CSE 156 | Lecture 12: Midterm Review

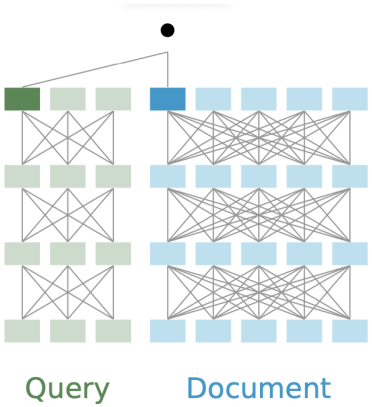
Ndapa Nakashole

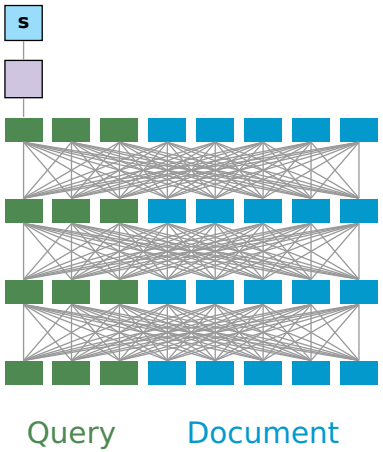
November 07, 2024

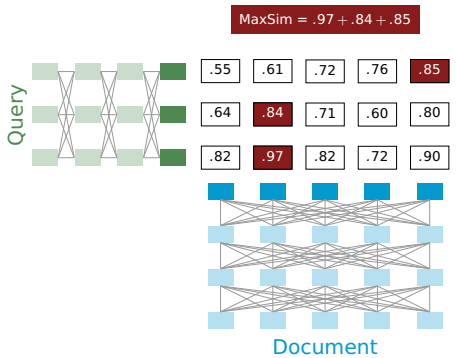
Administrative matters

- **PA2**: Last day to submit is today
- **PA3**: has been released, due Nov 18

Recap: Neural Retrieval Methods







Bi-Encoder

- ✓ Independent, Pre-computed Representations
- ✗ Coarse-Grained Representations

Cross-Encoder

- ✓ Fine-Grained Query-Document Interactions
- ✗ No Pre-computed Representations

Contextualized Late Interaction over BERT (**ColBERT**)

- ✓ Independent, Pre-computed Representations
- ✓ Fine-Grained Query Document Interactions

Table of Contents

MidTerm

- Midterm: next Tuesday, November 12th, 2024. Bring the following:
  - Your student ID
  - One double-sided note sheet
  - A pen

Midterm Exam Topics: 1/3

Foundations of Neural NLP

- **Softmax classifer**: model, training objective, gradient update
- **Classification**: how weights work in this setting
- **Optimization**: stochastic gradient descent, impact of step size on optimization
- **Feedforward neural networks**: definition, activation functions
- **Training neural networks**
- **Word embeddings**: skip-gram model, definition, properties
- **Subword tokenization**
- **Deep averaging networks**: model from PA1, limitations of the model

Midterm Exam Topics: 2/3

Modern Language Models: Background

- n-**gram language modeling**: basic definition, count-based parameter estimation
- **Recurrent neural networks**: model, training RNNs
- **Sequence to sequence models**: model, training, inference

Midterm Exam Topics: 3/3

Modern Language Models: Key Ingredients

- **Self-attention**: mathematical definition, how it works
- **Transformers**: architecture details
- **Transformer language modeling**: everything from PA2, including training, inference, and evaluation (perplexity)
- **Encoders**: masked language modeling and BERT
- **Decoders**: causal language modeling, training, inference
- **Inference in decoders**: nucleus sampling, beam search

Some Sample Questions

True or False Questions

1 point each

BERT uses a unidirectional language model.

- ○ True
- ○ False

In a Transformer architecture, the feedforward layer can be thought of as providing additional non-linear transformations that enhance each token's representation.

- ○ True
- ○ False

The softmax classifier can be used in any situation where the sigmoid classifier is applicable, but the sigmoid classifier cannot always replace the softmax classifier.

- ○ True
- ○ False

True or False Questions

1 point each

BERT uses a unidirectional language model.

- ○ True
- ✓ False

In a Transformer architecture, the feedforward layer can be thought of as providing additional non-linear transformations that improve each token's representation.

- ✓ True
- ○ False

The softmax classifier can be used in any situation where the sigmoid classifier is applicable, but the sigmoid classifier cannot always replace the softmax classifie

- ✓ True
- ○ False

Multiple Choice Questions (1)

Which of the following changes would likely reduce the effectiveness of a pre-trained GPT model for language generation tasks? Select all that apply.

- ○ Using bidirectional attention during pre-training
- ○ Decreasing the hidden dimension of the model
- ○ Pre-training with a unidirectional left-to-right objective
- ○ Using a smaller dataset for pre-training

Multiple Choice Questions (1)

Which of the following changes would likely reduce the effectiveness of a pre-trained GPT model for language generation tasks? Select all that apply.

- ✓ Using bidirectional attention during pre-training
- ✓ Decreasing the hidden dimension of the model
- ○ Pre-training with a unidirectional left-to-right objective
- ✓ Using a smaller dataset for pre-training

Multiple Choice Questions (2)

In a language model using causal masking, the objective is to maximize the probability of each token given only its preceding tokens in the sequence. Which of the following correctly represents the negative log-likelihood loss for a sequence of tokens $t_1, t_2, \ldots, t_n$?

- $-\sum_{i=1}^{n} \log P\left(t_i \mid t_1, t_2, \ldots, t_{i-1}\right)$
- $-\sum_{i=1}^{n} \log P\left(t_i \mid t_{i+1}, t_{i+2}, \ldots, t_n\right)$
- $-\prod_{i=1}^{n} P\left(t_i \mid t_1, t_2, \ldots, t_{i-1}\right)$
- $-\prod_{i=1}^{n} P\left(t_i \mid t_{i+1}, t_{i+2}, \ldots, t_n\right)$

Multiple Choice Questions (2)

In a language model using causal masking, the objective is to maximize the probability of each token given only its preceding tokens in the sequence. Which of the following correctly represents the negative log-likelihood loss for a sequence of tokens $t_1, t_2, \ldots, t_n$?

- ✓ $-\sum_{i=1}^{n} \log P\left(t_i \mid t_1, t_2, \ldots, t_{i-1}\right)$
- ○ $-\sum_{i=1}^{n} \log P\left(t_i \mid t_{i+1}, t_{i+2}, \ldots, t_n\right)$
- ○ $-\prod_{i=1}^{n} P\left(t_i \mid t_1, t_2, \ldots, t_{i-1}\right)$
- ○ $-\prod_{i=1}^{n} P\left(t_i \mid t_{i+1}, t_{i+2}, \ldots, t_n\right)$

Multiple Choice Questions (3)

Which of the following types of parameters are typically **not** learned in a Deep Averaging Network? Select all that apply.

- ○ Feedforward layer weights
- ○ Parameters for self-attention (e.g., W_k for key mappings)
- ○ Word embeddings
- ○ Attention map weights

Multiple Choice Questions (3)

Which of the following types of parameters are typically **not** learned in a Deep Averaging Network? Select all that apply.

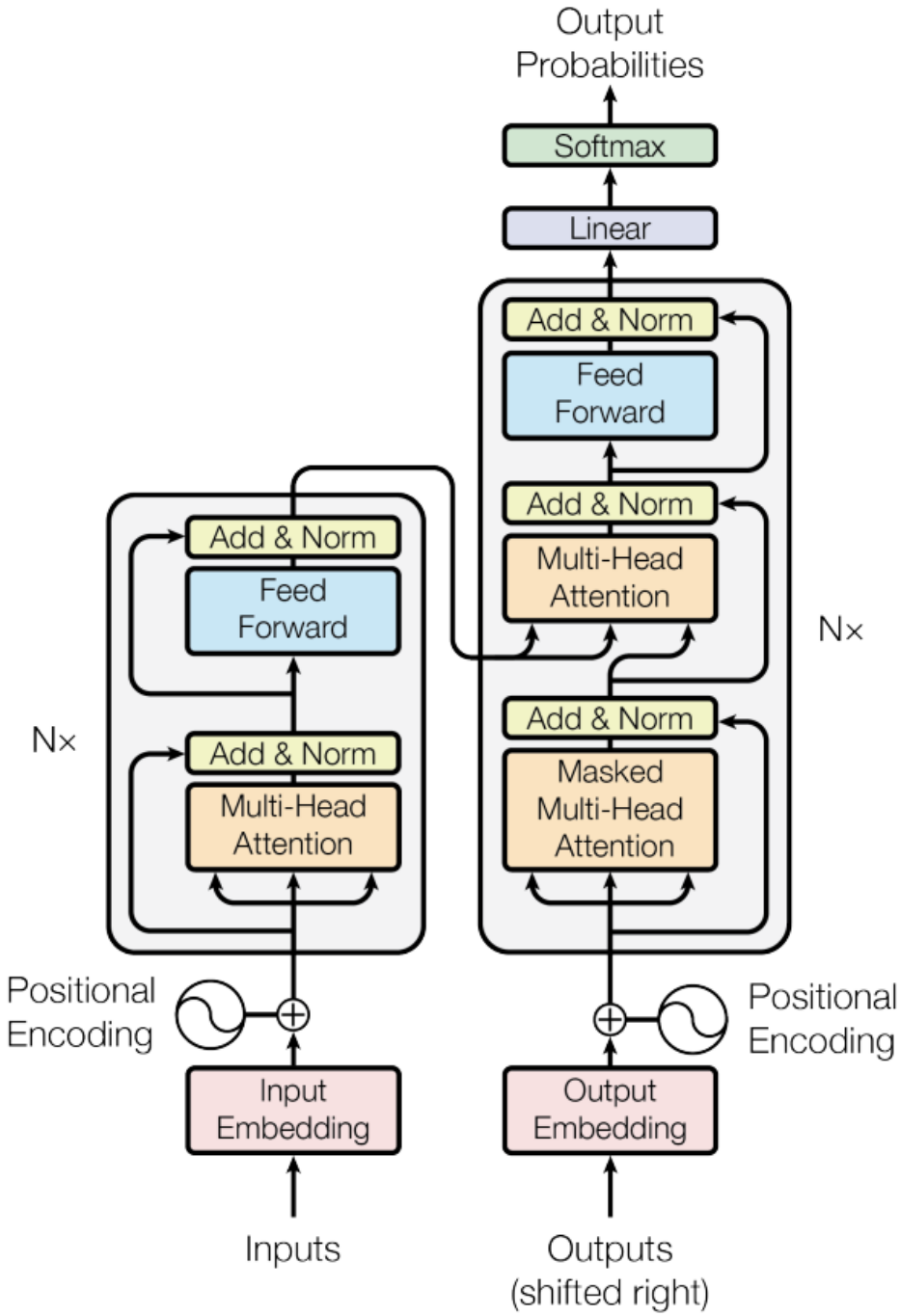
- ○ Feedforward layer weights
- ✓ Parameters for self-attention (e.g., W_k for key mappings)
- ○ Word embeddings
- ✓ Attention map weights

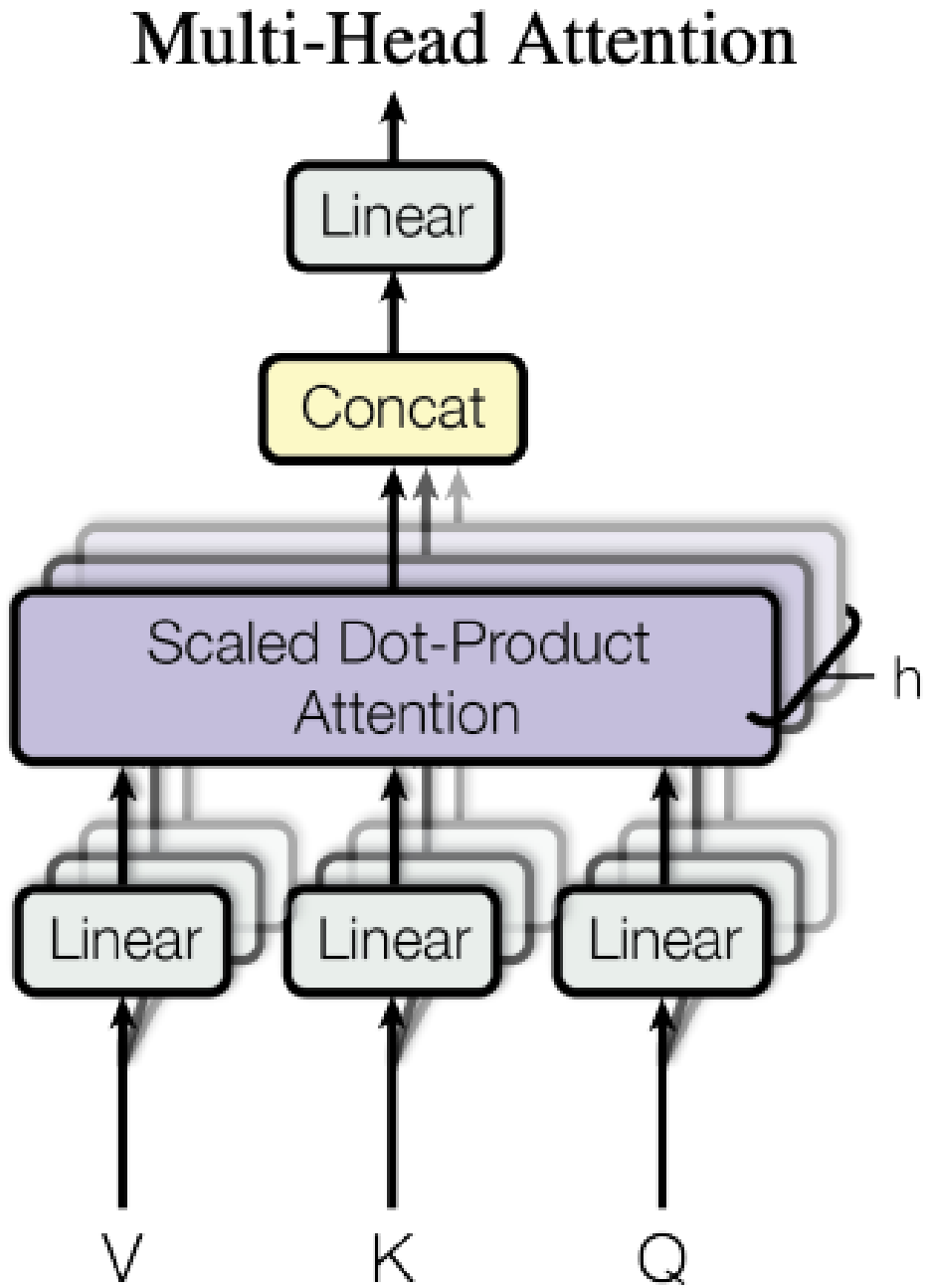
Explanation Question

- Both Feedforward neural networks and Softmax classifiers can output probabilities. What architectural feature allows Feedforward networks to capture more complex patterns than a simple Softmax classifier?
- **Points**: 2 points

Explanation Question

- **Question**: Why are positional embeddings necessary in the Transformer architecture but not in RNNs?
- **Points**: 2 points

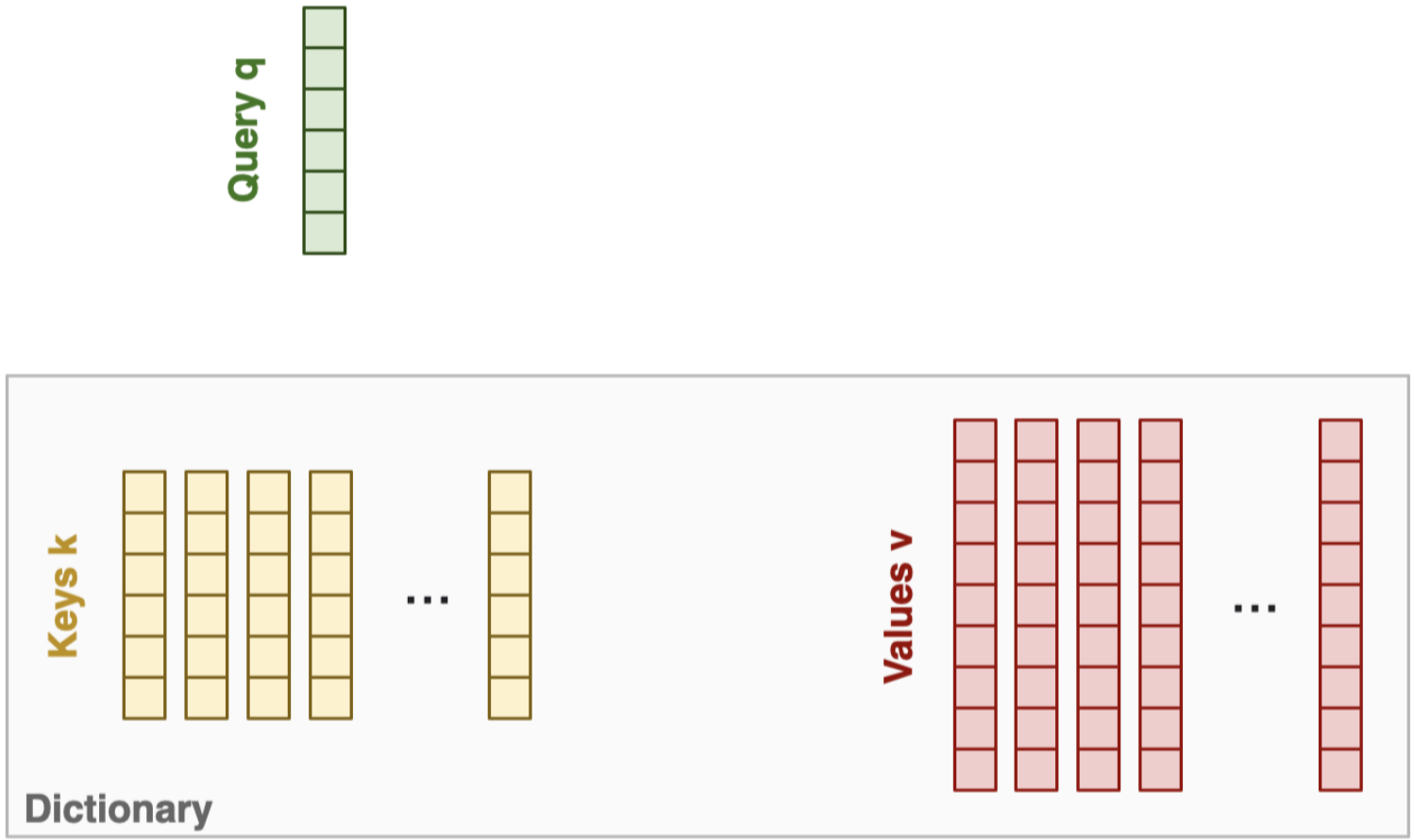

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)


Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

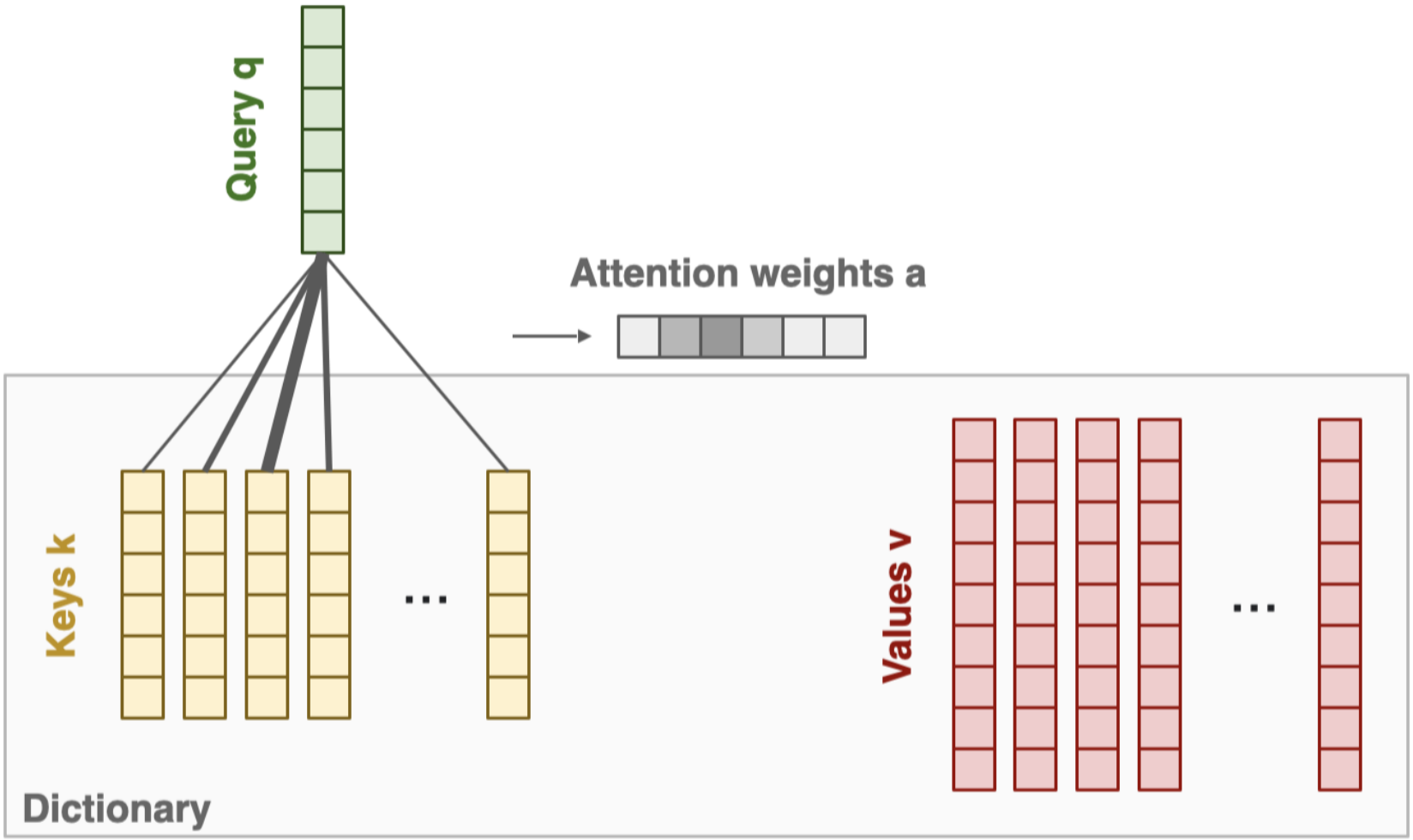
Self-Attention: Overview

- **Given**: Query vector $\boldsymbol{q}$ and a set of Key-Value ($\boldsymbol{k}$-$\boldsymbol{v}$) vector pairs.
- **Output**: a weighted sum of the values, where the weights are determined by the similarity of the query to the keys.
  - A selective summary of the values, with respect to the query.
- Query, Keys, Values, and Outputs are all vectors.

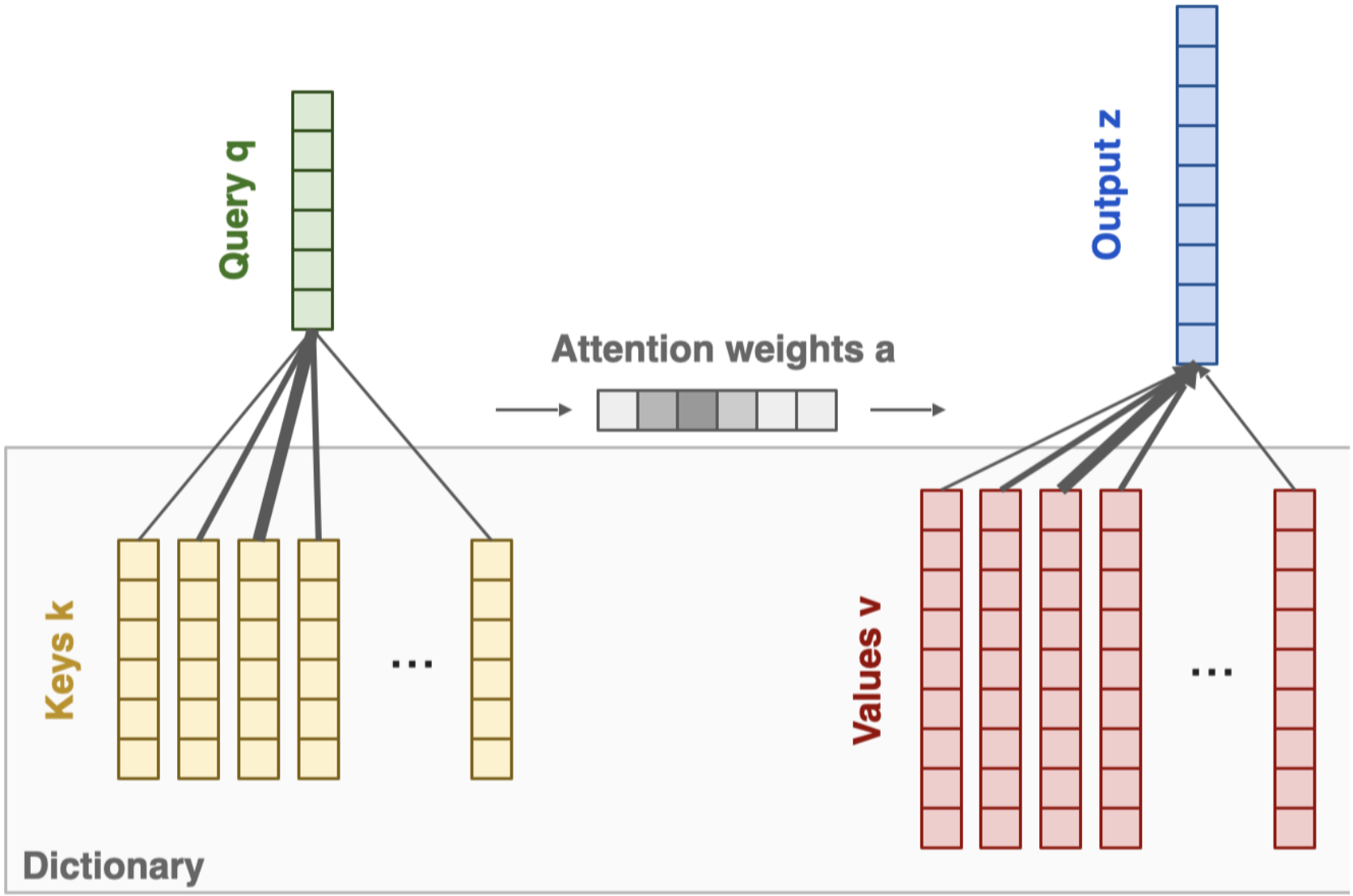
Queries, Keys, & Values: Example



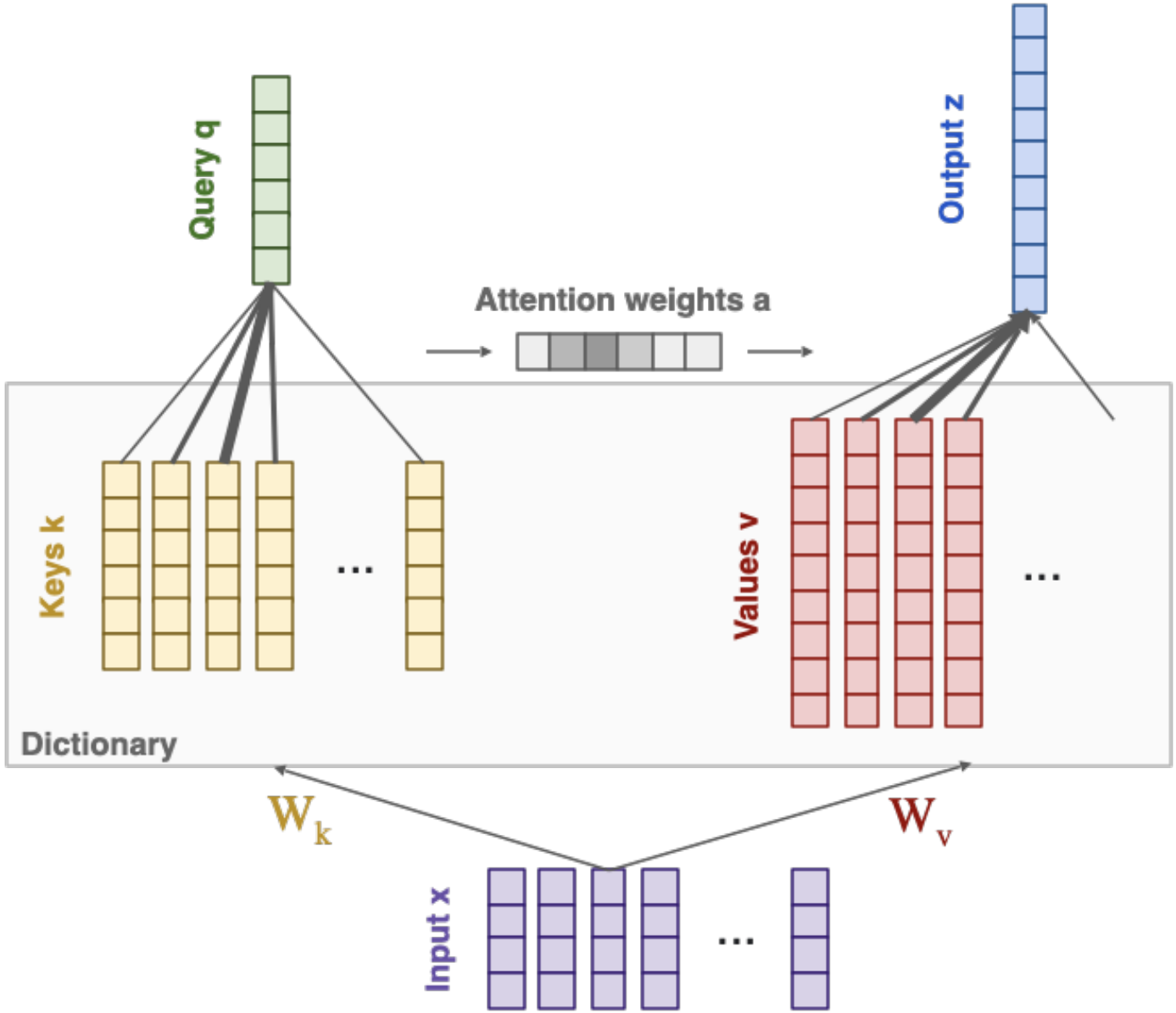
Computing the Attention Scores



Output


Query q
Output z
Attention weights a
Keys k
...
Values v
...
Dictionary

Where do the Keys, Queries, and Values come from?



Key, Value, and Query Derivation

k and v are derived from the same input x:

$$k = W_k \cdot x \quad v = W_v \cdot x$$

The query q can come from a separate input y:

$$q = W_q \cdot y$$

Or from the same input x, then we call it “self-attention":

$$q = W_q \cdot x$$

Attention Mechanism: 1 Query

$$\text{similarity}_i = \boldsymbol{q} \cdot \boldsymbol{k}_i$$

$$\text{attention}_i = \frac{e^{\boldsymbol{q} \cdot \boldsymbol{k}_i}}{\sum_j e^{\boldsymbol{q} \cdot \boldsymbol{k}_j}}$$

Compute the Weighted Sum of the Values:

$$A(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \text{attention}_i \cdot \boldsymbol{v}_i$$

Expands to:

$$A(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \sum_i \frac{e^{\boldsymbol{q} \cdot \boldsymbol{k}_i}}{\sum_j e^{\boldsymbol{q} \cdot \boldsymbol{k}_j}} \boldsymbol{v}_i$$

where $A \in \mathbb{R}^{d_v}$

Example Setup

Given:

- Query vector $\boldsymbol{q} \in \mathbb{R}^{1 \times d_k}$ (where $|Q| = 1$):

$$\boldsymbol{q} = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

- Key matrix $\boldsymbol{K} \in \mathbb{R}^{|K| \times d_k}$ (where $d_k = 2$ and $|K| = 2$):

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{k}_1 & \boldsymbol{k}_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

- Value matrix $\boldsymbol{V} \in \mathbb{R}^{|K| \times d_v}$ (where $|K| = 2$ and $d_v = 2$):

$$\boldsymbol{V} = \begin{bmatrix} 3 & 7 \\ 6 & 12 \end{bmatrix}$$

Step 1: Compute Similarity Scores then apply Softmax

We compute the dot product between the query vector $\boldsymbol{q}$ and each key vector $\boldsymbol{k}_i$:

$$\text{similarity}_1 = \boldsymbol{q} \cdot \boldsymbol{k}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 2$$

$$\text{similarity}_2 = \boldsymbol{q} \cdot \boldsymbol{k}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 3$$

Next, we apply the softmax function to the similarity scores:

$$\text{attention}_1 = \frac{e^2}{e^2 + e^3} = \frac{7.389}{7.389 + 20.086} = \frac{7.389}{27.475} \approx 0.269$$

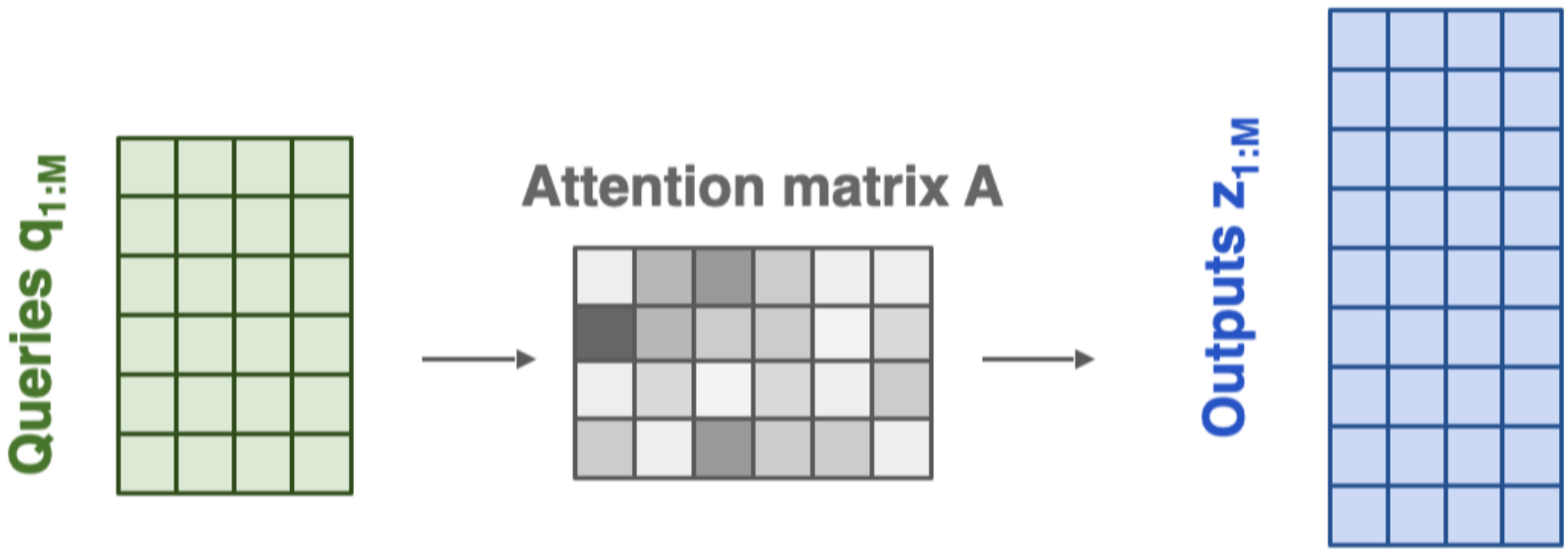
$$\text{attention}_2 = \frac{e^3}{e^2 + e^3} = \frac{20.086}{7.389 + 20.086} = \frac{20.086}{27.475} \approx 0.731$$

Step 2: Compute Weighted Sum of Values

$$
\begin{aligned}
A(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{attention}_1 \cdot \boldsymbol{v}_1 + \text{attention}_2 \cdot \boldsymbol{v}_2 \\
&= 0.269 \cdot \begin{bmatrix} 3 \\ 6 \end{bmatrix} + 0.731 \cdot \begin{bmatrix} 7 \\ 12 \end{bmatrix} \\
&= \begin{bmatrix} 0.807 \\ 1.614 \end{bmatrix} + \begin{bmatrix} 5.117 \\ 8.772 \end{bmatrix} \\
&= \begin{bmatrix} 5.924 \\ 10.386 \end{bmatrix}
\end{aligned}
$$

Matrix Notation: computing attention scores for multiple queries at once

Stacked queries $-->$ Attention matrix $-->$ Stacked outputs:



Attention matrix $= \text{softmax}\left(\boldsymbol{Q} \cdot \boldsymbol{K}^{\top}\right)$

Output $= \text{softmax}\left(\boldsymbol{Q} \cdot \boldsymbol{K}^{\top}\right) \cdot \boldsymbol{V}$

Attention Mechanism: Matrix Formulation

Stack queries in a matrix $\boldsymbol{Q}$: $[|Q| \times d_k]$ to compute attention matrix A, and then multiply by values $\boldsymbol{V}$:

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top\right) \cdot \boldsymbol{V}$$

$$[|Q| \times d_k] \times [d_k \times |K|] \times [|K| \times d_v]$$

Output: $|Q| \times d_v$

- Problem: scale of dot product increases as dimensions get larger
- Fix: scale by size of the vector

  Scaled dot-product :

$$\left(\boldsymbol{Q}\boldsymbol{K}^\top / \sqrt{d_k}\right)$$

Example Setup

Given:

- Query matrix $\boldsymbol{Q} \in \mathbb{R}^{|Q| \times d_k}$ (where $|Q| = 3$, $d_k = 2$):

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix}$$

- Key matrix $\boldsymbol{K} \in \mathbb{R}^{|K| \times d_k}$ (where $|K| = 2$, $d_k = 2$):

$$\boldsymbol{K} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

- Value matrix $\boldsymbol{V} \in \mathbb{R}^{|K| \times d_v}$ (where $|K| = 2$, $d_v = 2$):

$$\boldsymbol{V} = \begin{bmatrix} 4 & 8 \\ 6 & 12 \end{bmatrix}$$

Step 1: Compute Attention Weights

$$\boldsymbol{Q} \cdot \boldsymbol{K}^\top = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 \\ 3 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\text{attention}_{1,1} = \frac{e^1}{e^1 + e^0} \approx 0.731, \quad \text{attention}_{1,2} = \frac{e^0}{e^1 + e^0} \approx 0.269$$

$$\text{attention}_{2,1} = \frac{e^3}{e^3 + e^1} \approx 0.880, \quad \text{attention}_{2,2} = \frac{e^1}{e^3 + e^1} \approx 0.120$$

$$\text{attention}_{3,1} = \frac{e^1}{e^1 + e^1} = 0.5, \quad \text{attention}_{3,2} = \frac{e^1}{e^1 + e^1} = 0.5$$

$$\text{softmax}(\boldsymbol{Q} \cdot \boldsymbol{K}^\top) = \begin{bmatrix} 0.731 & 0.269 \\ 0.880 & 0.120 \\ 0.5 & 0.5 \end{bmatrix}$$

Step 2: Compute Weighted Sum of Values

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{Q} \cdot \boldsymbol{K}^\top) \cdot \boldsymbol{V}$$

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \begin{bmatrix} 0.731 & 0.269 \\ 0.880 & 0.120 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 4 & 8 \\ 6 & 12 \end{bmatrix}$$

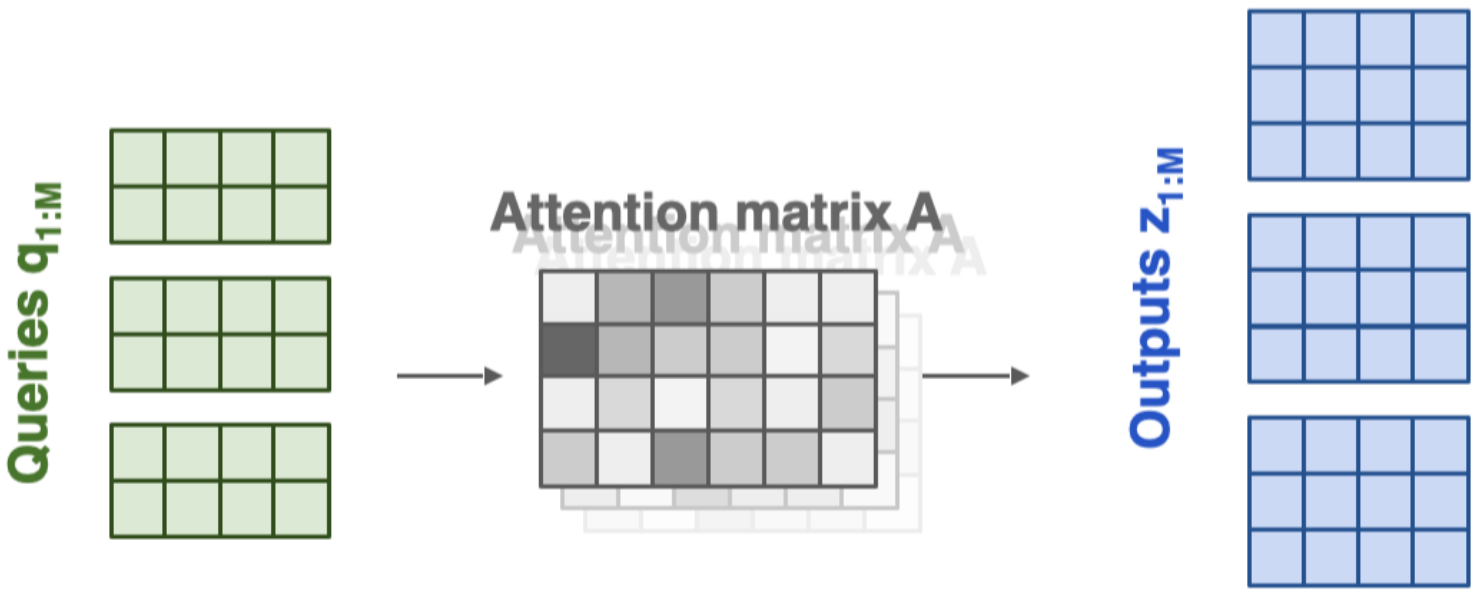
$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \begin{bmatrix} 0.731 \cdot 4 + 0.269 \cdot 6 & 0.731 \cdot 8 + 0.269 \cdot 12 \\ 0.880 \cdot 4 + 0.120 \cdot 6 & 0.880 \cdot 8 + 0.120 \cdot 12 \\ 0.5 \cdot 4 + 0.5 \cdot 6 & 0.5 \cdot 8 + 0.5 \cdot 12 \end{bmatrix}$$

$$= \begin{bmatrix} 4.538 & 9.076 \\ 4.240 & 8.480 \\ 5 & 10 \end{bmatrix}$$

Thus, the output $A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \in \mathbb{R}^{|Q| \times d_v}$, where $|Q| = 3$ and $d_v = 2$.

Multiple Heads

Repeat the process with different linear projections of Q, K, V:



Results are concatenated along the feature dimension.

Multi-Head Attention: Matrix Formulation

For each head:

$$A(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right) \cdot \boldsymbol{V}$$

- Each head computes: $\left[|Q| \times \frac{d_k}{h}\right] \times \left[\frac{d_k}{h} \times |K|\right] \times \left[|K| \times \frac{d_v}{h}\right]$
- Output per head: $|Q| \times \frac{d_v}{h}$

Concatenate the heads: h heads, concatenate their outputs:

$$\boldsymbol{O} = \text{Concat}(\boldsymbol{O}_1, \boldsymbol{O}_2, \ldots, \boldsymbol{O}_h)$$

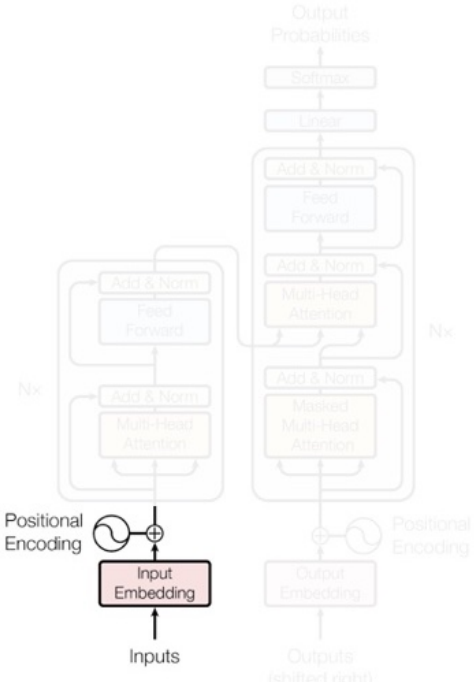
Where each $\boldsymbol{O}_i \in \mathbb{R}^{|Q| \times \frac{d_v}{h}}$.

Final output:

$$\boldsymbol{O} \in \mathbb{R}^{|Q| \times d_v}$$

Text Encoding in the Transformer

Input Processing



Input (Tokenization and) Embedding

Input text is first split into pieces. Can be characters, word, "tokens":

```
"The detective investigated" -> [The_] [detective_] [invest] [igat] [ed_]
```

Tokens are indices into the "vocabulary":

```
[The_] [detective_] [invest] [igat] [ed_] -> [3 721 68 1337 42]
```

Each vocab entry corresponds to a learned d_{model}-dimensional vector.

```
[3 721 68 1337 42] -> [ [0.123, -5.234, ...], [...], [...], [...], [...] ]
```

Positional Encoding

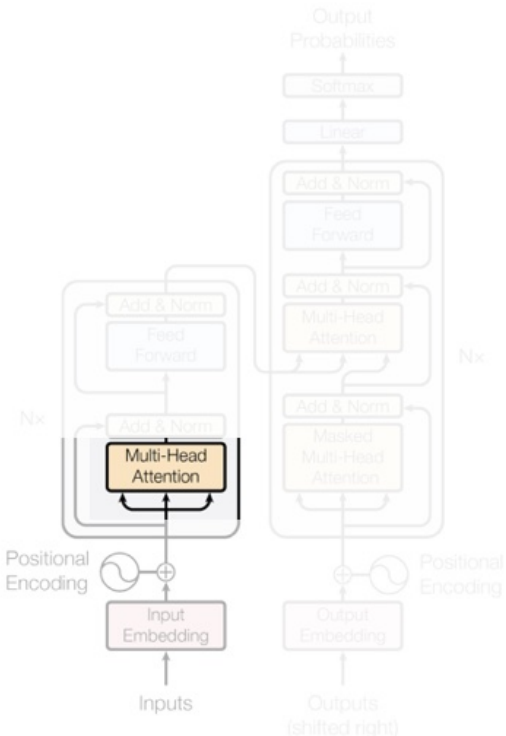
Remember attention is permutation invariant, but language is not!

("The mouse ate the cat" vs "The cat ate the mouse")

Need to encode position of each word; just add something.

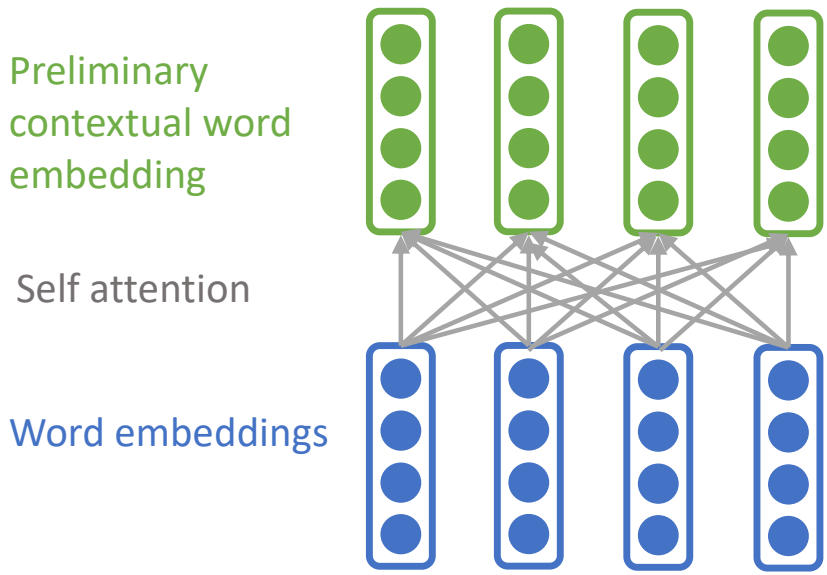
Think `[The_] + 10  [detective_] + 20  [invest] + 30` ... but smarter.

Attention: Communication Stage



Multi-headed Self-Attention

- Input sequence is used to create queries, keys, and values.
- Each token "looks around" the whole input, and decides how to update its representation based on what it sees.



Multi-Head Attention: Intuition

- Intuition: Information from different parts of the sentence can be useful to disambiguate in different ways.

  `I` **run** `a small business`

  `I` **run** `a mile in 10 minutes`

  `The robber made a` **run** `for it`

  `The stocking had a` **run**

- Each head can learn a different way to disambiguate (e.g., based on semantics, syntax, etc.)

Multi-Head Attention: Intuition

- Intuition: Information from different parts of the sentence can be useful to disambiguate in different ways.
- Syntax (nearby context)
- Semantics (farther context)

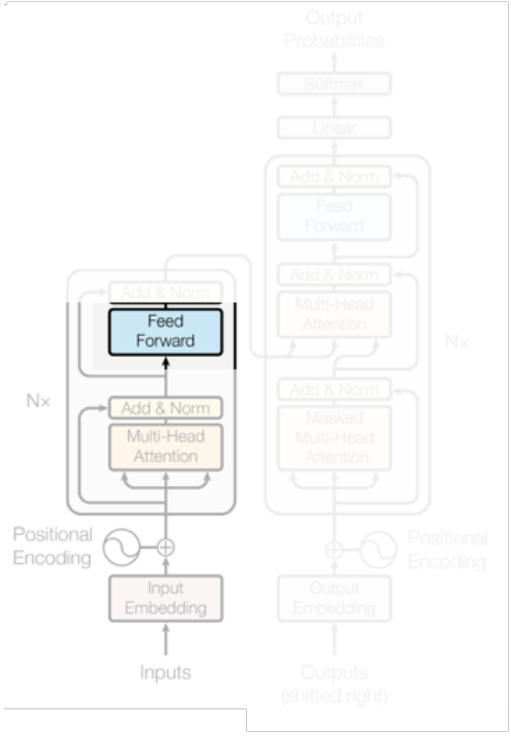
I **run** a small business

I **run** a mile in 10 minutes

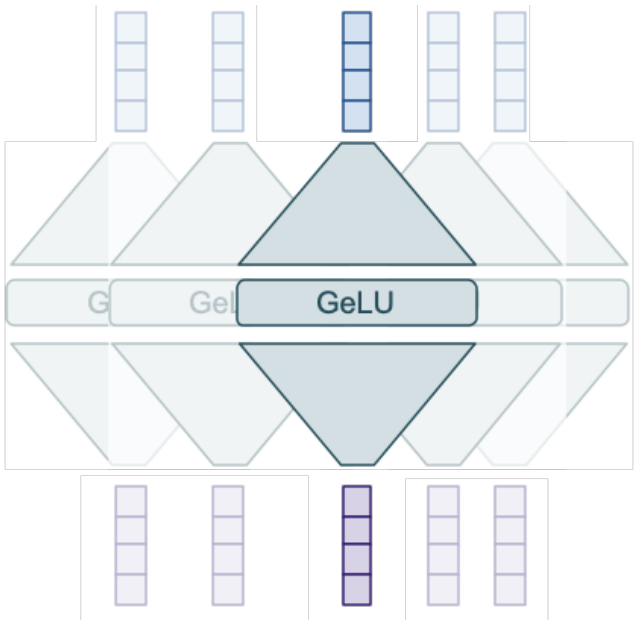
The robber made a **run** for it

The stocking had a **run**

Feedforward Layers: Position-wise

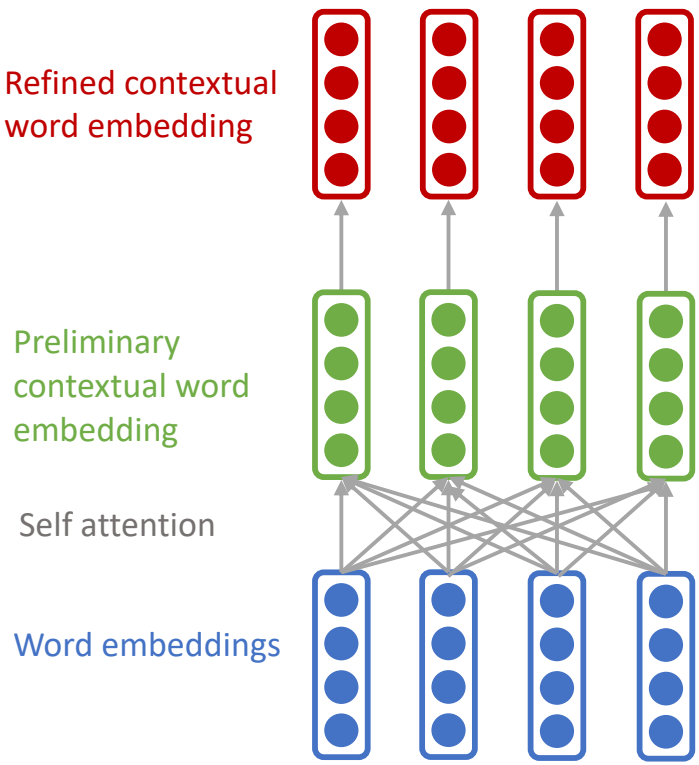


- A simple MLP applied to each token individually: $z_i = W_2\,\mathrm{GeLU}\,(W_1 x_i + b_1) + b_2$
- Bulk of the parameters in giant models, this is what becomes giant.

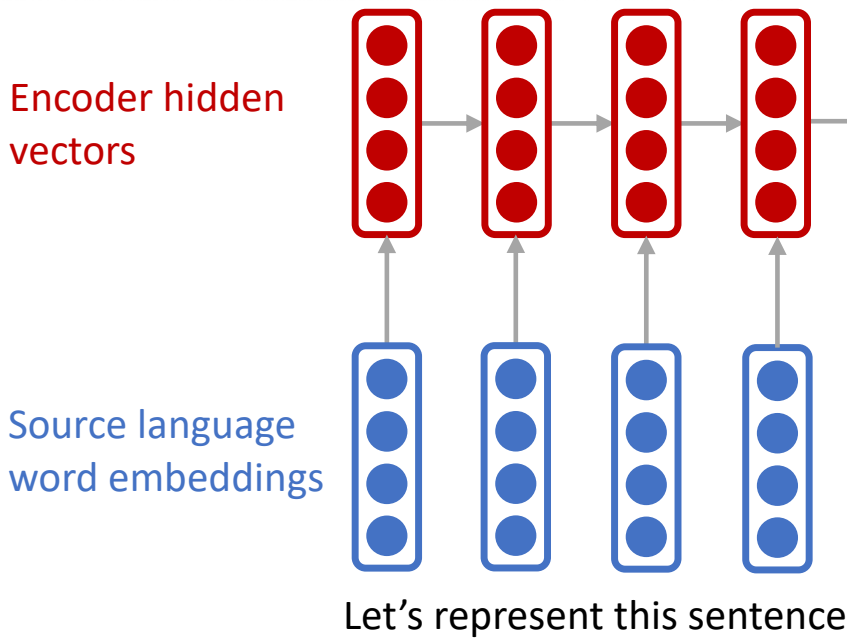


Encoder in a Transformer vs RNN

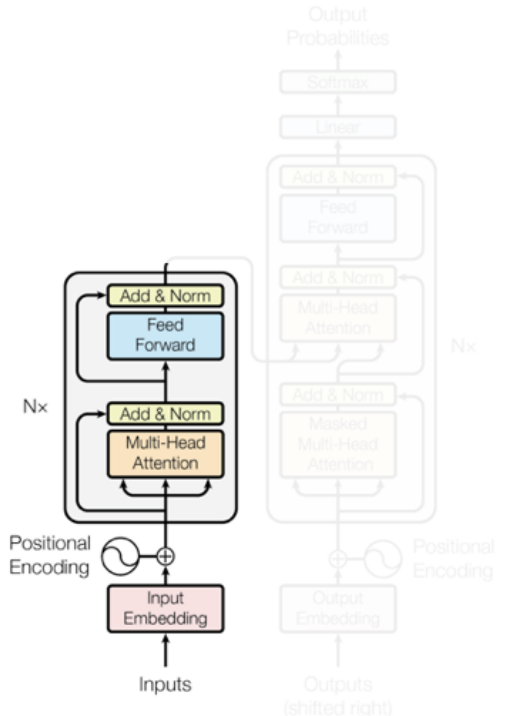
Transformer Encoder



RNN Encoder



The Encoder: Stacking Blocks

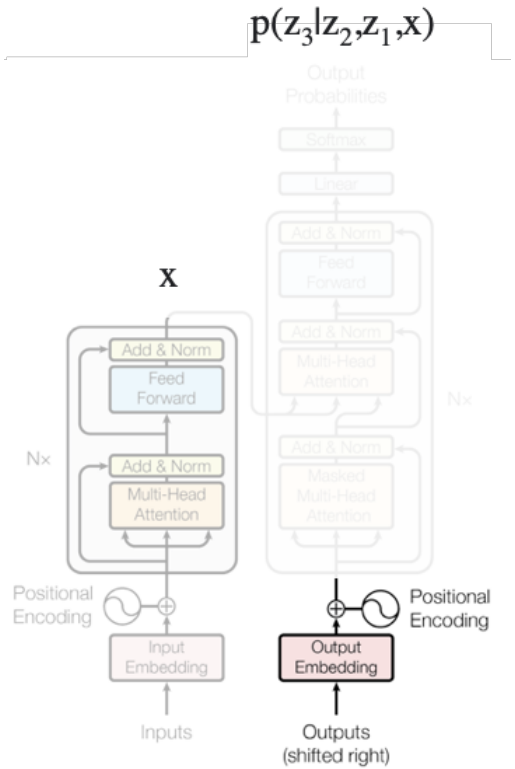


- Input & output shapes are identical, stack N such blocks.
- N=6 ("base"), N=12 ("large"), $>$ 100 ("LLMs").
- Encoder output is a heavily processed, contextualized version of the input tokens, i.e. a sequence.
- The encoder output is used by the decoder to generate the output sequence.

Decoding with the Transformer

Decoding

- Want to model: $p(z \mid x)$ decompose as:
  $p(z \mid x) = p(z_1 \mid x)\, p(z_2 \mid z_1, x)\, p(z_3 \mid z_2, z_1, x) \ldots$



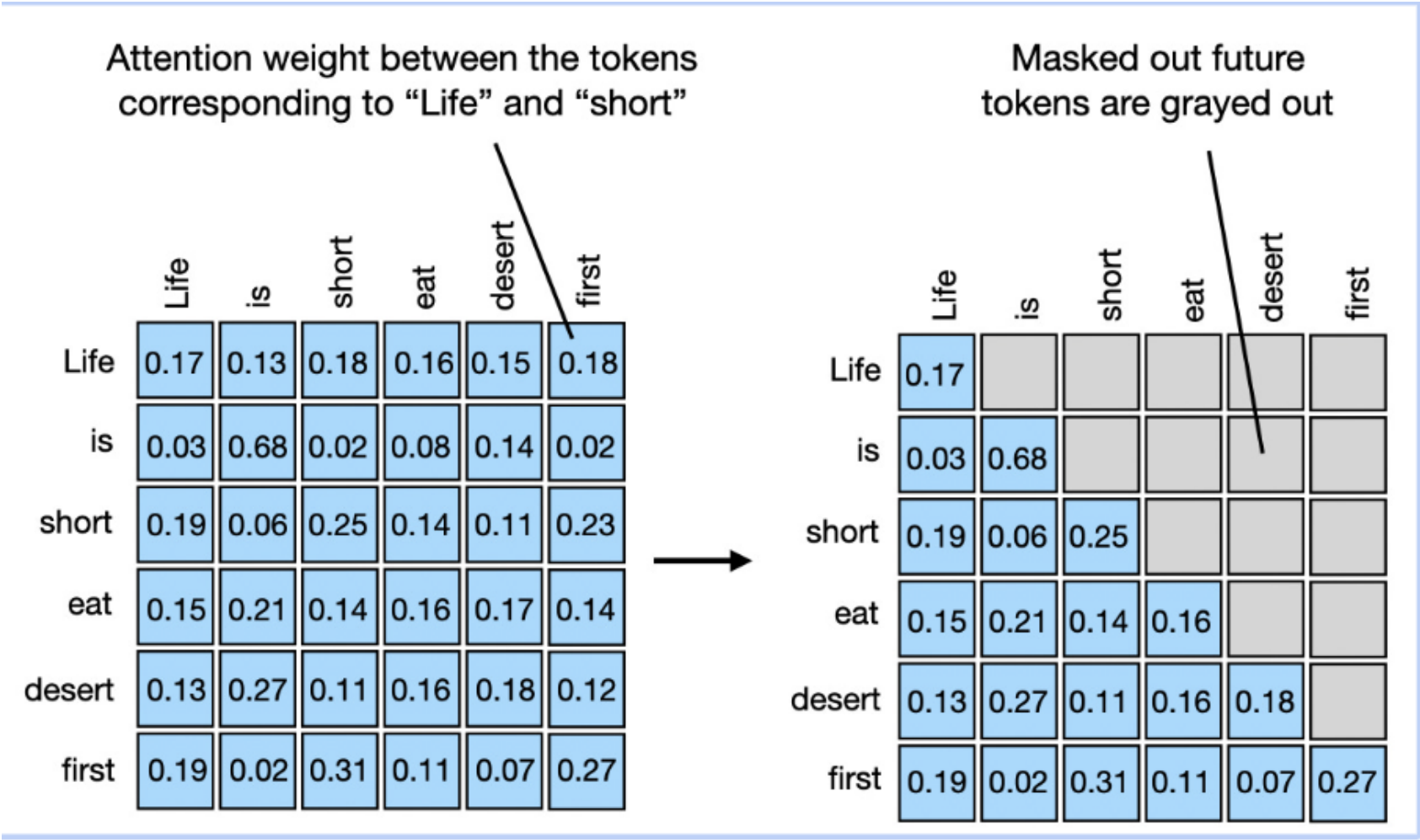
- Each p is a full pass through the model.
- For generating $p(z_3 \mid z_2, z_1, x)$:
  - x comes from the encoder,
  - z_1, z_2 is what we have predicted so far, goes into the decoder.
- Given: $p(z \mid x)$, still need to sample a sentence. Many strategies: greedy, beam-search, ...

Masked Self-Attention: Motivation

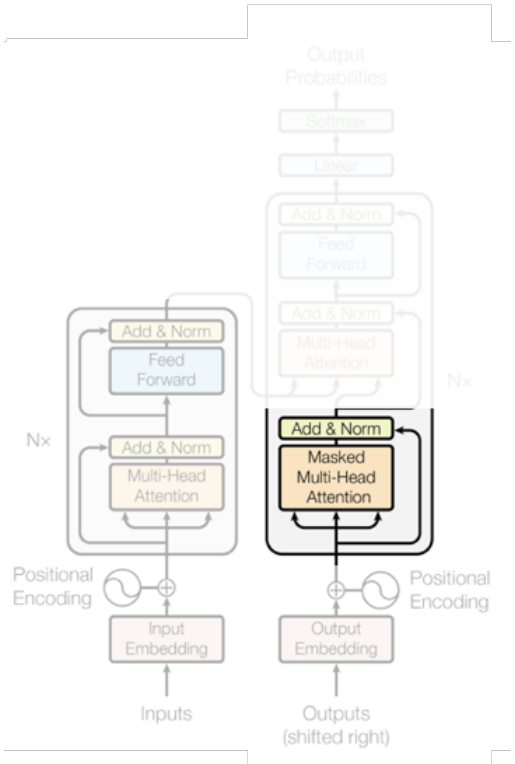
In language modeling, we want to predict the next token given the previous ones:

- Given a sentence from the training data: "The cat sat on the mat" we want to predict the next word for each word in the sentence:
  - $x =$ "The" ; $y =$ "cat"
  - $x =$ "The cat" ; $y =$ "sat"
  - ...
  - $x =$ "The cat sat on the" ; $y =$ "mat"
- We can train on all these examples simultaneously.

Masked Self-Attention: Motivation



Masked self-attention



At training time: Masked self-attention

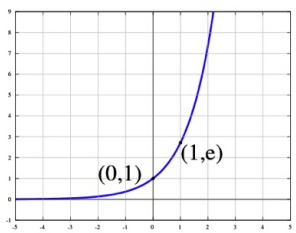
- If we had to train on one single $p\left(z_3 \mid z_2, z_1, x\right)$ at a time: SLOW! Instead, train on all $p\left(z_i \mid \boldsymbol{z}_{1:i}, x\right)$ simultaneously.
- How? Mask out the future tokens in the input sequence. In the attention weights for z_i, set all entries $i : N$ to 0 . Each token only sees the already generated ones.

At generation time:

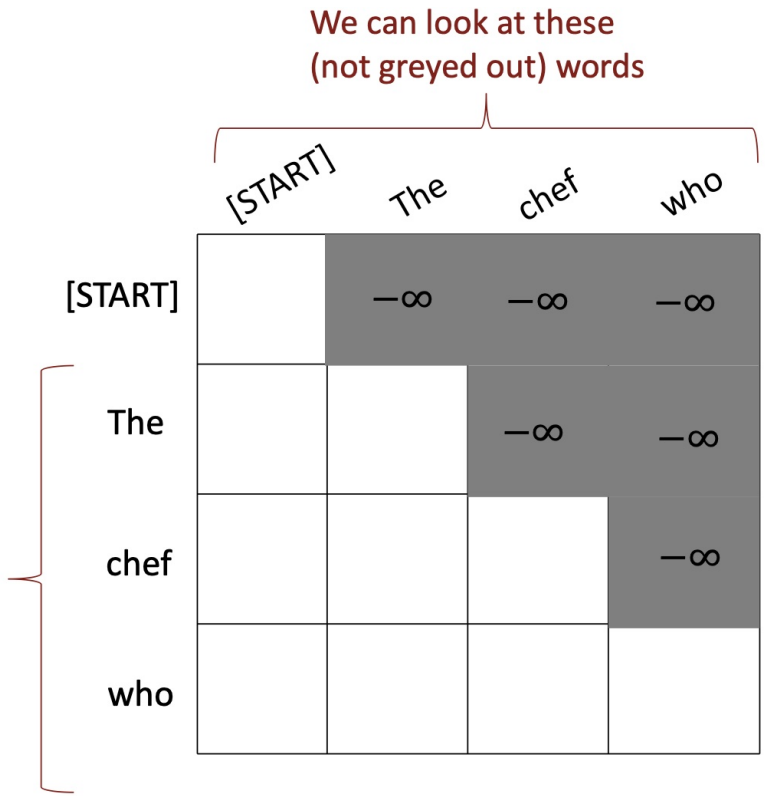
- There is no such trick. Must generate one z_i at a time. Hence, autoregressive decoding is really slow.

Attention Masking Details

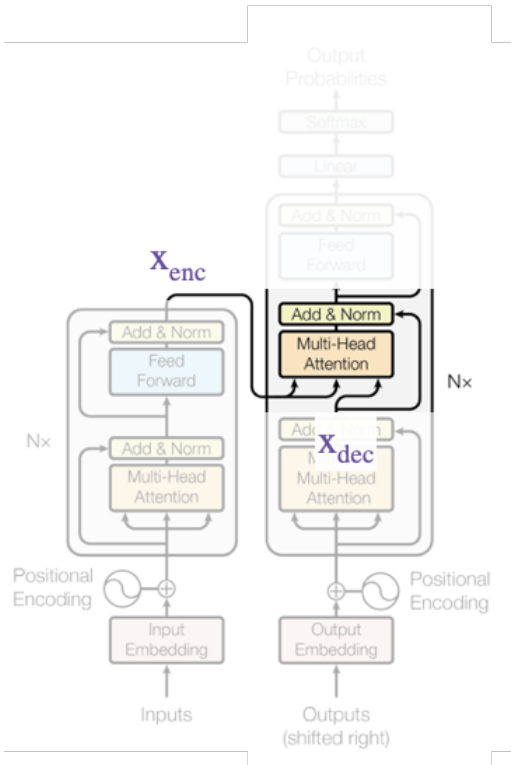
- We want to mask with upper triangular matrix. In PyTorch use: `torch.triu`
- Setting **attention score** to $-\infty$ ensures that attention probability will be zero.



$$e_{ij} = \begin{cases} q_i^\top k_j, j \leq i \\ -\infty, j > i \end{cases}$$

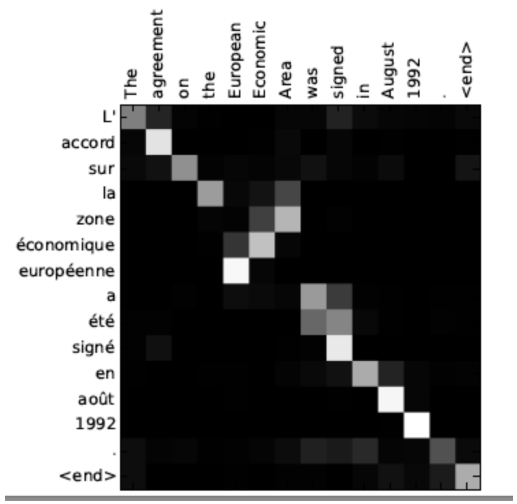


Encoder-Decoder Attention (Cross Attention)

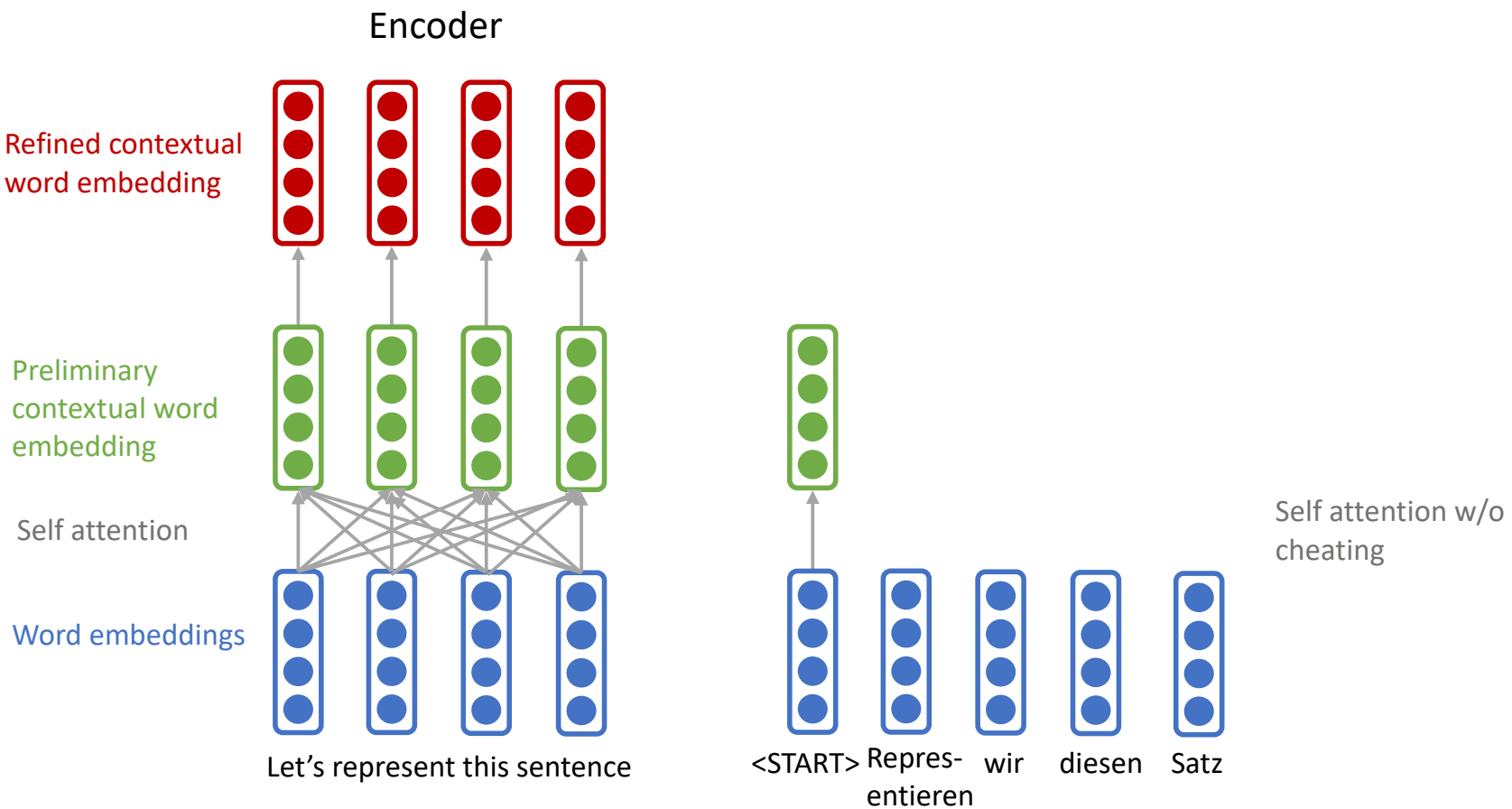


Encoder-Decoder Attention

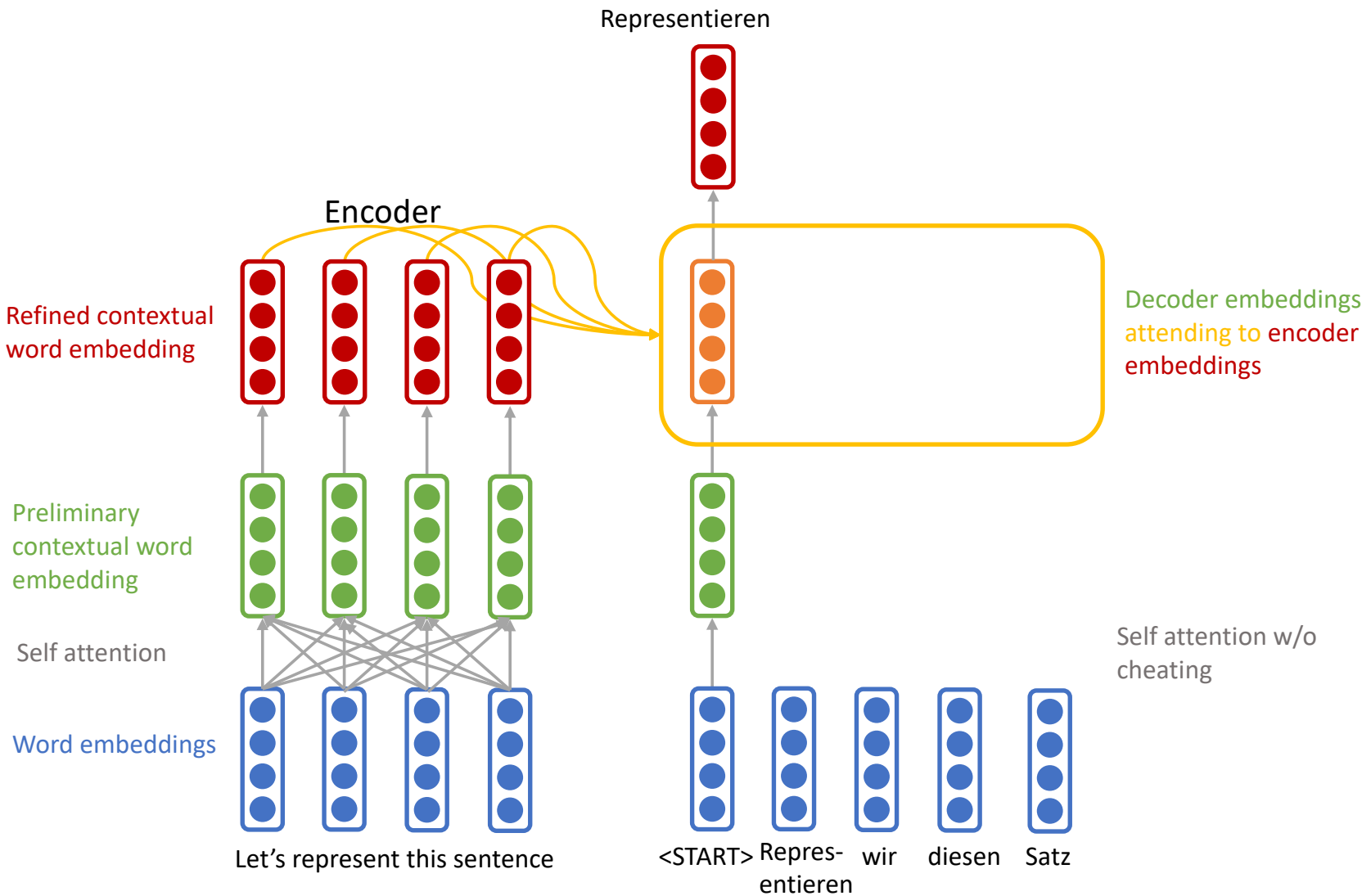
- Each decoded token can "look at" the encoder's output (as in seq2seq w/ attention)
- This allows the decoder to focus on relevant parts of the encoder's output.



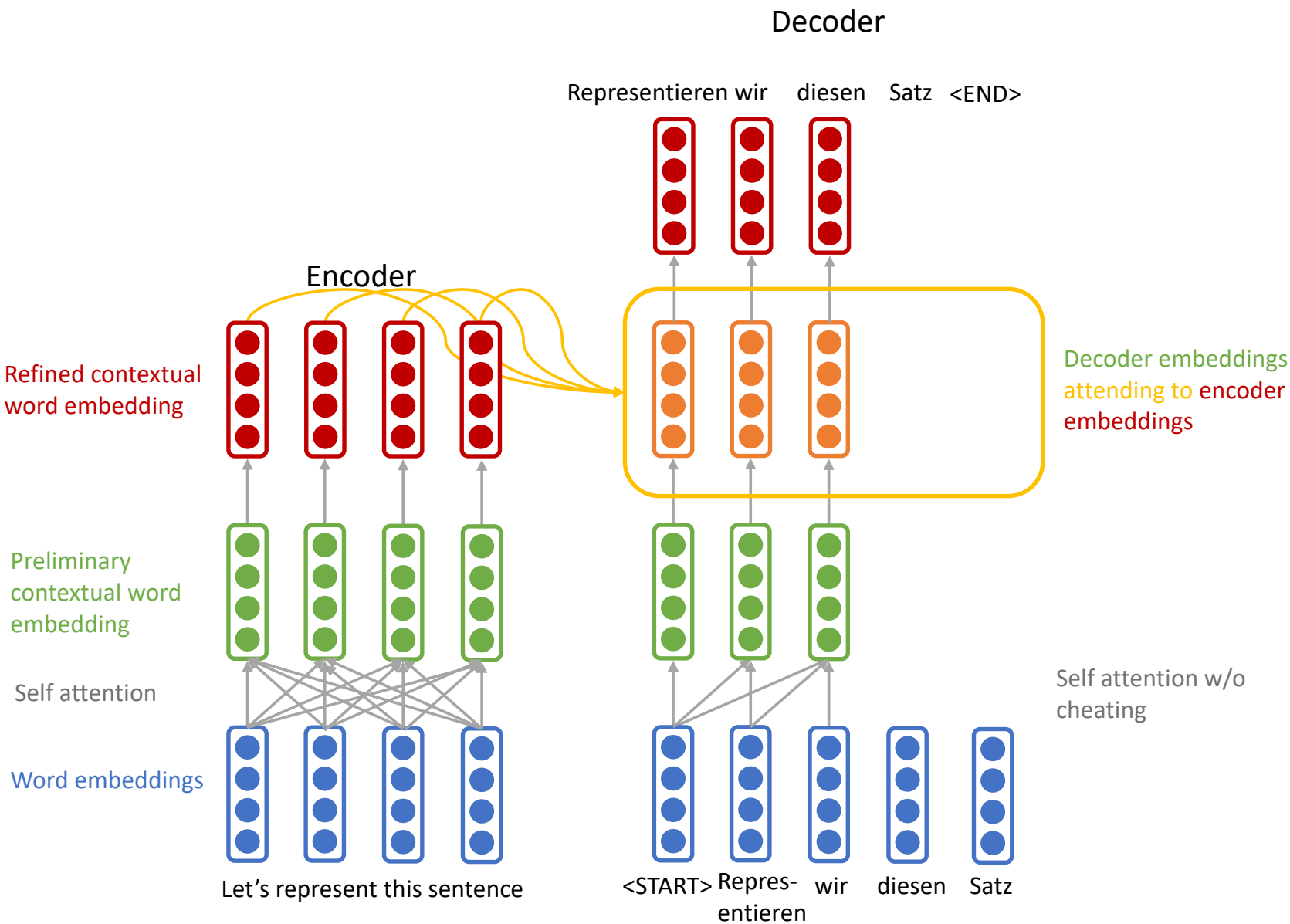
Transformer Encoder-Decoder



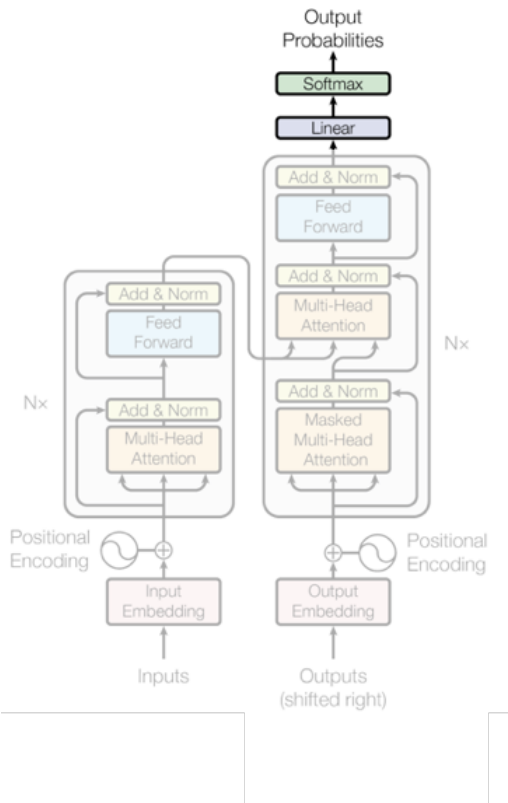
Transformer Encoder-Decoder



Transformer Encoder-Decoder



Output Layer

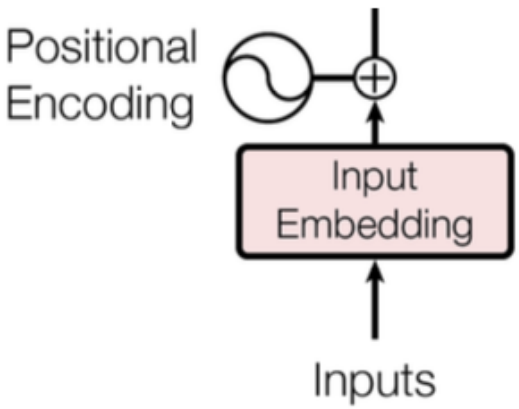


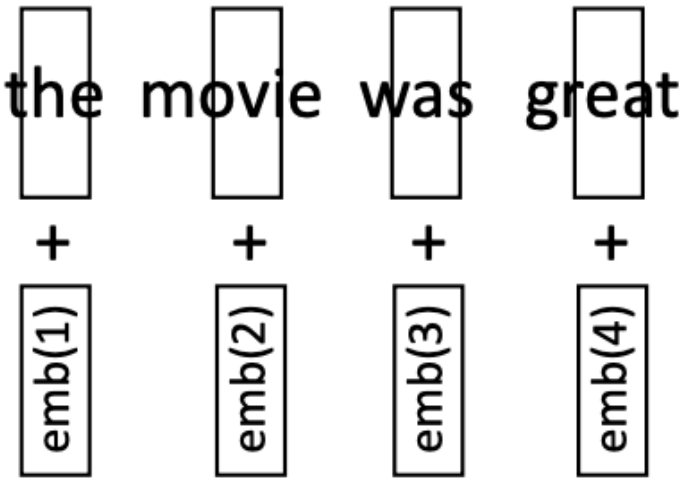
- The output layer converts the decoder output into a probability distribution over the vocabulary.
  - The output layer is a linear layer with a softmax activation.
- (Greedy decoding): The word with the highest probability is the predicted next word in the output sequence.

Positional Encoding

Position Sensitivity

"*the cat sat on the mat*"





- Encode each sequence position as an integer, add its embedding to the word embedding vector
- Seperate embedding matrix from the word embeddings

Using Trigonometric Functions for Positional Encoding

- Vaswani et al 2017: Calculate each dimension with sines/cosines of different frequencies

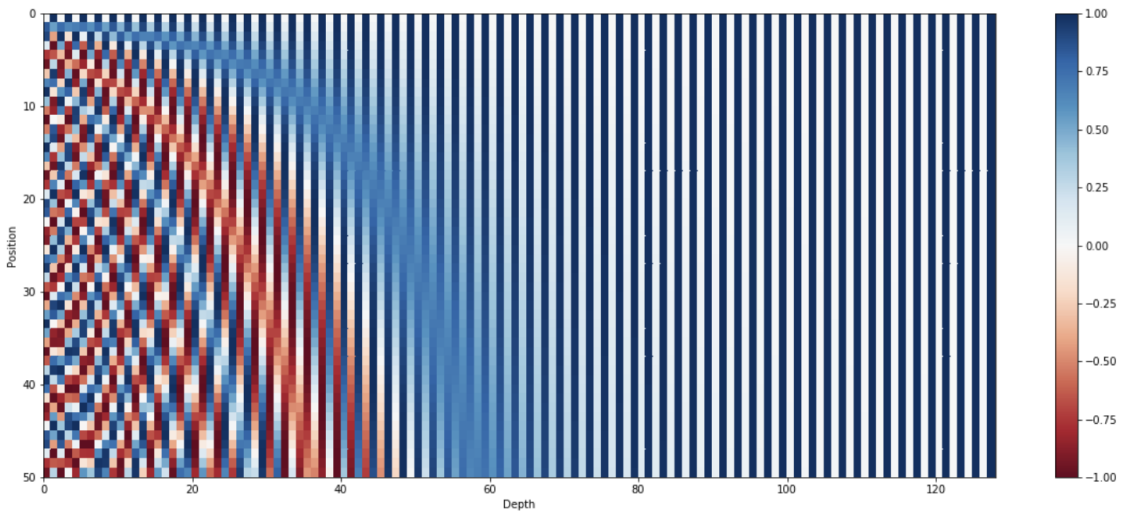


Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$

Why? closer words **get higher dot products**

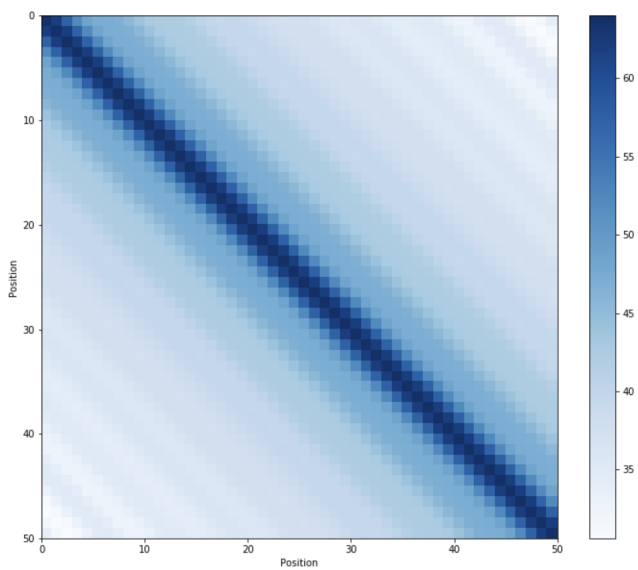


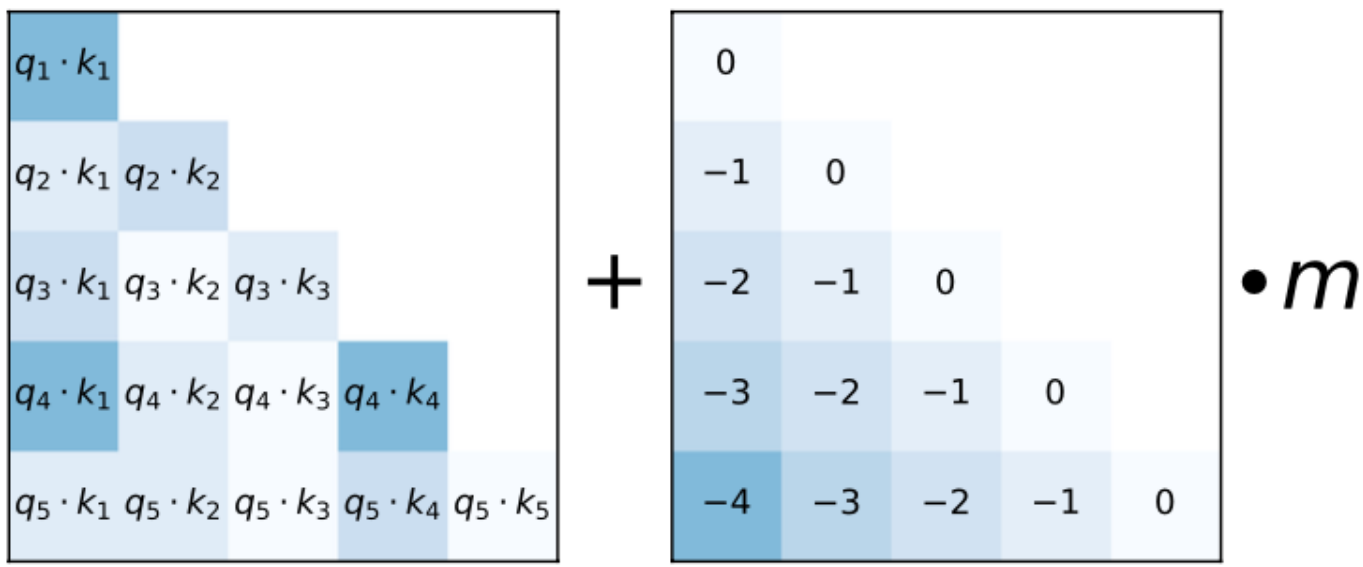
Figure 3 - Dot product of position embeddings for all time-steps

Illustrations, Kazemnejad, 2019

Absolute vs Relative Positional Encodings

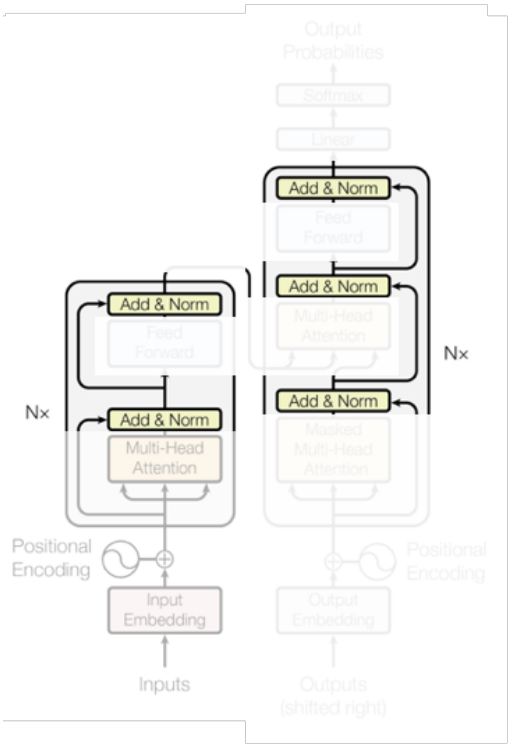
- **Absolute**: add an encoding to the input in **hope** that relative position will be captured
  - Vaswani et al. 2017: Positional Encodings
  - Fails to extrapolate to longer sequences

- **Relative**: **Explicitly** encode relative position e.g., "I" is 3 words away from "run"
  - Press et al. 2021: "Attention with Linear Biases" (ALiBi), **parameter-free** linear bias added to dot-product similarity. Extrapolates to longer sequences.



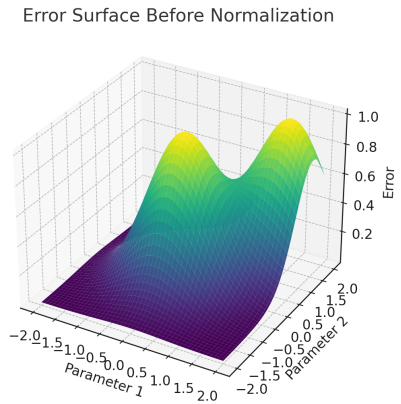
Training Transformers - Optimization Tricks

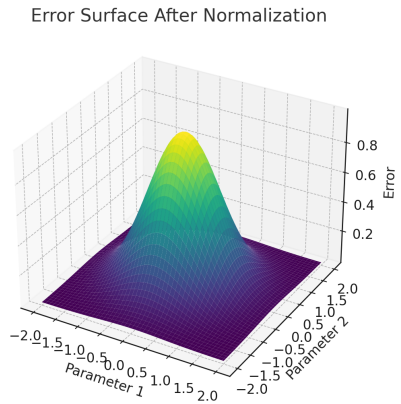
Residual Connections and Layer Normalization



Residual connections: training converges faster, and allows for deeper networks.

Layer normalization: Normalizes the outputs to be within a consistent range, preventing too much variance in scale of outputs





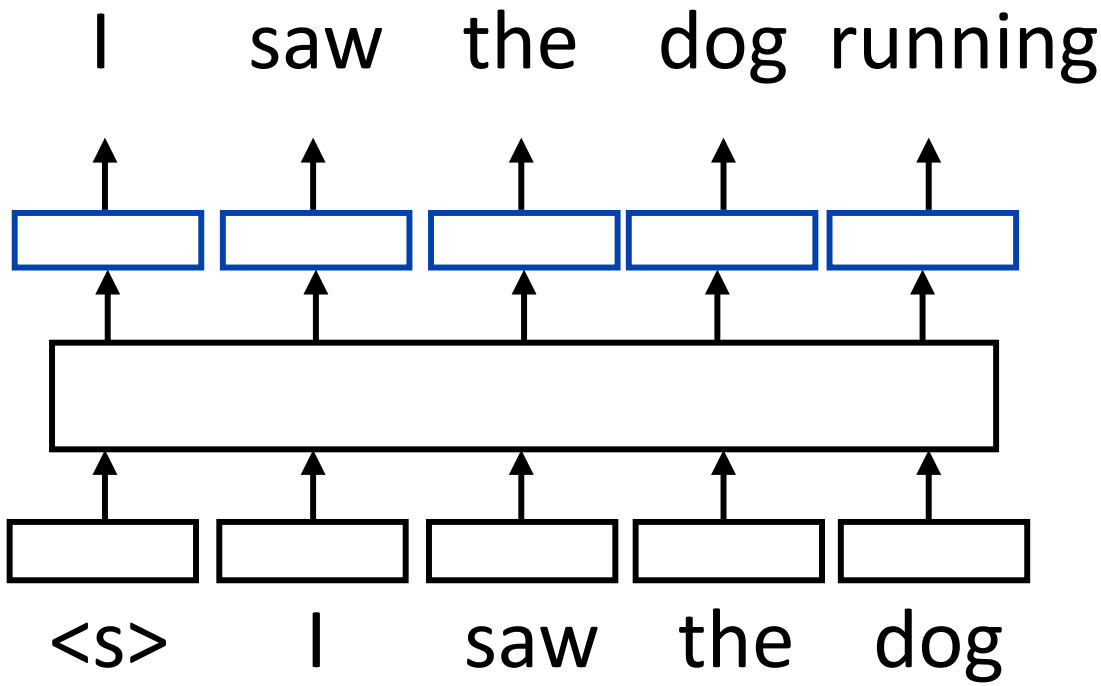
Transformer Summary and Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

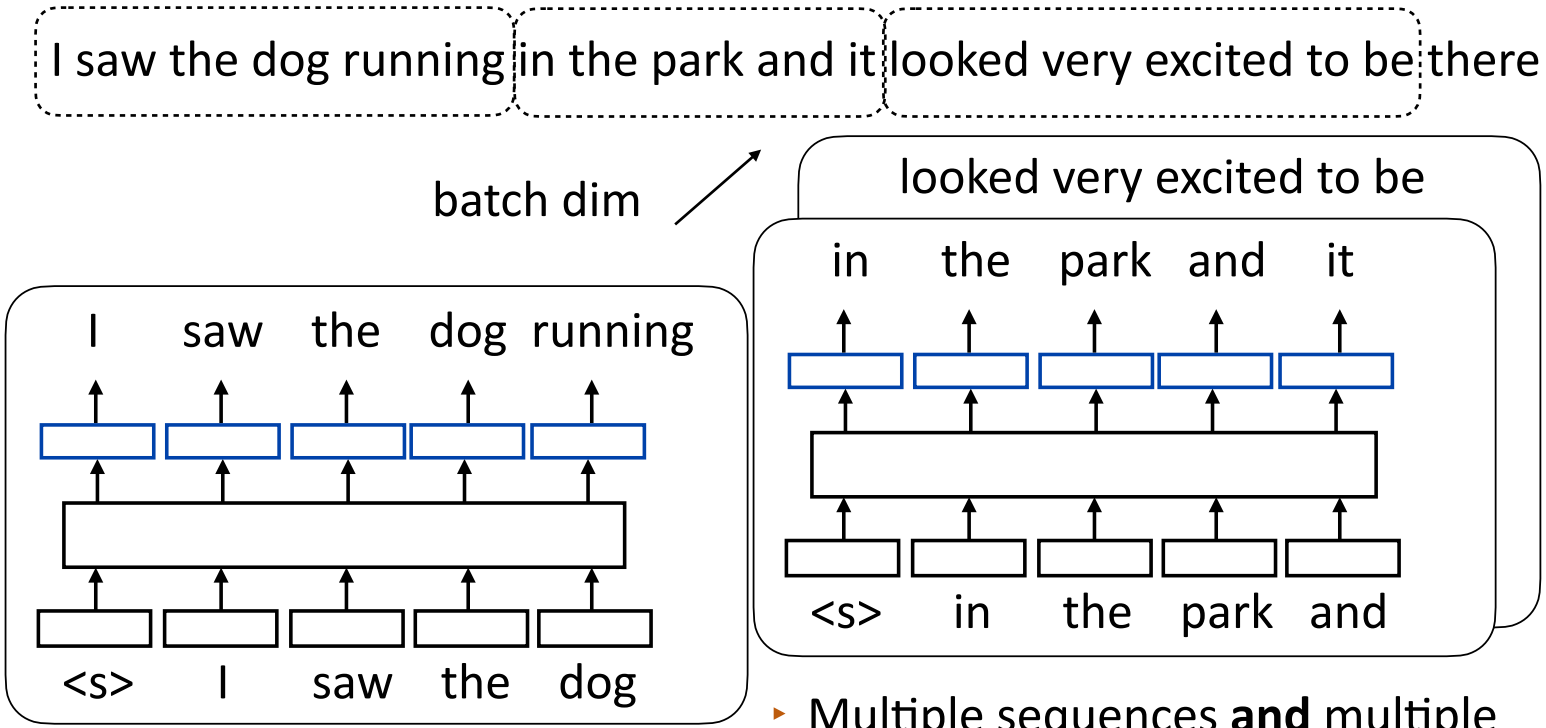
10^{21} is 100 times larger than 10^{19}. If something takes 40 days in the 10^{21} regime, it would take 0.4 days (or about 9.6 hours) in the 10^{19} regime.

Transformer Language Modeling: Training



- Input is a sequence of words, output is those words shifted by one,
- Allows us to train on predictions across several timesteps simultaneously (similar to batching but this is NOT what we refer to as batching)

Batched LM Training



- Multiple sequences **and** multiple timesteps per sequence

What happens with Multi-Head Attention?

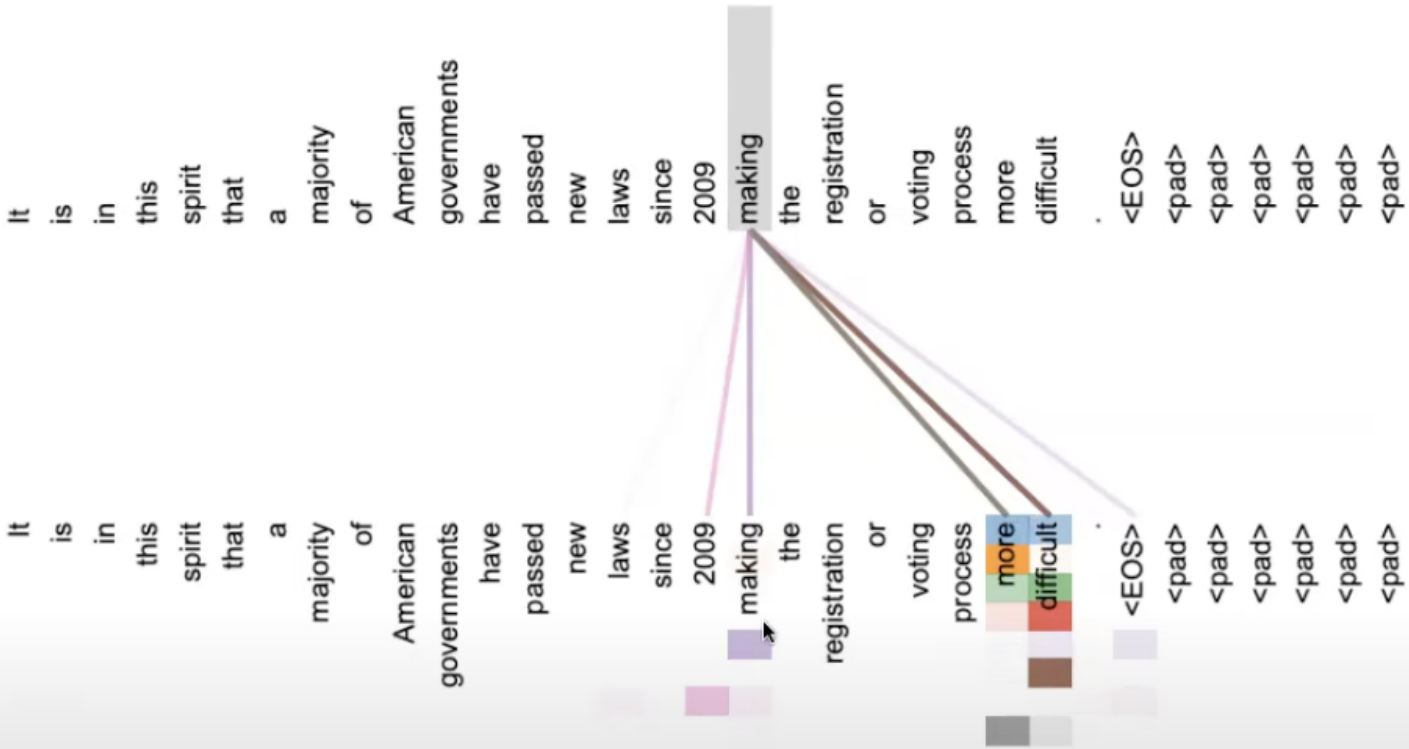


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Sequence Modeling

Challenges with RNNs

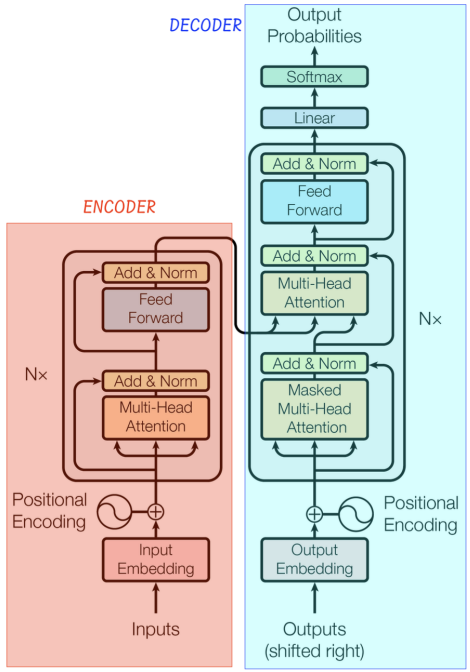
- Gradient vanishing and explosion
  - Long-range dependencies are not captured
- Recurrence prevents parallel computation
  - Slow training

Transformer Networks

- Facilitate long range dependencies
- No gradient vanishing and explosion
- No recurrence: facilitate parallel computation

The Transformer: versatile architecture

Domain	Data Type
Natural Languages	Text
Programming Languages	Code
Computer Vision	Images
Speech	Audio
Proteins Folding	Protein Sequences
Music	Audio
Video	Video
...	...



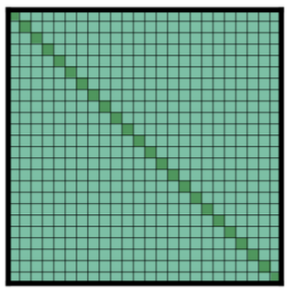
Takeaways

- Transformers are going to be the foundation for the much of the rest of this class and are a ubiquitous architecture nowadays
- Many details to get right, many ways to tweak and extend them, but core idea is the multi-head self attention and their ability to contextualize items in sequences

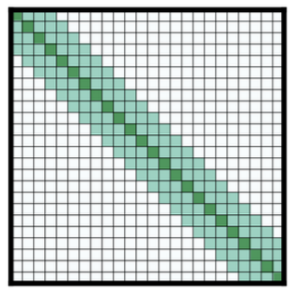
Transformer Runtime

- Even though most parameters and FLOPs are in feedforward layers, Transformers are still limited by quadratic $O(n^2)$ complexity of self-attention
- Quadratic complexity, but O(1) sequential operations (not linear like in RNNs) and O(1) "path" for words to inform each other ... so it's still fast
- Many tricks to reduce this complexity, e.g., Longformer, Reformer, Linformer, ...

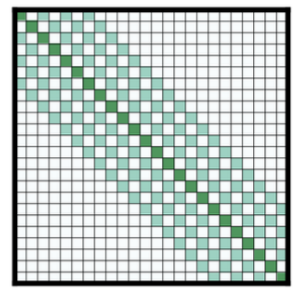
Longformer



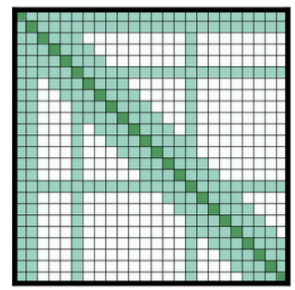
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



(d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer

- Use several pre-specified self-attention patterns that limit the number of operations
- Scales to 4096 -length sequences
- Engineering-based approaches like Flash Attention (which supports the "basic" Transformer) have superseded changing the Transformer model itself

---

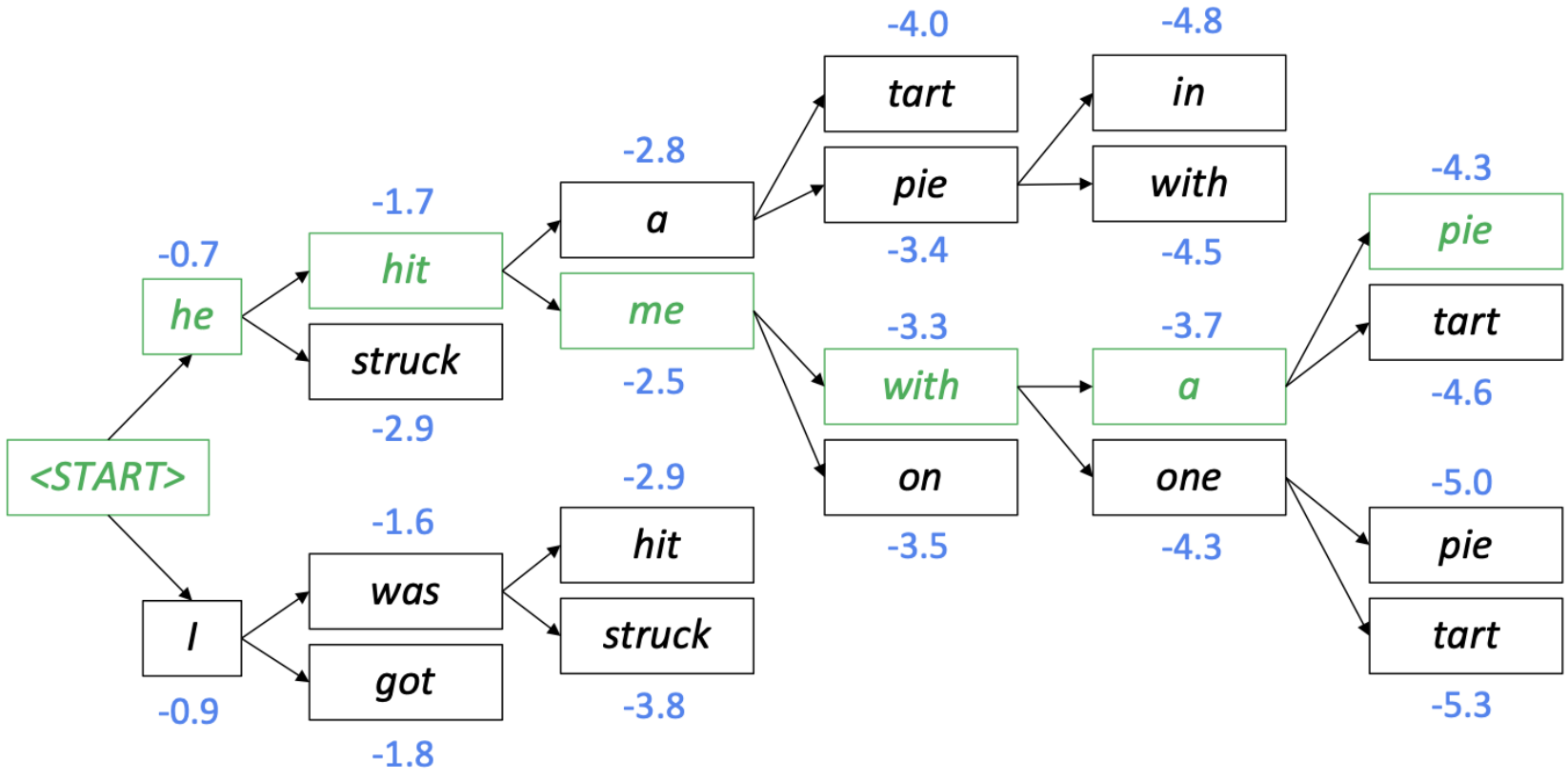
Beltagy et al. (2021)

Decoding Strategies

Method: Beam Search

Recall Example

Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



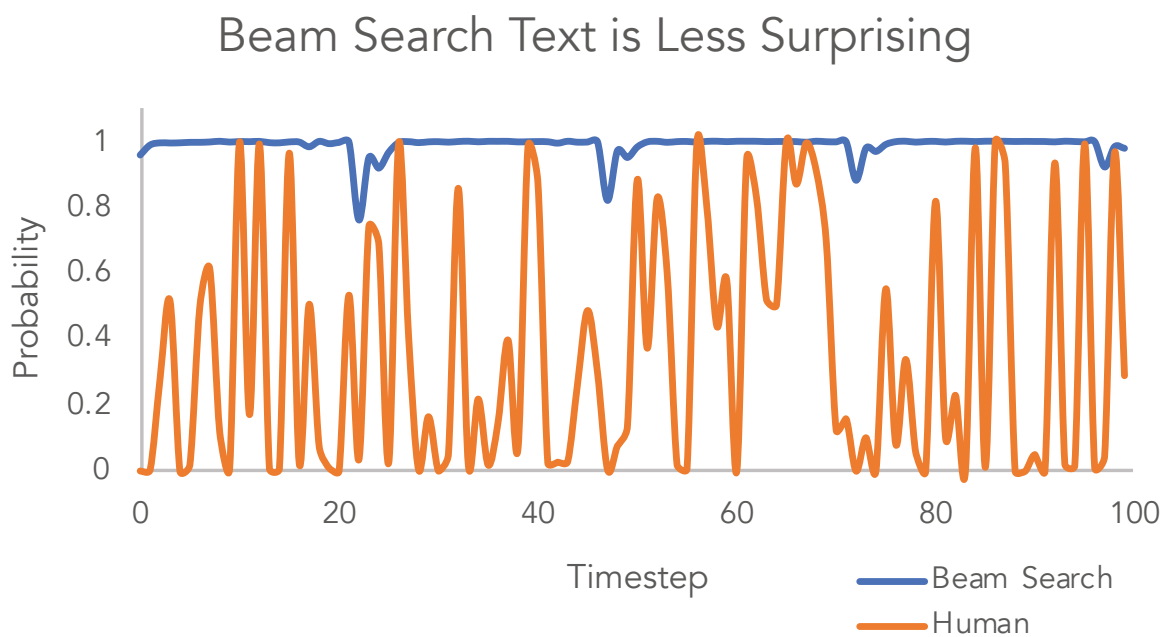
Backtrack to obtain the full hypothesis

Problems with Beam Search

Holtzman et al., 2019: "**The Curious Case of Neural Text Degeneration**"

- Maximization-based decoding (e.g., **beam search**) can cause **degeneration**
  - LMs assign high scores to well-formed text, yet
  - The highest scores for long texts can be **generic, repetitive, and incoherent**

.



Directed Text Generation (vs. Open-Ended)

- Directed text generation: Tasks have **(input, output)** pairs where the output is a transformation of the input:
  - **Machine translation**
  - **Data-to-text generation**
  - **Summarization**
- Generation is usually performed using Beam Search
  - Output is determined by the input, repetition and genericness are not as problematic.

Open-Ended Generation

- Open-ended generation: e.g. **conditional story generation** and **text continuation**.
- Generation has significant freedom in terms of what can plausibly follow.
- Some tasks are both open-ended and directed generation
  - **Goal-oriented dialogue**, e.g., conversation with a chatbot to book a flight

Open-Ended Text Generation

- Given a sequence of m context tokens, $x_1, \ldots, x_m$, the goal is to generate the next n continuation tokens, producing a completed sequence: $x_1, \ldots, x_m, x_{m+1}, \ldots, x_{m+n}$

$$P\left(x_{1:m}, x_{m+1:m+n}\right) = \prod_{i=1}^{m+n} P\left(x_i \mid x_1, \ldots, x_{i-1}\right)$$

- Generation proceeds token-by-token using a chosen decoding strategy (e.g., greedy, beam search, nucleus sampling).

Method: Pure Sampling

- Sampling directly from model-predicted probabilities often leads to **incoherent** and **contextually unrelated** text.

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town, and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV reporters. We don't even stick around to be interviewed by TV reporters. Maybe that's how they figured out that they're cosplaying as the Bolivian Cavalleros."

- Issue: **unreliable tail** in the probability distribution contains thousands of low-probability tokens
  - These tokens are **over-represented**, disrupting coherence and relevance.

Challenges with Pure Sampling

- **Sampling**: too random, introduces many grammatical errors

Pure Sampling:
They were cattle called Bolivian Cavalleros; they live in a remote desert uninterrupted by town and they speak huge, beautiful, paradisiacal Bolivian linguistic thing. They say, 'Lunch, marge.' They don't tell what the lunch is," director Professor Chuperas Omwell told Sky News. "They've only been talking to scientists, like we're being interviewed by TV

P(*y* | ... they live in a remote desert uninterrupted by)

0.01 roads
0.01 towns
0.01 people
0.005 civilization

Good options, maybe accounting for 90% of the total probability mass. So a 90% chance of getting something good

...
0.0005 town

Long tail with 10% of the mass

Holtzman et al. (2019)

Method: Nucleus Sampling

A Stochastic Decoding Method using Top-p Vocabulary

- Given a probability distribution $P\left(x \mid x_{1:i-1}\right)$, the **top-$p$ vocabulary** $V^{(p)} \subset V$ is defined as the smallest set of tokens satisfying:

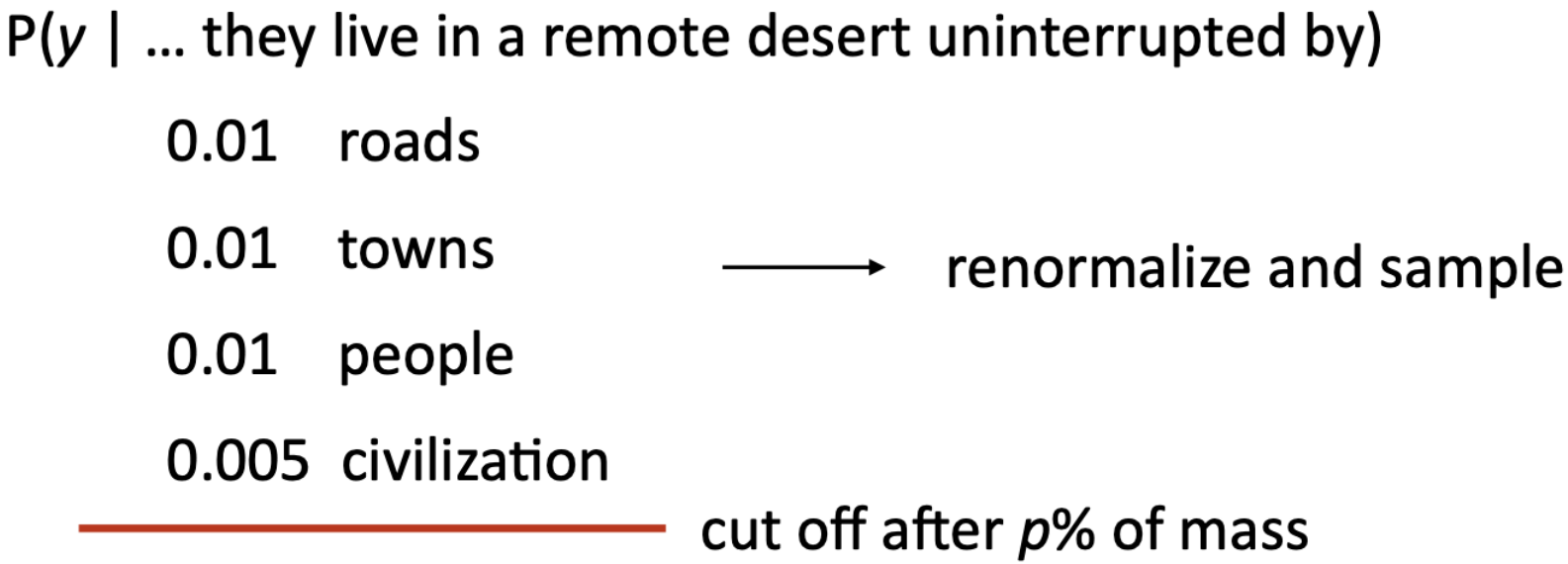
$$\sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right) \geq p$$

- Focus on (the nucleus) ensures sampling focuses on high-probability tokens while discarding the unreliable tail

- Let $p' = \sum_{x \in V^{(p)}} P\left(x \mid x_{1:i-1}\right)$. Original distribution is re-scaled to a new distribution, from which the next word is sampled:

$$P'\left(x \mid x_{1:i-1}\right) = \begin{cases} P\left(x \mid x_{1:i-1}\right) / p' & \text{if } x \in V^{(p)} \\ 0 & \text{otherwise} \end{cases}$$

Method: Nucleus Sampling

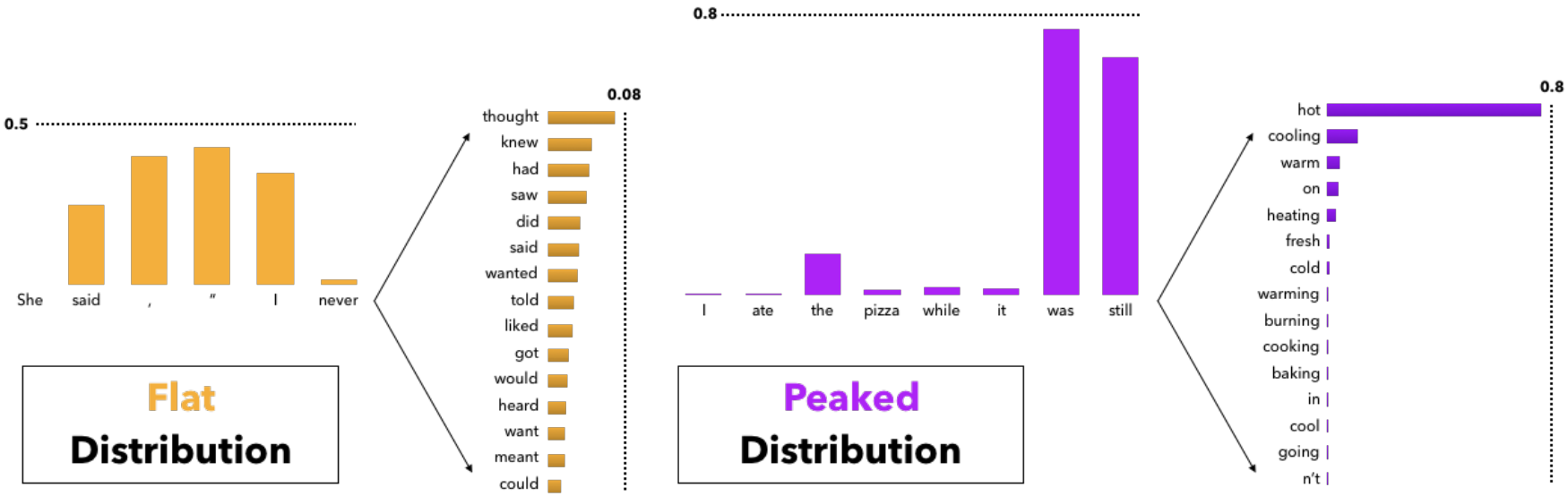


- Define a threshold p. Keep the most probable options account for $p\%$ of the probability mass (the nucleus), then sample among these
- To implement: sort options by probability, truncate the list once the total exceeds p, then renormalize and sample from it

Method: Top-k Sampling

- At each time step, top-k sampling considers only the k tokens with the highest probabilities, re-scaling these probabilities and sampling from them
- Both Top-k and Nucleus Sampling restrict sampling to a *trustworthy prediction zone*, but differ in truncation strategy

Top-k Sampling: Challenges with fixed k



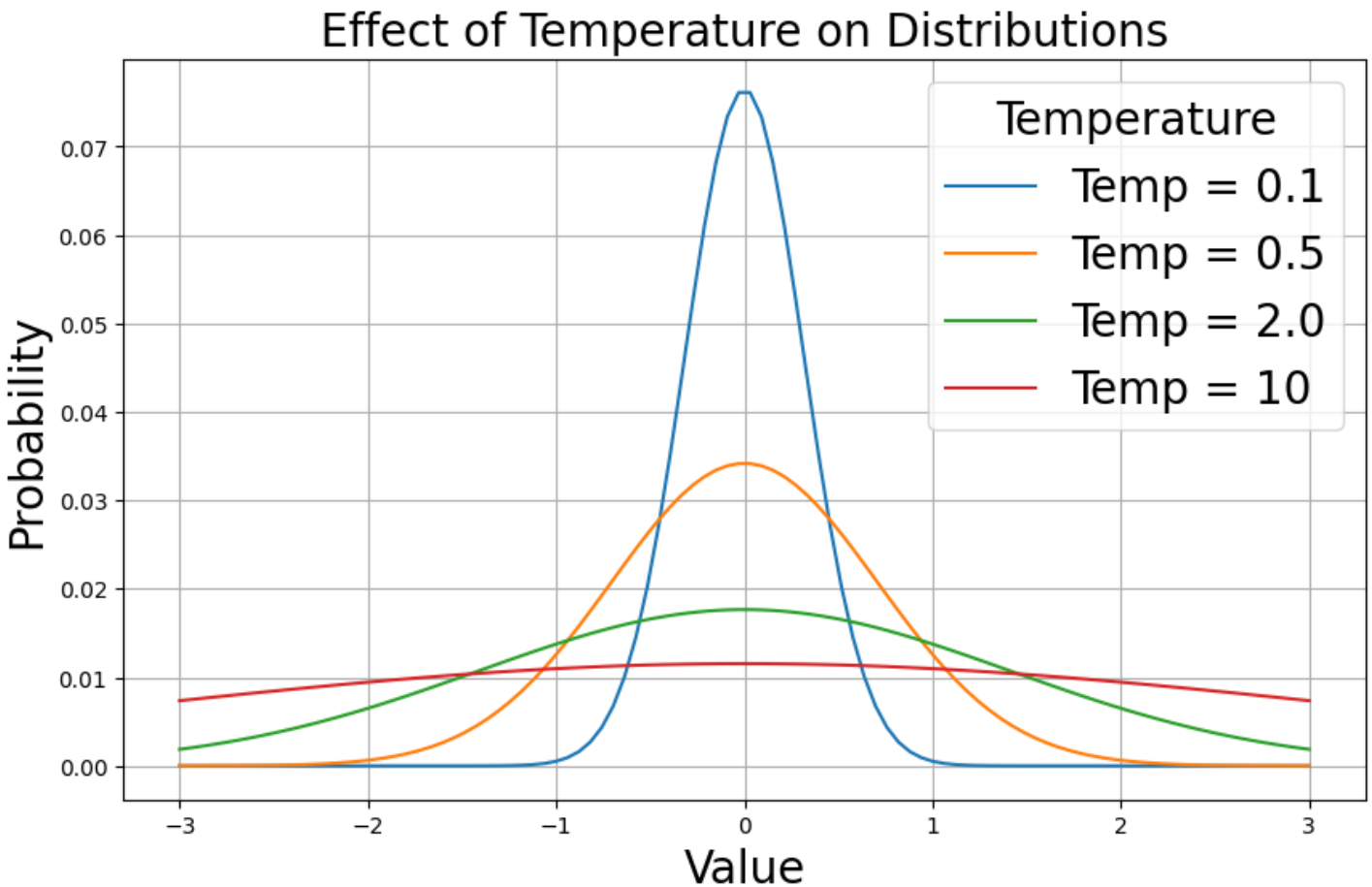
- The presence of flat distributions makes the use of a small k in top- k sampling problematic
- The presence of peaked distributions makes large k 's problematic.

Sampling with Temperature

High temperature: more randomness; **Low temperature**: less randomness

Given the scores $u_{1:|V|}$ and temperature t, adjust softmax as:

$$p\left(x = V_l \mid x_{1:i-1}\right) = \frac{\exp\left(u_l/t\right)}{\sum_{l'} \exp\left(u_{l'}/t\right)}$$



Sampling with Temperature

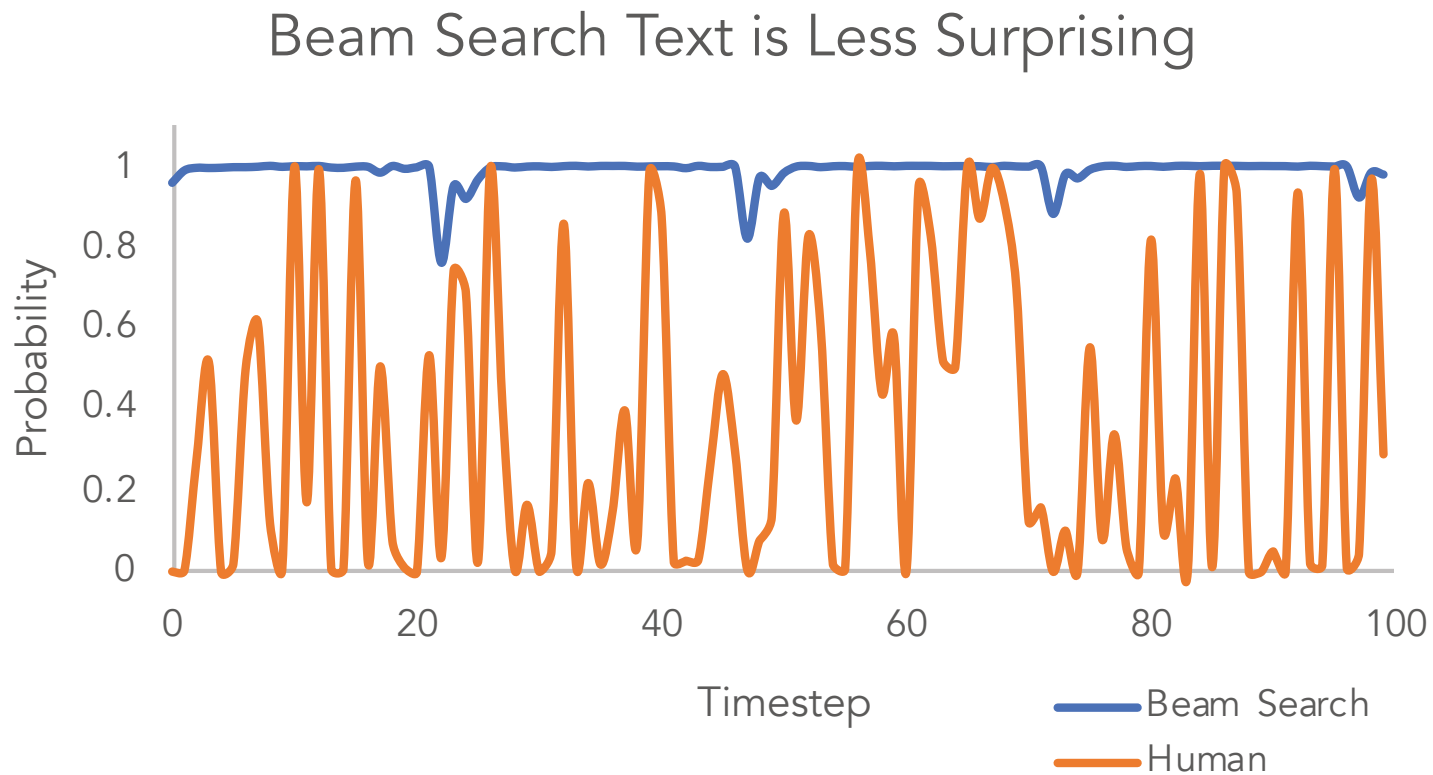
- Setting $t \in [0, 1)$ biases the distribution towards high probability events, reducing mass in the tail distribution.
- **Low temperature sampling** can address some issues of top-k sampling by pre-shaping the distribution (Radford et al., 2018; Fan et al., 2018).
- However, it was shown that lowering temperature improves quality but reduces diversity in generated text (Caccia et al., 2018; Hashimoto et al., 2019).

Decoding Strategies: Perplexity

Method	Perplexity
Human	12.38
Greedy	1.50
Beam, b=16	1.48
Stochastic Beam, b=16	19.20
Pure Sampling	22.73
Sampling, t=0.9	10.25
Top-k=40	6.88
Top-k=640	13.82
Top-k=40, t=0.7	3.48
Nucleus p=0.95	**13.13**

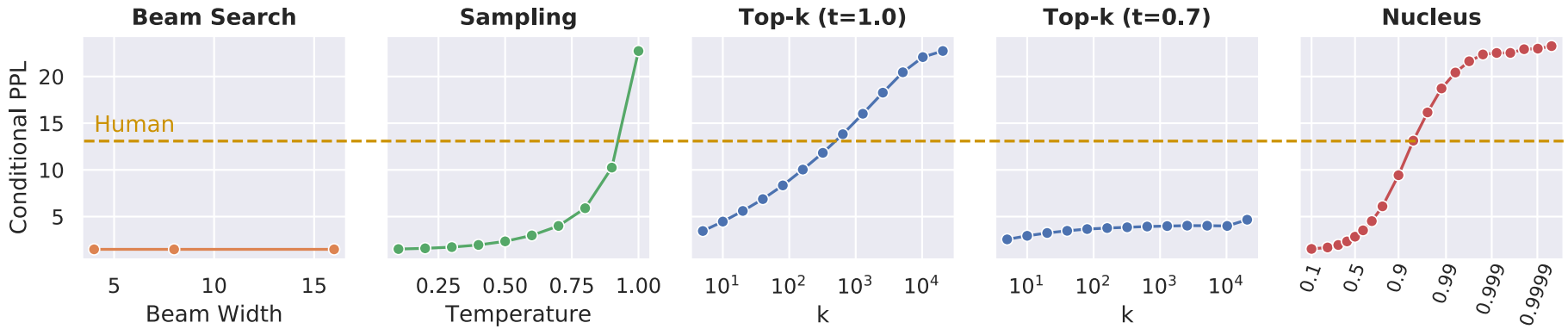
Natural Language Does Not Maximize Probability

- **Observation:** Natural language text, as written by humans, does not maximize per-token probability
- Natural language incorporates lower-probability but informative tokens



Holtzman et al. (2019)

Perplexities of Generations from Various Decoding Methods



- **Beam Search:** Shows unnaturally low perplexities, indicating overly predictable sequences.
- **Sampling, Top-k, and Nucleus** Sampling can be adjusted to achieve human-like perplexities.
- However, **Sampling and Top-** k struggle with coherency when parameters are tuned to reach these human-level perplexities.

Decoding Strategies

- LMs place a distribution $\mathrm{P}\left(\mathrm{x_i} \mid \mathrm{x_1}, \ldots, \mathrm{x_{i-1}}\right)$. How do we generate text from these?

- Maximization-based decoding:
  - **Greedy decoding**: pick the most likely word at each step
  - **Beam search**: keep track of the top k most likely sequences

- Sampling-based decoding:
  - **Random sampling**: sample from the full distribution
  - **Nucleus sampling**: sample from the top p fraction of the distribution
  - **Top-k sampling**: sample from the top k most likely words

That's it!