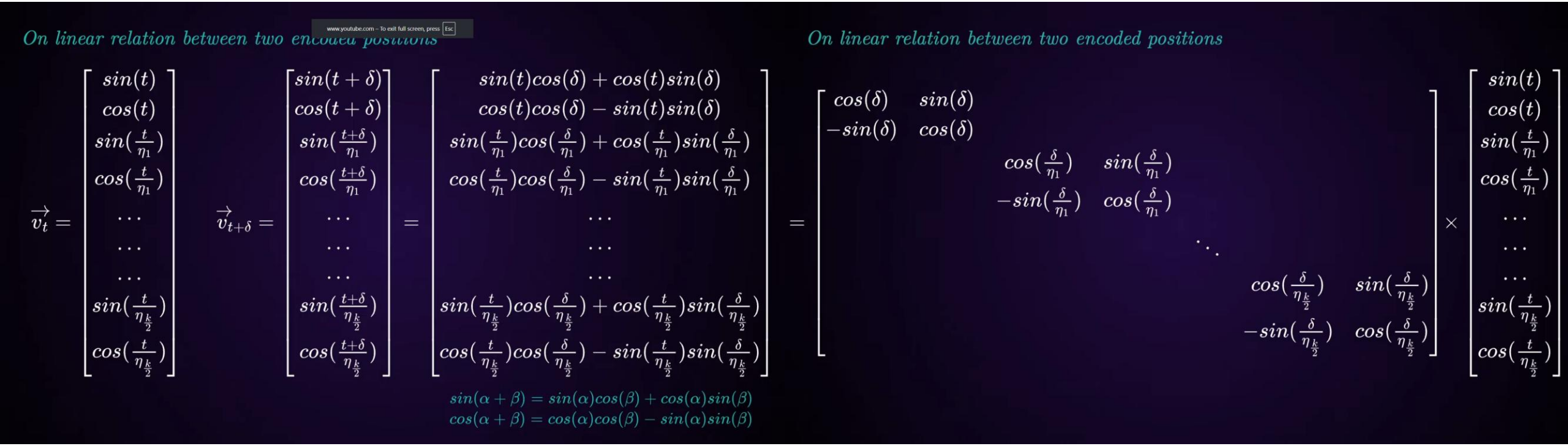
$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

where

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

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

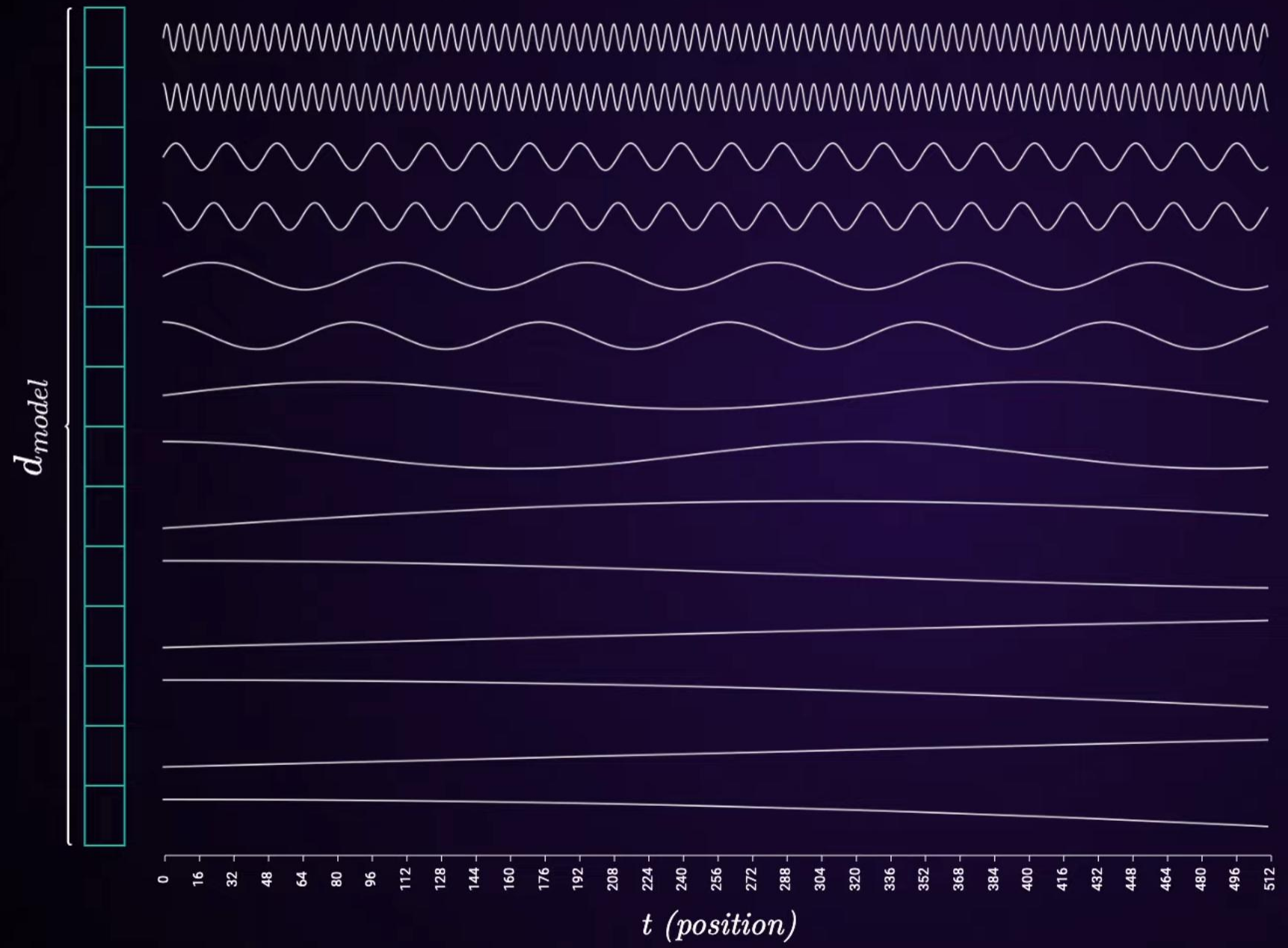
Sinusoidal: Linear Relative Offset



- For any fixed offset *k*, PE(*t*+*k*) can be represented as a linear function of PE(*t*).

https://www.youtube.com/watch?v=T3OT8kqoqjc

https://www.youtube.com/watch?v=T3OT8kqoqjc



k *is even:* $\sin\left(\frac{t}{N^{\frac{k}{d_{model}}}}\right)$

k *is odd:* $\cos\left(\frac{t}{N^{\frac{k}{d_{model}}}}\right)$

$L\ (context/sequence\ length) = 512$

$t\ (position) = \{1, \ldots, L\}$

$N = 10000$

$d_{model} = 14$

$k = \{1, 2, \ldots, d_{model}\}$

Not Really that Useful…

```python
class BertEmbeddings(nn.Module):
    """Construct the embeddings from word, position and token_type embeddings."""

    def __init__(self, config):
        super().__init__()
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_size, padding_idx=config.
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, config.hidden_size)
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config.hidden_size)

        self.LayerNorm = nn.LayerNorm(config.hidden_size, eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        # position_ids (1, len position emb) is contiguous in memory and exported when serialized
        self.register_buffer(
            "position_ids", torch.arange(config.max_position_embeddings).expand((1, -1)), persistent=F
        )
        self.register_buffer(
            "token_type_ids", torch.zeros(self.position_ids.size(), dtype=torch.long), persistent=Fals
        )
```

BERT

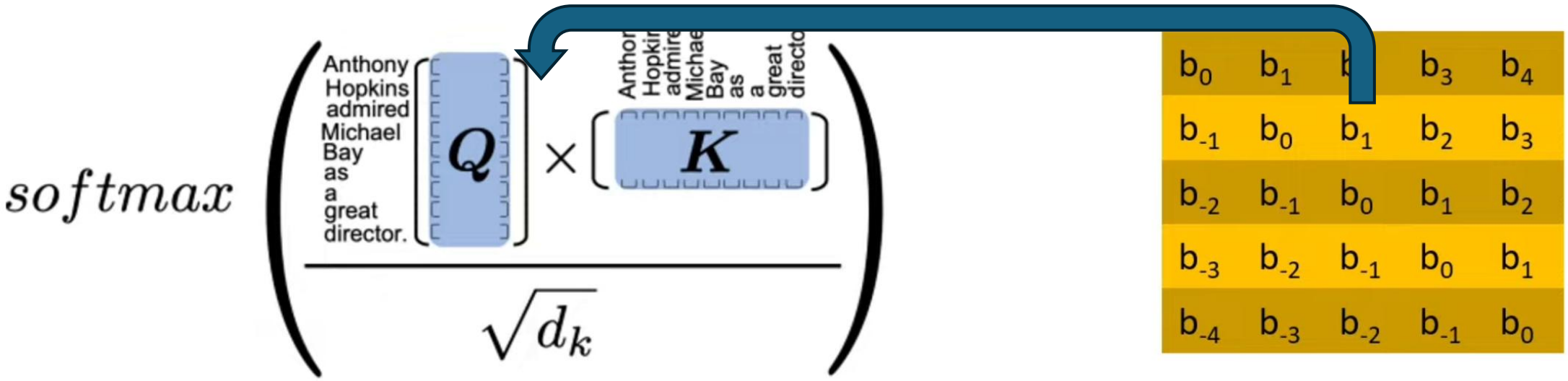
```python
@auto_docstring
class GPT2Model(GPT2PreTrainedModel):
    def __init__(self, config):
        super().__init__(config)

        self.embed_dim = config.hidden_size

        self.wte = nn.Embedding(config.vocab_size, self.embed_dim)
        self.wpe = nn.Embedding(config.max_position_embeddings, self.embed_dim)

        self.drop = nn.Dropout(config.embd_pdrop)
        self.h = nn.ModuleList([GPT2Block(config, layer_idx=i) for i in range(config.num_hidden_layers
        self.ln_f = nn.LayerNorm(self.embed_dim, eps=config.layer_norm_epsilon)

        self.gradient_checkpointing = False
        self._attn_implementation = config._attn_implementation

        # Initialize weights and apply final processing
        self.post_init()
```

GPT-2

Just treat positions 0,1,2,3, …, N as “tokens.”

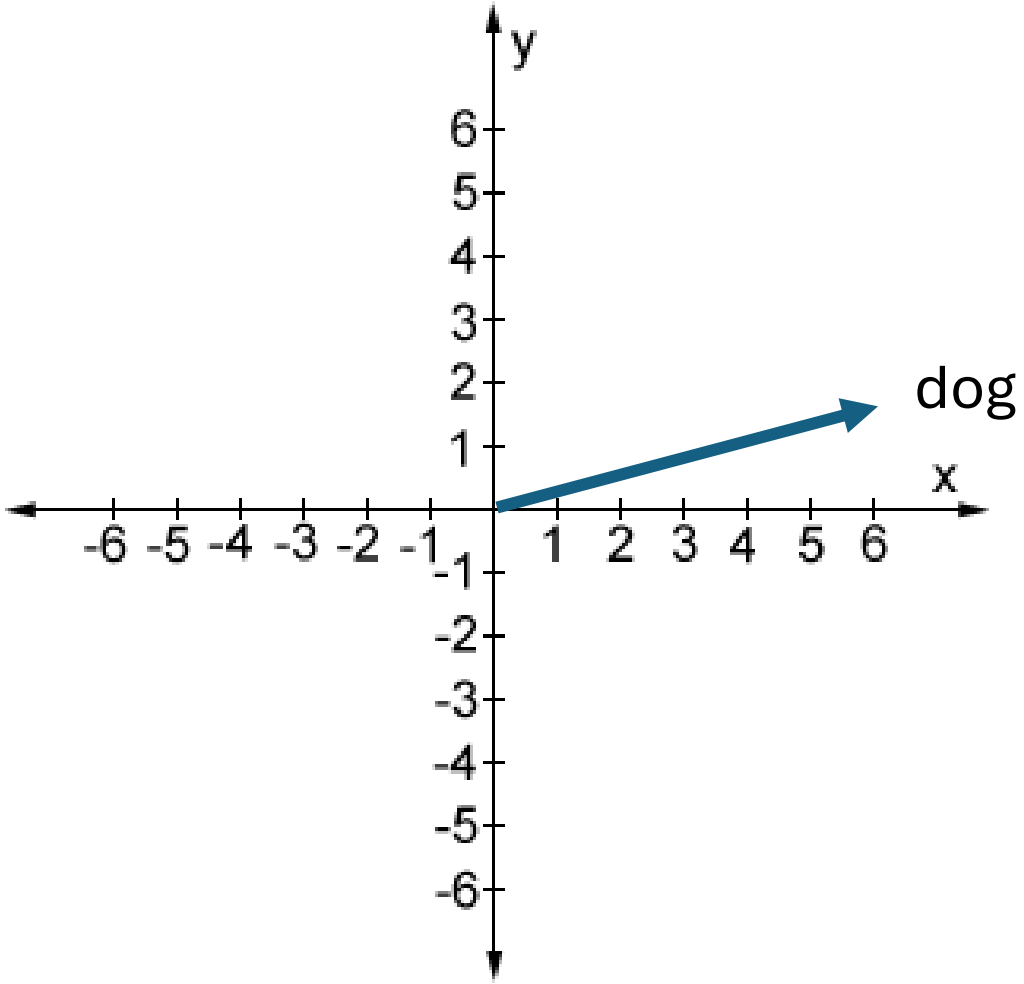
Relative Position Embedding: T5

- Absolute position embedding:
  - One position, one embedding. $E = E_{token} + E_{pos}$.
- Relative position embedding:

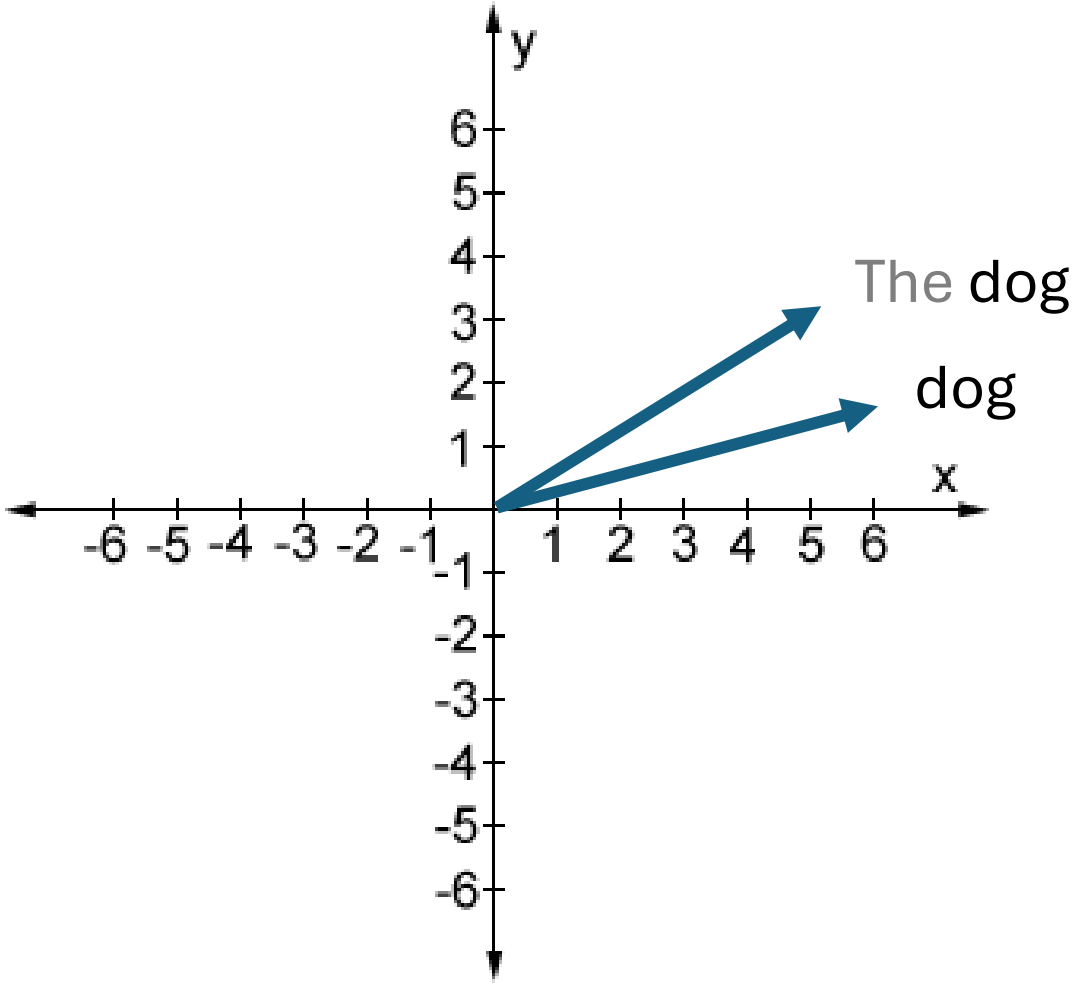


https://www.youtube.com/watch?v=o29P0Kpobz0

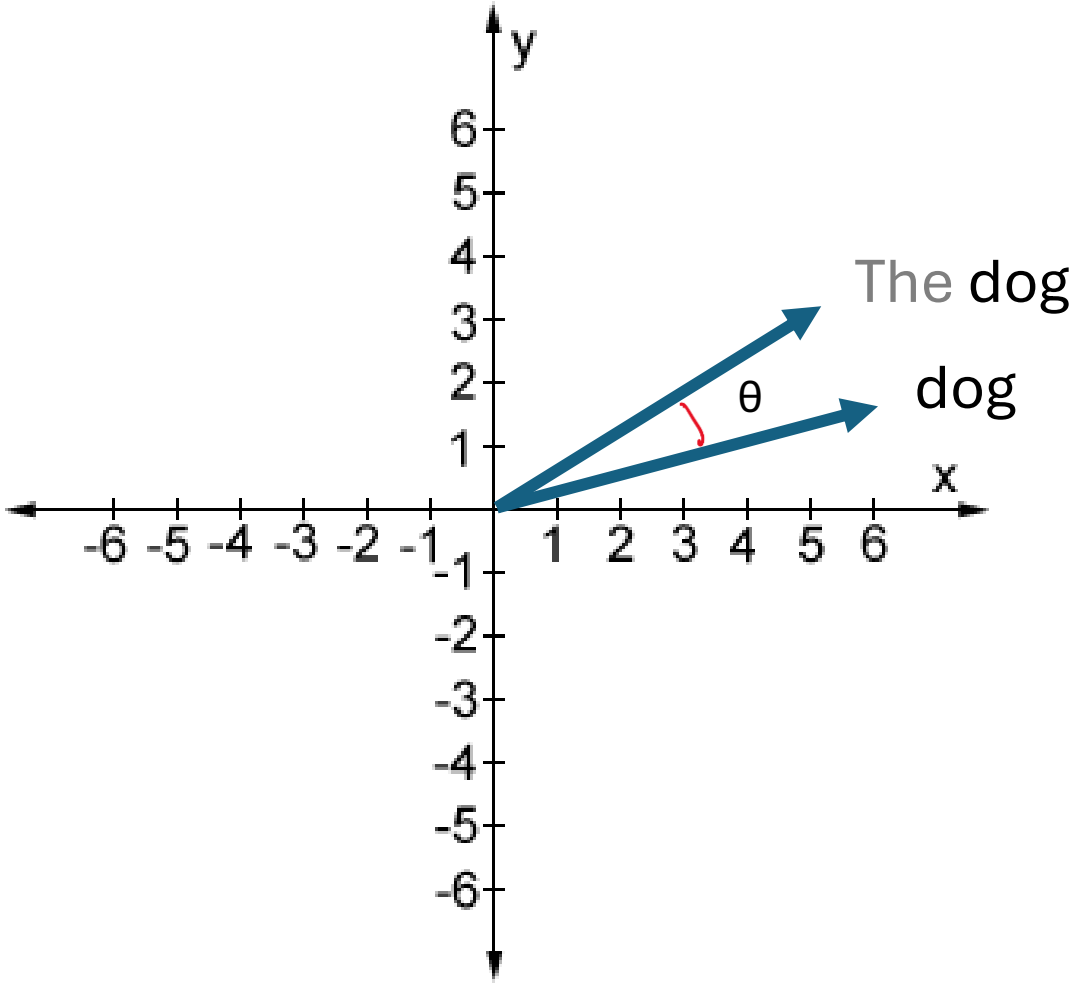
RoPE: Rotary Position Embedding



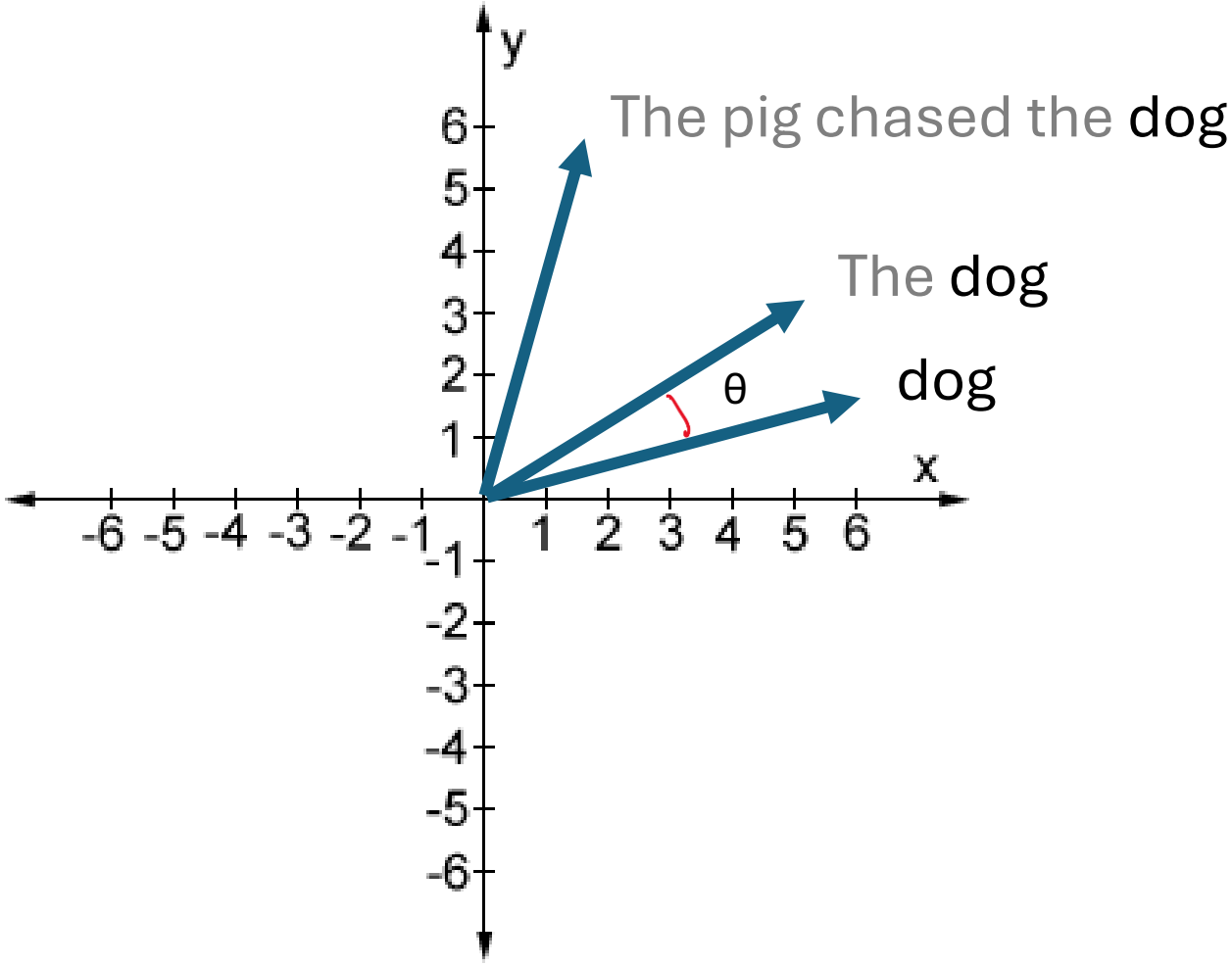
RoPE: Rotary Position Embedding



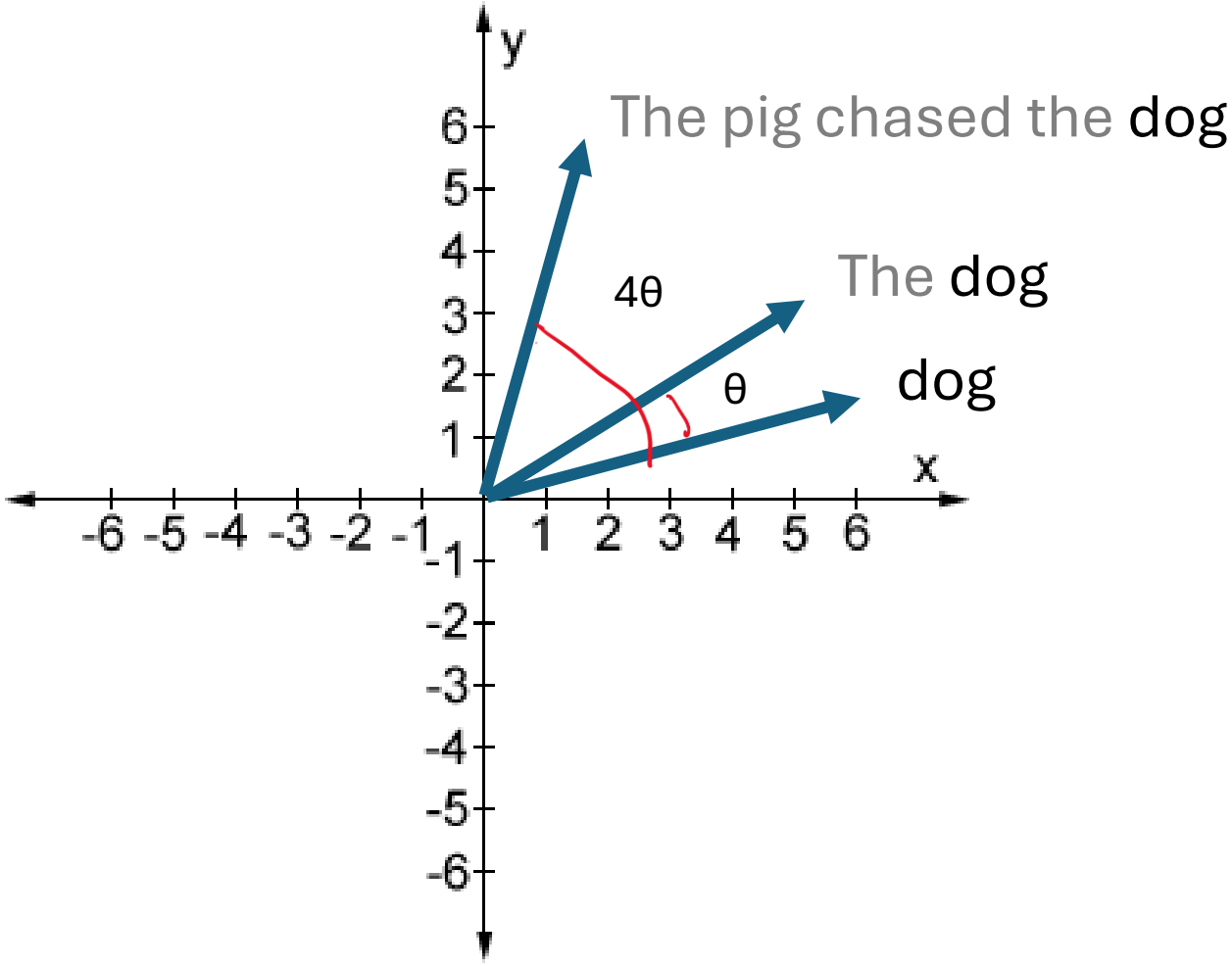
RoPE: Rotary Position Embedding



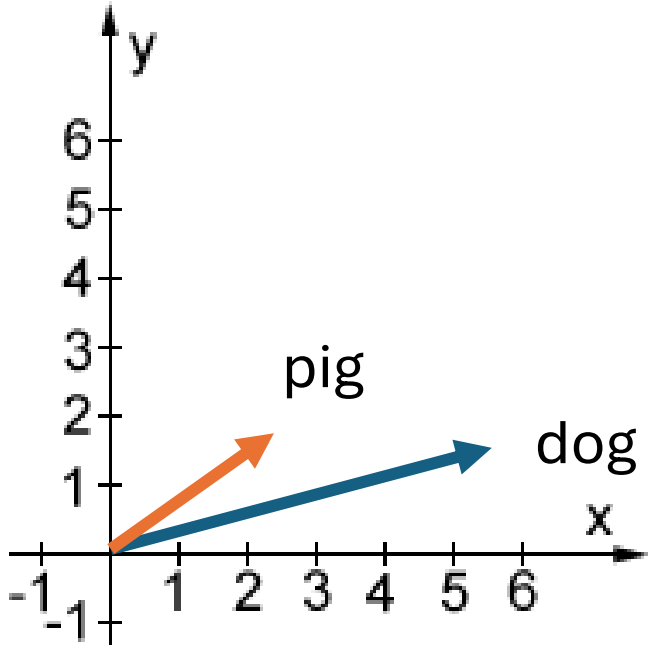
RoPE: Rotary Position Embedding



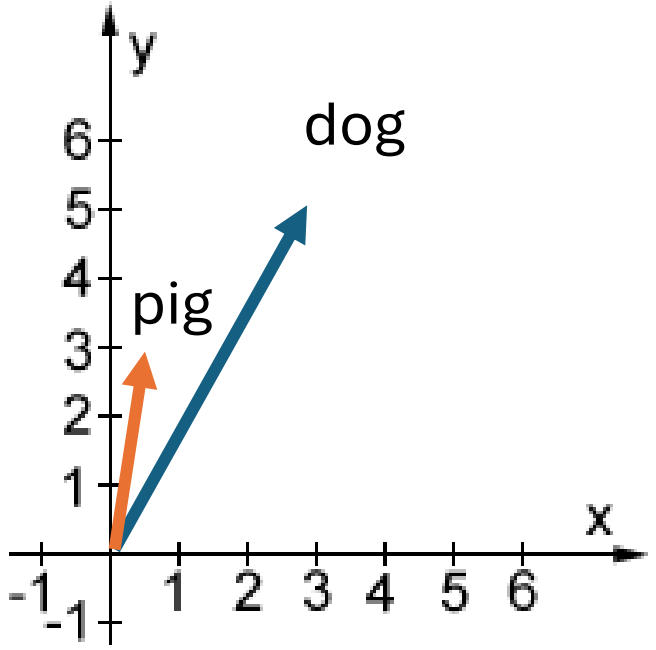
RoPE: Rotary Position Embedding



The pig chased the dog



Once upon a time,
the pig chased the dog



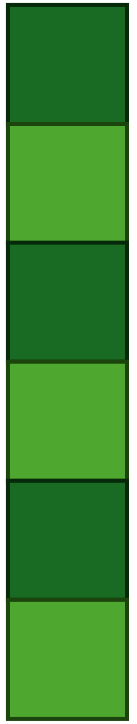
Implementation: 2D Example

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \underbrace{\begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}}_{\text{Rotate a vector by } m\theta} \begin{pmatrix} W^{(11)}_{\{q,k\}} & W^{(12)}_{\{q,k\}} \\ W^{(21)}_{\{q,k\}} & W^{(22)}_{\{q,k\}} \end{pmatrix} \begin{pmatrix} x^{(1)}_m \\ x^{(2)}_m \end{pmatrix}$$

Implementation: General Case

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$



Implementation: Actually…

```python
def rotate_half(x):
    """Rotates half the hidden dims of the input."""
    x1 = x[..., : x.shape[-1] // 2]
    x2 = x[..., x.shape[-1] // 2 :]
    return torch.cat((-x2, x1), dim=-1)


def apply_rotary_pos_emb(q, k, cos, sin, position_ids=None, unsqueeze_dim=1):
    """Applies Rotary Position Embedding to the query and key tensors."""
    cos = cos.unsqueeze(unsqueeze_dim)
    sin = sin.unsqueeze(unsqueeze_dim)
    q_embed = (q * cos) + (rotate_half(q) * sin)
    k_embed = (k * cos) + (rotate_half(k) * sin)
    return q_embed, k_embed
```

Llama – huggingface's implementation

RoPE Summary

- Su et al., (2021). RoFormer: Enhanced Transformer with Rotary Position Embedding.
- Good performance:
- Combines the strength of absolute and relative position encoding.
- The most widely used type of position encoding in contemporary LMs.
  - Basically all major models right now: LLaMA, Mistral, Qwen, GPT-NeoX, Falcon, Deepseek, Gemma…
  - ChatGPT (GPT-4/5), Grok, Claude, … probably also use RoPE.

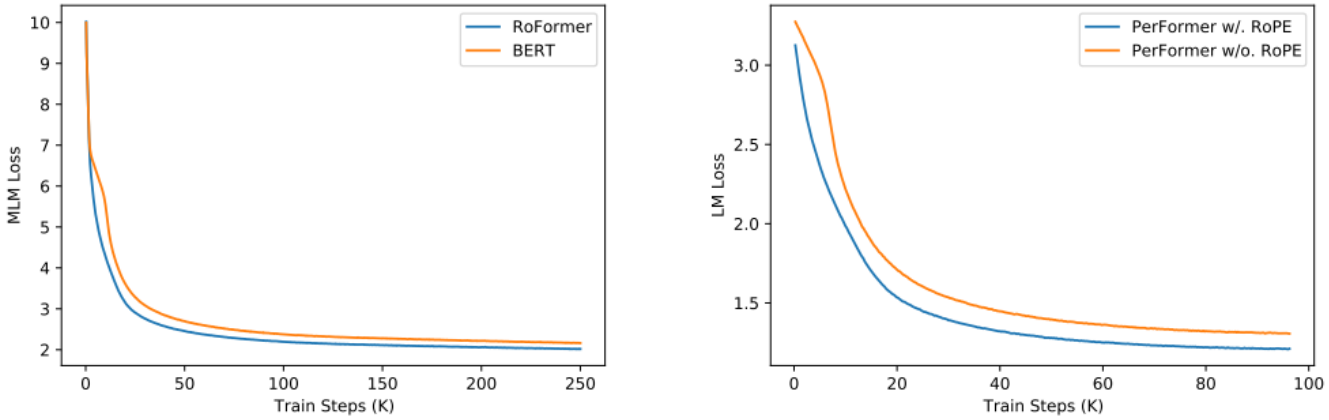
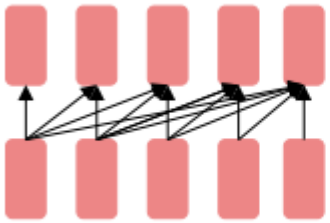


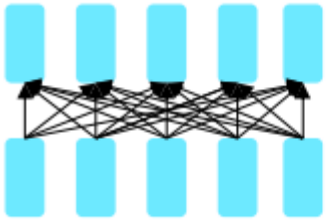
Figure 3: Evaluation of RoPE in language modeling pre-training. **Left**: training loss for BERT and RoFormer. **Right**: training loss for PerFormer with and without RoPE.

Three types of architectures



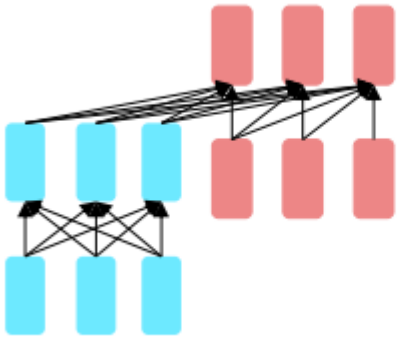
Decoders

- Next word prediction.
- Easy to train. Abundant amount of data.
- Nice to generate from; can’t condition on future words.



Encoders

- Gets bidirectional context – can condition on future!
- Good word embeddings.
- MLM, BERT.

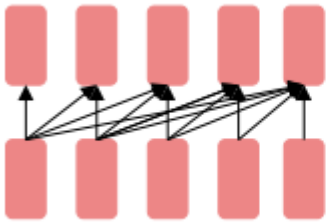


Encoder-Decoders

- Good parts of decoders and encoders?
- What’s the best way to pretrain them?

We will come to more details…

Three types of architectures



Decoders

- Next word prediction.
- Easy to train. Abundant amount of data.
- Nice to generate from; can't condition on future words.

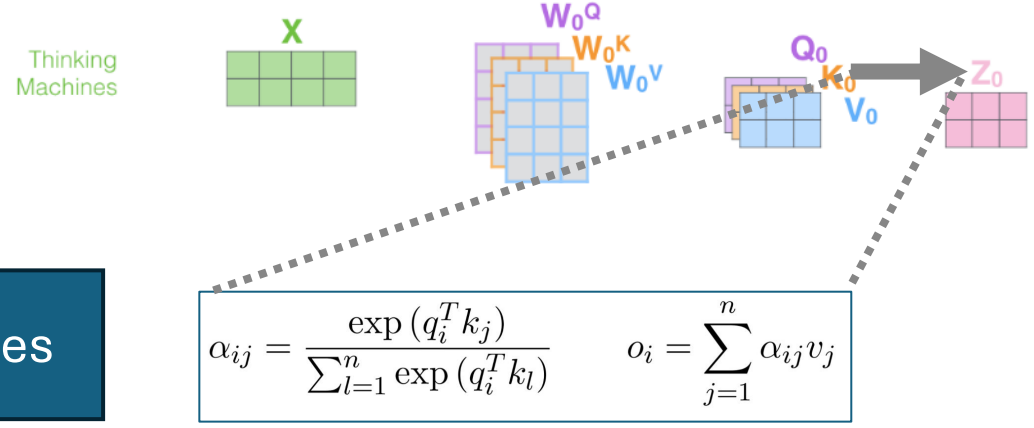
Encoders

- Gets bidirectional context – can condition on future!
- Good word embeddings.
- MLM, BERT.

Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

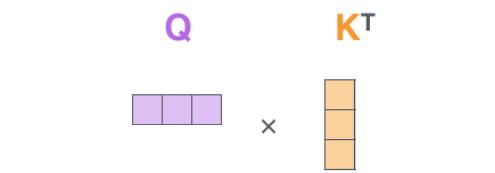
Self Attention

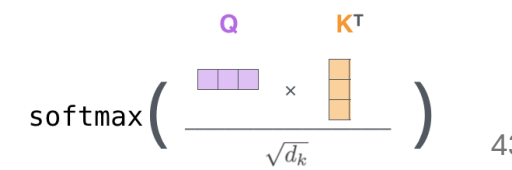


I	like	eating	apples

What is o(like)? I.e., the attention score from like to the sentence.

query				Q4 “apples”
key	K1 “I”	K2 “like”	K3 “eating”	K4 “apples”
attention weights*	14.2	18.1	10.3	7.9
attention scores* (o)	0.3	0.6	0.07	0.03

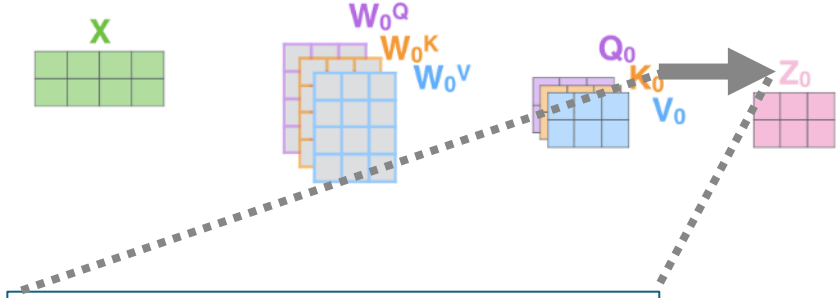




*: made-up numbers, not real.

Self Attention

Thinking Machines



$$\alpha_{ij} = \frac{\exp(q_i^T k_j)}{\sum_{l=1}^{n} \exp(q_i^T k_l)} \qquad o_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

I | like | eating | apples

What is z(like)?

attention scores* (o): 0.3 | 0.6 | 0.07 | 0.03

values: V1 “I” | V2 “like” | V3 “eating” | V4 “apples”

Weighted sum

Z4 “like”

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right)$$

$$X \times W^V = V$$

*: made-up numbers, not real.

GPT-2

Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Table 3. Zero-shot results on many datasets. No training or fine-tuning was performed for any of these results. PTB and WikiText-2 results are from (Gong et al., 2018). CBT results are from (Bajgar et al., 2016). LAMBADA accuracy result is from (Hoang et al., 2018) and LAMBADA perplexity result is from (Grave et al., 2016). Other results are from (Dai et al., 2019).

Perplexity

Normalized by the number of words.

$$\mathrm{PPL}(W) = P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}}$$

Inverse probability of a corpus, according to a LM.

Perplexity… Boring!

OpenAI's GPT-2 has been promoted as "an AI" that exemplifies an emergent understanding of language after mere unsupervised training on about 40GB of webpage text. It sounds really convincing in interviews:

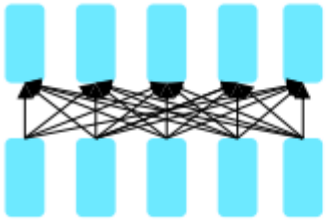
- *Q: Which technologies are worth watching in 2020?*
  A: I would say it is hard to narrow down the list. The world is full of disruptive technologies with real and potentially huge global impacts. The most important is artificial intelligence, which is becoming exponentially more powerful. There is also the development of self-driving cars. There is a lot that we can do with artificial intelligence to improve the world….
- *Q: Are you worried that ai* [sic] *technology can be misused?*
  A: Yes, of course. But this is a global problem and we want to tackle it with global solutions……

--- "AI can do that", *The World in 2020 – The Economist*

Three types of architectures

Decoders

- Next word prediction.
- Easy to train. Abundant amount of data.
- Nice to generate from; can't condition on future words.



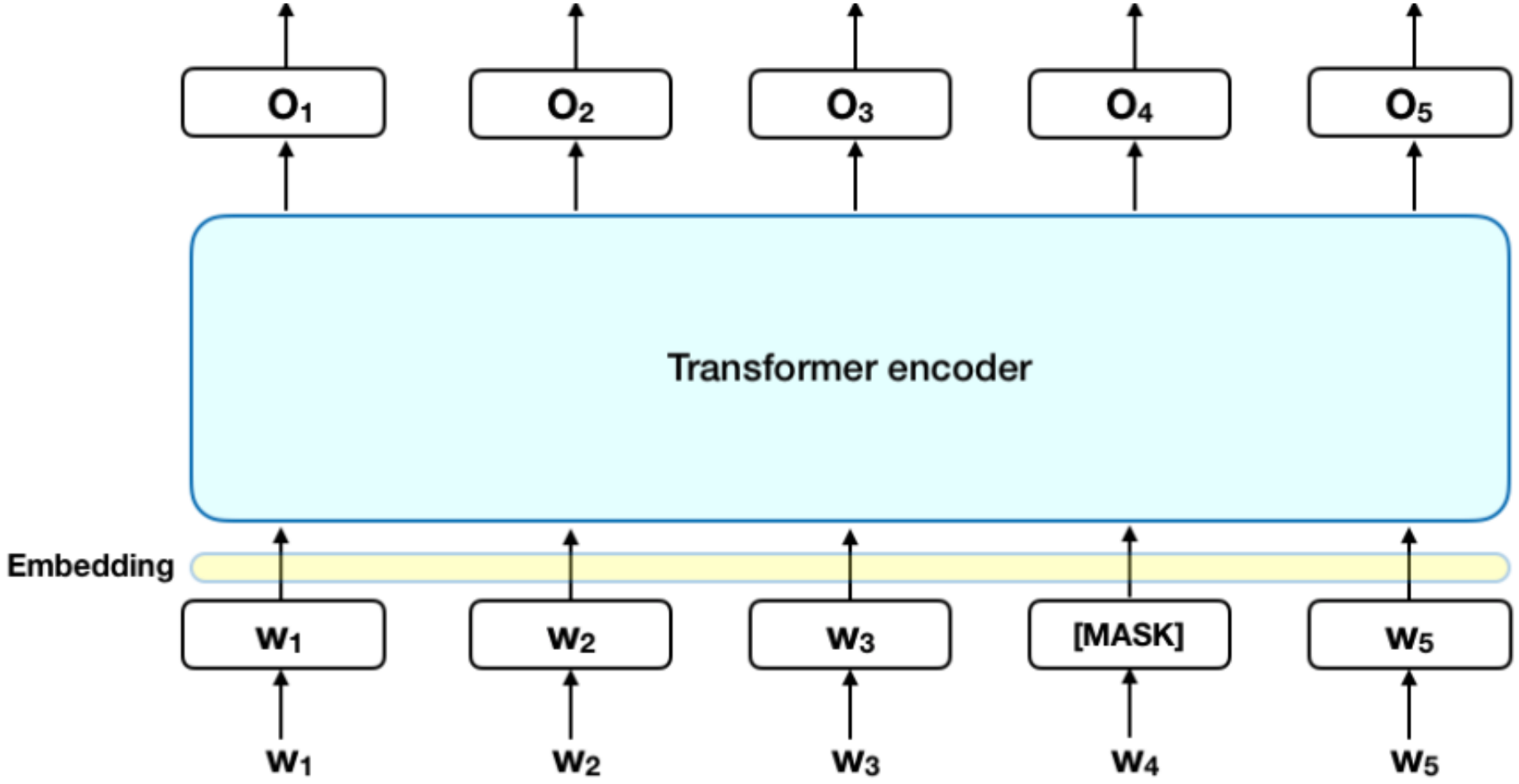
Encoders

- Gets bidirectional context – can condition on future!
- Good word embeddings.
- MLM, BERT.

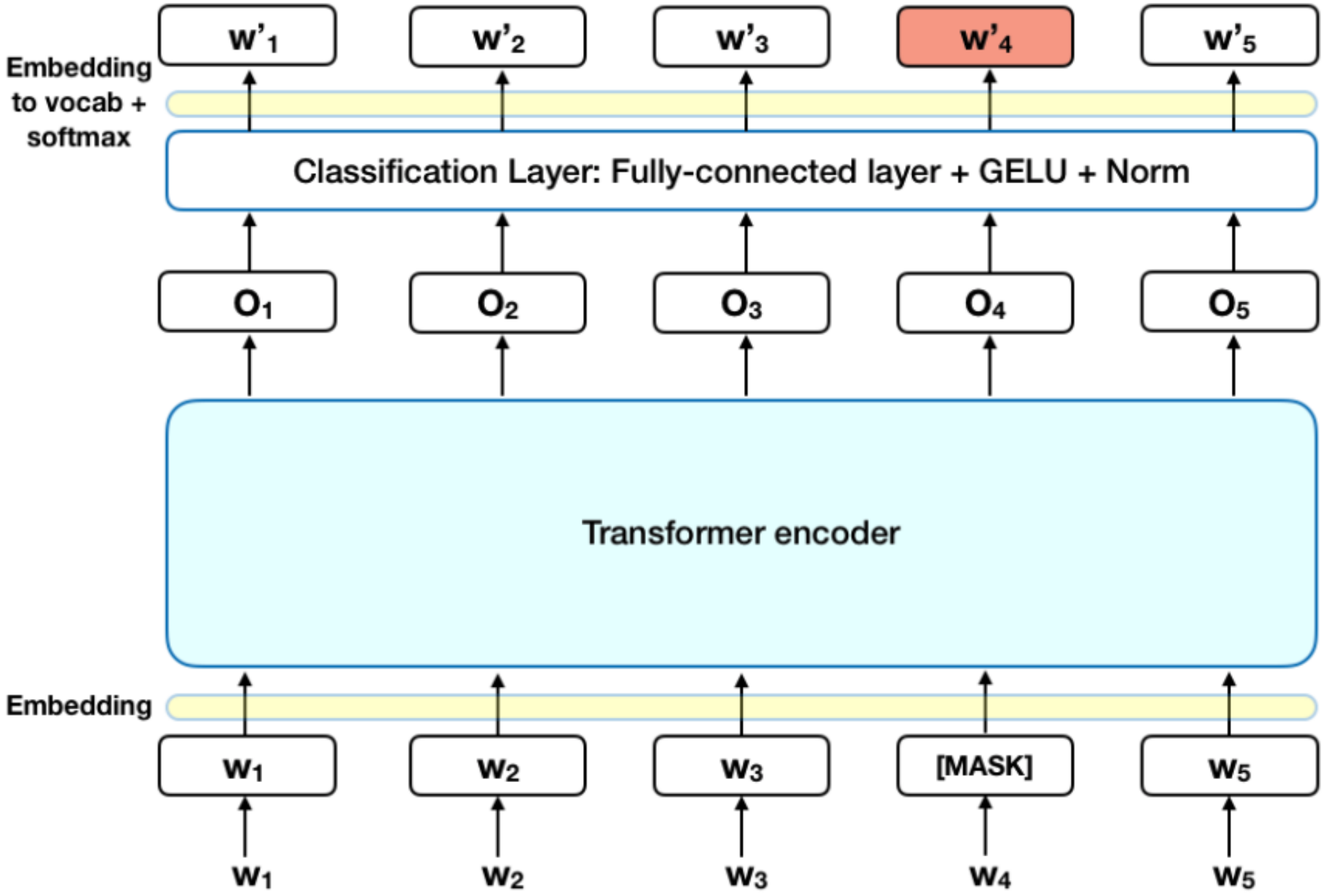
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

BERT


O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

BERT


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

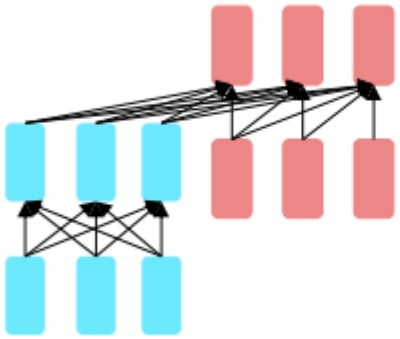
Three types of architectures

Decoders

- Next word prediction.
- Easy to train. Abundant amount of data.
- Nice to generate from; can't condition on future words.

Encoders

- Gets bidirectional context – can condition on future!
- Good word embeddings.
- MLM, BERT.



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretraining encoder-decoders: What pretraining objective to use?

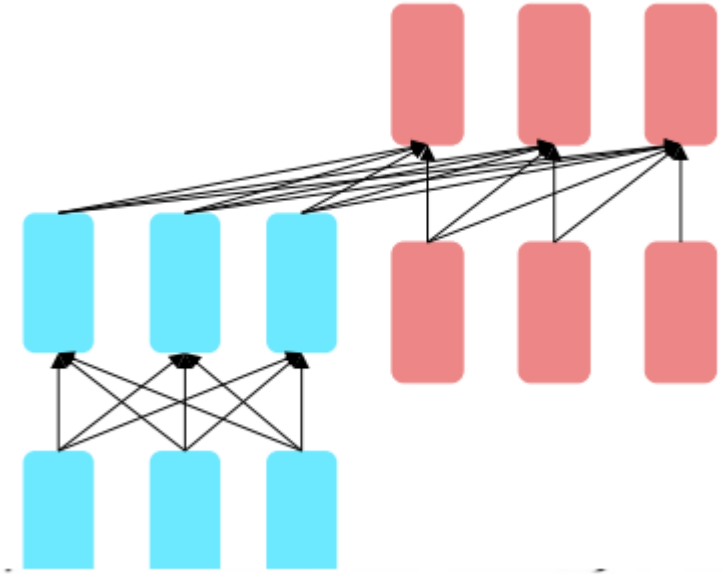
- What Raffel et al. (2018) found to work best was span corruption. Their model: T5.
- Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

Original text

Thank you ~~for inviting~~ me to your party ~~last~~ week.

Targets

<X> for inviting <Y> last <Z>



- This is implemented in text preprocessing: it's still an objective that looks like language modeling at the decoder side.

Inputs

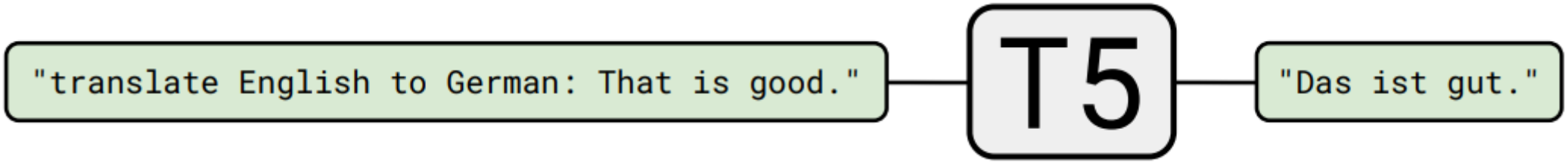
Thank you <X> me to your party <Y> week.

Pretraining encoder-decoders: What pretraining objective to use?

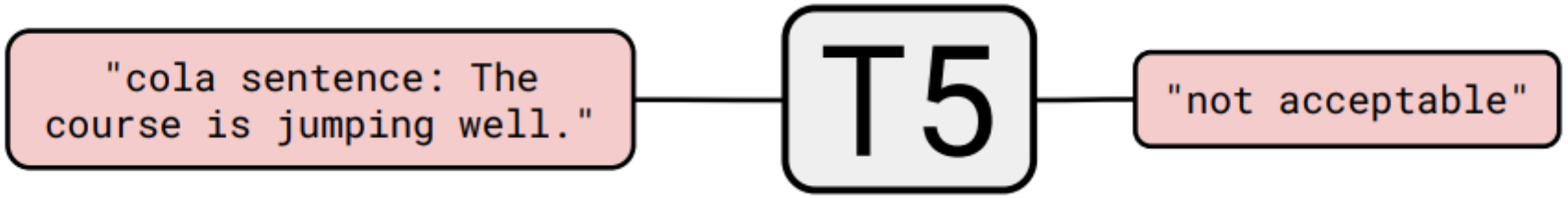
- Raffel et al., (2018) found encoder-decoders to work better than decoders for their tasks, and span corruption (denoising) to work better than language modeling.

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

One surprising finding



One surprising finding

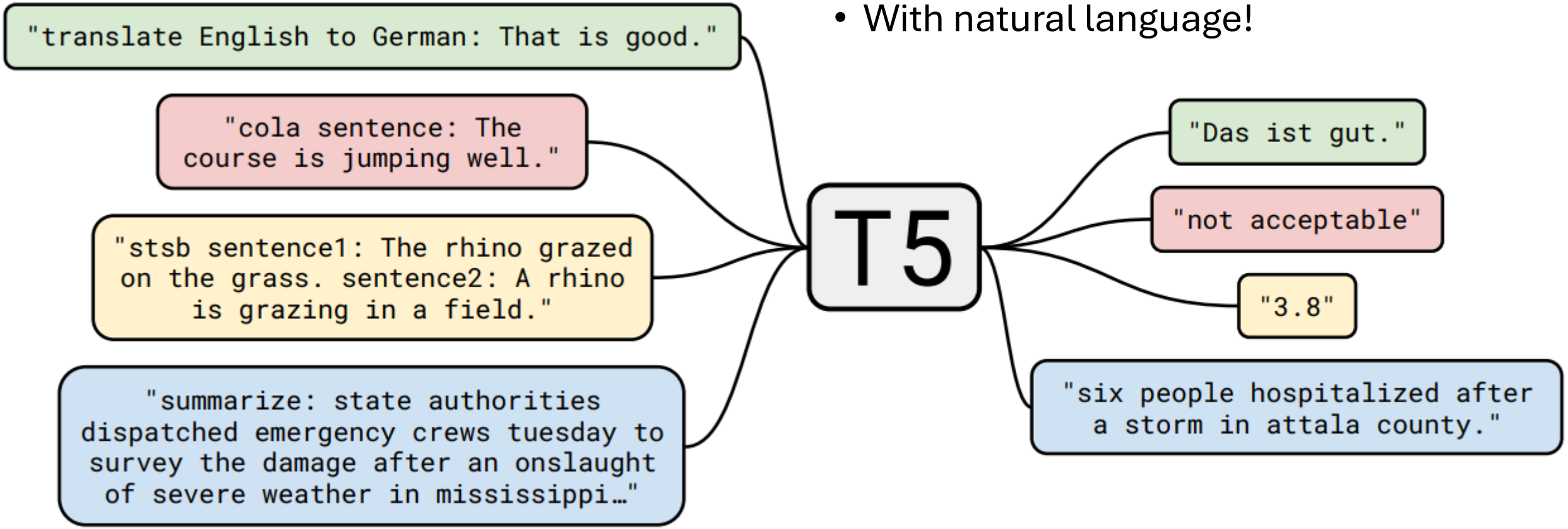


One surprising finding

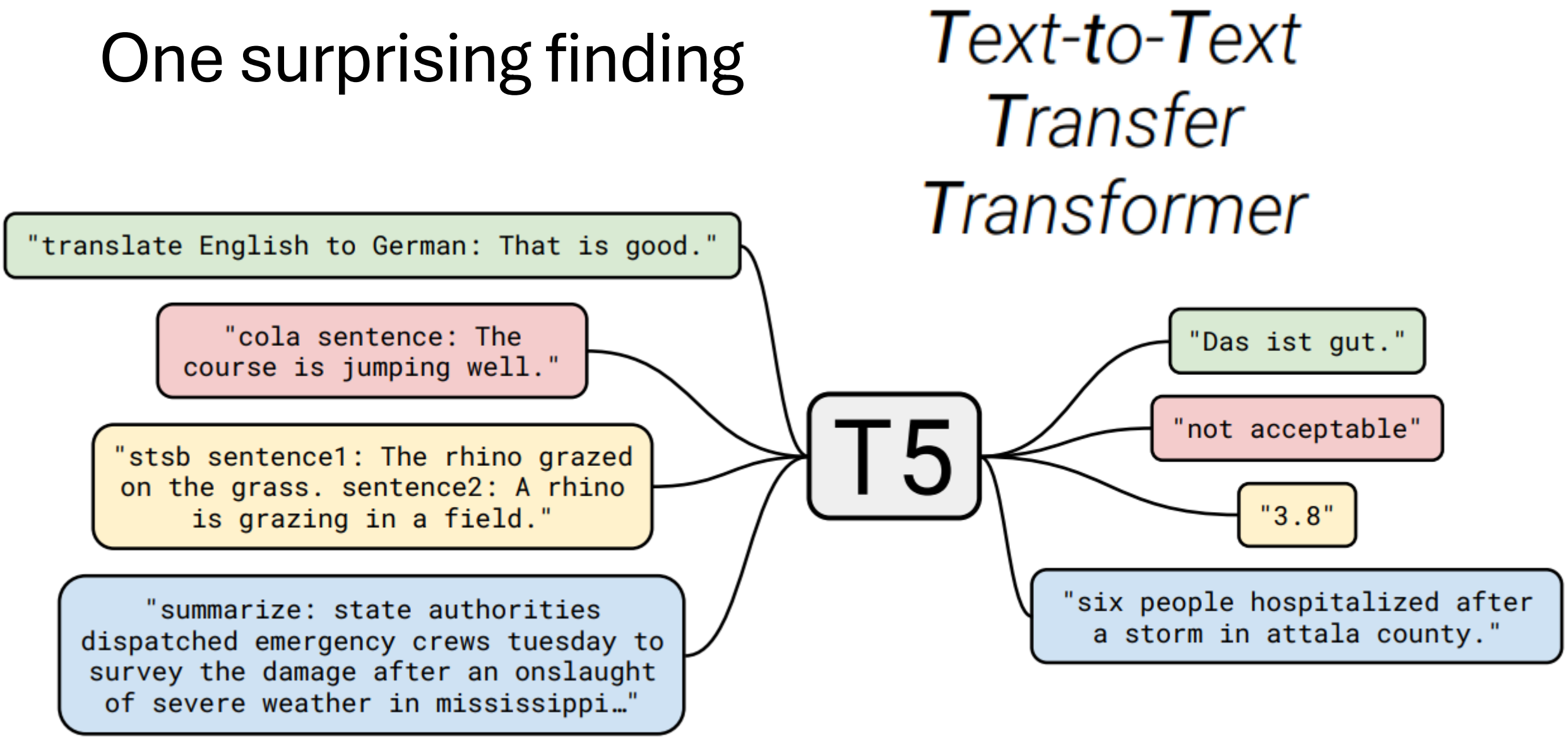
One surprising finding

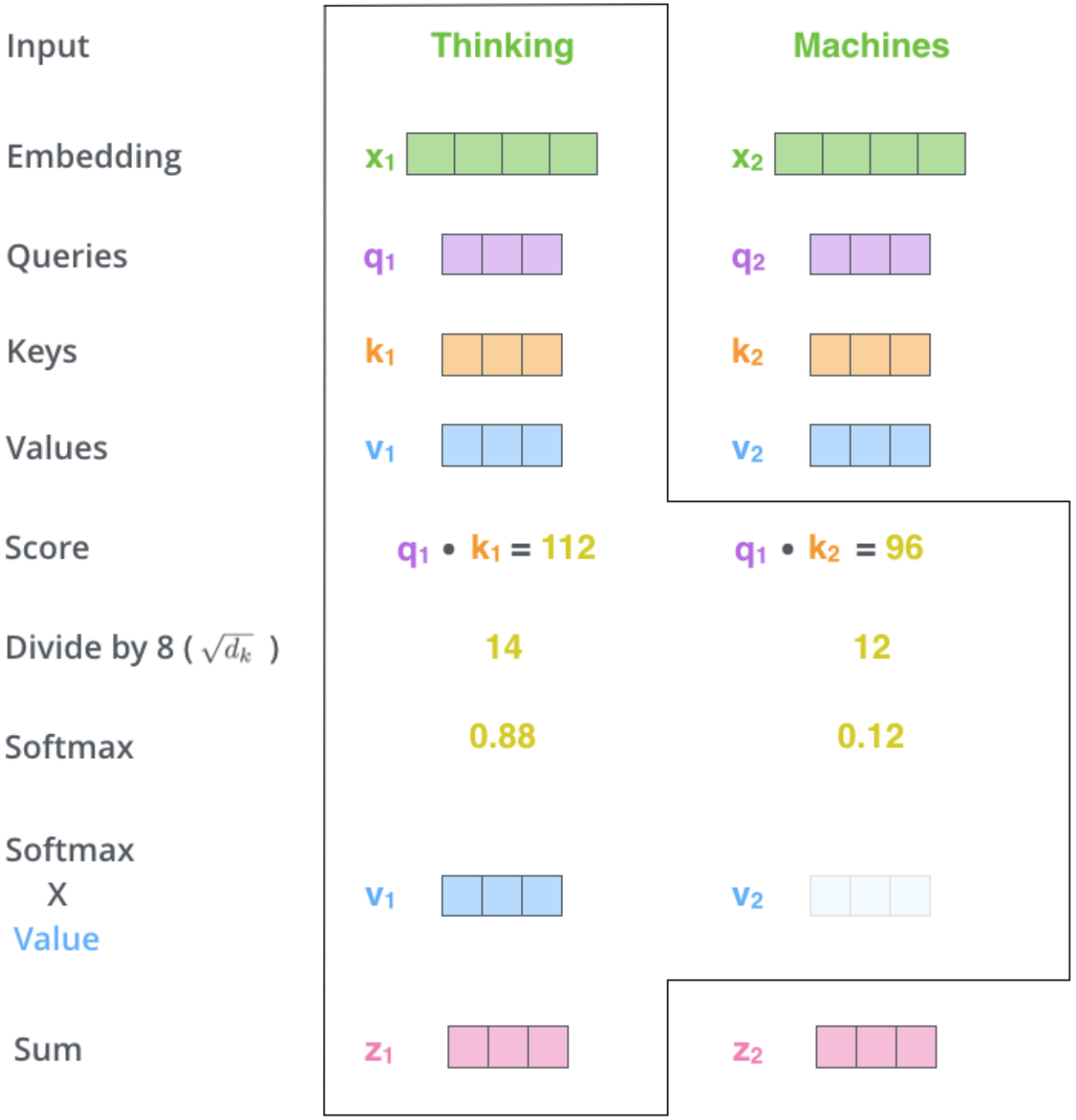
A fascinating property of T5:

- It can be finetuned to answer a wide range of questions, retrieving knowledge from its parameters.
- With natural language!



One surprising finding





Review

- Embeddings:
  - Token + Sentence + Position
- Multi-Head Attention
- Feed forward module (MLP)
- Layers