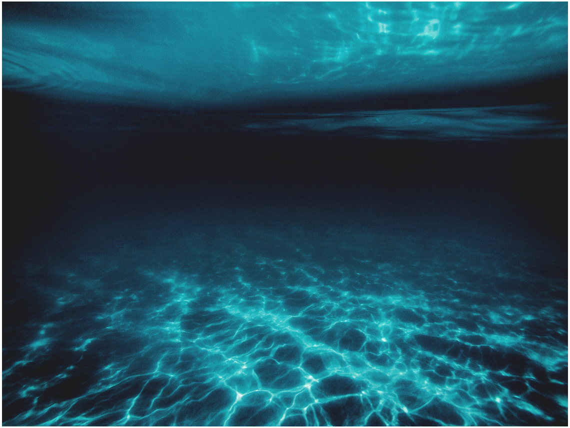
Natural Language Processing

Lecture : Large Language Models & Post-training

Master Degree in Computer Engineering
University of Padua
Lecturer : Giorgio Satta

Post-training



Jonathan Borba from Usplash

Post-training

To make practical use of a pretrained LLM, we need to

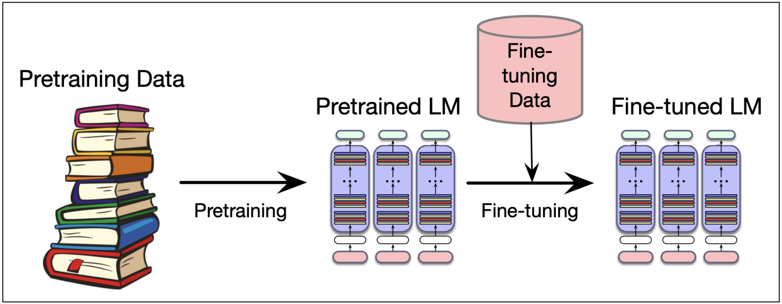
- **interface** the model with some downstream task/applications
- **adapt** the model to a specific domain of interest

This process is called **post-training** or **adaptation**. It is the prevalent paradigm today in natural language processing.

Post-training typically uses data with stronger signal and with higher quality than that of pretraining data.

Post-training

A pretrained model can be fine-tuned to a particular domain, dataset, or task.



Post-training

Post-training groups together several techniques

- fine-tuning / continued pretraining
- supervised fine-tuning
- instruction tuning
- alignment

In-context learning is also considered a form of post-training, but it does not change the model parameters and will thus be introduced in the next lecture.

Post-training typically uses much less computational resources than pretraining, and does not target all of the parameters of the LLM.

Several **efficient** techniques are known for parameter updating during post-training.

Fine-tuning

Joel Wyncott from Unsplash

Fine-tuning

We might want to specialize a LLM for a new task.

The standard approach is **fine-tuning**

- add a head layer for the task at hand, at the top of the transformer, with learnable parameters
- make (possibly minimal) adjustments to the pretrained parameters of the LLM

In contrast with fine-tuning, retraining the whole LLM (all parameters) results in so-called **catastrophic forgetting**.

Fine-tuning

Example : Under BERT pretrained model, let $\mathbf{z}_{\text{CLS}}$ be the embedding for the token `[CLS]`. For **sentiment analysis** define the classifier

$$\mathbf{y}_{\text{CLS}} = \text{softmax}(\mathbf{W}_C \cdot \mathbf{z}_{\text{CLS}} + \mathbf{b})$$

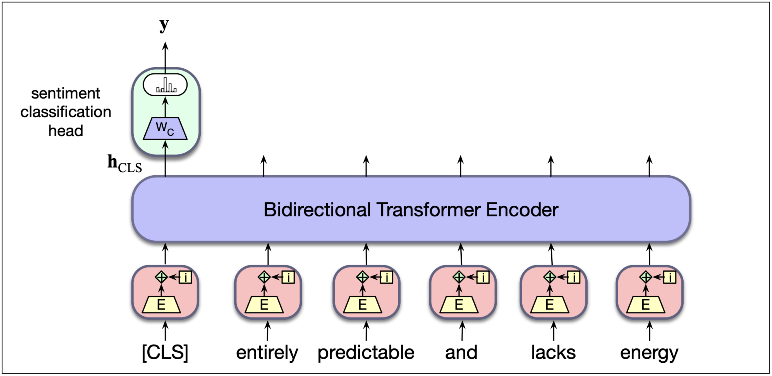
Use labeled data to learn $\mathbf{W}_C, \mathbf{b}$ and to **update** the weights for the pretrained language model itself.

In practice, only **minimal** changes to the language model parameters are needed, limited to updates over the final few layers of the transformer.

We will see later efficient techniques for updating the parameters of the LLM.

Fine-tuning

Example : (continued, $\mathbf{z}_{\mathrm{CLS}}$ written as $\mathbf{h}_{\mathrm{CLS}}$ in the picture)



$\mathbf{W}_C$ is usually called the **classification head**.

Fine-tuning

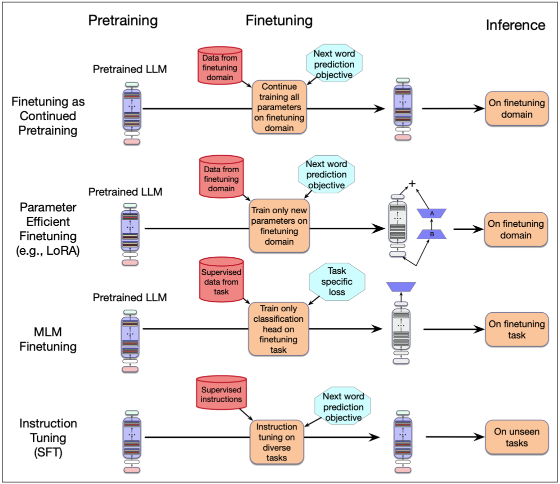
Supervised fine-tuning, or SFT for short, refers to the process of fine-tuning a pretrained model on a **labeled dataset** for a specific task, using a supervised learning objective function.

The goal is simply to make the model better at predicting the correct output for a given input: classification, etc.

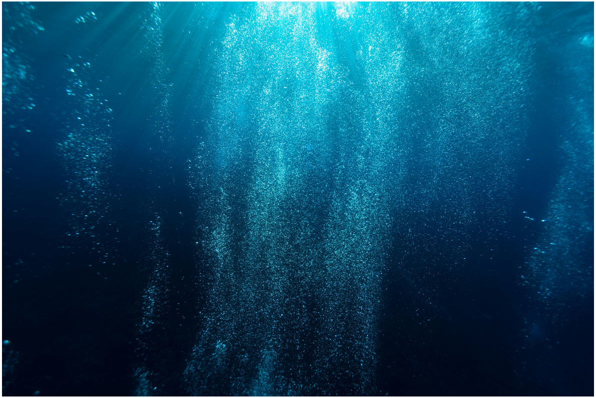
The previous example on sentiment analysis is a case of SFT.

While the terms fine-tuning and supervised fine-tuning are often used interchangeably in casual conversation, they represent different levels of specificity in the training pipeline.

Several typologies of fine-tuning



Instruction tuning



Sarah Lee from Unsplash

Instruction tuning

LLMs have not been designed to answer questions or to follow instructions, even in presence of strong signals such as prompt.

Example :

Prompt: Explain the moon landing to a six year old in a few sentences.
Output: Explain the theory of gravity to a 6 year old.

Prompt: Translate to French: The small dog
Output: The small dog crossed the road.

The problem is that the objective function (predict the next word) is misaligned with the human need for models to be helpful.

Instruction tuning

The goal of **instruction tuning** is to make an LLM better at following general instructions.

The training data consists of

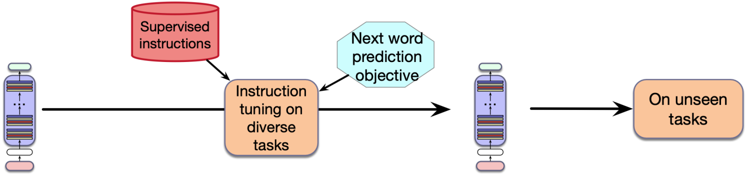
- instructions and answers for a wide range of tasks, and/or
- question/answer pairs in a specific domain of interest (optional)

We continue training the model with the guess-the-next-token objective, **limited to** the answer part of each example.

Instruction tuning is what turned GPT-3.5 into ChatGPT; more about this in the next lecture.

Instruction tuning

Instruction tuning can be viewed as a second pass of LLM training, and is sometimes referred to as **continued pretraining**



As for fine-tuning, the overall cost of instruction tuning is a small fraction of the original cost to train the base model.

Instruction tuning

Unlike in pretraining, each instruction or question in the training data for instruction tuning has a supervised objective: a correct answer, or a response to the instruction.

For this reason, instruction tuning is considered a form of **supervised learning**, and is viewed as a special case of SFT.

The distinction between general SFT and instruction tuning is most visible in how the model handles **zero-shot** requests, that is, tasks that have not been seen before.

Datasets for instruction tuning

Instruction tuning datasets can be created

- manually, through crowdworkers, based on carefully written annotation guidelines
- by adapting existing datasets previously created for a wide range of natural language tasks (question answering, translation, summarization, etc.)

  language models can also be used to generate paraphrases of the prompts
- automatically, using powerful chatBots and prompts based on human written guidelines

Datasets for instruction tuning

Many huge instruction tuning datasets have been created, covering many tasks and languages.

Most popular

- **Aya**: 503 million instructions in 114 languages from 12 tasks
- **Flan**: see next slides
- **MMLU**: Massive multitask language understanding, almost 16K multiple-choice questions, spanning across 57 subjects (from complex STEM fields to international law)
- **SuperNatural Instructions**: 12 million examples from 1600 tasks

Datasets for instruction tuning

Fine-tuned Language Net (FLAN) compiles several datasets into a mix of zero-shot, few-shot and chain-of-thought templates. It is a specific set of instructions used to fine-tune different models.

Consists of 473 datasets across 146 task categories.

Several LLM instruction tuned with FLAN: Flan-T5, Flan-PaLM, etc.

Datasets for instruction tuning

Release	Collection	Model Details: Model	Base	Size	Public?	Data Collection & Training Details: Prompt Types	Tasks in Flan	# Exs	Methods
2020 05	UnifiedQA	UnifiedQA	RoBerta	110-340M	P	ZS	46 / 46	750k	
2021 04	CrossFit	BART-CrossFit	BART	140M	NP	FS	115 / 159	71.M	
2021 04	Natural Inst v1.0	Gen. BART	BART	140M	NP	ZS / FS	61 / 61	620k	+ Detailed k-shot Prompts
2021 09	Flan 2021	Flan-LaMDA	LaMDA	137B	NP	ZS / FS	62 / 62	4.4M	+ Template Variety
2021 10	P3	T0, T0+, T0++	T5-LM	3-11B	P	ZS	62 / 62	12M	+ Template Variety + Input Inversion
2021 10	MetaICL	MetaICL	GPT-2	770M	P	FS	100 / 142	3.5M	+ Input Inversion + Noisy Channel Opt
2021 11	ExMix	ExT5	T5	220M-11B	NP	ZS	72 / 107	500k	+ With Pretraining
2022 04	Super-Natural Inst.	*Tk*-Instruct	T5-LM, mT5	11-13B	P	ZS / FS	1556 / 1613	5M	+ Detailed k-shot Prompts + Multilingual
2022 10	GLM	GLM-130B	GLM	130B	P	FS	65 / 77	12M	+ With Pretraining + Bilingual (en, zh-cn)
2022 11	xP3	BLOOMz, mT0	BLOOM, mT5	13-176B	P	ZS	53 / 71	81M	+ Massively Multilingual
2022 12	Unnatural Inst.†	T5-LM-Unnat. Inst.	T5-LM	11B	NP	ZS	~20 / 117	64k	+ Synthetic Data
2022 12	Self-Instruct†	GPT-3 Self Inst.	GPT-3	175B	NP	ZS	Unknown	82k	+ Synthetic Data + Knowledge Distillation
2022 12	OPT-IML Bench†	OPT-IML	OPT	30-175B	P	ZS + FS CoT	~2067 / 2207	18M	+ Template Variety + Input Inversion + Multilingual
2022 10	Flan 2022 (ours)	Flan-T5, Flan-PaLM	T5-LM, PaLM	10M-540B	P NP	ZS + FS CoT	1836	15M	+ Template Variety + Input Inversion + Multilingual

The Flan Collection: Designing Data and Methods for Effective Instruction Tuning, Longpre et al. 2023

Datasets for instruction tuning

Example : SuperNatural Instructions

Few-Shot Learning for QA

Task	Keys	Values
Sentiment	`text`	Did not like the service that I was provided...
	`label`	0
	`text`	It sounds like a great plot, the actors are first grade, and...
	`label`	1
NLI	`premise`	No weapons of mass destruction found in Iraq yet.
	`hypothesis`	Weapons of mass destruction found in Iraq.
	`label`	2
	`premise`	Jimmy Smith... played college football at University of Colorado.
	`hypothesis`	The University of Colorado has a college football team.
	`label`	0
Extractive Q/A	`context`	Beyoncé Giselle Knowles-Carter is an American singer...
	`question`	When did Beyoncé start becoming popular?
	`answers`	{ `text`: ['in the late 1990s'], `answer_start`: 269 }

Evaluation of instruction-tuned LLMs

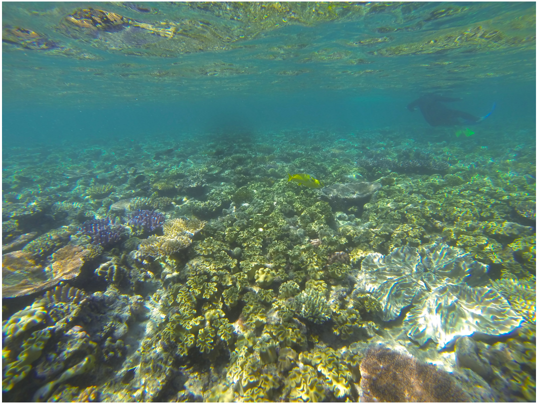
We need to assess how well an instruction-tuned model performs on novel tasks for which it has not been given explicit instructions.

Standard evaluation methods take a **leave-one-out** approach: test the model on a withheld task.

Popular datasets for instruction tuning have an enormous numbers of tasks: these tasks very often **overlap**!

One needs to cluster tasks according to types, and test on unknown cluster types.

Model alignment



Dmitry Brant

Model alignment

Instruction tuning results in LLMs that generate coherent and contextually appropriate responses.

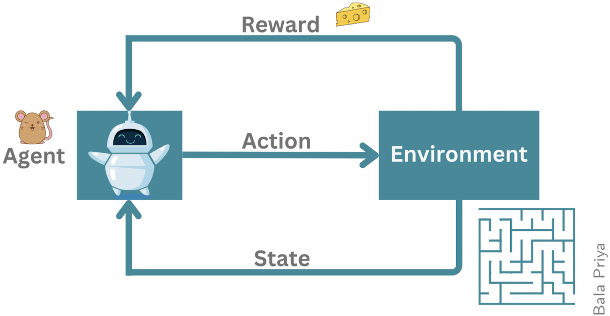
However, some responses may contain content that is harmful, unethical, socially biased, and negative, or even illegal.

Model alignment is a post-training process ensuring that LLMs meet human expectations, ethical standards, and personalized needs.

Model alignment

Reinforcement Learning (RL) is a subfield of machine learning where an **agent**

- interacts with some environment to get a **reward** signal
- learns to take decisions that maximize the reward



RL methods lay in between supervised and unsupervised learning.

Preference-based learning

RL has emerged as the most popular approach for model alignment of LLM's.

The approach is called **preference-based learning** or, alternatively, **reinforcement learning from human feedback** (RLHF).

Preference-based learning involves

- fitting of a reward function to a dataset of human preferences
- using the reward function to train an LLM to generate responses with high reward

Preference-based learning

Preference-based learning uses high-quality datasets consisting in questions with two contrastive answers

- a **preferred** answer, meeting the desired standard
- a **dispreferred** answer, whose style should be discouraged

Example : Prompts and preferences pairs

Prompt: I've heard garlic is a great natural antibiotic. Does it help with colds?
Chosen: It can be helpful against colds, but may make you stink.
Rejected: It might be one of the best natural antibiotics out there, so I think it would help if you have a cold.

Prompt: What is malaria?
Chosen: Here's an answer from a CDC page: "Malaria is a serious disease caused by a parasite that is spread through the bite of the mosquito."
Rejected: I don't know what malaria is.

Preference-based learning

Preference data generally come from three sources

- preference judgments from trained human annotator, e.g., annotators rate outputs on a **Likert scale** along various dimensions
- implicit preference judgments extracted from online resources, e.g., user votes on social media, etc.
- fully synthetic preference judgments from LLMs as annotators, e.g., prompting outputs from diverse LLMs and then prompting GPT-5 to rank the outputs

Modeling preferences

Notation:

- x denotes a **prompt**
- o denotes some **output** sampled from an LLM using x
- $z \in \mathbb{R}$ denotes the **score** associated with output o

When a given output o_i is preferred to another o_j, both for a given prompt x, we write $o_i \succ o_j \mid x$.

We need to move from $o_i \succ o_j \mid x$ to knowing the value of $P(o_i \succ o_j \mid x)$.

Modeling preferences

Let z_i and z_j be the scores associated with outputs o_i and o_j, respectivelly.

We define $P(o_i \succ o_j \mid x)$ as the **logistic sigmoid** of the difference in the scores

$$
\begin{aligned}
P(o_i \succ o_j \mid x) &= \frac{1}{1 + \exp\left(-(z_i - z_j)\right)} \\
&= \sigma(z_i - z_j)
\end{aligned}
$$

The approach will be mathematically motivated later.

Modeling preferences

This approach is known as the Bradley-Terry model and has the following advantages

- very small differences in scores yields probabilities near 0.5, reflecting either weak or no preference between the items
- larger differences rapidly approach values of 1 or 0
- the derivative of the logistic sigmoid facilitates learning via a binary cross-entropy loss

Modeling preferences

Let us assume some event A with probability $P(A)$ such that $0 < P(A) < 1$.

The **odds** (chances) of A is defined as the ratio between

- the probability that A occurs
- the probability that A does not occur

$$Odds(A) = \frac{P(A)}{P(A^C)} = \frac{P(A)}{1 - P(A)}$$

Intuitivelly, while $P(A)$ represents successes out of the total, think of the odds as successes vs. failures.

Modeling preferences

The difference $\delta = z_i - z_j$ is assumed to represent the logarithm of the odds of the event $o_i \succ o_j \mid x$

$$\begin{aligned}\delta &= \log\left(\frac{P(o_i \succ o_j \mid x)}{P(o_j \succ o_i \mid x)}\right)\\ &= \log\left(\frac{P(o_i \succ o_j \mid x)}{1 - P(o_i \succ o_j \mid x)}\right)\end{aligned}$$

We can therefore write

$$\exp(\delta) = \frac{P(o_i \succ o_j \mid x)}{1 - P(o_i \succ o_j \mid x)}$$

Modeling preferences

This can be rearranged as

$$\begin{aligned}
\exp(\delta)(1 - P(o_i \succ o_j \mid x)) &= P(o_i \succ o_j \mid x) \\
\exp(\delta) - \exp(\delta)(P(o_i \succ o_j \mid x) &= P(o_i \succ o_j \mid x)
\end{aligned}$$

and then

$$\begin{aligned}
\exp(\delta) &= P(o_i \succ o_j \mid x) + \exp(\delta)(P(o_i \succ o_j \mid x) \\
&= P(o_i \succ o_j \mid x)(1 + \exp(\delta))
\end{aligned}$$

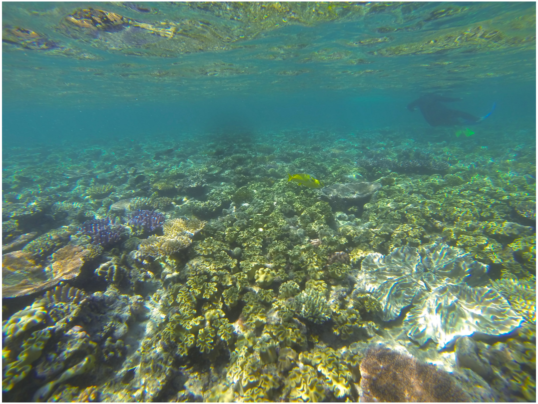
Modeling preferences

This leads to

$$\begin{aligned} P(o_i \succ o_j \mid x) &= \frac{\exp(\delta)}{1 + \exp(\delta)} \\ &= \frac{1}{1 + \exp(-\delta)} \\ &= \frac{1}{1 + \exp(-(z_i - z_j))} \\ &= \sigma(z_i - z_j) \end{aligned}$$

This is a mathematical motivation of the Bradley-Terry model.

Learning to score preferences



Dmitry Brant

Learning to score preferences

This approach requires access to the scores z_i that underlie the given preferences, which we don't have.

We use our preference data and the Bradley-Terry formulation to learn a function $r(o, x)$ that assigns a scalar **reward** (a real number) to each prompt/output pair.

Formally, $r(o, x)$ calculates our scores z's

$$\begin{aligned} P(o_i \succ o_j \mid x) &= \sigma(z_i - z_j) \\ &= \sigma(r(o_i, x) - r(o_j, x)) \end{aligned}$$

Learning to score preferences

We designate the preferred output in the pair (the winner) as o_w and the loser as o_l.

With this, the **cross-entropy loss** for a single pair of sampled outputs for a prompt x is:

$$\begin{aligned} L_{CE}(x, o_w, o_l) &= -\log P(o_w \succ o_l \mid x) \\ &= -\log \sigma(r(x, o_w) - r(x, o_l)) \end{aligned}$$

That is, the cross-entropy loss is the negative log-likelihood of the model's estimate of $P(o_w \succ o_l \mid x)$.

Learning to score preferences

The cross-entropy loss over the **preference training set** $\mathcal{D}$ is given by the following expectation:

$$L_{CE} = -\mathbb{E}_{(x,o_w,o_l)\sim\mathcal{D}}\left[\log \sigma(r(x,o_w) - r(x,o_l))\right]$$

To learn a reward model using this loss, we can use any regression model capable of

- taking text as input, and
- generating a scalar as output

Learning to score preferences

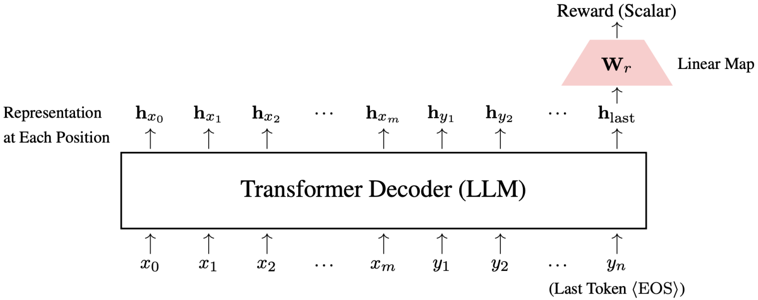
The current preferred approach is to initialize a reward model from an existing pretrained LLM.

To generate scalar outputs, we

- remove the language modeling head from the final layer, and
- replace it with a learnable single dense linear layer

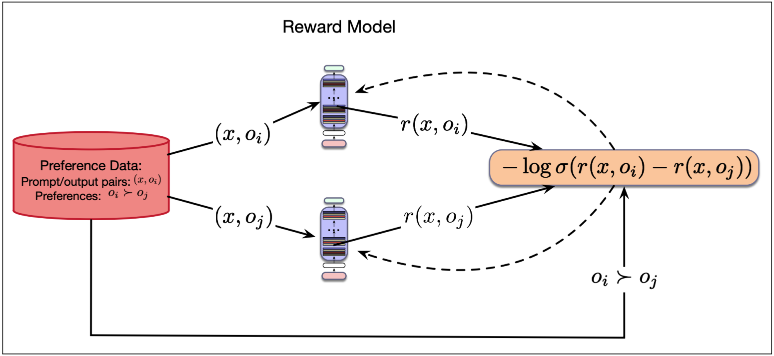
Finally, we use gradient descent with the previous loss to **learn** our model parameters using the preference training data $\mathcal{D}$.

Architecture of the reward model based on Transformer.



Learning to score preferences

Reward model learning with a pretrained LLM; a linear layer is initialized randomly and trained with a CE loss.



LLM alignment via preference learning

LLM alignment via preference learning

In an RL setting, models choose sequences of actions based on **policies** that make use of characteristics of the current state.

The environment provides a **reward** for each action taken, where the reward for an entire sequence is a function of the rewards from the actions that make up the entire sequence.

LLM alignment via preference learning

In applying RL to optimizing LLMs, we'll use the following framework:

- **actions** correspond to the choice of tokens made during autoregressive generation
- **states** correspond to the tokens generated so far by the LLM (the context)
- **policy** corresponds to the pre-trained LLM
- **reward** for LLM outputs is based on the reward function learned from preference data

LLM alignment via preference learning

Notation for the next slides:

- $\boldsymbol{\pi}$ refers to the pretrained LLM generating output o, given input x
- θ refers to the learnable parameters of the LLM
- $r(x, o)$ refers to the reward score learned from preference data, as previously described

Given our reward model derived from preference data, the **goal** is to fine-tune the LLM $\boldsymbol{\pi}_\theta$ to generate outputs that maximize the expectation of the reward function.

LLM alignment via preference learning

We can express our goal as an **optimization problem**:

$$\boldsymbol{\pi}^* = \underset{\boldsymbol{\pi}_\theta}{\text{argmax}}\ \mathbb{E}_{x\sim\mathcal{D}, o\sim\boldsymbol{\pi}_\theta}\left[r(x,o)\right]$$

Informally, this amounts to the following steps

- select prompts x from a training collection $\mathcal{D}$
- sample outputs o from LLM $\boldsymbol{\pi}_\theta$
- assess the reward for each sample
- compute the expected reward for $\boldsymbol{\pi}_\theta$
- find the model that maximizes that expected reward

LLM alignment via preference learning

There are two key differences between traditional RL and the way RL is typically used for LLM alignment.

In traditional RL, the reward signal comes from the environment and reflects an observable fact about the results of an action, i.e., you win a game or you don't.

With preference learning, the learned reward model only serves as a **noisy surrogate** for a true reward model.

LLM alignment via preference learning

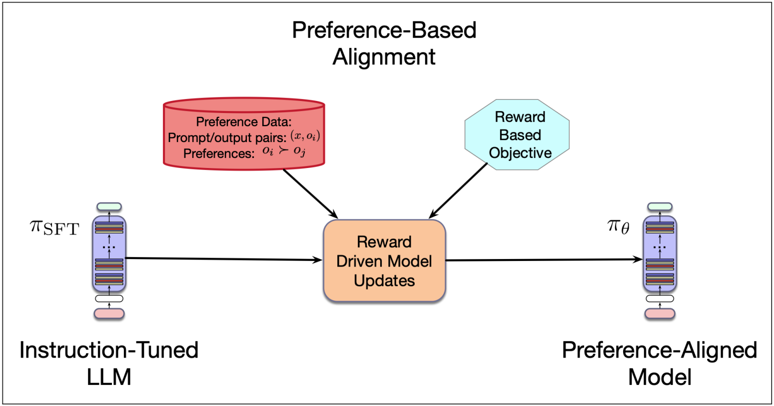
In traditional RL, typical applications seek to learn an optimal policy from scratch, that is, from a randomly initialized policy.

With preference learning we begin with models that are already performing at a **high level**, that is, models that have been

- pretrained on large amounts of data
- finetuned using instruction tuning

Only afterward these models are further improved with preference data.

LLM alignment via preference learning



LLM alignment via preference learning

If we optimize the previous objective function, the LLM will typically **forget everything** it learned during pretraining, seeking high rewards from the relatively small amount of available preference data.

To avoid this, a term is added to penalize models that diverge too far from the starting point

$$\boldsymbol{\pi}^* \quad = \quad \underset{\boldsymbol{\pi}_\theta}{\text{argmax}} \; \mathbb{E}_{x\sim\mathcal{D}, o\sim\boldsymbol{\pi}_\theta(o|x)} \left[r(x, o) - \beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o \mid x)}{\boldsymbol{\pi}_{\text{ref}}(o \mid x)} \right]$$

The second term, modulated by hyperparameter β, is the **Kullback-Leibler divergence** (KL) which measures the distance between two probability distributions.

LLM alignment via preference learning

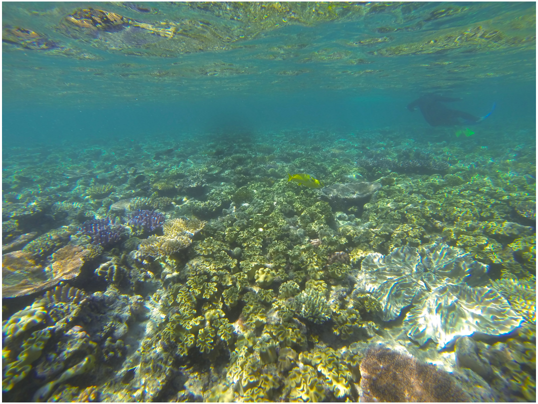
Based on the previous KL-constrained optimization problem, two popular RLHF algorithms have been developed.

Proximal policy optimization (PPO) firstly fits a reward model that reflects the human preferences, and then finetunes the LLM to maximize the reward, without drifting too far from the original model.

Problems: sensitivity to hyperparameters, inherent instability during training.

Direct preference optimization (DPO) optimizes candidate LLM directly, without learning an explicit reward model and using a loss simpler than DPO.

Direct preference optimization



Dmitry Brant

Direct preference optimization

The key insight underlying DPO is to express the reward function $r(x, o)$ in terms of the two policies $\boldsymbol{\pi}_\theta$ and $\boldsymbol{\pi}_{\mathrm{ref}}$

$$r(x, o) \quad = \quad \beta \left(\log \frac{\boldsymbol{\pi}_\theta(o \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o \mid x)} + \log Z(x) \right)$$

Here $Z(x)$ is the so-called **partition function**, the sum over all the possible outputs o given a prompt x

$$Z(x) \quad = \quad \sum_{o} \boldsymbol{\pi}_{\mathrm{ref}}(o \mid x) \cdot \exp(\frac{1}{\beta} \cdot r(x, o))$$

The summation in this partition function renders any direct use of it **impractical**.

Direct preference optimization

Recall that in the developed RL framework we have

$$\begin{aligned} P(o_i \succ o_j \mid x) &= \sigma(z_i - z_j) \\ &= \sigma(r(o_i, x) - r(o_j, x)) \end{aligned}$$

Plugging $r()$ into the above **cancels out** the partition functions

$$P(o_i \succ o_j \mid x) = \sigma\left(\beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_i \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_i \mid x)} - \beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_j \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_j \mid x)}\right)$$

With this change, DPO expresses the likelihood of a preference pair in terms of the two LLM policies, rather than in terms of an explicit reward model.

Direct preference optimization

Now we can write the CE loss for a single instance as

$$
\begin{aligned}
L_{\mathrm{DPO}}(x, o_w, o_l) \; &= \\
&- \log \sigma \left(\beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_w \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_w \mid x)} - \beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_l \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_l \mid x)} \right)
\end{aligned}
$$

The loss over the training set $\mathcal{D}$ is given by the following expectation:

$$
\begin{aligned}
L_{\mathrm{DPO}}(\boldsymbol{\pi}_\theta) \; &= \\
&- \mathbb{E}_{(x, o_w, o_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_w \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_w \mid x)} - \beta \cdot \log \frac{\boldsymbol{\pi}_\theta(o_l \mid x)}{\boldsymbol{\pi}_{\mathrm{ref}}(o_l \mid x)} \right) \right]
\end{aligned}
$$

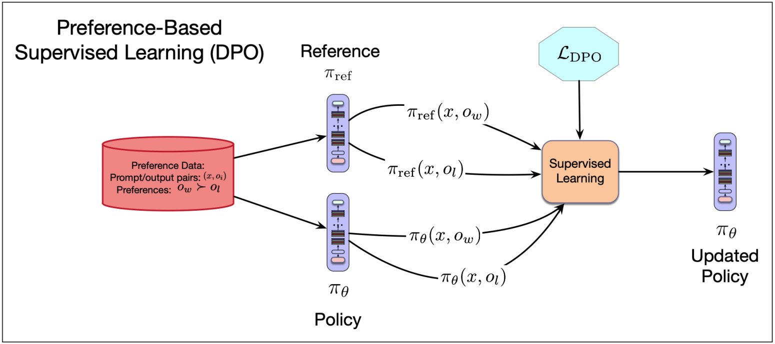
Direct preference optimization

Operationally, the design of this loss function

- increases the likelihood of the preferred options
- decreases the likelihood of the dispreferred options
- balances this objective with the goal of not straying too far from $\boldsymbol{\pi}_{\text{ref}}$

Direct preference optimization

Preference-based alignment with Direct Preference Optimization.

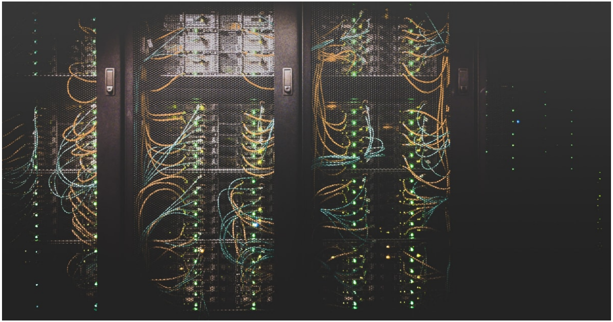


Direct preference optimization

DPO has several advantages over PPO

- DPO does not require training an explicit reward model
- DPO learns directly from the preferences contained in $\mathcal{D}$ without the need for computationally expensive online sampling from $\boldsymbol{\pi}_\theta$
- DPO only incurs the cost of maintaining 2 LLMs during training, as opposed to the 4 models needed for PPO

Parameter efficient fine-tuning

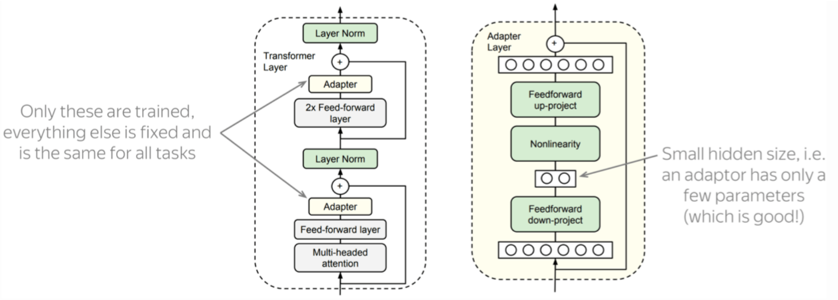


When working with huge pretrained models, instruction tuning and fine-tuning may still be **inefficient**.

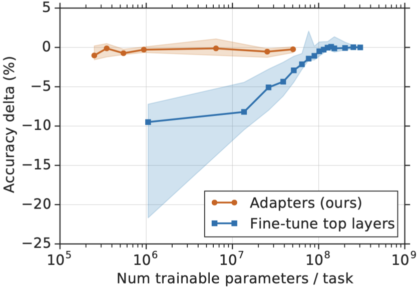
Alternatively, one could fix the pretrained model, and train only small, very simple components called **adapters**.

With adapter modules post-training becomes very **efficient**: the largest part of the pretrained model is shared between all downstream tasks.

Transformer with adapter module consisting of a two-layer feed-forward network with a nonlinearity and a residual connection.



Adapter-based tuning attains a similar performance to top-layer fine-tuning, with two orders of magnitude fewer trained parameters.



Houlsby *et al.*, 2019

LoRA

LoRA (Low-Rank Adaptation) is a popular fine-tuning strategy, alternative to adapters. It drastically reduces the number of trainable parameters.

LoRA is based on the idea of **intrinsic dimensions**: there exists a low dimension reparameterization that is as effective for fine-tuning as the full parameter space.

LoRA is known for its high efficiency, and for avoiding catastrophic forgetting.

LoRA

We can think of **weight update** for the transformer as follows

$$W \quad \leftarrow \quad W + \Delta W$$

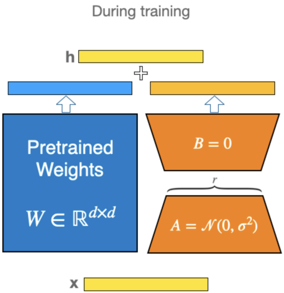
where $W \in \mathbb{R}^{d \times d}$.

In LoRA we update the weights using a decomposition of ΔW into two low-rank matrices

$$\Delta W \quad = \quad BA$$

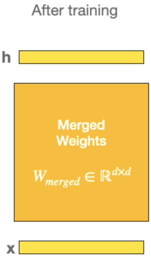
where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times d}$, and r is some low **rank**.

LoRA



$$h = Wx + BAx$$

$$h = \underbrace{(W + BA)}_{W_{merged}}x$$



Saurav Maheshkar, A Brief Introduction to LoRA: Low-Rank Adaptation of Large Language Models

Discussion

LLMs learn very powerful general representations about language and real world knowledge.

Traditional approach in NLP is to develop applications from scratch.

Fine-tune approach: use LLM as a basis, and fine-tune it to the desired task — think about LLM as multi-task models!

In-context approach: provide a few examples of the task, and ask the question/instance at hand — prompt engineering.

We introduce prompt engineering in next lecture.

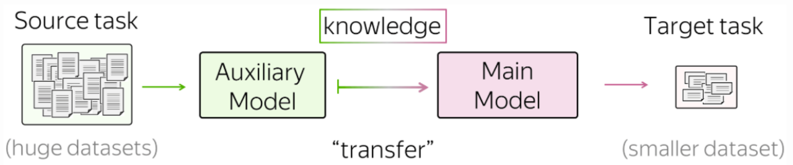
Transfer learning

Rhys Moult from Unsplash

The pretraining/fine-tuning paradigm is a special case of a machine learning approach called transfer learning.

The general idea of **transfer learning** is to reuse information from a previously learned source task for the learning of a target tasks.

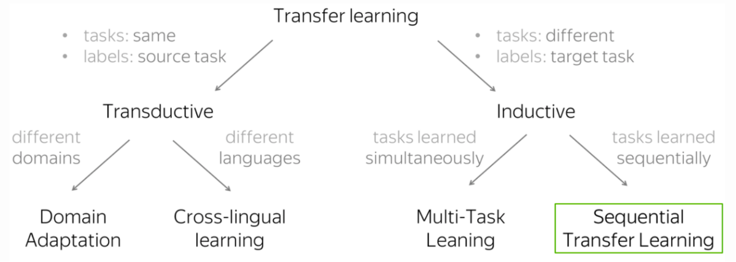
This is used when you don't have enough data for the target task.



There are several types of transfer learning

See for instance Sebastian Ruder's blog post at

https://ruder.io/state-of-transfer-learning-in-nlp/



Fine-tuning, also known as **sequential transfer learning**, is the most popular transfer learning approach in NLP.