11. Aligning Language Models (Basics)

Overview

What is alignment

- Prompting converts a task to a native LM task, but model performance is sensitive to prompts
- Goal: make human-AI communication natural and efficient
- So that we can just ask the model to do any task

Capability vs alignment

- **Capability**: What things is the model able to do?
- **Alignment**: What things does the model choose to do?

  — Align with human values
  - Provide truthful information and express uncertainty
  - Be careful with potentially harmful information
  - Clarify user intentions and preferences

Challenges in alignment

- **Implicit rules**: not articulated but assumed in human interaction
  - e.g. Explicit task: answer questions on topic X

    Implicit rules: Don't make up stuff. Don't use toxic language. Don't give information that's potentially harmful.
  - The implicit rules may be context dependent:
    - Translation: what if the source text is toxic?
- **Oversight**: provide supervision on alignment
  - One obvious way to align models is to train them on supervised data (later)
  - But how can we supervise models on tasks that beyond human capabilities?
- **Diversity**: whose values should the model be aligned with?
  - Different (cultural/ethnic/gender/religious/etc.) groups agree with different answers to the same question

Approaches to alignment

- **Prompting**: ask the model to behave according to human values
- **Finetuning / Supervised learning**: show the model the right response in various context
- **Reinforcement learning**: reward / punish the model when its behavior is aligned / unaligned with humans

Prompting

Prompting the model to answer questions truthfully

Prompts can be overwritten — ask it to ignore previous prompts

Summary

Prompt engineering: instruct the model to behave in a certain way

Pros:

- Easy to do—anyone can play around with it
- Efficient—no parameter updates
- First thing to try

Cons:

- Unprincipled—no idea why it works or doesn't work
- Unreliable—performance can have high variance
- Unsafe—easy to bypass

Supervised finetuning

- How do we teach the model the right behavior?
- Going back to supervised learning: demonstrate the right behavior
    - Input: user prompt (task specification)
    - Output: (aligned) response
- **Key challenge**: data collection

  How to get the prompts and responses?

What kind of data do we need?

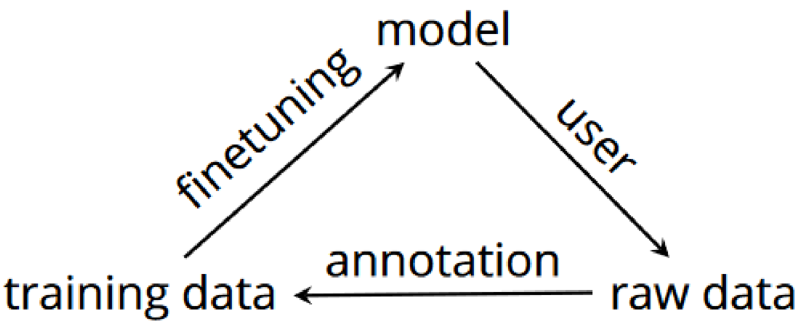
Idea 1: use existing NLP benchmarks

But this is not what we ask ChatGPT to do! Distribution shift.

- **Problem**: Gap between training and test data
- Straightforward **solution**: collect training data that is similar to test data

  How do we know what test data is like?
- Get some pilot data

  which requires a working-ish model first!



Tricky cases

- Recall that we want the model to infer user intention
- But also to make the right decisions that align with human values
- So it's important to include examples that invovle alignment decisions
- Open question: how to handle trade-off between helpfulness and harmfulness?

  e.g., user may request to generate toxic sentences for data augmentation

Summary

Supervised finetuning: train the model to respond in an aligned way on human-annotated prompt-response data

Pros:

- Relatively reliable—generalize to unseen data
- User friendly—doesn't require extensive prompt engineering
- Simple training pipeline—standard finetuning

Cons:

- Need a warm start—pilot data to decide what data to collect
- Expensive—data needs to cover many uses cases
- Compute—need to update very large models

Reinforcement learning

Motivation:

- Demonstrations are expensive to obtain—can we learn from weaker signals?
- For many tasks, humans (and animals) only get signal on whether they succeeded or not

Goal: learning from experience by maximizing the expected reward

At each time step t, an agent

- is in a **state** $s_t \in S$ (S is the state space)

- takes an **action** $a_t \in A$ (A is the action space)
- transitions to the next state s_{t+1} according to a **transition function** $p(\cdot\,|s_t, a_t)$
- obtains a **reward** $r(s_t, a_t)$ according to the **reward function** $r : S \times A \to \mathbb{R}$

The agent uses a **policy** π to decide which actions to take in a state:

- Deterministic: $\pi(s) = a$
- Stochastic: $\pi(a|s) = \mathbb{P}(A = a|S = s)$

A policy π_θ defines a distribution $p_\theta(\tau)$ over **trajectories** $\tau = (a_1, s_1, ..., a_T, s_T)$.

The agent's **objective** is to learn a policy π_θ (parametrized by θ) that maximizes the expected return:
$\text{maximize } \mathbb{E}_{\gamma\sim p_\theta(\gamma)}[\sum_{t=1}^{T} r(s_t, a_t)]$

Key steps:

- **Trial**: run policy to generate trajectories
- **Error**: estimate expected return
- **Learn**: improve the policy

Challenges:

- Trials could be expensive (e.g., healthcare, education)
- Reward signal could be expensive and sparse (e.g., expert feedback)
- May need many samples to learn a good policy

Policy gradient algorithms

While not converged

1. Sample trajectories from the current policy
2. Estimate return for each trajectories based on observed rewards
3. Take a gradient step on the expected return (w.r.t. the policy)

Notation: let $r(\tau) = \sum_{t=1}^{T} r(a_t, s_t)$ be the return.

$$\text{Our objective:} \quad J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[r(\tau)] = \sum_{\tau} p_\theta(\tau) r(\tau)$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau} p_\theta(\tau) r(\tau) \\
&= \sum_{\tau} \nabla_\theta p_\theta(\tau) r(\tau) \\
&= \sum_{\tau} p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) \\
&= \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]
\end{aligned}$$

log derivative trick

$$\begin{aligned}
& p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) \\
=& p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} \\
=& \nabla_\theta p_\theta(\tau)
\end{aligned}$$

Good news: the gradient is now inside the expectation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) r(\tau) \right] \quad \text{average gradient of sampled trajectory}$$

But what is $p_\theta(\tau)$?

$$p_\theta(\tau) = p_\theta(a_1, s_1, \dots, a_T, s_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t)$$

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]$$

Putting everything together

REINFORCE algorithm:

1. Sample N trajectories $\tau^1, \dots, \tau^N$ from π_θ
2. Estimate the gradient:

$$\nabla_\theta J(\theta) \approx \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i) \right) \left(\sum_{t=1}^{T} r(s_t^i, a_t^i) \right)$$

3. Update the policy with gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. Go back to 1

How is all this related to LLMs?

Think of tokens as actions:

- Action space: vocabulary $\quad a_t = x_t \in \mathcal{V}$
- State space: history / prefix $\quad s_t = (x_1, \dots, x_{t-1})$
- Policy: a language model $\quad p_\theta(x_t \mid x_{<t})$
- Trajectory: a sentence / generation $\quad x_1, \dots, x_T$

REINFORCE algorithm on text:

1. Sample N generations from the language model p_θ
2. Estimate the gradient: $\nabla_\theta J(\theta) \approx \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log p_\theta(x_t^i \mid x_{<t}^i) \right) r(x_{1:T})$
3. Update the policy with gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
4. Go back to 1

What is the algorithm doing?

If $r(x_{1:T})$ is positive, take a gradient step to increase $p_\theta(x_{1:T})$.
If $r(x_{1:T})$ is negative, take a gradient step to decrease $p_\theta(x_{1:T})$.

Supervised learning on model generations weighted by rewards

How to get the reward? Next lecture!

Summary

Reinforcement learning: align the model by giving it feedback on whether an output is good or bad

Pros:

- Cost-efficient—humans only need to provide judgments/rewards
- General—can be used to model all kinds of human preferences

Cons:

- Complex pipeline—RL algorithms need more engineering
- Reward hacking—models are good at finding ways to ”cheat”

  Generating polite and authorative nonsense
- Human judgments on some subjects are inherently diverse