Language Modeling II

CSE 5525: Foundations of Speech and Language Processing

https://shocheen.github.io/cse-5525-spring-2026/

THE OHIO STATE UNIVERSITY

Sachin Kumar (kumar.1145@osu.edu)

Slide Credits: Chris Tanner

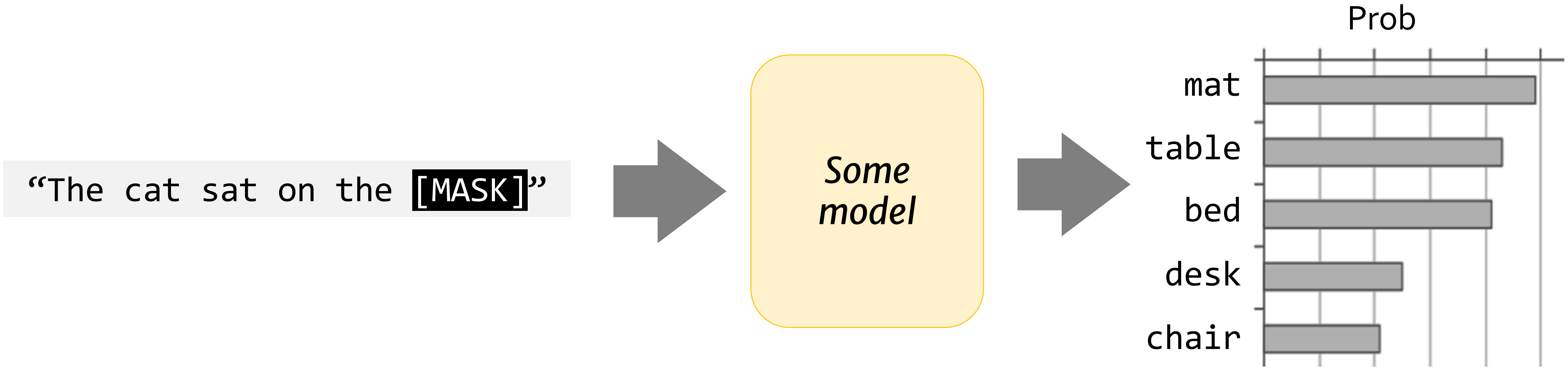
Logistics

- How was Hw1?
  - Any thoughts, questions, concerns?
- Homework 2 is released. Due in two weeks (Feb 11)
  - Topic: Language Modeling with Transformers
- Project information is released on teams/canvas
  - More details will be added for default project in the next two weeks.
  - I will provide a list of sample proposals also in the next few days.

$$\mathbf{P}(\underbrace{\text{mat}}_{\text{next word}} \,|\, \underbrace{\text{The cat sat on the}}_{\text{context or prefix}})$$

$$\mathbf{P}(\underbrace{X_t}_{\text{next word}} | \underbrace{X_1, \ldots, X_{t-1}}_{\text{context}})$$

$$\mathbf{P}(\underbrace{X_t}_{\text{next word}} \mid \underbrace{X_1, \ldots, X_{t-1}}_{\text{context}})$$



But more broadly, we want to model

$$P(X_1, \ldots, X_t)$$

Apply chain rule

$$P(X_1)P(X_2|X_1) \ldots P(X_t|X_1, \ldots, X_{t-1})$$

Doing Things with Language Model

- What is the probability of ....

"I like The Ohio State University"

"like State I University The Ohio State"

Doing Things with Language Model

- What is the probability of ….

“I like The Ohio State University”

“like State I University The Ohio State”

- LMs assign a probability to every sentence (or any string of words).

P(“I like The Ohio State University”) = 10^-5

P(“like State I University The Ohio State”) = 10^-15

Doing Things with Language Model (2)

- We can rank sentences.

- While LMs show "typicality", this may be a proxy indicator to other properties:
  - Grammaticality, fluency, factuality, etc.

P(*"I like The Ohio State University. EOS"*) > **P**(*"I like Ohio State University EOS"*)
P(*"OSU is located in Columbus. EOS"*) > **P**(*"OSU is located in Pittsburgh. EOS"*)

Doing Things with Language Model (3)

- Can also generate strings!

$$\mathbf{P}(\underbrace{X_t}_{\text{next word}} | \underbrace{X_1, \ldots, X_{t-1}}_{\text{context}})$$

- Let's say we start *"Ohio State is "*
- Using this prompt as an initial condition, recursively sample from an LM:

1. Sample from **P**(X | *"Ohio State is "*) → "located"
2. Sample from **P**(X | *"Ohio State is located"*) → "in"
3. Sample from **P**(X | *"Ohio State is located in"*) → "the"
4. Sample from **P**(X | *"Ohio State is located in the"*) → "state"
5. Sample from **P**(X | *"Ohio State is located in the state"*) → "of"
6. Sample from **P**(X | *"Ohio State is located in the state of"*) → "Ohio"
7. Sample from **P**(X | *"Ohio State is located in the state of Ohio"*) → "EOS"

Why Care About Language Modeling?

- Language Modeling is a part of many tasks:
  - Summarization
  - Machine translation
  - Spelling correction
  - Dialogue etc.
  - General purpose Instruction following (ala ChatGPT)

- Language Modeling is an effective proxy for language understanding.
  - Effective ability to predict forthcoming words requires on understanding of context/prefix.

Summary so far

- **Language modeling:** building probabilistic distribution over language.

- An accurate distribution of language enables us to solve many important tasks that involve language communication.

- **The remaining question**: how do you actually estimate this distribution?

Goals & Overview of Today's Lecture

Goal: Understand the language modeling task, which will be used for pretraining and the modern approach to treating many NLP applications as text generation

- Language modeling
- Intrinsic evaluation of language models
- n-gram language modeling
- Smoothing
- Neural language modeling

How good is our language model

- How good is our model?
  - At what?

- We want our model to prefer good sentences over bad ones
  - Higher probability to real or frequent sentences
    - Than ungrammatical or rare ones
    - Without overfitting to a training corpus

  - How does this relate to how we use the language model?

Evaluation

- We must test the model on data it hasn't seen during learning
  - Otherwise — overfitting!

- We need an evaluation metric — two options:
  - Extrinsic: focused on however the model will be used
  - Intrinsic: focused on the language model task — how good can the model assign probabilities to real unseen data?
  - Ideally, the two correlate, but reality is more complex

Perplexity (PPL) – Instrinsic Evaluation

Train the language model on a train corpus, then evaluate on a held-out test set

Using the **likelihood of held-out data**
… actually, using a function of the likelihood

inverse ⇒ higher likelihood means lower perplexity

$$\text{perplexity}(\boxed{w_1 w_2 \ldots w_n}) = \mathbb{P}(w_1 w_2 \ldots w_n)^{-\frac{1}{n}} = \left(\prod_{i=1}^{n} \mathbb{P}(w_i | w_1 \ldots w_{i-1})\right)^{-\frac{1}{n}}$$

held-out test set

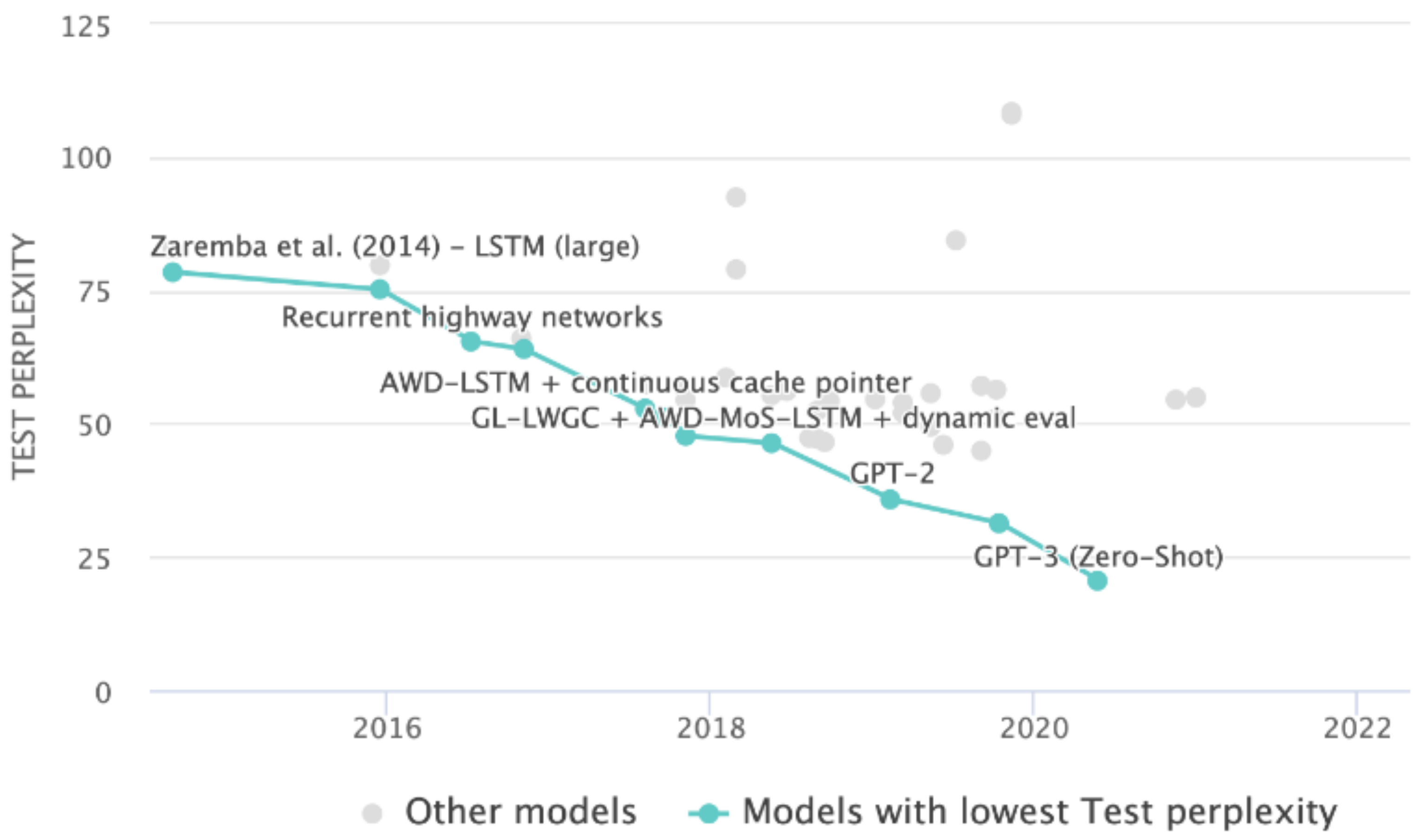
n-th root to normalize by the number of words; the likelihood gets smaller the longer the text (unwanted)

- Inverse comes from the original definition of perplexity in information theory
- Perplexity usually only reported in LM research papers: an (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the downstream tak performance
- Typically ranges from 5–200
- Perplexity of 2 LMs is only comparable if they use identical vocabularies

Perplexity of a Uniform Model

- Assume sentences consisting of random digits
- Assume M sentences with m random digits. Vocabulary size = 10
- What is the perplexity of this data for a model that assigns p=1/10 to each digit

Perplexity of contemporary models



Goals & Overview of Today's Lecture

Goal: Understand the language modeling task, which will be used for pretraining and the modern approach to treating many NLP applications as text generation

- Language modeling
- Intrinsic evaluation of language models
- n-gram language modeling
- Smoothing
- Neural language modeling

Language Models: A History

- Shannon (1950): The predictive difficulty (entropy) of English.

Prediction and Entropy of Printed English

By C. E. SHANNON

(*ManuscriptReceived Sept. 15, 1950*)

A new method of estimating the entropy and redundancy of a language is described. This method exploits the knowledge of the language statistics possessed by those who speak the language, and depends on experimental results in prediction of the next letter when the preceding text is known. Results of experiments in prediction are given, and some properties of an ideal predictor are developed.

Goal: What's the probability of a word w given some history h?

$$\mathrm{P}(X_1)P(X_2|\mathrm{X}_1)\ldots\mathrm{P}(\mathrm{X}_\mathrm{t}|\mathrm{X}_1,\ldots,X_{t-1})$$

$$\mathbb{P}(\text{the}|\text{its water is so transparent that}) = \frac{\text{count}(\text{its water is so transparent that the})}{\text{count}(\text{its water is so transparent that})}$$

Even the Web isn't big enough to give us good estimates in most cases

Simple extensions of the example sentence may have counts zero

N-gram Language Models

- **Terminology:** *n*-gram is a chunk of *n* consecutive words:
  - unigrams: "cat", "mat", "sat", …
  - bigrams: "the cat", "cat sat", "sat on", …
  - trigrams: "the cat sat", "cat sat on", "sat on the", …
  - four-grams: "the cat sat on", "cat sat on the", "sat on the mat", …

https://books.google.com/ngrams/

- *n*-gram language model: $P(X_t | X_1, \ldots, X_{t-1}) \approx P(X_t | X_{t-n+1}, \ldots, X_{t-1})$

$$\mathbf{P}(X_t | X_1, \ldots, X_{t-1})$$

Andrey Markov

Shannon (1950) build an approximate language model with word co-occurrences.

Markov assumptions: every node in a Bayesian network is conditionally independent of its nondescendants, given its parents.

1st order approximation: $\mathbf{P}(\text{mat} \mid \text{the cat sat on the}) \approx \mathbf{P}(\text{mat} \mid \text{the})$

2nd order approximation: $\mathbf{P}(\text{mat} \mid \text{the cat sat on the}) \approx \mathbf{P}(\text{mat} \mid \text{on the})$

$$\mathrm{P}(X_t | X_1, \ldots, X_{t-1}) \approx \mathrm{P}(X_t | \overbrace{X_{t-n+1}, \ldots, X_{t-1}}^{n-1 \text{ elements}})$$

Estimating N-gram probabilities

The probabilities can be computed by **relative frequency** estimation

E.g., for *bigram* model (N=2):

$$\mathbb{P}(w_n|w_{n-1}) = \frac{\text{count}(w_{n-1}w_n)}{\sum_w \text{count}(w_{n-1}w)} = \frac{\text{count}(w_{n-1}w_n)}{\text{count}(w_{n-1})}$$

The general case (with any N):

$$\mathbb{P}(w_n|w_{n-N+1:n-1}) = \frac{\text{count}(w_{n-N+1:n-1}w_n)}{\text{count}(w_{n-N+1:n-1})}$$

Increasing N-gram order & Sparsity

- **Gorillas** always like to groom **their** friends.
    - The likelihood of "their" depends on knowing that "gorillas" is plural
- The **computer** that's on the 3rd floor of our office building **crashed**.
    - The likelihood of "crashed" depends on knowing that the subject is a "computer"

With a low N, the resulting LM would offer probabilities that are too low for these sentences, and too high for sentences that fail basic linguistic tests like number agreement

In these examples we need a 6-gram model, but to estimate the probability of 6-grams, they must occur a sufficient number of times in our corpus

Sparsity: having many cases of "zero probability n-grams" that should really have some non-zero probability

Goals & Overview of Today's Lecture

Goal: Understand the language modeling task, which will be used for pretraining and the modern approach to treating many NLP applications as text generation

- Language modeling
- Intrinsic evaluation of language models
- n-gram language modeling
- Smoothing
- Neural language modeling

Smoothing & Discounting

unknown bigram

$$\mathbb{P}(w_n|w_{n-1}) = \frac{\text{count}(w_{n-1}w_n)}{\sum_w \text{count}(w_{n-1}w)} = \frac{\text{count}(\boxed{w_{n-1}\boxed{w_n}})}{\text{count}(w_{n-1})}$$

known word

➡ Zero probability and hence of the entire sequence

Lidstone smoothing: Add imaginary "pseudo" counts

- α=1 ⇒ **Laplace smoothing**
- α=0.5 ⇒ **Jeffreys-Perks law**
- V is vocabulary
- The probability mass is re-distributed equally

$$\mathbb{P}(w_n|w_{n-1}) = \frac{\text{count}(w_{n-1}w_n) + \alpha}{\text{count}(w_{n-1}) + |V| \cdot \alpha}$$

Smoothing & Discounting

unknown bigram

$$\mathbb{P}(w_n | w_{n-1}) = \frac{\text{count}(w_{n-1} w_n)}{\sum_w \text{count}(w_{n-1} w)} = \frac{\text{count}\boxed{(w_{n-1}\boxed{w_n})}}{\text{count}(w_{n-1})}$$

known word

➡ Zero probability and hence of the entire sequence

Absolute discounting: “shave off” a bit of probability from some more frequent n-grams and give it to the n-grams we’ve never seen

Reference: [Kneser-Ney Smoothing]

N-Gram Models in Practice

- You can build a simple **tri**gram Language Model over a 1.7 million words corpus in a few seconds on your laptop*

`today the ____`

get probability distribution

```
company      0.153
bank    0.153
price   0.077
italian      0.039
emirate      0.039
...
```

Sparsity problem: not much granularity in the probability distribution

Otherwise, seems reasonable!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

N-Gram Models in Practice

- Now we can sample from this mode:

```
today the ____
```

get probability distribution

```
company         0.153
bank    0.153
price   0.077
italian         0.039
emirate         0.039
...
```

* Try for yourself: https://nlpforhackers.io/language-models/ [adopted from Chris Manning]

N-Gram Models in Practice

- Now we can sample from this mode:

condition on this

`today the price ____`

get probability distribution

of	0.308
for	0.050
it	0.046
to	0.046
is	0.031
...	

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

N-Gram Models in Practice

- Now we can sample from this mode:

condition on this

`today the price of _`

get probability distribution

the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
...	

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

N-Gram Models in Practice

- Now we can sample from this mode:

```
today the price of gold per ton , while production of shoe
lasts and shoe industry , the bank intervened just after it
considered and rejected an imf demand to rebuild depleted
european stocks , sept 30 end primary 76 cts a share .
```

Surprisingly grammatical!

But quite incoherent! To improve coherence, one may consider increasing larger than 3-grams, but that would worsen the sparsity problem!

* Try for yourself: https://nlpforhackers.io/language-models/

[adopted from Chris Manning]

N-gram language models in practice

- Probabilistic n-gram models of text generation [Jelinek+ 1980's, ...]
  - Applications: Speech Recognition, Machine Translation

532 PROCEEDINGS OF THE IEEE, VOL. 64, NO. 4, APRIL 1976

Continuous Speech Recognition by Statistical Methods

FREDERICK JELINEK, FELLOW, IEEE

Abstract—Statistical methods useful in automatic recognition of continuous speech are described. They concern modeling of a speaker and of an acoustic processor, extraction of the models' statistical parameters, and hypothesis search procedures and likelihood computations of linguistic decoding. Experimental results are presented that indicate *the power of the methods.*

utterance models used will incorporate more grammatical features, and statistics will have been grafted onto grammatical models. Most methods presented here concern modeling of the speaker's and acoustic processor's performance and should, therefore, be universally useful.

Automatic recognition of continuous (English) speech is an

Goals & Overview of Today's Lecture

Goal: Understand the language modeling task, which will be used for pretraining and the modern approach to treating many NLP applications as text generation

- Language modeling
- Intrinsic evaluation of language models
- n-gram language modeling
- Smoothing
- Neural language modeling

This Class and Beyond: Neural Language Models

- Feedforward Neural Language Model
- Recurrent Neural Network (RNN)
- RNN + Attention
- Attention is all you need
  - Transformer Architecture

$$\mathbf{P}(\underbrace{X_t}_{\text{next word}} | \underbrace{X_1, \dots, X_{t-1}}_{\text{context}})$$

But more broadly,

$$\mathrm{P}(X_1, \dots, X_N)$$

A variant

$$\underbrace{\mathrm{P}(X_1, \dots, X_\mathrm{N} \,|\, \underbrace{Y_1, \dots, Y_M)}_{\text{additional input}}}_{\text{Conditional Language Model}}$$

Language Models: N-grams

- LMs so far: count-based estimates of probabilities
- Counts are brittle and generalize poorly, so we added smoothing
- The quantity that we are focused on estimating (e.g., for tri-gram model):

$$\prod_i P(X_i | X_{i-2}, X_{i-1})$$

Can we use more history by having a neural network predicting the next word?

Neural Language Models

A Very Simple Approach

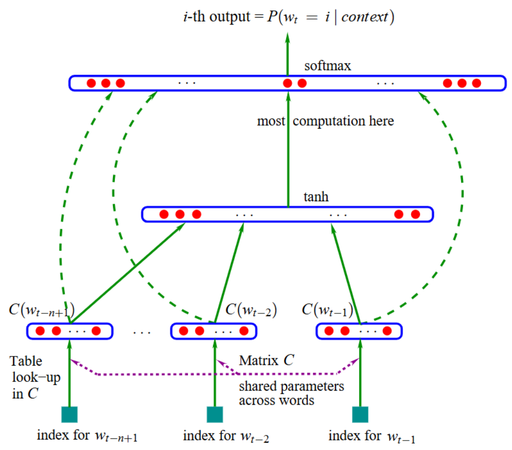
- Instead of having count-based distributions, parameterize them

$$P(X_i | X_{i-2}, X_{i-1}, \theta)$$

- How would we model this with a neural network?
    - Can we use a feedforward network?

Neural Language Models

A Very Simple Approach



- A simple MLP-ish model
  - $\mathbf{c} = [\phi(X_{i-1}); \phi(X_{i-2})]$ <- concatenate the two vectors
  - $l = W_2 \tanh(W_1 \boldsymbol{c} + b_1) + b_2$ (two layers with tanh activation)
  - $P(X_i | X_{i-2}, X_{i-1}, \theta)$ = softmax(l) (number of classes = vocabulary size)

ϕ is an embedding function, and $\theta = (W_1, b_1, W_2, b_2, \phi)$

- The parameters are estimated by maximizing the log probability of the data
- During inference, you compute the neural network every time you need a value from the probability distribution

[Bengio et al. 2003]

Neural Language Models

A Very Simple Approach

- A simple MLP-ish model
  - $\mathbf{x} = [\phi(X_{i-1}); \phi(X_{i-2})]$
  - $y = W_2 \tanh(W_1 \boldsymbol{x} + b_1) + b_2$ (two layers with tanh activation)
  - $P(X_i | X_{i-2}, X_{i-1}, \theta)$ = softmax(y) (number of classes = vocabulary size)

ϕ is an embedding function, and $\theta = (W_1, b_1, W_2, b_2, \phi)$

- What is the advantage over n-gram models?

  Think smoothing

[Bengio et al. 2003]

Neural Language Models

A Very Simple Approach

- A simple MLP-ish model
  - $\mathbf{x} = [\phi(X_{i-1}); \phi(X_{i-2})]$
  - $y = W_2 \tanh(W_1 \boldsymbol{x} + b_1) + b_2$ (two layers with tanh activation)
  - $P(X_i | X_{i-2}, X_{i-1}, \theta)$ = softmax(y) (number of classes = vocabulary size)

ϕ is an embedding function, and $\theta = (W_1, b_1, W_2, b_2, \phi)$

- What is the advantage over n-gram models?
  - Think smoothing
  - $softmax(y)_i = \frac{\exp(y_i)}{\sum_k \exp(y_k)}$
  - Why does softmax help with smoothing?
  - What are the costs?

[Bengio et al. 2003]

Feedforward Neural Language Models

- The MLP approach can help with smoothing at some costs
- But essentially makes the same modeling choices
  - Assuming a finite horizon — the Markov assumption
  - We adopted this assumption because of sparsity (i.e., smoothing) challenges
- Can neural networks allow us to revisit these assumptions?

Neural Language Models

Revisiting the Markov Assumption

- The Markov assumption was critical for generalization
- But: it’s terrible for natural language!
  - “I ate a **strawberry** with some **cream**”
  - “I ate a **strawberry** that was picked in the field by the best farmer in the world with some **cream**”
- It gets even worse beyond the single sentence

Neural Language Models

An MLP with No Markov Assumption

- We need to model the parameterized distribution
    - $P(X_i | X_1, \dots \, X_{i-2}, X_{i-1}, \theta)$
- Why not just treat the context as a bag of words → Deep Averaging Network
    - Then it doesn't matter how long it is

- Why is this a terrible idea?
    - Order matters a lot in language
    - But it worked well for text categorization …
    - What may work for tasks that just require focusing on salient words (e.g., topic categorization), is not sufficient for language models (i.e., <u>**next**</u>-word prediction)

Neural Language Models

Bag of Words

- BOW can handle arbitrary length
- But loses any notion of order
- Furthermore, dependencies are complex
  - Not following linear order
  - Importance follow complex patterns
    - "I ate a strawberry that was picked in the field by the best farmer in the world with some cream"
    - "I ate a strawberry that was picked in the field by the best farmer in the world with clippers"
  - The model needs to focus on different parts in the context to predict different words

[adopted from Chris Manning]

LMs w/ Recurrent Neural Nets

- Core idea: apply a model repeatedly

outputs

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

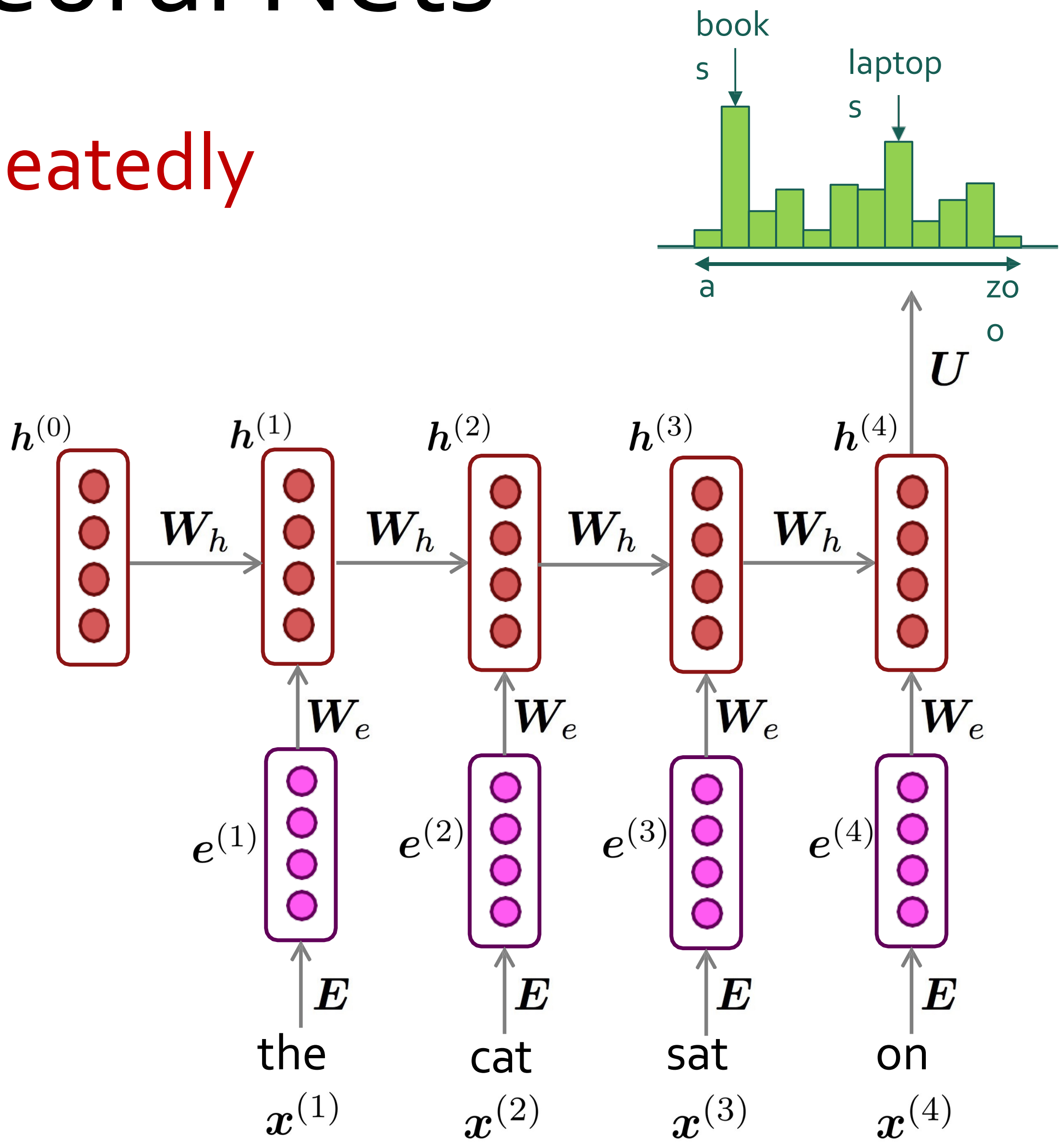
Input embedding

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Recurrent Neural Networks

- Applied to sequential data iteratively.
  - $h_t = f(h_{t-1}, x_t; \theta)$
  - there are many ways to define f (we will only talk about simple RNNs)
  - Note this theta is shared across all the items in the sequence

- Why RNNs
  - They allow modeling infinite context (in theory)
  - They can retain sequential information as opposed to bag of words models

- Intuitively, at every hidden state, the model encodes all the necessary information required to predict the next token at that position
  - At least that's the hope

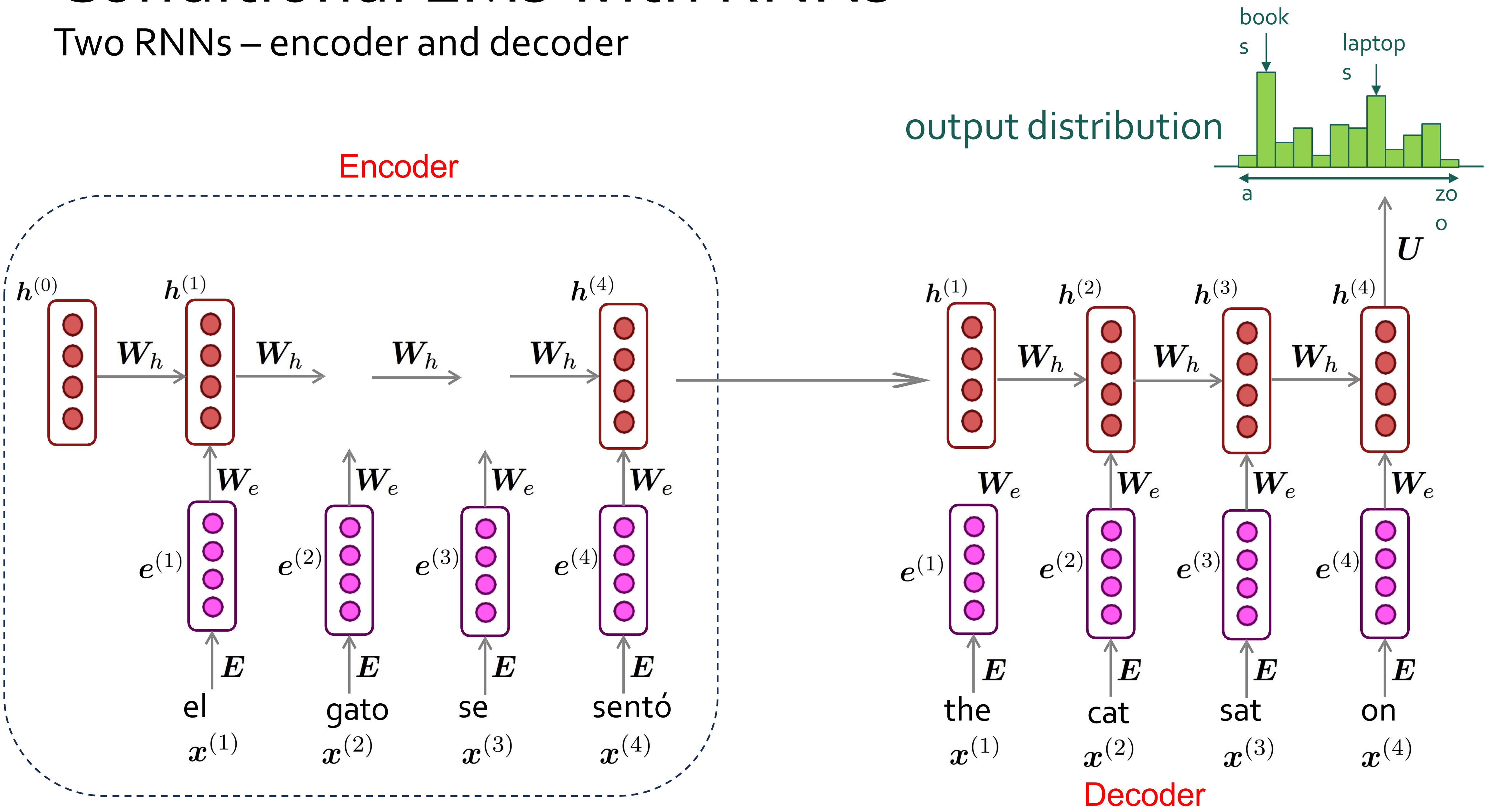
Recall: Conditional Language Models

- Useful for modeling tasks like machine translation, document summarization etc.

$$\mathrm{P}(X_1, \ldots, X_\mathrm{N} \mid Y_1, \ldots, Y_M)$$

Conditional LMs with RNNs

Two RNNs – encoder and decoder



How to train RNNs?

- Using our favorite algorithm: gradient descent.

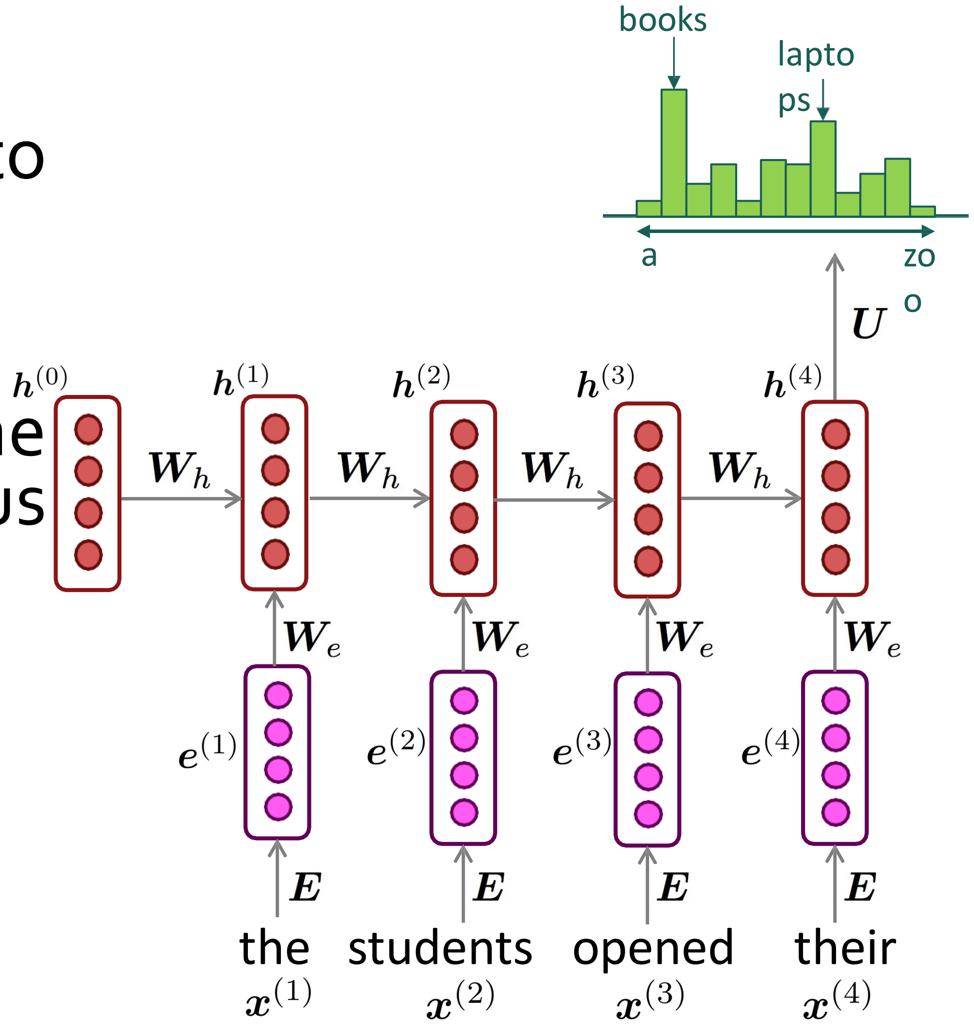
- Loss: classification loss at every step i (using cross-entropy loss)

- But backpropagation is applied over and over to the same parameters theta
  - Also known as backpropagation through time (BPTT)

- Issues with RNNs
  - Gradients can explode or vanish.
  - Solution: modify optimization algorithms / architectures (e.g. LSTMs) [won't discuss in this course, look at readings)

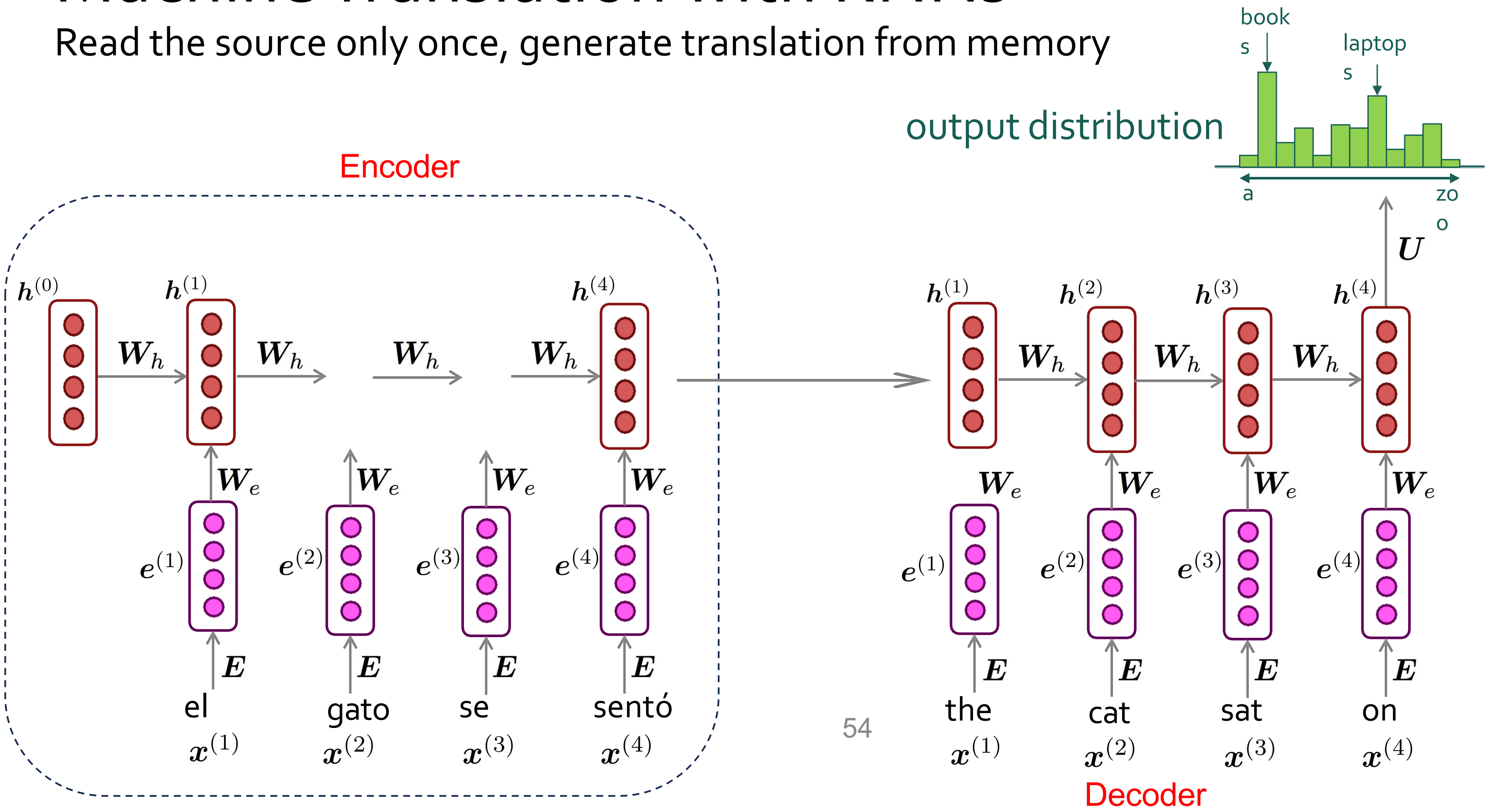
Other issues with RNNs

- Recurrent computation is slow, difficult to parallelize.
- Each hidden state is expected to store the entire information from the previous context
  - Is it even possible?



Machine Translation with RNNs

Read the source only once, generate translation from memory



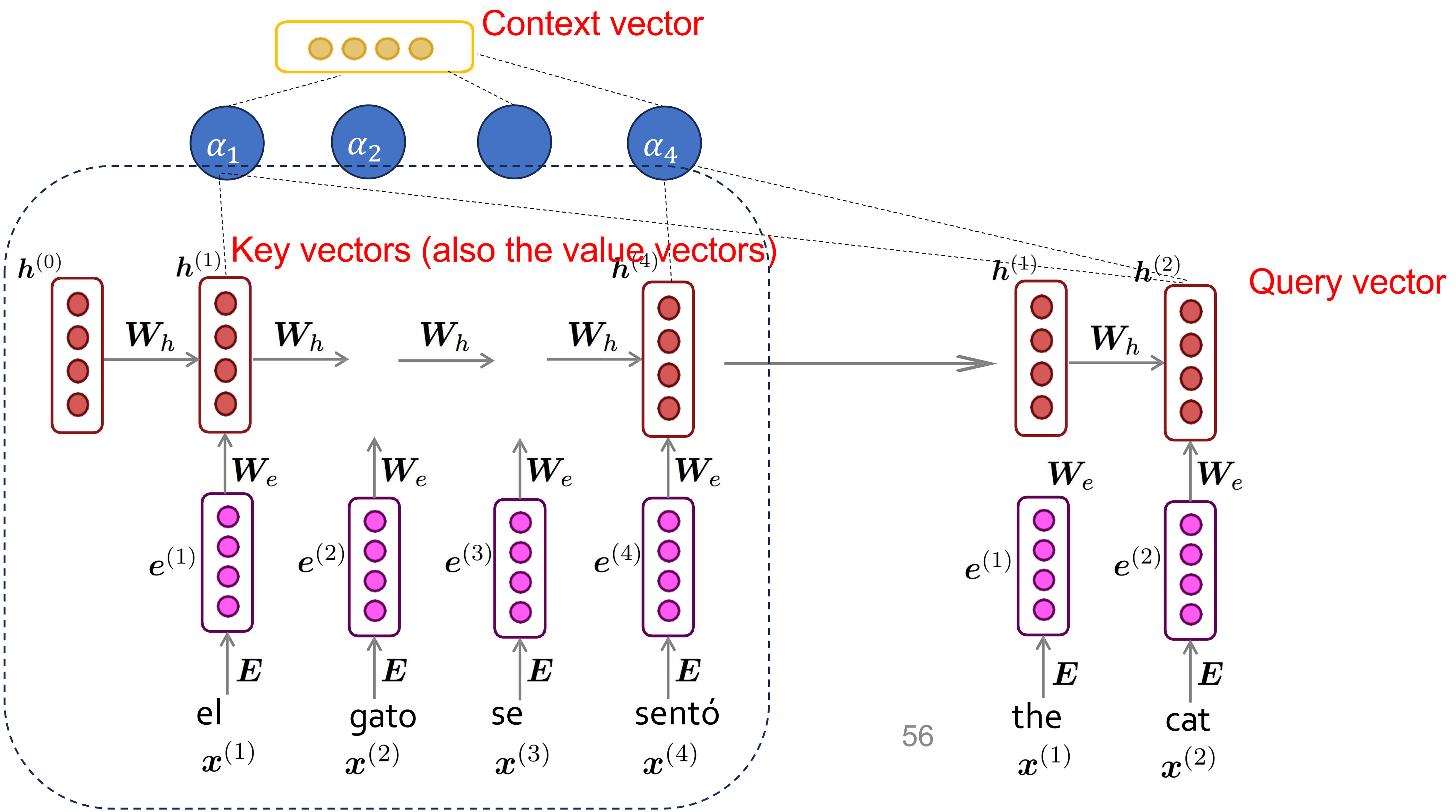
54

Solution: Attention

- What if the decoder at each step pays “attention” to a distribution of all of encoder’s hidden states?

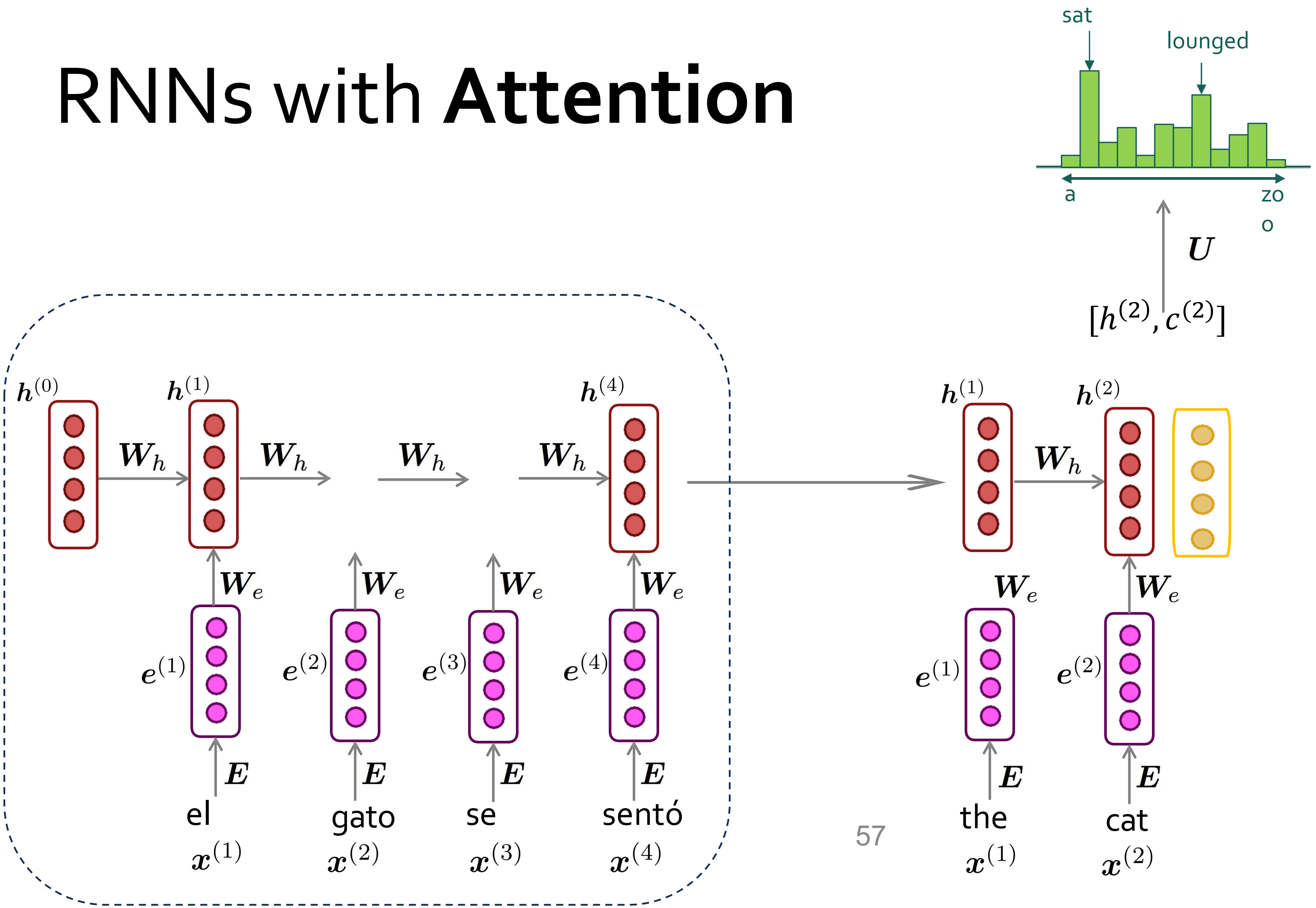
- Intuition: when we (humans) translate a sentence, we don’t just consume the original sentence then regurgitate in a new language; we continuously look back at the original while focusing on different parts

RNNs with **Attention**



56

RNNs with **Attention**



57

RNNs with Attention

- Attention allowed modelling longer context and obtain higher performance

- But
  - It is still slow because of linear computation in RNN
  - It still has gradient vanishing/exploding issues

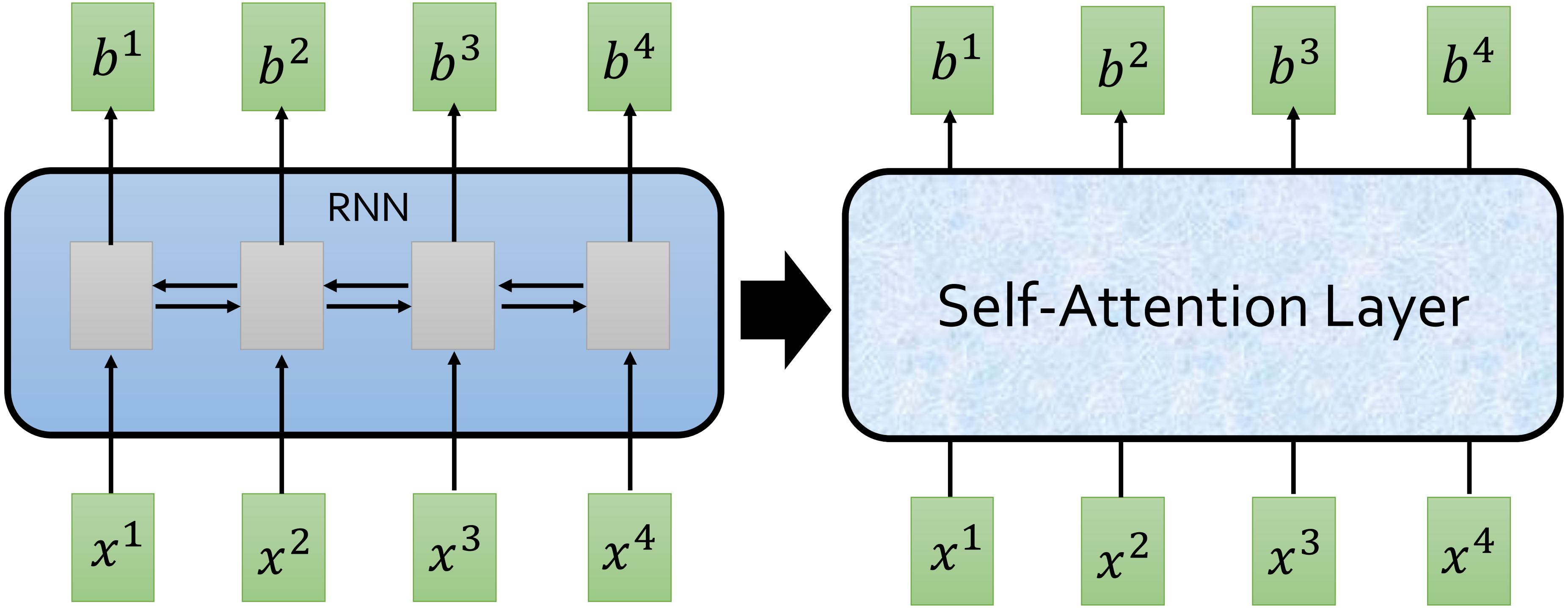
- Solution: what if we removed the RNN component and only use attention
  - Attention is all you need (Vaswani et al 2017)

Transformers

- Replace the linear part with **self-attention**
- Introduce **residual connections** to improve gradient flow (avoid gradient exploding / vanishing issues)
- Introduce **positional embeddings** to encode sequential order

Self-Attention

- b^t is obtained based on the whole input sequence.
- can be parallelly computed.



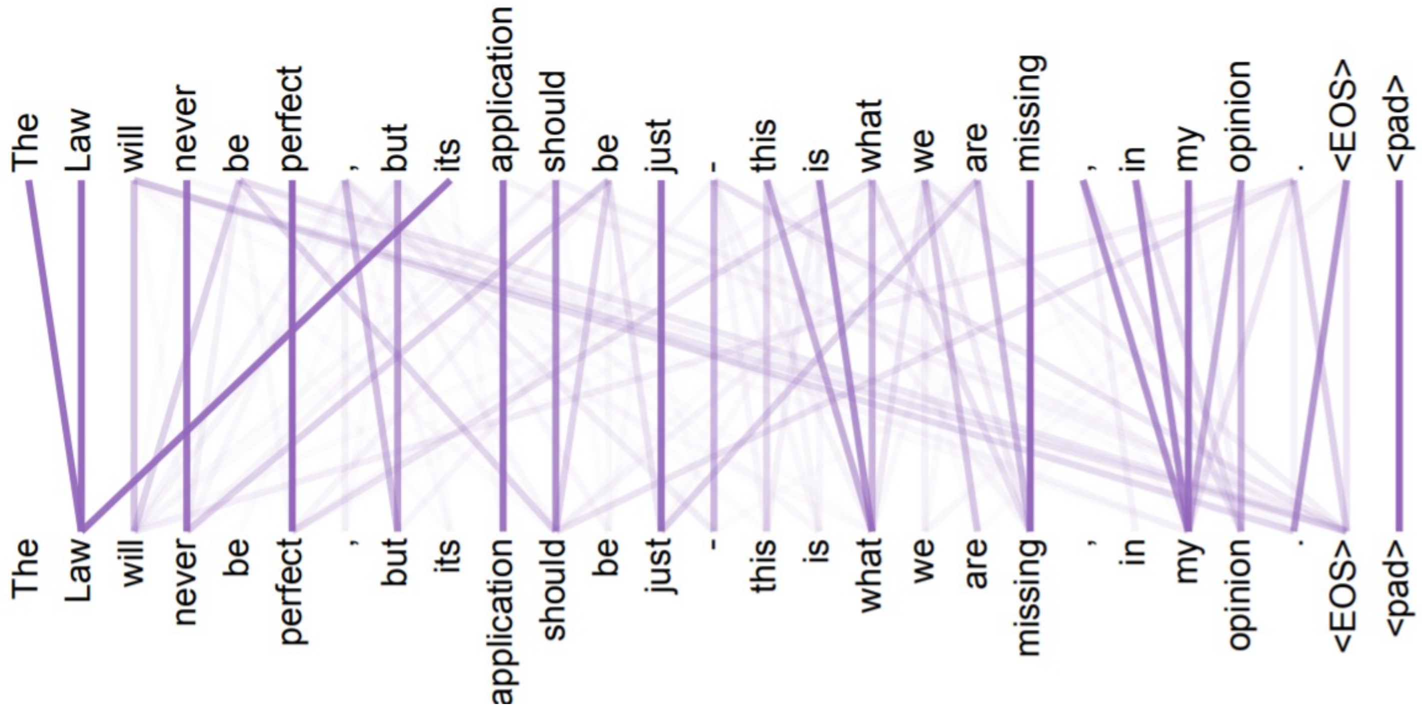
Idea: replace any thing done by RNN with self-attention.

"Neural machine translation by jointly learning to align and translate" Bahdanau etl. 2014;
"Attention is All You Need" Vaswani et al. 2017

[adopted from Hung-yi Lee]

Attention

- Core idea: on each step, *use direct connection* to *focus ("attend") on a particular part* of the context
  - Kind of similar to deep averaging networks but a "weighted average"



[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

Defining Self-Attention

- **Terminology:**
  - **Query**: to match others
  - **Key**: to be matched
  - **Value**: information to be extracted

- **Definition:** Given a set of vector **keys**, and a vector **query**, *attention* is a technique to compute a weighted sum of the **value**, dependent on the **query**.

[Vaswani et al. 2017: https://arxiv.org/abs/1706.03762]

q: query (to match others)

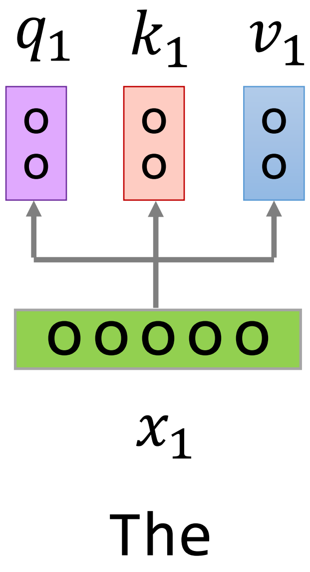
$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

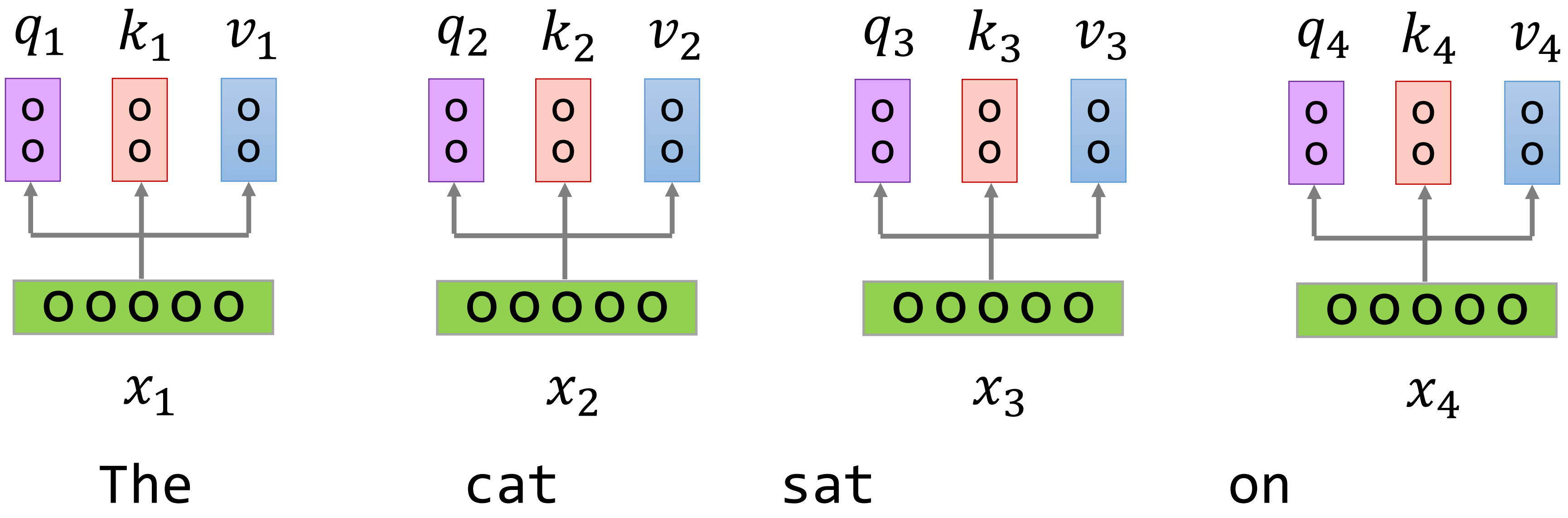
$$v_t = W^v x_t$$



q: query (to match others)
$$q_t = W^q x_t$$

k: key (to be matched)
$$k_t = W^k x_t$$

v: value (information to be extracted)
$$v_t = W^v x_t$$



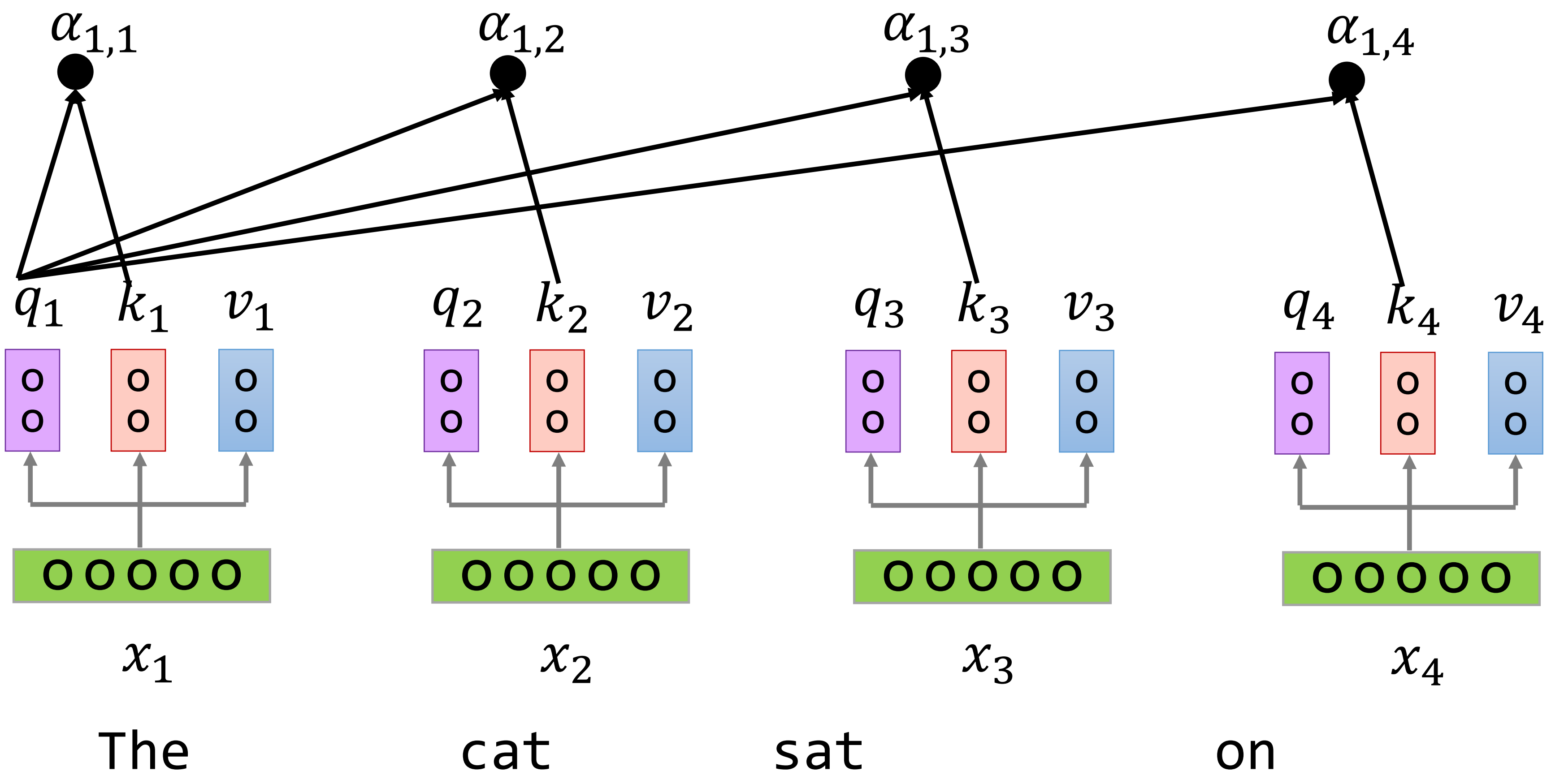
$$\alpha_{1,t} = \underbrace{q^1 \cdot k^t / \alpha}_{\text{Scaled dot product}}$$

q: query (to match others)

k: key (to be matched)

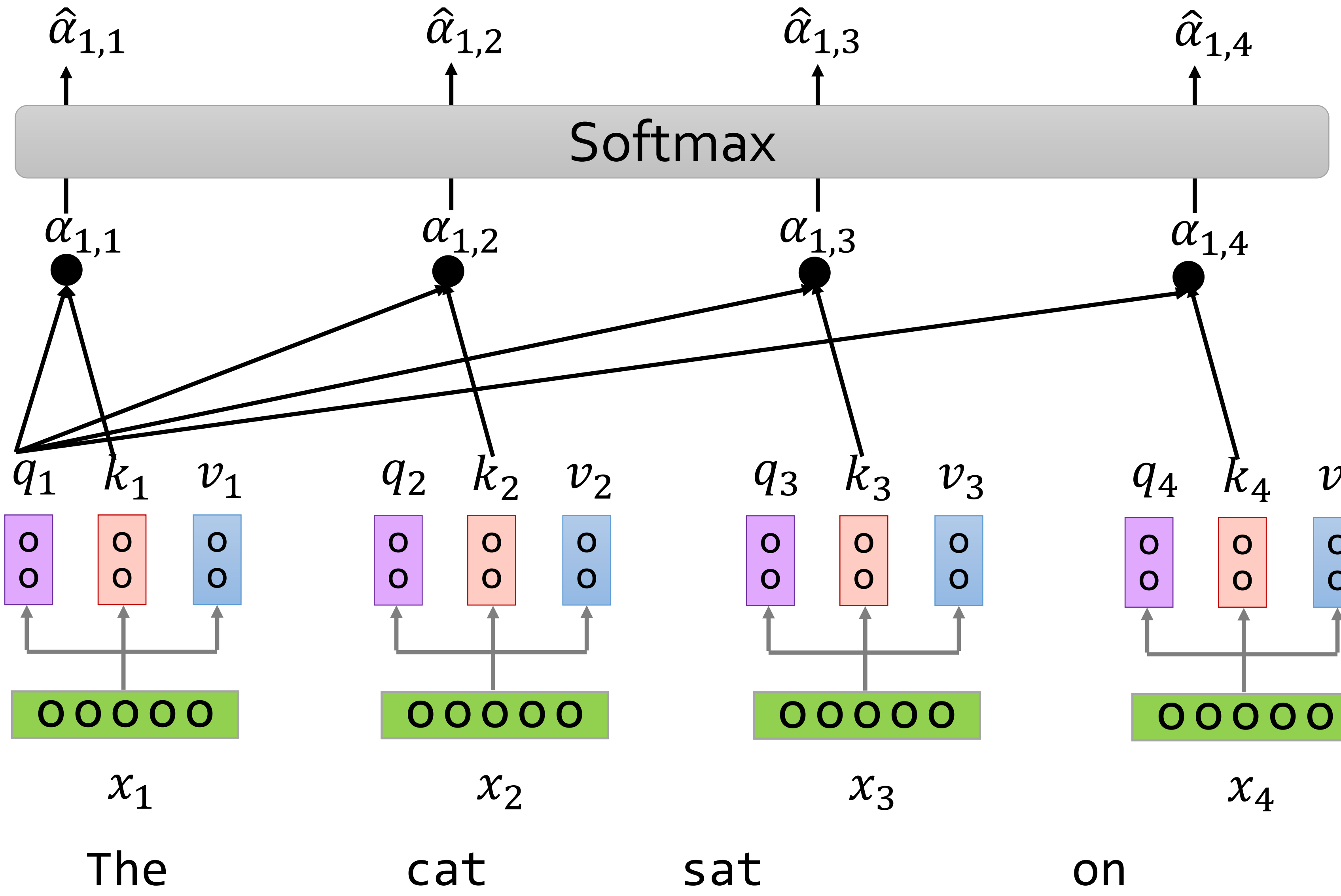
v: value (information to be extracted)

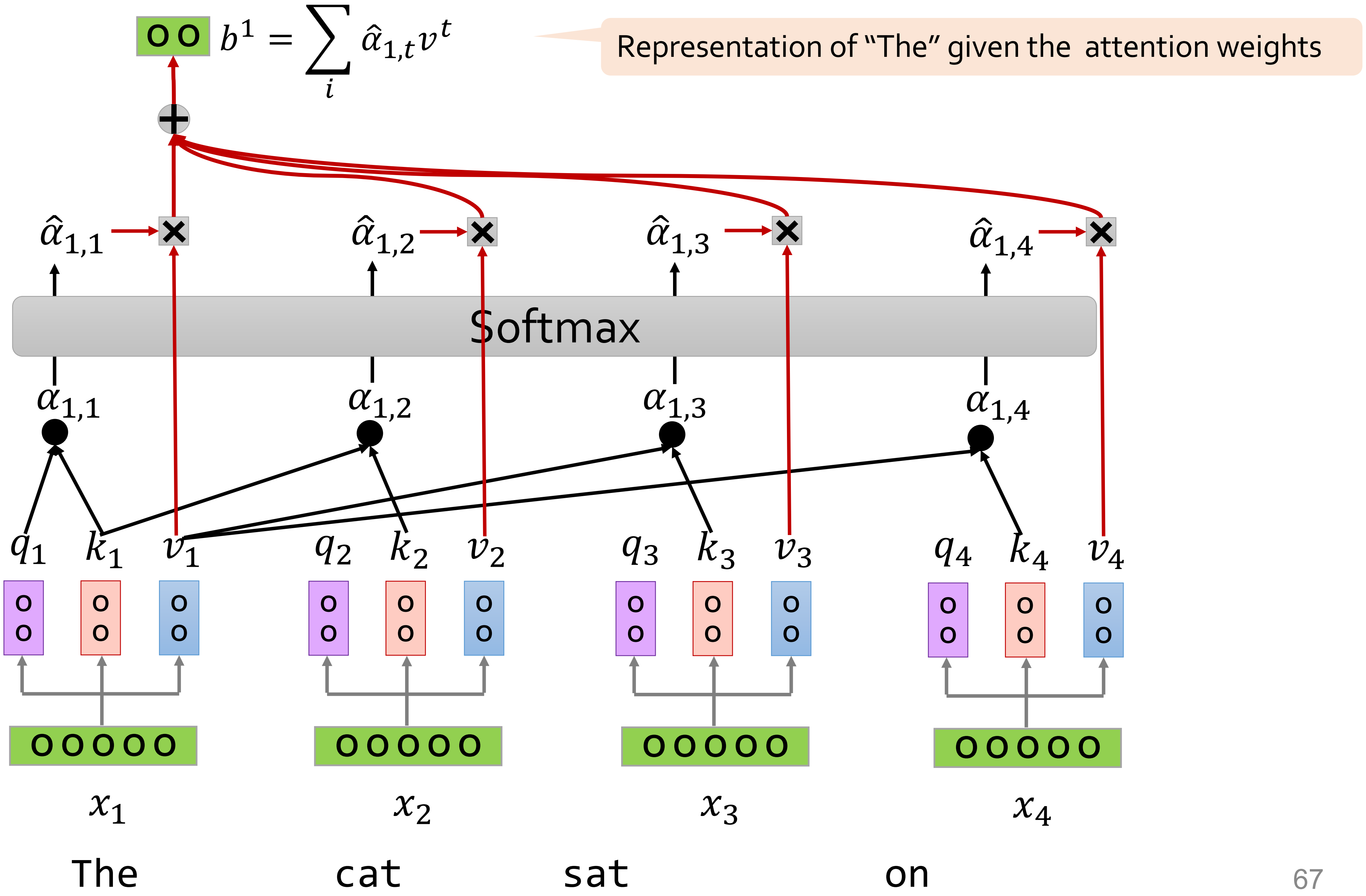
How much should "The" attend to other positions?



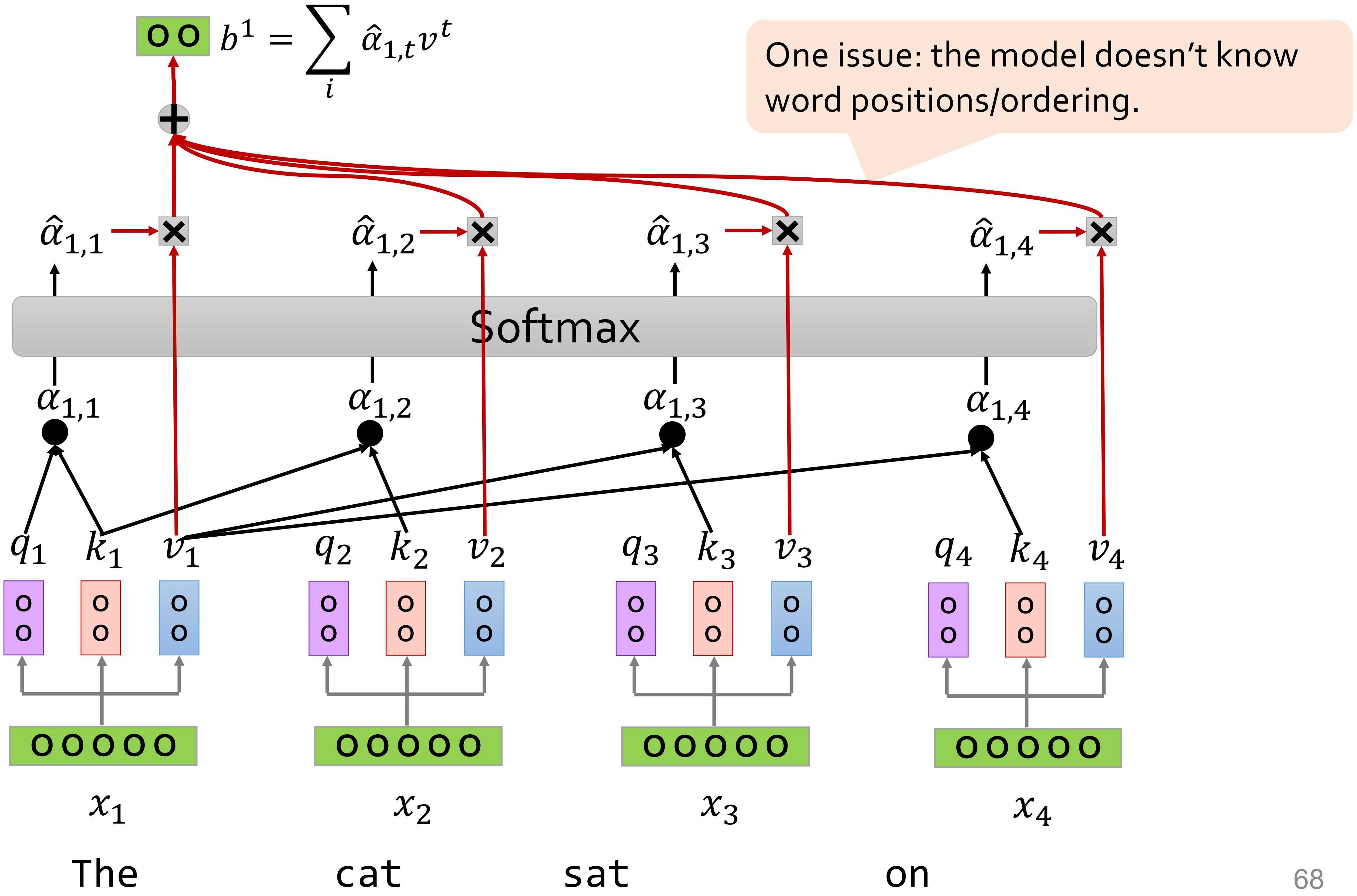
$$\sigma(z)_t = \frac{exp(z_t)}{\sum_j exp(z_j)}$$

How much should "The" attend to other positions?





$b^1 = \sum_i \hat{\alpha}_{1,t} v^t$
Representation of "The" given the attention weights
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
x_1
x_2
x_3
x_4
The
cat
sat
on



$b^1 = \sum_i \hat{\alpha}_{1,t} v^t$
One issue: the model doesn't know word positions/ordering.
$\hat{\alpha}_{1,1}$
$\hat{\alpha}_{1,2}$
$\hat{\alpha}_{1,3}$
$\hat{\alpha}_{1,4}$
Softmax
$\alpha_{1,1}$
$\alpha_{1,2}$
$\alpha_{1,3}$
$\alpha_{1,4}$
q_1 k_1 v_1
q_2 k_2 v_2
q_3 k_3 v_3
q_4 k_4 v_4
x_1
x_2
x_3
x_4
The
cat
sat
on

How to encode position information?

- Self attention doesn't have a way to know whether an input token comes before or after another
  - Position is important in sequence modeling in NLP

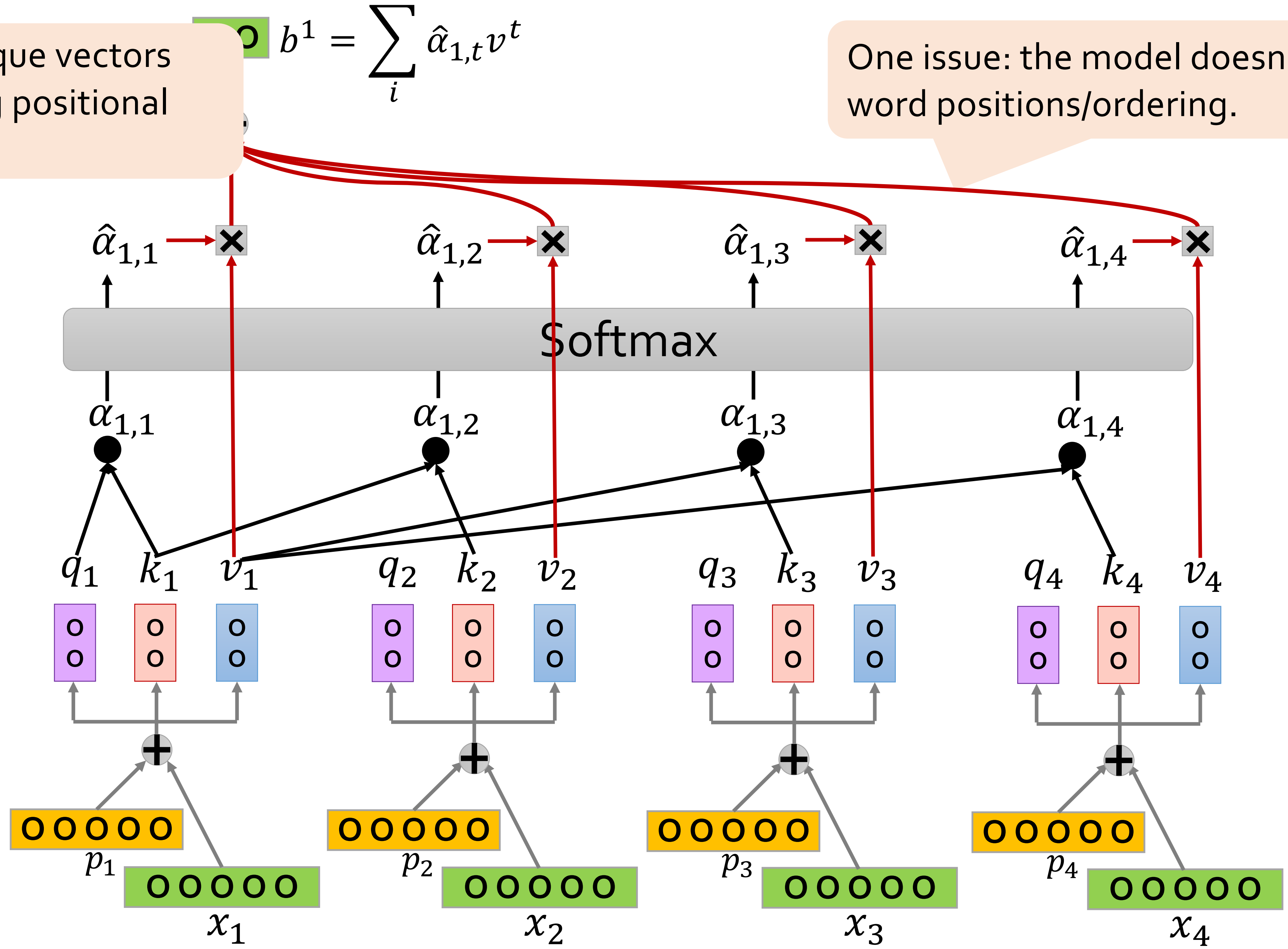
- A way to introduce position information is add individual position encodings to the input for each position in the sequence

$$x_t = x_t + pos_t$$

Where pos_{t} is a position vector

pos_i are unique vectors representing positional information

One issue: the model doesn't know word positions/ordering.



Properties of a good positional embedding

- It should output a unique encoding for each time-step (word’s position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
    - The cat sat on the mat
    - The happy cat sat on the mat
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.

Absolute position embeddings

- Define a maximum context length you model can encode: say 1000 tokens.
    - Create a separate embedding table for each position.
    - Each index 1, 2, 3, … gets an embedding.
    - Learn the embeddings with the model.

- Issues with Learned positions embeddings:
  - Maximum length that can be presented is limited (what if I get a 2000 token input)
  - Difficult to encode relative positions
    - The cat sat on the mat
    - The happy cat sat on the mat

Functional (and fixed) position embeddings

Sinusoidal embeddings

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

where

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

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1.\, t) \\ \cos(\omega_1.\, t) \\ \\ \sin(\omega_2.\, t) \\ \cos(\omega_2.\, t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2}.\, t) \\ \cos(\omega_{d/2}.\, t) \end{bmatrix}$$

The frequencies are decreasing along the vector dimension. It forms a geometric sequence on the wavelengths.

Sinusoidal Embeddings: Intuition

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

[Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog]

Variants of Positional Embeddings

- Rotary Positional Embeddings (RoPE): [2104.09864] RoFormer: Enhanced Transformer with Rotary Position Embedding (arxiv.org)

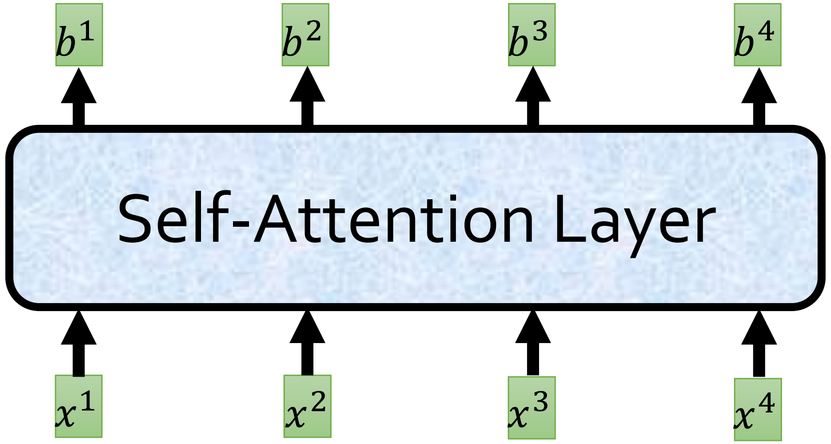
- AliBi: [2108.12409] Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (arxiv.org)

- No embeddings(!?): [2203.16634] Transformer Language Models without Positional Encodings Still Learn Positional Information (arxiv.org)

Self-Attention: Back to Big Picture

- Attention is a way to focus on particular parts of the input
- Can write it in matrix form:

$$\boldsymbol{b} = \mathrm{softmax}\left(\frac{QK^{\mathrm{T}}}{\alpha}\right)V$$



- Efficient implementations
- Better at maintaining long-distance dependencies in the context.

Self-Attention

$$\boldsymbol{b} = \mathrm{softmax}\left(\frac{QK^{\mathrm{T}}}{\alpha}\right)V$$

hardmaru
@hardmaru

The most important formula in deep learning after 2018

Self-Attention

What is self-attention? Self-attention calculates a weighted average of feature representations with the weight proportional to a similarity score between pairs of representations. Formally, an input sequence of n tokens of dimensions d, $X \in \mathbf{R}^{n \times d}$, is projected using three matrices $W_Q \in \mathbf{R}^{d \times d_q}$, $W_K \in \mathbf{R}^{d \times d_k}$, and $W_V \in \mathbf{R}^{d \times d_v}$ to extract feature representations Q, K, and V, referred to as query, key, and value respectively with $d_k = d_q$. The outputs Q, K, V are computed as

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V. \tag{1}$$

So, self-attention can be written as,

$$S = D(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_q}}\right)V, \tag{2}$$

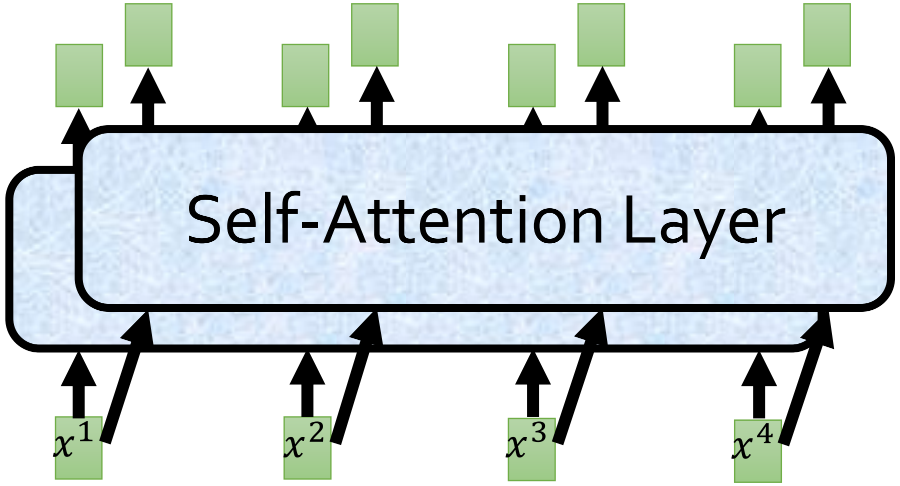
where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

9:08 PM · Feb 9, 2021 · Twitter Web App

553 Retweets **42** Quote Tweets **3,338** Likes

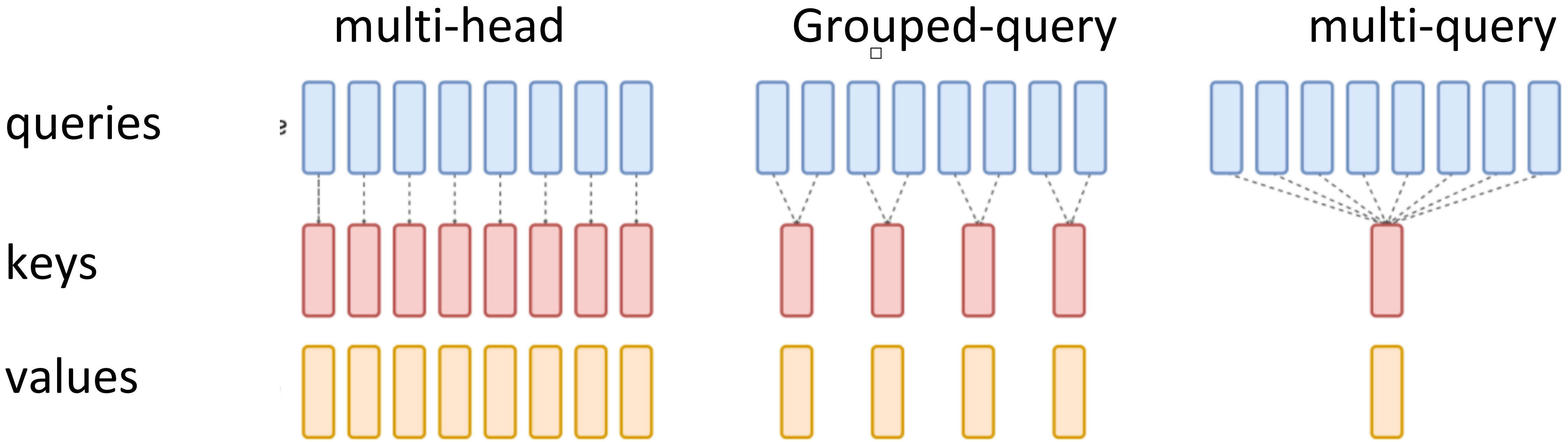
Multi-Headed Self-Attention

- Multiple parallel attention layers is quite common.
  - Each attention layer has its own parameters.



[Vaswani et al. 2017]

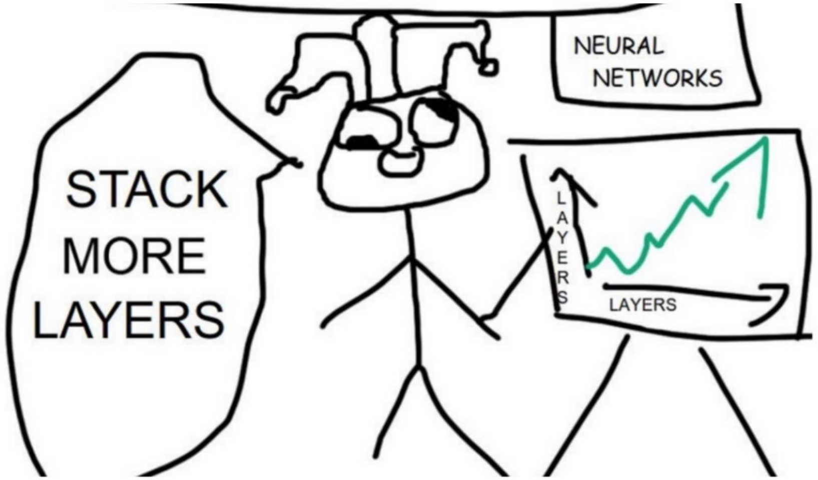
Variants of attention

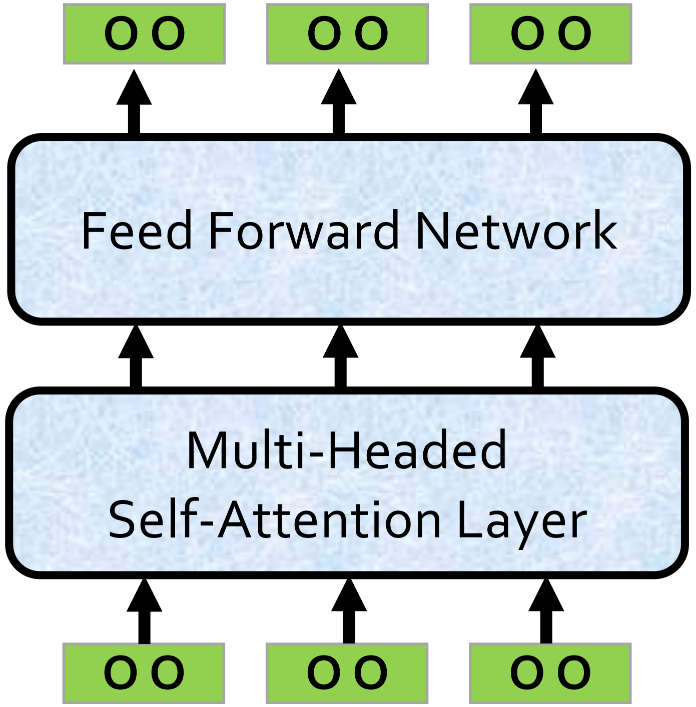


GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints (Ainslie et al., 2023)

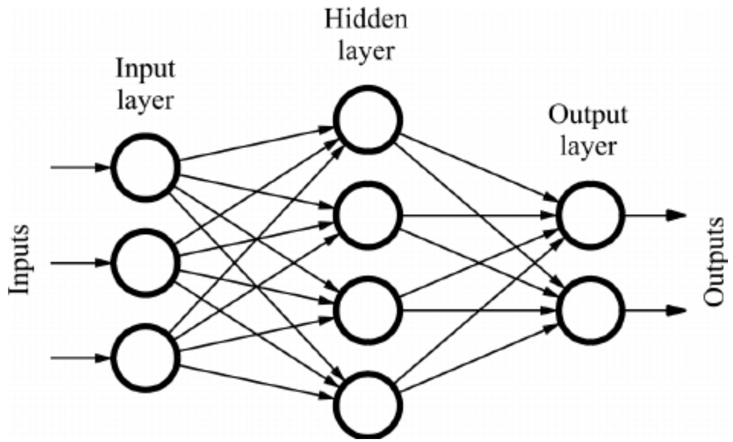
How Do We Make it Deep?

- Add a feed-forward network on top it to add more capacity/expressivity.
- Repeat!





Feedforward Net: Refresher



A fully-connected network of nodes and weights.

Feed forward layer in a transformer

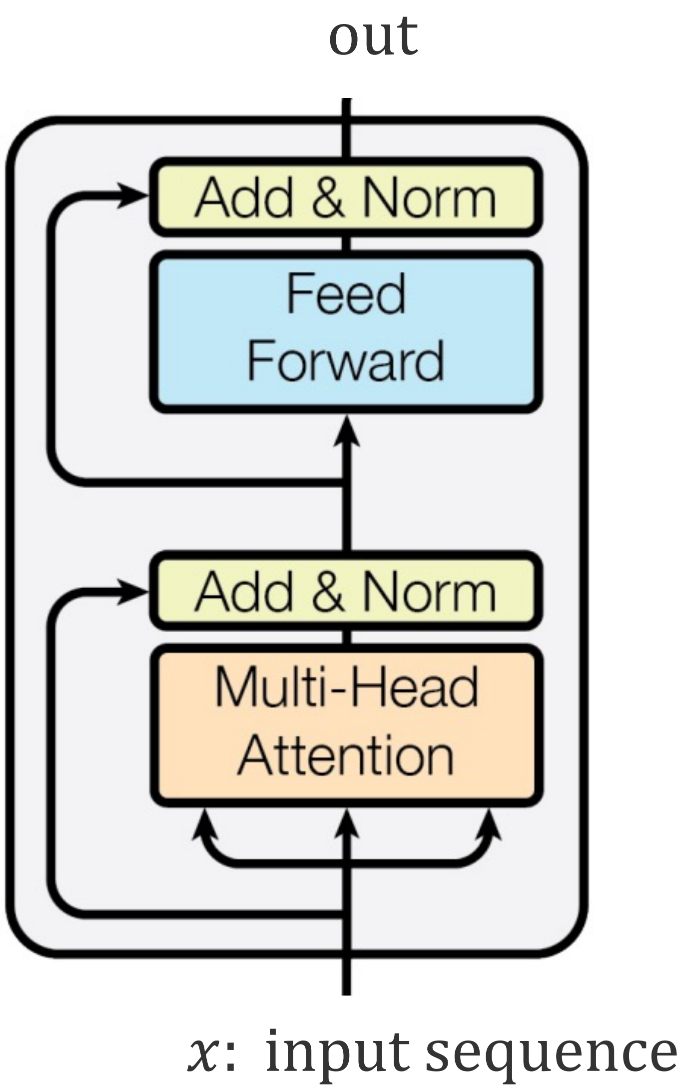
- A position-wise transformation consisting of:
  - A linear transformation, non-linear activation f (e.g., ReLU), and another linear transformation.

$$FF(c) = f(cW_1 + b_1)W_2 + b_2$$

- This allows the model to apply another transformation to the contextual representations (or “post-process” them)

- Usually the dimensionality of the hidden feedforward layer is 2-8 times larger than the input dimension

A transformer block



$\text{out} = LayerNorm\big(c' + \text{FF}(c')\big)$ (Residual connection)

$FF(c') = f(c'W_1 + b_1)W_2 + b_2$

$c' = LayerNorm(c + x)$

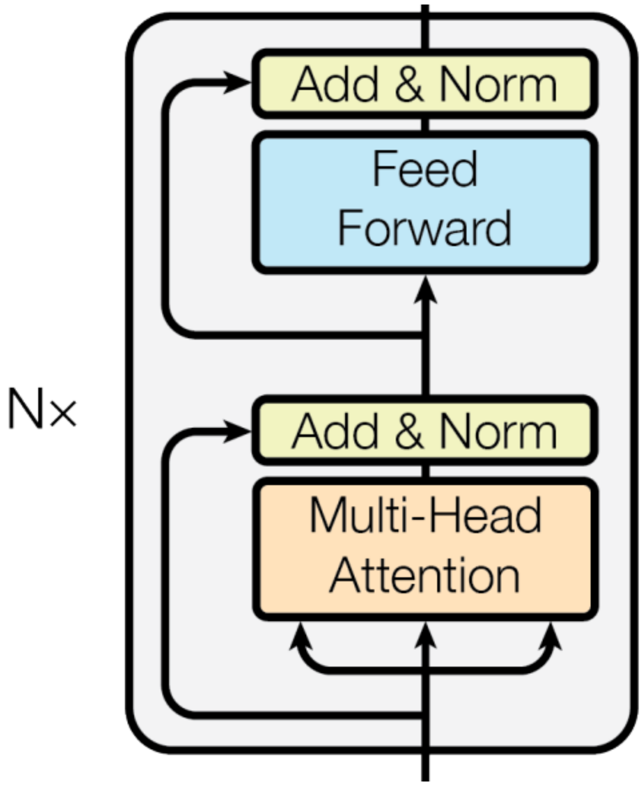
$c = \text{MultiHeadAttention}(\text{q}, \text{k}, \text{v})$

$q, k, v = \text{QKV\_Projection}(x)$

More details of LayerNorm and Residual Connection next week

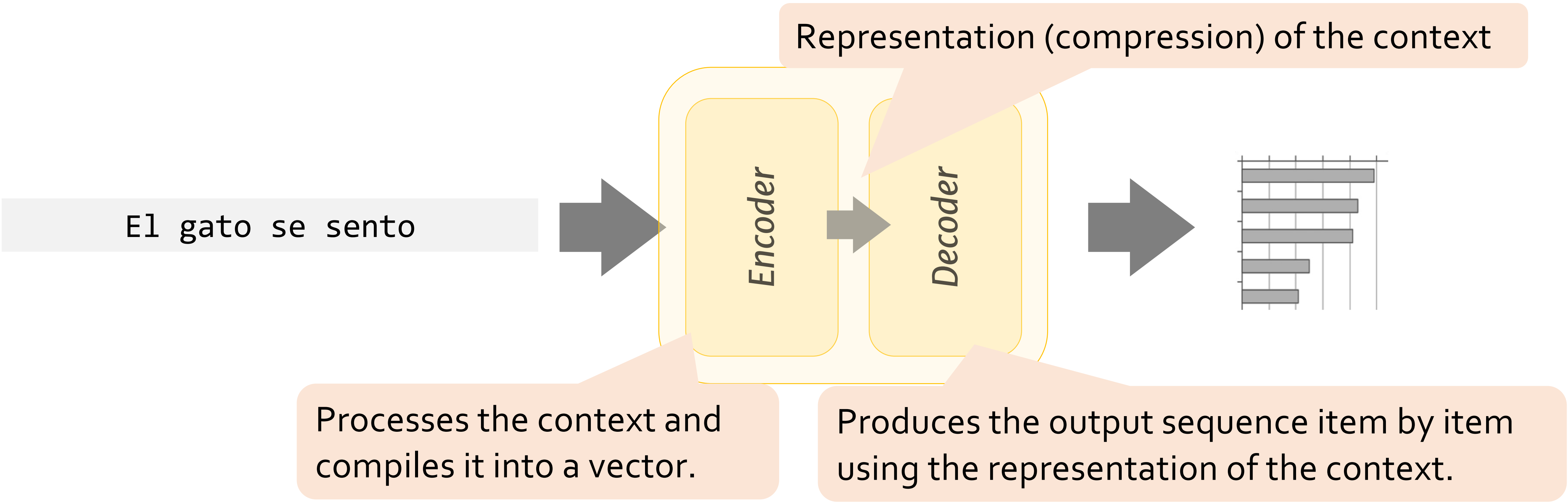
Transformer stack

- A stack of N transformer blocks (organized in N layers)

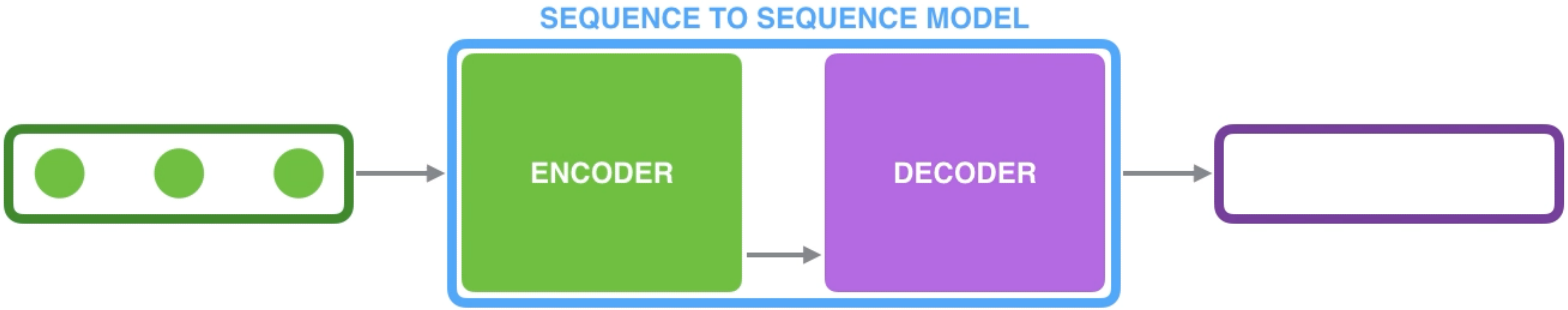


Encoder-Decoder Architectures

- Original transformer had two sub-models.



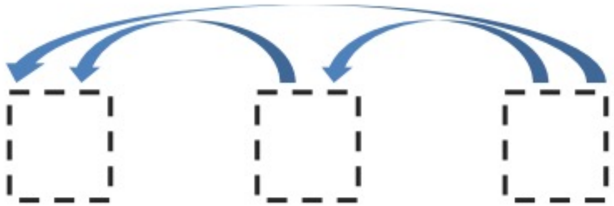
Encoder-Decoder Architectures

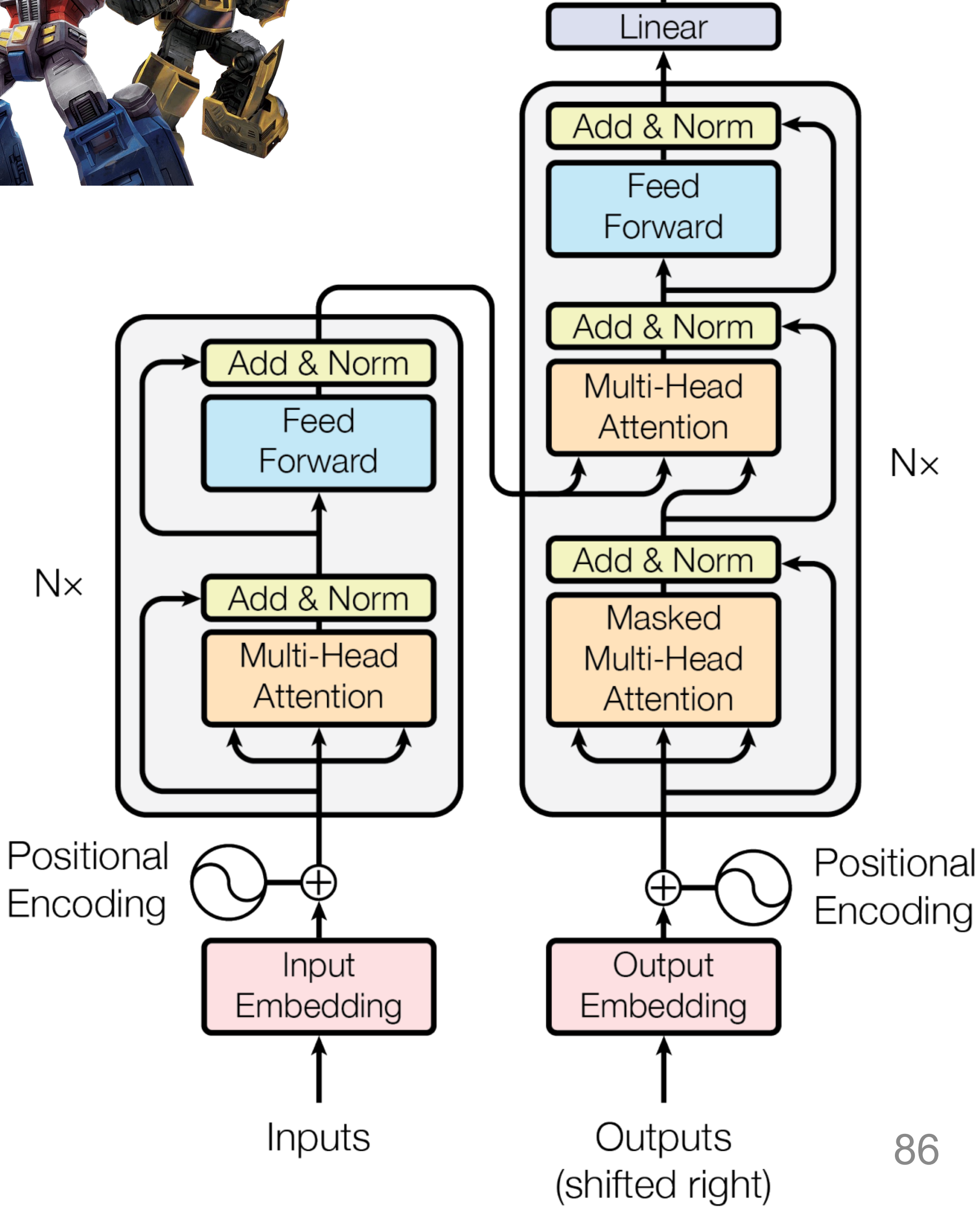


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Transformer [Vaswani et al. 2017]

- An encoder-decoder architecture built with attention modules.
- 3 forms of attention





Impact of Transformers

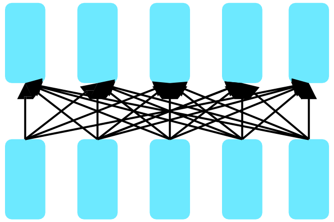
- Let to better predictive models of language ala GPTs!

Model	Layers	Heads	Perplexity
LSTMs (Grave et al., 2016)	-	-	40.8
QRNNs (Merity et al., 2018)	-	-	33.0
Transformer	16	16	19.8

[“Efficient Content-Based Sparse Attention with Routing Transformers” Roy et al. 2020]

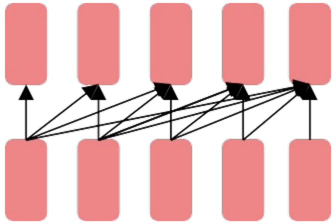
Impact of Transformers

- A building block for a variety of LMs



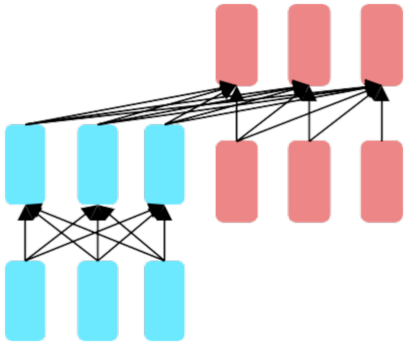
Encoders

- ❖ **Examples:** BERT, RoBERTa, SciBERT.
- ❖ Captures bidirectional context. How do we pretrain them?



Decoders

- ❖ **Examples:** GPT-2, GPT-3, Llama models, and many many more
- ❖ Other name: **causal or auto-regressive language model**
- ❖ Nice to generate from; can't condition on future words



Encoder-Decoders

- ❖ **Examples:** Transformer, T5, BART
- ❖ What's the best way to pretrain them?

Transformer LMs + Scale = LLMs

- 2 main dimensions:
- Model size, pretraining data size

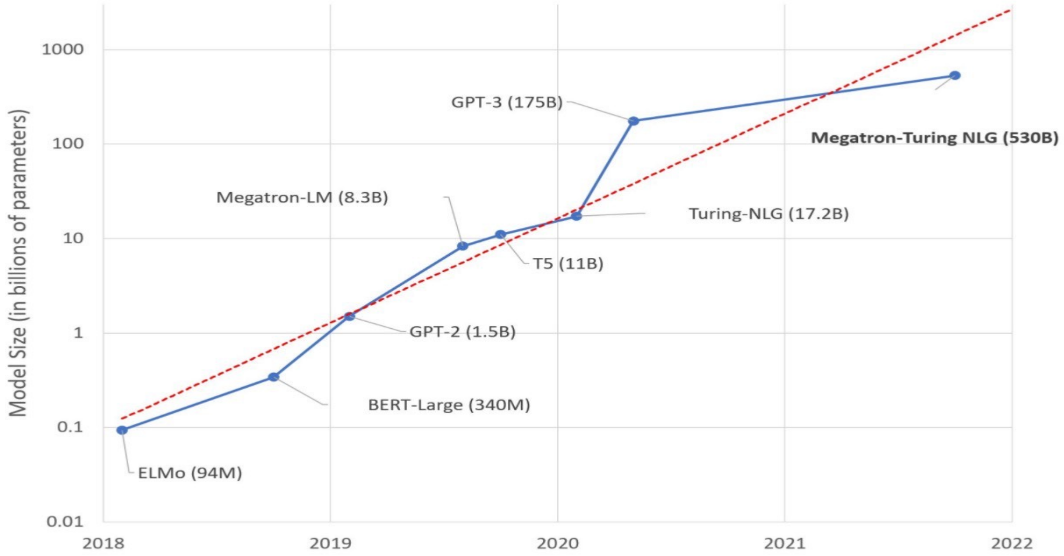
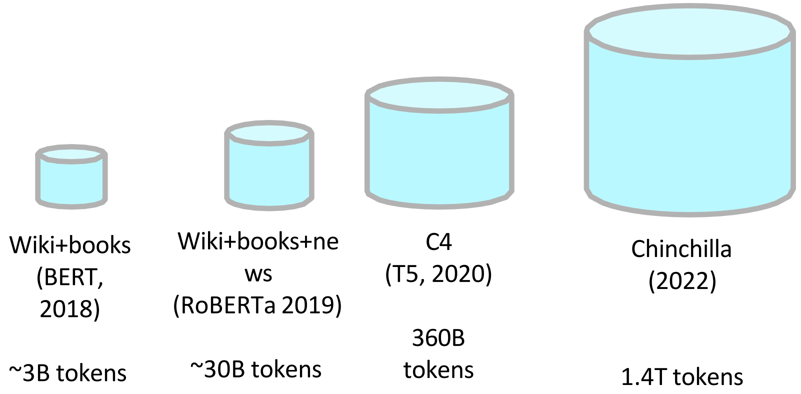
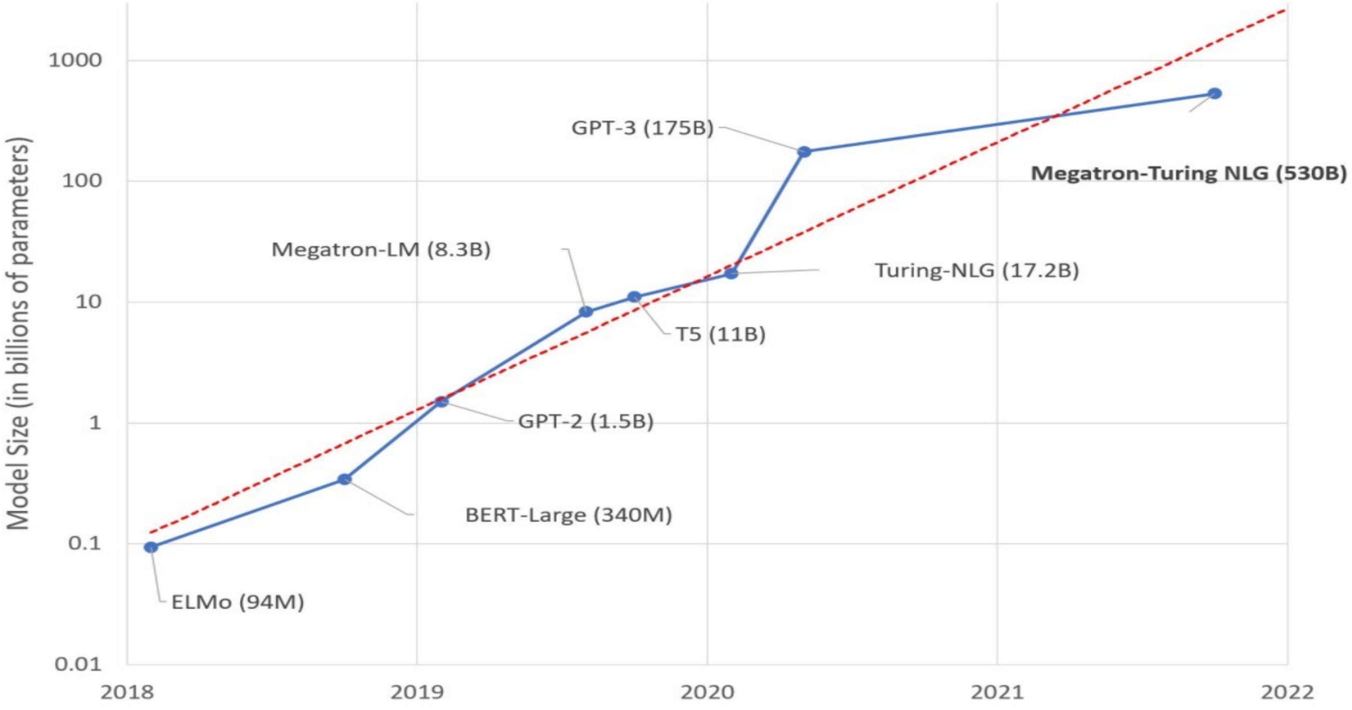


Photo credit: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/



Large Language Models

- Not only they improved performance on many NLP tasks, but exhibited new capabilities



Transformers - Summary

- Self-attention + positional embedding + others = NLP go brr
- Much faster to train than any previous architectures, much easier to scale
- Perform on par or better than previous RNN based models
  - Ease of scaling allows to extract much better performance