Language Modeling II

CSE 5525: Foundations of Speech and Language Processing

https://shocheen.github.io/cse-5525-spring-2025/

The Ohio State University

Sachin Kumar (kumar.1145@osu.edu)

Slide Credits: Chris Tanner

Logistics

- Homework 1 due date was Wednesday. How did everyone do?
  - Any thoughts, questions, concerns?

- Homework 2 is released. Due in two weeks (Feb 5)
  - Topic: Language Modeling with Transformers

Recap from last class

- What are language models
  - Distributions over sequences of “tokens”.
  - Tokens can be: words, character, something else (more about that soon)

- What are they useful for
  - Measure likelihood of given sequence, ranking different sequences, generating sequences, and more

- How do you measure if a given language model is good
  - Perplexity

- How do you train a language model
  - N-gram LMs

This Class and Beyond: Neural Language Models

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

The cat sat on the mat

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

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

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

But more broadly,

$$\underbrace{\mathrm{P}(X_1, \ldots, X_N) = \prod_t P(X_\mathrm{t} | \mathrm{X}_1, \ldots, X_{t-1})}_{\text{Chain rule}}$$

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

But more broadly,

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

A variant

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

Language Models: N-grams

- Probabilistic n-gram models of text generation
  - 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):

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

Neural Language Models

A Very Simple Approach

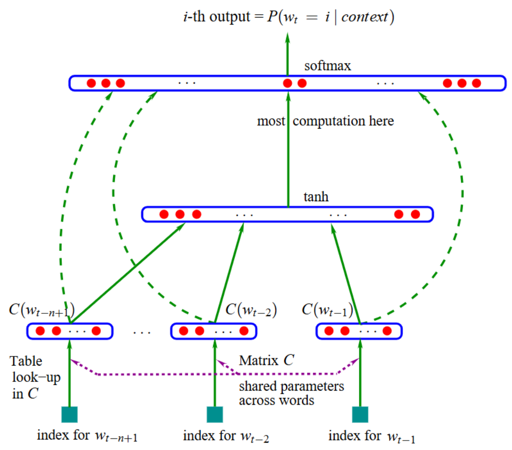
- Instead of having count-based distributions, parameterize them

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

- 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**"
- Dependencies can bridge arbitrarily long linear distances (similar to word2vec)
- It get even worse beyond the single sentence

Neural Language Models

An MLP with No Markov Assumption

- We need to model the parameterized distribution
    - $P(X_i|X_1, \ldots\ X_{i-2}\ , X_{i-1}, \theta)$
- 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 so 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 losses 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

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

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

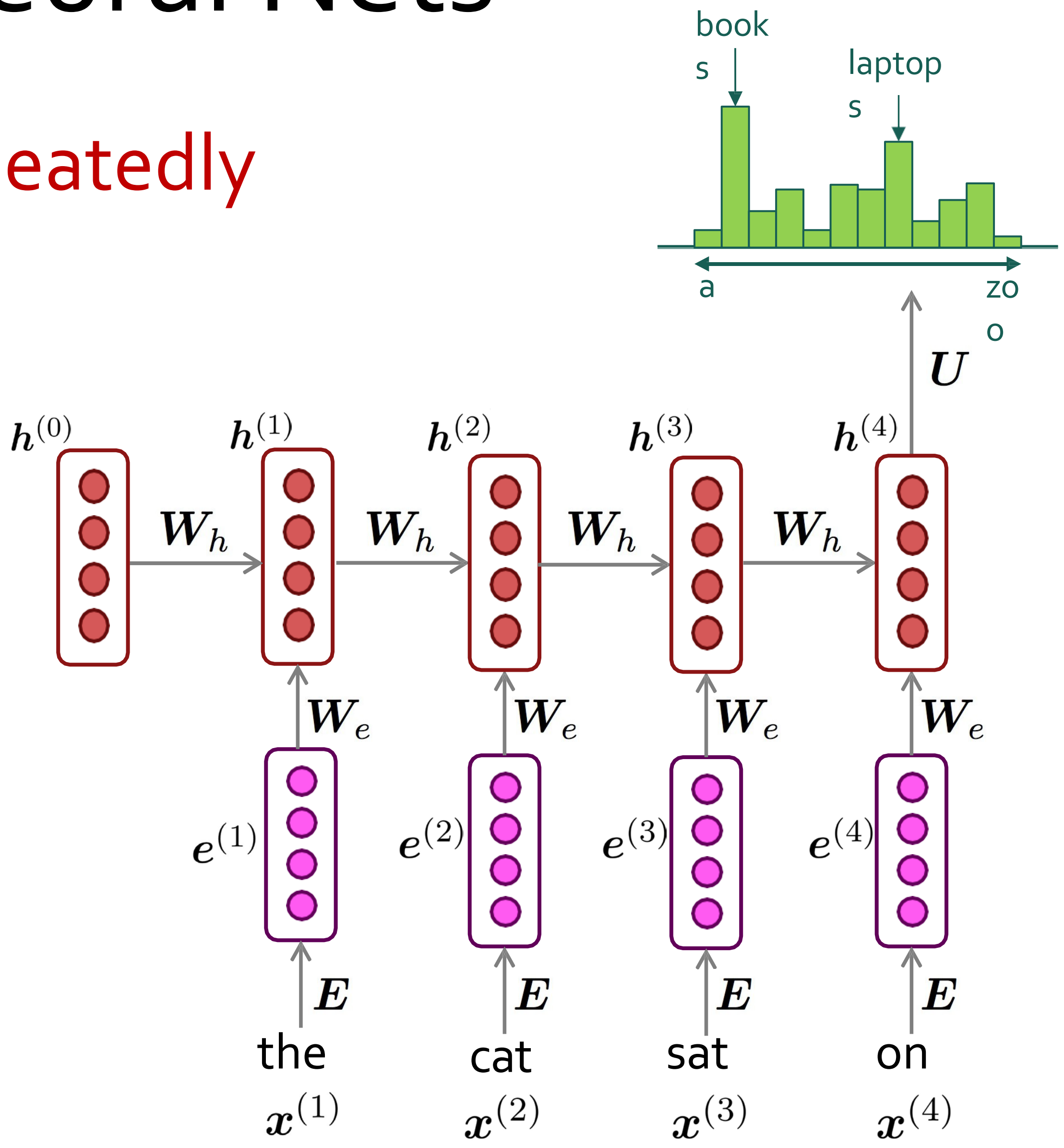
$\boldsymbol{h}^{(0)}$ is the initial hidden state

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

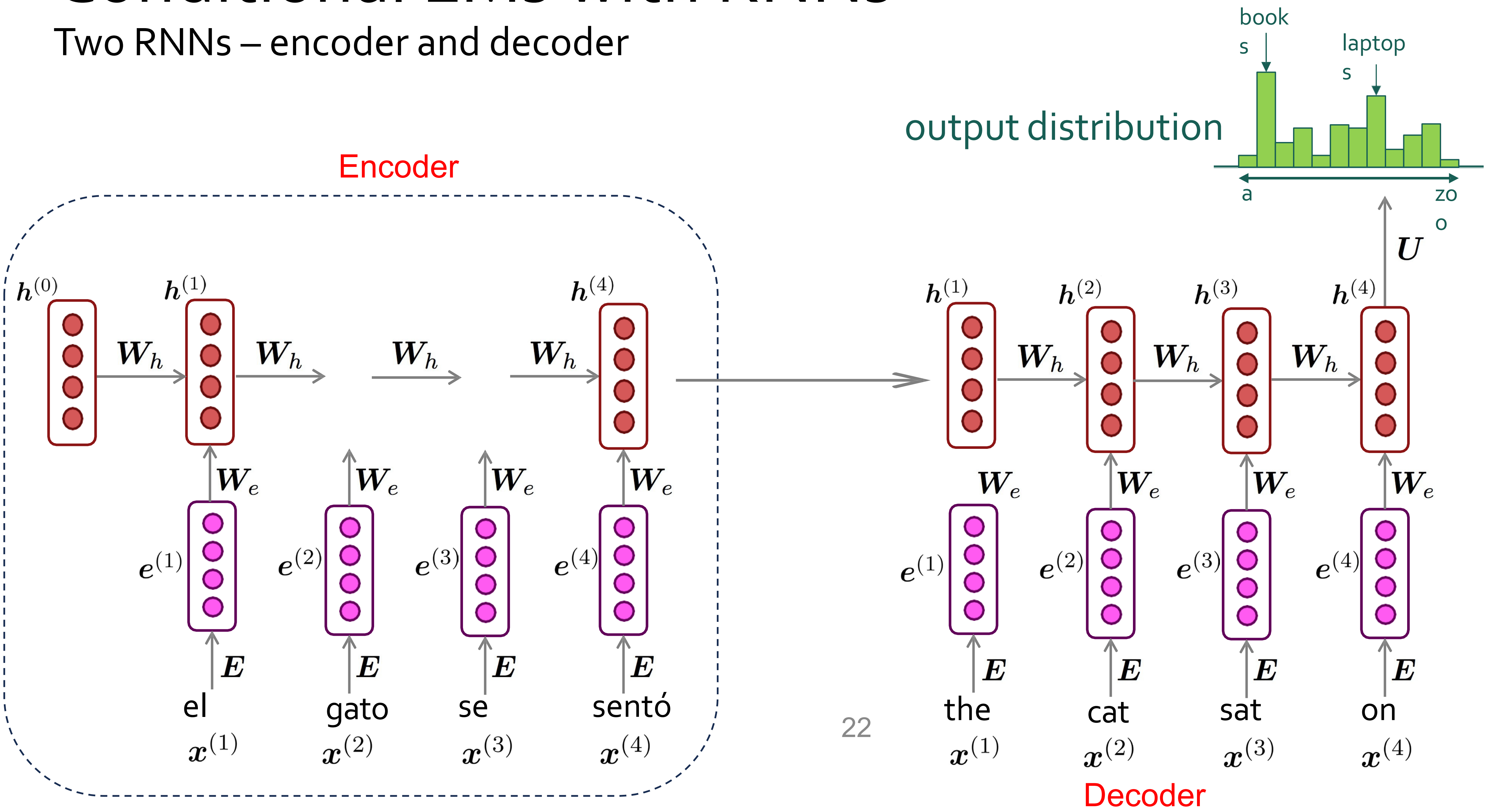
Recall: Conditional Language Models

- 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



22
22

How to train RNNs?

- Using our favorite algorithm: gradient descent using cross-entropy loss at every output step

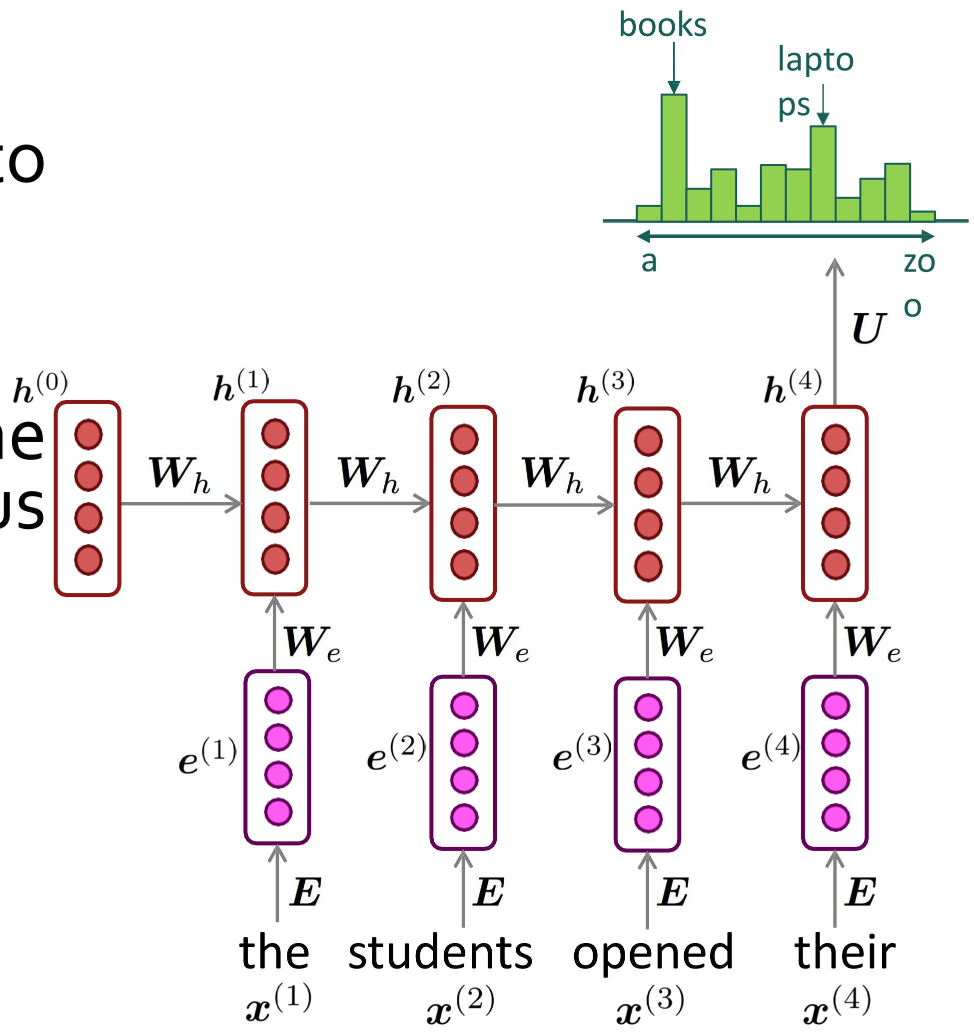
- But backpropagation is applied over and over to the same parameters theta
  - Also known as backpropagation through time (BPTT)

- 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

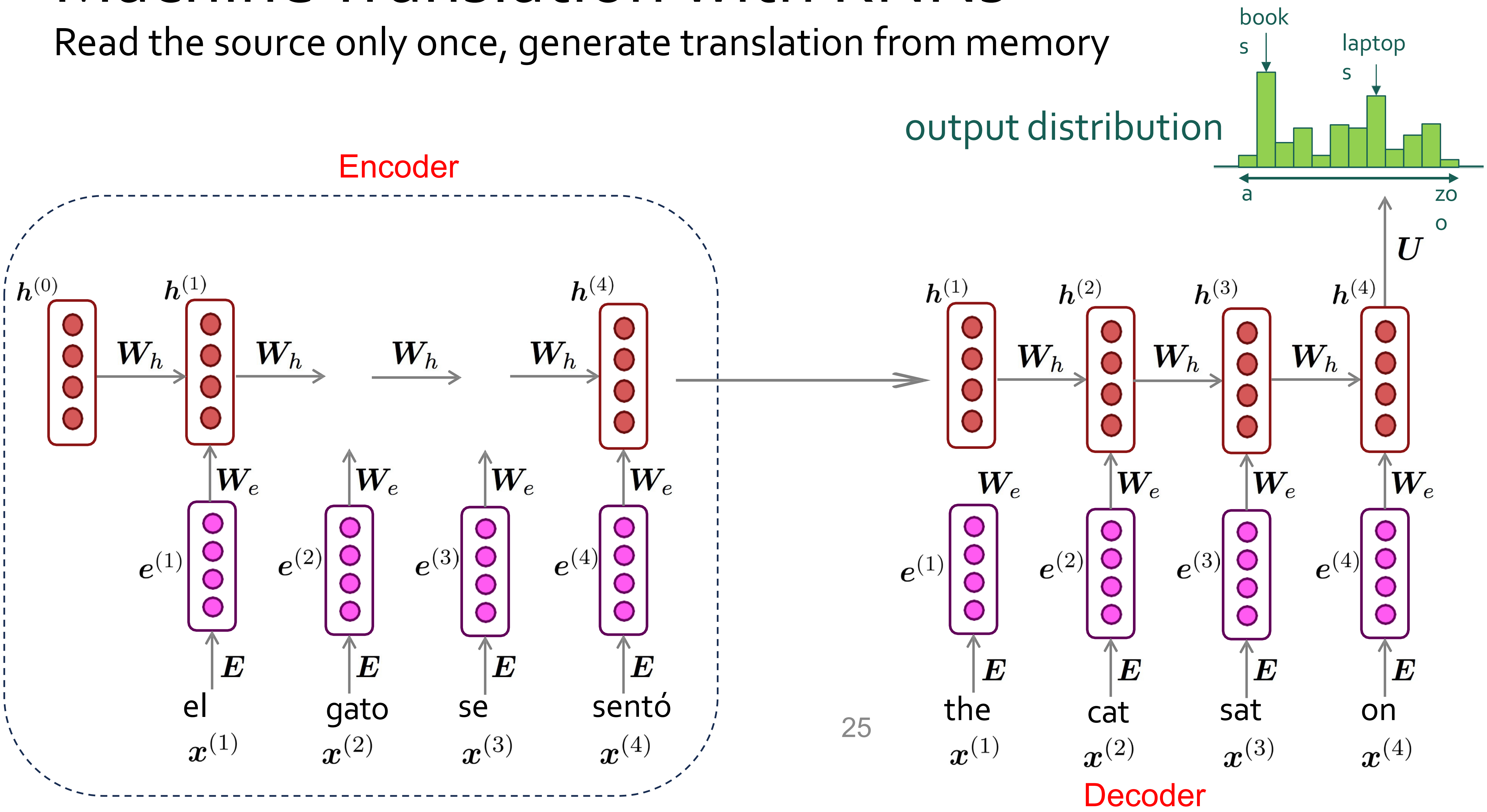
- Recurrent computation is slow, difficult to parallelize.

- 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



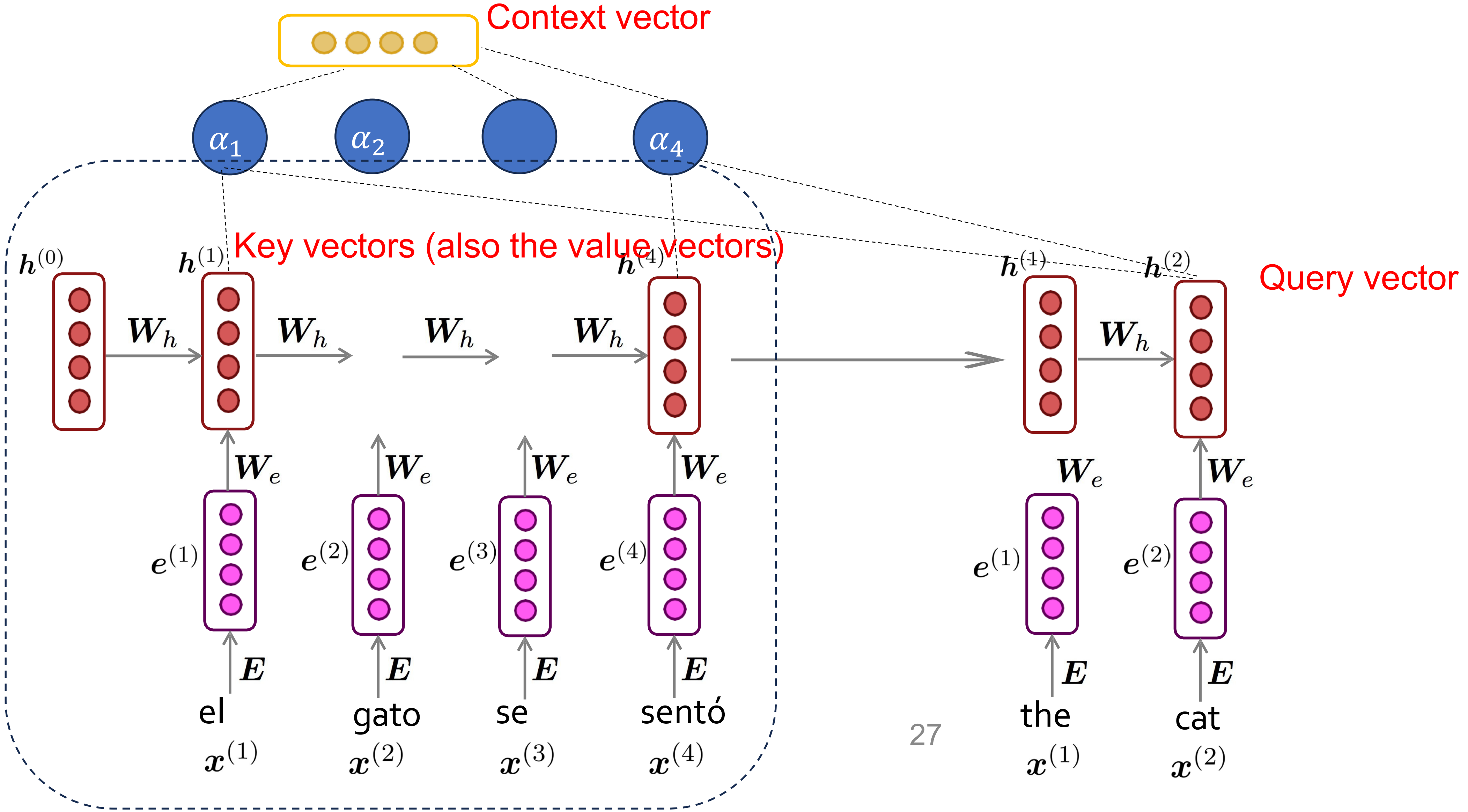
25

Solution: Attention

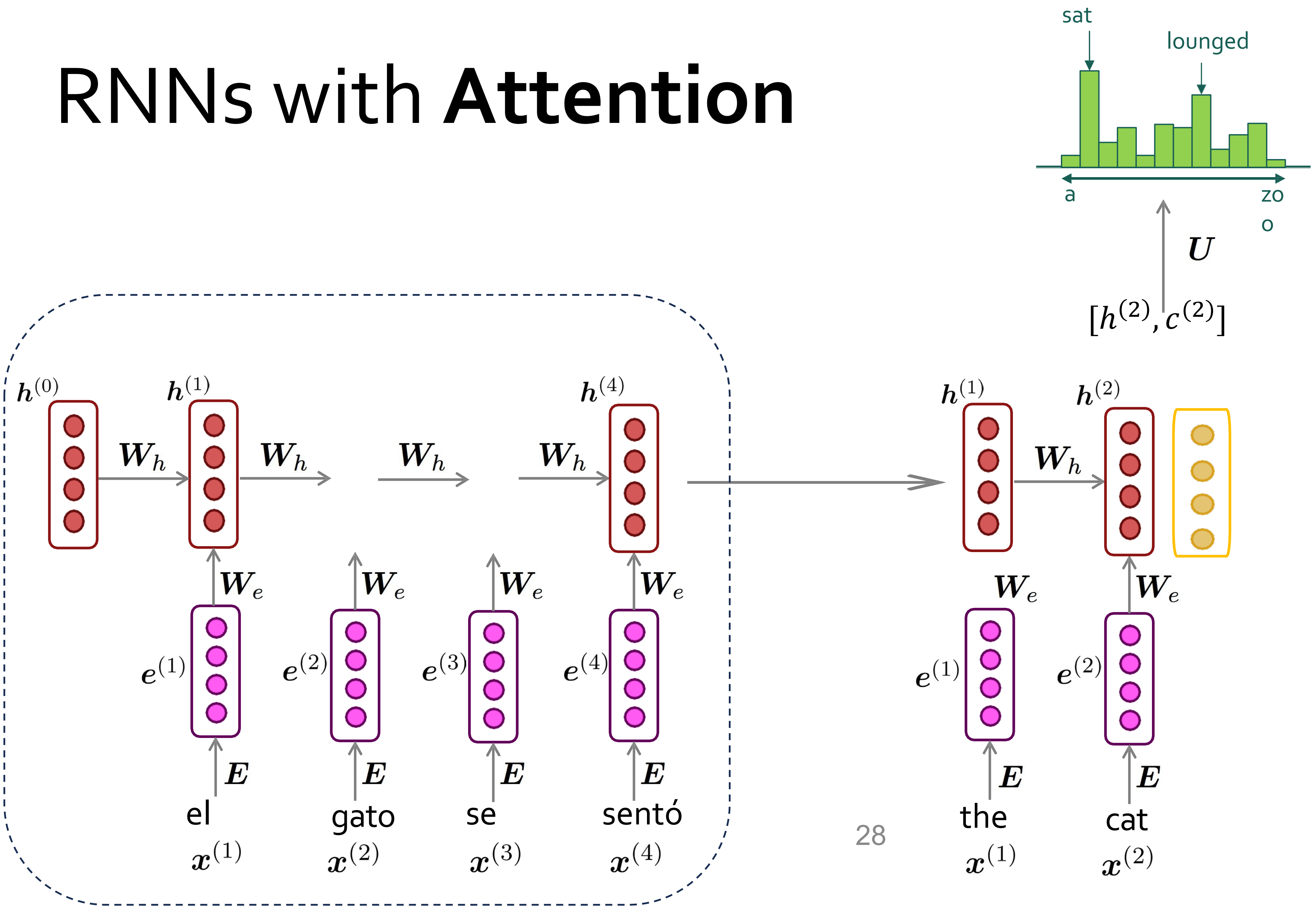
- What if the decoder at each step pays "attention" to a distribution of all of encoder's hidden states?

- 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**



RNNs with **Attention**



28
28

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

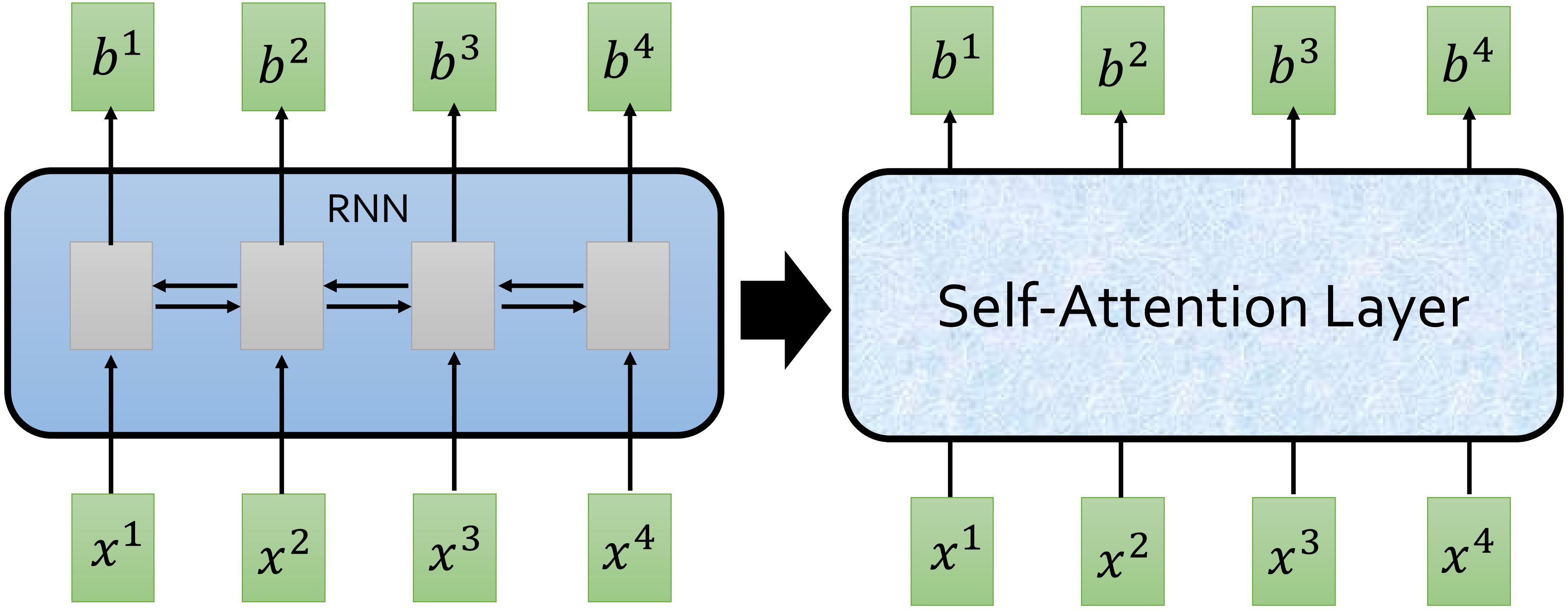
- Replace the linear part with **self-attention**

- 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.

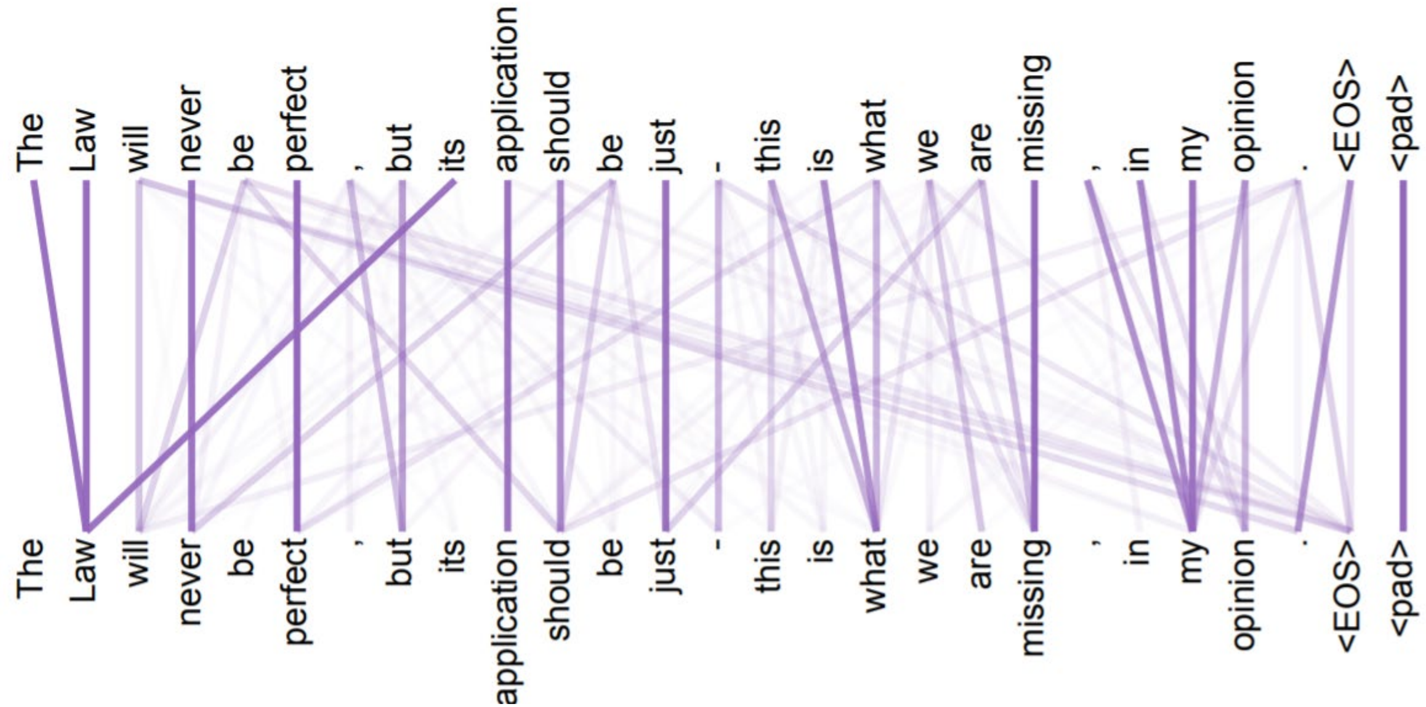


Idea: replace any thing done by RNN with self-attention.

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

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 **values**, 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)

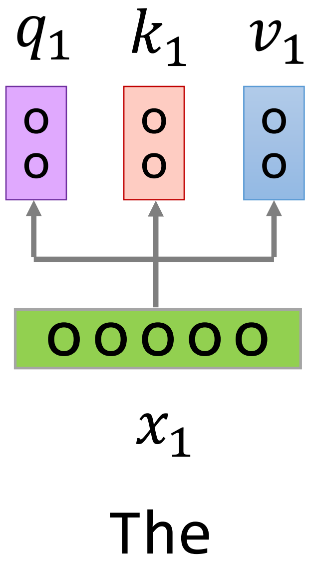
$$q_t = W^q x_t$$

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)

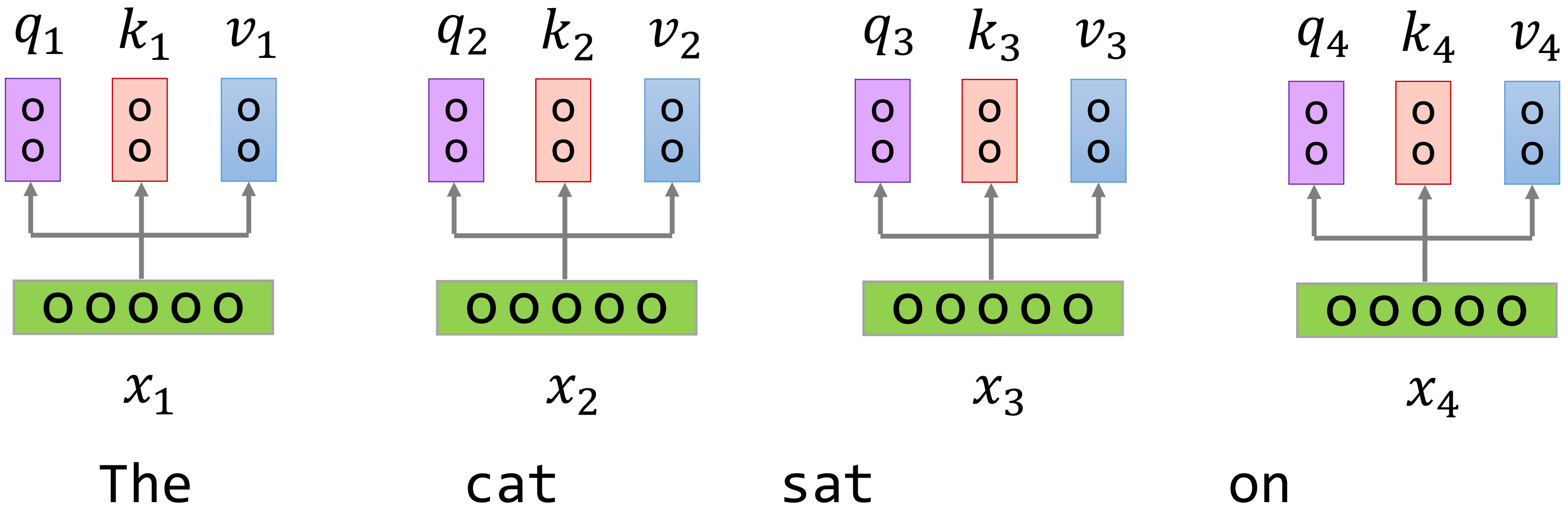
$$q_t = W^q x_t$$

k: key (to be matched)

$$k_t = W^k x_t$$

v: value (information to be extracted)

$$v_t = W^v x_t$$



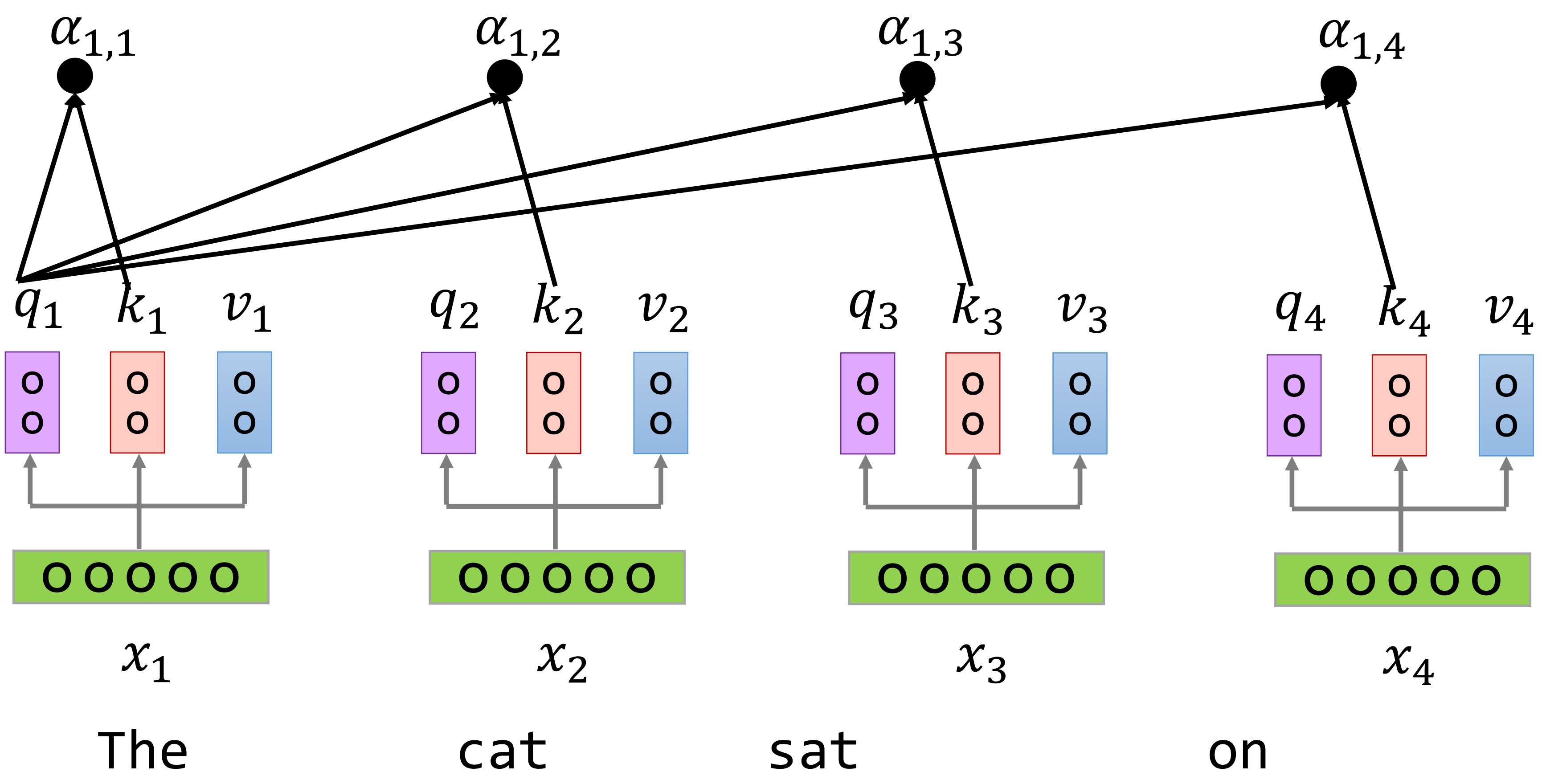
$$\alpha_{1,t} = \underbrace{q^1 \cdot k^t / \alpha}_{\text{Scaled dot product}}$$

q: query (to match others)

k: key (to be matched)

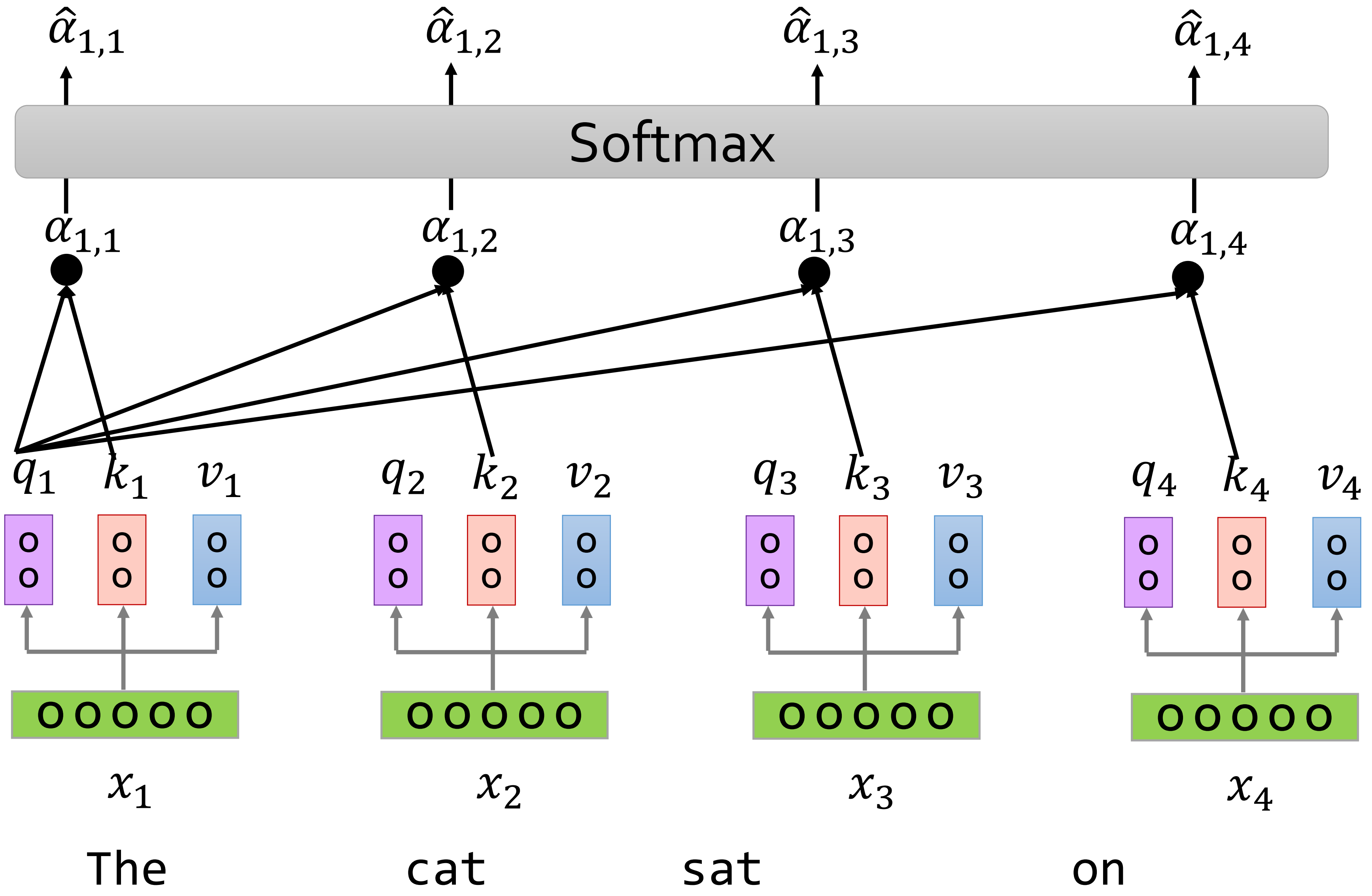
v: value (information to be extracted)

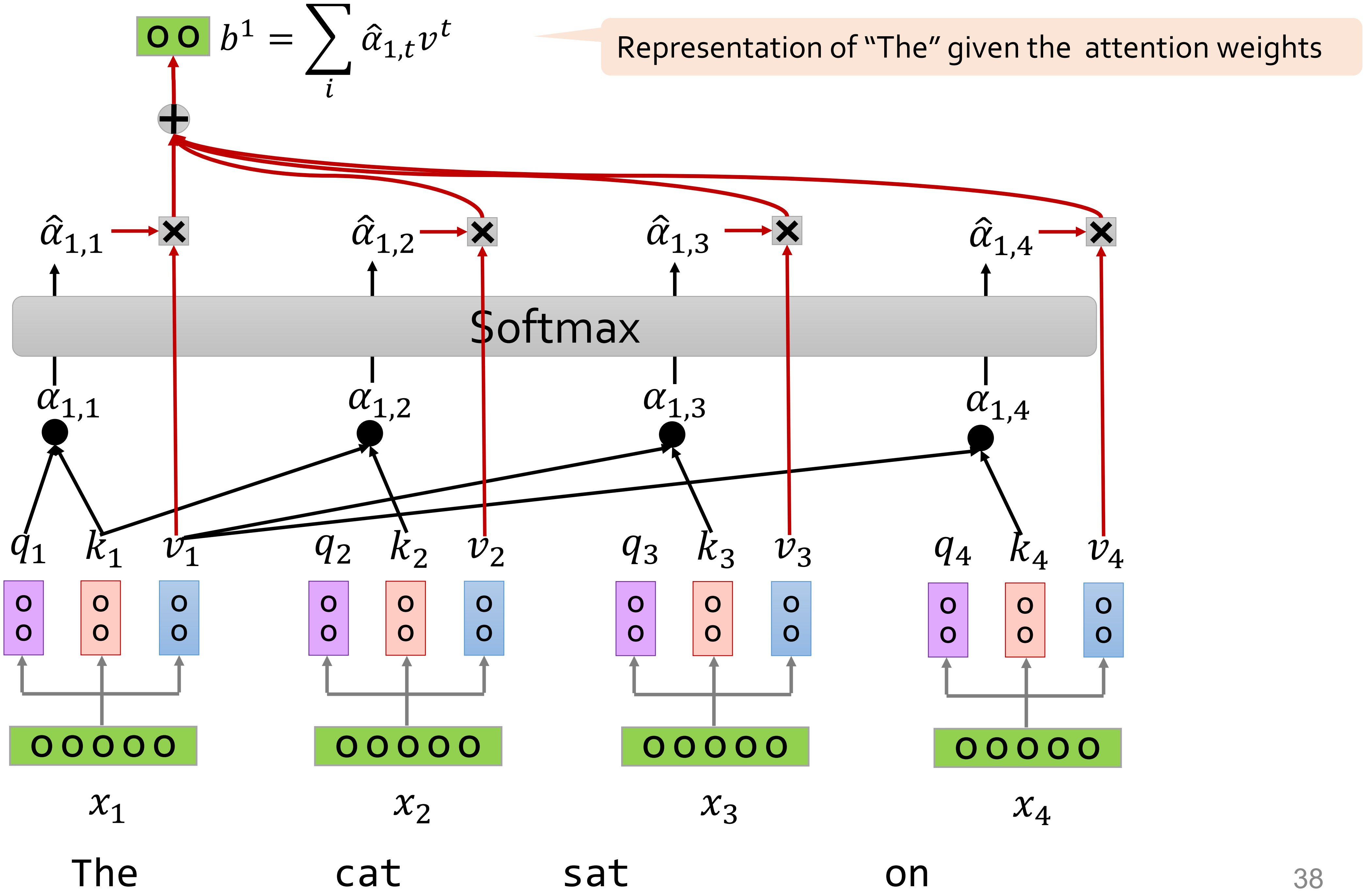
How much should "The" attend to other positions?



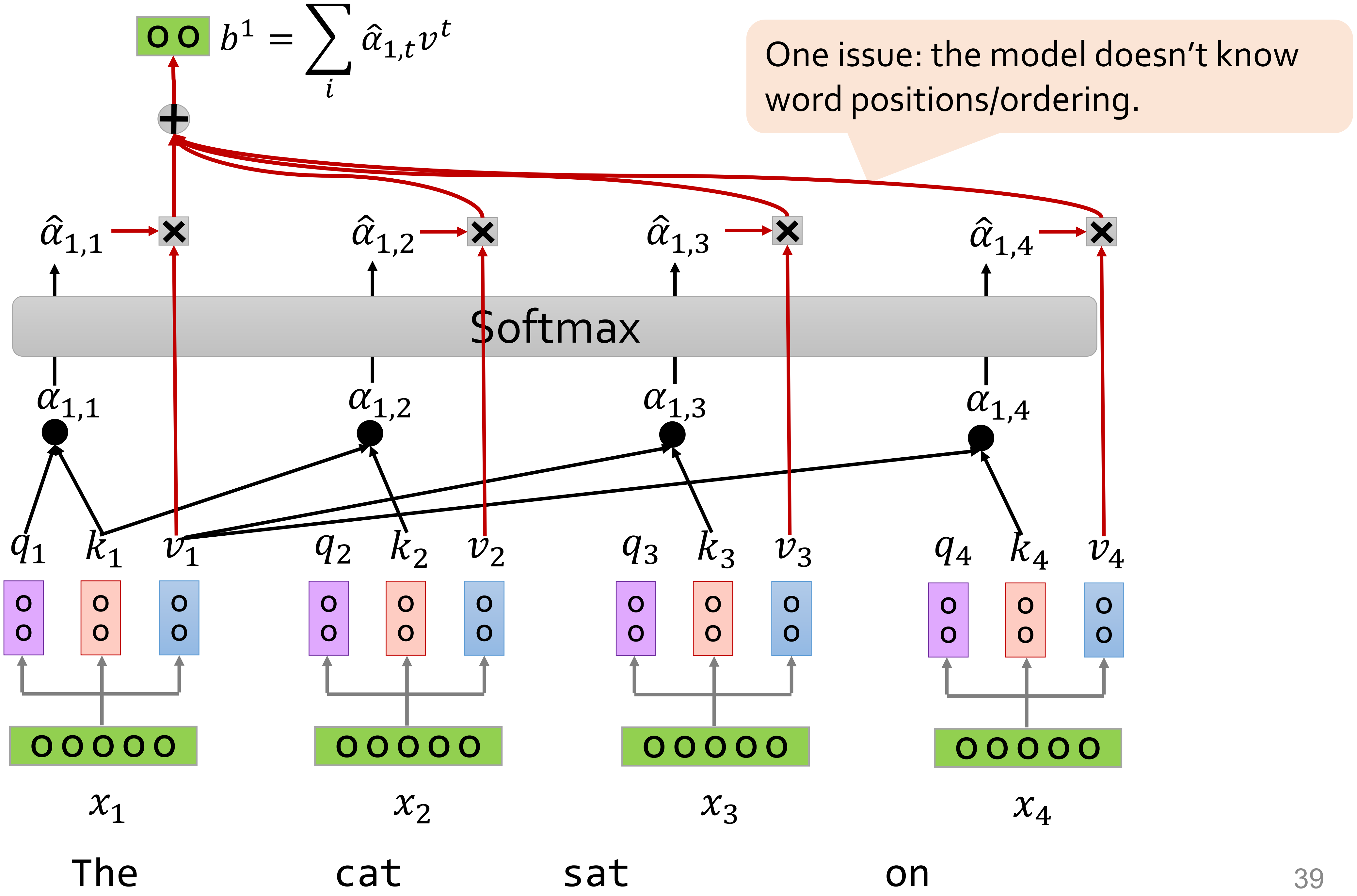
$$\sigma(z)_t = \frac{exp(z_t)}{\sum_j exp(z_j)}$$

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

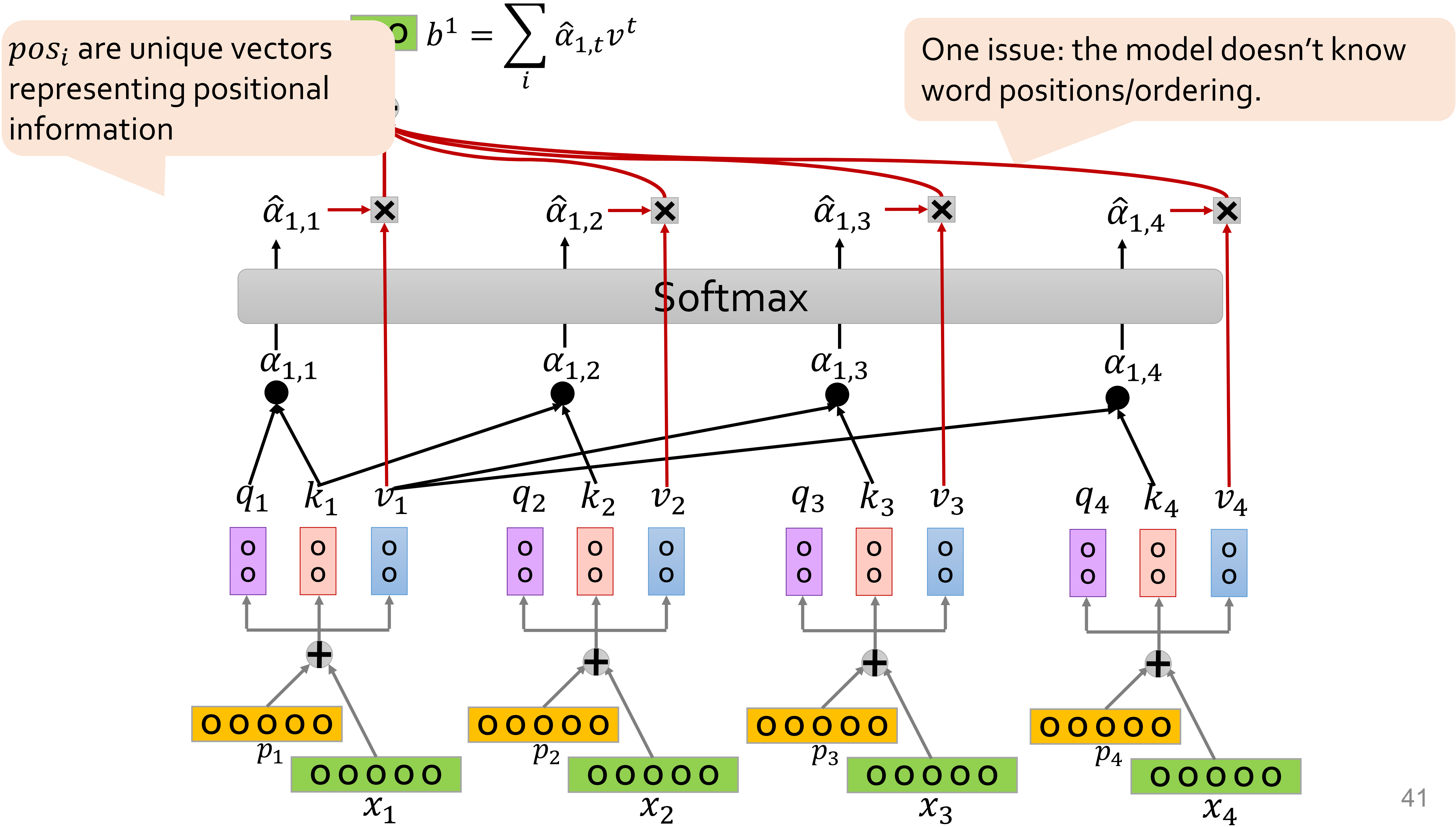
How to encode position information?

- 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
$b^1 = \sum_i \hat{\alpha}_{1,t} v^t$
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
p_1 x_1
p_2 x_2
p_3 x_3
p_4 x_4

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.
- It must be deterministic.

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 a enbedding.
    - 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 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)

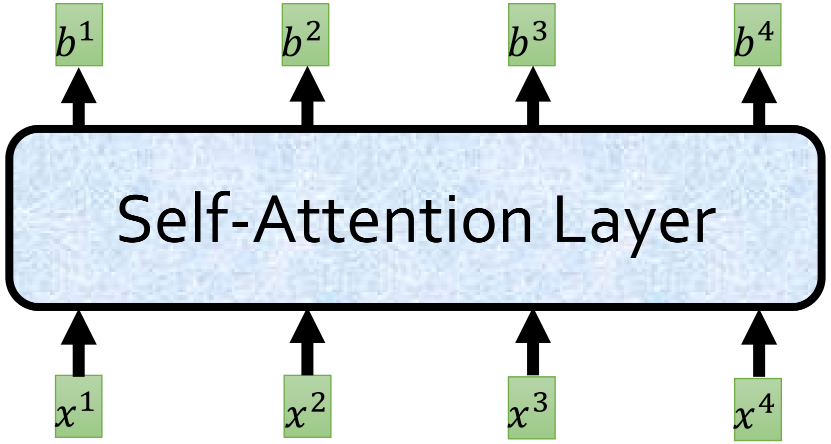
- AliBi: [2108.12409] Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (arxiv.org)

- 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

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

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,

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

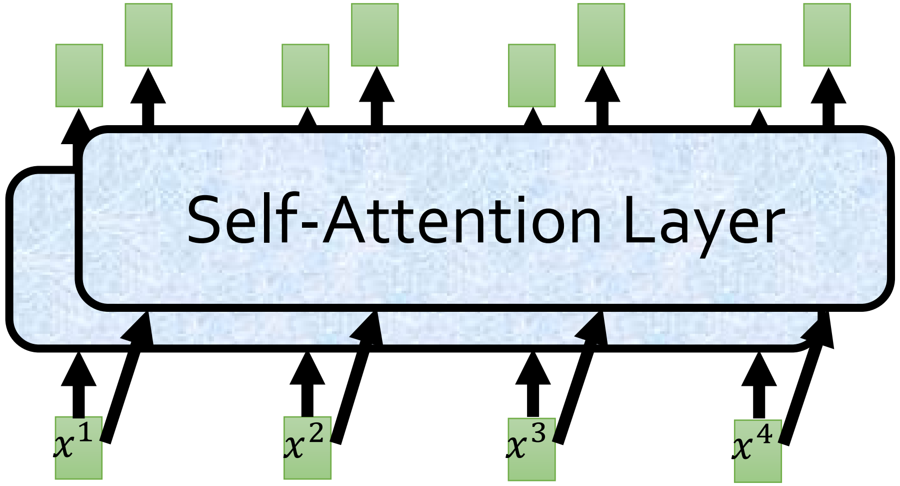
where softmax denotes a *row-wise* softmax normalization function. Thus, each element in S depends on all other elements in the same row.

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

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

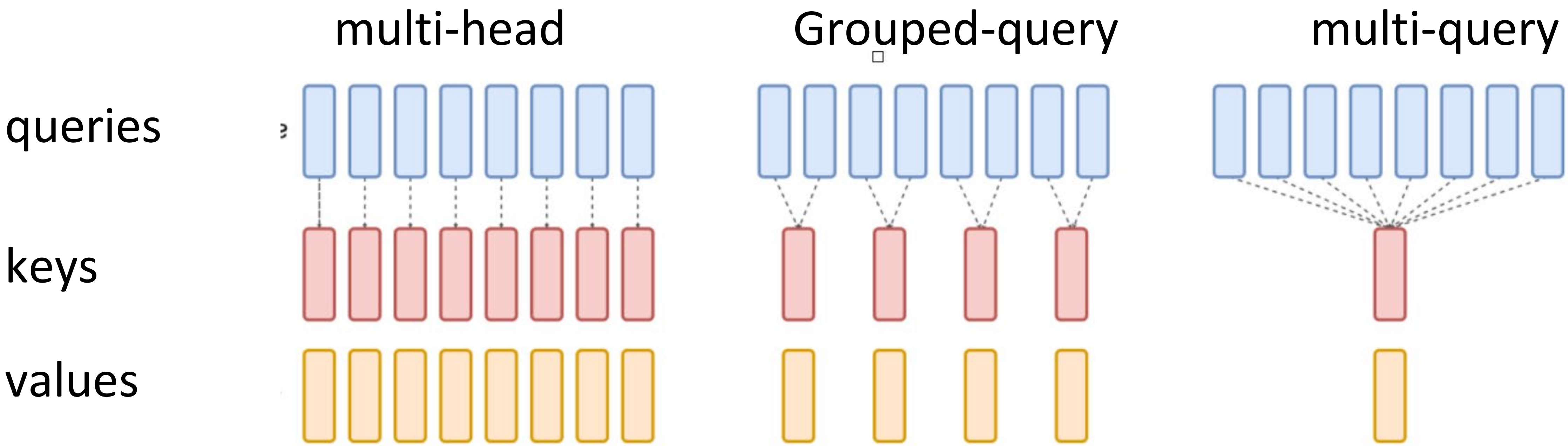
Multi-Headed Self-Attention

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



[Vaswani et al. 2017]

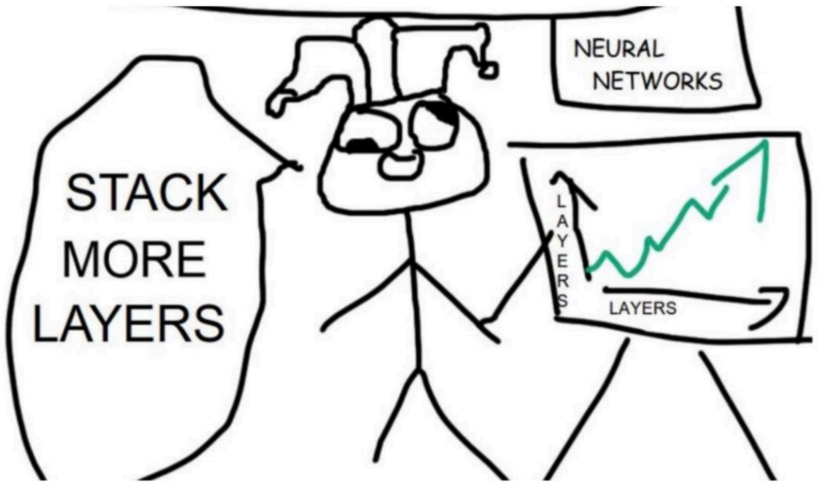
Variants of attention

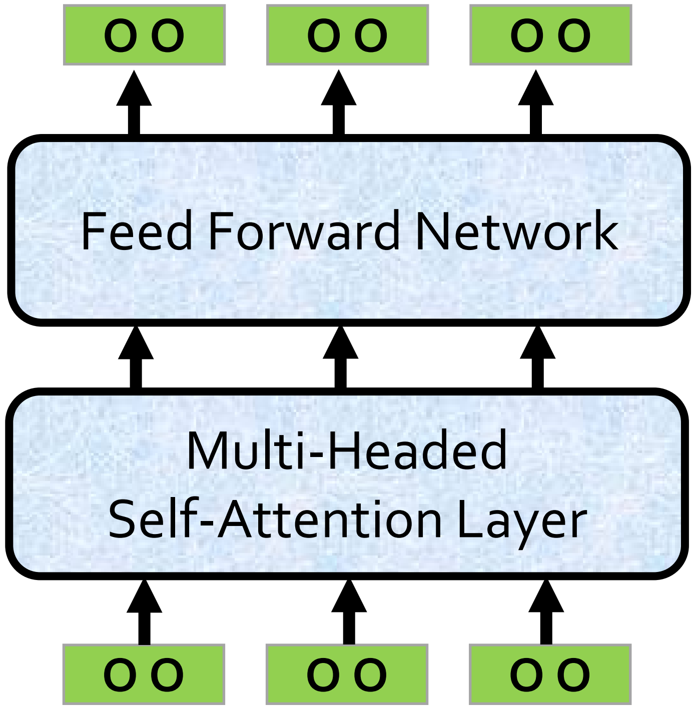


GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints (Ainslie et al., 2023)

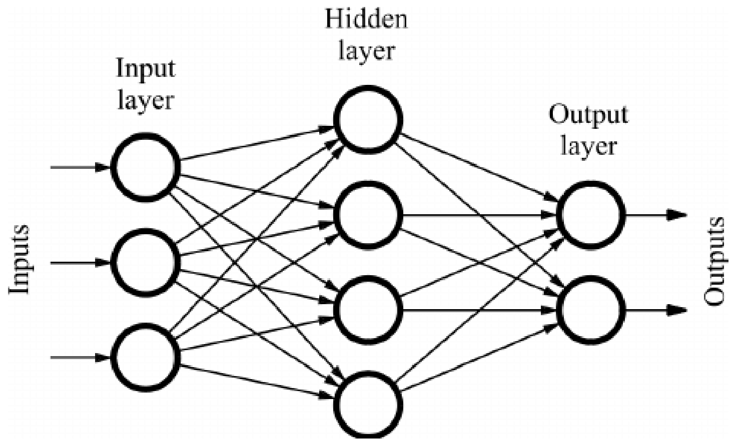
How Do We Make it Deep?

- 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.

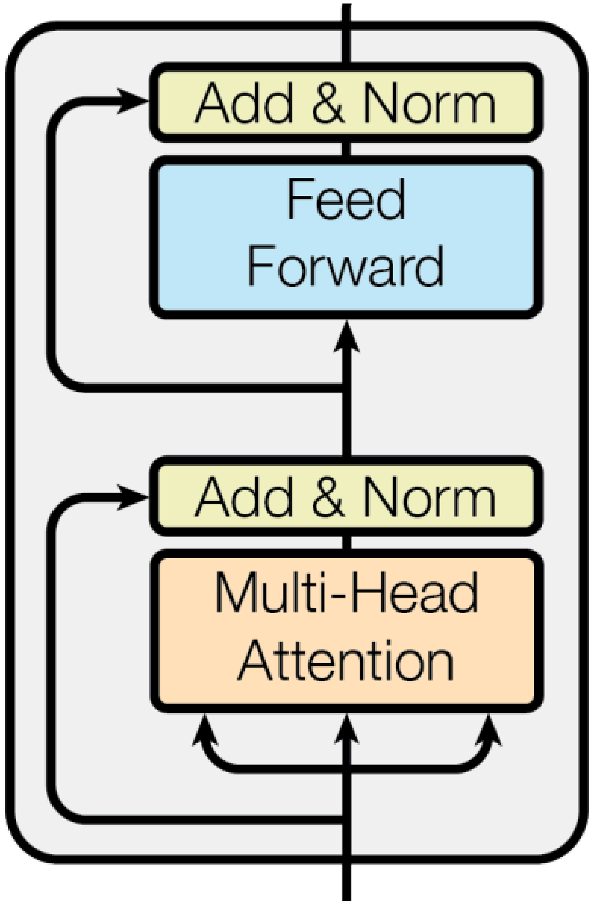
$$FF(c) = f(cW_1 + b_1)W_2 + b_2$$

  - 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

out



x: input sequence

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

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

$c' = LayerNorm(c + x)$

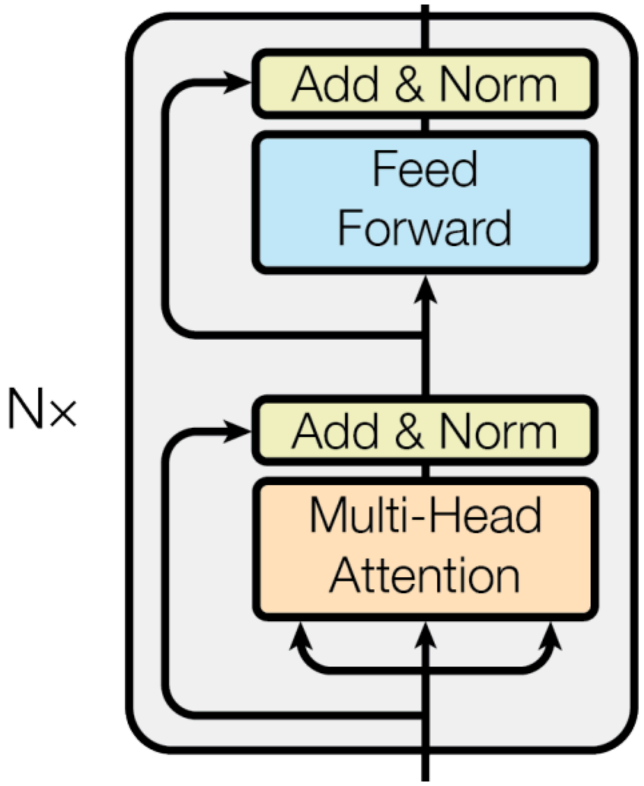
$c = \text{MultiHeadAttention}(\text{q}, \text{k}, \text{v})$

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

More details of LayerNorm and Residual Connection next week

Transformer stack

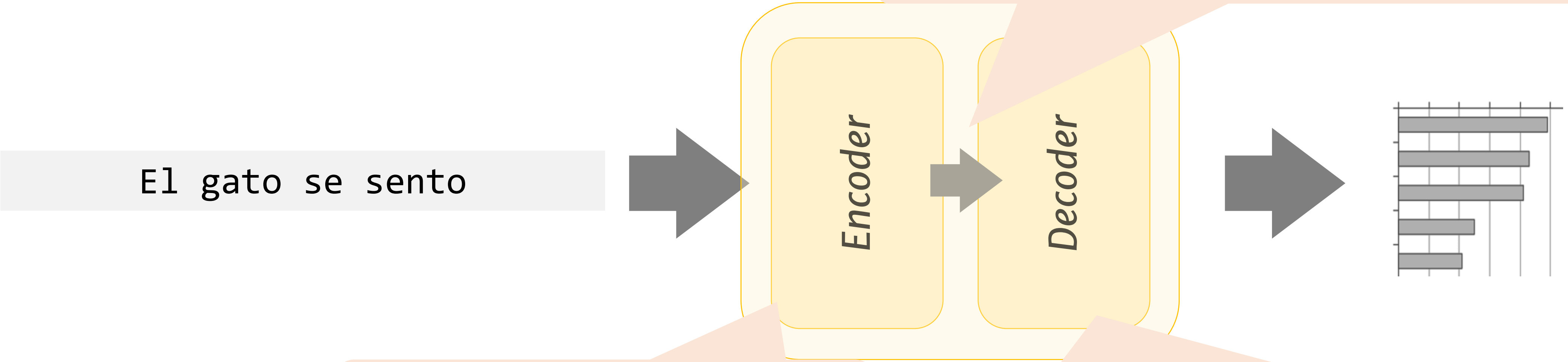
- A stack of N transformer blocks (organized in N layers)



Encoder-Decoder Architectures

- Original transformer had two sub-models.

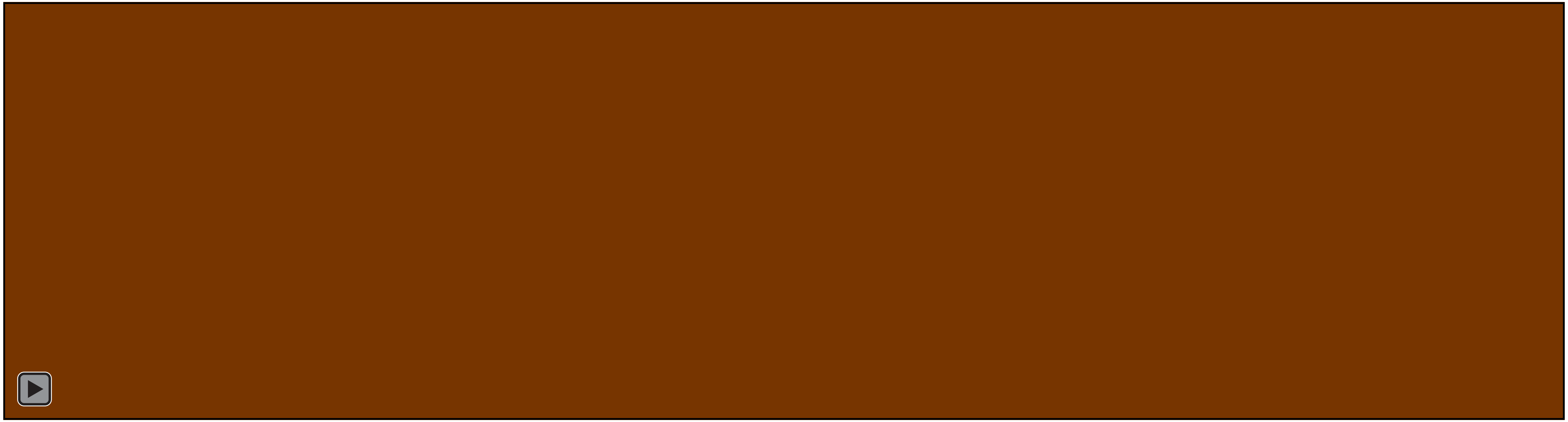
Representation (compression) of the context



Processes the context and compiles it into a vector.

Produces the output sequence item by item using the representation of the context.

Encoder-Decoder Architectures



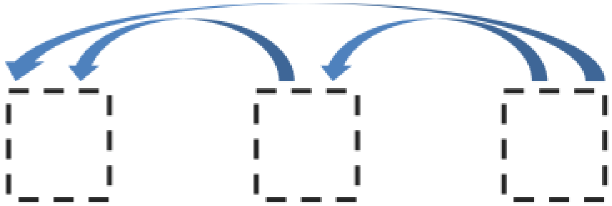
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Transformer [Vaswani et al. 2017]

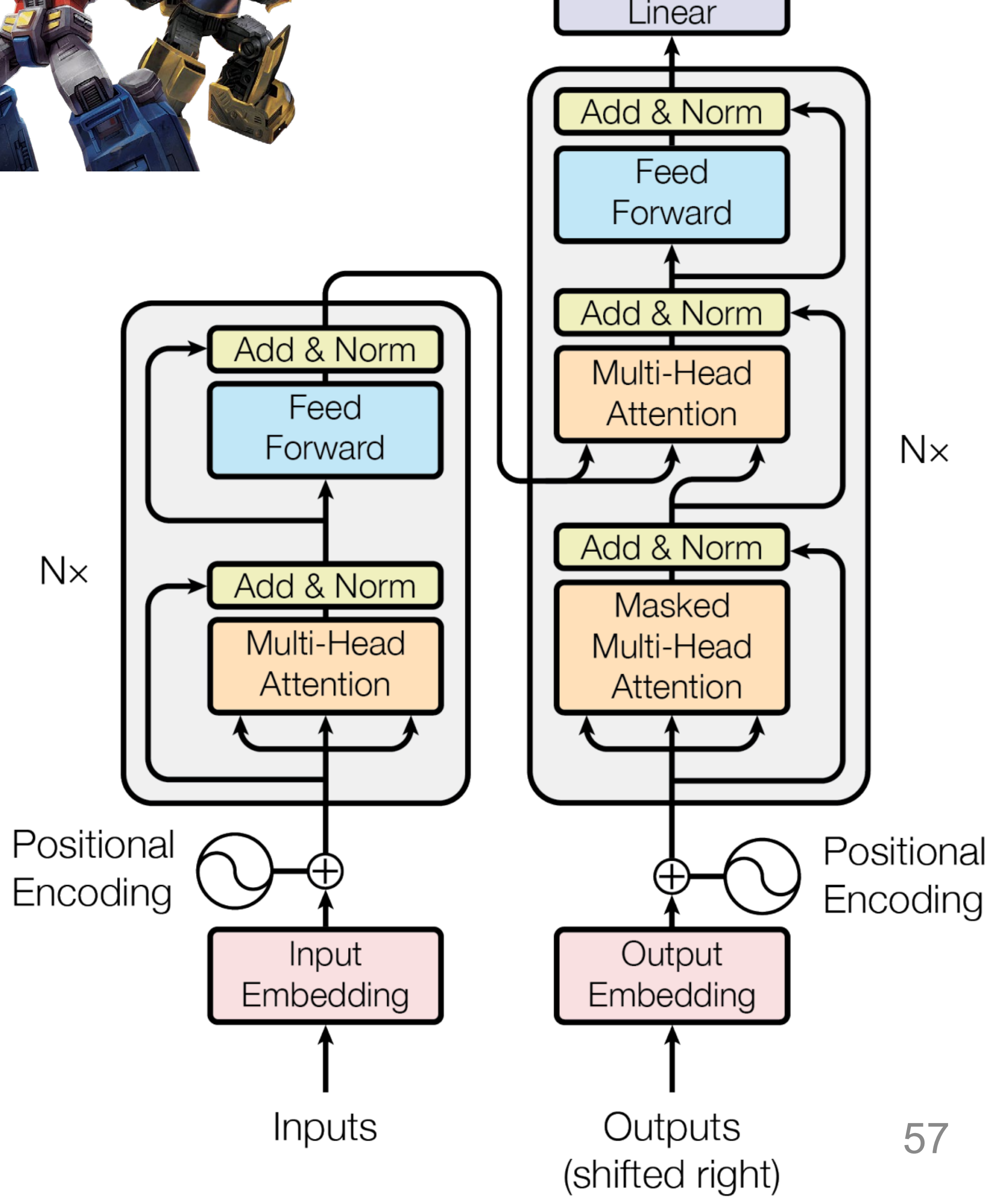
- An encoder-decoder architecture built with attention modules.
- 3 forms of attention

Encoder-Decoder Attention

Encoder Self-Attention



MaskedDecoder Self-Attention



Impact of Transformers

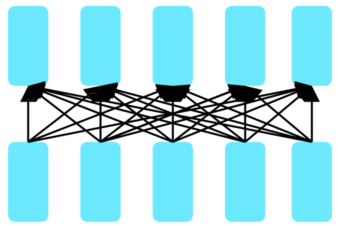
- Let to better predictive models of language ala GPTs!

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]

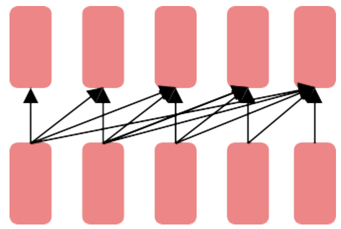
Impact of Transformers

- A building block for a variety of LMs



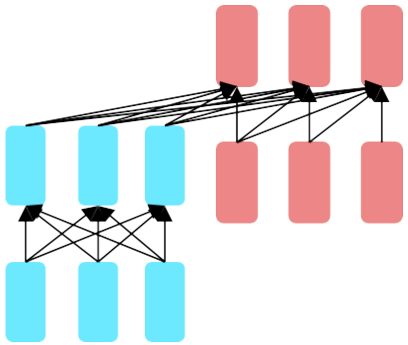
Encoders

- ❖ **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

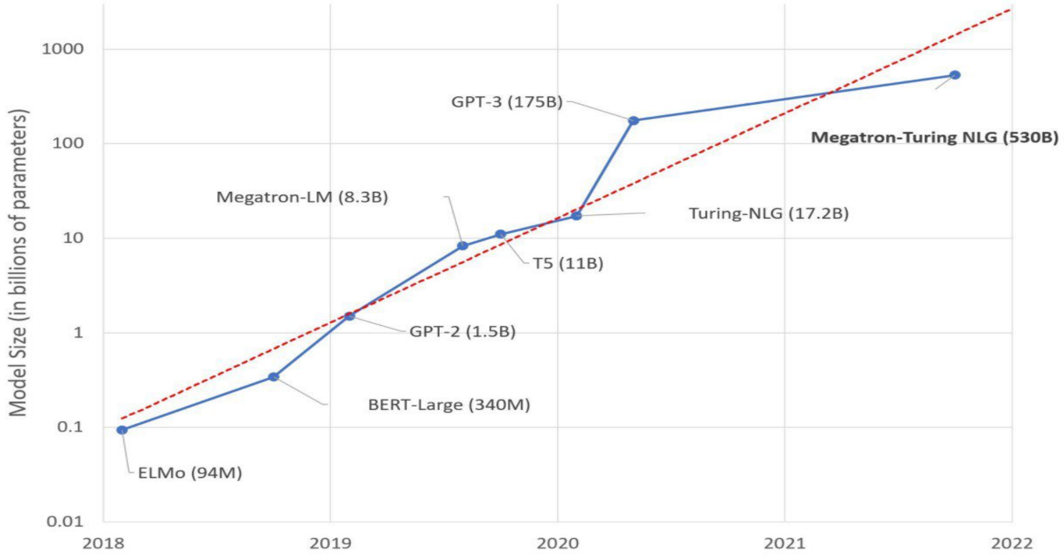
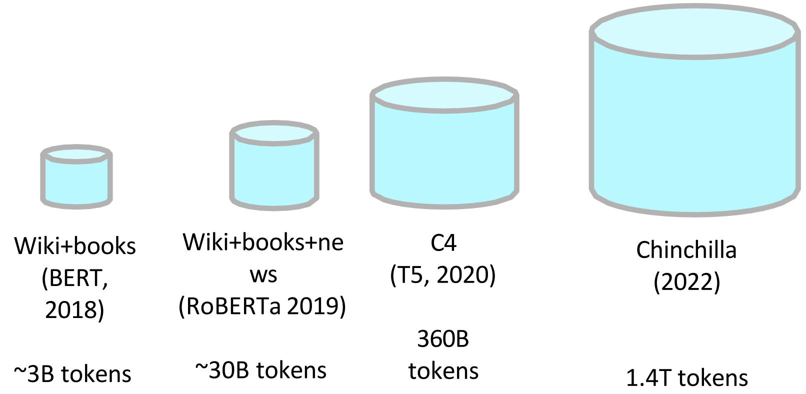
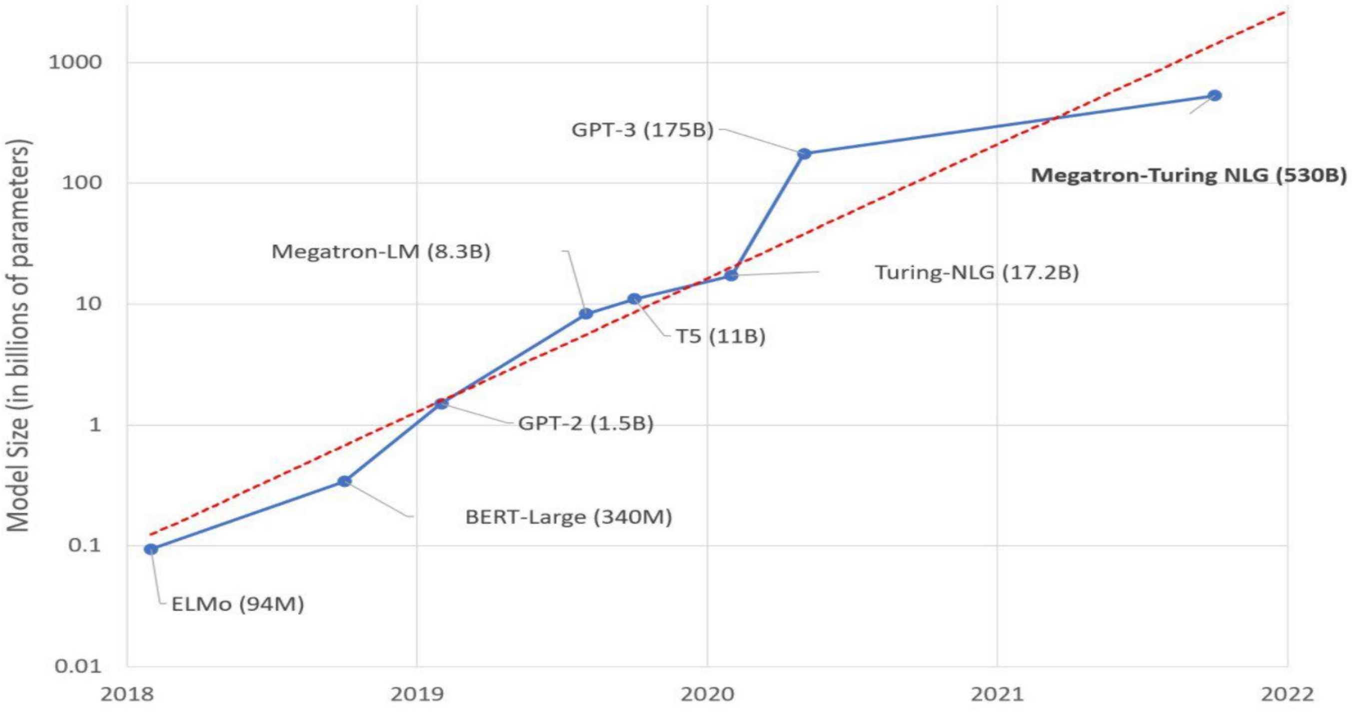


Photo credit: https://www.microsoft.com/en-us/research/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/



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