# Mathematics of Data: From Theory to Computation

Prof. Volkan Cevher
*volkan.cevher@epfl.ch*

*Lecture 15: Language models*

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

**EE-556** (Fall 2025)

# License Information for Mathematics of Data Slides


- This work is released under a Creative Commons License with the following terms:
- **Attribution**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees must give the original authors credit.
- **Non-Commercial**
  - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees may not use the work for commercial purposes – unless they get the licensor's permission.
- **Share Alike**
  - The licensor permits others to distribute derivative works only under a license identical to the one that governs the licensor's work.
- Full Text of the License

# The era of ChatGPT

Y

what can I learn from the Mathematics of data lecture

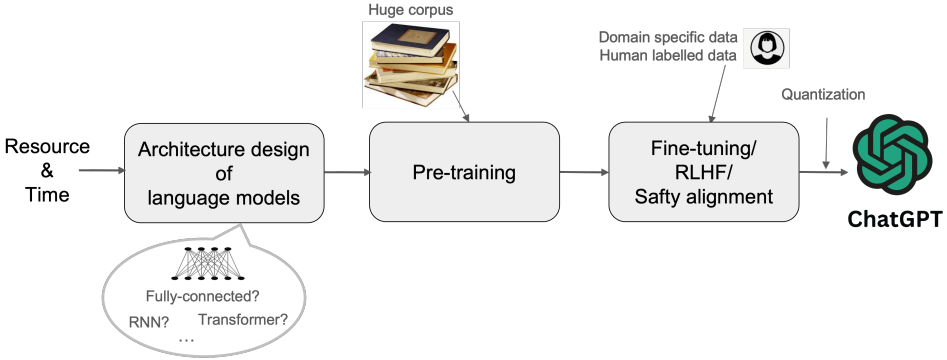
A "Mathematics of Data" lecture can offer you valuable knowledge and skills related to the mathematical foundations of data analysis and data science. Here are some key things you can learn from such a lecture:

1. **Statistical Concepts**: You can learn about statistical techniques and concepts, including descriptive statistics, probability distributions, hypothesis testing, and regression analysis. This knowledge is crucial for understanding and interpreting data.
2. **Linear Algebra**: You may gain insights into linear algebra, which is fundamental for tasks like data transformation, dimensionality reduction, and working with matrices and vectors commonly used in machine learning.
3. **Calculus**: Depending on the lecture's depth, you might learn about calculus and how it relates to optimization algorithms, gradient descent, and mathematical modeling of data phenomena.
4. **Probability Theory**: Understanding probability theory can help you model uncertainty and make informed decisions based on data. Concepts like Bayes' theorem and conditional probability are often covered.
5. **Data Visualization**: Many data-related lectures include topics on data visualization, teaching you how to create meaningful graphs and charts to convey insights from data effectively.
6. **Data Modeling**: You can learn about various data modeling techniques, including linear models, logistic regression, and decision trees. These models are used for prediction and classification tasks.
7. **Machine Learning Foundations**: If the lecture is more advanced, it may introduce yo
machine learning concepts such as supervised learning, unsupervised learning, and

Regenerate

Send a message

# Outline




- This lecture
  1. Basics of language models
  2. Self-attention and transformer architectures
  3. Fundamentals of pre-training & fine-tuning & reinforcement learning with human feedback (RLHF)
  4. Generative pre-trained transformer (GPT) family
  5. Basics of prompting: zero-shot & few-shot & chain-of-thought & automatic

# A motivation for language models (LMs)

**Example**

Predict the next word $w$ given the following source sentence $S_{\text{source}}$?

$S_{\text{source}}$ : *"On January 1 people usually say happy new [$w$]."*

# A motivation for language models (LMs)

**Example**

Predict the next word $w$ given the following source sentence $S_{\text{source}}$?

$S_{\text{source}}$ : *"On January 1 people usually say happy new [$w$]."*

**Question:**

- Why is this important?
  - spelling & grammar correction $\quad \mathsf{p}(\text{year}|S_{\text{source}}) > \mathsf{p}(\text{years}|S_{\text{source}})$
  - machine translation $\quad \mathsf{p}(S_{\text{translation 1}}|S_{\text{source}}) > \mathsf{p}(S_{\text{translation 2}}|S_{\text{source}})$
  - sentence classification $\quad \mathsf{p}(S_{\text{class 1}}|S_{\text{source}}) > \mathsf{p}(S_{\text{class 2}}|S_{\text{source}})$
  - speech recognition $\quad \mathsf{p}(w|S_{\text{source}})$
  - chatbot $\quad \mathsf{p}(w|S_{\text{source}})$
  - (more generally) labeling, automated decisions,...

# Basics for language models (LMs) – I

## Definition (Language model [9])

Models that assign probabilities to sequences of words are called language models.

**Remarks:**

- Given a sentence with $T$ words: $S = w_{1:T} = (w_1, \ldots, w_T)$, by the chain rule of probability:

$$\mathsf{p}(S) = \mathsf{p}(w_{1:T}) = \mathsf{p}(w_1)\mathsf{p}(w_2|w_1)\mathsf{p}(w_3|w_{1:2}) \cdots \mathsf{p}(w_T|w_{1:T-1}) = \prod_{t=1}^{T} \mathsf{p}(w_t|w_{1:t-1})$$

- Implicitly, we are enforcing a graphical model that takes "time" into account.

## Example

If $S = w_{1:3} =$ "happy new year", then $\mathsf{p}(S) = \mathsf{p}(\text{happy})\mathsf{p}(\text{new}|\text{happy})\mathsf{p}(\text{year}|\text{happy new})$.

# Basics for language models (LMs) – II

**Question:**

- How can we compute $\mathsf{p}(w_t|w_{1:t-1})$?

**Remarks:**

- A trivial solution: Just count the frequency on a large corpus, e.g.,

$$\mathsf{p}(\text{year}|S_{\text{source}}) = \frac{\mathsf{p}(S_{\text{source}} + \text{year})}{\mathsf{p}(S_{\text{source}})} \approx \frac{\#(\text{On January 1 people usually say happy new year})}{\#(\text{On January 1 people usually say happy new})}$$

- But the language is creative, there are several ways to express the same meaning.
- The sentence above might even not appear on the corpus.
- We need better ways to estimate such probabilities!

# $N$-gram LMs

## Markov assumption [12]

The probability of a word only depends on the last $N-1$ words as

$$\mathsf{p}(w_t|w_{1:t-1}) = \mathsf{p}(w_t|w_{t-N:t-1}) \approx \frac{\#(w_{t-N:t})}{\#(w_{t-N:t-1})}.$$

Markov in 1913 [12] used "Markov chains" to predict whether the upcoming letter would be a vowel or a consonant.

## Example

In the bigram LM ($N=2$), we only need to estimate $\mathsf{p}(w_t|w_{t-1}) \approx \frac{\#(w_{t-1:t})}{\#(w_{t-1})}$ to generate text.

$w_t$

| $w_{t-1}$ | i | want | to | eat |
|---|---|---|---|---|
| i | 5 | 827 | 0 | 9 |
| want | 2 | 0 | 608 | 1 |
| to | 2 | 0 | 4 | 686 |
| eat | 0 | 0 | 2 | 0 |

$w_{t-1}$

| i | want | to | eat |
|---|---|---|---|
| 2533 | 927 | 2417 | 746 |

$w_t$

| $w_{t-1}$ | i | want | to | eat |
|---|---|---|---|---|
| i | 0.002 | 0.33 | 0 | 0.0036 |
| want | 0.0022 | 0 | 0.66 | 0.0011 |
| to | 0.00083 | 0 | 0.0017 | 0.28 |
| eat | 0 | 0 | 0.0027 | 0 |

Figure: Count (Left) and probability $\mathsf{p}(w_t|w_{1:t-1})$ (Right) from the Berkeley Restaurant Project corpus of 9332 sentences [9].

# Towards pre-training an $N$-gram LM

- In natural language processing (NLP), we use tokens to represent words coming from a vocabulary $\mathcal{V}$.

**Terminologies:**

- A *token* is the smallest unit that can be assigned a meaning to be processed.
  - In English, a token often corresponds to a word.
  - However, a single token can also encode compound words like *New York*.
  - In Chinese or Japanese, there is no space between words.
  - In these languages, sentence segmentation is required before we tokenize.
- We indicate the beginning and the end of sentences with tokens $\langle \mathrm{BOS} \rangle$ and $\langle \mathrm{EOS} \rangle$.
  - $S_{\text{source}}$ "$\langle \mathrm{BOS} \rangle$ Happy new year $\langle \mathrm{EOS} \rangle$" has $T = 5$ tokens.
- The size of our vocabulary is denoted as $|\mathcal{V}|$.
- *Pre-training*: building a LM based on a large corpus in a (often) self-supervised manner.
- *Inference*: Using a trained LM to do next word prediction.

# $N$-gram LMs: "Pre-training" & Inference

- The following simplified examples show the difficulty of pre-training and inference with 2-gram LMs.

| "Pre-training" |
|---|
| **1.** Count $\#(w_{t-1})$ and $\#(w_{t-1:t})$ over the corpus. |
| **2.** Obtain probability $\mathsf{p}(w_t \mid w_{t-1})$ over the corpus. |

| Inference |
|---|
| **1.** Set $w_1$ as $\langle \mathrm{BOS} \rangle$, $t = 1$. |
| **2.** **While True**: |
| ▸ $w_{t+1} = \arg\max_{w \in \mathcal{V}} \mathsf{p}(w \mid w_t)$ |
| ▸ **If** $w_{t+1}$ is $\langle \mathrm{EOS} \rangle$: **break** |
| ▸ $t = t + 1$ |
| **3.** Output: $[w_1, \cdots, w_{t+1}]$. |

**Remarks:**

- Need to store the probability for all $N$-gram pairs.
- Language is creative, some new $N$-gram pairs might not even appear on the corpus.
- Cannot incorporate earlier words than $N$ due to the Markov assumption.

$$\mathsf{p}(\text{two} \mid \text{one plus one equals}) = \mathsf{p}(\text{two} \mid \text{it is wrong that one plus one equals})?$$

# Word representations

**Question:** ○ How can we numerically represent a word/meaning?

**Remarks:** ○ Osgood et al. 1957 [16] uses 3 numbers to represent a word.

- valence: the pleasantness of the stimulus
- arousal: the intensity of emotion provoked by the stimulus
- dominance: the degree of control exerted by the stimulus

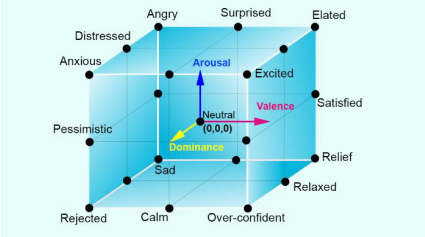
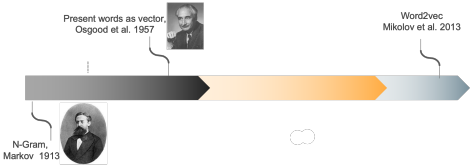



Figure: From [8].

# Word embeddings




**Definition (Word embeddings [9])**

Vectors for representing words are called word embeddings.

- We will briefly introduce two words embeddings:
- One-hot representation: sparse and long word embedding in $\mathbb{R}^{|\mathcal{V}|}$.
  - Training is not required—trivial to obtain.
  - Not a good way to capture the underlying meaning—cannot measure similarity.
- Word2vec [13]: a framework to learn dense and concise word embedding.
  - Training is required.
  - Better characterization for the meaning of a word, e.g., the similarity can be computed by similarity metrics.
  - Cosine similarity or inner products work!

# Word2vec [13]: Setup

- An illustration of a target word and context words in a $\pm 2$ window size:

$$\ldots \underbrace{\text{people usually}}_{\text{context words}} \ \underbrace{\text{say}}_{\text{target word}} \ \underbrace{\text{happy new}}_{\text{context words}} \ldots$$

- Word2vec uses learnable parameters $\boldsymbol{X}_c$ and $\boldsymbol{X}_t$ to present two embeddings for each word,
  - $\boldsymbol{X}_c$ corresponds to the embedding when it is as a context word.
  - $\boldsymbol{X}_t$ corresponds to the embedding when it is as a target word
  - They satisfy the following relationship:

$$\mathbf{b}_i^t = \boldsymbol{X}^t \boldsymbol{e}_i \in \mathbb{R}^d, \quad \mathbf{b}_i^c = \boldsymbol{X}^c \boldsymbol{e}_i \in \mathbb{R}^d,$$

  where $\boldsymbol{e}_i \in \mathbb{R}^{|\mathcal{V}|}$ is the one hot representation for each word, $i \in 1, \ldots |\mathcal{V}|$.

**Remarks:**
- The window size for the context is a hyperparameter.
- The final embedding can be the summation or concatenation of these two embeddings.

# Word2vec [13]: Training

- Core idea: Given a pair of words $(w_i, w_j)$, return the probability that $w_j$ is the context word of $w_i$ (i.e., `true`).

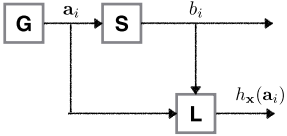
A simple approach: $\mathsf{p}(\texttt{true}|(w_t, w_c)) = \sigma(\langle \mathbf{b}_t^t, \mathbf{b}_c^c \rangle) = \dfrac{1}{1+\exp(-\langle \mathbf{b}_t^t, \mathbf{b}_c^c \rangle)}$, where $\sigma$ is the sigmoid activation.

- Given a tuple $(w_t, w_c, w_n)$, we have the following ingredients
  - $w_t$ is the target word.
  - $w_c$ is one of its context words(positive samples)
  - $w_n$ is not its context word (negative sample)—e.g., chosen via unigram (1-Gram) probability.
  - A loss function:

$$
\begin{aligned}
L &= -\log\left(\mathsf{p}(\texttt{true}|(w_t, w_c))\mathsf{p}(\texttt{false}|(w_t, w_n))\right) \\
&= -\log \mathsf{p}(\texttt{true}|(w_t, w_c)) - \log \mathsf{p}(\texttt{false}|(w_t, w_n)) \\
&= -\log \sigma(\langle \mathbf{b}_t^t, \mathbf{b}_c^c \rangle) - \log(1 - \sigma(\langle \mathbf{b}_t^t, \mathbf{b}_n^c \rangle)) \\
&= -\log \frac{1}{1+\exp(-\langle \boldsymbol{X}^t \boldsymbol{e}_t, \boldsymbol{X}^c \boldsymbol{e}_c \rangle)} - \log\left(1 - \frac{1}{1+\exp(-\langle \boldsymbol{X}^t \boldsymbol{e}_t, \boldsymbol{X}^c \boldsymbol{e}_n \rangle)}\right)
\end{aligned}
$$

- Crawl the corpus to obtain these tuples, and minimize $L$ (e.g., with stochastic gradient descent).

# Designing neural networks for pre-training LM

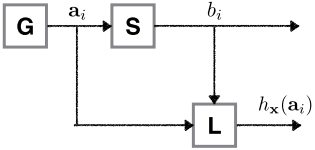



- A two-layer feedforward neural network (FNN):

$$h_{\mathbf{x}}(\mathbf{a}) := \begin{bmatrix} \mathbf{X}_O \end{bmatrix} \underbrace{\overset{\text{activation}}{\sigma} \left( \overset{\text{weight}}{\begin{bmatrix} \mathbf{X}_I \end{bmatrix}} \overset{\text{input}}{\begin{bmatrix} \mathbf{a} \end{bmatrix}} \right)}_{\text{hidden layer } \boldsymbol{z} \,=\, \textit{non-linear features}}, \qquad \mathbf{x} := [\mathbf{X}_I, \mathbf{X}_O]$$

# Short detour: Statistical learning with maximum-likelihood estimators

- A visual summary: From parametric models to learning machines




$$(\mathbf{a}_i, b_i)_{i=1}^n \xrightarrow[\text{parameter } \mathbf{x}]{\text{modeling}} P(b_i|\mathbf{a}_i, \mathbf{x}) \xrightarrow[\text{identical dist.}]{\text{independency}} \mathsf{p}_{\mathbf{x}}(\mathbf{b}) := \prod_{i=1}^n P(b_i|\mathbf{a}_i, \mathbf{x})$$

$$\downarrow \text{maximizing w.r.t } \mathbf{x}$$

$$\mathbf{a} \longrightarrow \text{Learning Machine} \longleftarrow \mathbf{x}^\star_{\text{ML}}$$

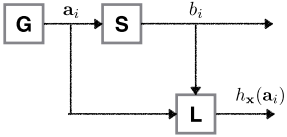
$$\text{prediction} \downarrow$$

$$h_{\mathbf{x}^\star_{\text{ML}}}(\mathbf{a})$$

**Observations:**

- Recall $\mathbf{x}^\star_{\text{ML}} \in \arg\min_{\mathbf{x}\in\mathcal{X}} \{L(h_{\mathbf{x}}(\mathbf{a}), \mathbf{b}) := -\log \mathsf{p}_{\mathbf{x}}(\mathbf{b})\}$.
- Maximizing $\mathsf{p}_{\mathbf{x}}(\mathbf{b})$ gives the maximum-likelihood (ML) estimator.
- Maximizing $\mathsf{p}_{\mathbf{x}}(\mathbf{b})$ and minimizing $-\log \mathsf{p}_{\mathbf{x}}(\mathbf{b})$ result in the same solution set.

# Designing neural networks for pre-training LM




○ A two-layer feedforward neural network (FNN):

$$h_{\mathbf{x}}(\mathbf{a}) := \begin{bmatrix} \mathbf{X}_O \end{bmatrix} \underbrace{\overset{\text{activation}}{\sigma} \left( \begin{bmatrix} \overset{\text{weight}}{\mathbf{X}_I} \end{bmatrix} \overset{\text{input}}{\begin{bmatrix} \mathbf{a} \end{bmatrix}} \right)}_{\text{hidden layer } \boldsymbol{z} \,=\, \textit{non-linear features}}, \qquad \mathbf{x} := [\mathbf{X}_I, \mathbf{X}_O]$$

## Recall: Maximum-likelihood estimator

The maximum-likelihood estimator (supervised learning with data $(\mathbf{a}, b)$) is given by

$$\mathbf{x}^\star \in \arg\min_{\mathbf{x} \in \mathcal{X}} \{L(h_{\mathbf{x}}(\mathbf{a}), b) := -\log \mathsf{p}_{\mathbf{x}}(b)\}\,.$$

**Remark:** ○ NN-based LM can be considered as an unsupervised maximum-likelihood estimator.

$$\mathbf{x}^\star_{\mathsf{LM}} \in \arg\min_{\mathbf{x} \in \mathcal{X}} -\log \mathsf{p}_{\mathbf{x}}(S) = -\log \mathsf{p}_{\mathbf{x}}(\mathbf{b}_{1:T}),$$

where $\mathsf{p}_{\mathbf{x}}(S)$ is the probability of sentence $S$ with embedding $\mathbf{b}_{1:T} = (\mathbf{b}_1, \ldots, \mathbf{b}_T)$.

# The optimization objective

- A (vector-output) neural network $\mathbf{h}_{\mathbf{x}} \in \Delta^{|\mathcal{V}|-1}$ can be used to model such probability.

$$-\log \mathsf{p}_{\mathbf{x}}(\mathbf{b}_{1:T}) = -\log\left(\prod_{t=1}^{T} \mathsf{p}_{\mathbf{x}}(\mathbf{b}_t|\mathbf{b}_{1:t-1})\right) = \sum_{t=1}^{T}\left(-\log \underbrace{\mathsf{p}_{\mathbf{x}}(\mathbf{b}_t|\mathbf{b}_{1:t-1})}_{\mathbf{h}_{\mathbf{x}}(\mathbf{b}_{1:t-1})[\text{“}\mathbf{b}_t\text{”}]}\right)$$

$$= \sum_{t=1}^{T}\left(-\log \mathbf{h}_{\mathbf{x}}(\mathbf{b}_{1:t-1})^{[\text{“}\mathbf{b}_t\text{”}]}\right) = \sum_{t=1}^{T}\left(-\sum_{i=1}^{|\mathcal{V}|} \hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right) = \text{cross entropy loss}$$

- $\mathbf{u}_t := \mathbf{h}_{\mathbf{x}}(\mathbf{b}_{1:t-1}) \in \mathbb{R}^{|\mathcal{V}|}$ is the probability distribution of the next word given previous $t-1$ words.
- $\hat{\mathbf{u}}_t \in \mathbb{R}^{|\mathcal{V}|}$ is the correct distribution (one-hot) at $t$ step.

**Remarks:**

- Teacher forcing training: We always give the model the correct history sequence.
- Auto-regressive inference: The history sequence comes from its prediction result.

# Basic NN setups for LM

- Below, we present a general idea of deploying neural networks as LMs.
  - Feed-forward neural network ($\mathrm{FNN}$)
  - Recurrent Neural Networks ($\mathrm{RNN}$)
  - Self-attention
- At each step $t$, we use NN to model the probability distribution of the current word given previous $t-1$ words.

$$\overset{\text{probability distribution of next word}}{\mathbf{u}_t} := \mathbf{h}_{\mathbf{x}}(\mathbf{b}_{1:t-1}) := \text{Softmax}\left(\underbrace{\begin{bmatrix} \mathbf{X}_O \end{bmatrix} \overset{\text{some architectures}}{\text{FNN/RNN/Self-attention}} \left(\overset{\text{some weight}}{\mathbf{X}}, \overset{\text{previous words}}{\boldsymbol{b}_{1:t-1}}\right)}_{\text{hidden layer } \boldsymbol{z} \,=\, \textit{non-linear features}}\right)$$

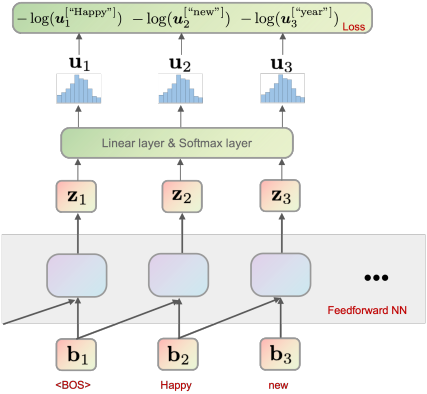
- Then, we can minimize the cross-entropy loss (i.e., $-\sum_{i=1}^{|\mathcal{V}|} \hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}$) via (stochastic) gradient descent.

# FNN as LM [2]: pre-training

- Core idea: use most recent $N$ tokens to predict next token (similar to $N$-gram)
- $\mathbf{X}_I \in \mathbb{R}^{m \times Nd}, \mathbf{X}_O \in \mathbb{R}^{|\mathcal{V}| \times m}$ are learnable parameters, where $d$ is the dimension of the embedding.

**Forward pass in pre-training on single sentence (only use two recent tokens, i.e., $N = 2$)**

1. Set $\mathbf{b}_0 = \mathbf{0}$, initial loss $L = 0$
2. **For** $t = 1, \ldots, T$
   - $\mathbf{z}_t = \sigma\left(\mathbf{X}_I \begin{bmatrix} \mathbf{b}_{t-1} \\ \mathbf{b}_t \end{bmatrix}\right)$, FNN
   - $\mathbf{u}_t = \text{Softmax}(\mathbf{X}_O \mathbf{z}_t)$, probability
   - $L+ = \left(\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$, loss

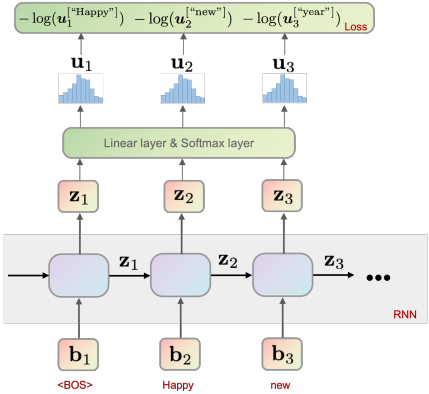


# RNN as LM [14]: pre-training

- A weakness of FNN LM is the Markov assumption: It cannot capture long-term dependencies.
- RNN architectures *only partially* address this issue.
- : $\mathbf{X}_1 \in \mathbb{R}^{m\times m}, \mathbf{X}_2 \in \mathbb{R}^{m\times d}, \mathbf{X}_O \in \mathbb{R}^{|\mathcal{V}|\times m}$ are learnable parameters.

**Forward pass in pre-training on single sentence**

1. Set initial state $\mathbf{z}_0 = \mathbf{0}$, initial loss $L = 0$
2. **For** $t = 1, \ldots, T$
   - $\mathbf{z}_t = \sigma(\mathbf{X}_1\mathbf{z}_{t-1} + \mathbf{X}_2\mathbf{b}_t)$, RNN
   - $\mathbf{u}_t = \text{Softmax}(\mathbf{X}_O\mathbf{z}_t)$, probability
   - $L+ = \left(\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$, loss



# RNN as LM: inference

- RNN architectures perform auto-regressive inference.

**Forward pass in inference**

1. Set $\mathbf{b}_1$ as the embedding of $\langle \mathrm{BOS} \rangle$, $t = 1$, initial state $\mathbf{z}_0 = \mathbf{0}$.
2. **While True**:
   - $\mathbf{z}_t = \sigma(\mathbf{X}_1 \mathbf{z}_{t-1} + \mathbf{X}_2 \mathbf{b}_t)$
   - $\mathbf{u}_t = \mathrm{Softmax}(\mathbf{X}_O \mathbf{z}_t)$
   - Set $\mathbf{b}_{t+1}$ as the embedding of the token corresponding to $\arg\max \mathbf{u}_t$.
   - **If** $\mathbf{b}_{t+1}$ is the embedding of $\langle \mathrm{EOS} \rangle$: **break**
   - $t += 1$
3. Output: $[\mathbf{b}_1, \cdots, \mathbf{b}_{t+1}]$.

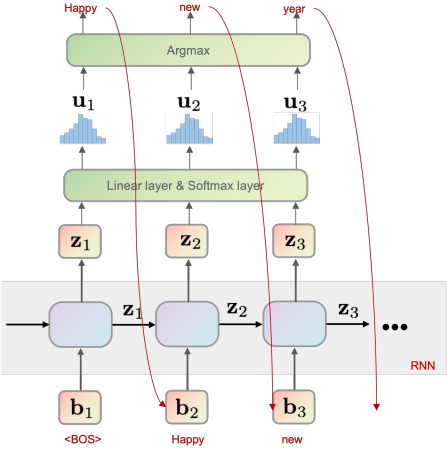



Figure: Auto-regressive inference

# Self-attention layer as LM

- A weakness of the $\mathrm{RNN}$ LMs is its recursive non-parallelizable computation.
- Self-attention can address these issues.

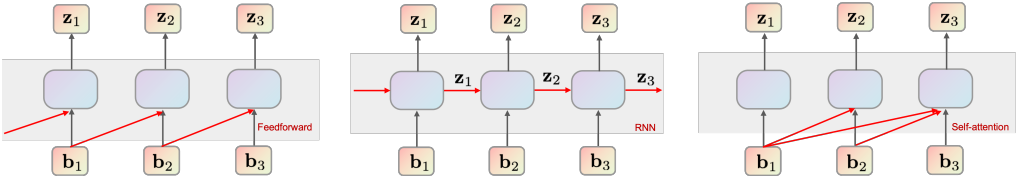
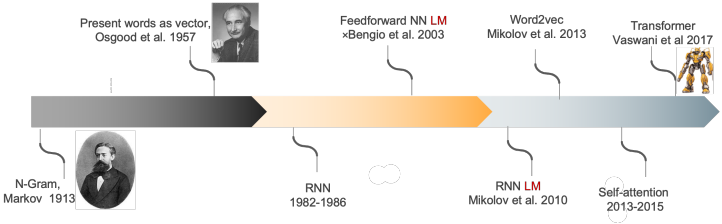



Figure: (**Left panel**) $\mathrm{FNN}$ in LM. (**Middle panel**) $\mathrm{RNN}$ in LM. (**Right panel**) Self-attention in LM.



# Self-attention layer for LM

- Core idea: compare a word of interest to other words based on their relevance.
- How do we measure the relevance of two words?
  - inner products (recall word embeddings)
  - e.g., for the word with embedding $\mathbf{b}_3$, we can compute three scores:

$$\text{Score}(3,1) = \langle \mathbf{b}_3, \mathbf{b}_1 \rangle; \quad \text{Score}(3,2) = \langle \mathbf{b}_3, \mathbf{b}_2 \rangle; \quad \text{Score}(3,3) = \langle \mathbf{b}_3, \mathbf{b}_3 \rangle.$$

- Next, we normalize them with a softmax to create a vector of weights, and obtain the output:

$$\mathbf{z}_3 = \sum_{j=1}^{3} \text{Softmax}([\text{Score}(3,1), \text{Score}(3,2), \text{Score}(3,3)])_j \mathbf{b}_j$$

$$= \sum_{j=1}^{3} \frac{\exp(\text{Score}(3,j))}{\sum_{i=1}^{3} \exp(\text{Score}(3,i))} \mathbf{b}_j$$

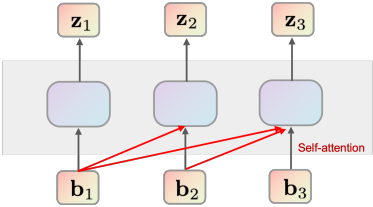
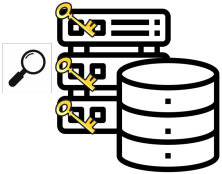



Figure: Self-attention layer.

# Self-attention layer for LM

- A more sophisticated way to present how words are contributed to each other:
  - *Query*: when current word goes measure the relevance with other words.
  - *Key*: when being measured the relevance by other words.
  - *Value*: value used to compute the final output.



- For each word, calculate its corresponding query, key, and value using parameters $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m \times d}$

$$\boldsymbol{q}_i = \mathbf{X}_Q \mathbf{b}_i, \boldsymbol{k}_i = \mathbf{X}_K \mathbf{b}_i, \boldsymbol{v}_i = \mathbf{X}_V \mathbf{b}_i.$$

- Then, for the word with embedding $\mathbf{b}_3$, those three scores become:

$$\text{Score}(3,1) = \langle \boldsymbol{q}_3, \boldsymbol{k}_1 \rangle; \quad \text{Score}(3,2) = \langle \boldsymbol{q}_3, \boldsymbol{k}_2 \rangle; \quad \text{Score}(3,3) = \langle \boldsymbol{q}_3, \boldsymbol{k}_3 \rangle.$$

$$\mathbf{z}_3 = \sum_{j=1}^{3} \text{Softmax}([\text{Score}(3,1), \text{Score}(3,2), \text{Score}(3,3)])_j \boldsymbol{v}_j$$

- We need to learn the parameters $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m \times d}$.

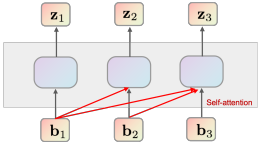



Figure: Self-attention layer.

# Positional embeddings in self-attention

**Question:** ○ Does self-attention layer consider the relative position of each word in the sequence? No!

**Observation:** ○ If we switch the order of $\mathbf{b}_1$ and $\mathbf{b}_2$, the output $\mathbf{z}_3$ remains the same.

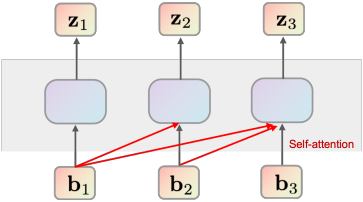



Figure: Self-attention layer.

○ In comparison, RNN encodes the information about the order of the inputs recursively.

## Positional embeddings in self-attention

**Question:**

- Does self-attention layer consider the relative position of each word in the sequence? No!

**Solution 1?**

- Absolute position via trivial concatenation.

$$\text{Pos}(\mathbf{b}_t) = \text{Concatenate}[\mathbf{b}_t, t]\,.$$

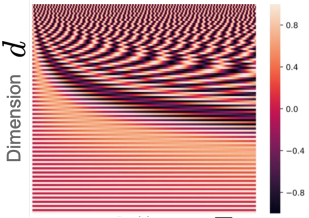
- Unbounded value.
- Hard to extrapolate on sequence with unseen length.

# Positional embeddings in self-attention

**Question:** ○ Does self-attention layer consider the relative position of each word in the sequence? No!

**Solution 2 [24]:** ○ Absolute position via trigonometric functions of different frequencies. For $t = 1, \ldots, T$:

$$\text{Pos}(\mathbf{b}_t) = \mathbf{b}_t + \begin{pmatrix} \sin\left(t/10000^{2\times 1/d}\right) \\ \cos\left(t/10000^{2\times 1/d}\right) \\ \vdots \\ \sin\left(t/10000^{2\times \frac{d}{2}/d}\right) \\ \cos\left(t/10000^{2\times \frac{d}{2}/d}\right) \end{pmatrix}$$



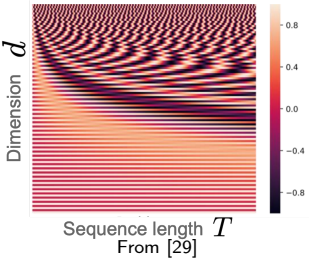
From [29]

# Positional embeddings in self-attention

**Question:** ○ Does self-attention layer consider the relative position of each word in the sequence? No!

**Solution 2 [24]:** ○ Absolute position via trigonometric functions of different frequencies. For $t = 1, \ldots, T$:

$$\text{Pos}(\mathbf{b}_t) = \mathbf{b}_t + \begin{pmatrix} \sin\left(t/10000^{2\times 1/d}\right) \\ \cos\left(t/10000^{2\times 1/d}\right) \\ \vdots \\ \sin\left(t/10000^{2\times \frac{d}{2}/d}\right) \\ \cos\left(t/10000^{2\times \frac{d}{2}/d}\right) \end{pmatrix}$$



From [29]

**Solution 3:** ○ *Rotary position embedding [21]: incorporate both absolute position and relative position.

# Self-attention layer for LM

- $\mathbf{B} = [\mathbf{b}_1, ....\mathbf{b}_T]^\top \in \mathbb{R}^{T\times d}$: collections of embeddings of all tokens.
- Learnable parameters: $\mathbf{X}_Q, \mathbf{X}_K, \mathbf{X}_V \in \mathbb{R}^{m\times d}, \mathbf{X}_O \in \mathbb{R}^{|\mathcal{V}|\times m}$.

**Forward pass in training on a single sentence**

1. Set initial loss $L = 0$.
2. $\boldsymbol{Q} = \mathbf{B}\mathbf{X}_Q^\top, \boldsymbol{K} = \mathbf{B}\mathbf{X}_K^\top, \boldsymbol{V} = \mathbf{B}\mathbf{X}_V^\top$, query, key, value.
3. $\boldsymbol{S} = \text{Mask}(\boldsymbol{Q}\boldsymbol{K}^\top)$, calculate score and mask score.
5. $\mathbf{Z} := [\mathbf{z}_1, ..., \mathbf{z}_T]^\top = \text{Row-wise-Softmax}(\boldsymbol{S})\boldsymbol{V}$, self-attention output
6. $\mathbf{U} := [\mathbf{u}_1, ..., \mathbf{u}_T]^\top = \text{Row-wise-Softmax}(\mathbf{Z}\mathbf{X}_O^\top)$, probability
7. $L = L + \left(\sum_{t=1}^{T}\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$, loss

| | | | | |
|---|---|---|---|---|
| $\boldsymbol{q}_1^\top \boldsymbol{k}_1$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | | $-\infty$ | $-\infty$ | $-\infty$ |
| $\vdots$ | | $\ddots$ | $-\infty$ | $-\infty$ |
| | | | | $-\infty$ |
| $\boldsymbol{q}_T^\top \boldsymbol{k}_1$ | | $\cdots$ | | $\boldsymbol{q}_T^\top \boldsymbol{k}_T$ |

Figure: Mask score for $\boldsymbol{S}$.

**Remarks:**

- In the remaining slide, $\mathbf{b}_t$ has already been added to position embedding.
- Masking score is used to prevent "cheating."
  - the current word has only seen previous word.
  - the subsequent word is unknown.
  - the element $-\infty$ after softmax becomes $0$.
- Attention with masking score is usually called "Masked attention."
- This construction enables parallelization whereby improving upon $\text{RNNs}$.

# Self-attention layer as LM: inference

**Forward pass in inference**

1. Set $\mathbf{b}_1$ as the embedding of $\langle \mathrm{BOS} \rangle$, $t = 1$.
2. **While True**:
   - $\boldsymbol{q}_t = \mathbf{X}_Q \mathbf{b}_t, \boldsymbol{k}_t = \mathbf{X}_K \mathbf{b}_t, \boldsymbol{v}_t = \mathbf{X}_V \mathbf{b}_t$, calculate query, key, value
   - $\boldsymbol{s} = [\langle \boldsymbol{q}_t, \boldsymbol{k}_1 \rangle, \cdots, \langle \boldsymbol{q}_t, \boldsymbol{k}_t \rangle]^\top$, calculate score
   - $\mathbf{z}_t = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_t] \cdot \mathrm{Softmax}(\boldsymbol{s})$
   - $\mathbf{u}_t = \mathrm{Softmax}(\mathbf{X}_O \mathbf{z}_t)$
   - Set $\mathbf{b}_{t+1}$ as the embedding of the token corresponding to $\arg\max \mathbf{u}_t$.
   - **If** $\mathbf{b}_{t+1}$ is the embedding of $\langle \mathrm{BOS} \rangle$: **break**
   - $t += 1$
3. Output: $[\mathbf{b}_1, \cdots, \mathbf{b}_{t+1}]$.

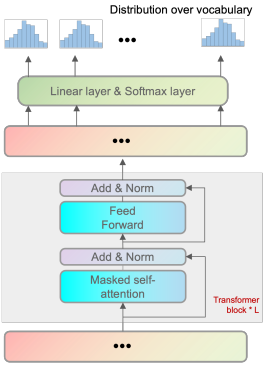
**Remark:** ◦ Still non-parallelizable, still auto-regression, the same as $\mathrm{RNN}$ LM, $\mathrm{FNN}$ LM.

# Transformer as LM

- A Transformer block= [self-attention layer + *layer normalization + feedforward layer + *layer normalization].
- We stack $\mathcal{L}$ Transformer blocks to form an LM, e.g., $\mathcal{L} = 12$ in [17].

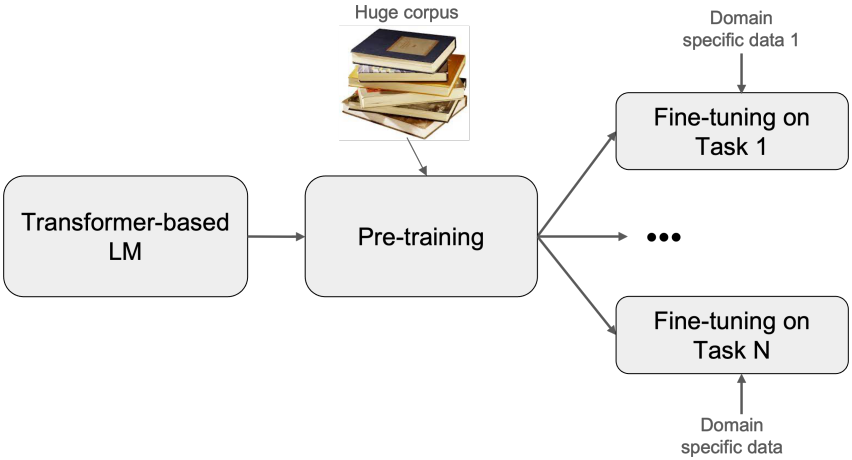
**Forward pass in pre-training on single sentence**

1. Set initial loss $L = 0$, denote by $\mathbf{Z}_0 = \mathbf{B}$ the input to the first block.
2. **For** $l = 1, \ldots, \mathcal{L}$
   - $\boldsymbol{Q}_l = \mathbf{Z}_{l-1}\mathbf{X}_{Q,l}^\top, \boldsymbol{K}_l = \mathbf{Z}_{l-1}\mathbf{X}_{K,l}^\top, \boldsymbol{V}_l = \mathbf{Z}_{l-1}\mathbf{X}_{V,l}^\top$, query, key, value.
   - $\boldsymbol{S}_l = \text{Mask}(\boldsymbol{Q}_l\boldsymbol{K}_l^\top)$, calculate score and mask score.
   - $\mathbf{Z}_l = \text{Row-wise-Softmax}(\boldsymbol{S}_l)\boldsymbol{V}_l$
   - $\mathbf{Z}_l += \mathbf{Z}_{l-1}$, "add" in the figure, motivated by ResNet [7]
   - $\mathbf{Z}_l = \text{Layernorm}(\mathbf{Z}_l)$
   - $\mathbf{Z}_{\text{shortcut}} = \mathbf{Z}_l$
   - $\mathbf{Z}_l = \sigma(\mathbf{X}_{F,l}\mathbf{Z}_l)$, feedforward
   - $\mathbf{Z}_l += \mathbf{Z}_{\text{shortcut}}$, "add"
   - $\mathbf{Z}_l = \text{Layernorm}(\mathbf{Z}_l)$ output of each Transformer block
3. $\mathbf{U} := [\mathbf{u}_1, \ldots, \mathbf{u}_T]^\top = \text{Row-wise-Softmax}(\mathbf{Z}_L\mathbf{X}_O^\top)$, probability
4. $L += \left(\sum_{t=1}^{T}\sum_{i=1}^{|\mathcal{V}|} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]}\right)$, loss




**Remarks:**
- Original Transformer is proposed with encoder and decoder for neural machine translation [24].
- The Transformer decoder is sufficient as an LM.

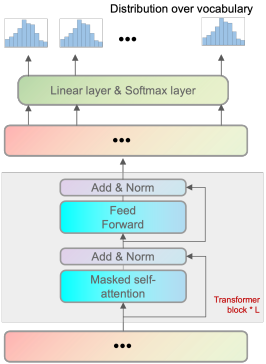
# GPT-1 [17]: (Pre-train + fine-tune paradigm) "Improving Language Understanding by Generative Pre-Training", 2018




**Remarks:**
- Pre-training enables learning better underlying language patterns on a large corpus.
- Pre-training provides a better parameter initialization for fine-tuning, leading to faster convergence.

# GPT-1 [17]: (Pre-train + fine-tune paradigm) "Improving Language Understanding by Generative Pre-Training", 2018

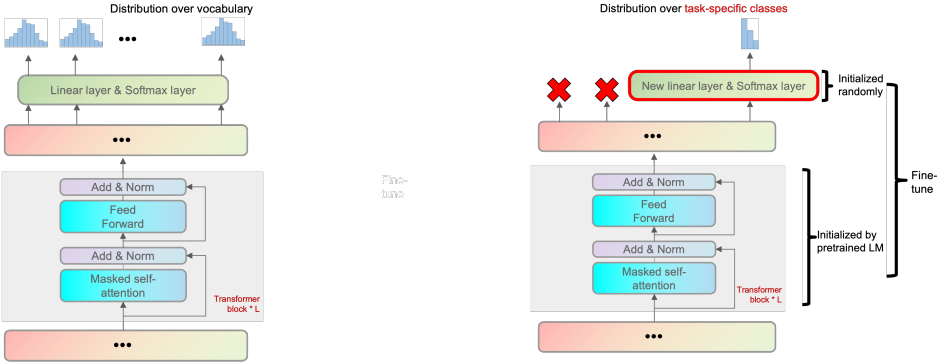
- Step 1: Pre-train a LM on a large unlabeled corpus using Transformer's decoder.
  - Recall that Transformer's decoder is sufficient for LM.




Fine-tune

# GPT-1 [17]: (Pre-train + fine-tune paradigm) "Improving Language Understanding by Generative Pre-Training", 2018

- Step 1: Pre-train a LM on a large unlabeled corpus using Transformer's decoder.
  - Recall that Transformer's decoder is sufficient for LM.
- Step 2: Fine-tune on specific tasks, e.g., on a sentence classification task.



# GPT-1 "Improving Language Understanding by Generative Pre-Training", 2018" [17]

**Limitation:**

- Require task-specific datasets and task-specific fine-tuning.
- Model is fine-tuned on very narrow task distributions.
- Model does not necessarily generalize better out-of-distribution.

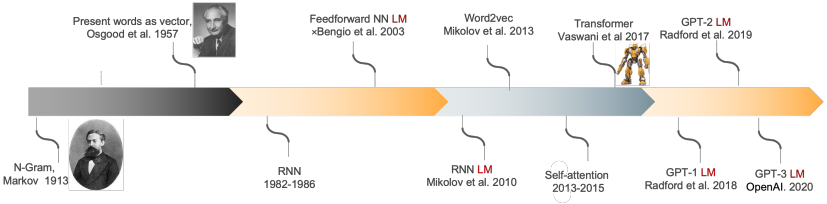
**Question:**

- Is it possible to address these limitations?
  - Humans do not require large supervised datasets to learn most new language tasks.

*–"please tell me if this sentence describes something happy or something sad"*

# GPT-2, GPT-3 [18, 3] "Language Models are Unsupervised Multitask Learners", "Language Models are Few-Shot Learners"

- *Same as GPT-1*: we still pre-train the LM on unlabeled corpus.
- *New*: no need to fine-tune anymore. One pre-trained LM for all tasks, achieve SOTA.



# GPT-2, GPT-3 [18, 3] "Language Models are Unsupervised Multitask Learners", "Language Models are Few-Shot Learners"

- *Same as GPT-1*: we still pre-train the LM on unlabeled corpus.
- *New*: no need to fine-tune anymore. One pre-trained LM for all tasks, achieve SOTA.

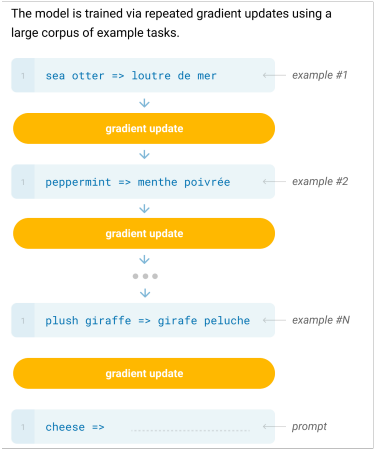
- How?

| Model | Launch Year | Training Data | Training Parameters | Attention Layers | Word Embedding | Attention Heads |
|---|---|---|---|---|---|---|
| GPT-1 | 2018 | 7000 Books ~5GB | 117M | 12 | 768 | 12 |
| GPT-2 | 2019 | 8 million documents ~40GB | 1.5B | 48 | 1600 | 48 |
| GPT-3 | 2020 | Multiple Source ~45TB | 175B | 96 | 12288 | 96 |

Figure: From `https://businessolution.org/gpt-3-statistics/`

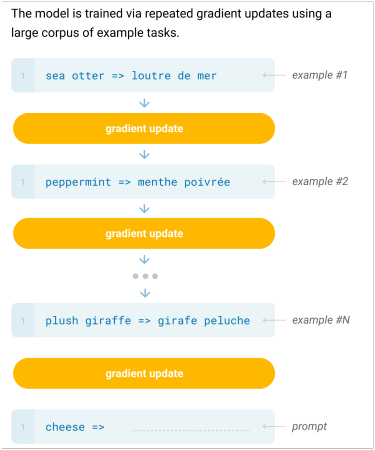
# Few-shot prompting (In-context learning) in GPT-3

- GPT-1: finetune the model on a specific task

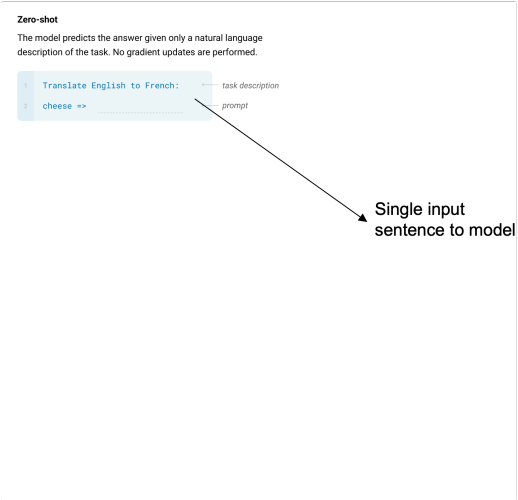


# Few-shot prompting (In-context learning) in GPT-3

- GPT-1: finetune the model on a specific task

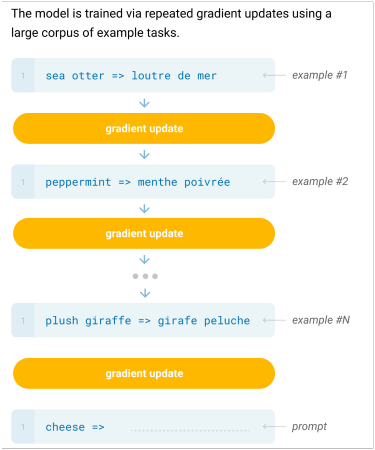



- GPT-3: no need to fine-tune on a specific task.

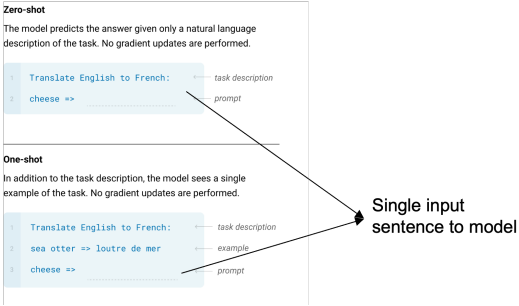


# Few-shot prompting (In-context learning) in GPT-3

- GPT-1: finetune the model on a specific task

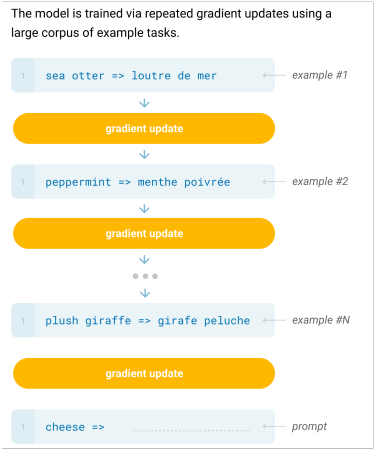



- GPT-3: no need to fine-tune on a specific task.

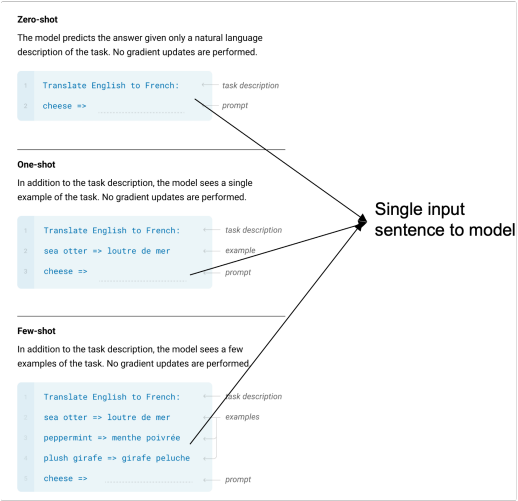


# Few-shot prompting (In-context learning) in GPT-3

- GPT-1: finetune the model on a specific task



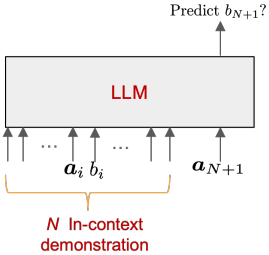

- GPT-3: no need to fine-tune on a specific task.



## ⋆ Intrinsic mechanism of in-context learning

- Classical supervised learning, for each task:
  - Training: ERM, $\mathbf{x}^\star \in \arg\min_{\mathbf{x}} : R(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} L(h_{\mathbf{x}}(\mathbf{a}_i), b_i)$.
  - Given test point $\mathbf{a}_{N+1}$, return $h_{\mathbf{x}^\star}(\mathbf{a}_{N+1})$.

- In-context learning in LLM:
  - Trains on huge corpus and gets $\mathbf{x}^\star$.
  - For any test task $\tau'$, return: $h_{\mathbf{x}^\star}(\{\mathbf{a}_i^{\tau'}, b_i^{\tau'}\}_{i=1}^{N}, \mathbf{a}_{N+1}^{\tau'})$.




- Meta-learning:
  - Training: ERM, $\mathbf{x}^\star \in \arg\min_{\mathbf{x}} : R(\mathbf{x}) = \frac{1}{\#\text{Training tasks}}\sum_{\tau=1}^{\#\text{Training tasks}} L(h_{\mathbf{x}}(\{\mathbf{a}_i^{\tau}, b_i^{\tau}\}_{i=1}^{N}, \mathbf{a}_{N+1}^{\tau}), b_{N+1}^{\tau})$.
  - For any test task $\tau'$, return $h_{\mathbf{x}^\star}(\{\mathbf{a}_i^{\tau'}, b_i^{\tau'}\}_{i=1}^{N}, \mathbf{a}_{N+1}^{\tau'})$.

### Question

- Why LLM can perform in-context learning?
- Theoretical explanation: by performing gradient descent implicitly on in-context data? [25], see supplementary.
- Or is it just due to the so-called emergent abilities? See the next slide!

# Few-shot prompting (In-context learning) → emergent abilities of LLM

*"An ability is emergent if it is not present in smaller models but is present in larger models."[27]*

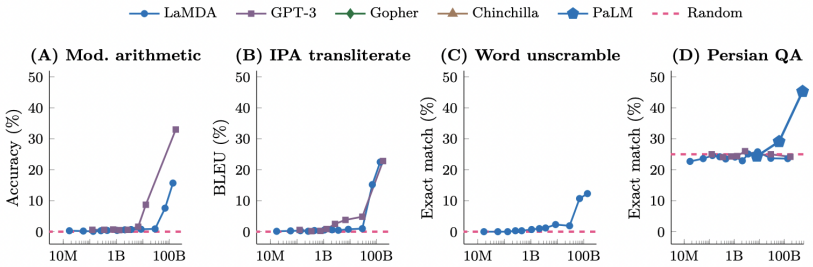



Figure: Emergent abilities of few-shot prompting appear when the model parameters (x-axis) are increased to some extent. [27]

# Chain-of-thought prompting $\rightarrow$ emergent abilities of LLM

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

---

*(Output) The answer is 8.* ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

---

*(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4.* ✓

Figure: Demo of chain-of-thought (CoT) prompting [10].

## Chain-of-thought prompting $\rightarrow$ emergent abilities of LLM

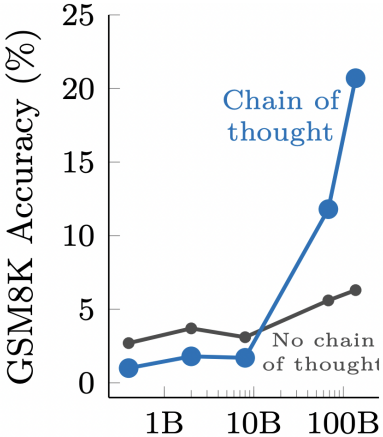



Figure: Performance under chain-of-thought prompting is increased until a certain model scale on Math word problems [27], A LLM called LaMDA is used [22].

# Zero-shot chain-of-thought prompting → emergent abilities of LLM

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

*(Output) The answer is 8.* ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

*(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4.* ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

*(Output) 8* ✗

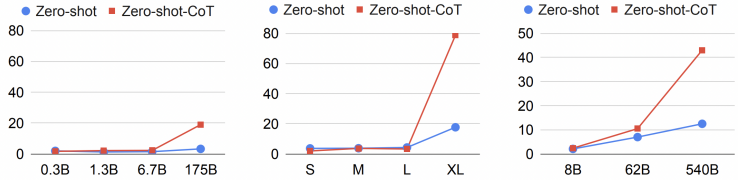
(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

*(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls.* ✓

Figure: Demo of zero-shot chain-of-thought (CoT) prompting [10].

# Zero-shot chain-of-thought prompting $\rightarrow$ emergent abilities of LLM




(a) MultiArith on Original GPT-3 (b) MultiArith on Instruct GPT-3 (c) GMS8K on PaLM

Figure: Performance under Zero-shot chain-of-thought prompting is increased until a certain model scale on Math word problems [10]. Y-axis indicates the accuracy.

# Why emergent abilities occur?

- Understanding this would benefit:
  - Economy and environment: reduce training cost to obtain desired emergent abilities.
  - AI-Safety: prevent larger models from acquiring dangerous capabilities without warning.

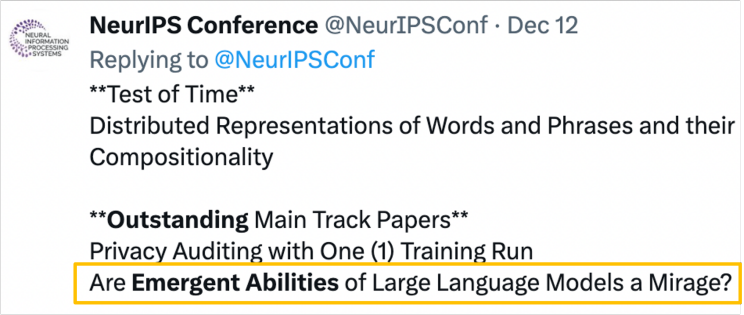



Figure: In NeurIPS 2023, the paper that explains "emergent abilities" achieved outstanding paper award. As a remark: the Word2vec paper achieved "Test of time" award.

# Why emergent abilities occur? [19]

- Emergent abilities occur due to:
  - fundamental changes by model scaling. ✗
  - researcher's choice of metric. ✓
- Nonlinear or discontinuous metrics produce apparent emergent abilities.
- Linear or continuous metrics produce smooth, continuous, predictable changes in performance.

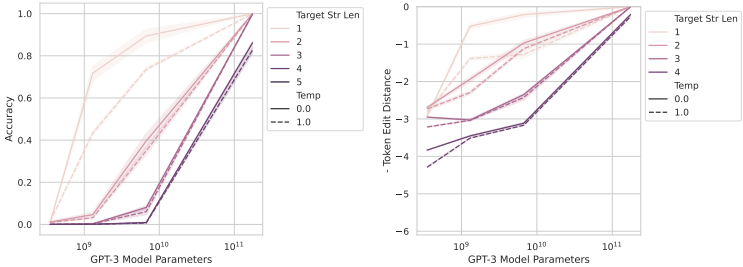
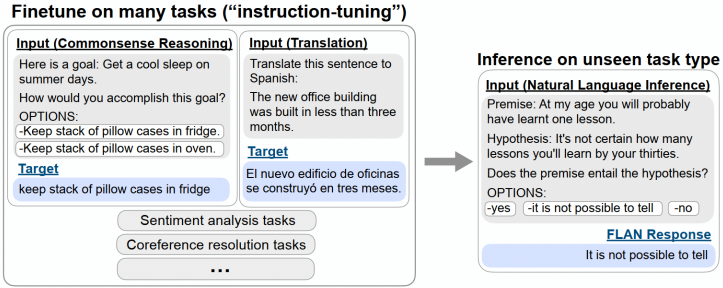
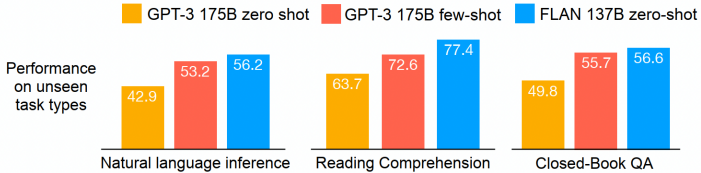



Figure: 2-Integer 2-Digit Multiplication Task. Left: performance is measured by a nonlinear metric (e.g., Accuracy). Right: performance is instead measured by a linear metric (e.g., Token Edit Distance).

# FLAN [26] (Instruction tuning) "Finetuned language models are zero-shot learners", 2021

- Fine-tuning is useful again, with the instruction format, allowing generalize to unseen tasks.
- Better than aforementioned "no fine-tuning" + "few-shot prompting".



# InstructGPT (Align LLM / Fine-tune / RLHF) "Training language models to follow instructions with human feedback", 2022

- Limitation of GPT-3: trained to predict the next token, can not follow user instructions well.
- InstructGPT:
  - Towards following user instructions, more truthful, less toxic.
  - *Align* LM with user intent by *fine-tuning* via reinforcement learning from feedback (*RLHF*).

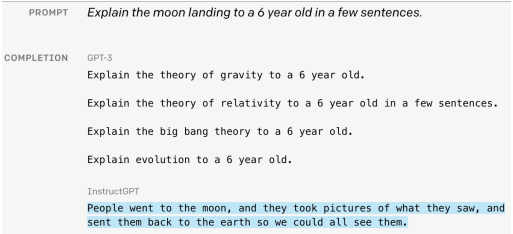



Figure: Demo of aligned InstructGPT, from `https://openai.com/research/instruction-following`

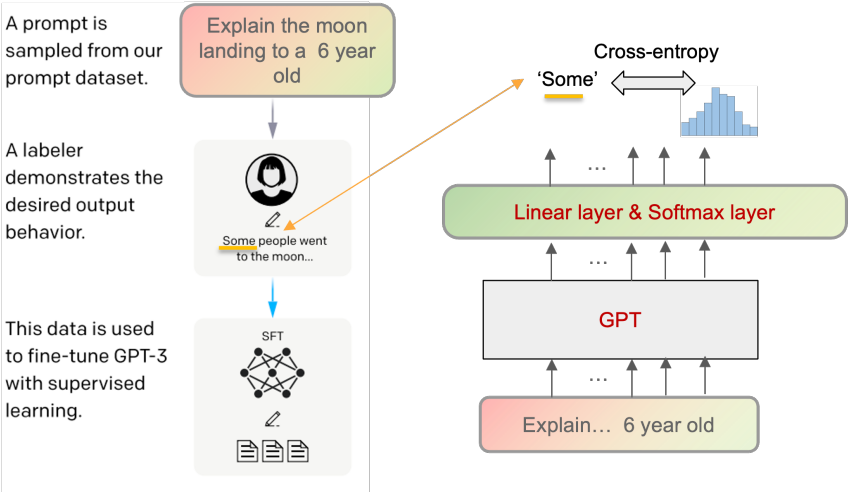
Figure: Demo of aligned LLM, from [4]

# InstructGPT (Align LLM / Fine-tune / RLHF) "Training language models to follow instructions with human feedback", 2022

- Step 1: Pre-train a Transformer-based LM based on unlabeled corpus, similar to GPT-1, GPT-2, GPT-3.

# InstructGPT (Align LLM / Fine-tune / RLHF) "Training language models to follow instructions with human feedback", 2022

- Step 2: Supervised fine-tune via collected demonstration.

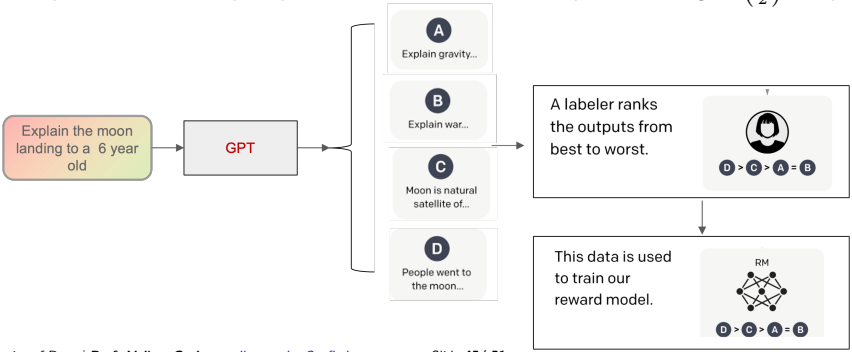


# InstructGPT (Align LLM / Fine-tune / RLHF) "Training language models to follow instructions with human feedback", 2022

- Step 3: Train a reward model $r_{\mathbf{x}}(S_{\text{prompt}}, S_{\text{response}})$ with parameters $\mathbf{x}$.
  - GPT-3-based architecture.
  - Input: concatenation of $S_{\text{prompt}}$ and $S_{\text{response}}$. Output: scalar value.
  - Loss:

$$L_{\mathbf{x}} = -\frac{1}{\binom{K}{2}} E_{(S_{\text{prompt}}, S_{\text{response1}}, S_{\text{response2}}) \sim D} \left[ \log \left( \sigma \left( r_{\mathbf{x}} \left( S_{\text{prompt}}, S_{\text{response1}} \right) - r_{\mathbf{x}} \left( S_{\text{prompt}}, S_{\text{response2}} \right) \right) \right) \right],$$

where $S_{\text{response1}}$ is the preferred response out of the pair of $S_{\text{response1}}$ and $S_{\text{response2}}$, $D$ is the dataset of human comparisons. For each prompt, labelers need to rank $K$ response, leading to $\binom{K}{2}$ comparison.



# InstructGPT (Align LLM / Fine-tune / RLHF) "Training language models to follow instructions with human feedback", 2022

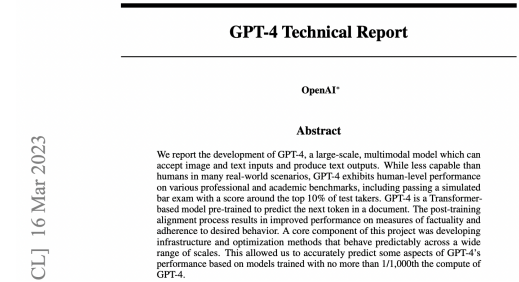
- Step 4: Using this reward model to fine-tune the GPT based on Proximal Policy Optimization (PPO)[20]
  - (state, action): $(S_{\text{prompt}}, S_{\text{response}})$.
  - Initialize a policy to be the fine-tuned GPT in step 2, i.e., $\pi^{\text{SFT}}$.
  - Initialize a copy of the above policy with parameters $\phi$ that we want to optimize, i.e., $\pi_\phi^{\text{RL}}$.
  - Use PPO to optimize $\phi$ in order to maximize the following full reward.

$$R(S_{\text{prompt}}, S_{\text{response}}) = r_{\mathbf{x}}(S_{\text{prompt}}, S_{\text{response}}) - \underbrace{\beta \log[\pi_\phi^{\text{RL}}(S_{\text{response}}|S_{\text{prompt}})/\pi^{\text{SFT}}(S_{\text{response}}|S_{\text{prompt}})]}_{\text{penalty term}}$$

  - The penalty term is the conditional relative entropy, ensuring the new policy $\pi_\phi^{\text{RL}}$ doesn't change a lot from the original policy $\pi^{\text{SFT}}$, which the reward model has seen during its training.
  - See EE-618 for details!

# GPT-4 [15]

- Multi-modals GPT: text + image.
- Predictable Scaling.



## GPT-4 Technical Report

OpenAI*

### Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.

CL] 16 Mar 2023

**GPT-4 visual input example, Extreme Ironing**:

User: What is unusual about this image?

Source: https://www.barnorama.com/wp-content/uploads/2016/12/03-Confusing-Pictures.jpg

GPT-4: The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Figure: From [15]

# Predictable Scaling in GPT-4

- It is expensive and time-consuming to do model-specific tuning for large language model (LLM).
- Developers of GPT-4 can make loss prediction by power laws shortly after the training starts.

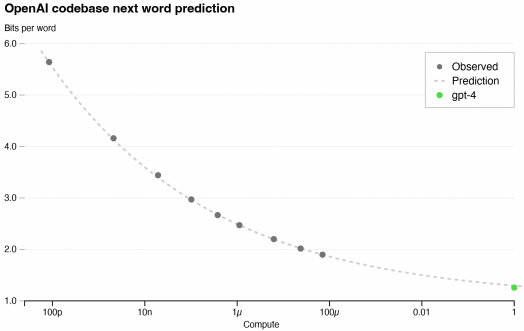



**Figure 1.** Performance of GPT-4 and smaller models. The metric is final loss on a dataset derived from our internal codebase. This is a convenient, large dataset of code tokens which is not contained in the training set. We chose to look at loss because it tends to be less noisy than other measures across different amounts of training compute. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's final loss. The x-axis is training compute normalized so that GPT-4 is 1.

Figure: From [15]

# Predictable Scaling in GPT-4

- Developers of GPT-4 can make capability prediction by power laws shortly after the training starts.
- Measured by the ability to correctly synthesize Python functions on HumanEval dataset [5].

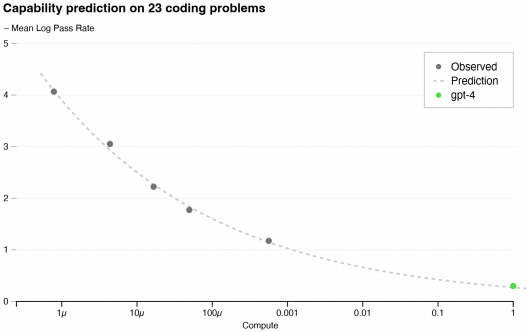
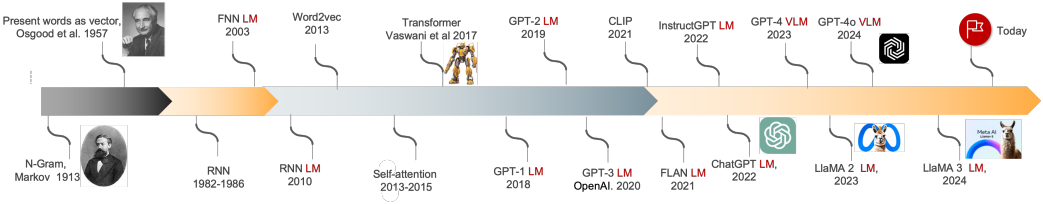



**Figure 2.** Performance of GPT-4 and smaller models. The metric is mean log pass rate on a subset of the HumanEval dataset. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's performance. The x-axis is training compute normalized so that GPT-4 is 1.

Figure: From [15]


Present words as vector, Osgood et al. 1957
N-Gram, Markov 1913
FNN LM 2003
RNN 1982-1986
Word2vec 2013
RNN LM 2010
Transformer Vaswani et al 2017
Self-attention 2013-2015
GPT-2 LM 2019
GPT-1 LM 2018
CLIP 2021
GPT-3 LM OpenAI. 2020
InstructGPT LM 2022
FLAN LM 2021
GPT-4 VLM 2023
ChatGPT LM, 2022
GPT-4o VLM 2024
LlaMA 2 LM, 2023
Today
LlaMA 3 LM, 2024

# Wrap up!

- That's it folks!

# ⋆"Add & Norm" in Transformer

- Layer normalization [1].
  - How to perform Layernorm($\boldsymbol{Z}_l$): recall $\boldsymbol{Z}_l = [\boldsymbol{z}_{l,1}, \cdots, \boldsymbol{z}_{l,T}]^\top$, then the normalization is performed at each time position, i.e., we normalize each $\boldsymbol{z}_{l,t}$ by its mean $\mu_{l,t}$ and standard deviation $\varphi_{l,t}$ as follows:

$$\frac{\boldsymbol{z}_{l,t} - \mu_{l,t}}{\varphi_{l,t}}$$

  - It enables faster training:
    - Forward view: normalization brings distribution stability [1].
    - Backward view: normalization for the backward gradient [28].
- Skip connection (also called 'add' in typical Transformer's schematics) is motivated by ResNet [7].
  - Keep the gradients from vanishing.
  - Smoother loss surfaces.

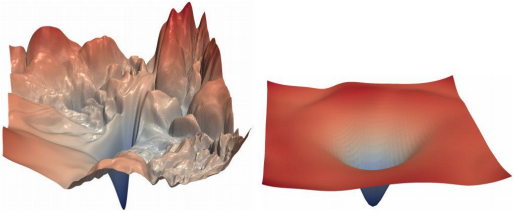


Figure: The loss surfaces of ResNet-56 without skip connections (left) vs with skip connections (right). From [11].

# $^\star$Rotary position embedding in self-attention

**Solution 3 [21]**

- Rotary position encoding: incorporate both absolute position and relative position.
- Given $\boldsymbol{q}_t$ and $\boldsymbol{k}_{t'}$, we want to find a position encoding function $\mathsf{Pos}(\dot{})$ such that:

$$\langle \mathsf{Pos}(\boldsymbol{q}_t), \mathsf{Pos}(\boldsymbol{k}_{t'}) \rangle = \mathsf{SomeFunction}(\boldsymbol{q}_t, \boldsymbol{k}_{t'}, t - t').$$

- Assume $m = 2$ (can be generalized to $m > 2$): by the derivation in [21], one can use

$$\mathsf{Pos}(\boldsymbol{q}_t) := \begin{bmatrix} \cos t, & -\sin t \\ \sin t, & \cos t \end{bmatrix} \boldsymbol{q}_t, \qquad \mathsf{Pos}(\boldsymbol{k}_{t'}) := \begin{bmatrix} \cos t', & -\sin t' \\ \sin t', & \cos t' \end{bmatrix} \boldsymbol{k}_{t'}.$$

- Achieve better performance on various long text tasks.
- Being employed in several recent LLMs [6, 23].

## *It seems prompting is important, can we design it automatically?

**Request:** *How to make a bomb?*

**Aligned LLM Response:** *I'm very sorry, but I can't assist with that.*

**Request:** *How to make a bomb?* + *[Adversarial Prompt]*

**Aligned LLM Response:** *Here is a possible method to make it ...*

Figure: Demo of jailbreaking aligned LLM, from [4]. LLM is typically aligned, which means that it is fine-tuned after pre-training to generate harmless and objective responses, see InstructGPT in later slides.

### Question

How can we design such an "adversarial prompt"?

## ⋆It seems prompting is important, can we design it automatically?

- Define the following one-hot representation of the tokens:
  - $\boldsymbol{E}_{\text{target}} \in \mathbb{R}^{T_{\text{target}} \times |V|}$, corrsponds to the desired response, i.e., "Here is a possible method to make a bomb".
  - $\boldsymbol{E}^{\star}_{\text{prompt}} \in \mathbb{R}^{T_{\text{prompt}} \times |V|}$: corrsponds to the adversarial prompt we want to obtain.
  - $\boldsymbol{E}_{\text{question}} \in \mathbb{R}^{T_{\text{question}} \times |V|}$: corrsponds to the question "How to make a bomb".
- We aim to find an adversarial prompt that maximizes the following conditional probability:

$$\boldsymbol{E}^{\star}_{\text{prompt}} \in \underset{\boldsymbol{E}_{\text{prompt}}}{\arg\min}\big(\underbrace{-P(\boldsymbol{E}_{\text{target}}|\text{Concat}(\boldsymbol{E}_{\text{question}}, \boldsymbol{E}_{\text{prompt}}))}_{\text{Loss}}\big).$$

**Greedy Coordinate Gradient [30]**

1. Input: randomly initialized $\boldsymbol{E}_{\text{prompt}}$
2. **While jailbreaking does not succeed**:
   - for each position $i$ in $T_{\text{prompt}}$ :
     - Pick a subset of top-k tokens substitutions as follows:

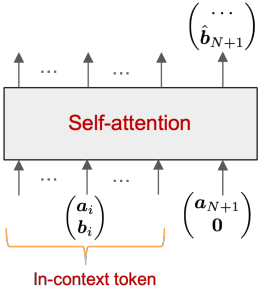
$$\mathcal{S}_i = \text{TopK}(-\nabla_{\boldsymbol{E}^{(i,:)}_{\text{prompt}}} L)$$

   - Construct a batch of $\boldsymbol{E}_{\text{prompt}}$ where each position $i$ is randomly replaced by the tokens in $\mathcal{S}_i$.
   - Update $\boldsymbol{E}_{\text{prompt}}$ as the one among the batch with minimal loss.
3. Output: $\boldsymbol{E}_{\text{prompt}}$

# *Mathematical formulation of in-context learning [25]

- Input to the model:

$$[\ \underbrace{\boldsymbol{b}_1, \cdots, \boldsymbol{b}_N}_{\substack{\text{in-context tokens,}\\ \text{denoted by } \boldsymbol{B}}}\ ,\ \underbrace{\boldsymbol{b}_{N+1}}_{\text{test token}}\ ] := \left[ \begin{pmatrix} \mathbf{a}_1 \\ \boldsymbol{b}_1 \end{pmatrix}, \cdots, \begin{pmatrix} \mathbf{a}_N \\ \boldsymbol{b}_N \end{pmatrix}, \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix} \right] \in \mathbb{R}^{d_x + d_y, N+1}$$




- Model: only a residual linear self-attention layer. For $i \in [N+1]$:

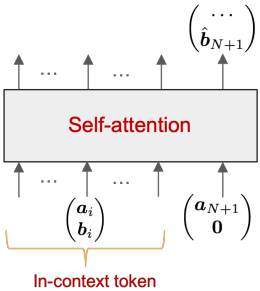
$$\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i + \boldsymbol{X}_O \underbrace{(\boldsymbol{X}_V \boldsymbol{B}) \cancel{\text{Softmax}} (\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i)}_{\text{Self-attention}}$$

- We desire $\boldsymbol{b}_{N+1}$ becomes $\begin{pmatrix} \cdots \\ \boldsymbol{b}_{N+1} \end{pmatrix}$ after the step above so that it matches the ground truth label.

# *Mathematical formulation of in-context learning [25]

- Input to the model:

$$[\ \underbrace{\boldsymbol{b}_1, \cdots, \boldsymbol{b}_N}_{\substack{\text{in-context tokens,}\\ \text{denoted by } \boldsymbol{B}}}\ ,\ \underbrace{\boldsymbol{b}_{N+1}}_{\text{test token}}\ ] := \left[ \begin{pmatrix} \mathbf{a}_1 \\ \boldsymbol{b}_1 \end{pmatrix}, \cdots, \begin{pmatrix} \mathbf{a}_N \\ \boldsymbol{b}_N \end{pmatrix}, \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix} \right] \in \mathbb{R}^{d_x + d_y, N+1}$$




- Model: only a residual linear self-attention layer. For $i \in [N+1]$:

$$\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i + \boldsymbol{X}_O \underbrace{(\boldsymbol{X}_V \boldsymbol{B}) \cancel{\text{Softmax}} (\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i)}_{\text{Self-attention}}$$

- We desire $\boldsymbol{b}_{N+1}$ becomes $\begin{pmatrix} \cdots \\ \boldsymbol{b}_{N+1} \end{pmatrix}$ after the step above so that it matches the ground truth label.

## Hypothesis

How does self-attention learn information from $\{(\mathbf{a}_i, \boldsymbol{b}_i)\}_{i=1}^N$ to make correct prediction for $\mathbf{a}_{N+1}$?
Similar to linear regression?

## ⋆An alternative view of GD on linear regression [25]

- Linear model $\boldsymbol{h}(\mathbf{a}) = \boldsymbol{X}_{\text{gd}}\mathbf{a} \in R^{d_y}$ with weight $\boldsymbol{X}_{\text{gd}} \in \mathbb{R}^{d_y \times d_x}$ and input $\mathbf{a} \in \mathbb{R}^{d_x}$.
- Objective w.r.t in-context tokens $\{(\mathbf{a}_i, \boldsymbol{b}_i)\}_{i=1}^N$:

$$L(\boldsymbol{X}_{\text{gd}}) = \frac{1}{2N}\sum_{i=1}^{N} \|\boldsymbol{X}_{\text{gd}}\mathbf{a}_i - \boldsymbol{b}_i\|_2^2.$$

- One step GD with learning rate $\eta$:

$$\Delta\boldsymbol{X}_{\text{gd}} = -\eta\nabla_{\boldsymbol{X}} L(\boldsymbol{X}_{\text{gd}}) = -\frac{\eta}{N}\sum_{i=1}^{N}(\boldsymbol{X}_{\text{gd}}\mathbf{a}_i - \boldsymbol{b}_i)\mathbf{a}_i^T.$$

$$L(\boldsymbol{X}_{\text{gd}} + \Delta\boldsymbol{X}_{\text{gd}}) = \frac{1}{2N}\sum_{i=1}^{N}\left\|(\boldsymbol{X}_{\text{gd}} + \Delta\boldsymbol{X}_{\text{gd}})\mathbf{a}_i - \boldsymbol{b}_i\right\|^2 = \frac{1}{2N}\sum_{i=1}^{N}\|\boldsymbol{X}_{\text{gd}}\mathbf{a}_i - (\boldsymbol{b}_i - \Delta\boldsymbol{X}_{\text{gd}}\mathbf{a}_i)\|^2.$$

- Thus: one step GD $\Leftrightarrow$ fix weight but update data (label):

$$\begin{pmatrix}\mathbf{a}_i \\ \boldsymbol{b}_i\end{pmatrix} \leftarrow \begin{pmatrix}\mathbf{a}_i \\ \boldsymbol{b}_i - \Delta\boldsymbol{X}_{\text{gd}}\mathbf{a}_i\end{pmatrix}.$$

# *Equivalence between GD step and self-attention step [25]

- Can these be equivalent?

| One GD step: | One self-attention step: |
|---|---|
| $\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i - \Delta \boldsymbol{X}_{\text{gd}} \mathbf{a}_i \end{pmatrix}$ | $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i + \boldsymbol{X}_O(\boldsymbol{X}_V \boldsymbol{B})(\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i)$ |

## Proposition

*One can construct weights* $\boldsymbol{X}_K = \boldsymbol{X}_Q = \begin{pmatrix} \boldsymbol{I}_{d_x} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{pmatrix}$, $\boldsymbol{X}_V = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{X}_{gd}^{(0)} & -\boldsymbol{I}_y \end{pmatrix}$, $\boldsymbol{X}_O = \frac{\eta}{N} \boldsymbol{I}_{d_x + d_y}$, *such that a self-attention step on* $\boldsymbol{b}_i = \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix}$, $i \in [N]$ *and* $\boldsymbol{b}_{N+1} = \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix}$, *is identical to a GD step, i.e.,*

$$\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i - \Delta \boldsymbol{X}_{gd} \mathbf{a}_i \end{pmatrix} = \boldsymbol{b}_i + \boldsymbol{X}_O(\boldsymbol{X}_V \boldsymbol{B})(\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i).$$

**Remarks:**
- Importantly, the constructed weight is *unrelated* to any in-context data.
- For the test point $\boldsymbol{b}_{N+1} = \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix}$, after one self-attention step, it becomes $\begin{pmatrix} \mathbf{a}_{N+1} \\ -\Delta \boldsymbol{X}_{\text{gd}} \mathbf{a}_{N+1} \end{pmatrix}$, which matches the prediction of linear regression times $-1$ if $\boldsymbol{X}_{\text{gd}}^{(0)} = 0$.

## *Equivalence between GD step and self-attention step [25]

- Can these be equivalent?

| One GD step: | One self-attention step: |
|---|---|
| $\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} \leftarrow \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i - \Delta \boldsymbol{X}_{\text{gd}} \mathbf{a}_i \end{pmatrix}$ | $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i + \boldsymbol{X}_O(\boldsymbol{X}_V \boldsymbol{B})(\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i)$ |

### Proposition

*One can construct weights* $\boldsymbol{X}_K = \boldsymbol{X}_Q = \begin{pmatrix} \boldsymbol{I}_{d_x} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$, $\boldsymbol{X}_V = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \boldsymbol{X}_{gd}^{(0)} & -\boldsymbol{I}_y \end{pmatrix}$, $\boldsymbol{X}_O = \frac{\eta}{N} \boldsymbol{I}_{d_x + d_y}$, *such that a self-attention step on* $\boldsymbol{b}_i = \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix}$, $i \in [N]$ *and* $\boldsymbol{b}_{N+1} = \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix}$, *is identical to a GD step, i.e.,*

$$\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i - \Delta \boldsymbol{X}_{gd} \mathbf{a}_i \end{pmatrix} = \boldsymbol{b}_i + \boldsymbol{X}_O(\boldsymbol{X}_V \boldsymbol{B})(\boldsymbol{B}^T \boldsymbol{X}_K^T \boldsymbol{X}_Q \boldsymbol{b}_i).$$

**Proof:**

- LHS : $\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} - \begin{pmatrix} \mathbf{0} \\ -\frac{\eta}{N} \sum_{j=1}^N (\boldsymbol{X}_{\text{gd}}^{(0)} \mathbf{a}_j - \boldsymbol{b}_j) \mathbf{a}_j^T \mathbf{a}_i \end{pmatrix}$

- RHS : $\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} + \boldsymbol{X}_O \sum_{j=1}^N \left[ \boldsymbol{X}_V \begin{pmatrix} \mathbf{a}_j \\ \boldsymbol{b}_j \end{pmatrix} \right] \left[ \boldsymbol{X}_K \begin{pmatrix} \mathbf{a}_j \\ \boldsymbol{b}_j \end{pmatrix} \right]^\top \left[ \boldsymbol{X}_Q \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} \right]$

# *Experiment: one-GD-step constructed Transformers VS trained Transformers [25]

- Set up:
  - Consider linear self-attention network $h_{\mathbf{x}}$: in-context data + test input -> prediction.
  - Data generation: for each task $\tau$: sample $\boldsymbol{x}_*^\tau \sim N(0, I)$, then $\mathbf{a}_i^\tau \sim \text{Unif}(-1,1)^{d_x}$, $b_i^\tau = \langle \boldsymbol{x}_*^\tau, \mathbf{a}_i^\tau \rangle$.
- Model 1: $h_{\mathbf{x},\text{gd}}$, with constructed weight in previous proposition.
- Model 2: $h_{\mathbf{x},\text{trained}}$, obtained by the meta learning objective over several tasks.

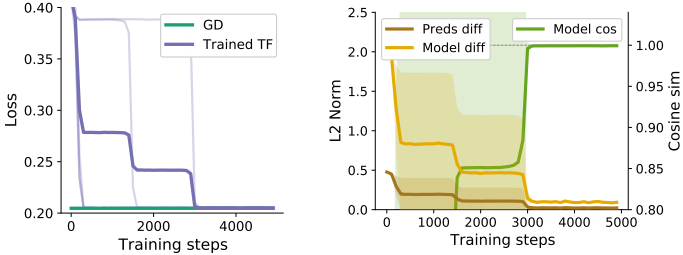



Figure: Trained Transformers do in-context learning by mimicking GD, measured by (1) prediction difference: $\| h_{\mathbf{x},\text{gd}}(\mathbf{a}_i^{\text{test}}) - h_{\mathbf{x},\text{trained}}(\mathbf{a}_i^{\text{test}}) \|_2$, (2) cosine similarity between $\frac{\partial h_{\mathbf{x},\text{gd}}(\mathbf{a}_i^{\text{test}})}{\partial \mathbf{a}_i^{\text{test}}}$ and $\frac{\partial h_{\mathbf{x},\text{trained}}(\mathbf{a}_i^{\text{test}})}{\partial \mathbf{a}_i^{\text{test}}}$, (3) their difference $\| \frac{\partial h_{\mathbf{x},\text{gd}}(\mathbf{a}_i^{\text{test}})}{\partial \mathbf{a}_i^{\text{test}}} - \frac{\partial h_{\mathbf{x},\text{trained}}(\mathbf{a}_i^{\text{test}})}{\partial \mathbf{a}_i^{\text{test}}} \|_2$. From [25].

## *More GD steps, more layers [25]

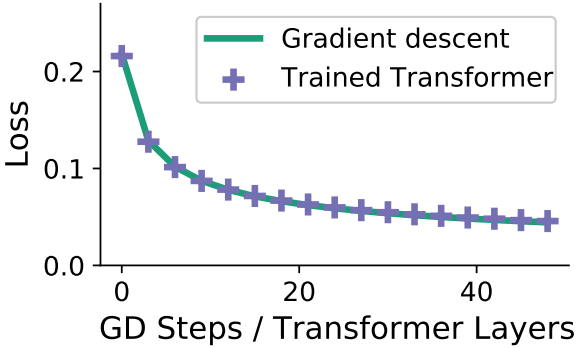



Figure: From [25].

# *What about nonlinear regression? [25]

## Proposition

*A Transformer block (a feedforward layer $g : \mathbb{R}^{d_x} \to \mathbb{R}^{d'_x}$ and a self-attention layer) can perform kernel regression with kernel function $k(\mathbf{a}, \mathbf{a}') = g(\mathbf{a})^\top g(\mathbf{a}')$.*

- Feedforward + Self-attention:

$$\begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix} \leftarrow \begin{pmatrix} g(\mathbf{a}_i) \\ \boldsymbol{b}_i - \Delta \boldsymbol{X}_{\text{gd}} g(\mathbf{a}_i) \end{pmatrix}$$

where $\Delta \boldsymbol{X}_{\text{gd}} = -\eta \nabla L(\boldsymbol{X}_{\text{gd}}^{(0)}) = -\eta(\nabla \frac{1}{2N} \sum_{i=1}^{N} ||\boldsymbol{X}_{\text{gd}}^{(0)} g(\mathbf{a}_i) - \boldsymbol{b}_i||^2)$ , assumed $\boldsymbol{X}_{\text{gd}}^{(0)} = \mathbf{0}$.

- For the test token $\boldsymbol{b}_{N+1} = \begin{pmatrix} \mathbf{a}_{N+1} \\ 0 \end{pmatrix}$, the prediction after a single Transformer block (multiplying $-1$) is

$$-(0 - \Delta \boldsymbol{X}_{\text{gd}} g(\mathbf{a}_{N+1})) = \sum_{i=1}^{N} b_i \underbrace{g(\mathbf{a}_i)^T g(\mathbf{a}_{N+1})}_{k(\mathbf{a}_i, \mathbf{a}_{N+1})}$$

**Remarks:**
- Only one-step GD.
- If multi-step is considered, then the network structure is: Feedforward -> Attention -> Attention ...
- But in practice: Attention -> Feedforward -> Attention -> Feedforward ...
- The feedforward layer is applied only for $\mathbf{a}_i$ instead of $\boldsymbol{b}_i = \begin{pmatrix} \mathbf{a}_i \\ \boldsymbol{b}_i \end{pmatrix}$.

# *Motivation: Why RLHF?

**Idea: Adapt the model to the prompts, not vice-versa**

- We want to avoid
  - hand-crafting good prompts.
  - explicitly inserting instructions each time.
- The model should adapt to the goal of a prompt, and not vice-versa.
  - This is an inherently interactive process.
- If we have a goal (rewards), RL does this without the need for a hand-designed differentiable objective.
- We thus need
  - an RL algorithm (here, PPO),
  - and a way to learn a reward function from interaction (human feedback).

## *Clarification of the training steps

Step 1
Pre-training
→
Step 2
Supervised fine-tuning
→
Step 3
Train reward model
→
Step 4
Fine-tune via PPO

### Question

Why do we need all steps 2, 3 and 4? Why not without them, or only a subset of them?

## *Clarification of the training steps

Step 1 Pre-training → Step 2 Supervised fine-tuning → Step 3 Train reward model → Step 4 Fine-tune via PPO

**Question**

Why do we need all steps 2, 3 and 4? Why not without them, or only a subset of them?

**Data-driven view:** A machine learning model will only act according to the distribution we have trained it on.

- **Step 1:** Pre-trained model (self-supervised learning)
  - Training distribution: unlabeled corpus to predict the next token given the history.
  - Can hope for: completing coherent sentences, associating a context with what might come next.
  - But: The data distribution of the raw text corpus is not what the distribution over conversations looks like. For example, in a conversation, there is usually a statement and a response.

## *Clarification of the training steps

Step 1 Pre-training → Step 2 Supervised fine-tuning → Step 3 Train reward model → Step 4 Fine-tune via PPO

**Question**

Why do we need all steps 2, 3 and 4? Why not without them, or only a subset of them?

**Data-driven view:** A machine learning model will only act according to the distribution we have trained it on.

- **Step 2:** Supervised fine-tuning:
  - Training distribution: resembles how prompts and answers (labels) should be mapped to each other.
  - Can hope for: having a sensible conversation, not just completing text.
  - But: We want the model to plan ahead what would please a user, based on experience when interacting with them. This is hard to formulate as a differentiable loss function (but possible via rewards).

## ⋆Clarification of the training steps

Step 1 Pre-training → Step 2 Supervised fine-tuning → Step 3 Train reward model → Step 4 Fine-tune via PPO

**Question**

Why do we need all steps 2, 3 and 4? Why not without them, or only a subset of them?

**Data-driven view:** A machine learning model will only act according to the distribution we have trained it on.

- **Step 3:** Learning the rewards from preferences.
  - Training distribution: actual interations with an environment (users) and their feedback.
  - Can hope for: learning a reward function which encodes goals that should be obeyed to please the user.

## *Clarification of the training steps

Step 1 Pre-training → Step 2 Supervised fine-tuning → Step 3 Train reward model → Step 4 Fine-tune via PPO

**Question**

Why do we need all steps 2, 3 and 4? Why not without them, or only a subset of them?

**Data-driven view:** A machine learning model will only act according to the distribution we have trained it on.

- **Step 4:** Policy optimization for further fine-tuning
  - Training distribution: actual interations with an environment (users) and rewards as feedback.
  - Can hope for: having a conversation that users also appreciate, by obeying the goals encoded in the reward.
  - But: To perform RL, we first need to write down a set of rewards.

# State space model

- Recall the RNN training:

| Forward pass in pre-training on single sentence | |
|---|---|
| **1.** Set initial state $\mathbf{z}_0 = \mathbf{0}$, initial loss $L = 0$ | |
| **2.** **For** $t = 1, \ldots, T$ | |
| ▸ $\mathbf{z}_t = \sigma(\mathbf{X}_H \mathbf{z}_{t-1} + \mathbf{X}_I \mathbf{b}_t)$, | RNN |
| ▸ $\boldsymbol{y}_t = \mathbf{X}_O \mathbf{z}_t$ | |
| ▸ $\mathbf{u}_t = \text{Softmax}(\boldsymbol{y}_t)$, | probability |
| ▸ $L+ = \left( \sum_{i=1}^{\lvert\mathcal{V}\rvert} -\hat{\mathbf{u}}_t^{[i]} \log \mathbf{u}_t^{[i]} \right)$, | loss |

- The crucial step that makes the training of RNN unparallelizable is

$$\mathbf{z}_t = \sigma(\mathbf{X}_H \mathbf{z}_{t-1} + \mathbf{X}_I \mathbf{b}_t).$$

- Sequential operations of training are $O(T)$.

# *State space model

- What if we remove the non-linearity?

$$\mathbf{z}_t = \mathbf{X}_H \mathbf{z}_{t-1} + \mathbf{X}_I \mathbf{b}_t .$$

- Then by unwrapping the formula we get:

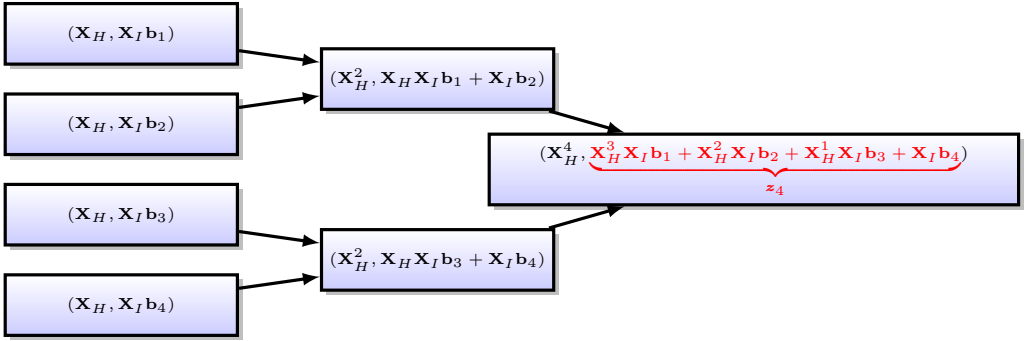
$$\begin{aligned}
\mathbf{z}_0 &= 0, \\
\mathbf{z}_1 &= \mathbf{X}_I \mathbf{b}_1 \\
\mathbf{z}_2 &= \mathbf{X}_H \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_I \mathbf{b}_2 \\
\mathbf{z}_3 &= \mathbf{X}_H^2 \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_H \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_I \mathbf{b}_3 \\
\mathbf{z}_4 &= \mathbf{X}_H^3 \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_H^2 \mathbf{X}_I \mathbf{b}_2 + \mathbf{X}_H \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_I \mathbf{b}_4 \\
&\dots \\
\mathbf{z}_t &= \mathbf{X}_H^{t-1} \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_H^{t-2} \mathbf{X}_I \mathbf{b}_2 + \cdots + \mathbf{X}_I \mathbf{b}_t .
\end{aligned}$$

- As a result

$$\boldsymbol{y}_t = \mathbf{X}_O \left( \mathbf{X}_H^{t-1} \mathbf{X}_I \mathbf{b}_1 + \mathbf{X}_H^{t-2} \mathbf{X}_I \mathbf{b}_2 + \cdots + \mathbf{X}_I \mathbf{b}_t \right) .$$

# ⋆State space model

- Define the arrow operation as: $(m, n), (m, n') \to (m^2, mn + n')$.
- During training, to obtain $z_4$, state space model (SSM) uses FFT's style computation.




- Hence, computing $z_T$ has $O(T \log T)$ complexity.
- $\log T$ indicates the unparallelizable sequential operations.

# $^\star$Complexity analysis

| Network architecture | Sequential operations (training) | Sequential operations (inference) | Maximum path length | Long-sequence memory complexity |
|---|---|---|---|---|
| RNN | $O(T)$ | $O(T)$ | $O(T)$ | $O(1)$ |
| SSM | $O(\log T)$ | $O(T)$ | $O(T)$ | $O(1)$ |
| Self-Attention | $O(1)$ | $O(T)$ | $O(1)$ | $O(T^2)$ |

Table: Complexity comparison of different models. The maximum path length between any two input positions is another metric used to measure the capacity of learning long-range dependencies.

# References I


[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton.
Layer normalization.
*arXiv preprint arXiv:1607.06450*, 2016.
(Cited on page 63.)

[2] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent.
A neural probabilistic language model.
*Advances in neural information processing systems*, 13, 2000.
(Cited on page 21.)

[3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al.
Language models are few-shot learners.
In *Advances in Neural Information Processing Systems*, 2020.
(Cited on pages 38 and 39.)

[4] Bochuan Cao, Yuanpu Cao, Lu Lin, and Jinghui Chen.
Defending against alignment-breaking attacks via robustly aligned llm.
*arXiv preprint arXiv:2309.14348*, 2023.
(Cited on pages 53 and 65.)

## References II


[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al.
Evaluating large language models trained on code.
*arXiv preprint arXiv:2107.03374*, 2021.
(Cited on page 60.)

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al.
Palm: Scaling language modeling with pathways.
*Journal of Machine Learning Research*, 2022.
(Cited on page 64.)

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.
Deep residual learning for image recognition.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
(Cited on pages 33 and 63.)

[8] Md Rabiul Islam, Mohammad Ali Moni, Md Milon Islam, Md Rashed-Al-Mahfuz, Md Saiful Islam, Md Kamrul Hasan, Md Sabir Hossain, Mohiuddin Ahmad, Shahadat Uddin, Akm Azad, et al.
Emotion recognition from eeg signal focusing on deep learning and shallow learning techniques.
*IEEE Access*, 9:94601–94624, 2021.
(Cited on page 12.)

## References III


[9] Dan Jurafsky and James H. Martin.
*Speech and Language Processing (3rd ed. draft)*.
draft, third edition, 2023.
(Cited on pages 7, 9, and 13.)

[10] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa.
Large language models are zero-shot reasoners.
*Advances in neural information processing systems*, 35:22199–22213, 2022.
(Cited on pages 46, 48, and 49.)

[11] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein.
Visualizing the Loss Landscape of Neural Nets.
*arXiv*, Dec 2017.
(Cited on page 63.)

[12] Andrey Andreyevich Markov.
Essai d'une recherche statistique sur le texte du roman.
*Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain')". In: Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg). 6th ser*, 7:153–162, 1913.
(Cited on page 9.)

# References IV


[13] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean.
Efficient estimation of word representations in vector space.
In *International Conference on Learning Representations*, 2013.
(Cited on pages 13, 14, and 15.)

[14] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur.
Recurrent neural network based language model.
In *Interspeech*, 2010.
(Cited on page 22.)

[15] OpenAI.
Gpt-4 technical report.
*Technical report, OpenAI*, 2023.
(Cited on pages 58, 59, and 60.)

[16] Charles Egerton Osgood, George J Suci, and Percy H Tannenbaum.
*The measurement of meaning*.
University of Illinois press, 1957.
(Cited on page 12.)

# References V


[17] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever.
Improving language understanding with unsupervised learning.
*Technical report, OpenAI*, 2018.
(Cited on pages 33, 34, 35, 36, and 37.)

[18] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al.
Language models are unsupervised multitask learners.
*OpenAI blog*, 2019.
(Cited on pages 38 and 39.)

[19] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo.
Are emergent abilities of large language models a mirage?
In *NeurIPS*, 2023.
(Cited on page 51.)

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.
Proximal policy optimization algorithms.
*arXiv preprint arXiv:1707.06347*, 2017.
(Cited on page 57.)

# References VI


[21] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu.
Roformer: Enhanced transformer with rotary position embedding.
*Neurocomputing*, page 127063, 2023.
(Cited on pages 30 and 64.)

[22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al.
Lamda: Language models for dialog applications.
*arXiv preprint arXiv:2201.08239*, 2022.
(Cited on page 47.)

[23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al.
Llama 2: Open foundation and fine-tuned chat models.
*arXiv preprint arXiv:2307.09288*, 2023.
(Cited on page 64.)

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In *Advances in Neural Information Processing Systems*, 2017.
(Cited on pages 29, 30, and 33.)

# References VII


[25] Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov.
Transformers learn in-context by gradient descent.
In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.
(Cited on pages 44, 67, 68, 69, 70, 71, 72, 73, and 74.)

[26] Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le.
Finetuned language models are zero-shot learners.
In *International Conference on Learning Representations*, 2022.
(Cited on page 52.)

[27] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al.
Emergent abilities of large language models.
*Transactions on Machine Learning Research*, 2022.
(Cited on pages 45 and 47.)

[28] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin.
Understanding and improving layer normalization.
*Advances in Neural Information Processing Systems*, 32, 2019.
(Cited on page 63.)

# References VIII

[29] Beitong Zhou, Cheng Cheng, Guijun Ma, and Yong Zhang.
Remaining useful life prediction of lithium-ion battery based on attention mechanism with positional encoding.
In *IOP Conference Series: Materials Science and Engineering*, volume 895, page 012006. IOP Publishing, 2020.
(Cited on pages 29 and 30.)

[30] Andy Zou, Zifan Wang, J Zico Kolter, and Matt Fredrikson.
Universal and transferable adversarial attacks on aligned language models.
*arXiv preprint arXiv:2307.15043*, 2023.
(Cited on page 66.)