Lecture 04: Sequence-to-sequence Models & Transformers

Xiang Ren

USC CSCI 662 Advanced NLP

Spring 2026

Some slides adapted from Chris Manning and Dan Jurafsky

Announcements

Announcements + Logistics

- Project proposal due today 11:59pm PST!
- Project pitch feedback & grades sent out on Monday.
- Paper selection is finalized. Schedule is out! (also on website)
  - https://docs.google.com/spreadsheets/d/10OVp0QWnyy9chi12cTJUpzZQc6Ah34cWnFKAo0jxo-8/edit?gid=0#gid=0

Paper Presentation Schedule (starting Feb 18)

Name of Paper	Name of Pr	Discussion	Discussion	Day	Date
Collaborative gym: A framework for enabling and evaluating human-agent collaboration	Sheryl Mathev	Hong-En Chen	Anthony Liang	1	2/18/26
Sys2Bench: Inference-Time Computations for LLM Reasoning and Planning (arXiv:2502.12	I-Chun Liu	Zeyu Shanggua	Xinlei Yu	1	2/18/26
VisuLogic: A Benchmark for Evaluating Visual Reasoning in Multi-modal Large Language M	Zongjian Li	Ziyan Yang	Muzi Tao	2	2/25/26
Search-o1: Agentic Search-Enhanced Large Reasoning Models	Dimitrios Andr	Qihan Zhang	Efthymios Tsap	2	2/25/26
Humanity's Last Exam	Toan Nguyen	Didem Zeynep	Yifan Wu	2	2/25/26
rStar-Math: Small LLMs Can Master Math Reasoning with Self-Evolved Deep Thinking	Mohammad H	Zihan Wang	Didem Zeynep	3	3/4/26
MemoryAgentBench: Evaluating Memory in LLM Agents	Ziyan Yang	Zongjian Li	Jike Zhong	3	3/4/26
CoT-Self-Instruct: Building high-quality synthetic prompts for reasoning and non-reasoning t	Gengpei Qi	Yipeng Gao	Mohammad Ho	3	3/4/26
Constitutional Classifiers: Defending against Universal Jailbreaks across Thousands of Hou	Michael Duan	Sheryl Mathew	Anu Soneye	3	3/4/26
Can MLLMs Reason in Multimodality? EMMA: An Enhanced MultiModal ReAsoning Benchn	Efthymios Tsa	Muzi Tao	Chen Chu	3	3/4/26
Stop Overthinking: A Survey on Efficient Reasoning for Large Language Models	Stop Overthinl	Ania Serbina	Zhaoyuan Den	3	3/4/26
ReTool: Reinforcement Learning for Strategic Tool Use in LLMs	Qihan Zhang	I-Chun Liu	Sunwoo Lim	4	3/25/26
ToolRL: Reward is All Tool Learning Needs	Xinlei Yu	Haolin Xiong	Zhangyu Jin	4	3/25/26
TTRL: Test-Time Reinforcement Learning	Alexios Rusto	Sunwoo Lim	Letao Chen	4	3/25/26
It's Not That Simple: An Analysis of Simple Test-Time Scaling (arXiv:2507.14419)	Jike Zhong	Toan Nguyen	Zhaoyuan Den	4	3/25/26
The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models	Yifan Wu	DJ Bell	Chen Chu	4	3/25/26
Red-Teaming LLM Multi-Agent Systems via Communication Attacks	Qixin Hu	Michael Duan	Jike Zhong	4	3/25/26
Training Language Models to Reason Efficiently	Haolin Xiong	Anu Soneye	Zihan Wang	5	4/1/26
Layer by Layer: Uncovering Hidden Representations in Language Models	Ryan Swift	Hong-En Chen	Yanchen Liu	5	4/1/26
Reasoning Models Don't Always Say What They Think	Letao Chen	Efthymios Tsap	DJ Bell	5	4/1/26
Training Large Language Model to Reason in a Continuous Latent Space	Zhaoyuan Der	Bo-Ruei Huang	Ryan Swift	5	4/1/26
Understanding R1-Zero-Like Training: A Critical Perspective	Sunwoo Lim	Dimitrios Andre	Alexios Ruston	5	4/1/26
LIMO: Less is More for Reasoning	Yanchen Liu	Sheryl Mathew	Ania Serbina	5	4/1/26
Reinforcement Learning Teachers of Test Time Scaling (RLT) (arXiv:2506.08388)	Bo-Ruei Huan	Anthony Liang	Zeyu Shanggua	6	4/8/26
ROBOTOUILLE: Asynchronous Planning Benchmark for LLM Agents (ICLR 2025)	Didem Zeyne	I-Chun Liu	Ryan Swift	6	4/8/26
PaperArena: Benchmarking Tool-Using Agents	Yipeng Gao	Yifan Wu	Ziyan Yang	6	4/8/26

Paper Presentation Requirement

20% of the grade

Each student will complete **one in-depth research paper presentation (15-17 mins presentation + 5-7 mins discussion)** during the semester using slides,

Each student will also serve as a **designated discussion lead for *two* additional paper presentations.**

Paper Presentation: Evaluation

The presentation will be assessed on the student's ability to clearly explain (50%):

- the paper's motivation
- technical approach
- key results

To critically evaluate its assumptions, limitations, and contributions (30%)

To situate the work within the broader LLM and NLP research landscape (20%)

Paper Presentation: Evaluation

Students will also be evaluated on ***the quality of the discussion they facilitate:***

- through insightful questions
- engagement with peers
- ability to surface open problems and future research directions

both during their own presentation and when leading discussions for others.

Paper Presentation

The slides of the presentation need to be shared a day before the presentation to the class.

Please add to the slide deck TA sent the night before, and TA will distribute before the class in the morning.

Recap

Data: Feedforward Language Model

- Self-supervised
- Computation is divided into time steps t, where different sliding windows are considered
- $x_t = (w_{t-1}, \ldots, w_{t-M+1})$ for the context
  - represent words in this prior context by their embeddings, rather than just by their word identity as in n-gram LMs
  - allows neural LMs to generalize better to unseen data / similar data
  - All embeddings in the context are concatenated
- $y_t = w_t$ for the next word
  - Represented as a one hot vector of vocabulary size where only the ground truth gets a value of 1 and every other element is a 0

One-hot vector

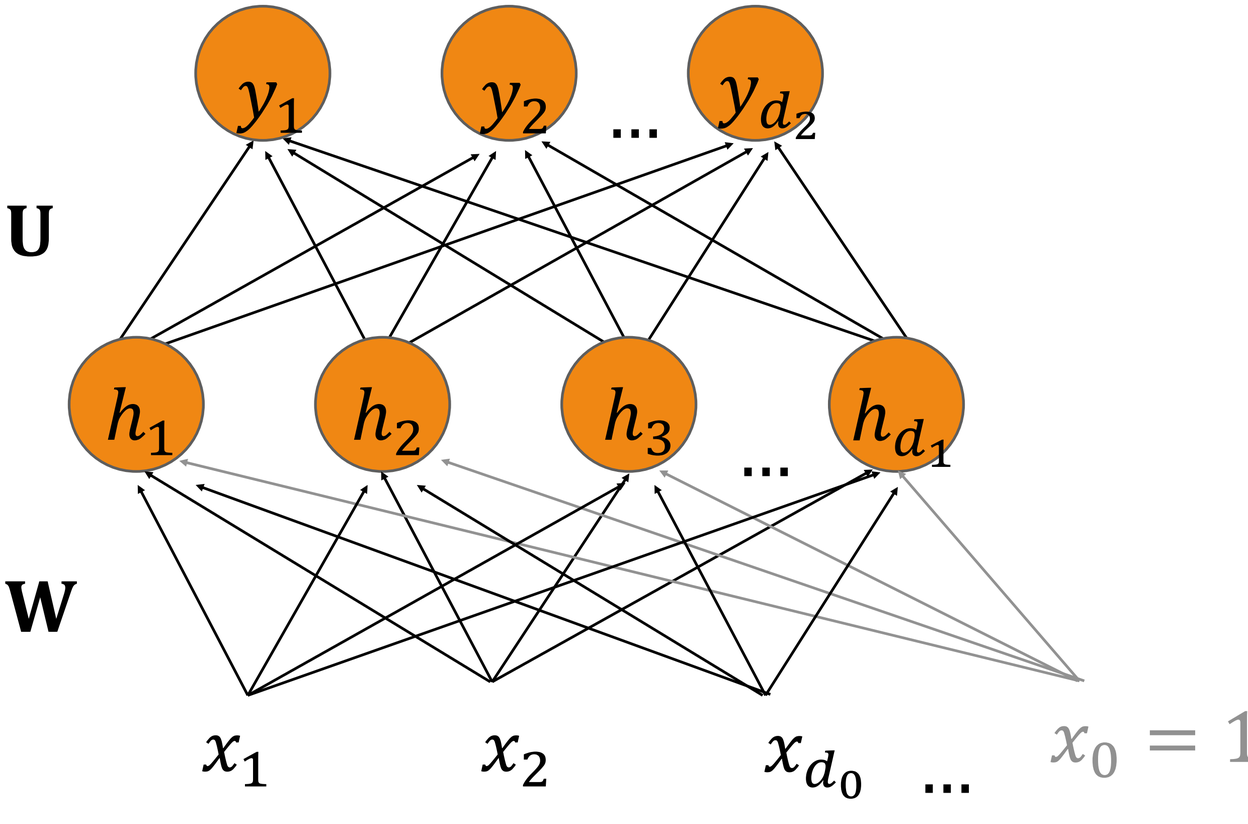
Two-layer FFNN: Notation

Output layer: $\mathbf{y} = \mathit{softmax}(\mathbf{U} \cdot \mathbf{h})$

Hidden layer: $\mathbf{h} = g(\mathbf{W}\mathbf{x}) = g(\sum_{i=0}^{d_0} W_{ji} x_i)$

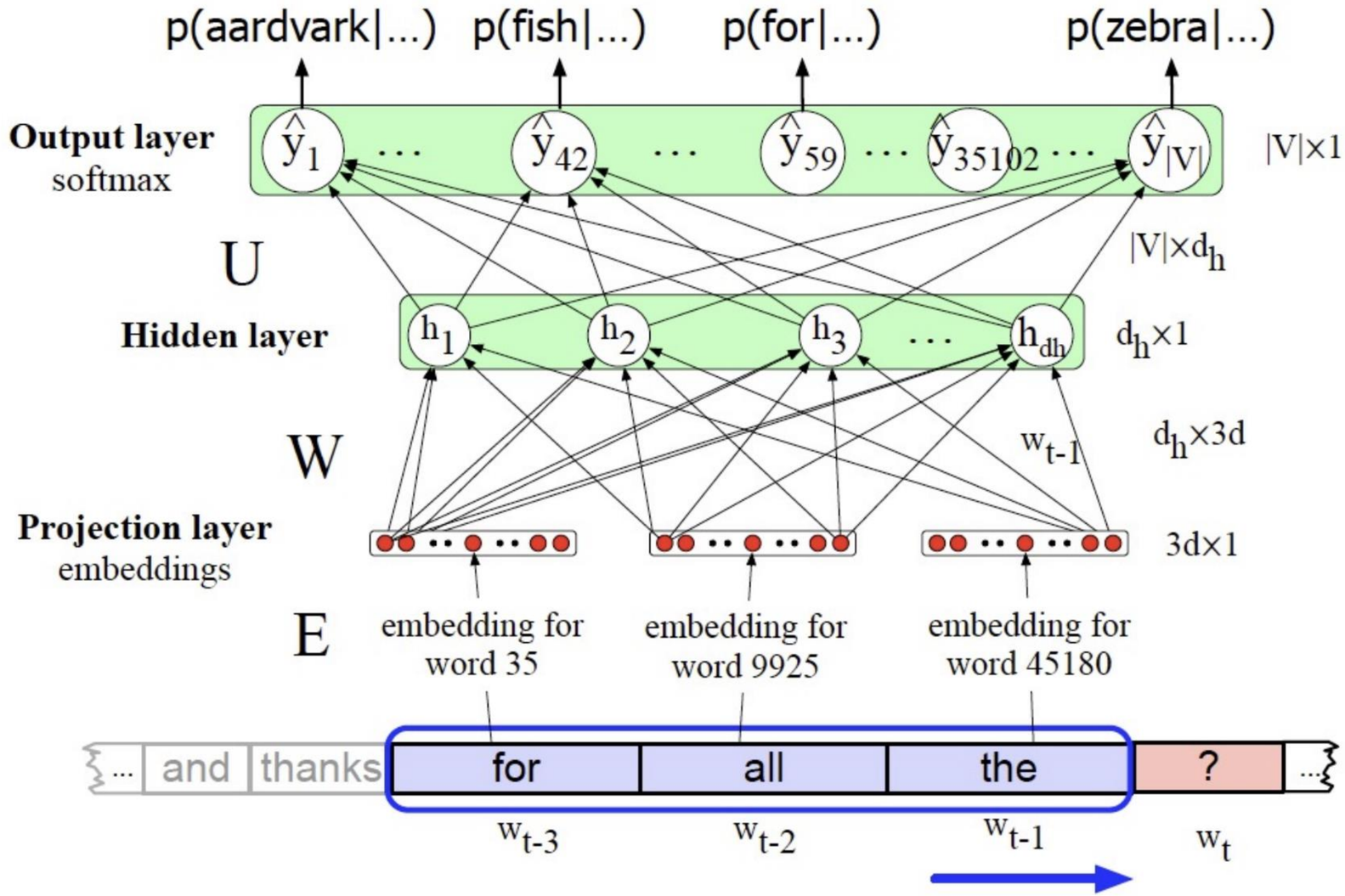
Usually ReLU or tanh

Input layer: vector $\mathbf{x}$

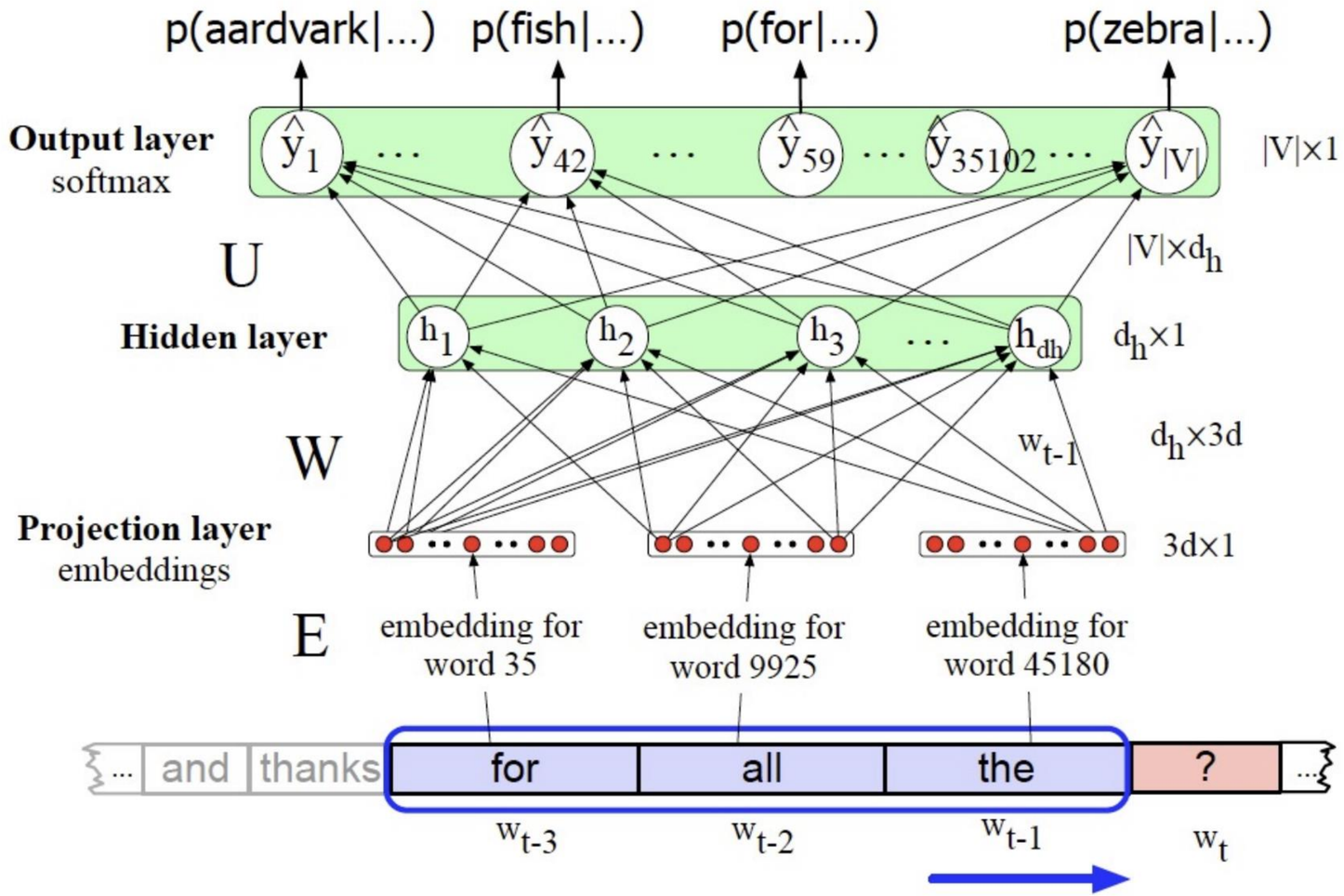


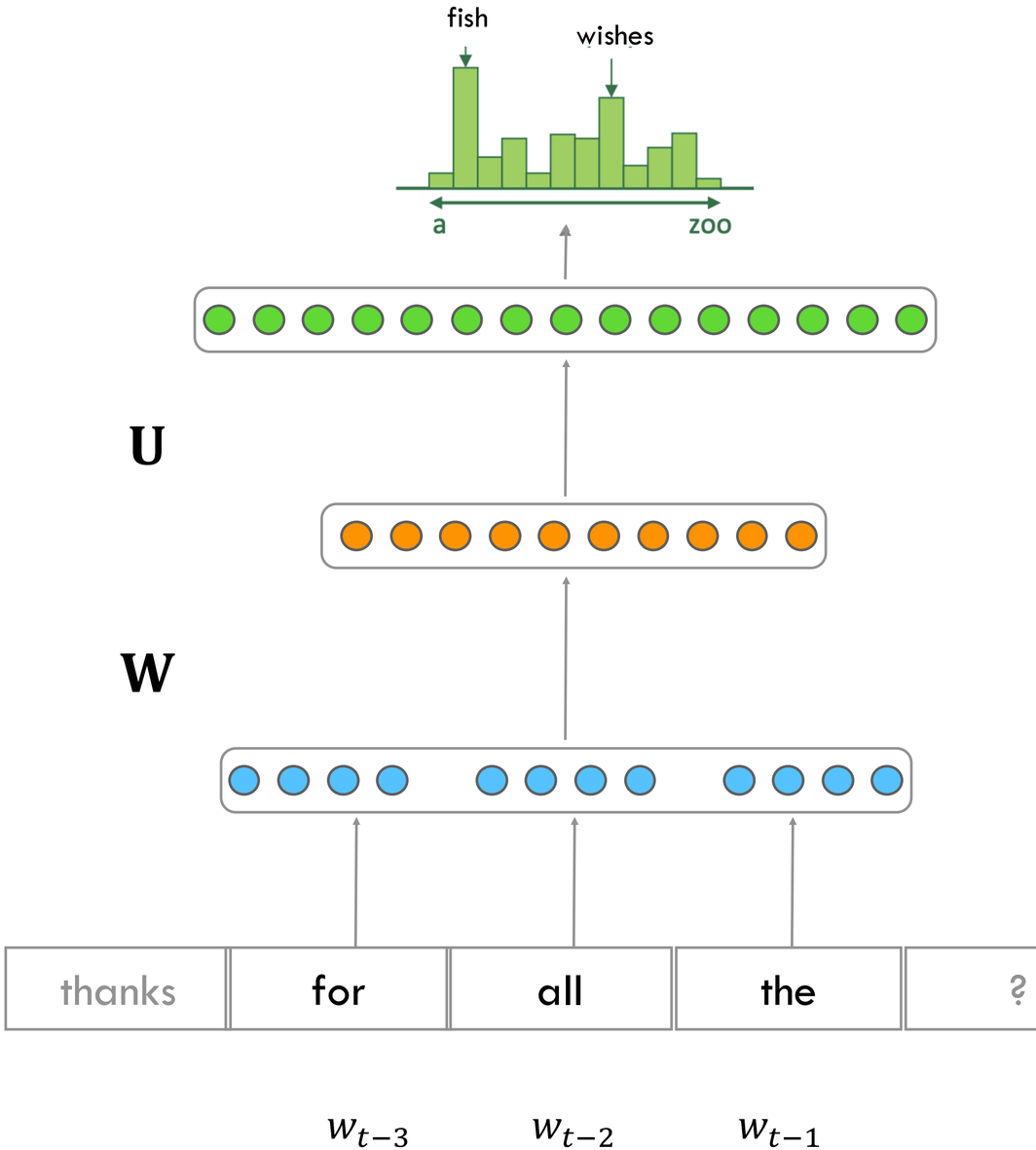
Feedforward Neural LM

- Sliding window of size 4 (including the target word)
- Every feature in the embedding vector connected to every single hidden unit
- Projection / embedding layer is a kind of input layer
  - This is where we plug in our word2vec embeddings
  - May or may not update embedding weights



Simplified Representation



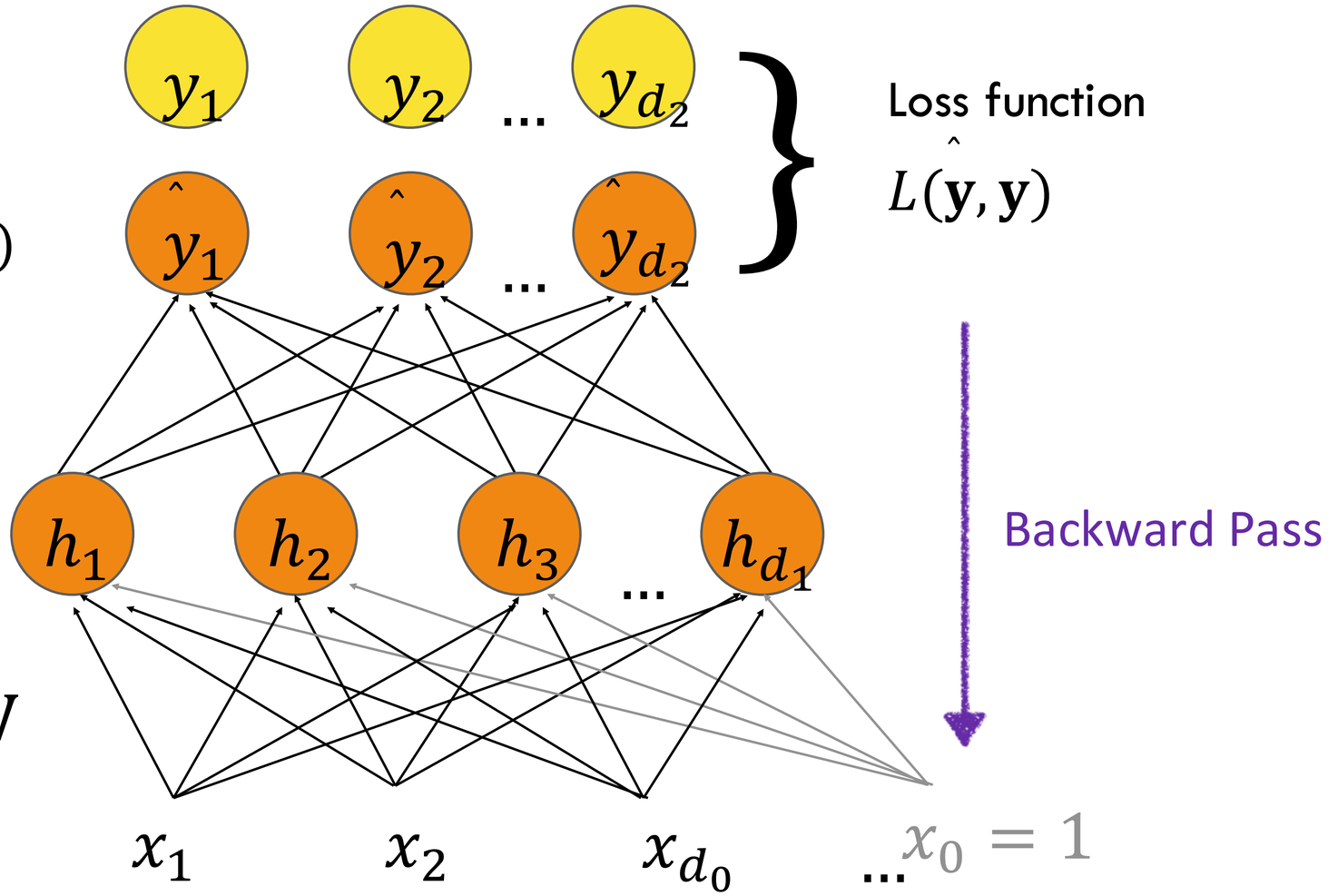


Intuition: Training a 2-layer Network

Training instance $\mathbf{y}$

Model Output $\hat{\mathbf{y}} = softmax(\mathbf{U} \cdot \mathbf{h})$

Forward Pass



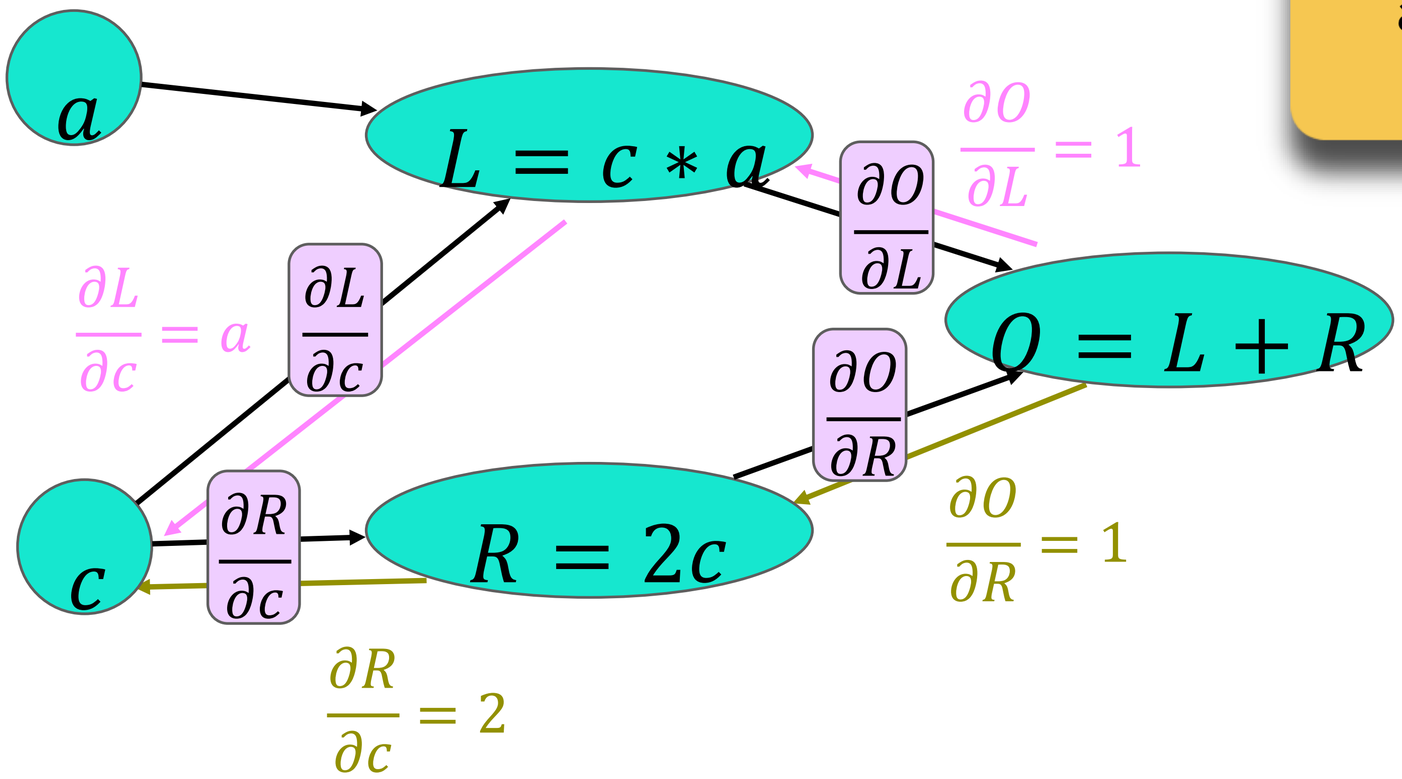
Loss function $L(\hat{\mathbf{y}}, \mathbf{y})$

Backward Pass

Training instance $\mathbf{x}$

Example: Two Paths

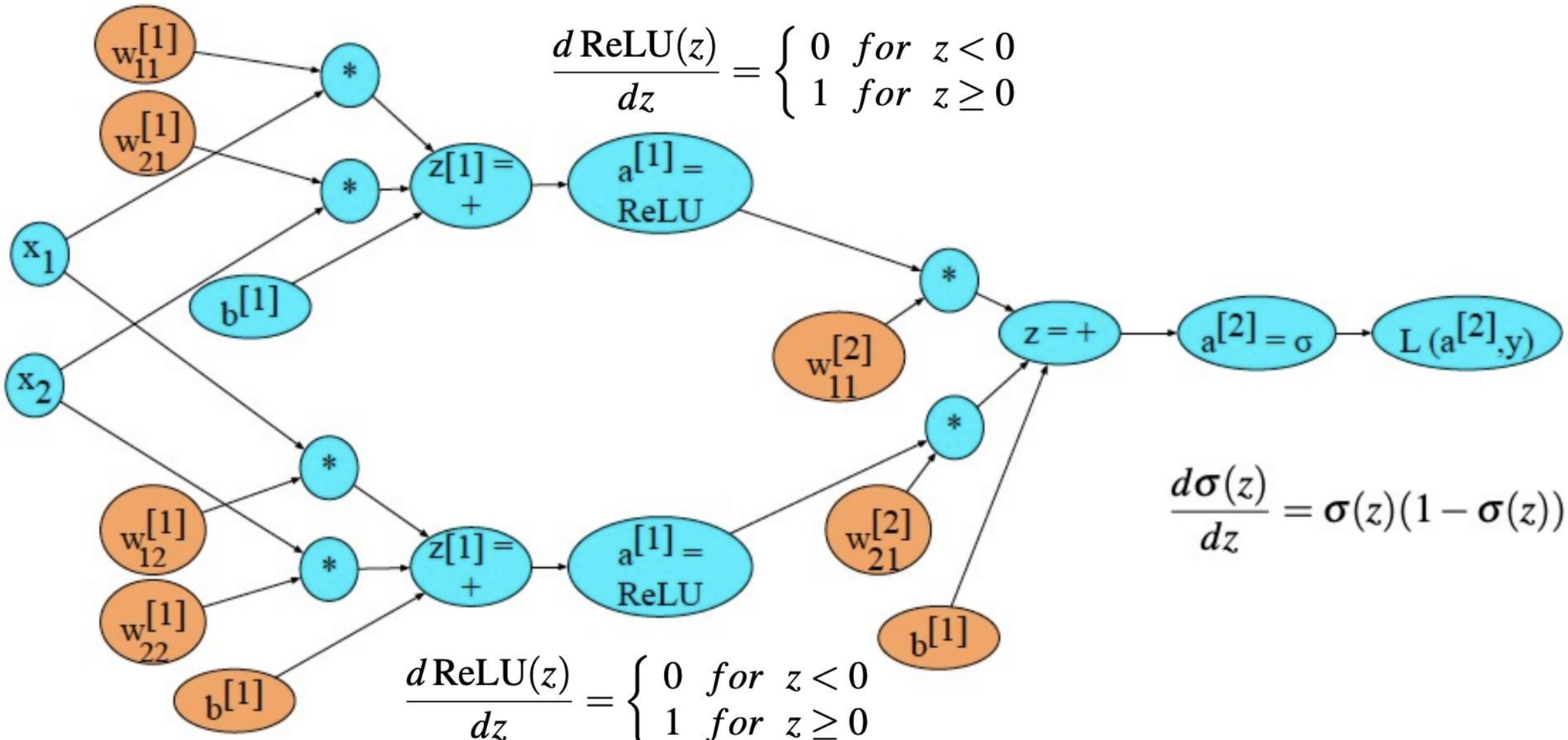
When multiple branches converge on a single node we will add these branches



$$\frac{\partial O}{\partial c} = \frac{\partial O}{\partial L}\frac{\partial L}{\partial c} + \frac{\partial O}{\partial R}\frac{\partial R}{\partial c}$$

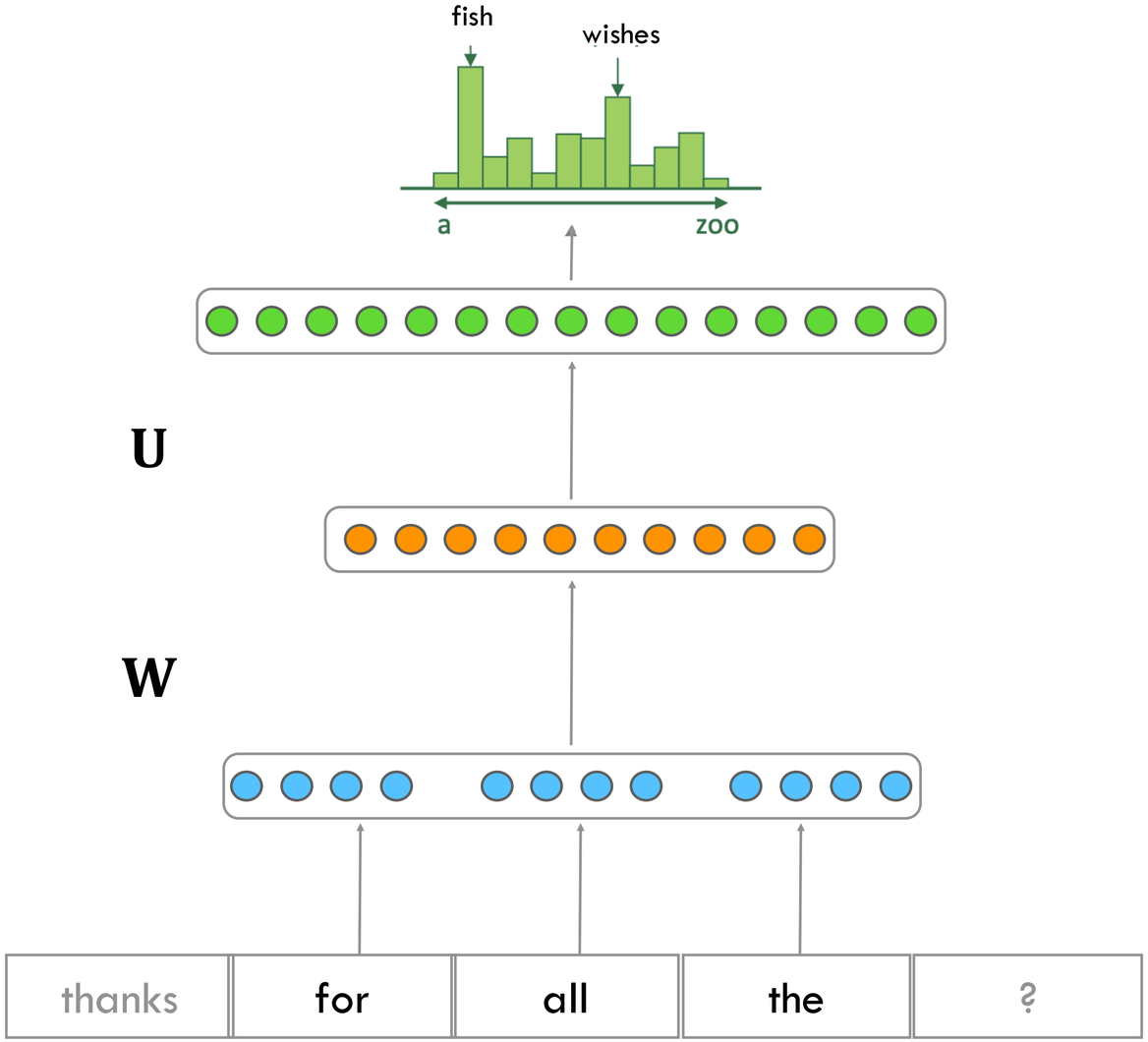
Such cases arise when considering regularized loss functions

2 layer MLP with 2 input features



Feedforward LMs: Windows

- The goodness of the language model depends on the size of the sliding window!
- Fixed window can be too small
- Enlarging window enlarges $\mathbf{W}$
- Each word uses different rows of $\mathbf{W}$. We don't share weights across the window.
- Window can never be large enough!



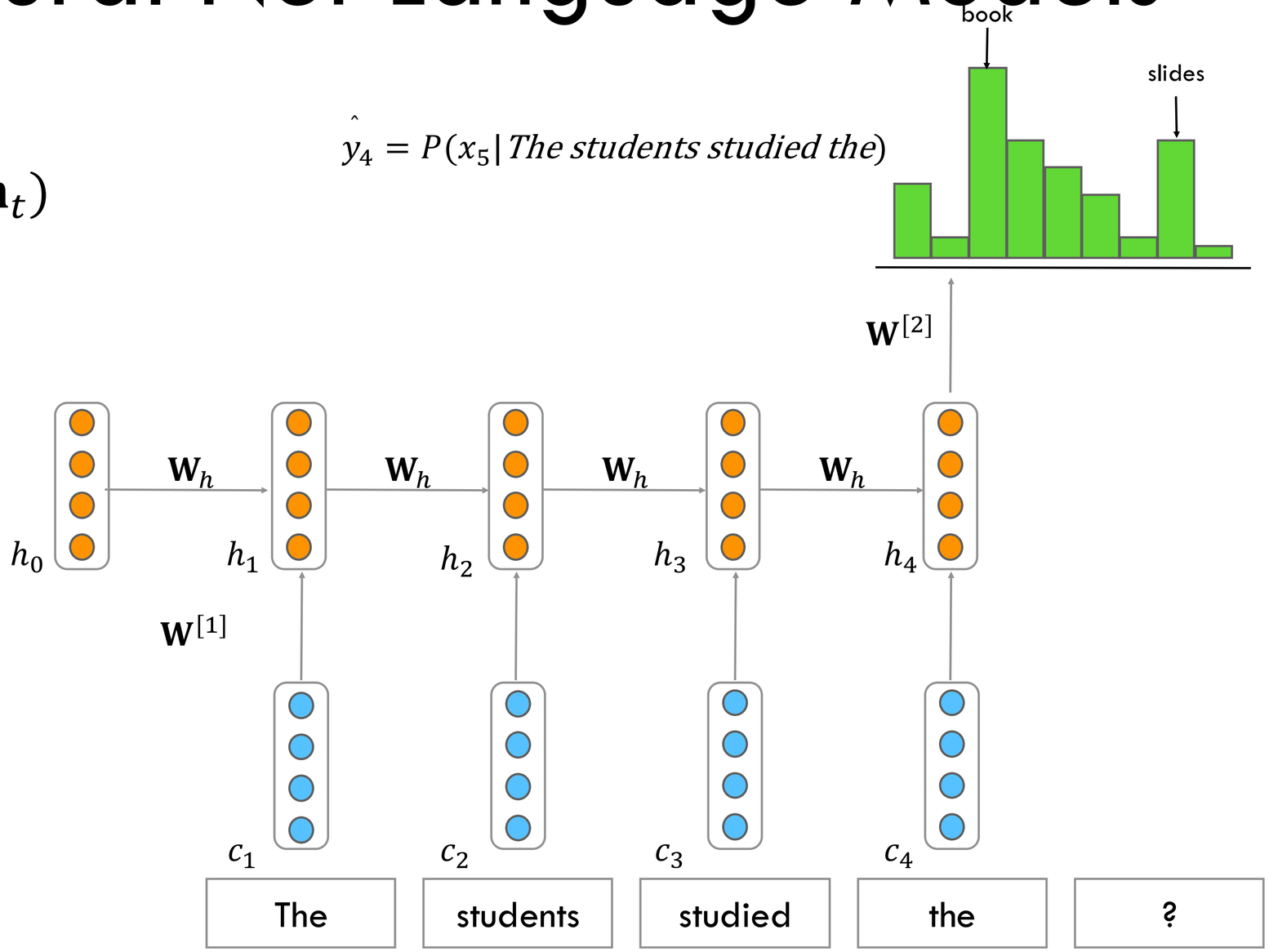
Recurrent Neural Net Language Models

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

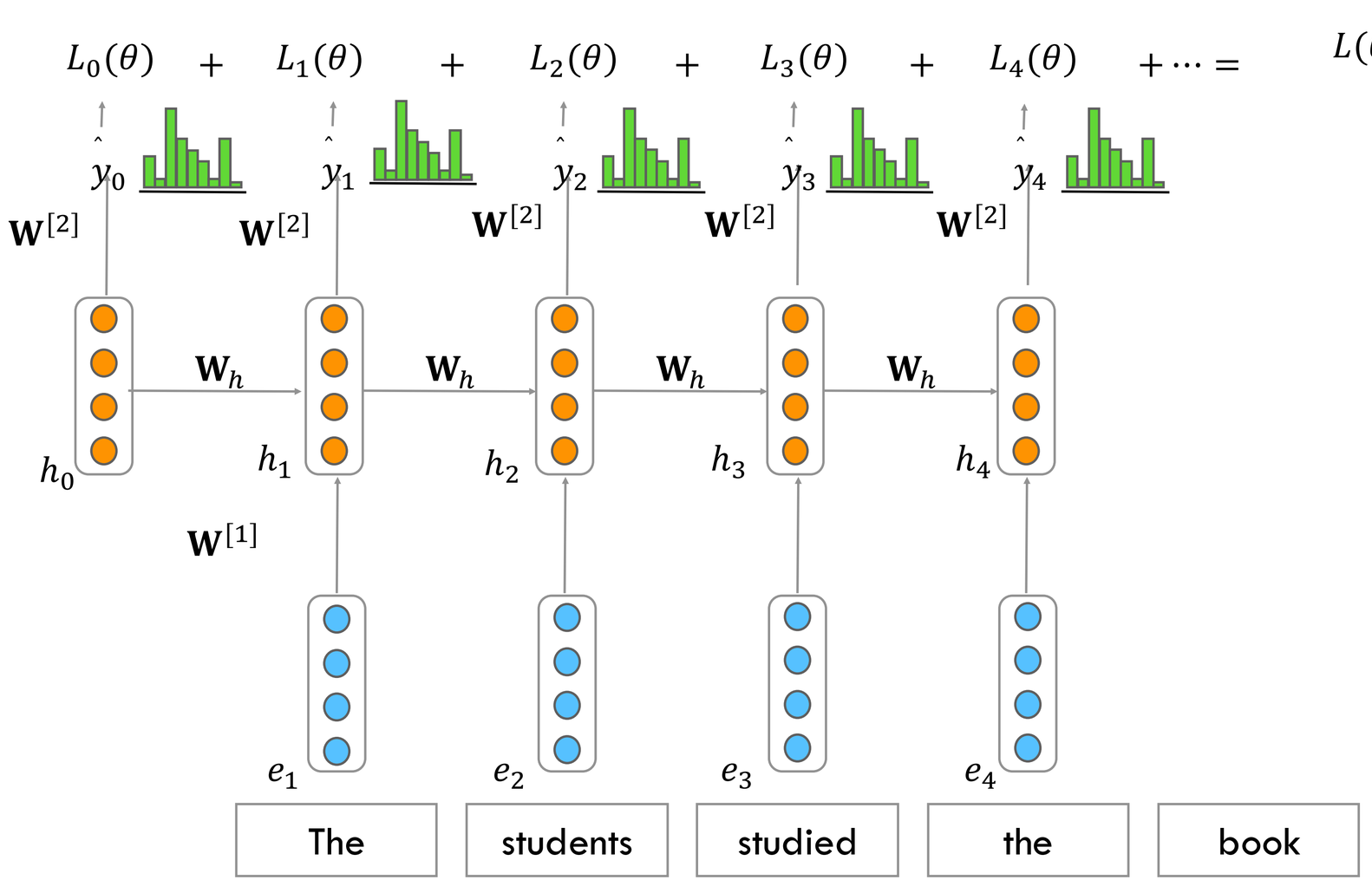
Hidden layer: $\mathbf{h}_t = g(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}^{[1]}\mathbf{c}_t)$

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

Word Embeddings, $\mathbf{c}_i$



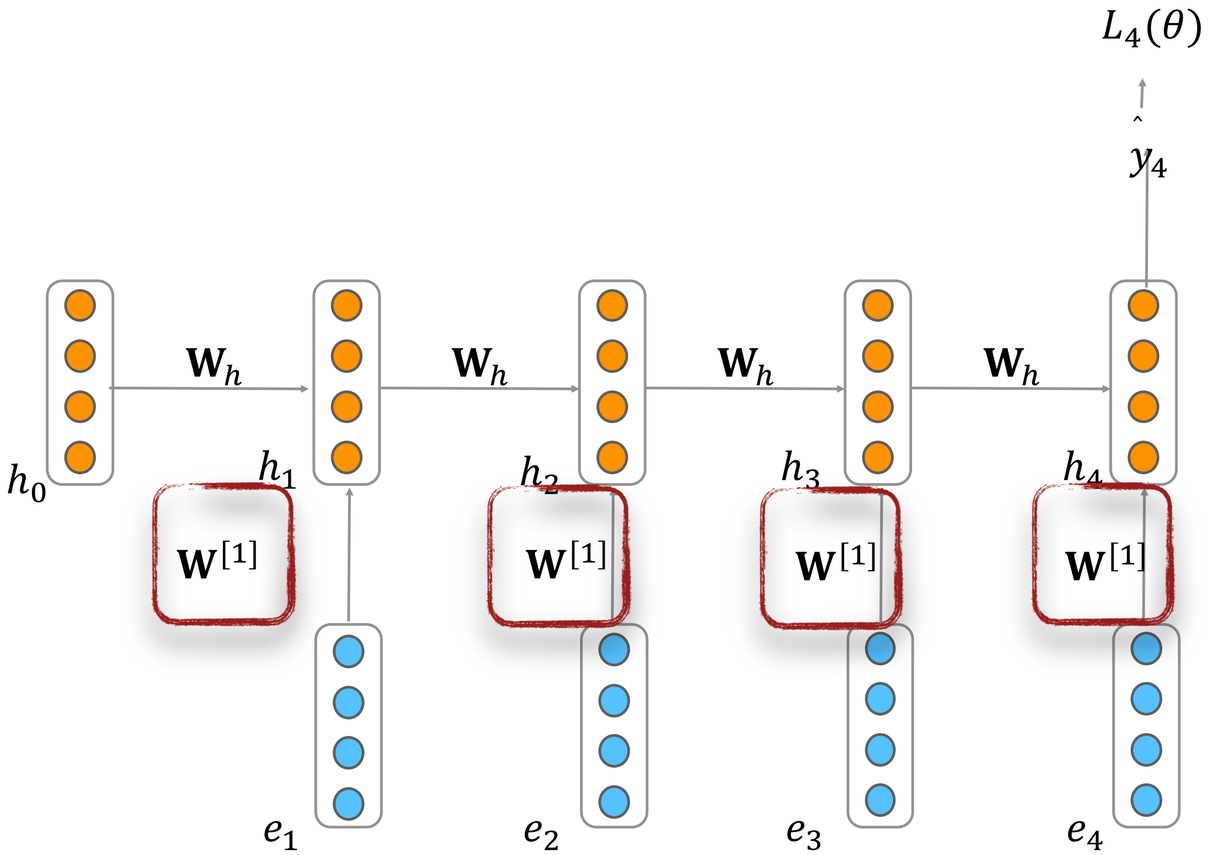
Loss



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

Training RNNs is hard

- Multiply the same matrix at each time step during forward propagation
- Ideally inputs from many time steps ago can modify output y
- This leads to something called the vanishing gradient problem



Practical Issues with training RNNs

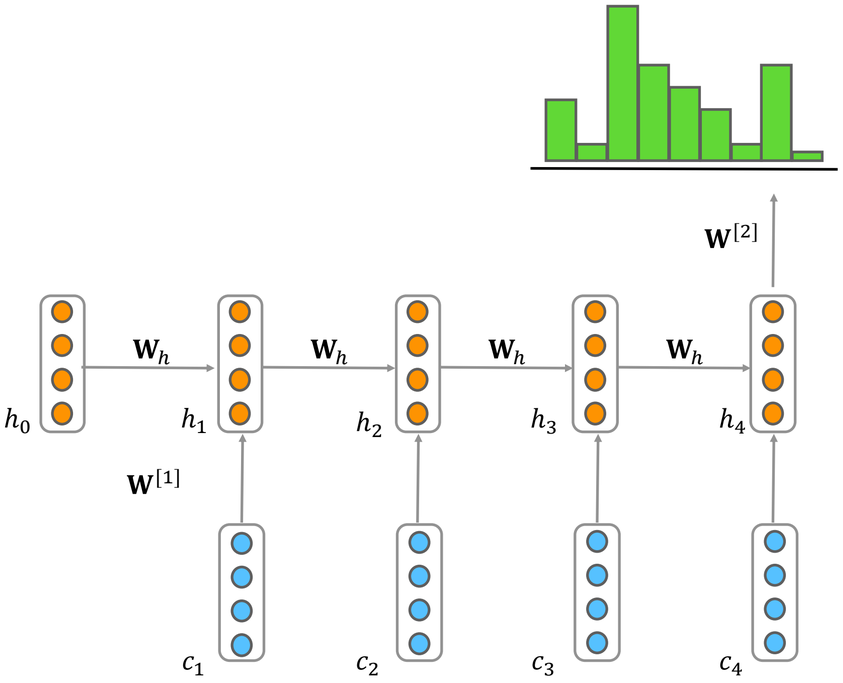
- Computing loss and gradients across entire corpus is too expensive!
- Recall: mini-batch Stochastic Gradient Descent allows us to compute loss and gradients for small chunk of data, and update.
- Solution: consider chunks of text.
  - In practice, consider $x_1, x_2, \ldots x_T$ for some T as a "sentence" or "single data instance"

$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_{CE}(\hat{y}_t, y_t)$$

- Compute loss for a sentence (actually usually a batch of sentences), compute gradients and update weigh

Summarizing RNNs

- RNNs do not have
  - the limited context problem of n-gram models
  - the fixed context limitation of feedforward LMs
  - since the hidden state can in principle represent information about all of the preceding words all the way back to the beginning of the sequence
- Training can be expensive and might lead to vanishing gradients
- More advanced architectures: LSTMs (Long Short-Term Memories)



Can be applied to both classification and generation tasks

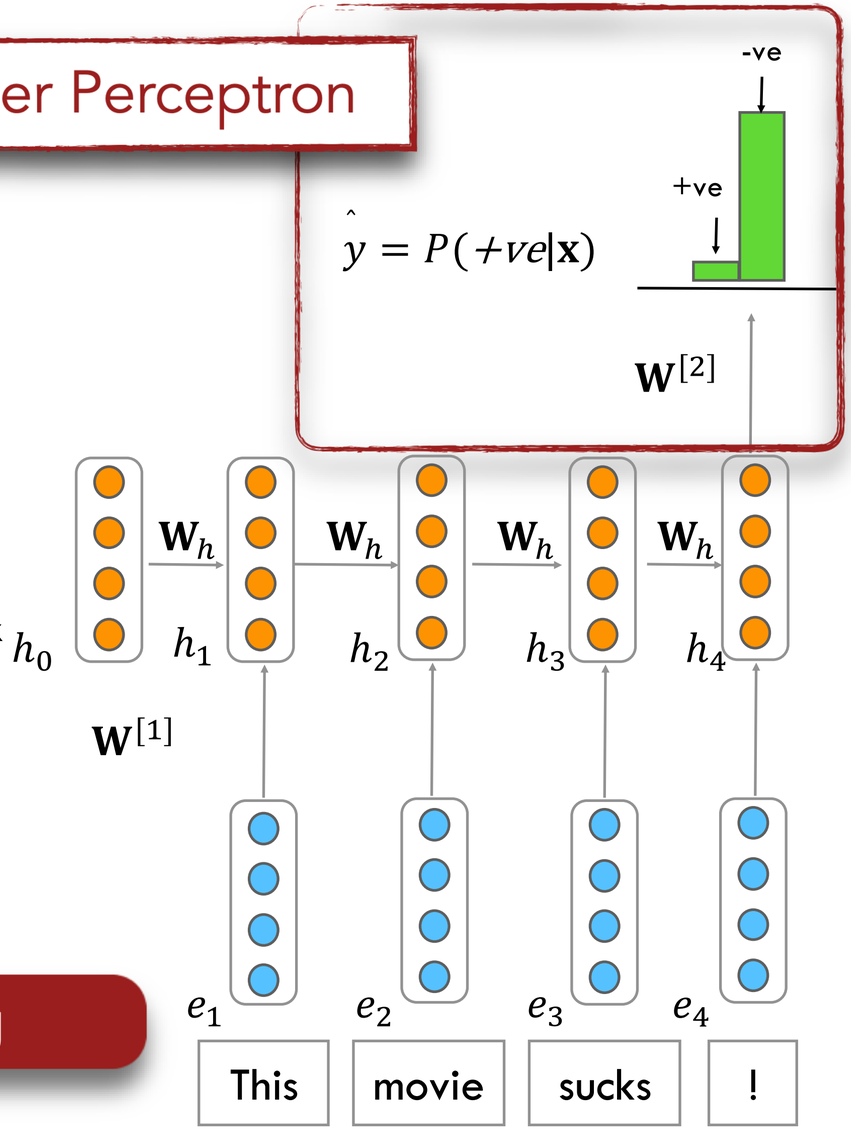
Applications

RNNs for Sequence Classification

- $\mathbf{x}$ = Entire sequence / document of length n
- y = (Multivariate) labels
- Pass $\mathbf{x}$ through the RNN one word at a time generating a new hidden layer at each time step
- Hidden layer for the last token of the text, $\mathbf{h}_n$ is a compressed representation of the entire sequence
- Pass $\mathbf{h}_n$ to a feedforward network (or multilayer perceptron) that chooses a class via a softmax over the possible classes
- Better sequence representations?
  - could also average all $\mathbf{h}_i$'s or
  - consider the maximum element along each dimension

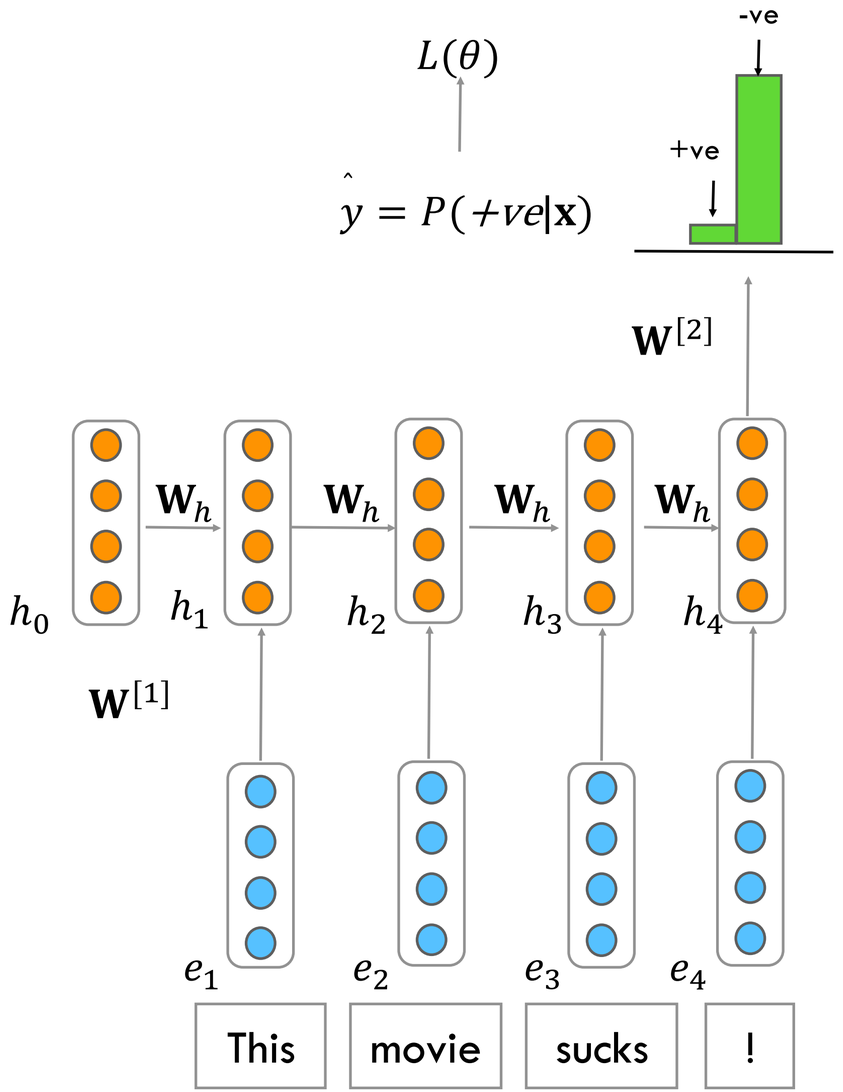
Mean pooling

Max pooling



Training RNNs for Sequence Classification

- Don't need intermediate outputs for the words in the sequence preceding the last element
- Loss function used to train the weights in the network is based entirely on the final text classification task
  - Cross-entropy loss
- Backprop: error signal from the classification is backpropagated all the way through the weights in the feedforward classifier through, to its input, and then through to the three sets of weights in the RNN



Generation with RNNLMs

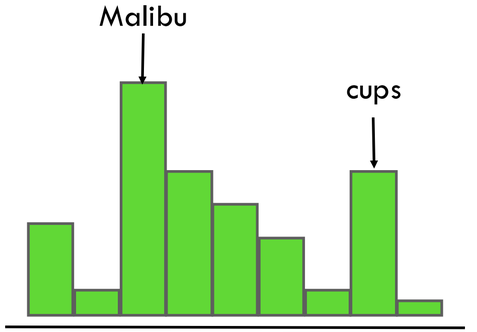
- Similar to sampling from n-gram LMs
- First randomly sample a word to begin a sequence based on its suitability as the start of a sequence
- Then continue to sample words conditioned on our previous choices until
  - we reach a pre-determined length,
  - or an end of sequence token is generated

Remember sampling from n-gram LMs?

1. Choose a random bigram (<s>, w) according to its probability
2. Now choose a random bigram (w, x) according to its probability…and so on until we choose </s>

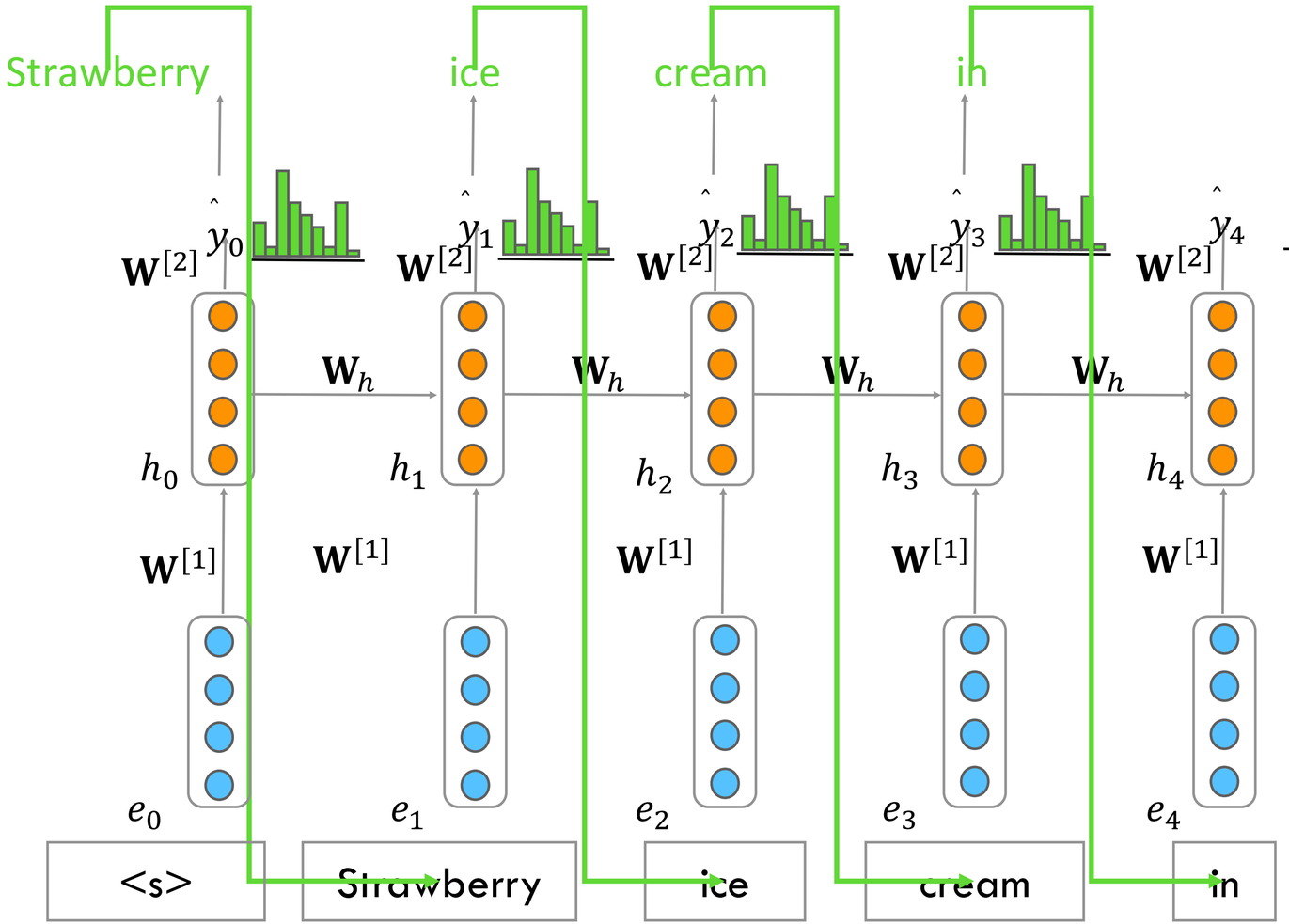
```
<s> I
    I want
      want to
           to eat
              eat Chinese
                  Chinese food
                          food  </s>
I want to eat Chinese food
```

$\hat{y}_4 = P(x_5|Strawberry\ ice\ cream\ in)$



$\text{arg}max\hat{y}_i$

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



Generation with RNNLMs

1. Sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>, as the first input.

2. Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.

3. Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.

Repeated sampling of the next word conditioned on previous choices

Autoregressive Generation

RNNLMs are Autoregressive Models

- Model that predicts a value at time t based on a function of the previous values at times $t-1$, $t-2$, and so on
- 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!
    - Machine translation, question answering, summarization
- Key technique: prime the generation with the most suitable context

Can do better than <s>!

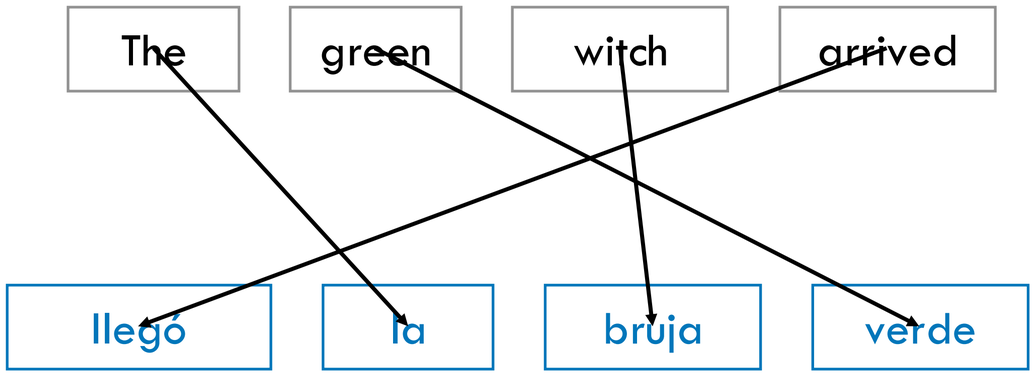
Provide rich task-appropriate context!

(Neural) Machine Translation

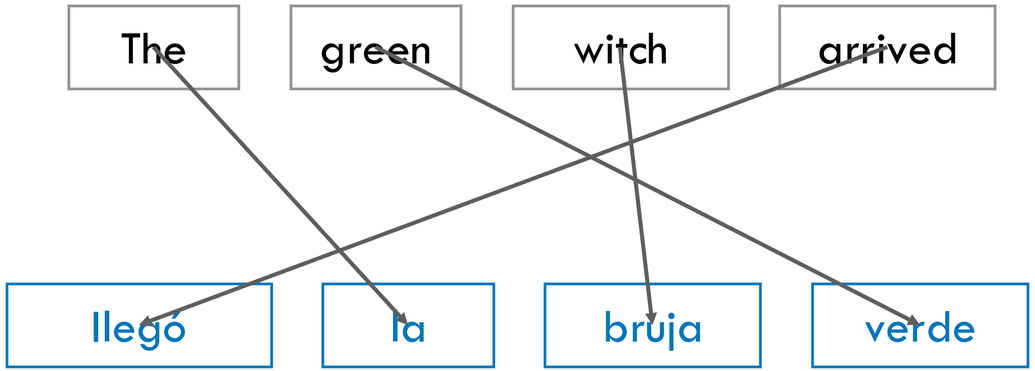
Provide rich task-appropriate context!

- Sequence Generation Problem (as opposed to sequence classification)
  - $\mathbf{x}$ = Source sequence of length n
  - $\mathbf{y}$ = Target sequence of length m
- Different from regular generation from an LM
  - Since we expect the target sequence to serve a specific utility (translate the source)

Sequence-to-Sequence (Seq2seq)



Sequence-to-Sequence Generation



- Mapping between a token in the input and a token in the output can be very indirect
  - in some languages the verb appears at the beginning of the sentence; e.g. Arabic, Hawaiian
  - in other languages at the end; e.g. Hindi
  - in other languages between the subject and the object; e.g. English
- Does not necessarily align in a word-word way!

Need a special architecture to summarize the entire context!

Sequence-to-Sequence Models

- Models capable of generating contextually appropriate, arbitrary length, output sequences given an input sequence.
- The key idea underlying these networks is the use of an encoder network that takes an input sequence and creates a contextualized representation of it, often called the context.
- This representation is then passed to a decoder network which generates a task- specific output sequence.

Encoder-Decoder Networks

Sequence-to-Sequence Modeling with Encoder-Decoder Networks

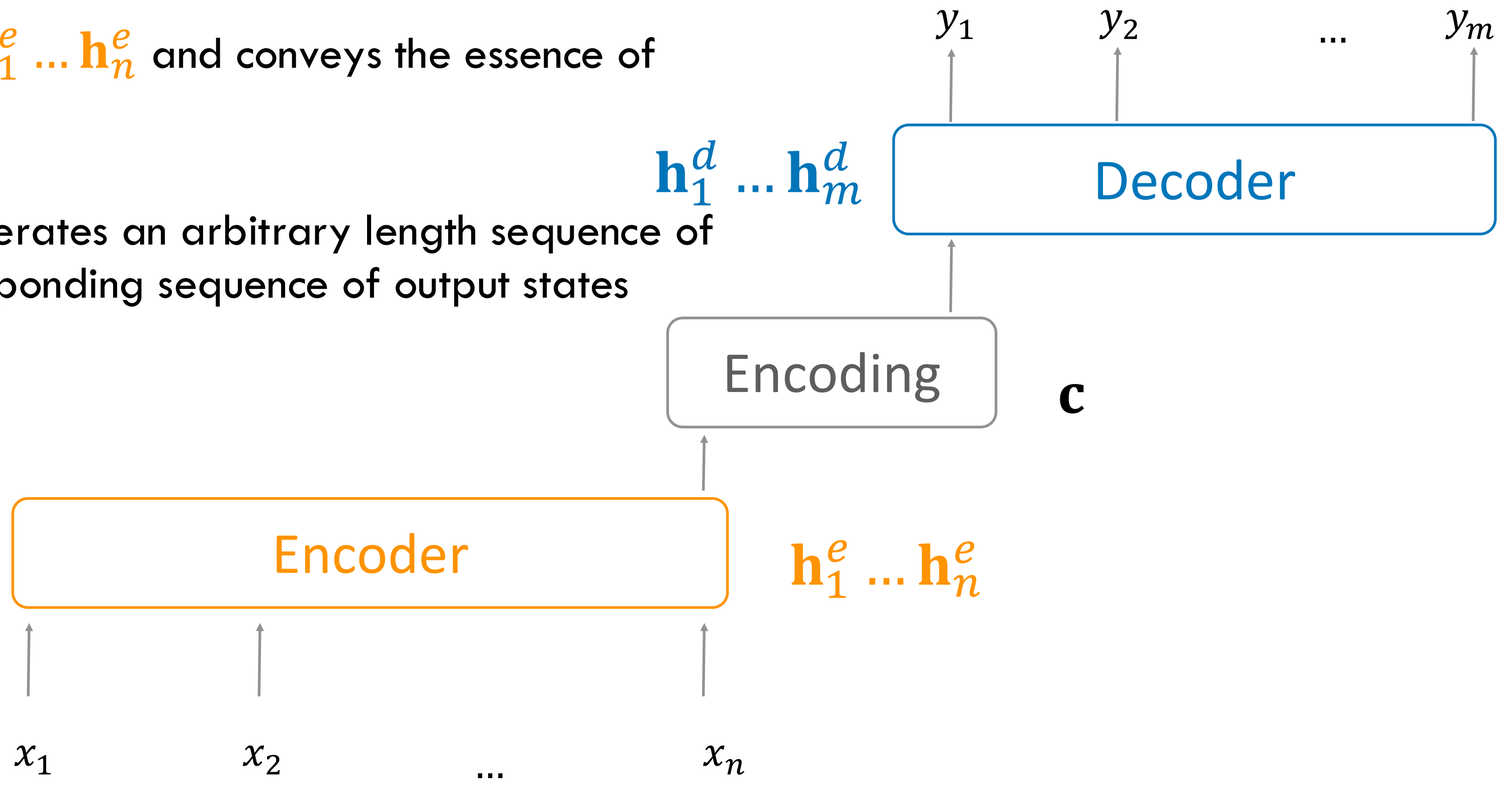
Encoder-Decoder Networks

Encoder-decoder networks consist of three components:

1. An encoder that accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_1^e \dots \mathbf{h}_n^e$

2. A encoding vector, $\mathbf{c}$ which is a function of $\mathbf{h}_1^e \dots \mathbf{h}_n^e$ and conveys the essence of the input to the decoder

3. A decoder which accepts $\mathbf{c}$ as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_1^d \dots \mathbf{h}_m^d$, from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained

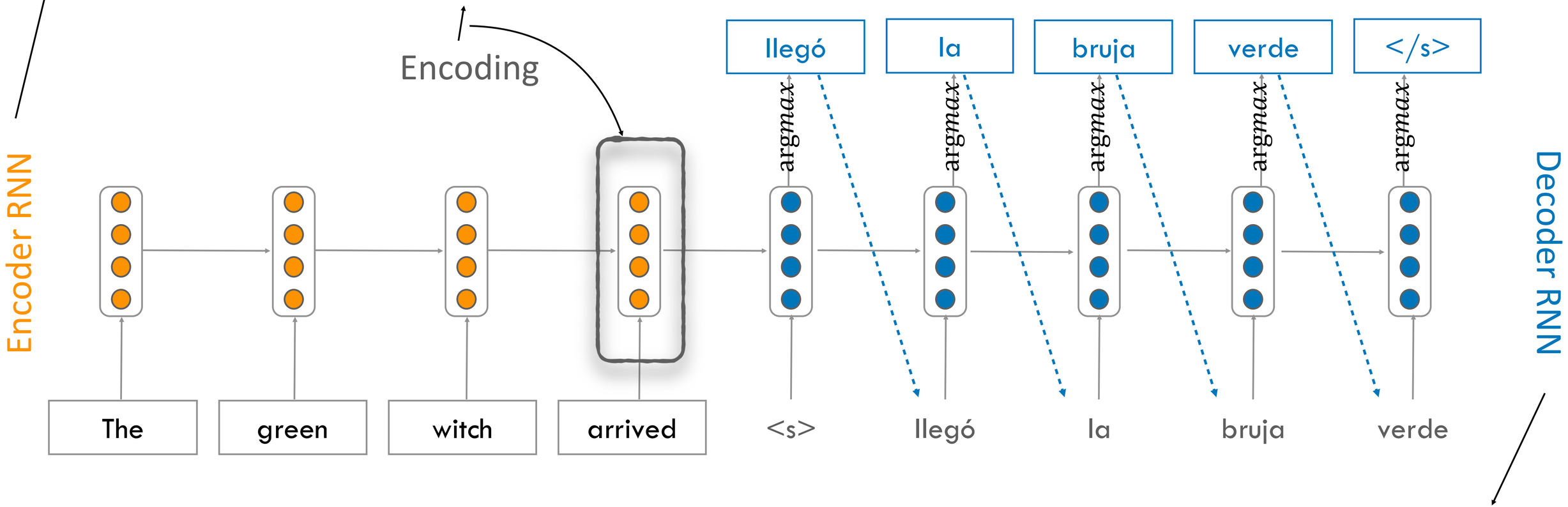


Encoders and decoders can be made of FFNNs, RNNs, or Transformers

Produces an encoding of the source sequence

Represents input sequence. Provides initial hidden state for Decoder RNN

Target Sentence **y**



Source Sentence **x**

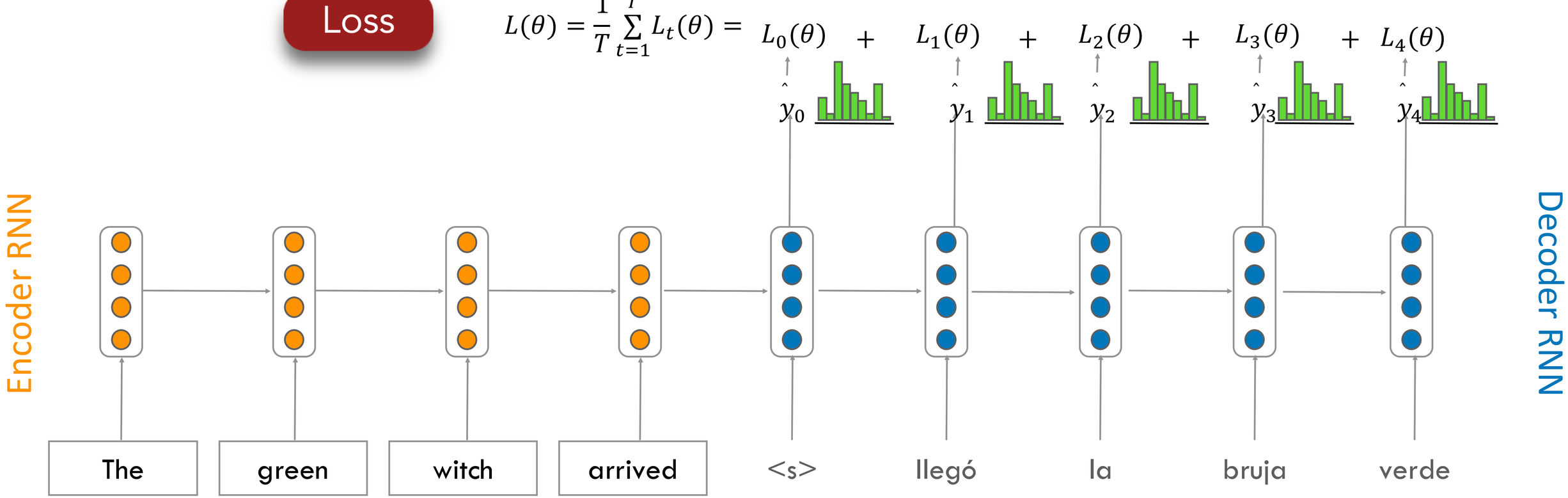
Language Model that produces the target sentence conditioned on the encoding

negative log prob. of "llegó"

negative log prob. of "</s>"

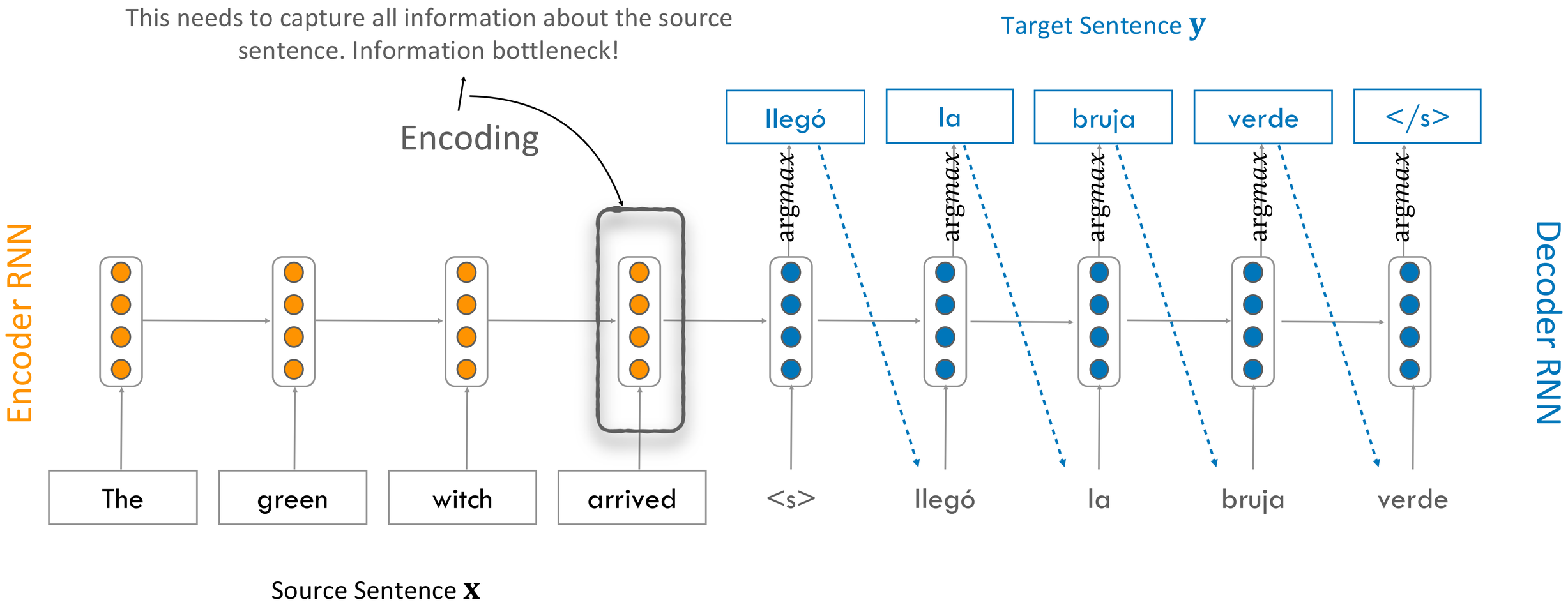
Loss

$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L_t(\theta) = L_0(\theta) + L_1(\theta) + L_2(\theta) + L_3(\theta) + L_4(\theta)$$



Source Sentence $\mathbf{x}$

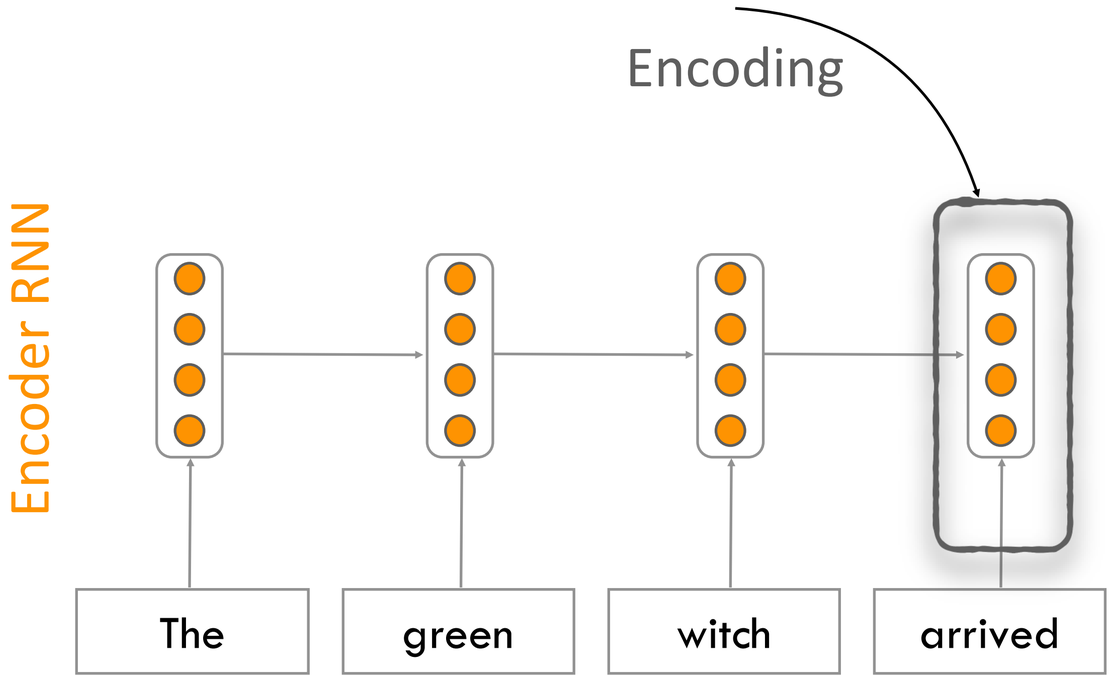
Target Sentence $\mathbf{y}$

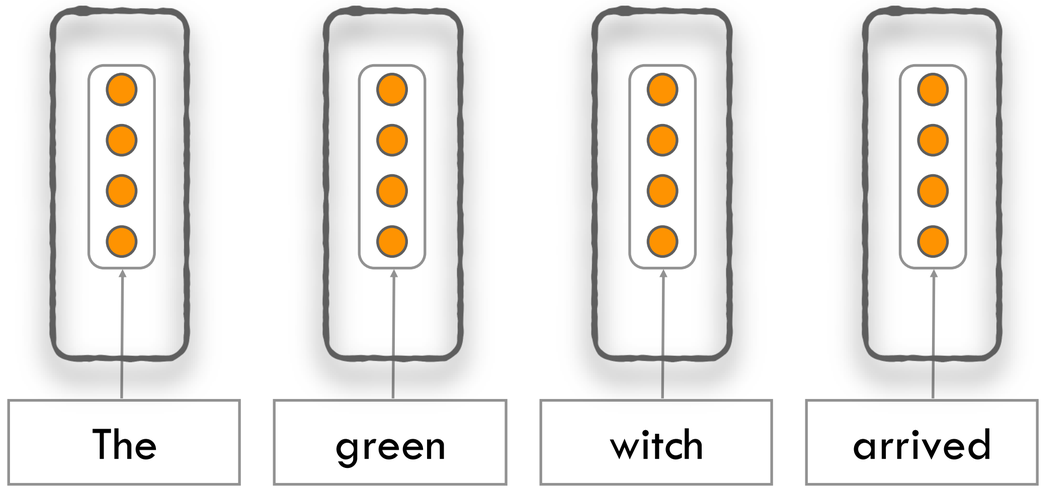

This needs to capture all information about the source sentence. Information bottleneck!
Encoding
Target Sentence y
Encoder RNN
Decoder RNN
llegó
la
bruja
verde
</s>
argmax
The
green
witch
arrived
<s>
llegó
la
bruja
verde
Source Sentence x

"you can't cram the meaning of a whole %&@#&ing sentence into a single $*(&@ing vector!"

– Ray Mooney, Professor of Computer Science, UT Austin

Information Bottleneck: One Solution





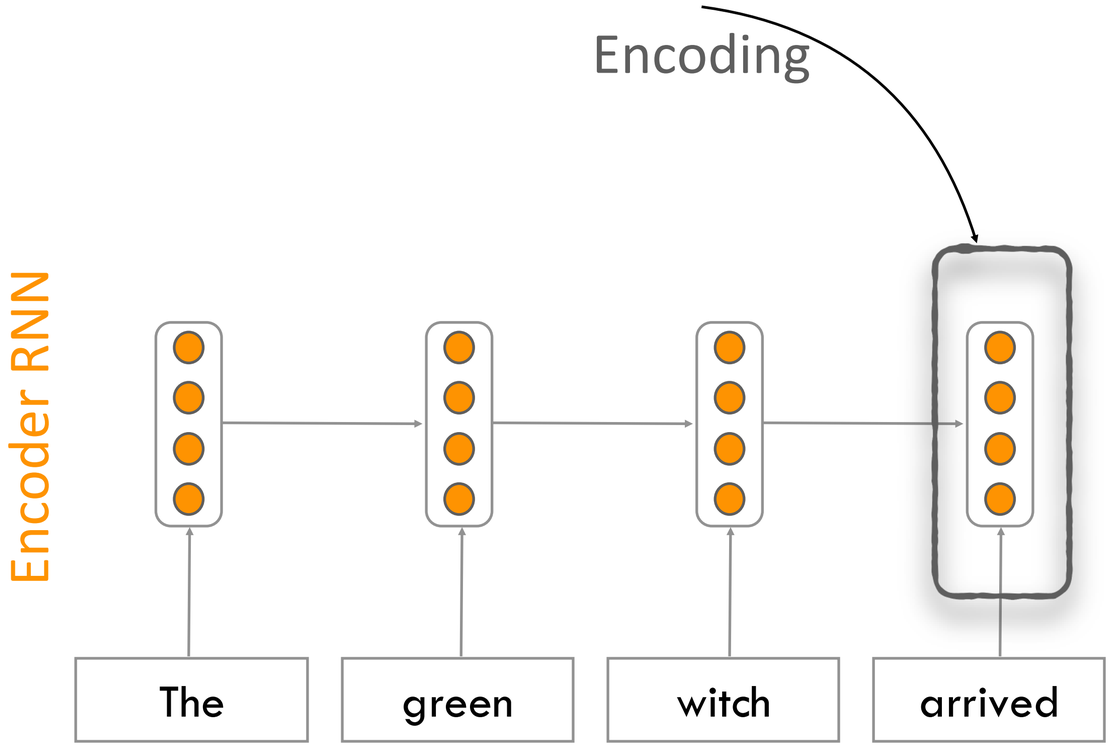
What if we had access to all hidden states?

How to create this?

Attention Mechanism

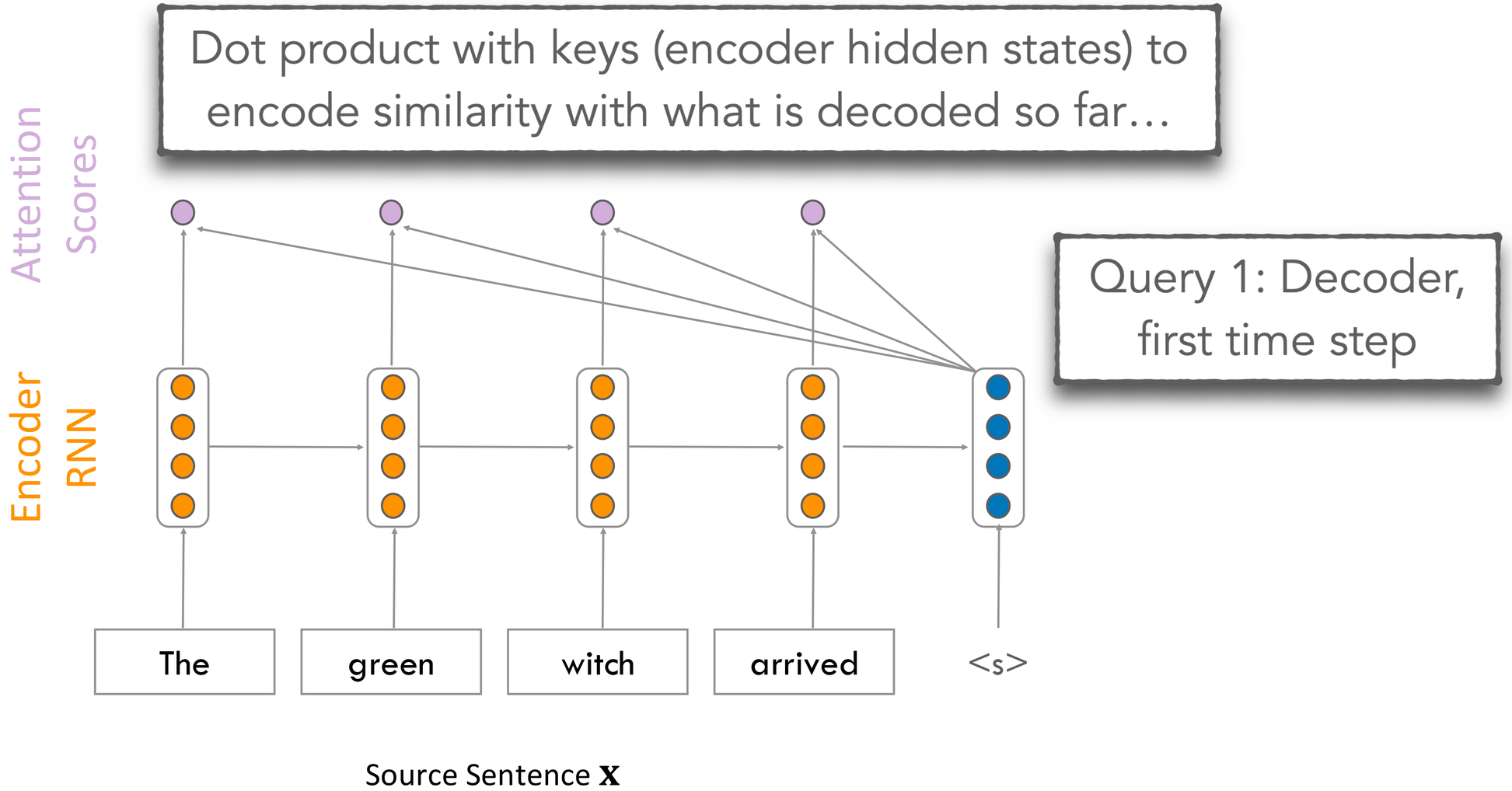
Attention Mechanism

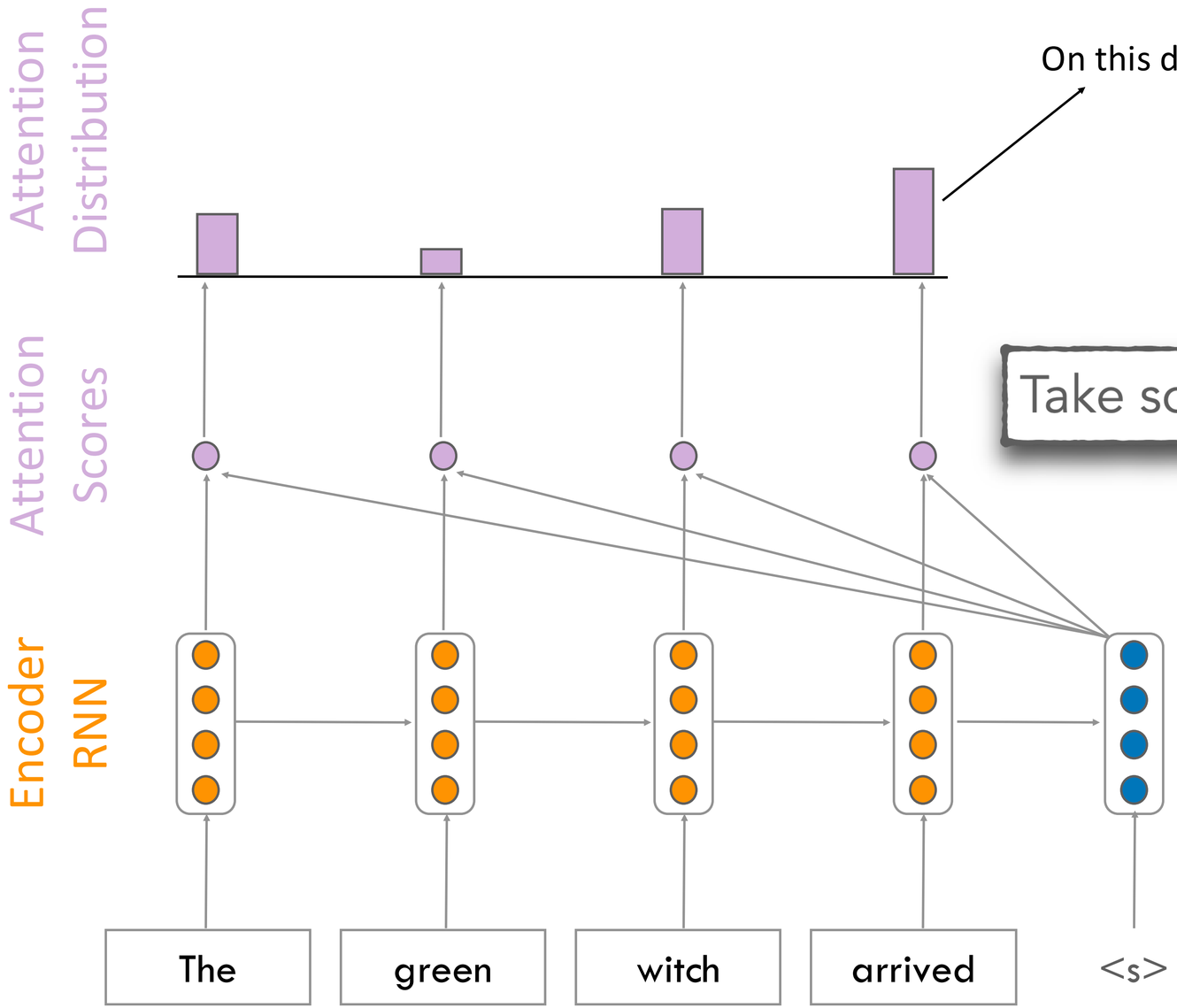
- Attention mechanisms allow the decoder to focus on a particular part of the source sequence at each time step
- Single fixed-length vector $\mathbf{c}_t$ by taking a weighted sum of all the encoder hidden states
  - One per time step of the decoder!
- In general, we have a single query vector and multiple key vectors.
- We want to score each query-key pair



Bahdanau et al., 2015

Seq2Seq with Attention

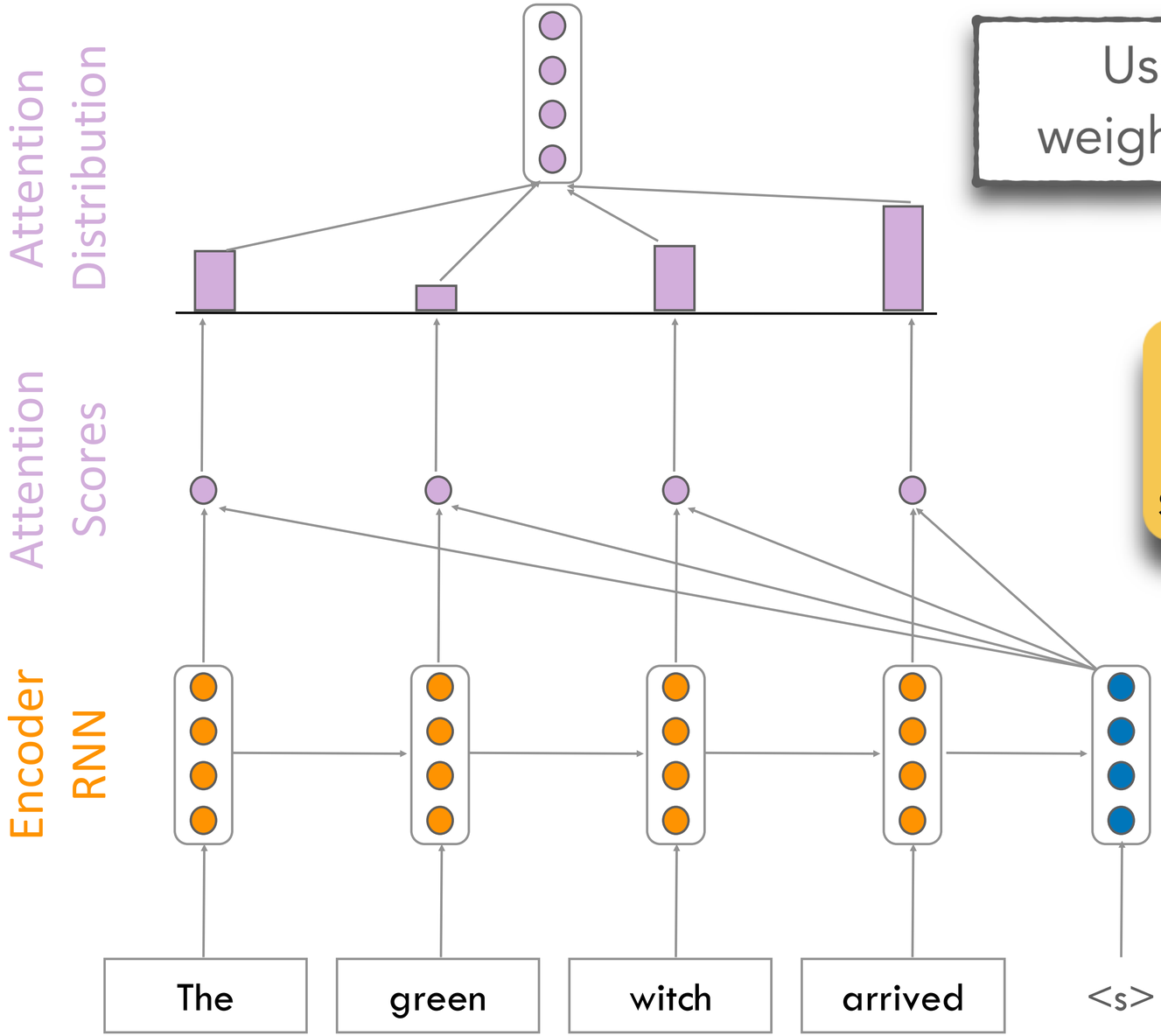




On this decoder tilmestep we are mostly focusing on the source token “arrived”

Take softmax to turn into probability distribution

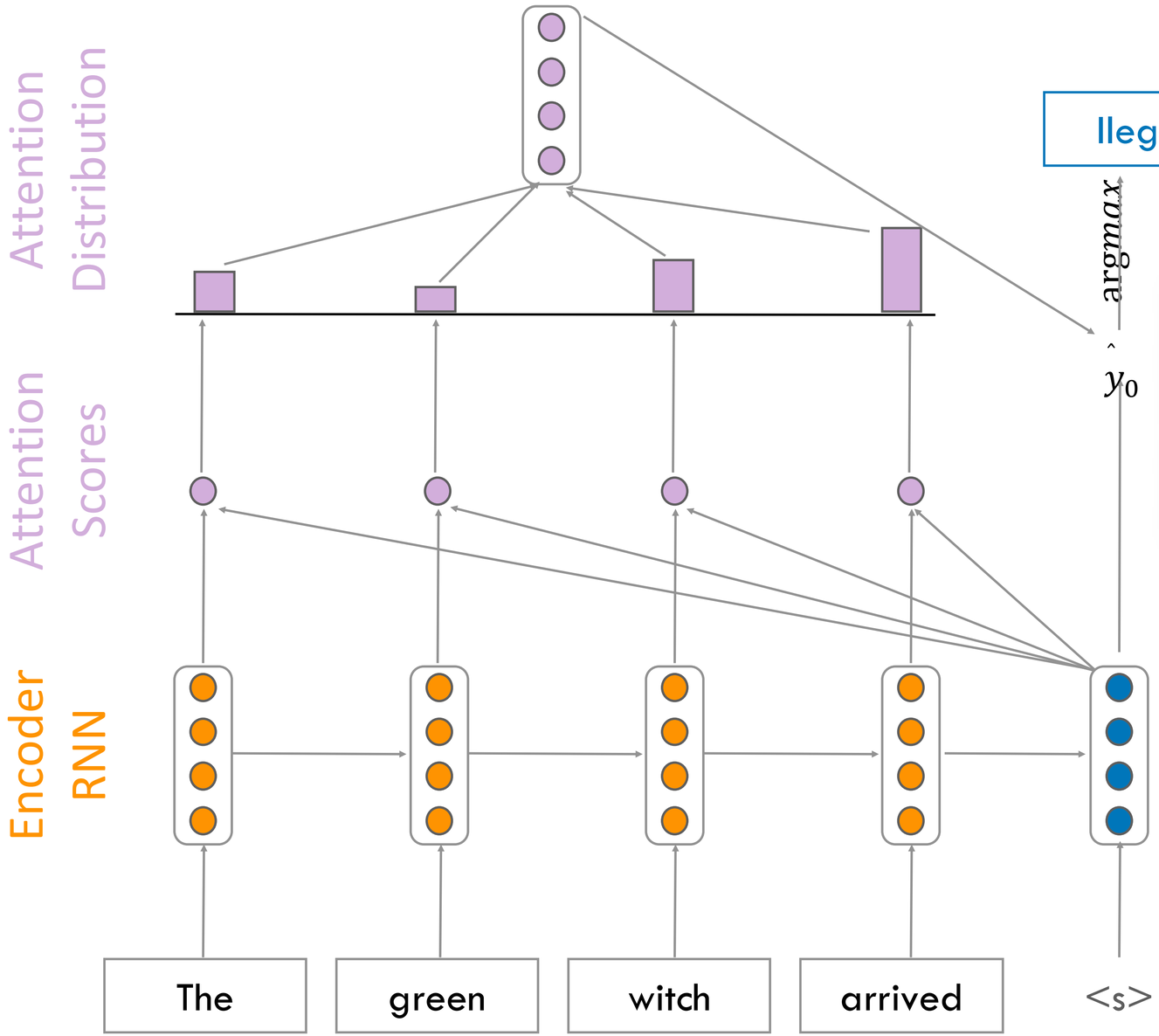
Source Sentence $\mathbf{x}$



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

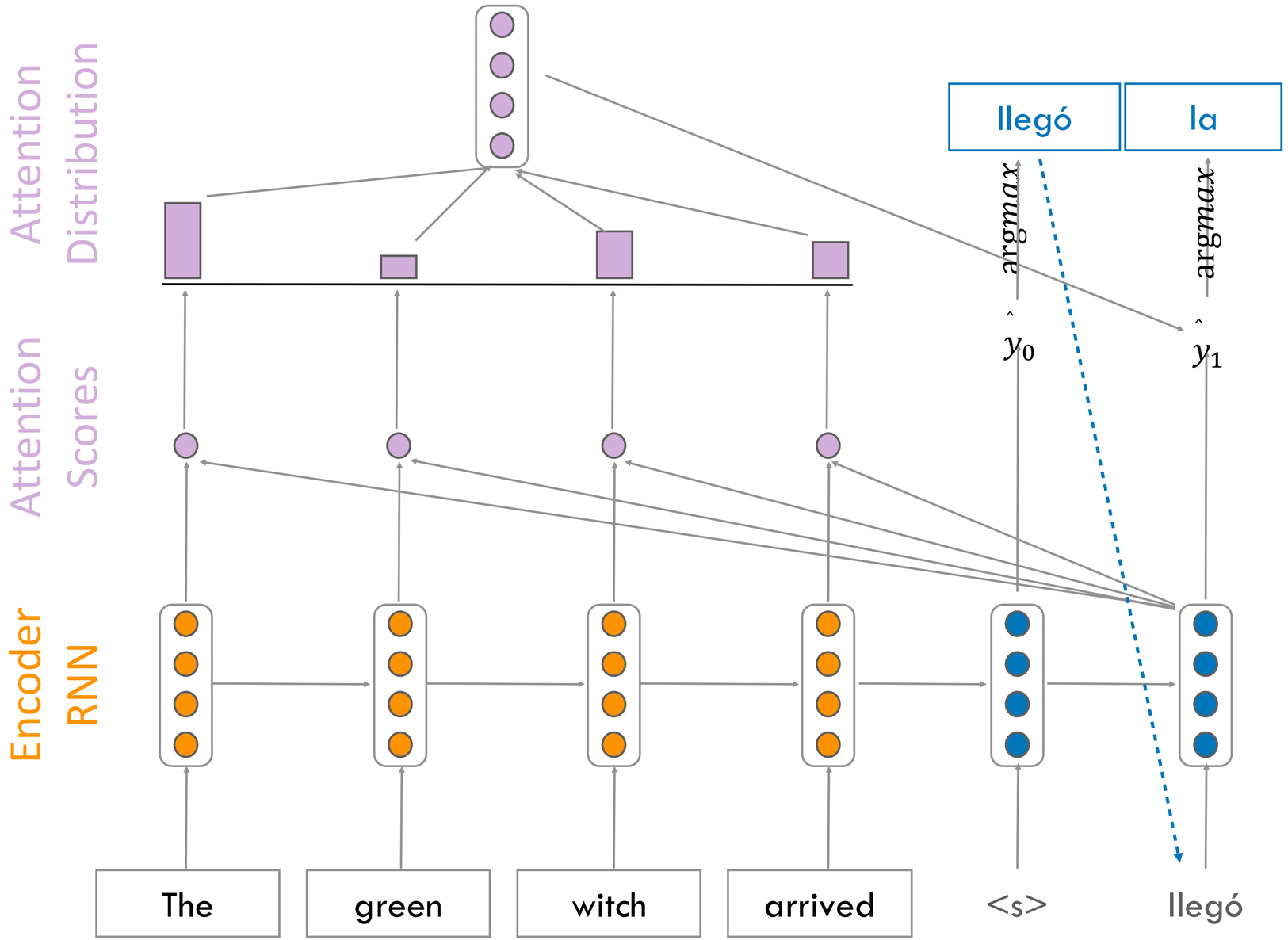
The attention output mostly contains information the hidden states that received high attention.

Source Sentence $\mathbf{x}$



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

Source Sentence $\mathbf{x}$

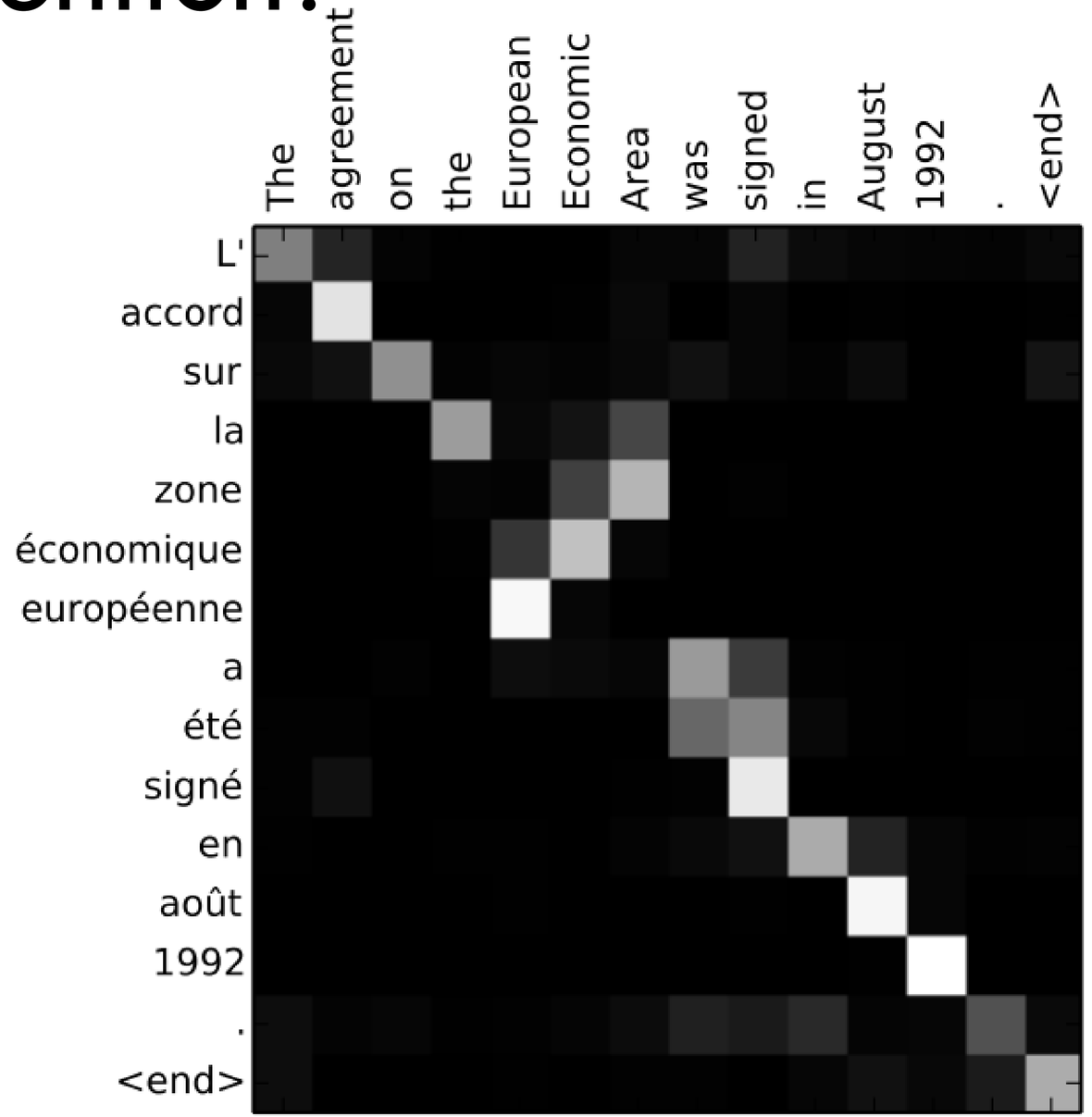


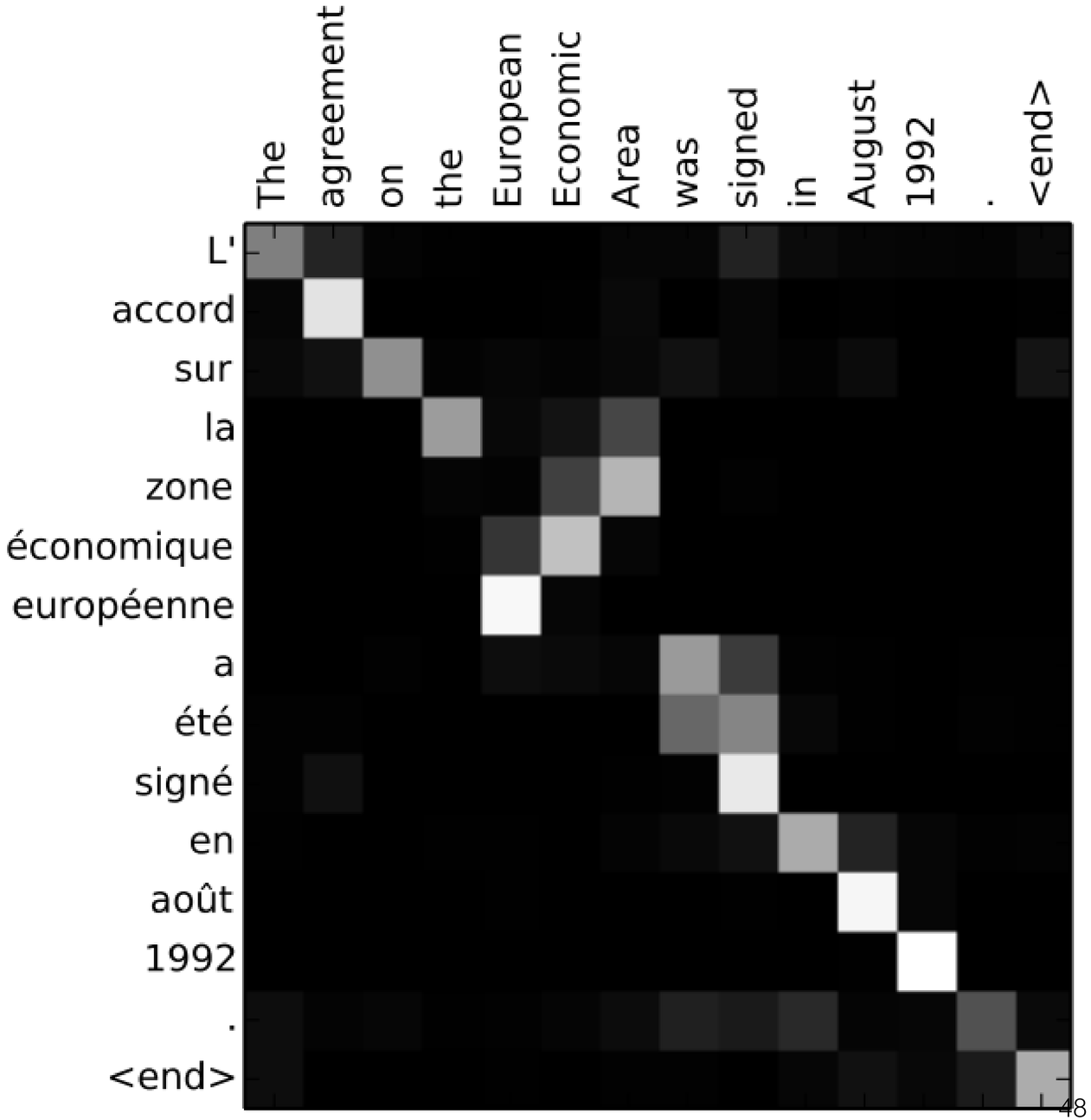
Query 2: Decoder, second time step

Source Sentence **x**

Why Attention?

- Attention significantly improves neural machine translation performance
  - Very useful to allow decoder to focus on certain parts of the source
- Attention solves the information bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on →
  - We get alignment for free! We never explicitly trained an alignment system! The network just learned alignment by itself





Seq2Seq Summary

- Seq2Seq modeling is popular for close-ended generation tasks
  - MT, Summarization, QA
  - Involves an encoder and a decoder
    - Can be any neural architecture!
- Popular Seq2Seq Models using Transformers: BART, T5
- Secret Sauce: Attention

More on Attention

Attention Variants

- In general, we have some values $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and a query $\mathbf{q} \in \mathbb{R}^{d_2}$
- Attention always involves

1. Computing the attention scores, $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$

Can be done in multiple ways!

2. Taking softmax to get attention distribution $\alpha_t = softmax(e(\mathbf{q}, \mathbf{h}_{1:N})) \in [0,1]^N$

3. Using attention distribution to take weighted sum of values:

$$\mathbf{c}_t^{att} = \sum_{i=1}^{N} \alpha_{t,i} \mathbf{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the attention output $\mathbf{c}_t^{att}$ (sometimes called the context vector)

Attention Variants

- There are several ways you can compute $e(\mathbf{q}, \mathbf{h}_{1:N}) \in \mathbb{R}^N$ from $\mathbf{h}_1 \ldots \mathbf{h}_N \in \mathbb{R}^{d_1}$ and $\mathbf{q} \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q} \cdot \mathbf{h}_j]_{j=1:N}$
  - This assumes $d_1 = d_2$
  - We applied this in encoder-decoder RNNs
- Multiplicative attention: $e(\mathbf{q}, \mathbf{h}_{1:N}) = [\mathbf{q}^T \mathbf{W} \mathbf{h}_j]_{j=1:N}$
  - Where $\mathbf{W} \in \mathbb{R}^{d_1 \times d_2}$ is a learned weight matrix.
  - Also called "bilinear attention"

More on Attention

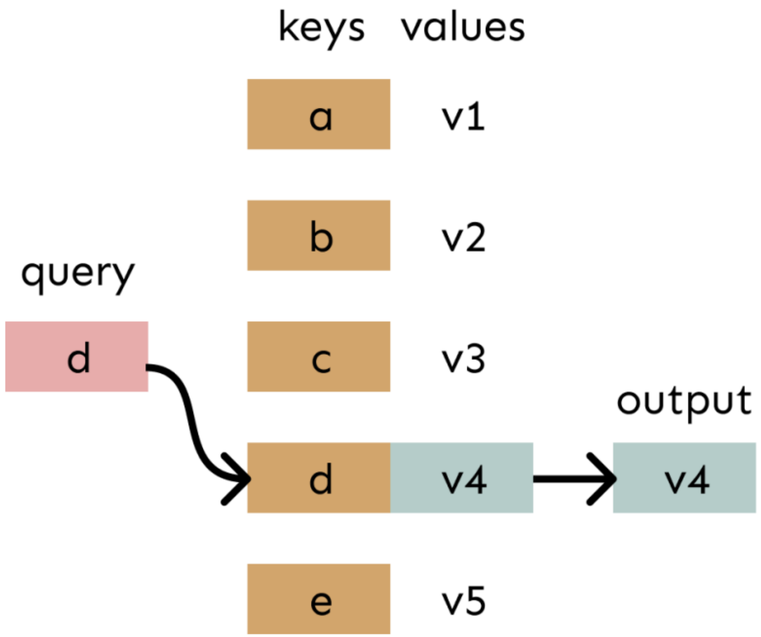
Given a set of vector values, and a vector query, attention is a technique to compute a weighted sum of the values, dependent on the query

- We sometimes say that the query attends to the values.
  - For example, in the seq2seq + attention model, each decoder hidden state (query) attends to all the encoder hidden states (values)
- The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
- Attention is a powerful, flexible, general deep learning technique in all deep learning models.
  - A new idea from after 2010! Originated in NMT

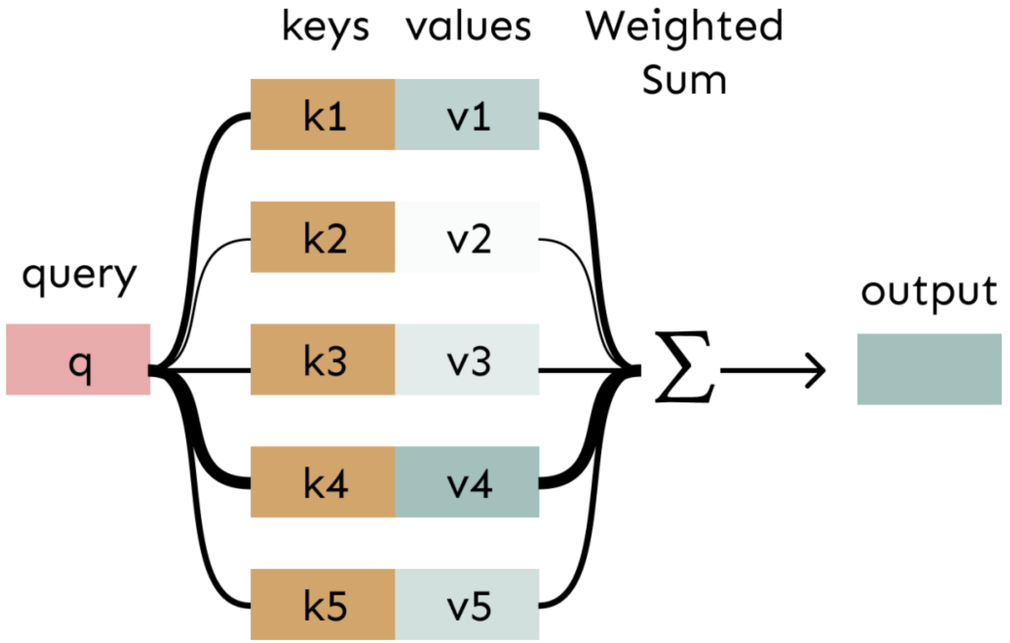
Attention and lookup tables

Attention performs fuzzy lookup in a key-value store

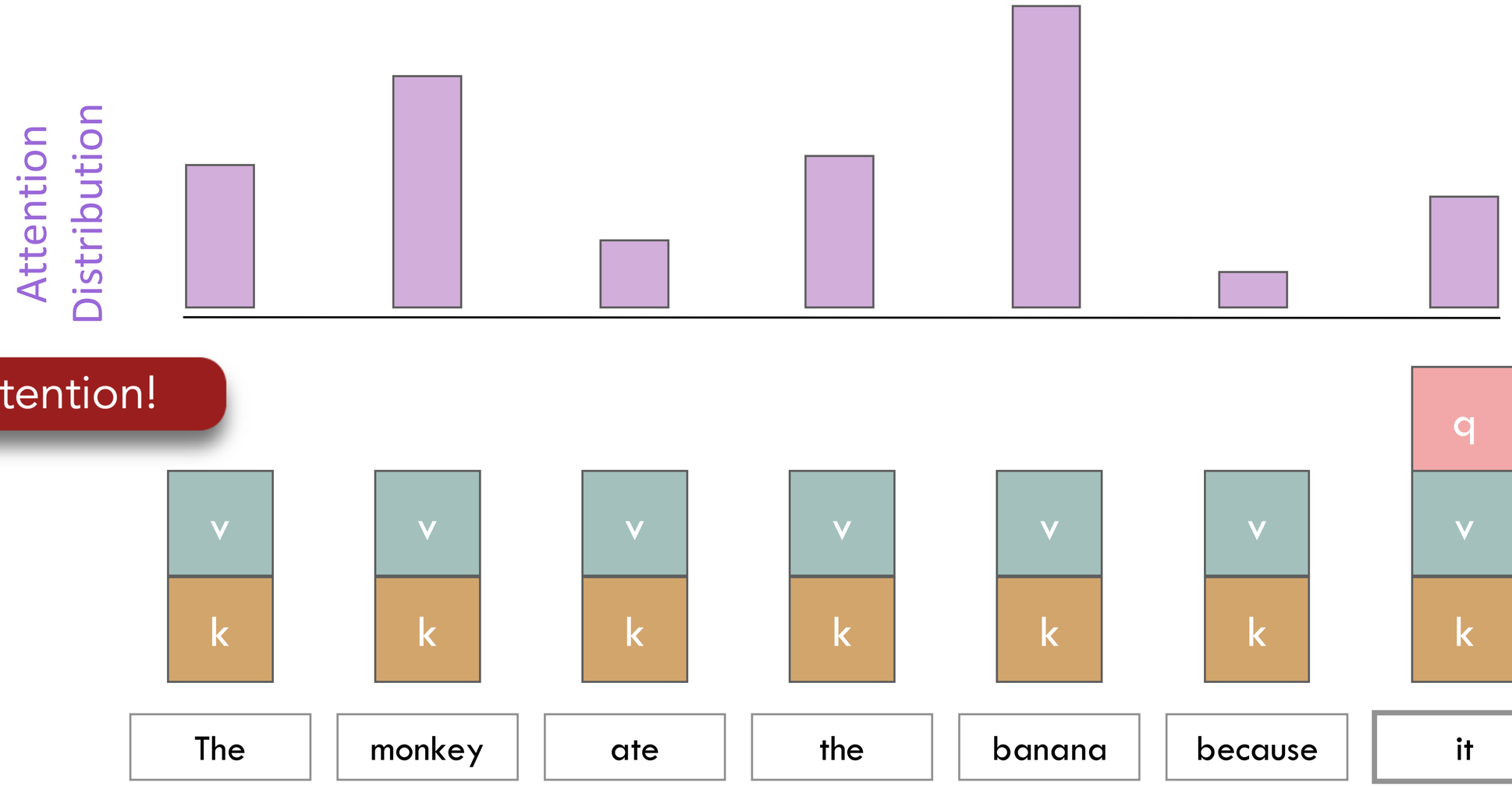
In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.



In attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.

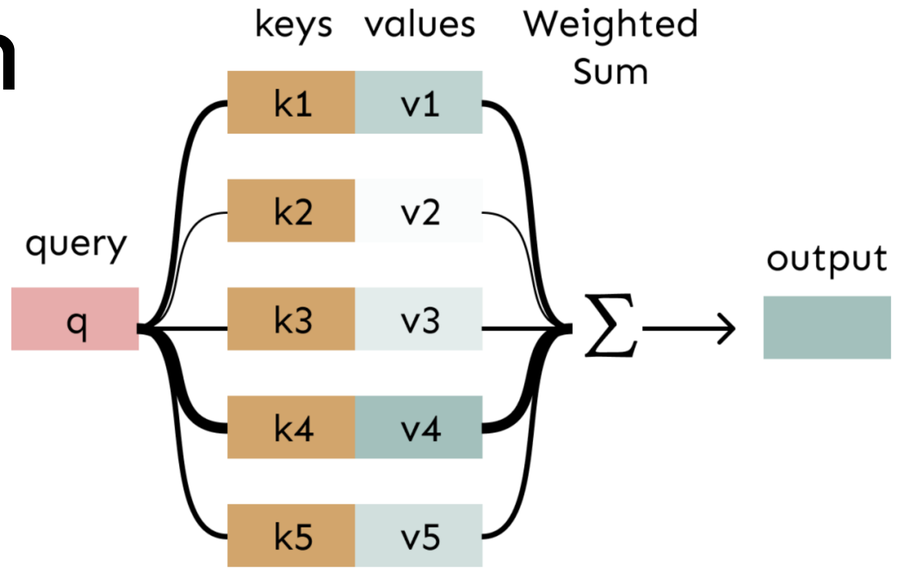


Attention in the decoder



Transformers: Self-Attention

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 $\mathbb{R}^{d \times d}$

$q_i = Q\boldsymbol{x_i}$ (queries) $\quad k_i = K\boldsymbol{x_i}$ (keys) $\quad v_i = V\boldsymbol{x_i}$ (values)

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

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

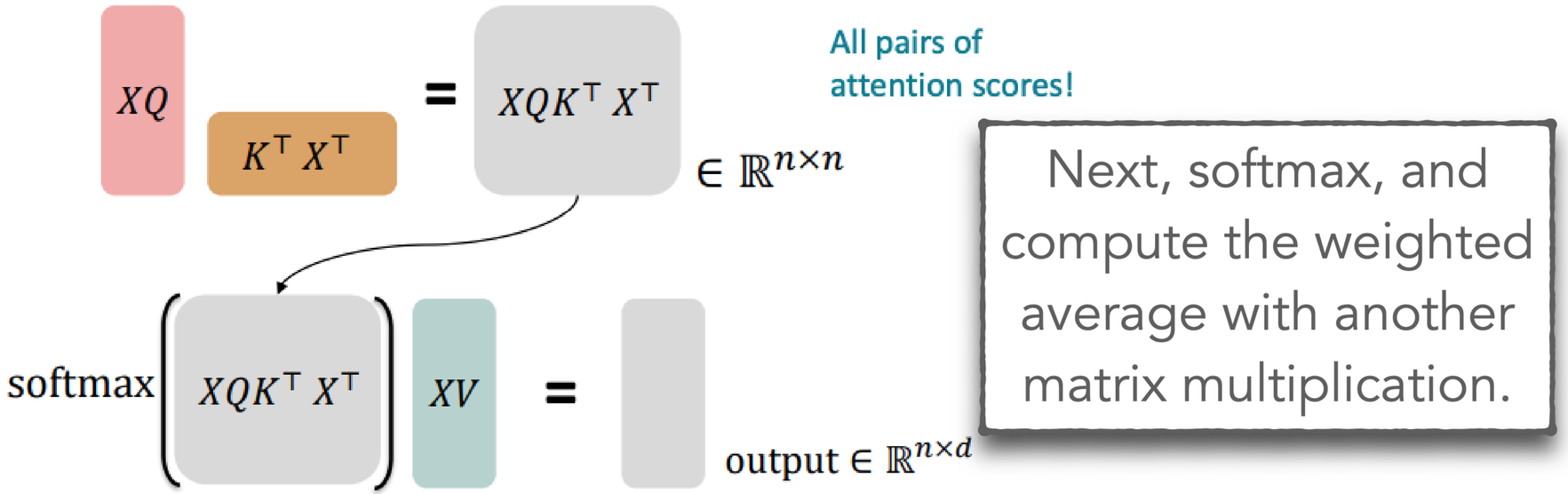
3. Compute output for each word as weighted sum of values

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

Self-Attention as Matrix Multiplications

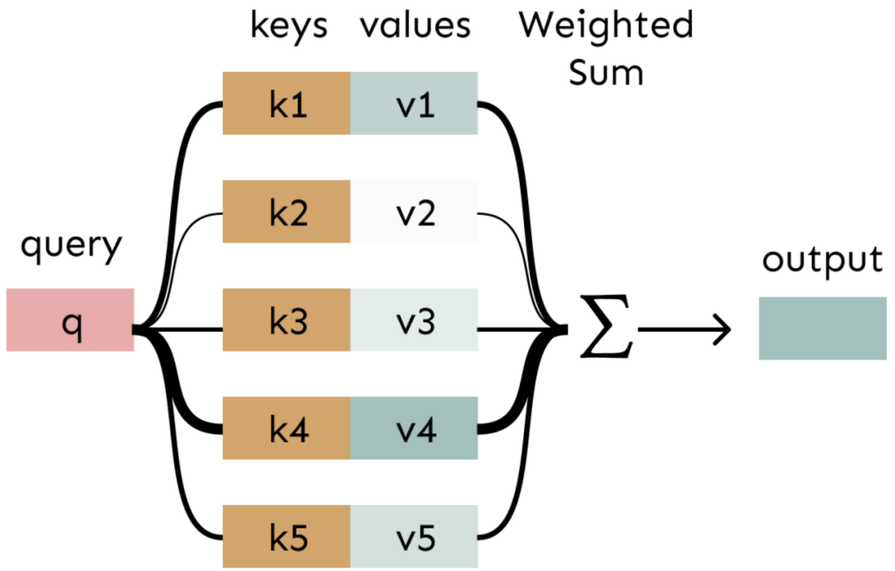
- Key-query-value attention is typically computed as matrices.
  - Let $\mathbf{X} = [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - First, note that $\mathbf{XK} \in \mathbb{R}^{n \times d}$, $\mathbf{XQ} \in \mathbb{R}^{n \times d}$, and $\mathbf{XV} \in \mathbb{R}^{n \times d}$
  - The output is defined as $\mathit{softmax}(\mathbf{XQ}(\mathbf{XK})^T)\mathbf{XV} \in \mathbb{R}^{n \times d}$

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



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

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
- Used often with feedforward networks!

Transformers are Self-Attention Networks

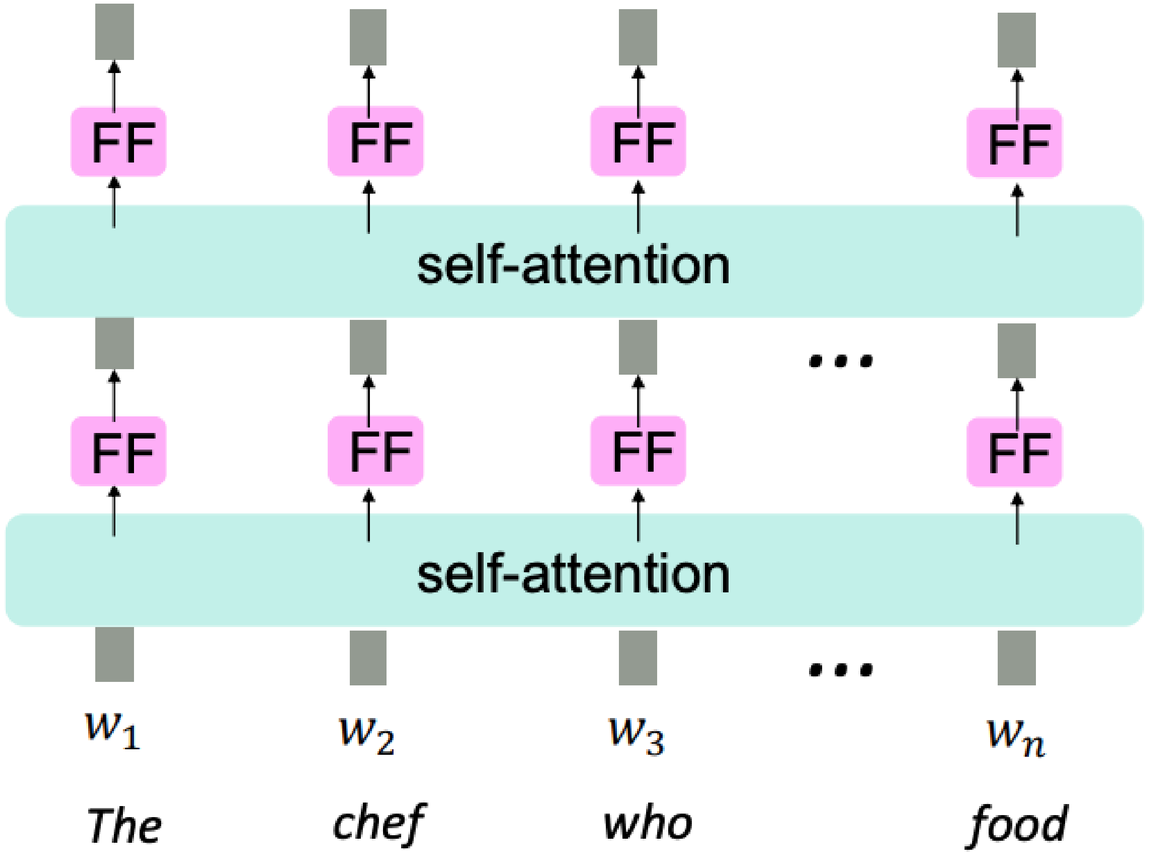
- Self-Attention is the key innovation behind Transformers!
- Transformers map sequences of input vectors $(x_1, \ldots, x_n)$ to sequences of output vectors $(y_1, \ldots, y_n)$ of the same length.
- Made up of stacks of Transformer blocks
  - each of which is a multilayer network made by combining
    - simple linear layers,
    - feedforward networks, and
    - self-attention layers

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.

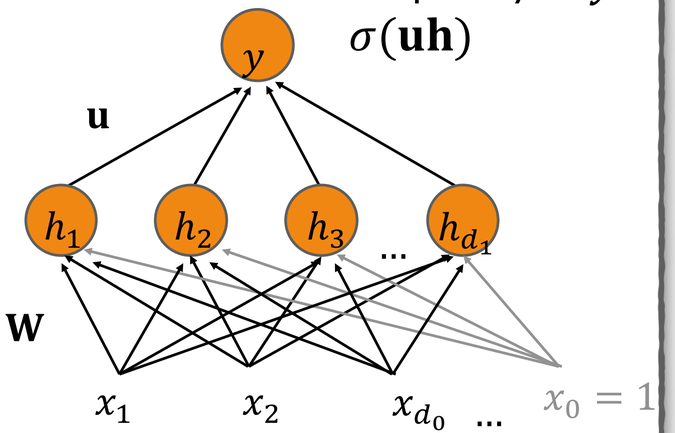


Hidden layer: $\mathbf{h} = g(\mathbf{W}\mathbf{x}) = g(\sum_{i=0}^{d_0} \mathbf{W}_{ji}\mathbf{x}_i)$

Usually ReLU or tanh

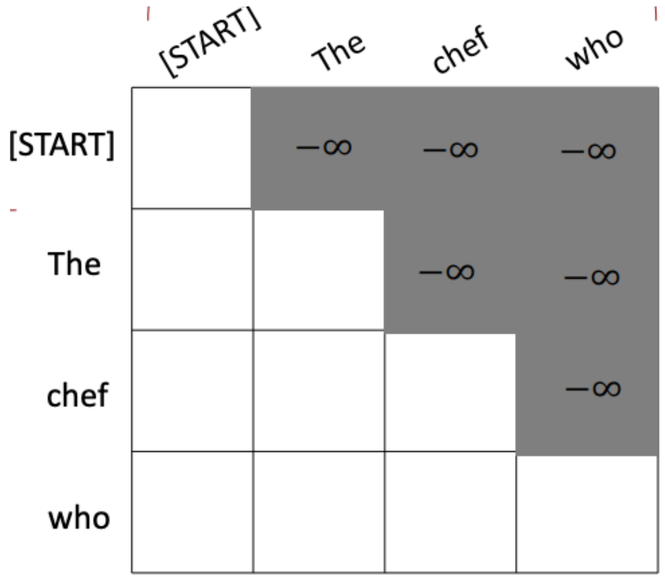
Input layer: vector $\mathbf{x}$

Output layer: $y = \sigma(\mathbf{u}\mathbf{h})$



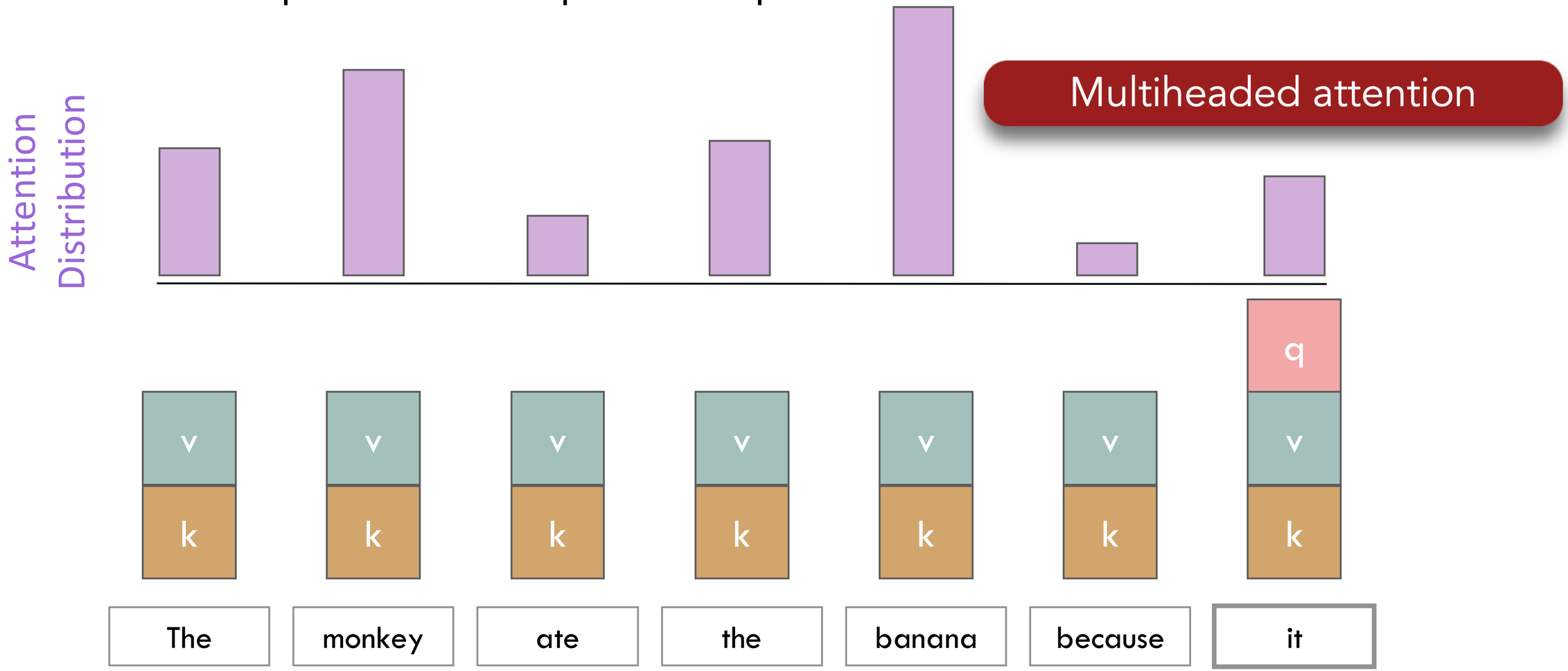
Self Attention and Future Information

- 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.
- 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 $-\infty$



Self-Attention and Heads

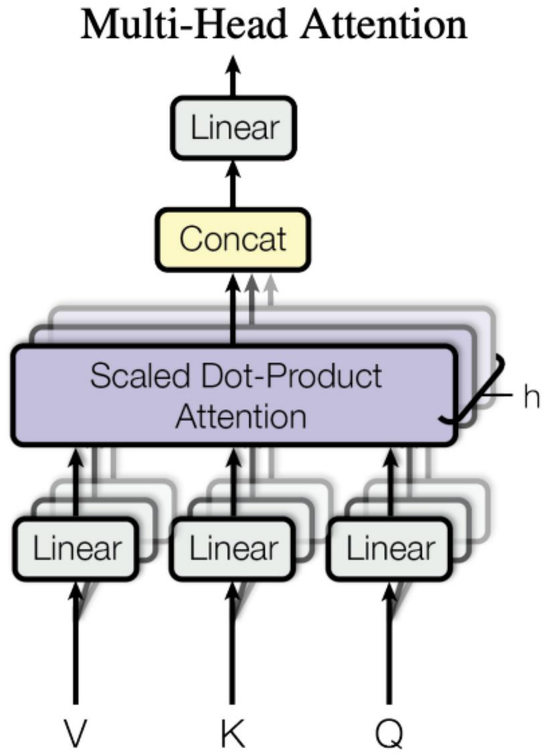
- What if we needed to pay attention to multiple different kinds of things e.g. entities, syntax
- Solution: Consider multiple attention computations in parallel



Transformers: 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?
- 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!



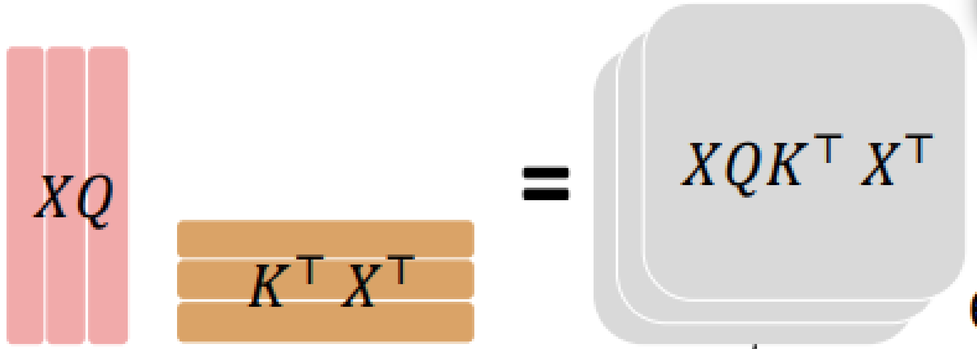
Each head gets to "look" at different things, and construct value vectors differently

Multiheaded Attention: Visualization

Still efficient, can be parallelized!

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

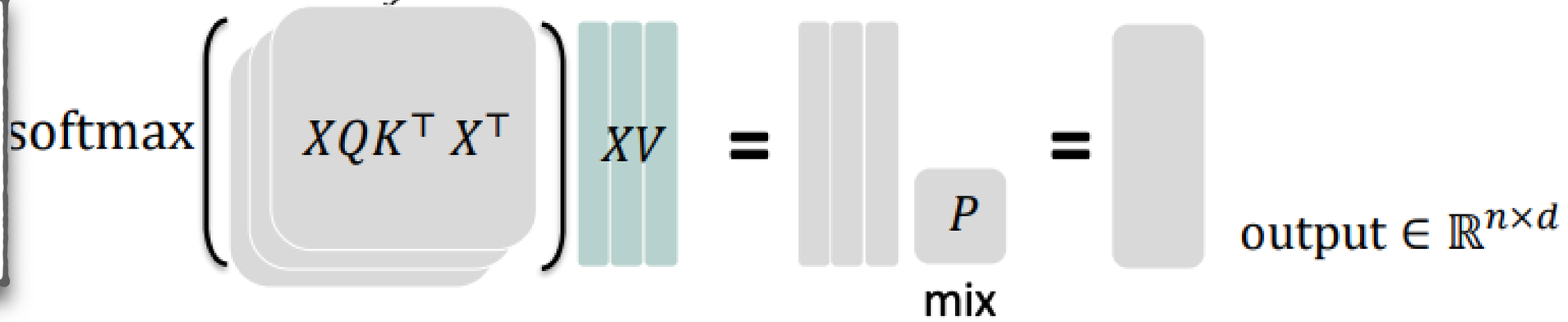
Tensor!



3 sets of all pairs of attention scores!

$\in \mathbb{R}^{3\times n\times n}$

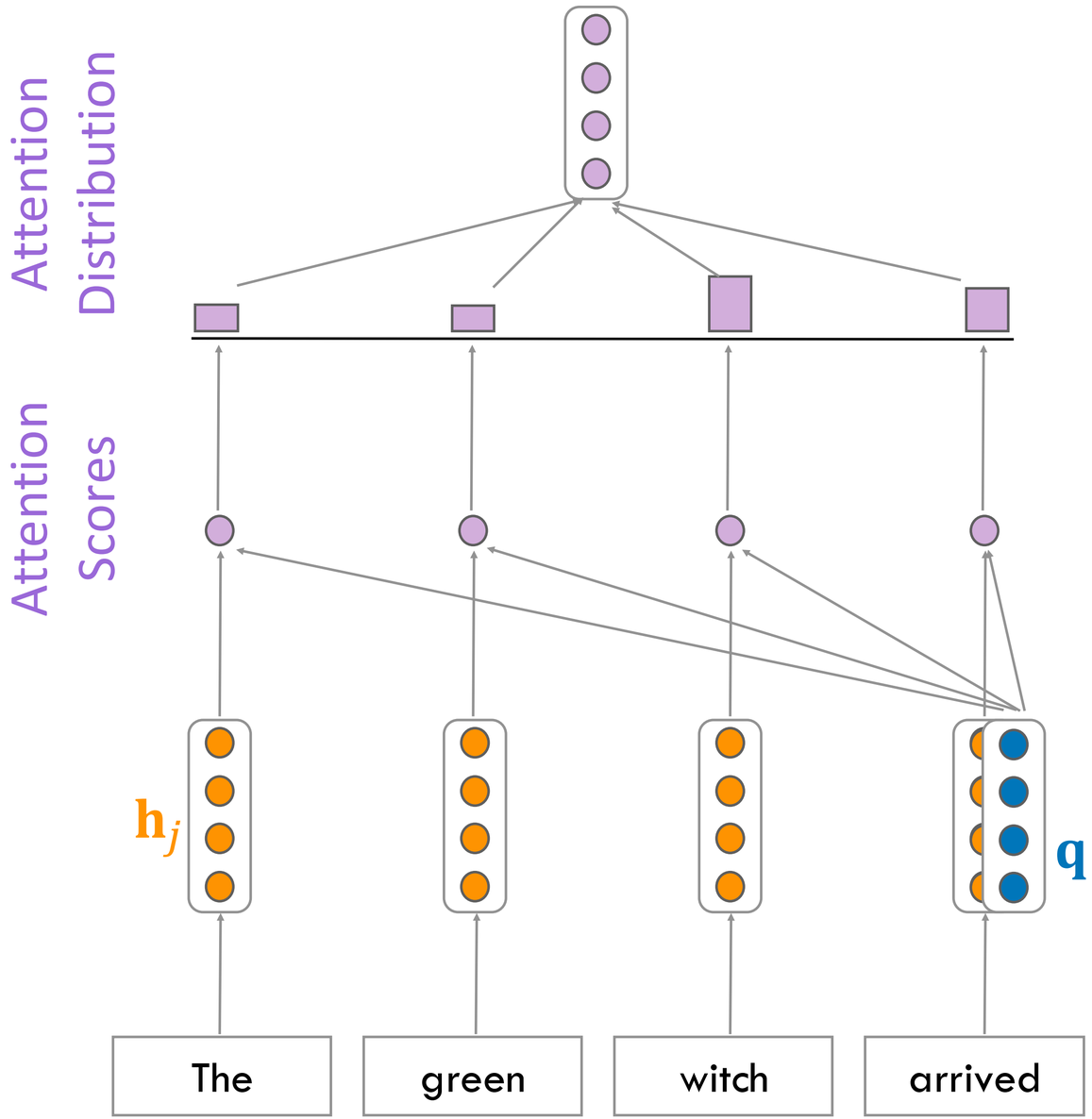
Next, softmax, and compute the weighted average with another matrix multiplication.



Self-Attention: Order Information?

- 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?



Transformers: Positional Embeddings

Missing: Order Information

- Consider representing each sequence index as a vector
  - $\mathbf{p}_i \in \mathbb{R}^d$, for $i \in \{1,2, \ldots, n\}$ are position vectors
- Don't worry about what the $\mathbf{p}_i$ are made of yet!
- Easy to incorporate this info: just add the $\mathbf{p}_i$ to our inputs!
- Recall that $\mathbf{x}_i$ is the embedding of the word at index i. The positioned embedding is:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

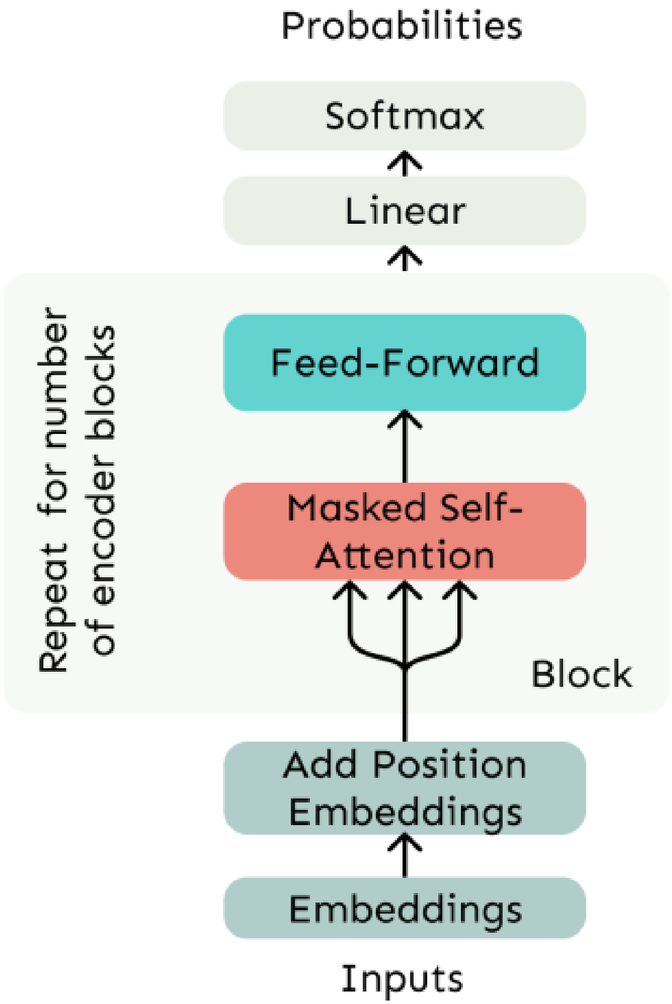
Positional Embeddings

- 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)
- Pros:
    - Flexibility: each position gets to be learned to fit the data
- Cons:
    - Definitely can't extrapolate to indices outside $1, \ldots, n$.
    - There will be plenty of training examples for the initial positions in our inputs and correspondingly fewer at the outer length limits

Putting it all together: Transformer Blocks

Self-Attention Transformer Building Block

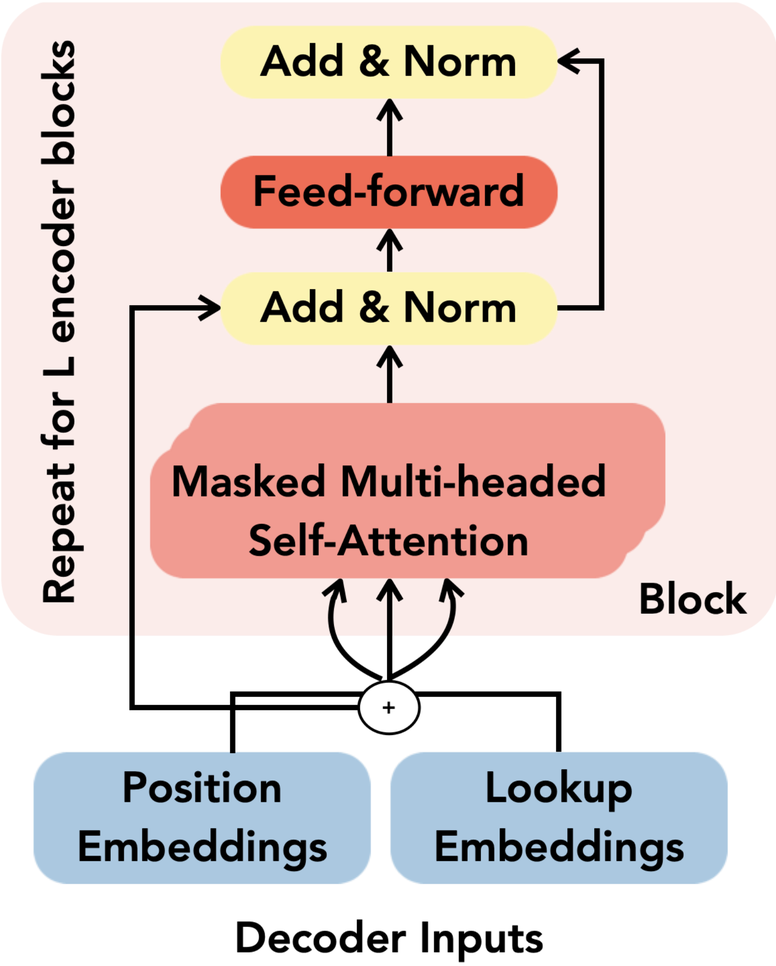
- Self-attention:
  - the basis of the method; with multiple heads
- Position representations:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- Nonlinearities:
  - At the output of the self-attention block
  - Frequently implemented as a simple feedforward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



Transformers as Language Models

The Transformer Model

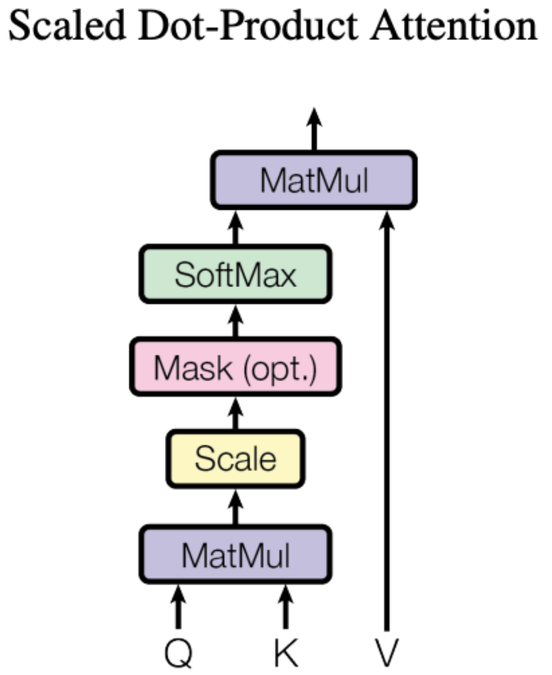
- Transformers are made up of stacks of transformer blocks, each of which is a multilayer network made by combining feedforward networks and self-attention layers, the key innovation of self-attention transformers
- The Transformer Decoder-only model corresponds to
  - a Transformer language model
- Lookup embeddings can be randomly initialized (more common) or taken from existing resources such as word2vec
  - We will look at tokenization (next class)



Transformer Decoder

Scaled Dot Product Attention

$$output_\ell = softmax(XQ_\ell K_\ell^T X^T) * XV_\ell$$



- 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

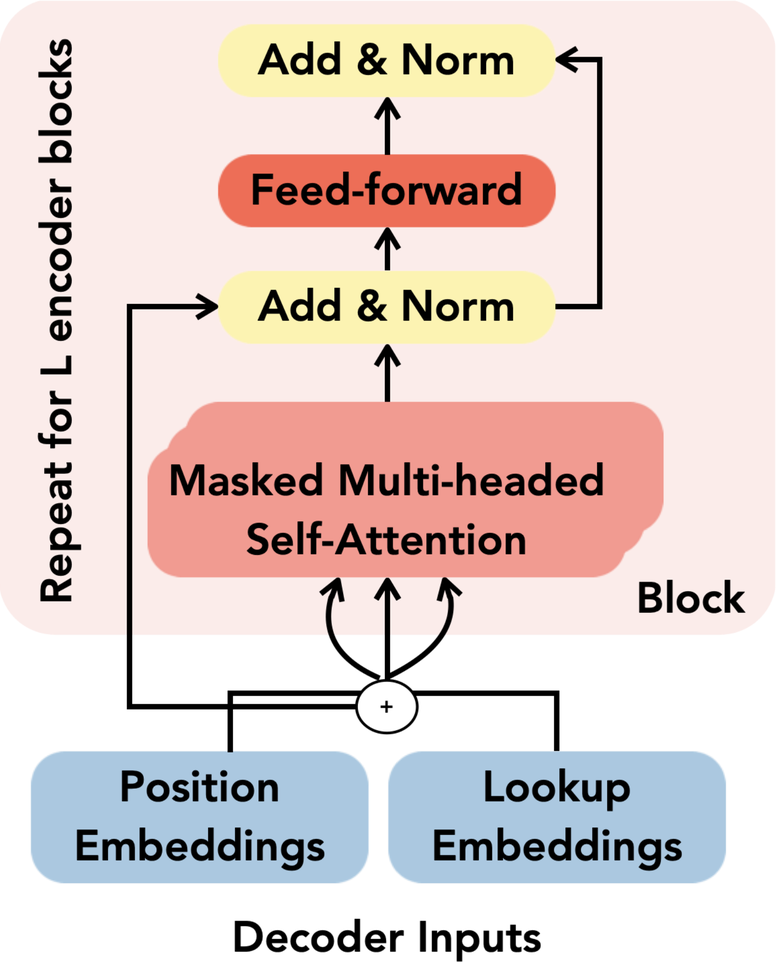
$$output_\ell = softmax(\frac{XQ_\ell K_\ell^T X^T}{\sqrt{d/h}}) * XV_\ell$$

- We divide the attention scores by 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)

The Transformer Decoder

- Two optimization tricks that help training:
  - Residual Connections
  - Layer Normalization
- In most Transformer diagrams, these are often written together as "Add & Norm"
  - Add: Residual Connections
  - Norm: Layer Normalization



Transformer Decoder

Residual Connections

- Original Connections: $X^{(i)} = Layer(X^{(i-1)})$ where i represents the layer
- Residual Connections : trick to help models train better.
  - We let $X^{(i)} = X^{(i-1)} + Layer(X^{(i-1)})$
    - 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.

$$\mu = \frac{1}{d}\sum_{j=1}^{d} x_j; \mu \in \mathbb{R}$$

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

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

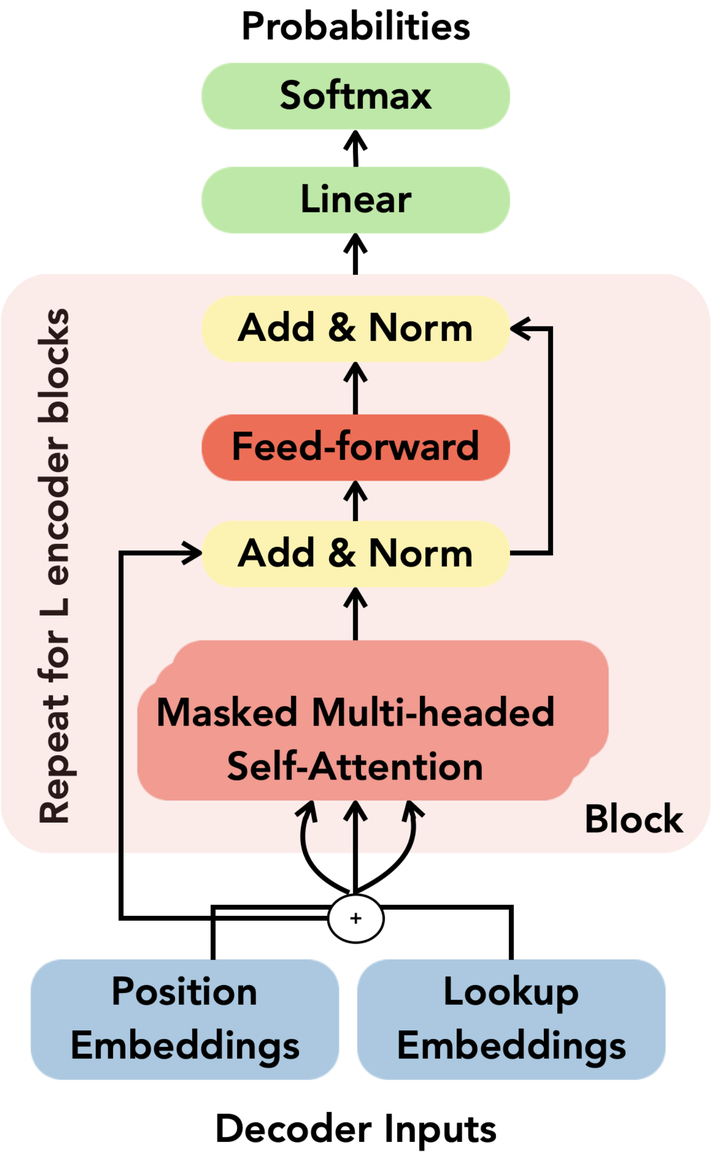
$$\hat{x} = \frac{x - \mu}{\sigma}$$

Component-wise subtraction

Xu et al., 2019

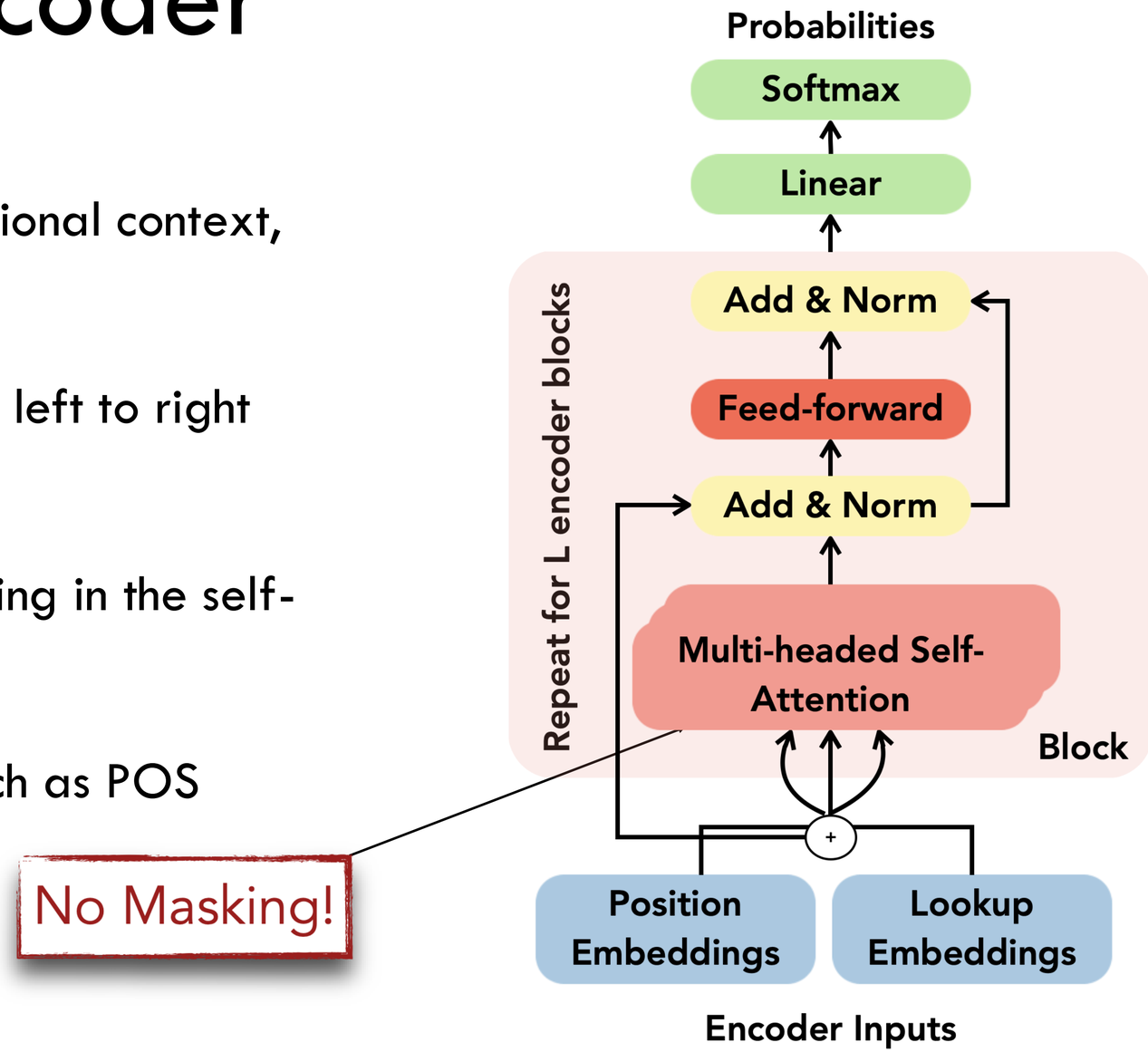
The Transformer Decoder

- 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



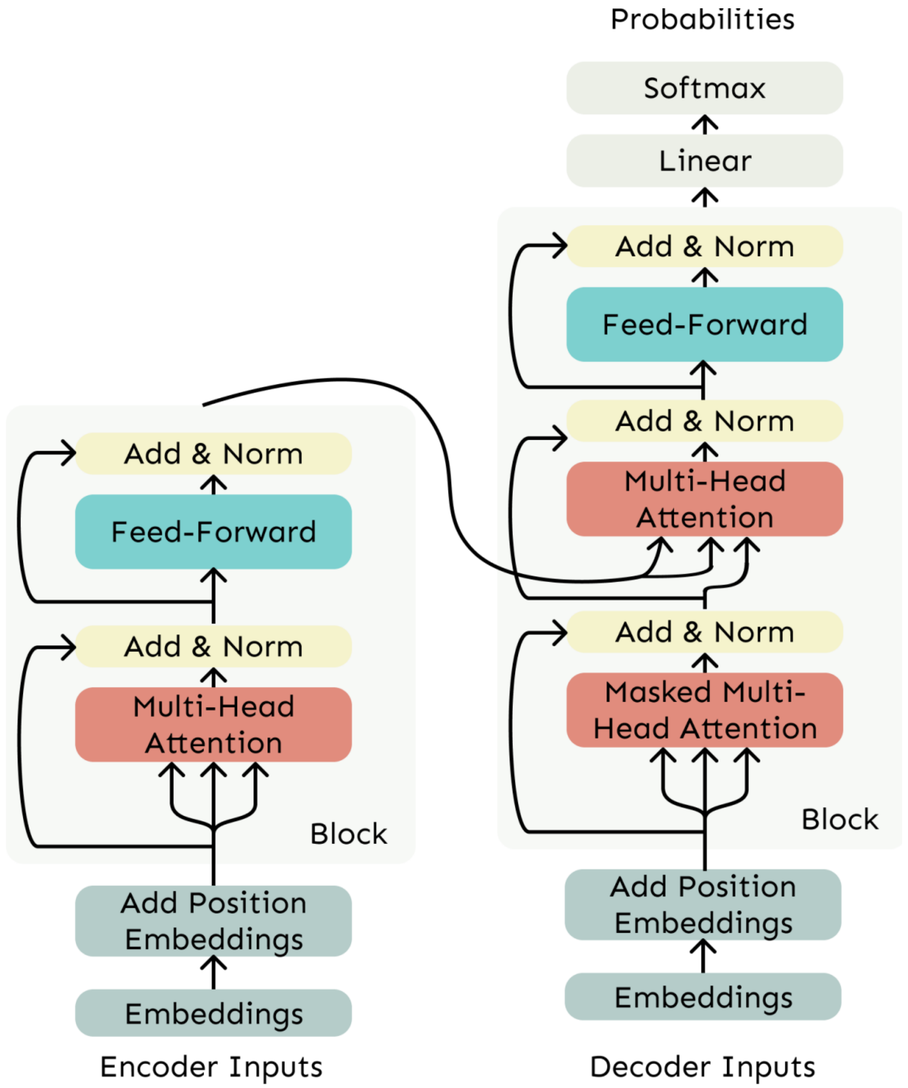
The Transformer Encoder

- 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



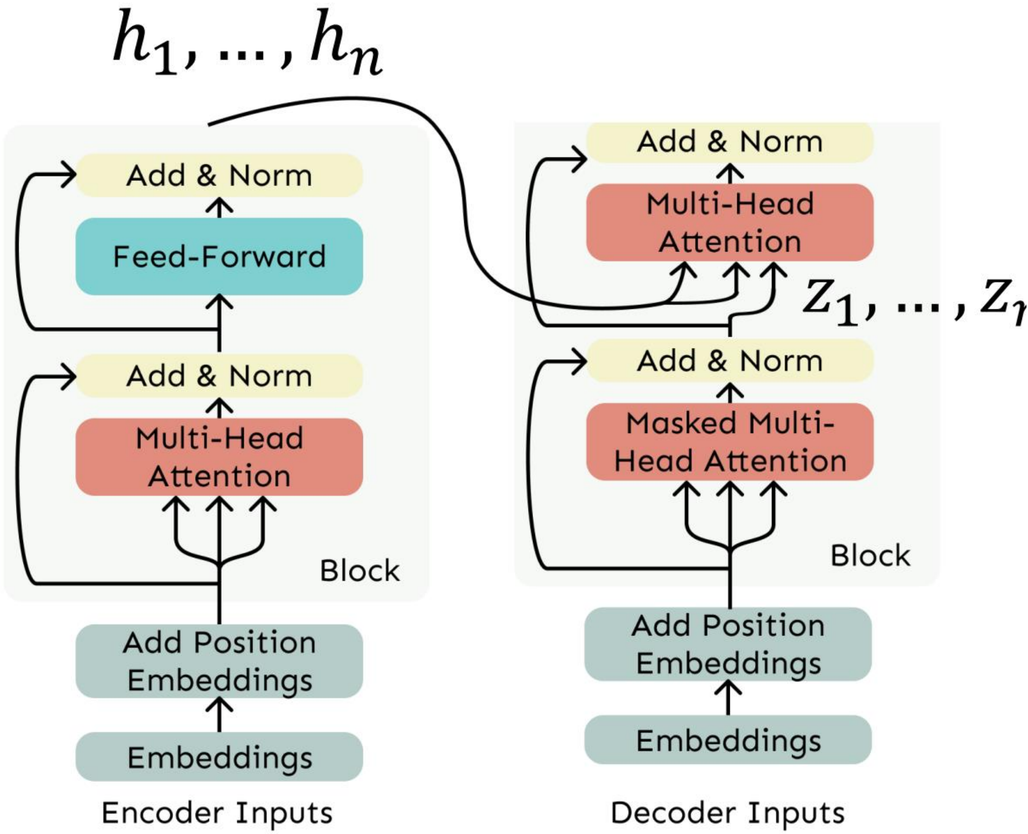
The Transformer Encoder-Decoder

- 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.

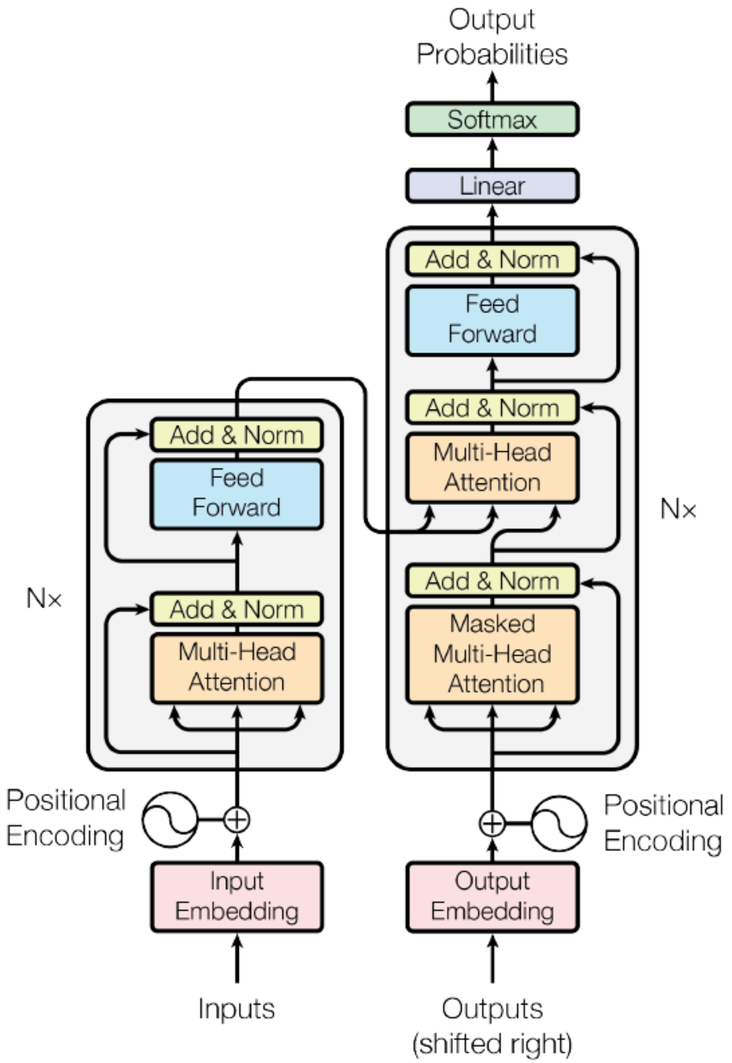


Cross Attention

- 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, $\mathbf{q}_i = \mathbf{Q}\mathbf{z}_i$



Transformer Diagram



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

Transformers: Performance

Machine Translation

Language Modeling

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**	$2.3 \cdot 10^{19}$	

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The real power of Transformers comes from pretraining language models which are then adapted for different tasks

Next Class!

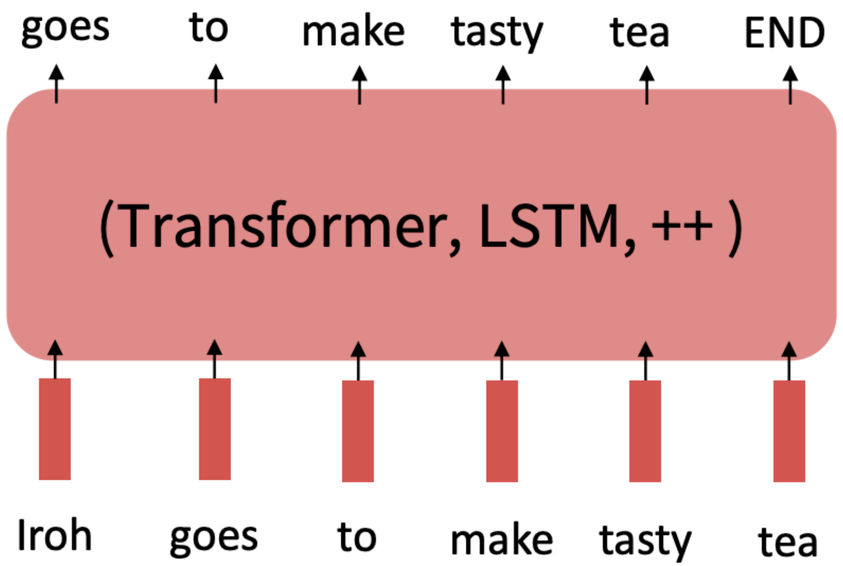
The Pre-training and Fine-tuning Paradigm

The Pretraining / Finetuning Paradigm

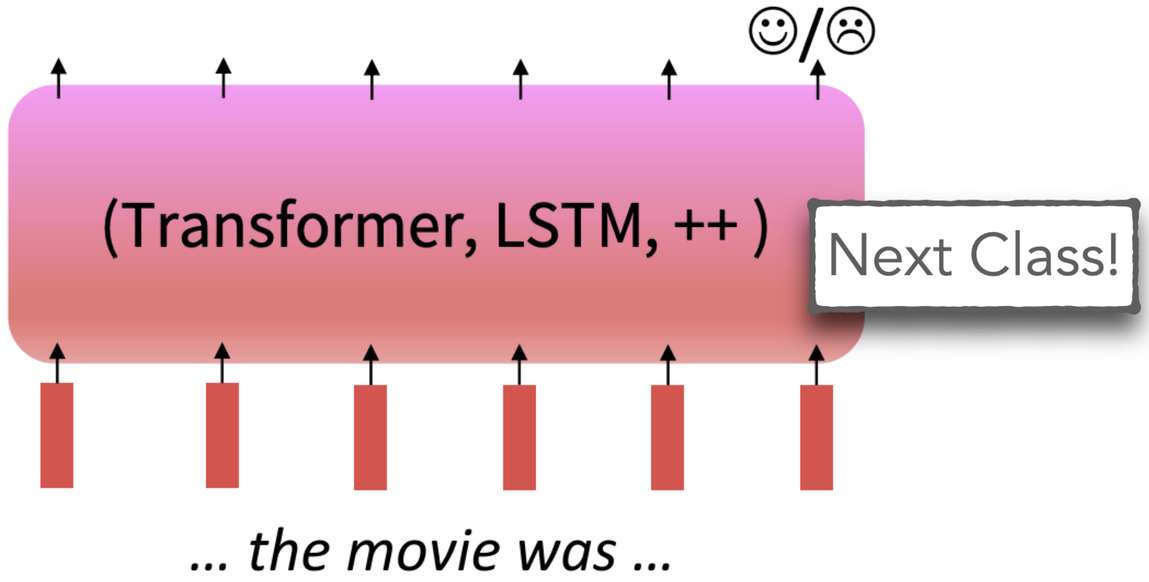
- Pretraining can improve NLP applications by serving as parameter initialization.

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

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



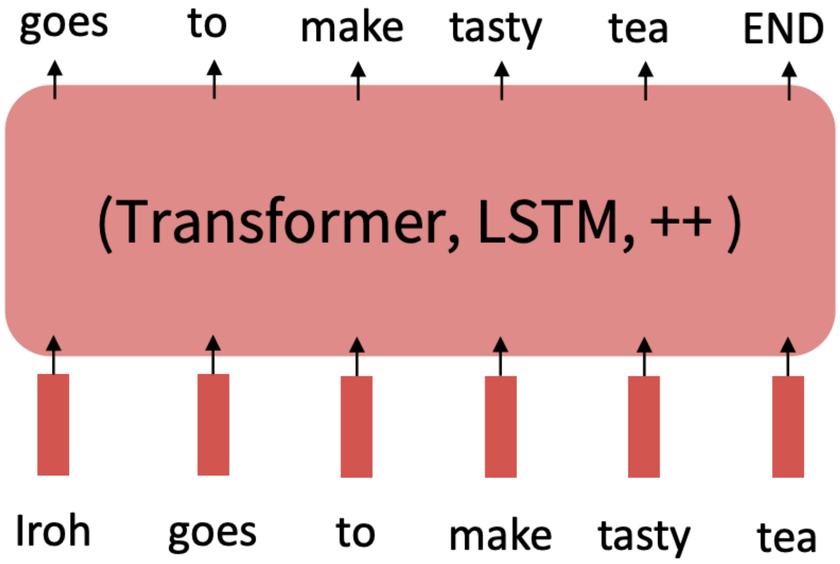
Step 2: Finetune (on your task data)
Not many labels; adapt to the task!



Pretraining

- 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"

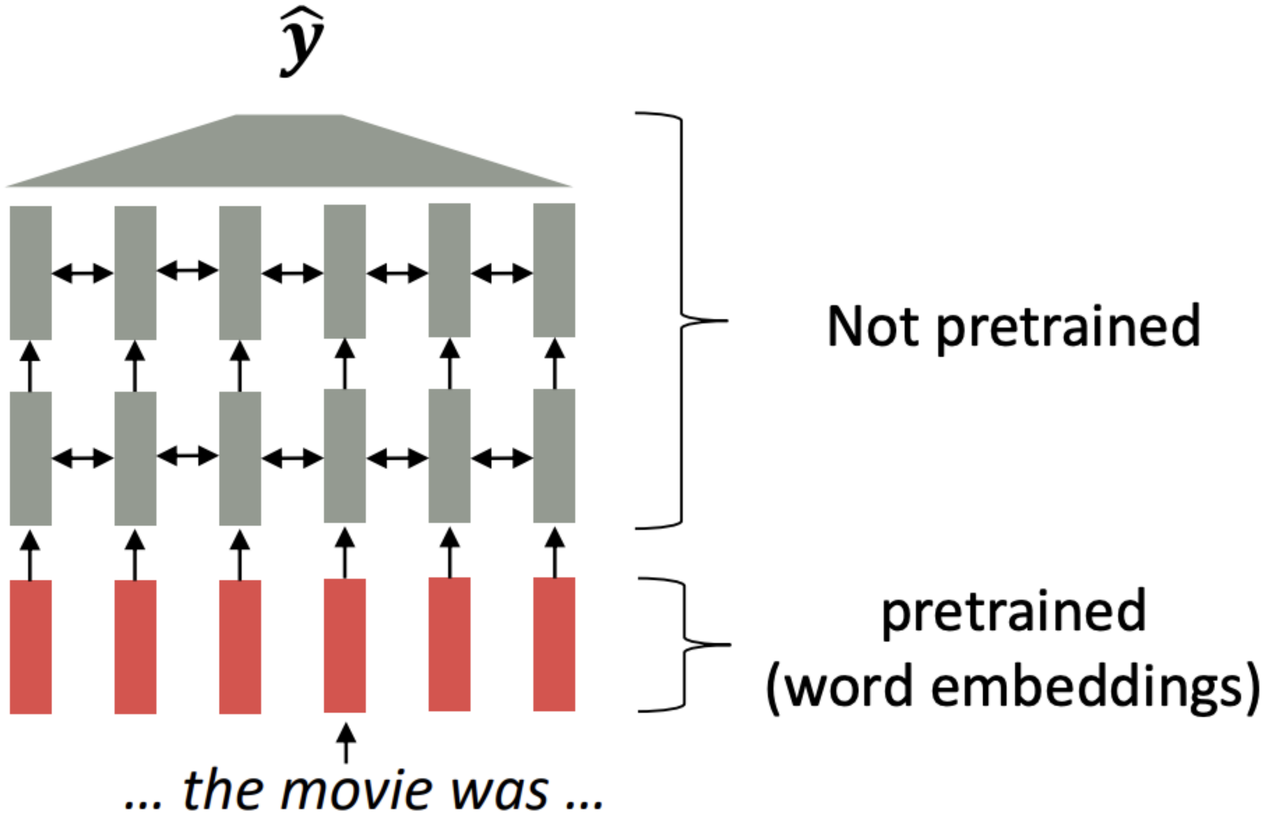
Step 1: Pretrain (on language corpora)
Lots of text; learn general things!



Word embeddings were pretrained too!

Previously:

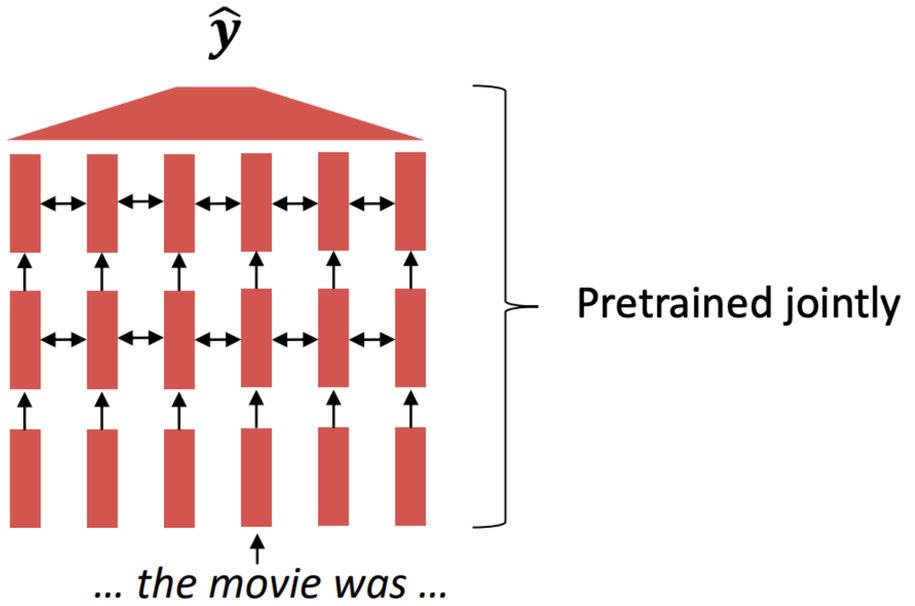
- Start with pretrained word embeddings
  - word2vec
  - GloVe
  - Trained with limited context (windows)
- Learn how to incorporate context in an LSTM or Transformer while training on the task (e.g. sentiment classification)
- Paradigm till 2017



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 $\min_{\theta} \mathcal{L}_{pretrain}(\theta)$
  - $\mathcal{L}_{pretrain}(\theta)$ is the pretraining loss

Pretraining: Intuition from SGD

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

- Pretraining provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{pretrain}(\theta)$
  - $\mathcal{L}_{pretrain}(\theta)$ is the pretraining loss
- Then, finetuning approximates $\min_{\theta} \mathcal{L}_{finetune}(\theta)$,but starting at $\hat{\theta}$
- $\mathcal{L}_{finetune}(\theta)$ is the finetuning loss

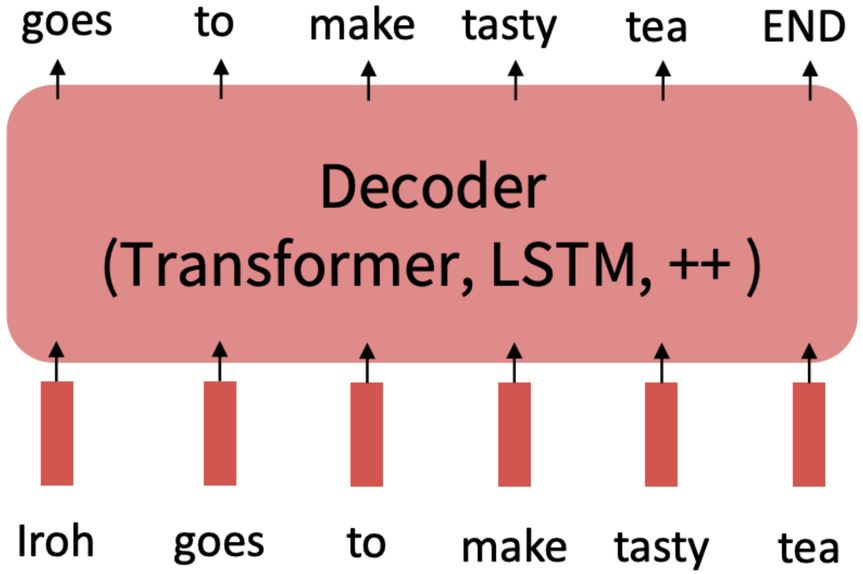
Pretraining: Intuition from SGD

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

- Pretraining provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{pretrain}(\theta)$
  - $\mathcal{L}_{pretrain}(\theta)$ is the pretraining loss
- Then, finetuning approximates $\min_{\theta} \mathcal{L}_{finetune}(\theta)$,but starting at $\hat{\theta}$
- $\mathcal{L}_{finetune}(\theta)$ 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!

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.



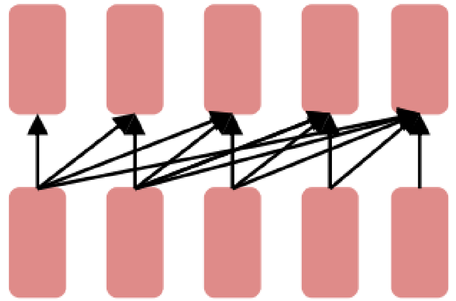
Semi-supervised Sequence Learning

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

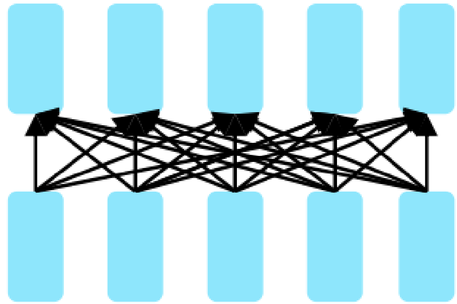
Quoc V. Le
Google Inc.
qvl@google.com

Pretraining

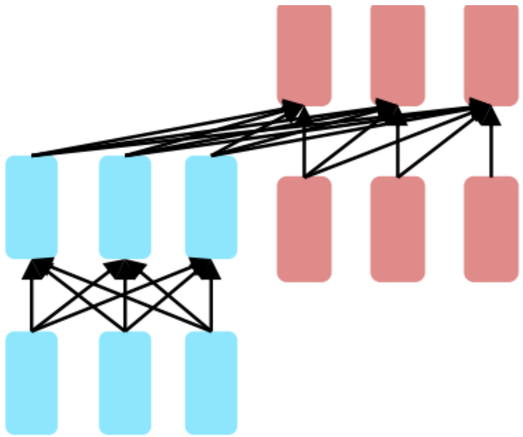
- Not restricted to language modeling!
- Can be any task
- But most successful if the task definition is very general
- Hence, language modeling is a great pretraining option
- Three options!



Decoders

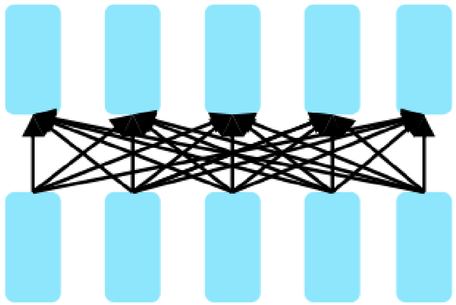


Encoders



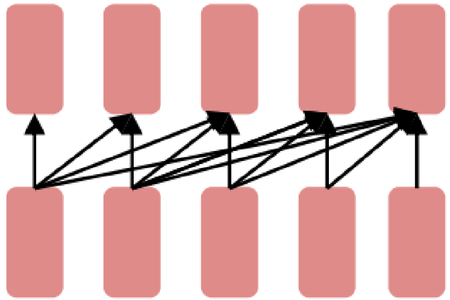
Encoder-Decoders

Pretraining for three types of architectures



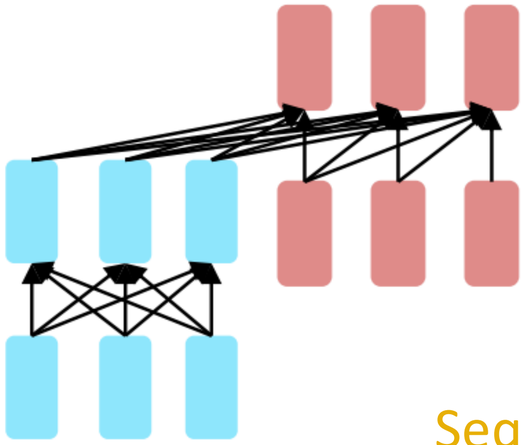
Encoders

Bidirectional Context



Decoders

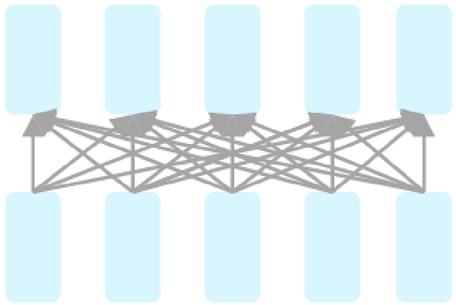
Language Models



Encoder-Decoders

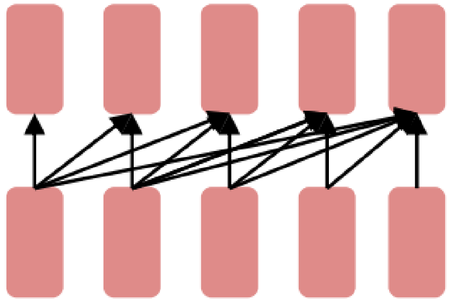
Sequence-to-sequence

Pretraining for three types of architectures



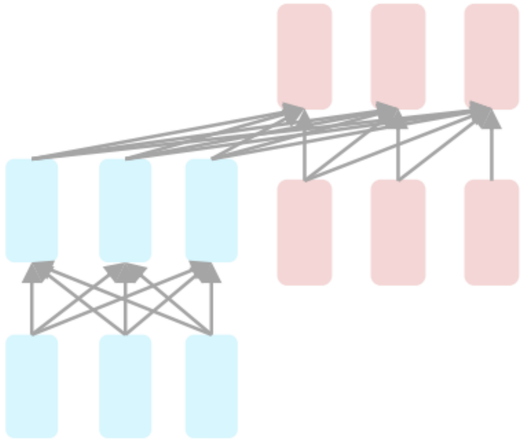
Encoders

Bidirectional Context



Decoders

Language Models

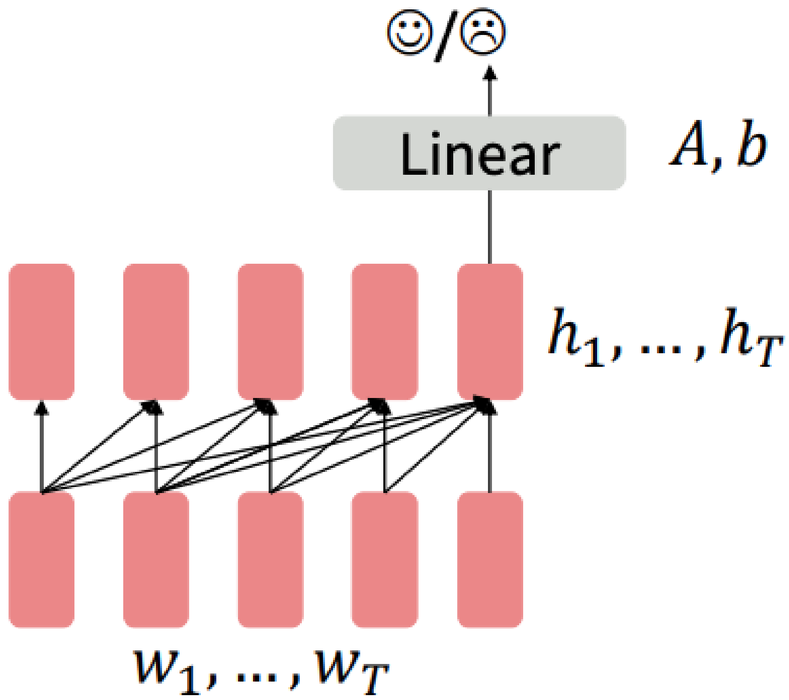


Encoder-Decoders

Sequence-to-sequence

Pretrained Decoders → Classifiers

- When using language model pretrained decoders, we can ignore that they were trained to model $p_\theta(w_t|w_{1:t-1})$
- We can finetune them by training a classifier on the last word's hidden state
  - $h_1, \dots, h_T = Decoder(w_1, \dots, w_T)$
  - $y \approx Ah_T + b$
  - Where A and b are randomly initialized and specified by the downstream task.
- Gradients backpropagate through the whole network.



The linear layer hasn't been pretrained and must be learned from scratch.

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"

OpenAI

[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]

Announcements + Logistics

- Project proposal due today 11:59pm PST!
- Project pitch feedback & grades sent out on Monday.
- Paper selection is finalized. Schedule is out! (also on website)
  - https://docs.google.com/spreadsheets/d/10OVp0QWnyy9chi12cTJUpzZQc6Ah34cWnFKAo0jxo-8/edit?gid=0#gid=0