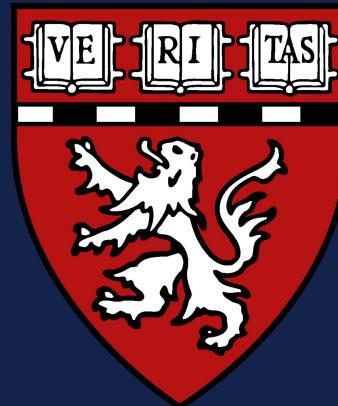


# BMI 702: Biomedical Informatics (Large) Language Models

Jonathan Richard Schwarz  jonathan\_schwarz@hms.harvard.edu

Slides



03/21/2024

# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

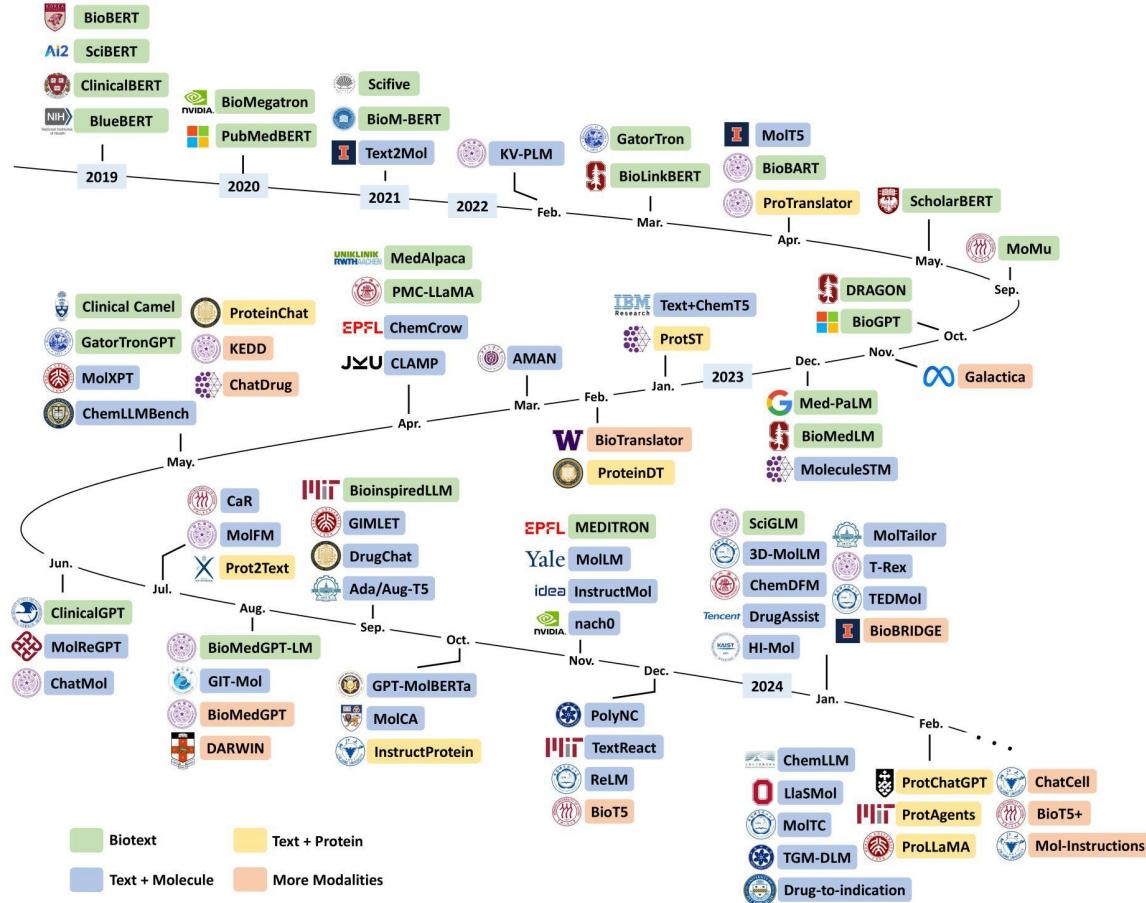
## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations

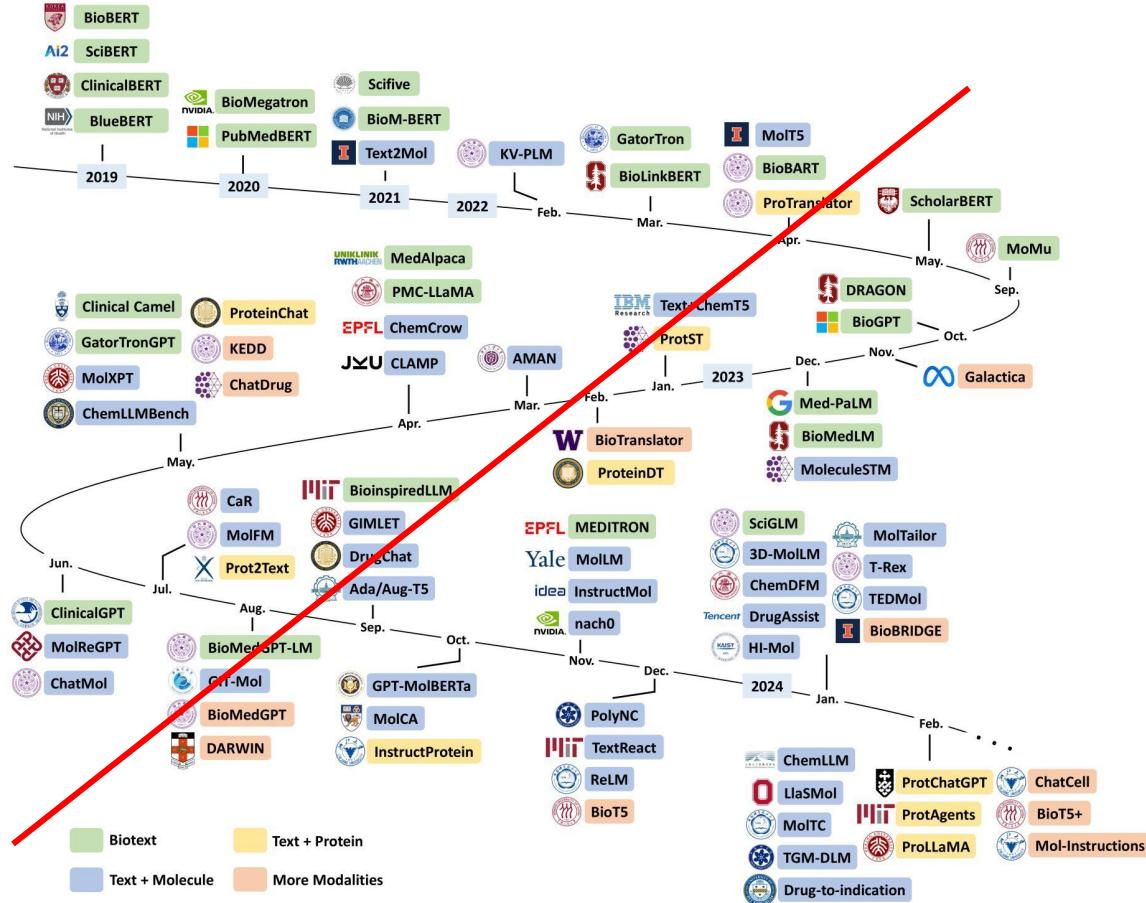
+ Glossary of new ideas (RLHF, RAG, Instruction Tuning), time permitting



# This class



# This class



# Outline for today

## Part 1:

- **N-Gram Language Models**
- Transformers

[break]

## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations



# A famous quote

*It must be recognized that the notion “probability of a sentence” is an entirely useless one, under any known interpretation of this term.*

- Noam Chomsky, 1969



# Intuitive interpretation

“Probability of a sentence” = how likely is it to occur in natural language

Example 1: Grammatical knowledge

$$p(\textit{the cat purrs}) > p(\textit{cat purrs the})$$

Example 2: World knowledge

$$p(\textit{the cat purrs}) > p(\textit{the cat smokes})$$



# Intuitive interpretation

"Probability of a sentence" = how likely is it to occur in natural language

Example 1: Grammatical knowledge

$$p(\text{the cat purrs}) > p(\text{cat purrs the})$$

Example 2: World knowledge

$$p(\text{the cat purrs}) > p(\text{the cat smokes})$$

What about the probability of "the Archaeopteryx winged jaggedly amidst foliage"?



# Probabilistic Models of Language

- A vocabulary  $\Sigma$  is a (finite, non-empty) set of symbols (result of tokenization).
- Kleene closure  $\Sigma^*$  of a vocabulary: Set of all possible (finite-length) sequences including the empty sequence.
- Language  $L \subseteq \Sigma^*$ : Subset of the Kleene closure.



# Probabilistic Models of Language

- A vocabulary  $\Sigma$  is a (finite, non-empty) set of symbols (result of tokenization).
- Kleene closure  $\Sigma^*$  of a vocabulary: Set of all possible (finite-length) sequences including the empty sequence.
- Language  $L \subseteq \Sigma^*$ : Subset of the Kleene closure.

Probability model:

1.  $p(L) = 1$
2.  $p(\bigcup_{i=1}^n \mathcal{E}_i) = \sum_i^n p(\mathcal{E}_i)$  if  $\mathcal{E}_1, \mathcal{E}_2, \dots$  is a countable sequence of disjoint sets of  $\mathcal{P}(L)$ , the power set (=set of all subsets) of  $L$ .



# Probabilistic Models of Language

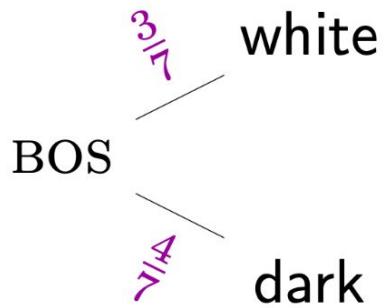
- A vocabulary  $\Sigma$  is a (finite, non-empty) set of symbols (result of tokenization).
- Kleene closure  $\Sigma^*$  of a vocabulary: Set of all possible (finite-length) sequences including the empty sequence.
- Language  $L \subseteq \Sigma^*$ : Subset of the Kleene closure.

Probability model:

1.  $p(L) = 1$
2.  $p(\bigcup_{i=1}^n \mathcal{E}_i) = \sum_i^n p(\mathcal{E}_i)$  if  $\mathcal{E}_1, \mathcal{E}_2, \dots$  is a countable sequence of disjoint sets of  $\mathcal{P}(L)$ , the power set (=set of all subsets) of  $L$ .
3. (Conditional probability)  $p(\mathbf{x}) = p(\text{EOS} \mid \mathbf{x}) \prod_{t=1}^n p(x_t \mid \mathbf{x}_{<t}) \cdot \text{EOS} \notin \Sigma^*$



## Example



- lotus  $p(\text{lotus} \mid \text{white})p(\text{white}) = \frac{3}{7} \cdot \frac{6}{9}$
- knight  $p(\text{knight} \mid \text{white})p(\text{white}) = \frac{3}{7} \cdot \frac{3}{9}$
- knight  $p(\text{knight} \mid \text{dark})p(\text{dark}) = \frac{4}{7} \cdot \frac{5}{9}$
- materials  $p(\text{materials} \mid \text{dark})p(\text{dark}) = \frac{4}{7} \cdot \frac{4}{9}$



# Estimation

We assume there is some true  $p^*$  which we estimate/approximate with a (parametric) estimator  $\hat{p}$  which is an element of  $\{p_\theta \mid \theta \in \Theta\}$ .



# Estimation

We assume there is some true  $p^*$  which we estimate/approximate with a (parametric) estimator  $\hat{p}$  which is an element of  $\{p_\theta \mid \theta \in \Theta\}$ .

This is done by learning from data  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq L$ , e.g. by minimizing some loss:

$$\hat{\theta} \triangleq \arg \min_{\theta \in \Theta} \ell(\theta, \theta^*)$$



# Estimation

We assume there is some true  $p^*$  which we estimate/approximate with a (parametric) estimator  $\hat{p}$  which is an element of  $\{p_\theta \mid \theta \in \Theta\}$ .

This is done by learning from data  $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq L$ , e.g. by minimizing some loss:

$$\hat{\theta} \triangleq \arg \min_{\theta \in \Theta} \ell(\theta, \theta^*)$$

Since the optimal model is unknown, we use the data as an estimate:

$$p_{\theta^*} \approx \frac{1}{|\mathcal{D}|} \sum_{i=1}^{\mathcal{D}} \delta_{\mathbf{x}_i}(\mathbf{x}) \quad \delta_{\mathbf{x}_i}(\mathbf{x}) \triangleq \begin{cases} 1 & \text{if } \mathbf{x}_i = \mathbf{x} \\ 0 & \text{else} \end{cases}$$

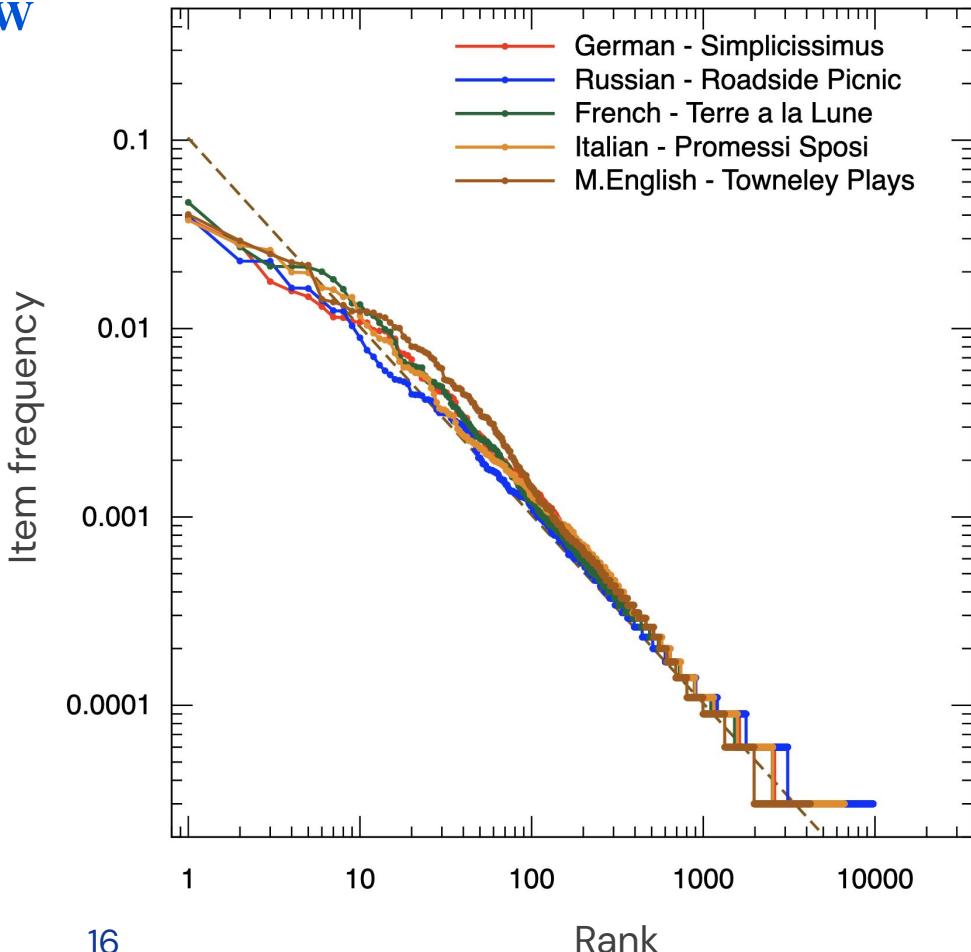


# A note about data - Zipf's Law

Word frequency approximately inversely proportional to its rank:

$$\text{frequency} \propto \frac{1}{(\text{rank} + b)^a}$$

with **a, b** fitted. (Zipf-Mandelbrot law)



# A note about data - Zipf's Law

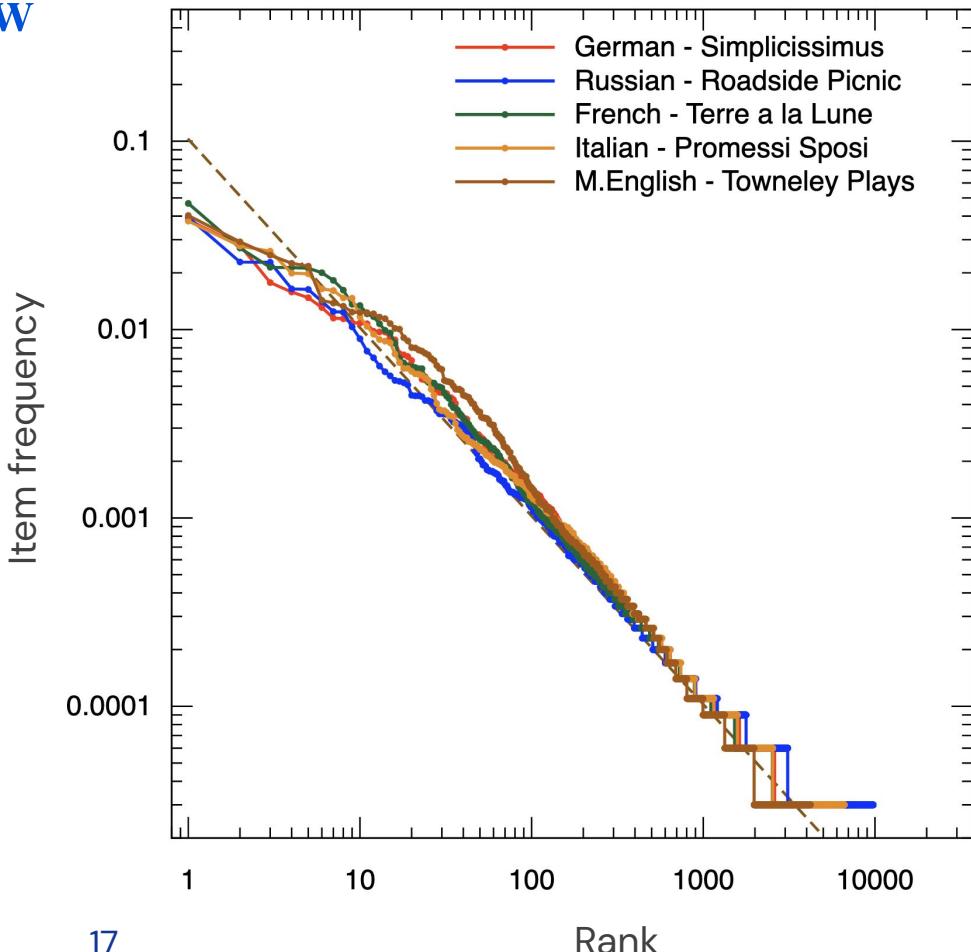
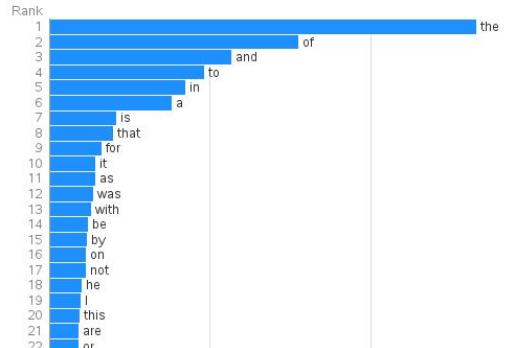
Word frequency approximately inversely proportional to its rank:

$$\text{frequency} \propto \frac{1}{(\text{rank} + b)^a}$$

with **a, b** fitted. (Zipf–Mandelbrot law)

50 Most Frequent Words in English Writing

Based on Google books data



# Cross-Entropy

A suitable loss function is the KL-Divergence (divergence between prob. distributions):

$$\text{KL}(p_{\theta^*}, p_{\hat{\theta}}) \triangleq - \sum_{\mathbf{x} \in L} p_{\theta^*}(\mathbf{x}) \log p_{\hat{\theta}}(\mathbf{x}) + \underbrace{p_{\theta^*}(\mathbf{x}) \log p_{\theta^*}(\mathbf{x})}_{-H(p_{\theta^*})}$$



# Cross-Entropy

A suitable loss function is the KL-Divergence (divergence between prob. distributions):

$$\text{KL}(p_{\theta^*}, p_{\hat{\theta}}) \triangleq \underbrace{- \sum_{\mathbf{x} \in L} p_{\theta^*}(\mathbf{x}) \log p_{\hat{\theta}}(\mathbf{x})}_{\text{Cross-Entropy}} + \underbrace{p_{\theta^*}(\mathbf{x}) \log p_{\theta^*}(\mathbf{x})}_{-H(p_{\theta^*})}$$

constant wrt. model param.

## Justification:

From Information Theory: Measures the excess number of bits we pay by encoding our data with a sub-optimal model. The optimum is just the entropy (Shannon, 1948).



# N-Gram models

We can obtain a very simple form for  $\{p_\theta \mid \theta \in \Theta\}$  by making the Markov assumption:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, \dots, x_n) \\ &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_{n-1}|x_1, x_2, \dots, x_{n-2}) \dots p(x_1) \\ &\approx p(x_n|x_{n-2}, x_{n-1})p(x_{n-1}|x_{n-3}, x_{n-2}) \dots p(x_1) \end{aligned}$$



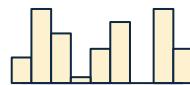
# N-Gram models

We can obtain a very simple form for  $\{p_\theta \mid \theta \in \Theta\}$  by making the Markov assumption:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, \dots, x_n) \\ &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_{n-1}|x_1, x_2, \dots, x_{n-2}) \dots p(x_1) \\ &\approx p(x_n|x_{n-2}, x_{n-1})p(x_{n-1}|x_{n-3}, x_{n-2}) \dots p(x_1) \end{aligned}$$

This is a tri-gram model (history of two). Straightforwardly estimated using the Maximum-Likelihood Estimate of a categorical distribution:

$$p_{\text{ML}}(x_3|x_1, x_2) = \frac{C(x_1, x_2, x_3)}{C(x_1, x_2)}$$



What's the problem with this model?



## N-Gram models

We can obtain a very simple form for  $\{p_\theta \mid \theta \in \Theta\}$  by making the Markov assumption:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, \dots, x_n) \\ &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_{n-1}|x_1, x_2, \dots, x_{n-2}) \dots p(x_1) \\ &\approx p(x_n|x_{n-2}, x_{n-1})p(x_{n-1}|x_{n-3}, x_{n-2}) \dots p(x_1) \end{aligned}$$

This is a tri-gram model (history of two). Assumes all of these are equal:

- $p(\text{slept}|\text{the cat})$
- $p(\text{slept}|\text{after lunch the cat})$
- $p(\text{slept}|\text{the dog chased the cat})$
- $p(\text{slept}|\text{except for the cat})$



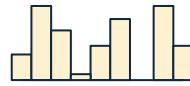
# N-Gram models

We can obtain a very simple form for  $\{p_\theta \mid \theta \in \Theta\}$  by making the Markov assumption:

$$\begin{aligned} p(\mathbf{x}) &= p(x_1, \dots, x_n) \\ &= p(x_n|x_1, x_2, \dots, x_{n-1})p(x_{n-1}|x_1, x_2, \dots, x_{n-2}) \dots p(x_1) \\ &\approx p(x_n|x_{n-2}, x_{n-1})p(x_{n-1}|x_{n-3}, x_{n-2}) \dots p(x_1) \end{aligned}$$

This is a tri-gram model (history of two). Straightforwardly estimated using the Maximum-Likelihood Estimate of a categorical distribution:

$$p_{\text{ML}}(x_3|x_1, x_2) = \frac{C(x_1, x_2, x_3)}{C(x_1, x_2)}$$



Zero-Probability events!



# Bayesian N-Gram models

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

Likelihood (Categorical)      Prior (Dirichlet)  
Evidence



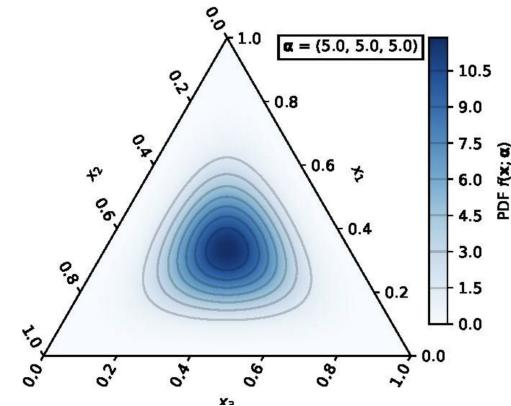
# Bayesian N-Gram models

Likelihood (Categorical)

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

Prior (Dirichlet)

Evidence



Turns out this is an example of a "conjugate prior". A choice of prior for which the posterior has the same shape as the prior.

$$p_1, \dots, p_K \sim \text{Dir}(\alpha_1, \dots, \alpha_K)$$

$$y \sim \text{Cat}(p_1, \dots, p_K)$$



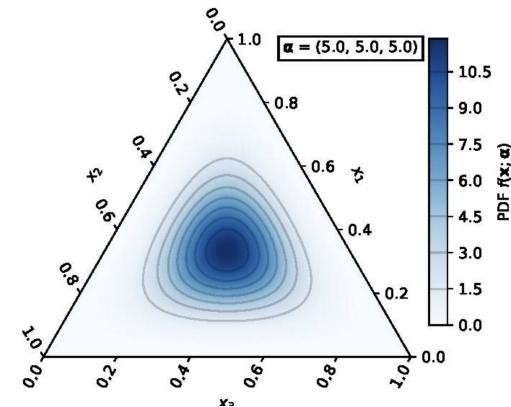
# Bayesian N-Gram models

Likelihood (Categorical)

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

Prior (Dirichlet)

Evidence



Turns out this is an example of a "conjugate prior". A choice of prior for which the posterior has the same shape as the prior.

$$p_1, \dots, p_K \sim \text{Dir}(\alpha_1, \dots, \alpha_K)$$

$$p(\theta|\mathcal{D}) = \text{Dir}(\alpha'_1, \dots, \alpha'_K)$$

$$y \sim \text{Cat}(p_1, \dots, p_K)$$

$$\alpha'_j = \cancel{\alpha_j} + \sum_{y_i \in D} \mathbb{1}\{y_i = j\}$$

You can think of those as "pseudo counts"



# Evaluation

Two popular evaluation metrics evaluated on a held-out/test set:

(1) Cross entropy (per word):

$$H(p_{\theta^*}, p_{\hat{\theta}}) = -\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \frac{1}{n} \log_2 p_{\hat{\theta}}(x_1, \dots, x_n)$$



# Evaluation

Two popular evaluation metrics evaluated on a held-out/test set:

(1) Cross entropy (per word):

$$H(p_{\theta^*}, p_{\hat{\theta}}) = -\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \frac{1}{n} \log_2 p_{\hat{\theta}}(x_1, \dots, x_n)$$

(2) Perplexity (captures a notion of surprise):

$$PPL(p_{\hat{\theta}}) = 2^{H(p_{\theta^*}, p_{\hat{\theta}})}$$



# Evaluation

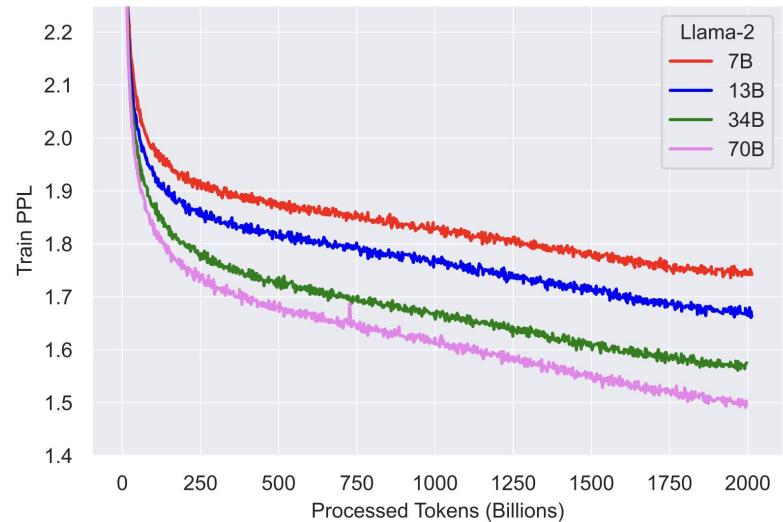
Two popular evaluation metrics evaluated on a held-out/test set:

(1) Cross entropy (per word):

$$H(p_{\theta^*}, p_{\hat{\theta}}) = -\frac{1}{|\mathcal{D}|} \sum_{\mathbf{x} \in \mathcal{D}} \frac{1}{n} \log_2 p_{\hat{\theta}}(x_1, \dots, x_n)$$

(2) Perplexity (captures a notion of surprise):

$$PPL(p_{\hat{\theta}}) = 2^{H(p_{\theta^*}, p_{\hat{\theta}})}$$



# Outline for today

## Part 1:

- N-Gram Language Models
- **Transformers**

[break]

## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations

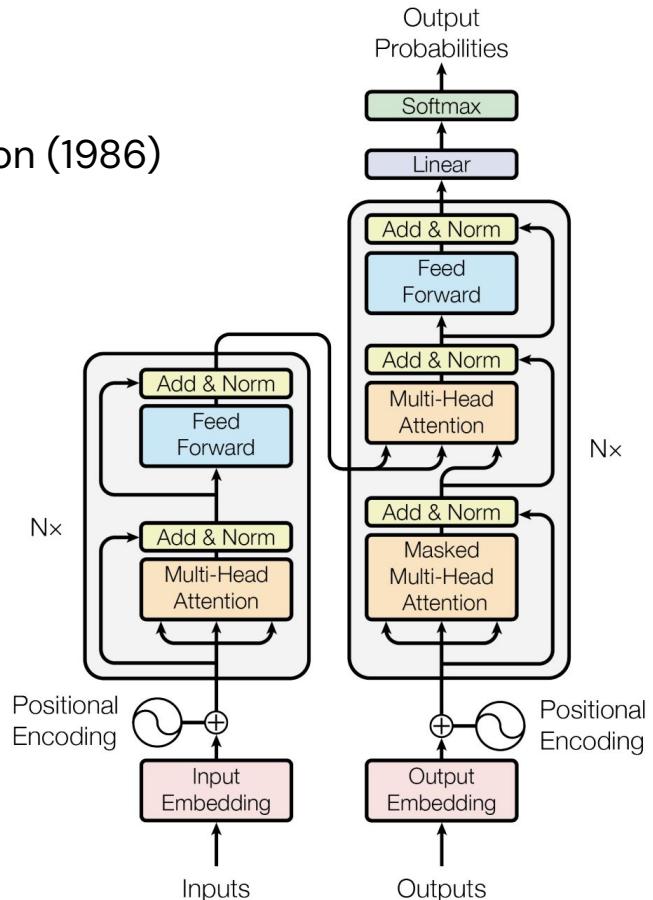


# (Encoder-Decoder) Transformers

Probably the most influential ML paper since Backpropagation (1986)

→ Over 112k citations since 2017

→ Essentially replaced RNNs for most purposes



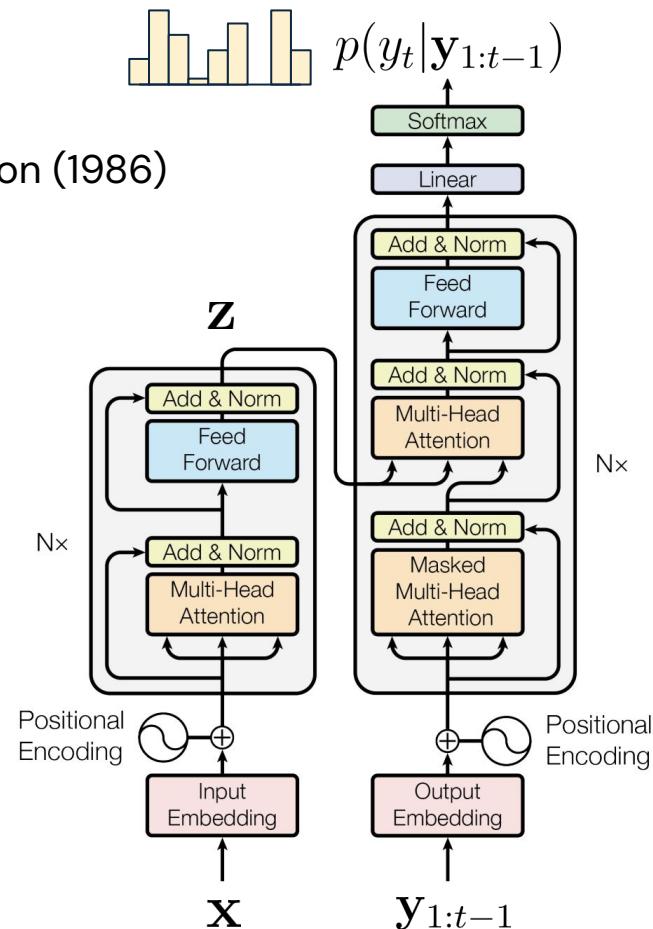
# (Encoder-Decoder) Transformers

Probably the most influential ML paper since Backpropagation (1986)

→ Over 112k citations since 2017

→ Essentially replaced RNNs for most purposes

A simple sequence to sequence model mapping an input  $(x_1, \dots, x_n)$  (tokenized and "embedded") into a continuous representation  $\mathbf{z} = (z_1, \dots, z_n)$  based on which the decoder produces  $(y_1, \dots, y_m)$  autoregressively, i.e. one symbol at a time.

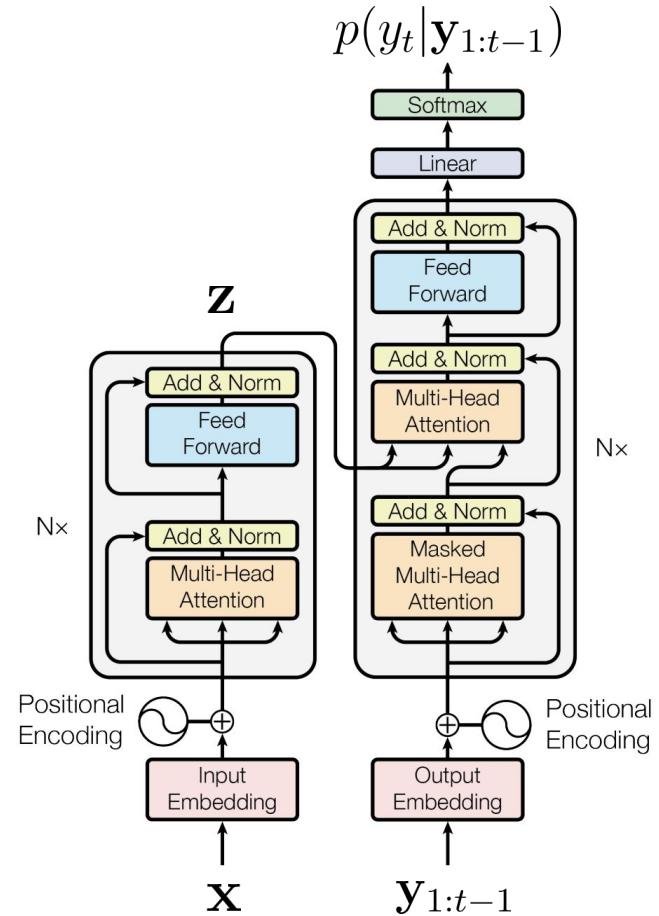


# The Transformer Building Blocks

1. Multi-head Attention

2. Position Encodings

3. Residual connections + Normalization

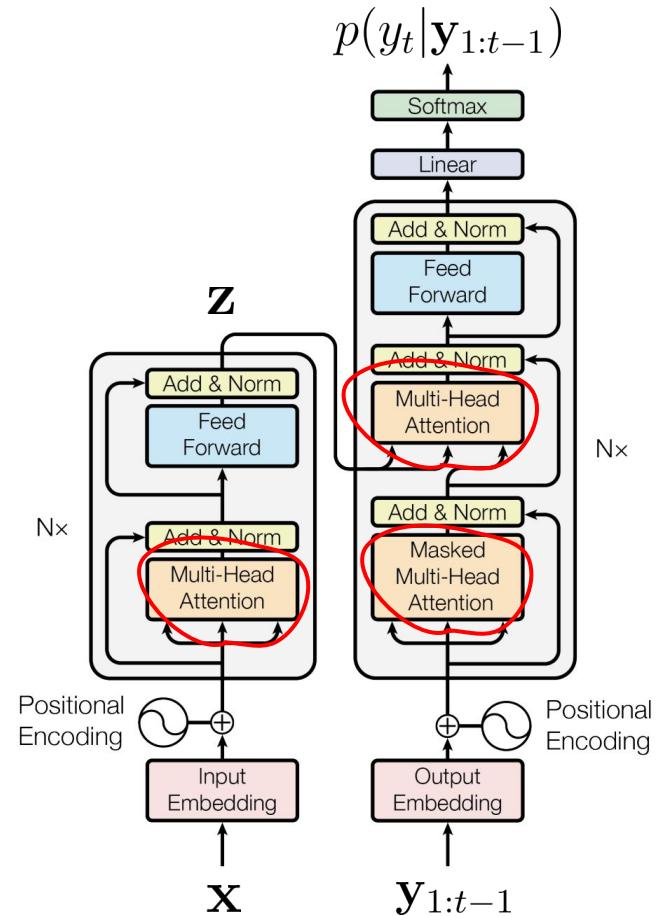


# The Transformer Building Blocks

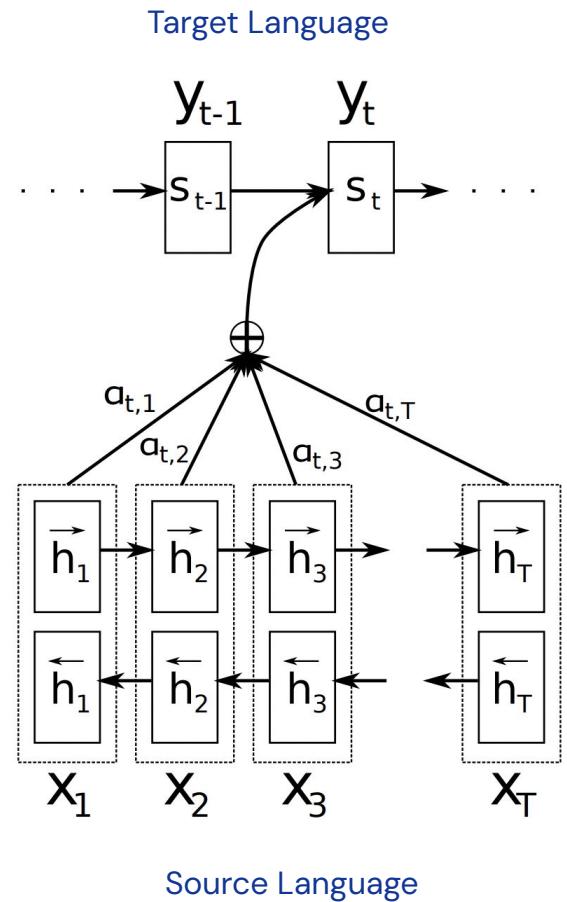
## 1. Multi-head Attention

## 2. Position Encodings

## 3. Residual connections + Normalization



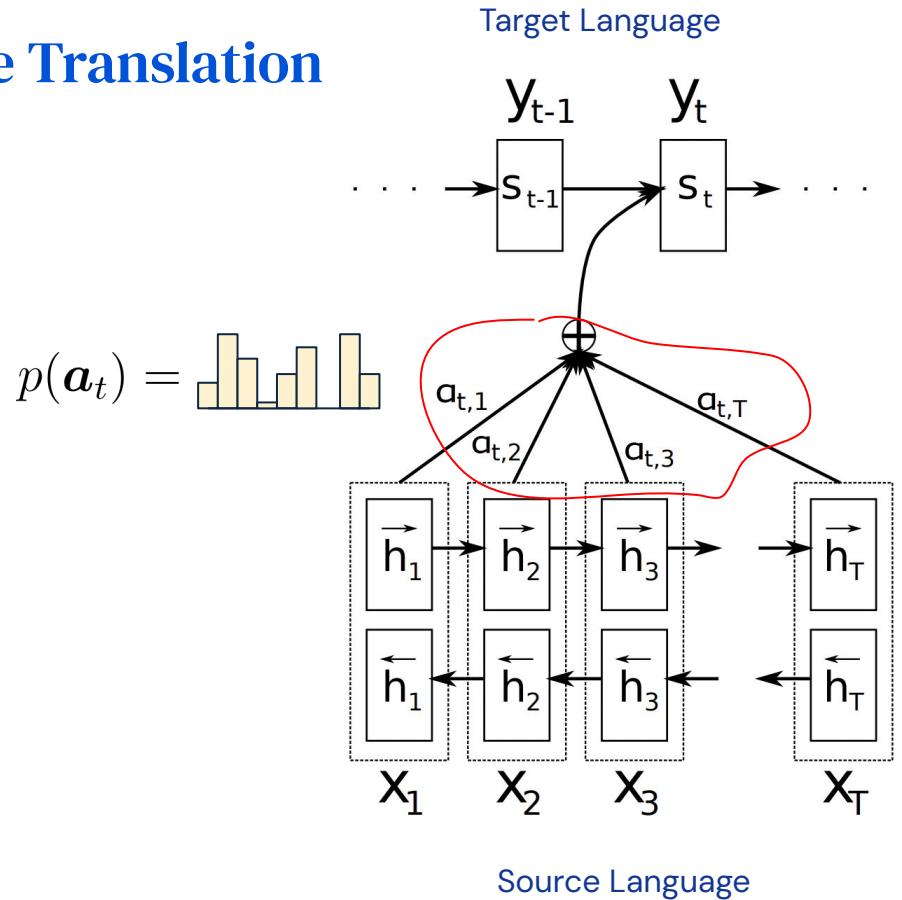
# Attention: An idea from Machine Translation



Source Language

Target Language

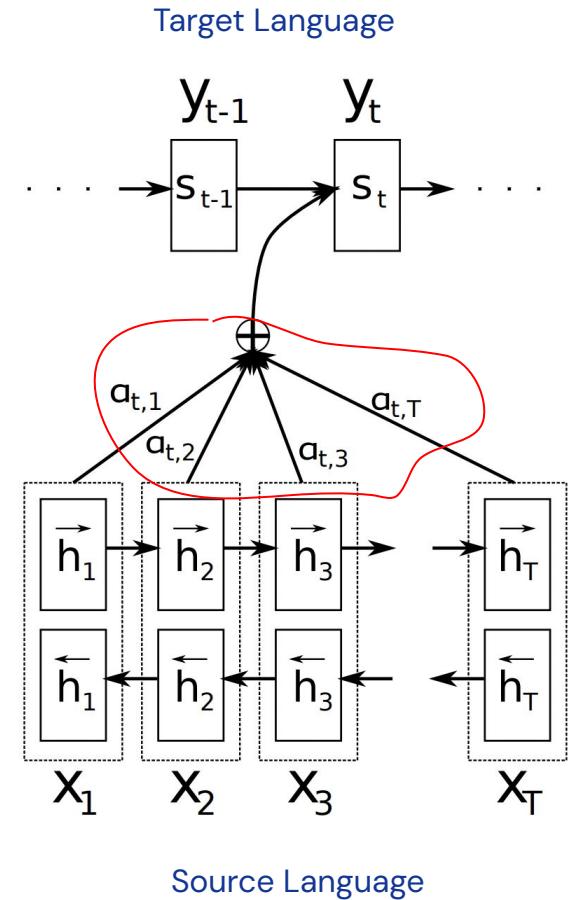
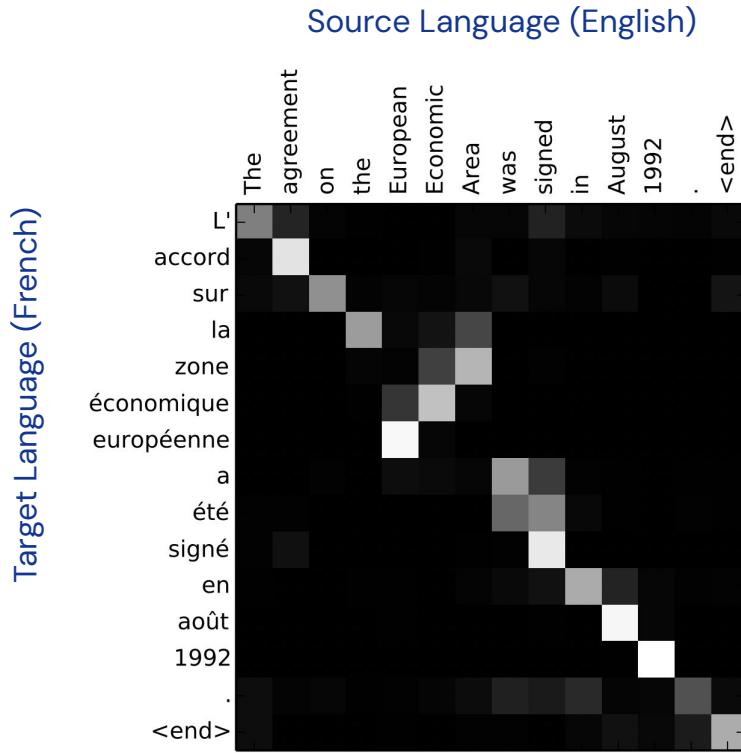
# Attention: An idea from Machine Translation



Source Language

Target Language

# Attention: An idea from Machine Translation



## Scaled Dot-Product Attention

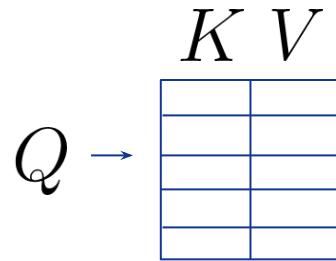
$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$

$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



## Scaled Dot-Product Attention

$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$



Think of this as a soft "look-up" operation in an associative memory using dot-products as a similarity measure.

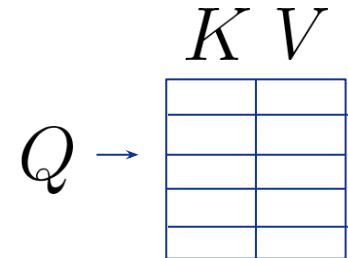
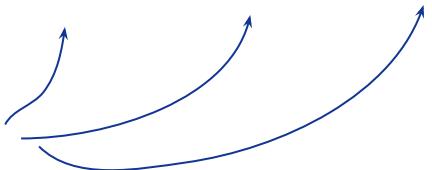
$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



## Scaled Dot-Product Attention

$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$

Where do they come from?



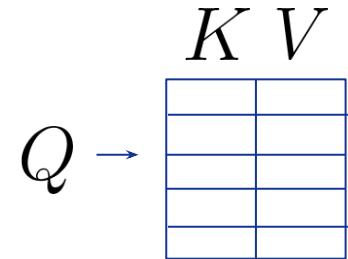
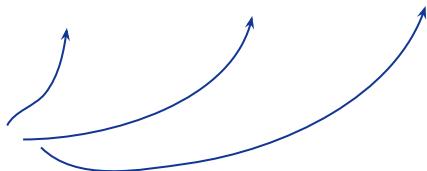
$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



# Scaled Dot-Product Attention

$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$

Where do they come from?



In Machine Translation, Keys and Values come from the source language, queries from the target language processed so far

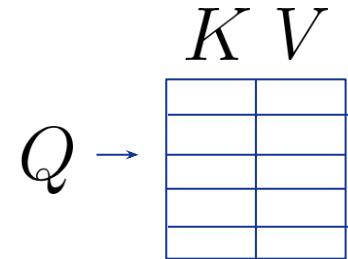
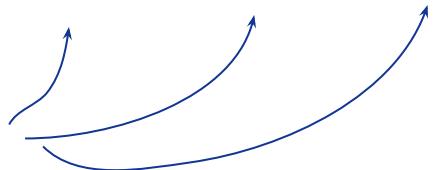
$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



# Scaled Dot-Product Attention

$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$

Where do they come from?



For now, let's think of them as equal, i.e. the input (or previous hidden layer) sequence:

$$Q = K = V = X$$

$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

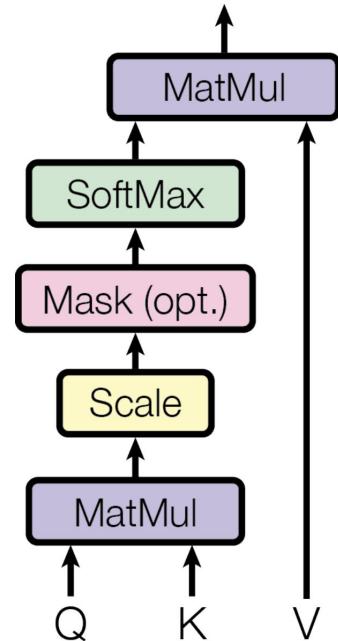


## Scaled Dot-Product Attention

$$H = \text{Attention}(QW^Q, KW^K, VW^V)$$

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

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{\dim(x)} e^{x_j}}$$



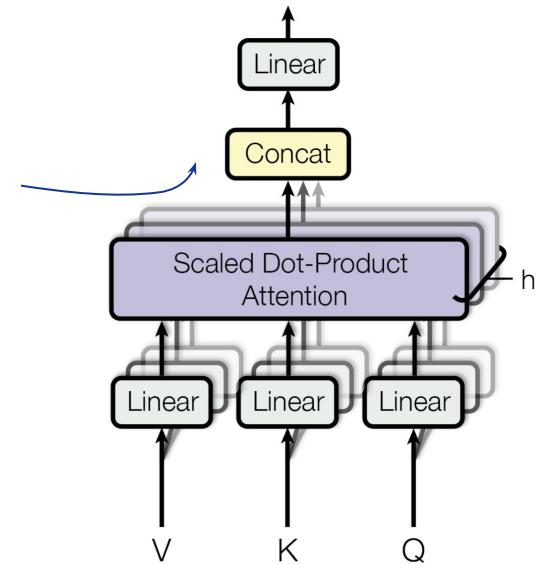
$$W^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



# Multi-Head Attention

$$H_i = \text{Attention}(QW_i^Q, K_iW^K, V_iW^V), i = 1, \dots, H$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, \dots, H_H)W^O$$



$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

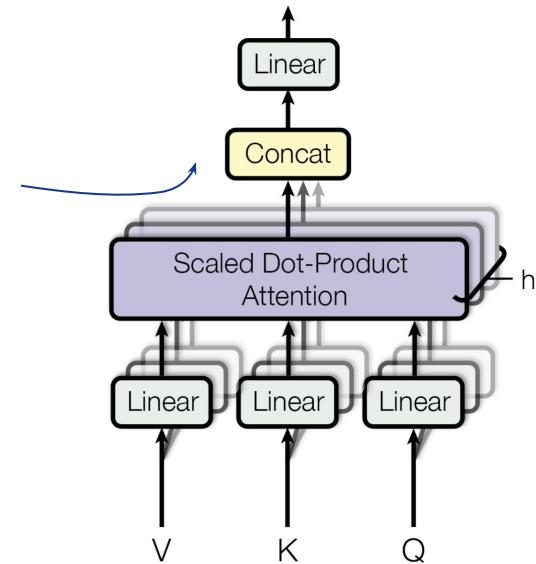


# Multi-Head Attention

$$H_i = \text{Attention}(QW_i^Q, K_iW^K, V_iW^V), i = 1, \dots, H$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, \dots, H_H)W^O$$

What's the shape of the output weights?



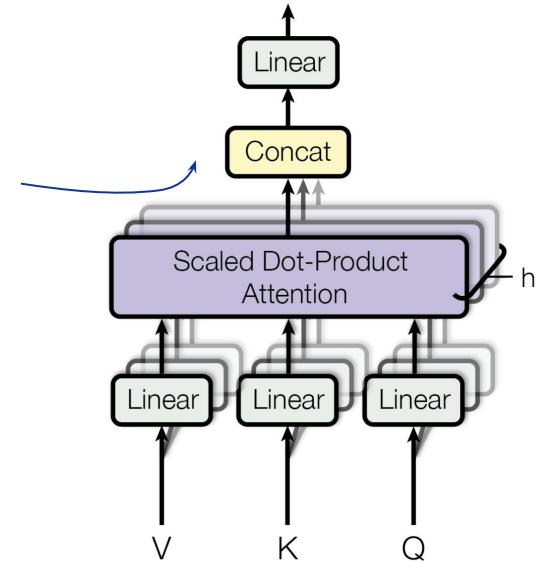
$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$



# Multi-Head Attention

$$H_i = \text{Attention}(QW_i^Q, K_iW^K, V_iW^V), i = 1, \dots, H$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H_1, \dots, H_H)W^O$$

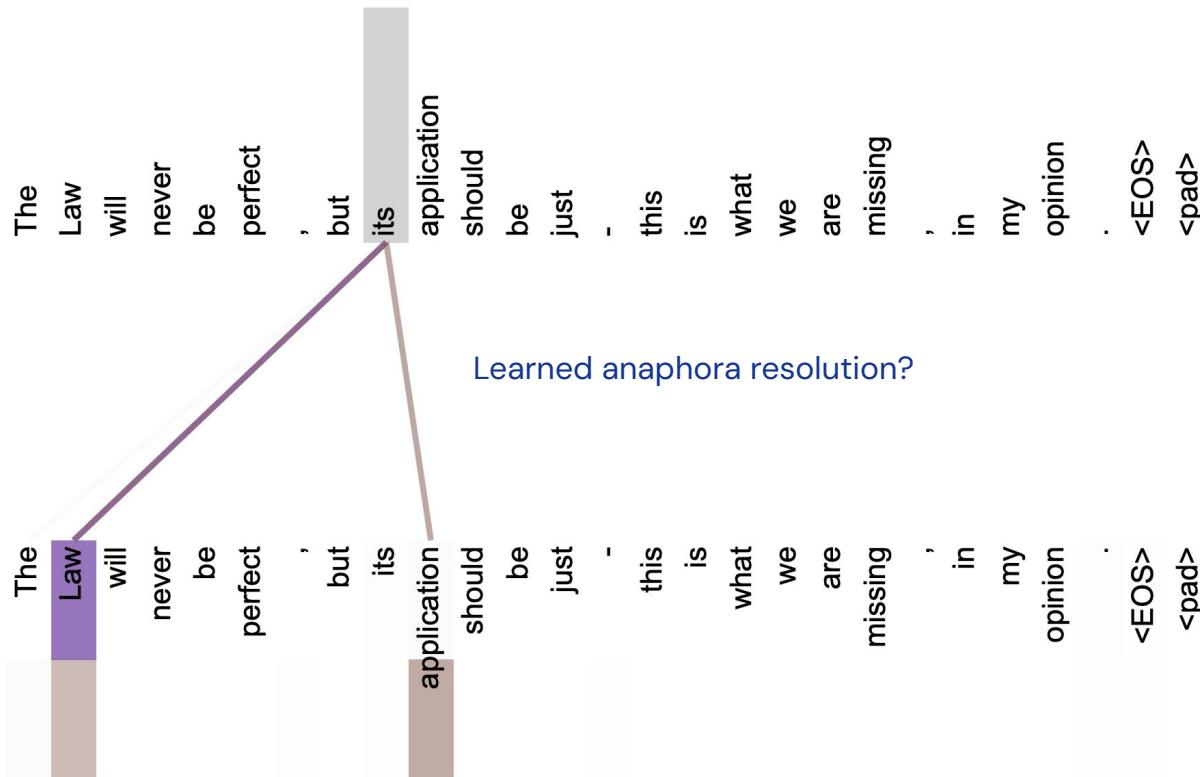


$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v} \text{ and } W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$



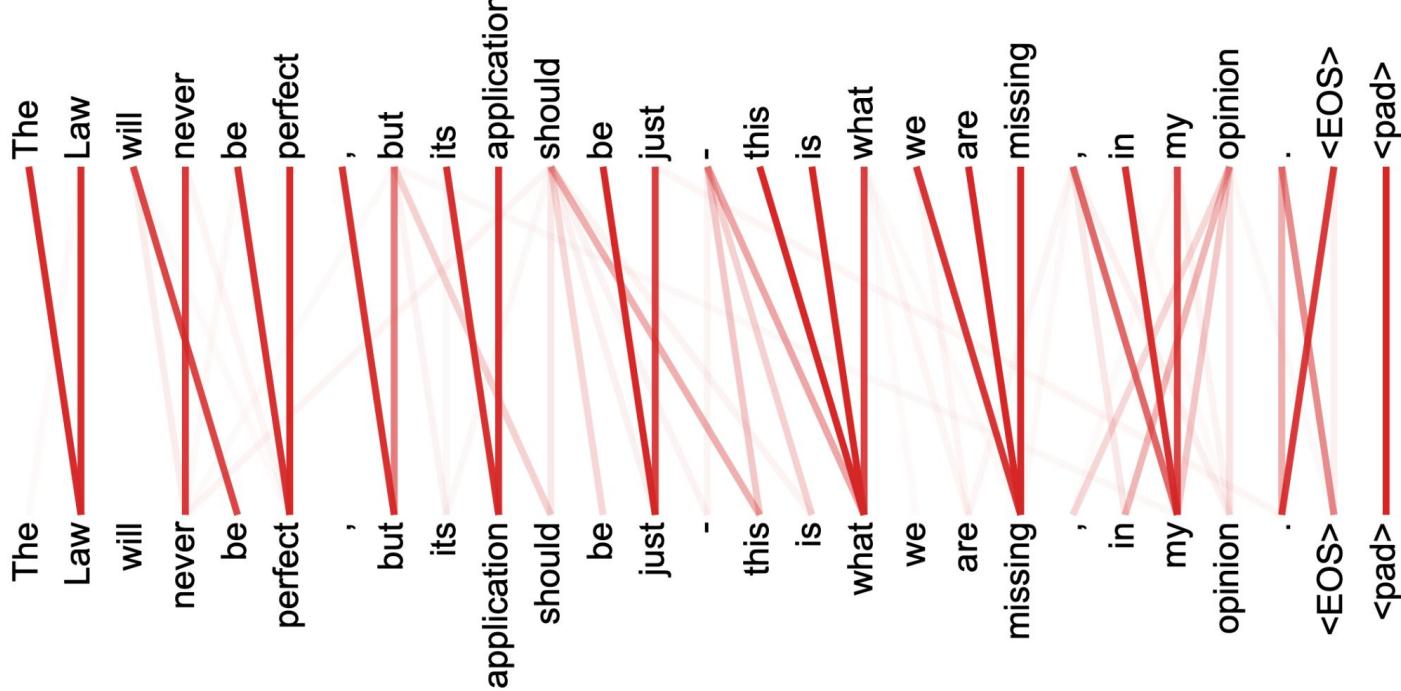
## Scaled Dot-Product Attention

# Scaled Dot-Product Attention



# Scaled Dot-Product Attention

Sequence and structure information?



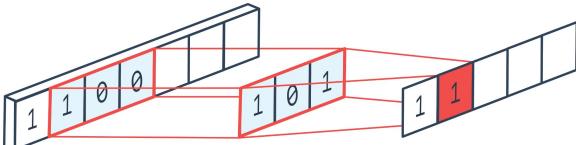
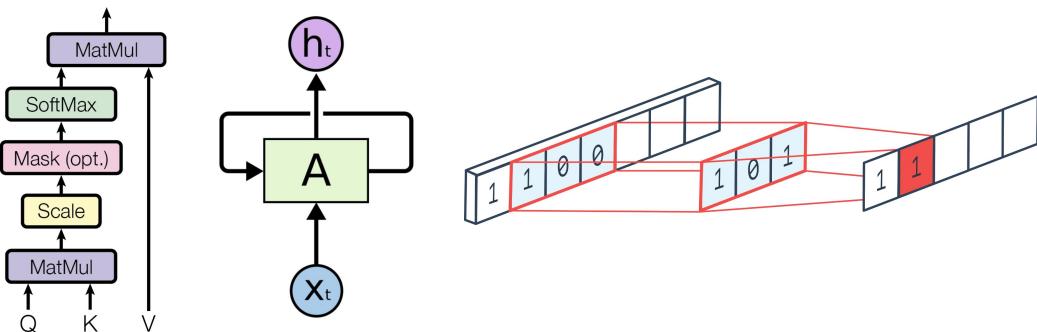
# The complexity of self-attention

For full dependency between all elements

---

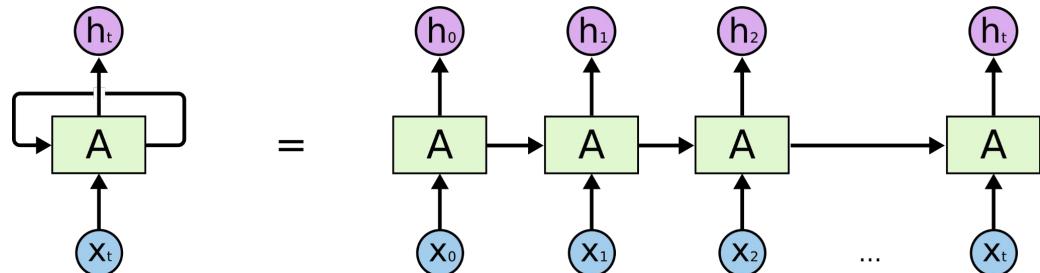
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention			
Recurrent			
Convolutional			

---



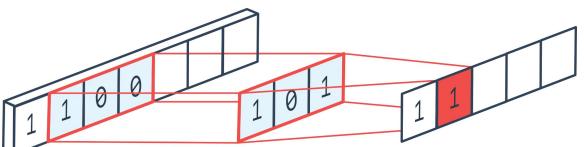
# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Recurrent	$O(n \cdot d^2)$ sequence length dimensionality	$O(n)$	$O(n)$



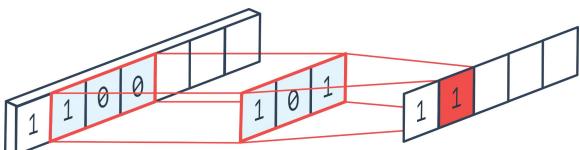
# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Recurrent Convolutional	$O(n \cdot d^2)$ $O(k \cdot n \cdot d^2)$	$O(n)$ $O(1)$	$O(n)$ $O(\log_k(n))$



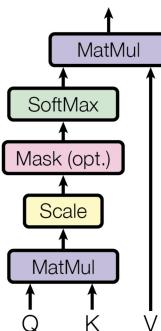
# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Recurrent Convolutional	$O(n \cdot d^2)$ $O(k \cdot n \cdot d^2)$	$O(n)$ $O(1)$	$O(n)$ $O(\log_k(n))$



# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention			
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

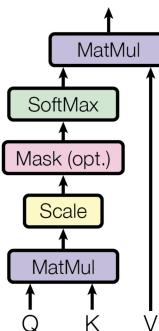


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
$$Q, K, V \in \mathbb{R}^{N \times d_k}$$



# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$		
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

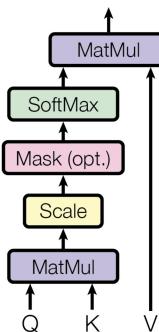


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
$$Q, K, V \in \mathbb{R}^{N \times d_k}$$



# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

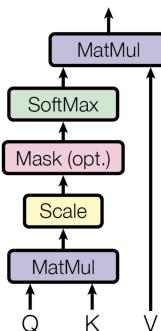


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
$$Q, K, V \in \mathbb{R}^{N \times d_k}$$



# The complexity of self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

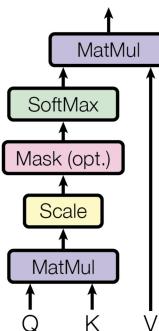


$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
$$Q, K, V \in \mathbb{R}^{N \times d_k}$$



# The complexity of self-attention

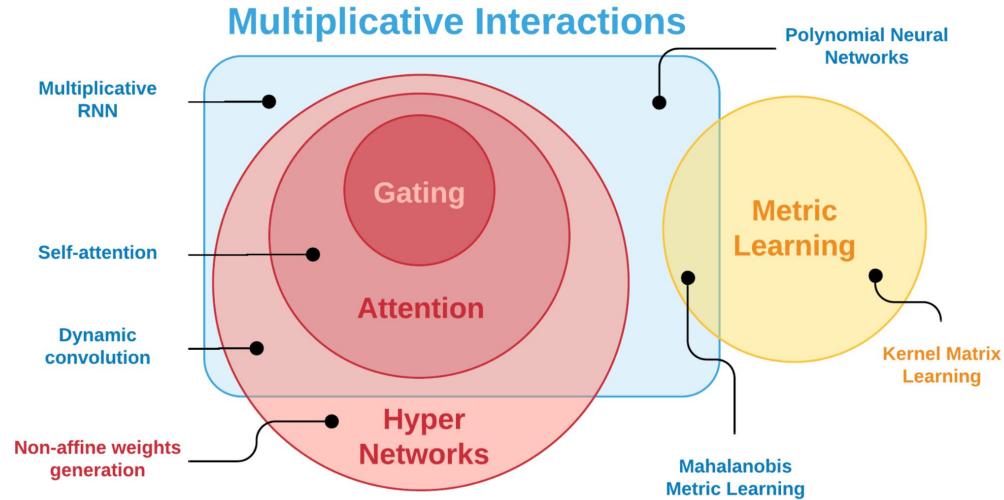
Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$
$$Q, K, V \in \mathbb{R}^{N \times d_k}$$



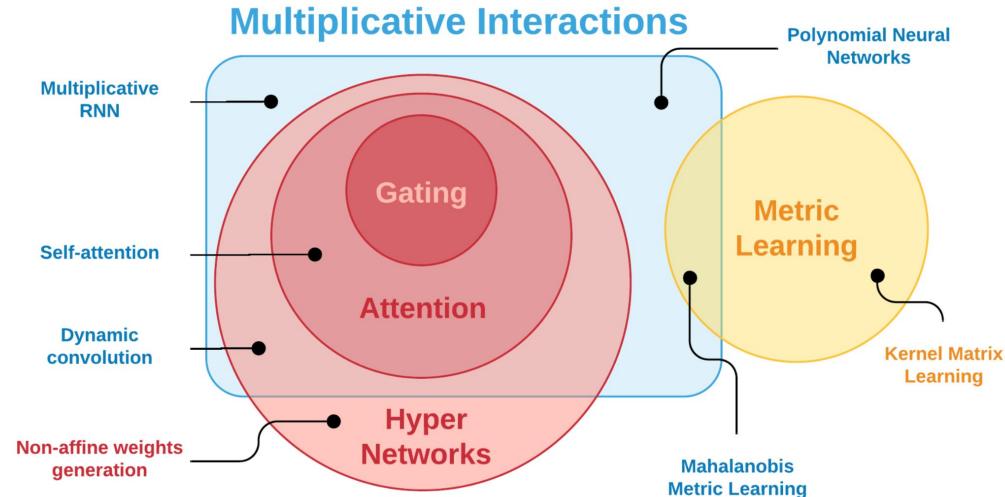
# Understanding Self-Attention



# Understanding Self-Attention

General class of multiplicative interactions:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{z}^T \mathbb{W} \mathbf{x} + \mathbf{z}^T \mathbf{U} + \mathbf{V} \mathbf{x} + \mathbf{b}$$



# Understanding Self-Attention

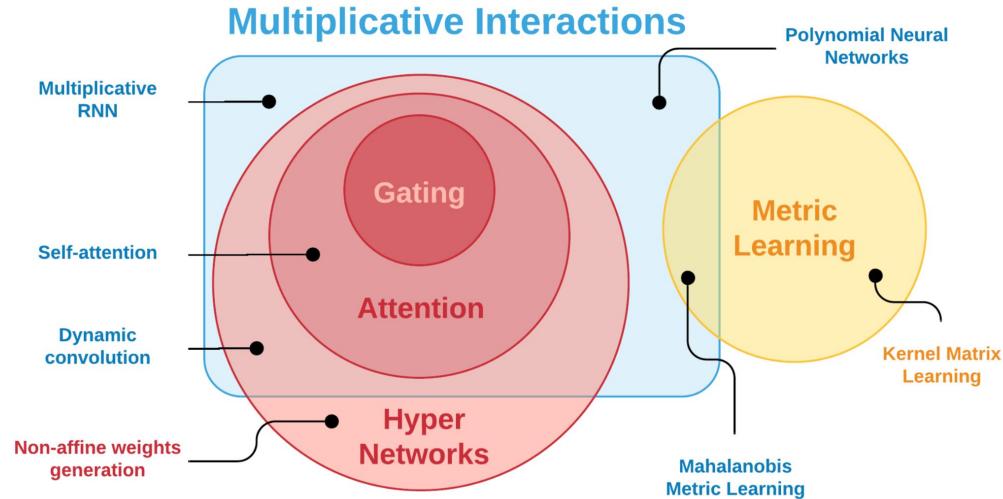
General class of multiplicative interactions:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{z}^T \mathbb{W} \mathbf{x} + \mathbf{z}^T \mathbf{U} + \mathbf{V} \mathbf{x} + \mathbf{b}$$

can be written as:

$$\mathbf{W}' = \mathbf{z}^T \mathbb{W} + \mathbf{V} \quad \mathbf{b}' = \mathbf{z}^T \mathbf{U} + \mathbf{b}$$

$$\mathbf{y} = \mathbf{W}' \mathbf{x} + \mathbf{b}'$$



# Understanding Self-Attention

General class of multiplicative interactions:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{z}^T \mathbb{W} \mathbf{x} + \mathbf{z}^T \mathbf{U} + \mathbf{V} \mathbf{x} + \mathbf{b}$$

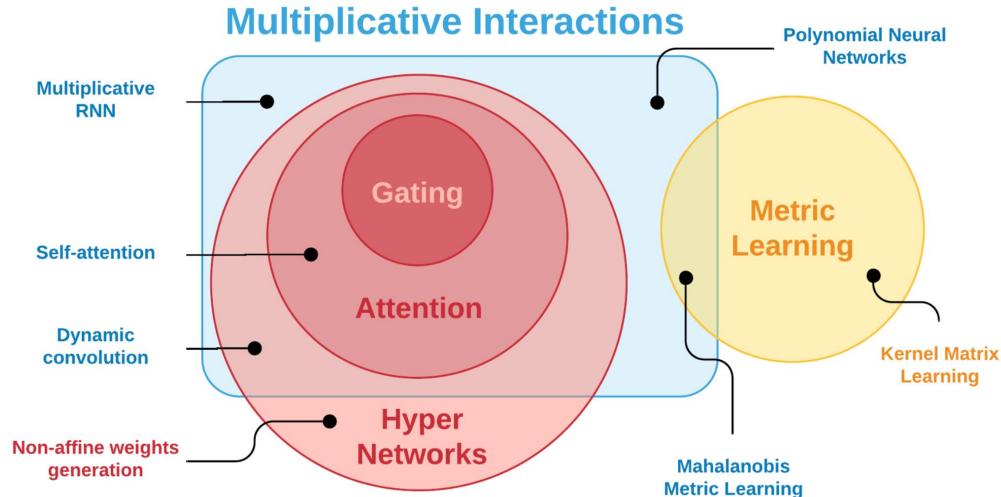
can be written as:

$$\begin{aligned}\mathbf{W}' &= \mathbf{z}^T \mathbb{W} + \mathbf{V} & \mathbf{b}' &= \mathbf{z}^T \mathbf{U} + \mathbf{b} \\ \mathbf{y} &= \mathbf{W}' \mathbf{x} + \mathbf{b}'\end{aligned}$$

Consider diagonal approximation:

$$\mathbf{W}' = \text{diag}(a_1, \dots, a_n) \quad f = \mathbf{a} \odot \mathbf{x}$$

(similarly for biases)



# Understanding Self-Attention

General class of multiplicative interactions:

$$f(\mathbf{x}, \mathbf{z}) = \mathbf{z}^T \mathbb{W} \mathbf{x} + \mathbf{z}^T \mathbf{U} + \mathbf{V} \mathbf{x} + \mathbf{b}$$

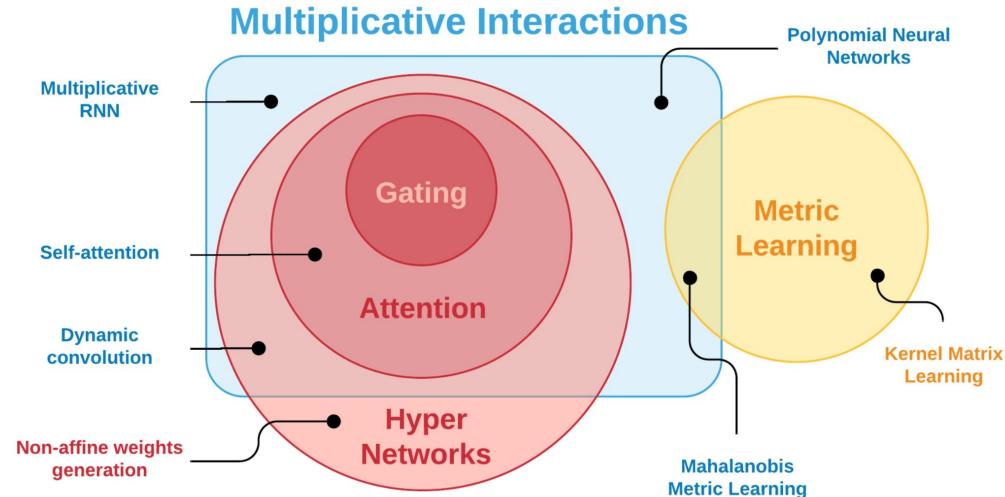
can be written as:

$$\begin{aligned}\mathbf{W}' &= \mathbf{z}^T \mathbb{W} + \mathbf{V} & \mathbf{b}' &= \mathbf{z}^T \mathbf{U} + \mathbf{b} \\ \mathbf{y} &= \mathbf{W}' \mathbf{x} + \mathbf{b}'\end{aligned}$$

Consider diagonal approximation:

$$\mathbf{W}' = \text{diag}(a_1, \dots, a_n) \quad f = \mathbf{a} \odot \mathbf{x}$$

(similarly for biases)

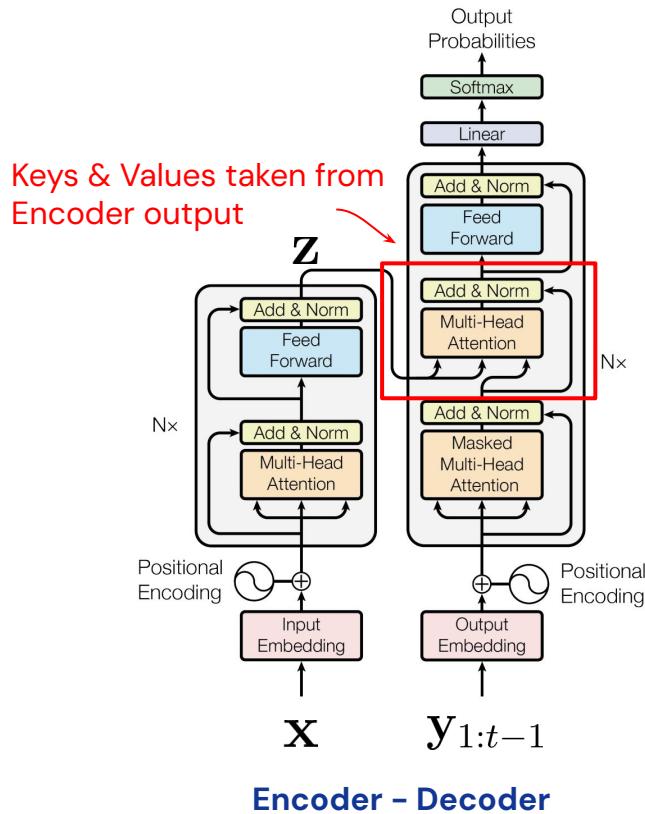


Then, Self-Attention is (with  $\mathbf{m}$  bounded):

$$\mathbf{m} = f(\mathbf{x}, \mathbf{z}) \quad \mathbf{y} = \mathbf{m} \odot \mathbf{x}$$



# Encoder-Decoder v Decoder-Only Transformers



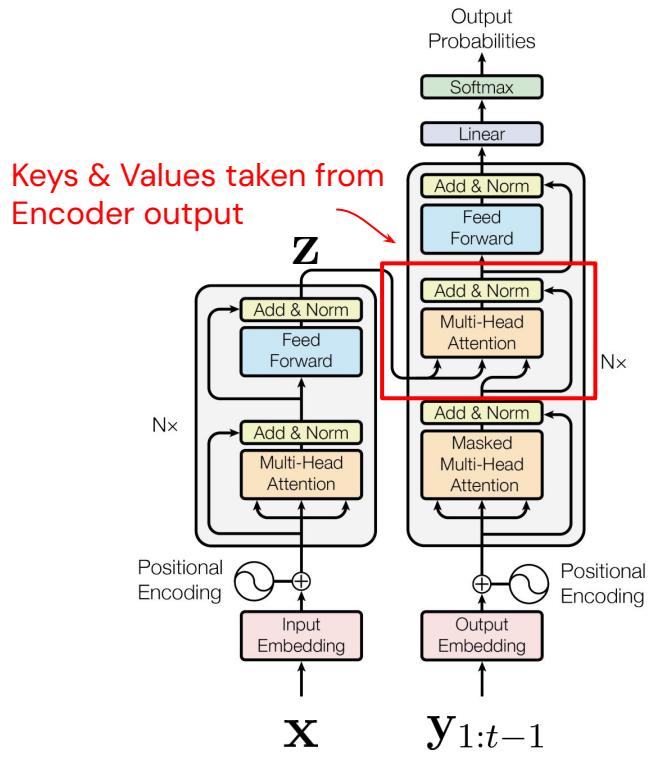
$$H_i^{(l)} = \text{Attention}(QW_i^Q, K_iW^K, V_iW^V)$$

$$Q = Y^{(l-1)}$$

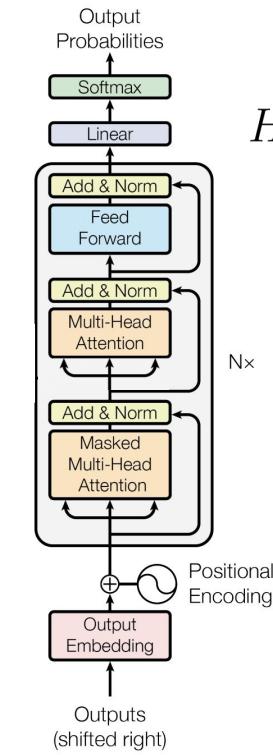
$$K = V = Z^{(l-1)}$$



# Encoder-Decoder v Decoder-Only Transformers



Keys & Values taken from  
Encoder output



$$H_i^{(l)} = \text{Attention}(QW_i^Q, K_iW^K, V_iW^V)$$

$$Q = K = V = Y^{(l-1)}$$

Encoder - Decoder

Decoder only



# Homework for Everyone (Credit to Harvard NLP)

## The Annotated Transformer

```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
        / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

- *v2022: Austin Huang, Suraj Subramanian, Jonathan Sum, Khalid Almubarak, and Stella Biderman.*
- Original: [Sasha Rush](#).

The Transformer has been on a lot of people's minds over the last year five years. This post presents an annotated version of the paper in the form of a line-by-line implementation. It reorders and deletes some sections from the original paper and adds comments throughout. This document itself is a working notebook, and should be a completely usable implementation. Code is available [here](#).

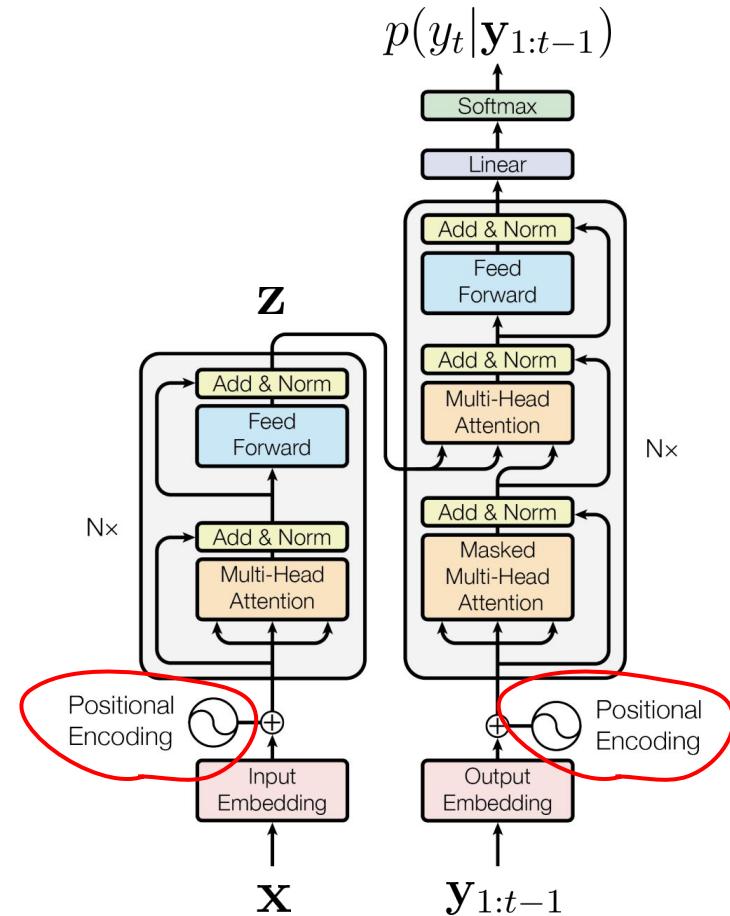


# The Transformer Building Blocks

1. Multi-head Attention

2. Position Encodings

3. Residual connections + Normalization



# Positional Encodings

In order to allow the model to distinguish between sequence positions, we use positional encodings.



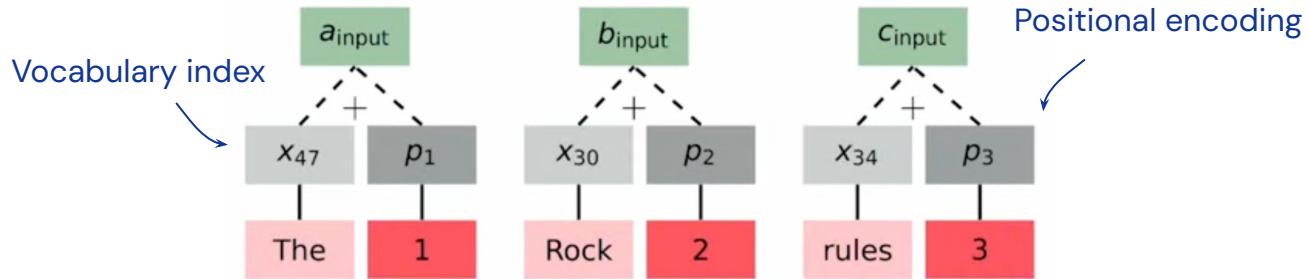
# Positional Encodings

In order to allow the model to distinguish between sequence positions, we use positional encodings. Key questions:

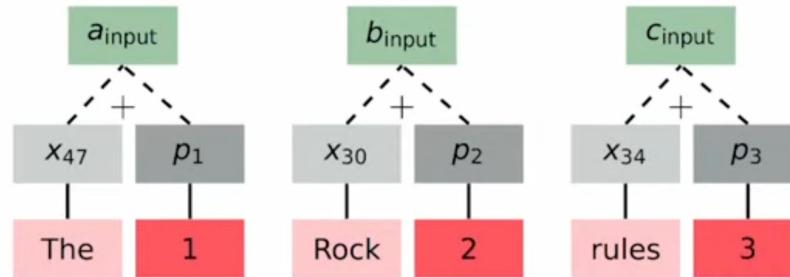
1. Does the set of positions need to be decided ahead of time?
2. Does the scheme hinder generalization to new positions?



# Absolute Positional Encodings



# Absolute Positional Encodings



1. Set of positions need to be decided ahead of time (to normalize).
2. Scheme hinders generalization to new positions:

$$\begin{matrix} \text{The} & 1 & \text{Rock} & 2 \end{matrix} \neq \begin{matrix} \text{The} & 15 & \text{Rock} & 16 \end{matrix}$$



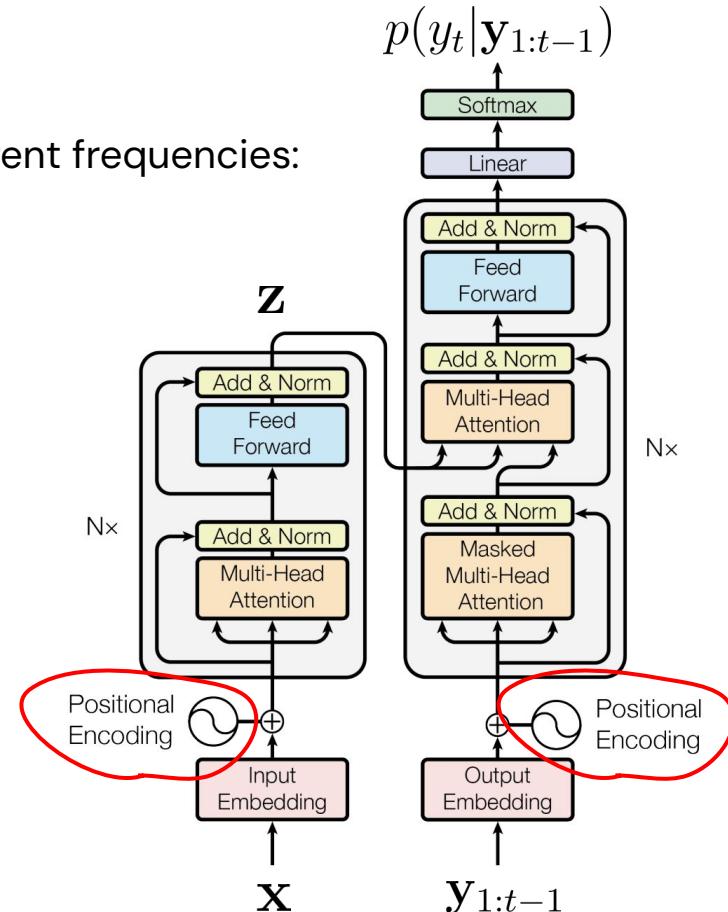
# Frequency Positional Encodings

Instead, the original transformer used additive values at different frequencies:

$$PE_{t,2i} = \sin\left(\frac{t}{10,000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{t,2i+1} = \cos\left(\frac{t}{10,000^{\frac{2i+1}{d_{model}}}}\right)$$

where PE has the same **dimensionality** as our embeddings  
and we have a wavelength from  $(2\pi, 10000 \cdot 2\pi)$



# Frequency Positional Encodings

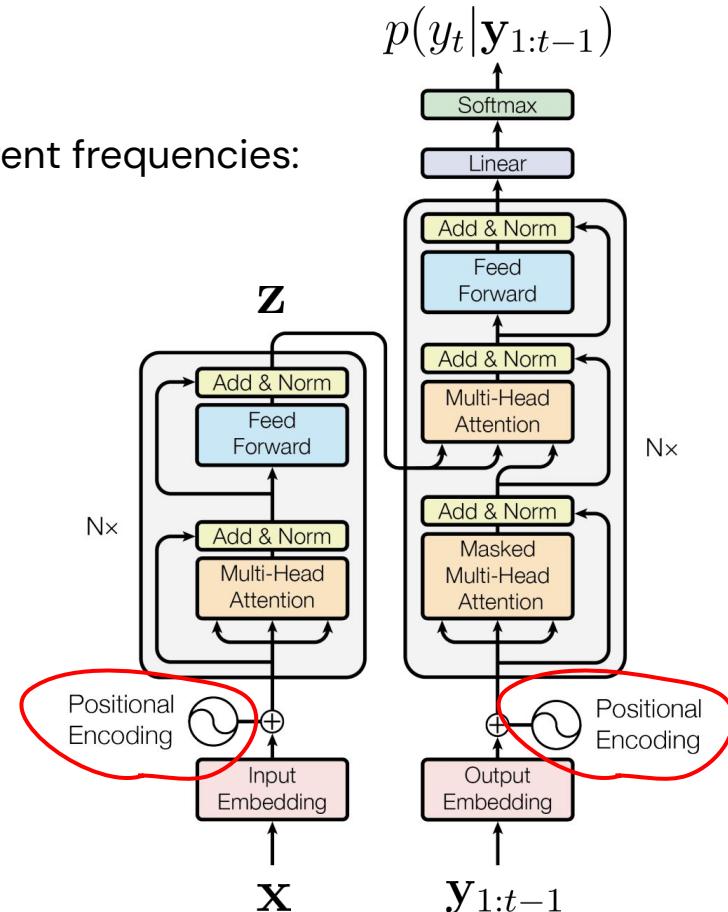
Instead, the original transformer used additive values at different frequencies:

$$PE_{t,2i} = \sin\left(\frac{t}{10,000^{\frac{2i}{d_{model}}}}\right)$$

Position, embedding id

$$PE_{t,2i+1} = \cos\left(\frac{t}{10,000^{\frac{2i+1}{d_{model}}}}\right)$$

where PE has the same **dimensionality** as our embeddings  
and we have a wavelength from  $(2\pi, 10000 \cdot 2\pi)$



# Modern Positional Encodings

Frequency Positional Encodings have essentially been replaced in modern Transformers.  
Popular alternatives

(1) Relative positional encodings (Example with window size 1):



# Modern Positional Encodings

Frequency Positional Encodings have essentially been replaced in modern Transformers.  
Popular alternatives

(1) Relative positional encodings (Example with window size 1):

y1	y2	y3	y4	y5	
$p(3,1) = w(-1)$	$p(3,2) = w(-1)$	$p(3,3) = w(0)$	$p(3,4) = w(1)$	$p(3,4) = w(1)$	$\mathbf{v}_i^{(l)} = \sum_{j=1}^N (\alpha_{i,j} \cdot \mathbf{v}_j^{l-1}) + \mathbf{p}_{i,j}^v$
$p(2,1) = w(-1)$	$p(2,2) = w(0)$	$p(2,3) = w(1)$	$p(2,4) = w(1)$	$p(2,5) = w(1)$	

[\[Vaswani et al. "Attention is all you need", 2017\]](#)

[\[More details on positional encodings, Stanford XCS224U\]](#)

[\[A gentle introduction into positional encodings\]](#)



# Modern Positional Encodings

Frequency Positional Encodings have essentially been replaced in modern Transformers.  
Popular alternatives

(1) Relative positional encodings (Example with window size 1):

y1	y2	y3	y4	y5
p(3,1) = w(-1)	p(3,2) = w(-1)	p(3,3) = w(0)	p(3,4) = w(1)	p(3,4) = w(1)
p(2,1) = w(-1)	p(2,2) = w(0)	p(2,3) = w(1)	p(2,4) = w(1)	p(2,5) = w(1)

$$\mathbf{v}_i^{(l)} = \sum_{j=1}^N (\alpha_{i,j} \cdot \mathbf{v}_j^{l-1}) + \mathbf{p}_{i,j}^v$$

(2) Rotary positional encodings (RoPE, Su et al., 2021)

[\[Vaswani et al. "Attention is all you need", 2017\]](#)

[\[More details on positional encodings, Stanford XCS224U\]](#)

[\[A gentle introduction into positional encodings\]](#)

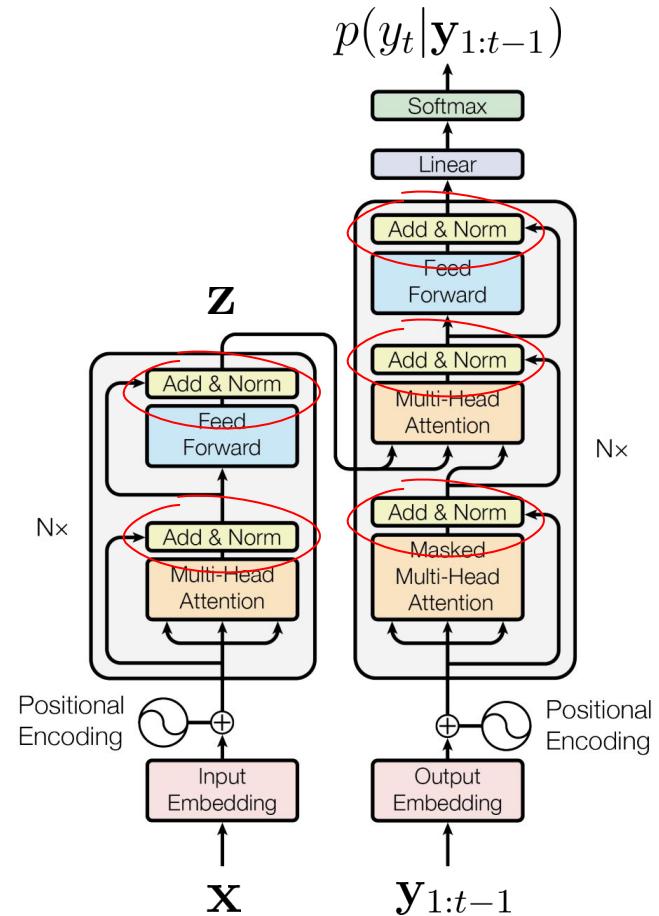


# The Transformer Building Blocks

1. Multi-head Attention

2. Position Encodings

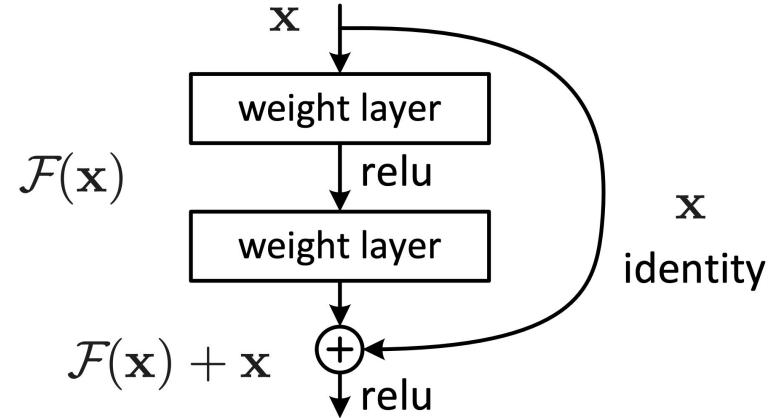
3. Residual connections + Normalization



# Residual Connections + Normalization

$\mathcal{H}(\mathbf{x}) \rightarrow$  desired mapping

$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x} \rightarrow$  chosen mapping

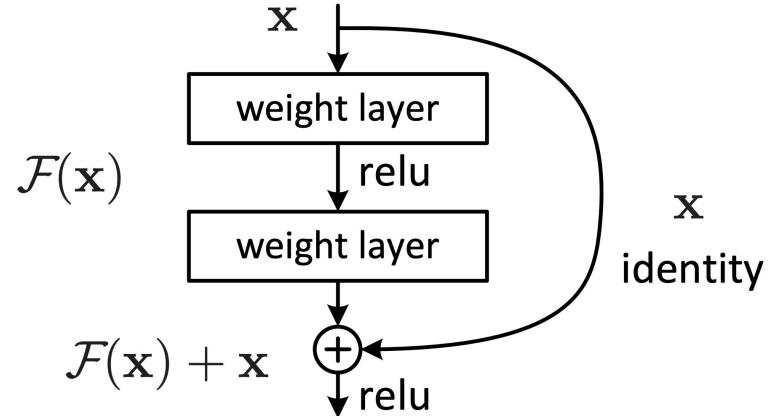


# Residual Connections + Normalization

$\mathcal{H}(\mathbf{x}) \rightarrow$  desired mapping

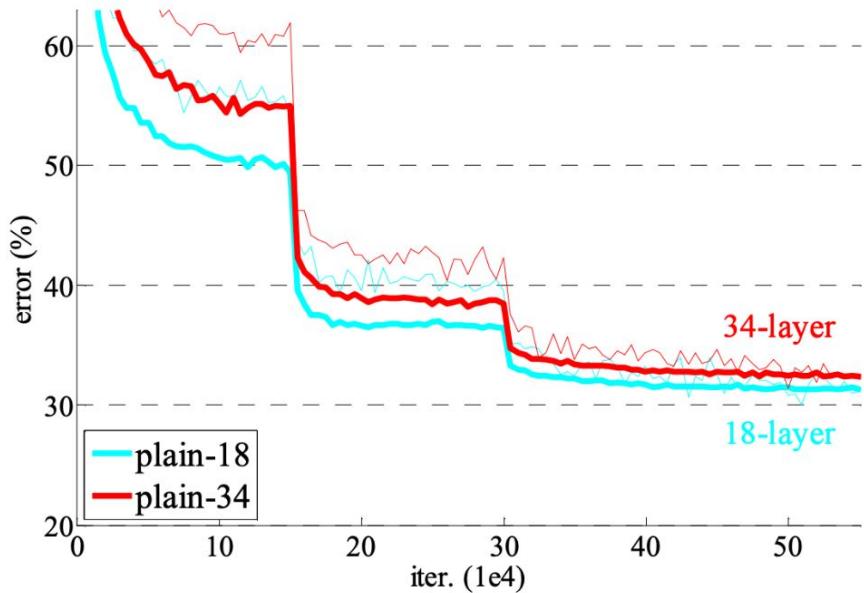
$\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x} \rightarrow$  chosen mapping

The motivation for this are "skip-connections", which had empirically been observed to help train deeper networks in many previous studies (see vanishing gradient problem (LSTM, Hochreiter et al., 1997)).

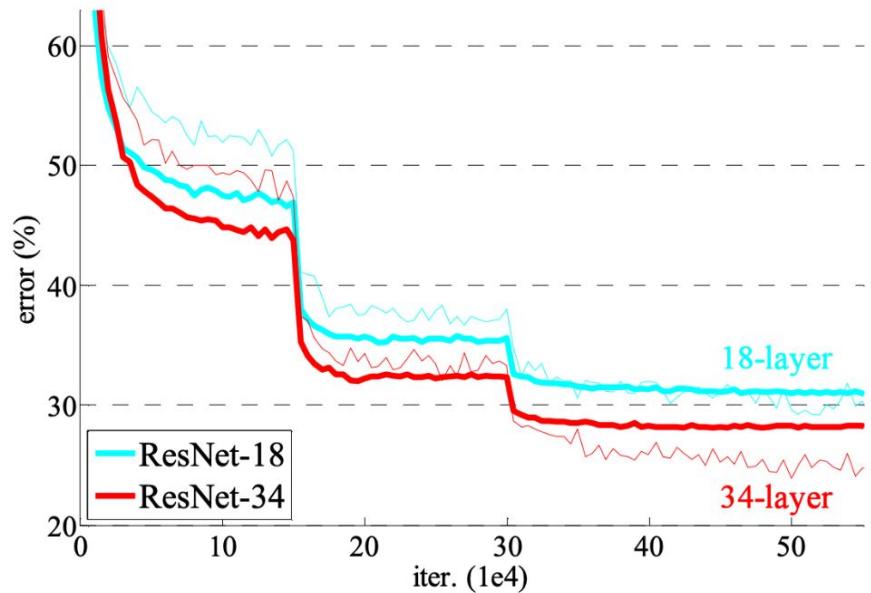


# Residual Connections + Normalization

Plain network



Residual Network



# Batch Normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

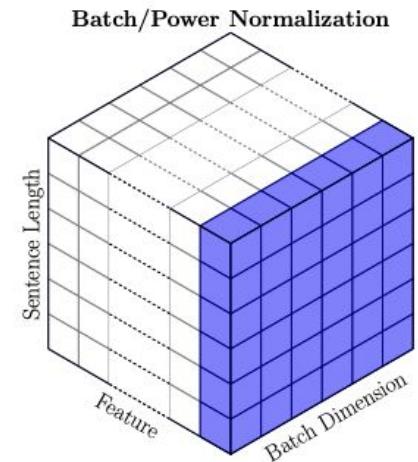
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Pre-Activations



# Batch Normalization

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

This is a bug magnet:

1. Introduces a dependency between examples in the batch
2. Implementations are stateful, to track the statistics over the course of training

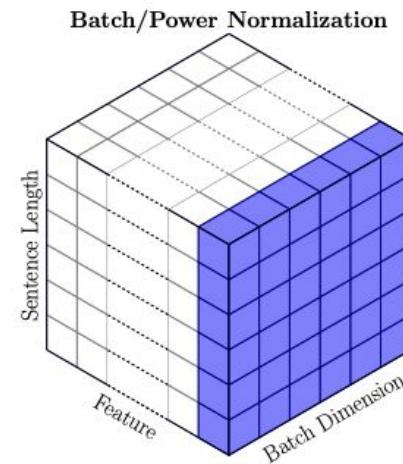
This can lead to:

- Information leakage (e.g. in autoregressive % contrastive models)
- Unattractive batch size effects
- Affects optimization (improving gradient propagation, but adding randomness in a tightly coupled way)



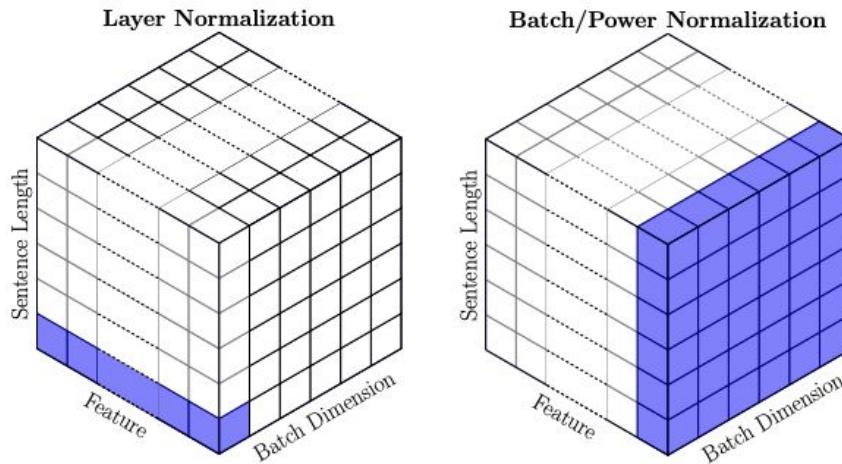
# Layer Normalization

$$\mathbf{h}^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b} \right] \quad \mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t \quad \sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu^t)^2}$$



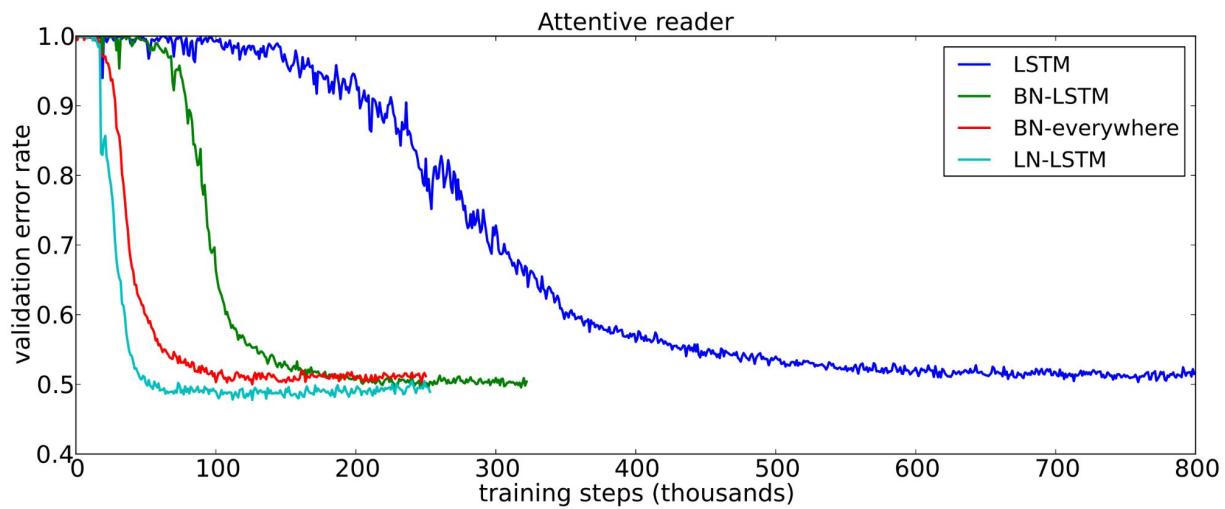
# Layer Normalization

$$\mathbf{h}^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b} \right] \quad \mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t \quad \sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu^t)^2}$$



# Layer Normalization

$$\mathbf{h}^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b} \right] \quad \mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t \quad \sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu^t)^2}$$



# Layer Normalization in Transformers

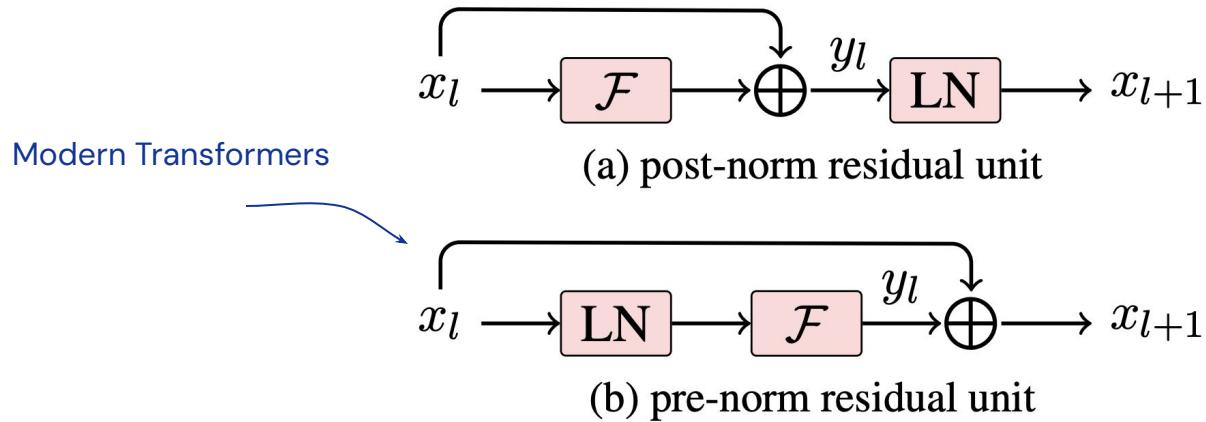


Figure 1: Examples of pre-norm residual unit and post-norm residual unit.  $\mathcal{F}$  = sub-layer, and LN = layer normalization.



# Modern Transformers: Architecture & Training Tricks

## Training:

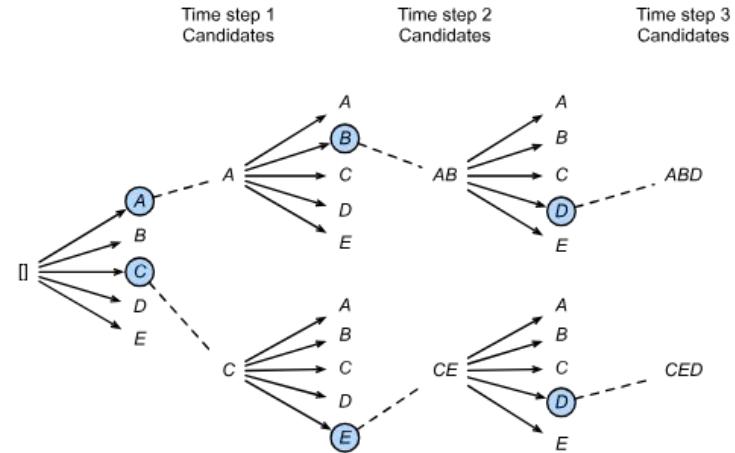
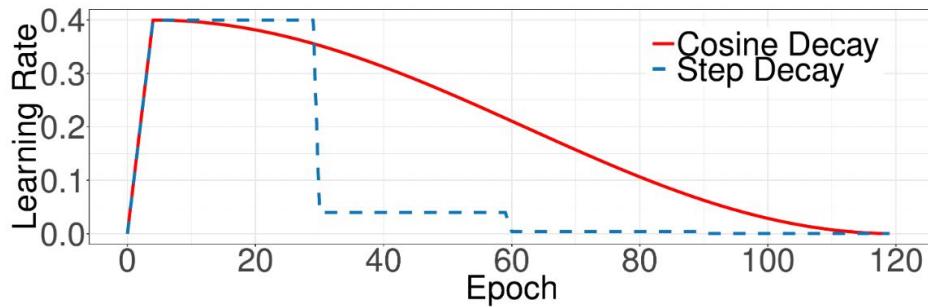
- Dropout (Srivastava et al., 2014) during at every layer just before adding residual
- AdamW optimizer with warmup and cosine decay (Loshchilov & Hutter, 2017)
- Label smoothing (Müller et al, 2019)
- Auto-regressive decoding with beam search and length penalties (Graves, 2012)
- Checkpoint-averaging (Izmailov et al., 2018)



# Modern Transformers: Architecture & Training Tricks

## Training:

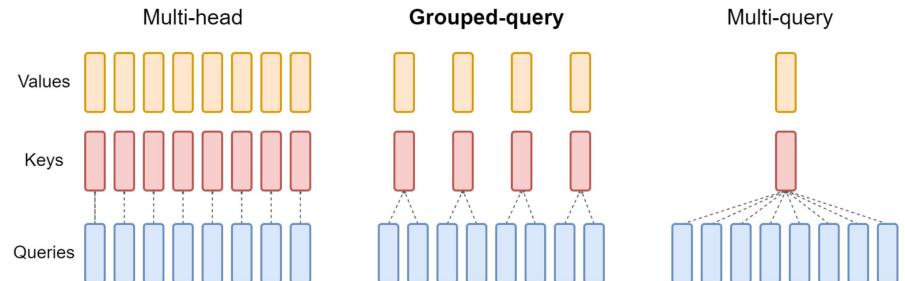
- Dropout (Srivastava et al., 2014) during at every layer just before adding residual
- AdamW optimizer with warmup and cosine decay (Loshchilov & Hutter, 2017)
- Label smoothing (Müller et al, 2019)
- Auto-regressive decoding with beam search and length penalties (Graves, 2012)
- Checkpoint-averaging (Izmailov et al., 2018)



# Modern Transformers: Architecture & Training Tricks

## Architecture:

- Pre-normalization using RMSNorm (Zhang and Sennrich, 2019)
- SwiGLU activation function (Shazeer, 2020)
- Rotary positional embeddings (RoPE, Su et al. 2022)
- Grouped-query attention (GQA, Ainslie et al., 2023)
- Flash or Ring Attention (Dao et al, 2022; Liu et al, 2023)
- Mixture of Experts (MoE, Mistral AI., 2023)



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

## Part 2:

- **In-Context Learning & Prompting**
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations



# Language Models as Few-Shot Learners (GPT-3)

Prior to GPT-2: Rich literature on so called few-shot learning algorithms

→ Goal: Develop ML methods that can perform well on a novel, unseen task for which we only have a small number of labeled examples.



# Language Models as Few-Shot Learners (GPT-3)

Prior to GPT-2: Rich literature on so called few-shot learning algorithms

→ Goal: Develop ML methods that can perform well on a novel, unseen task for which we only have a small number of labeled examples.

→ Main strategy: Episodic training

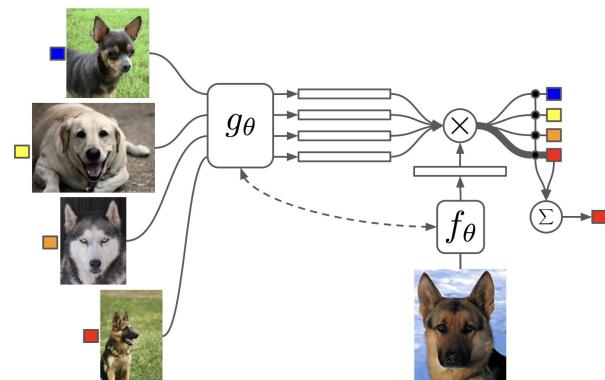
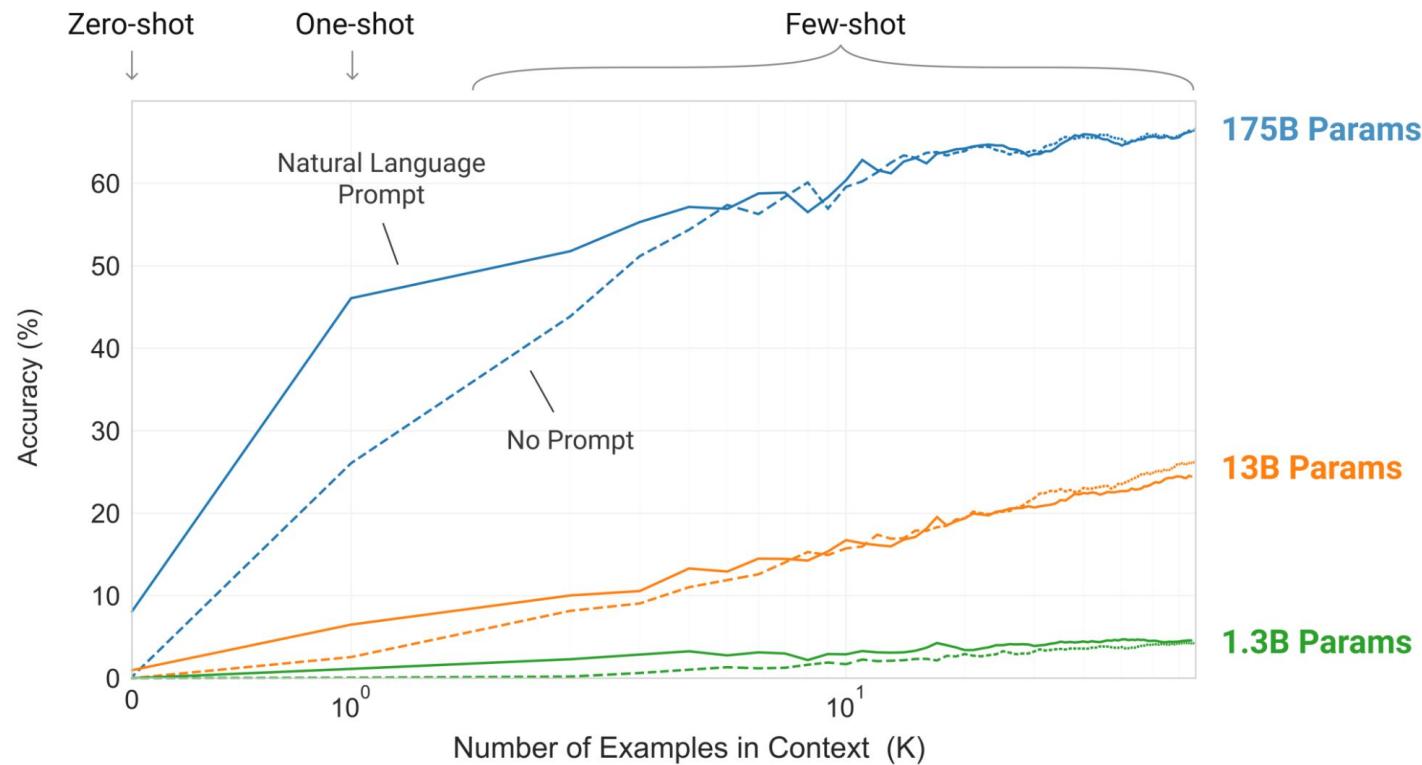


Figure 1: Matching Networks architecture

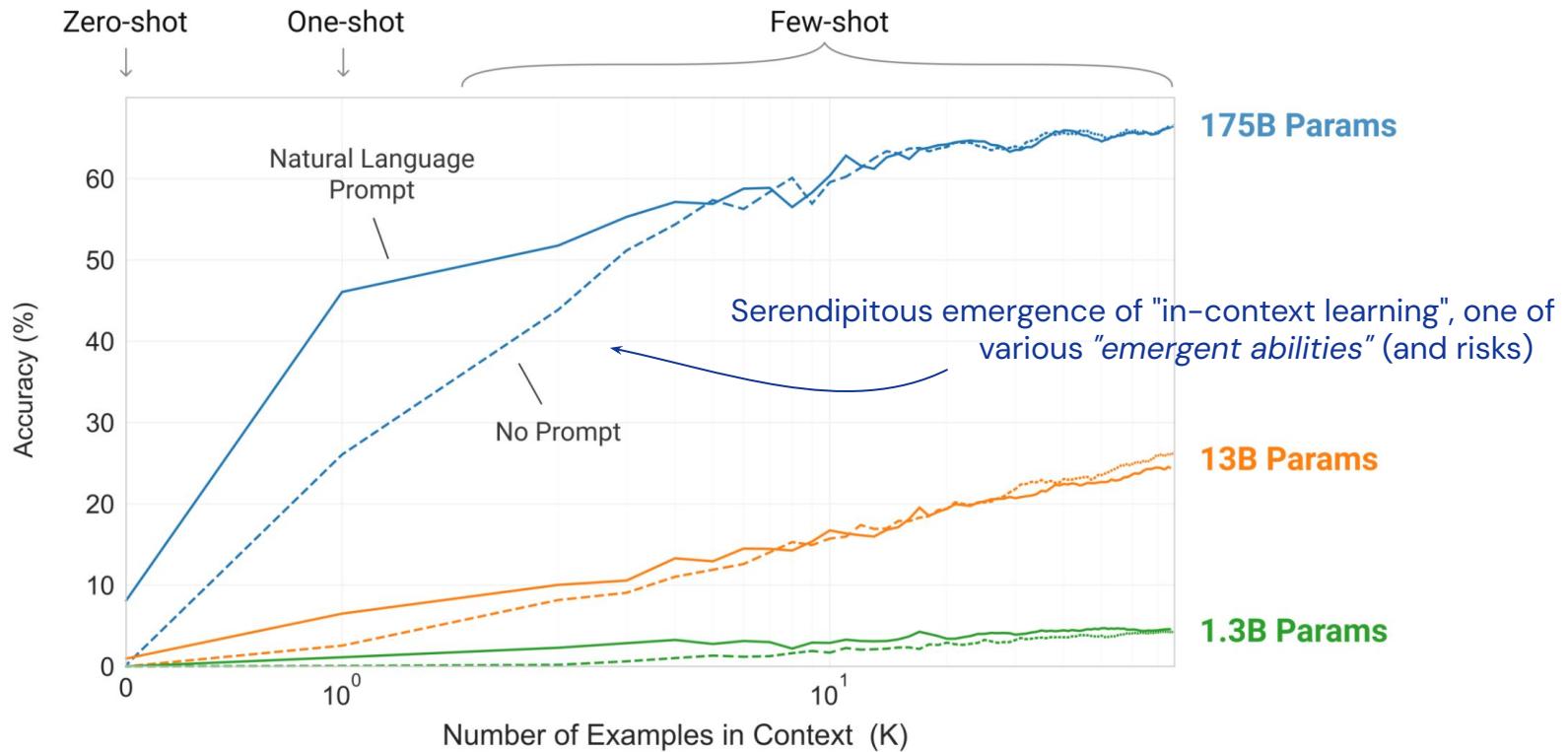
*"our training procedure is based on a simple machine learning principle: test and train conditions must match"*  
*(Vinyals et al., 2017)*



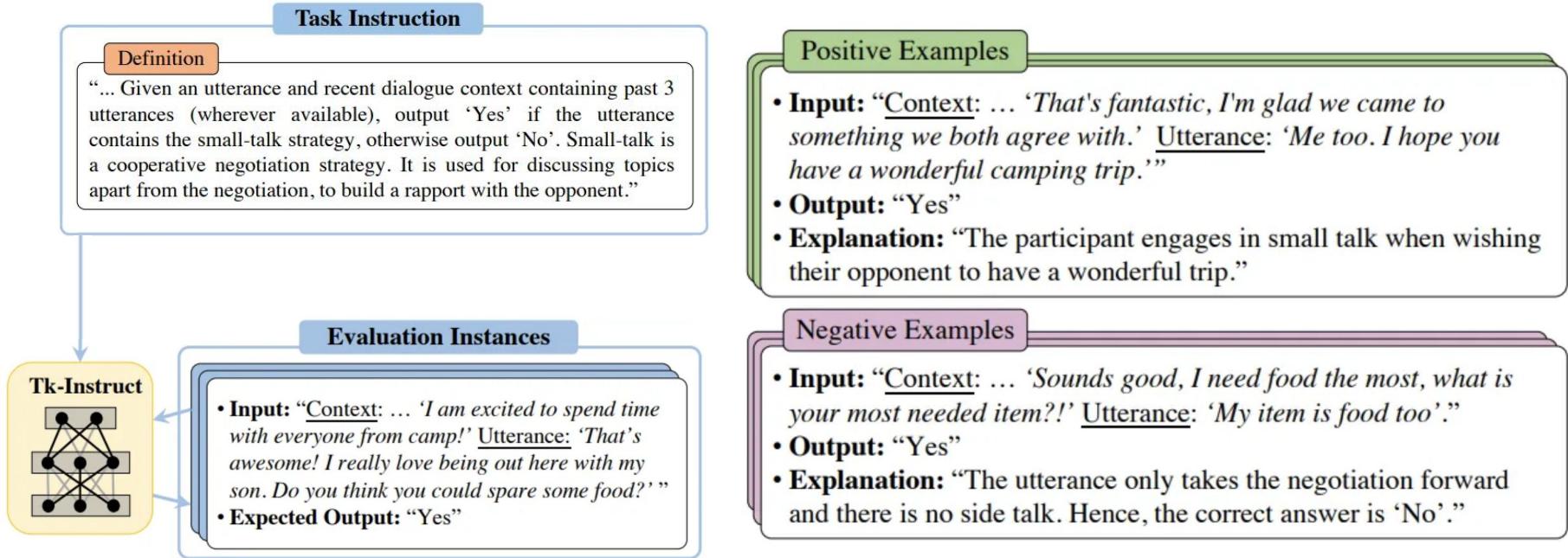
# Language Models as Few-Shot Learners (GPT-3)



# Language Models as Few-Shot Learners (GPT-3)



# An In-Context Learning example



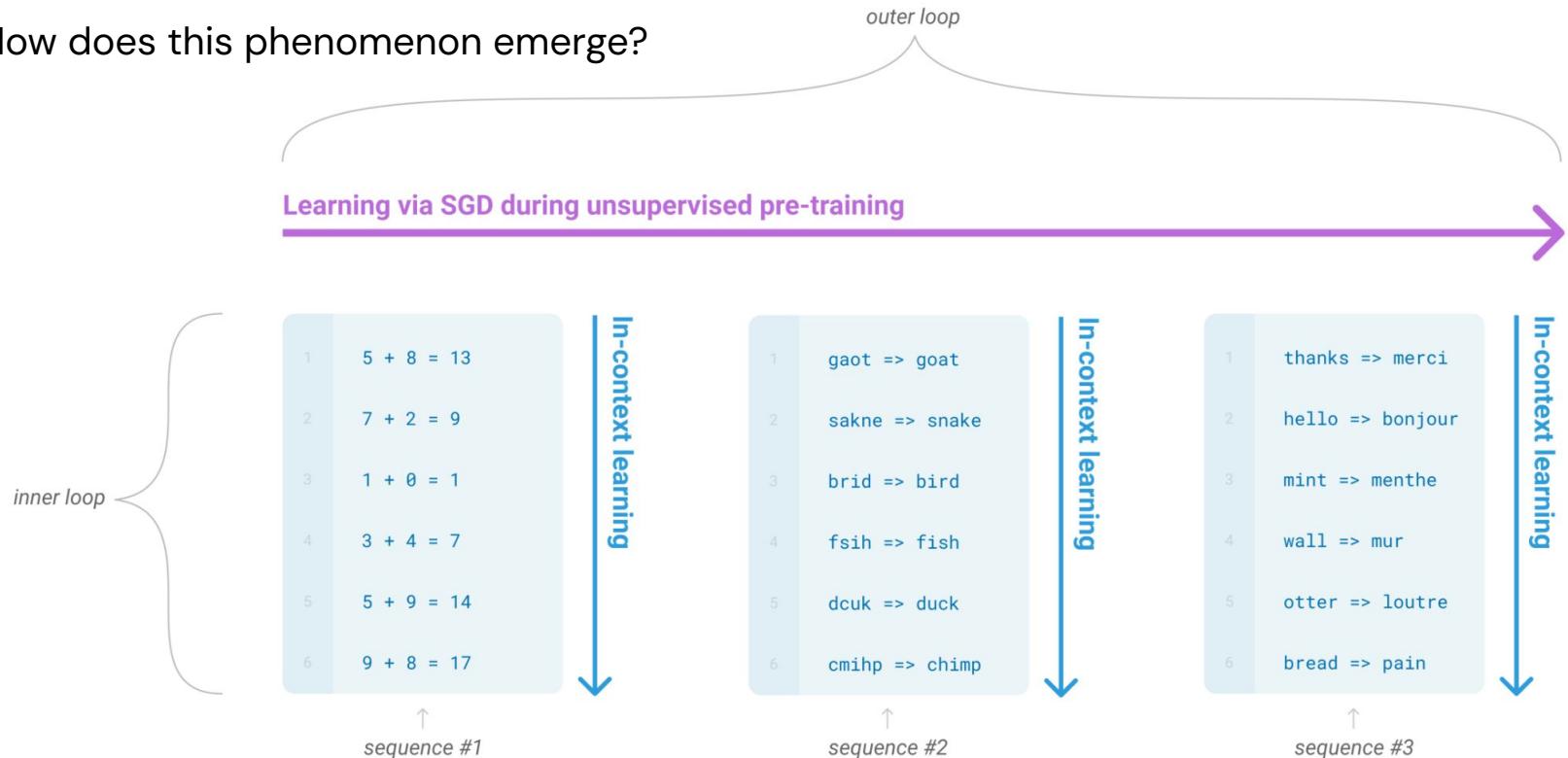
# Language Models as Few-Shot Learners (GPT-3)

How does this phenomenon emerge?

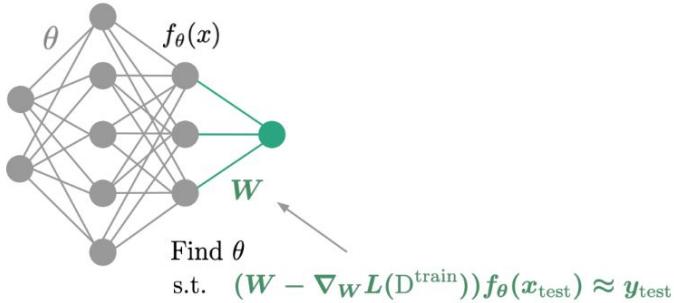


# Language Models as Few-Shot Learners (GPT-3)

How does this phenomenon emerge?



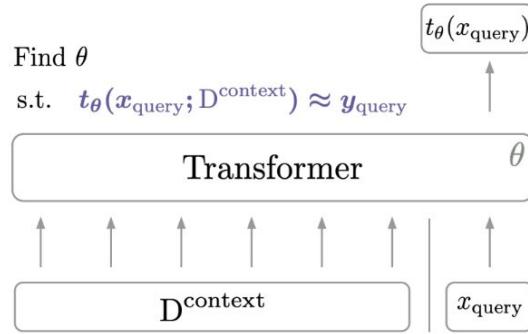
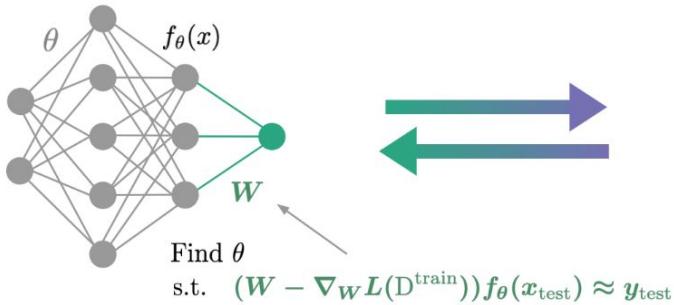
# Understanding In-Context Learning



Could Self-Attention implement a gradient-based learning algorithm?



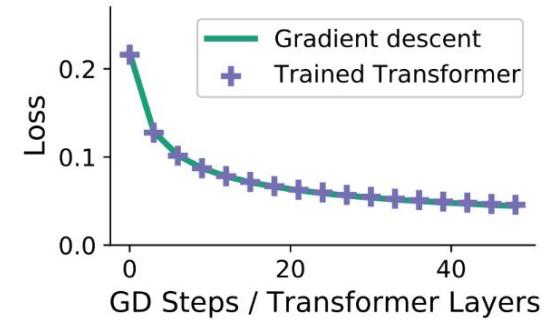
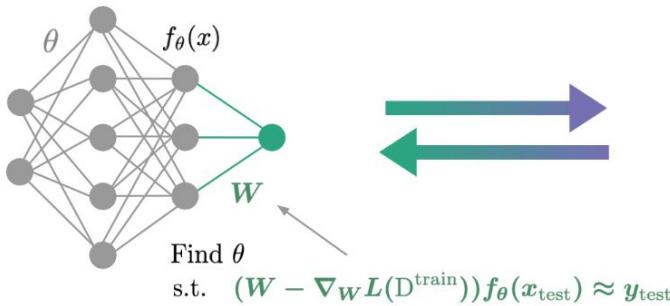
# Understanding In-Context Learning



Could Self-Attention implement a gradient-based learning algorithm?



# Understanding In-Context Learning



Could Self-Attention implement a gradient-based learning algorithm?



# Understanding In-Context Learning in Linear Transformers

Squared error for a linear model:

$$L(W) = \frac{1}{2N} \sum_{i=1}^N \|Wx_i - y_i\|^2$$

Update with Gradient descent:

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (Wx_i - y_i)x_i^T$$



# Understanding In-Context Learning in Linear Transformers

Squared error for a linear model:

$$L(W) = \frac{1}{2N} \sum_{i=1}^N \|Wx_i - y_i\|^2$$

Update with Gradient descent:

$$\Delta W = -\eta \nabla_W L(W) = -\frac{\eta}{N} \sum_{i=1}^N (Wx_i - y_i)x_i^T$$

New Loss:

$$\begin{aligned} L(W + \Delta W) &= \frac{1}{2N} \sum_{i=1}^N \|(W + \Delta W)x_i - y_i\|^2 \\ &= \frac{1}{2N} \sum_{i=1}^N \|Wx_i - (y_i - \Delta y_i)\|^2 \end{aligned}$$

"transformed targets"



# Understanding In-Context Learning in Linear Transformers

$$\begin{aligned} e_j &\leftarrow e_j + \text{SA}_\theta(j, \{e_1, \dots, e_N\}) \\ &= e_j + \sum_h P_h V_h \text{softmax}(K_h^T q_{h,j}) \end{aligned} \quad \begin{aligned} q_{h,j} &= W_{h,Q} e_j \\ &\quad (\text{similarly for Values and Keys}) \end{aligned}$$

Tokens



# Understanding In-Context Learning in Linear Transformers

$$\begin{aligned} e_j &\leftarrow e_j + \text{SA}_\theta(j, \{e_1, \dots, e_N\}) \\ &= e_j + \sum_h P_h V_h \text{softmax}(K_h^T q_{h,j}) \end{aligned}$$

Tokens

$$q_{h,j} = W_{h,Q} e_j$$

(similarly for Values and Keys)

Let's consider a linear Transformer:

$$e_j \leftarrow e_j + \text{LSA}_\theta(j, \{e_1, \dots, e_N\}) = e_j + \sum_h P_h V_h K_h^T q_{h,j}$$



# Understanding In-Context Learning in Linear Transformers

$$\begin{aligned} e_j &\leftarrow e_j + \text{SA}_\theta(j, \{e_1, \dots, e_N\}) \\ &= e_j + \sum_h P_h V_h \text{softmax}(K_h^T q_{h,j}) \end{aligned} \quad \begin{aligned} q_{h,j} &= W_{h,Q} e_j \\ &\quad \text{(similarly for Values and Keys)} \end{aligned}$$

Tokens

Let's consider a linear Transformer:

$$e_j \leftarrow e_j + \text{LSA}_\theta(j, \{e_1, \dots, e_N\}) = e_j + \sum_h P_h V_h K_h^T q_{h,j}$$

Let's set up an in-context regression problem:

$$\begin{aligned} e_j &= (x_j, y_j) \in \mathbb{R}^{N_x+N_y} \\ e_{N+1} &= (x_{N+1}, y_{N+1}) = (x_{\text{test}}, \hat{y}_{\text{test}}) = e_{\text{test}} \end{aligned}$$



# Understanding In-Context Learning in Linear Transformers

**Proposition 1.** *Given a 1-head linear attention layer and the tokens  $e_j = (x_j, y_j)$ , for  $j = 1, \dots, N$ , one can construct key, query and value matrices  $W_K, W_Q, W_V$  as well as the projection matrix  $P$  such that a Transformer step on every token  $e_j$  is identical to the gradient-induced dynamics  $e_j \leftarrow (x_j, y_j) + (0, -\Delta W x_j) = (x_j, y_j) + P V K^T q_j$  such that  $e_j = (x_j, y_j - \Delta y_j)$ . For the test data token  $(x_{N+1}, y_{N+1})$  the dynamics are identical.*

Details in the paper



# Prompting

In practice, it turns out that in-context learning is extremely sensitive to the way prompts are phrased:



# Prompting

In practice, it turns out that in-context learning is extremely sensitive to the way prompts are phrased. These all give very different results using the same data:

<b>Model</b>	<b>Prompt</b>
CoT	““Let’s think step by step.””
PS	“Let’s first understand the problem and devise a plan to solve the problem. Then, let’s carry out the plan and solve the problem step by step.””
PS+	“Let’s first understand the problem, extract relevant variables and their corresponding numerals, and make a plan. Then, let’s carry out the plan, calculate intermediate variables (pay attention to correct numerical calculation and commonsense), solve the problem step by step, and show the answer.””
APE	“Let’s work this out in a step by step way to be sure we have the right answer.””
OPRO	“Take a deep breath and work on this problem step-by-step.””



# Prompting

In practice, it turns out that in-context learning is extremely sensitive to the way prompts are phrased:

Method	LLM	MultiArith*	SingleEq*	AddSub*	SVAMP*	SQA	CSQA	AQuA-RAT	GSM8K	
Zero-shot	CoT	text-davinci-003	(83.8)	(88.1)	(85.3)	(69.9)	(63.8)	(65.2)	(38.9)	(56.4)
	PoT	text-davinci-003	(92.2)	(91.7)	(85.1)	(70.8)	—	—	(43.9)	(57.0)
	PS	text-davinci-003	(87.2)	(89.2)	(88.1)	(72.0)	—	—	(42.5)	(58.2)
	PS+	text-davinci-003	(91.8)	(94.7)	<b>(92.2)</b>	(75.7)	(65.4)	(71.9)	(46.0)	(59.3)
	PS	PaLM 2-L	97.7	90.6	72.4	83.8	50.0	77.9	40.2	59.0
	PS+	PaLM 2-L	92.5	94.7	74.4	86.3	50.1	73.3	39.4	60.5
	APE	PaLM 2-L	95.8	82.2	72.2	73.0	38.4	67.3	45.7	77.9
	OPRO	PaLM 2-L	—	—	—	—	—	—	—	80.2
Few-shot	PB (ours)	PaLM 2-L	<b>99.7</b>	<b>96.4</b>	87.8	<b>90.2</b>	<b>71.8</b>	<b>85.4</b>	<b>62.2</b>	<b>83.9</b>
	Manual-CoT	text-davinci-003	(93.6)	(93.5)	<b>(91.6)</b>	(80.3)	(71.2)	(78.3)	(48.4)	(58.4)
	Auto-CoT	text-davinci-003	(95.5)	(92.1)	(90.8)	(78.1)	—	—	(41.7)	(57.1)
	PB (ours)	PaLM 2-L	<b>100.0</b>	<b>98.9</b>	87.1	<b>93.7</b>	<b>80.2</b>	<b>85.9</b>	<b>64.6</b>	<b>83.5</b>

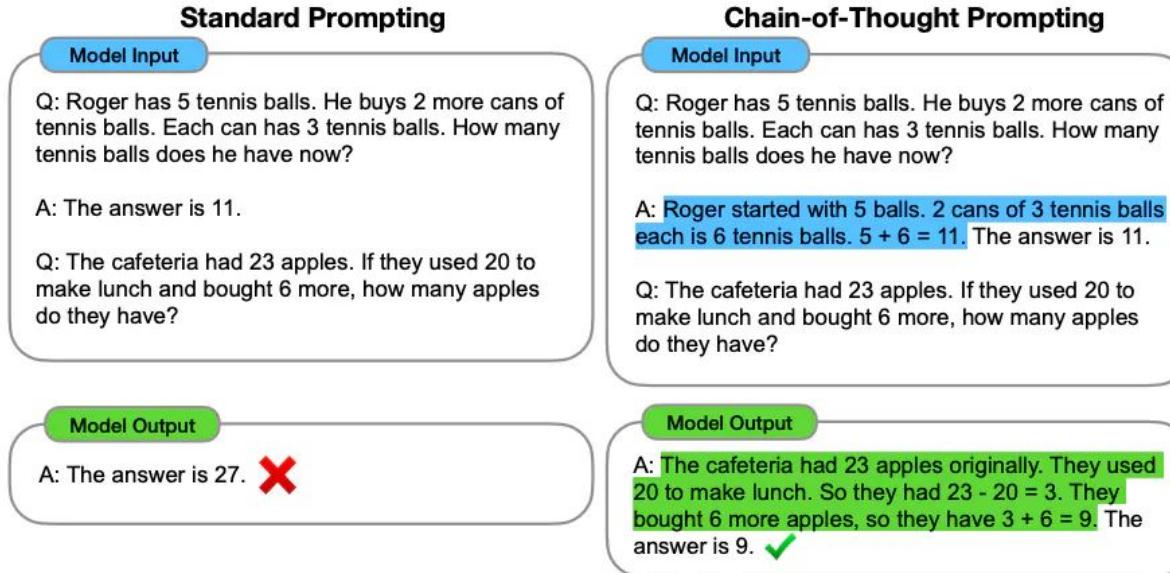


Same data, same model



# Chain-of-Thought Prompting

In practice, it turns out that in-context learning is extremely sensitive to the way prompts are phrased. One common trick that almost always works in adding explanations:



These can be even auto-generated (using another LLM!)



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

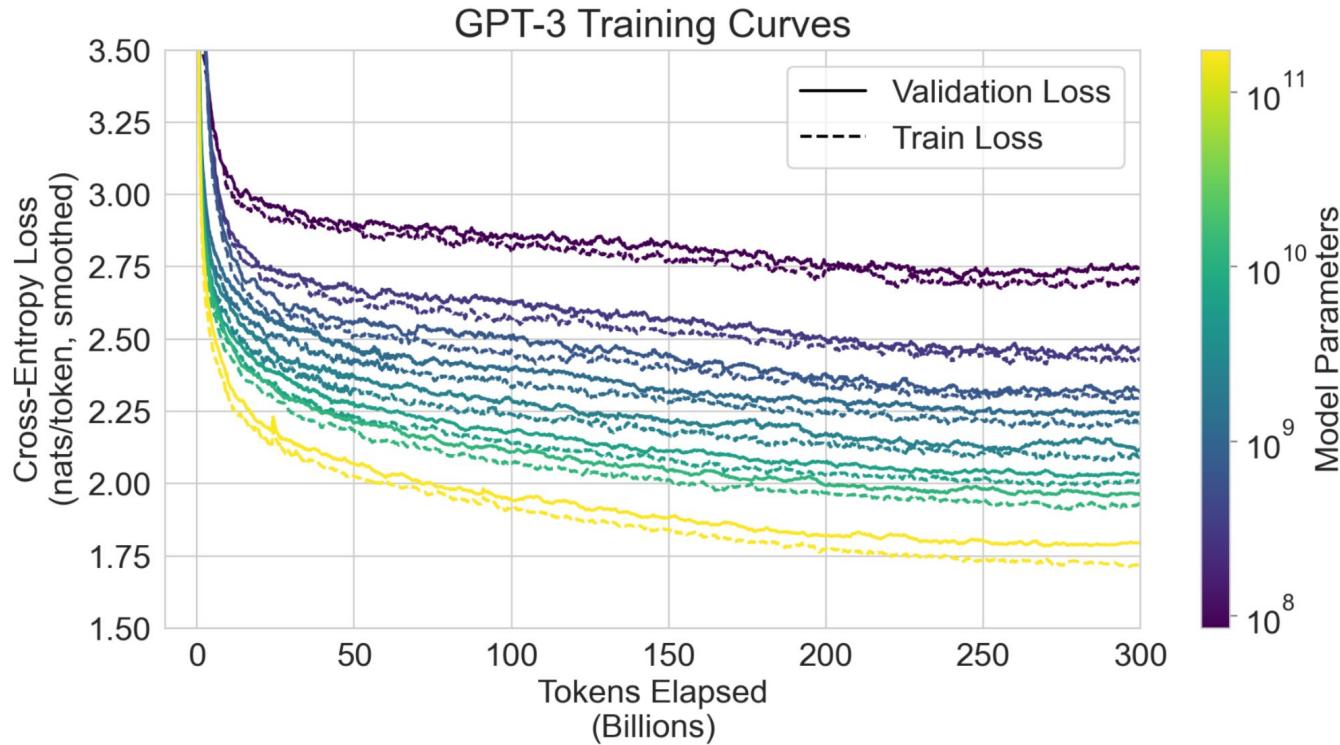
[break]

## Part 2:

- In-Context Learning & Prompting
- **Scaling Laws**
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations



# Can we predict the expected improvement?



# Training Compute Optimal Models

$$N_{opt}(C), D_{opt}(C) = \underset{N,D \text{ s.t. } \text{FLOPs}(N,D)=C}{\operatorname{argmin}} L(N, D)$$

Number of datapoints  
Number of parameters  
FLOPS budget



# Scaling Laws from Classical Risk Decomposition

$$N_{opt}(C), D_{opt}(C) = \operatorname{argmin}_{N, D \text{ s.t. FLOPs}(N, D)=C} L(N, D)$$

Number of data seen so far

Estimated loss

Number of parameters

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$



# Scaling Laws from Classical Risk Decomposition

$$N_{opt}(C), D_{opt}(C) = \operatorname{argmin}_{N,D \text{ s.t. FLOPs}(N,D)=C} L(N, D)$$

Loss for an ideal generative process (entropy of natural text)

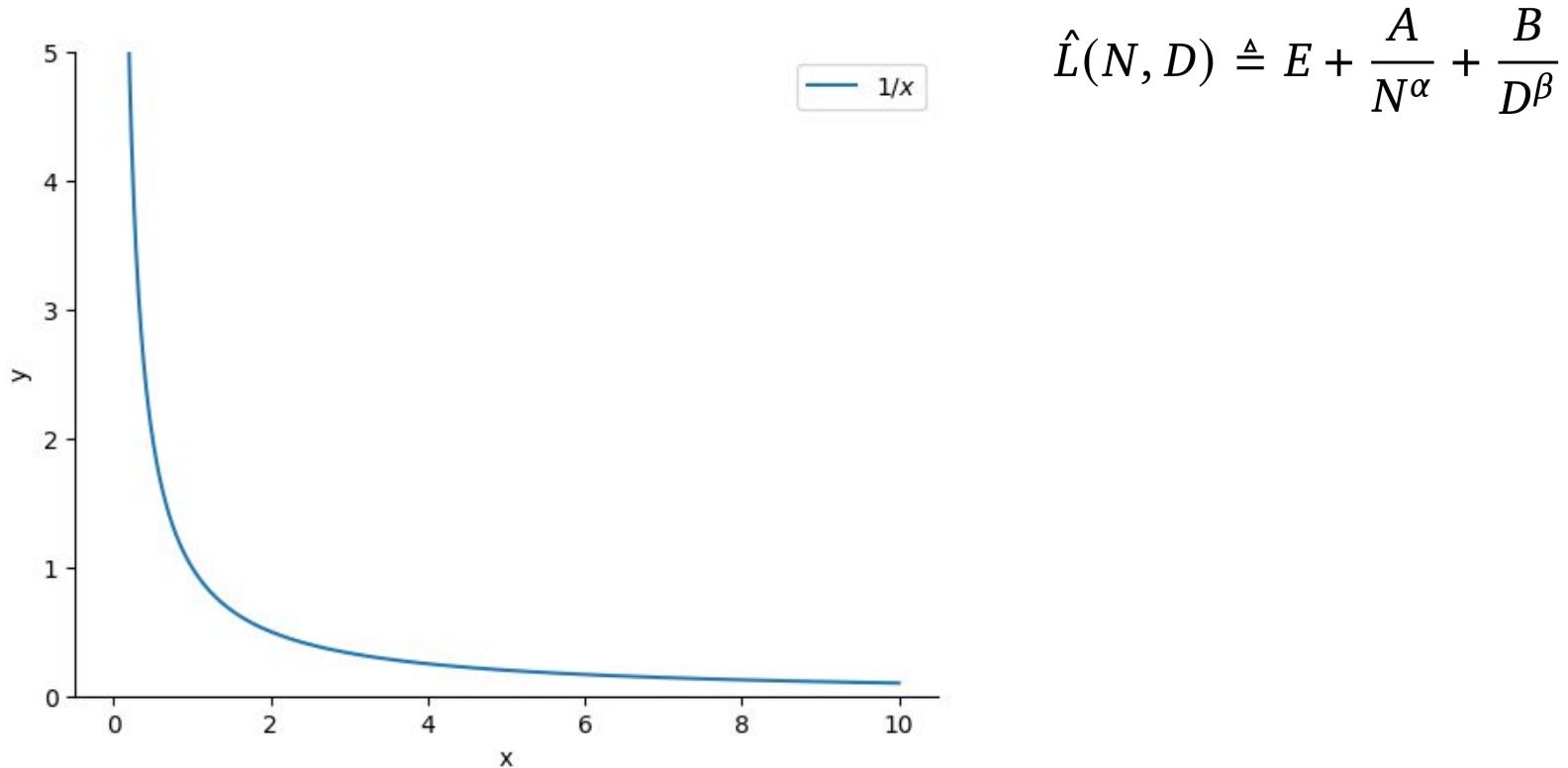
$$\hat{L}(N, D) \triangleq [E] + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

Perfectly trained transformer with N parameters underperforms the ideal setting

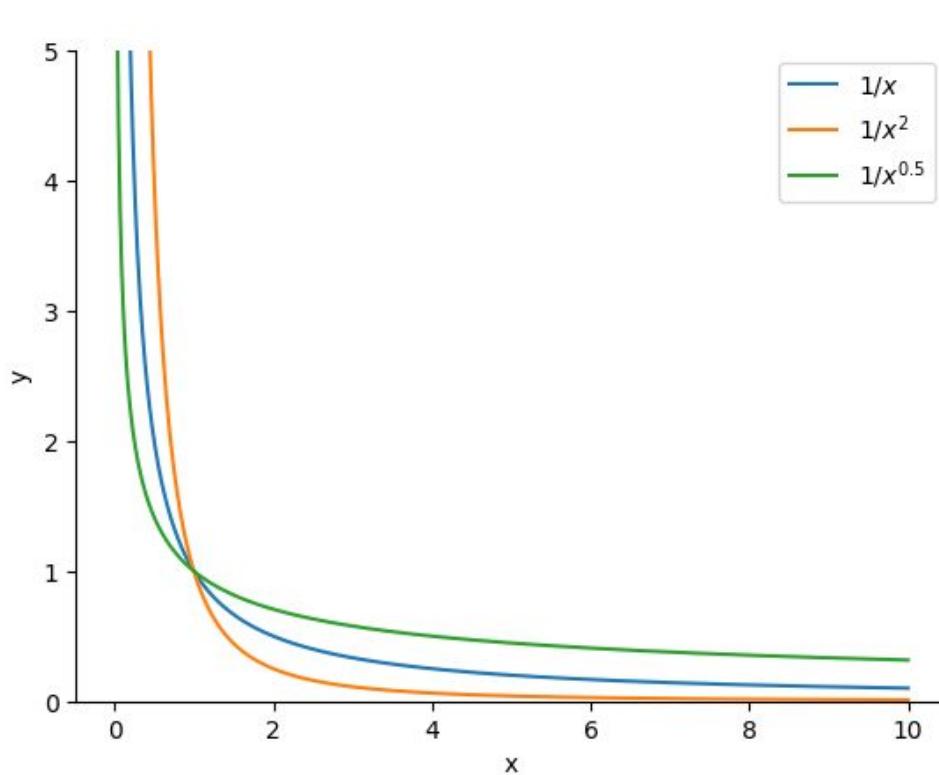
Transformer is not trained to convergence, as we only make a finite number of optimisation steps, on a sample of the dataset distribution



# Scaling Laws from Classical Risk Decomposition



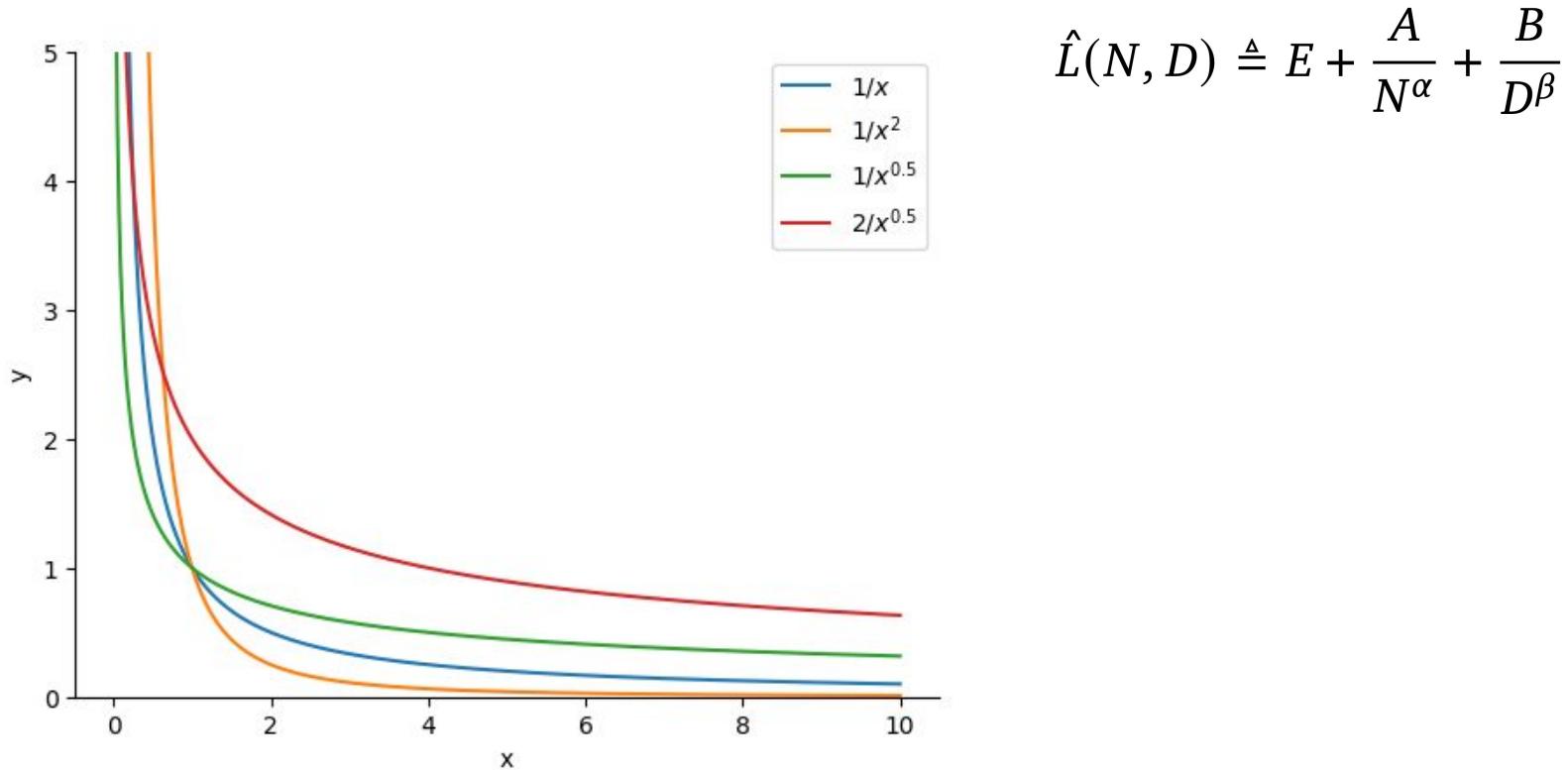
# Scaling Laws from Classical Risk Decomposition



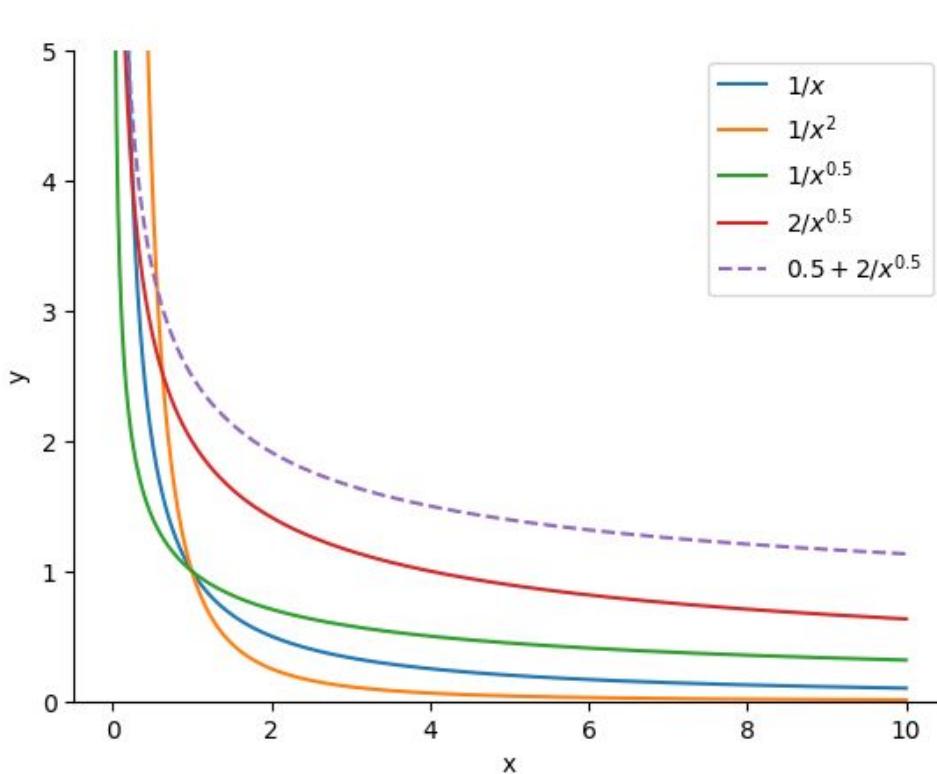
$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$



# Scaling Laws from Classical Risk Decomposition



# Scaling Laws from Classical Risk Decomposition



$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$



# Scaling Laws from Classical Risk Decomposition

Loss for an ideal generative process (entropy of natural text)

$$\hat{L}(N, D) \triangleq E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}$$

Perfectly trained transformer with  $N$  parameters underperforms the ideal setting

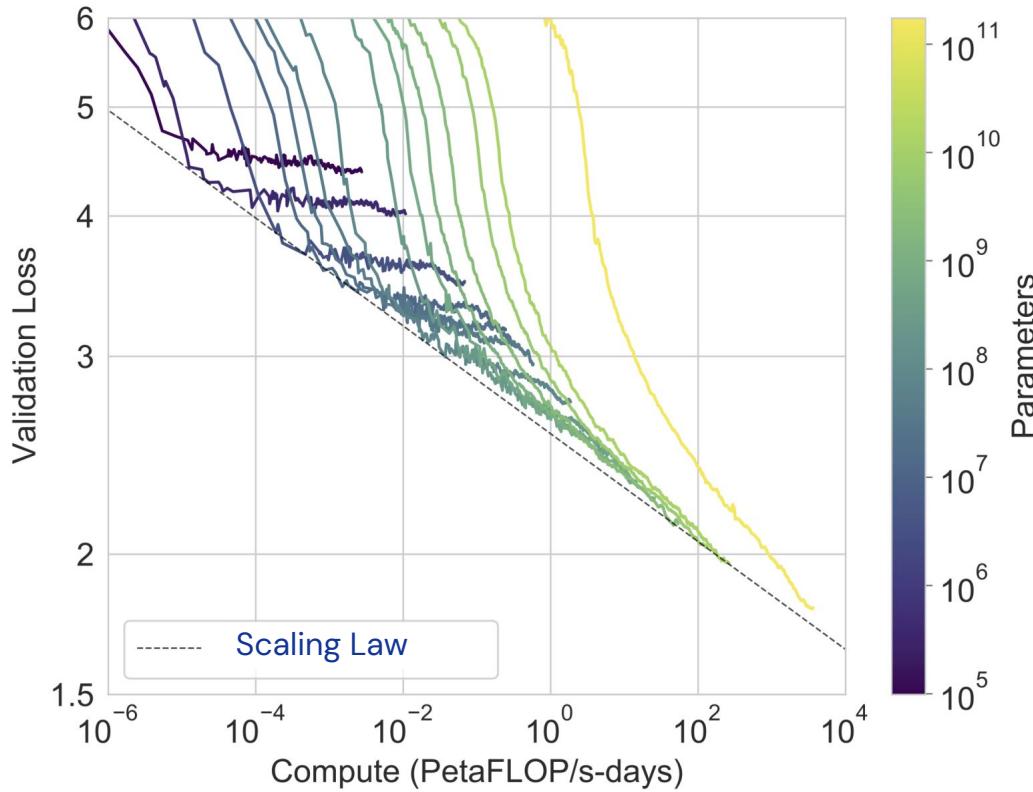
Transformer is not trained to convergence, as we only make a finite number of optimisation steps, on a sample of the dataset distribution

Parameters are learned from existing training curves:

$$\min_{A,B,E,\alpha,\beta} \sum_{\text{Runs } i} \text{Huber}_\delta \left( \log \hat{L}(N_i, D_i) - \log L_i \right)$$



# Can we predict the expected improvement?



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- **Parameter Efficient Fine-Tuning & Quantization**
- Capabilities & Limitations



## Using these models in practice:

```
from transformers import AutoModelForCausalLM  
  
if 'mixtral' in model_type:  
    model = AutoModelForCausalLM.from_pretrained(  
        'mistralai/Mixtral-8x7B-v0.1',  
        device_map=device_map,  
        cache_dir=CACHE_PATH,  
    )
```



## Using these models in practice:

```
from transformers import AutoModelForCausalLM  
  
if 'mixtral' in model_type:  
    model = AutoModelForCausalLM.from_pretrained(  
        'mistralai/Mixtral-8x7B-v0.1',  
        device_map=device_map,  
        cache_dir=CACHE_PATH,  
    )
```



```
RuntimeError: CUDA out of memory. Tried to allocate 200.00 MiB (GPU 0; 15.78 GiB total  
capacity; 14.56 GiB already allocated; 38.44 MiB free; 14.80 GiB reserved in total by  
PyTorch) If reserved memory is >> allocated memory try setting max_split_size_mb to avoid  
fragmentation. See documentation for Memory Management and PYTORCH_CUDA_ALLOC_CONF
```



## (Post-training) Quantization

$$\operatorname{argmin}_{\widehat{\mathbf{W}}} \|\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}\|_2^2$$

quantized + low-precision matrix multiplication



## (Post-training) Quantization

$$\operatorname{argmin}_{\widehat{\mathbf{W}}} \|\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}\|_2^2$$

quantized + low-precision matrix multiplication

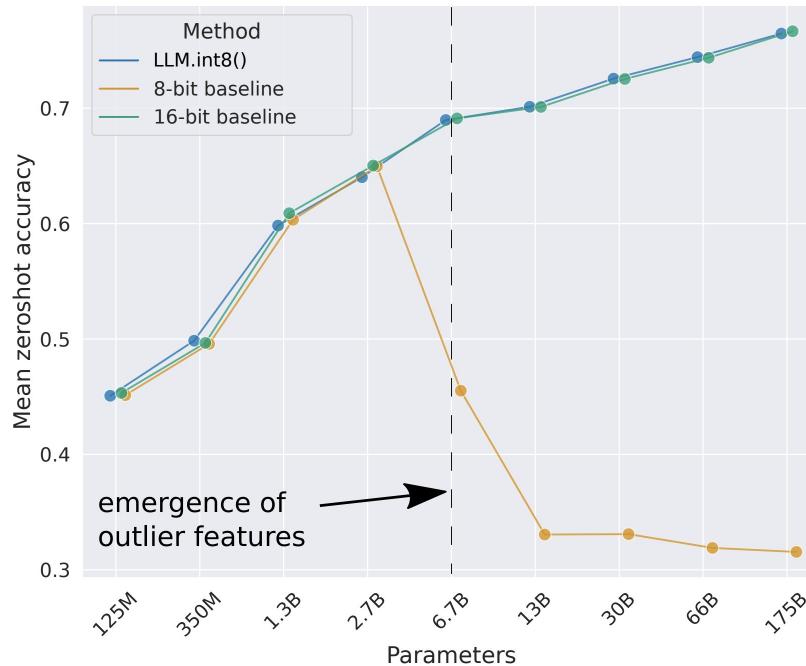
Specifically, GPTQ can quantize GPT models with 175 billion parameters in approximately four GPU hours, reducing the bitwidth down to 3 or 4 bits per weight, with negligible accuracy degradation relative to the uncompressed baseline. Our method more than doubles the compression gains relative to previously-proposed one-shot quantization methods, preserving accuracy, allowing us for the first time to execute an 175 billion-parameter model inside a single GPU for generative inference.



# (Post-training) Quantization

$$\operatorname{argmin}_{\widehat{\mathbf{W}}} \|\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}\|_2^2$$

quantized + low-precision matrix multiplication



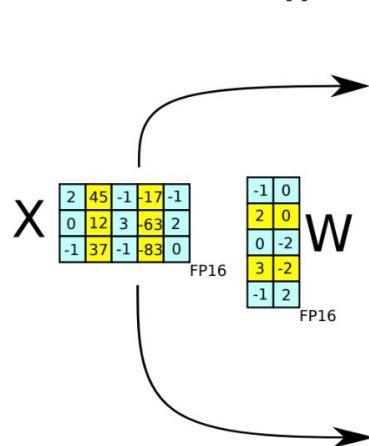
128

# (Post-training) Quantization

$$\operatorname{argmin}_{\widehat{\mathbf{W}}} \|\mathbf{W}\mathbf{X} - \widehat{\mathbf{W}}\mathbf{X}\|_2^2$$

quantized + low-precision matrix multiplication

`LLM.int8()`



8-bit Vector-wise Quantization

(1) Find vector-wise constants:  $C_W$  &  $C_X$

$X$	$1 \ 2 \leftarrow C_W$
$\begin{matrix} 2 & -1 & -1 \\ 0 & 3 & 2 \\ -1 & -1 & 0 \end{matrix}$ FP16	$\begin{matrix} -1 & 0 \\ 0 & -2 \\ -1 & 2 \end{matrix}$ FP16

$C_X$

(2) Quantize

$$X_{\text{FP16}} * (127/C_X) = X_{\text{I8}}$$

$$W_{\text{FP16}} * (127/C_W) = W_{\text{I8}}$$

(3) Int8 Matmul

$$X_{\text{I8}} \ W_{\text{I8}} = \text{Out}_{\text{I32}}$$

(4) Dequantize

$$\frac{\text{Out}_{\text{I32}} * (C_X \otimes C_W)}{127*127} = \text{Out}_{\text{FP16}}$$

16-bit Decomposition

(1) Decompose outliers

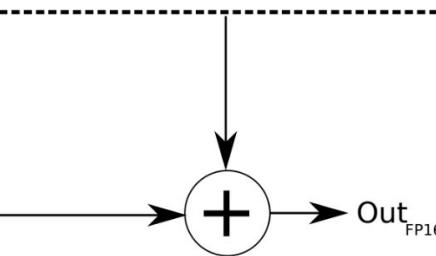
$X$	$45 \ 17$
$12 \ 63$	$2 \ 0$
$37 \ 83$	$3 \ -2$

FP16

(2) FP16 Matmul

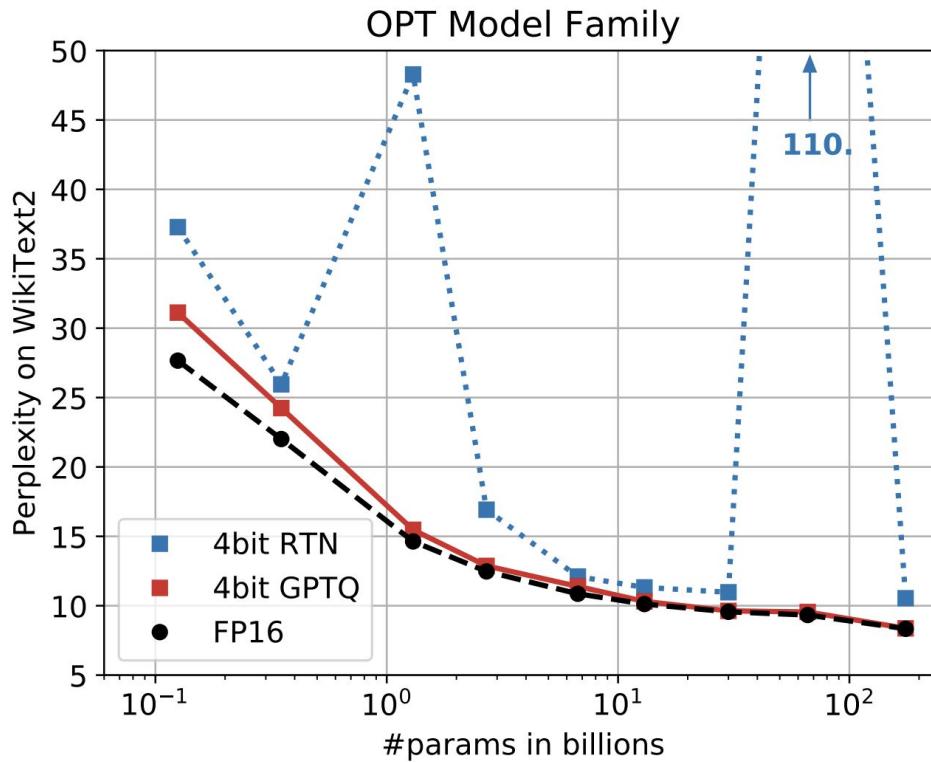
$$X_{\text{FP16}} \ W_{\text{FP16}} = \text{Out}_{\text{FP16}}$$

Legend:  
█ Regular values  
█ Outliers



# (Post-training) Quantization

$$\operatorname{argmin}_{\widehat{\mathbf{W}}} \|\mathbf{WX} - \widehat{\mathbf{W}}\mathbf{X}\|_2^2$$



quantized + low-precision matrix multiplication



# Parameter-Efficient Fine-Tuning (PEFT)

Common Workflow:

- (1) Download state-of-the-art pre-trained LLM trained on internet text for general "world knowledge" and reasoning abilities
- (2) Test in-context learning abilities for sufficient few-shot performance.



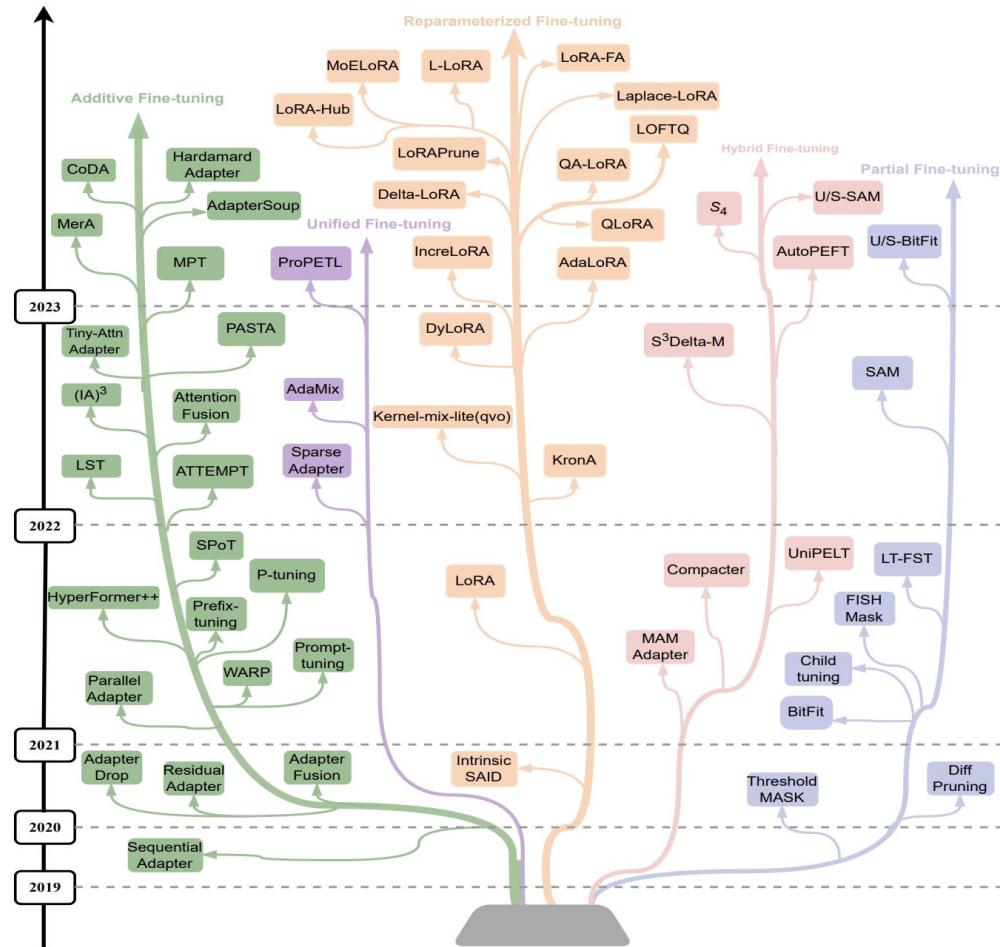
# Parameter-Efficient Fine-Tuning (PEFT)

Common Workflow:

- (1) Download state-of-the-art pre-trained LLM trained on internet text for general "world knowledge" and reasoning abilities
- (2) Test in-context learning abilities for sufficient few-shot performance.
- (3) Use a parameter-efficient fine-tuning scheme to update a subset of parameters (~1%).



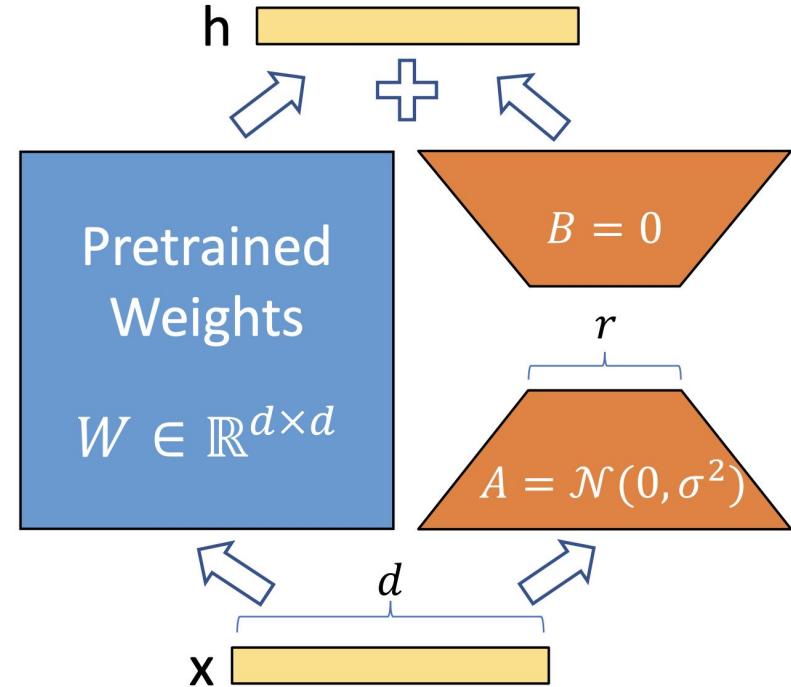
# PEFT



# Low-Rank Adaptation (LoRA)

$$h = W_0x + \Delta Wx = W_0x + BAx$$

Zero at initialization

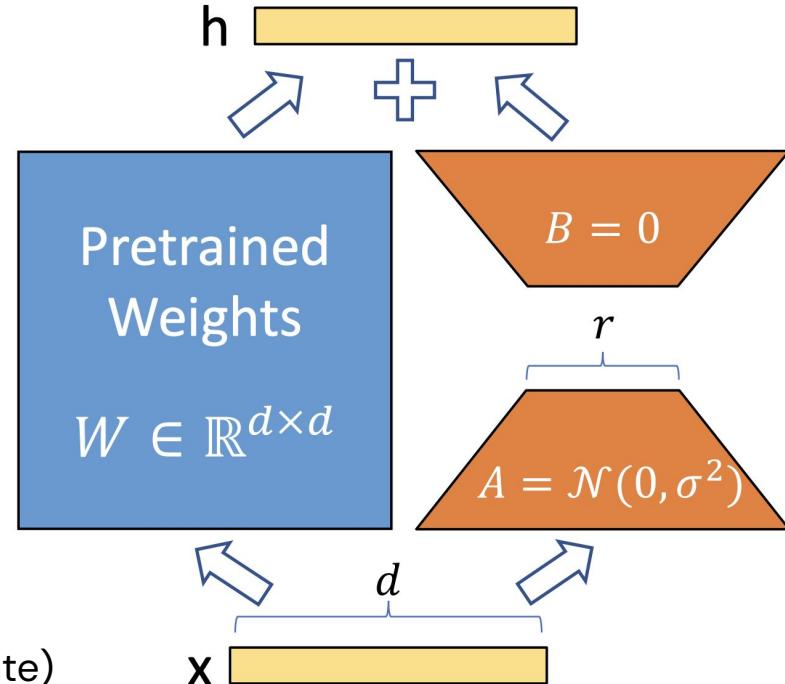


# Low-Rank Adaptation (LoRA)

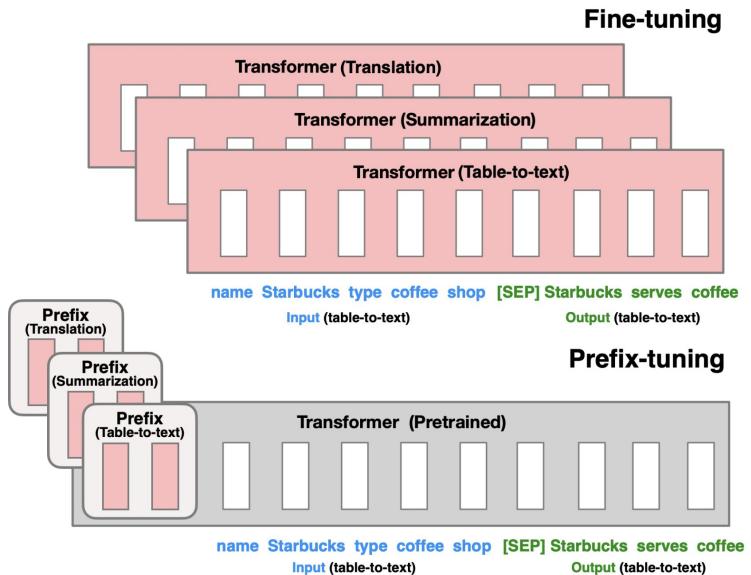
$$h = W_0x + \Delta Wx = W_0x + BAx$$

Zero at initialization

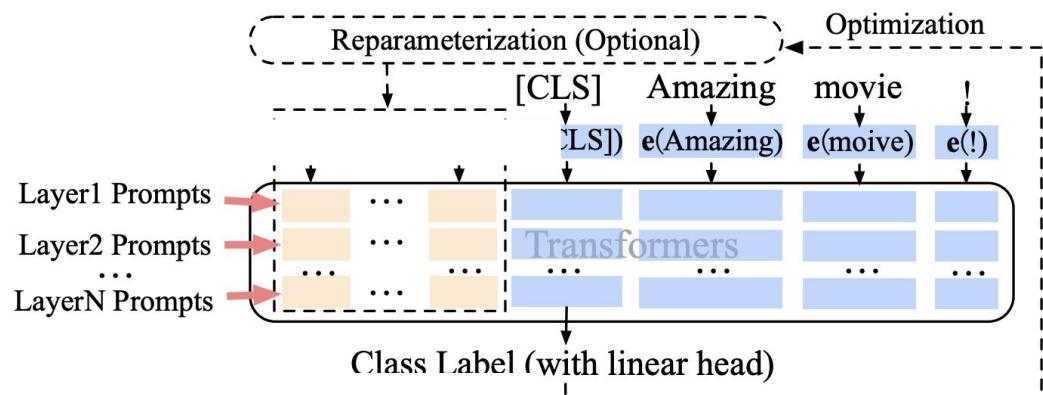
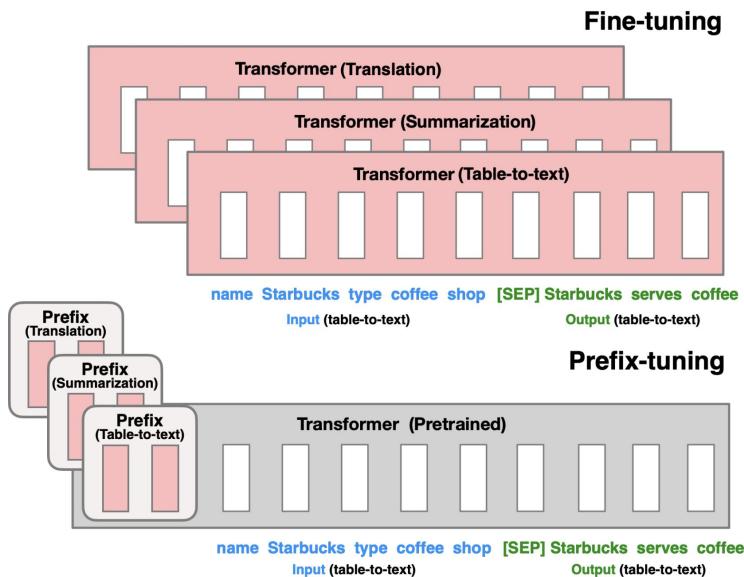
- Can easily train and share different LoRA modules for various tasks, only need to store  $B, A$
- No changes to inference speed
- Efficient Training (No need to calculate and store gradients for full model, no need to store optimizer state)



# Prefix Tuning v1 & v2



# Prefix Tuning v1 & v2



# Implementing Prefix Tuning v2

```
batch_size = input_ids.shape[0]
past_key_values = self.get_prompt(batch_size=batch_size)
prefix_attention_mask = torch.ones(batch_size, self.pre_seq_len).to(self.bert.device)
attention_mask = torch.cat((prefix_attention_mask, attention_mask), dim=1)

outputs = self.bert(           ← Call to a popular LLM (Devlin et al., 2018)
    input_ids,
    attention_mask=attention_mask,
    token_type_ids=token_type_ids,
    position_ids=position_ids,
    head_mask=head_mask,
    inputs_embeds=inputs_embeds,
    output_attentions=output_attentions,
    output_hidden_states=output_hidden_states,
    return_dict=return_dict,
    past_key_values=past_key_values,
)
```



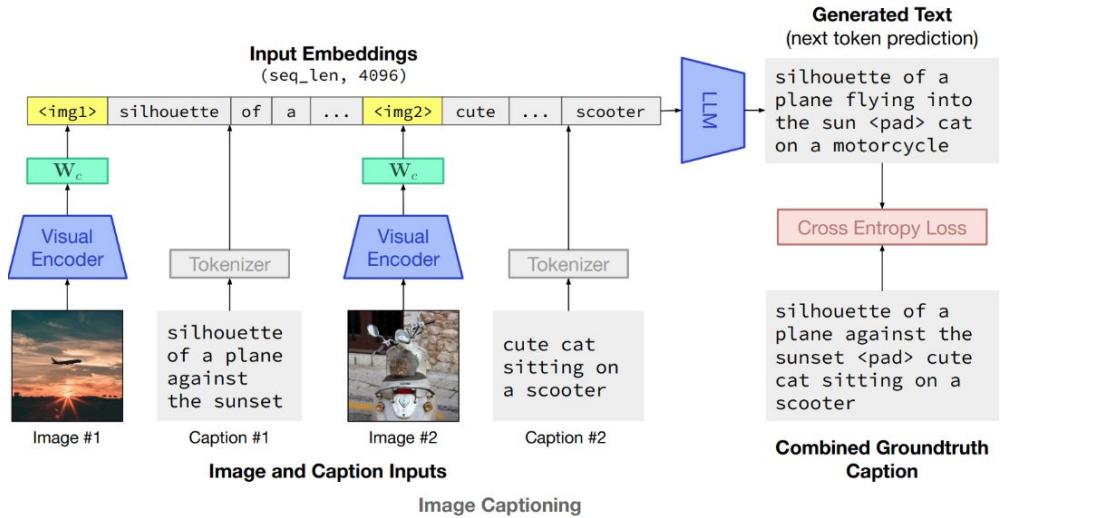
# Implementing Prefix Tuning v2

```
batch_size = input_ids.shape[0]
past_key_values = self.get_prompt(batch_size=batch_size)
prefix_attention_mask = torch.ones(batch_size, self.pre_seq_len).to(self.bert.device)
attention_mask = torch.cat((prefix_attention_mask, attention_mask), dim=1)

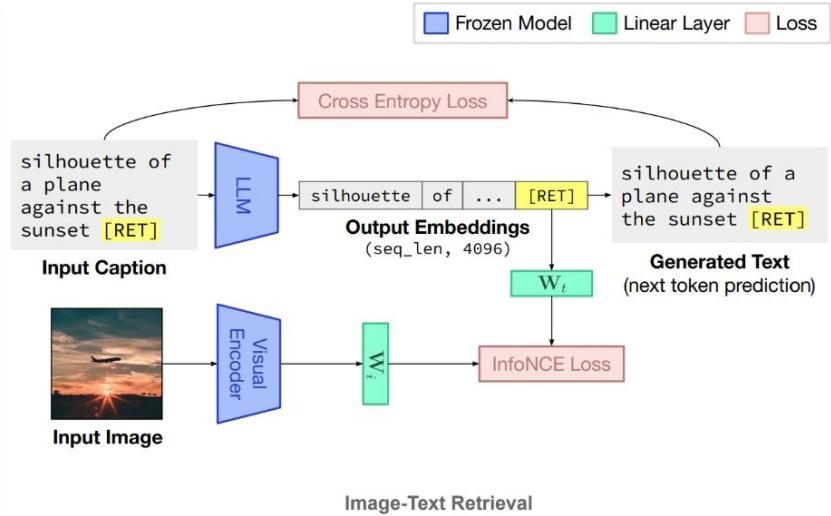
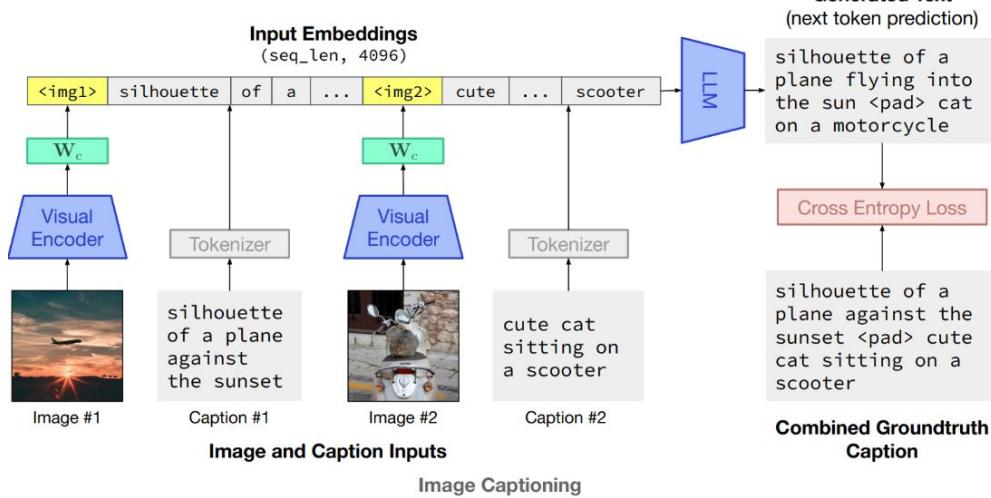
outputs = self.bert(
    input_ids,
    attention_mask=attention_mask,
    token_type_ids=token_type_ids,
    position_ids=position_ids,
    head_mask=head_mask,
    inputs_embeds=inputs_embeds,
    output_attentions=output_attentions,
    output_hidden_states=output_hidden_states,
    return_dict=return_dict,
    past_key_values=past_key_values,
)
```



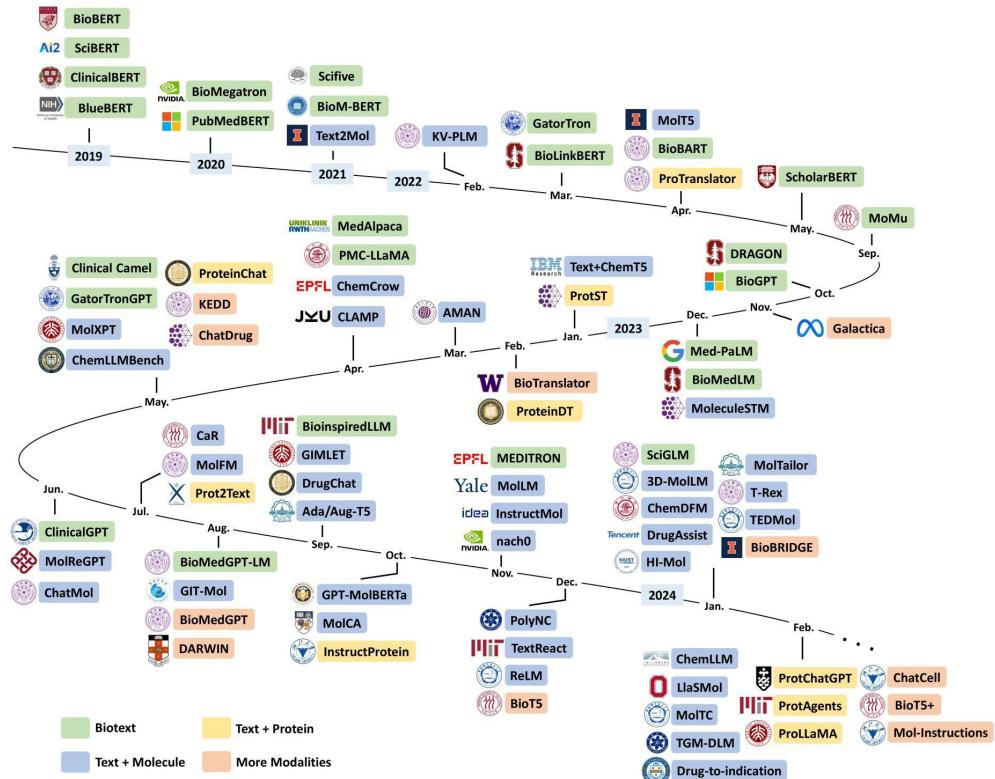
# Prefix Tuning for Multi-Modal models



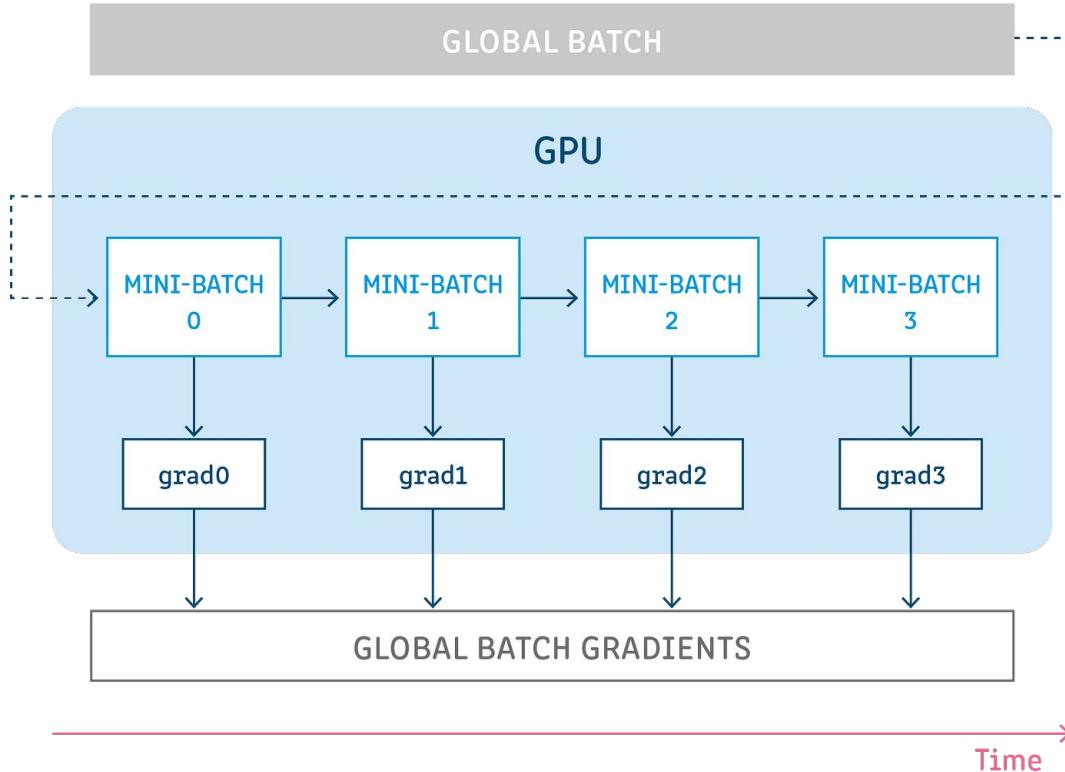
# Prefix Tuning for Multi-Modal models



# Prefix Tuning for Multi-Modal models



# Other Tricks: Gradient accumulation



# Other Tricks: Gradient checkpointing

Assume 1x 16GB GPU and a Pretrained model of size 1.5 GB

Gradient Checkpoint off

1.5GB Neurons \* Layers \* Batch Size

Static

Dynamic (Activations)

Mem Usage



$O(n)$

All Intermediate Activations stored



# Other Tricks: Gradient checkpointing

Assume 1x 16GB GPU and a Pretrained model of size 1.5 GB

Gradient Checkpoint off

1.5GB Neurons \* Layers \* Batch Size



Mem Usage



$O(n)$

Gradient Checkpoint On

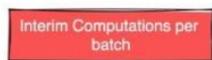
1.5GB Neurons \* Layers \* Batch Size



$O(\sqrt{n})$

Total GPU Memory - 16GB

Legend



# Other Tricks: Gradient checkpointing

```
# Set training parameters
training_args = TrainingArguments(
    output_dir=ckpt_path,
    num_train_epochs=FLAGS.n_epochs,
    max_steps=FLAGS.n_max_steps,
    per_device_train_batch_size=FLAGS.batch_size,
    per_device_eval_batch_size=FLAGS.eval_batch_size,
    # Optimization settings
    learning_rate=FLAGS.learning_rate,
    weight_decay=0.01,
    max_grad_norm=0.3,
    lr_scheduler_type='cosine',
    warmup_ratio=0.03,
    # Logging & Validation settings
    logging_steps=25,
    evaluation_strategy="steps",
    eval_steps=100,
    save_strategy="steps",
    save_steps=100,
    save_total_limit=2,
    metric_for_best_model=FLAGS.best_model_metric,
    greater_is_better=False if 'eval_loss' == FLAGS.best_model_metric else True,
    load_best_model_at_end=False,
    # Efficiency settings
    fp16=False,
    bf16=use_bf16,
    gradient_checkpointing=False,
    gradient_accumulation_steps=1,
    group_by_length=False,
    optim=optim,
    report_to="wandb" if FLAGS.wandb_track else "none",
    run_name='{}_finetune {}'.format(FLAGS.dataset, FLAGS.model),
)
```



# Other Tricks: Gradient checkpointing

```
# Set training parameters
training_args = TrainingArguments(
    output_dir=ckpt_path,
    num_train_epochs=FLAGS.n_epochs,
    max_steps=FLAGS.n_max_steps,
    per_device_train_batch_size=FLAGS.batch_size,
    per_device_eval_batch_size=FLAGS.eval_batch_size,
    # Optimization settings
    learning_rate=FLAGS.learning_rate,
    weight_decay=0.01,
    max_grad_norm=0.3,
    lr_scheduler_type='cosine',
    warmup_ratio=0.03,
    # Logging & Validation settings
    logging_steps=25,
    evaluation_strategy="steps",
    eval_steps=100,
    save_strategy="steps",
    save_steps=100,
    save_total_limit=2,
    metric_for_best_model=FLAGS.best_model_metric,
    greater_is_better=False if 'eval_loss' == FLAGS.best_model_metric else True,
    load_best_model_at_end=False,
    # Efficiency settings
    fp16=False,
    bf16=use_bf16,
    gradient_checkpointing=False,
    gradient_accumulation_steps=1,
    group_by_length=False,
    optim=optim,
    report_to="wandb" if FLAGS.wandb_track else "none",
    run_name='{}_finetune {}'.format(FLAGS.dataset, FLAGS.model),
)
```

Think of both as a  
memory/speed trade-off



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

## Part 2:

- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- **Capabilities & Limitations**



## Sparks of Artificial General Intelligence: Early experiments with GPT-4

Sébastien Bubeck Varun Chandrasekaran Ronen Eldan Johannes Gehrke  
Eric Horvitz Ece Kamar Peter Lee Yin Tat Lee Yuanzhi Li Scott Lundberg  
Harsha Nori Hamid Palangi Marco Tulio Ribeiro Yi Zhang

Microsoft Research

3.12712v5 [cs.CL] 13 Apr 2023

### Abstract

Artificial intelligence (AI) researchers have been developing and refining large language models (LLMs) that exhibit remarkable capabilities across a variety of domains and tasks, challenging our understanding of learning and cognition. The latest model developed by OpenAI, GPT-4 [Ope23], was trained using an unprecedented scale of compute and data. In this paper, we report on our investigation of an early version of GPT-4, when it was still in active development by OpenAI. We contend that (this early version of) GPT-4 is part of a new cohort of LLMs (along with ChatGPT and Google's PaLM for example) that exhibit more general intelligence than previous AI models. We discuss the rising capabilities and implications of these models. We demonstrate that, beyond its mastery of language, GPT-4 can solve novel and difficult tasks that span mathematics, coding, vision, medicine, law, psychology and more, without needing any special prompting. Moreover, in all of these tasks, GPT-4's performance is strikingly close to human-level performance, and often vastly surpasses prior models such as ChatGPT. Given the breadth and depth of GPT-4's capabilities, we believe that it could reasonably be viewed as an early (yet still incomplete) version of an artificial general intelligence (AGI) system. In our exploration of GPT-4, we put special emphasis on discovering its limitations, and we discuss the challenges ahead for advancing towards deeper and more comprehensive versions of AGI, including the possible need for pursuing a new paradigm that moves beyond next-word prediction. We conclude with reflections on societal influences of the recent technological leap and future research directions.



# Sparks of AGI

**Prompt:** Can you write a proof that there are infinitely many primes, with every line that rhymes?

**GPT-4:**

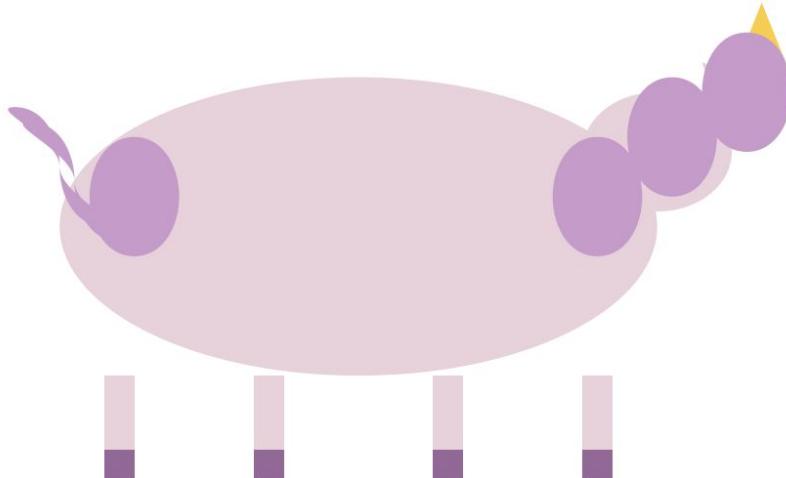
Yes, I think I can,  
Though it might take a clever plan.  
I'll start by noting Euclid's proof,  
Which shows that primes aren't just aloof.  
Assume we have a finite list,  
Of primes, and that none have been missed.  
Multiply them all together,  
And add one, just to be clever.  
The result will be a number,  
That has no prime factors, oh wonder!  
But every number has a prime,  
That divides it, it's just a matter of time.  
So we've found a contradiction,  
And our finite list needs eviction.  
There must be infinitely many primes,  
And that's the end of my rhyming lines.s



# Sparks of AGI

**Prompt:** Draw a unicorn in TiKZ.

**GPT-4:** [Produces L<sup>A</sup>T<sub>E</sub>X compiling to following picture.]

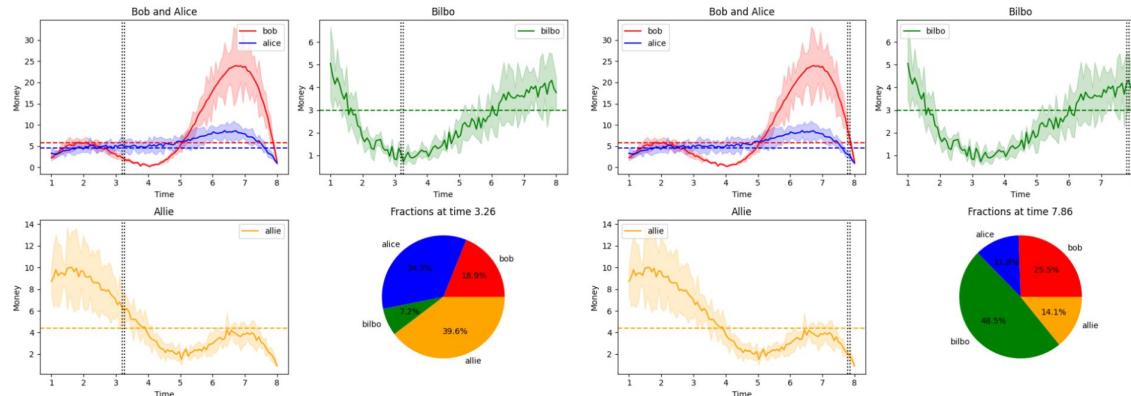


# Sparks of AGI

**Prompt:** Can you generate a pyplot for the following data:  $x = [1, 3, 5, 6, 8]$ ,  $y1 = [2, 3, 5, 18, 1]$ ,  $y2 = [3, 5, 6, 8, 1]$ ,  $y3 = [5, 1, 2, 3, 4]$ ,  $y4 = [9, 7, 2, 3, 1]$ . I want  $y1, y2$  to be in the same plot, but  $y3$  is in another plot next to that plot,  $y4$  is in below. I want the legend of  $y1$  to be "bob",  $y2$  to be "alice",  $y3$  to be "bilbo",  $y4$  to be "allie". I want the  $x$ -axis to be labeled with "time" and  $y$  axis to be labeled with "money". I want to add a 10 %-40% random error bar to each curve, through all times (including non-integers). I want smoothed curves to show the plot, and smoothed error bar. Do not use linear interpolation, use smooth interpolation! I want to also add some small zig-zag to the smoothed curve to make it look more real. I want to put a baseline as the mean of each line. I want to put a pie chart below indicating the fraction of the four people in each time step. I also want an animation to show how the fractions are changing in the pie chart in \*\*continuous time\*\*. Interpolate the missing fractions! I also want vertical line animation in other three plots to match the pie chart. I want the fanciest plot. Please add as many fancy things as possible.

Here is a possible code to generate a pyplot for the following data, with some comments to explain the steps:

**GPT-4:** [We give two snapshots from the animation resulting from the produced code]

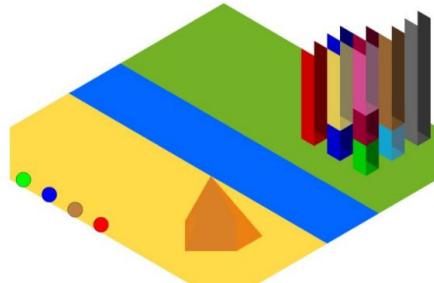


# Sparks of AGI

## GPT-4 and Stable Diffusion

**Prompt:** A screenshot of a city-building game in 3D. The screenshot is showing a terrain where there is a river from left to right, there is a desert with a pyramid below the river, and a city with many highrises above the river. The bottom of the screen has 4 buttons with the color green, blue, brown, and red respectively.

**GPT-4:**



**Stable Diffusion v2.1:**



**Stable Diffusion v2.1  
with GPT-4's sketch:**



# Sparks of AGI

Can you compose a short tune (say four to eight bars) using ABC notation?

There are many possible ways to compose a short tune in abc notation, but here is one example:

```
X:1
T:Example Tune
M:4/4
L:1/8
K:C
|:G2AB c2BA | G2AB c2c2 | edcB c2BA | G4-G2z2 :|
|:c2de f2ed | c2de f2f2 | g2fe d2cB | c4-c2z2 :|
```

This tune has two four-bar phrases, each repeated once, forming an A part and a B part. The key is C major.

Can you describe the tune in musical terms?

Here is one possible description:

The tune starts with a rising arpeggio of the tonic chord (C major), followed by a descending scale that returns to the tonic. This pattern is repeated, but the second time the scale continues to the higher tonic note. The first phrase ends with a long tonic note.



# Sparks of AGI

## Prompt:

A function  $f$  is called {increasing} \_{emphasize!} if  $f(s) \leq f(t)$  holds for every  $s \leq t$ . Consider the formula:  
 $\lim_{x \rightarrow 0} \lim_{n \rightarrow \infty}$  underscore{  
 $\begin{matrix} [ \sum_{i=1}^n (f(nx) - \int_0^x \sin(nx) dx), \\ ((x + 2/8)^2 - 5)/((4x-1/x^2) / (n \text{ xor } 3)), \\ \dots, \text{partial } f(x/n) \\ \text{over } \text{partial } x * (e^{-i\omega n x} + 3(x\#7)), \\ f \text{ inverse of } (x \bmod n) ] \\ \text{multiply } \begin{matrix} [ 1, 1, \dots, 1 ] \\ [ 1, 1, \dots, 1 ] \end{matrix} \\ \text{grey_color{this gives us a good impression about the function}} \end{matrix}$ }

Convert the above paragraph to LaTeX.

## GPT-4:

A function  $f$  is called *increasing* if  $f(s) \leq f(t)$  holds for every  $s \leq t$ . Consider the formula:

$$\lim_{x \rightarrow 0} \lim_{n \rightarrow \infty} \underbrace{\left[ \begin{array}{c} \sum_{i=1}^n (f(nx) - \int_0^x \sin(nx) dx), & \frac{(x+\frac{2}{8})^2-5}{\frac{4x-1/x^2}{n \oplus 3}}, \\ \vdots & \vdots \\ \frac{\partial f(x/n)}{\partial x} \cdot (e^{-i\omega n x} + 3(x\#7)), & f^{-1}(x \bmod n) \end{array} \right]}_{\text{this gives us a good impression about the function}} \times \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$



# Embers of Autoregression

## Embers of Autoregression: Understanding Large Language Models Through the Problem They are Trained to Solve

R. Thomas McCoy      Shunyu Yao      Dan Friedman      Matthew Hardy      Thomas L. Griffiths

Princeton University

### One-sentence summary:

To understand what language models are, we must understand what we have trained them to be.

### Abstract:

The widespread adoption of large language models (LLMs) makes it important to recognize their strengths and limitations. We argue that in order to develop a holistic understanding of these systems we need to consider the problem that they were trained to solve: next-word prediction over Internet text. By recognizing the pressures that this task exerts we can make predictions about the strategies that LLMs will adopt, allowing us to reason about when they will succeed or fail. This approach—which we call the teleological approach—leads us to identify three factors that we hypothesize will influence LLM accuracy: the probability of the task to be performed, the probability of the target output, and the probability of the provided input. We predict that LLMs will achieve higher accuracy when these probabilities are high than when they are low—even in deterministic settings where probability should not matter. To test our predictions, we evaluate two LLMs (GPT-3.5 and GPT-4) on eleven tasks, and we find robust evidence that LLMs are influenced by probability in the ways that we have hypothesized. In many cases, the experiments reveal surprising failure modes. For instance, GPT-4’s accuracy at decoding a simple cipher is 51% when the output is a high-probability word sequence but only 13% when it is low-probability. These results show that AI practitioners should be careful about using LLMs in low-probability situations. More broadly, we conclude that we should not evaluate LLMs as if they are humans but should instead treat them as a distinct type of system—one that has been shaped by its own particular set of pressures.

).13638v1 [cs.CL] 24 Sep 2023





# Embers of Autoregression

Ember of autoregression	Definition	Example
<b>Sensitivity to task frequency</b>	LLMs perform better on tasks that are frequent than ones that are rare, even when the tasks have an equivalent level of complexity.	When asked to translate English sentences into Pig Latin, GPT-4 gets 42% accuracy when using the most common variant of Pig Latin but only 23% accuracy when using a rare variant.
<b>Sensitivity to output probability</b>	LLMs achieve higher accuracy when the correct answer is high-probability text than when it is low-probability text, even when the task is deterministic.	When asked to reverse a sequence of words, GPT-4 gets 97% accuracy when the answer is a high-probability sentence yet 53% accuracy when the output is low probability.
<b>Sensitivity to input probability</b>	Even when the task is deterministic, LLMs sometimes achieve higher accuracy when the input text is high-probability than when it is low-probability, but input probability is less influential than output probability.	When asked to encode sentences in a simple cipher (rot-13), GPT-4 gets 21% accuracy when the input is a high-probability sentence yet 11% accuracy when the input is low probability.



# Embers of Autoregression

## Shift cipher: Task probability

**Common task: Rot-13.** Decode the message by shifting each letter thirteen positions backward in the alphabet.

Input: Jryy, vs gurl qba'g pbzr, fb or vg.

Correct: Well, if they don't come, so be it.

✓ GPT-4: Well, if they don't come, so be it.

**Uncommon task: Rot-2.** Decode the message by shifting each letter two positions backward in the alphabet.

Input: Ygnn, kh vjga fqp'v eqog, uq dg kv.

Correct: Well, if they don't come, so be it.

✗ GPT-4: Well, if there isn't cake, to be it.



# Embers of Autoregression

## Shift cipher: Task probability

**Common task: Rot-13.** Decode the message by shifting each letter thirteen positions backward in the alphabet.

Input: Jryy, vs gurl qba'g pbzr, fb or vg.

Correct: Well, if they don't come, so be it.

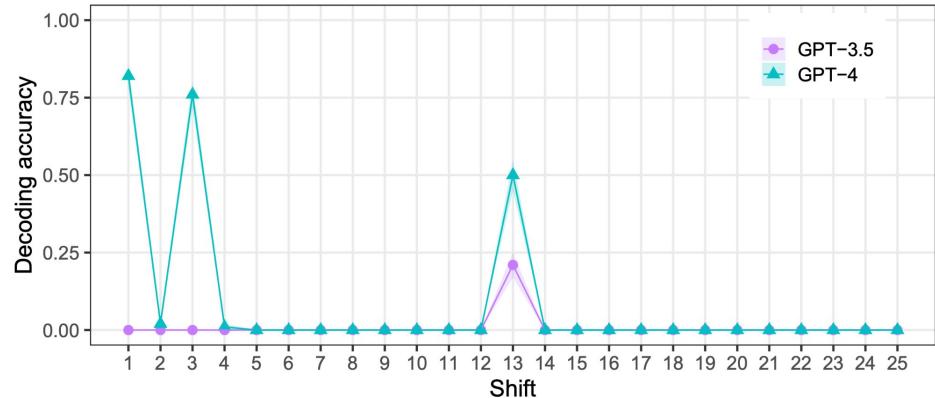
✓ GPT-4: Well, if they don't come, so be it.

**Uncommon task: Rot-2.** Decode the message by shifting each letter two positions backward in the alphabet.

Input: Ygnn, kh vjga fqp'v eqog, uq dg kv.

Correct: Well, if they don't come, so be it.

✗ GPT-4: Well, if there isn't cake, to be it.



# Outline for today

## Part 1:

- N-Gram Language Models
- Transformers

[break]

## Part 2:

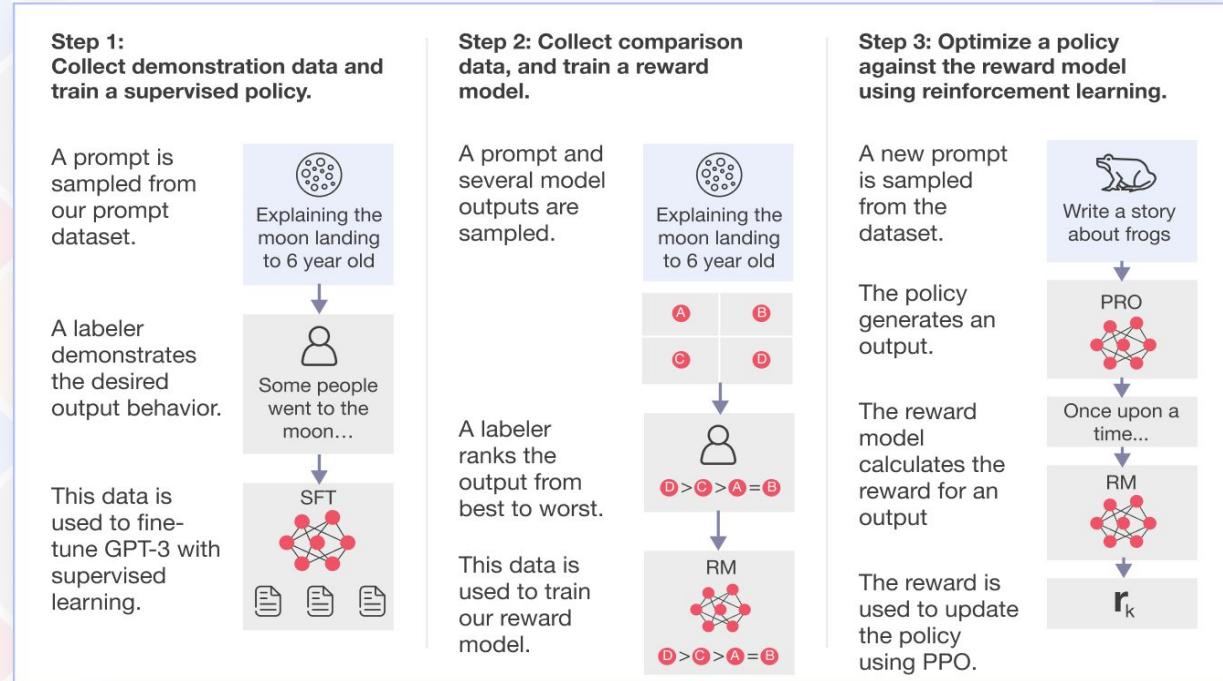
- In-Context Learning & Prompting
- Scaling Laws
- Parameter Efficient Fine-Tuning & Quantization
- Capabilities & Limitations

+ Glossary of new ideas (RLHF, RAG, Instruction Tuning), time permitting



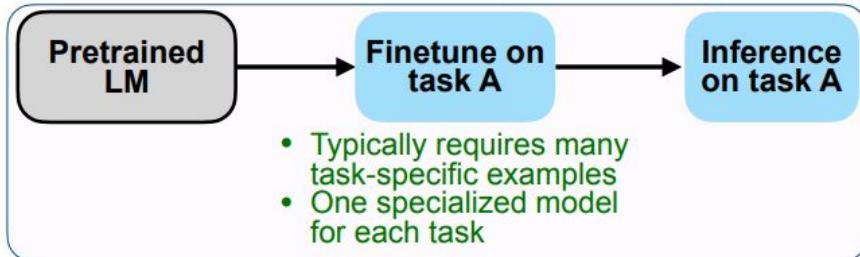
# Bonus: Reinforcement Learning from Human Feedback

## The role of RLHF in ChatGPT

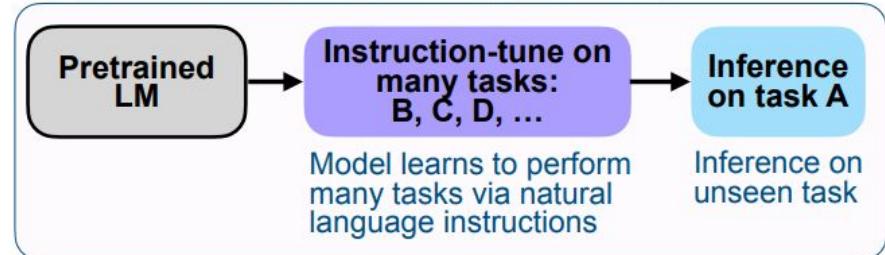


# Bonus: Instruction Tuning

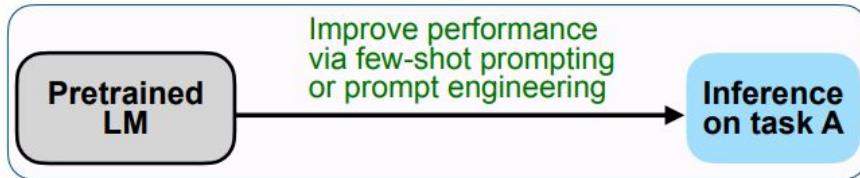
## Parameter-Efficient (or full) Finetuning



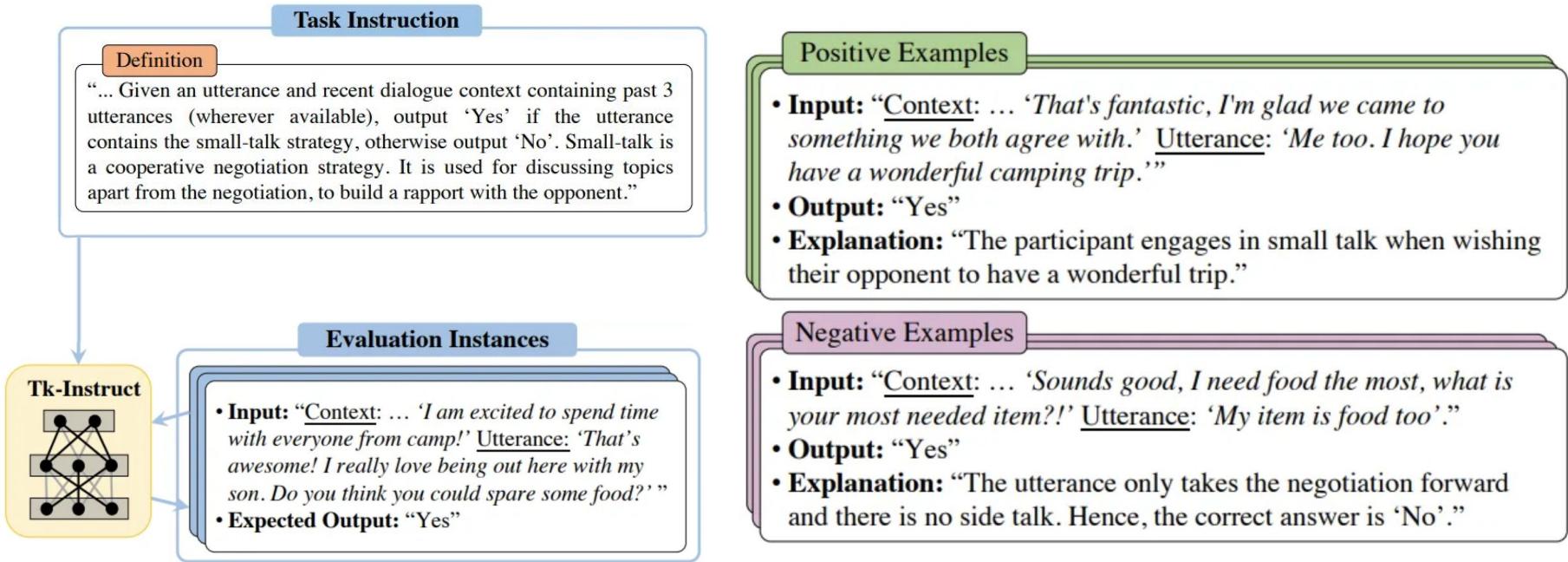
## Instruction Tuning



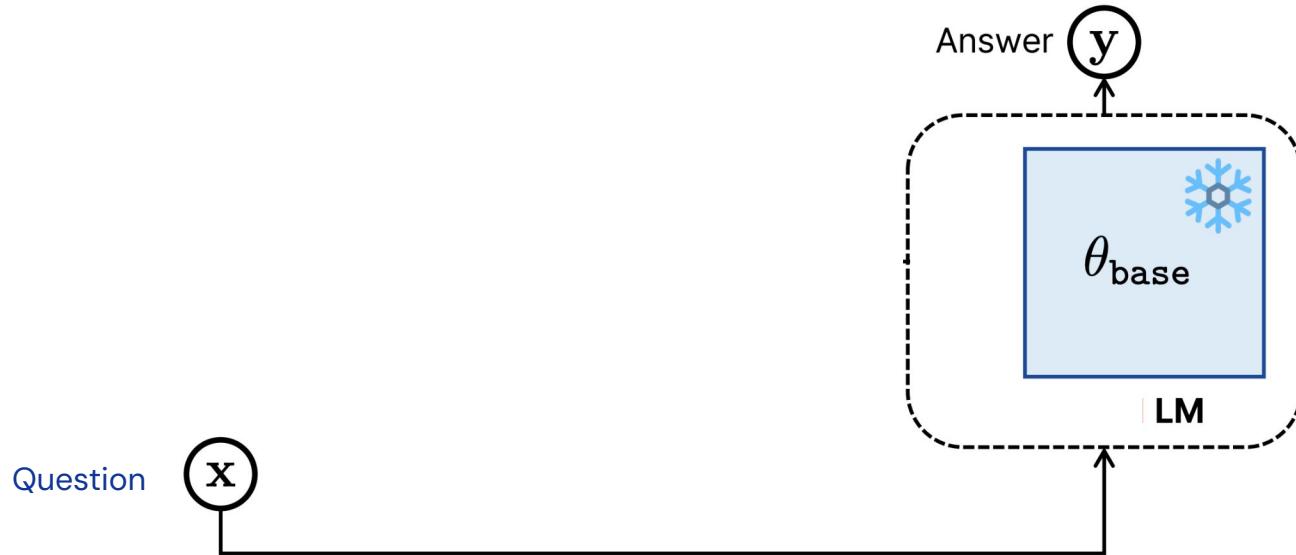
## In-Context Learning / Prompt Engineering



# Bonus: Instruction Tuning



## Bonus: Retrieval Augmentation



## Bonus: Retrieval Augmentation

