EECS 487: Introduction to Natural Language Processing

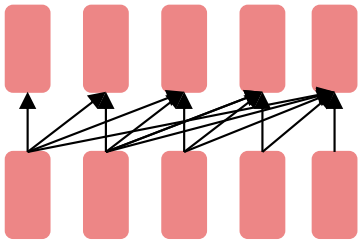
Instructor: Prof. Lu Wang
Computer Science and Engineering
University of Michigan
Webpage: web.eecs.umich.edu/~wangluxy

Big idea

- Many tasks can be turned into tasks of predicting words!

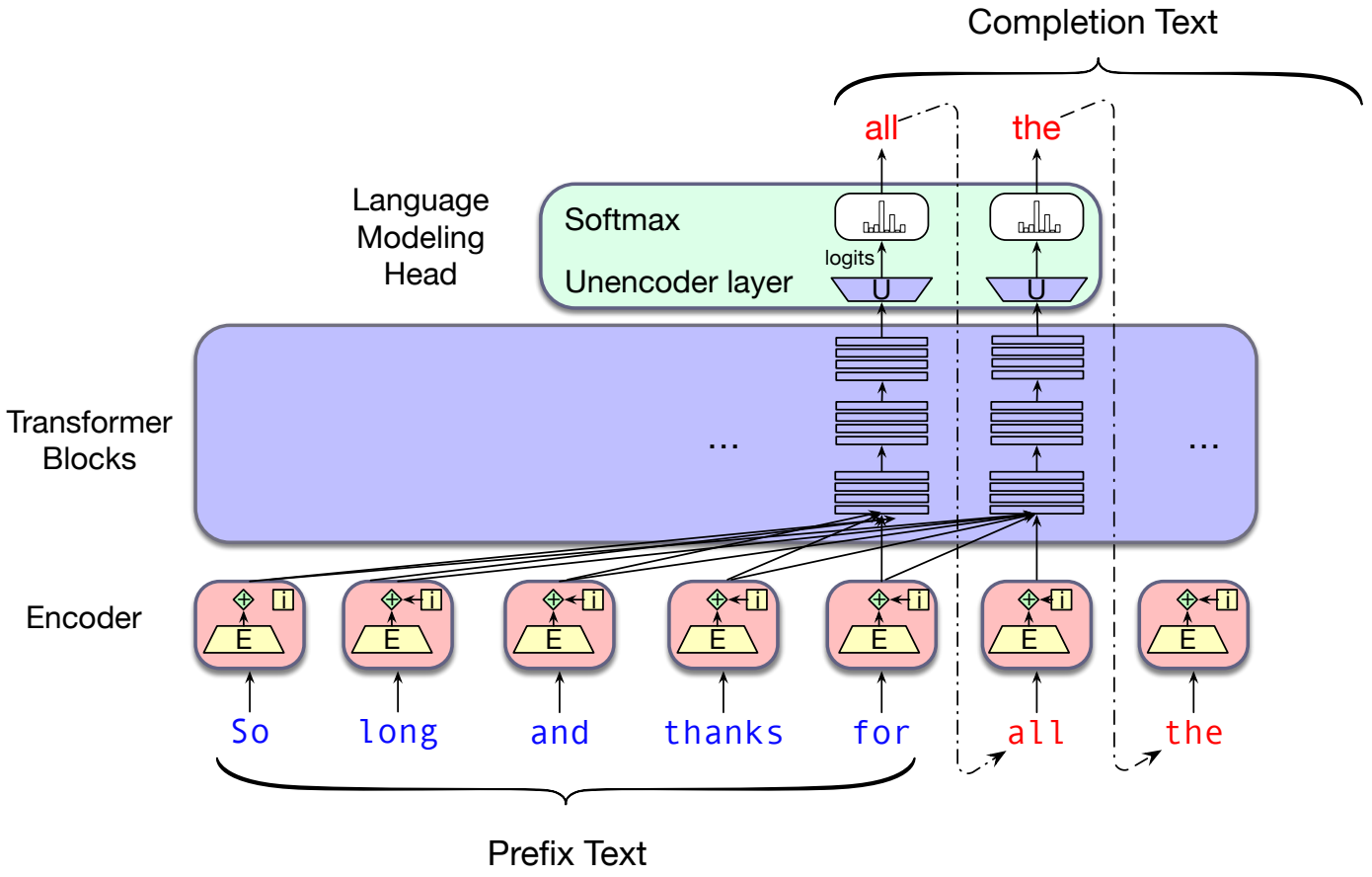
Let's focus on decoder-only models

- Also called:
- Causal LLMs
- Autoregressive LLMs
- Left-to-right LLMs



- Predict words left to right

Conditional Generation: Generating text conditioned on previous text!



Many practical NLP tasks can be cast as word prediction!

Sentiment analysis: “I like Jackie Chan”

1. We give the language model this string:
   `The sentiment of the sentence "I like Jackie Chan" is:`
2. And see what word it thinks comes next:

$$P(\text{positive}|\texttt{The sentiment of the sentence ‘‘I like Jackie Chan" is:})$$
$$P(\text{negative}|\texttt{The sentiment of the sentence ‘‘I like Jackie Chan" is:})$$

Framing lots of tasks as conditional generation

- QA: “Who wrote The Origin of Species”

1. We give the language model this string:

`Q: Who wrote the book ‘‘The Origin of Species"? A:`

2. And see what word it thinks comes next:

$P(w|$`Q: Who wrote the book ‘‘The Origin of Species"? A:`$)$

3. And iterate:

$P(w|$`Q: Who wrote the book ‘‘The Origin of Species"? A: Charles`$)$

Summarization

Original

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

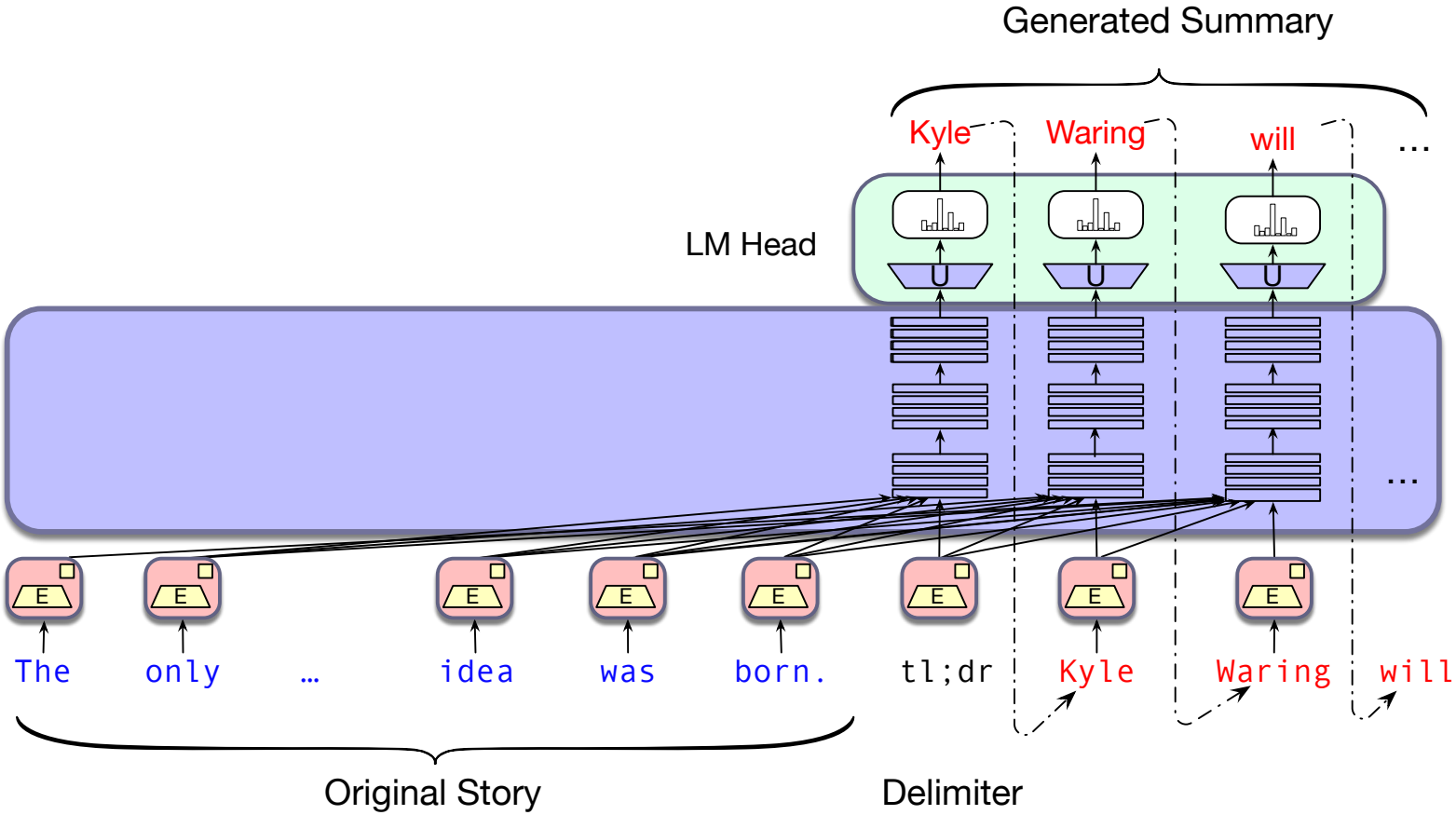
His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. [...]

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

LLMs for summarization (using tl;dr)



Outline

- Sampling for large language model (LLM) generation
- Pretraining data for LLMs
- Dealing with scale

Decoding and Sampling

- This task of choosing a word to generate based on the model's probabilities is called **decoding**.
- The most common method for decoding in LLMs: **sampling**.
- Sampling from a model's distribution over words:
  - choose random words according to their probability assigned by the model.
- After each token we'll sample words to generate according to their probability *conditioned on our previous choices*
  - A transformer language model will give the probability

Random sampling

$\mathrm{i} \leftarrow 1$

$w_i \sim \mathrm{p}(w)$

while w_i != EOS

$\quad \mathrm{i} \leftarrow \mathrm{i} + 1$

$\quad w_i \sim \mathrm{p}(w_i \mid w_{<i})$

Random sampling doesn't work very well

- Even though random sampling mostly generate sensible, high-probable words
- There are many odd, low-probability words in the tail of the distribution
- Each one is low- probability but added up they constitute a large portion of the distribution
- So they get picked enough to generate weird sentences

Factors in word sampling: **quality** and **diversity**

Emphasize **high-probability** words

+ **quality**: more accurate, coherent, and factual,

- diversity: boring, repetitive.

Emphasize **middle-probability** words

+ **diversity**: more creative, diverse,

- **quality**: less factual, incoherent

Top-k sampling:

1. Choose # of words *k*
2. For each word in the vocabulary *V* , use the language model to compute the likelihood of this word given the context $p(w_t \mid w_{<t})$
3. Sort the words by likelihood, keep only the top *k* most probable words.
4. Renormalize the scores of the *k* words to be a legitimate probability distribution.
5. Randomly sample a word from within these remaining *k* most-probable words according to its probability.

Top-p sampling (= nucleus sampling)

Holtzman et al., 2020

- Problem with top-*k:* *k* is fixed so may cover very different amounts of probability mass in different situations
- Idea: Instead, keep the top p percent of the probability mass
- Given a distribution $P(w_t \mid \mathbf{w}_{<t})$, the top-*p* vocabulary $V(p)$ is the smallest set of words such that

$$\sum_{w \in V^{(p)}} P(w|\mathbf{w}_{<t}) \geq p$$

Temperature sampling

- Reshape the distribution instead of truncating it
- Intuition from thermodynamics,

  a system at high temperature is flexible and can explore many possible states,

  a system at lower temperature is likely to explore a subset of lower energy (better) states.

- In **low-temperature sampling** ($\tau \leq 1$) we smoothly

  increase the probability of the most probable words

  decrease the probability of the rare words.

Temperature sampling

- Divide the logit by a temperature parameter τ before passing it through the softmax.

- Instead of

$$\mathbf{y} = \text{softmax}(u)$$

- We do

$$\mathbf{y} = \text{softmax}(u/\tau)$$

Temperature sampling

$$\mathbf{y} = \text{softmax}(u/\tau) \quad 0 \leq \tau \leq 1$$

Why does this work?

- When τ is close to 1 the distribution doesn't change much.
- The lower τ is, the larger the scores being passed to the softmax
- Softmax pushes high values toward 1 and low values toward 0.
- Large inputs pushes high-probability words higher and low probability word lower, making the distribution more greedy.
- As τ approaches 0, the probability of most likely word approaches 1

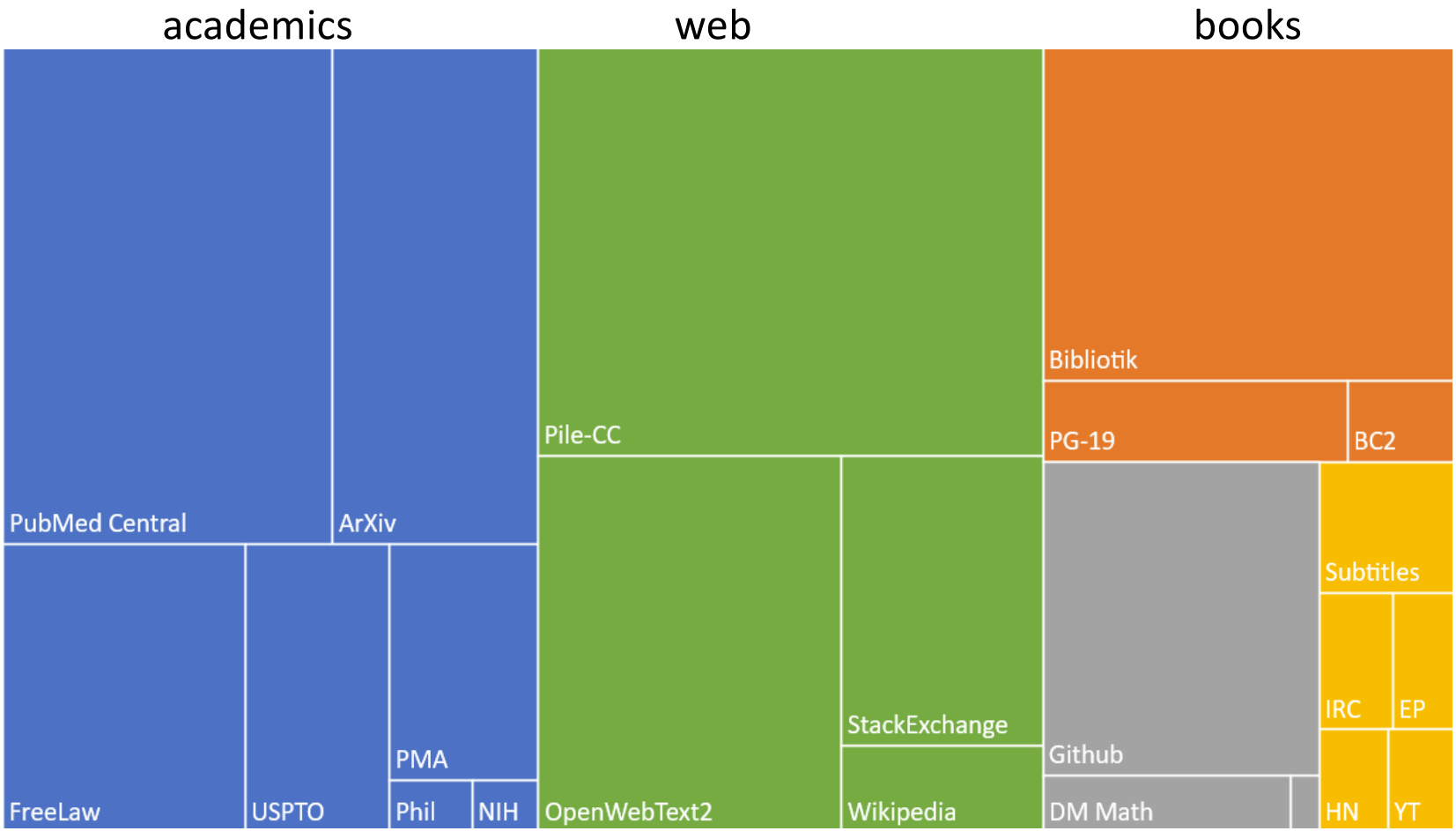
Outline

- Sampling for large language model (LLM) generation
- Pretraining data for LLMs
- Dealing with scale

LLMs are mainly trained on the web

- Common crawl, snapshots of the entire web produced by the non-profit Common Crawl with billions of pages
- Colossal Clean Crawled Corpus (C4; Raffel et al. 2020), 156 billion tokens of English, filtered
- What's in it? Mostly patent text documents, Wikipedia, and news sites

The Pile: a pretraining corpus



Filtering for quality and safety

Quality is subjective

- Many LLMs attempt to match Wikipedia, books, particular websites
- Need to remove boilerplate, adult content
- Deduplication at many levels (URLs, documents, even lines)

Safety also subjective

- Toxicity detection is important, although that has mixed results
- Can mistakenly flag data written in dialects like African American English

What does a model learn from pretraining?

- There are canines everywhere! One dog in the front room, and two dogs
- It wasn't just big it was enormous
- The author of "A Room of One's Own" is Virginia Woolf
- The doctor told me that he
- The square root of 4 is 2

Big idea

- Text contains enormous amounts of knowledge
- Pretraining on lots of text with all that knowledge is what gives language models their ability to do so much

But there are problems with scraping from the web

Copyright: much of the text in these datasets is copyrighted

- Not clear if fair use doctrine in US allows for this use
- This remains an open legal question

Data consent

- Website owners can indicate they don't want their site crawled

Privacy:

- Websites can contain private IP addresses and phone numbers

Outline

- Sampling for large language model (LLM) generation
- Pretraining data for LLMs
- Dealing with scale

Scaling Laws

LLM performance depends on

- Model size: the number of parameters not counting embeddings
- Dataset size: the amount of training data
- Compute: Amount of compute (in FLOPS or etc)
- can improve a model by adding parameters (more layers, wider contexts), more data, or training for more iterations

The performance of a large language model (the loss) scales as a power-law with each of these three

Scaling Laws

- Loss *L* as a function of # parameters *N*, dataset size *D*, compute budget *C (*if other two are held constant)

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}$$

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}$$

$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C}$$

Scaling laws can be used early in training to predict what the loss would be if we were to add more data or increase model size.

Number of non-embedding parameters N

$$
\begin{aligned}
N &\approx 2\, d\, n_{\text{layer}}(2\, d_{\text{attn}} + d_{\text{ff}}) \\
&\approx 12\, n_{\text{layer}}\, d^2 \\
&\quad (\text{assuming } \; d_{\text{attn}} = d_{\text{ff}}/4 = d)
\end{aligned}
$$

Thus GPT-3, with n = 96 layers and dimensionality d = 12288, has 12 × 96 × 122882 ≈ 175 billion parameters.

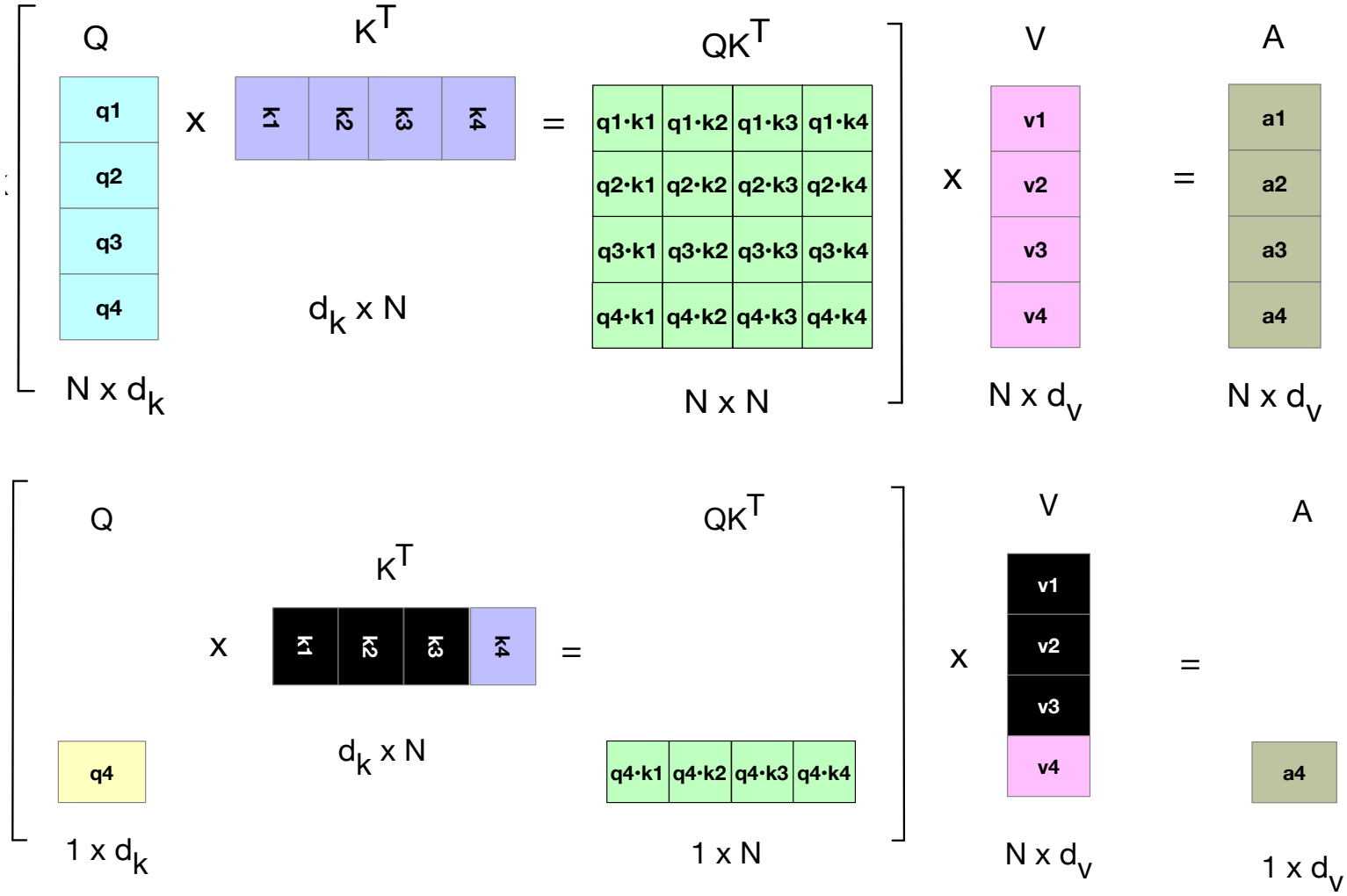
KV Cache

- In training, we can compute attention very efficiently in parallel:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

- But not at inference! We generate the next tokens **one at a time!**
- For a new token x, need to multiply by W^Q, W^K, and W^V to get query, key, values
- But don't want to **recompute** the key and value vectors for all the prior tokens $x_{<i}$
- Instead, store key and value vectors in memory in the KV cache, and then we can just grab them from the cache

KV Cache


Q
K^T
QK^T
V
A
q1
q2
q3
q4
x
k1
k2
k3
k4
=
q1·k1 q1·k2 q1·k3 q1·k4
q2·k1 q2·k2 q2·k3 q2·k4
q3·k1 q3·k2 q3·k3 q3·k4
q4·k1 q4·k2 q4·k3 q4·k4
x
v1
v2
v3
v4
=
a1
a2
a3
a4
N x d_k
d_k x N
N x N
N x d_v
N x d_v
Q
K^T
QK^T
V
A
q4
x
k1
k2
k3
k4
=
q4·k1 q4·k2 q4·k3 q4·k4
x
v1
v2
v3
v4
=
a4
1 x d_k
d_k x N
1 x N
N x d_v
1 x d_v

Parameter-Efficient Finetuning

Adapting to a new domain by continued pretraining (finetuning) is a problem with huge LLMs.

- Enormous numbers of parameters to train
- Each pass of batch gradient descent has to backpropagate through many many huge layers.
- Expensive in processing power, in memory, and in time.

Instead, **parameter-efficient fine tuning** (PEFT)

- Efficiently select a subset of parameters to update when finetuning.
- E.g., freeze some of the parameters (don't change them),
- And only update some a few parameters.

LoRA (Low-Rank Adaptation)

- Transformers have many dense matrix multiply layers
  - Like $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$, $\mathbf{W}^O$ layers in attention
- Instead of updating these layers during finetuning,
  - Freeze these layers
  - Update a low-rank approximation with fewer parameters.

LoRA

- Consider a **matrix W** (shape [$N \times d$]) that needs to be updated during finetuning via gradient descent.
  - Normally updates are ΔW (shape [$N \times d$])
- In LoRA, we freeze W and update instead a low-rank decomposition of W:
  - A of shape [$N \times r$],
  - B of shape [$r \times d$], r is very small (like 1 or 2)
  - That is, during finetuning we update A and B instead of W.
  - Replace $W + \Delta W$ with $W + BA$.

Forward pass: instead of

$$\mathbf{h} = \mathbf{xW}$$

We do

$$\mathbf{h} = \mathbf{xW} + \mathbf{xAB}$$

LoRA