Language Model Pretraining, Fine-Tuning & Decoding

Slido: https://app.sli.do/event/6B2AZpesRh69naQSbun3oX

Yu Meng
University of Virginia
yumeng5@virginia.edu

Oct 20, 2025

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling & Recurrent Neural Networks (RNNs)
- Week 6: Language Modeling with Transformers
- Week 8: Transformer and Pretraining
- Week 9: Large Language Models (LLMs) & In-context Learning
- Week 10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Reinforcement Learning for LLM Post-Training
- Week 13: LLM Agents + Course Summary
- Week 15 (after Thanksgiving): Project Presentations

Reminder

- Midterm report due today 11:59pm! (guideline on Canvas)

(Recap) Tokenization: Overview

- Tokenization: splitting a string into a sequence of tokens
- Simple approach: use whitespaces to segment the sequence
  - One token = one word
  - We have been using “tokens” and “words” interchangeably
- However, segmentation using whitespaces is not the approach used in modern large language models

> Multiple models, each with different capabilities and price points. Prices can be viewed in units of either per 1M or 1K tokens. You can think of tokens as pieces of words, where 1,000 tokens is about 750 words.

(Recap) Limitation of Word-Based Segmentation

- Out-of-vocabulary (OOV) issues:
  - Cannot handle words never seen in our training data
  - Reserving an [UNK] token for unseen words is a remedy
- Subword information:
  - Loses subword information valuable for understanding word meaning and structure
  - Example: “unhappiness” -> “un” + “happy” + “ness”
- Data sparsity and exploded vocabulary size:
  - Require a large vocabulary (vocabulary size = number of unique words)
  - The model sees fewer examples of each word (harder to generalize)

(Recap) Single-Character Segmentation?

- How about segmenting sequences by character?
  - No OOV issue
  - Small vocabulary size
- Increased sequence length:
  - Significantly increases the length of input sequences
  - Transformer's self-attention has quadratic complexity w.r.t. sequence length!
- Loss of word-level semantics:
  - Characters alone often don't carry semantic meaning/linguistic patterns

(Recap) Subword Tokenization

- Strike a balance between character-level and word-level tokenization
  - Capture meaningful subword semantics
  - Handle out-of-vocabulary words better
  - Efficient sequence modeling
- Three common algorithms:
  - Byte-Pair Encoding (BPE): Sennrich et al. (2016)
  - WordPiece: Schuster and Nakajima (2012)
  - SentencePiece: Kudo and Richardson (2018)
- Subword tokenization usually consists of two parts:
  - A token learner that takes a raw training corpus and induces a **vocabulary** (a set of tokens)
  - A token segmenter that takes a raw sentence and **tokenizes** it according to that vocabulary

(Recap) Byte-Pair Encoding (BPE) Overview

- BPE is the most commonly used tokenization algorithm in modern LLMs
- Intuition: start with a character-level vocabulary and iteratively merge the most frequent pairs of tokens
- **Initialization**: let vocabulary be the set of all individual characters: {A, B, C, D, ..., a, b, c, d, ...}
- **Frequency counting**: count all adjacent symbol pairs (could be a single character or a previously merged pair) in the training corpus
- **Pair merging**: merge the most frequent pair of symbols (e.g. ‘t’, ‘h’ => “th”)
- **Update corpus**: replace all instances of the merged pair in the corpus with the new token & update the frequency of pairs
- **Repeat**: repeat the process of counting, merging, and updating until a predefined number of merges (or vocabulary size) is reached

(Recap) BPE: Token Learner

Token learner of BPE

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Figure source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE Example

Suppose we have the following corpus

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

Special “end-of-word” character (distinguish between subword units vs. whole word)

vocabulary

_, d, e, i, l, n, o, r, s, t, w

Example source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE: Counting & Merging

The adjacent symbol pair with the highest frequency is "**er**" (count = 9)

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

Merge "er"

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

Example source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE: Counting & Merging

The adjacent symbol pair with the highest frequency is "**er_**" (count = 9)

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

corpus

5 l o w _
2 l o w e s t _
6 n e w er _
3 w i d er _
2 n e w _

Merge "er_"

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

Example source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE: Counting & Merging

The adjacent symbol pair with the highest frequency is "**ne**" (count = 8)

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

corpus

5 l o w _
2 l o w e s t _
6 n e w er_
3 w i d er_
2 n e w _

Merge "ne"

corpus

5 l o w _
2 l o w e s t _
6 ne w er_
3 w i d er_
2 ne w _

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_

vocabulary

_, d, e, i, l, n, o, r, s, t, w, er, er_, ne

Example source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE: Counting & Merging

Continue the process to merge more adjacent symbols

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

corpus	
5	l o w _
2	l o w e s t _
6	n e w er_
3	w i d er_
2	n e w _

merge	current vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er_)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer_, low_

Example source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

(Recap) BPE: Token Segmenter

- Once we learn our vocabulary, we need a token segmenter to tokenize an unseen sentence (from test set)
- Just run (greedily based on training data frequency) on the merge rules we have learned from the training data on the test data
- Example:
  - Assume the merge rules: [(e, r), (er, _), (n, e), (ne, w), (l, o), (lo, w), (new, er_), (low, _)]
  - First merge all adjacent “er”, then all adjacent “er_”, then all adjacent “ne”…
  - “newer_” from the test set will be tokenized as a whole word
  - “lower_” from the test set will be tokenized as “low” + “er_”

low low low low low lowest lowest newer newer newer
newer newer newer wider wider wider new new

“lower_” is an unseen word from the training set

(Recap) Transformer Block

- Modules in Transformer layers:
  - Multi-head attention
  - Layer normalization (LayerNorm)
  - Feedforward network (FFN)
  - Residual connection

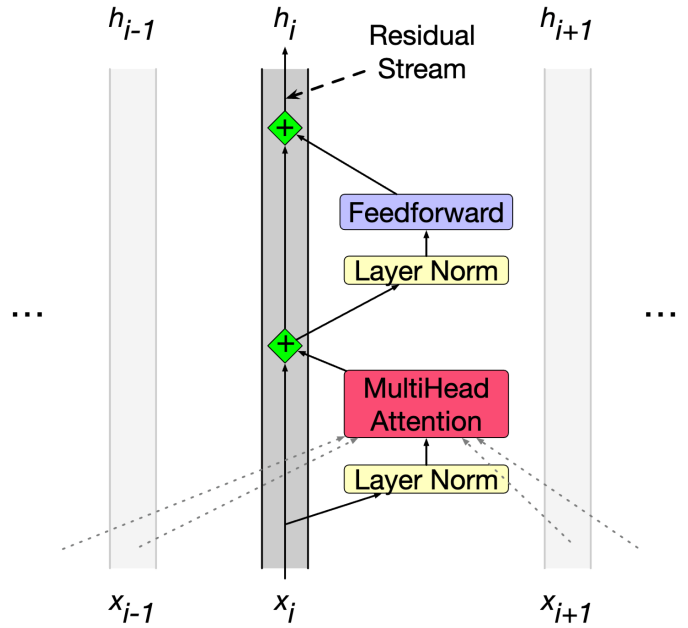


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) Layer Normalization: Motivation

- Proposed in Ba et al. (2016)
- The distribution of inputs to DNN can change during training – “internal covariate shift”
- Slow down the training process: the model constantly adapts to changing distributions

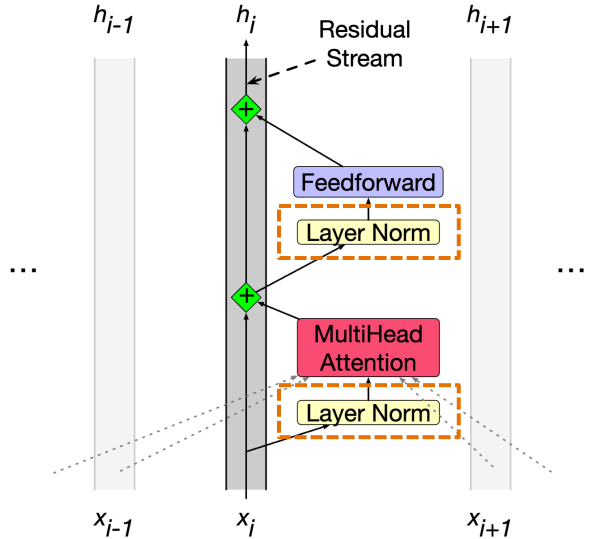


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) Layer Normalization: Solution

- Normalize the input vector $\boldsymbol{x}$
  - Calculate the mean & standard deviation over the input vector dimensions

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i \qquad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

  - Apply normalization

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \mu}{\sigma}$$

- Learn to scale and shift the normalized output with parameters

$$\text{LayerNorm}(\boldsymbol{x}) = \gamma\frac{\boldsymbol{x} - \mu}{\sigma} + \beta$$

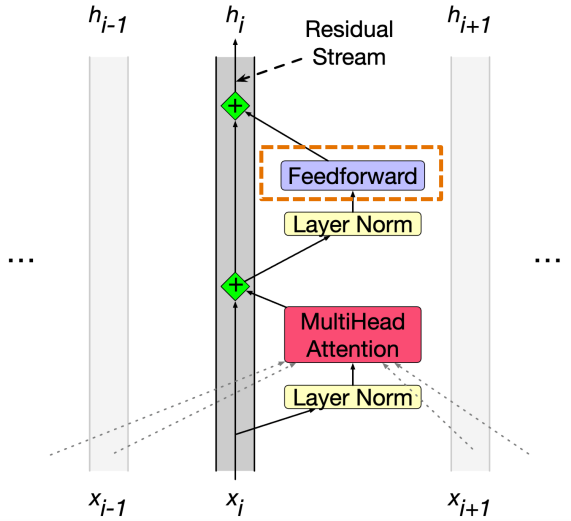
Learnable parameters

(Recap) Feedforward Network (FFN)

- FFN in Transformer is a 2-layer network (one hidden layer, two weight matrices)

$$\mathrm{FFN}(\boldsymbol{x}_i) = \mathrm{ReLU}(\boldsymbol{x}_i \boldsymbol{W}_1)\boldsymbol{W}_2$$

- Apply non-linear activation after the first layer
- Same weights applied to every token
- Weights are different across different Transformer layers

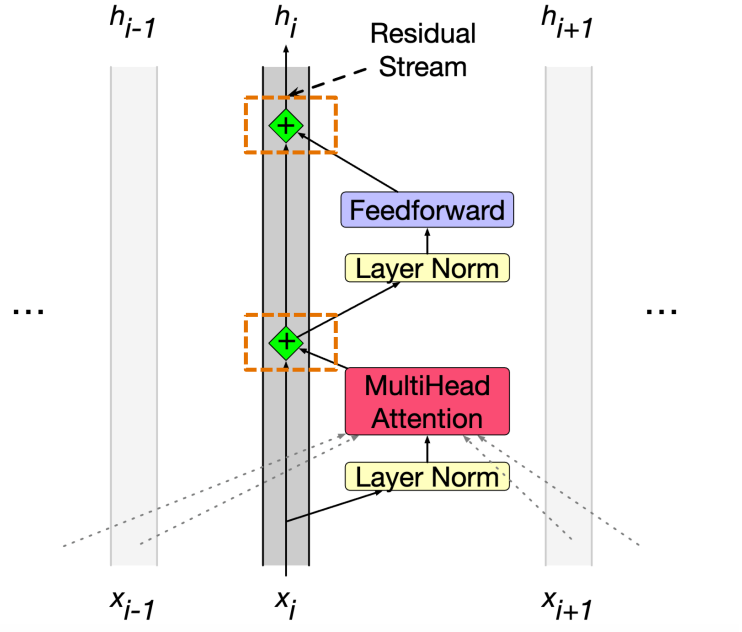


(Recap) Residual Connections

- Add the original input to the output of a sublayer (e.g., attention/FFN)

$$\boldsymbol{y} = \boldsymbol{x} + f(\boldsymbol{x})$$

- Benefits
  - Address the vanishing gradient problem
  - Facilitate information flow across the network
  - Help scale up model



(Recap) Language Model Head

- Language model head is added to the final layer
- Usually apply the weight tying trick (share weights between input embeddings and the output embeddings)

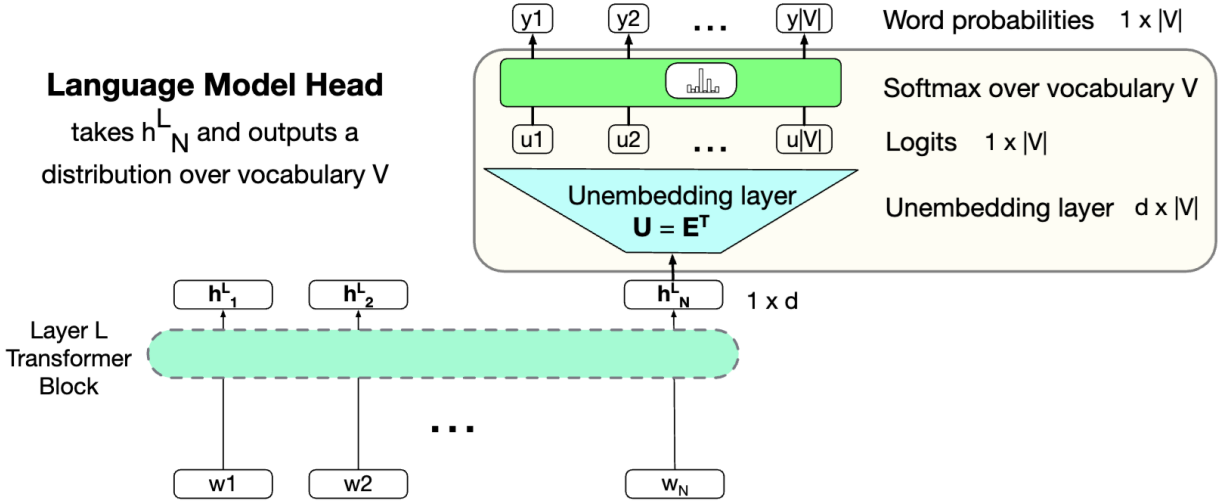


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) Transformer Language Model: Overview

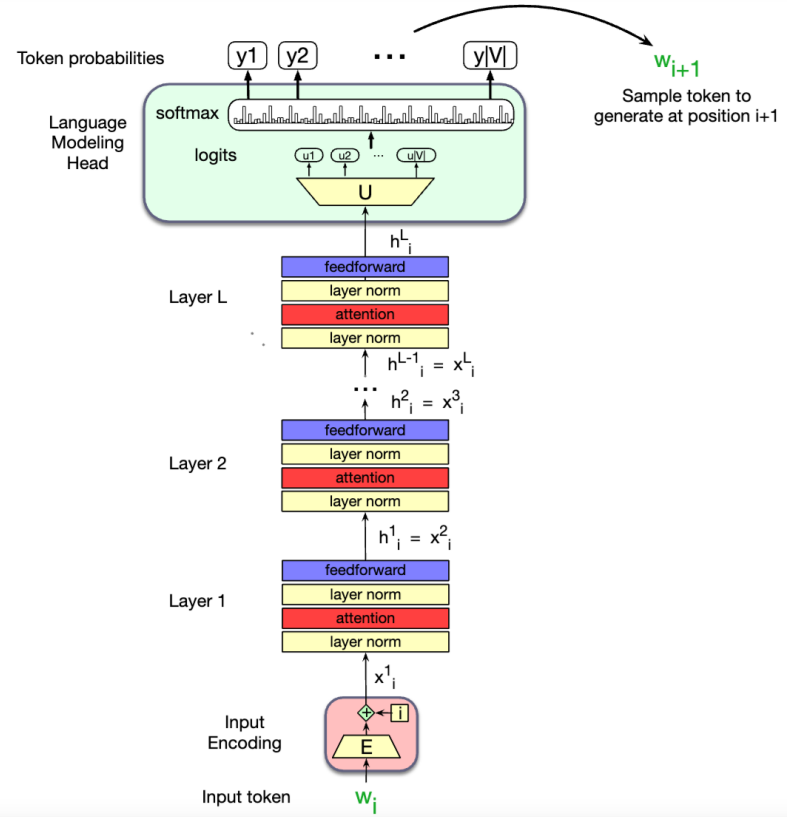


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

(Recap) Transformer Language Model Training

Use cross-entropy loss to train Transformers for language modeling (like RNN LMs)

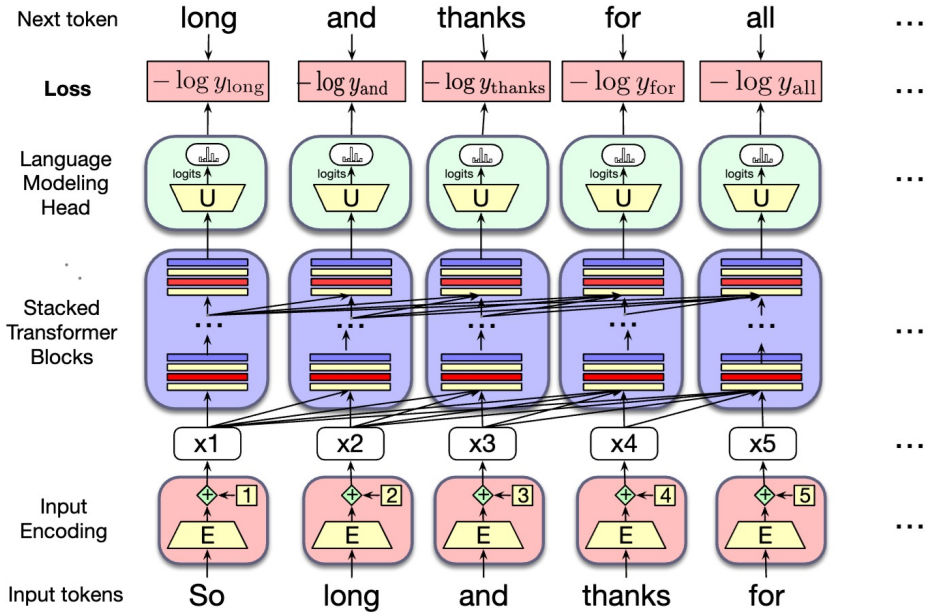


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Agenda

- Language Model Pretraining: Overview
- Pretraining for Different Transformer Architectures
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation

Pretraining: Motivation

- Before pretraining became prevalent in NLP, most NLP models were trained from scratch on downstream task data
- **Data scarcity**: many NLP tasks do not have large labeled datasets available (costly to obtain)
- **Poor generalization**: models trained from scratch on specific tasks do not generalize well to unseen data or other tasks
- **Sensitivity to noise and randomness**: models are more likely to learn spurious correlations or be affected by annotation errors/randomness in training

Pretraining: Motivation

- There are abundant text data on the web, with rich information of linguistic features and knowledge about the world
- Learning from these easy-to-obtain data greatly benefits various downstream tasks

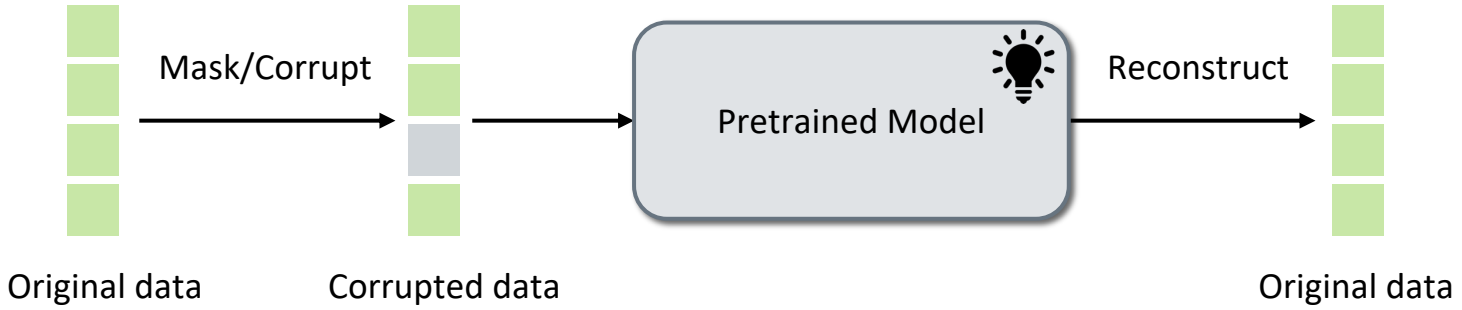
Pretraining: Multi-Task Learning

- In my free time, I like to **{run, banana}** (*Grammar*)
- I went to the zoo to see giraffes, lions, and **{zebras, spoon}** (*Lexical semantics*)
- The capital of Denmark is **{Copenhagen, London}** (*World knowledge*)
- I was engaged and on the edge of my seat the whole time. The movie was **{good, bad}** (*Sentiment analysis*)
- The word for "pretty" in Spanish is **{bonita, hola}** (*Translation*)
- 3 + 8 + 4 = **{15, 11}** (*Math*)
- …

Examples from: https://docs.google.com/presentation/d/1hQUd3pF8_2Gr2Obc89LKjmHL0DlH-uof9M0yFVd3FA4/edit#slide=id.g28e2e9aa709_0_1

Pretraining: Self-Supervised Learning

- Pretraining is a form of **self-supervised** learning
- Make a part of the input unknown to the model
- Use other parts of the input to reconstruct/predict the unknown part



No Human Supervision Needed!

Pretraining + Fine-Tuning

- Pretraining: trained with pretext tasks on large-scale text corpora
- Fine-tuning (continue training): adjust the pretrained model's parameters with fine-tuning data
- Fine-tuning data can have different forms:
  - Task-specific labeled data (e.g., sentiment classification, named entity recognition)
  - (Multi-turn) dialogue data (i.e., instruction tuning)

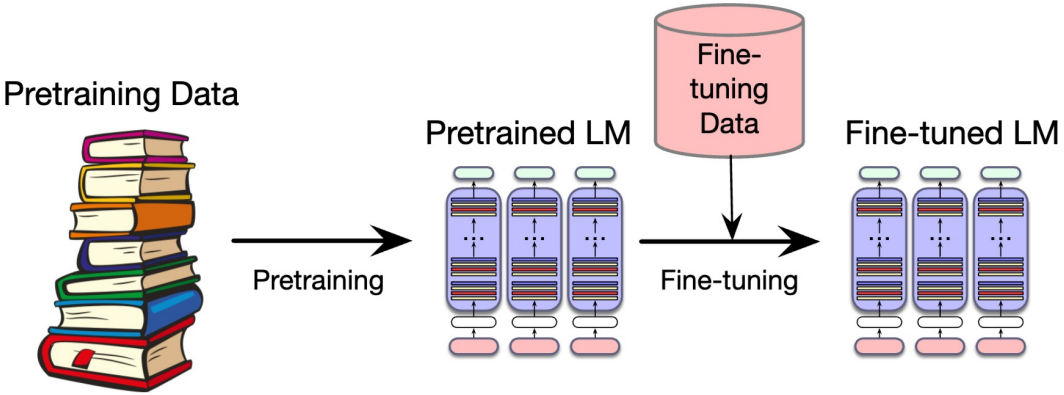


Figure source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Transformer for Pretraining

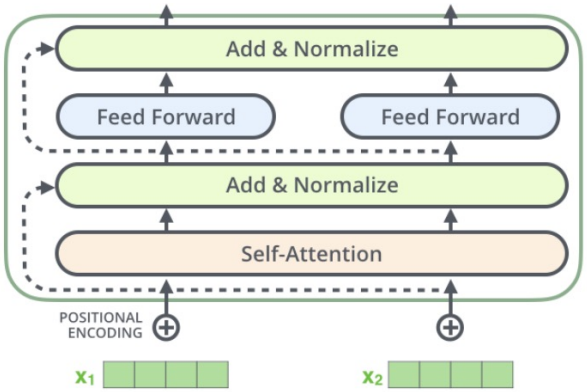
- Transformer is the common backbone architecture for language model pretraining
- **Efficiency**: Transformer processes all tokens in a sequence simultaneously – fast and efficient to train, especially on large datasets
- **Scalability**: Transformer architectures have shown impressive scaling properties, with performance improving as model size and training data increase (more on this later!)
- **Versatility**: Transformer can be adapted for various tasks and modalities beyond just text, including vision, audio, and other multimodal applications

Agenda

- Language Model Pretraining: Overview
- Pretraining for Different Transformer Architectures
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation

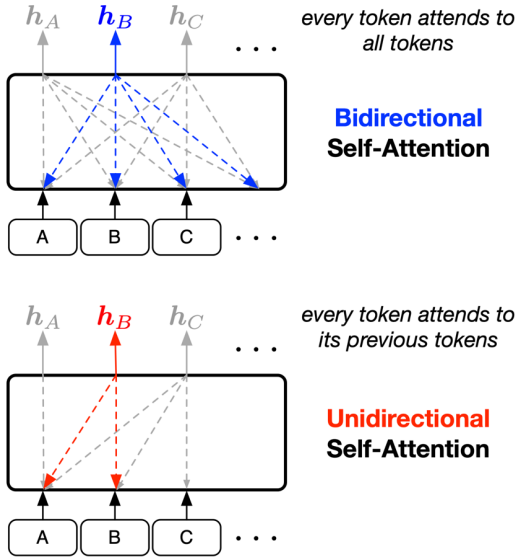
Transformer Architectures

- Based on the type of self-attention, Transformer can be instantiated as
  - Encoder: Bidirectional self-attention
  - Decoder: Unidirectional self-attention



Encoder

Decoder



N

q1·k1	q1·k2	q1·k3	q1·k4
q2·k1	q2·k2	q2·k3	q2·k4
q3·k1	q3·k2	q3·k3	q3·k4
q4·k1	q4·k2	q4·k3	q4·k4

N

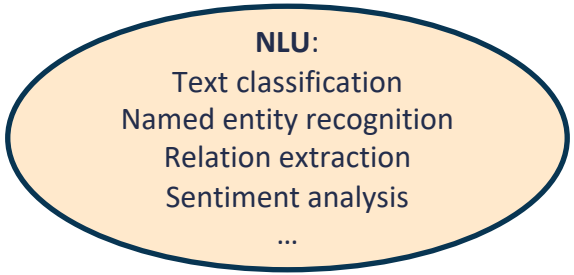
N

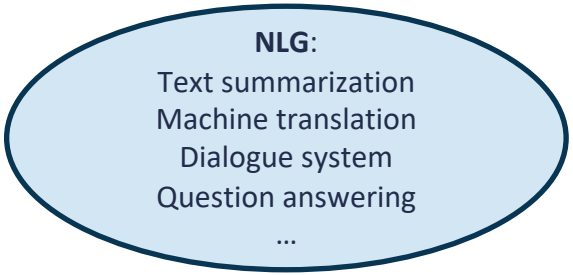
q1·k1	−∞	−∞	−∞
q2·k1	q2·k2	−∞	−∞
q3·k1	q3·k2	q3·k3	−∞
q4·k1	q4·k2	q4·k3	q4·k4

N

Applications of Transformer Architectures

- Encoder (e.g., BERT):
  - Capture bidirectional context to learn each token representations
  - Suitable for natural language understanding (NLU) tasks
- Decoder (modern large language models, e.g., GPT):
  - Use prior context to predict the next token (conventional language modeling)
  - Suitable for natural language generation (NLG) tasks
  - Can also be used for NLU tasks by generating the class labels as tokens





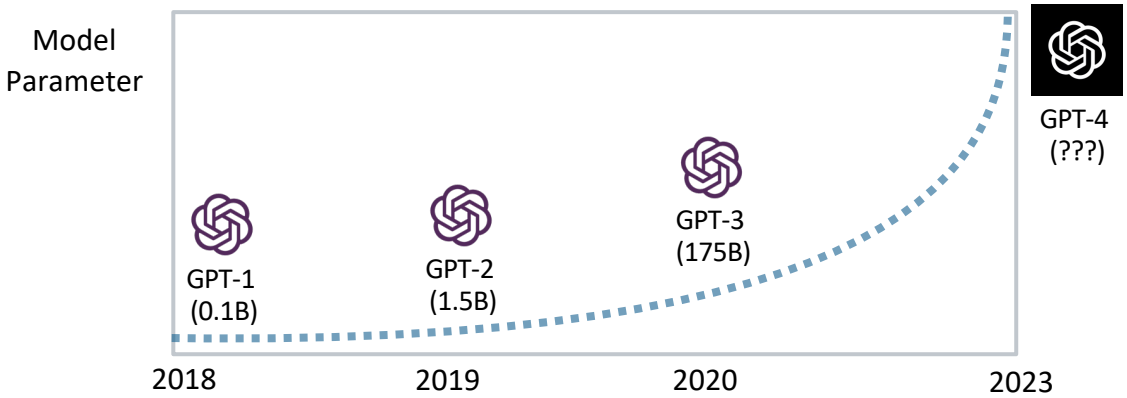
Decoder Pretraining

- Decoder architecture is the prominent choice in large language models
- Pretraining decoders is first introduced in GPT (generative pretraining) models
- Follow the standard language modeling (cross-entropy) objective

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(x_i | x_1, x_2, \ldots, x_{i-1})$$

GPT Series

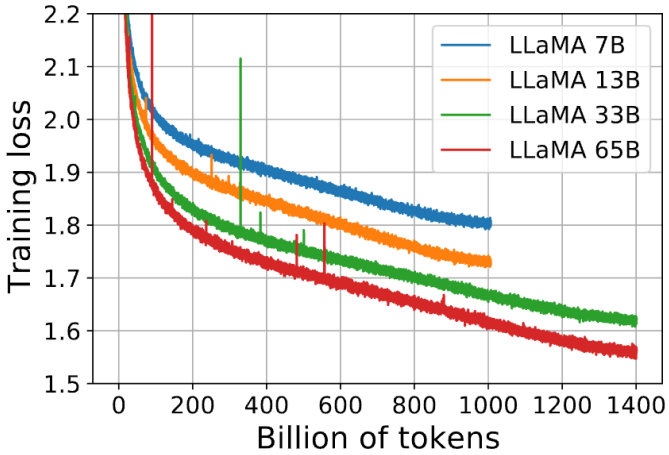
- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



Papers: (GPT-1) https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
(GPT-2) https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
(GPT-3) https://arxiv.org/pdf/2005.14165.pdf

Llama Series

- Llama-1 (2023/02): 7B/13B/33B/65B
- Llama-2 (2023/07): 7B/13B/70B
- Llama-3 (3.1 & 3.2) (2024/07): 1B/3B/8B/70B/405B w/ multi-modality



Larger models learn pretraining data better

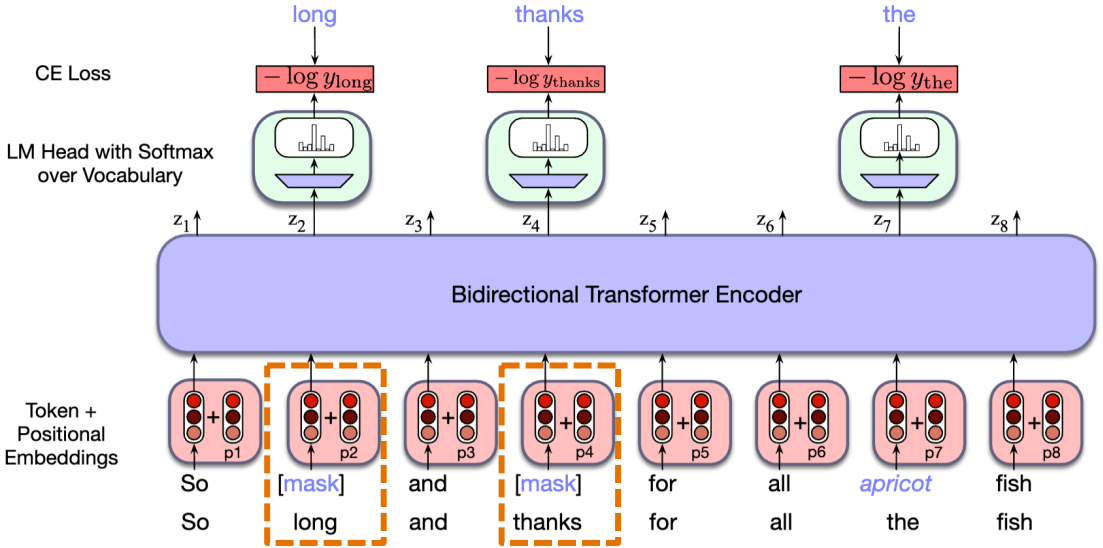
Papers: (Llama-1) https://arxiv.org/pdf/2302.13971
(Llama-2) https://arxiv.org/pdf/2307.09288
(Llama-3) https://arxiv.org/pdf/2407.21783

Further Reading on Decoder LMs

- Mistral 7B [Jiang et al., 2023]
- Qwen Technical Report [Bai et al., 2023]
- GPT-4 Technical Report [OpenAI, 2023]
- Gemma: Open Models Based on Gemini Research and Technology [Gemma Team, 2024]

Encoder Pretraining: BERT

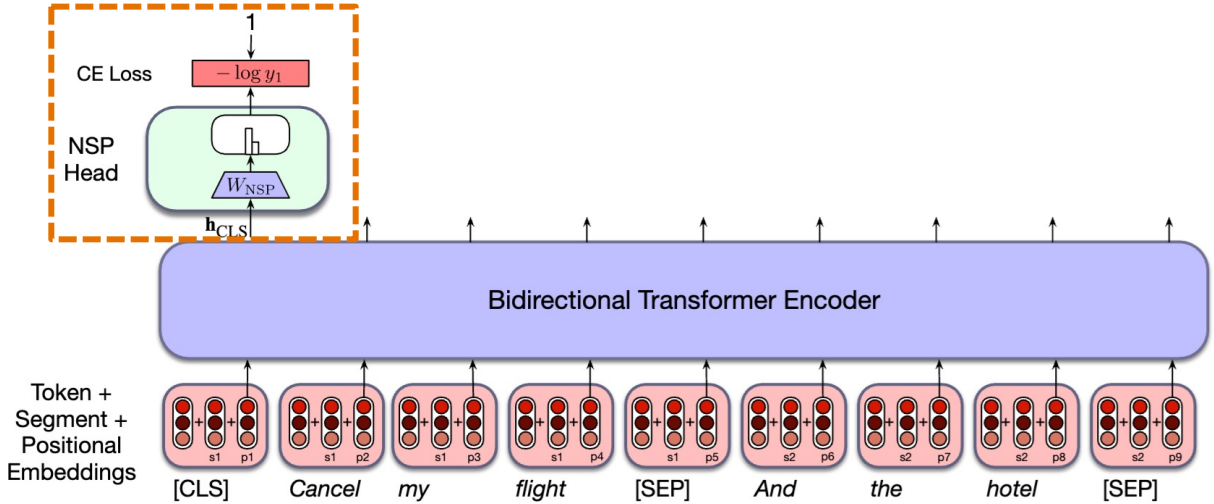
- BERT pretrains encoder models with bidirectionality
- **Masked language modeling** (MLM): With 15% words randomly masked, the model learns bidirectional contextual information to predict the masked words



BERT: https://arxiv.org/pdf/1810.04805.pdf Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

Encoder Pretraining: BERT

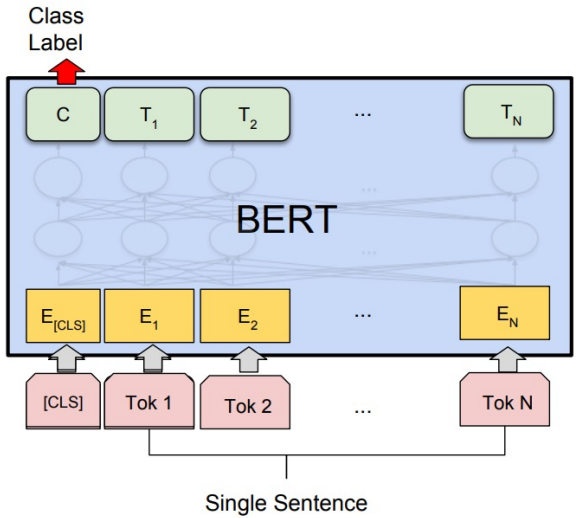
- **Next sentence prediction** (NSP): the model is presented with pairs of sentences
- The model is trained to predict whether each pair consists of an actual pair of adjacent sentences from the training corpus or a pair of unrelated sentences



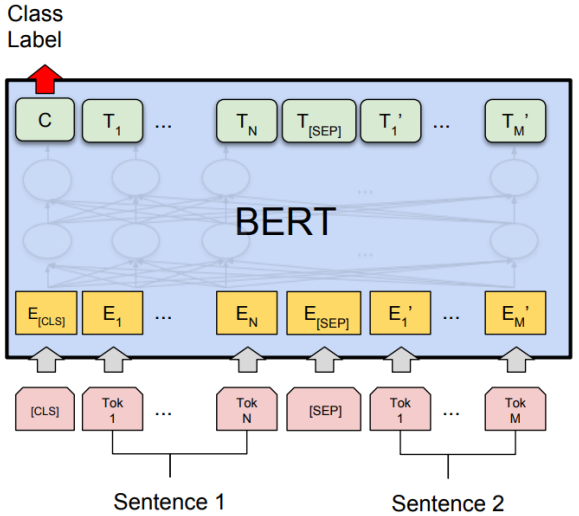
BERT: https://arxiv.org/pdf/1810.04805.pdf Figure source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

BERT Fine-Tuning

- Fine-tuning pretrained BERT models takes different forms depending on task types
- Usually replace the LM head with a linear layer fine-tuned on task-specific data



Single sequence classification



Sequence-pair classification

BERT vs. GPT on NLU tasks

- BERT outperforms GPT-1 on a set of NLU tasks
- Encoders capture **bidirectional** contexts – build a richer understanding of the text by looking at both preceding and following words
- Are encoder models still better than state-of-the-art (large) decoder models?
  - LLMs can be as good as (if not better than) encoder models on NLU: Can ChatGPT Understand Too?
  - The sheer model size + massive amount of pretraining data compensate for LLMs' unidirectional processing

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

BERT Variant I: RoBERTa

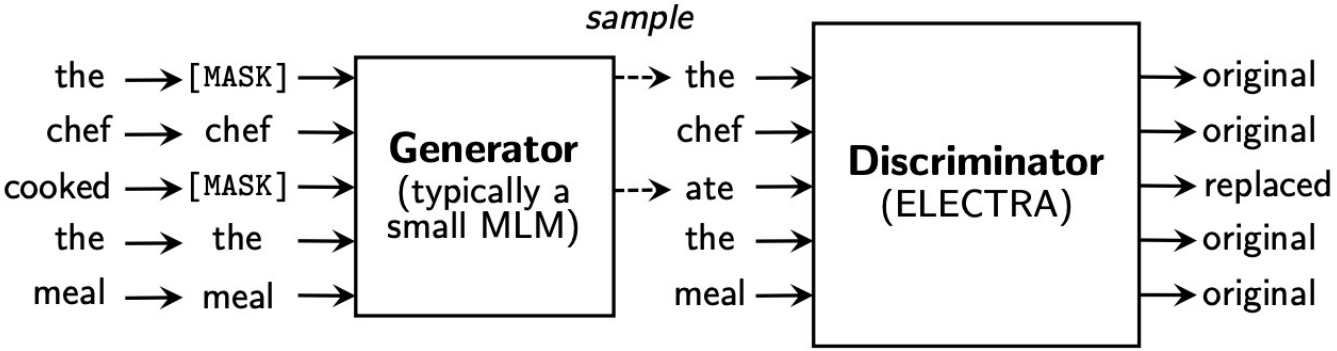
- Pretrain the model for longer, with bigger batches over more data
- Pretrain on longer sequences
- Dynamically change the masking patterns applied to the training data in each epoch

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

RoBERTa: https://arxiv.org/pdf/1907.11692.pdf

BERT Variant II: ELECTRA

- Use a small MLM model as an auxiliary generator (discarded after pretraining)
- Pretrain the main model as a discriminator
- The small auxiliary MLM and the main discriminator are jointly trained
- The main model's pretraining task becomes more and more challenging in pretraining
- Major benefits: sample efficiency + learning curriculum



ELECTRA: https://arxiv.org/pdf/2003.10555.pdf

ELECTRA Performance

- ELECTRA pretraining incurs lower computation costs compared to MLM
- Better downstream task performance

Model	Train FLOPs	Params	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	Avg.
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	**91.4**	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa-500K	3.2e21 (4.5x)	356M	68.0	96.4	90.9	92.1	92.2	90.2	94.7	86.6	88.9
XLNet	3.9e21 (5.4x)	360M	69.0	**97.0**	90.8	92.2	92.3	90.8	94.9	85.9	89.1
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA-400K	7.1e20 (1x)	335M	**69.3**	96.0	90.6	92.1	**92.4**	90.5	94.5	86.8	89.0
ELECTRA-1.75M	3.1e21 (4.4x)	335M	69.1	96.9	90.8	**92.6**	**92.4**	**90.9**	**95.0**	**88.0**	**89.5**

Further Reading on Encoder LMs

- XLNet: Generalized Autoregressive Pretraining for Language Understanding [Yang et al., 2019]
- ALBERT: A Lite BERT for Self-supervised Learning of Language Representations [Lan et al., 2020]
- DeBERTa: Decoding-enhanced BERT with Disentangled Attention [He et al., 2020]
- COCO-LM: Correcting and Contrasting Text Sequences for Language Model Pretraining [Meng et al. 2021]

Agenda

- Language Model Pretraining: Overview
- Pretraining for Different Transformer Architectures
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation

Prompting

- **Prompt**: initial user input/instructions given to the model to guide text generation
- Example (sentiment analysis):

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

prompt

- Example (question answering):

$P(w|\text{Q: Who wrote the book ‘‘The Origin of Species"? A:})$ prompt

- **Prompting**: directly use trained LMs to generate text given user prompts (no fine-tuning)

For good prompting performance, we need **instruction-tuning** (later lectures)

Example source: https://web.stanford.edu/~jurafsky/slp3/10.pdf

Prompt Engineering

- Some LMs (especially small ones) can be sensitive to specific formats of prompts
- Multiple prompts can make sense for the same task, but the resulting model performance might differ

$P_1(a) =$ It was ____. a $P_2(a) =$ Just ____! $\|$ a

$P_3(a) =$ a. All in all, it was ____.

$P_4(a) =$ a $\|$ In summary, the restaurant is ____.

Model predicts the masked word

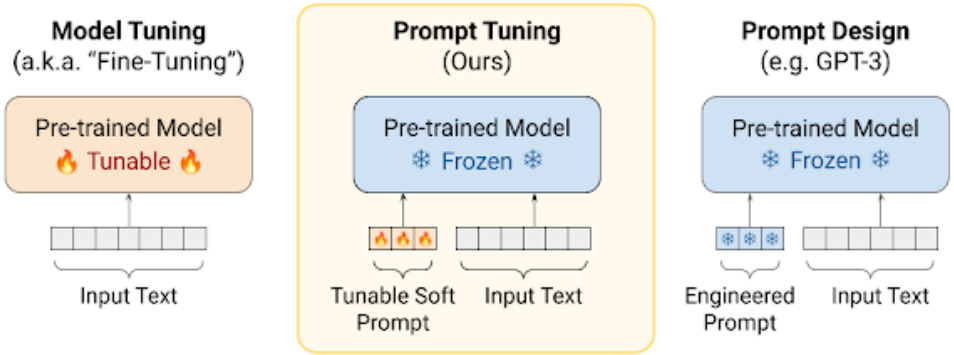
Prompt templates for BERT sentiment classification

- **Prompt engineering**: designing and refining prompts to achieve desired outcomes from LMs (e.g., manually tune on a validation set)
- A guide on prompt engineering: https://www.promptingguide.ai/

Figure source: https://aclanthology.org/2021.eacl-main.20.pdf

Prompt Tuning

- **Prompt tuning**: instead of manually testing the prompt design, consider prompt tokens as learnable model parameters ("soft prompts")
- Optimize a small amount of prompt token embeddings while keeping the LM frozen



- Prompt tuning is a parameter efficient fine-tuning (PEFT) method

Figure source: https://www.googblogs.com/guiding-frozen-language-models-with-learned-soft-prompts/

Parameter Efficient Fine-tuning (PEFT)

- Fine-tuning all model parameters is expensive

Pretrained weight
(can represent any module)

$$\boldsymbol{W}_0 \in \mathbb{R}^{d \times d}$$

Fine-tuned weight

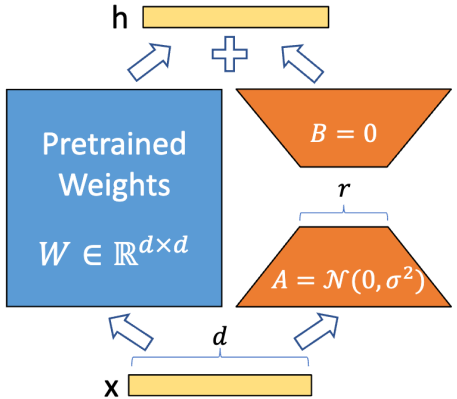
$$\boldsymbol{W}^* = \boldsymbol{W}_0 + \Delta \boldsymbol{W}, \quad \Delta \boldsymbol{W} \in \mathbb{R}^{d \times d}$$

- Can we update only a small number of model parameters on fine-tuning data?

Parameter Efficient Fine-tuning: LoRA

- Assume the parameter update is **low-rank**
  - **Overparameterization**: large language models typically have many more parameters than strictly necessary to fit the training data
  - **Empirical observation**: parameter updates in neural networks tend to be low-rank in practice
- Solution: approximate weight updates with low-rank factorization

Freeze pretrained weights



$$\Delta \boldsymbol{W} \approx \boldsymbol{B}\boldsymbol{A}, \quad \boldsymbol{B} \in \mathbb{R}^{d \times r}, \quad \boldsymbol{A} \in \mathbb{R}^{r \times d}, \quad r \ll d$$

Low-rank approximation

LoRA: https://arxiv.org/pdf/2106.09685

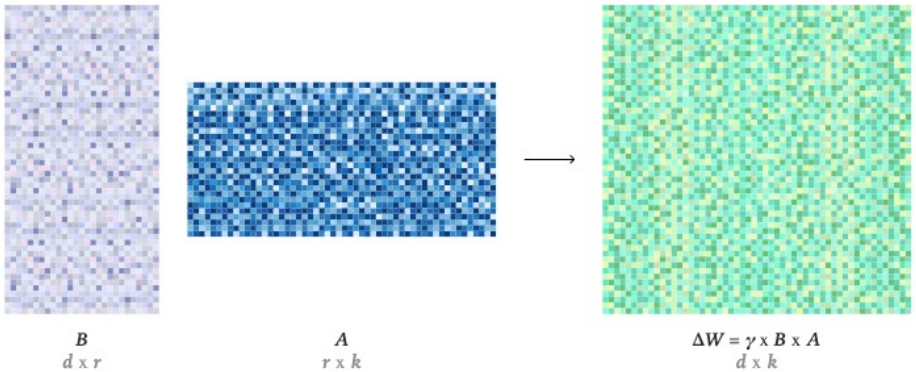
LoRA in Frontier Research

LoRA fine-tuning is effective for frontier LLM post-training (e.g., reinforcement learning)

LoRA Without Regret

John Schulman in collaboration with others at Thinking Machines

Sep 29, 2025



Blog post: https://thinkingmachines.ai/blog/lora/

Further Reading on PEFT

- Parameter-Efficient Transfer Learning for NLP [Houlsby et al., 2019]
- Prefix-Tuning: Optimizing Continuous Prompts for Generation [Li & Liang, 2021]
- The Power of Scale for Parameter-Efficient Prompt Tuning [Lester et al., 2021]
- GPT Understands, Too [Liu et al., 2021]

Agenda

- Language Model Pretraining: Overview
- Pretraining for Different Transformer Architectures
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation

Large Language Models (LLMs)

- The field of LLMs is rapidly evolving!
  - In 2018, BERT-large with 340 million parameters was considered large
  - In 2019, GPT-2 with 1.5 billion parameters was considered very large
  - In 2020, GPT-3 with 175 billion parameters set a new standard for “large”
- In 2025, how should we define LLMs?
- General definition:
  - Transformer-decoder architecture (or variants) that can generate text
  - Pretrained on vast and diverse general-domain corpora
  - With (at least) billions of parameters
  - General-purpose solvers for a wide range of NLP tasks and beyond

Decoding with LLMs

- **Decoding**: convert Transformer representations into natural language tokens
- Autoregressive decoding typically involves iterative **sampling** from LMs' output distributions, until an [EOS] token is generated

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1}) = \text{softmax}(\boldsymbol{U}\boldsymbol{h}_{i-1}) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1})}, \ldots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1})}\right]$$

Model parameters — Unembedding matrix — Hidden states at token $i-1$

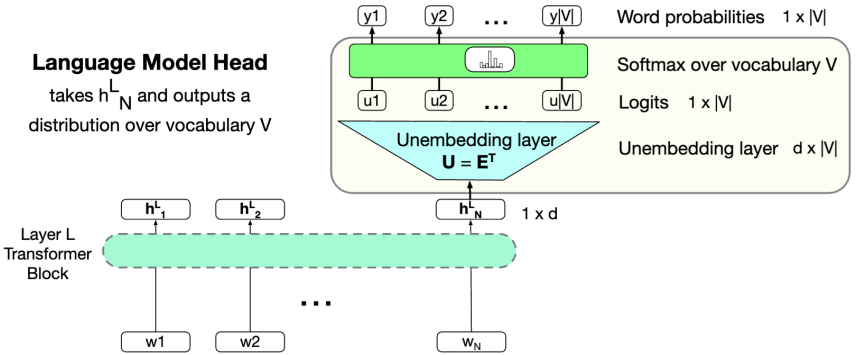


Figure source: https://web.stanford.edu/~jurafsky/slp3/8.pdf

Greedy Decoding

- Always pick the token with the highest probability estimated by the LM for every step

$$x_i \leftarrow \arg\max_w p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1})$$

- Pros:
  - Simplicity: easy to implement and understand
  - Deterministic: guarantee the same output given the same input
  - Efficient: makes only one (simple) decision at each step w/o additional operations
- Cons:
  - Suboptimal solutions: may not find the globally optimal sequence
  - Lack of diversity: cannot produce multiple outputs given the same input

Top-k Sampling

- Motivation: Instead of choosing the single most probable word to generate, sample from the top-k most likely tokens (candidates) – avoid generating low probability tokens
- k is a hyperparameter (typically 5-10)

Compute the probability distribution only over the top-k tokens

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1}) = \text{softmax}(\boldsymbol{U}_{\text{top-}k}\boldsymbol{h}_{i-1}) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{k}\exp(\boldsymbol{u}_{\text{top-}j} \cdot \boldsymbol{h}_{i-1})}, \ldots, \frac{\exp(\boldsymbol{u}_{\text{top-}k} \cdot \boldsymbol{h}_{i-1})}{\sum_{j=1}^{k}\exp(\boldsymbol{u}_{\text{top-}j} \cdot \boldsymbol{h}_{i-1})}\right]$$

Sample from the top-k tokens $x_i \sim p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1})$

- With $k = 1$, top-k sampling is equivalent to greedy decoding

Nucleus (Top-p) sampling

- Top-k sampling does not account for the shape of the probability distribution
  - For the next-token distribution of "the 46th US president Joe", top-k sampling may consider more tokens than necessary
  - For the next-token distribution of "the spacecraft", top-k sampling may consider fewer tokens than necessary
- Nucleus sampling sets cutoff based on the top-p percent of the probability mass
- p is a hyperparameter (typically 0.9)
- Top-p vocabulary is the smallest set of words such that

$$\sum_{w \in \mathcal{V}_{\text{top-p}}} p(w|x_1, x_2, \ldots, x_{i-1}) \geq p$$

- Sample from the top-p vocabulary in a similar way as top-k sampling

Temperature Sampling

- Intuition comes from thermodynamics
  - A system at a high temperature is flexible and can explore many possible states
  - A system at a lower temperature is likely to explore a subset of lower energy (better) states
- Reshape the probability distribution by incorporating a temperature hyperparameter

$$p_{\boldsymbol{\theta}}(w|x_1, x_2, \ldots, x_{i-1}) = \text{softmax}(\boldsymbol{U}\boldsymbol{h}_{i-1}/\tau) = \left[\frac{\exp(\boldsymbol{u}_1 \cdot \boldsymbol{h}_{i-1}/\tau)}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1}/\tau)}, \ldots, \frac{\exp(\boldsymbol{u}_{|\mathcal{V}|} \cdot \boldsymbol{h}_{i-1}/\tau)}{\sum_{j=1}^{|\mathcal{V}|}\exp(\boldsymbol{u}_j \cdot \boldsymbol{h}_{i-1}/\tau)}\right]$$

- With $\tau \to 0$, temperature sampling approaches greedy decoding

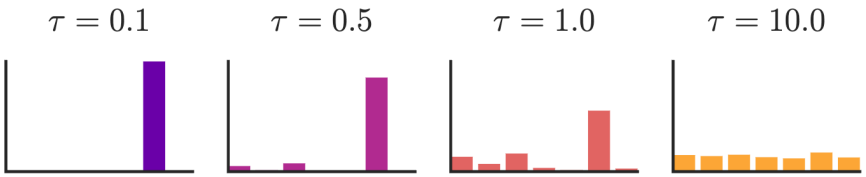


Figure source: https://arxiv.org/pdf/1611.01144v5

Practical Considerations of Decoding Algorithms

- If aiming for simplicity and efficiency without diversity requirements, use greedy decoding
- If multiple responses are required for the same input, use sampling-based decoding
  - Top-p is usually better than Top-k
  - Temperature sampling is commonly used
  - Top-p can be used together with temperature sampling

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu