Machine Learning Department
School of Computer Science
Carnegie Mellon University

Pretraining vs. finetuning + Modern Transformers

(RoPE, GQA, Longformer)

Matt Gormley & Aran Nayebi

Lecture 4

Sep. 8, 2025

Reminders

- Homework 0: PyTorch + Weights & Biases
  - Out: Wed, Aug 27
  - Due: Mon, Sep 8 at 11:59pm
- Quiz 1: Wed, Sep 10
- Homework 1: Generative Models of Text
  - Out: Mon, Sep 8
  - Due: Mon, Sep 22 at 11:59pm

Two parts: Deep Learning and Language Modeling

Recap So Far

Deep Learning

- AutoDiff
  - is a tool for **computing gradients** of a differentiable function, b = f(a)
  - the key building block is a **module** with a forward() and backward()
  - sometimes define f as **code** in forward() by chaining existing modules together
- Computation Graphs
  - are another way to define f (more conducive to slides)
  - so far, we saw two (deep) computation graphs
    - 1) RNN-LM
    - 2) Transformer-LM
    - (Transformer-LM was kind of complicated)

Language Modeling

- key idea: condition on previous words to **sample the next word**
- to define the **probability** of the next word…
  - … n-gram LM uses collection of massive 50k-sided **dice**
  - … RNN-LM or Transformer-LM use a **neural network**

- Learning an LM
  - n-gram LMs are easy to learn: just **count** co-occurrences!
  - a RNN-LM / Transformer-LM is trained by optimizing an objective function with SGD; compute gradients with AutoDiff

PRE-TRAINING VS. FINE-TUNING

The Start of Deep Learning

- The architectures of modern deep learning have a long history:
  - 1960s: Rosenblatt's 3-layer multi-layer perceptron, ReLU)
  - 1970-80s: RNNs and CNNs
  - 1990s: linearized self-attention
- The spark for deep learning came in 2006 thanks to **pre-training** (e.g., Hinton & Salakhutdinov, 2006)

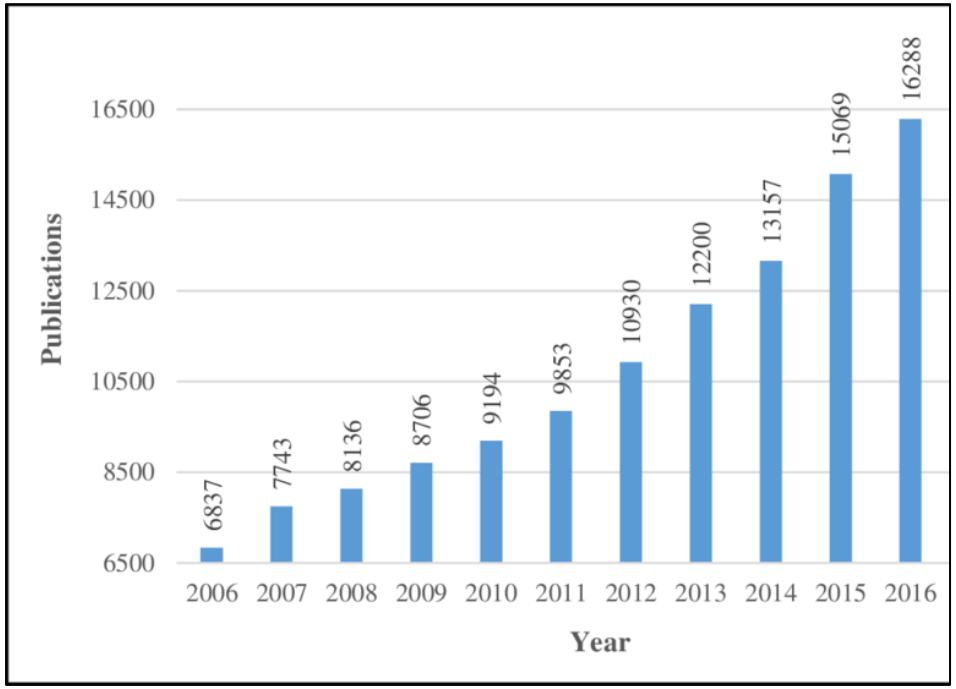


Figure from Vargas et al. (2017)

Pre-Training vs. Fine-Tuning

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

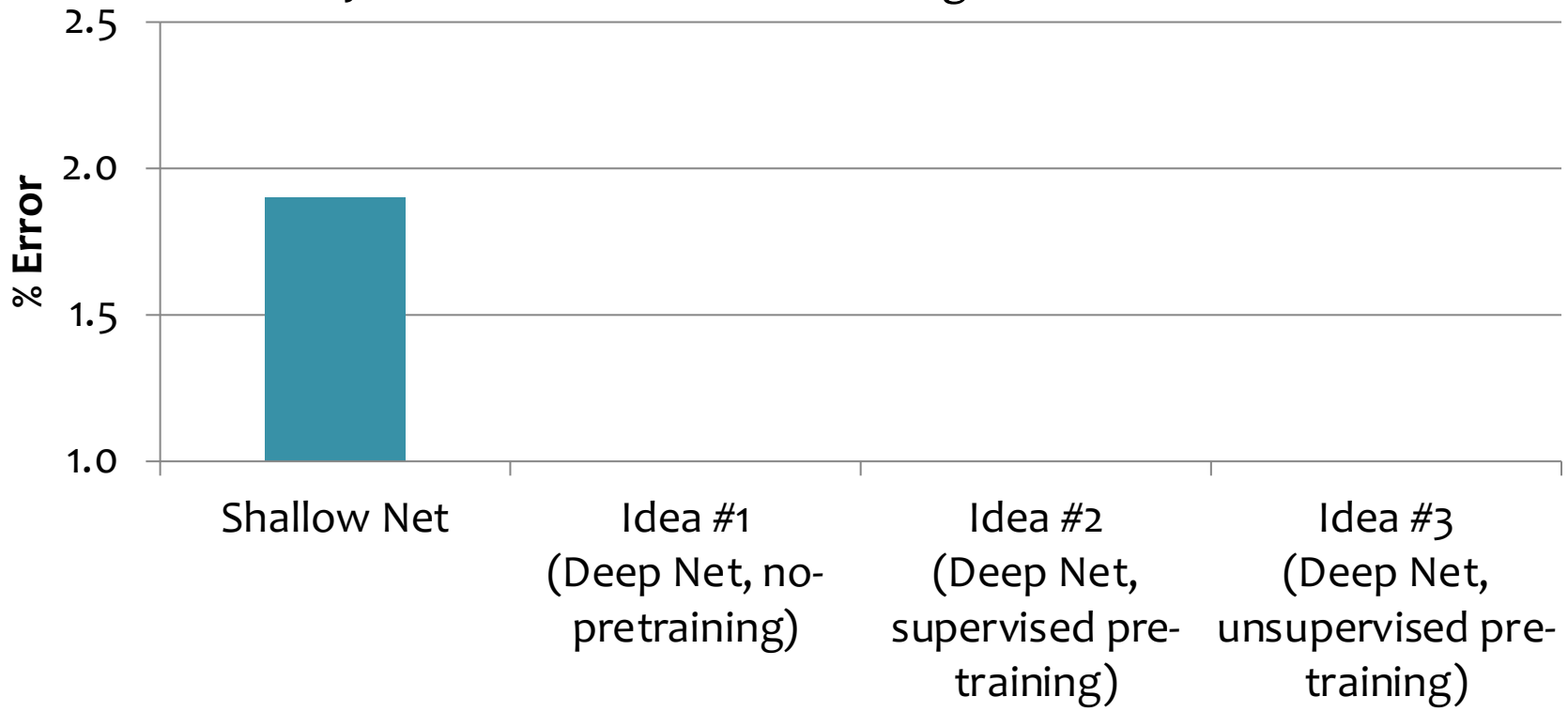
- unsupervised pre-training by maximizing likelihood of a large set of unlabeled sentences such as…
- The Pile (800 Gb of text)
- Dolma (3 trillion tokens)

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

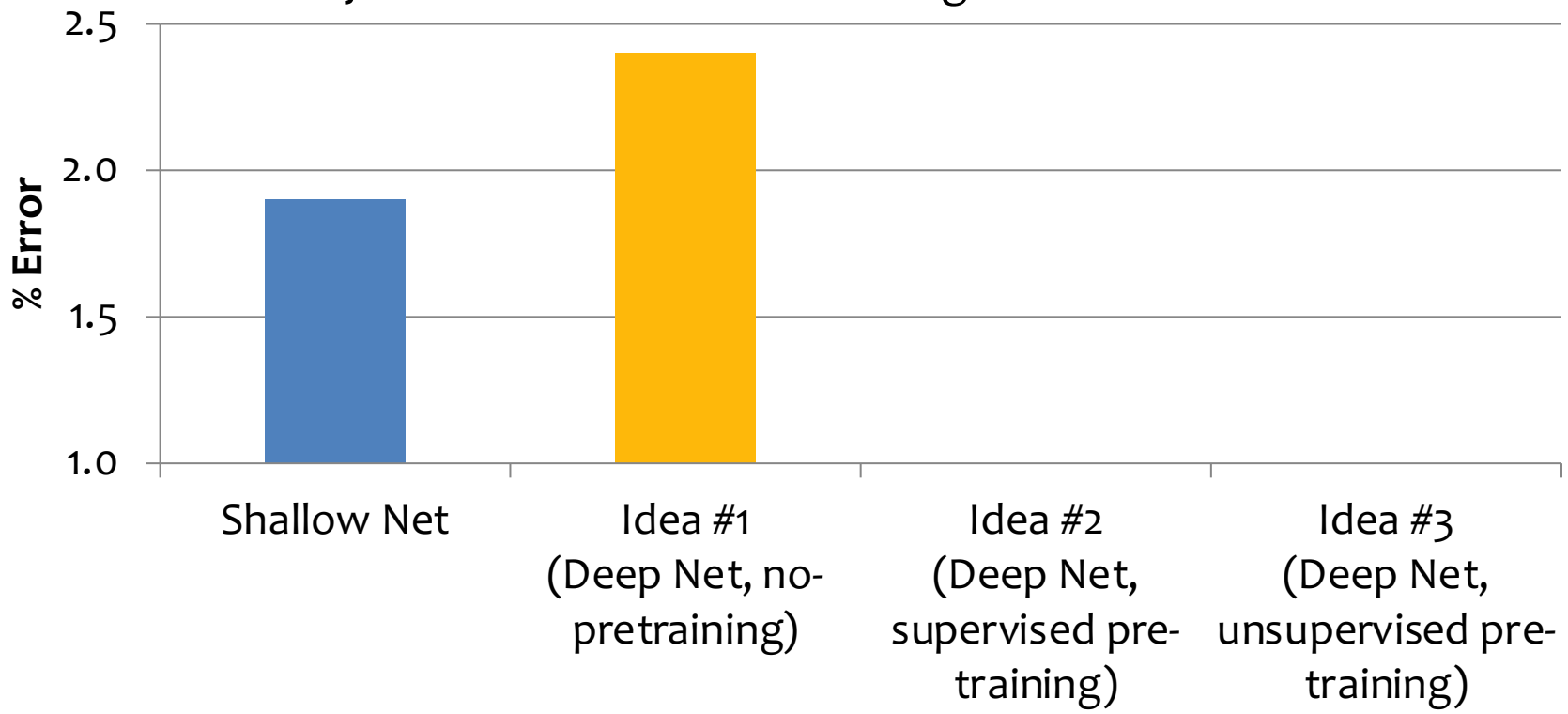
Pre-Training and Fine-Tuning on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)
- Some methods first do pre-training
- Every method includes fine-tuning on labeled data



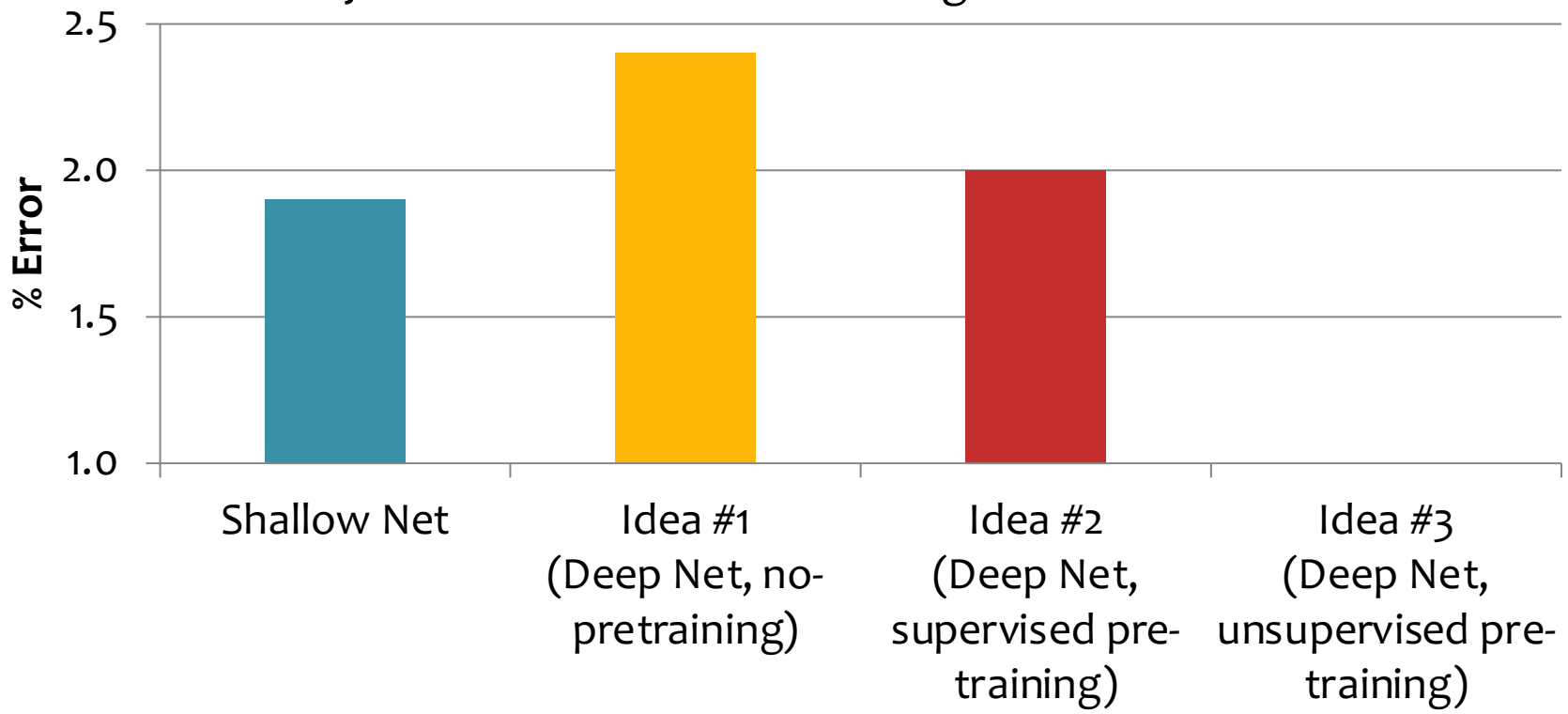
Pre-Training and Fine-Tuning on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)
- Some methods first do pre-training
- Every method includes fine-tuning on labeled data



Pre-Training and Fine-Tuning on MNIST

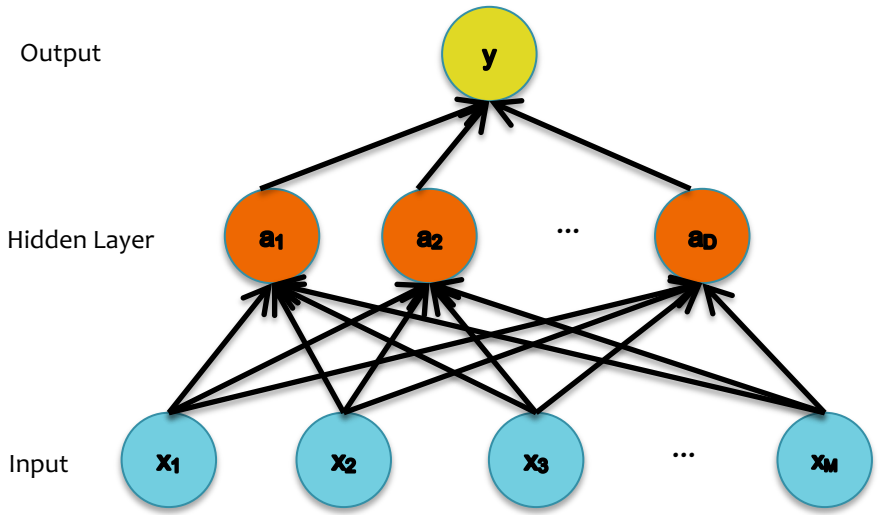
- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)
- Some methods first do pre-training
- Every method includes fine-tuning on labeled data



Unsupervised Autoencoder Pre-Training for Vision

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

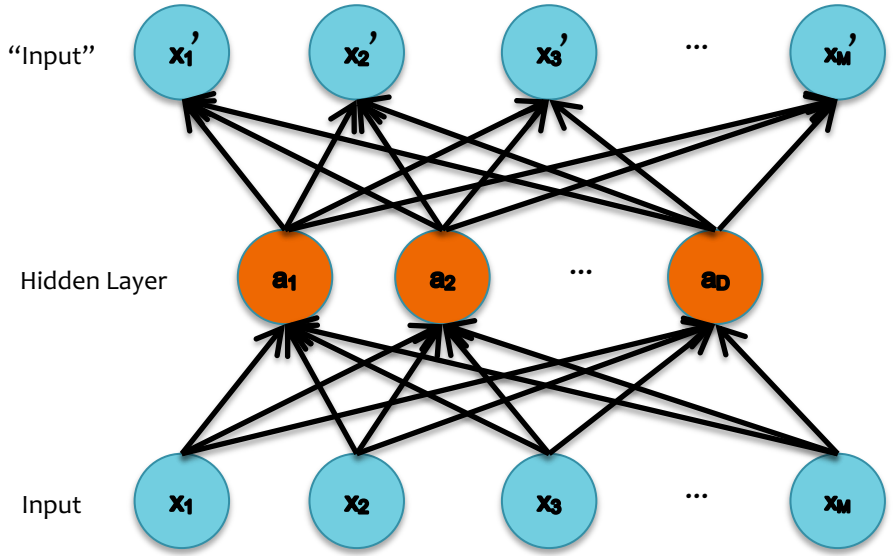


Unsupervised Autoencoder Pre-Training for Vision

Unsupervised pre-training of the first layer:

- What should it predict?
- What else do we observe?
- **The input!**

This topology defines an Auto-encoder.



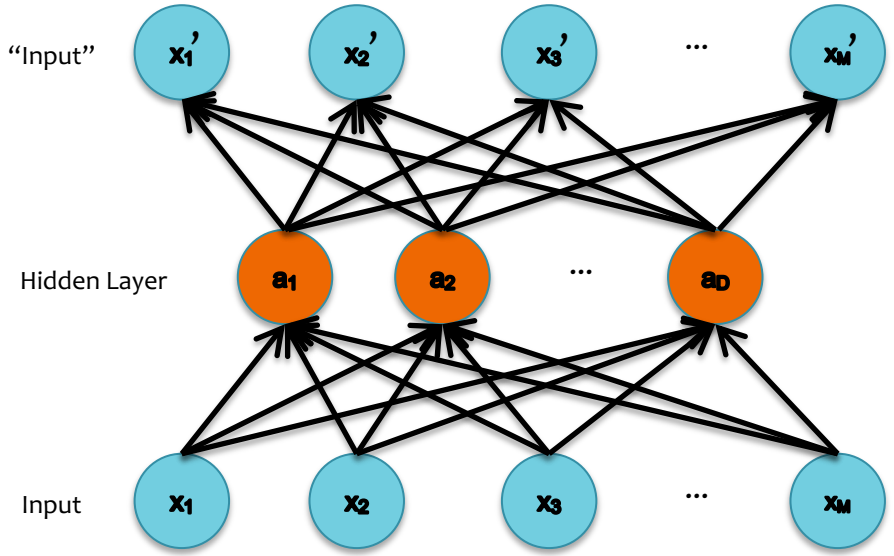
Unsupervised Autoencoder Pre-Training for Vision

Key idea: Encourage z to give small reconstruction error:

- x' is the *reconstruction* of x
- Loss = $|| x - \text{DECODER}(\text{ENCODER}(x)) ||^2$
- Train with the same backpropagation algorithm for 2-layer Neural Networks with x_m as both input and output.

DECODER: $x' = h(W'z)$

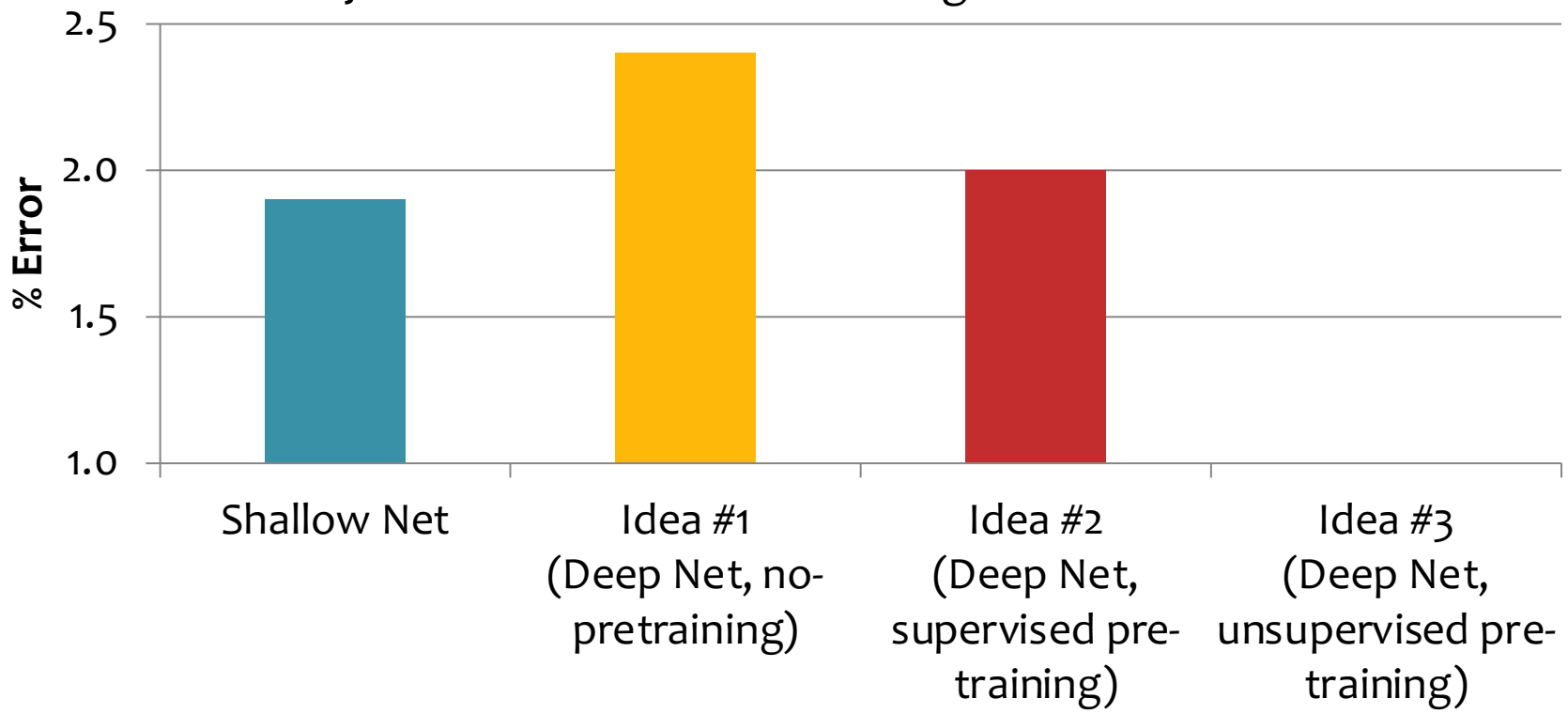
ENCODER: $z = h(Wx)$



Slide adapted from Raman Arora

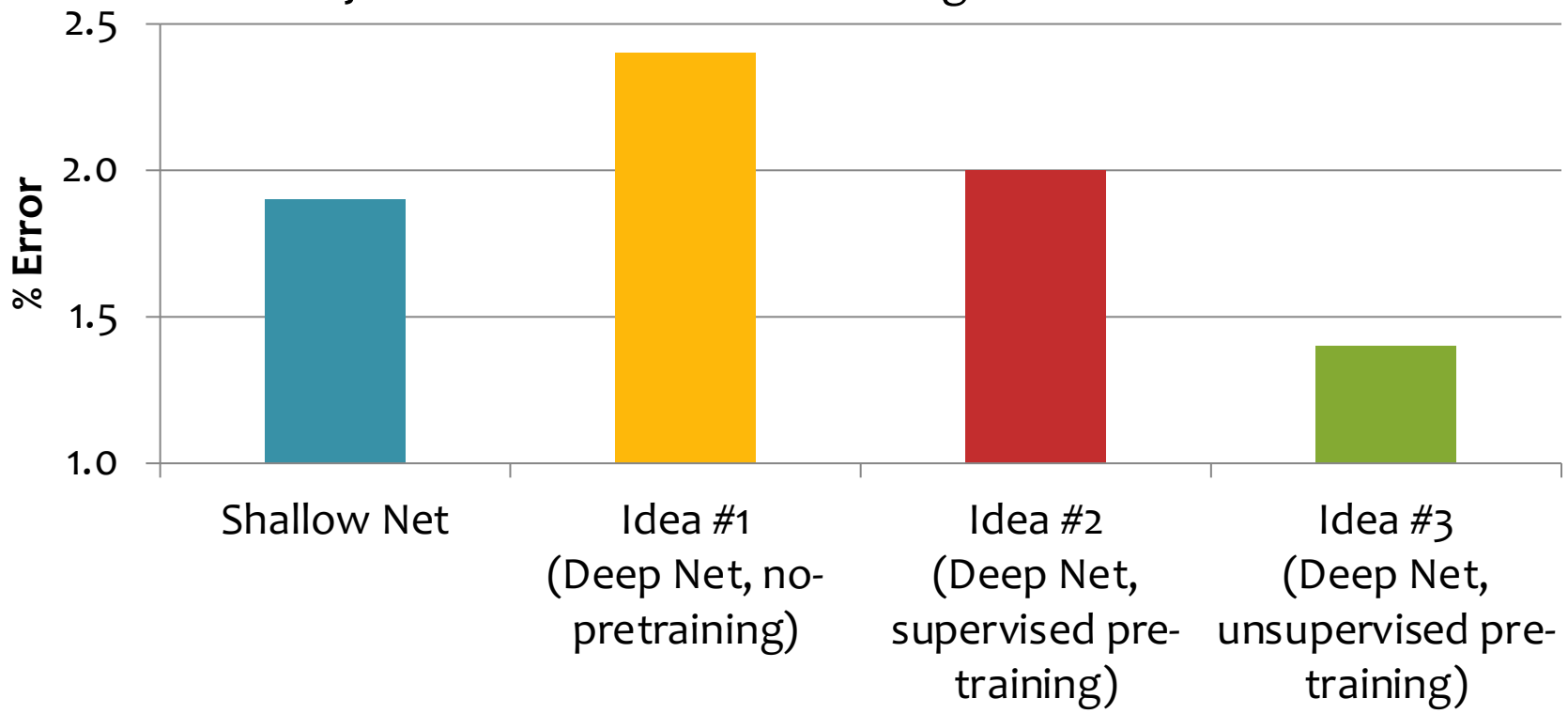
Pre-Training and Fine-Tuning on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)
- Some methods first do pre-training
- Every method includes fine-tuning on labeled data



Pre-Training and Fine-Tuning on MNIST

- Results from Bengio et al. (2006) on MNIST digit classification task
- Percent error (lower is better)
- Some methods first do pre-training
- Every method includes fine-tuning on labeled data



Supervised Pre-Training for Vision

- Nowadays, we tend to just do supervised pre-training on a massive labeled dataset
- Vision Transformer's success was largely due to using a much larger pre-training dataset

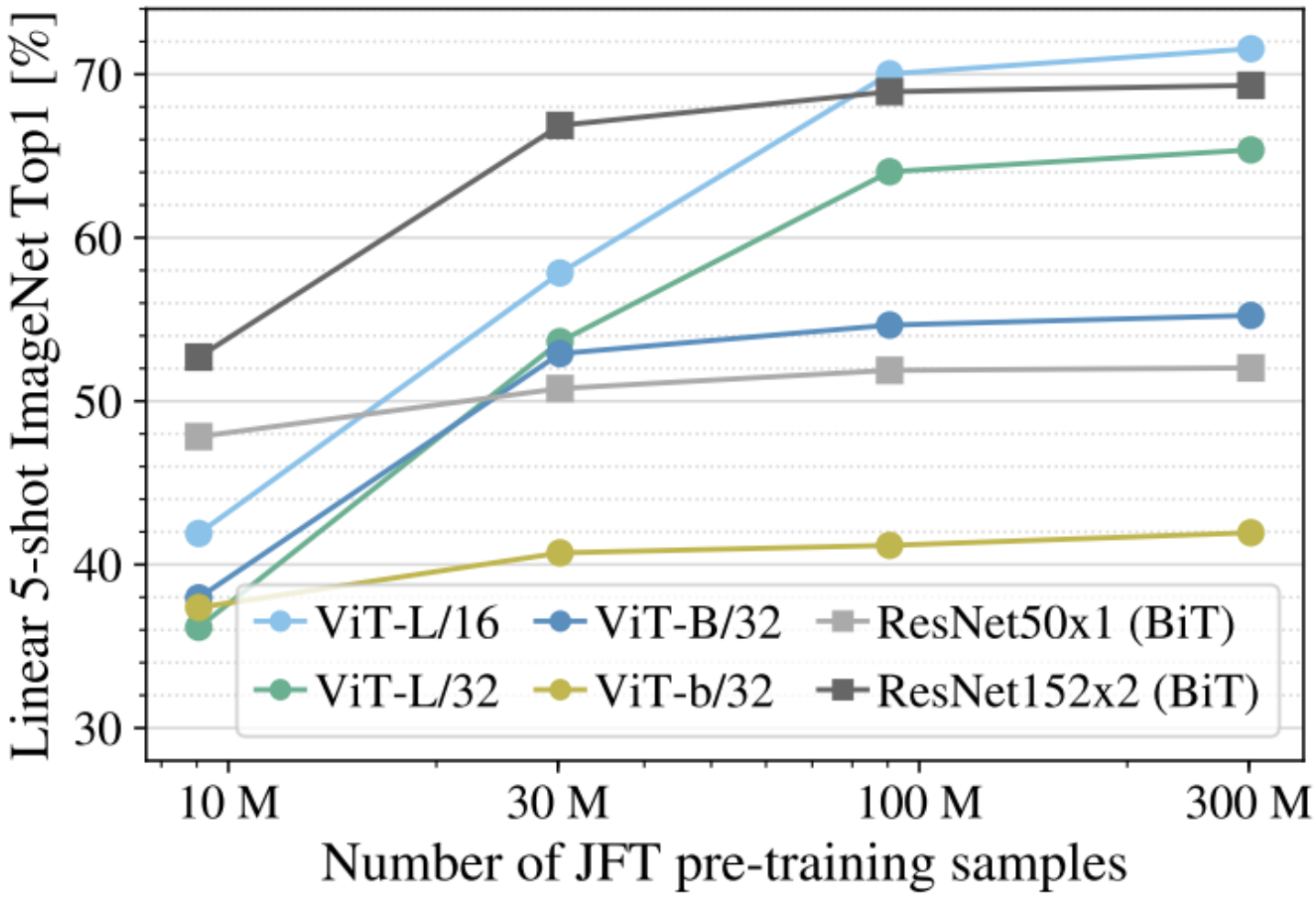


Figure from https://arxiv.org/pdf/2010.11929

Pre-Training vs. Fine-Tuning

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

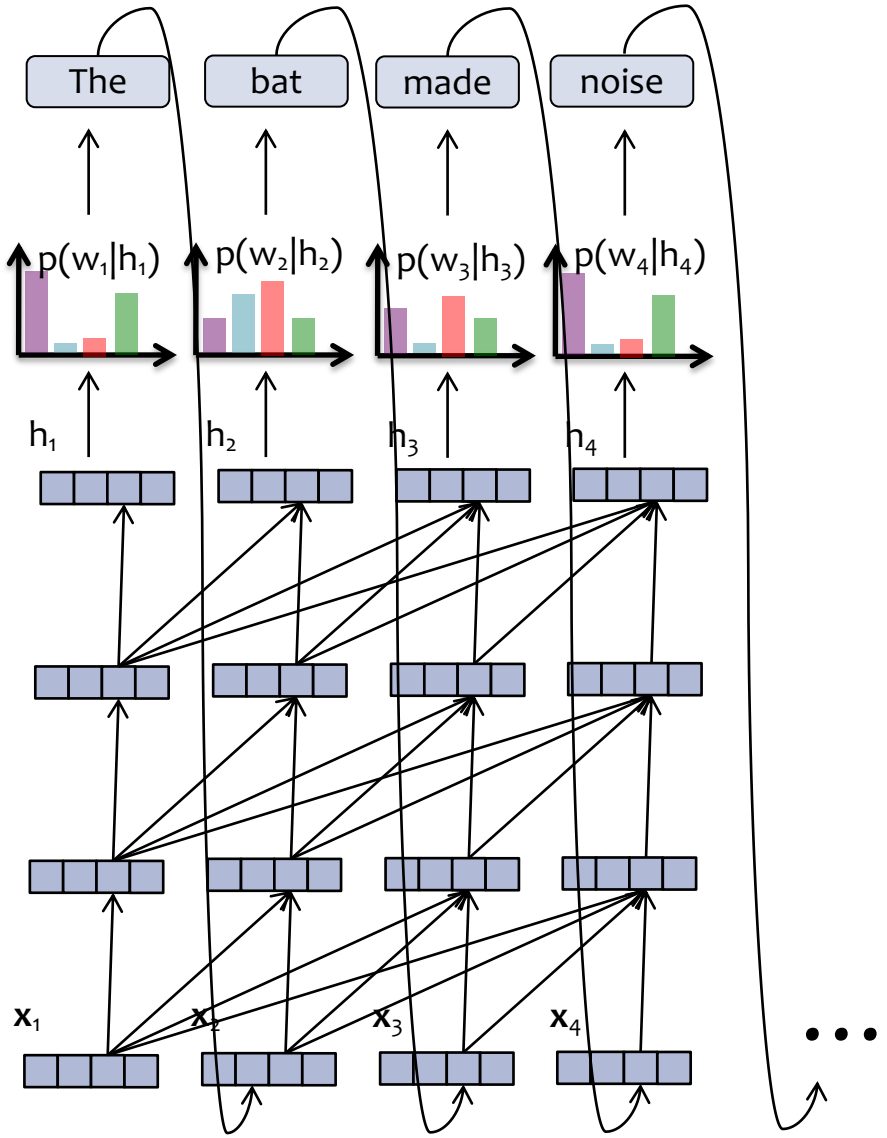
Pre-training

- 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

Unsupervised Pre-Training for an LLM



Generative pre-training for a deep language model:

- each training example is an (unlabeled) sentence
- the objective function is the likelihood of the observed sentence

Practically, we can **batch** together many such training examples to make training more efficient

Training Data for LLMs

GPT-3 Training Data:

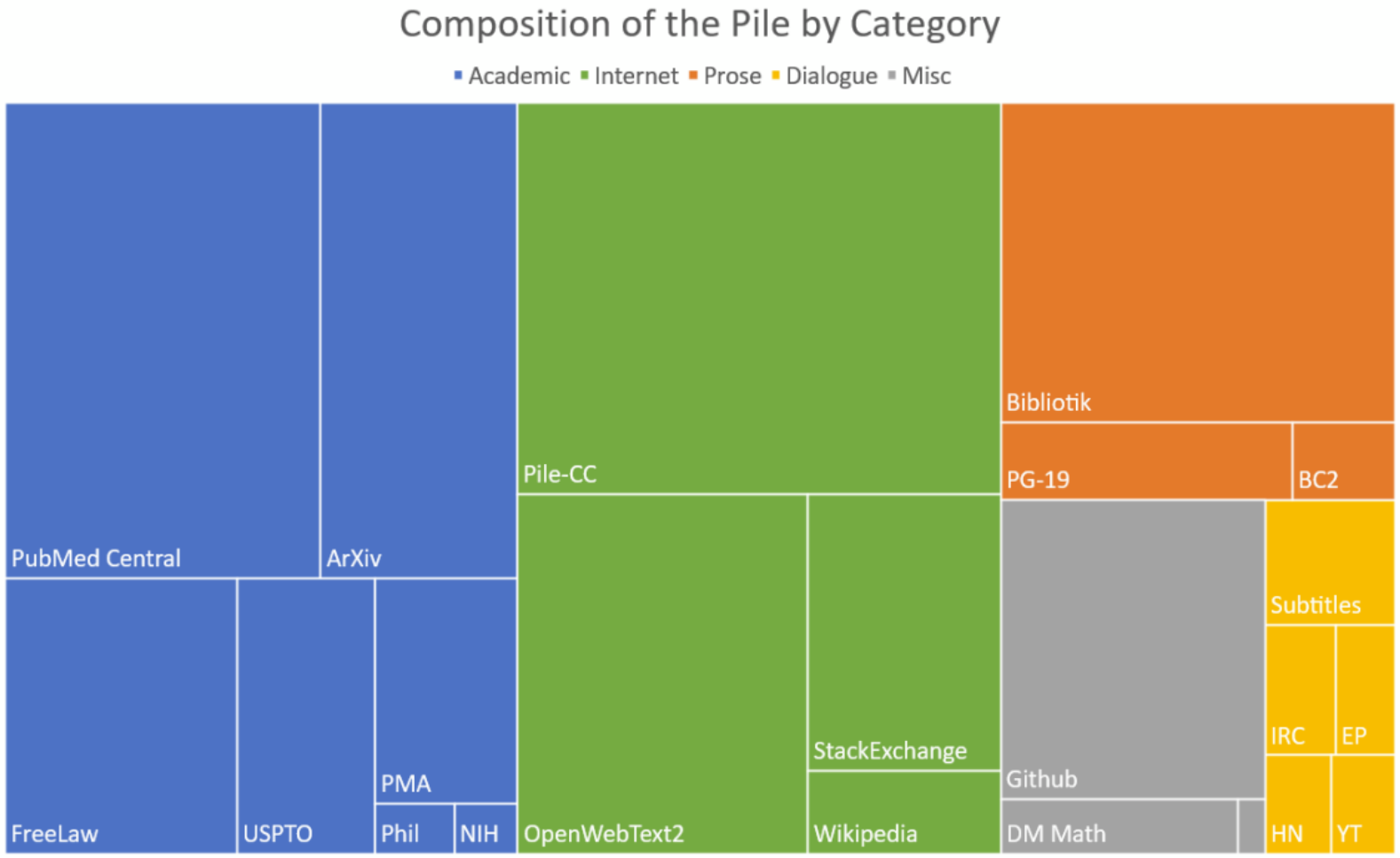
Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table from http://arxiv.org/abs/2005.14165

Training Data for LLMs

The Pile:

- An open source dataset for training language models
- Comprised of 22 smaller datasets
- Favors high quality text
- 825 Gb ≈ 1.2 trillion tokens



Pre-Training vs. Fine-Tuning

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)

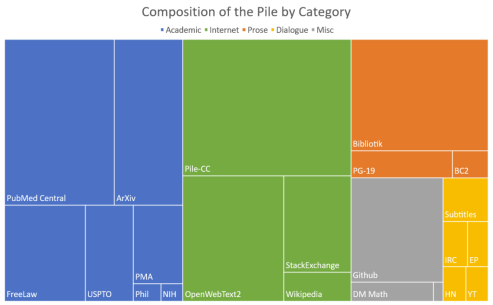
Fine-tuning

- 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

LLM Training Pipeline

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)



Mid-training

- just like pre-training but on higher-quality data
- Dolmino Mix 1124 (100B – 300B tokens)

Name	Category	Tokens	Bytes (uncompressed)	Documents	License
DCLM	HQ Web Pages	752B	4.56TB	606M	CC-BY-4.0
Flan	HQ Web Pages	17.0B	98.2GB	57.3M	ODC-BY
Pes2o	STEM Papers	58.6B	413GB	38.8M	ODC-BY
Wiki	Encyclopedic	3.7B	16.2GB	6.17M	ODC-BY
StackExchange	CodeText	1.26B	7.72GB	2.48M	CC-BY-SA-{2.5, 3.0, 4.0}
TuluMath	Synth Math	230M	1.03GB	220K	ODC-BY
DolminoSynthMath	Synth Math	28.7M	163MB	725K	ODC-BY
TinyGSM-MIND	Synth Math	6.48B	25.52GB	17M	ODC-BY
MathCoder2	Synth Math	3.87B	18.48GB	2.83M	Apache 2.0
Metamath-owmfilter	Math	84.2M	741MB	383K	CC-BY-SA-4.0
CodeSearchNet-owmfilter	Math	1.78M	29.8MB	7.27K	ODC-BY
GSM8K	Math	2.74M	25.3MB	17.6K	MIT
Total		843B	5.14TB	732M	ODC-BY

Post-Training

- some combination of:
  - instruction tuning
  - RLHF / DPO
  - Reinforcement learning with verifiable rewards (RLVR)
  - safety tuning
  - task-specific fine-tuning
- usually involves small quantities of supervised data

MODERN TRANSFORMER MODELS

Modern Tranformer Models

- PaLM (Oct 2022)
  - *Parameters*: 540B parameters
  - *Licensing*: closed source
  - *Model*:
    - SwiGLU instead of ReLU, GELU, or Swish
    - **multi-query attention** (MQA) instead of multi-headed attention
    - **rotary position embeddings**
    - **shared input-output embeddings** instead of separate parameter matrices
  - *Training*: **Adafactor** on 780 billion tokens
- Llama-1 (Feb 2023)
  - *Parameters*: models of various parameter sizes: 7B, 13B, 32B, 65B
  - *Licensing*: semi-open source
  - Llama-13B outperforms GPT-3 on average
  - *Model* compared to GPT-3:
    - **RMSNorm** on inputs instead of LayerNorm on outputs
    - **SwiGLU** activation function instead of ReLU
    - **rotary position embeddings (RoPE)** instead of absolute
  - *Training*: **AdamW** on 1.0 – 1.4 trillion tokens
- Falcon (June - Nov 2023)
  - *Parameters*: models of size 7B, 40B, 180B
  - *Licensing*: first fully open source model, Apache 2.0
  - *Model* compared to Llama-1:
    - (GQA) instead of multi-headed attention (MHA) or **grouped query attention multi-query attention** (MQA)
    - **rotary position embeddings** (worked better than Alibi)
    - **GeLU** instead of SwiGLU
  - *Training*: AdamW on up to 3.5 trillion tokens for 180B model, using **z-loss** for stability and **weight decay**
- Llama-2 (Aug 2023)
  - *Parameters*: collection of models of varying parameter sizes: 7B, 13B, 70B.
  - introduced Llama 2-Chat, fine-tuned as a dialogue agent
  - *Model* compared to Llama-1:
    - **grouped query attention (GQA)** instead of multi-headed attention (MHA)
    - context length of 4096 instead of 2048
  - *Training*: AdamW on 2.0 trillion tokens
- Mistral (Oct 2023)
  - *Parameters*: 7B
  - introduced Mistral 7B – Instruct, fine-tuned as a dialogue agent
  - outperforms Llama-2 13B on average
  - *Licensing*: truly open source: Apache 2.0 license
  - *Model* compared to Llama-2
    - **sliding window attention** (with W=4096) and grouped-query attention (GQA) instead of just GQA
    - context length of 8192 instead of 4096 (can generate sequences up to length 32K)
    - **rolling buffer cache** (grow the KV cache and the overwrite position i into position i mod W)
  - variant **Mixtral** offers a **mixture of experts** (roughly 8 Mistral models)

In this section we'll look at four techniques:

1. key-value cache (KV cache)
2. rotary position embeddings (RoPE)
3. grouped query attention (GQA)
4. sliding window attention

Modern Tranformer Models

- Qwen (Sep 2023)
- OLMo (Feb 2024)
  - *Parameters*: 1B and 7B
  - *Licensing*: fully open; includes weights, data, training code
  - *Model*:
    - no biases (following LLaMa, PaLM)
    - non-parametric layer norm, SwiGLU
    - **rotary position embeddings**
    - QKV clipping
  - *Training*: AdamW on 2T tokens, with **4M tokens per batch**
- Gemma (Feb 2024)
  - *Parameters*: 2B, 7B
  - *Licensing*: open weights
  - *Model*:
    - Multi-query attention (related to GQA, but different)
    - RoPE
    - GeGLU, RMSNorm
  - *Training*: 3T - 6T tokens
- Gemma 2 (June 204)
  - *Parameters*: 2B, 9B, 27B
  - *Licensing*: open weights
  - *Model*:
    - GQA, RoPE
    - pre-norm and post-norm with RMSNorm, QK-Norm
    - interleaved local sliding window attention with global self-attention
  - *Training*: 2T - 13T tokens for different model sizes
- Qwen 2 (July 2024)
- LLaMa 3 and 3.1 (Apr 2024)
  - *Parameters*: 8B, 70B, and 405B
  - *Model* compared to Llama-2
    - GQA uses fewer key-value heads
    - Context length significantly increased—up to **128K tokens** for Llama 3.1 (vs. 4K in Llama 2, 8K in Llama 3 base)
    - Intro-doc masking so one doc doesn't attend to another
  - *Training*: 405B model was trained on 15.6T tokens
- OLMo 2 (Dec 2024 – Mar 2025)
  - *Parameters*: 1B, 7B, 13B, 32B
  - *Licensing*: fully open: weights, data, training code, logs, recipes
  - *Model* compared to OlMO 1
    - RMSNorm instead of LayerNorm, Reordered norm, and QK-norm
    - RoPE but with much larger base value
  - *Training*: AdamW with Z-Loss on 4-6T tokens
- DeepSeek-R1 (Jan 2025)
  - *Parameters*: 1.5 B, 7 B, 8 B, 14 B, 32 B, 70 B
  - Licensing: fully open source, MIT
  - Training:
    - DeepSeek-R1-Zero: pure RL training
    - DeepSeek-R1: SFT + RL training
- Gemma 3 (Mar 2025)
  - *Parameters*: 1B, 4B, 12B, 27B
  - *Similar to Gemma 2*
  - *Training*: 2T - 14T tokens for different model sizes
    - Quantization aware training (QAT)
- Llama 4 (Apr 2025)
- Qwen 3 (May 2025)
- gpt-oss (Aug 2025)
  - *Parameters*: 20B, 120B
  - *Model:*
    - RoPE, GQA, SwiGLU
    - mixture of experts (MoE)
  - *Training*: 2.1 million H100 GPU hours to complete pre-training

In this section we'll look at four techniques:

1. key-value cache (KV cache)
2. rotary position embeddings (RoPE)
3. grouped query attention (GQA)
4. sliding window attention

Modern Tranformer Models

Q: It sure would be nice if we could keep this list of Modern LLMs automatically updated throughout the year as new models come out. Can an LLM do this for me?

I'm making a summary of some modern transformer LLMs. The goal is to provide a somewhat chronological picture of what the major model choices were in some of the big name models that are open-source enough that we have information about them.

Here's what I have so far:

* PaLM (Oct 2022\)
  * 540B parameters
  * closed source
  * Model:
    * SwiGLU instead of ReLU, GELU, or Swish
    * multi-query attention (MQA) instead of multi-headed attention
    * rotary position embeddings
    * shared input-output embeddings instead of separate parameter matrices
  * Training: Adafactor on 780 billion tokens
* Llama-1 (Feb 2023\)
  * collection of models of varying parameter sizes: 7B, 13B, 32B, 65B
  * semi-open source
  * Llama-13B outperforms GPT-3 on average
  * Model compared to GPT-3:

A: You would think so, but…

OLMo (February 2024) — the Allen Institute Open Lang

- **Parameters**: early releases unspecified; focus is on openness and reproducibility Ai2 Documentation arXiv
- **Licensing**: fully open; includes weights, data, training code Ai2 Documentation arXiv
- **Model differences**:
  - No explicit architectural innovations listed (e.g., no mention of RoPE, GQA) — emphasis is on transparency rather than new transformer variants
- **Training**: fully open pipeline for reproducibility Ai2 Documentation

wrong

wrong

OLMo 2 (December 2024 / Early 2025)

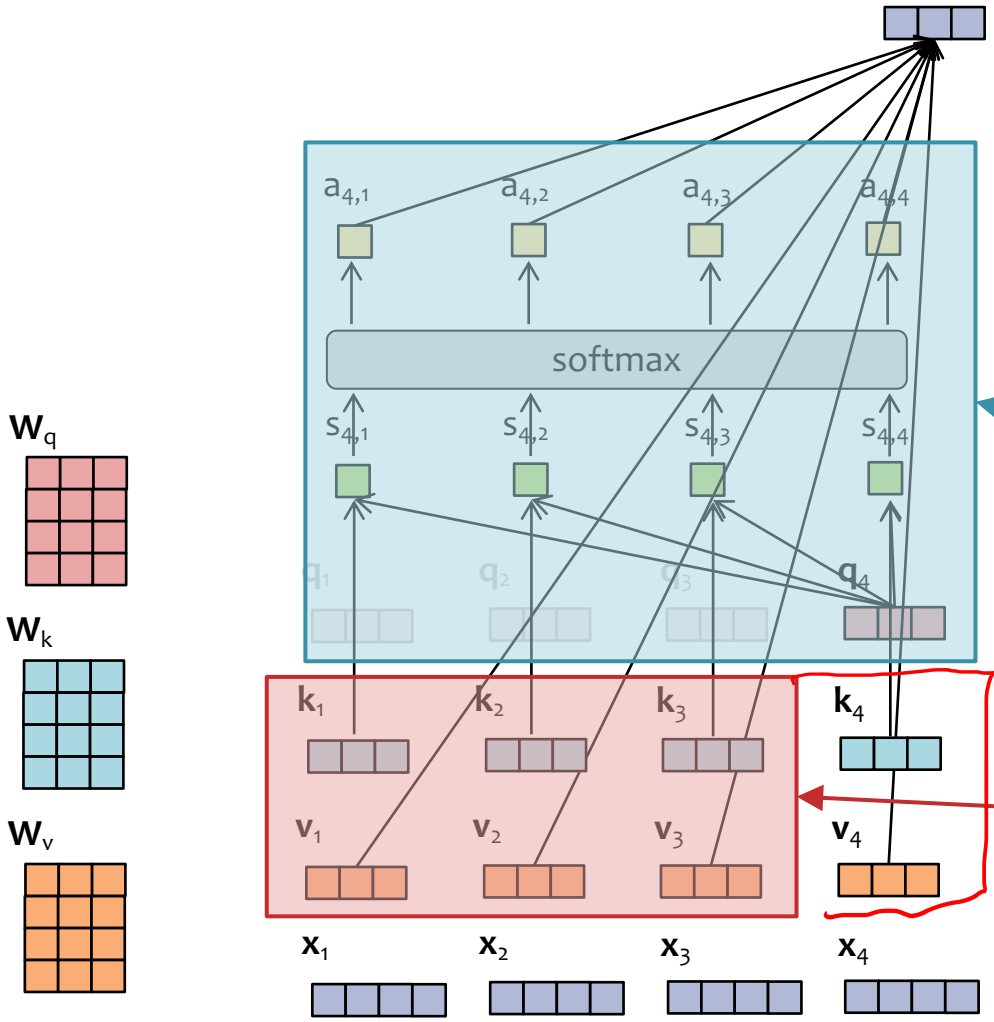
- **Parameters**: 7 B, 13 B, and 32 B (initial batch), plus a 1 B variant released May 2025 Hugging Face Ken Yeung
- **Licensing**: Fully open (Apache 2.0), includes all code, weights, logs, recipes Ai2 Documentation Hugging Face Ken Yeung
- **Model differences**:
  - Still no explicit notes of using **RoPE**, **GQA**, or sliding window attention—seems architecturally standard autoregressive transformer
- **Training**:
  - 32 B and 13 B versions: trained on ~4–5 trillion tokens, context length 4096 Hugging Face Ken Yeung
  - 1 B variant trained on 4 trillion tokens, with post-training series of variants (Instruct, SFT, DPO, RLHF, GGUF) Ken Yeung

wrong

wrong

Key-Value Cache

$$\mathbf{x}'_4 = \sum_{j=1}^{4} a_{4,j}\mathbf{v}_j$$



- 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)

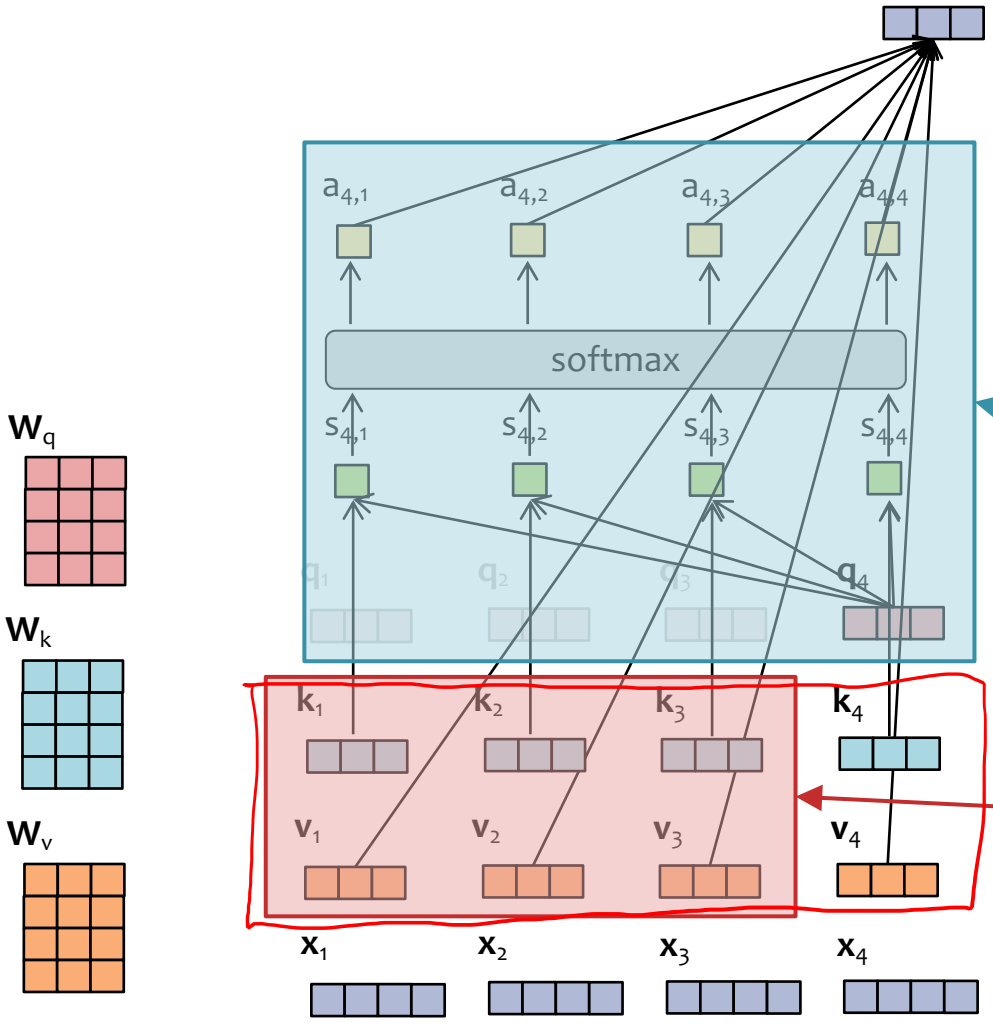
Discarded after this timestep

Computed for previous time-steps and reused for this timestep

Key-Value Cache

$$\mathbf{X}'_t = \mathbf{A}_t\mathbf{V} = \text{softmax}(\mathbf{Q}_t\mathbf{K}^T/\sqrt{d_k})\mathbf{V}$$

- 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)



$$\mathbf{A}_t = \text{softmax}(\mathbf{S}_t)$$

$$\mathbf{S}_t = \mathbf{Q}_t\mathbf{K}^T/\sqrt{d_k}$$

$$\mathbf{Q}_t = \mathbf{X}_t\mathbf{W}_q$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_v$$

$$\mathbf{X} = \left[\mathbf{x}_1, \ldots, \mathbf{x}_t\right]^T$$

Discarded after this timestep

Computed for previous time-steps and reused for this timestep

ROTARY POSITION EMBEDDINGS (ROPE)

Rotary Position Embeddings (RoPE)

Q: Why does this slide have so many typos?

A: I'm really not sure. I very meticulously type up the latex for my slides myself and think carefully about all the things I put in them.

RoPE attention:

$$f_q(\mathbf{x}_t, m) \triangleq \mathbf{R}_{\Theta,m} \mathbf{W}_q^T \mathbf{x}_t$$

$$f_k(\mathbf{x}_j, m) \triangleq \mathbf{R}_{\Theta,m} \mathbf{W}_k^T \mathbf{x}_j$$

$$s_{t,j} = f_k(\mathbf{x}_j, m)^T f_q(\mathbf{x}_t, m) / \sqrt{|\mathbf{k}|},$$

$$\forall j, t \text{ where } m = t - j$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

wrong

wrong

wrong

wrong

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$, and the rotary matrix $\mathbf{R}_{\Theta,m} \in \mathbb{R}^{d_k \times d_k}$ is given by:

$$R_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos m\theta_{d_k/2} & -\sin m\theta_{d_k/2} \\ 0 & 0 & 0 & 0 & \dots & \sin m\theta_{d_k/2} & \cos m\theta_{d_k/2} \end{pmatrix}$$

The θ_i parameters are fixed ahead of time and defined as below:

$$\Theta = \{\theta_i = 10000^{-2^{i-1}/d}, i \in [1, 2, \dots, d/2]\}$$

wrong

Rotary Position Embeddings (RoPE)

Q: Why does this slide have so many typos?

A: I'm really not sure. I very meticulously type up the latex for my slides myself and think carefully about all the things I put in them.

Rotary Position Embeddings (RoPE)

- Rotary position embeddings are a kind of **relative** position embeddings
- Key idea:
  - break each d-dimensional input vector into d/2 vectors of length 2
  - rotate each of the d/2 vectors by an amount scaled by m
  - m is the absolute position of the query or the key

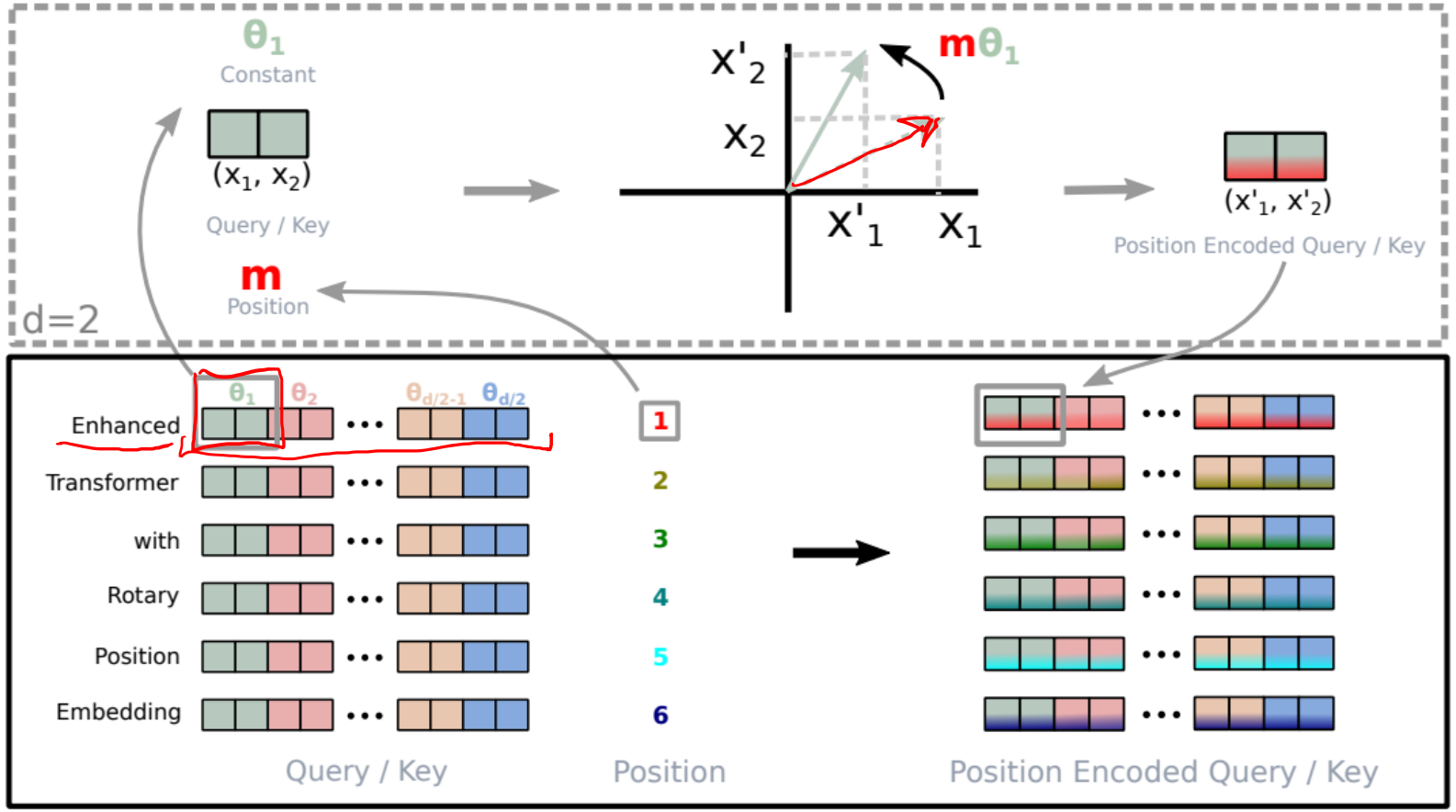


Figure from http://arxiv.org/abs/2104.09864

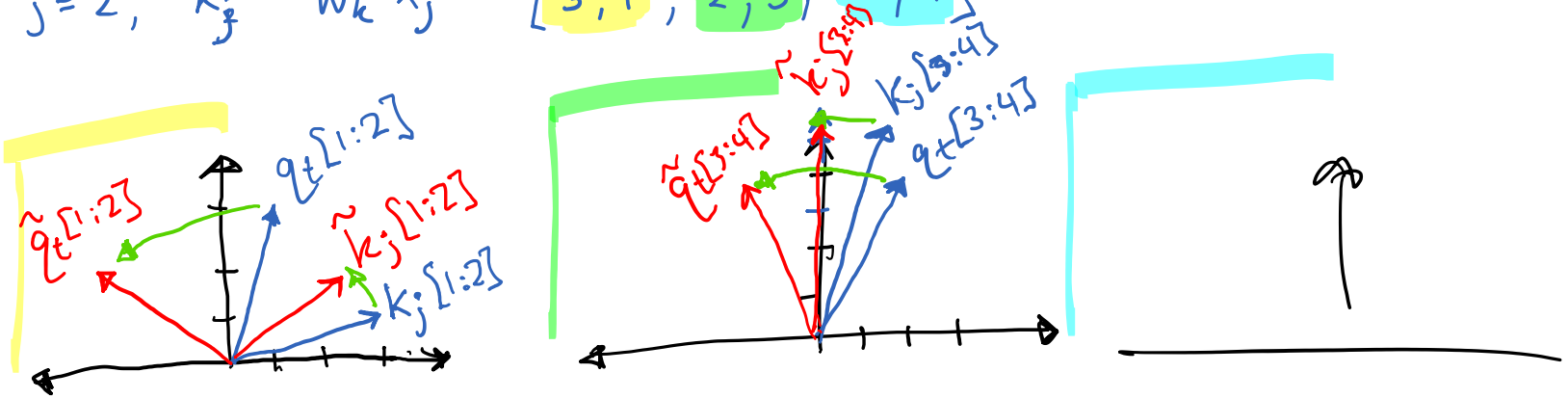
Rotary Position Embeddings (RoPE)

the bird flew out

1 2 3 4

$t=4,\quad q_t = W_q x_t = [1, 3, 2, 4, 4, 7]$

$j=2,\quad k_j = W_k x_j = [3, 1, 2, 5, -2, 1]$



$$\tilde{q}_t = f(q_t, t) = R_{\Theta,t}\, q_t$$

$$\tilde{k}_j = f(k_j, j) = R_{\Theta,j}\, k_j$$

Rotary Position Embeddings (RoPE)

Standard attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$
$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$
$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j \qquad \mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$
$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j \qquad \tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$
$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

where $\mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d_{model} \times d_k}$. Herein we use $d = d_k$ for brevity.

For some fixed absolute position m, the rotary matrix $\mathbf{R}_{\Theta,m} \in \mathbb{R}^{d_k \times d_k}$ is given by:

$$R_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \dots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & \cos m\theta_{d_k/2} & -\sin m\theta_{d_k/2} \\ 0 & 0 & 0 & 0 & \dots & \sin m\theta_{d_k/2} & \cos m\theta_{d_k/2} \end{pmatrix}$$

The θ_i parameters are fixed ahead of time and defined as below.

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$$

Rotary Position Embeddings (RoPE)

Standard attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j$$
$$\mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$
$$s_{t,j} = \mathbf{k}_j^T \mathbf{q}_t / \sqrt{|\mathbf{k}|}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j \qquad \mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$
$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j \qquad \tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$
$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$
$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

Because of the block sparse pattern in $\mathbf{R}_{\theta,m}$, we can efficiently compute the matrix-vector product of $\mathbf{R}_{\theta,m}$ with some arbitrary vector $\mathbf{y}$ in a more efficient manner:

$$\mathbf{R}_{\Theta,m}\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{d-1} \\ y_d \end{pmatrix} \odot \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{d/2} \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -y_2 \\ y_1 \\ -y_4 \\ y_3 \\ \vdots \\ -y_d \\ y_{d-1} \end{pmatrix} \odot \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{d/2} \\ \sin m\theta_{d/2} \end{pmatrix}$$

Matrix Version of RoPE

RoPE attention:

$$\mathbf{q}_j = \mathbf{W}_q^T \mathbf{x}_j, \forall j \qquad \mathbf{k}_j = \mathbf{W}_k^T \mathbf{x}_j, \forall j$$

$$\tilde{\mathbf{q}}_j = \mathbf{R}_{\Theta,j} \mathbf{q}_j \qquad \tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$

$$s_{t,j} = \tilde{\mathbf{k}}_j^T \tilde{\mathbf{q}}_t / \sqrt{d_k}, \forall j, t$$

$$\mathbf{a}_t = \text{softmax}(\mathbf{s}_t), \forall t$$

Matrix Version:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\tilde{\mathbf{Q}} = g(\mathbf{Q}; \Theta) \qquad \tilde{\mathbf{K}} = g(\mathbf{K}; \Theta)$$

$$\mathbf{S} = \tilde{\mathbf{Q}}\tilde{\mathbf{K}}^T / \sqrt{d_k}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$

Goal: to construct a new matrix $\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$ such that $\tilde{\mathbf{Y}}_{m,.} = \mathbf{R}_{\Theta,m} \mathbf{y}_m$

$$\mathbf{C} = \left[\begin{array}{ccc|ccc} 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} & 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ N\theta_1 & \cdots & N\theta_{\frac{d}{2}} & N\theta_1 & \cdots & N\theta_{\frac{d}{2}} \end{array}\right]$$

$$\begin{aligned} \tilde{\mathbf{Y}} &= g(\mathbf{Y}; \Theta) \\ &= \left[\begin{array}{c|c} \mathbf{Y}_{.,1:d/2} & \mathbf{Y}_{.,d/2+1:d} \end{array}\right] \odot \cos(\mathbf{C}) \\ &\quad + \left[\begin{array}{c|c} -\mathbf{Y}_{.,d/2+1:d} & \mathbf{Y}_{.,1:d/2} \end{array}\right] \odot \sin(\mathbf{C}) \end{aligned}$$

Matrix Version of RoPE

Q: Is this slide correct?

A: I'm really not sure.

But I did write it myself!

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

$$\tilde{\mathbf{k}}_j = \mathbf{R}_{\Theta,j} \mathbf{k}_j$$

Matrix Version:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_k$$

$$\tilde{\mathbf{Q}} = g(\mathbf{Q}; \Theta) \qquad \tilde{\mathbf{K}} = g(\mathbf{K}; \Theta)$$

$$\mathbf{S} = \tilde{\mathbf{Q}}\tilde{\mathbf{K}}^T / \sqrt{d_k}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S})$$

Goal: to construct a new matrix $\tilde{\mathbf{Y}} = g(\mathbf{Y}; \Theta)$ such that $\tilde{\mathbf{Y}}_{m,.} = \mathbf{R}_{\Theta,m}\mathbf{y}_m$

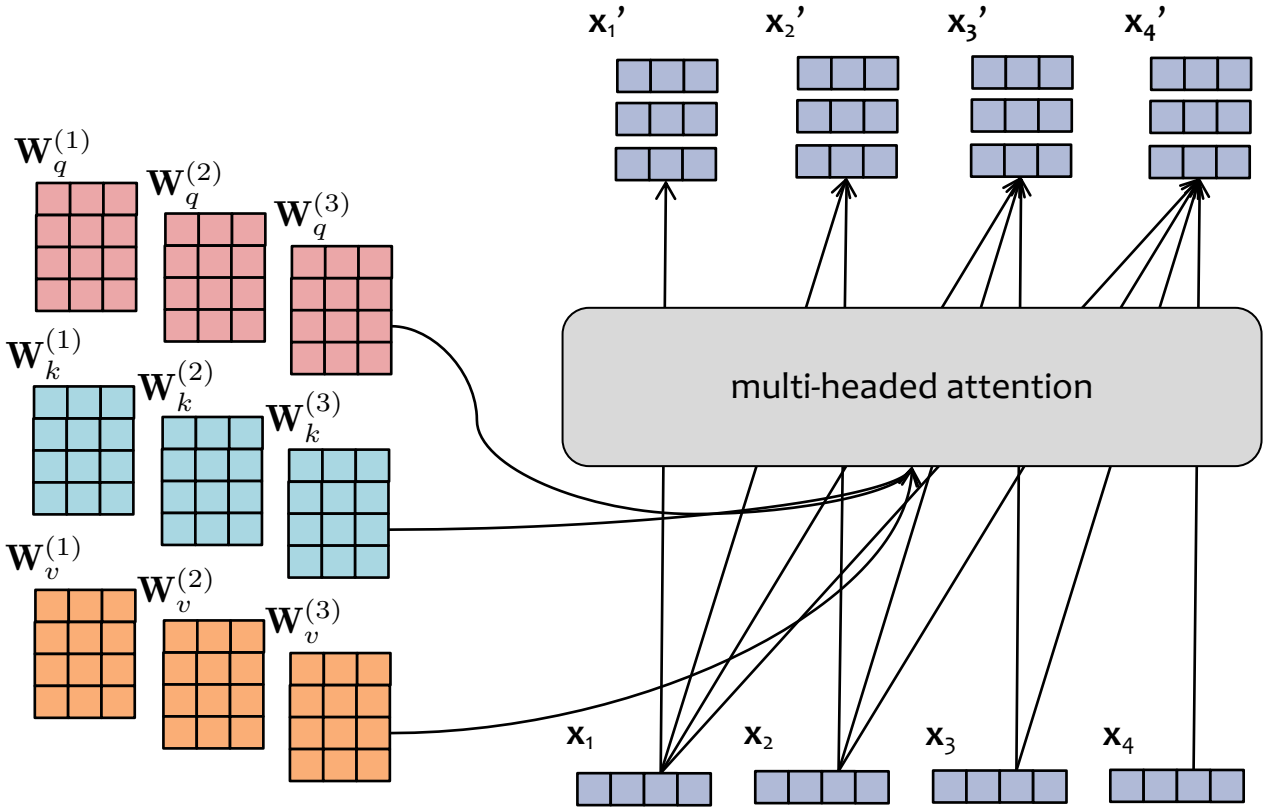
$$\mathbf{C} = \left[\begin{array}{ccc|ccc} 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} & 1\theta_1 & \cdots & 1\theta_{\frac{d}{2}} \\ \vdots & & \vdots & \vdots & & \vdots \\ N\theta_1 & \cdots & N\theta_{\frac{d}{2}} & N\theta_1 & \cdots & N\theta_{\frac{d}{2}} \end{array}\right]$$

$$\begin{aligned} \tilde{\mathbf{Y}} &= g(\mathbf{Y}; \Theta) \\ &= \left[\begin{array}{c|c} \mathbf{Y}_{.,1:d/2} & \mathbf{Y}_{.,d/2+1:d} \end{array}\right] \odot \cos(\mathbf{C}) \\ &\quad + \left[\begin{array}{c|c} -\mathbf{Y}_{.,d/2+1:d} & \mathbf{Y}_{.,1:d/2} \end{array}\right] \odot \sin(\mathbf{C}) \end{aligned}$$

GROUPED QUERY ATTENTION (GQA)

Matrix Version of Multi-Headed (Causal) Attention

Recall…



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

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

$$\mathbf{Q}^{(i)} = \mathbf{X}\mathbf{W}_q^{(i)}$$

$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}$$

$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}$$

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_4]^T$$

Grouped Query Attention (GQA)

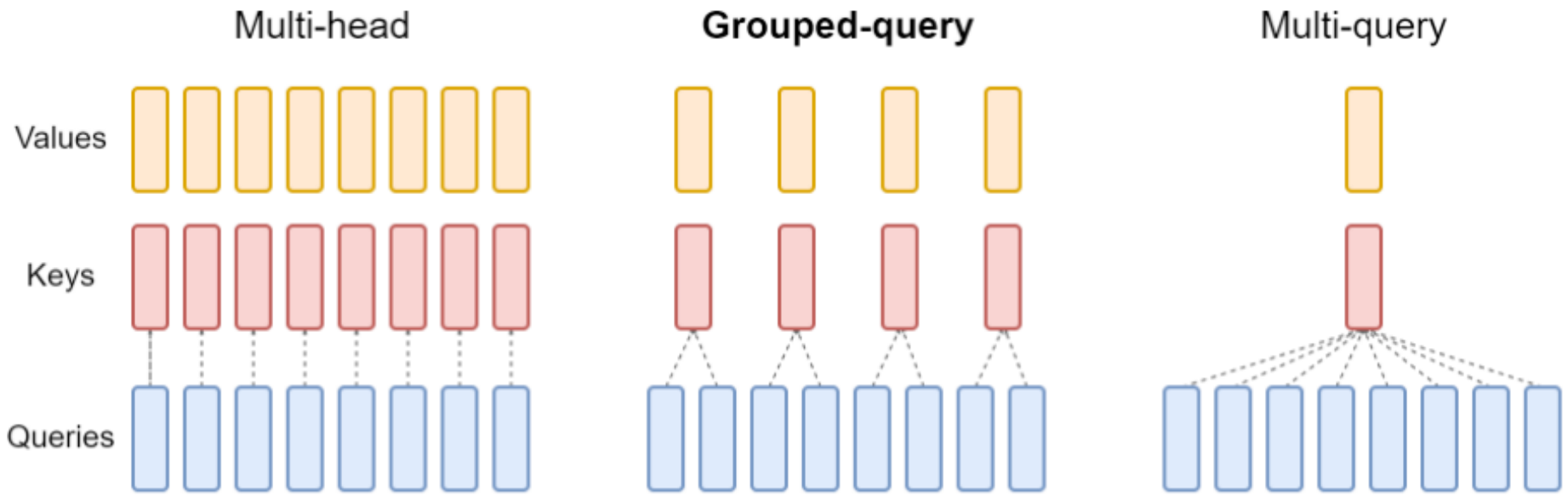
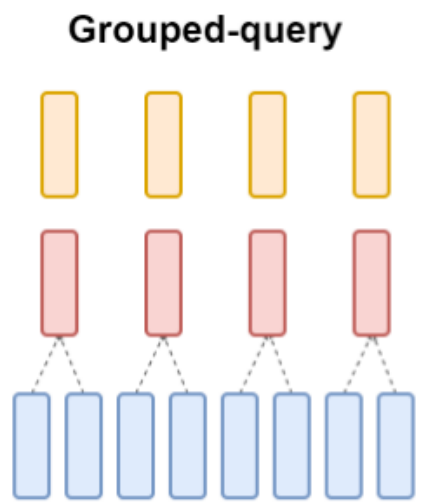


Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

Figure from http://arxiv.org/abs/2305.13245

Grouped Query Attention (GQA)



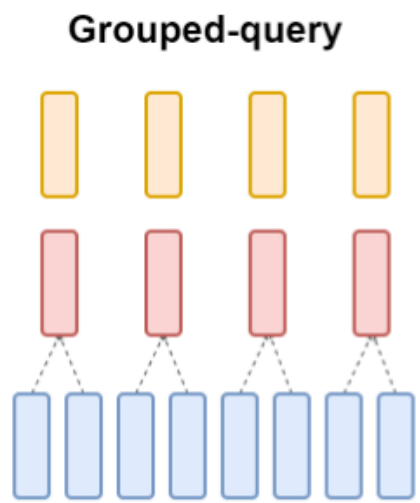
- **Key idea:** reuse the same key-value heads for multiple different query heads
- **Parameters:** The parameter matrices are all the same size, but we now have fewer key/value parameter matrices (heads) than query parameter matrices (heads)

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^T$$
$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$
$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$
$$\mathbf{Q}^{(i,j)} = \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

- h_q = the number of query heads
- h_{kv} = the number of key/value heads
- Assume h_q is divisible by h_{kv}
- $g = h_q / h_{kv}$ is the size of each group (i.e. the number of query vectors per key/value vector).

Figure from http://arxiv.org/abs/2305.13245

Grouped Query Attention (GQA)



- **Key idea:** reuse the same key-value heads for multiple different query heads
- **Parameters:** The parameter matrices are all the same size, but we now have fewer key/value parameter matrices (heads) than query parameter matrices (heads)

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]^T$$
$$\mathbf{V}^{(i)} = \mathbf{X}\mathbf{W}_v^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$
$$\mathbf{K}^{(i)} = \mathbf{X}\mathbf{W}_k^{(i)}, \forall i \in \{1, \ldots, h_{kv}\}$$
$$\mathbf{Q}^{(i,j)} = \mathbf{X}\mathbf{W}_q^{(i,j)}, \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$

$$\mathbf{S}^{(i,j)} = \mathbf{Q}^{(i,j)}(\mathbf{K}^{(i)})^T/\sqrt{d_k}, \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$
$$\mathbf{A}^{(i,j)} = \text{softmax}(\mathbf{S}^{(i,j)}), \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$
$$\mathbf{X}'^{(i,j)} = \mathbf{A}^{(i,j)}\mathbf{V}^{(i)}, \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$
$$\mathbf{X}' = \text{concat}(\mathbf{X}'^{(i,j)}), \quad \forall i \in \{1, \ldots, h_{kv}\}, \forall j \in \{1, \ldots, g\}$$
$$\mathbf{X} = \mathbf{X}'\mathbf{W}_o \qquad (\text{where } \mathbf{W}_o \in \mathbb{R}^{d_{model} \times d_{model}})$$

Figure from http://arxiv.org/abs/2305.13245

SLIDING WINDOW ATTENTION

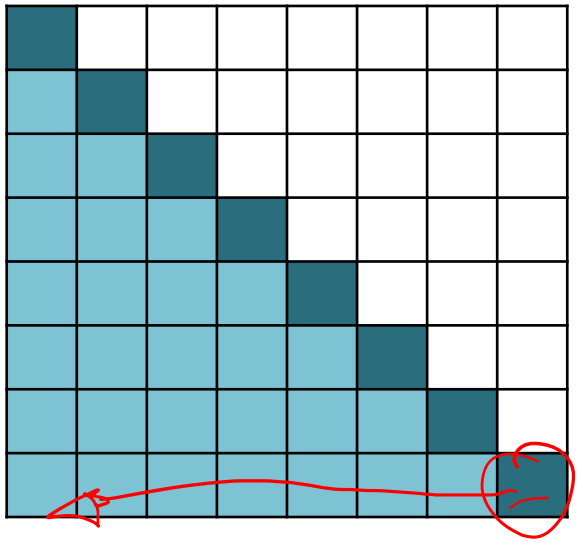
Sliding Window Attention

Sliding Window Attention

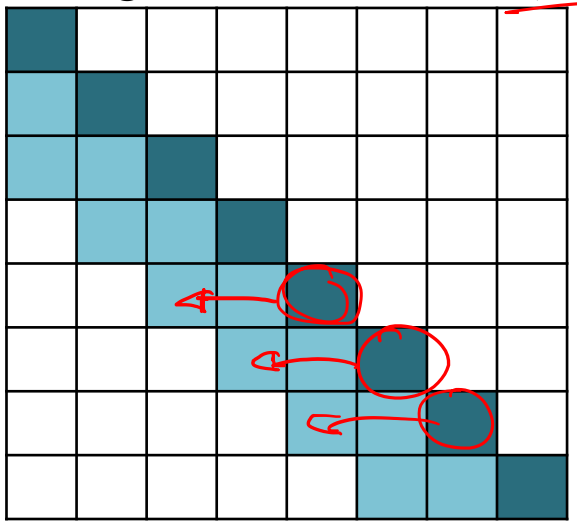
- also called "local attention" and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

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

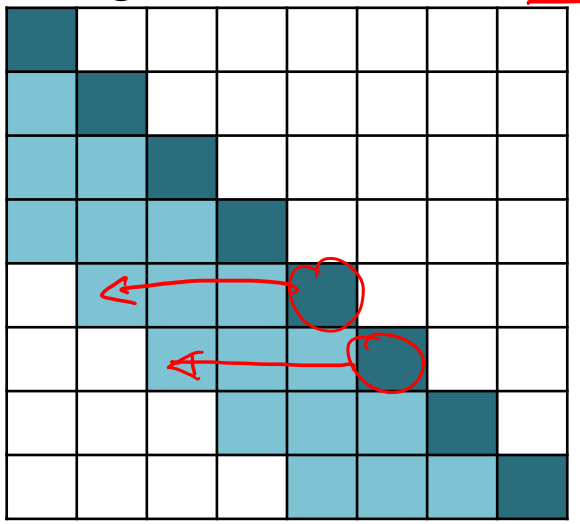
regular causal attention



sliding window attention (w=4)



sliding window attention (w=6)



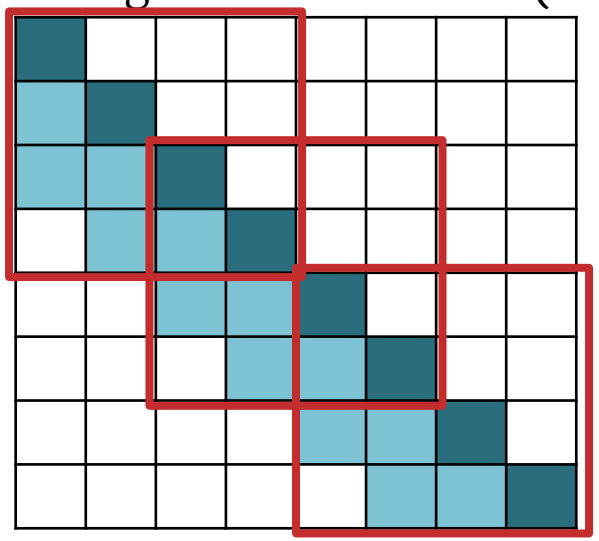
Sliding Window Attention

Sliding Window Attention

- also called "local attention" and introduced for the Longformer model (2020)
- **The problem:** regular attention is computationally expensive and requires a lot of memory
- **The solution:** apply a causal mask that only looks at the include a window of (½w+1) tokens, with the rightmost window element being the current token (i.e. on the diagonal)

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

sliding window attention (w=4)



3 ways you could implement

1. *naïve implementation:* just do the matrix multiplication, but this is still slow
2. *for-loop implementation:* asymptotically faster / less memory, but unusable in practice b/c for-loops in PyTorch are too slow
3. *sliding chunks implementation:* break into Q and K into chunks of size w x w, with overlap of ½w; then compute full attention within each chunk and mask out chunk (very fast/low memory in practice)