LLM Optimization 2

Haw-Shiuan Chang

Deadlines

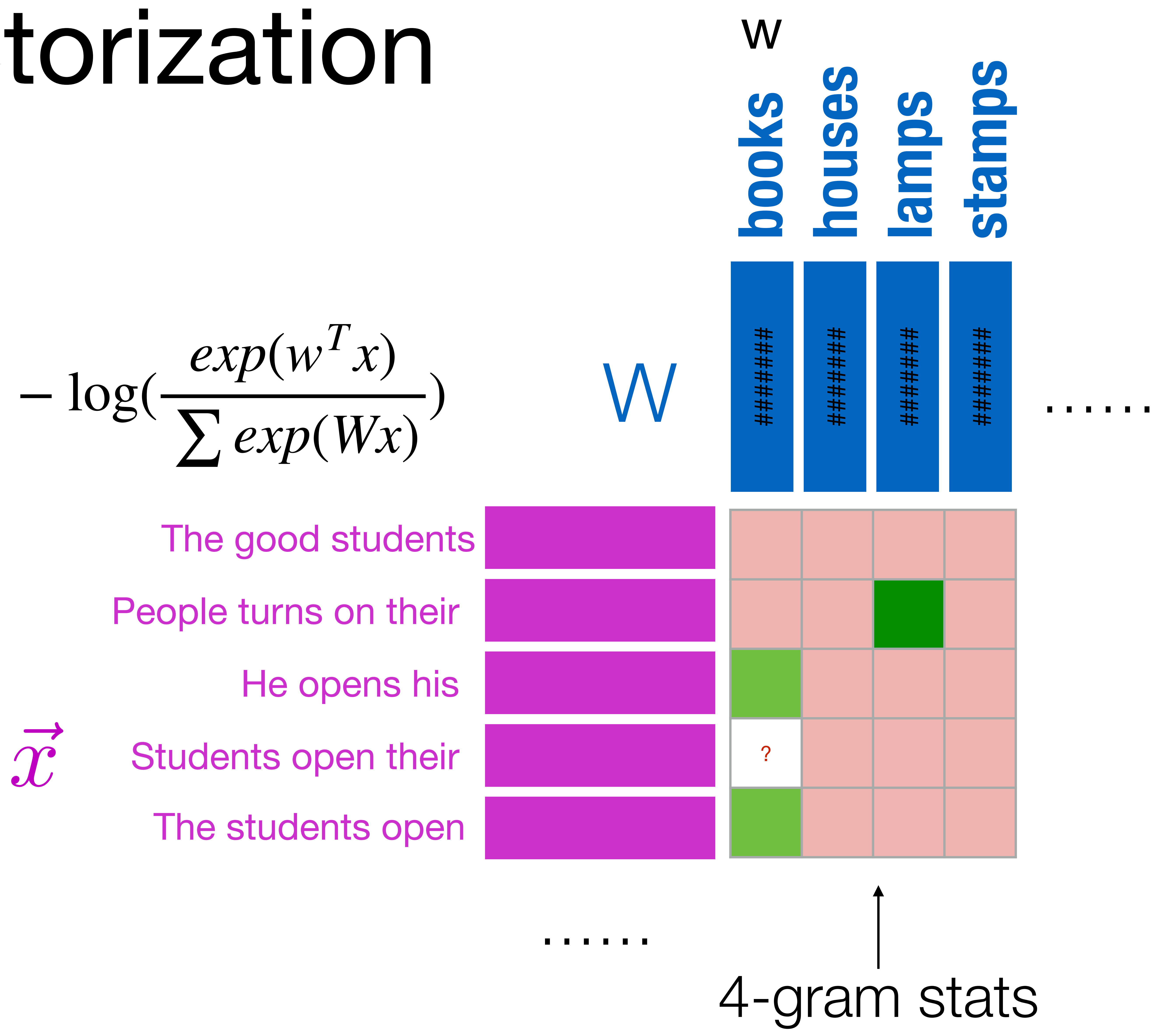
- **https://people.cs.umass.edu/~hschang/cs685/schedule.html**
  - Gradescope version might be outdated

- **3/3:** Quiz 2 due
  - Will release it soon
- **3/7**: Project proposals due
  - I have assigned every student into one group. Please start to work on project proposal asap
  - If you cannot reach all of your group members this week, please report the situation on Piazza
- **3/14**: HW 1 due
  - Will release it soon
  - If you have collected the dataset at CS 485, feel free to use it here
- **5/9**: Last day to submit extra credit
  - If the recording of the extra credits won't be available, I will provide some YouTube talk for you to watch.

Matrix Factorization

- Prob = Softmax(Wx)
- Loss = -log(Prob(w|x)) $= -\log\left(\frac{exp(w^T x)}{\sum exp(Wx)}\right)$
  - Assuming w is the word we actually observe

- Hallucination comes from errors in matrix factorization



Why softmax is called softmax?

$$z = \log(\sum exp(Wx)) = \log(y) \qquad \frac{dz}{dy} = \frac{1}{y}$$

$$y = \sum_j b_j$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = \frac{dz}{dy}\sum_j \frac{db_j}{dx} = \frac{dz}{dy}\sum_j \frac{db_j}{da_j}\frac{da_j}{dx}$$

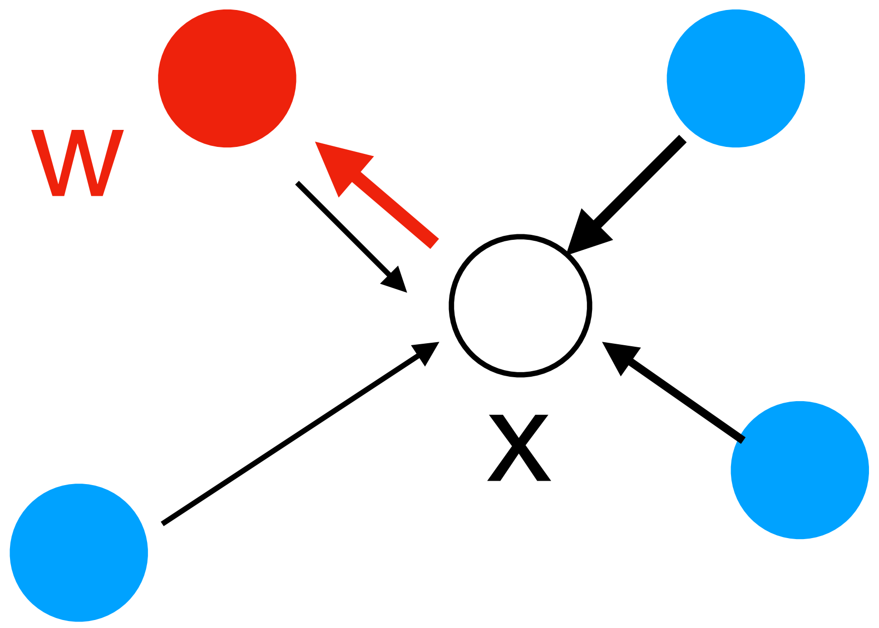
$$b_j = exp(a_j) \qquad \frac{db_j}{da_j} = exp(a_j)$$

$$= \frac{\sum_j exp(w_j^T x) w_j}{\sum_j exp(w_j^T x)} = \sum_j [Softmax(Wx)]_j w_j$$

$$a_j = w_j^T x \qquad \frac{da_j}{dx} = w_j$$

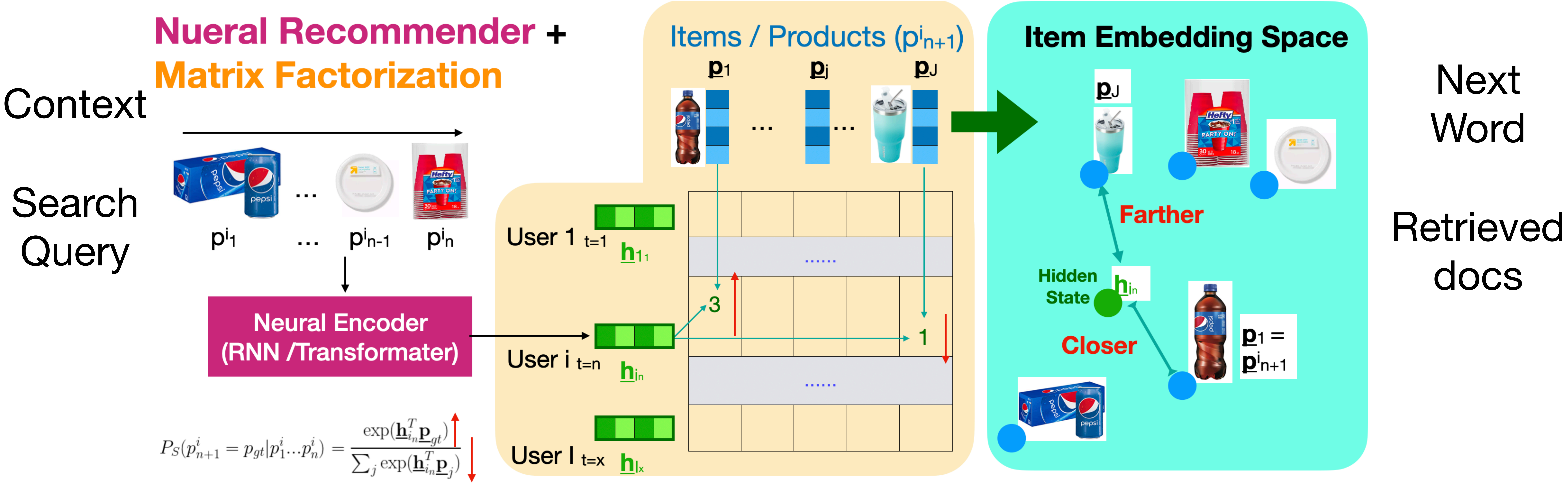
$$\frac{d - \log(\frac{exp(w^T x)}{\sum exp(Wx)})}{dx} = -\frac{dw^T x}{dx} + \frac{d\log(\sum exp(Wx))}{dx} \quad = -w + \sum_j [Softmax(Wx)]_j w_j$$

$$= -(1 - [Softmax(Wx)]_i)w + \sum_{j \neq i} [Softmax(Wx)]_j w_j$$



Matrix cookbook (https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf)

Intuitions of Predicting a Sequence

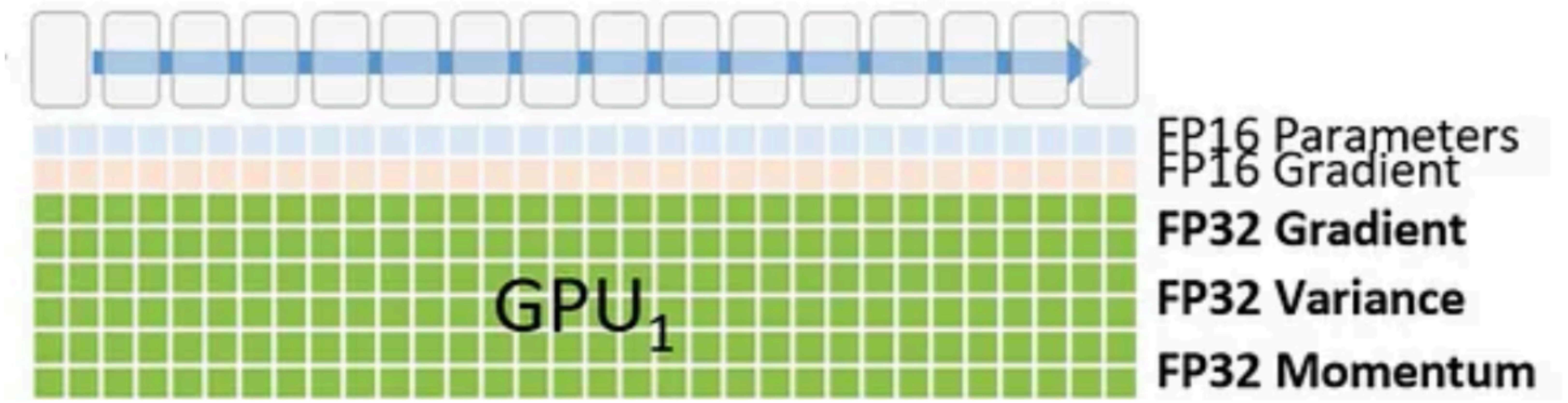


$$P_S(p^i_{n+1} = p_{gt}|p^i_1 \dots p^i_n) = \frac{\exp(\mathbf{h}^T_{in}\mathbf{p}_{gt})}{\sum_j \exp(\mathbf{h}^T_{in}\mathbf{p}_j)}$$

- Contrastive learning (c, w, all other w_j)
  - No good negative examples
- Optimal context hidden state is roughly the average of all its co-occurred word/product/document embeddings
  - Optimal word/product/document embedding is roughly the average of all its co-occurred context hidden state

Memory Usage in Optimization

3B LLM



6B Parameters

6B Gradient

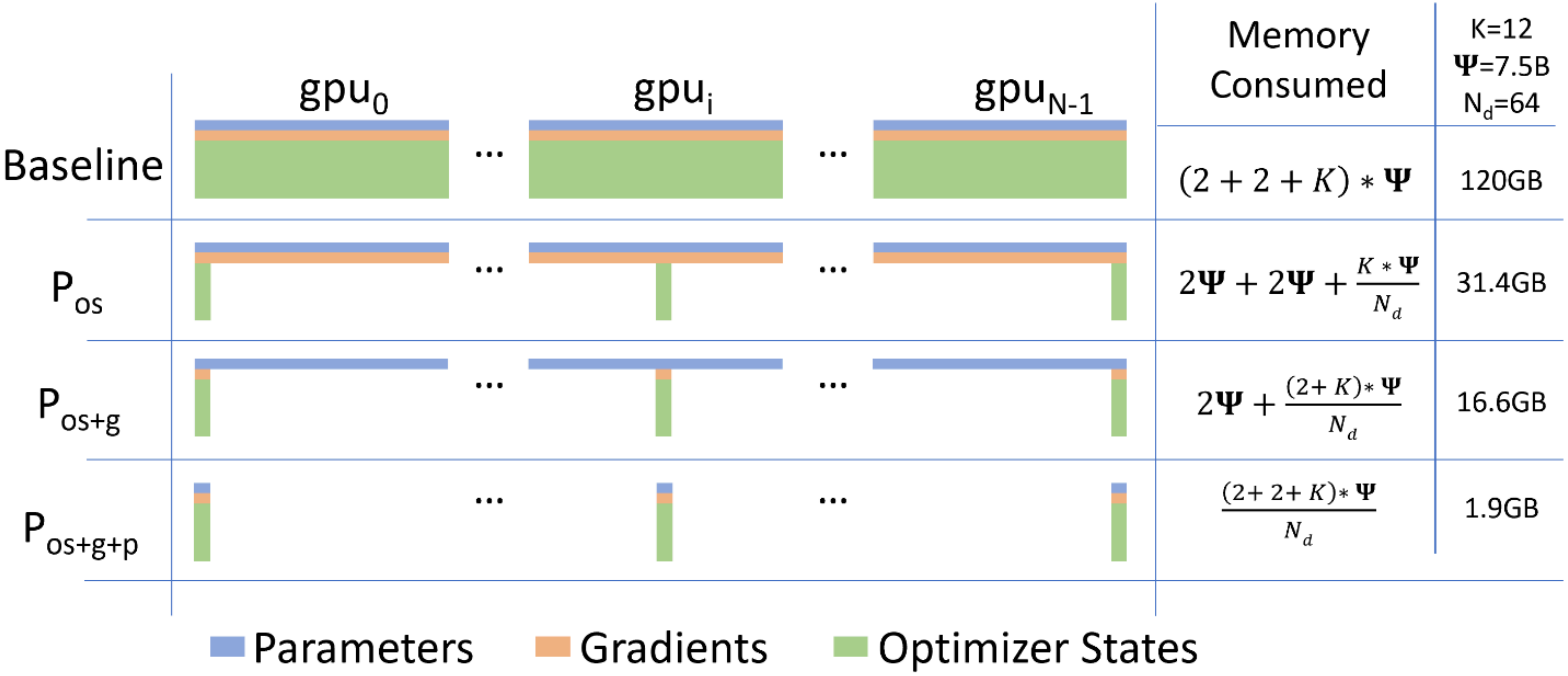
12B*3 Adam

3B * 16 = 48GB

We need some extra space to store the hidden state of the forward pass

Why? For computing backward pass

ZeRO



DeepSpeed (https://huggingface.co/docs/accelerate/en/usage_guides/deepspeed)

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models (https://arxiv.org/abs/1910.02054)

https://github.com/vllm-project/vllm https://github.com/unslothai/unsloth https://github.com/hiyouga/LLaMA-Factory

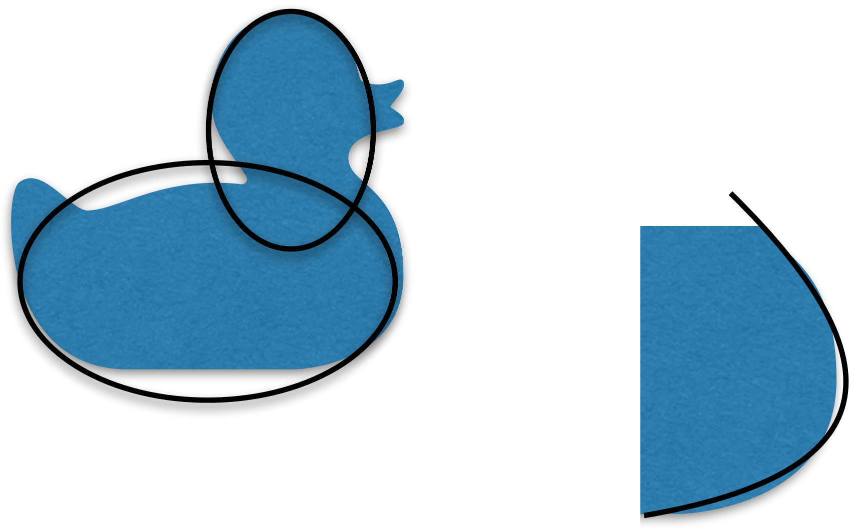
Self-Attention

Haw-Shiuan Chang

Task -> Loss -> Model -> Optimization

- Task:
  - Predict the next token
- Loss:
  - Maximal Likelihood / Cross-entropy
- Model:
  - Tables -> Neural Network -> **Transformer**

- Optimization:
  - Counting -> Gradient Descent



A RNN Language Model

output distribution

$\hat{y} = \text{softmax}(W_2 h^{(t)})$

hidden states

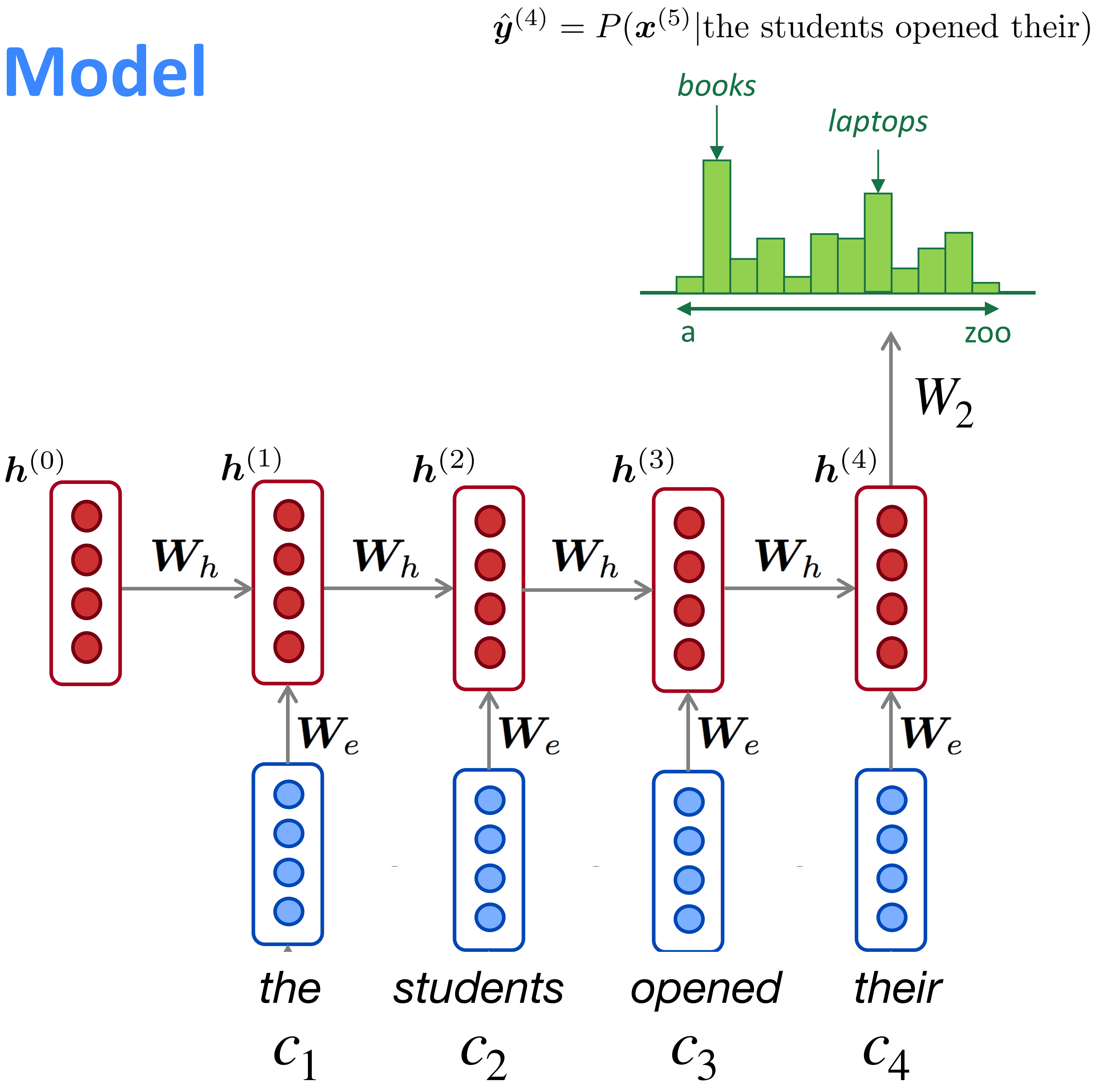
$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$

$h^{(0)}$ is initial hidden state!

word embeddings

c_1, c_2, c_3, c_4

Training time



A RNN Language Model

hidden states

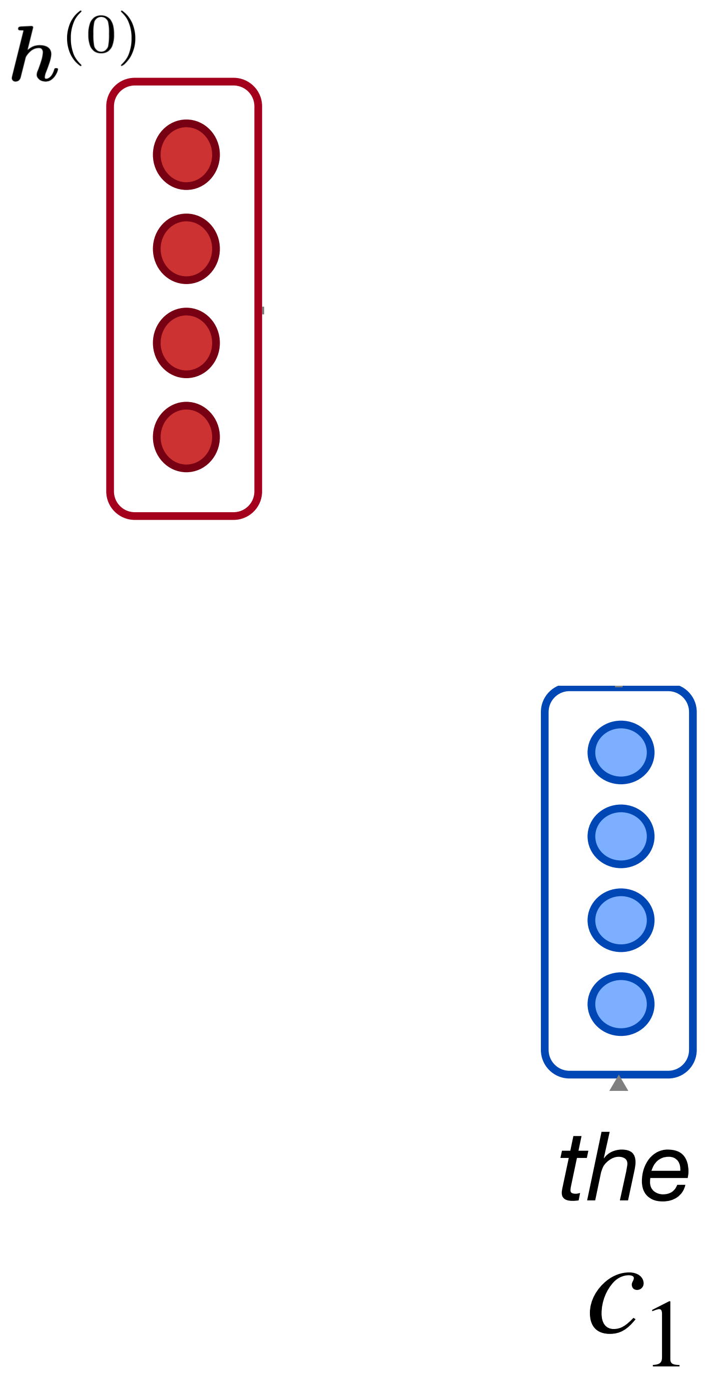
$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$

Testing time



A RNN Language Model

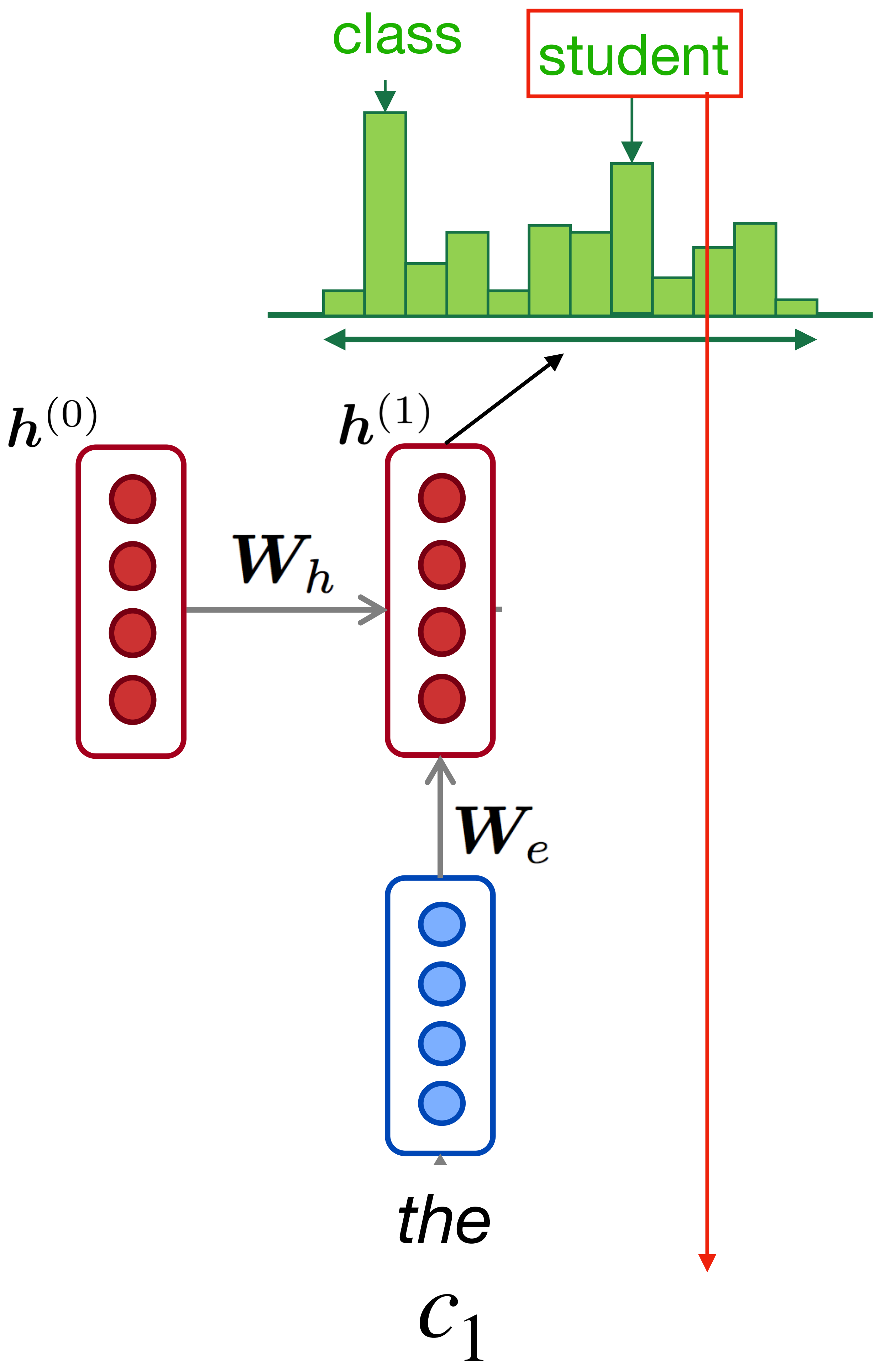
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

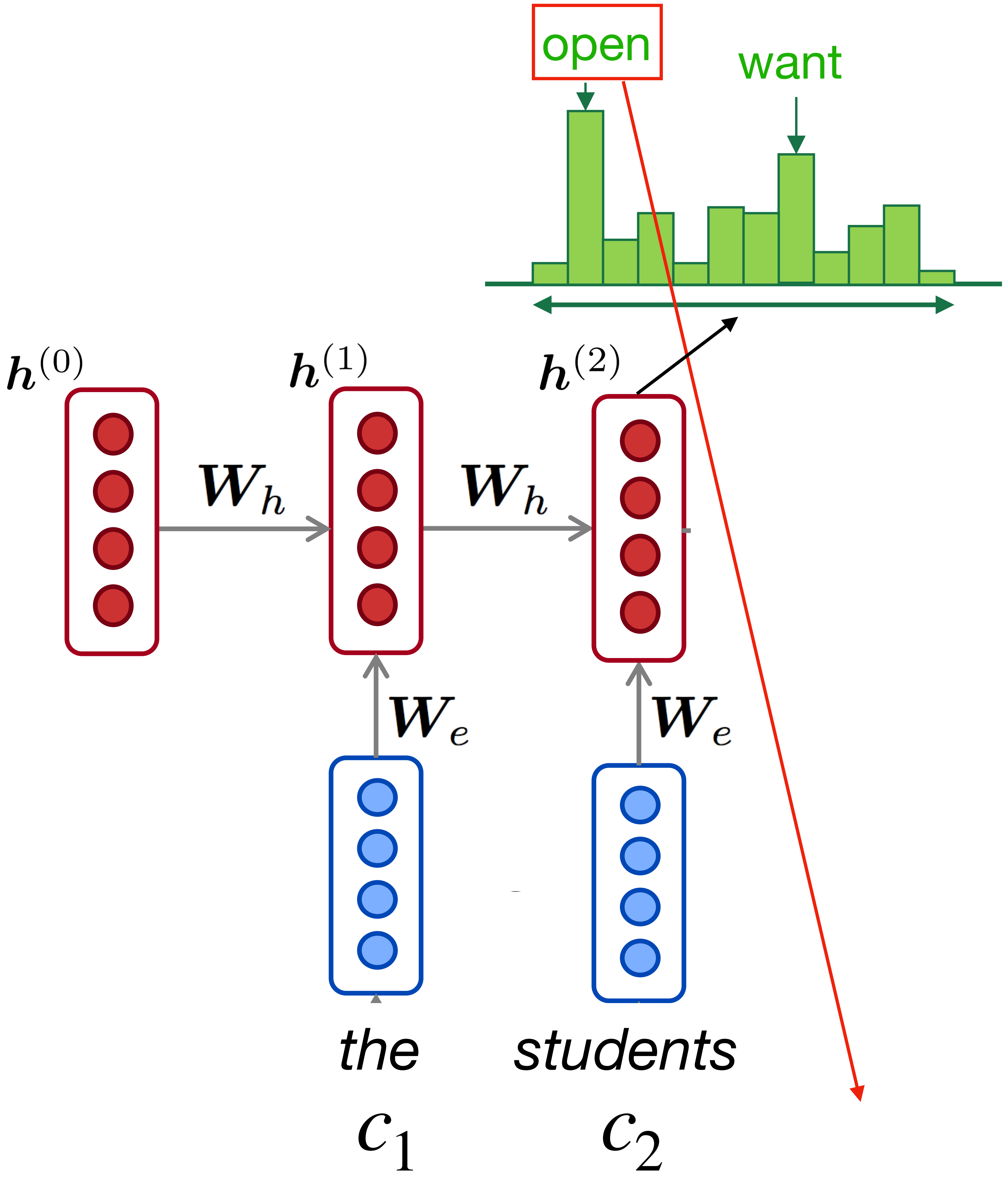
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

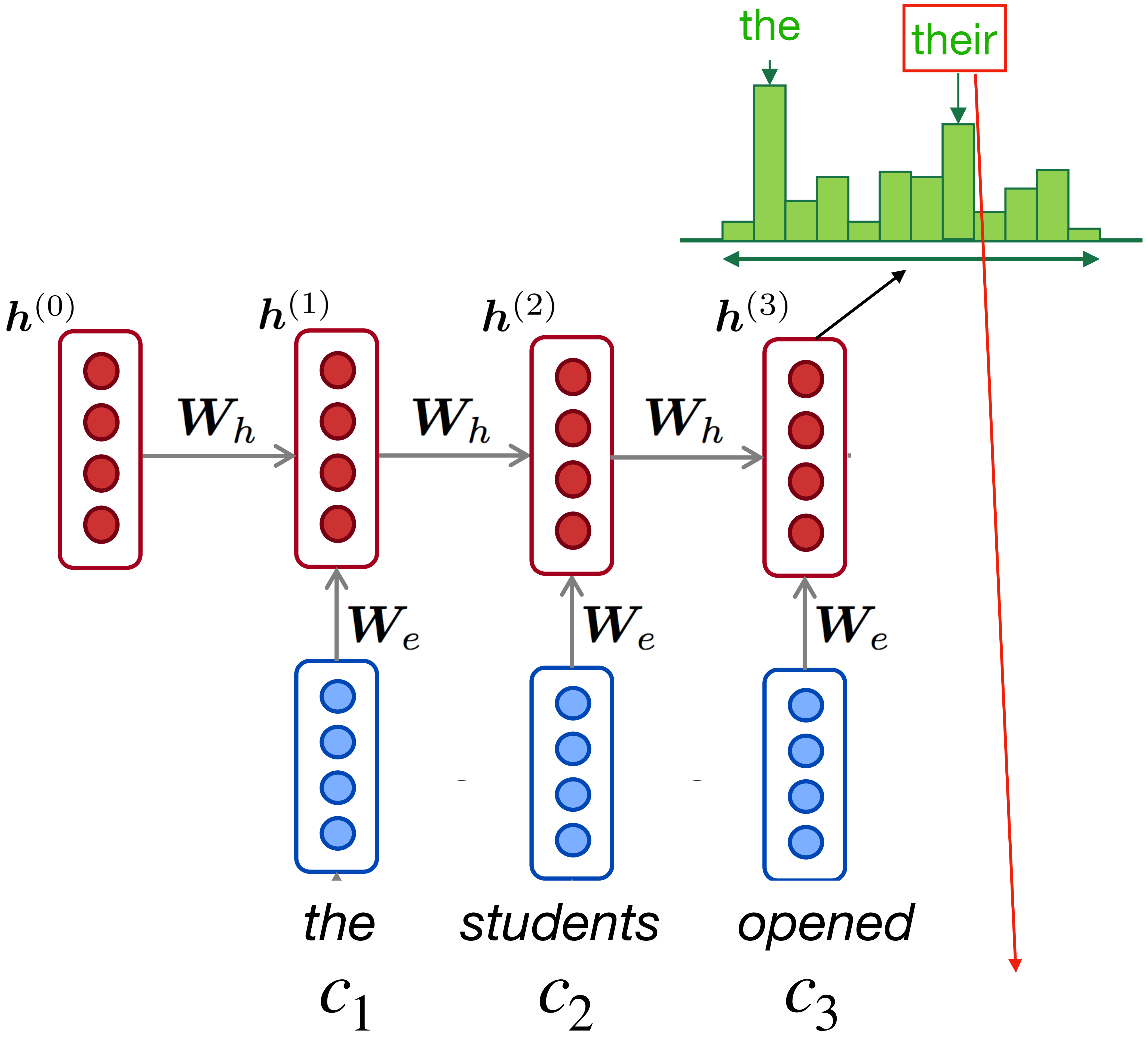
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

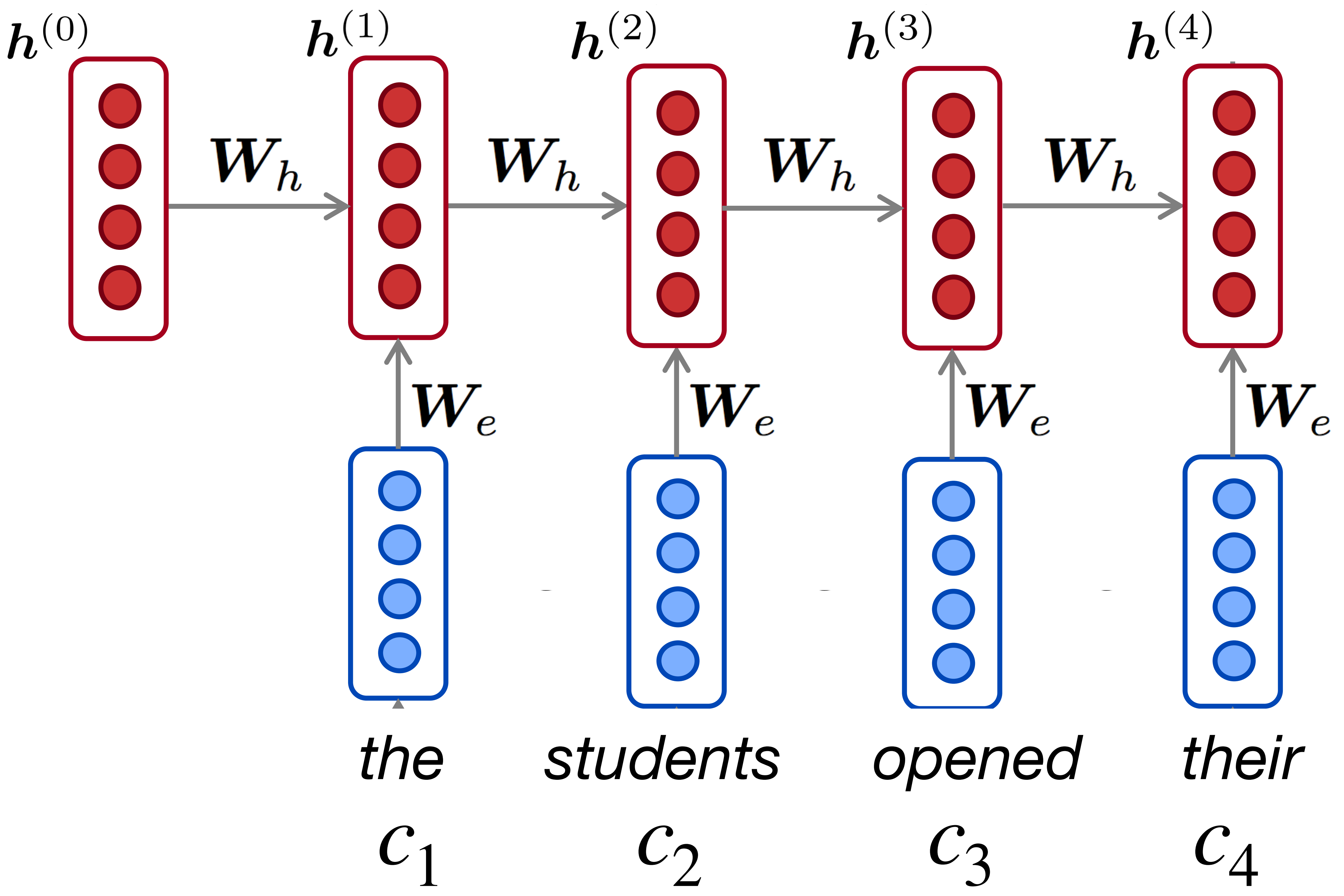
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



A RNN Language Model

output distribution

$$\hat{y} = \text{softmax}(W_2 h^{(t)})$$

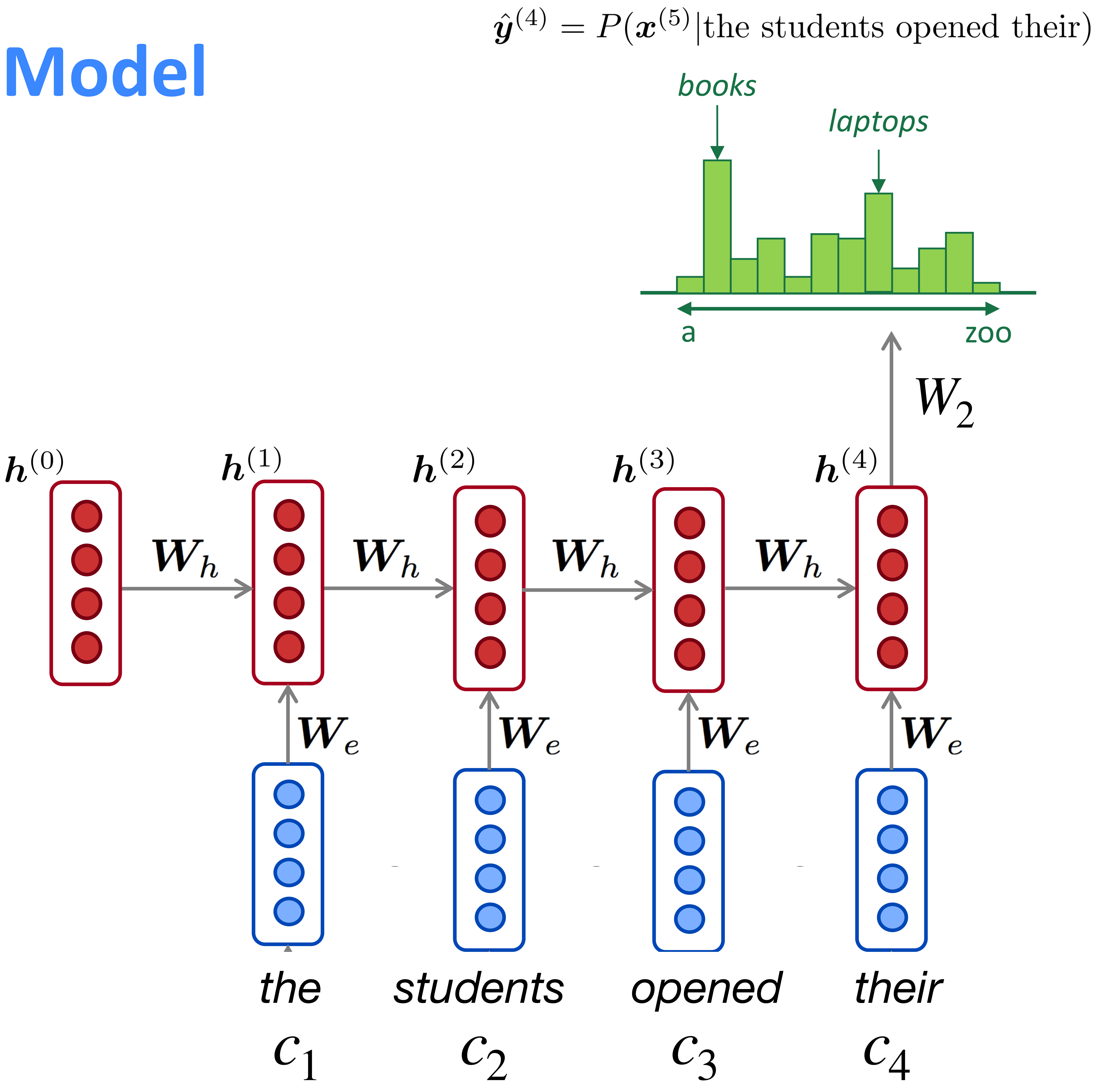
hidden states

$$h^{(t)} = f(W_h h^{(t-1)} + W_e c_t)$$

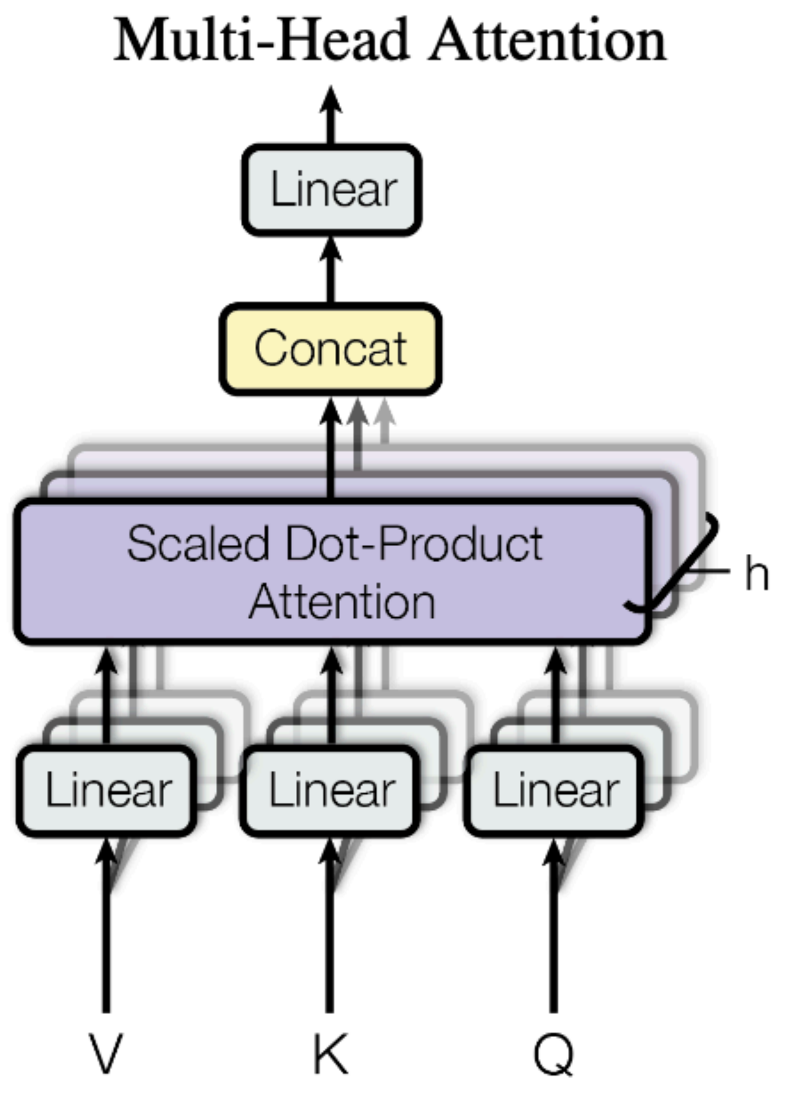
$h^{(0)}$ is initial hidden state!

word embeddings

$$c_1, c_2, c_3, c_4$$



Last Year Notes



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

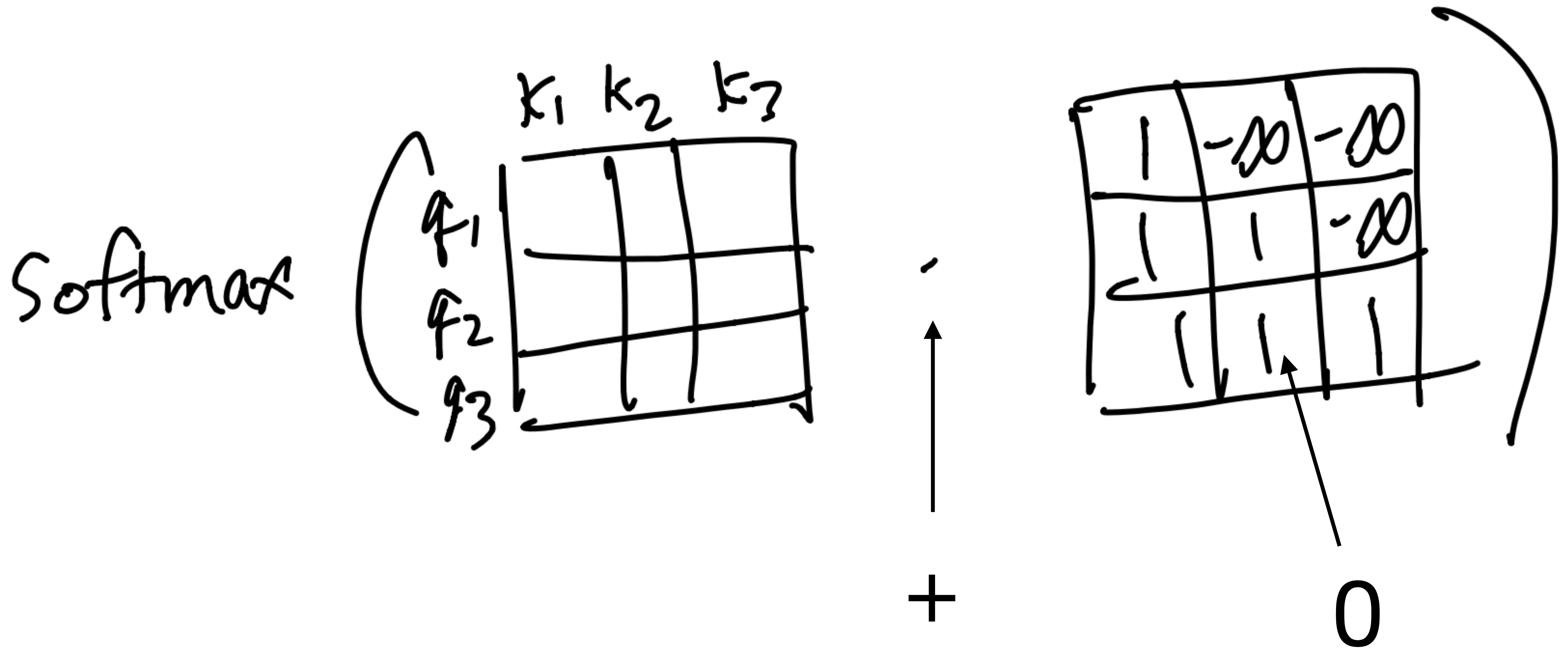
Typos Correction

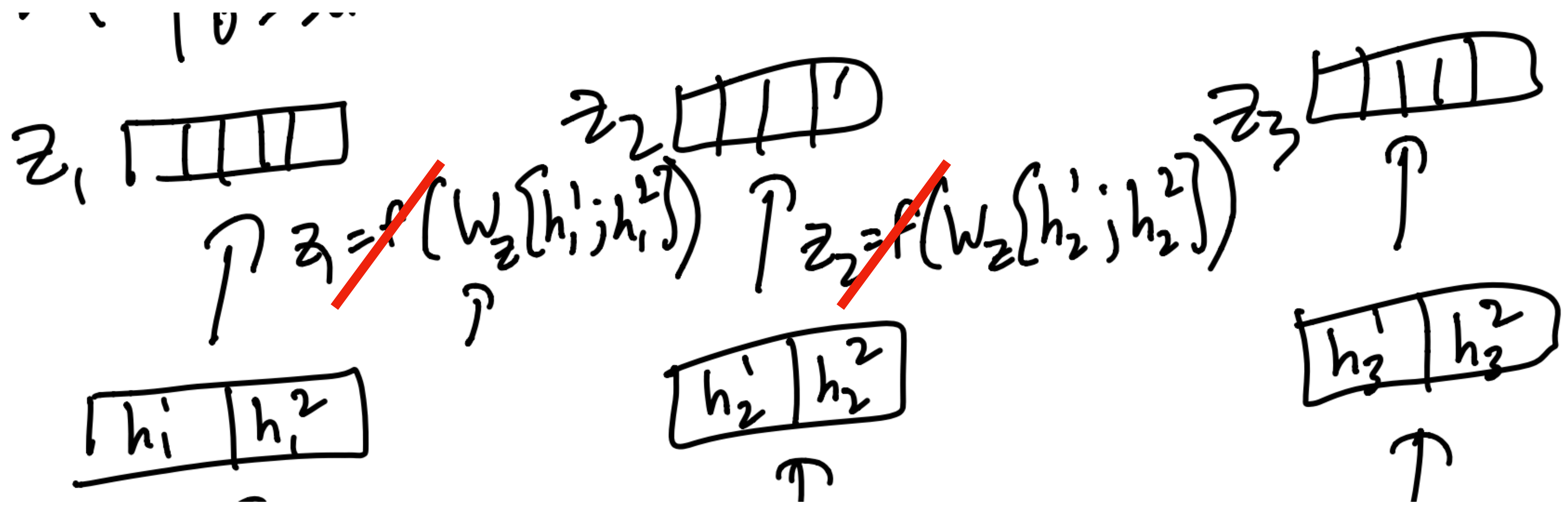
query : $q_1 = \cancel{f}(W_q c_1)$

key : $k_1 = \cancel{f}(W_k c_1)$

value : $v_1 = \cancel{f}(W_v c_1)$

$$\mathbf{q}_t = W^Q \mathbf{h}_t,$$
$$\mathbf{k}_t = W^K \mathbf{h}_t,$$
$$\mathbf{v}_t = W^V \mathbf{h}_t,$$

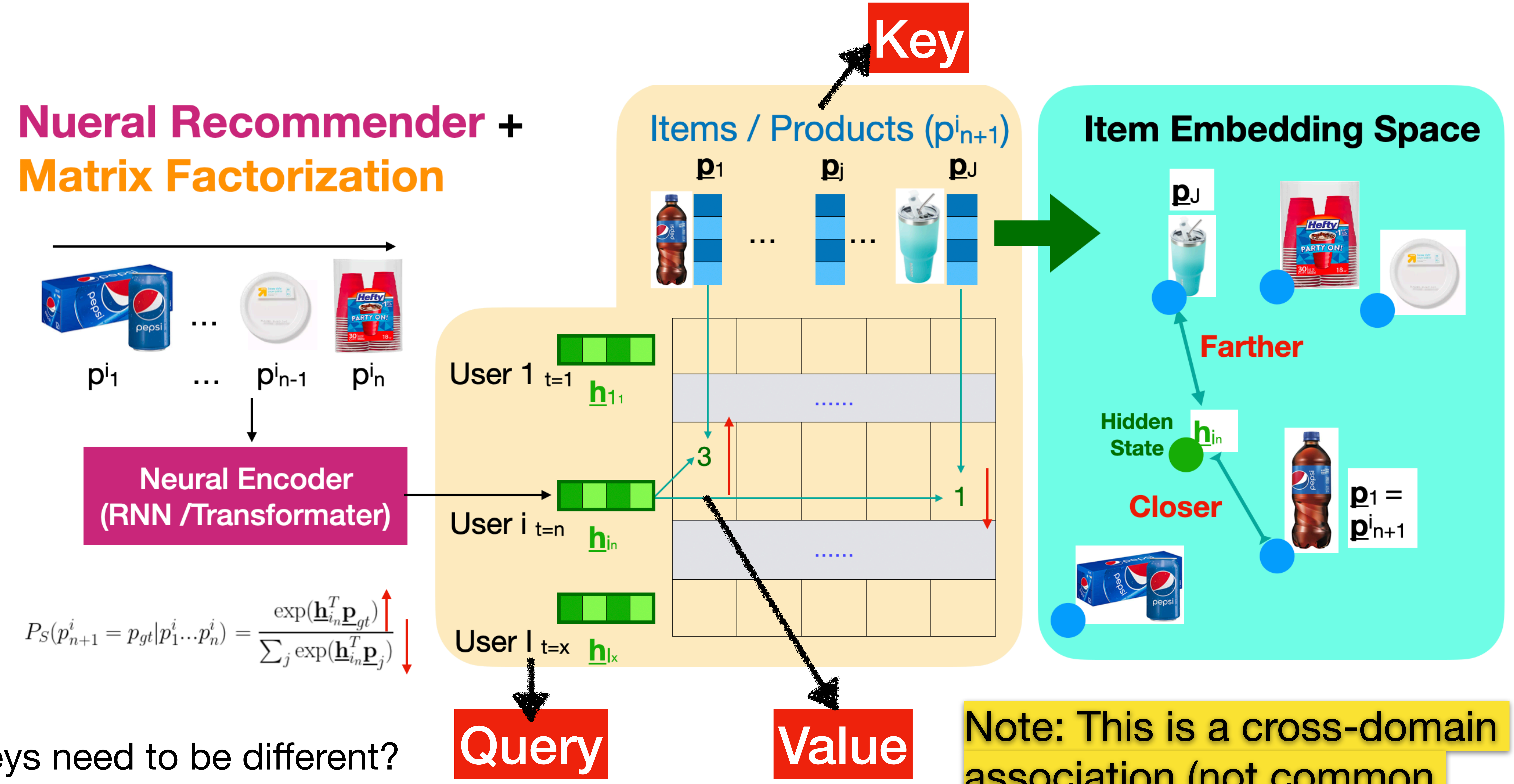




$$[\mathbf{q}_{t,1};\mathbf{q}_{t,2};\ldots;\mathbf{q}_{t,n_h}] = \mathbf{q}_t,$$
$$[\mathbf{k}_{t,1};\mathbf{k}_{t,2};\ldots;\mathbf{k}_{t,n_h}] = \mathbf{k}_t,$$
$$[\mathbf{v}_{t,1};\mathbf{v}_{t,2};\ldots;\mathbf{v}_{t,n_h}] = \mathbf{v}_t,$$
$$\mathbf{o}_{t,i} = \sum_{j=1}^{t} \mathrm{Softmax}_j\left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h}}\right)\mathbf{v}_{j,i},$$
$$\mathbf{u}_t = W^O[\mathbf{o}_{t,1};\mathbf{o}_{t,2};\ldots;\mathbf{o}_{t,n_h}],$$

https://arxiv.org/pdf/2405.04434

A Metaphor



$$P_S(p^i_{n+1} = p_{gt} | p^i_1 \ldots p^i_n) = \frac{\exp(\mathbf{h}_{i_n}^T \mathbf{p}_{gt})}{\sum_j \exp(\mathbf{h}_{i_n}^T \mathbf{p}_j)}$$

Why do queries and keys need to be different?

1. Matrix factorization's need
2. Diagonal value cannot be low

Note: This is a cross-domain association (not common knowledge among NLP researchers)

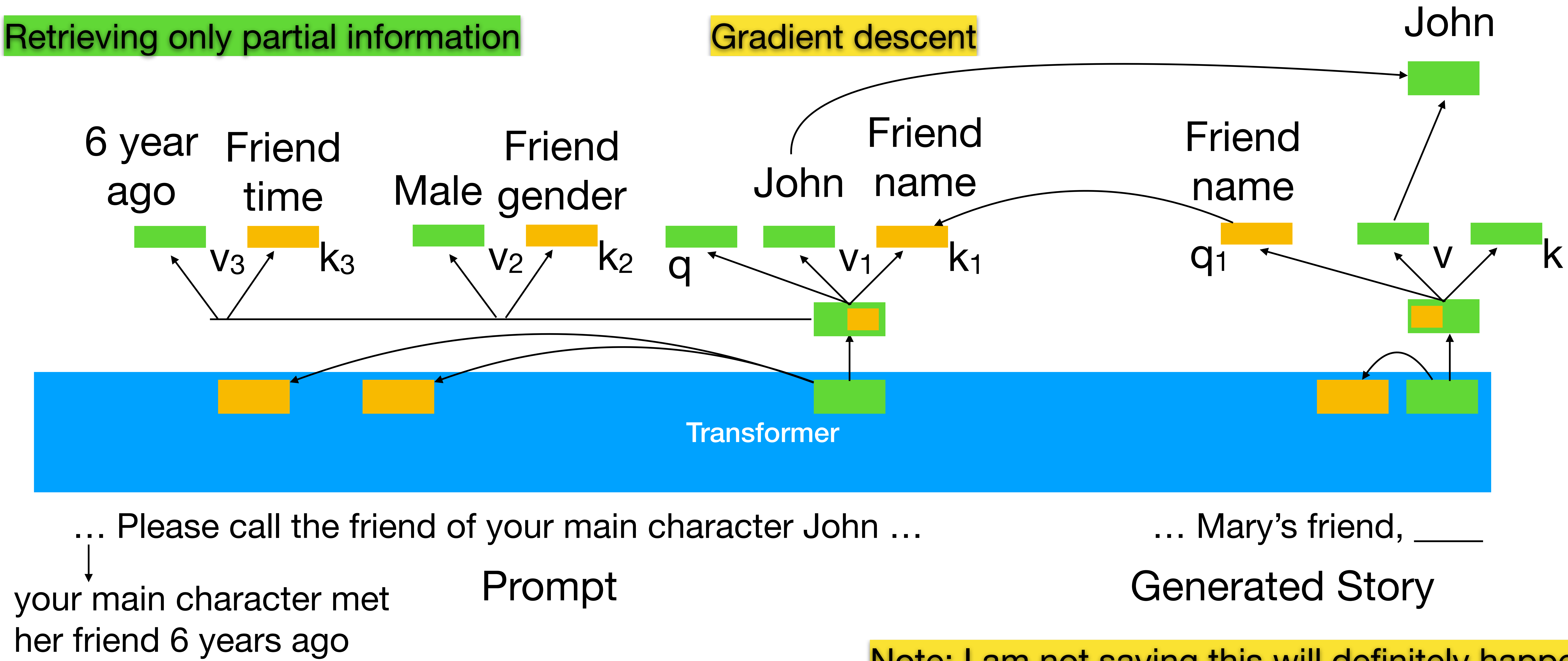
Self-Attention Example

Why do we need multiple heads?

Retrieving only partial information

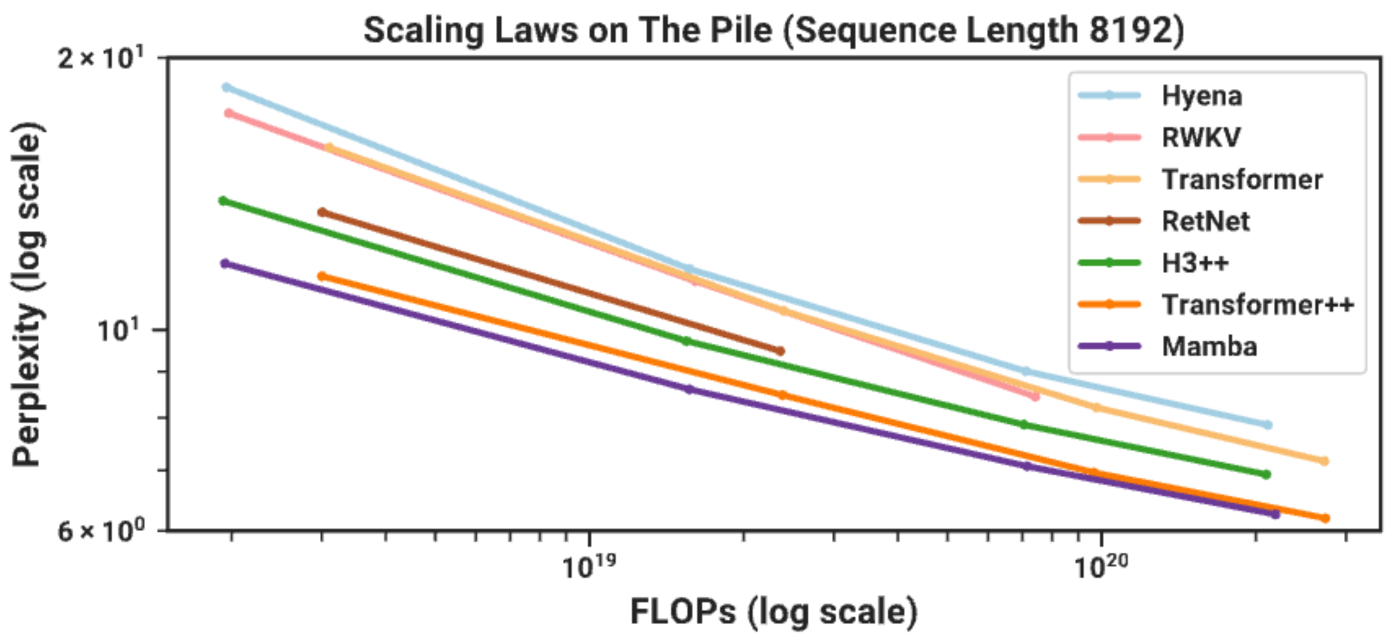
How do they learn these?

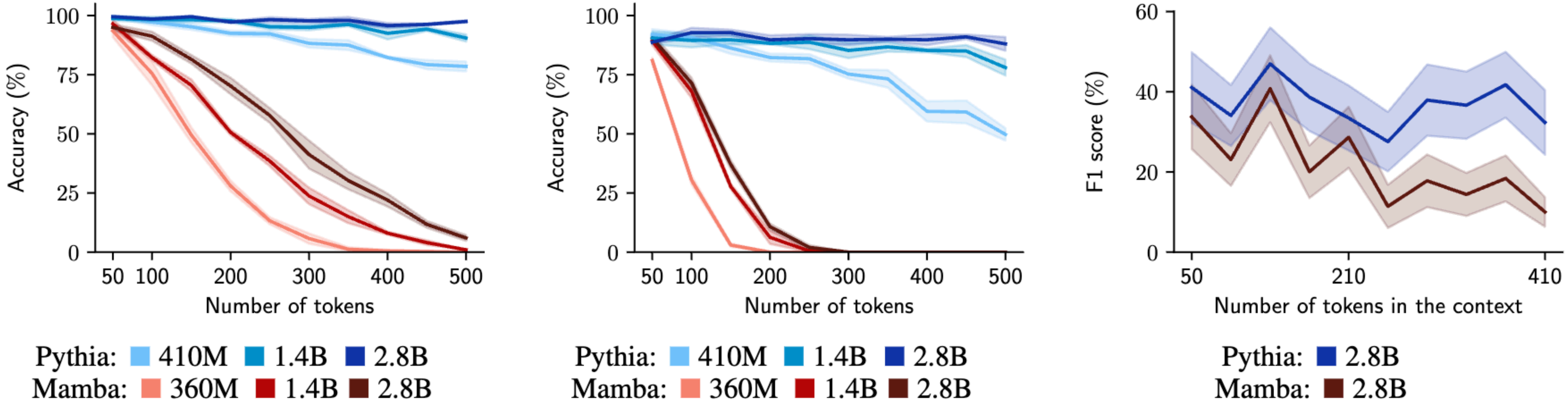
Gradient descent



Note: I am not saying this will definitely happen.

Why is Attention Effective?





(a) Copy: natural language strings

(b) Copy: shuffled strings

(c) Question answering (SQUAD)

Figure 7. **(a) Copy: natural language strings.** We compare pretrained models on their ability to copy natural language strings sampled from C4 of varying lengths and report string-level accuracy. The transformer models substantially outperform the GSSMs. **(b) Copy: shuffled strings.** To test whether it mattered that the strings were in natural language, we randomly shuffle the word order of the strings from the previous experiment. We find that this degrades performance, especially for the Mamba models. **(c) Question answering (SQUAD).** We compare Pythia and Mamba on a standard question answering dataset where we bin the dataset based on the length of the context paragraph. We find that Mamba performance decays more quickly with the length of the context.

Repeat After Me: Transformers are Better than State Space Models at Copying (https://arxiv.org/abs/2402.01032)

Why Same Weight for All Tokens?

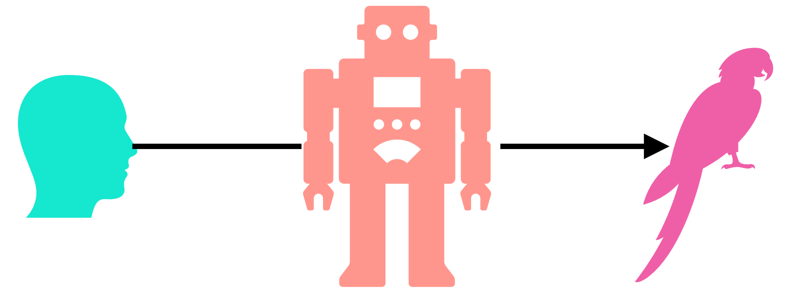
- Gradients naturally update less for easy examples

$$\frac{d - \log(\frac{exp(w^T x)}{\sum exp(Wx)})}{dx} = -\frac{dw^T x}{dx} + \frac{d \log(\sum exp(Wx))}{dx} = -(1 - \overset{1}{[Softmax(Wx)]_i})w + \overset{0}{\sum_{j\neq i} [Softmax(Wx)]_j w_j}$$

- Focus on easy tokens
  - Learning slowly
  - Focus too much on things you have known
- Focus on hard tokens
  - Easily affected by noise / unstable
  - Focus too much on things you cannot memorize

Humans learn more when starting from easier things

For LLMs, it doesn't matter



Midterm Example Question

Q1: RNN represents one sequence using one embedding, but Transformer also represents one sequence using one embedding. Why does Transformer mitigate the embedding bottleneck problem?

Q2: Which model is more expensive to train? RNN or Self-attention?

Multi-Head Latent Attention (MLA)

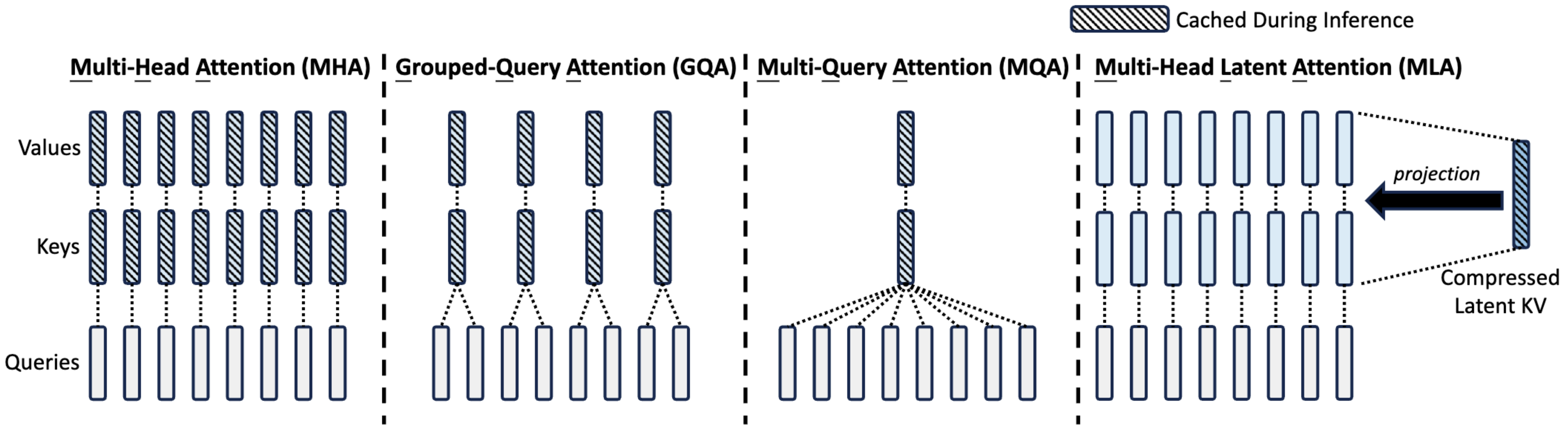
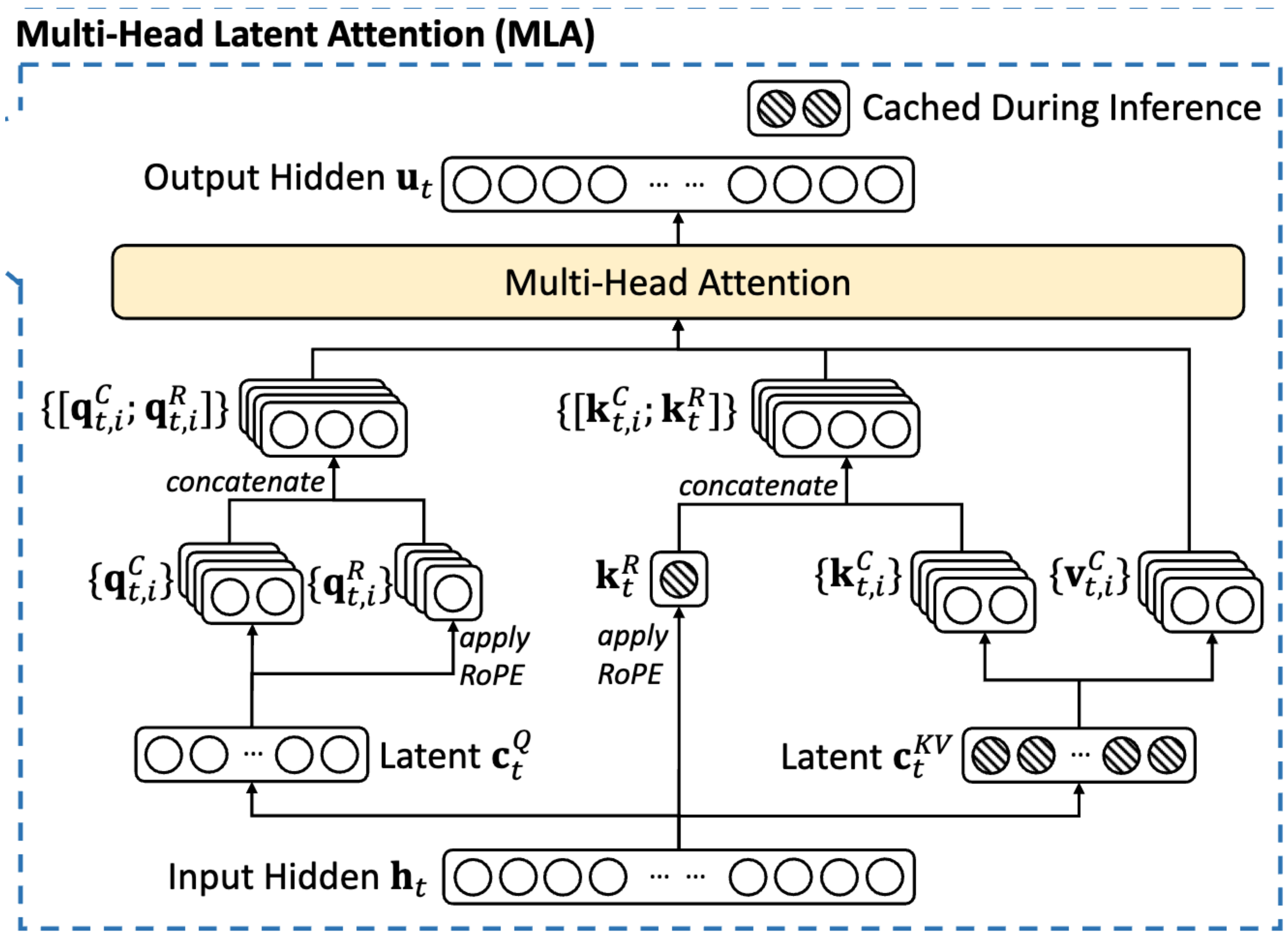


Figure 3 | Simplified illustration of Multi-Head Attention (MHA), Grouped-Query Attention (GQA), Multi-Query Attention (MQA), and Multi-head Latent Attention (MLA). Through jointly compressing the keys and values into a latent vector, MLA significantly reduces the KV cache during inference.

Deepseek V2 (https://arxiv.org/pdf/2405.04434)

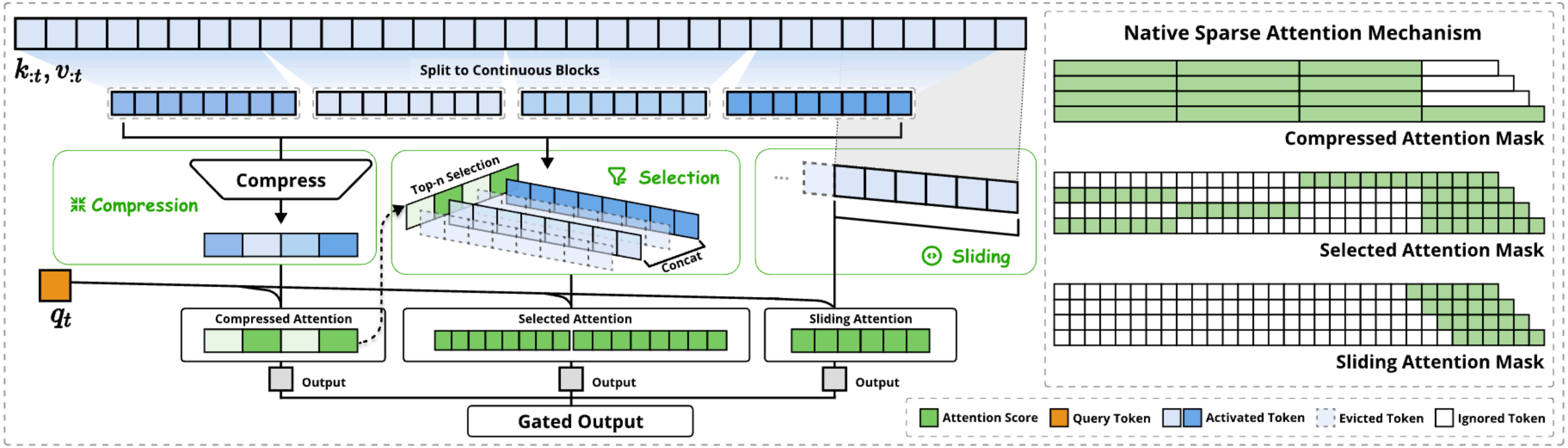
Multi-Head Latent Attention (MLA)

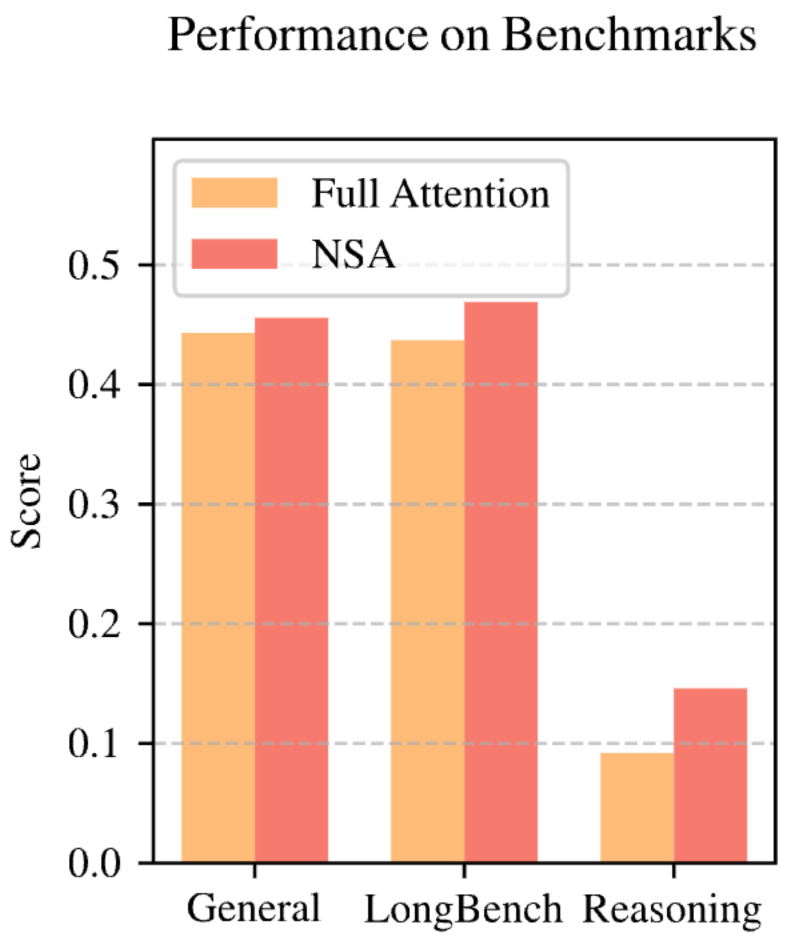


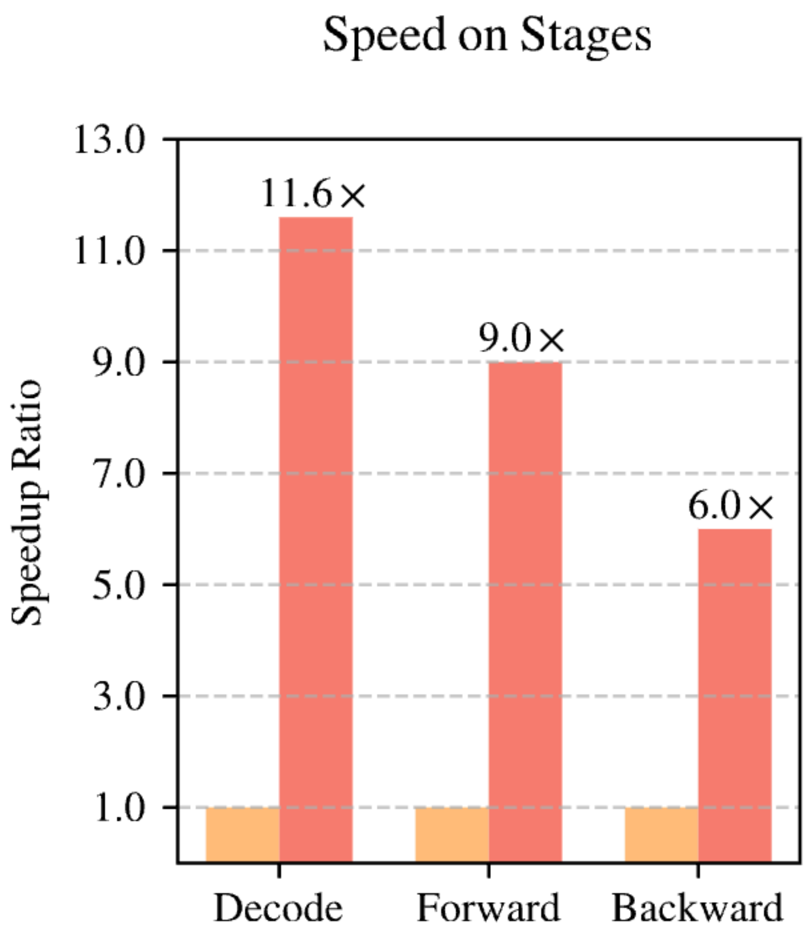
Attention Mechanism	KV Cache per Token (# Element)	Capability
Multi-Head Attention (MHA)	$2n_h d_h l$	Strong
Grouped-Query Attention (GQA)	$2n_g d_h l$	Moderate
Multi-Query Attention (MQA)	$2d_h l$	Weak
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2}d_h l$	Stronger

Table 1 | Comparison of the KV cache per token among different attention mechanisms. n_h denotes the number of attention heads, d_h denotes the dimension per attention head, l denotes the number of layers, n_g denotes the number of groups in GQA, and d_c and d_h^R denote the KV compression dimension and the per-head dimension of the decoupled queries and key in MLA, respectively. The amount of KV cache is measured by the number of elements, regardless of the storage precision. For DeepSeek-V2, d_c is set to $4d_h$ and d_h^R is set to $\frac{d_h}{2}$. So, its KV cache is equal to GQA with only 2.25 groups, but its performance is stronger than MHA.

Hierarchical Attention







Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention (https://arxiv.org/pdf/2502.11089)

(From deepseek AI)

Cross-Entropy Review

model's predicted distribution q

training data distribution: $\Rightarrow$

def of cross entropy

$$-\sum_{w \in V} p(w) \log q(w)$$

1 when w = books
0 otherwise

Cross-Entropy and Perplexity

$$
\begin{array}{llc}
 & w_t = a & q(w_t = a \,|\, w_{<t}) = 1/3 \\
w_1 \ldots w_{t-1} \rightarrow & w_t = b & 1/3 \\
 & w_t = c & 1/3 \\
w_{<t} & w_t = d \ldots & 0
\end{array}
$$

$$H(q,p) = -\sum_x q(w_t = x \,|\, w_{<t}) \log p(w_t = x \,|\, w_{<t}) = -\frac{1}{3} \sum_{x=a,b,c} \log p(w_t = x \,|\, w_{<t})$$

$$perplexity = \exp(H(q,p))$$

Negative log likelihood

Cross-Entropy, Entropy, and KL Divergence

$$H(q,p) = H(q) + D_{KL}(q||p)$$

- Cross-Entropy

$$H(q,p) = -\sum_x q(w_t = x | w_{<t}) \log p(w_t = x | w_{<t})$$

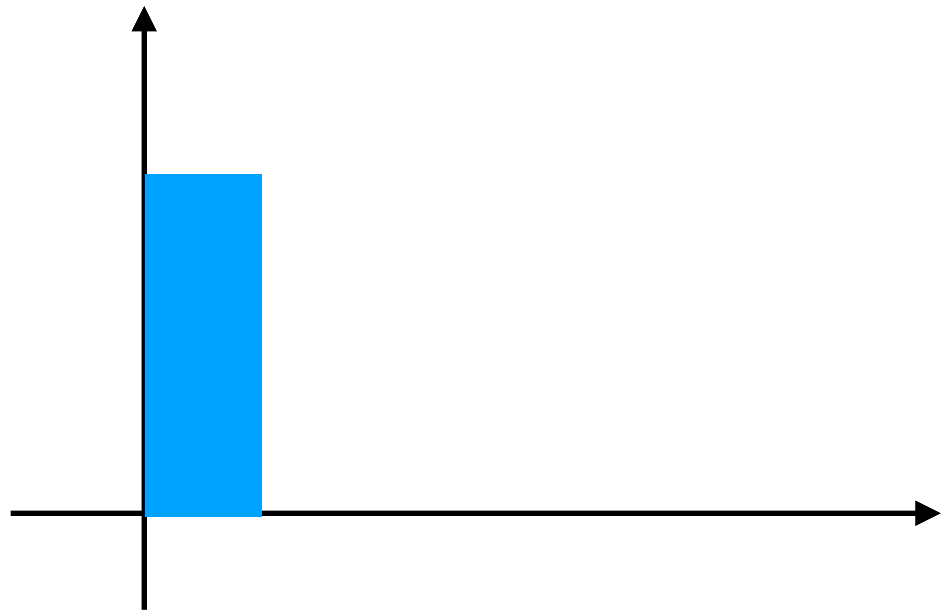
- Entropy

$$H(q) = -\sum_x q(w_t = x | w_{<t}) \log q(w_t = x | w_{<t})$$

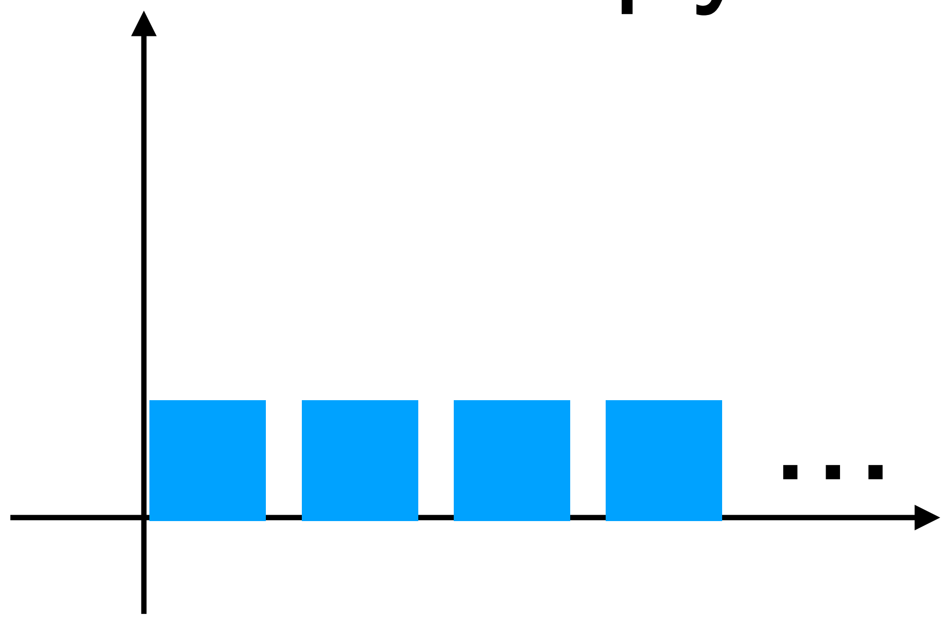
- KL Divergence

$$D_{KL}(q||p) = -\sum_x q(w_t = x | w_{<t}) \log \frac{p(w_t = x | w_{<t})}{q(w_t = x | w_{<t})}$$

Entropy = 0



Largest Entropy



- Cross-Entropy = KL Divergence when entropy is 0