Introduction to Large Language Models (LLMs)

Yu Meng
University of Virginia
yumeng5@virginia.edu

Oct 16, 2024

Reminder

Join at
slido.com
#3140 184

Midterm report due this Friday! (Guideline: https://docs.google.com/document/d/12-f2KQRH2kYBohxJLj_E6gzfj1vulmnuaEVBbyXBAiY/edit?usp=sharing)

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 and Neural Language Models
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

(Recap) Tokenization

- Segmenting input sequences based on words suffer from several limitations
  - Out-of-vocabulary issue
  - Massive vocabulary size
  - Failure to capture subword information
- Subword tokenization is the common approach to segment input sequences
- Start from single-character vocabulary, iteratively merge adjacent symbols based on frequency in the training set
- Apply the merge rules to test sequences in the order as learned from the training set

(Recap) Transformer

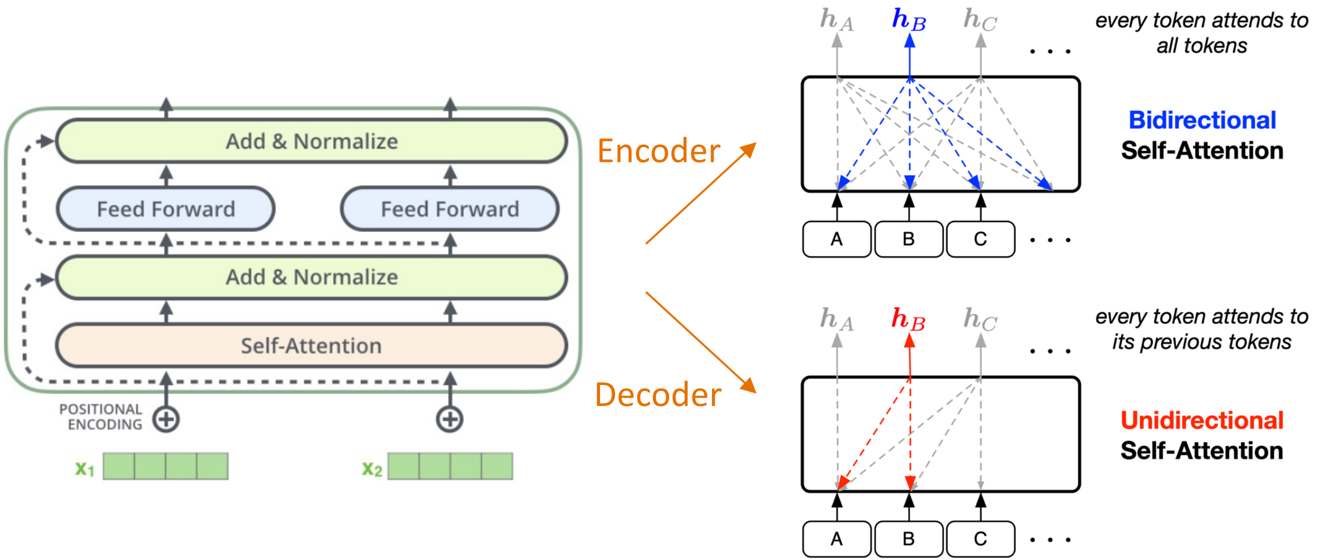
- Transformer is the most commonly-used architecture for language models
- (Multi-head) self-attention
  - Allows every token to directly attend to other tokens in the same input (parallel processing)
  - Can be either bidirectional or unidirectional
  - Quadratic complexity w.r.t. sequence length
- Input embedding
  - Add Token embedding with positional encoding
- Layer normalization
  - Normalize the input across the features to stabilize and speed up training
- Residual connection
  - Add the input of a layer to its output – facilitate information & gradient flow
- Feedforward network
  - Help store factual knowledge

(Recap) Pretraining & Fine-tuning

- Pretraining: train LMs with pretext tasks on large-scale text corpora
  - A form of self-supervised learning – no human supervision needed
  - A form of multi-task learning – learn from diverse domains
  - Different training objectives based on different Transformer architecture
- Fine-tuning: adjust the pretrained model's parameters with fine-tuning data
  - A form of continue training/transfer learning
  - Can use different types of data: task-specific/dialogue annotated data
  - Can apply parameter-efficient techniques (e.g., LoRA) to bring down optimization costs

(Recap) Transformer Architectures

- Based on the type of self-attention, Transformer can be instantiated as
  - Encoder: Bidirectional self-attention
  - Decoder: Unidirectional self-attention
  - Encoder-decoder: Use both encoder and 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	$-\infty$	$-\infty$	$-\infty$
q2·k1	q2·k2	$-\infty$	$-\infty$
q3·k1	q3·k2	q3·k3	$-\infty$
q4·k1	q4·k2	q4·k3	q4·k4

N

(Recap) Applications of Different 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
- Encoder-decoder (e.g., BART, T5):
  - Use the encoder to process input, and use the decoder to generate outputs
  - Can conduct all tasks that encoders/decoders can do

NLU:
Text classification
Named entity recognition
Relation extraction
Sentiment analysis
…

NLG:
Text summarization
Machine translation
Dialogue system
Question answering
…

(Recap) Decoder Pretraining & Fine-tuning

- 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

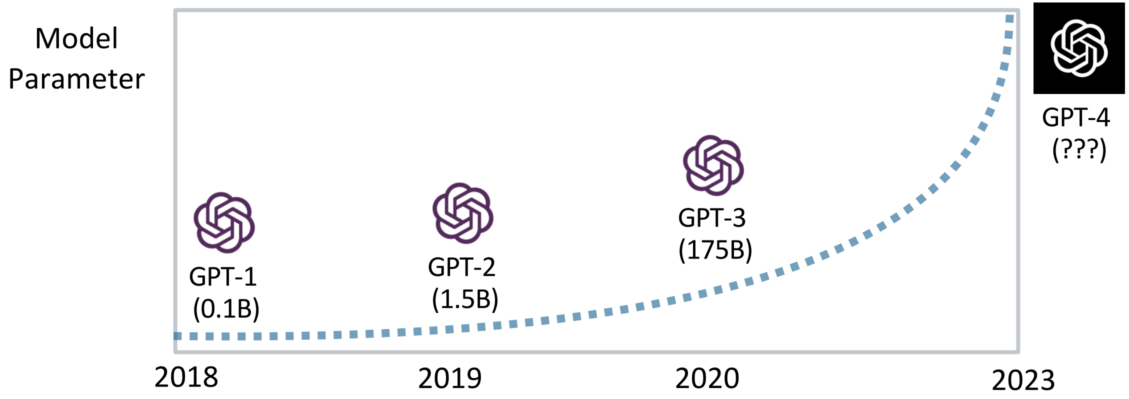
$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(x_i | x_1, x_2, \ldots, x_{i-1})$$

- Fine-tuning decoder is straightforward: apply the same cross-entropy loss to fine-tuning data

Join at
slido.com
#3140 184

(Recap) GPT Series

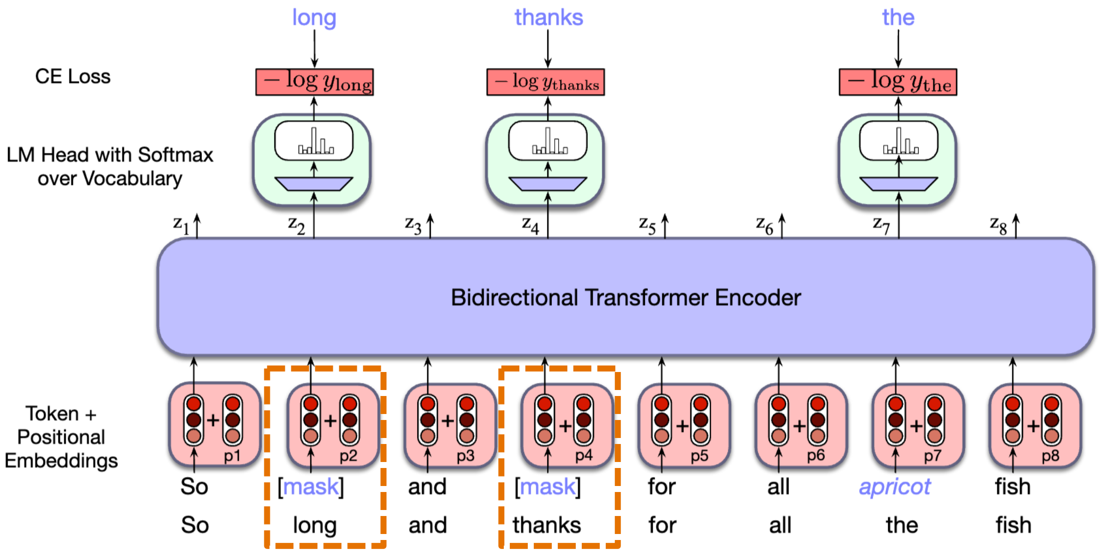
- GPT-1 (2018): 12 layers, 117M parameters, trained in ~1 week
- GPT-2 (2019): 48 layers, 1.5B parameters, trained in ~1 month
- GPT-3 (2020): 96 layers, 175B parameters, trained in several months



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

(Recap) 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/slp3/11.pdf

Agenda

- Encoder-decoder Pretraining (Continued)
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning

UNIVERSITY of VIRGINIA

Join at slido.com #3140 184

Encoder-Decoder Architecture: BART

- Pretraining: Apply a series of noising schemes (e.g., masks, deletions, permutations…) to input sequences and train the model to recover the original sequences
- Fine-tuning:
  - For NLU tasks: Feed the same input into the encoder and decoder, and use the final decoder token for classification
  - For NLG tasks: The encoder takes the input sequence, and the decoder generates outputs autoregressively

BART: https://arxiv.org/pdf/1910.13461.pdf

BART Performance

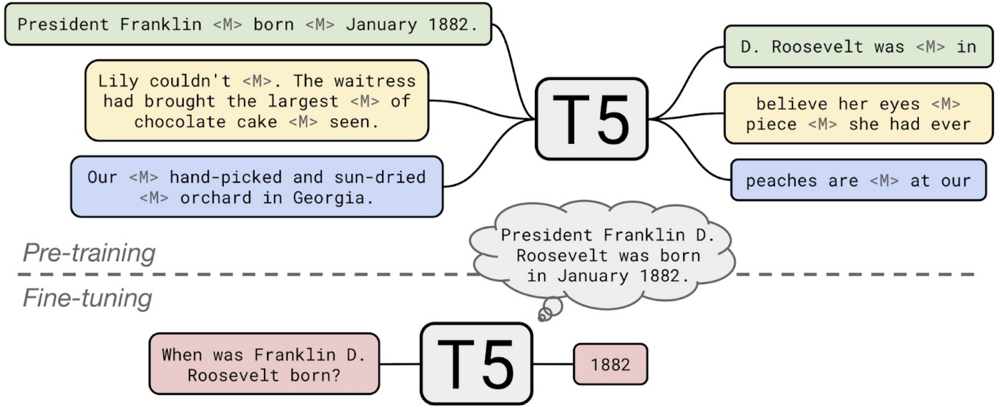
- Comparable to encoders on NLU tasks
- Good performance on NLG tasks

	SQuAD 1.1 EM/F1	SQuAD 2.0 EM/F1	MNLI m/mm	SST Acc	QQP Acc	QNLI Acc	STS-B Acc	RTE Acc	MRPC Acc	CoLA Mcc
BERT	84.1/90.9	79.0/81.8	86.6/-	93.2	91.3	92.3	90.0	70.4	88.0	60.6
UniLM	-/-	80.5/83.4	87.0/85.9	94.5	-	92.7	-	70.9	-	61.1
XLNet	**89.0**/94.5	86.1/88.8	89.8/-	95.6	91.8	93.9	91.8	83.8	89.2	63.6
RoBERTa	88.9/**94.6**	**86.5/89.4**	**90.2/90.2**	96.4	92.2	94.7	**92.4**	86.6	**90.9**	**68.0**
BART	88.8/**94.6**	86.1/89.2	89.9/90.1	**96.6**	**92.5**	**94.9**	91.2	**87.0**	90.4	62.8

	CNN/DailyMail			XSum		
	R1	R2	RL	R1	R2	RL
Lead-3	40.42	17.62	36.67	16.30	1.60	11.95
PTGEN (See et al., 2017)	36.44	15.66	33.42	29.70	9.21	23.24
PTGEN+COV (See et al., 2017)	39.53	17.28	36.38	28.10	8.02	21.72
UniLM	43.33	20.21	40.51	-	-	-
BERTSUMABS (Liu & Lapata, 2019)	41.72	19.39	38.76	38.76	16.33	31.15
BERTSUMEXTABS (Liu & Lapata, 2019)	42.13	19.60	39.18	38.81	16.50	31.27
BART	**44.16**	**21.28**	**40.90**	**45.14**	**22.27**	**37.25**

Encoder-Decoder Architecture: T5

- T5: **T**ext-**to**-**T**ext **T**ransfer **T**ransformer
- Pretraining: Mask out spans of texts; generate the original spans
- Fine-tuning: Convert every task into a sequence-to-sequence generation problem
- We’ll see this model again in the instruction tuning lectures



T5: https://arxiv.org/pdf/1910.10683

UNIVERSITY of VIRGINIA

Join at
slido.com
#3140 184

T5 Performance

- Good performance across various tasks
- T5 vs. BART performance: unclear comparison due to difference in model sizes & training setups

Model	GLUE Average	CoLA Matthew's	SST-2 Accuracy	MRPC F1	MRPC Accuracy	STS-B Pearson	STS-B Spearman
Previous best	89.4[a]	69.2[b]	97.1[a]	**93.6**[b]	**91.5**[b]	92.7[b]	92.3[b]
T5-Small	77.4	41.0	91.8	89.7	86.6	85.6	85.0
T5-Base	82.7	51.1	95.2	90.7	87.5	89.4	88.6
T5-Large	86.4	61.2	96.3	92.4	89.9	89.9	89.2
T5-3B	88.5	67.1	97.4	92.5	90.0	90.6	89.8
T5-11B	**90.3**	**71.6**	**97.5**	92.8	90.4	**93.1**	**92.8**

Model	QQP F1	QQP Accuracy	MNLI-m Accuracy	MNLI-mm Accuracy	QNLI Accuracy	RTE Accuracy	WNLI Accuracy
Previous best	74.8[c]	**90.7**[b]	91.3[a]	91.0[a]	**99.2**[a]	89.2[a]	91.8[a]
T5-Small	70.0	88.0	82.4	82.3	90.3	69.9	69.2
T5-Base	72.6	89.4	87.1	86.2	93.7	80.1	78.8
T5-Large	73.9	89.9	89.9	89.6	94.8	87.2	85.6
T5-3B	74.4	89.7	91.4	91.2	96.3	91.1	89.7
T5-11B	**75.1**	90.6	**92.2**	**91.9**	96.9	**92.8**	**94.5**

Encoder-Decoder vs. Decoder-Only

- Modern LLMs are mostly based on the decoder-only Transformer architecture
- Simplicity:
  - Decoder-only models are simpler in structure (one Transformer model)
  - Encoder-decoder models require two Transformer models
- Efficiency:
  - Decoder-only models are more parameter-efficient for text generation
  - Encoder-decoder models' encoder part does not contribute to generation
- Scalability:
  - Decoder-only models scale very well with increased model size and data
  - Encoder-decoder models do not outperform decoder-only models at large model sizes

Join at
slido.com
#3140 184

Agenda

- Encoder-decoder Pretraining (Continued)
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning

Prompting

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

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

prompt

- Example (question answering):

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

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

Join at
slido.com
#3140 184

- 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

$$P_1(a) = \text{It was } \_\_\_\_.\ a \qquad P_2(a) = \text{Just } \_\_\_\_!\ \|\ a$$

$$P_3(a) = a.\ \text{All in all, it was } \_\_\_\_.$$

$$P_4(a) = a\ \|\ \text{In summary, the restaurant is } \_\_\_\_.$$

Model predicts the masked word

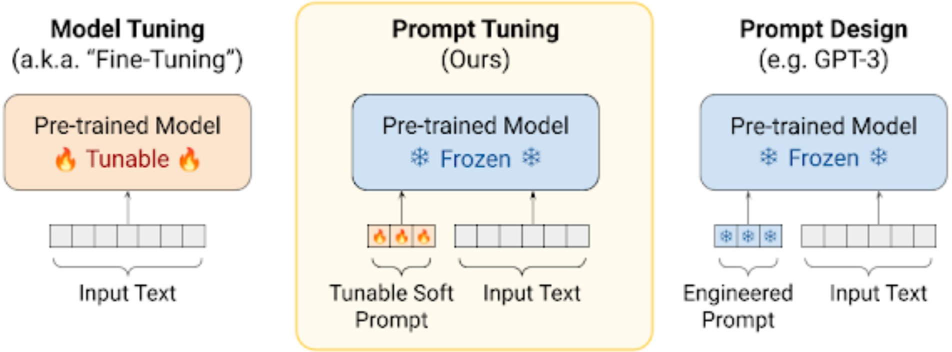
Prompt templates for BERT sentiment classification

- **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/

UNIVERSITY of VIRGINIA

Join at
slido.com
#3140 184

Parameter Efficient Fine-tuning (PEFT)

- Fine-tuning all model parameters is expensive

Pretrained weight (can represent any module) $\boldsymbol{W}_0 \in \mathbb{R}^{d\times d}$

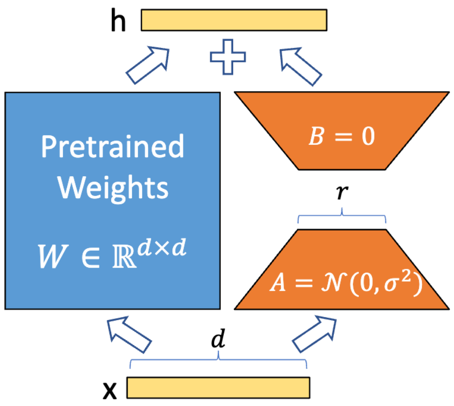
Fine-tuned weight $\boldsymbol{W}^* = \boldsymbol{W}_0 + \Delta\boldsymbol{W}, \quad \Delta\boldsymbol{W} \in \mathbb{R}^{d\times d}$

- 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

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

- Encoder-decoder Pretraining (Continued)
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning

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 2024, 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

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

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

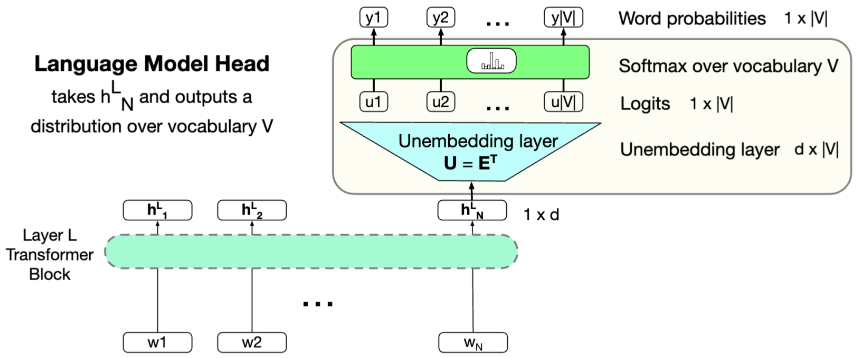


Figure source: https://web.stanford.edu/~jurafsky/slp3/9.pdf

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

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

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

Join at slido.com #3140 184

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

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

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

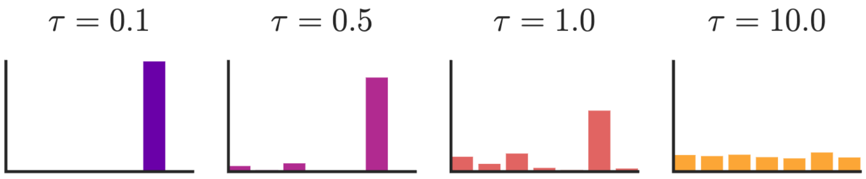


Figure source: https://arxiv.org/pdf/1611.01144v5

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

Agenda

- Encoder-decoder Pretraining (Continued)
- Prompting and Parameter Efficient Fine-tuning
- Large Language Models (LLMs) for Text Generation
- In-context Learning

Join at
slido.com
#3140 184

In-context Learning

- In-context learning is a type of few-shot learning
  - User provides a few examples of input-output pairs in the prompt
  - The model uses given examples to predict the output for new, similar inputs
- First studied in the GPT-3 paper
- No model parameter updates

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

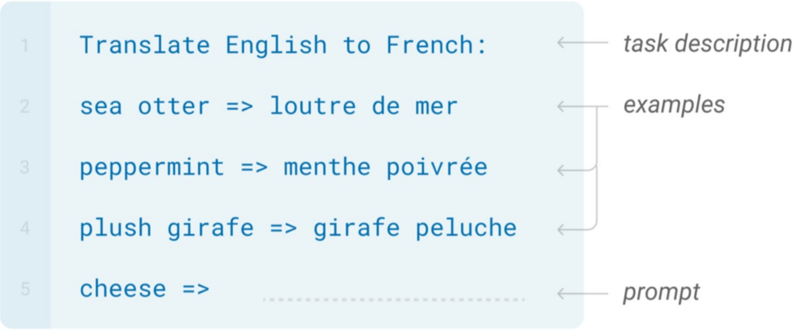
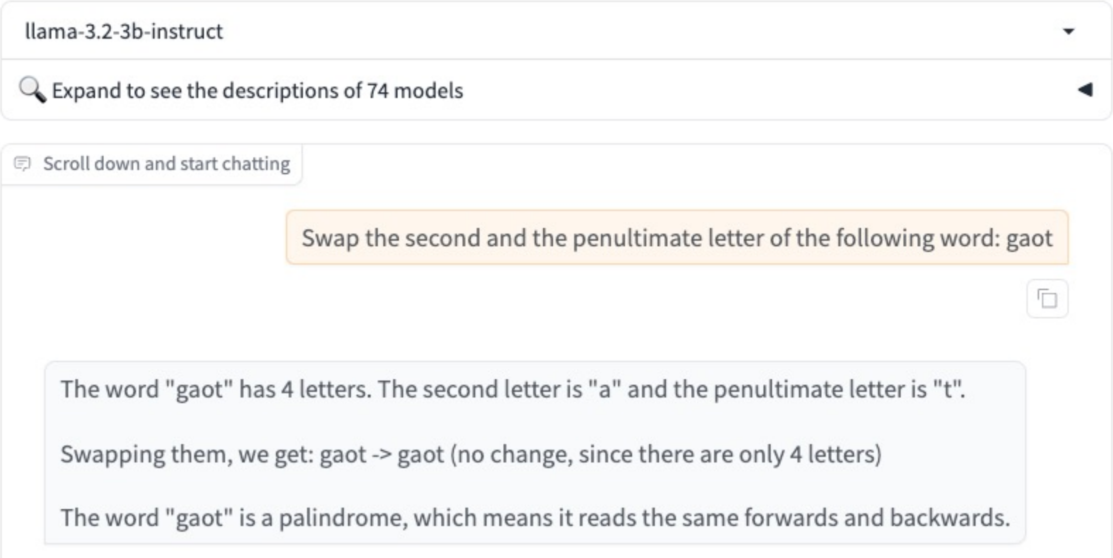


Figure source: https://arxiv.org/pdf/2005.14165

In-context Learning Demo

Prompt: “Swap the second and the penultimate letter of the following word: gaot”



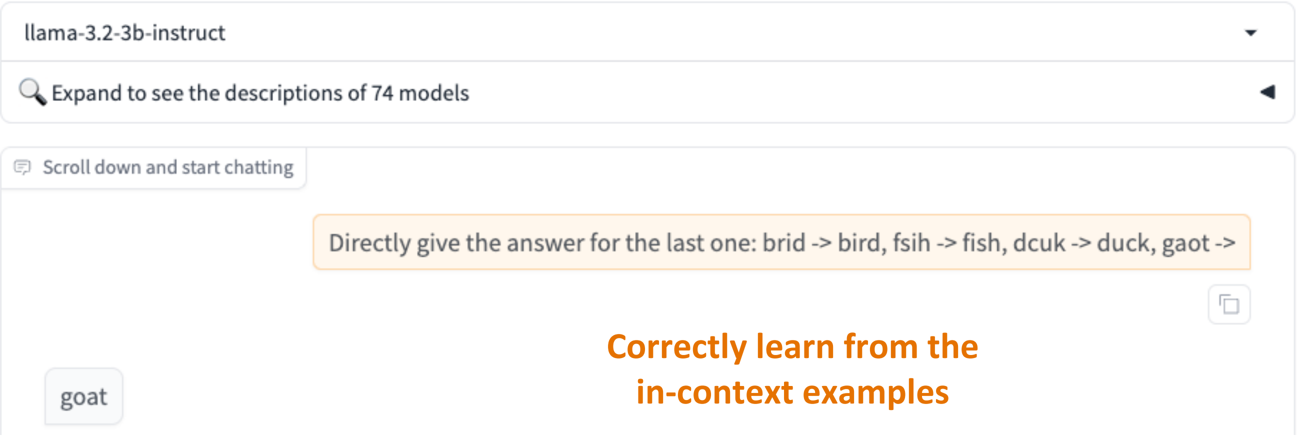
Wrong generation only given the prompt

Generated with greedy decoding
(temperature = 0)

Figure source: https://lmarena.ai/?model=llama-3.2-3b-instruct

In-context Learning Demo

Prompt: “Directly give the answer for the last one: brid -> bird, fsih -> fish, dcuk -> duck, gaot ->”



Correctly learn from the in-context examples

Generated with greedy decoding (temperature = 0)

Figure source: https://lmarena.ai/?model=llama-3.2-3b-instruct

UNIVERSITY of VIRGINIA

Join at
slido.com
#3140 184

Further Reading on In-context Learning

- An Explanation of In-context Learning as Implicit Bayesian Inference [Xie et al., 2021]
- Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? [Min et al., 2022]
- What Can Transformers Learn In-Context? A Case Study of Simple Function Classes [Garg et al., 2022]
- What learning algorithm is in-context learning? Investigations with linear models [Akyurek et al., 2023]

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu