Lecture 11: Transformer Language Models

Instructor: Swabha Swayamdipta

USC CSCI 444 NLP

Oct 13, 2025

Some slides adapted from Dan Jurafsky and Chris Manning

Announcements + Logistics

- Wed: HW2 due
- Next Mon: Flipped Classroom with Project Discussions
  - You work on your project in class! Every team takes turns to come chat with me regarding your proposal feedback, where you are feeling stuck, etc.

Quiz 2
Solutions (Redacted)

Overall Performance

Question	Average Grade	Standard Deviation
Question 1	57.14 %	49.49 %
Question 2	71.43 %	45.18 %
Question 3	21.43 %	41.03 %
Question 4	21.43 %	41.03 %
Question 5	71.43 %	45.18 %
Question 6	9.52 %	19.63 %

Lecture Outline

- Quiz 2: Solutions
- Recap: Transformers
- The Pretraining / Post-training Paradigm
- Encoder-only Transformer LMs
  - Masked Language Modeling with Transformers

Recap: Transformers

Transformers are Self-Attention Networks

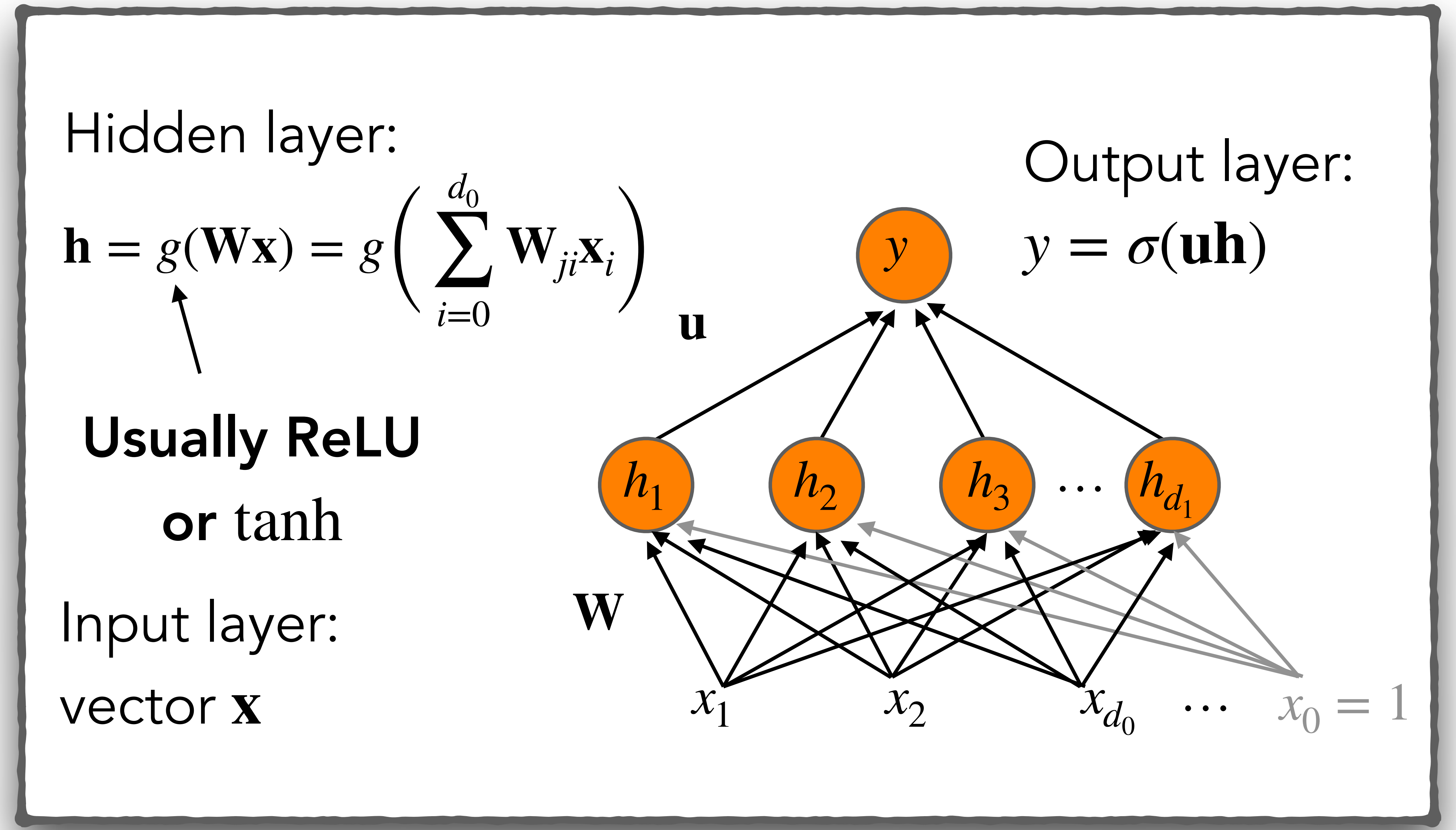
- Self-Attention is the key innovation behind Transformers!
- Transformers map sequences of input vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ to sequences of output vectors $(\mathbf{y}_1, \ldots, \mathbf{y}_n)$ of the same length.
- Made up of stacks of Transformer blocks
  - each of which is a multilayer network made by combining
    - simple linear layers,
    - feedforward networks, and
    - self-attention layers
  - No recurrent connections!

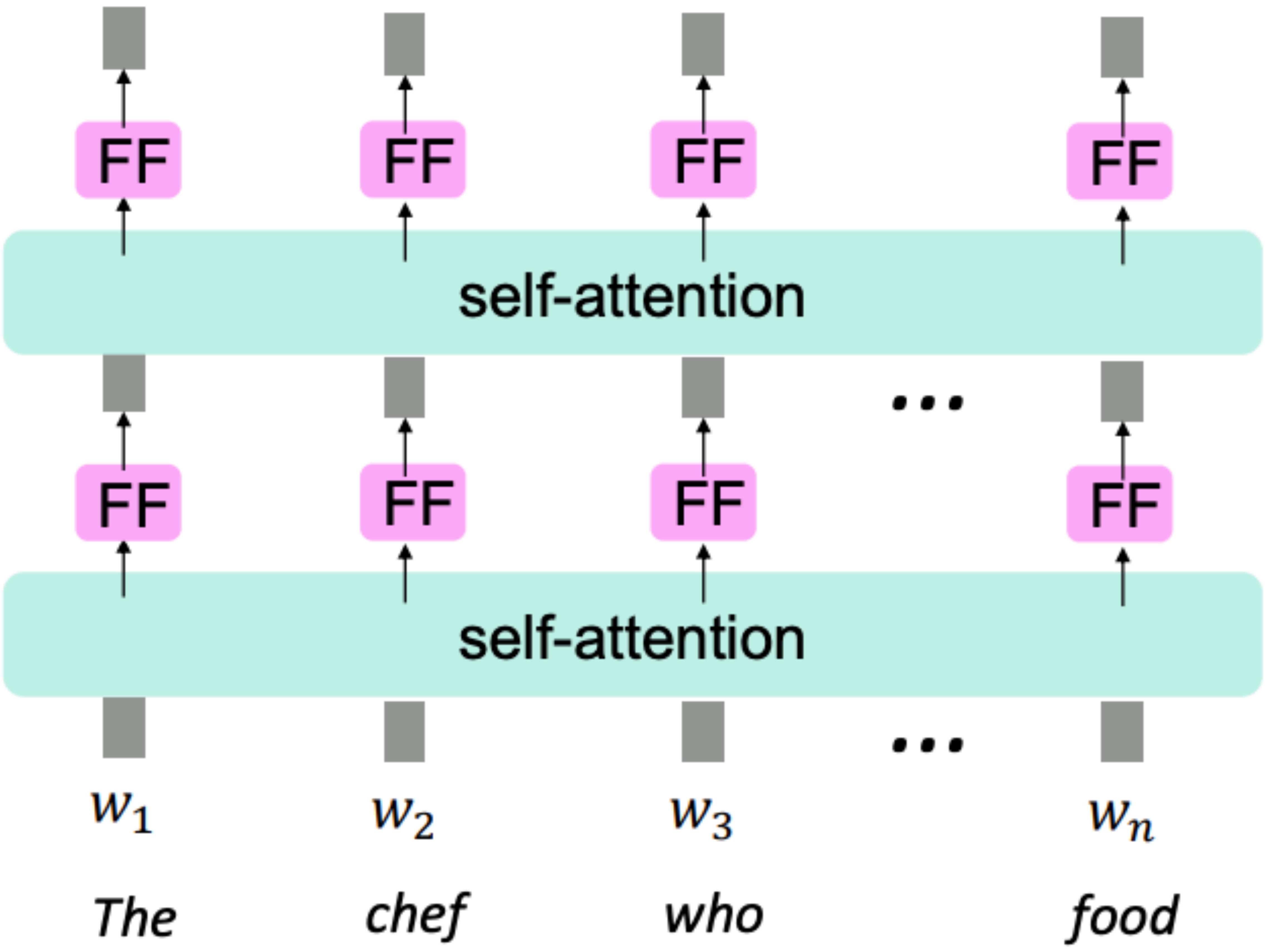
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Self-Attention and Weighted Averages

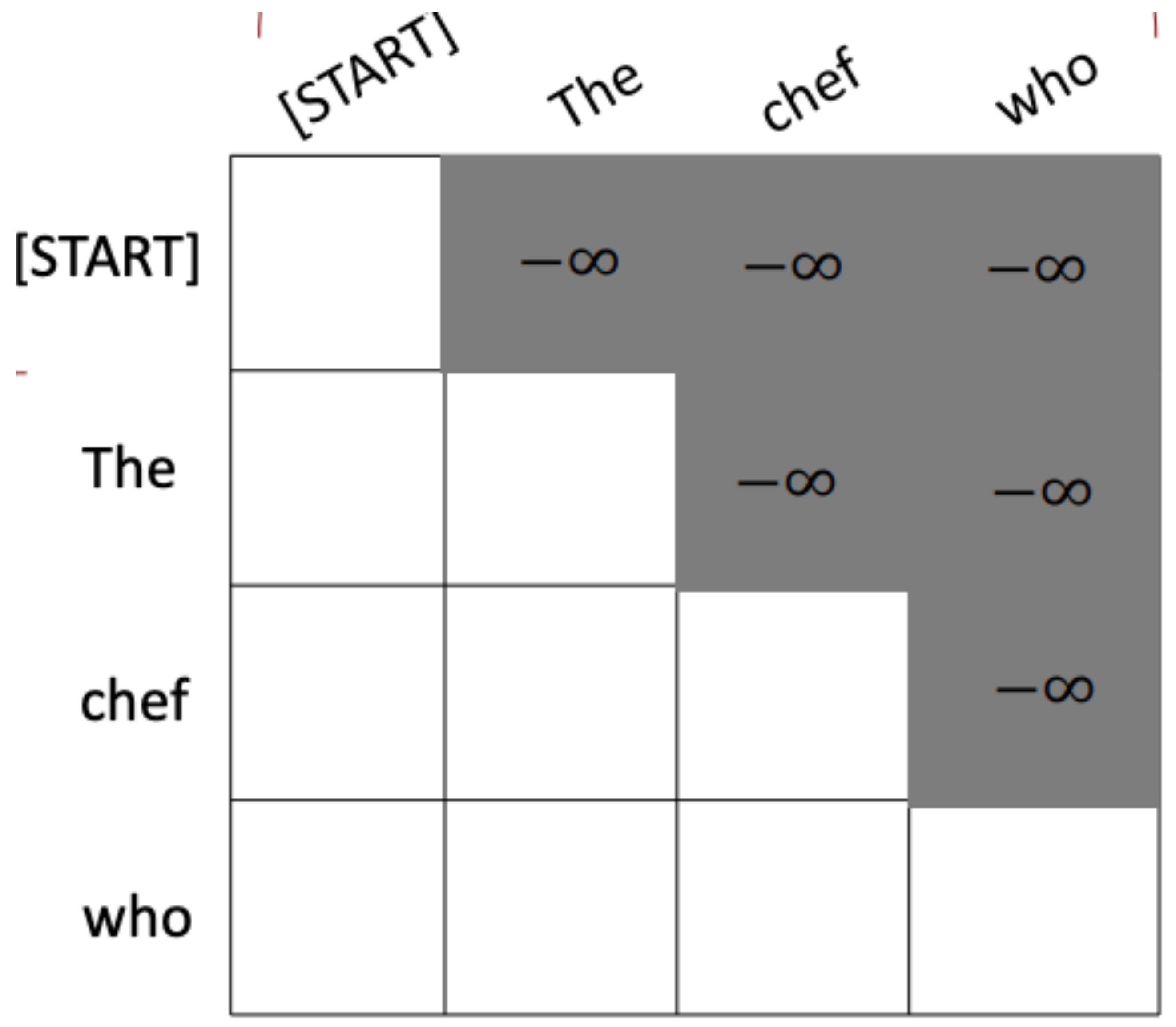
- **Problem**: there are no element-wise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors
- **Solution**: add a feed-forward network to post-process each output vector.





Self Attention and Future Information

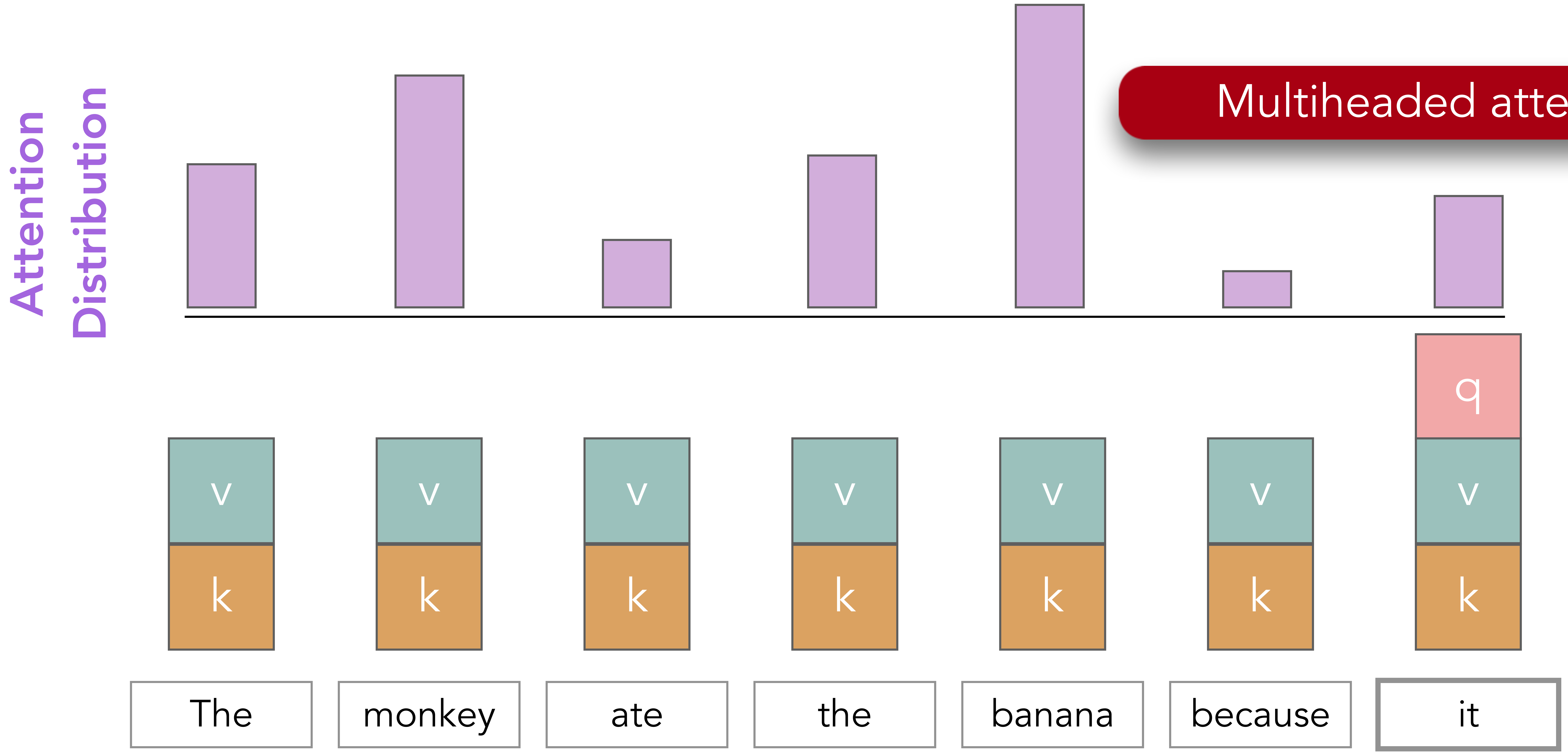
- **Problem**: Need to ensure we don't "look at the future" when predicting a sequence
  - e.g. Target sentence in machine translation or generated sentence in language modeling
  - To use self-attention in decoders, we need to ensure we can't peek at the future, *during training*
- **Solution (Naïve)**: At every time step, we could change the set of keys and queries to include only past words.
  - (Inefficient!)
- **Solution:** To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$



USC Viterbi

Self-Attention and Heads

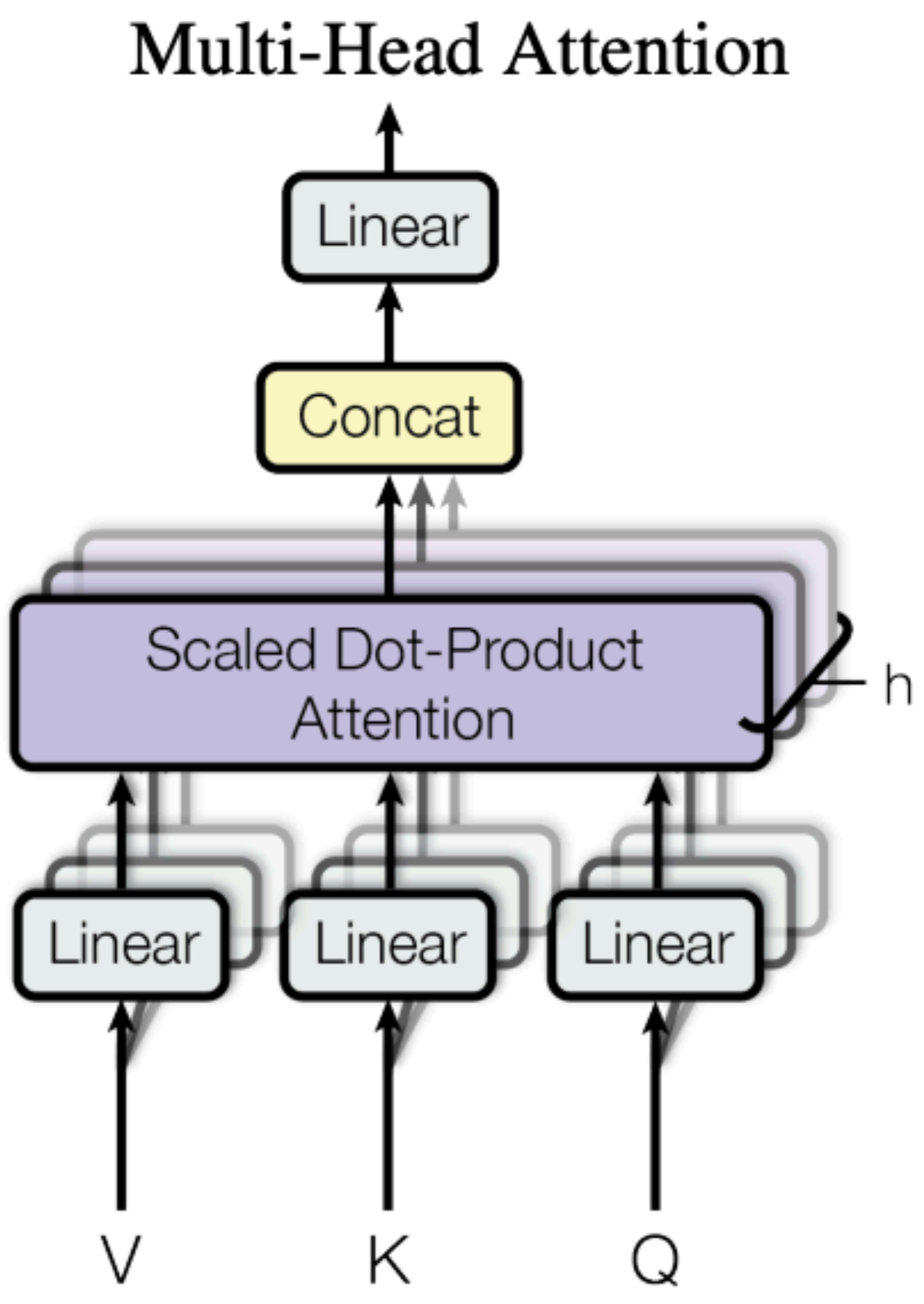
- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax
- **Solution**: Consider multiple attention computations in parallel



Multiheaded attention

Multi-headed attention

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $\mathbf{x}_i^T \mathbf{Q}^T (\mathbf{K}\mathbf{x}_j)$ is high, but maybe we want to focus on different j for different reasons?
- Define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and $1 \leq l \leq h$.
- Each attention head performs attention independently:
- Then the outputs of all the heads are combined!



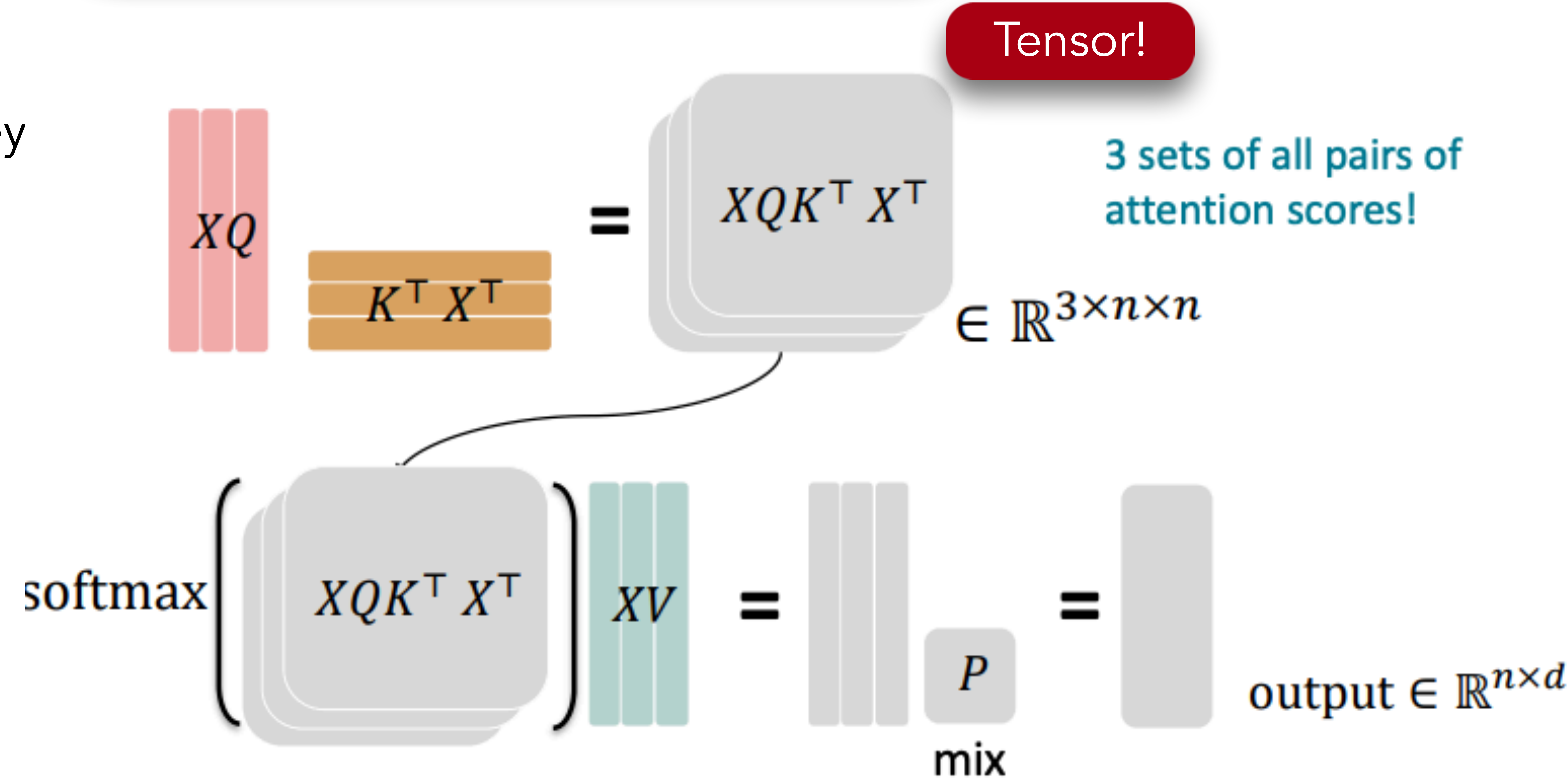
Each head gets to "look" at different things, and construct value vectors differently

Multiheaded Attention: Visualization

Still efficient, can be parallelized!

First, take the query-key dot products in one matrix multiplication: $\mathbf{XQ}_l(\mathbf{XK}_l)^T$

Next, softmax, and compute the weighted average with another matrix multiplication.

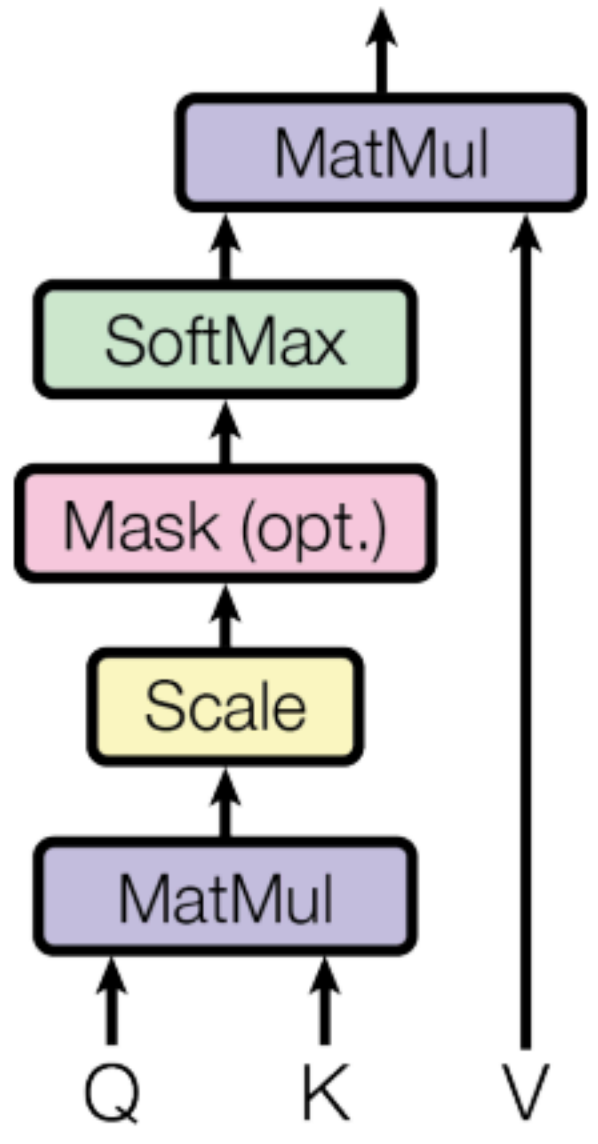


Scaled Dot Product Attention

$$\textbf{output}_\ell = \textbf{softmax}(XQ_\ell K_\ell^T X^T) * XV_\ell$$

- So far: Dot product self-attention
- When dimensionality d becomes large, dot products between vectors tend to become large
- Because of this, inputs to the softmax function can be large, making the gradients small
- Now: Scaled Dot product self-attention to aid in training

$$\textbf{output}_\ell = \textbf{softmax}\left(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}\right) * XV_\ell$$



- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h, where h is the number of heads

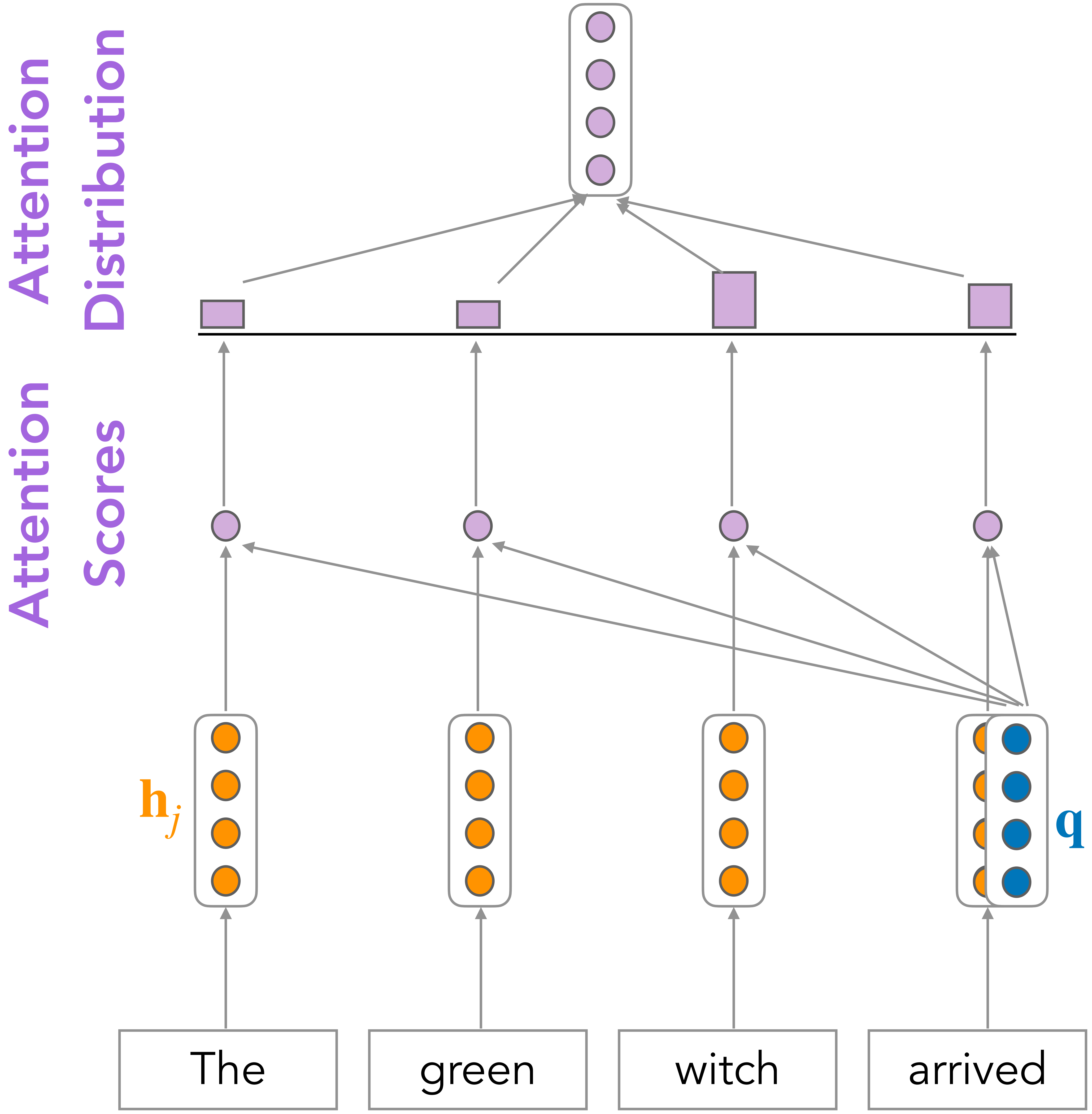
Attention is all you need (Vaswani et al., 2017)

USC Viterbi

Self-Attention: Order Information?

- Self-attention networks are not necessarily (and not typically) based on Recurrent Neural Nets
  - No more order information!
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.

Do feedforward nets contain order information?

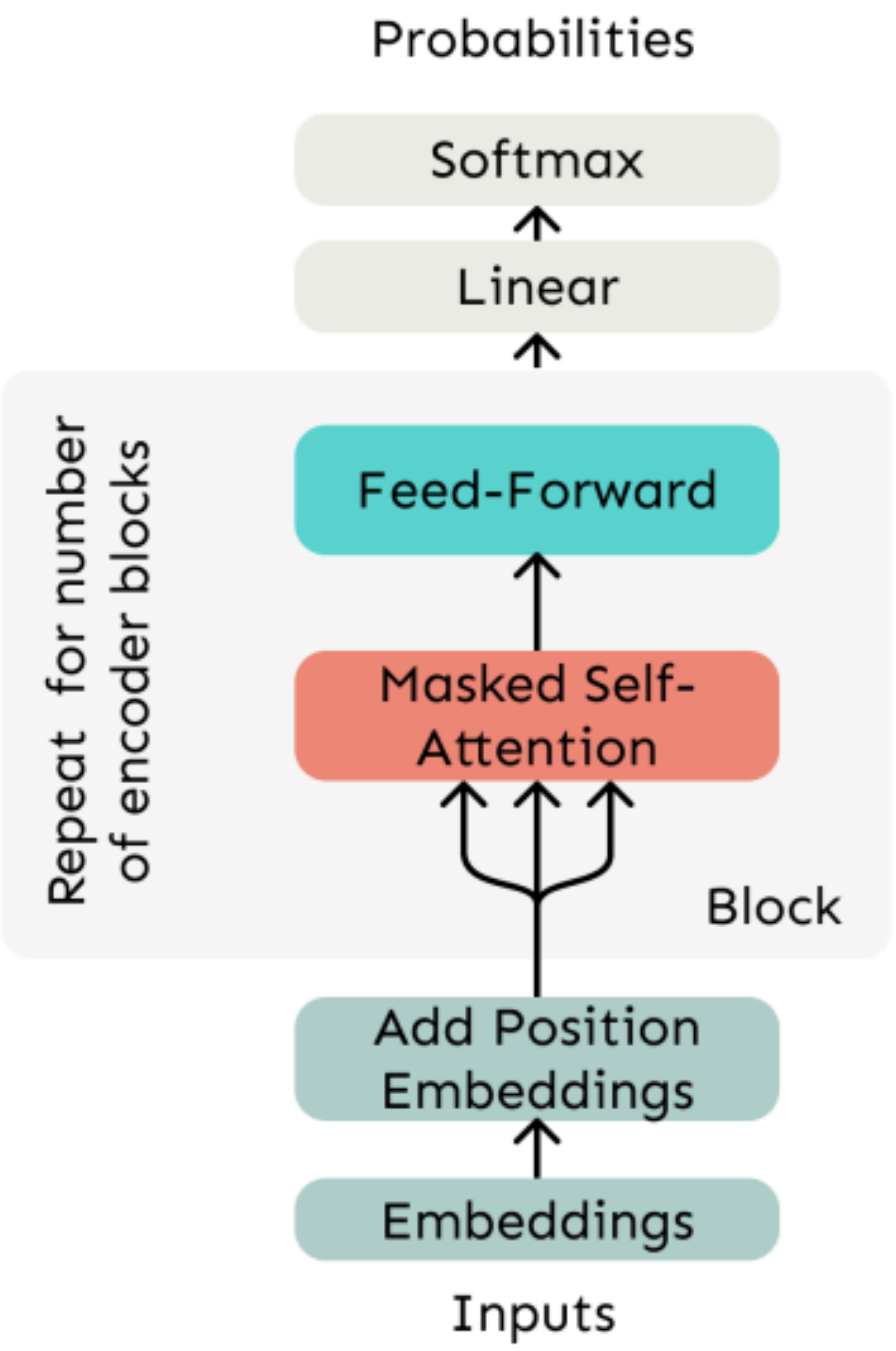


Positional Embeddings

- Recall that $\mathbf{x}_i$ is the embedding of the word at index i. The positioned embedding is:
    - $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$
- Maps integer inputs (for positions) to real-valued vectors, $\mathbf{p}_i$
    - one per position, i in the entire context
- Can be randomly initialized and can let all $\mathbf{p}_i$ be learnable parameters (most common)
- Pros:
    - Flexibility: each position gets to be learned to fit the data
- Cons:
    - Definitely can't extrapolate to indices outside 1, … , n, where n is the maximum length of the sequence allowed under the architecture
    - There will be plenty of training examples for the initial positions in our inputs and correspondingly fewer at the outer length limits

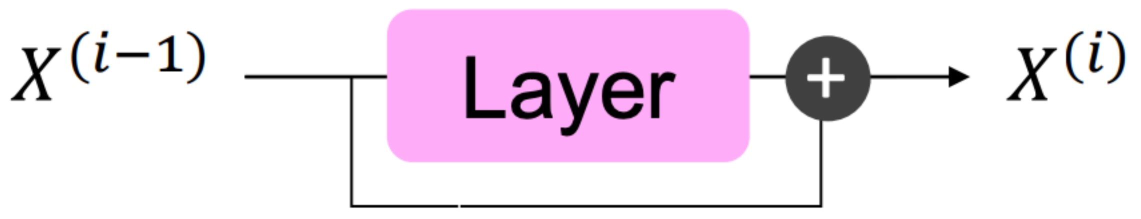
Self-Attention Transformer Building Block

- Self-attention:
  - the basis of the method; with multiple heads
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- Nonlinearities:
  - At the output of the self-attention block
  - Frequently implemented as a simple feedforward network.
- Masking:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.



Residual Connections

- Original Connections: $X^{(i)} = \text{Layer}(X^{(i-1)})$ where i represents the layer
- **Residual Connections** : trick to help models train better.
  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$
    - so we only have to learn “the residual” from the previous layer



Allowing information to skip a layer improves learning and gives higher level layers **direct access to information** from lower layers (He et al., 2016).

Layer Normalization

- Layer normalization is another trick to help models train faster
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.

$$\mu = \frac{1}{d}\sum_{j=1}^{d} x_j; \quad \mu \in \mathbb{R} \qquad\qquad \sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2}; \quad \sigma \in \mathbb{R}$$

Result: New vector with zero mean and a standard deviation of one

$$\hat{x} = \frac{x - \mu}{\sigma}$$

Component-wise subtraction

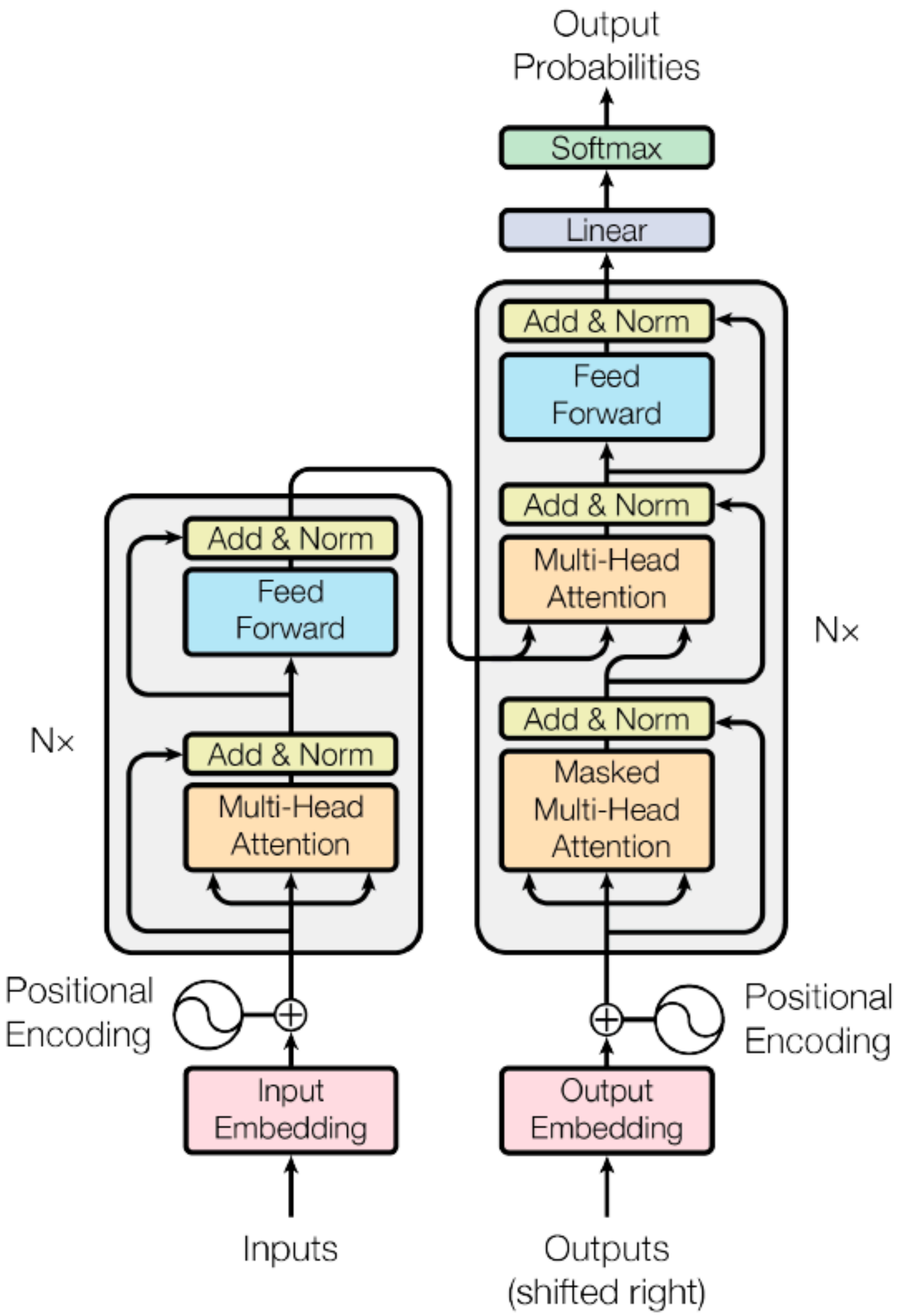
- Let $\gamma \in \mathbb{R}$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)

$$\textbf{LayerNorm} = \gamma\hat{x} + \beta$$

Xu et al., 2019

USC Viterbi

Transformer Diagram



Attention is all you need (Vaswani et al., 2017)

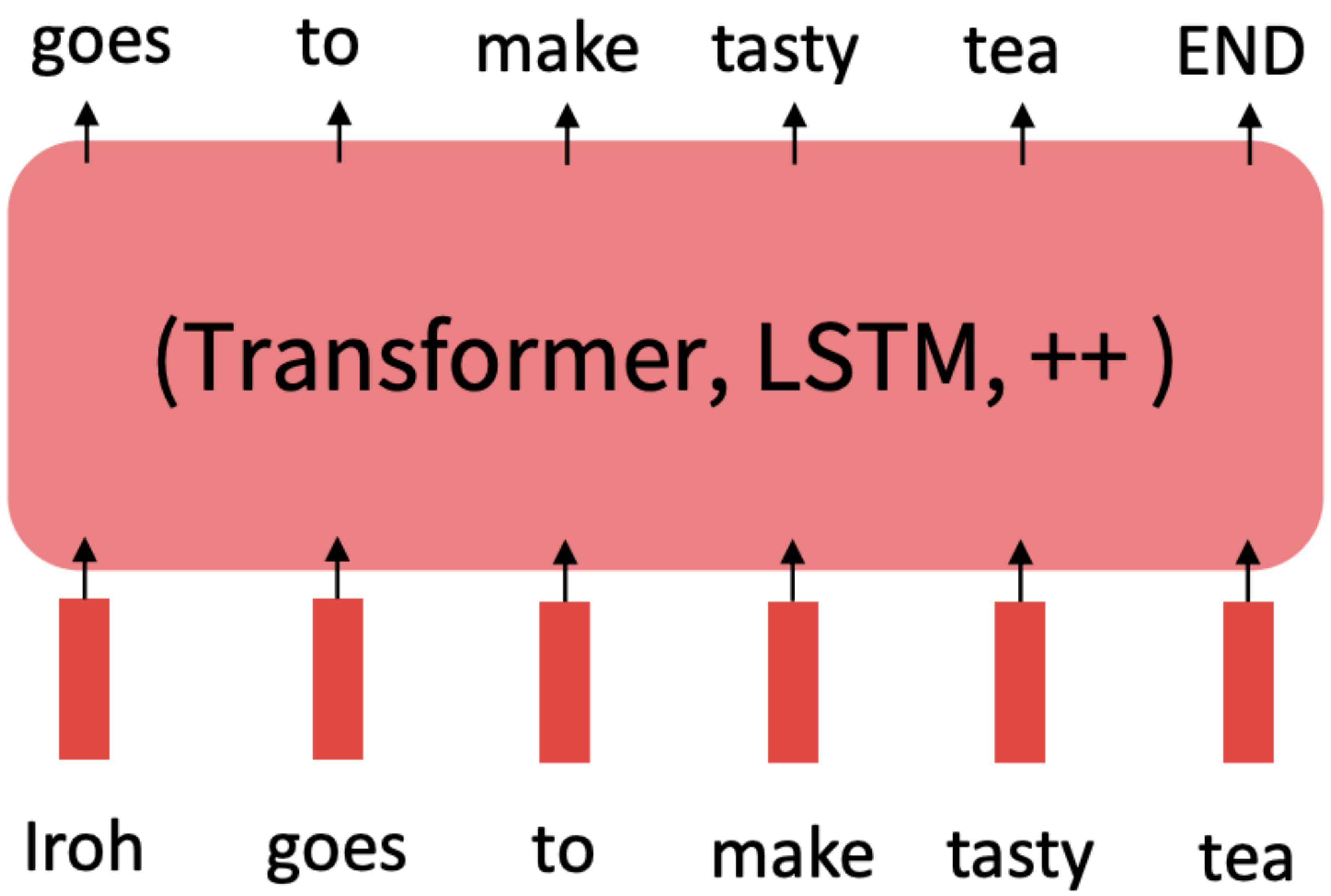
The Pretraining and Post-training Paradigm

The Pretraining / Post-training Paradigm

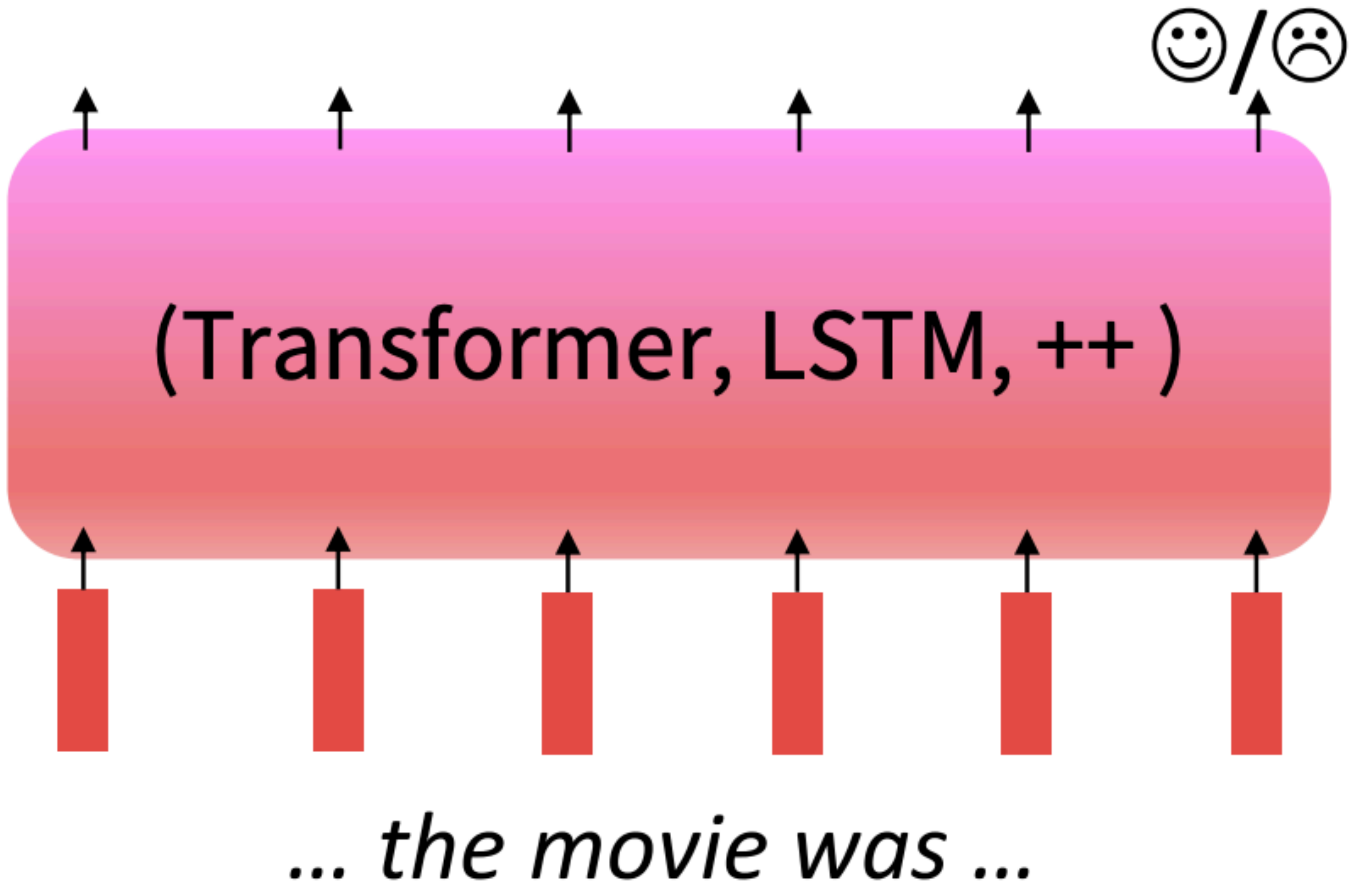
- Pretraining can improve NLP applications by serving as parameter initialization.

Key idea: "Pretrain and Post-train once, Prompt many times."

Step 1: Pretrain (on language corpora)
Lots of text; learn general information!
Results in a "**base**" model



Step 2: Post-train for solving tasks
Instruction Tuning / Supervised Fine Tuning
+ Model Alignment

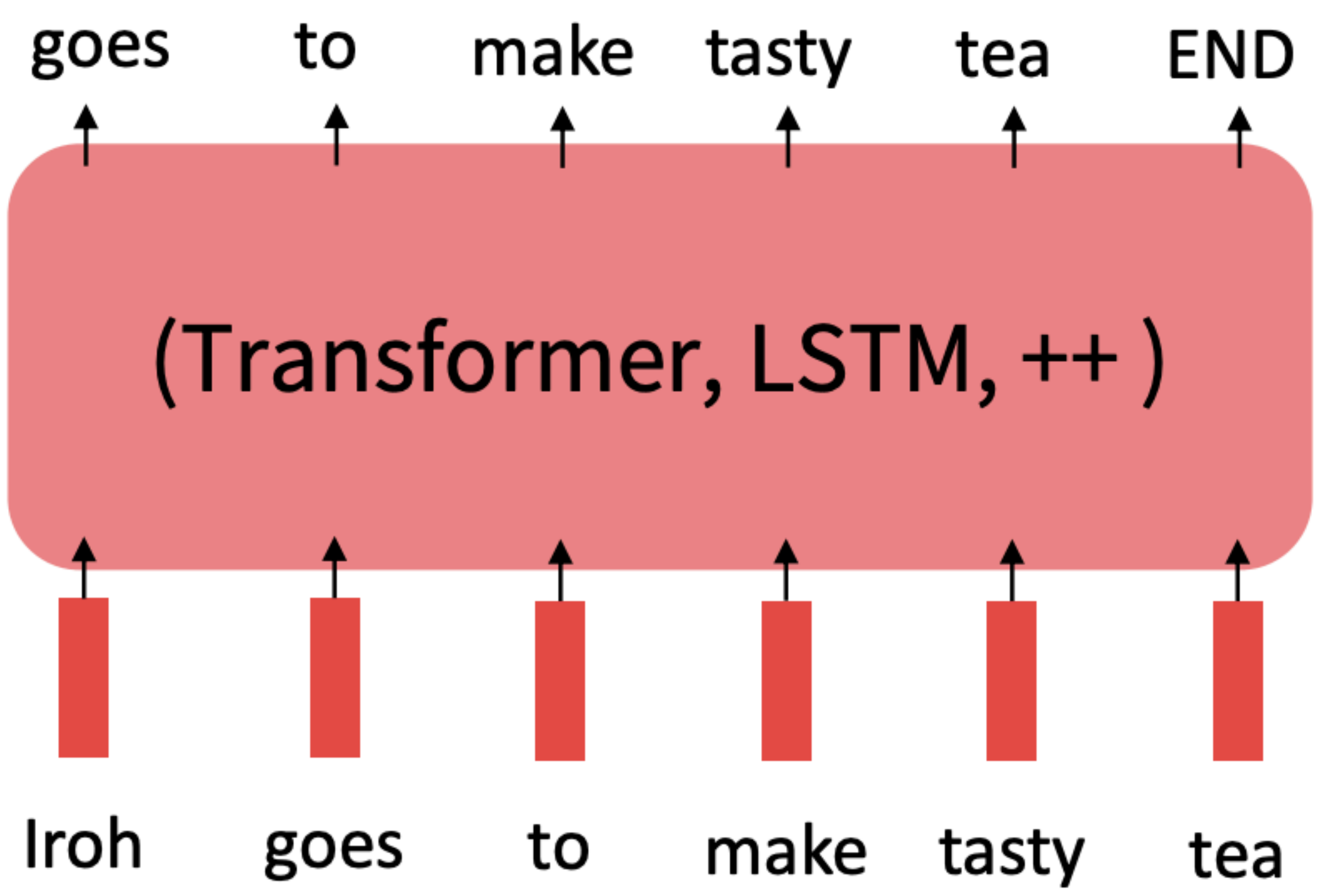


Pretraining

- Central Approach: Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.
- Used for parameter initialization
  - Part of network
  - Full network
- Abstracts away from the task of "learning the language"

Step 1: Pretrain (on language corpora)

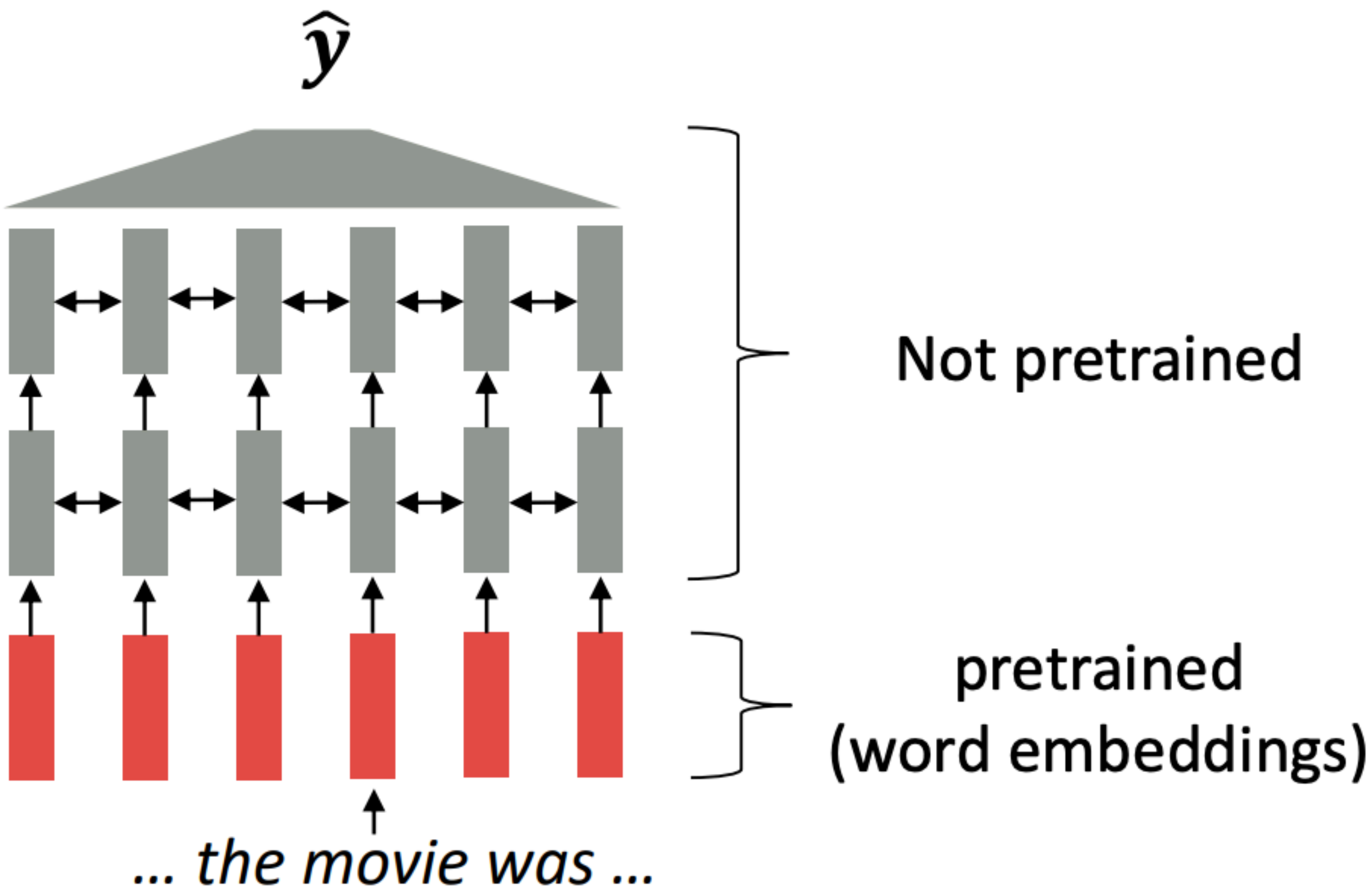
Lots of text; learn general things!



Word embeddings were pretrained too!

Previously:

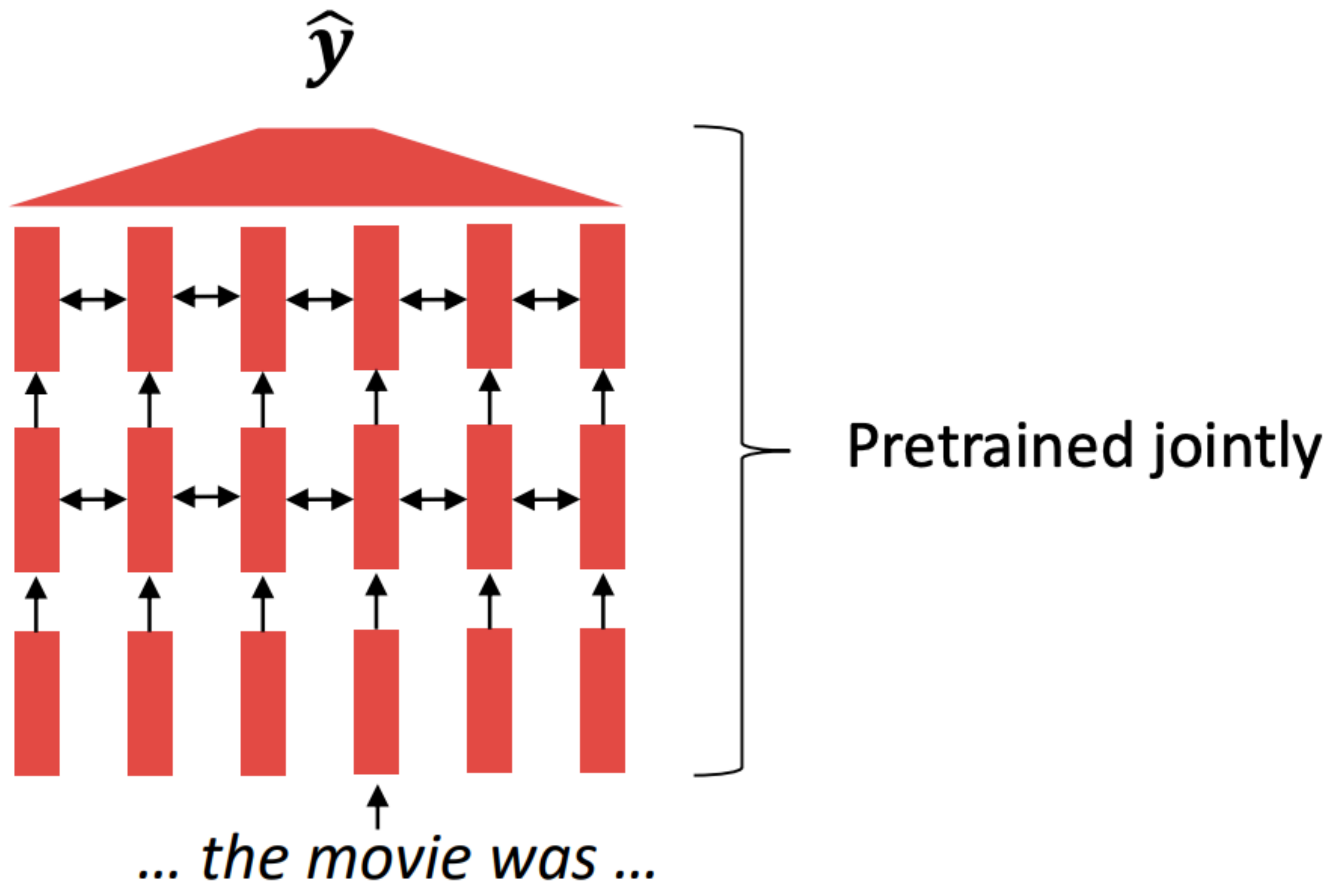
- Start with pretrained word embeddings
  - word2vec
  - GloVe
  - Trained with limited context (windows)
- Learn how to incorporate context in an LSTM or Transformer while training on the task (e.g. sentiment classification)
- Paradigm till 2017



However, the word "movie" gets the same word embedding, no matter what sentence it shows up in!

Pretraining Entire Models

- In modern NLP:
  - All (or almost all) parameters in NLP networks are initialized via pretraining.
  - This has been exceptionally effective at building strong:
    - representations of language
    - parameter initializations for strong NLP models.
    - probability distributions over language that we can sample from



[This model has learned how to represent entire sentences through pretraining]

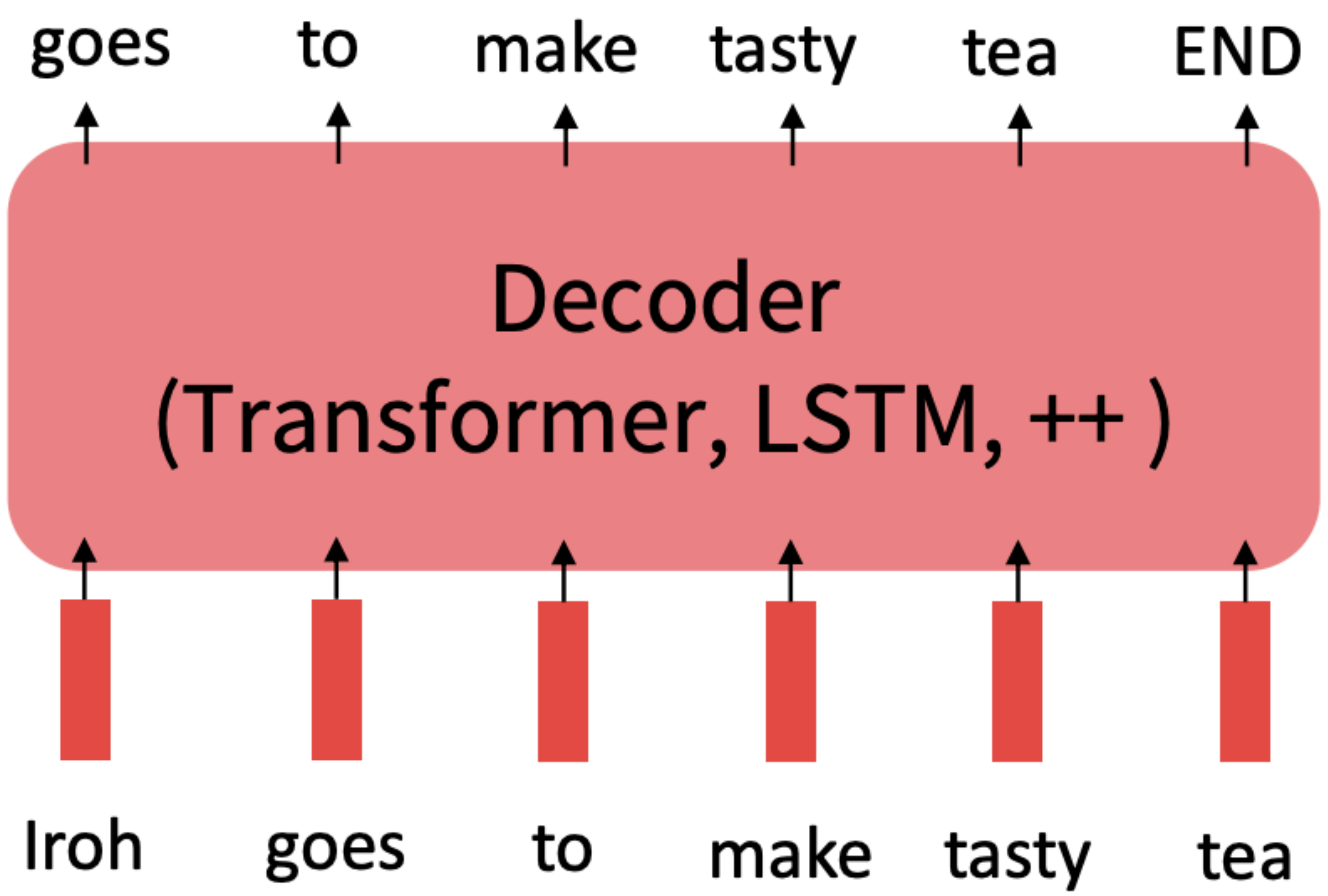
Pretraining: Intuition from SGD

Why should pretraining and finetuning help, from a "training neural nets" perspective?

- Pretraining provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathscr{L}_{\text{pretrain}}(\theta)$
  - $\mathscr{L}_{\text{pretrain}}(\theta)$ is the pretraining loss
- Then, finetuning approximates $\min_{\theta} \mathscr{L}_{\text{finetune}}(\theta)$, **but starting at** $\hat{\theta}$.
  - $\mathscr{L}_{\text{finetune}}(\theta)$ is the finetuning loss
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to $\hat{\theta}$ during finetuning
  - It is possible that the finetuning local minima near $\hat{\theta}$ tends to generalize well!
  - And/or, maybe the gradients of finetuning loss near $\hat{\theta}$ propagate nicely!

Pretraining: Language Models

- Recall the language modeling task:
  - Model $p_\theta(w_t | w_{1:t-1})$, the probability distribution over words given their past contexts.
  - There's lots of data for this! (In English.)
- Pretraining through language modeling:
  - Train a neural network to perform language modeling on a large amount of text.
  - Save the network parameters
  - Called a causal model



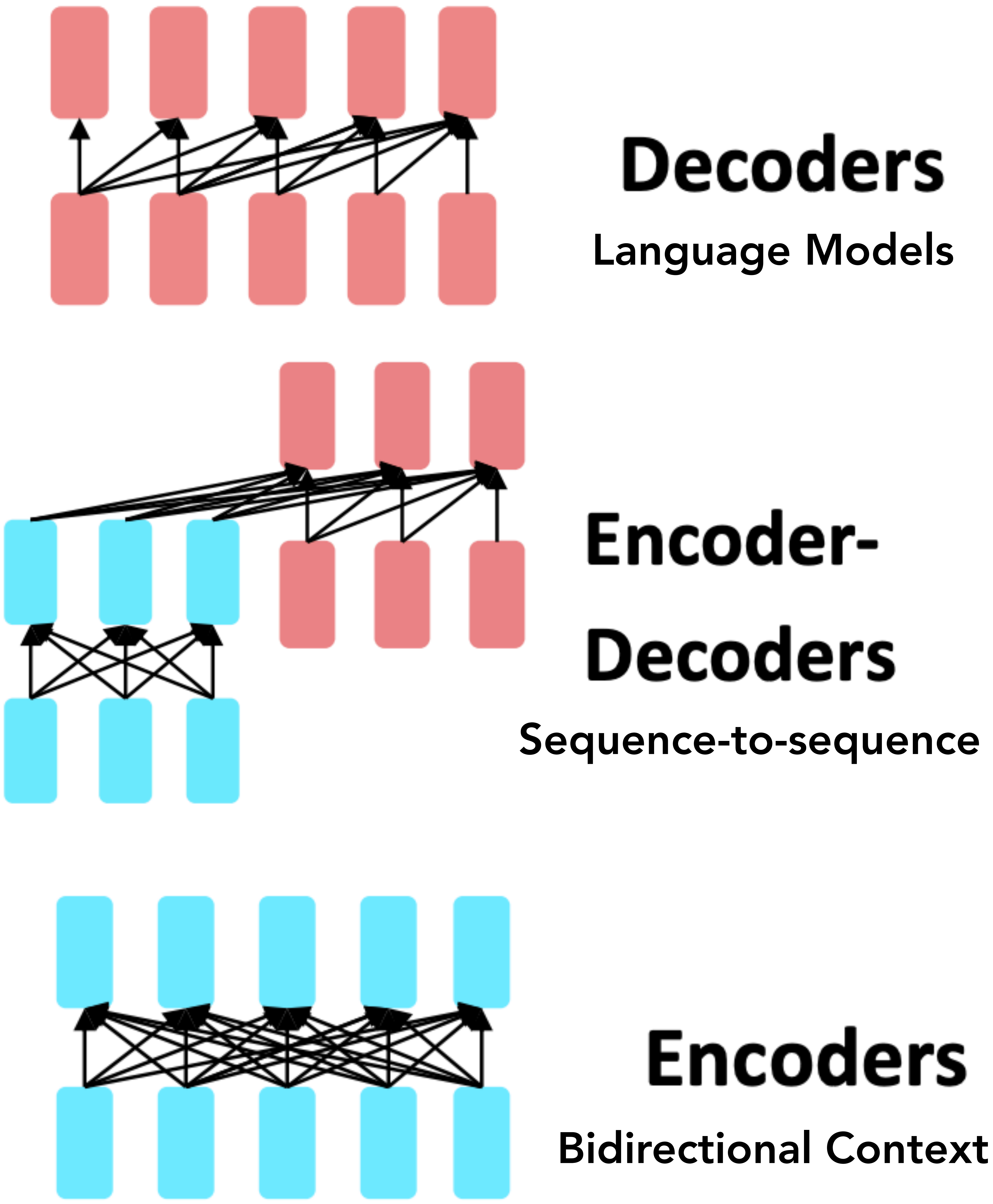
Semi-supervised Sequence Learning

Andrew M. Dai
Google Inc.
adai@google.com

Quoc V. Le
Google Inc.
qvl@google.com

Pretraining

- Can be any task, not just language modeling
- But most successful if the task definition is very general. Hence, language modeling is a great pretraining option
- Three options!

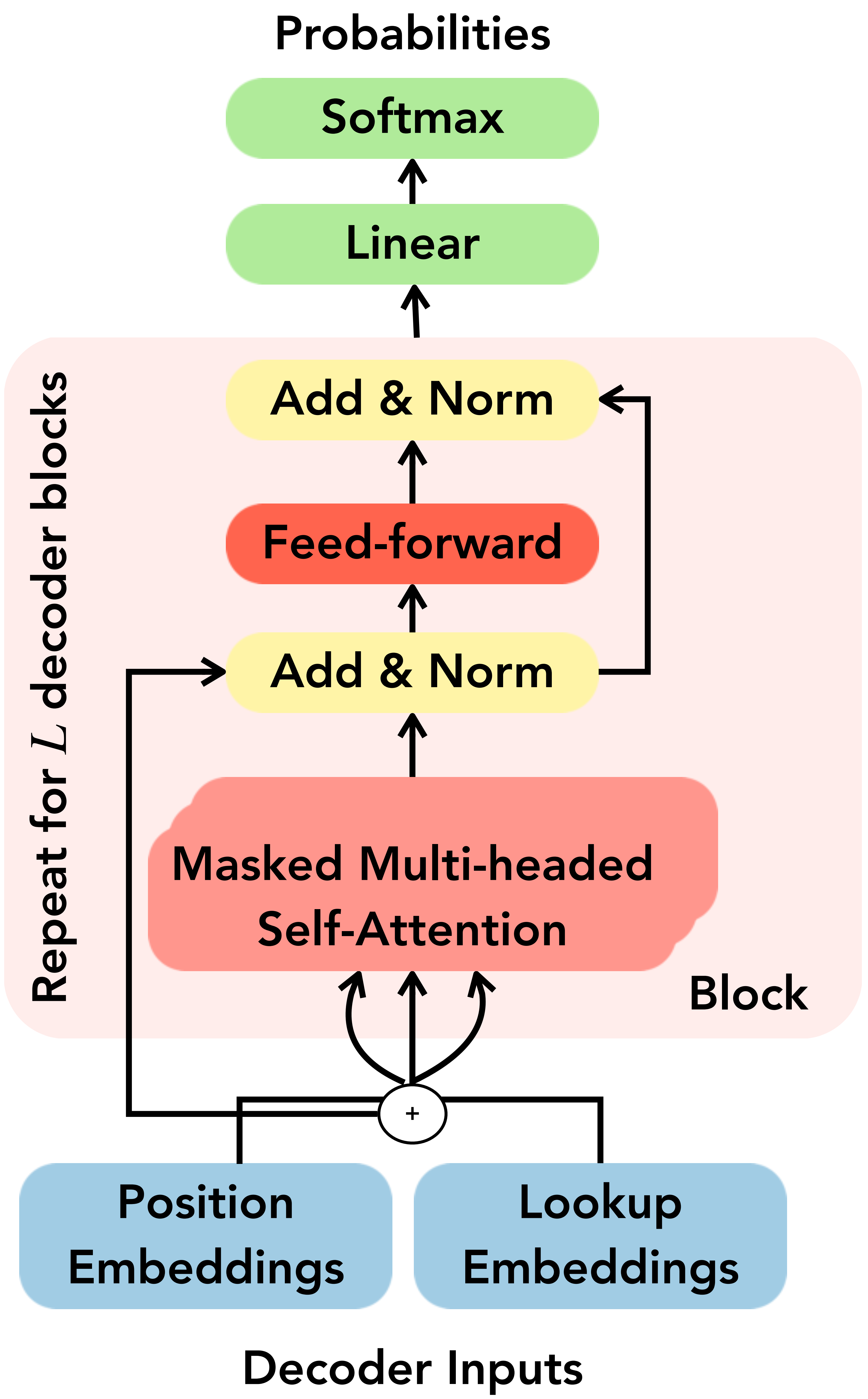


Decoders, Encoder-Decoders and Encoder-only Transformer LMs

USC Viterbi

The Transformer Decoder

- The Transformer Decoder is a stack of Transformer Decoder Blocks.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- Output layer is as always a softmax layer
  - Sometimes called an **unembedding** layer



USC Viterbi

GPT-2

- GPT-2, a larger version (1.5B) of GPT trained on more data, was shown to produce relatively convincing samples of natural language.
- Moved away from classification, only generation

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

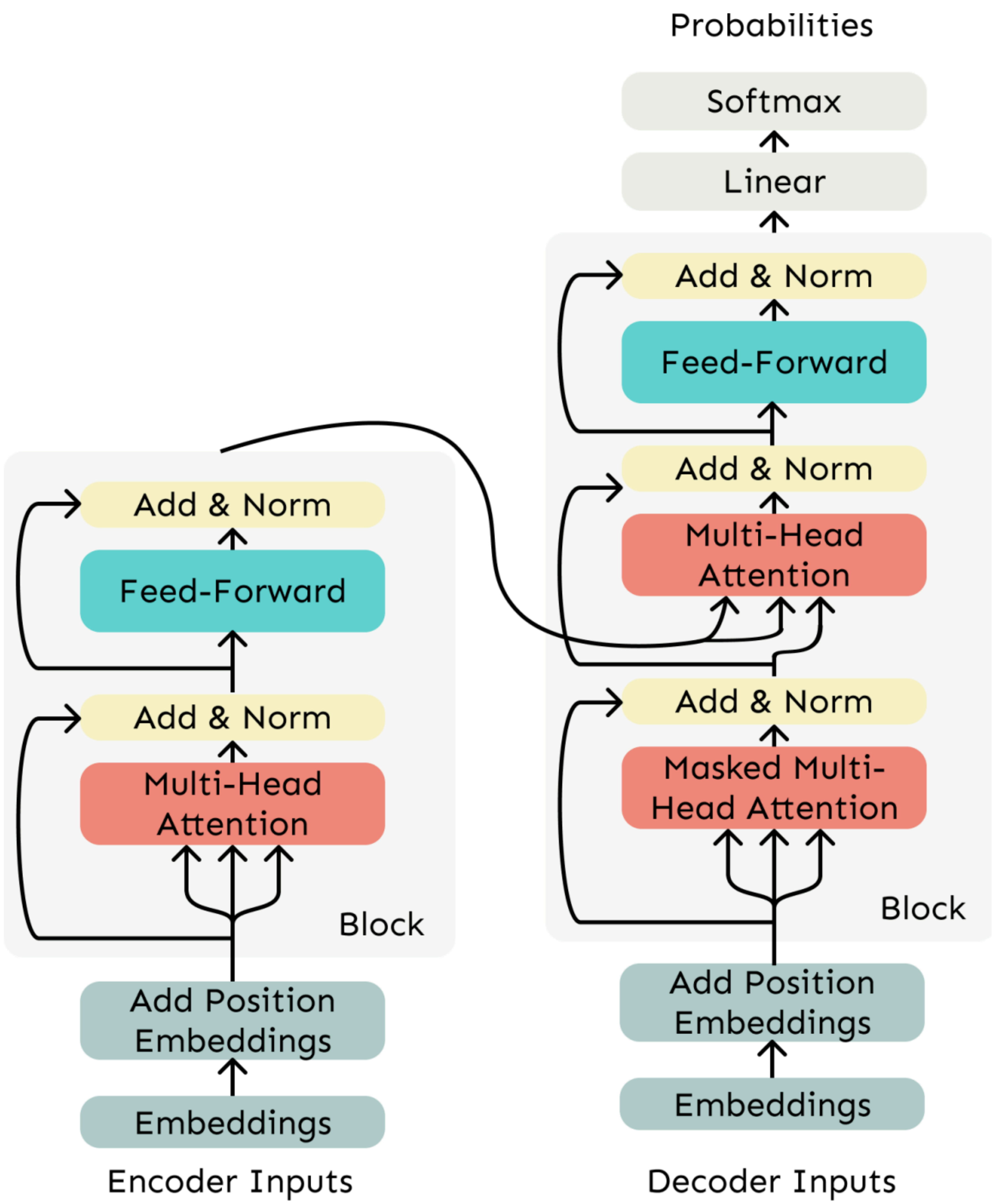
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

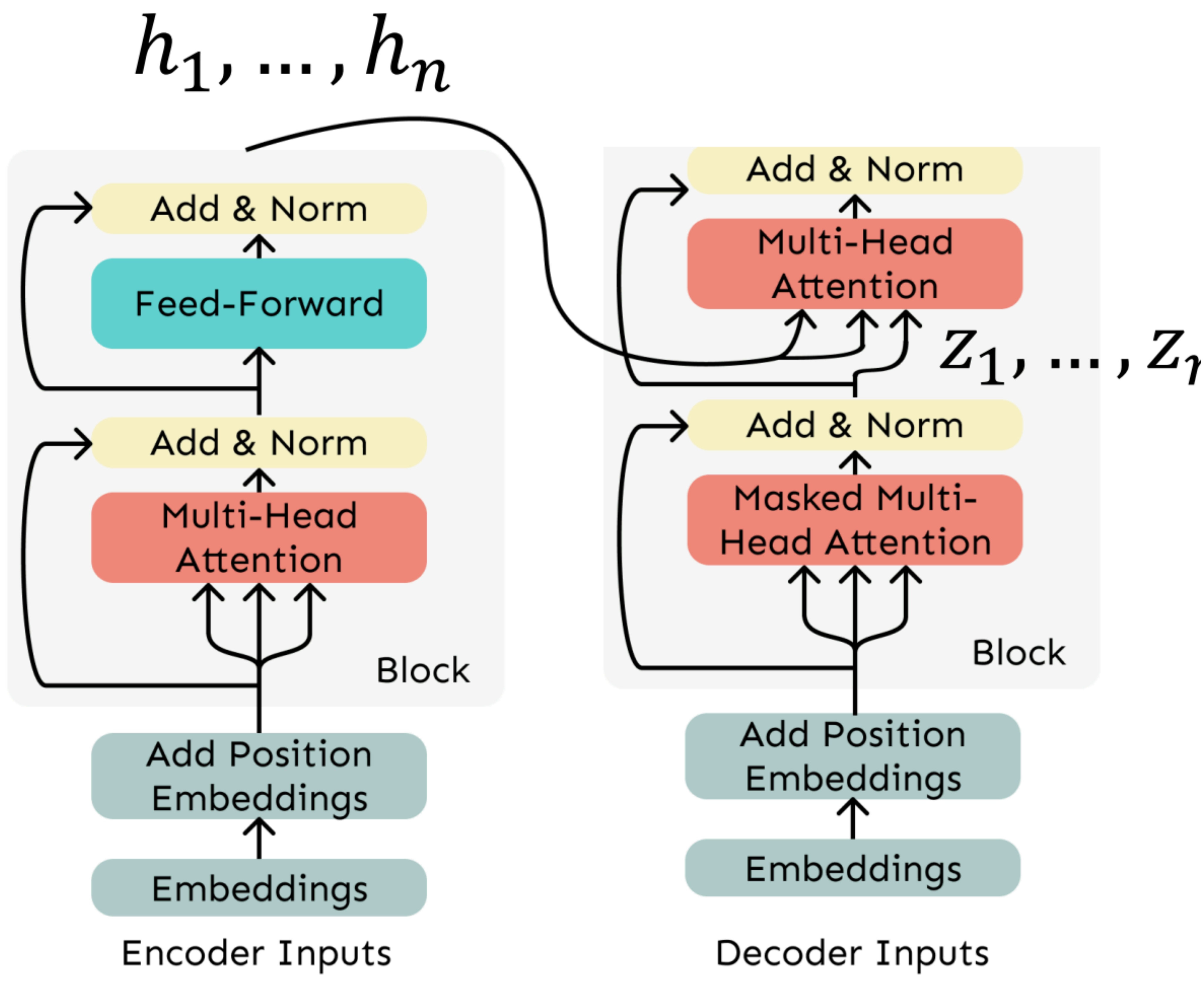
The Transformer Encoder-Decoder

- Recall that in machine translation, we processed the source sentence with a bidirectional model and generated the target with a unidirectional model.
- For this kind of seq2seq format, we often use a Transformer Encoder-Decoder.
- We use a normal Transformer Encoder.
- Our Transformer Decoder is modified to perform **cross-attention** to the output of the Encoder.



Cross Attention

- We saw that self -attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $\mathbf{h}_1, \ldots, \mathbf{h}_n$ be output vectors from the Transformer encoder; $\mathbf{h}_i \in \mathbb{R}^d$
- Let $\mathbf{z}_1, \ldots, \mathbf{z}_n$ be input vectors from the Transformer decoder, $\mathbf{h}_i \in \mathbb{R}^d$
- Then keys and values are drawn from the encoder (like a memory):
  - $\mathbf{k}_i = \mathbf{K}\mathbf{h}_i, \mathbf{v}_i = \mathbf{V}\mathbf{h}_i$
- And the queries are drawn from the decoder, $\mathbf{q}_i = \mathbf{Q}\mathbf{z}_i$

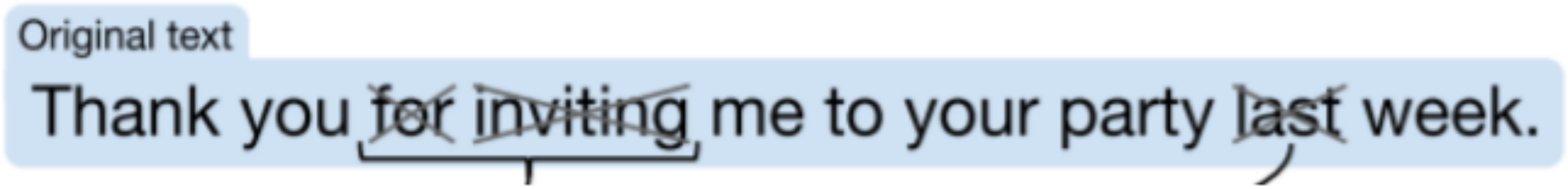


USC Viterbi

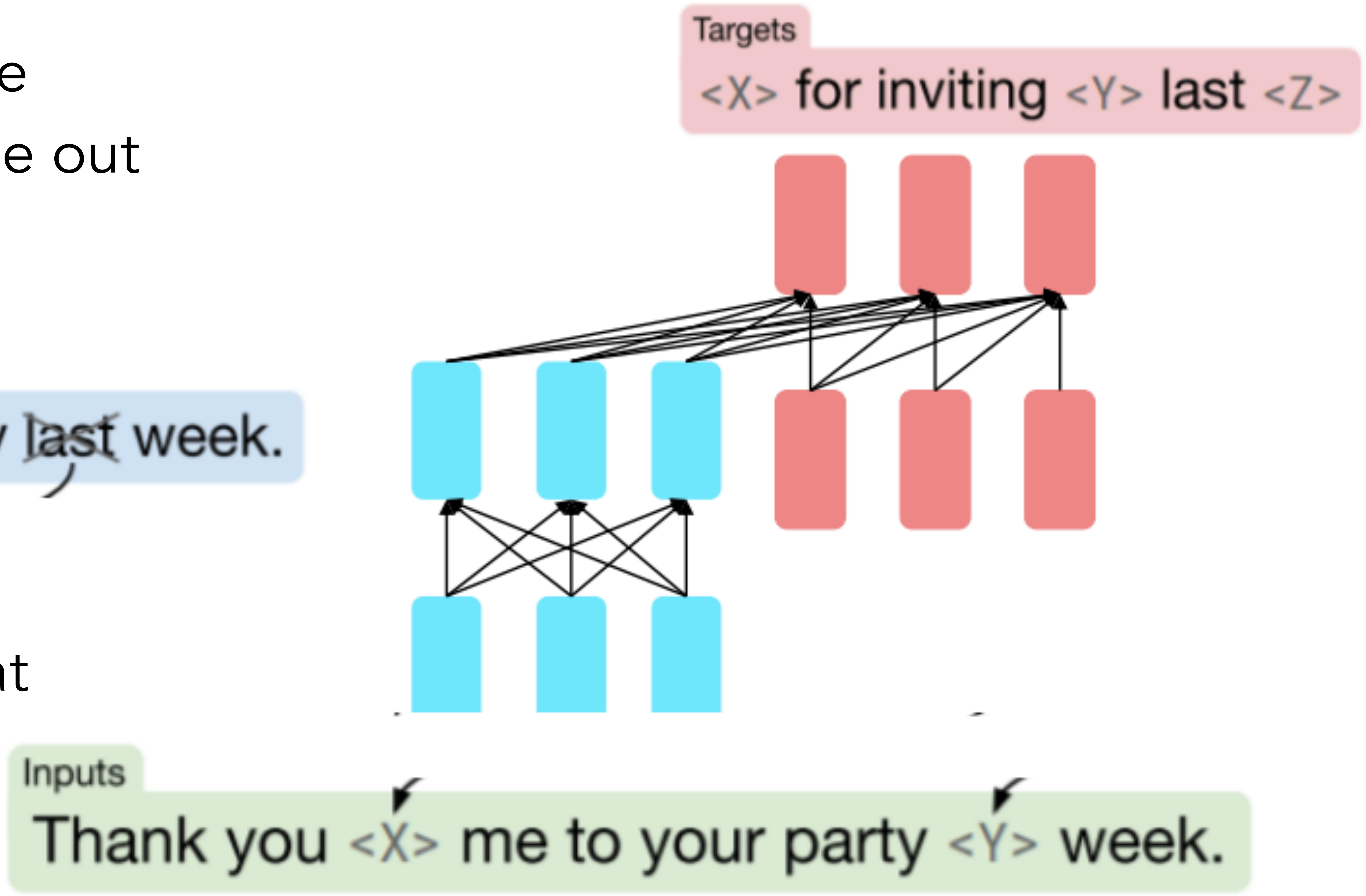
T5: A Pretrained Encoder-Decoder Model

- Raffel et al., 2018 built T5, which uses as a span corruption pretraining objective

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!



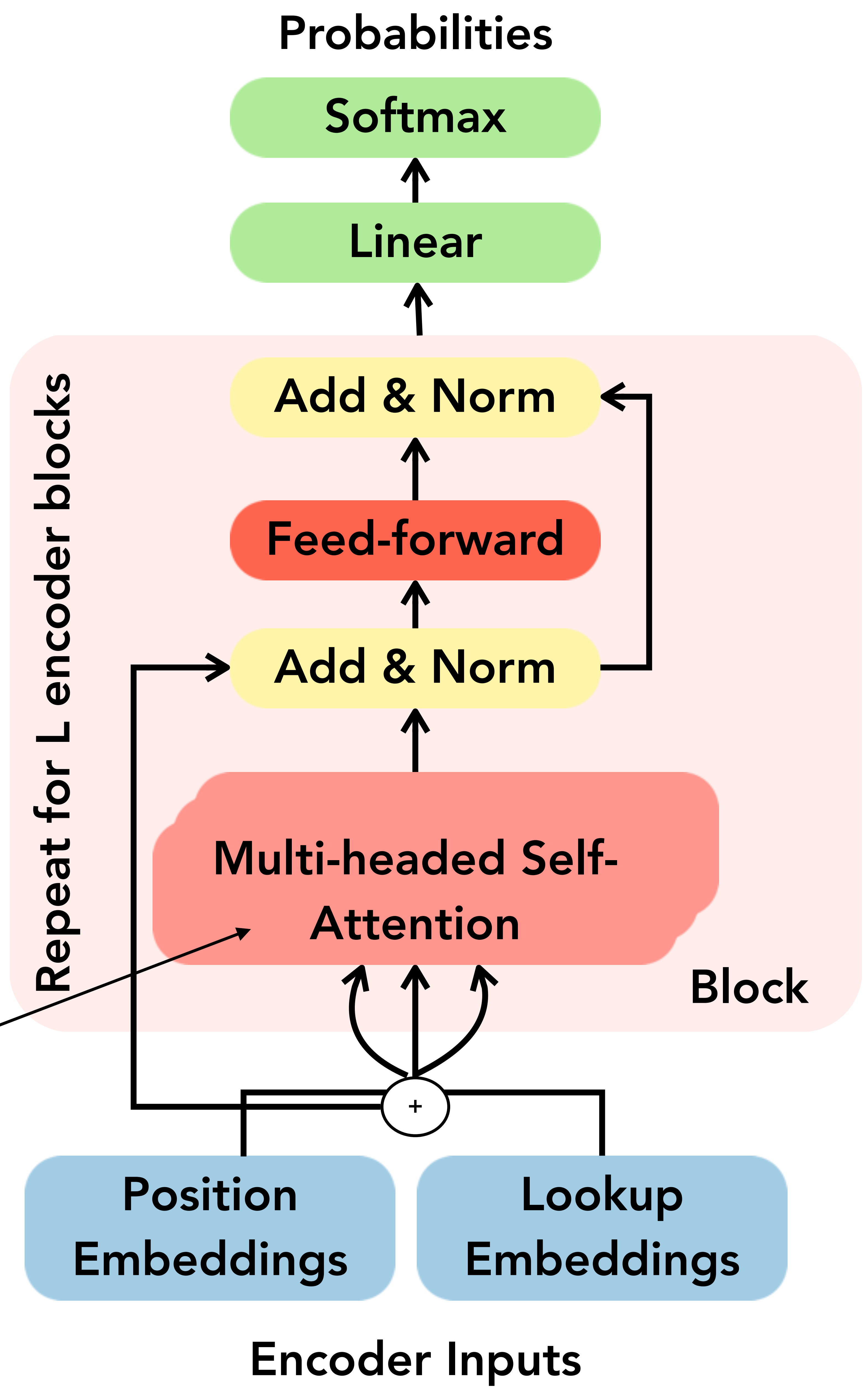
This is implemented in text preprocessing: it's still an objective that looks like language modeling at the decoder side.



The Transformer Encoder

- The Transformer Decoder constrains to unidirectional context, as for language models.
- What if we want bidirectional context, i.e. both left to right as well as right to left?
- The only difference is that we remove the masking in the self-attention.
- Commonly used in sequence prediction tasks such as POS tagging
  - One output token y per input token x

No Masking!



Pre-training
Encoder-Only Language Models

Pretraining Encoders: Bidirectional Context

I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____

Universal Studios Theme Park is located in ______________, California

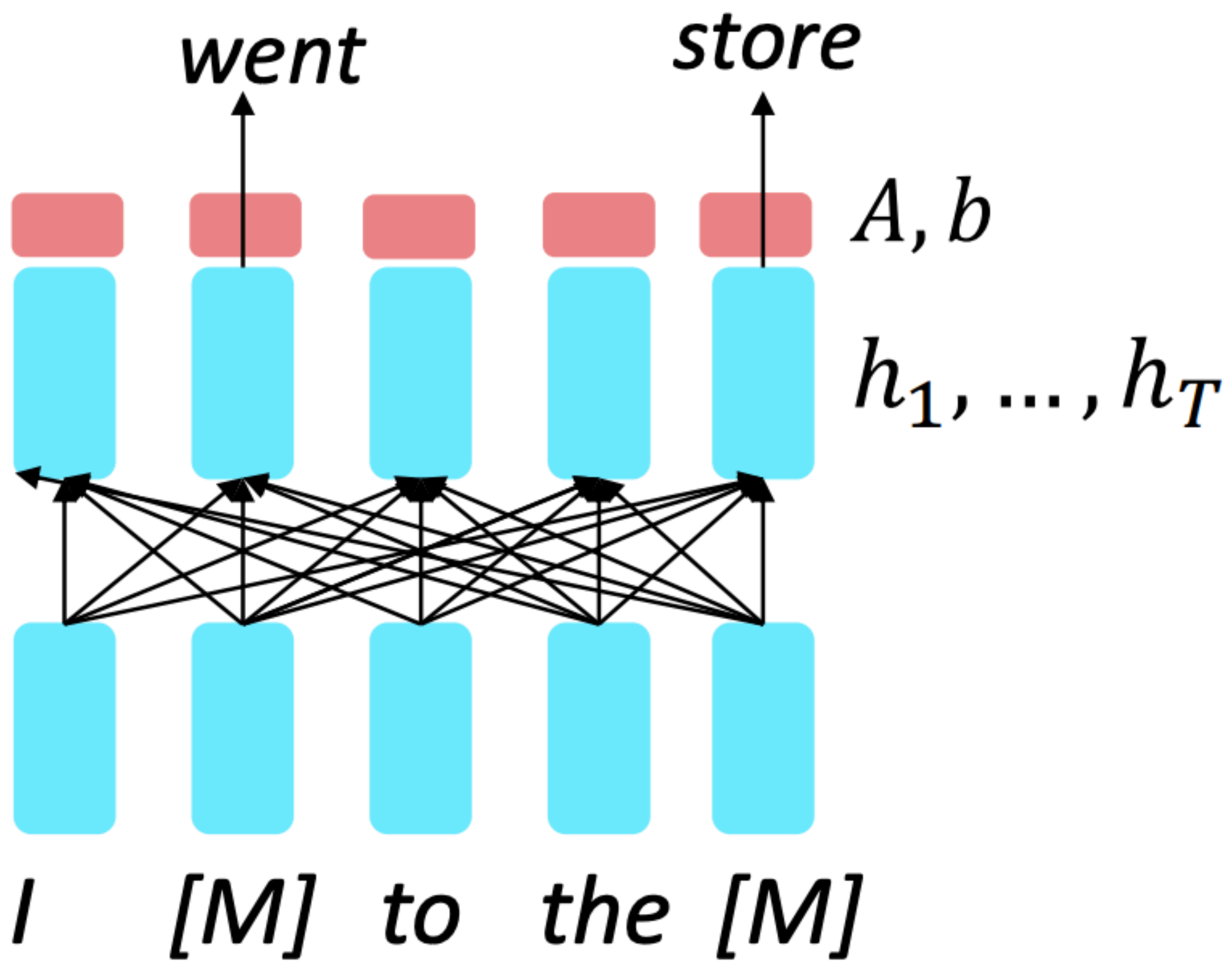
Problem: Input Reconstruction

‘Cause darling i'm a _______ dressed like a daydream

Bidirectional context is important to reconstruct the input!

Pretraining Encoders: Objective

- Encoders get bidirectional context, so we can't do language modeling!
- Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.
  - $\mathbf{h}_1, \ldots, \mathbf{h}_T = \text{Encoder}(w_1, \ldots, w_T)$
  - $\mathbf{y}_i \approx \text{softmax}(A\mathbf{h}_i + b)$



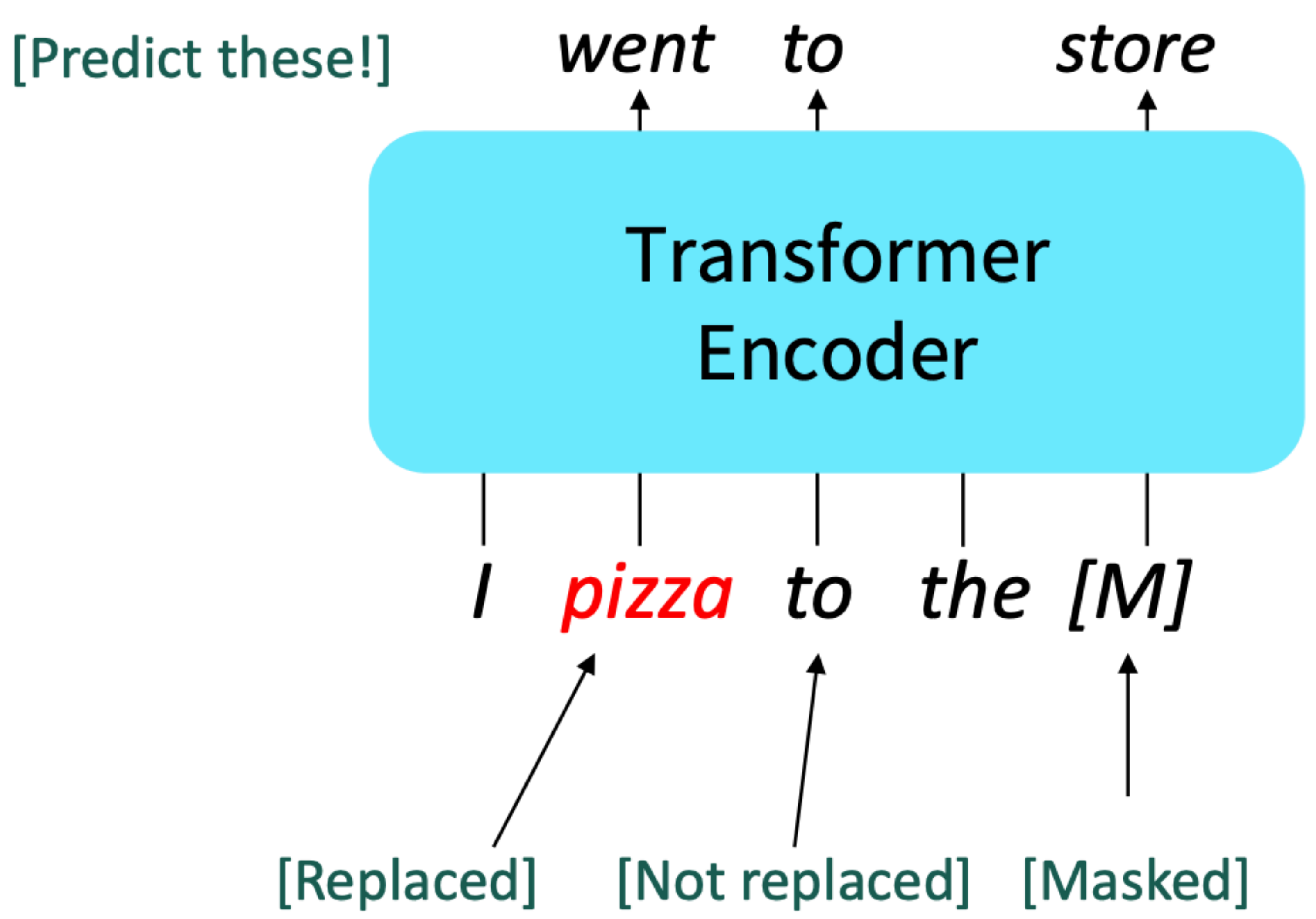
- Only add loss terms from words that are "masked out."
- If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(\tilde{x} \mid x)$.
- Called Masked LM
- Special type of language modeling

Masked Language Modeling

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective and released BERT, a Transformer, pretrained to:

- 15% of the input tokens in a training sequence are sampled for learning, these are to be predicted by the model
- Of these
  - 80% are replaced with [MASK]
  - 10% are replaced with randomly selected tokens,
  - Remaining 10% are left unchanged

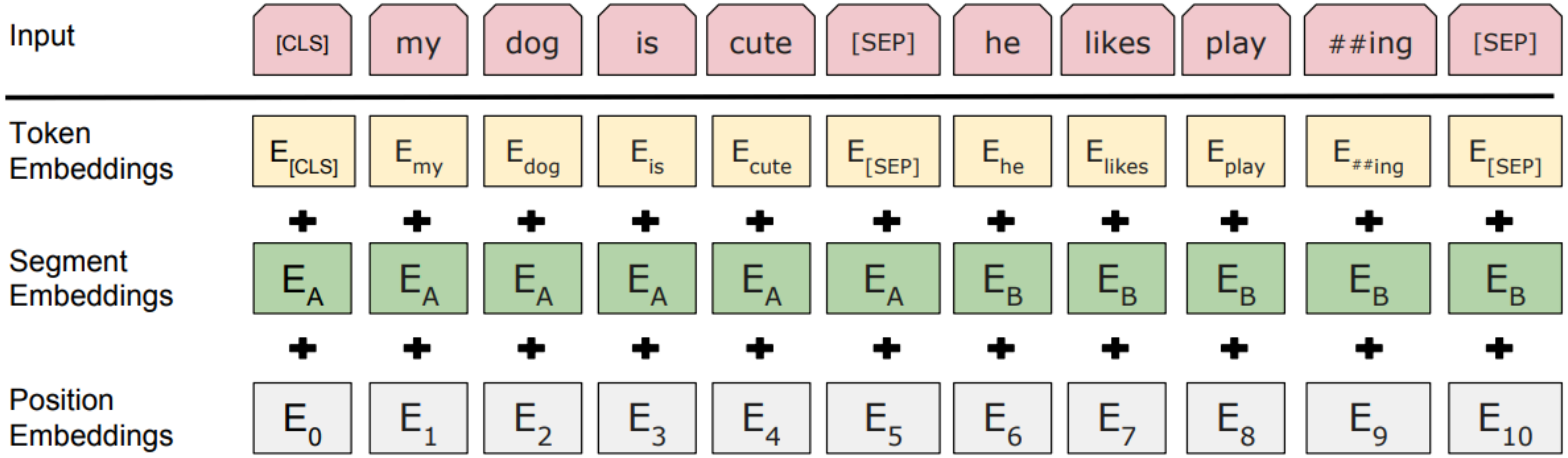


Why?

Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)

BERT: Bidirectional Encoder Representations from Transformers

- The pretraining input to BERT was two separate contiguous chunks of text:

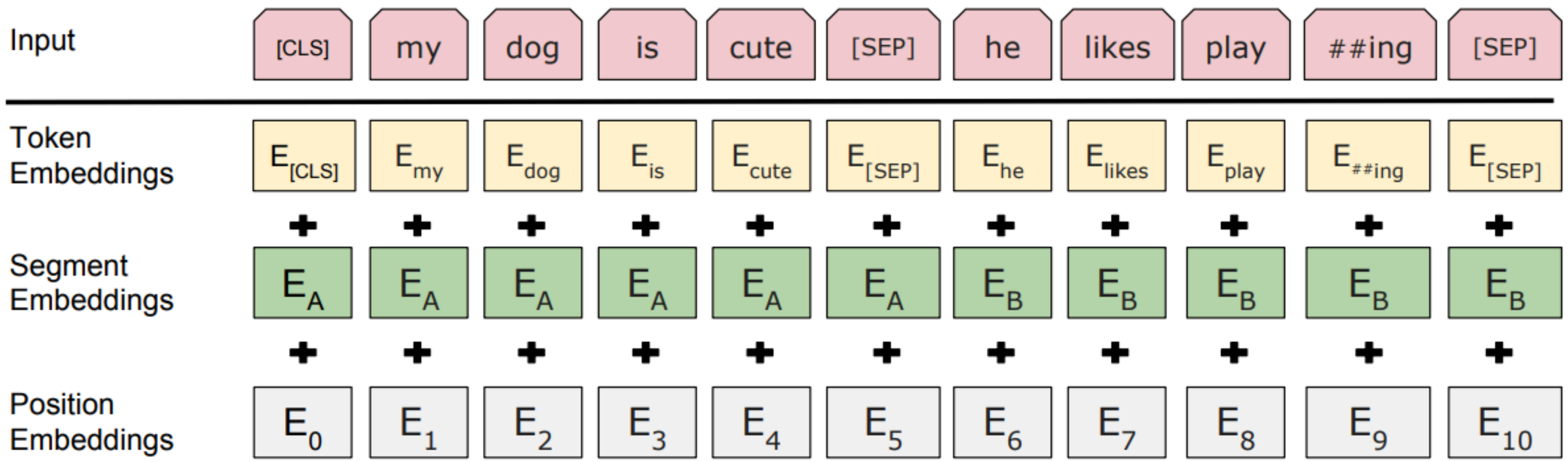


- BERT was trained to predict whether one chunk follows the other or is randomly sampled.
  - [CLS] and [SEP] tokens
  - [SEP] is used for next sentence prediction - do these sentences follow each other?
  - [CLS] for text classification / connection to fine-tuning

BERT: Training Details

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params.
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
    - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - "Pretrain once, finetune many times."

BERT: Contextual Embeddings



- BERT results in contextual embeddings
  - Embeddings for tokens in context, not just type embeddings like word2vec, GloVe
  - Can be used for measuring the semantic similarity of two words in context
  - Useful in linguistic tasks that require *precise* models of word meaning

USC Viterbi

BERT: Results

- BERT was massively popular and hugely versatile; finetuning BERT led to new state-of-the-art results on a broad range of tasks

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Various Text Classification tasks like sentiment classification

BERT: Overview

- [SEP]: Later work has argued this “next sentence prediction” is not necessary
- RoBERTa: A variant of BERT that was just better trained (careful hyper parameter optimization, etc.)
  - In general, more compute, more data can improve pretraining even when not changing the underlying Transformer encoder
- Results in contextual embeddings
- Key Limitations:
  - Cannot be used for generation. No pretraining encoders can be used for autoregressive generation very naturally
    - There are clunky ways in which you could try…but not a natural fit
    - For this, we need to have a decoder!