6 – Scaling LLMs

IASD Apprentissage – LLMs course

Florian Le Bronnec

December 3, 2024

Table of Contents

Table of Contents

Exam

Information

- Date: 10/12
- 2h.
- No computer, no internet.
- 2 pages (recto-verso) of handwritten notes.
- Exam will feature questions about an article AND the course.
- The article will be made available on the 03/12 at 12h.
- No article during the exam.

Advice

- No need to learn by heart.
- If a question is about a specific point of the article, it will be reminded in the exam subject.

Mock exam on Attention is all you need

Question 1: Encoder Input in Translation

What is the input to the encoder in translation?

Question 1: Encoder Input in Translation

What is the input to the encoder in translation?

Answer: The input to the encoder is a sequence of symbol representations (e.g., embeddings) of the source sentence. These embeddings are combined with positional encodings to inject information about the order of tokens in the sequence.

Question 2: Decoder Input in Translation

During training, what is the input of the decoder? What are its labels?

Question 2: Decoder Input in Translation

During training, what is the input of the decoder? What are its labels?

Answer: The input to the decoder is the target sequence's embeddings, combined with positional encodings. The labels are the target sequence shifted by one position to the right.

Question 3: Step-by-Step Operations in the Decoder

Describe step-by-step the operations performed in the decoder.

Question 3: Step-by-Step Operations in the Decoder

Describe step-by-step the operations performed in the decoder.

Answer:

- Input Preparation (shifted target sequence + word embeddings + positional encodings).
- Masked Multi-Head Self-Attention
- Encoder-Decoder Multi-Head Attention
- Feed-Forward Network
- Residual Connections and Layer Normalization
- Output Projection

Question 4: ROUGE and BLEU Metrics and Its Limitations

Without giving a precise algorithm, how do the BLEU and ROUGE metric work? What are their limitations?

Question 4: ROUGE and BLEU Metrics and Its Limitations

Without giving a precise algorithm, how do the BLEU and ROUGE metric work? What are their limitations?

Answer:

- **How it works:**
  - BLEU and ROUGE measure the overlap of n-grams (subsequences of tokens) between a generated sequence and one or more reference sequences.
- **Limitations:**
  - BLEU and ROUGE do not consider semantic meaning, focusing only on surface-level token matches.

Question 5: Sampling Decoding Algorithm

Describe in detail the sampling decoding algorithm.

Question 5: Sampling Decoding Algorithm

Describe in detail the sampling decoding algorithm.

Answer:

- Initialize the generated sequence with a start token `<BOS>`.
- Repeat until the end token `<EOS>` is generated or a maximum length is reached:
  - **Step 1: Input Preparation.** Feed the current sequence into the model to obtain the probability distribution $P(y_t \mid \text{previous tokens})$ over the vocabulary.
  - **Step 2: Sampling.** Use the probability distribution to sample the next token y_t. This can involve:
    - Random multinomial sampling from $P(y_t)$.
  - **Step 3: Append Token.** Add the sampled token to the sequence.
- Return the generated sequence.

Table of Contents

Last time – Finetuning

Finetuning of encoder models

Encoder (BERT-like models) are tuned for specific tasks.

- **Classification**: add a classification head on top of the model, on the `[CLS]` token.
- **Token classification (or tagging)**: add a classification head on top of the model, on each token.
- **Sequence pairs classification**: add a classification head on top of the model, on the `[SEP]` token.

Tasks: NER, POS tagging, sentiment analysis, etc.

Last time – Finetuning

Finetuning of decoder models

Decoder (GPT-like models) are tuned for specific tasks.

- Models are still trained with a **next token prediction** objective.
- Concatenate the source with the target, and train the model to predict the target.

Tasks: translation, summarization, question answering, etc.

Table of Contents

What's under the hood?

Impressive pretrained models

- BERT-like models achieve **better performances than humans** on the SuperGLUE dataset [6].
- BART-like models can **summarize texts, answer questions**, etc. [2].
- GPT3-3.5-4 are few-shot learners, they can answer questions based on their **general knowledge** [4].
- ChatGPT.

What's under the hood?

Impressive pretrained models

- BERT-like models achieve **better performances than humans** on the SuperGLUE dataset [6].
- BART-like models can **summarize texts, answer questions**, etc. [2].
- GPT3-3.5-4 are few-shot learners, they can answer questions based on their **general knowledge** [4].
- ChatGPT.

How?

- Data.
- Scaling the models.

What's under the hood?

Impressive pretrained models

- BERT-like models achieve **better performances than humans** on the SuperGLUE dataset [6].
- BART-like models can **summarize texts, answer questions**, etc. [2].
- GPT3-3.5-4 are few-shot learners, they can answer questions based on their **general knowledge** [4].
- ChatGPT.

How?

- Data.
- Scaling the models.

$\Longrightarrow$ What does it represent in practice?

Data

Model	Params	Context	Batch	Steps
BERT [1]	355M	512	256[1]	1M
BART [2]	406M	1024	8000	500K
LLama2 [11]	7-13-70B	4096	4000[1]	500K

Table 1: Comparison of NLP models. Steps are the number of training steps.

[1]Batch size were indicated in terms of number of tokens, I approximately converted it to number of documents.

Data

Model	Params	Context	Batch	Steps
BERT [1]	355M	512	256[1]	1M
BART [2]	406M	1024	8000	500K
LLama2 [11]	7-13-70B	4096	4000[1]	500K

Table 1: Comparison of NLP models. Steps are the number of training steps.

Demo!

[1]Batch size were indicated in terms of number of tokens, I approximately converted it to number of documents.

Pretraining in practice

Biggest NVIDIA GPUs are $\sim \mathbf{8}\times$ bigger than Colab's GPUs.

Even with bigger GPUs, processing batches of **8000 documents** is an issue. How can we do that in practice?

Pretraining in practice

Biggest NVIDIA GPUs are $\sim \mathbf{8}\times$ bigger than Colab's GPUs.

Even with bigger GPUs, processing batches of **8000 documents** is an issue. How can we do that in practice?

$\implies$ Let's review some methods to perform these heavy trainings.

Data parallelism

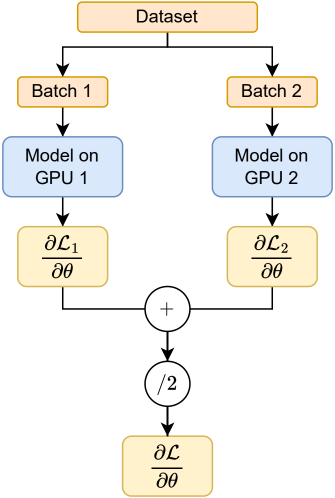

Figure 1: Data parallelism.

Gradient accumulation

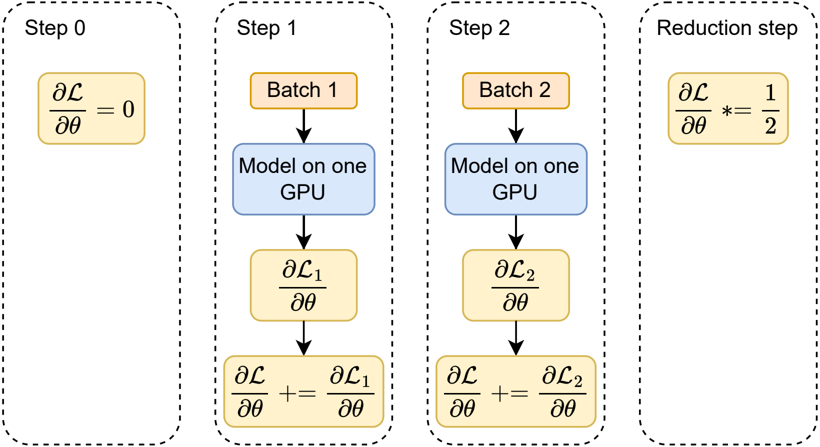

Figure 2: Gradient accumulation.

Summary

Data parallelism and **gradient accumulation** are two very common methods in NLP (and deep learning) to increase the effective size of the batch.

Data Parallelism

Per Device Batch Size	**Number of Devices**	**Effective Batch Size**
32	1	32
32	2	64
32	4	128

Gradient accumulation

Batch Size	**Accumulation Steps**	**Effective Batch Size**
32	1	32
32	2	64
32	4	128

Comparison

Aspect	Data Parallelism	Gradient Accumulation
Parallelism	Yes, across GPUs	No, sequential computations
Several GPUs	Yes	Can work on 1 GPU
Time	$1 \times (\texttt{forward} + \texttt{backward})$ $+$ communication across GPUs $+$ parameters update	$N \times (\texttt{forward} + \texttt{backward})$ $+$ parameters update

Table 2: Comparison of Data Parallelism and Gradient Accumulation. N is the number of accumulation steps.

Comparison

Aspect	Data Parallelism	Gradient Accumulation
Parallelism	Yes, across GPUs	No, sequential computations
Several GPUs	Yes	Can work on 1 GPU
Time	$1 \times (\texttt{forward} + \texttt{backward})$ $+$ communication across GPUs $+$ parameters update	$N \times (\texttt{forward} + \texttt{backward})$ $+$ parameters update

Table 2: Comparison of Data Parallelism and Gradient Accumulation. N is the number of accumulation steps.

$\Longrightarrow$ Data parallelism is useful when you have **several GPUs** and want to **speed up** the training.

$\Longrightarrow$ Gradient accumulation is useful when you have **limited resources.**

Both can of course be combined (and are combined in practice).

In practice?

Everything is quite easy with PyTorch.

Demo!

Scaling the number of parameters

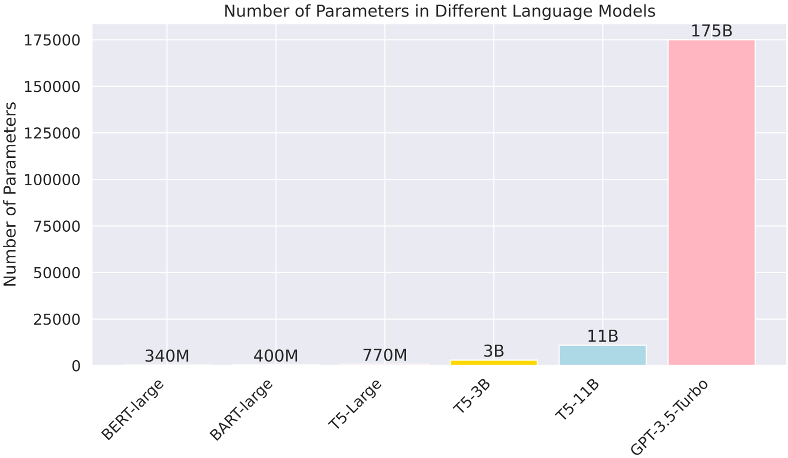

Figure 3: Different models and their scales.

What does it imply?

Demo on GPT2 size.

What does it imply?

Demo on GPT2 size.

GPT2-Large on a GPU is $\sim$ 3.5GiB. GPT-3 is $\sim$ $\times$250 bigger than GPT2-Large.

Biggest GPUs are 80GiB.

$\Longrightarrow$ How does it fit?

What does it imply?

Demo on GPT2 size.

GPT2-Large on a GPU is $\sim$ 3.5GiB. GPT-3 is $\sim \times 250$ bigger than GPT2-Large.

Biggest GPUs are 80GiB.

$\implies$ How does it fit?

We're going to speak about **training** and **finetuning / inference**.

Model parallelism

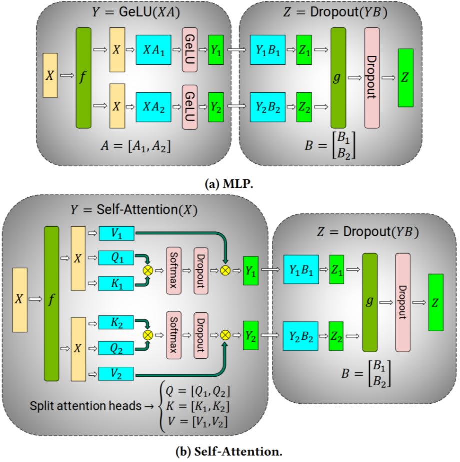

Figure 4: Idea: split the computations across several GPUs. Tensor parallelism, figure from [5].

Pretraining

For pretraining, there are some heavy hardware optimizations, like the ones presented in [5].

The idea is to split independant and costly operations across devices, then aggregate the final result.

Even if there are solutions, as a rule of thumb remind that **model parallelism is not easy**.

And for end-users?

Most people are not interested in what we presented in previous slides.

But still, we might want to use these models for at least:

- running inference as is,
- finetuning on a specific task.

$\Longrightarrow$ Let's describe some solutions for practitioners.

Some solutions: Quantization

GPUs standard precision is `float32`. A solution is to **reduce this precision**.

Data Type	Bit Width	Hardware Capability	Use for training
`float32`	32	General-purpose CPUs/GPUs	Yes
`float16`	16	GPUs with FP16 support	Yes
`bfloat16`	16	NVIDIA Ampere GPUs, TPUs	Yes
`int8`	8	CPUs, GPUs	No

Table 3: Non-exhaustive list of mixed-precision data types and hardware support.

Some solutions: Quantization

GPUs standard precision is `float32`. A solution is to **reduce this precision**.

Data Type	Bit Width	Hardware Capability	Use for training
`float32`	32	General-purpose CPUs/GPUs	Yes
`float16`	16	GPUs with FP16 support	Yes
`bfloat16`	16	NVIDIA Ampere GPUs, TPUs	Yes
`int8`	8	CPUs, GPUs	No

Table 3: Non-exhaustive list of mixed-precision data types and hardware support.

$\Longrightarrow$ In practice the models maintain their performances (empirical statement).

Be careful when using quantization

- Make sure the operations you use are well supported.
- Read the documentation, especially for training.
- There can be some instabilities (ex: T5 does not work with `float16`).

Reduce the cost of autodiff

Component	Memory Cost	Inference
Model Parameters	$O(P)$	Yes
Activations	$O(B \times L)$	Yes / No
Gradients	$O(P)$	No
Optimizer States	$O(P)$	No

Table 4: Memory cost during training and inference.

Huge dependency on P, the number of parameters to update.

Reduce the cost of autodiff

Component	Memory Cost	Inference
Model Parameters	$O(P)$	Yes
Activations	$O(B \times L)$	Yes / No
Gradients	$O(P)$	No
Optimizer States	$O(P)$	No

Table 4: Memory cost during training and inference.

Huge dependency on P, the number of parameters to update.

$\implies$ Reduce the number of parameters to update!

Parameter efficient finetuning

Let $\theta \in \mathbb{R}^P$ be the parameters of a base model: BERT, Llama, etc.

The goal of **parameter efficient finetuning (PEFT)** is to select a subset of parameters $\theta' \in \mathbb{R}^{P'}$ with $P' \ll P$ such that the model's performances are maintained.

Example

Train only the last layer of the model.

Break!

Table of Contents

Which length of texts can we process?

How many tokens can fit in the models?

There are 3 types of limitations:

- **architectural**, because of position embeddings of fixed size,
- **training setup**, the maximal length seen during training, independently of the position embeddings,
- **computational cost**, because the costs scale with the context length.

Attention is $\mathcal{O}(L^2)$

Let L be the input length.

$$\boldsymbol{QK}^T \in \mathbb{R}^{L \times L}.$$

Demo!

Attention is $\mathcal{O}(L^2)$

Let L be the input length.

$$\boldsymbol{QK}^T \in \mathbb{R}^{L \times L}.$$

Demo!

$\implies$ **Quadratic** cost w.r.t L.

How can we avoid that?

Scaling the context

Simple idea: **reduce the size of the attention matrix**.

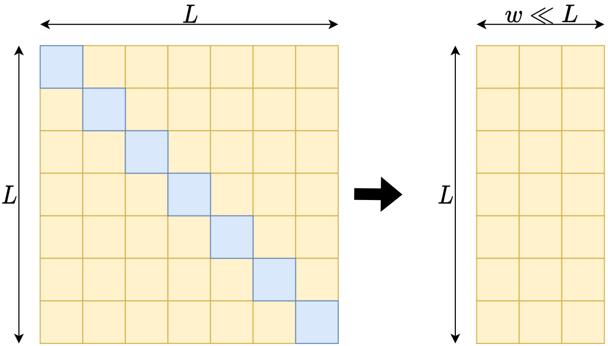

Figure 5: Reducing the size of the attention matrix.

We can therefore reach a $\mathcal{O}(Lw)$ memory cost.

Sparse attention

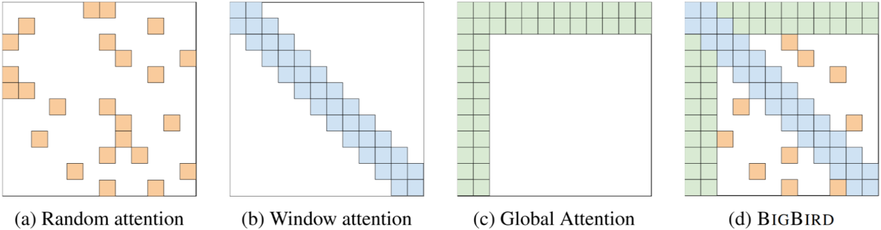

Figure 6: Sparse attention patterns, figure from [7].

Whole litterature on sparse transformers:

- LongT5 [9],
- BigBird [7],
- Longformer [3] (extends pretrained models),
- etc.

Other solutions

Hardware improvements

- FlashAttention [8] proposed recently a new CUDA kernel optimized for computing attenion on recent NVIDIA GPUs.

Other solutions

Hardware improvements

- FlashAttention [8] proposed recently a new CUDA kernel optimized for computing attenion on recent NVIDIA GPUs.

Stick to the model's base length

- 4k tokens is already a lot.
- Models have **not been pretrained on such lengths**.
- LLMs **do not use their full context** [10].
- Use **retrieval techniques**.
- Process the documents by **chunks** (e.g. summarize chapter by chapter).

What about other atchitectures?

New architectures are being proposed

State-space models, MAMBA, etc.

Table of Contents

Introduction to Training Chat Models

- **Goal**: Develop models capable of understanding and responding to user instructions in a conversational manner.
- **Challenge**: Achieving natural, helpful, and aligned responses requires more than basic fine-tuning.
- **Two-Phase Training Approach**:
  - **Phase 1 – Instruction-Tuning**: Fine-tune the model on instruction-based datasets to improve task understanding.
  - **Phase 2 – Reinforcement Learning from Human Feedback (RLHF)**: Use human feedback to further align the model's responses with user preferences.
- This combination makes chat models able of going beyond basic instruction-following, incorporating nuanced human feedback.

Instruction-Tuning for Chat Models

- **Objective**: Adapt generative models to follow user instructions accurately.
- **Method**: Fine-tune models on datasets where each example includes an instruction and a correct response.

- Examples of instruction-tuning datasets: FLAN, Super-NaturalInstructions.

Example of Supervised Instruction-Tuning

- **Task**: Fine-tune the model to respond accurately to user instructions.

Instruction	**Model Input**	**Output Prediction**
Instruction: "List three benefits of regular exercise." Expected Output: "Improves cardiovascular health, boosts mental well-being, and strengthens muscles."	`[BOS] List three benefits of regular exercise.[EOS]`	`Improves cardiovascular health, boosts mental well-being, and strengthens muscles.`

- **Objective**: Model learns to generate responses aligned with the instruction prompt.

Limitations of Instruction-Tuning

- **Instruction-Tuning Provides a Solid Start**:
  - Models are trained to follow directions and respond to a wide range of prompts.
  - Instruction datasets enable models to generalize across many basic tasks.

Limitations of Instruction-Tuning

- **Instruction-Tuning Provides a Solid Start**:
  - Models are trained to follow directions and respond to a wide range of prompts.
  - Instruction datasets enable models to generalize across many basic tasks.
- **Key Limitations of Instruction-Tuning**:
  - **Models are trained using only next token prediction objective**: Models will learn the dataset distribution. This does not encompass directly any notion of "quality", like helpfulness, factuality, toxicity, etc.
  - **Datasets are not perfect**: After learning, models might reflects some imperfections of the training data, like generic tone or writing style that might not be appropriate in all contexts.
  - **No filtering on sensitive domains**: Instruction datasets are often filtered, and often contain generic or neutral content. Therefore, models should be further trained to handle sensitive domains, like avoiding harmful content.

Reinforcement Learning from Human Feedback (RLHF)

- **Solution**:

Reinforcement Learning from Human Feedback (RLHF)

- **Solution**: Improve response quality and relevance through direct human feedback, guiding models to align closer with human expectations.
- **Goal of RLHF**:
  - Use human feedback to enhance the model's ability to produce "human-like" responses.

Reinforcement Learning from Human Feedback (RLHF)

- **Solution**: Improve response quality and relevance through direct human feedback, guiding models to align closer with human expectations.
- **Goal of RLHF**:
  - Use human feedback to enhance the model's ability to produce "human-like" responses.

Why RLHF Matters

- **Why RLHF Matters**:
  - Aligns model responses more closely with human-like preferences and expectations.
  - Provides a method for refining responses beyond the limitations of instruction-tuning.
- **Examples of RLHF Datasets**: Anthropic's HH-RLHF, OpenAI's feedback datasets.

Correcting Model Behavior with Human Feedback

Objective: Improve model responses by correcting cases where generated outputs are suboptimal or misaligned with user expectations.

Ideal Approach:

1. Generate a variety of responses to a prompt.
2. Make a human rate each response, on a scale from 0 to 5.
3. Use a reinforcement learning (RL) algorithm to adjust the model's behavior, encouraging preferred responses and discouraging low-quality ones.

Such RL algorithms can be REINFORCE, PPO, etc.

These methods might help correct and refine model behavior, aligning responses more closely with human expectations.

Why Direct Human Ratings are Impractical

Challenge with Human Ratings: While human ratings provide valuable feedback, rating every response generated by the model is unrealistic.

Limitations of Relying on Human Ratings:

- **High Cost and Time Demand**: Scaling human feedback for all responses is resource-intensive.
- **Slow Iteration**: Relying on human feedback would slows down model training.

To make reinforcement learning feasible at scale, we need a way to approximate these ratings efficiently.

Approximating Human Feedbackk with a Value Model

Instead of using human, we will train a model to approximate human preferences. **Step 1: Collect human feedback**:

- For a given prompt, we generate 2 responses.
- Human annotators are asked to rank the responses based on quality (a preferred response and a less-preferred response).
- We train the value model (typically a value head on top of the finetuned LM) to predict whether a response is a good one or not (independently of each other), using standard binary classification loss.

After trainnig, we have a value model r which is able to give a "score" to a response, $r(y)$, which corresponds to the probability of the response being the preferred one.

Using a Value Model with REINFORCE

Goal: Use the value model to approximate rewards for generated responses, guiding the main model's updates.

REINFORCE Algorithm with Value Model:

- For a prompt x, the model generates a response y.
- The value model assigns a score $r(y)$ to the response, approximating human preference.
- Update the main model's parameters θ to maximize expected reward $\mathbb{E}[r(y)]$ using the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{y \sim \pi_\theta(y|x)} \left[r(y) \nabla_\theta \log \pi_\theta(y|x) \right]$$

RLHF – Summary

- Learn a **value model** to approximate human feedback scores.
- Use **REINFORCE** or **PPO** to update the main model based on the value model's scores.
  - Generate responses to intrusctions.
  - Compute the score using the value model.
  - Update the main model using REINFORCE or PPO.

Motivation for Direct Preference Optimization (DPO)

Limitations of RL with Value Models:

- REINFORCE or PPO require a value model to approximate human feedback scores.
- Training with RL involves balancing multiple components, such as reward scaling and policy stability, making it complex and computationally intensive.

Can we get rid of RL?:

- Yes! Use directly the important piece of information: preference dataset.
- This is the idea behind Direct Preference Optimization (DPO).

Transition from RL to DPO

In RL (e.g., REINFORCE), we maximize the expected reward:

$$J(\theta) = \mathbb{E}_{y \sim \pi_\theta(y|x)} [r(y)]$$

In DPO, rather than optimizing for reward scores, we optimize preferences directly. For a prompt x and two responses $y_{\text{preferred}}$ and $y_{\text{less\_preferred}}$, DPO adjusts the model to make $y_{\text{preferred}}$ more likely.

DPO Loss:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\log \frac{\pi_\theta(y_{\text{preferred}}|x)}{\pi_\theta(y_{\text{less\_preferred}}|x)} \right)$$

where σ is the sigmoid function.

Complete RL and DPO objective

Regularized RL Loss with Reference Model:

- In reinforcement learning, we often use a regularization term to keep the fine-tuned model π_θ close to a base reference model π_{ref}, which helps maintain stability and prevents excessive deviation.

$$\mathcal{L}_{\text{RL}} = -\mathbb{E}_{y \sim \pi_\theta(y|x)}\left[r(y)\right] - \lambda \text{KL}(\pi_\theta \| \pi_{\text{ref}})$$

Regularized DPO Loss with Reference Model:

- In DPO, this regularization translate to using a base model π_{ref} to stabilize training (full proof in original DPO paper).

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\log \frac{\pi_\theta(y_{\text{preferred}}|x)/\pi_{\text{ref}}(y_{\text{preferred}}|x)}{\pi_\theta(y_{\text{less\_preferred}}|x)/\pi_{\text{ref}}(y_{\text{less\_preferred}}|x)} \right)$$

where π_{ref} keeps π_θ anchored to the reference model's distribution.

Contrastive Learning in DPO – Introduction

Contrastive Learning Overview:

- In contrastive learning, the goal is to make similar samples closer while pushing dissimilar samples apart in the model's representation space.

Applying Contrastive Learning to Preferences:

- In DPO, we have a prompt x with two responses:
  - $y_{\text{preferred}}$: the response preferred by human feedback.
  - $y_{\text{less\_preferred}}$: a lower-rated alternative.
- The objective is to make $y_{\text{preferred}}$ more likely than $y_{\text{less\_preferred}}$, similar to bringing closer and pushing apart in contrastive learning.

DPO as a Contrastive Objective

DPO Loss as a Contrastive Loss:

- The DPO loss function:

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\log \frac{\pi_\theta(y_{\text{preferred}} | x)}{\pi_\theta(y_{\text{less\_preferred}} | x)} \right)$$

  optimizes the model to assign higher probability to preferred responses.

- This can be interpreted as a contrastive objective:
  - Maximizing $\pi_\theta(y_{\text{preferred}}|x)$ pulls preferred responses "closer" (higher probability).
  - Minimizing $\pi_\theta(y_{\text{less\_preferred}}|x)$ pushes less-preferred responses "farther" (lower probability).

Conclusion: DPO vs. RL for Human Alignment

- **Simplicity**: DPO directly optimizes for preferences, avoiding the complexity of reward modeling in RL.
- **Efficiency**: DPO's contrastive objective is computationally lighter, allowing faster, more stable training.

Conclusion: DPO vs. RL for Human Alignment

- **Simplicity**: DPO directly optimizes for preferences, avoiding the complexity of reward modeling in RL.
- **Efficiency**: DPO's contrastive objective is computationally lighter, allowing faster, more stable training.

Performance Gains from Human Alignment:

- DPO and other human-alignment methods have significantly improved model quality, with benchmarks showing up to 20-30% gains in user satisfaction and response relevance.
- DPO now sets the standard for aligning large language models with human feedback, achieving higher consistency and responsiveness.

Conclusion

Thank you!

References I

[1] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[2] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL].

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL].

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[5] Mohammad Shoeybi et al. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. 2020. arXiv: 1909.08053 [cs.CL].

[6] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL].

References II

[7] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences*. 2021. arXiv: 2007.14062 [cs.LG].

[8] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG].

[9] Mandy Guo et al. *LongT5: Efficient Text-To-Text Transformer for Long Sequences*. 2022. arXiv: 2112.07916 [cs.CL].

[10] Mathieu Ravaut et al. *On Position Bias in Summarization with Large Language Models*. 2023. arXiv: 2310.10570 [cs.CL].

[11] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].