CS182: Introduction to Machine Learning – Transformer LMs

Yujiao Shi

SIST, ShanghaiTech

Spring, 2025

LARGE LANGUAGE MODELS

How large are LLMs?

上海科技大学
ShanghaiTech University

Comparison of some recent **large language models** (LLMs)

Model	Creators	Year of release	Training Data (# tokens)	Model Size (# parameters)
GPT-2	OpenAI	2019	~10 billion (40Gb)	1.5 billion
GPT-3	OpenAI	2020	300 billion	175 billion
PaLM	Google	2022	780 billion	540 billion
Chinchilla	DeepMind	2022	1.4 trillion	70 billion
LaMDA (cf. Bard)	Google	2022	1.56 trillion	137 billion
LLaMA	Meta	2023	1.4 trillion	65 billion
LLaMA-2	Meta	2023	2 trillion	70 billion
GPT-4	OpenAI	2023	?	? (1.76 trillion)
Gemini (Ultra)	Google	2023	?	? (1.5 trillion)
LLaMA-3	Meta	2024	15 trillion	405 billion

3

What is ChatGPT?

- ChatGPT is a large (in the sense of having many parameters) language model, fine-tuned to be a dialogue agent
- The base language model is GPT-3.5 which was trained on a large quantity of text

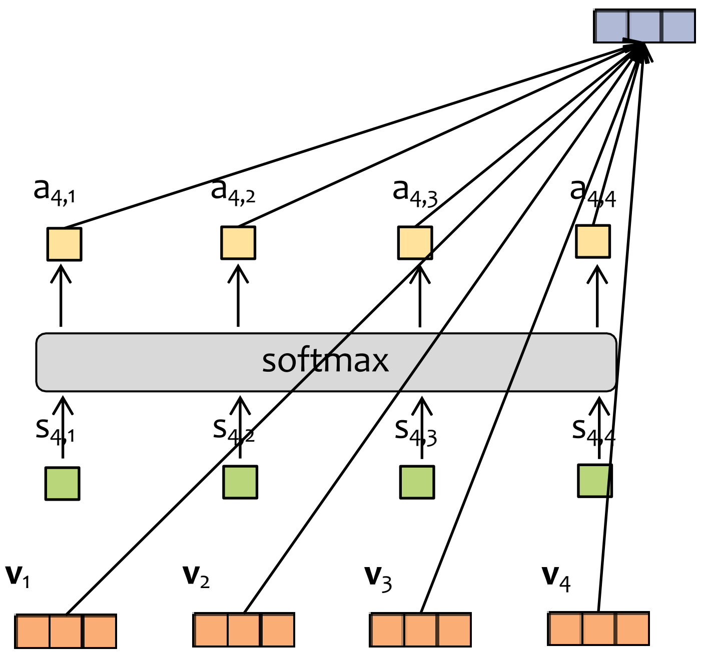
Transformer Language Models

MODEL: GPT

5

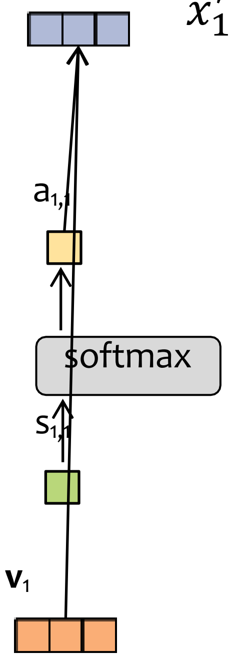
Attention

上海科技大学
ShanghaiTech University



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

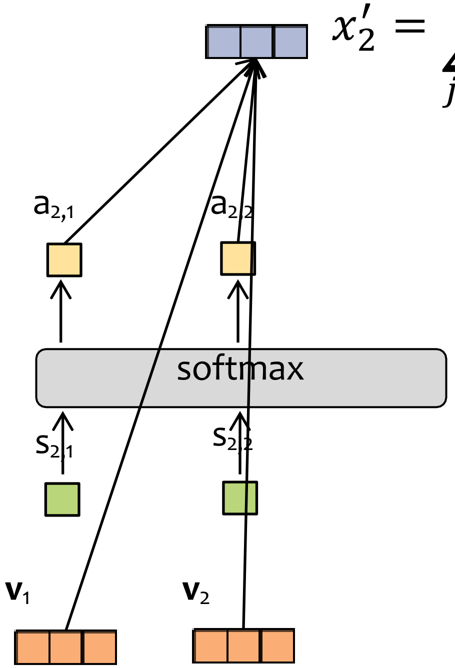
Attention



$$x'_1 = \sum_{j=1}^{1} a_{1,j} v_j$$

Attention

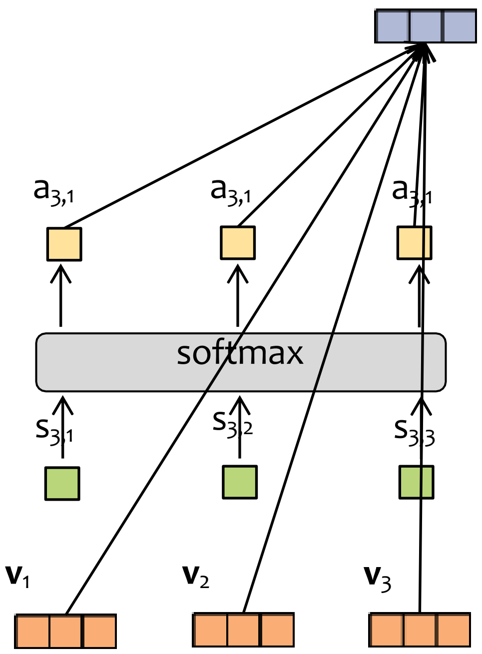
上海科技大学 ShanghaiTech University



$$x'_2 = \sum_{j=1}^{2} a_{2,j} v_j$$

Attention

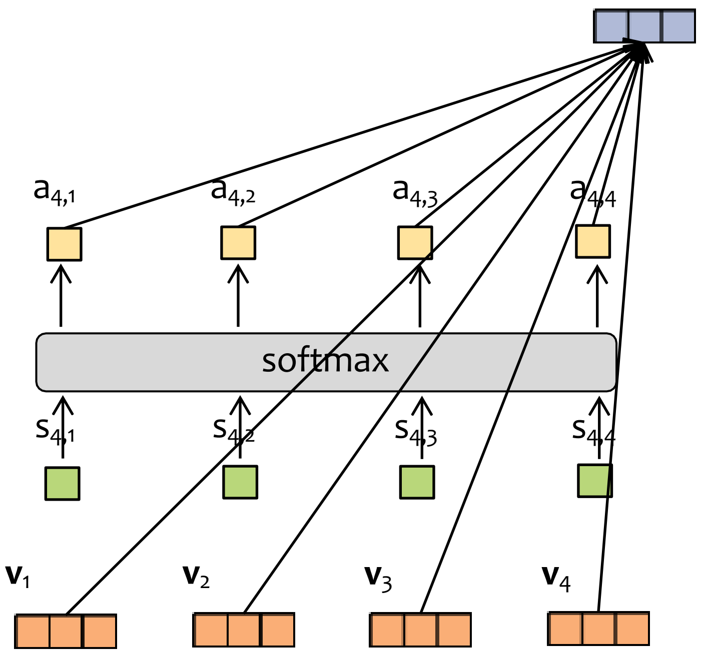
上海科技大学
ShanghaiTech University



$$x'_3 = \sum_{j=1}^{3} a_{3,j} v_j$$

Attention

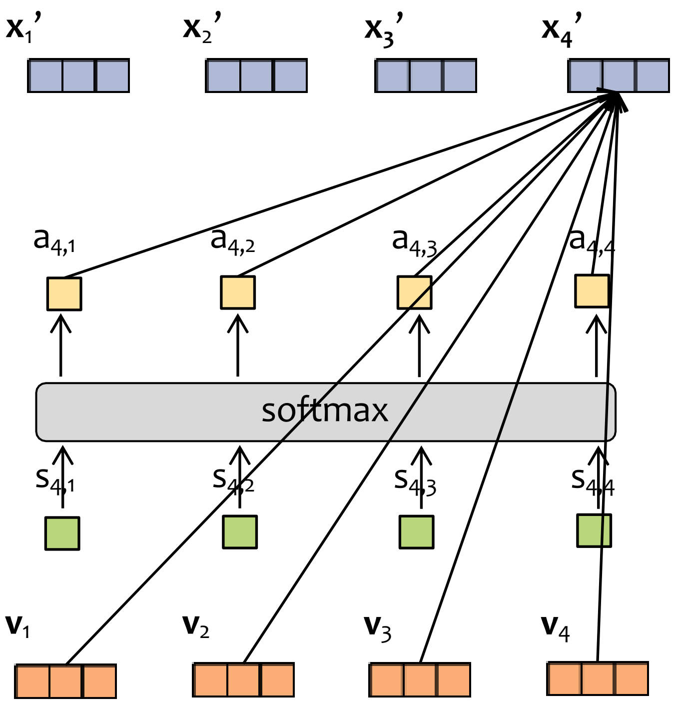
上海科技大学
ShanghaiTech University



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

Attention

上海科技大学
ShanghaiTech University



$$x'_t = \sum_{j=1}^{t} a_{t,j} v_j$$

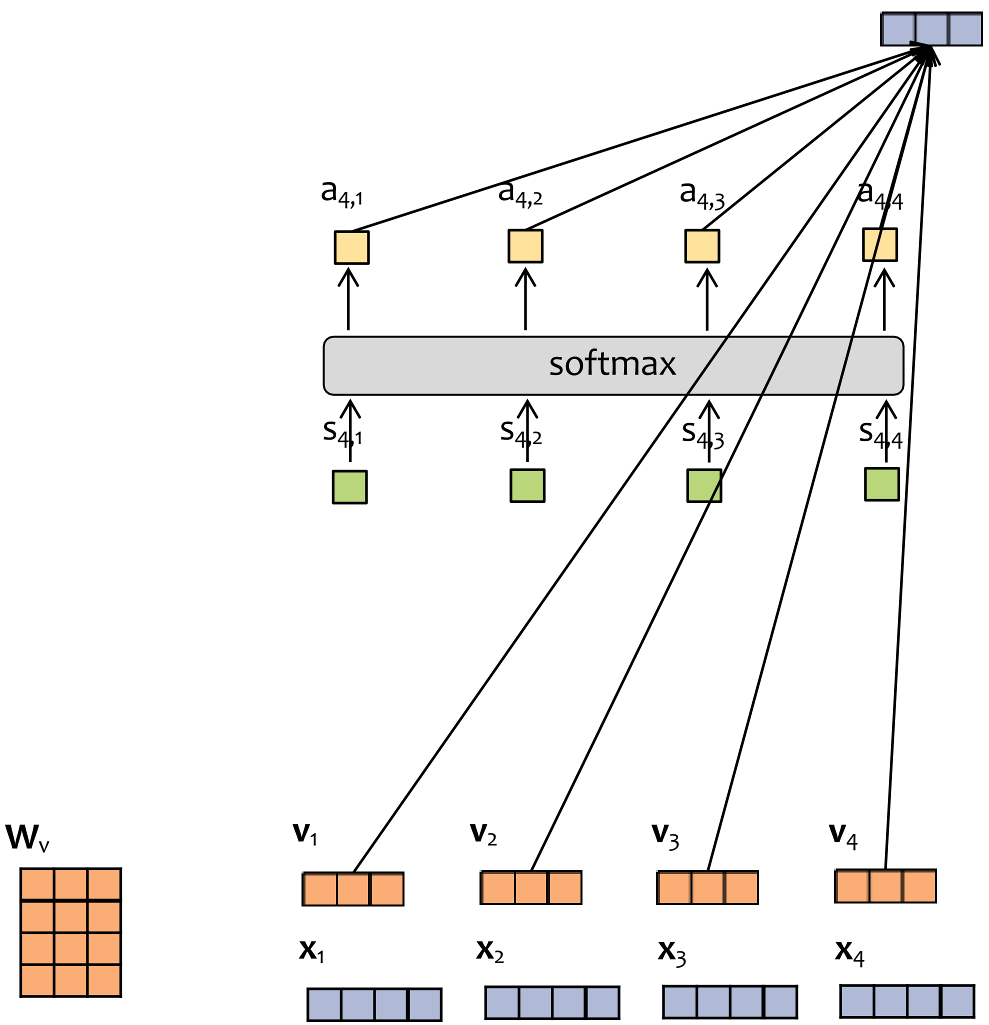
attention weights

scores

values

Scaled Dot-Product Attention

上海科技大学
ShanghaiTech University



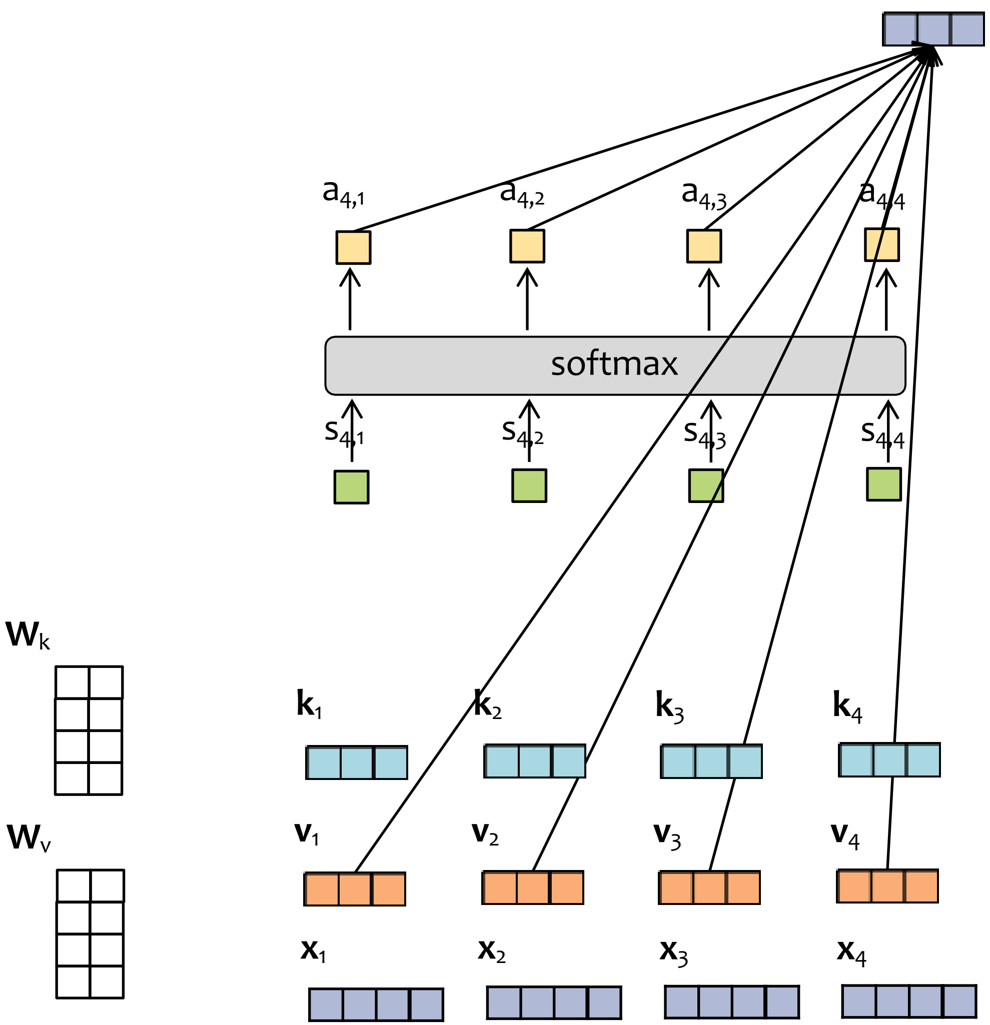
$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$

values

Scaled Dot-Product Attention

上海科技大学
ShanghaiTech University



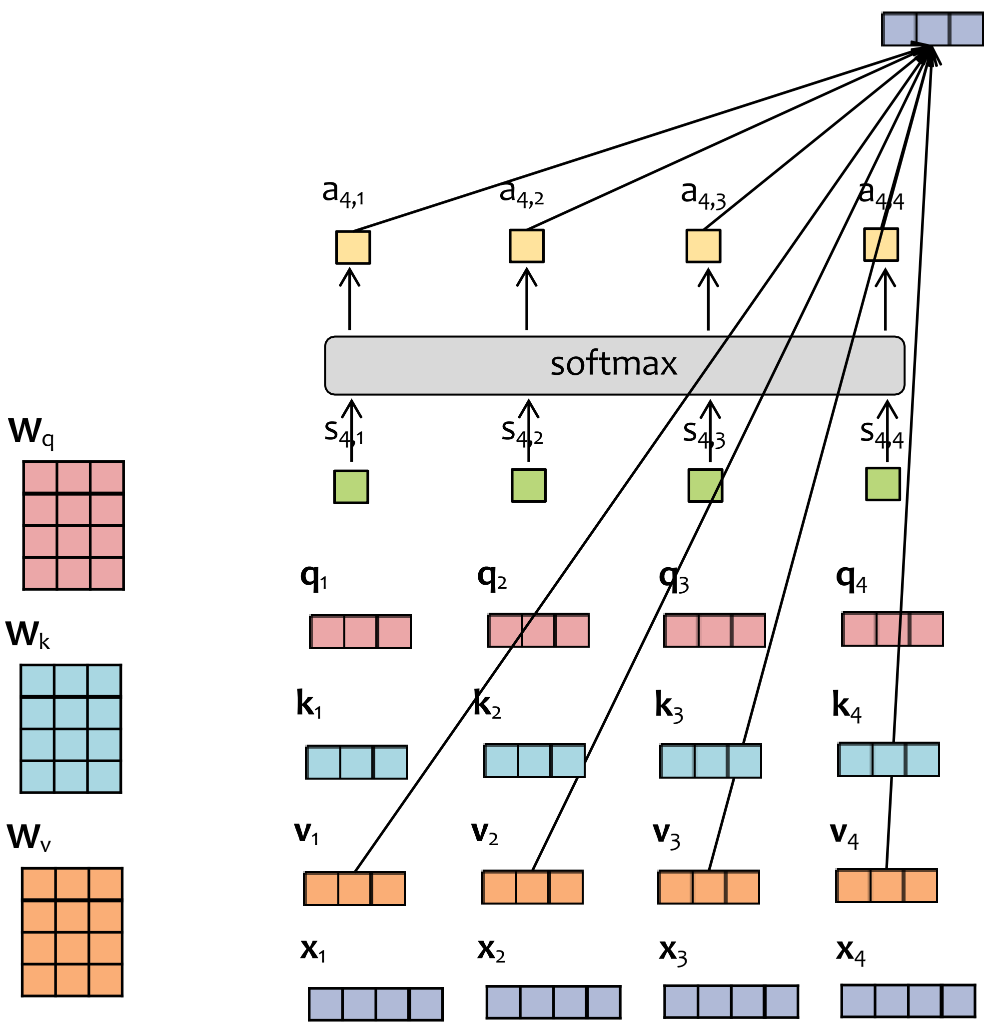
$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Scaled Dot-Product Attention

上海科技大学 ShanghaiTech University



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

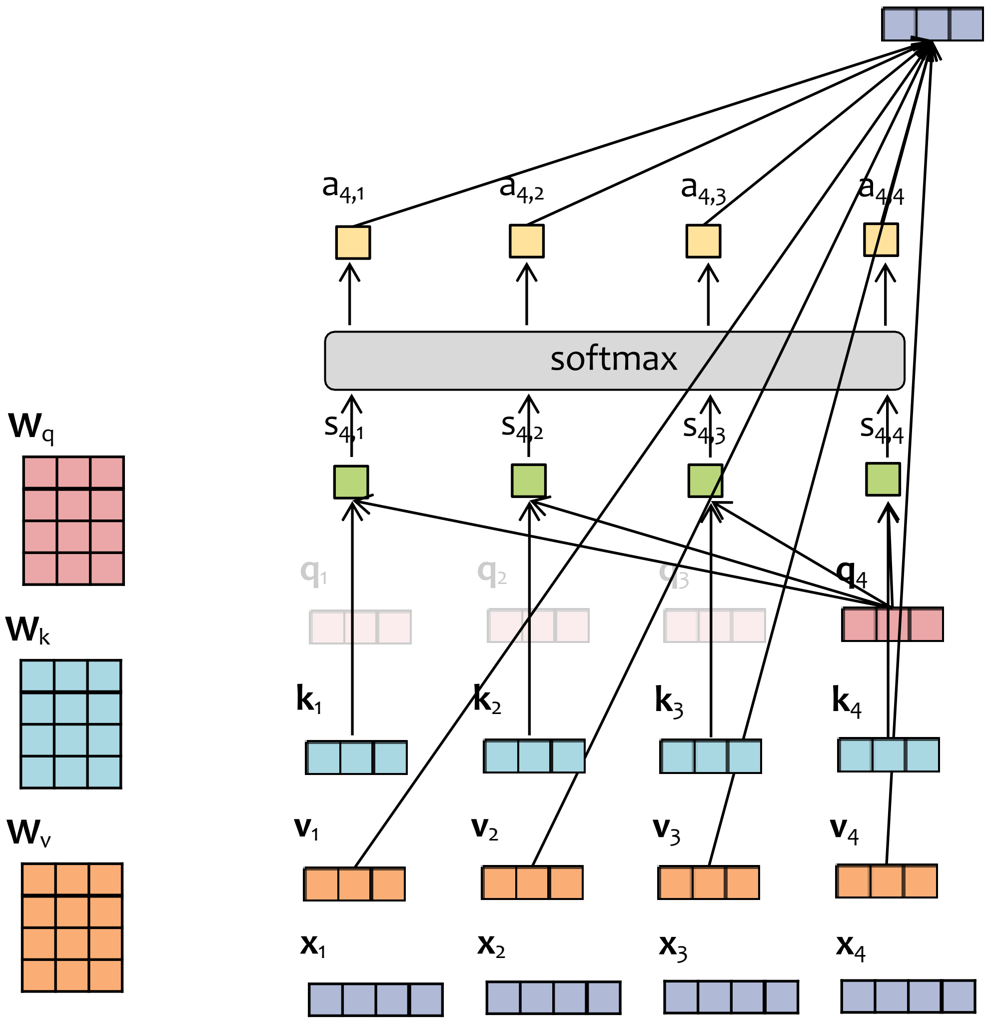
$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$$ queries

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$$ keys

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$ values

Scaled Dot-Product Attention

上海科技大学 ShanghaiTech University



$$x_4' = \sum_{j=1}^{4} a_{4,j} v_j$$

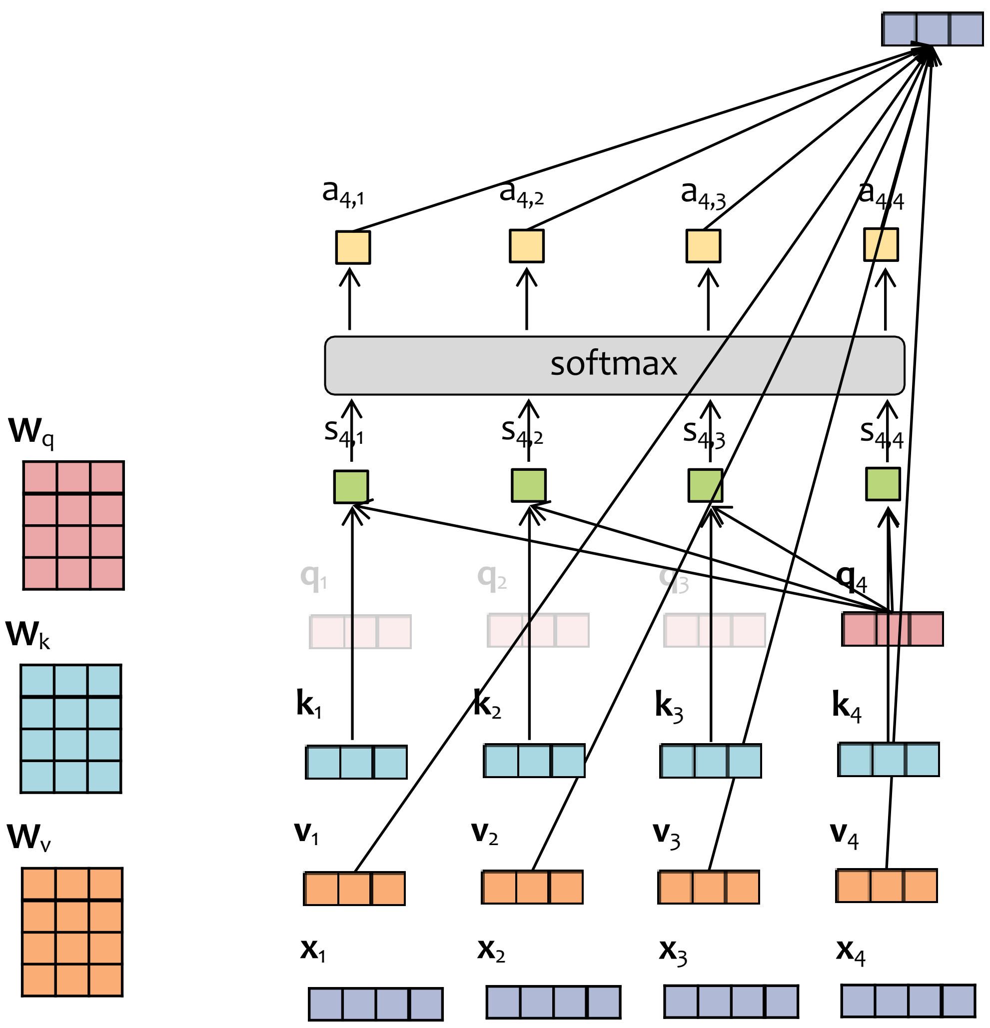
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Scaled Dot-Product Attention



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

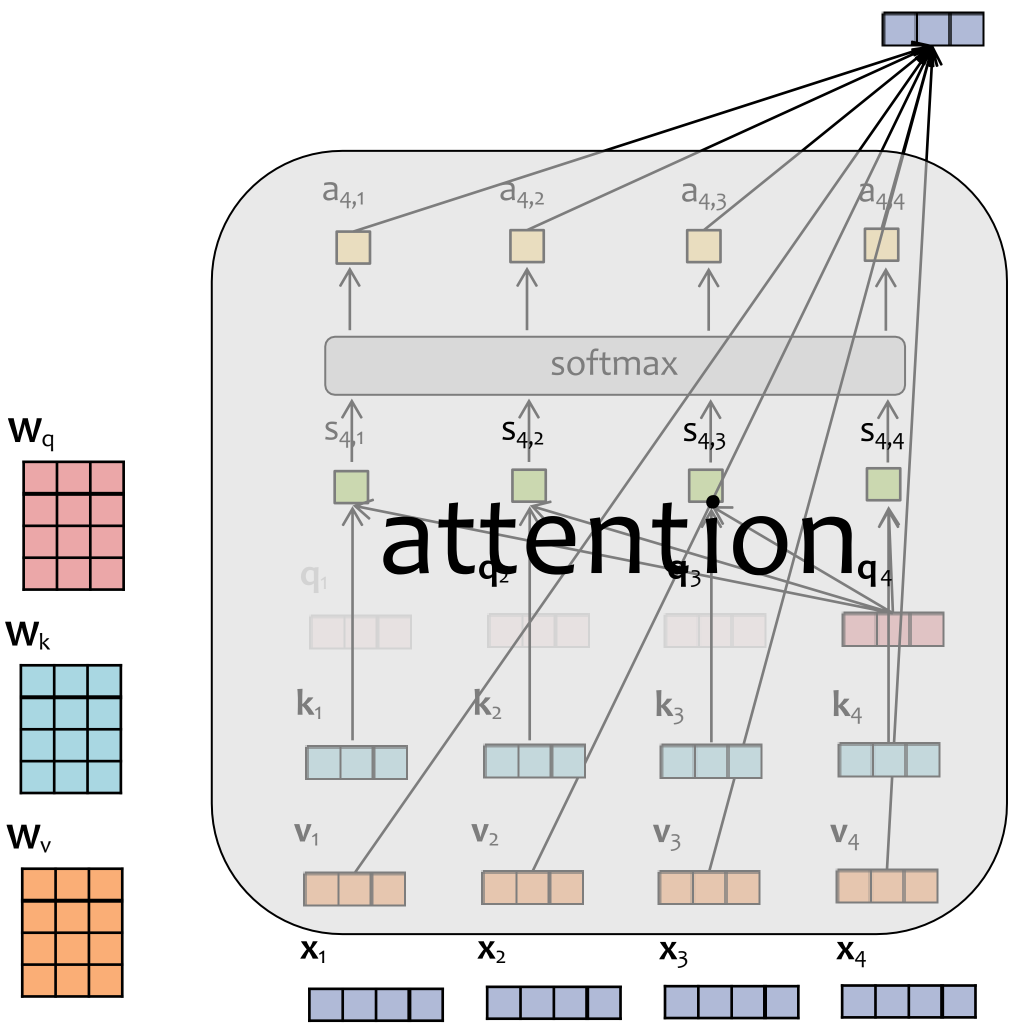
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Scaled Dot-Product Attention

上海科技大学 ShanghaiTech University



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \mathrm{softmax}(\mathbf{s}_4)$attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

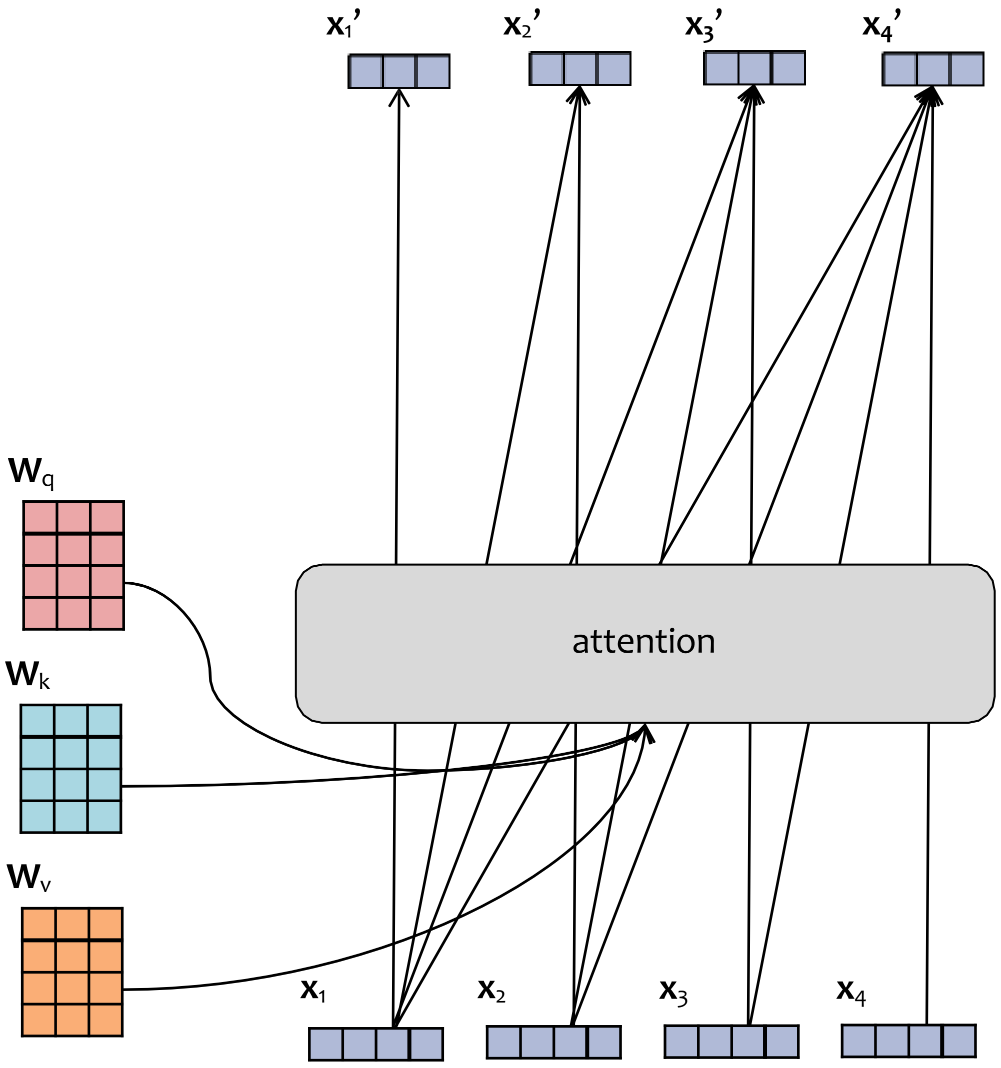
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Scaled Dot-Product Attention

上海科技大学 ShanghaiTech University



$$x'_t = \sum_{j=1}^{t} a_{t,j} v_j$$

$\mathbf{a}_t = \mathrm{softmax}(\mathbf{s}_t)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Animation of 3D Convolution

上海科技大学 ShanghaiTech University

Recall...

http://cs231n.github.io/convolutional-networks/

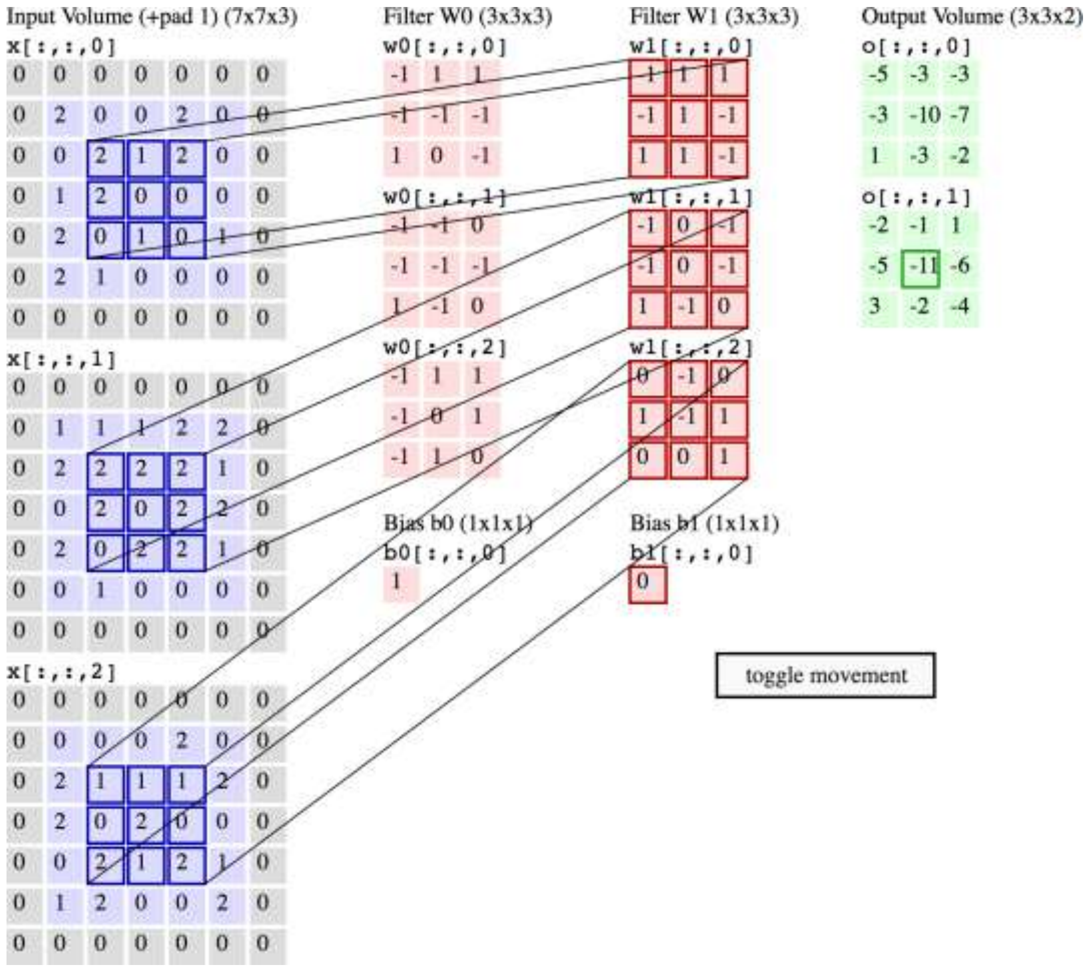
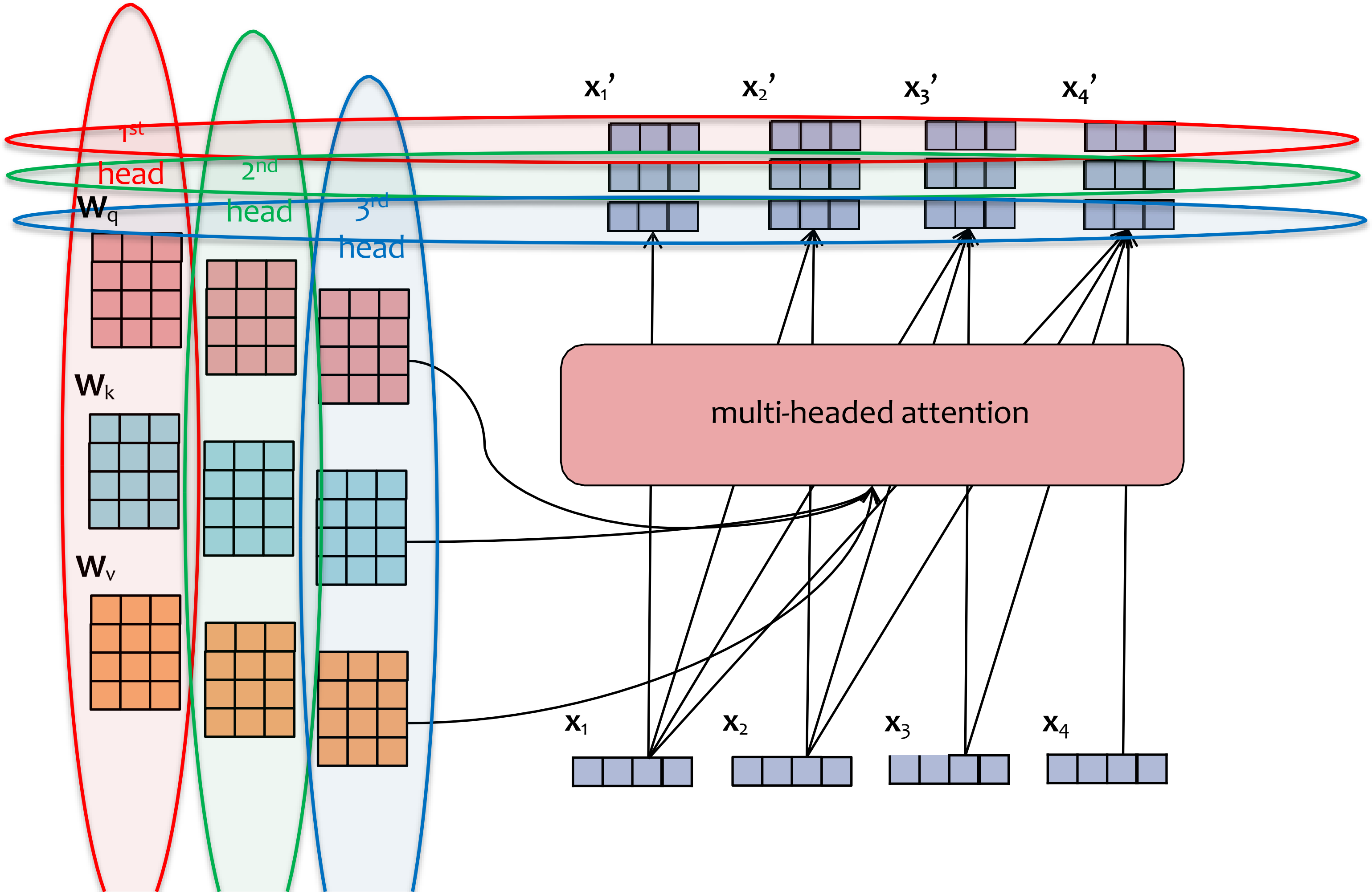


Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)

19

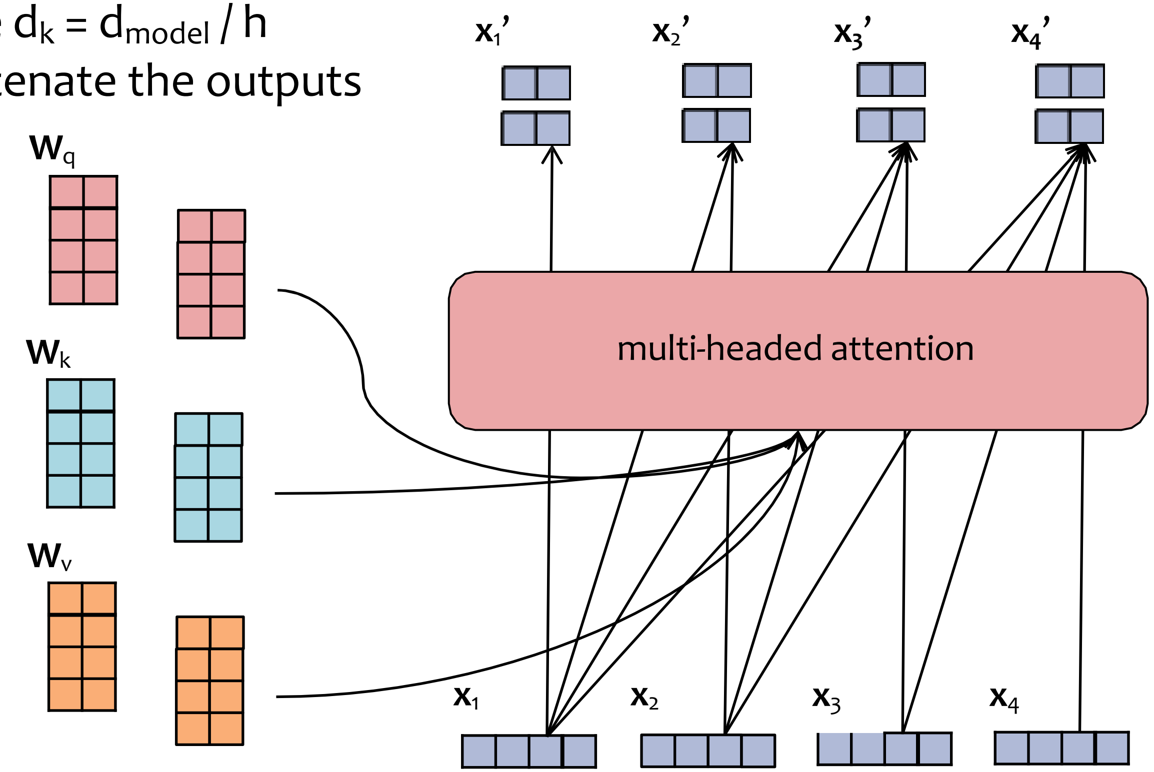
Multi-headed Attention



- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

Multi-headed Attention

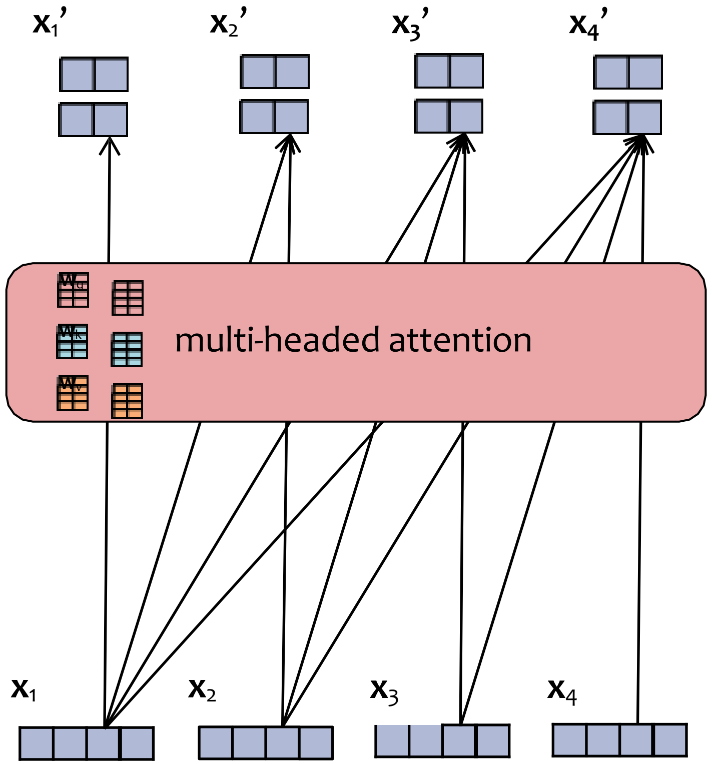
- To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t'$, Transformers usually choose the embedding sizes and number of heads appropriately:
  - d_{model} = dim. of inputs
  - d_k = dim. of each output
  - h = # of heads
  - Choose $d_k = d_{model}$ / h
- Then concatenate the outputs



- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

Multi-headed Attention

- To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t'$, Transformers usually choose the embedding sizes and number of heads appropriately:
  - d_{model} = dim. of inputs
  - d_k = dim. of each output
  - h = # of heads
  - Choose $d_k = d_{model} / h$
- Then concatenate the outputs



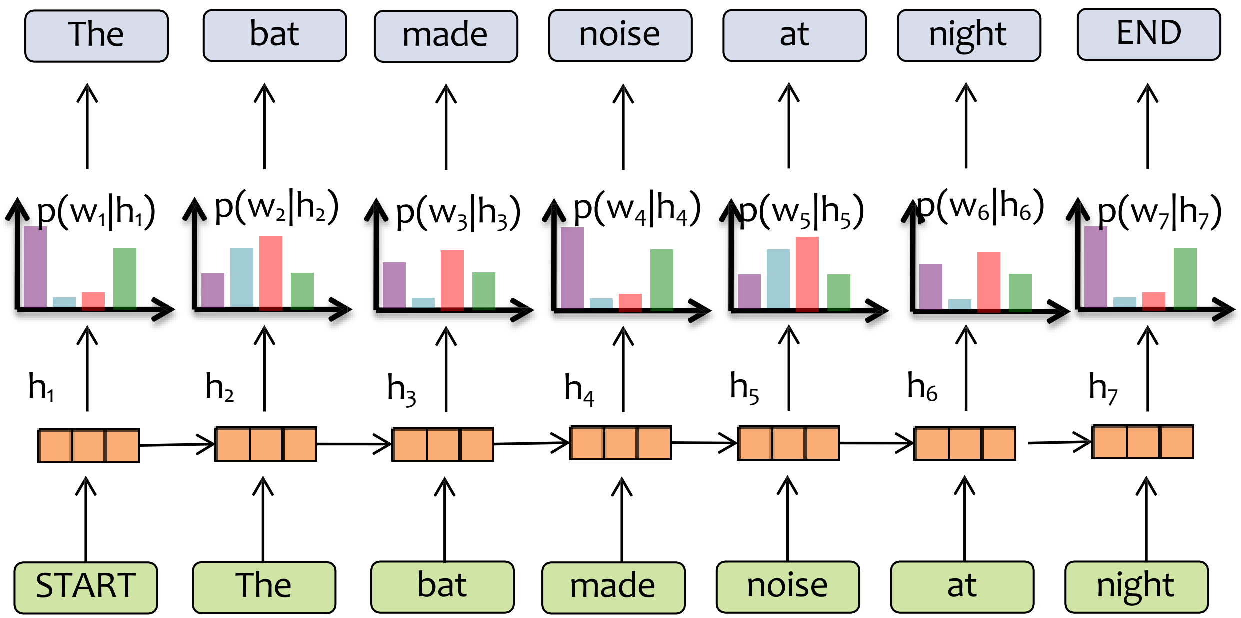
- Just as we can have **multiple channels** in a **convolution** layer, we can use **multiple heads** in an **attention** layer
- Each head gets **its own parameters**
- We can **concatenate** all the outputs to get a single vector for each time step

22

RNN Language Model

Recall...

上海科技大学 ShanghaiTech University



Key Idea:

(1) convert all previous words to a **fixed length vector**

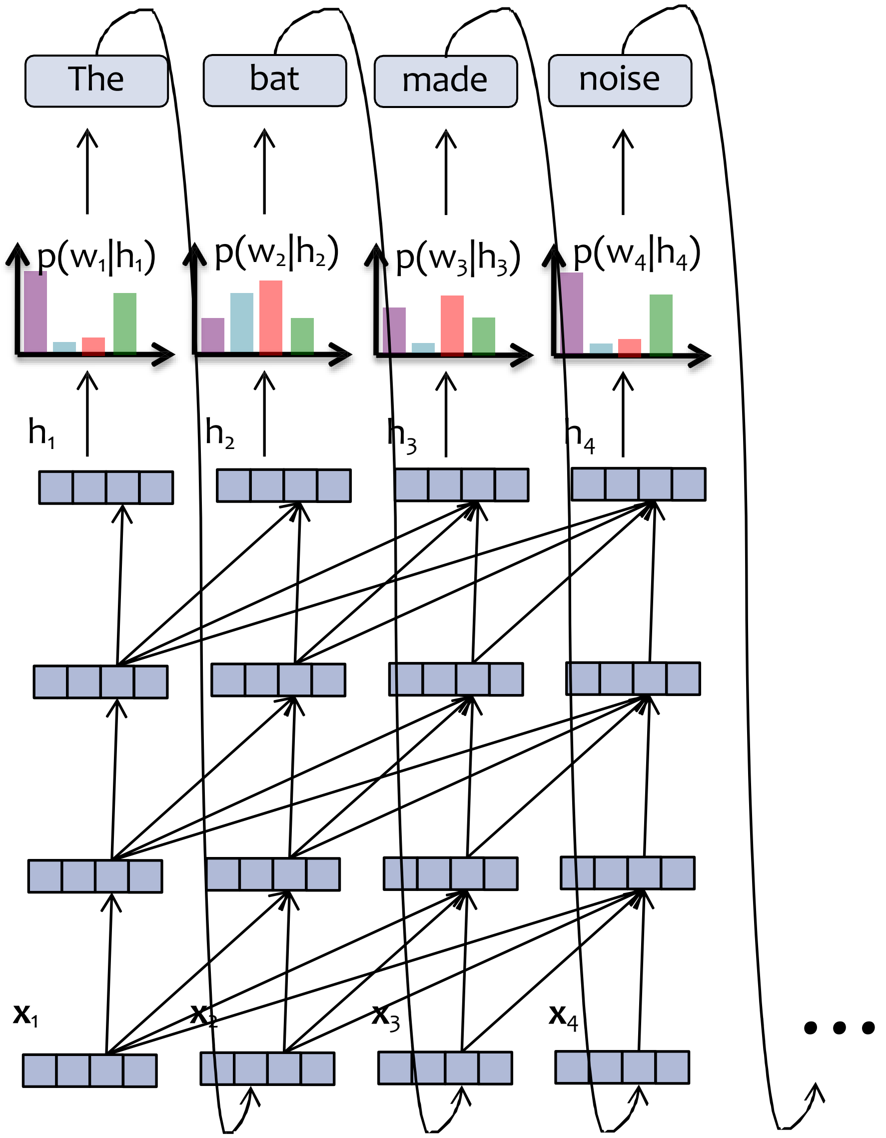
(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector $\mathbf{h}_t = f_\theta(w_{t-1}, \ldots, w_1)$

Transformer Language Model

上海科技大学
ShanghaiTech University

Important!

- RNN computation graph grows **linearly** with the number of input tokens
- Transformer-LM computation graph grows **quadratically** with the number of input tokens



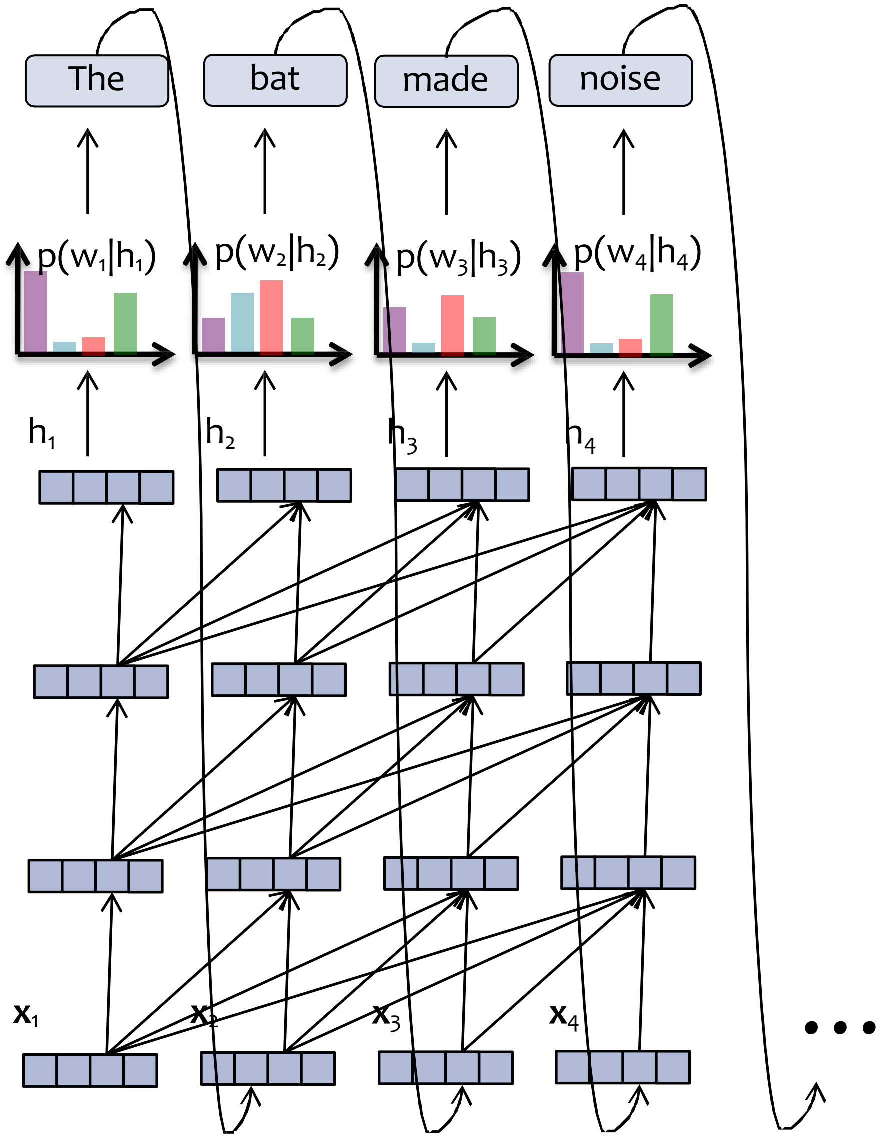
Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM!

Transformer Language Model

上海科技大学
ShanghaiTech University

Important!
- RNN computation graph grows **linearly** with the number of input tokens
- Transformer-LM computation graph grows **quadratically** with the number of input tokens



Each layer of a Transformer LM consists of several **sublayers:**
1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM!

Layer Normalization

上海科技大学 ShanghaiTech University

- *The Problem*: **internal covariate shift** occurs during training of a deep network when a small change in the low layers amplifies into a large change in the high layers
- *One Solution:* **Layer normalization** normalizes each layer and learns elementwise gain/bias
- Such normalization allows for higher learning rates (for **faster convergence**) without issues of diverging gradients

Given input a $\in \mathbb{R}^K$, LayerNorm computes output b $\in \mathbb{R}^K$:

$$b = \gamma \odot \frac{a - u}{\sigma} \oplus \beta$$

where we have mean $\mu = \frac{1}{K} \Sigma_{k=1}^{K} a_k$,

standard deviation $\sigma = \sqrt{\frac{1}{K} \Sigma_{k=1}^{K} (a_k - \mu)^2}$,

and parameters γ $\in \mathbb{R}^K$, β $\in \mathbb{R}^K$.

$\odot$ and $\oplus$ denote elementwise multiplication and addition.

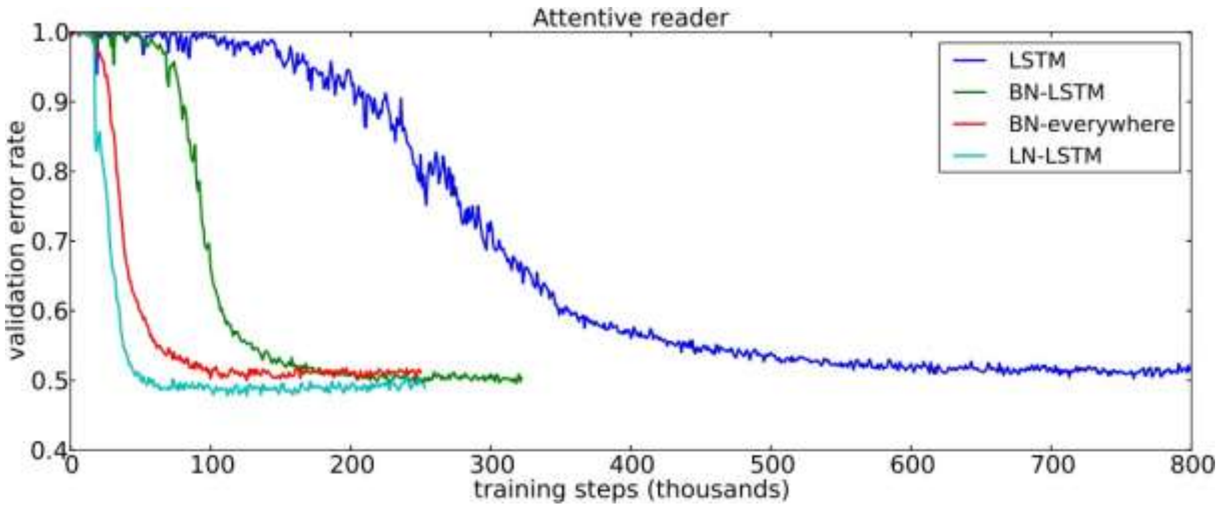


Figure from https://arxiv.org/pdf/1607.06450.pdf

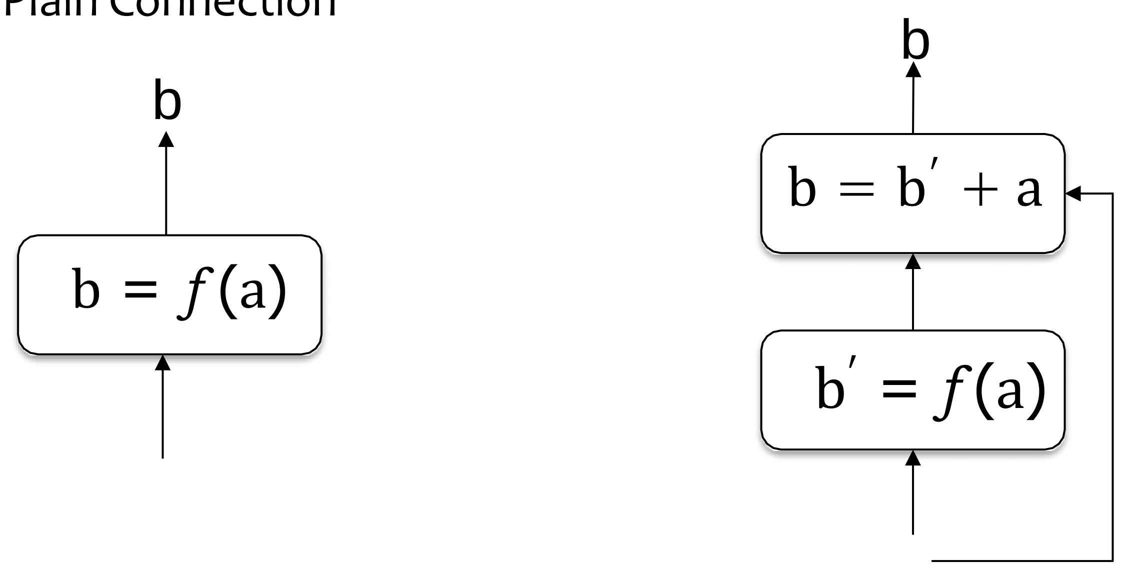
Residual Connections

上海科技大学
ShanghaiTech University

- *The Problem:* as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)
- *One Solution:* **Residual connections** pass a copy of the input alongside another function so that information can flow more directly
- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts

Plain Connection

Residual Connection



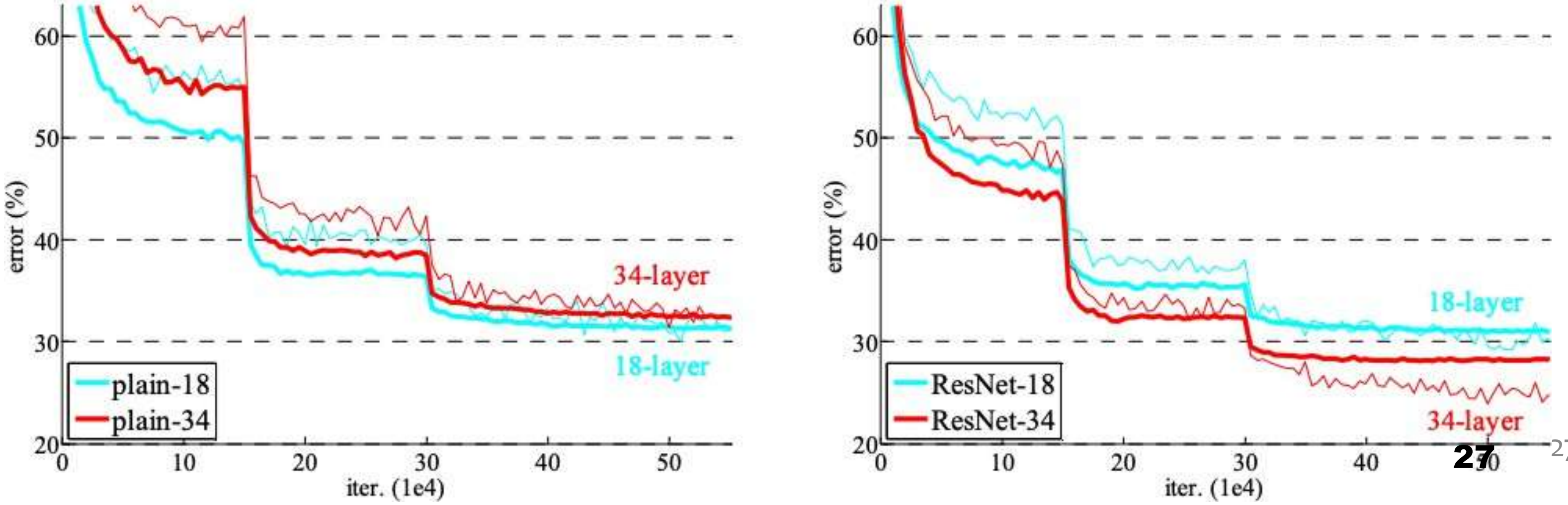
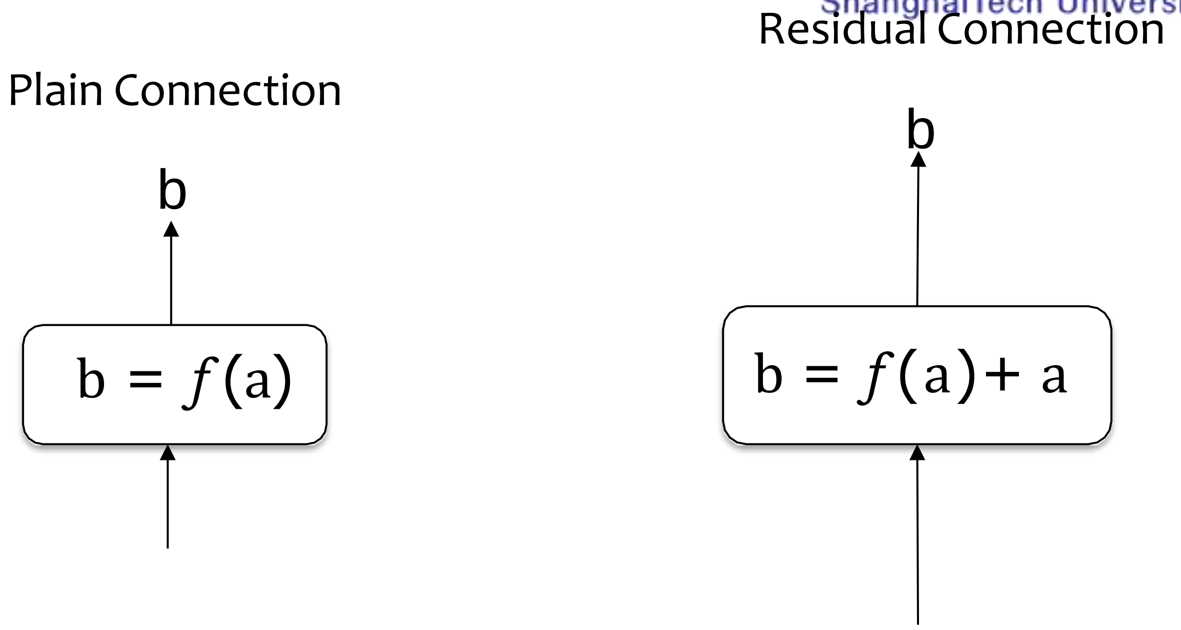


Figure from https://arxiv.org/pdf/1512.03385.pdf

Residual Connections

- *The Problem:* as network depth grows very large, a **performance degradation** occurs that is not explained by overfitting (i.e. train / test error both worsen)
- *One Solution:* **Residual connections** pass a copy of the input alongside another function so that information can flow more directly
- These residual connections allow for **effective training of very deep networks** that perform better than their shallower (though still deep) counterparts



Why are residual connections helpful?

Instead of f(a) having to learn a full transformation of a, f(a) only needs to learn an additive modification of a (i.e. the residual).

Figure from https://arxiv.org/pdf/1512.03385.pdf

28

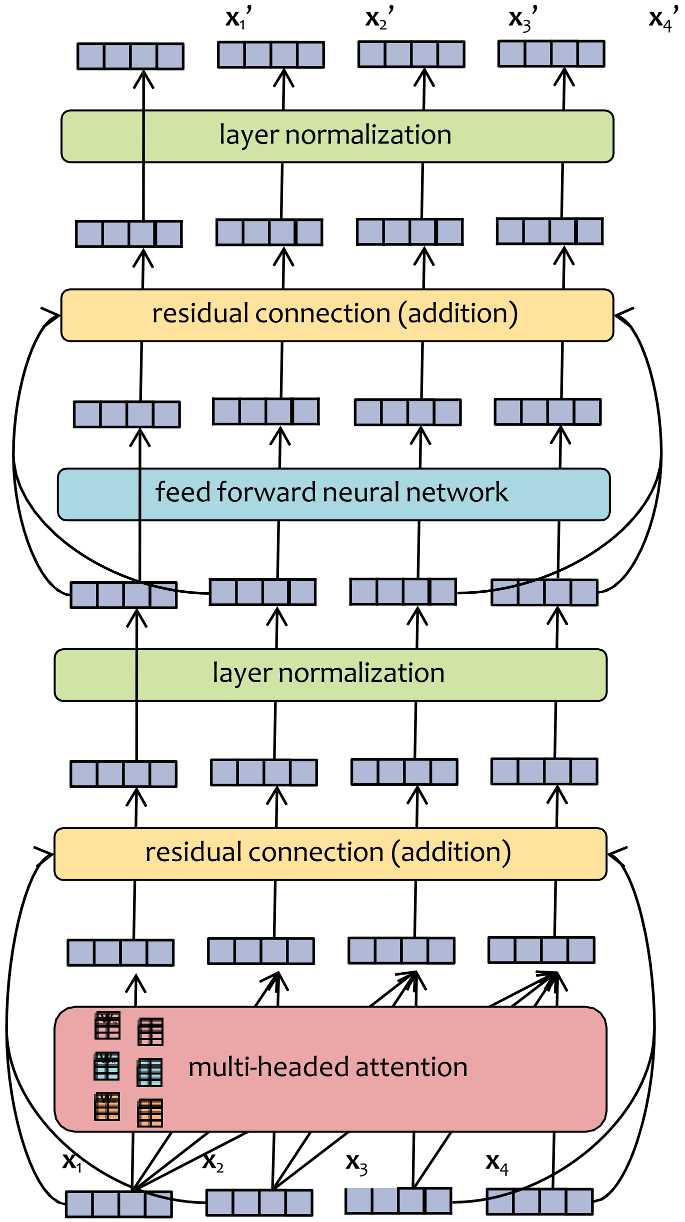
Transformer Layer

上海科技大学
ShanghaiTech University

Post-LN Version:

This is the version of the Transformer Layer that was introduced in the original paper in 2017.

The LayerNorm modules occur at the end of each set of 3 layers.



Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

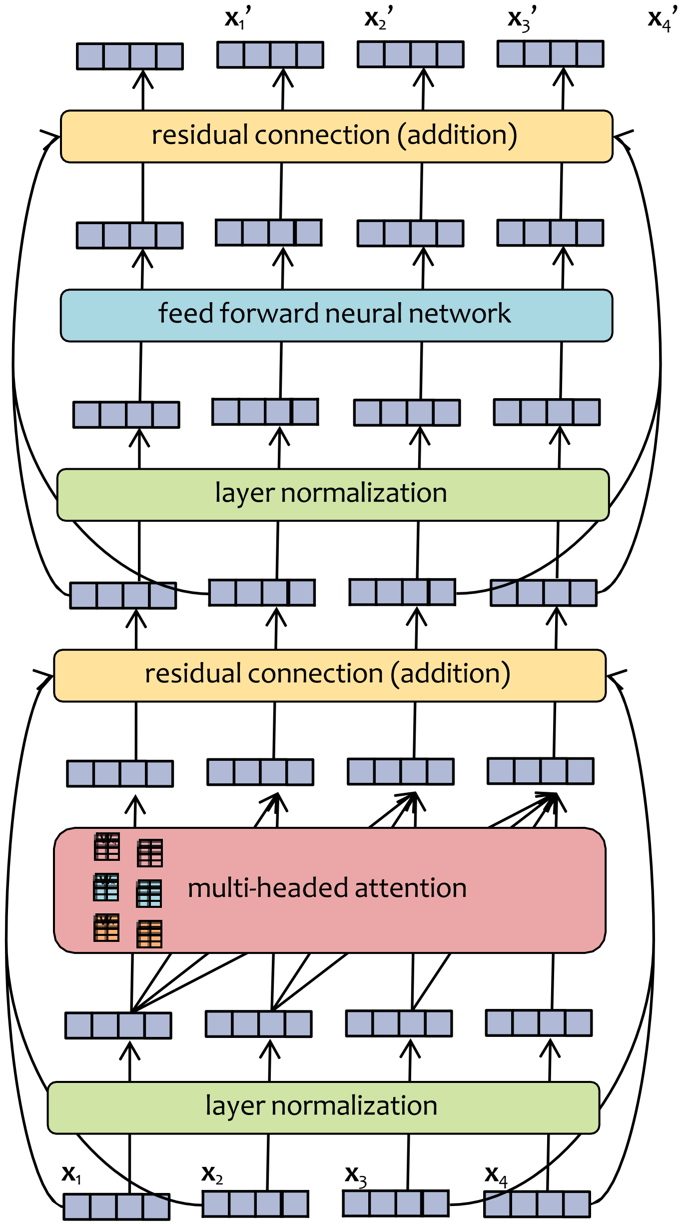
29

Transformer Layer

上海科技大学
ShanghaiTech University

Pre-LN Version:

However, subsequent work found that reordering such that the LayerNorm's came at the beginning of each set of 3 layers, the multi-headed attention and feed-forward NN layers tend to be better behaved (i.e. tricks like warm-up are less important).

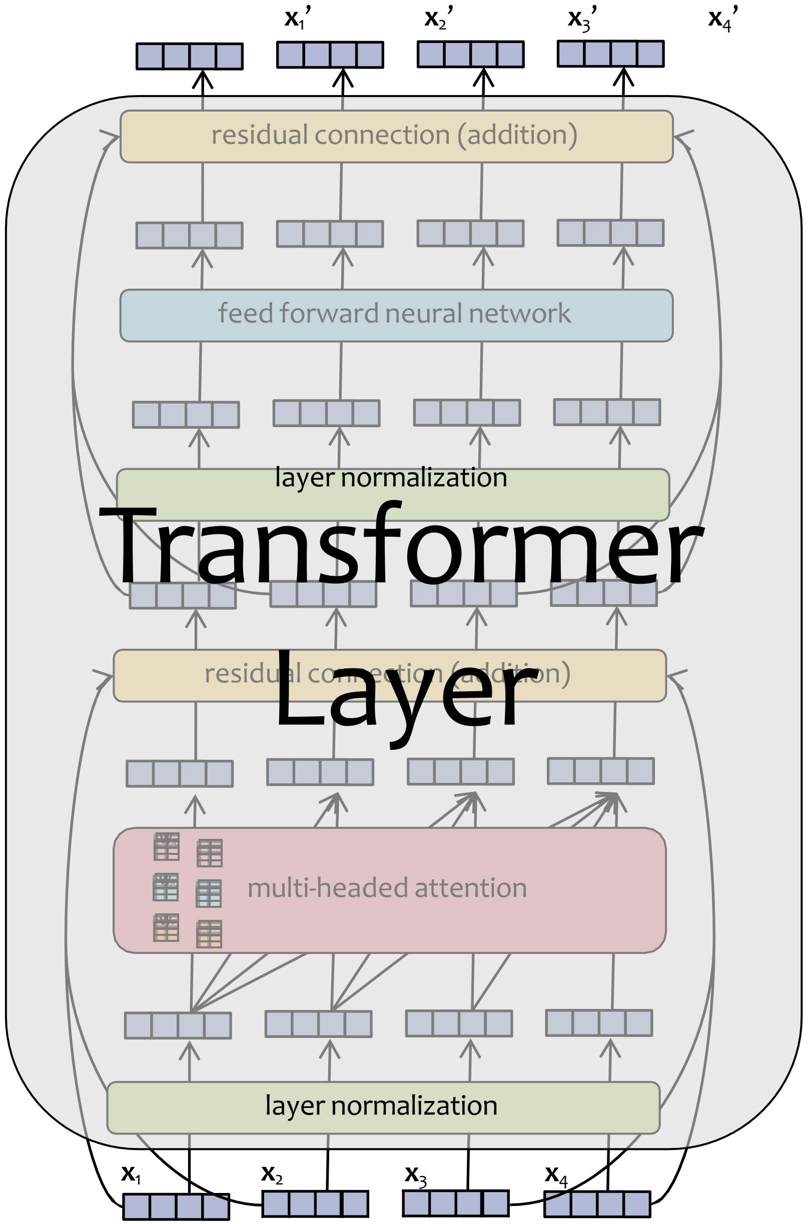


Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

30

Transformer Layer

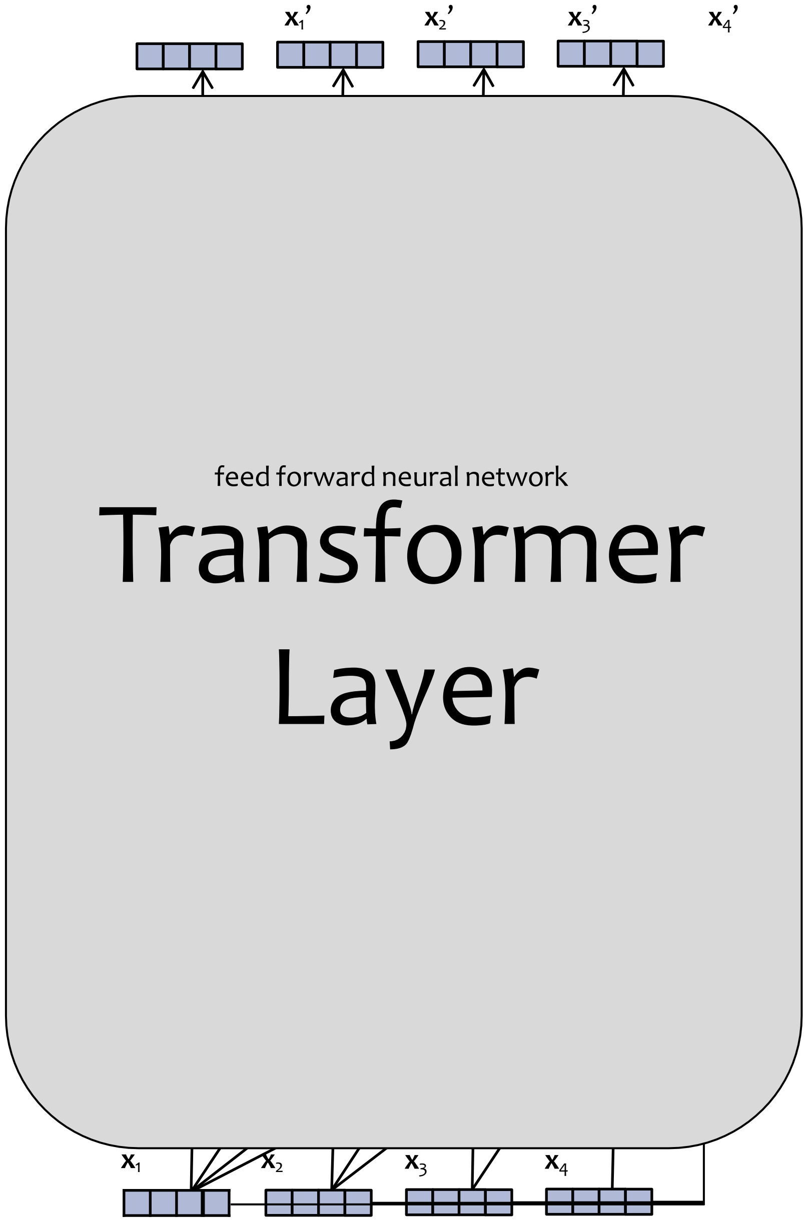


Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

31

Transformer Layer



上海科技大学
ShanghaiTech University

Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

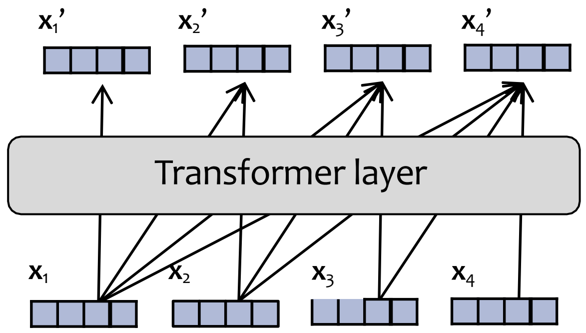
32

Transformer Layer

上海科技大学 ShanghaiTech University

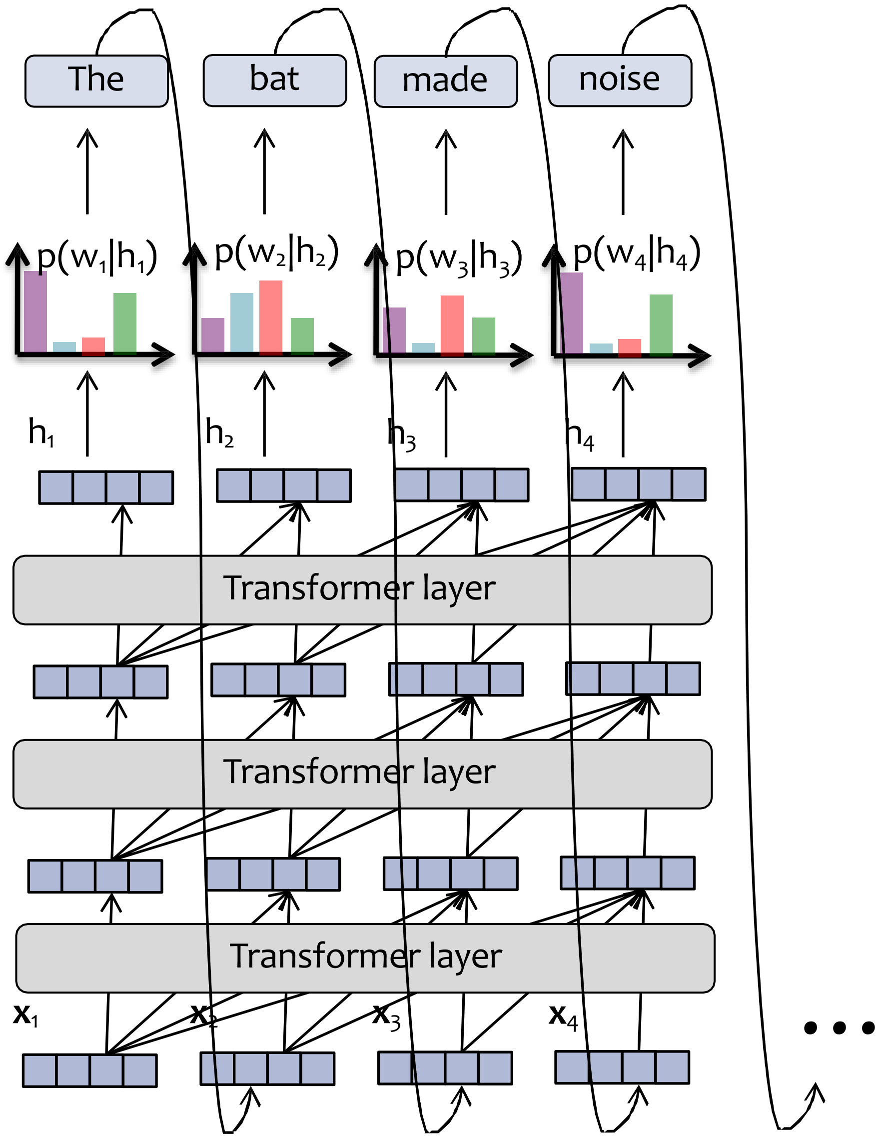
Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections



Transformer Language Model

上海科技大学 ShanghaiTech University



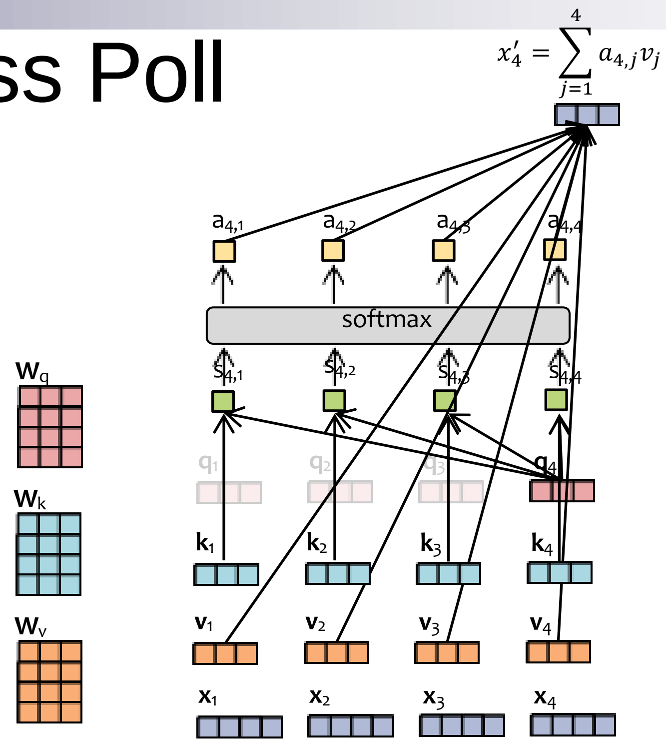
Each layer of a Transformer LM consists of several **sublayers:**

1. attention
2. feed-forward neural network
3. layer normalization
4. residual connections

Each hidden vector looks back at the hidden vectors of the **current and previous timesteps in the previous layer.**

The language model part is just like an RNN-LM.

In-Class Poll



$\mathbf{a}_4 = \mathrm{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Question:

Suppose we have the following input embeddings and attention weights:

- $x_1 = [1,0,0,0]$ $a_{4,1} = 0.1$
- $x_2 = [0,1,0,0]$ $a_{4,2} = 0.2$
- $x_3 = [0,0,2,0]$ $a_{4,3} = 0.6$
- $x_4 = [0,0,0,1]$ $a_{4,4} = 0.1$

And $W_v = I$. Then we can compute x_4'.

Now suppose we swap the embeddings x_2 and x_3 such that

- $x_2 = [0,0,2,0]$
- $x_3 = [0,1,0,0]$

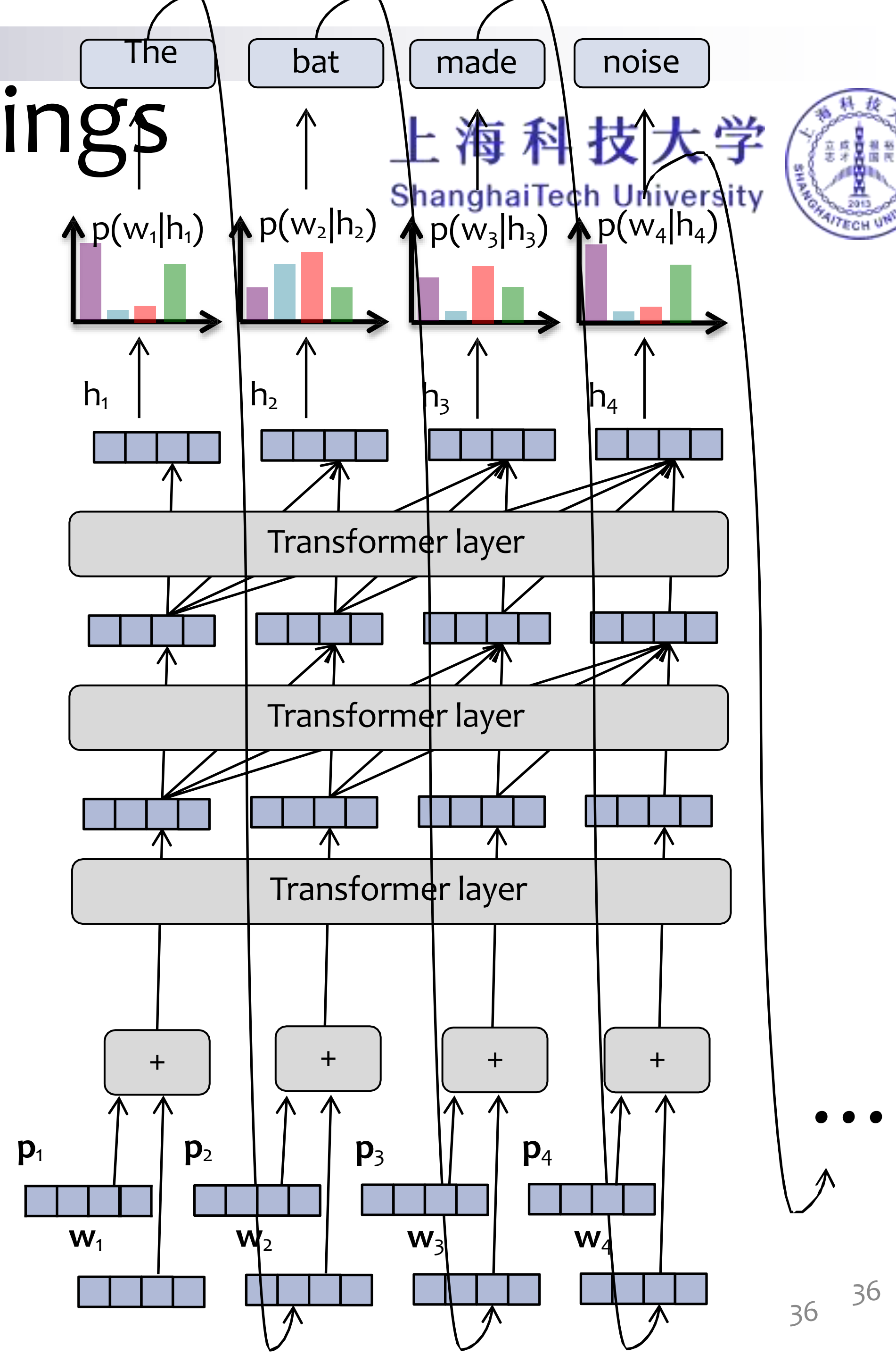
What is the new value of x_4'?

Answer:

35

Position Embeddings

- The Problem: Because attention is position invariant, we **need** a way to learn about positions
- The Solution: Use (or learn) a collection of position specific embeddings: $\mathbf{p}_t$ represents what it means to be in position t. And add this to the word embedding $\mathbf{w}_t$.
  The **key idea** is that every word that appears in position t uses the same position embedding $\mathbf{p}_t$
- There are a number of varieties of position embeddings:
  - Some are fixed (based on sine and cosine), whereas others are learned (like word embeddings)
  - Some are absolute (as described above) but we can also use relative position embeddings (i.e. relative to the position of the query vector)



36

LEARNING A TRANSFORMER LM

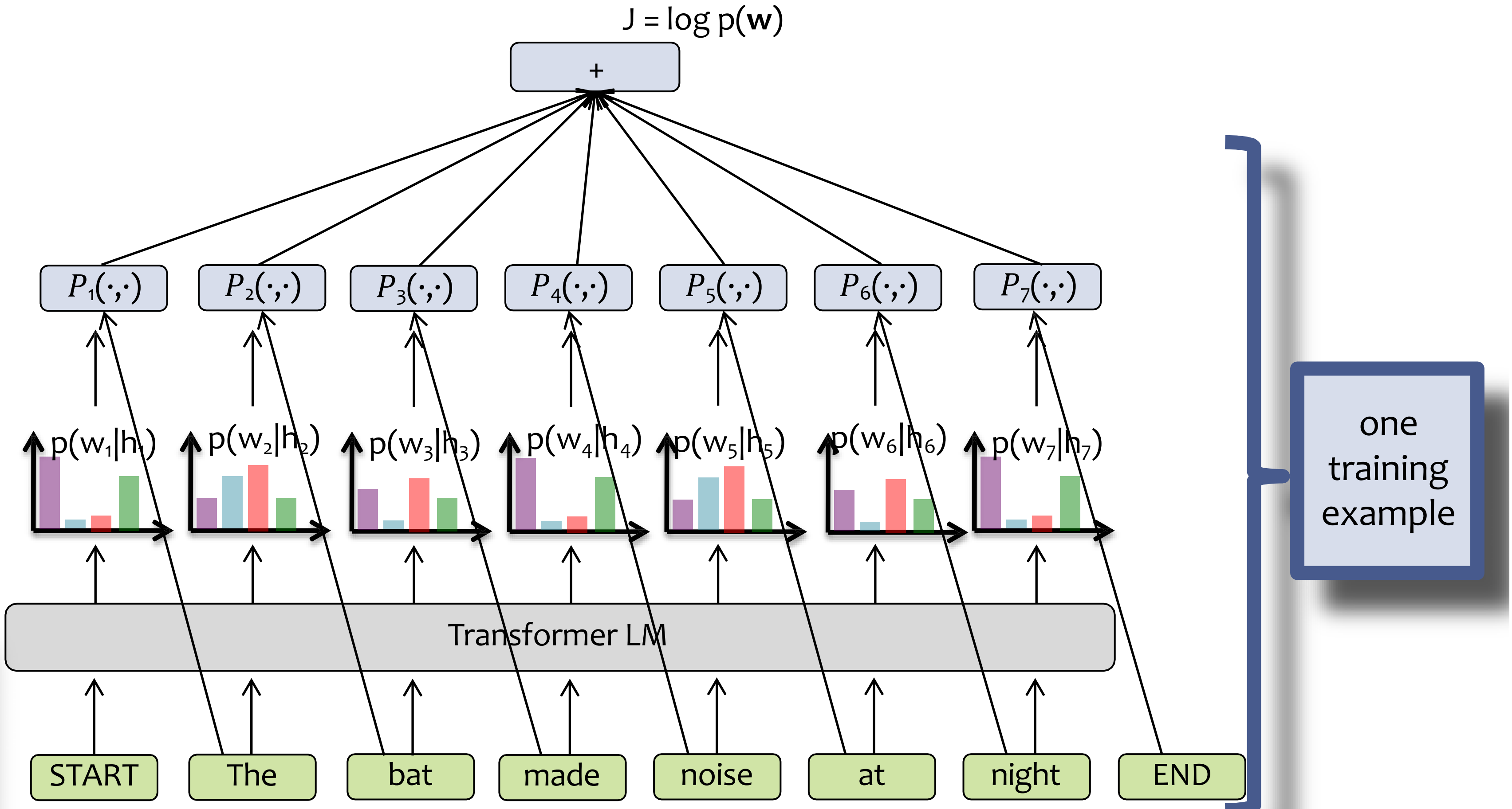
37

Learning a Transformer LM

- Each training example is a sequence (e.g. sentence), so we have training data D = {$\mathbf{w}^{(1)}$, $\mathbf{w}^{(2)}$, … , $\mathbf{w}^{(N)}$}
- The objective function for a Deep LM (e.g. RNN-LM or Tranformer-LM) is typically the log-likelihood of the training examples:
  $J(\boldsymbol{\theta}) = \Sigma_i \log p_{\boldsymbol{\theta}}(\mathbf{w}^{(i)})$
- We train by mini-batch SGD (or your favorite flavor of mini-batch SGD)

Training a Transformer-LM is the same, except we swap in a different deep language model.

$$\log p(\mathbf{w}) = \log p(w_1, w_2, w_3, \dots, w_T) = \log p(w_1 \mid h_1) + \log p(w_2 \mid h_2) + \dots + \log p(w_2 \mid h_T)$$



Language Modeling

An aside:

- State-of-the-art language models currently tend to rely on **transformer networks** (e.g. GPT-3)
- RNN-LMs comprised most of the early neural LMs that **led to** current SOTA architectures

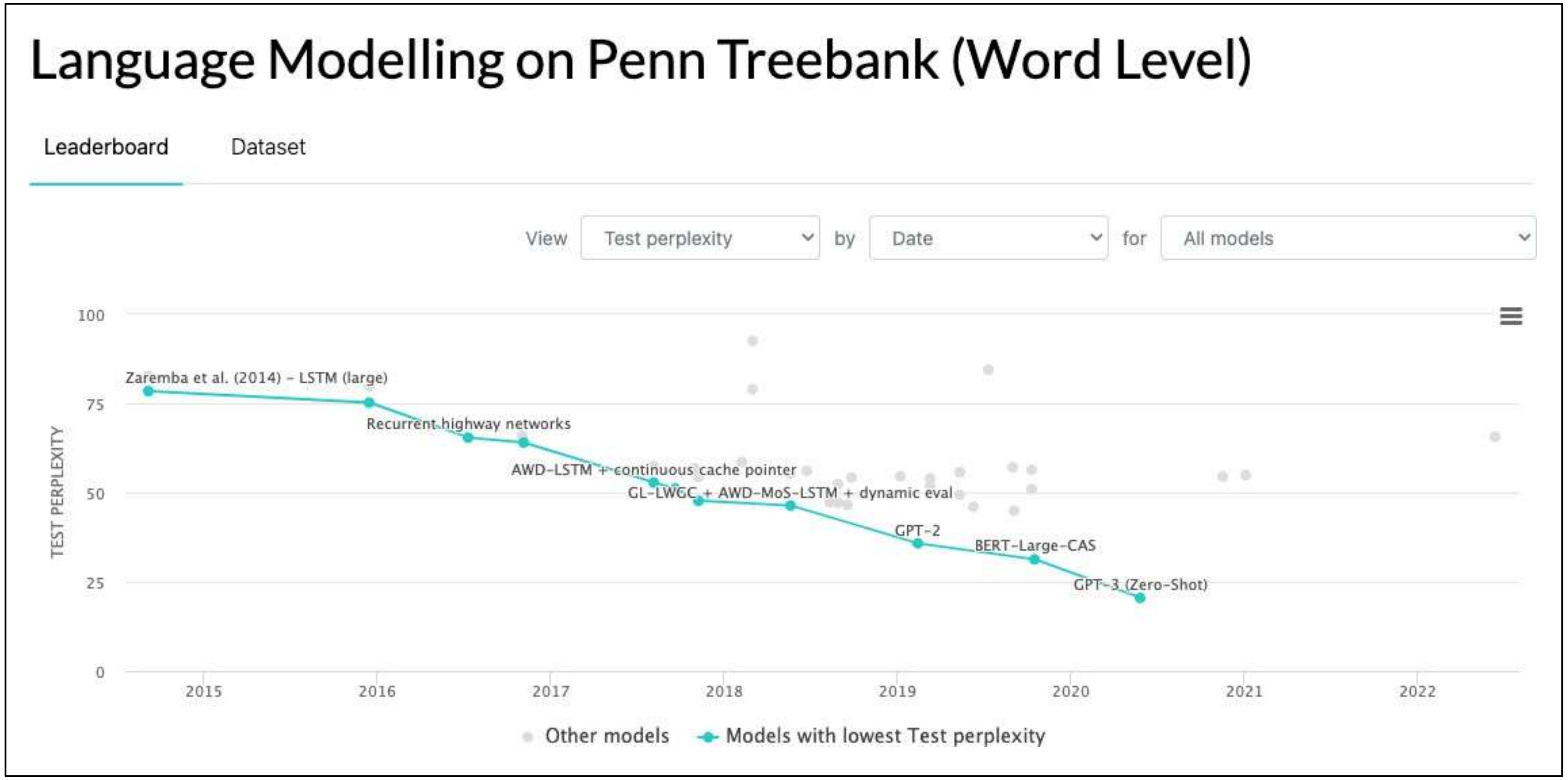


Figure from https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

GPT-3

上海科技大学 ShanghaiTech University

- GPT stands for Generative Pre-trained Transformer
- GPT is just a Transformer LM, but with a huge number of parameters

Model	# layers	dimension of states	dimension of inner states	# attention heads	# params
GPT (2018)	12	768	3072	12	117M
GPT-2 (2019)	48	1600	--	--	1542M
GPT-3 (2020)	96	12288	4*12288	96	175000M

Why does efficiency matter?

上海科技大学
ShanghaiTech University

Case Study: GPT-3

- # of training tokens = 500 billion
- # of parameters = 175 billion
- # of cycles = 50 petaflop/s-days (each of which are 8.64e+19 flops)

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

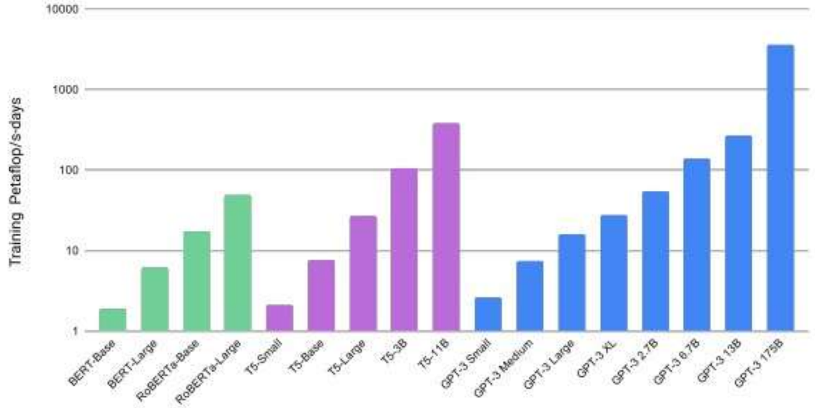


Figure 2.2: Total compute used during training. Based on the analysis in Scaling Laws For Neural Language Models [KMH+20] we train much larger models on many fewer tokens than is typical. As a consequence, although GPT-3 3B is almost 10x larger than RoBERTa-Large (355M params), both models took roughly 50 petaflop/s-days of compute during pre-training. Methodology for these calculations can be found in Appendix D.

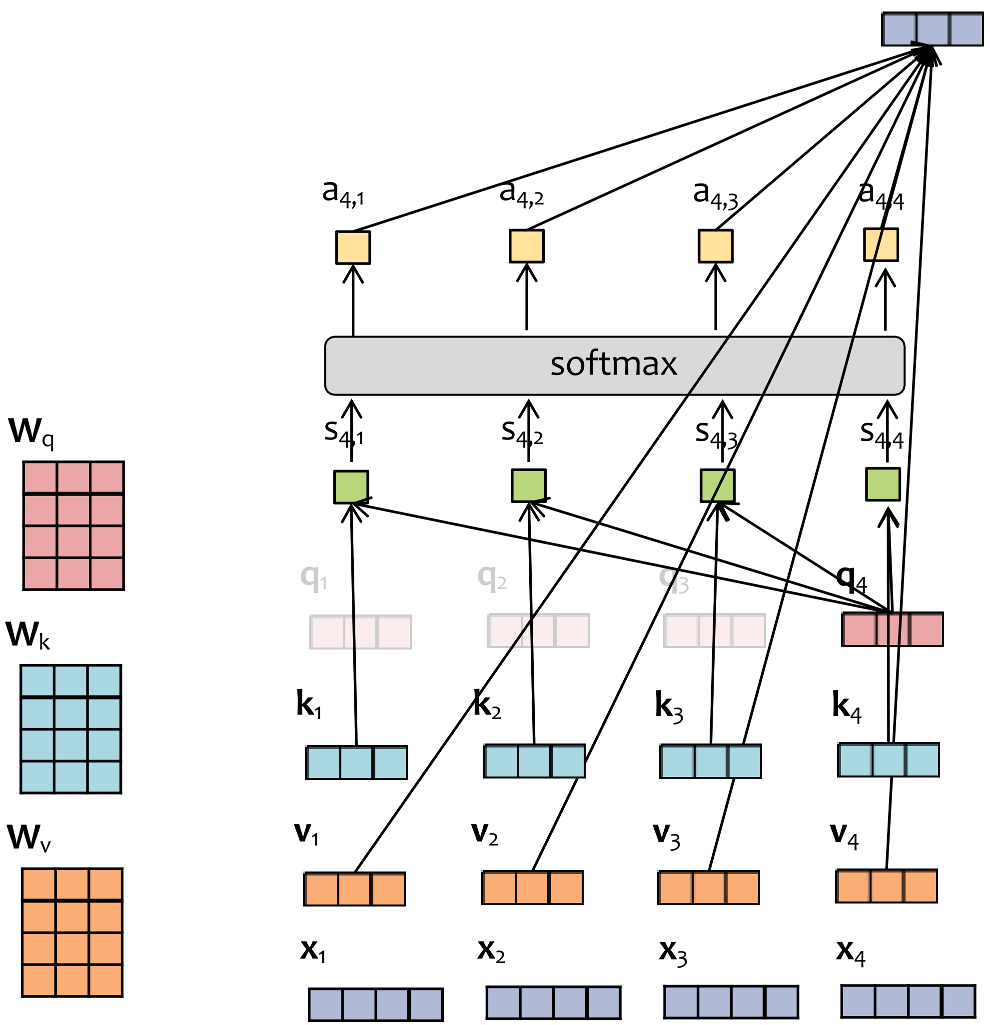
Figure from https://arxiv.org/pdf/2005.14165.|

41

IMPLEMENTING A TRANSFORMER LM

42

Matrix Version of Single-Headed Attention



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

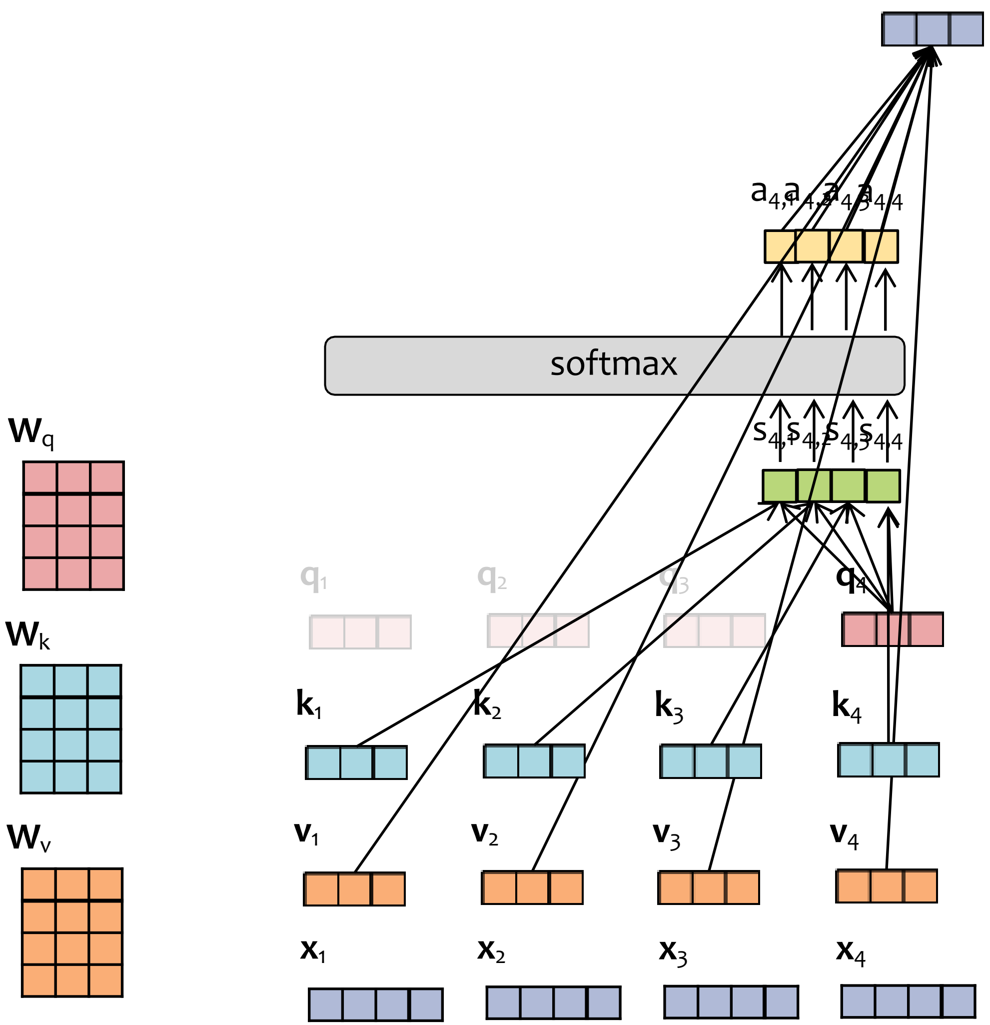
$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Matrix Version of Single-Headed Attention



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$ attention weights

$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$ scores

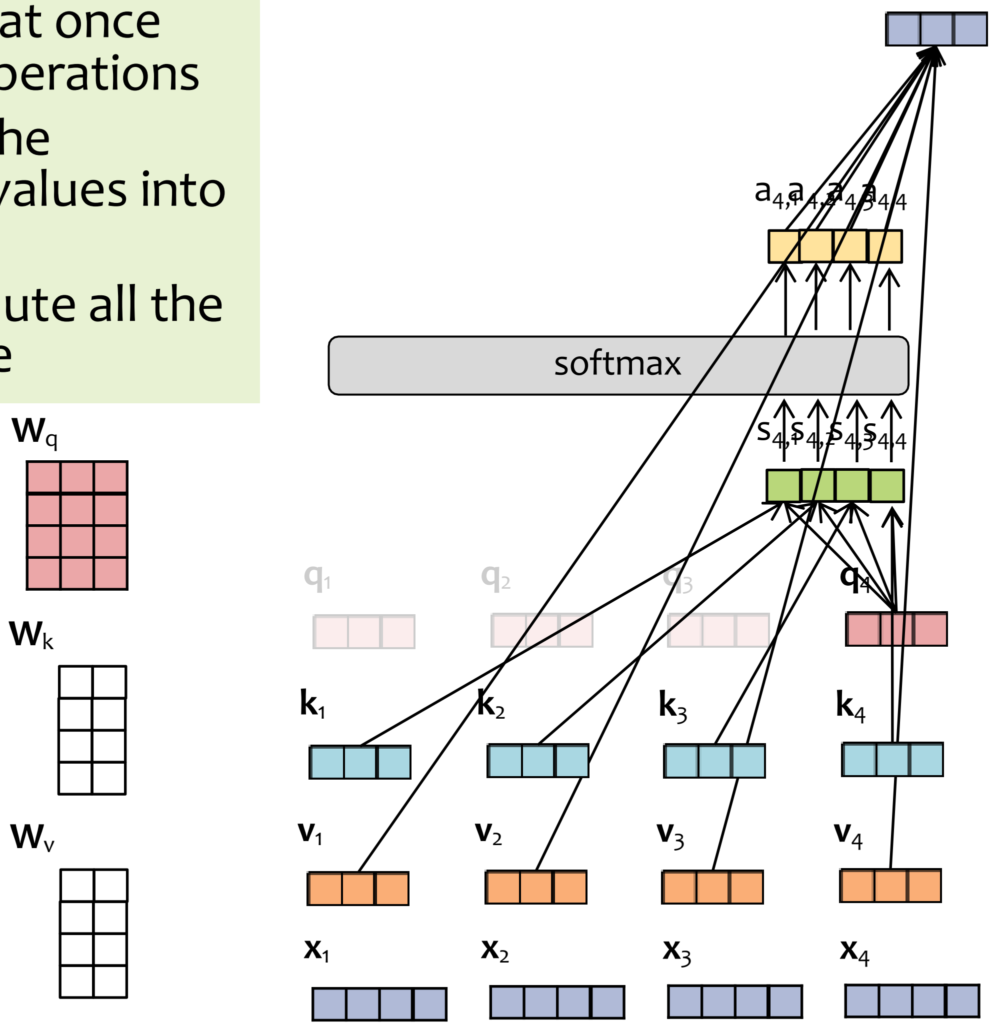
$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$ queries

$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$ keys

$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$ values

Matrix Version of Single-Headed Attention

- For speed, we compute all the queries at once using matrix operations
- First we pack the queries, keys, values into matrices
- Then we compute all the queries at once



$$X' = AV = \mathrm{softmax}(QK^T/\sqrt{d_k})V$$

$$A = [a_1, \ldots, a_4]^T = \mathrm{softmax}(S)$$

$$S = [s_1, \ldots, s_4]^T = QK^T/\sqrt{d_k}$$

$$Q = [q_1, \ldots, q_4]^T = XW_q$$

$$K = [k_1, \ldots, k_4]^T = XW_k$$

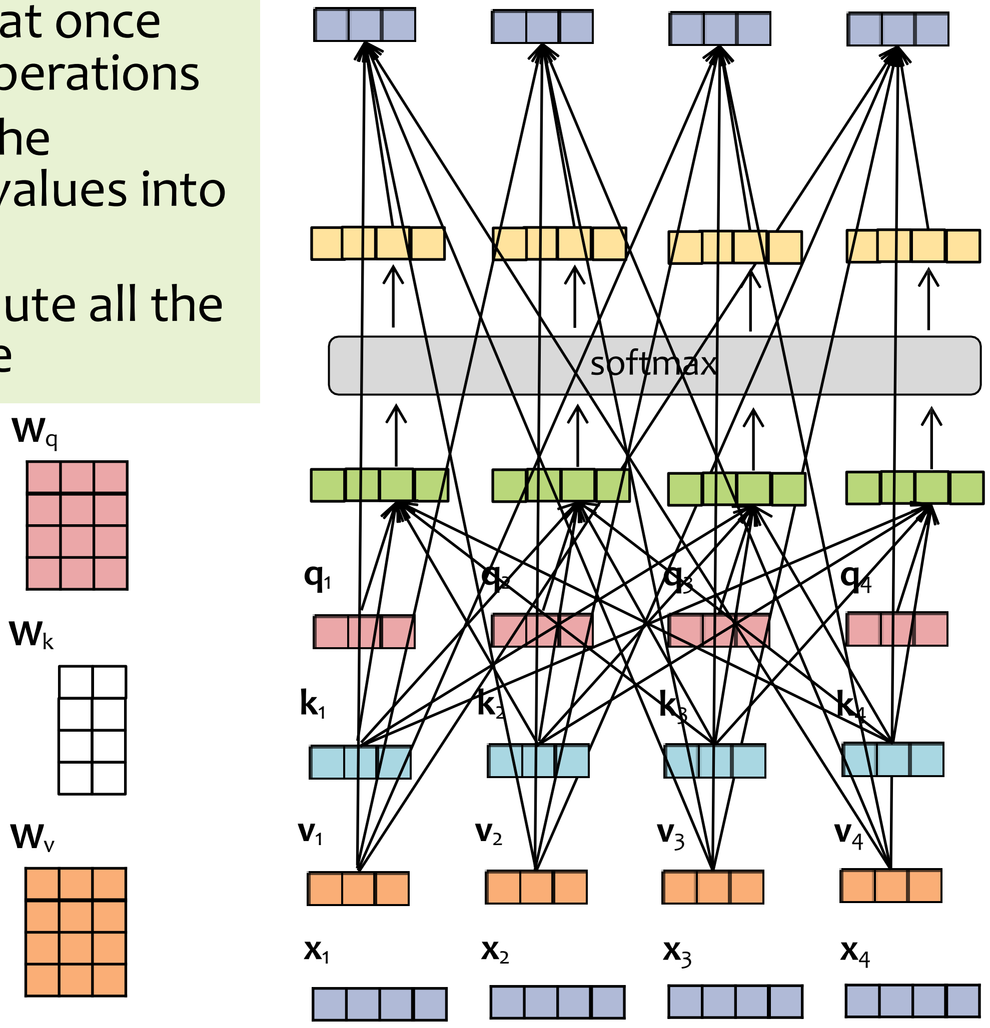
$$V = [v_1, \ldots, v_4]^T = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

Matrix Version of Single-Headed Attention

上海科技大学 ShanghaiTech University

- For speed, we compute all the queries at once using matrix operations
- First we pack the queries, keys, values into matrices
- Then we compute all the queries at once



$$X' = AV = \mathrm{softmax}(QK^T/\sqrt{d_k})V$$

$$A = [a_1, \ldots, a_4]^T = \mathrm{softmax}(S)$$

$$S = [s_1, \ldots, s_4]^T = QK^T/\sqrt{d_k}$$

$$Q = [q_1, \ldots, q_4]^T = XW_q$$

$$K = [k_1, \ldots, k_4]^T = XW_k$$

$$V = [v_1, \ldots, v_4]^T = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

Matrix Version of Single-Headed Attention

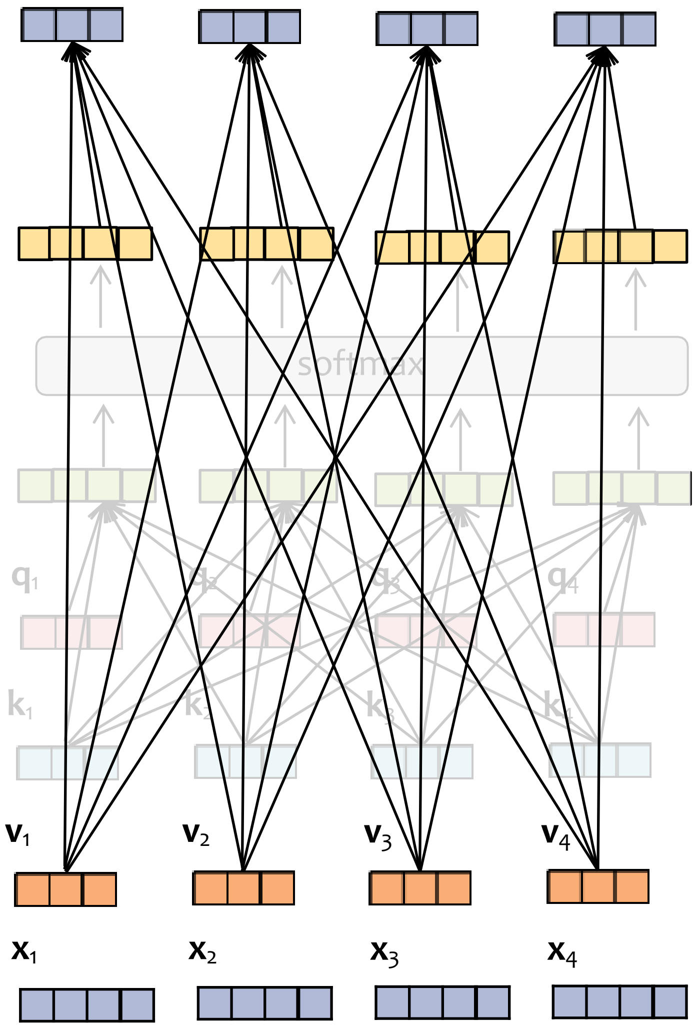
上海科技大学
ShanghaiTech University

Holy cow, that's a lot of new arrows… do we always want/need all of those?

- Suppose we're training our transformer to predict the next token(s) given the input…
- … then attending to tokens that come after the current token is cheating!

So what is this model?

- This version is the *standard* Transformer block. (more on this later!)
- But we want the Transformer LM block
- And that requires masking!



$$X' = AV = \mathrm{softmax}(QK^T / \sqrt{d_k})V$$

$$A = [a_1, \ldots, a_4]^T = \mathrm{softmax}(S)$$

$$S = [s_1, \ldots, s_4]^T = QK^T / \sqrt{d_k}$$

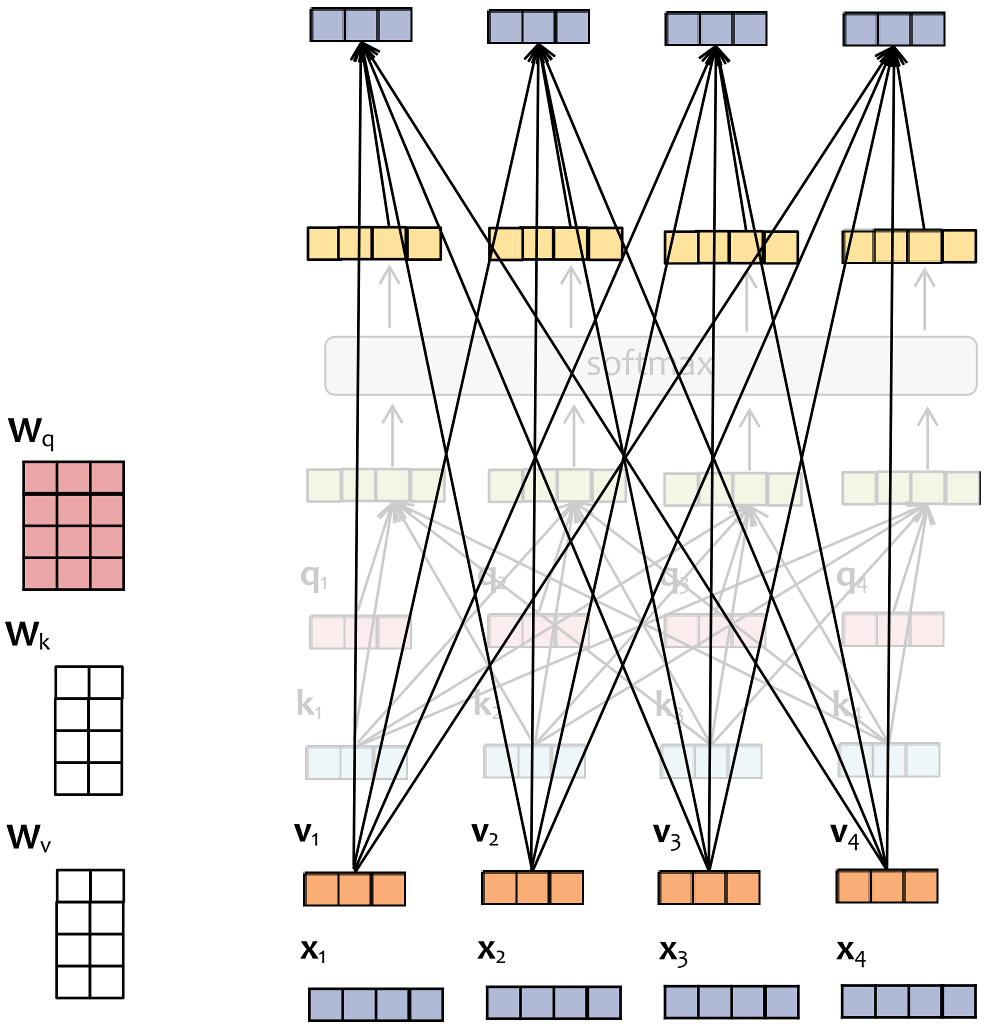
$$Q = [q_1, \ldots, q_4]^T = XW_q$$

$$K = [k_1, \ldots, k_4]^T = XW_k$$

$$V = [v_1, \ldots, v_4]^T = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

Matrix Version of Single-Headed Attention



$$X' = AV = \mathrm{softmax}(QK^T / \sqrt{d_k})V$$

$$A = \mathrm{softmax}(S)$$

$$S = QK^T / \sqrt{d_k}$$

$$Q = XW_q$$

$$K = XW_k$$

$$V = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

Question: How is the softmax applied?

A. column-wise

B. row-wise

Answer:

Matrix Version of Single-Headed (Causal) Attention

Insight: if some element in the input to the softmax is -∞, then the corresponding output is 0!

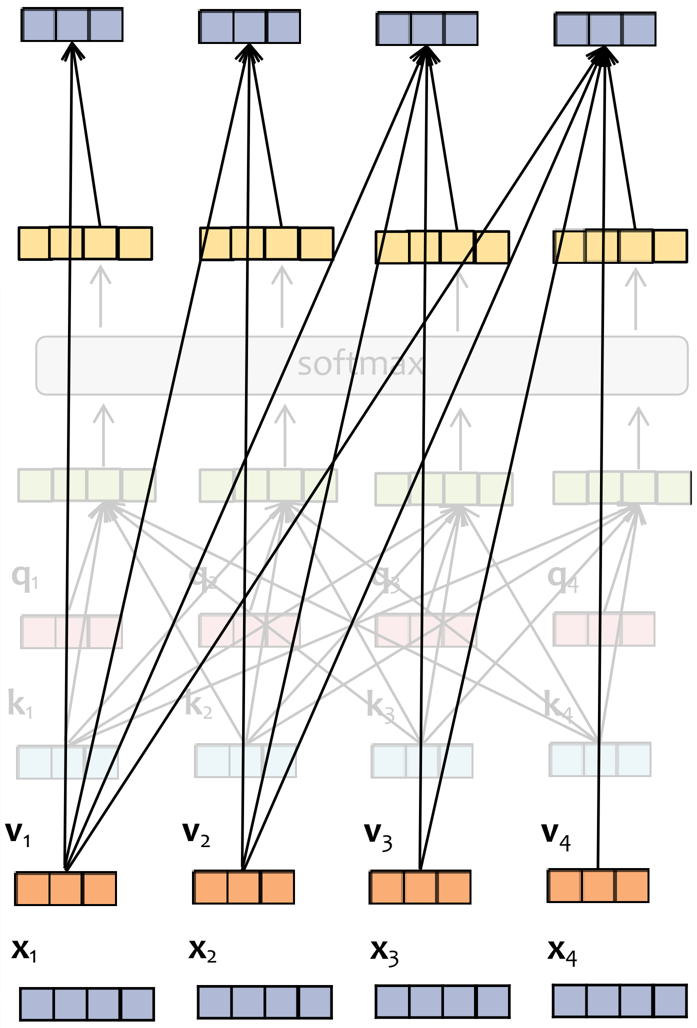
Question: For a causal LM which is the correct matrix?

A:
$$M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -\infty & 0 & 0 & 0 \\ -\infty & -\infty & 0 & 0 \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

B:
$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

C:
$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & -\infty & 0 \end{bmatrix}$$

Answer:



$$X' = AV = \text{softmax}(QK/\sqrt{d_k})V$$

$$A_{causal} = \text{softmax}(S + M)$$

$$S = QK^T/\sqrt{d_k}$$

$$Q = XW_q$$

$$K = XW_k$$

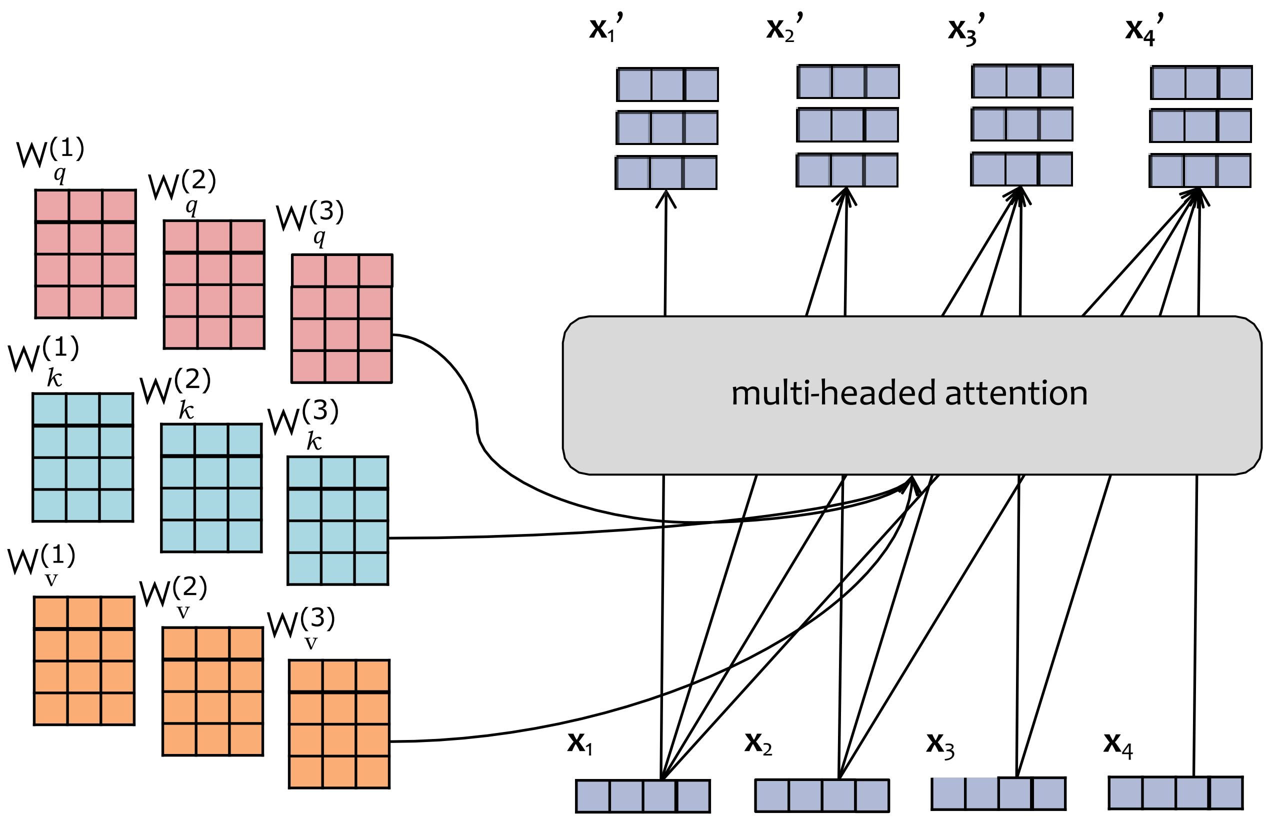
$$V = XW_v$$

$$X = [x_1, \ldots, x_4]^T$$

In practice, the attention weights are computed for all time steps T, then we mask out (by setting to –inf) all the inputs to the softmax that are for the timesteps to the right of the query.

Matrix Version of Multi-Headed (Causal) Attention

上海科技大学 ShanghaiTech University



$$X = \text{concat}(X'^{(1)}, X'^{(2)}, X'^{(3)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$

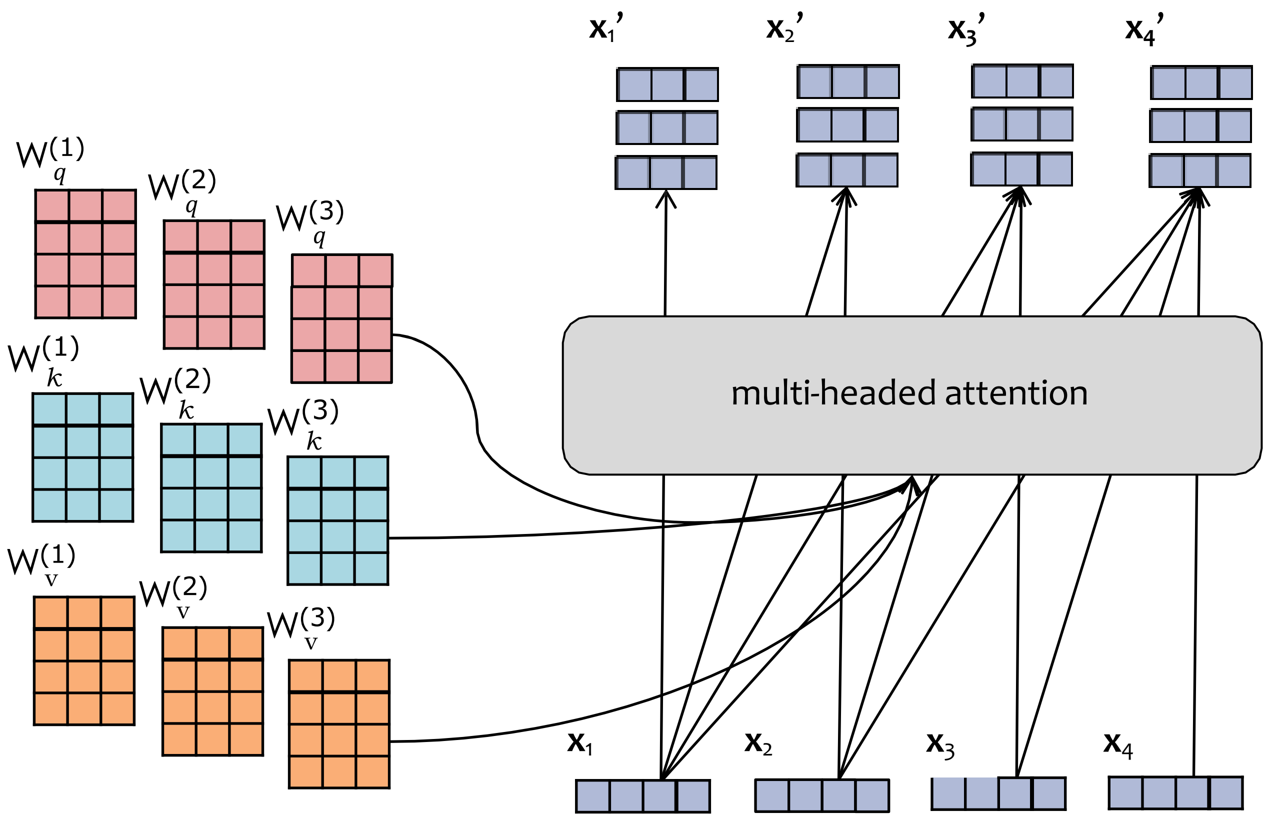
$$Q^{(i)} = XW_q^{(i)}$$

$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

$$X = [x_1, \dots, x_4]^T$$

Matrix Version of Multi-Headed (Causal) Attention



$$X = \text{concat}(X'^{(1)}, \dots, X'^{(h)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$

$$Q^{(i)} = XW_q^{(i)}$$

$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

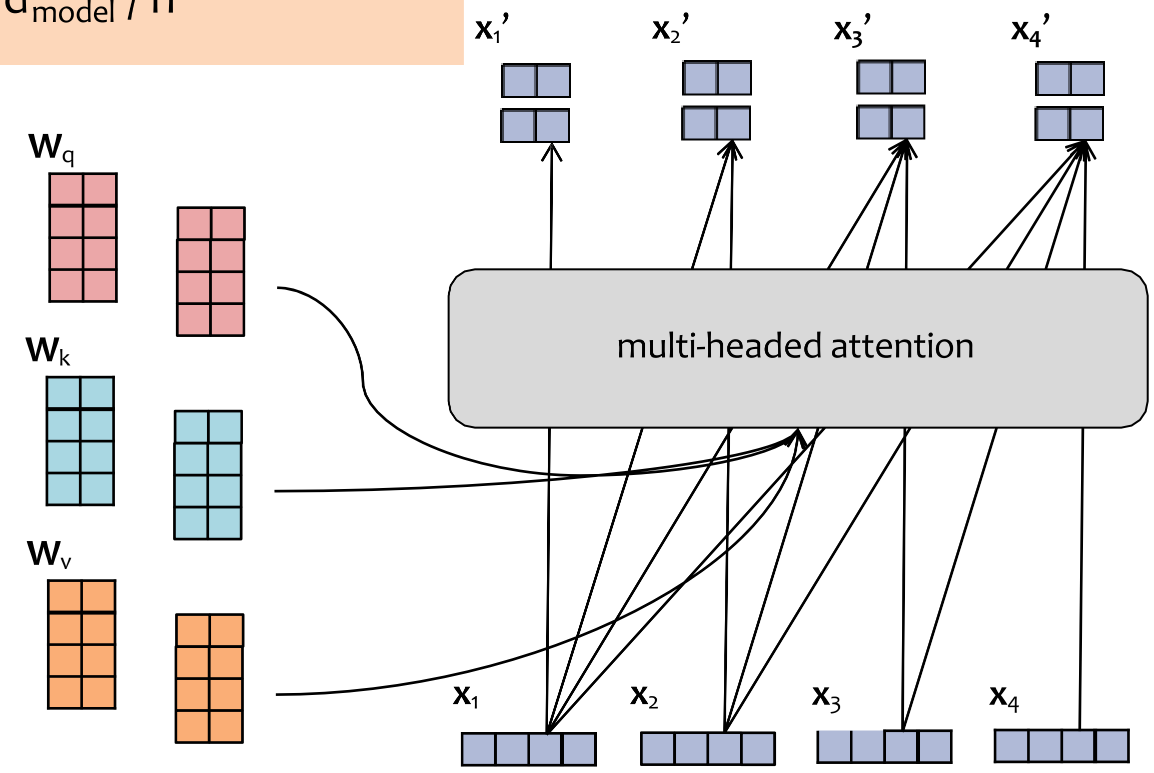
$$X = [x_1, \dots, x_4]^T$$

Recall:
To ensure the dimension of the **input** embedding $\mathbf{x}_t$ is the same as the **output** embedding $\mathbf{x}_t$', Transformers usually choose the embedding sizes and number of heads appropriately:

- d_{model} = dim. of inputs
- d_k = dim. of each output
- h = # of heads
- Choose $d_k = d_{model} / h$

Multi-Headed (Causal) Attention

上海科技大学 ShanghaiTech University



$$X = \text{concat}(X'^{(1)}, \ldots, X'^{(h)})$$

$$X'^{(i)} = \text{softmax}\left(\frac{Q^{(i)}(K^{(i)})^T}{\sqrt{d_k}} + M\right) V^{(i)}$$

$$Q^{(i)} = XW_q^{(i)}$$

$$K^{(i)} = XW_k^{(i)}$$

$$V^{(i)} = XW_v^{(i)}$$

$$X = [x_1, \ldots, x_4]^T$$

PRACTICALITIES OF TRANSFORMER LMS

Batching: Padding and Truncation

上海科技大学 ShanghaiTech University

- Transformers can be trained very efficiently!
  (This is arguably one of the key reasons they have been so successful.)
- **Batching**: Rather than processing one sentence at a time, Transformers take in a batch of B sentences at a time. The computation is identical for each batch and is trivially parallelized.

i	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}
1	In	the	hole	in	the	ground	there	lived	a	hobbit		
2	It	is	our	choices	that	show	what	we	truly	are		
3	It	was	the	best	of	times	it	was	the	worst	of	times
4	Even	miracles	take	a	little	time						
5	The	more	that	you	read	the	more	things	you	will	know	
6	We'll	always	have	each	other	no	matter	what	happens			
7	The	sun	did	not	shine	it	was	too	wet	to	play	
8	The	important	thing	is	to	never	stop	questioning				

Batching: Padding and Truncation

上海科技大学 ShanghaiTech University

- Suppose we have 8 training sentences
- We set our block size (maximum sequence length) to 10
- Before collecting them into a batch, we:
  1. truncate those sentences that are too long
  2. pad the sentences that are too short
  3. convert each token to an integer via a lookup table (vocabulary)
  4. convert each token to an embedding vector of fixed length

i	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}
1	In	the	hole	in	the	ground	there	lived	a	hobbit		
2	It	is	our	choices	that	show	what	we	truly	are		
3	It	was	the	best	of	times	it	was	the	worst	of	times
4	Even	miracles	take	a	little	time						
5	The	more	that	you	read	the	more	things	you	will	know	
6	We'll	always	have	each	other	no	matter	what	happens			
7	The	sun	did	not	shine	it	was	too	wet	to	play	
8	The	important	thing	is	to	never	stop	questioning				

Batching: Padding and Truncation

- Suppose we have 8 training sentences
- We set our block size (maximum sequence length) to 10
- Before collecting them into a batch, we:
  1. truncate those sentences that are too long
  2. pad the sentences that are too short
  3. convert each token to an integer via a lookup table (vocabulary)
  4. convert each token to an embedding vector of fixed length

i	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}
1	In	the	hole	in	the	ground	there	lived	a	hobbit		
2	It	is	our	choices	that	show	what	we	truly	are		
3	It	was	the	best	of	times	it	was	the	worst	of	times
4	Even	miracles	take	a	little	time	<PAD>	<PAD>	<PAD>	<PAD>		
5	The	more	that	you	read	the	more	things	you	will	know	
6	We'll	always	have	each	other	no	matter	what	happens	<PAD>		
7	The	sun	did	not	shine	it	was	too	wet	to	play	
8	The	important	thing	is	to	never	stop	questioning	<PAD>	<PAD>		

Batching: Padding and Truncation

上海科技大学 ShanghaiTech University

- Suppose we have 8 training sentences
- We set our block size (maximum sequence length) to 10
- Before collecting them into a batch, we:
  1. truncate those sentences that are too long
  2. pad the sentences that are too short
  3. convert each token to an integer via a lookup table (vocabulary)
  4. convert each token to an embedding vector of fixed length

i	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
1	2	41	17	19	41	13	42	23	6	16
2	3	20	32	10	40	36	53	51	49	8
3	3	50	41	9	30	46	21	50	41	55
4	1	25	39	6	22	45	0	0	0	0
5	4	26	40	56	34	41	26	44	56	54
6	5	7	15	12	31	28	24	53	14	0
7	4	38	11	29	35	21	50	48	52	47
8	4	18	43	20	47	27	37	33	0	0

Vocabulary:

```
{
    '<PAD>': 0,
    'Even': 1,
    'In': 2,
    'It': 3,
    'The': 4,
    "We'll": 5,
    'a': 6,
    'always': 7,
    'are': 8,
    'best': 9,
     …
    'what': 53,
    'will': 54,
    'worst': 55,
    'you': 56
}
```

Batching: Padding and Truncation

- Suppose we have 8 training sentences
- We set our block size (maximum sequence length) to 10
- Before collecting them into a batch, we:
  1. truncate those sentences that are too long
  2. pad the sentences that are too short
  3. convert each token to an integer via a lookup table (vocabulary)
  4. convert each token to an embedding vector of fixed length

i	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
1										
2										
3										
4										
5										
6										
7										
8										

Embeddings:

```
{
    0 :
    1 :
    2 :
    3 :
    4 :
    5 :
    6 :
    7 :
    ...
    55 :
    56 :
}
```

TOKENIZATION

Tokenization

Word-based Tokenizer:

Input: “Henry is giving a lecture on transformers”

Output: [“henry”, “is”, “giving”, “a”, “lecture”, “on”, “transformers”]

Pros/Cons:

- Can have difficulty trading off between vocabulary size and computational tractability
- Similar words e.g., “transformers” and “transformer” can get mapped to completely disparate representations
- Typos will typically be out-of-vocabulary (OOV)

Tokenization

Word-based Tokenizer:

Input: “Henry is givin’ a lectrue on transformers”

Output: [“henry”, “is”, <OOV>, “a”, <OOV>, “on”, “transformers”]

Pros/Cons:

- Can have difficulty trading off between vocabulary size and computational tractability
- Similar words e.g., “transformers” and “transformer” can get mapped to completely disparate representations
- Typos will typically be out-of-vocabulary (OOV)

Tokenization

Character-based Tokenizer:

Input: “Henry is givin’ a lectrue on transformers”

Output: [“h”, “e”, “n”, “r”, “y”, “i”, “s”, “g”, “i”, “v”, “i”, “n”, “ ’ ”, …]

Pros/Cons:

- Much smaller vocabularies but a lot of semantic meaning is lost…
- Sequences will be much longer than word-based tokenization, potentially causing computational issues
- Can do well on logographic languages e.g., 汉字

Tokenization

上海科技大学 ShanghaiTech University

Subword-based Tokenizer:

Input: “Henry is givin’ a lectrue on transformers”

Output: [“henry”, “is”, “giv”, “##in”, “ ‘ ”, “a”, “lec” “##true”, “on”, “transform”, “##ers”]

Pros/Cons:

- Split long or rare words into smaller, semantically meaningful components or subwords
- No out-of-vocabulary words – any non-subword token can be constructed from other subwords (always includ all characters as subwords)
- Examples algorithms for learning a subword tokenization:
  - Byte-Pair-Encoding (BPE), WordPiece, SentencePiece

Slide adapted from Henry Chai

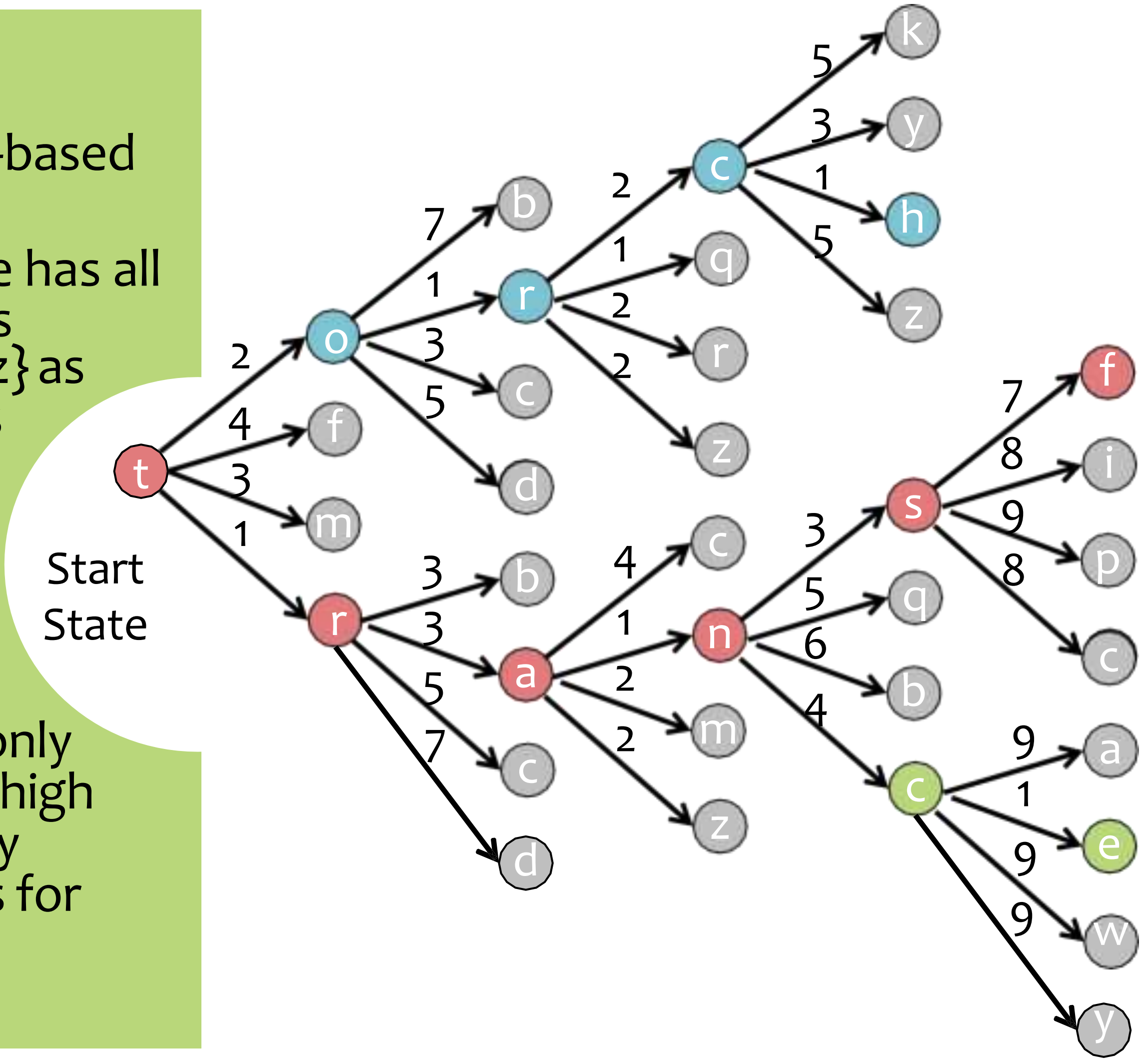
GREEDY DECODING FOR A LANGUAGE MODEL

Greedy Decoding for a Language Model

上海科技大学 ShanghaiTech University

Setup:

- Assume a character-based tokenizer
- Each node has all characters {a,b,c,…,z} as neighbors
- Here we only show the high probability neighbors for space



Goal:

- Search space consists of nodes (partial sentences) and weighted by negative log probability
- Goal is to find the highest probably (lowest negative log probability) path from root to a leaf

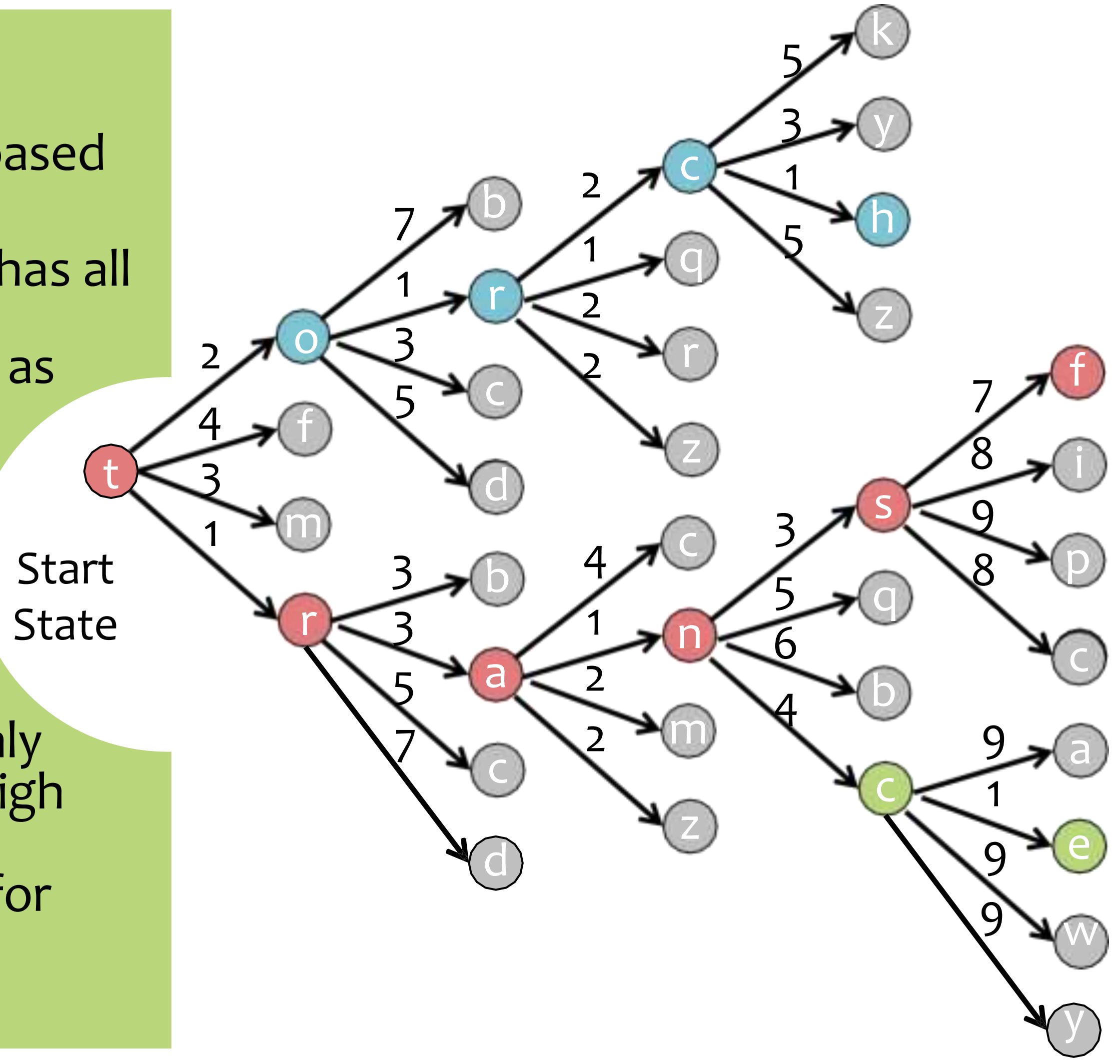
Greedy Search:

- At each node, selects the edge with lowest negative log probability
- **Heuristic** method of search (i.e. does *not* necessarily find the best path)
- Computation time: **linear** in max path length

Sampling from a Language Model

Setup:

- Assume a character-based tokenizer
- Each node has all characters {a,b,c,…,z} as neighbors
- Here we only show the high probability neighbors for space



Goal:

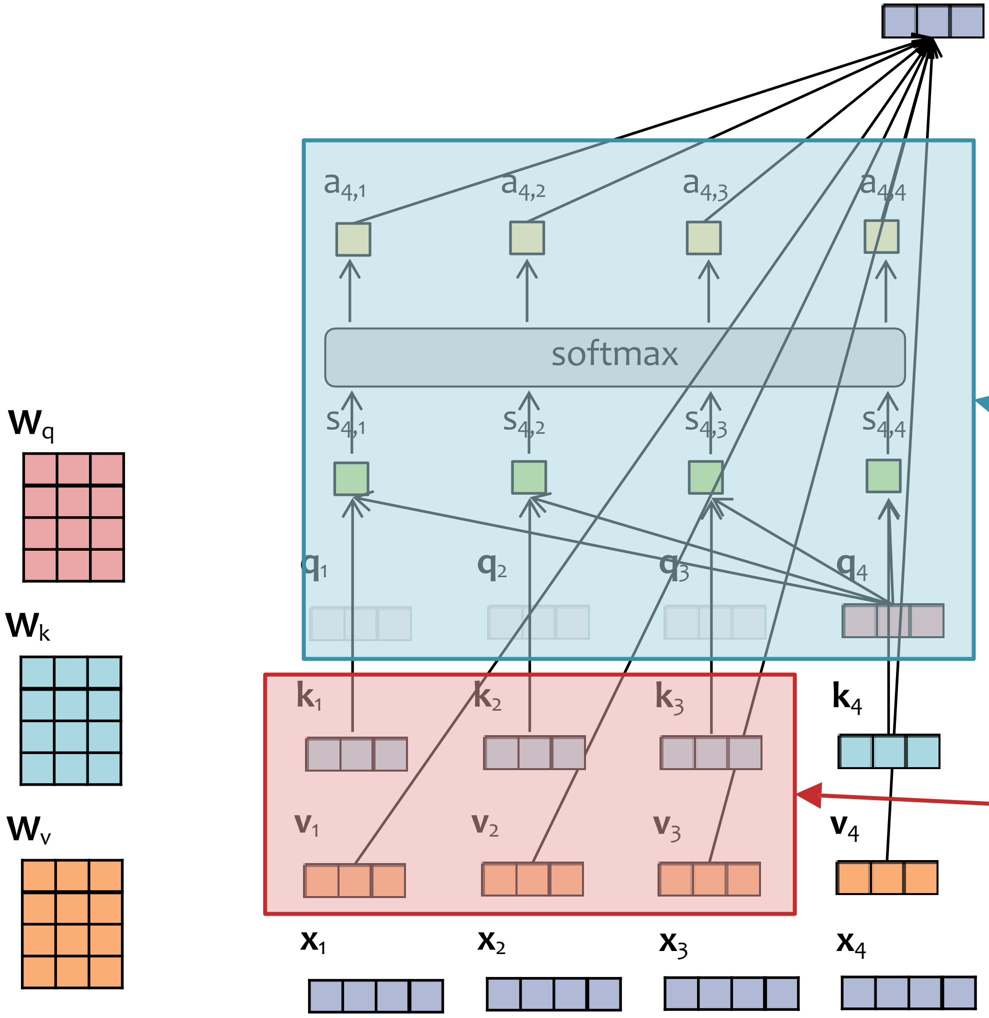
- Search space consists of nodes (partial sentences) and weighted by negative log probability
- Goal is to sample a path from root to a leaf with probability according to the probability of that path

Ancestral Sampling:

- At each node, randomly pick an edge with probability (converting from negative log probability)
- **Exact** method of sampling, assuming a locally normalized distribution (i.e. samples a path according to its total probability)
- Computation time: **linear** in max path length

Key-Value Cache

上海科技大学 ShanghaiTech University



$$x'_4 = \sum_{j=1}^{4} a_{4,j} v_j$$

$$\mathbf{a}_4 = \text{softmax}(\mathbf{s}_4)$$

$$s_{4,j} = \mathbf{k}_j^T \mathbf{q}_4 / \sqrt{d_k}$$

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j$$

$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j$$

$$\mathbf{v}_j = \mathbf{W}_v^T \mathbf{x}_j$$

- At each timestep, we reuse all previous keys and values (i.e. we need to cache them)
- But we can get rid of the queries, similarity scores, and attention weights (i.e. we can let them fall out of the cache)

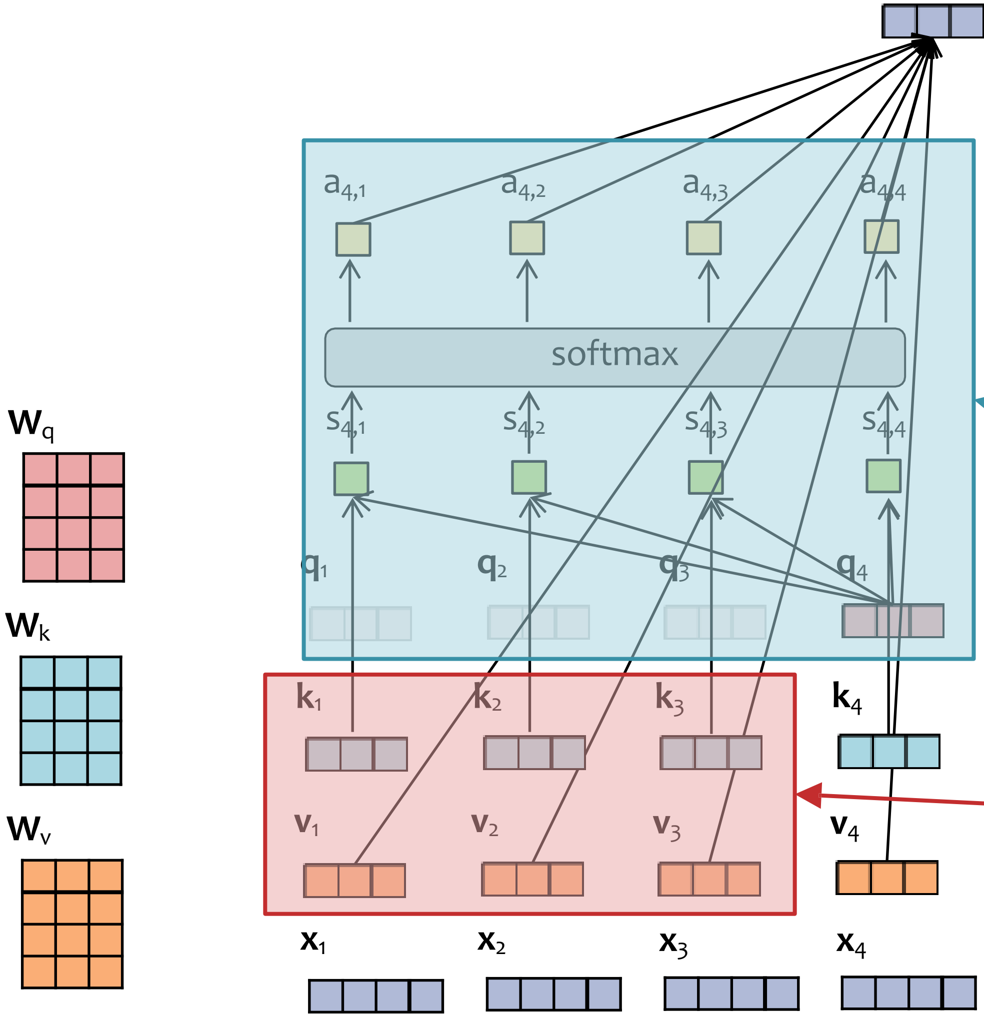
Discarded after this timestep

Computed for previous time-steps and reused for this timestep

Key-Value Cache

$$X'_t = A_t V = \mathrm{softmax}(Q_t K^T / \sqrt{d_k}) V$$

- At each timestep, we reuse all previous keys and values (i.e. we need to cache them)
- But we can get rid of the queries, similarity scores, and attention weights (i.e. we can let them fall out of the cache)



$$A_t = \mathrm{softmax}(S_t)$$

$$S_t = Q_t K^T / \sqrt{d_k}$$

$$Q_t = X_t W_q$$

$$K = X W_k$$

$$V = X W_v$$

$$X = [x_1, \ldots, x_t]^T$$

Discarded after this timestep

Computed for previous time-steps and reused for this timestep

上海科技大学
ShanghaiTech University

PRE-TRAINING VS. FINE-TUNING

Pre-Training vs. Fine-Tuning

上海科技大学 ShanghaiTech University

Definitions

Pre-training

- randomly initialize the parameters, then…
- *option A*: unsupervised training on very large set of unlabeled instances
- *option B*: supervised training on a very large set of labeled examples

Fine-tuning

- initialize parameters to values from pre-training
- (optionally), add a prediction head with a small number of randomly initialized parameters
- train on a specific task of interest by backprop

Example: Vision Models

Pre-training

- Example A: unsupervised autoencoder training on very large set of unlabeled images (e.g. MNIST digits)
- Example B: supervised training on a very large image classification dataset (e.g. ImageNet w/21k classes and 14M images)

Fine-tuning

- object detection, training on 200k labeled images from COCO
- semantic segmentation, training on 20k labeled images from ADE20k

Example: Language Models

Pre-training

- unsupervised pre-training by maximizing likelihood of a large set of unlabeled sentences such as…
- The Pile (800 Gb of text)
- Dolma (3 trillion tokens)

Fine-tuning

- MMLU benchmark: a few training examples from 57 different tasks ranging from elementary mathematics to genetics to law
- code generation, training on ~400 training examples from MBPP