ATCI: Reinforcement Learning

About the reward Function

Mario Martin

CS-UPC

March 4, 2025

Motivation

- We assume that reward function is easy to design, but usually we observe the *cobra effect*

[1] List of agents cheating in AI

Motivation

- We assume that reward function is easy to design, but usually we observe the *cobra effect* [1]
  - Some cases take profit of glitches in the simulator (Atari)
  - Some others from bad definition of the reward (Stack lego blocks / Pancake / Romba / Pausing / Grasping / Minotaur / Walking flipward)
- Corollary of Goodhart's law:
  - *"When a measure becomes a target, it ceases to be a good measure"*
- ...because all metrics of scientific evaluation are bound to be abused.
- Examples of such problems not only in RL also NLP (ROUGE, BLEU measures)

[1] List of agents cheating in AI

Goal of this lecture

- In other cases, we have problems defining safety conditions (you have to define what you should not do! Sometimes a long list)
- In other cases, we cannot apply RL because the reinforcement function is unknown or too complex (f.i. driving)

Goal of this lecture

- In other cases, we have problems defining safety conditions (you have to define what you should not do! Sometimes a long list)
- In other cases, we cannot apply RL because the reinforcement function is unknown or too complex (f.i. driving)

- What should we do in these cases? Can we learn a behavior agent-like without a reward function?
- We will see two different solution for to solve this problem:
  1. Inverse Reinforcement Learning
  2. Reinforcement learning with human Feedback

Goal of this lecture

- In other cases, we have problems defining safety conditions (you have to define what you should not do! Sometimes a long list)
- In other cases, we cannot apply RL because the reinforcement function is unknown or too complex (f.i. driving)

- What should we do in these cases? Can we learn a behavior agent-like without a reward function?
- We will see two different solution for to solve this problem:
  1. **Inverse Reinforcement Learning**
  2. Reinforcement learning with human Feedback

Inverse Reinforcement Learning

Goal of this lecture

- In other cases, we have problems defining safety conditions (you have to define what you should not do! Sometimes a long list)
- In other cases, we cannot apply RL because the reinforcement function is unknown or too complex (f.i. driving)

- What should we do in these cases? Can we learn a behavior agent-like without a reward function?
- We will see two different solution for to solve this problem:
  1. Inverse Reinforcement Learning
  2. **Reinforcement learning with human Feedback**

Reinforcement learning with human Feedback

Reinforcement learning with human Feedback

- We know that we have problems with undesired behavior due to flaws or under-specification of the reward function.
- Also that in some cases it is complex to define a reward function

Reinforcement learning with human Feedback

- We know that we have problems with undesired behavior due to flaws or under-specification of the reward function.
- Also that in some cases it is complex to define a reward function
- What can we do? ... One approach learn from examples both directly or with IRL
- But **what happens when we don't have examples or they are very costly to obtain?**

Reinforcement learning with human Feedback

- We know that we have problems with undesired behavior due to flaws or under-specification of the reward function.
- Also that in some cases it is complex to define a reward function
- What can we do? ... One approach learn from examples both directly or with IRL
- But **what happens when we don't have examples or they are very costly to obtain?**

- In this case we can use another approach: Reward modelling using human feedback

DRL from Human Preferences (Christiano et al. 17)

- *Deep reinforcement learning from human preferences* (Christiano et al. 17)
- Problem: we don't know how to define a good reward function for a task,
- ... but we know to recognize a good trajectory
- Idea: Learn a reward function with human help
- However, some problems:
  - We don't have examples for learning

DRL from Human Preferences (Christiano et al. 17)

- *Deep reinforcement learning from human preferences* (Christiano et al. 17)
- Problem: we don't know how to define a good reward function for a task,
- ... but we know to recognize a good trajectory
- Idea: Learn a reward function with human help
- However, some problems:
  - We don't have examples for learning $\rightarrow$ let's the agent generate examples for evaluation

DRL from Human Preferences (Christiano et al. 17)

- *Deep reinforcement learning from human preferences* (Christiano et al. 17)
- Problem: we don't know how to define a good reward function for a task,
- ... but we know to recognize a good trajectory
- Idea: Learn a reward function with human help
- However, some problems:
  - We don't have examples for learning $\rightarrow$ let's the agent generate examples for evaluation
  - We don't know to assign numbers to each trajectory

DRL from Human Preferences (Christiano et al. 17)

- *Deep reinforcement learning from human preferences* (Christiano et al. 17)
- Problem: we don't know how to define a good reward function for a task,
- ... but we know to recognize a good trajectory
- Idea: Learn a reward function with human help
- However, some problems:
  - We don't have examples for learning $\rightarrow$ let's the agent generate examples for evaluation
  - We don't know to assign numbers to each trajectory ... but we know to generate *preferences* (if a trajectory is better than another)

DRL from Human Preferences (Christiano et al. 17)

Proposal

Learn a DNN that **modelize the reward function** and that **adheres to preferences of the human** for **trajectories generated by the agent**!

DRL from Human Preferences (Christiano et al. 17)

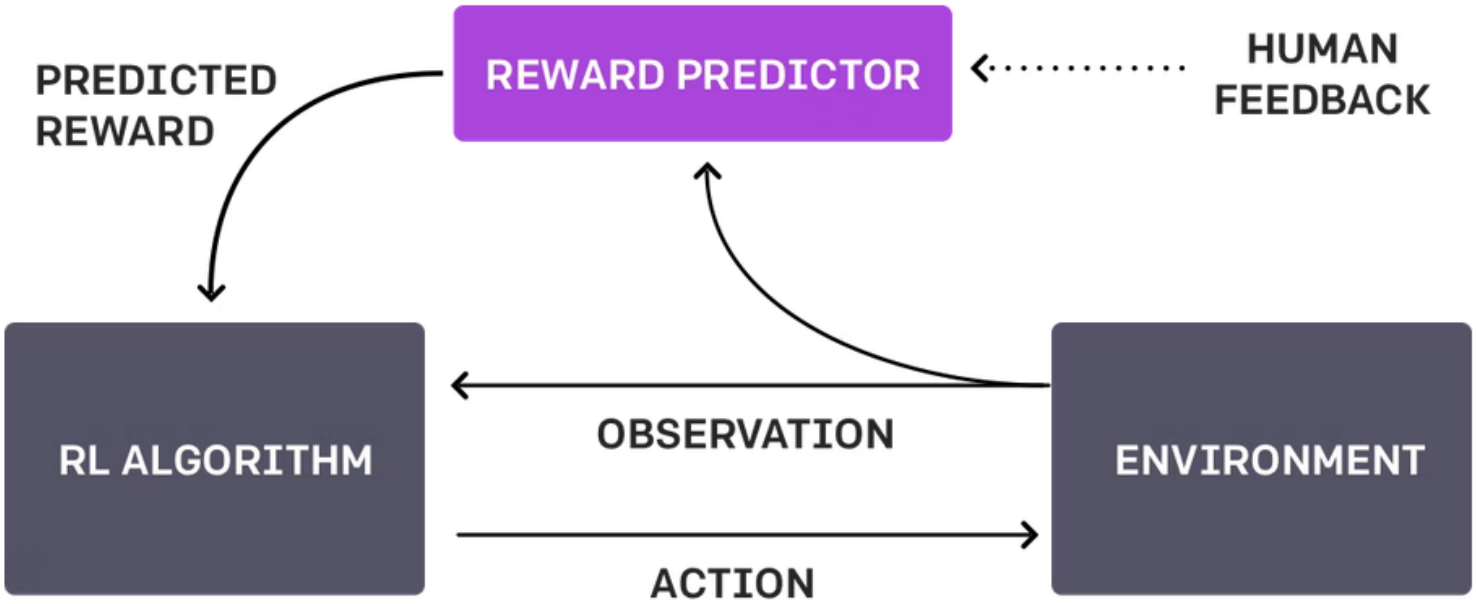
Proposal

Learn a DNN that **modelize the reward function** and that **adheres to preferences of the human** for **trajectories generated by the agent**!

- Still some problems:
  - Humans don't have time to evaluate a lot of examples
  - Examples change with the policy (become more competitive)
- Solution will be to include the human in learning loop

DRL from Human Preferences (Christiano et al. 17)

Reinforcement Learning with Human Feedback (RLHF)



DRL from Human Preferences (Christiano et al. 17)

- Example: Train Hopper to make flips
- Not defined task in Mujoco for Hopper[2]
- Authors applied the RLHF to the task.
- Repeat until good behavior is obtained:
  1. Agent generates two trajectories and ask for preferences to the human
  2. Human shows preference and it is added to set of preferences
  3. Reward function is trained to generate rewards according to preferences
  4. For some steps, agent is trained using RL with that reward function
- See video here

[2] Reward function in fact, can be defined by hand. However, it took two hours of hard work for researchers to develop it.

DRL from Human Preferences (Christiano et al. 17)

- Loss function for reward function:

$$\text{loss}(\hat{r}) = -\sum_{(\sigma^1,\sigma^2,\mu)\in\mathcal{D}} \mu(1)\log\hat{P}\left[\sigma^1 \succ \sigma^2\right] + \mu(2)\log\hat{P}\left[\sigma^2 \succ \sigma^1\right]$$

  where σ are trajectories (or pieces of trajectory) and $\mu(i)$ indicated preferences of trajectory i or not.

- Probability $\hat{P}\left[\sigma^1 \succ \sigma^2\right]$ of preference according r is defined as soft-max:

$$\hat{P}\left[\sigma^1 \succ \sigma^2\right] = \frac{\exp\sum\hat{r}\left(o_t^1, a_t^1\right)}{\exp\sum\hat{r}\left(o_t^1, a_t^1\right) + \exp\sum\hat{r}\left(o_t^2, a_t^2\right)}$$

  where the sums are done for all pairs o_i, a_i of state, action of the trajectory

DRL from Human Preferences (Christiano et al. 17)

- Other tricks in implementation (see paper): ensembles, normalization,

DRL from Human Preferences (Christiano et al. 17)

- Other tricks in implementation (see paper): ensembles, normalization,

- For Hopper to learn to flip 900 preferences were needed. Each preference was decided in 3-5 seconds $\rightarrow$ in approx. 1 hour they trained the agent.
- Compared with standard approach, only generating the reward function took two hours
- The behavior obtained was not so elegant (something common in Mujoco) compared with RLHF

DRL from Human Preferences (Christiano et al. 17)

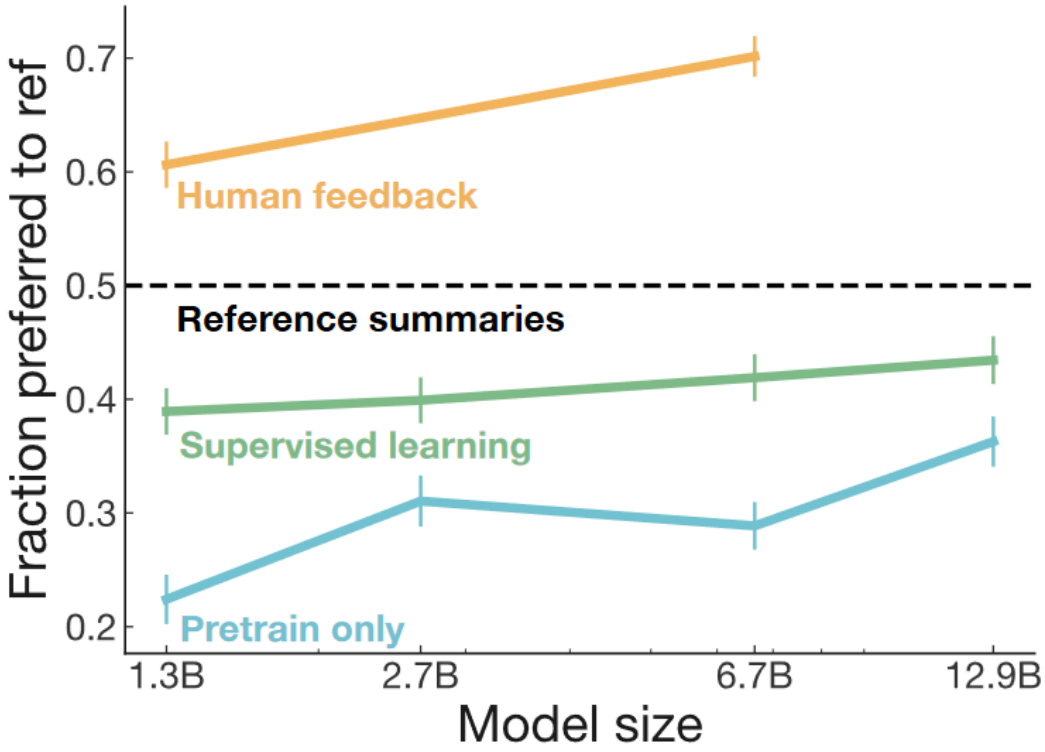
- Other tricks in implementation (see paper): ensembles, normalization,
- For Hopper to learn to flip 900 preferences were needed. Each preference was decided in 3-5 seconds $\rightarrow$ in approx. 1 hour they trained the agent.
- Compared with standard approach, only generating the reward function took two hours
- The behavior obtained was not so elegant (something common in Mujoco) compared with RLHF
- However, some problems: Still too many labeling due to a lot of time spent in warming the policy

Summarizing application (Stiennon et al. 20)

- *Learning to summarize from human feedback* (Stiennon et al. 20)
- Application of RLHF to NLP (generic proposal (Ziegler et al. 20))
- Task: Generate summaries of texts
- Generator of the summary is an autoregressive Transformer NN implementing a Language Model
- Procedure they propose is the following:
  1. Start from a standard language GPT-like model trained on perplexity (TL;DR prompt)
  2. The language model is trained to generate summaries using a training dataset of texts and trying to reproduce summaries generated by human experts
  3. After that, RL is used to fine tune the supervised trained model to generate better summaries

Summarizing application (Stiennon et al. 20)

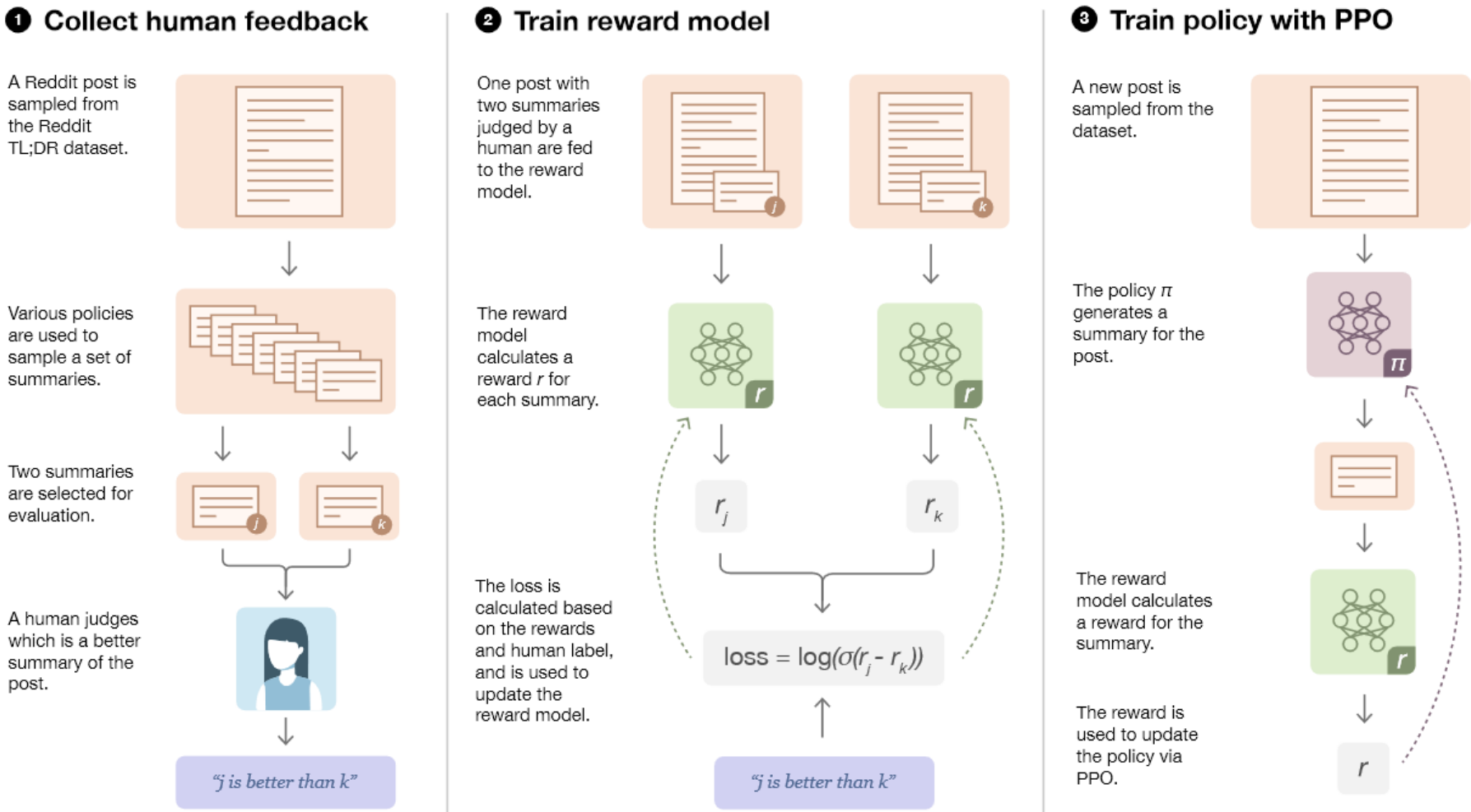
- Each stage is able to produce summaries.
- ROUGE evaluation (based on n-grams) of summaries is too simple so evaluation is done by humans



Summarizing application (Stiennon et al. 20)

- Empirically it worked better than supervised learnt model (see paper results)
- But how we obtained this result?

Summarizing application (Stiennon et al. 20)



Summarizing application (Stiennon et al. 20)

- Reward model is learnt trying to minimize the following loss:

$$\text{loss}\left(r_\theta\right) = -\mathbb{E}_{(x,y_0,y_1,i)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_i\right) - r_\theta\left(x,y_{1-i}\right)\right)\right)\right]$$

where $r_\theta(x,y)$ is reward model with parameters θ for text x and summary y. D is the dataset of human judgments.

Summarizing application (Stiennon et al. 20)

- Reward model is learnt trying to minimize the following loss:

$$\text{loss}\left(r_\theta\right) = -\mathbb{E}_{(x,y_0,y_1,i)\sim D}\left[\log\left(\sigma\left(r_\theta\left(x,y_i\right) - r_\theta\left(x,y_{1-i}\right)\right)\right)\right]$$

  where $r_\theta(x,y)$ is reward model with parameters θ for text x and summary y. D is the dataset of human judgments.

- However, PPO is trained in this another reward function:

$$R(x,y) = r_\theta(x,y) - \beta \log\left[\pi_\phi^{\text{RL}}(y \mid x)/\pi^{\text{SFT}}(y \mid x)\right]$$

  where the substracting term refer to the KL divergence between learnt policy π_ϕ^{RL} and initial policy with supervised fine learning π^{SFT} over GPT-like model

- Again, to avoid RL policy to take profit of flaws in the reward function

Summarizing application (Stiennon et al. 20)

- Some observations:
  - No warmup problem
  - No "human in the loop" (so need for KL divergence)
  - No concept of state, huge state space (50k actions)
  - **Summaries aligned with human preferences**

Summarizing application (Stiennon et al. 20)

- Some observations:
  - No warmup problem
  - No "human in the loop" (so need for KL divergence)
  - No concept of state, huge state space (50k actions)
  - **Summaries aligned with human preferences**
- Can you see what comes next?

InstructGPT (Ouyang et al. 22)

- *Training language models to follow instructions with human feedback* paper describes InstructGPT version of ChatGPT (text-davinci-002) we use nowadays
- It uses RLHF to align language model (GPT-3) to do what humans ask.
- A lot labeling of data done.
- Follow the three steps methodology: Supervised + Learn reward model + Train language model to generate reward

InstructGPT (Ouyang et al. 22)

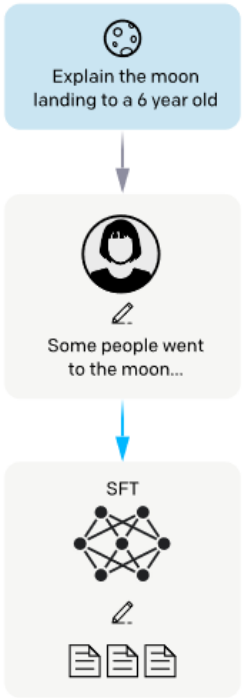
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



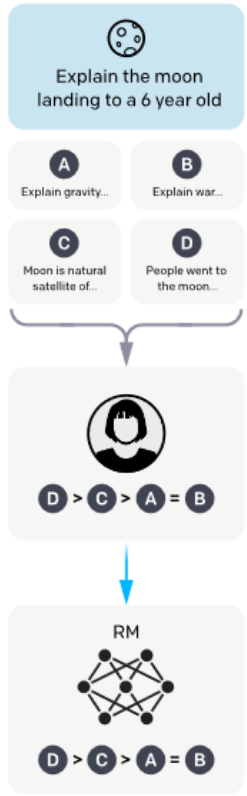
Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

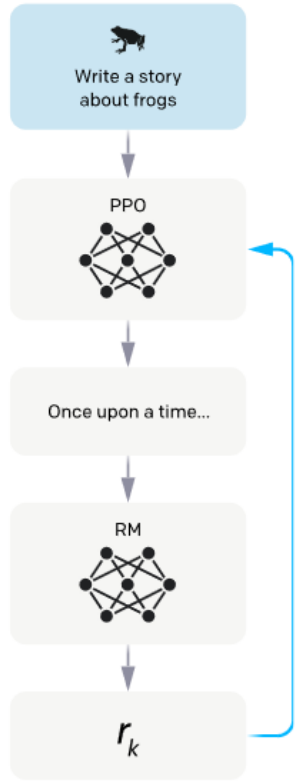
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



InstructGPT (Ouyang et al. 22)

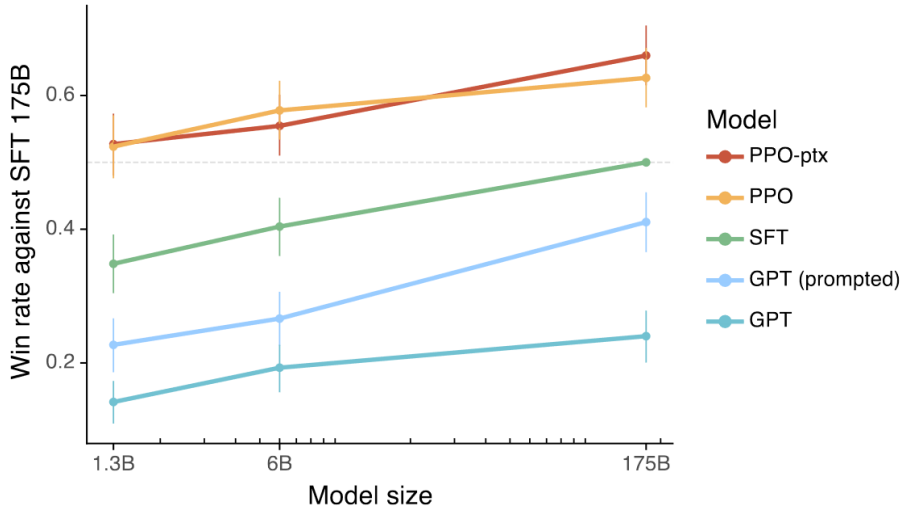
- Not a lot of details on how the RL is trained.
- Only some small differences:
  - Amount of answers for each prompt (more data for training)
  - In a variation of PPO (PPO-ptx), they add in the loss an extra term to not deviate further away from dataset used for training

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}} \left[r_\theta(x,y) - \beta \log \left(\pi_\phi^{\mathrm{RL}}(y \mid x) / \pi^{\mathrm{SFT}}(y \mid x) \right) \right] + \gamma \mathbb{E}_{x\sim D_{\text{pretrain}}} \left[\log \left(\pi_\phi^{\mathrm{RL}}(x) \right) \right]$$

- They still do not iterate the RLHF

InstructGPT (Ouyang et al. 22)

- Results:



- Awesome results.
- Team of 40 labelers
- Small costs of finetuning compared to train GPT3
  - 3600 petaflops-day to train GPT3
  - 5 petaflops-day for SL + 60 petaflops-day to RL fine-tuning

Final words about RLHF on Language models

- A lot of implementation details. See here for a review here
- ChatGPT and InstructGPT are not the only trained models using this approach (Claude, Mistral, Gemini).
- RL method is on-policy (why?)!
- People usually do not close the loop
- Not a lot of detail in paper! Papers start to enclose information about their research because of the possible economical impact

Some of latest developments

- Variations of RLHF:
  - **Safe RLHF** (Dai et al. 23): Combine alignment criteria that might conflict with each other sometimes
  - Reinforcement Learning from AI Feedback (**RLAIF**) (Lee et al. 23): Reduce human labelling using LLM to generate preferences. Also **Self-Rewarding Language Models** (Lee et al. 24)
  - **Constitutional AI**: (Bai et al 22)
- **Pairwise PPO**: (Tianhao Wu et al. 23)
- **DPO**: Direct Preference Optimization translates the RL problem to a classification problem (Rafailov et al. 23)
- **WARM**: Ensembles of reward functions to avoid *Reward Hacking* (Ramé et al. 24)

Direct Preference Optimization (DPO)

- Very popular because it should return same results from preferences like RLHF but easier because it does not involve the learning of a reward model simplifying the process

Direct Preference Optimization (DPO)

- Very popular because it should return same results from preferences like RLHF but easier because it does not involve the learning of a reward model simplifying the process
- Wait... How?

Direct Preference Optimization (DPO)

1. RLHF Objective: The goal is to maximize the expected reward while regularizing with KL-divergence:

$$\max_{\pi} \mathbb{E}_{\pi}[r(x, y)] - \beta \mathrm{KL}\left(\pi \| \pi_{\mathrm{ref}}\right)$$

The solution is the optimal policy:

$$\pi^*(y \mid x) = \frac{1}{Z} \pi_{\mathrm{ref}}(y \mid x) \exp\left(\frac{r(x, y)}{\beta}\right)$$

2. Express Reward in Terms of Policy: Rearrange the optimal policy to solve for $r(x, y)$:

$$r(x, y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta \log Z(x)$$

where $Z(x)$ is the partition function. For pairwise preferences, $Z(x)$ cancels out.

Direct Preference Optimization (DPO)

3. Bradley-Terry Preference Model: Human preferences are modeled as:

$$P\left(y_w \succ y_l \mid x\right) = \sigma\left(r\left(x, y_w\right) - r\left(x, y_l\right)\right)$$

Substitute the expression for $r(x, y)$ into this model:

$$r\left(x, y_w\right) - r\left(x, y_l\right) = \beta \log \frac{\pi^*\left(y_w \mid x\right)}{\pi_{\text{ref}}\left(y_w \mid x\right)} - \beta \log \frac{\pi^*\left(y_l \mid x\right)}{\pi_{\text{ref}}\left(y_l \mid x\right)}$$

Simplify to:

$$r\left(x, y_w\right) - r\left(x, y_l\right) = \beta \log \frac{\pi^*\left(y_w \mid x\right) \pi_{\text{ref}}\left(y_l \mid x\right)}{\pi^*\left(y_l \mid x\right) \pi_{\text{ref}}\left(y_w \mid x\right)}$$

4. DPO Reparameterization: Replace π^* with the learnable policy π_θ, yielding:

$$P\left(y_w \succ y_l \mid x\right) = \sigma\left(\beta \log \frac{\pi_\theta\left(y_w \mid x\right) \pi_{\text{ref}}\left(y_l \mid x\right)}{\pi_\theta\left(y_l \mid x\right) \pi_{\text{ref}}\left(y_w \mid x\right)}\right)$$

Direct Preference Optimization (DPO)

5. DPO Loss Function: Maximize the log-likelihood of observed preferences:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta\left(y_w \mid x\right)\pi_{\text{ref}}\left(y_l \mid x\right)}{\pi_\theta\left(y_l \mid x\right)\pi_{\text{ref}}\left(y_w \mid x\right)}\right)\right]$$

That brings us the final DPO expression:

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta\left(y_w \mid x\right)}{\pi_{\text{ref}}\left(y_w \mid x\right)} - \beta\log\frac{\pi_\theta\left(y_l \mid x\right)}{\pi_{\text{ref}}\left(y_l \mid x\right)}\right)\right]$$

Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training

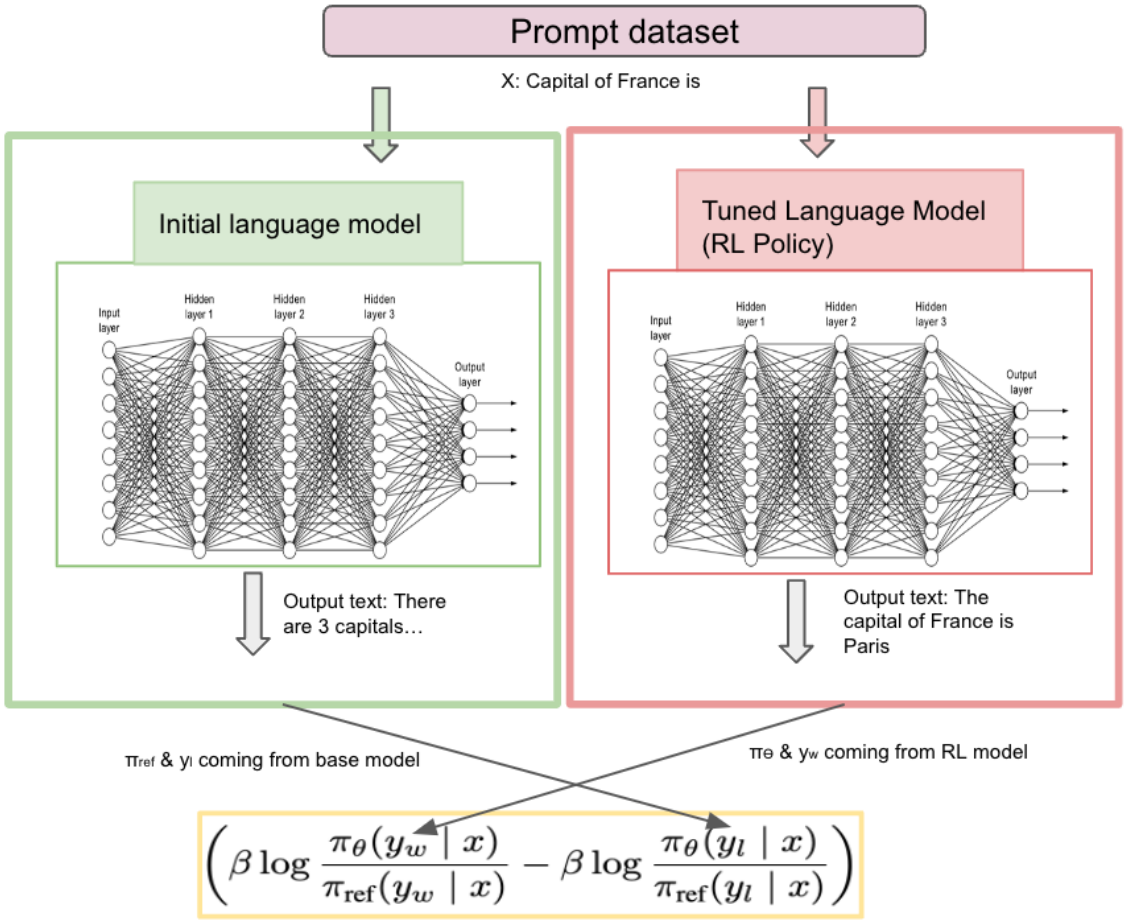
Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training
- A lot used in fine-tuning LLM together with LORA on attention weights

Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training
- A lot used in fine-tuning LLM together with LORA on attention weights
- But we don't have the reward function anymore with associated problems (explainability, guarantee checks, ...)

Direct Preference Optimization (DPO)



Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training

Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training
- A lot used in fine-tuning LLM together with LORA on attention weights

Direct Preference Optimization (DPO)

- Like in RLHF you only need pairs of preferences
- Easy to implement: We can find the policy without the reward function and no need to run RL
- More efficient in training
- A lot used in fine-tuning LLM together with LORA on attention weights
- But we don't have the reward function anymore with associated problems (explainability, guarantee checks, ...)

Deepseek and GRPO

- Recently Deepseek(Deepseek 25) did an interesting work on LLM and RL
- They trained different LLM, all of them trained with a new RL algorithm Group Relative Policy Optimization (GRPO)
  - DeepSeek-V3
  - DeepSeek-R1-Zero
  - DeepSeek-R1
- DeepSeek-V3 uses a Mixture-of-Experts dynamically activating only a subset of parameters per token, optimizing computational efficiency and reducing costs

Deepseek and GRPO

- DeepSeek-R1-Zero, built upon DeepSeek-V3 **architecture**, pushes boundaries of RL application to LLM by **avoiding the Supervised Fine-Tuning step** and it is trained to reason
- DeepSeek-R1 did some finetuning on reasoning with pairs of examples
- These methods have been trained using GRPO
- They propose several changes to PPO in GRPO:
  - Use an ensemble of reward functions that are hand made (no discovery of the reward function)
  - Compare groups of answers instead of pairs
  - Avoid Value functions (usually the size of the policy)
  - Adding a KL term in the PPO loss function

Deepseek and GRPO

GRPO loss:

$$J(\theta) = \frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta\left(a_i \mid s\right)}{\pi_{\theta_{\text{old}}}\left(a_i \mid s\right)}A_i, \text{clip}\left(\frac{\pi_\theta\left(a_i \mid s\right)}{\pi_{\theta_{\text{old}}}\left(a_i \mid s\right)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta D_{KL}\left(\pi_\theta \,\|\, \pi_{\text{ref}}\right)\right)$$

- Uses PPO clip loss with Importance Sampling but:
- Notice sum for G Advantages (see next slide)
- Also notice introduction in PPO loss of KL term

Deepseek and GRPO

- Given a prompt the system returns a **group** of G answers
- Each answer is evaluated using a set of hand made reward functions (when it can be done, f.i accuracy of mathematical computations) or learnt models trained on DeepSeek-V3 SFT checkpoints[3]).
- Advantage of one answer is compared as a standarized advantage over the reward of other answers
- Advantages computation:

$$A_i = \frac{r_i - \text{mean}\left(r_1, r_2, \cdots, r_G\right)}{\text{std}\left(r_1, r_2, \cdots, r_G\right)}$$

[3]Not very clear for me. See end of page 29 of (Deepseek 25) paper.

Deepseek-R1

- Deepseek-R1-Zero worked well but had some problems in format and answers
- Deepseek-R1 was Fine tuned *after* the RL training with a set of 600k examples generated from other sources (including itself)