Lecture 9: Transformer Language Model (conts.) & Pretraining

Instructor: Xiang Ren

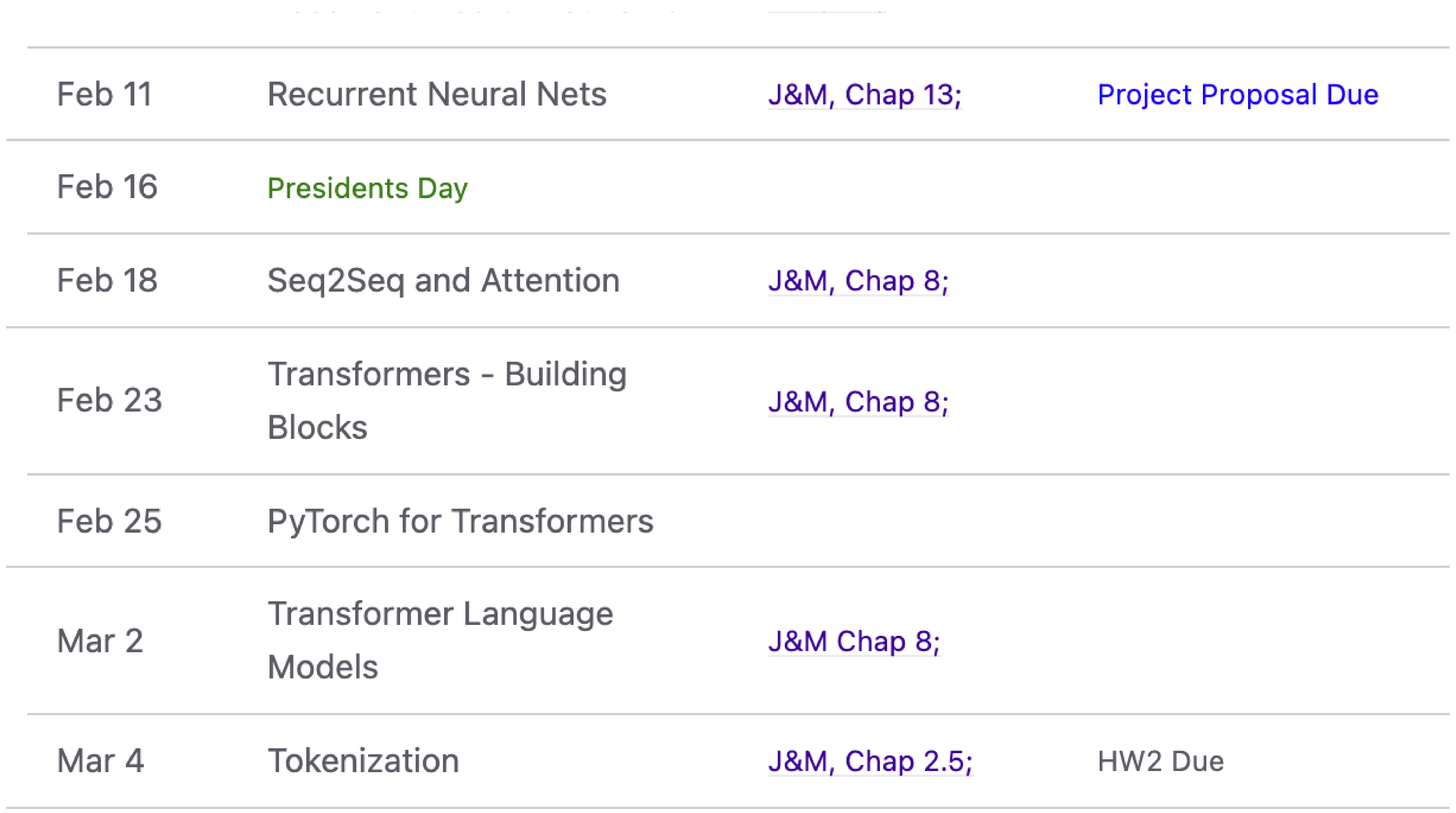
USC CSCI 444 NLP

2026 Spring

Slides mostly adapted from Dan Jurafsky, some from Mohit Iyyer

Logistics / Announcements

- Project Proposal graded by end of the week
- HW1 grades released EOD



Feb 11	Recurrent Neural Nets	J&M, Chap 13;	Project Proposal Due
Feb 16	Presidents Day		
Feb 18	Seq2Seq and Attention	J&M, Chap 8;	
Feb 23	Transformers - Building Blocks	J&M, Chap 8;	
Feb 25	PyTorch for Transformers		
Mar 2	Transformer Language Models	J&M Chap 8;	
Mar 4	Tokenization	J&M, Chap 2.5;	HW2 Due

USC Viterbi

Recap: RNN & Autoregressive Generation & Seq2seq model

Recurrent Neural Net Language Models

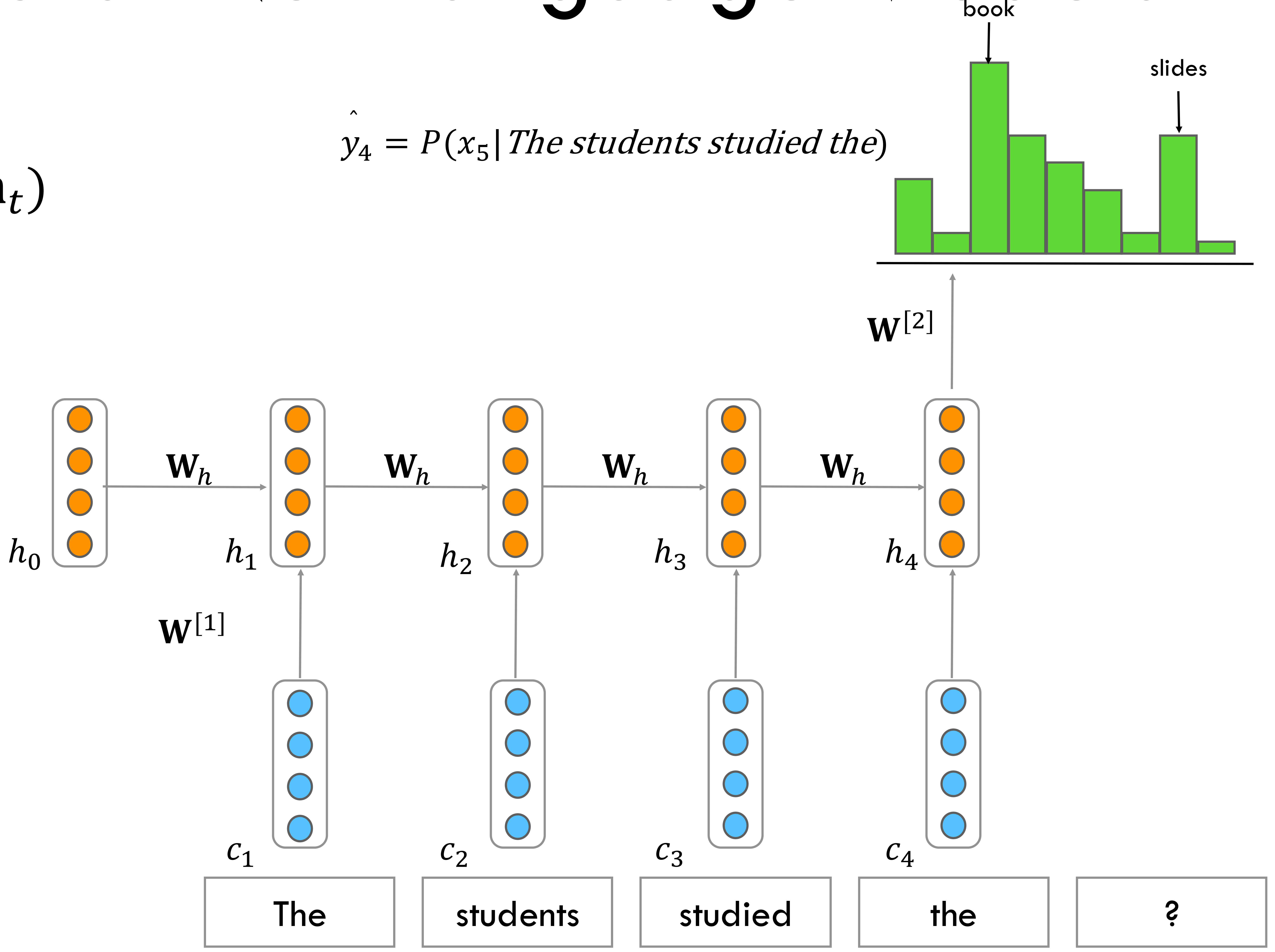
Output layer: $\hat{\mathbf{y}}_t = softmax(\mathbf{W}^{[2]}\mathbf{h}_t)$

Hidden layer: $\mathbf{h}_t = g(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}^{[1]}\mathbf{c}_t)$

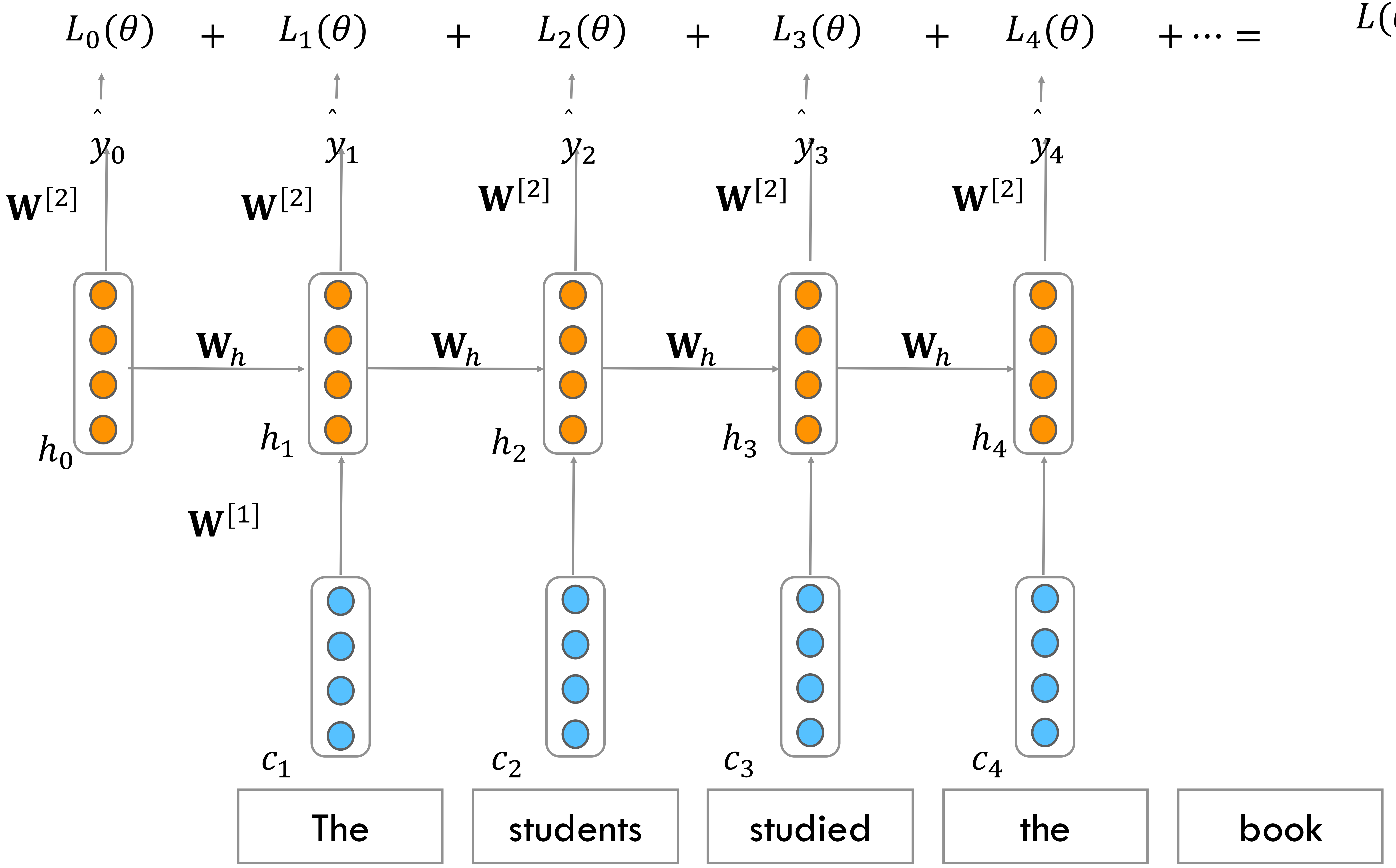
Initial hidden state: $\mathbf{h}_0$

Word Embeddings, $\mathbf{c}_i$

$\hat{y}_4 = P(x_5 | The\ students\ studied\ the)$



Loss



$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta)$$

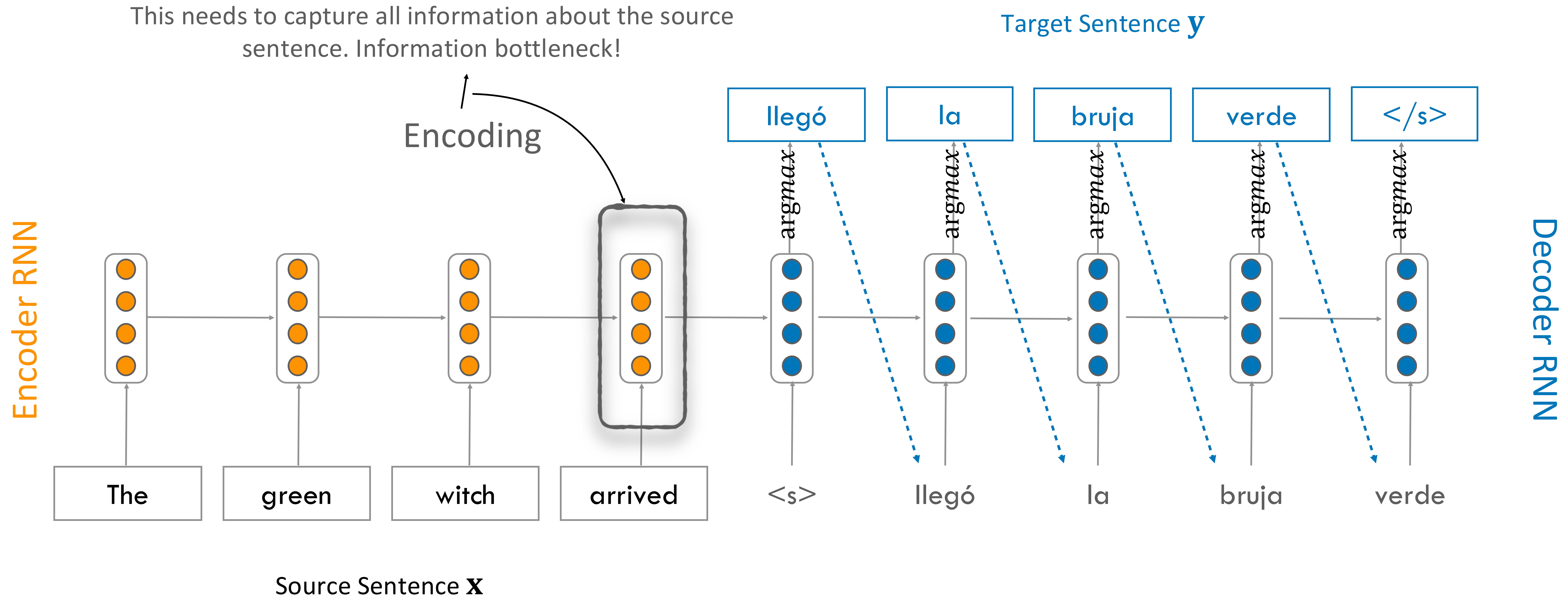
RNNLMs are Autoregressive Models

- Word generated at each time step is conditioned on the word selected by the network from the previous step
- State-of-the-art generation approaches are all autoregressive!
- Key technique: prime the generation with the most suitable context

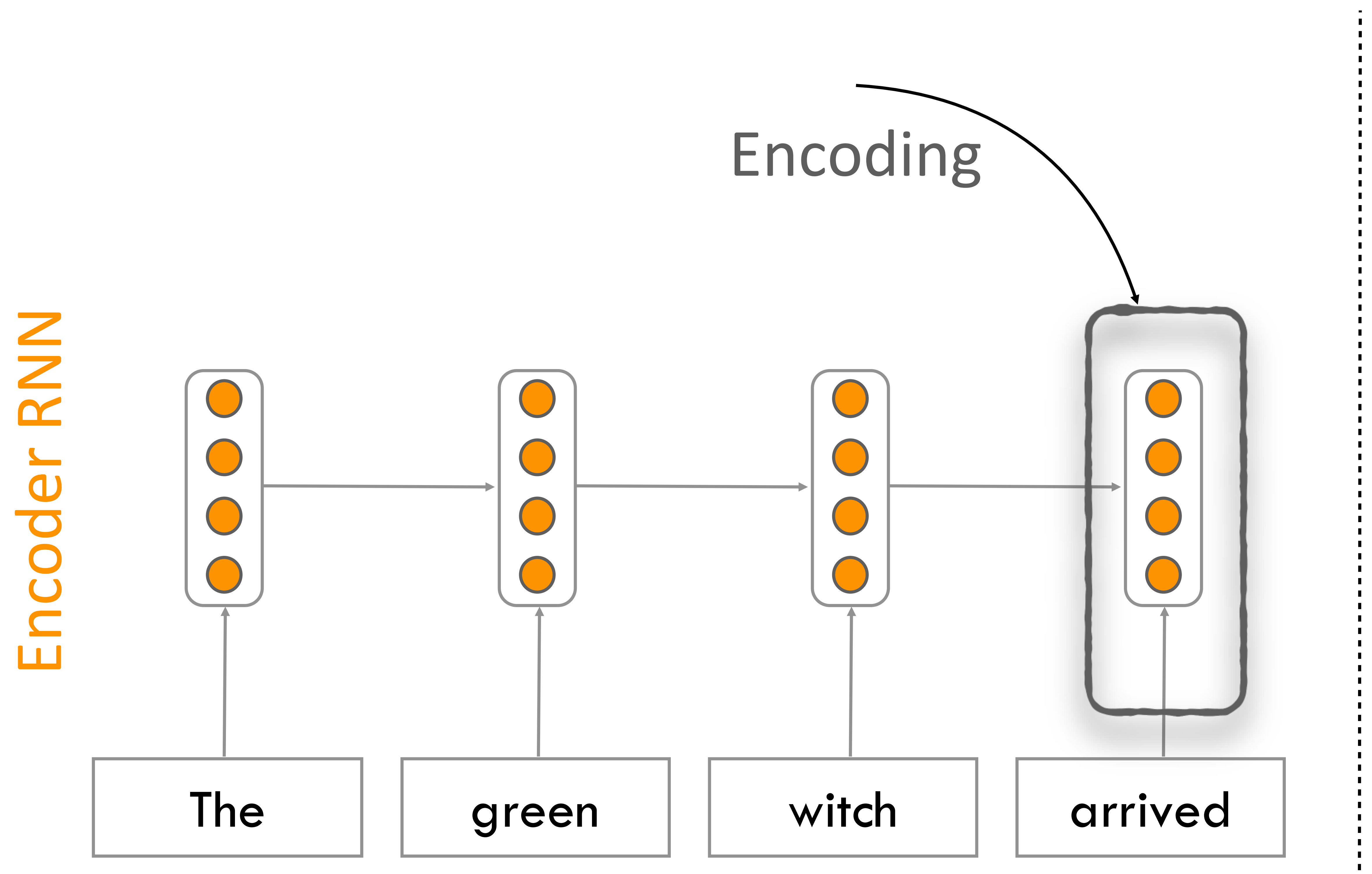
Can do better than <s>!

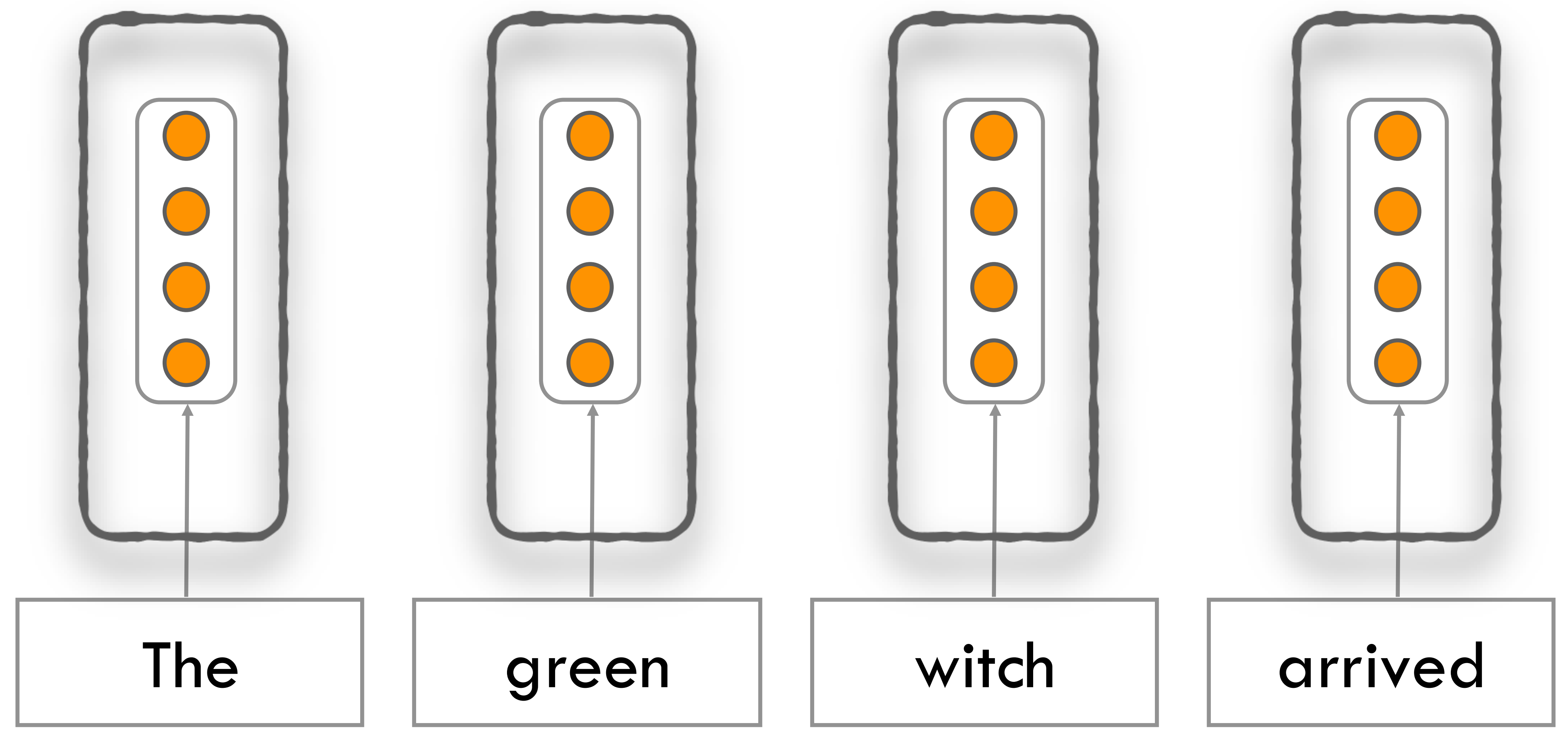
Provide rich task-appropriate context!

Encoder-Decoder Networks


This needs to capture all information about the source sentence. Information bottleneck!
Encoding
Target Sentence $\mathbf{y}$
llegó
la
bruja
verde
</s>
argmax
Encoder RNN
Decoder RNN
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence $\mathbf{x}$

Information Bottleneck: One Solution



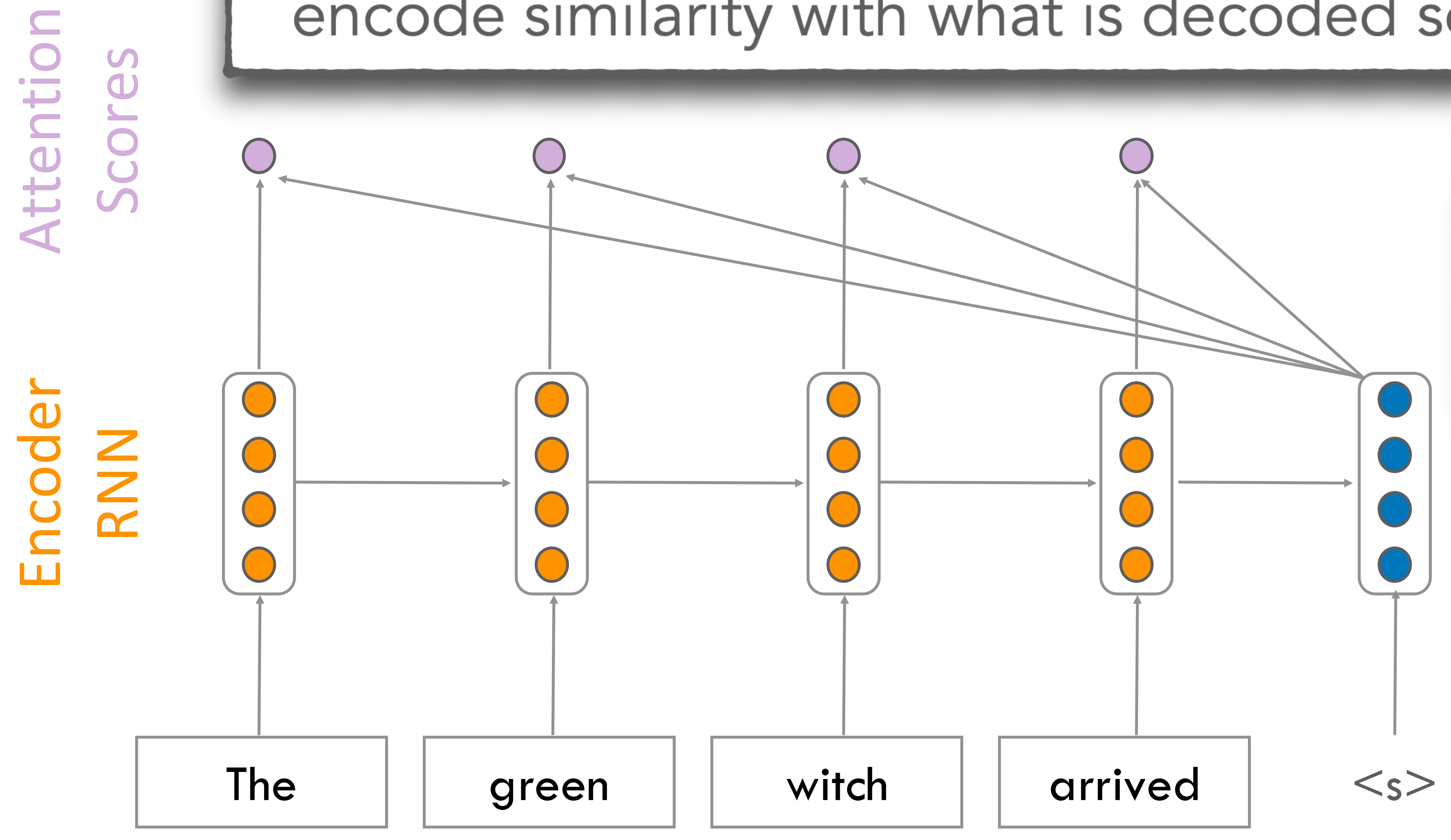


What if we had access to all hidden states?

How to create this?

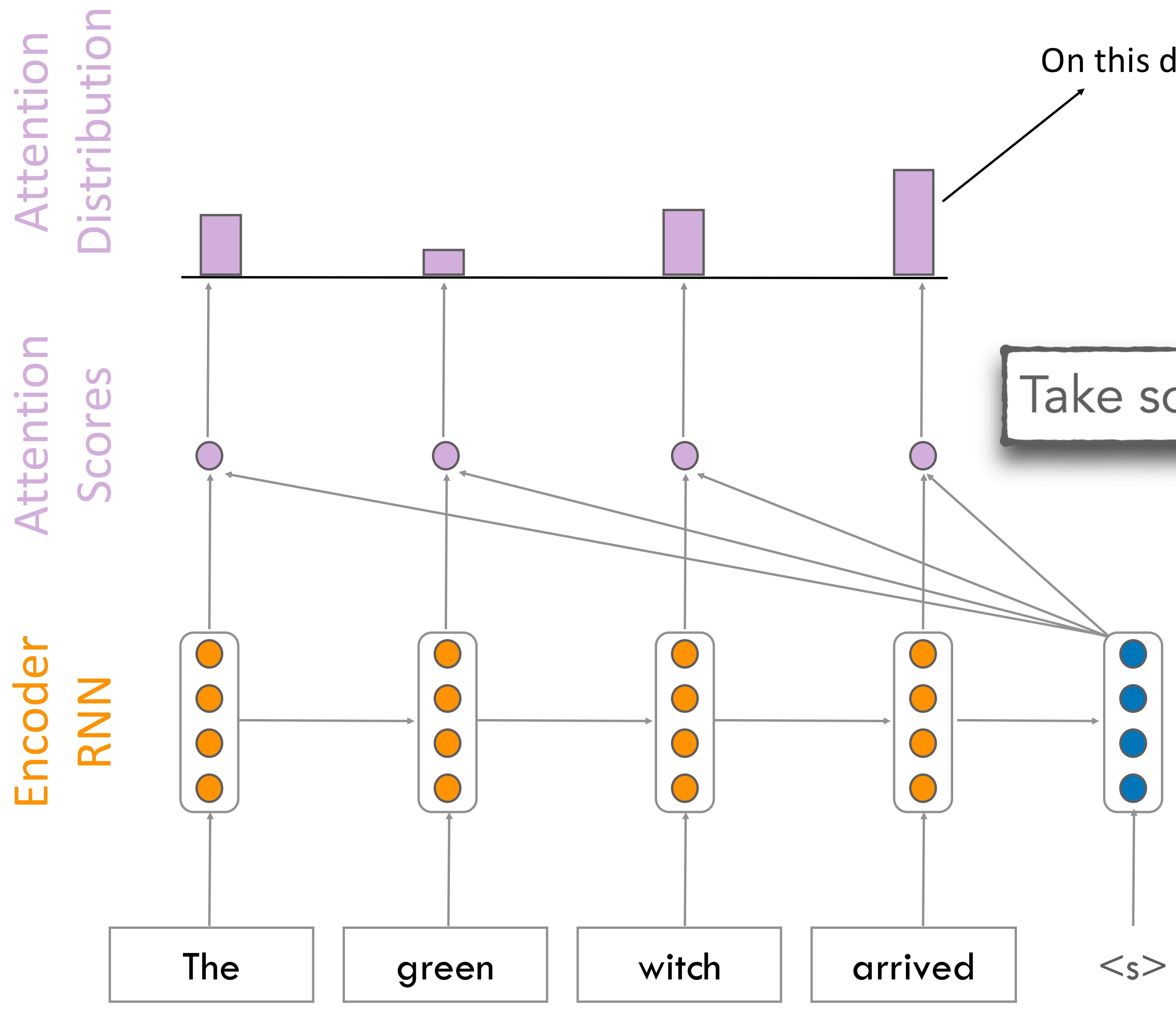
Seq2Seq with Attention

Dot product with keys (encoder hidden states) to encode similarity with what is decoded so far…



Query 1: Decoder, first time step

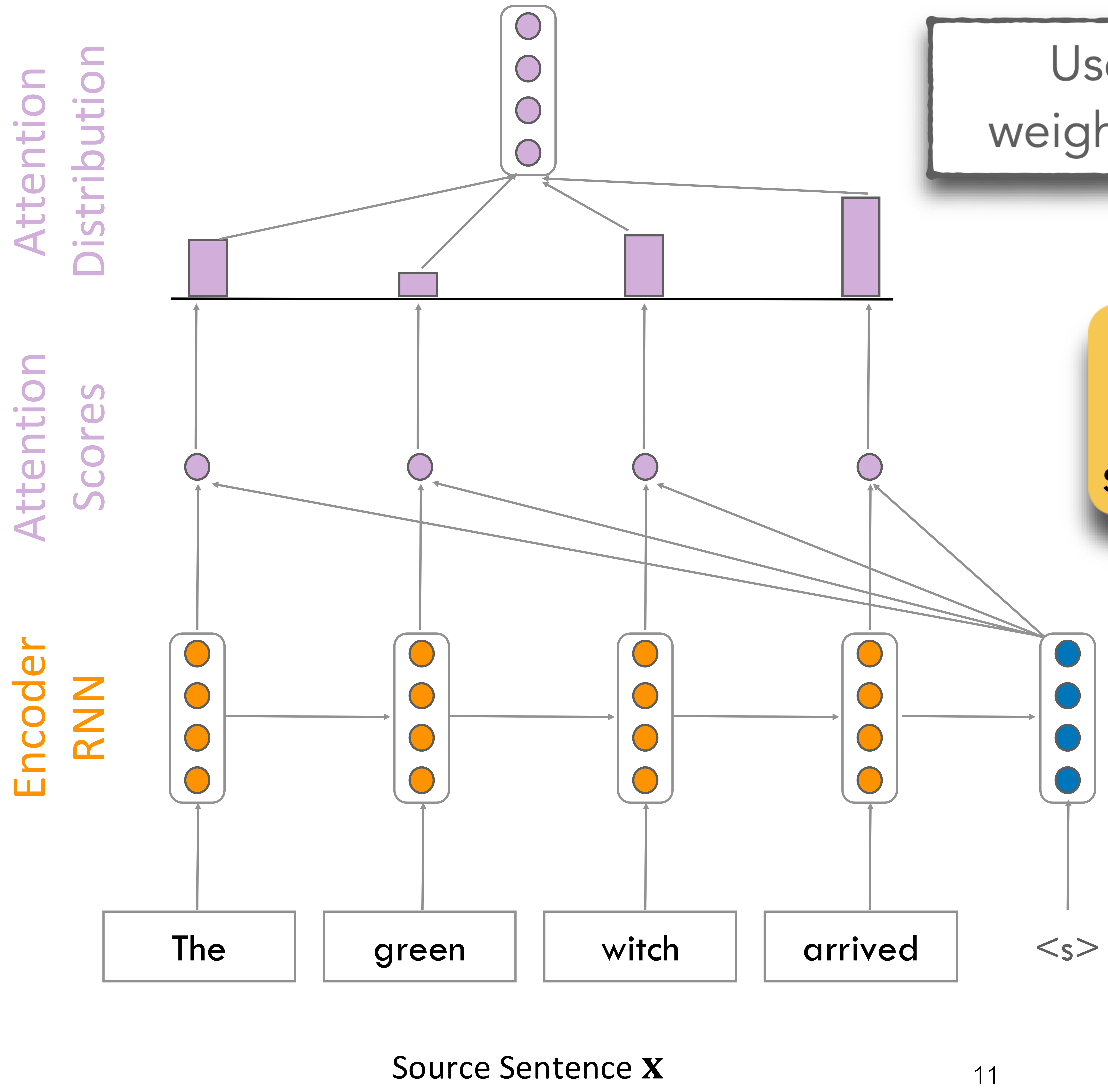
Source Sentence $\mathbf{x}$



On this decoder tilmestep we are mostly focusing on the source token “arrived”

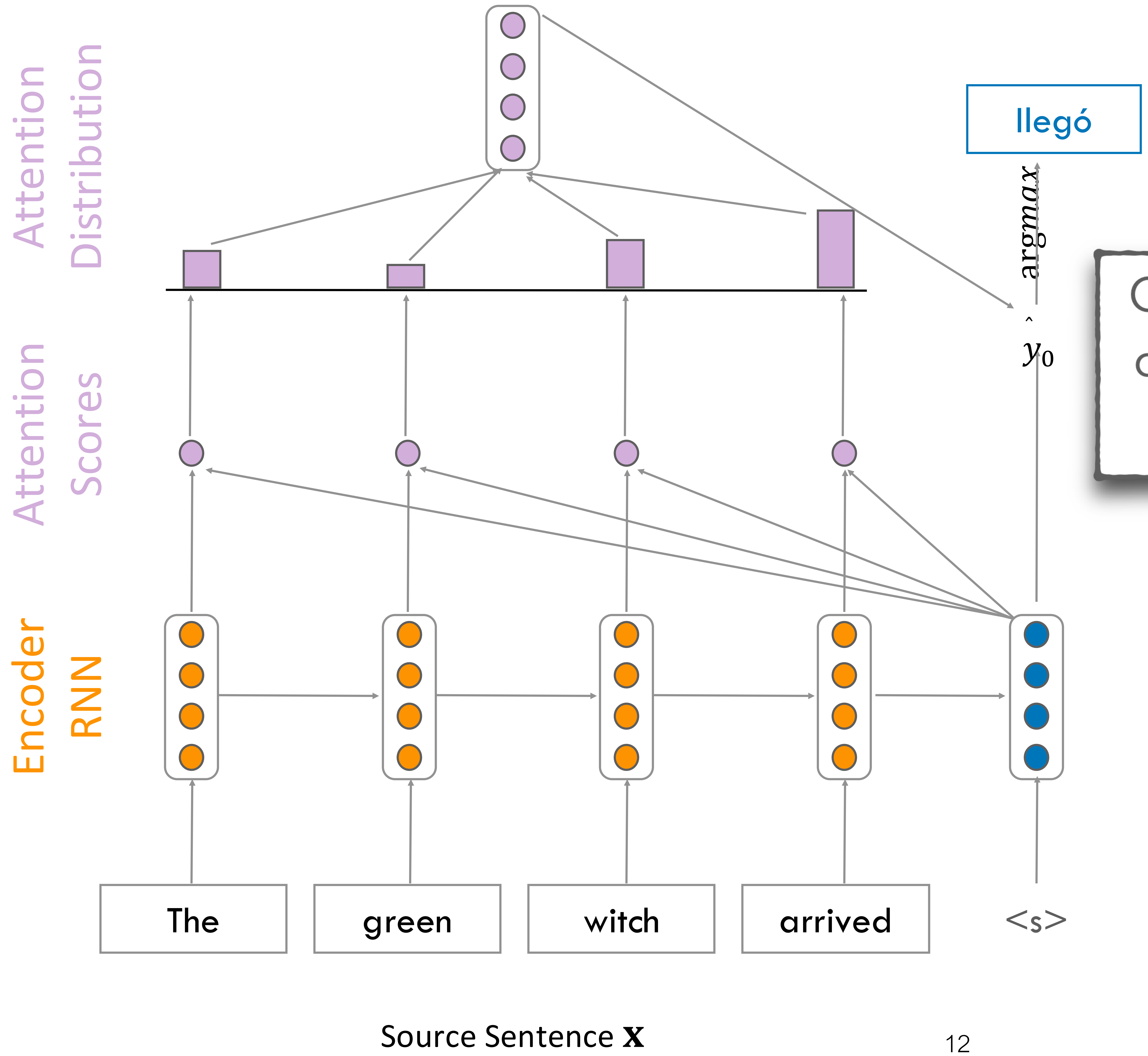
Take softmax to turn into probability distribution

Source Sentence $\mathbf{x}$

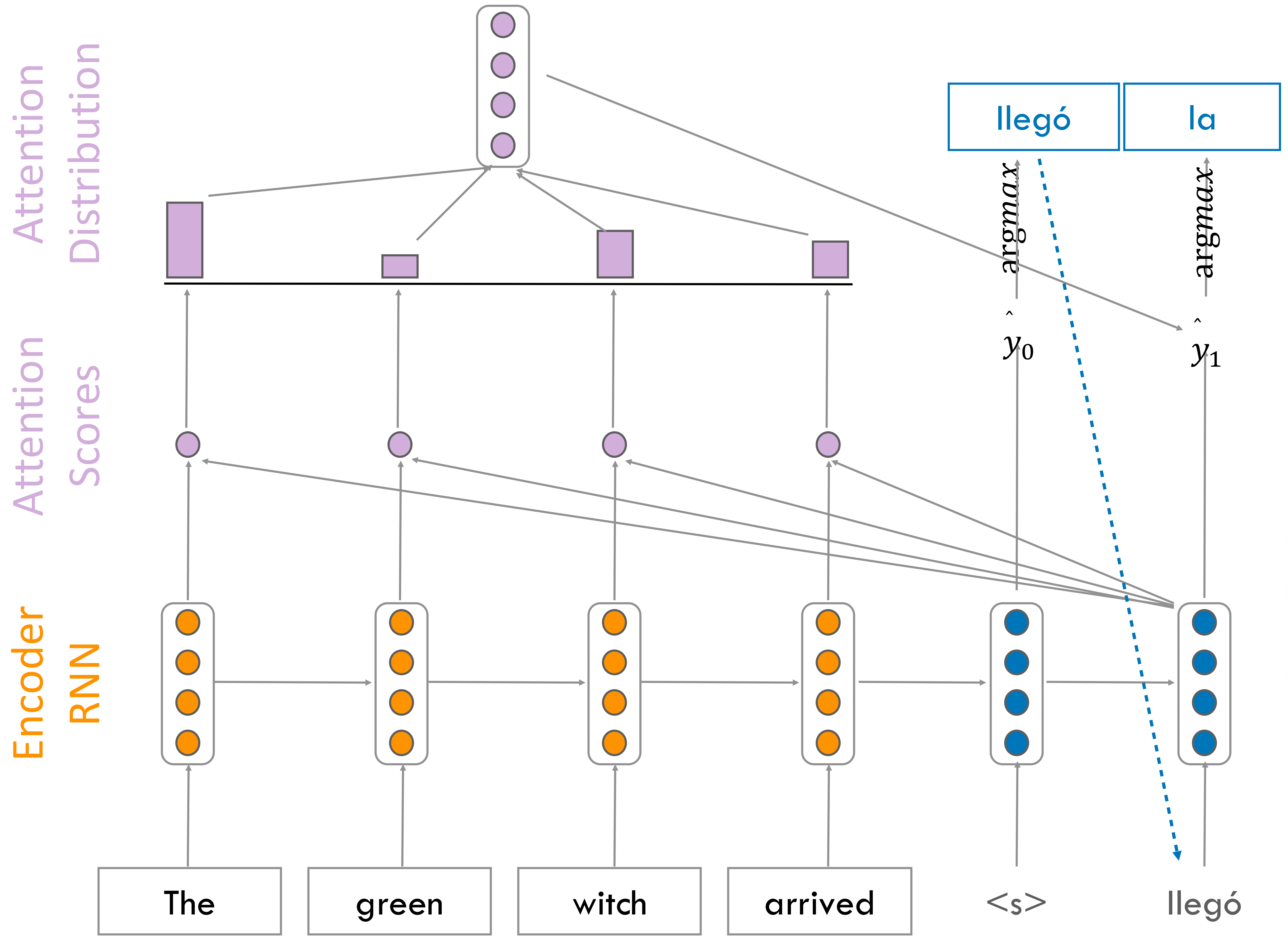


Use the attention distribution to take a weighted sum of the encoder hidden states.

The attention output mostly contains information the hidden states that received high attention.



Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_0$ as before

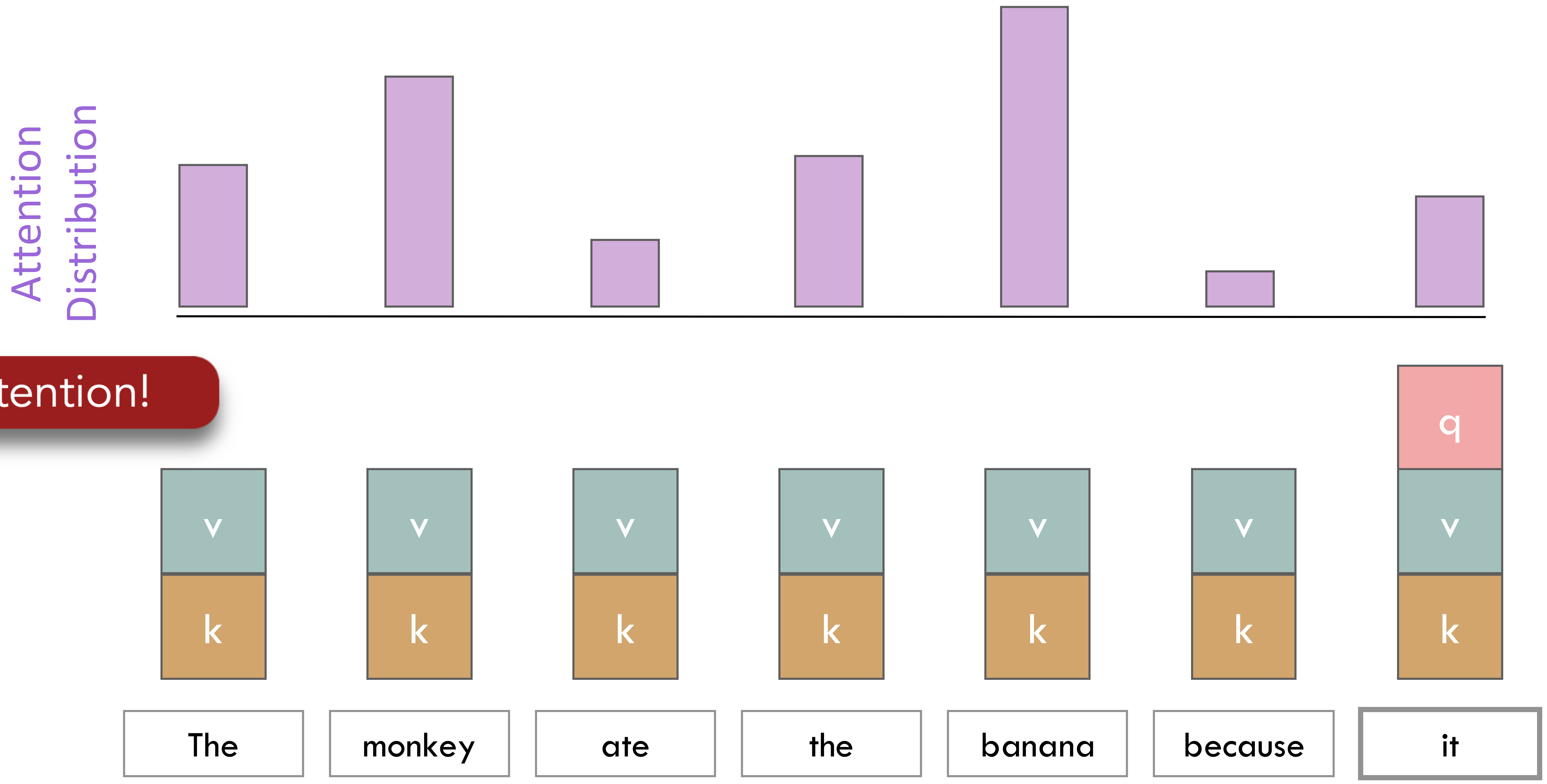


Query 2: Decoder, second time step

Source Sentence $\mathbf{x}$

Recap: Transformers Language Models

Attention in the decoder



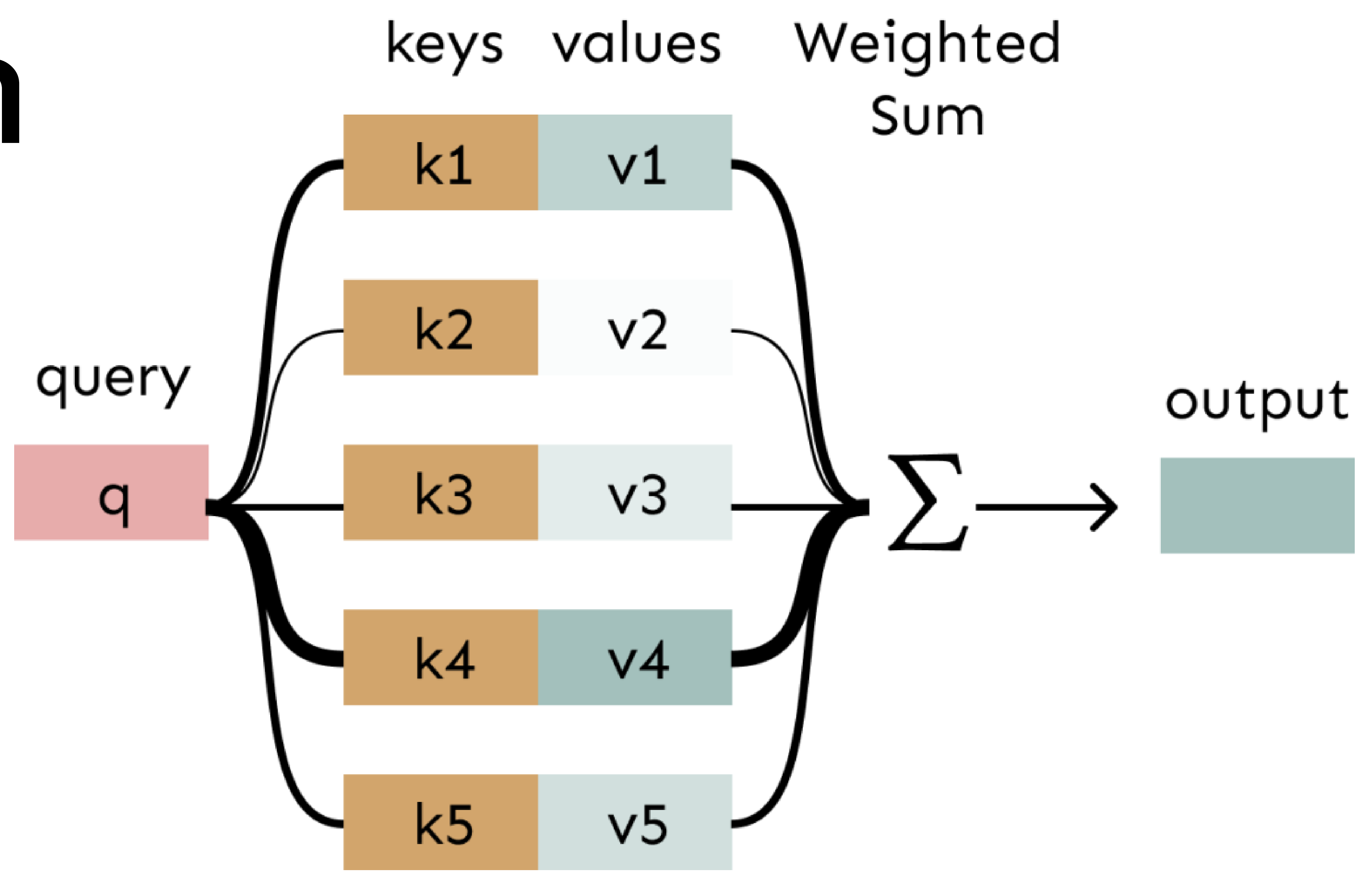
USC Viterbi

Self-Attention

Keys, Queries, Values from the same sequence

Let $\mathbf{w}_{1:N}$ be a sequence of words in vocabulary V

For each $\mathbf{w}_i$, let $\mathbf{x}_i = \mathbf{E}_{w_i}$, where $\mathbf{E} \in \mathbb{R}^{d\times V}$ is an embedding matrix.



1. Transform each word embedding with weight matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, each in $\mathbb{R}^{d\times d}$

$q_i = Qx_i$ (queries) $k_i = Kx_i$ (keys) $v_i = Vx_i$ (values)

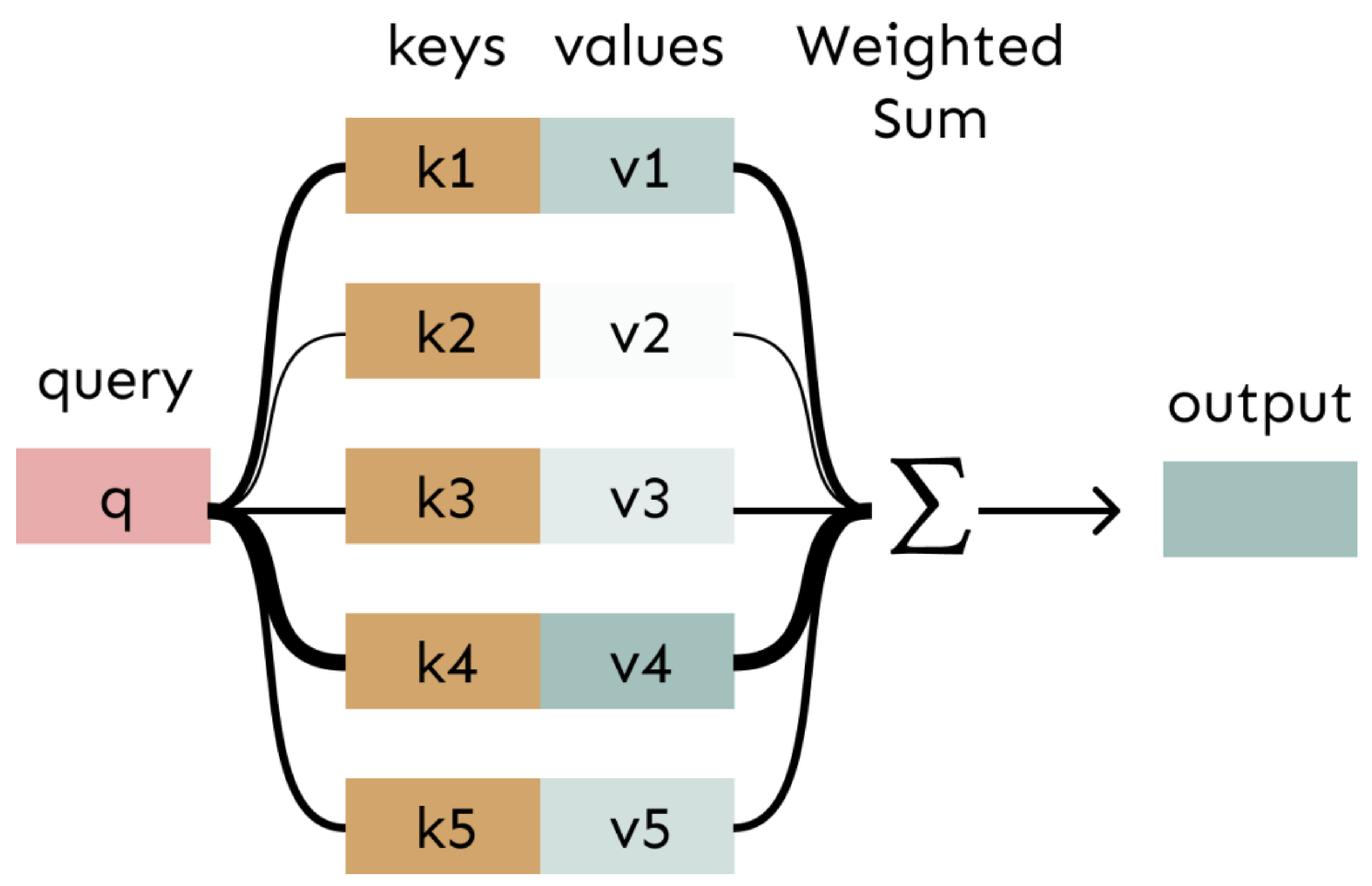
2. Compute pairwise similarities between keys and queries; normalize with softmax

$$e_{ij} = q_i^\top k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$o_i = \sum_j \alpha_{ij} v_i$$

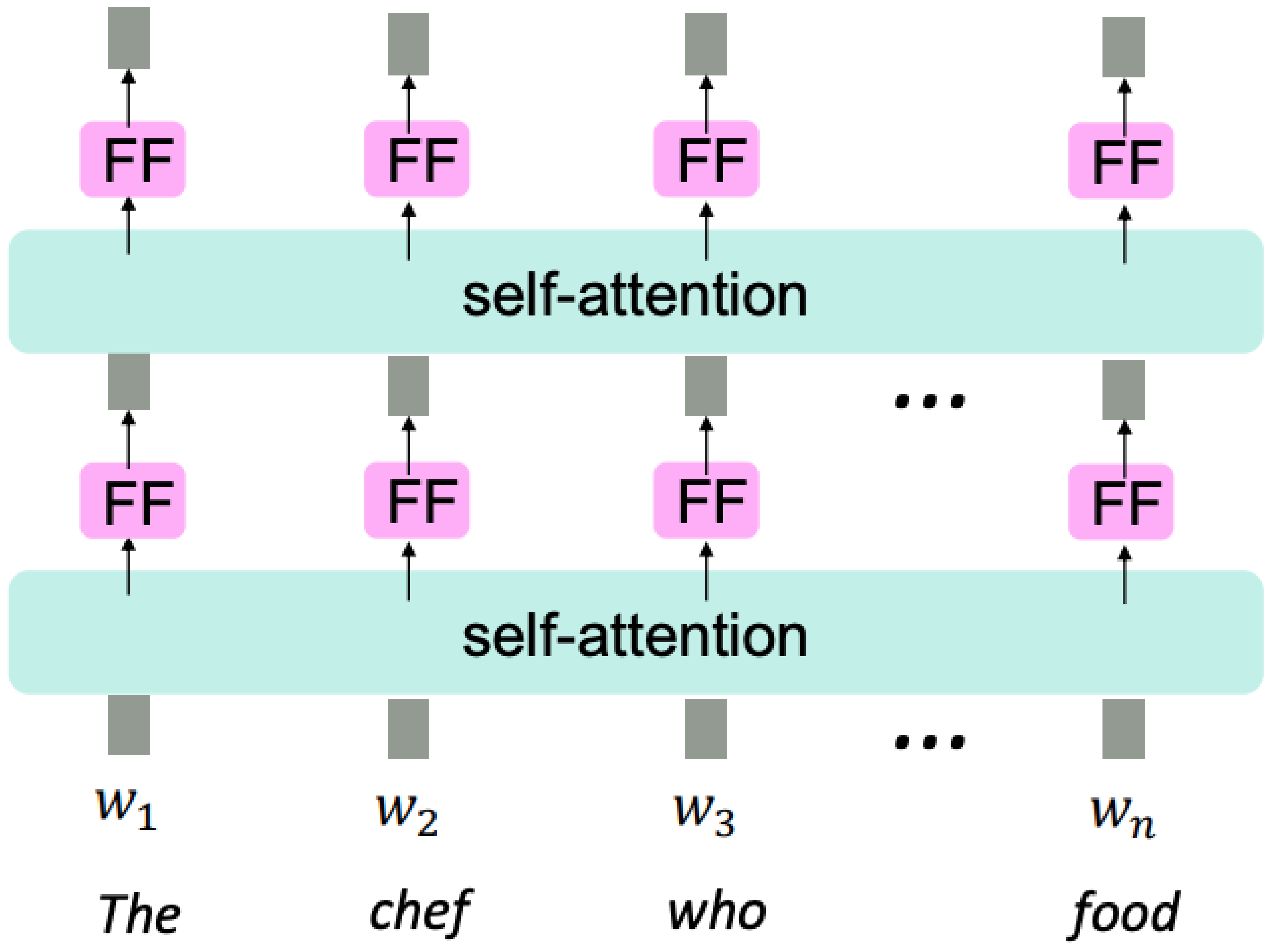
Why Self-Attention?



- Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs

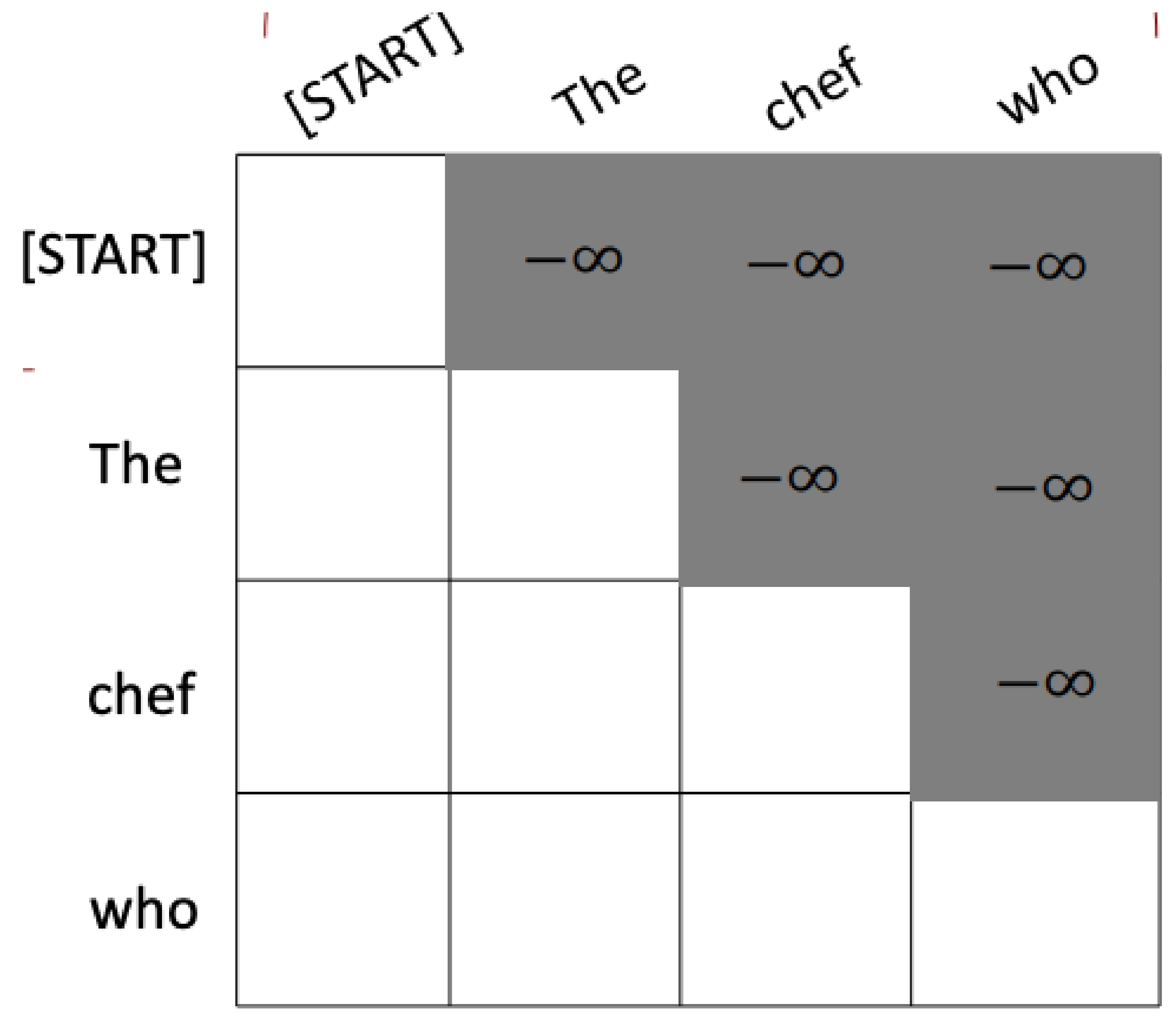
Self-Attention and Weighted Averages

- 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

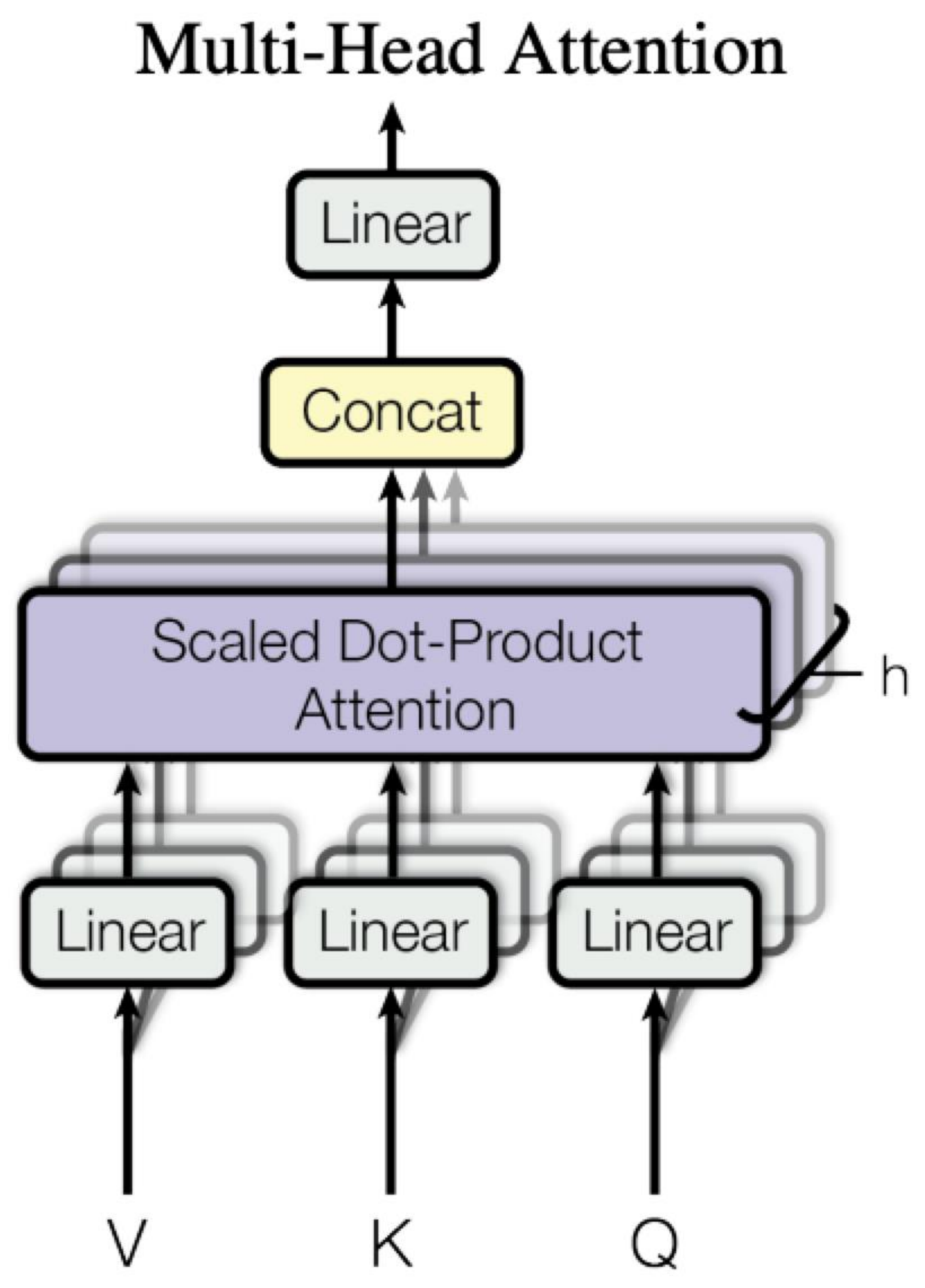
- Problem: Need to ensure we don't "look at the future" when predicting a sequence
  - To use self-attention in decoders, we need to ensure we can't peek at the future.
- Solution: To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$



$$\begin{array}{c|cccc} & \text{[START]} & \text{The} & \text{chef} & \text{who} \\ \hline \text{[START]} & & -\infty & -\infty & -\infty \\ \text{The} & & & -\infty & -\infty \\ \text{chef} & & & & -\infty \\ \text{who} & & & & \end{array}$$

Multi-headed attention

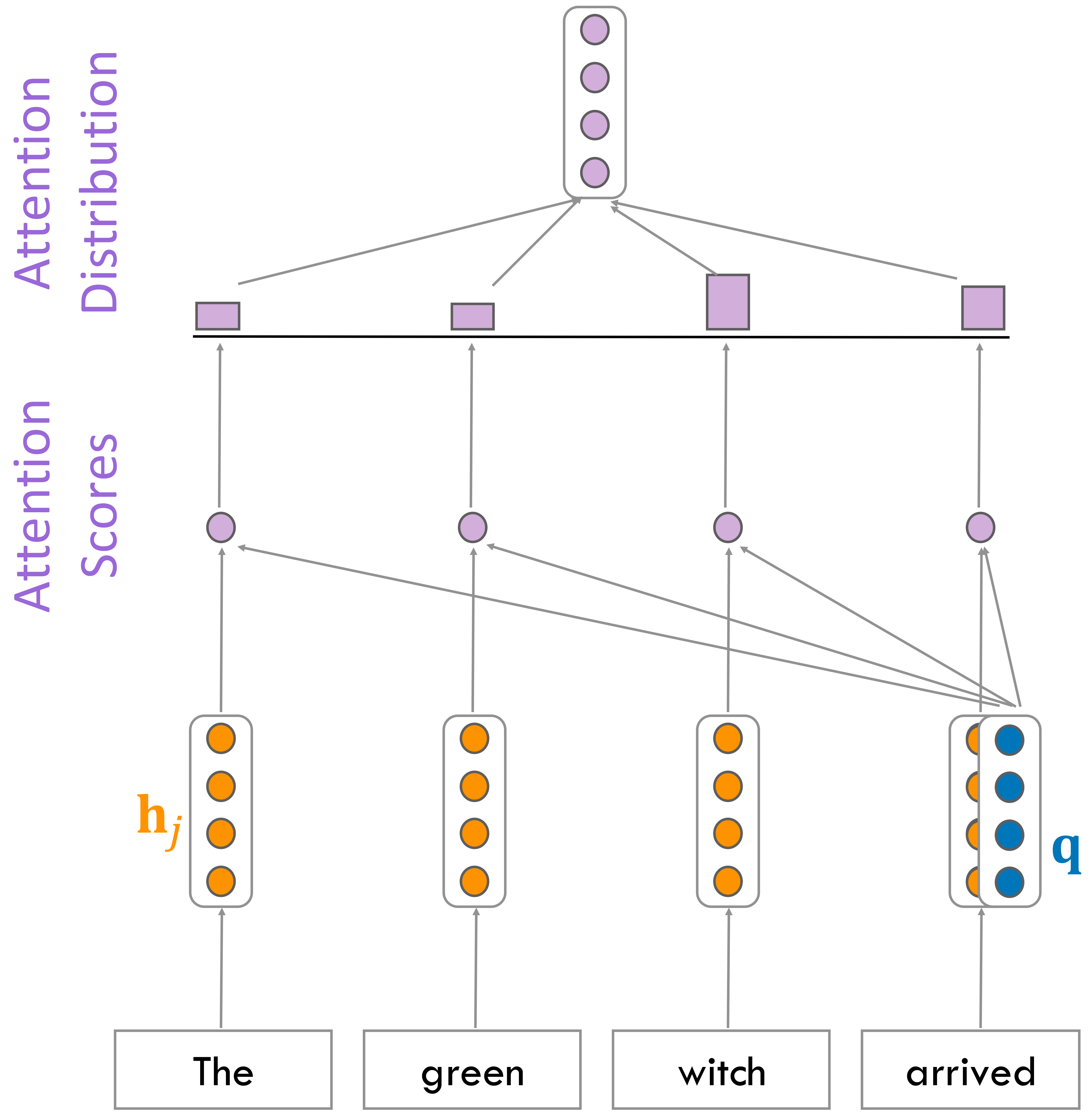
- What if we want to look in multiple places in the sentence at once?
- We'll define multiple attention "heads" through multiple $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ matrices
- Let $\mathbf{Q}_l, \mathbf{K}_l, \mathbf{V}_l$, each in $\mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention 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

Self-Attention: Order Information?

- 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.



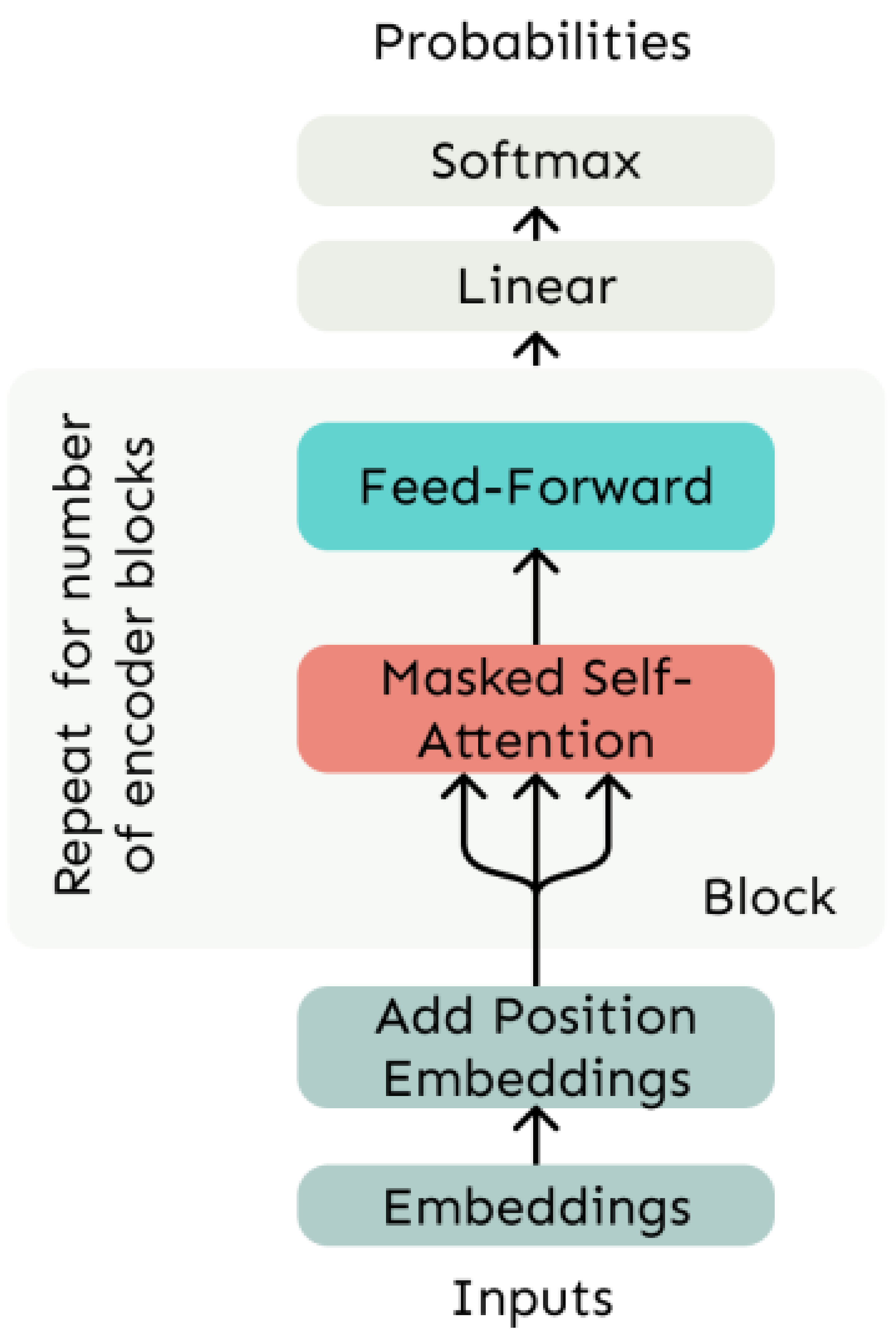
Positional Embeddings

- Maps integer inputs (for positions) to real-valued vectors
  - one per position in the entire context
- Can be randomly initialized and can let all $\mathbf{p}_i$ be learnable parameters (most common)

Putting it all together: Transformer Blocks

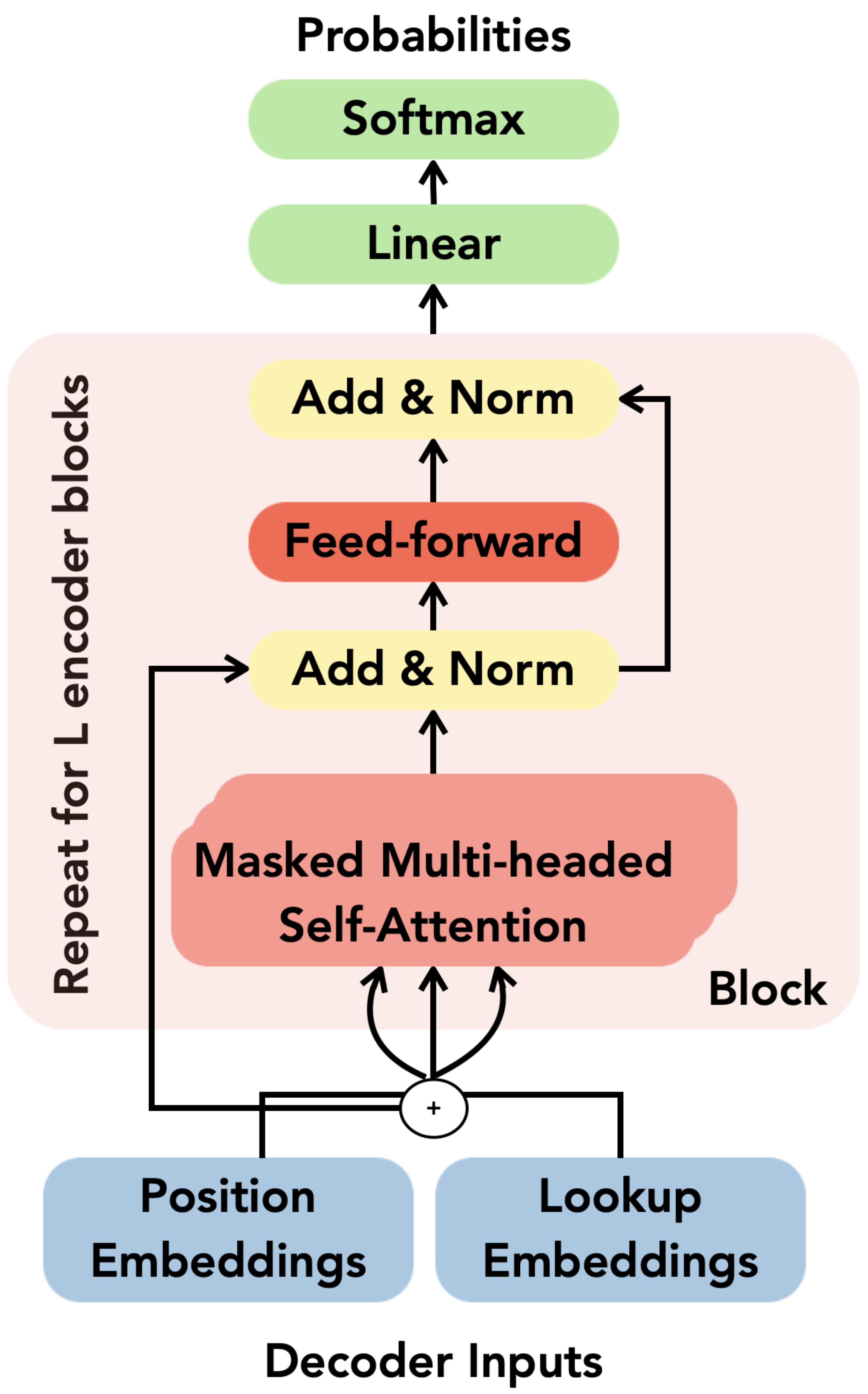
Self-Attention Transformer Building Block

- 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.



The Transformer Decoder

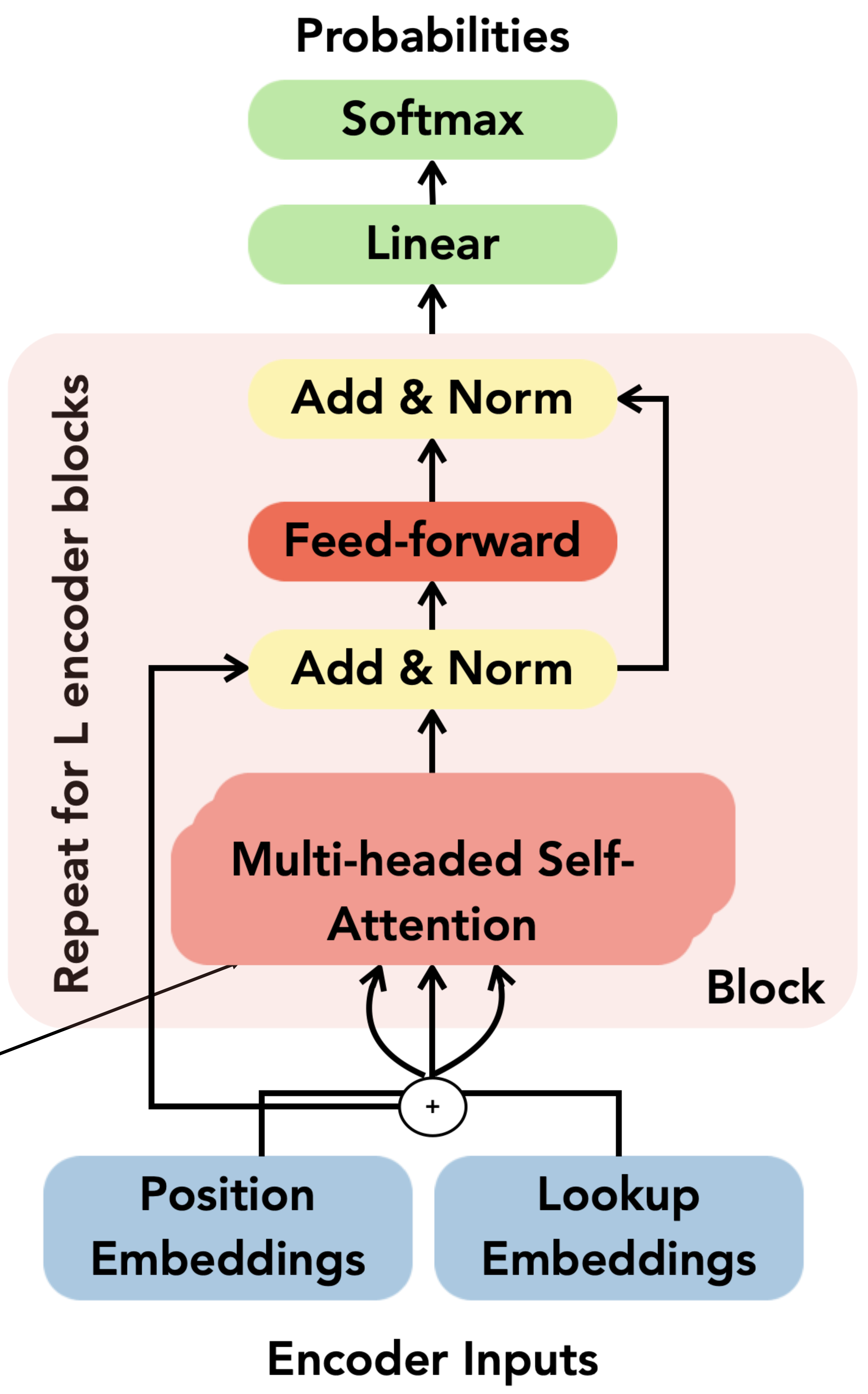
- The Transformer Decoder is a stack of Transformer Decoder Blocks.
- Each Block consists of:
  - Self-attention
  - Add & Norm
  - Feed-Forward
  - Add & Norm
- Output layer is as always a softmax layer



The Transformer Encoder

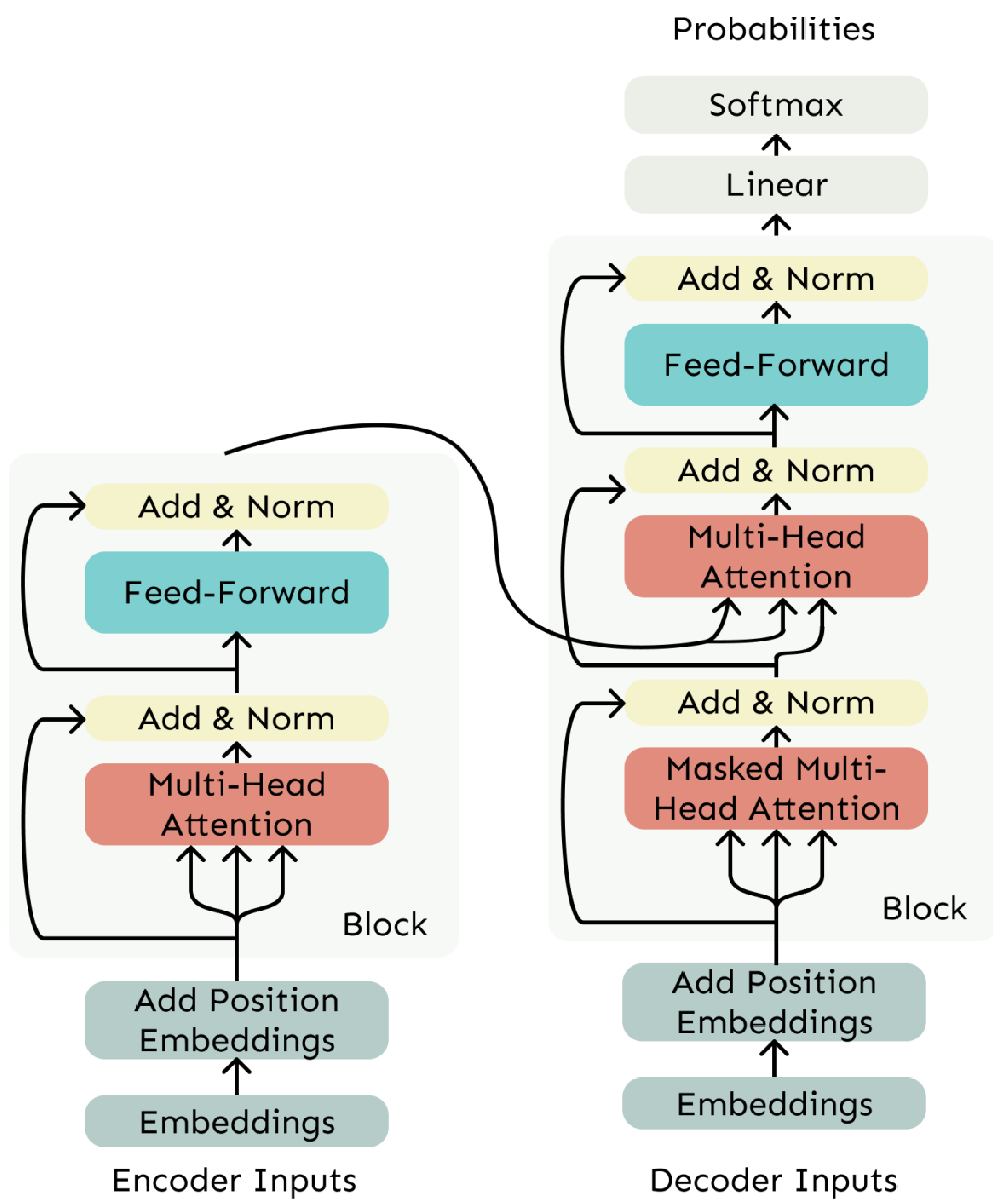
- What if we want **bidirectional** context, i.e. both left to right as well as right to left?
- The only difference is that we **remove** the masking in the self-attention.

No Masking!



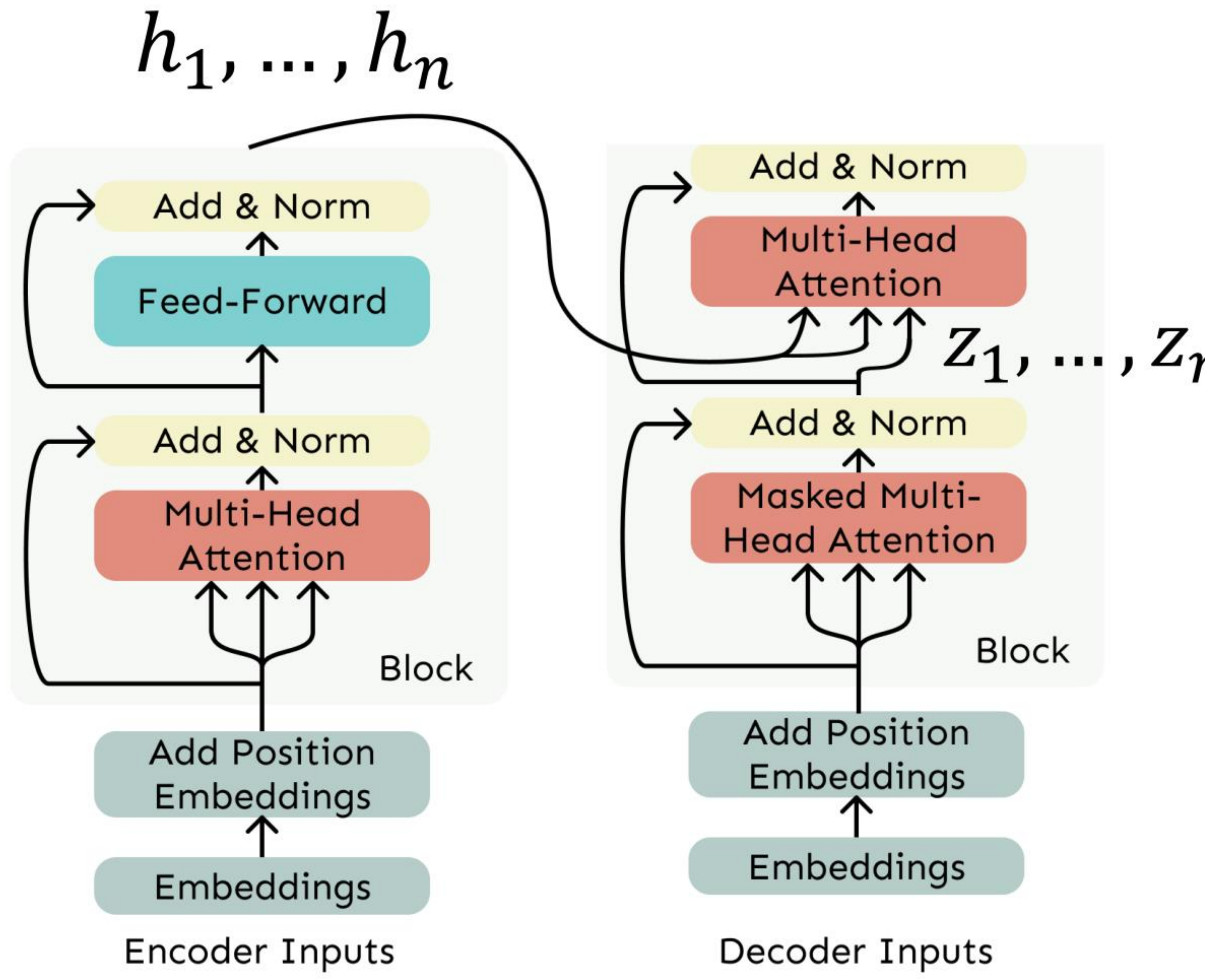
The Transformer Encoder-Decoder

- 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 $\mathbf{h}_1, \dots, \mathbf{h}_n$ be output vectors from the Transformer encoder; $\mathbf{h}_i \in \mathbb{R}^d$
- Let $\mathbf{z}_1, \dots, \mathbf{z}_n$ 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, $\mathbf{q}_i = \mathbf{Q}\mathbf{z}_i$



The Pre-training and Fine-tuning Paradigm

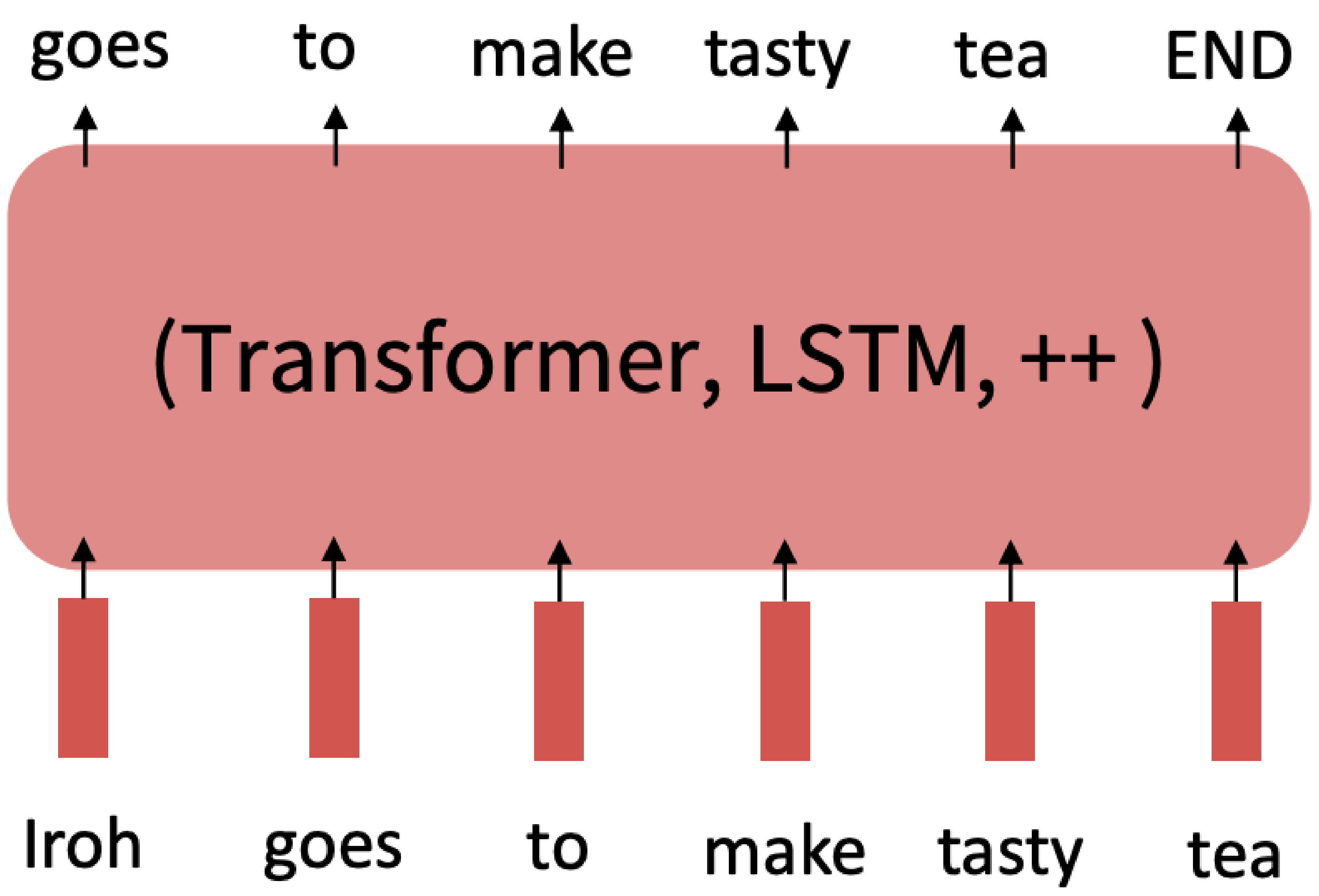
The Pretraining / Finetuning Paradigm

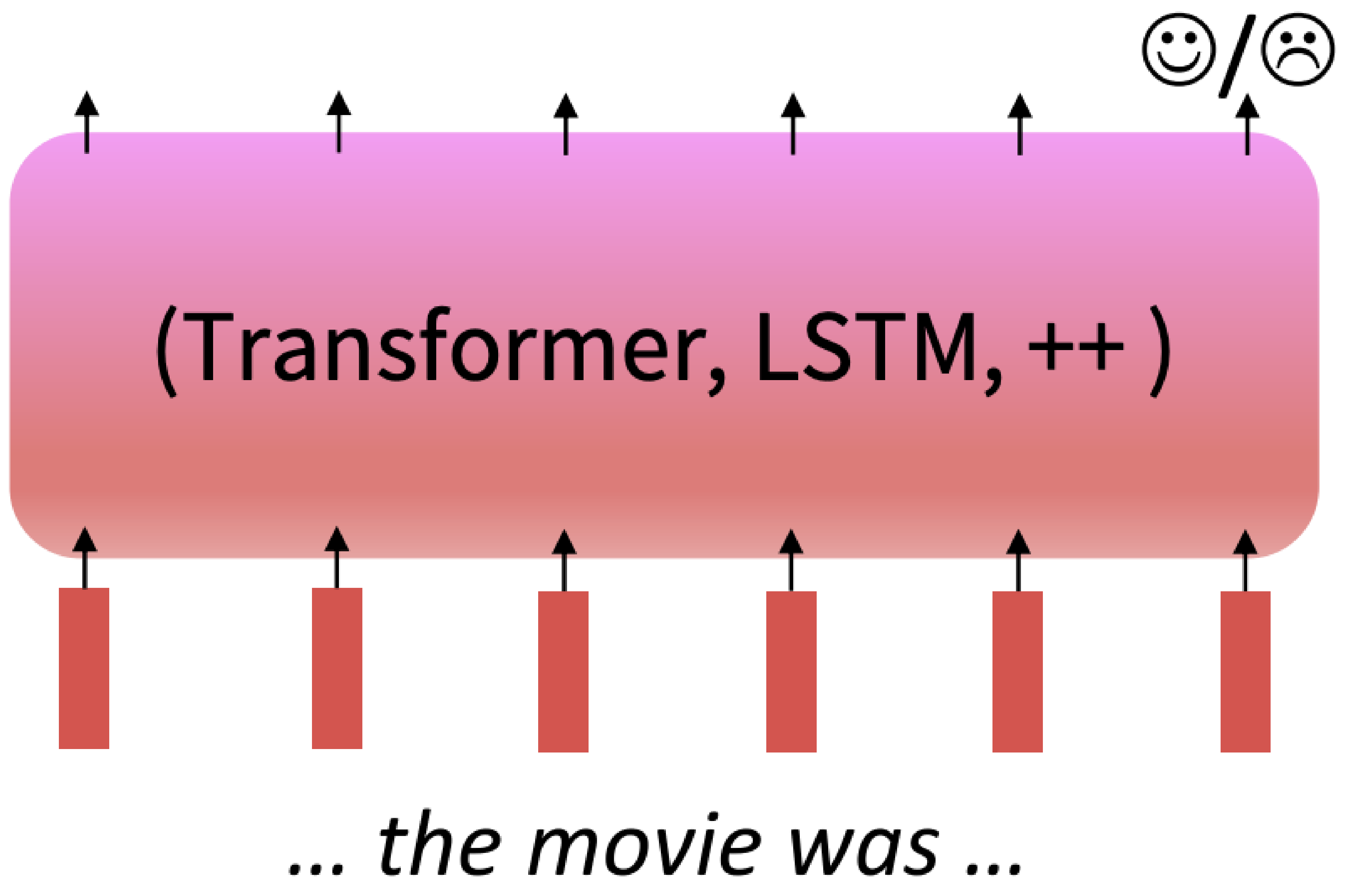
- Pretraining can improve NLP applications by serving as parameter initialization.

Key idea: "Pretrain once, finetune many times."

Step 1: Pretrain (on language corpora)
Lots of text; learn general things!

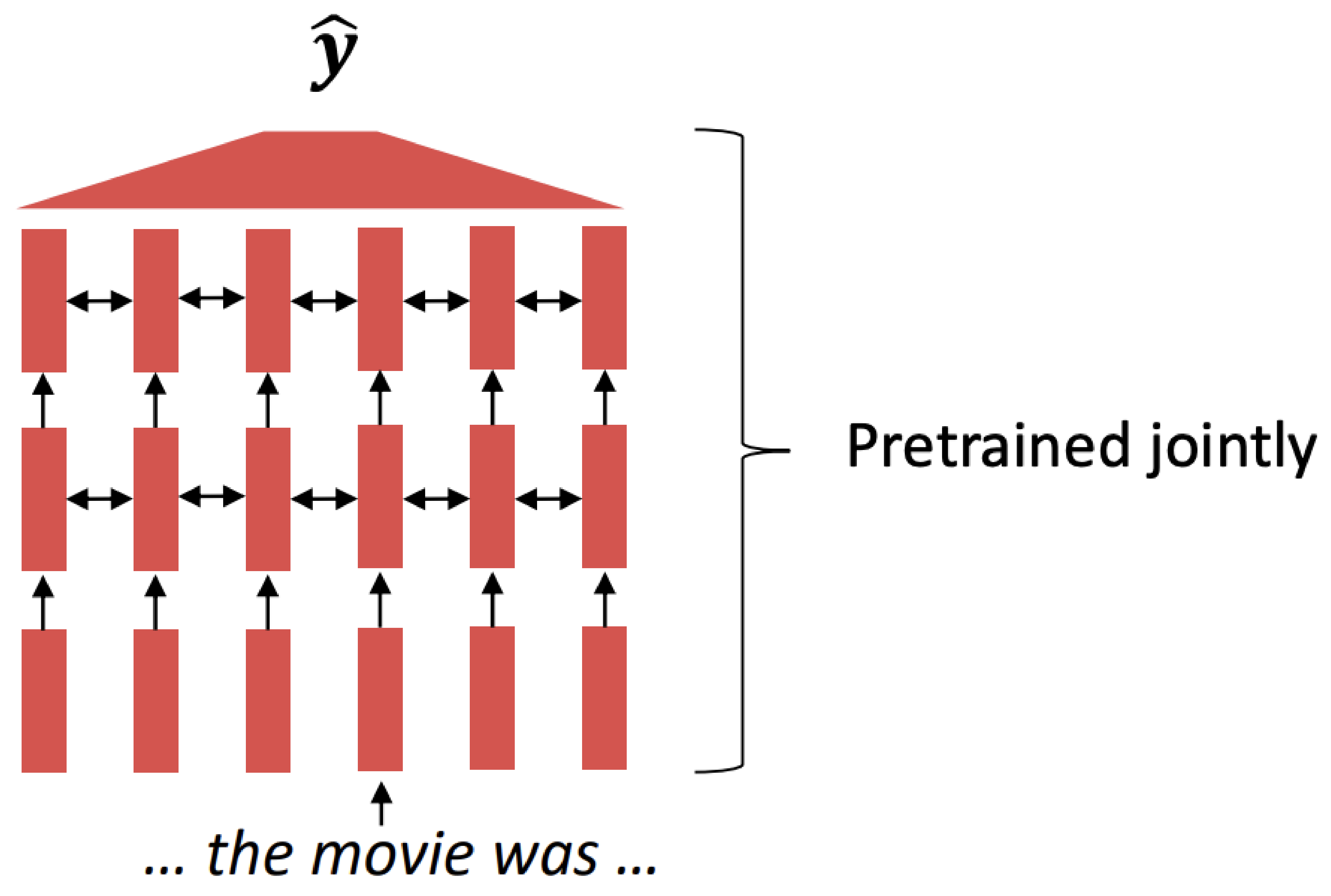
Step 2: Finetune (on your task data)
Not many labels; adapt to the task!





Pretraining Entire Models

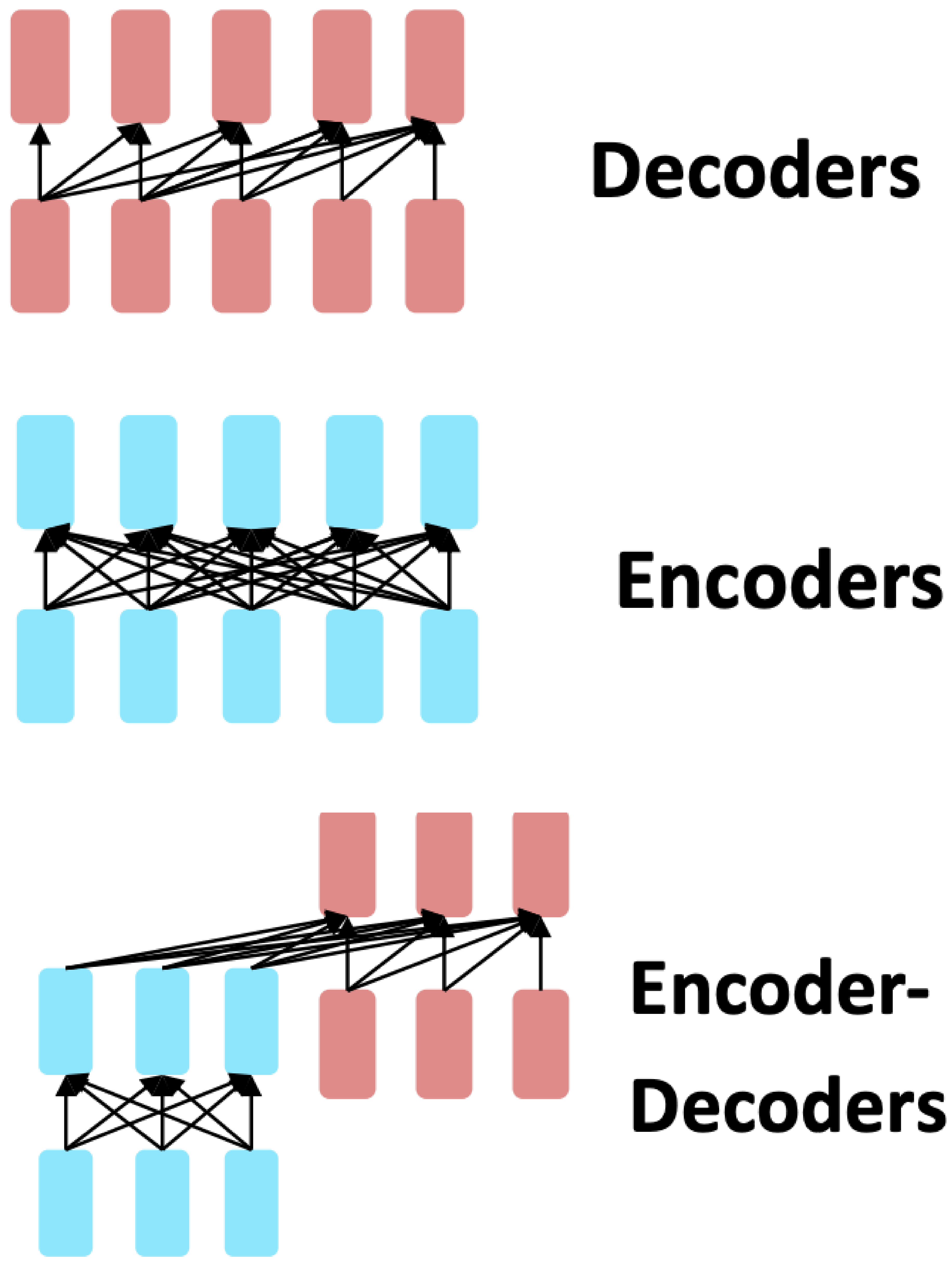
- In modern NLP:
  - All (or almost all) parameters in NLP networks are initialized via pretraining.
  - This has been exceptionally effective at building strong:
    - representations of language
    - parameter initializations for strong NLP models.
    - probability distributions over language that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



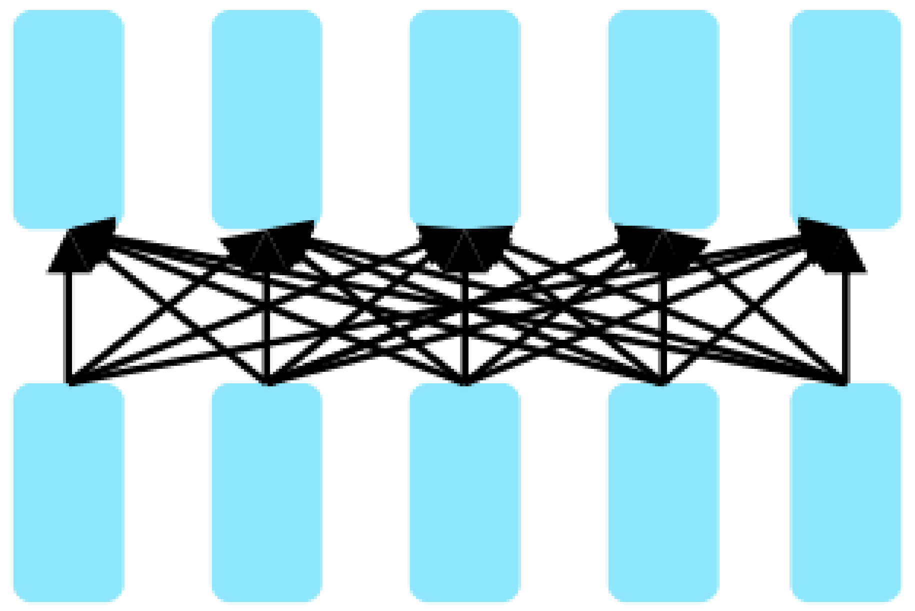
[This model has learned how to represent entire sentences through pretraining]

Pretraining

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

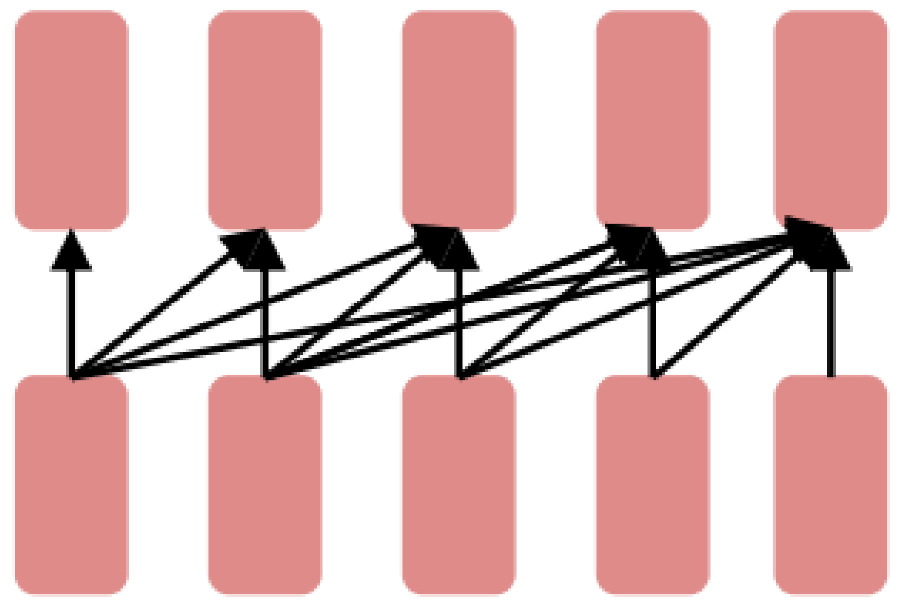


Pretraining for three types of architectures



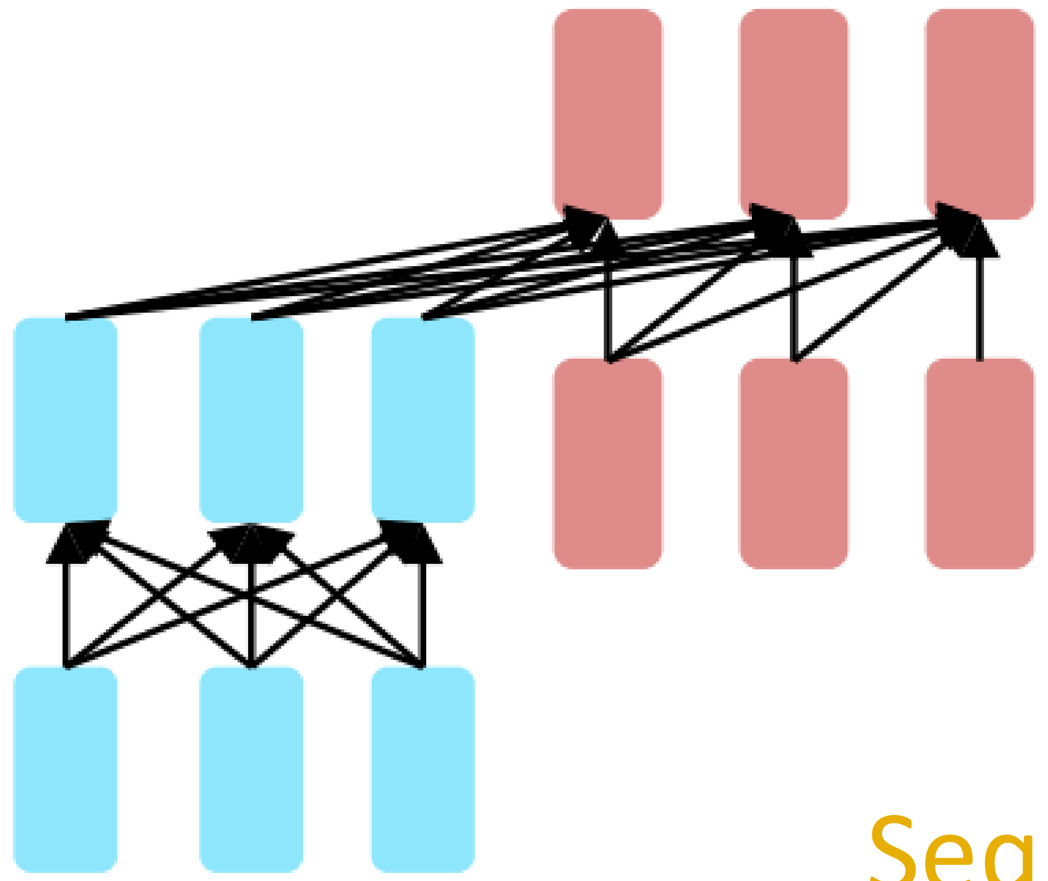
Encoders

Bidirectional Context



Decoders

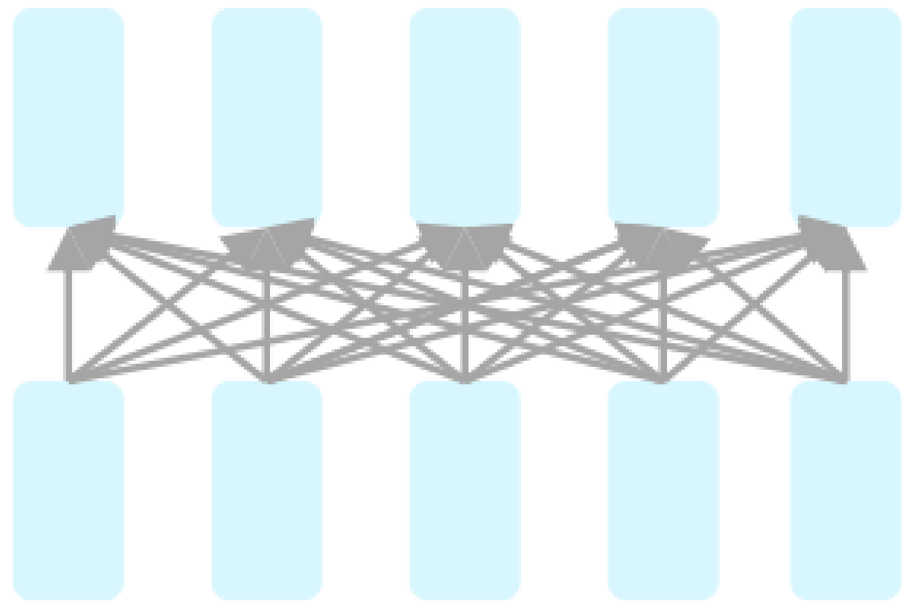
Language Models



Encoder-Decoders

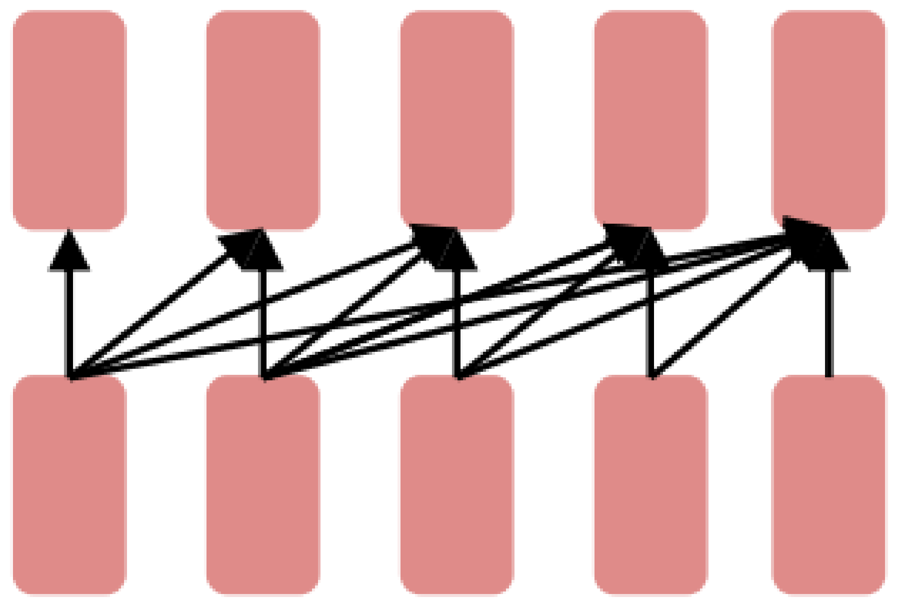
Sequence-to-sequence

Pretraining for three types of architectures



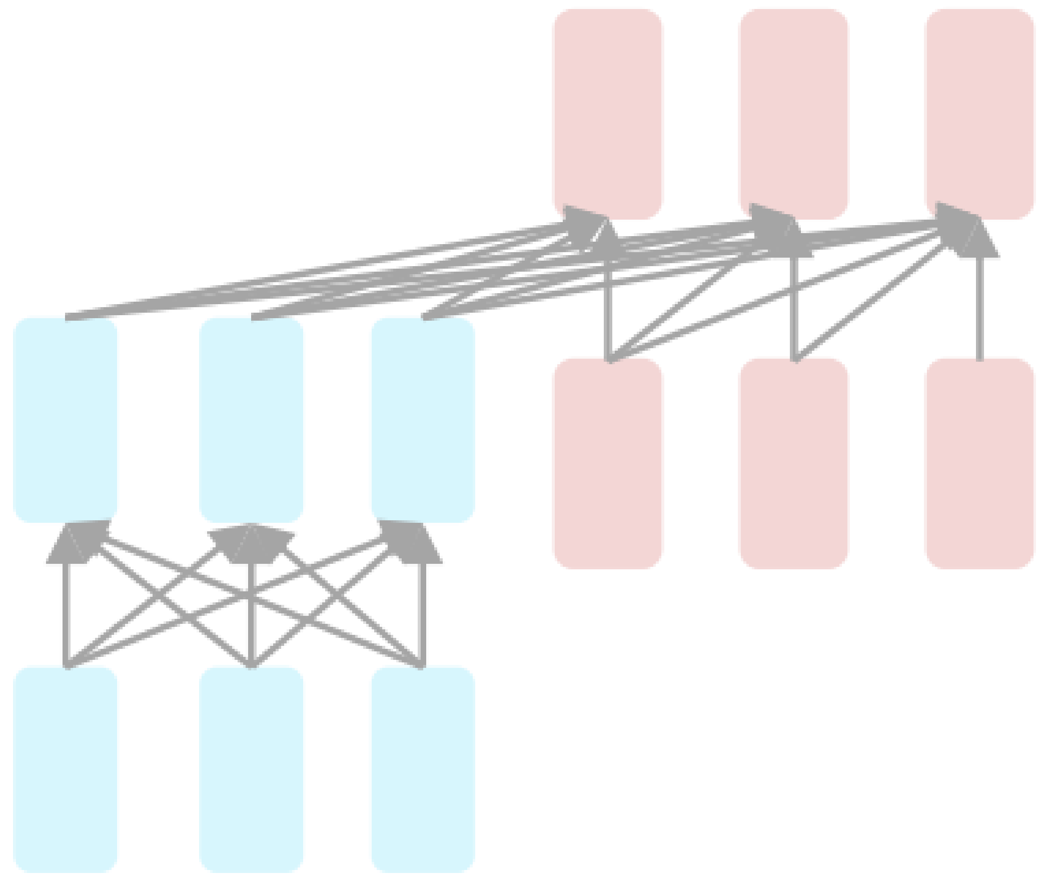
Encoders

Bidirectional Context



Decoders

Language Models



Encoder-Decoders

Sequence-to-sequence

Generative Pretrained Transformer (GPT)

- 2018's GPT was a big success in pretraining a decoder!
  - Transformer decoder with 12 layers, 117M parameters.
  - 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
  - Byte-pair encoding with 40,000 merges
  - Trained on BooksCorpus: over 7000 unique books.
    - Contains long spans of contiguous text, for learning long-distance dependencies.
  - The acronym "GPT" never showed up in the original paper; it could stand for "Generative PreTraining" or "Generative Pretrained Transformer"

[Radford et al., 2018]

Adapting GPT

- How do we format inputs to our decoder for finetuning tasks?

- Natural Language Inference: Label pairs of sentences as entailing/contradictory/neutral
  - Premise: The man is in the doorway
  - Hypothesis: The person is near the door

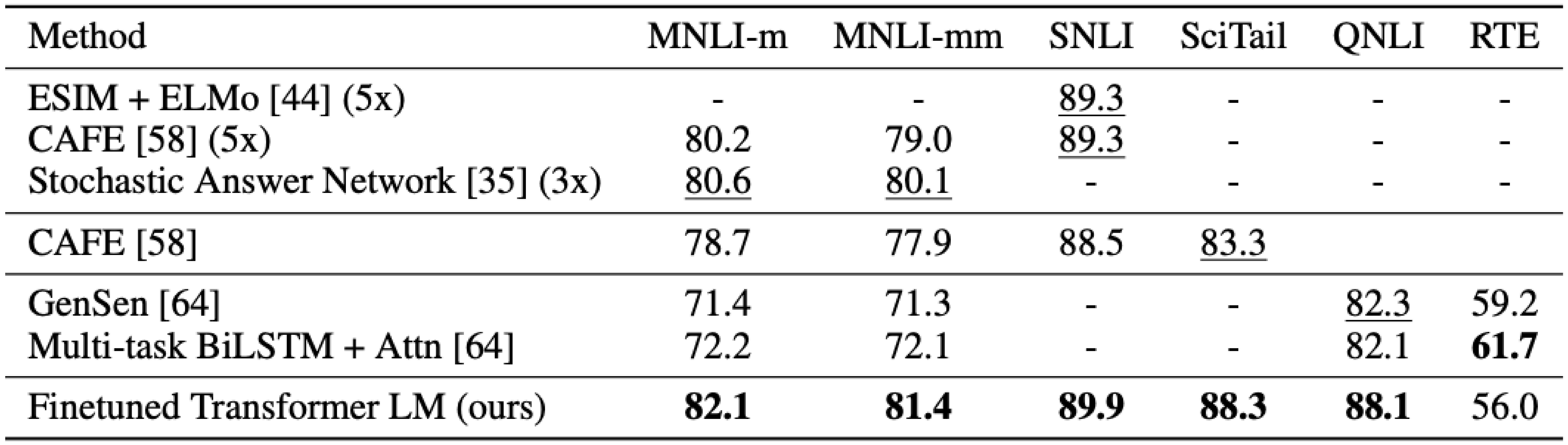
Entailment

- Radford et al., 2018 evaluate on natural language inference by formatting the input as a sequence of tokens for the decoder
  - [START] The man is in the doorway [DELIM] The person is near the door [EXTRACT]
  - The linear classifier is applied to the representation of the [EXTRACT] token.

[Radford et al., 2018]

GPT: Results on Classification

- Outperforms Recurrent Neural Nets



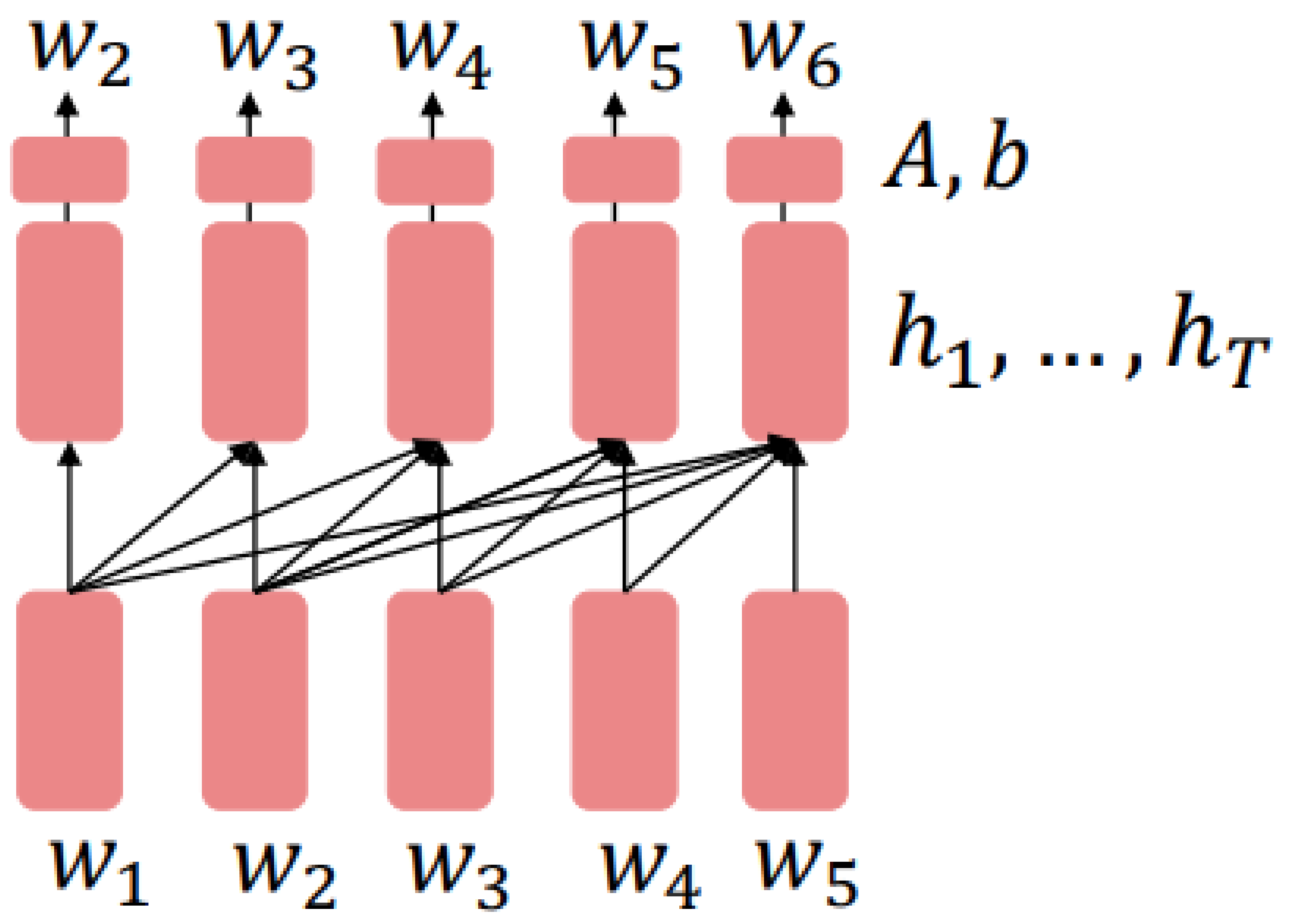
Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

[Radford et al., 2018]

Pretraining Decoders: Generators

- More natural: pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t | w_{1:t-1})$
  - $h_1, \dots, h_T = Decoder(w_1, \dots, w_T)$
- $w_t \approx Ah_{t-1} + b$
- Where A, b were pretrained in the language model!

- This is helpful in tasks where the output is a sequence with a vocabulary like that at pretraining time!
  - Dialogue (context=dialogue history)
  - Summarization (context=document)

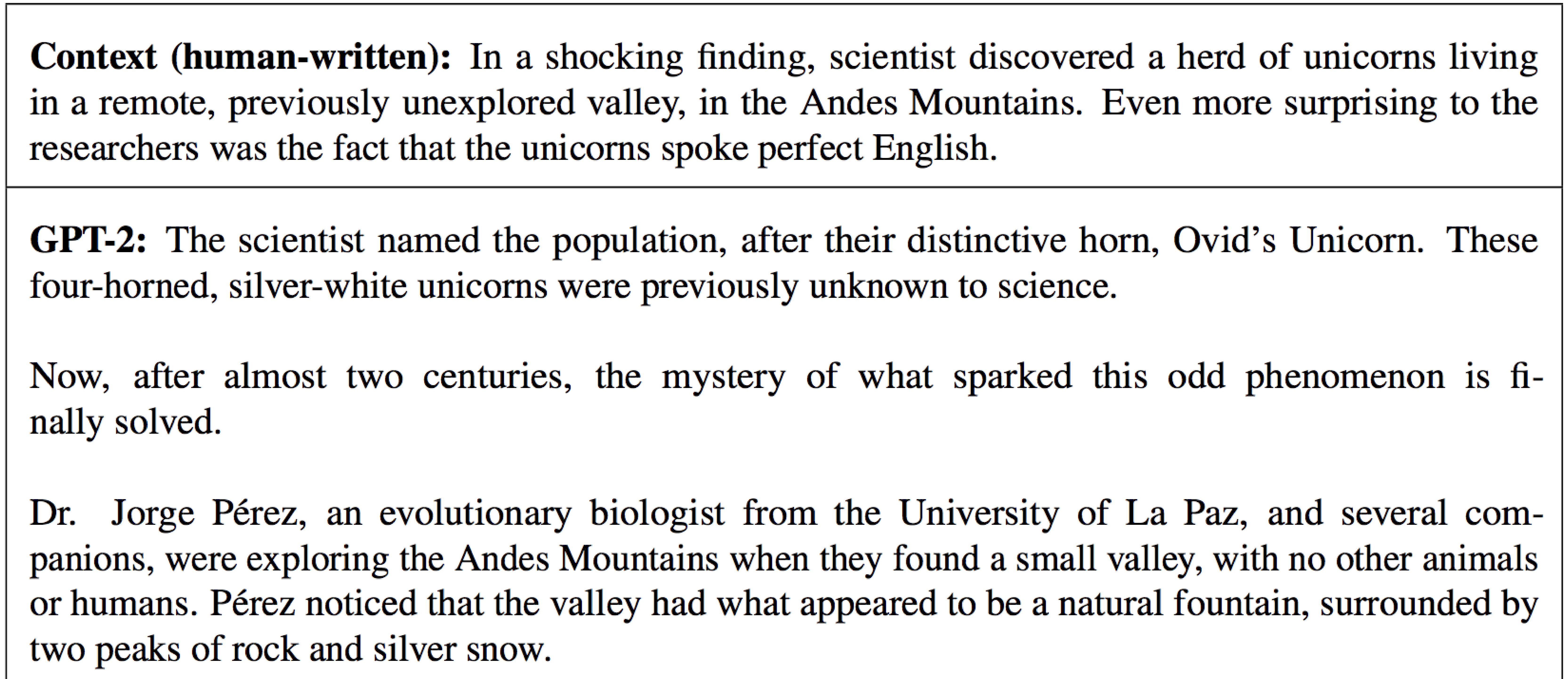


The linear layer has been pretrained

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.

GPT-3 and beyond

- Markedly improved generation capabilities by greatly increasing data and model scale
- Solving all tasks through generation
- Even obviating the need to fine-tune!

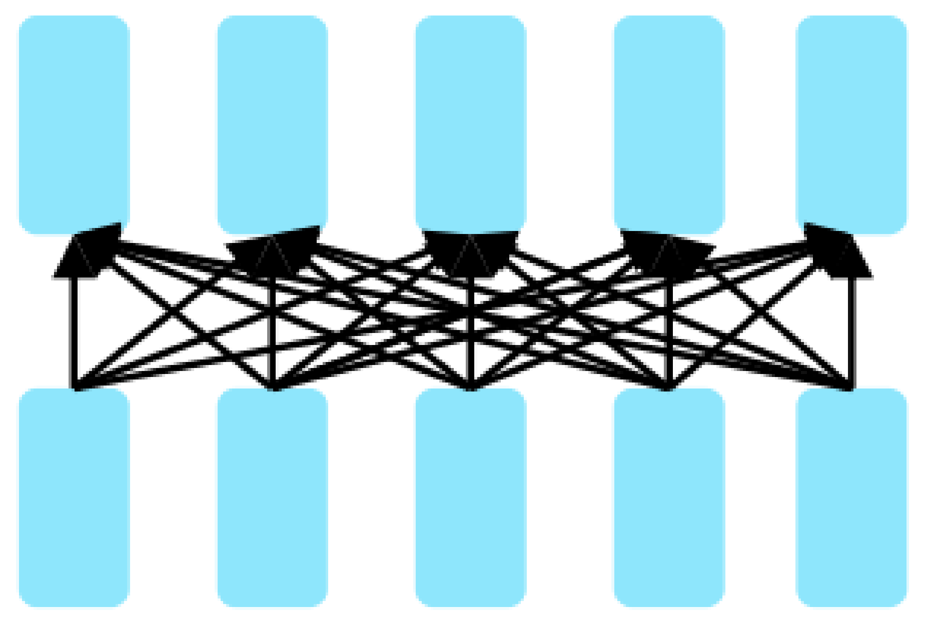
More in future weeks!

GPT-3

ChatGPT

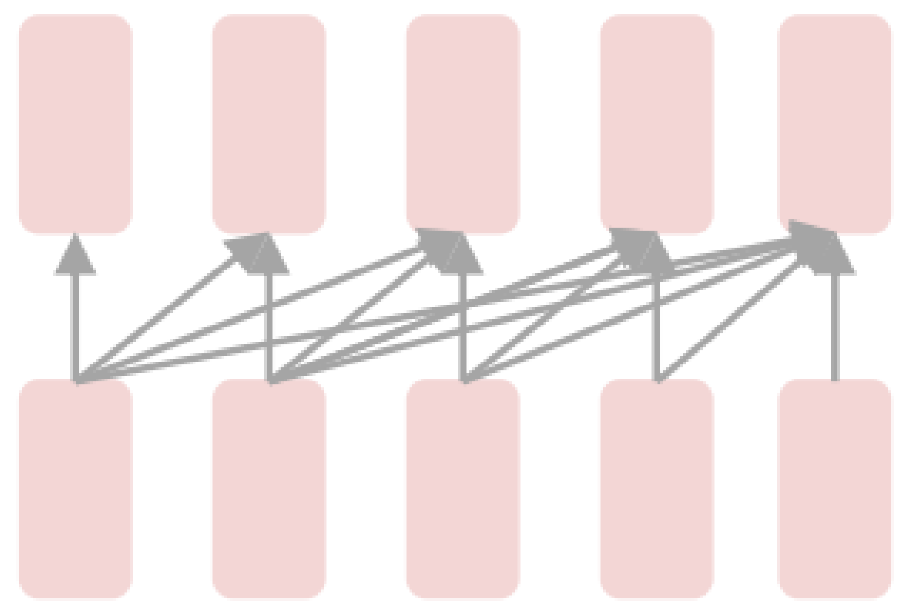
GPT-4

Pretraining for three types of architectures



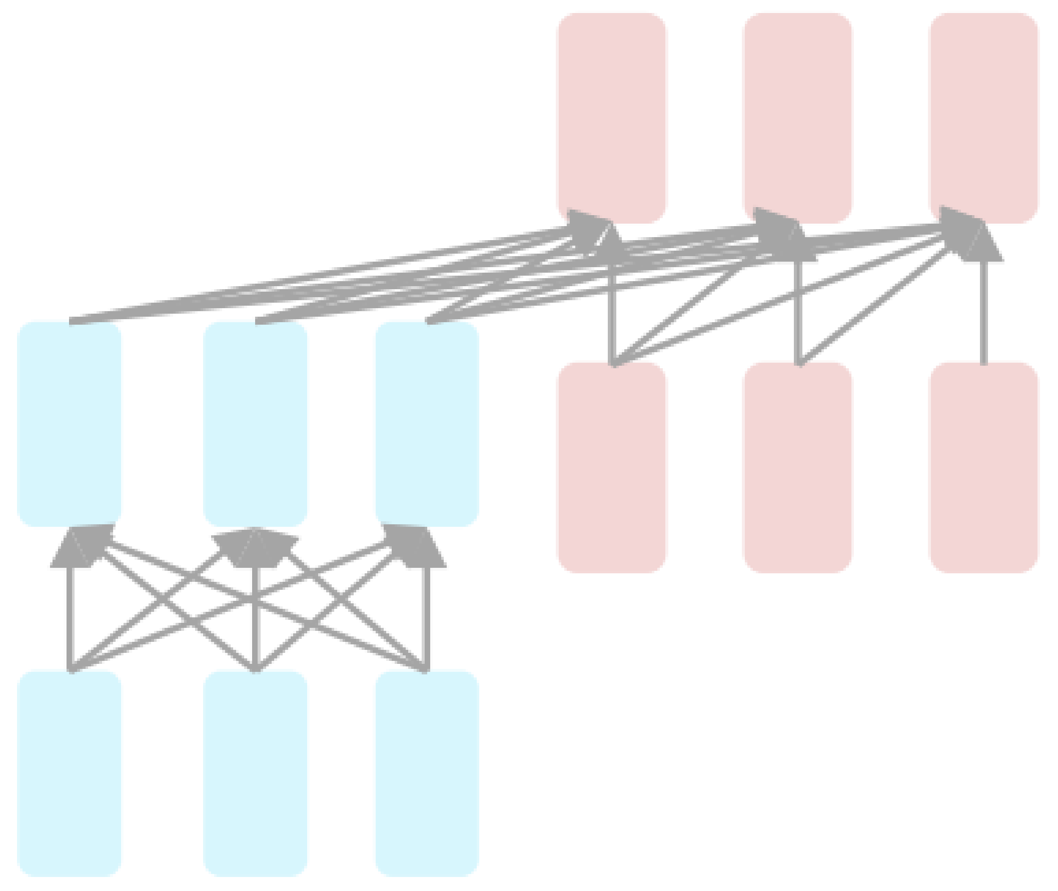
Encoders

Bidirectional Context



Decoders

Language Models



Encoder-Decoders

Sequence-to-sequence

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

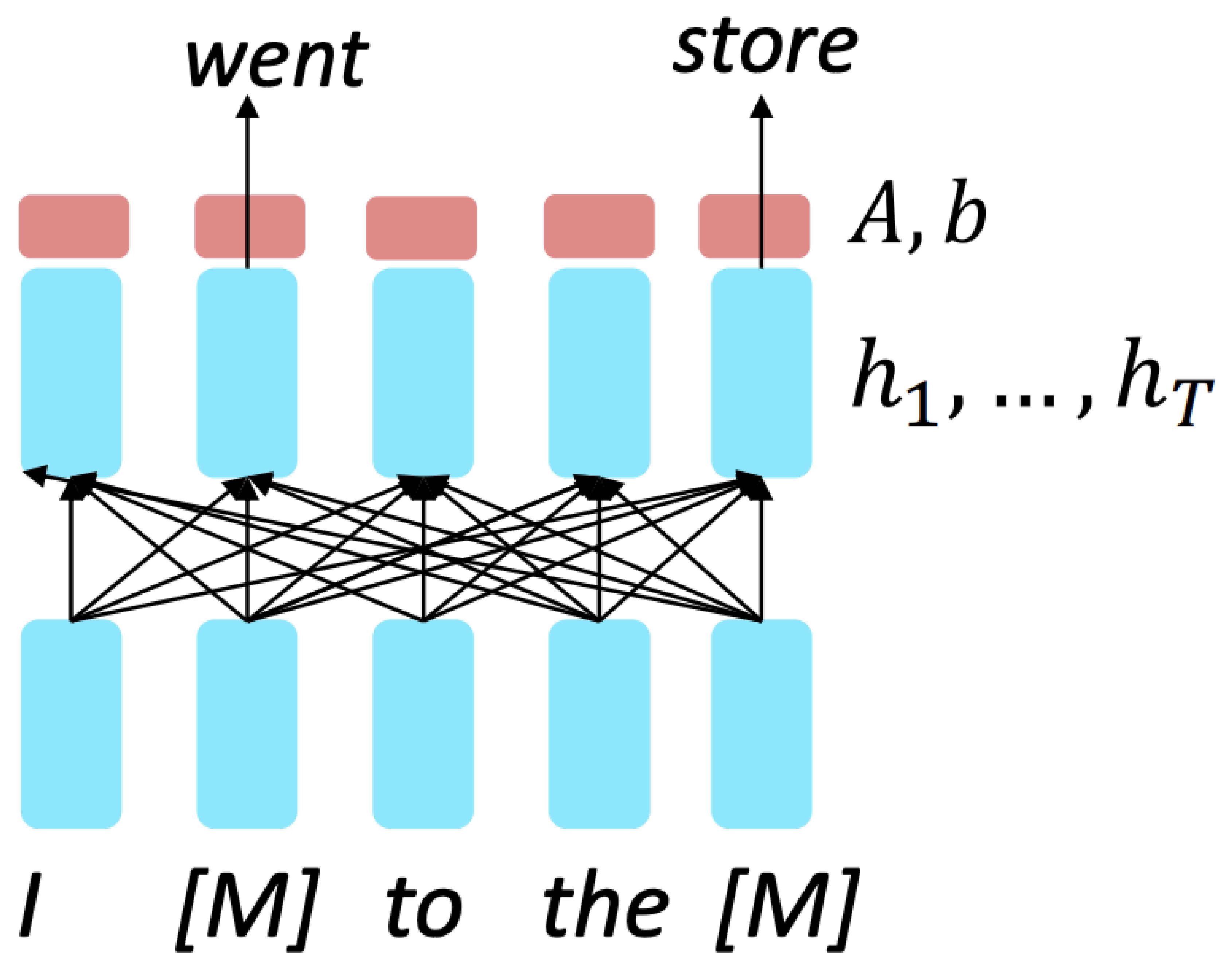
Problem: Input Reconstruction

'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.
  - $h_1, \dots, h_T = Encoder(w_1, \dots, w_T)$
  - $y_i \approx Ah_i + b$
- Only add loss terms from words that are "masked out."
- If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$.
- Called Masked LM
- Special type of language modeling

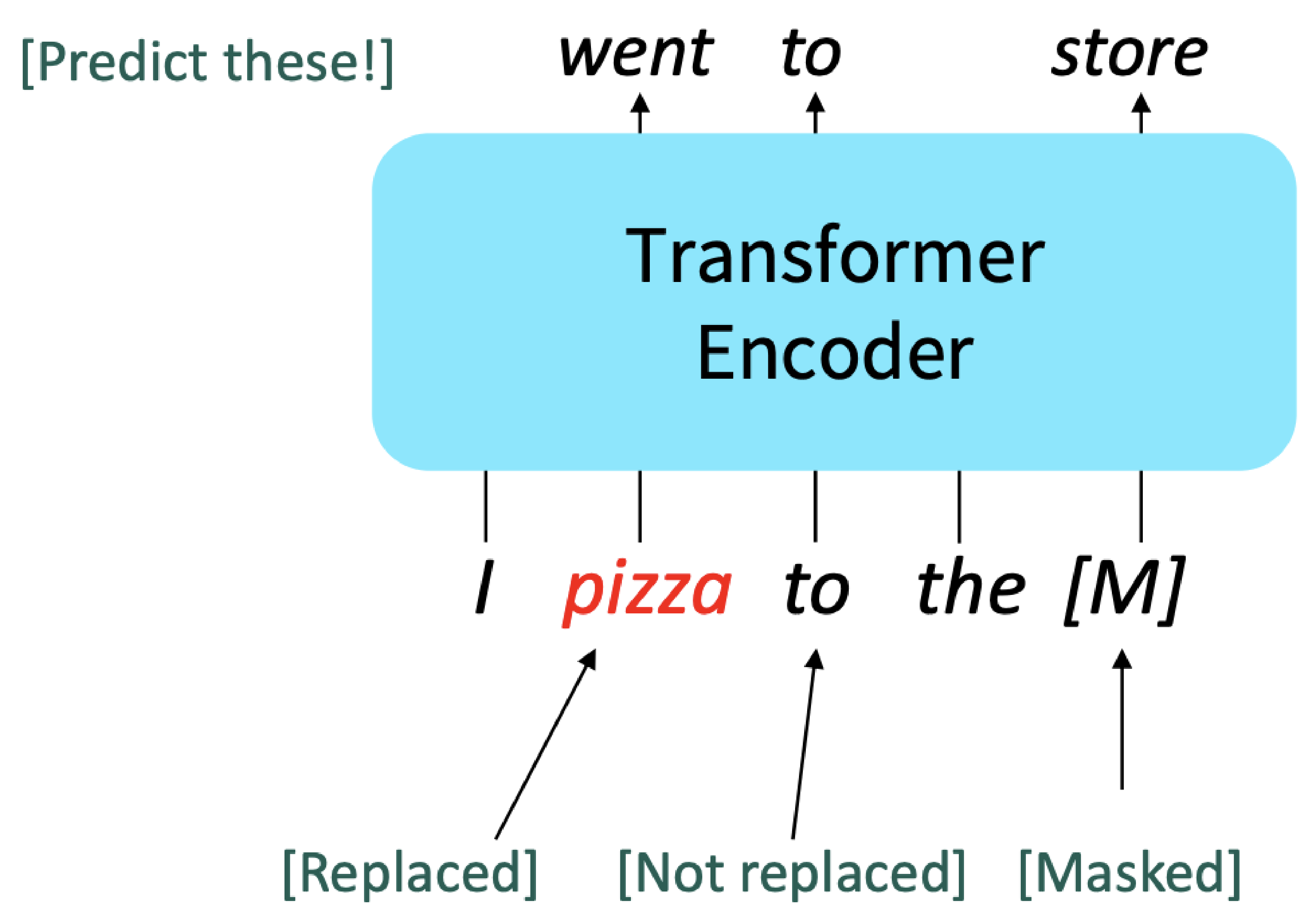


Masked Language Modeling

BERT: Bidirectional Encoder Representations from Transformers

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

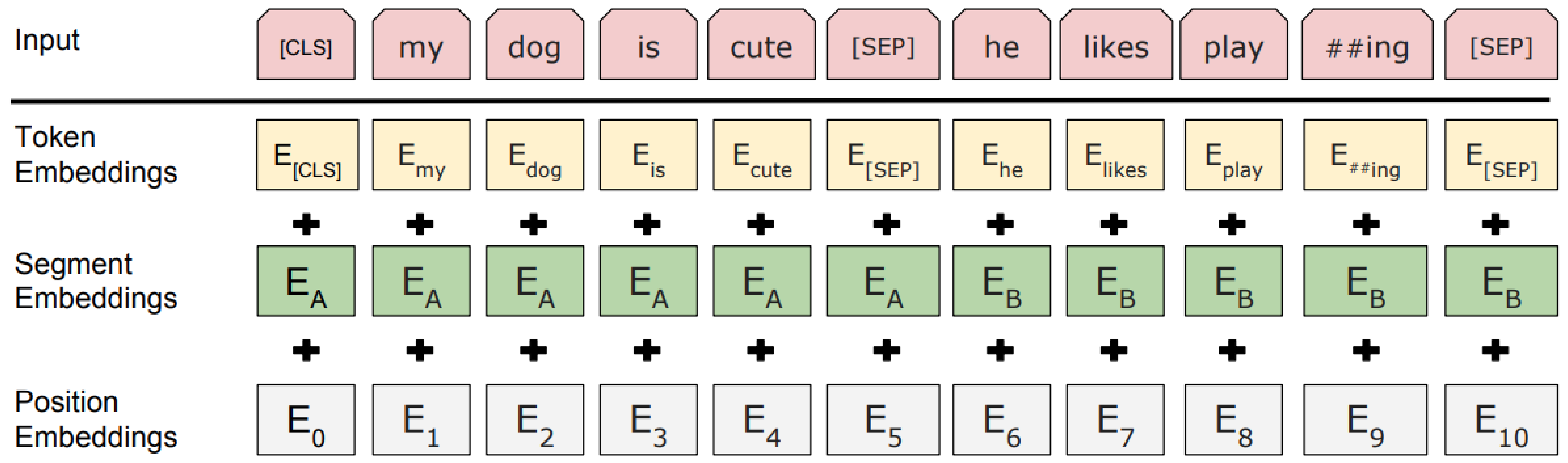
- 15% of the input tokens in a training sequence are sampled for learning, these are to be predicted by the model
- Of these
  - 80% are replaced with [MASK]
  - 10% are replaced with randomly selected tokens,
  - Remaining 10% are left unchanged



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:

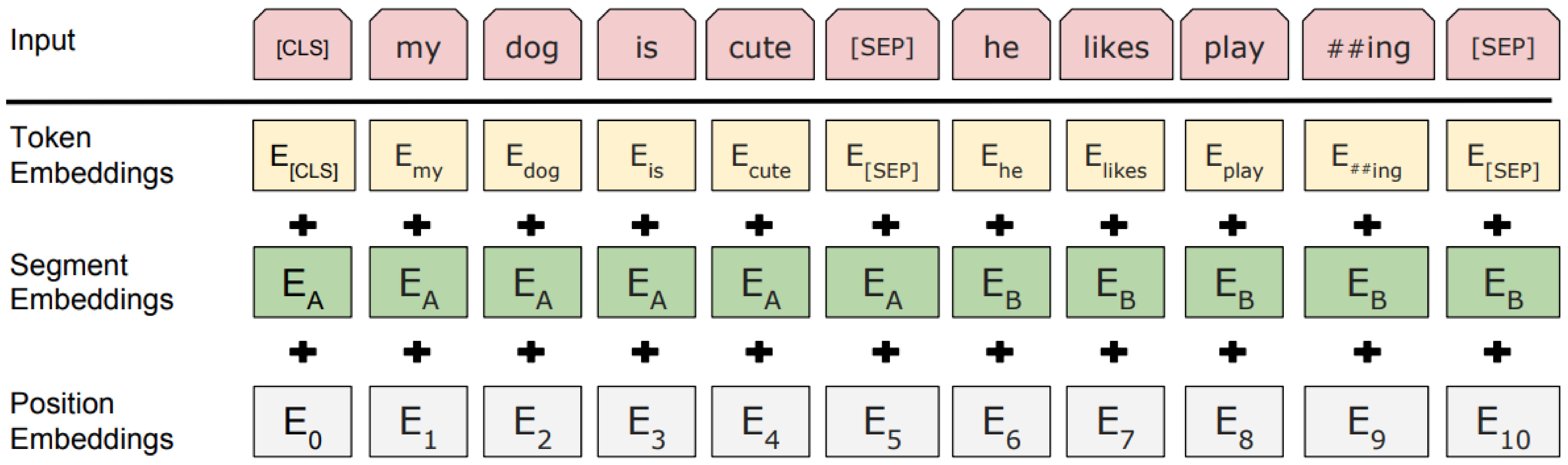

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

- 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


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

- 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

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)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
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

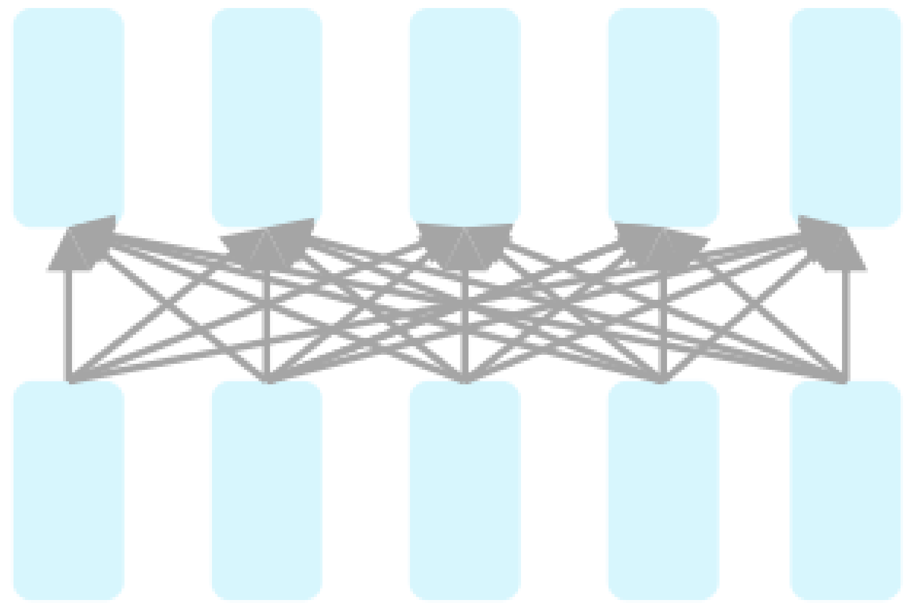
BERT: Extensions

- Some generally accepted improvements to the BERT pretraining formula:
  - RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
  - SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task
- A lot of BERT variants that used the BERT formula
  - ALBERT: BERT with parameter-reduction techniques
  - DistilBERT:
  - DeBERTa: Decoding-enhanced BERT with disentangled attention
  - FlauBERT: BERT for French
  - XLNet: Multilingual BERT
  - Etc.
- BERTology: How and why BERT worked so well

BERT: Overview

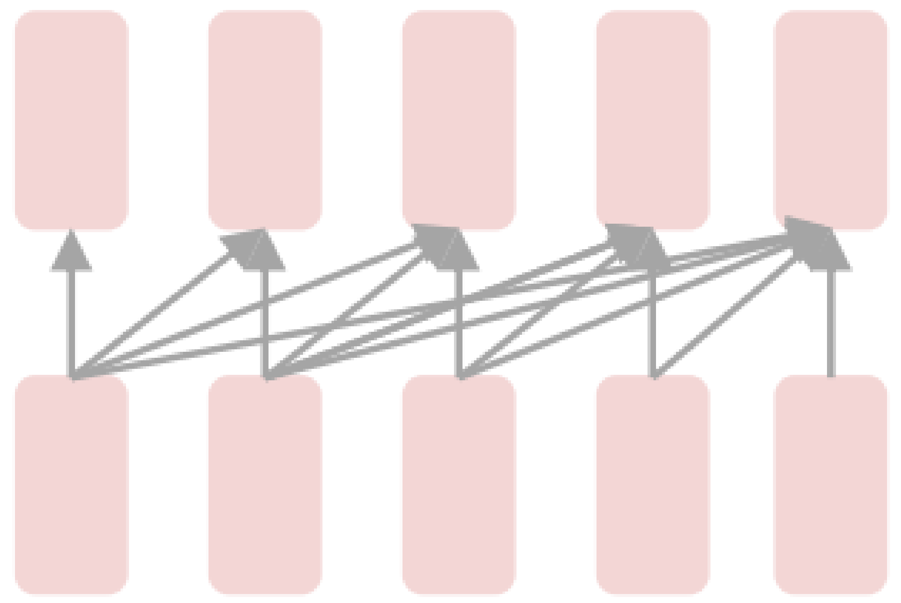
- [SEP]: Later work has argued this "next sentence prediction" is not necessary
- In general, more compute, more data can improve pretraining even when not changing the underlying Transformer encoder
- Results in contextual embeddings
- Key Limitation:
  - Cannot be used for generation
  - No pretraining 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!

Pretraining for three types of architectures



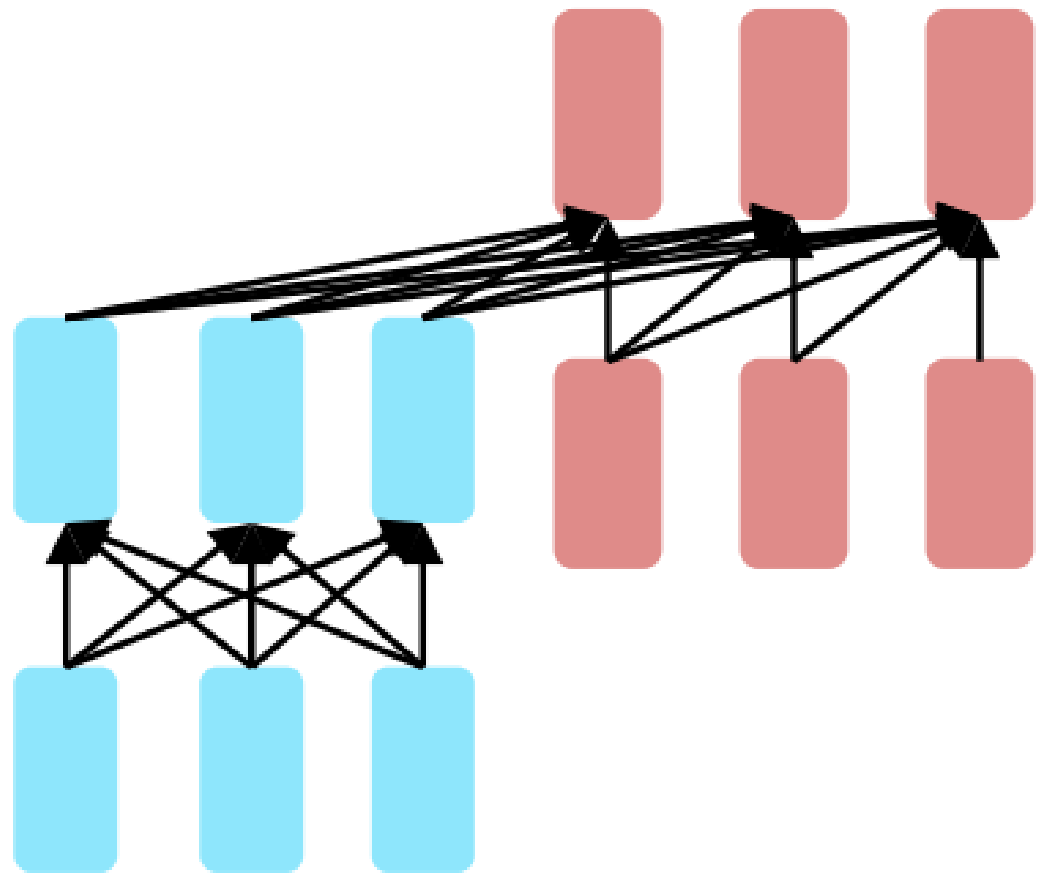
Encoders

Bidirectional Context



Decoders

Language Models



Encoder-Decoders

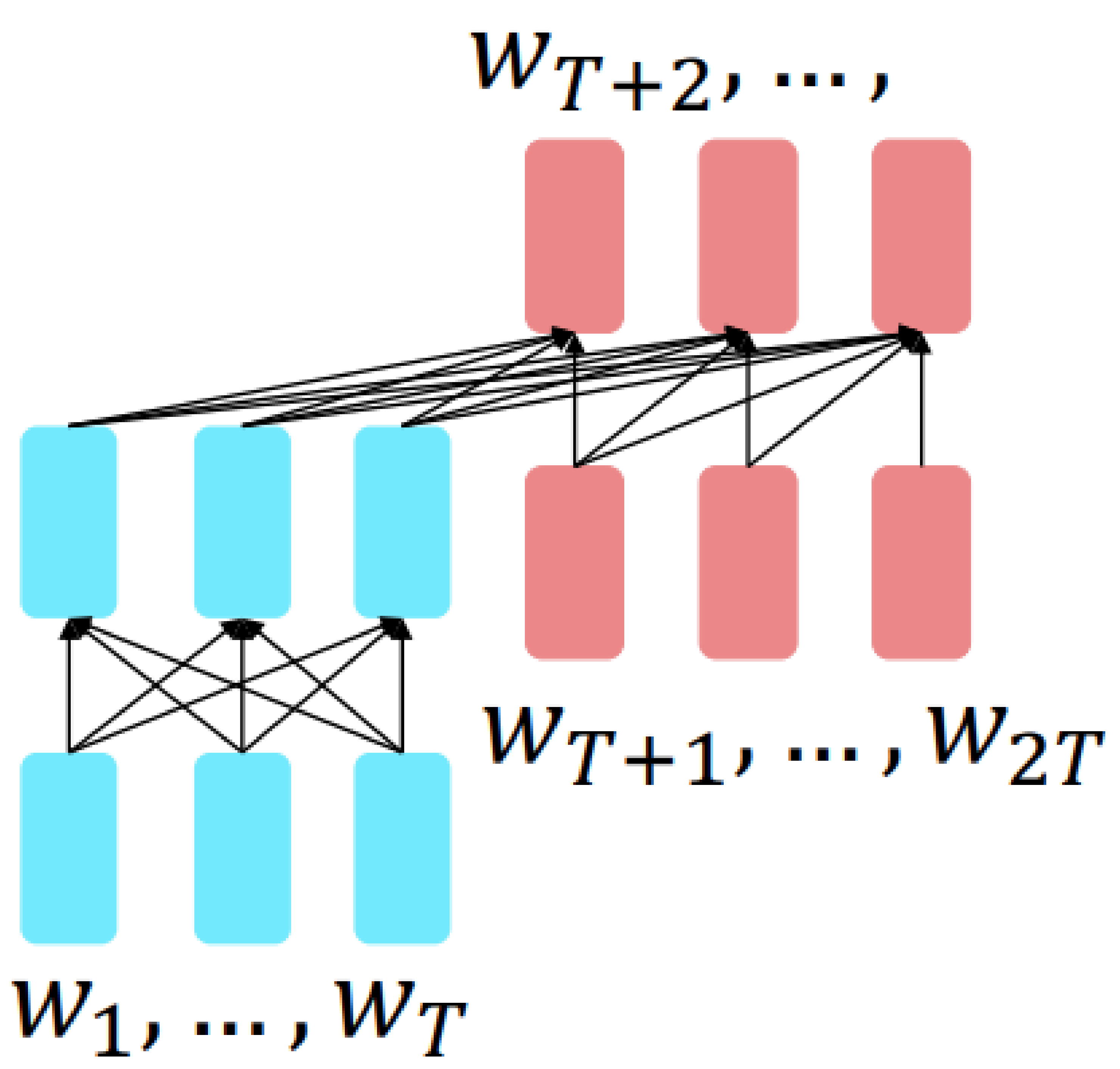
Sequence-to-sequence

Pretraining Encoder-Decoder Models

- For encoder-decoders, we could do something like language modeling, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$

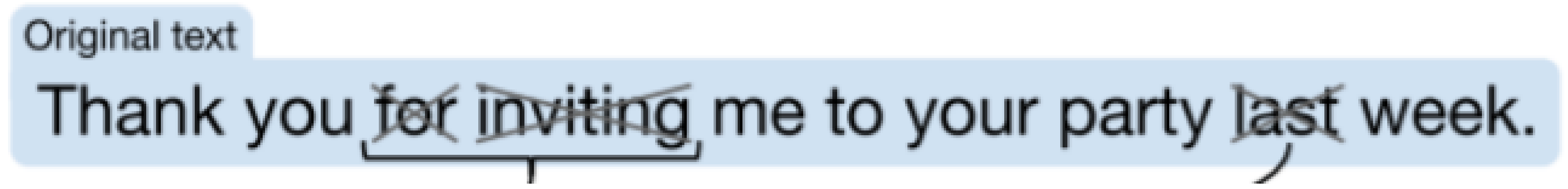
The encoder portion benefits from bidirectional context; the decoder portion is used to train the whole model through language modeling.



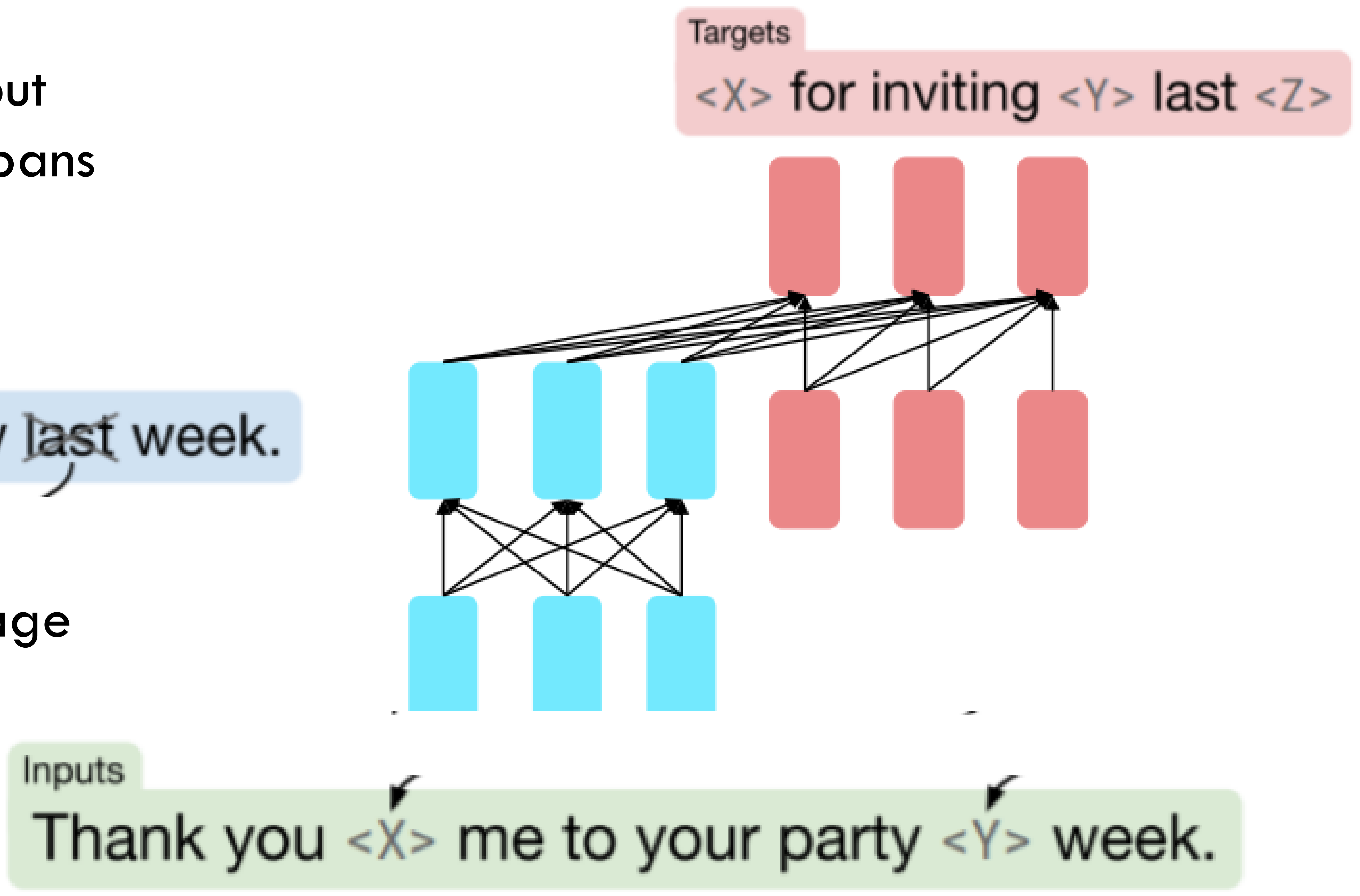
T5: A Pretrained Encoder-Decoder Model

- 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!

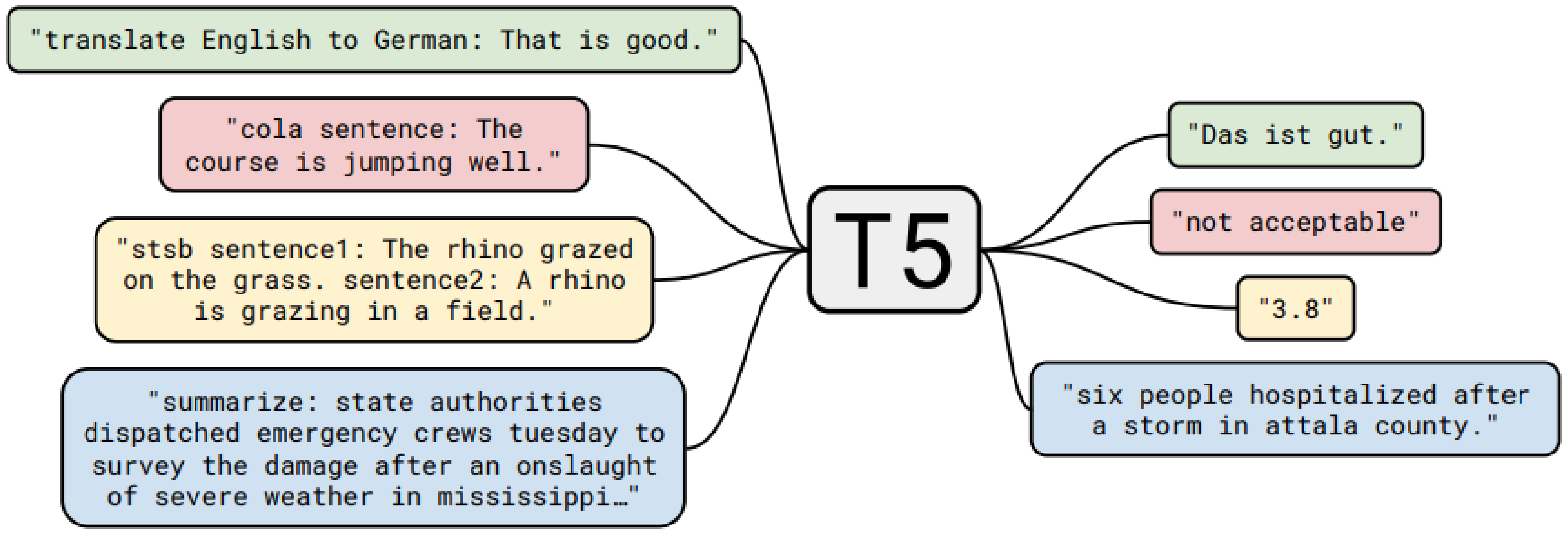


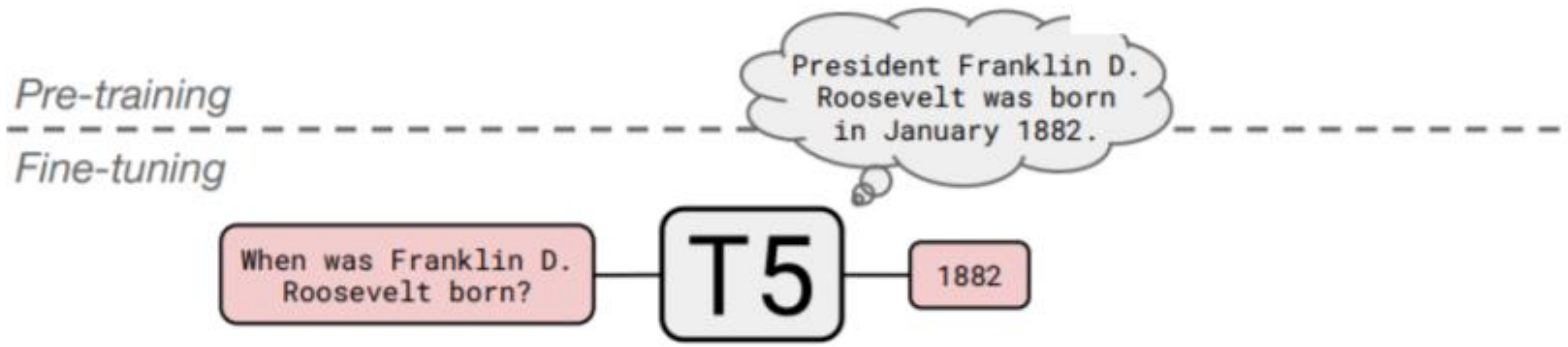
Still uses an objective that looks like language modeling at the decoder side.


Targets
<X> for inviting <Y> last <Z>
Inputs
Thank you <X> me to your party <Y> week.

T5: Task Preparation

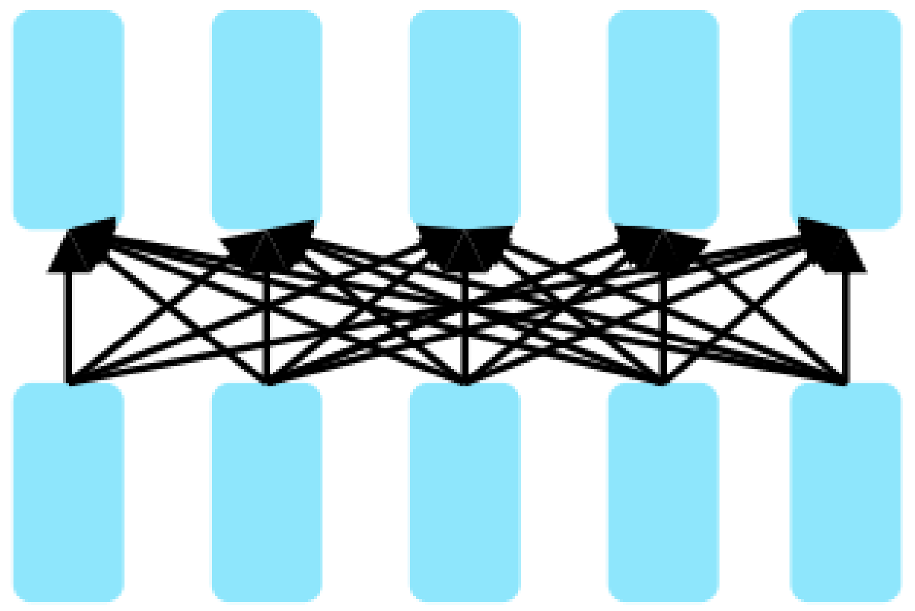
T5 can be finetuned to answer a wide range of tasks, where the input and output are expressed as a sequence of tokens


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."


Pre-training
Fine-tuning
President Franklin D. Roosevelt was born in January 1882.
When was Franklin D. Roosevelt born?
T5
1882

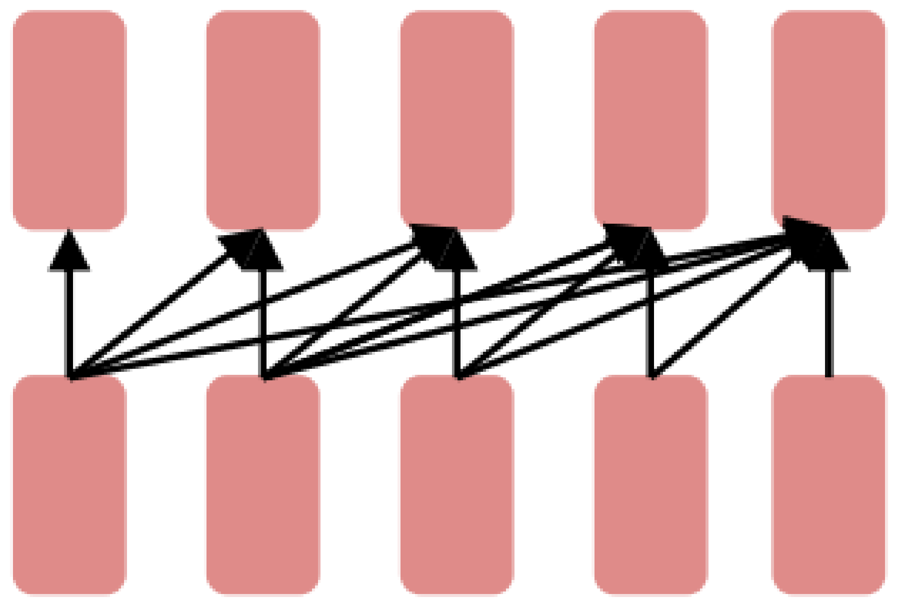
USC Viterbi

Pretraining for three types of architectures



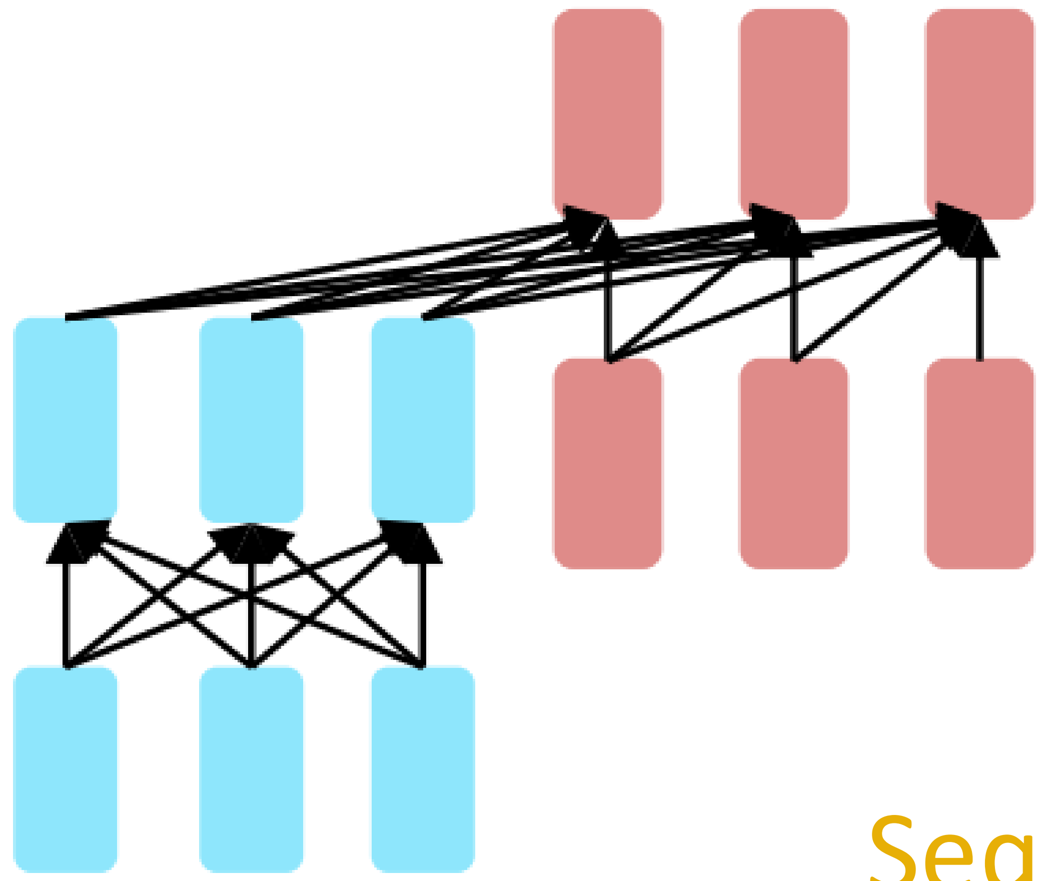
Encoders

Bidirectional Context



Decoders

Language Models



Encoder-Decoders

Sequence-to-sequence