Reinforcement Learning from Human Feedback (Preference-based RL)

Classical RL with a reward function

In standard RL, we assume a scalar reward function $r(s, a)$ for each state and action, with cumulative return:

$$R(\tau) = \sum_{t=1}^{H} r(s_h, a_h).$$

- The trajectory τ is a sequence of H state-action pairs $(s_1, a_1, s_2, a_2, \cdots, s_H, a_H)$.
- This works well when the reward function truly captures the task objective.
- In many real problems, we only have a hand-crafted reward **proxy** for what we actually want (e.g., goal + reward shaping).

Reward misspecification and reward hacking

Reward misspecification

The handcrafted reward fails to capture the true objective.

Reward hacking

The agent identifies a strategy that maximizes the proxy reward but fails to accomplish the task.

Reward misspecification and reward hacking

Reward misspecification

The handcrafted reward fails to capture the true objective.

Reward hacking

The agent identifies a strategy that maximizes the proxy reward but fails to accomplish the task.

RL often fails because optimization is **too successful on the wrong reward**.

Examples of reward hacking in classical RL

Example

- **Navigation/robotics:** We reward progress to the goal, but the agent learns to trigger the progress bonuses repeatedly without reaching the goal.

Challenge

It is hard to design a scalar reward to capture safety, robustness, and human satisfaction.

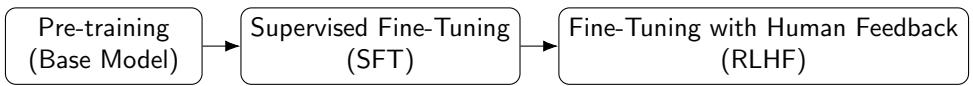
Preference-based RL

Agents should learn from human feedback rather than from handcrafted reward signals.

A "Famous" Example

GPT 3

Both the base model and SFT are based on supervised learning, and RLHF is based on reinforcement learning.



What is a GPT Transformer?

Next token prediction: Example

The cat sat "?"

GPT Transformer

- Step 1: Input text $\Rightarrow$ tokens $\Rightarrow$ token embeddings and position embeddings
- Step 2: Embedding $\Rightarrow$ Transformer block 1 $\Rightarrow$ Transformer block 2 $\cdots$ Transformer block L.
- Step 3: Linear layer $\Rightarrow$ Softmax over vocabulary $\Rightarrow$ Next-token distribution

GPT Transformer: Step 1

Tokenization and Embedding: Example

Word	Token	Embedding
The	101	$[0.2,\ -1.1,\ 0.7,\ 0.4]^\top$
Cat	572	$[-0.5,\ 0.3,\ 1.4,\ -0.8]^\top$
Sat	913	$[1.0,\ 0.1,\ -0.6,\ 0.9]^\top$

GPT Transformer: Step 1

Tokenization and Embedding: Example

Word	Token	Embedding
The	101	$[0.2,\ -1.1,\ 0.7,\ 0.4]^\top$
Cat	572	$[-0.5,\ 0.3,\ 1.4,\ -0.8]^\top$
Sat	913	$[1.0,\ 0.1,\ -0.6,\ 0.9]^\top$

Why embedding?

We can measure similarity and closeness of two words based on the embedding vectors.

GPT Transformer: Step 1

Position embedding

Position	Position Embedding
1	$[0.01,\ 0.02,\ 0.03,\ 0.04]^\top$
2	$[0.05,\ 0.06,\ 0.07,\ 0.08]^\top$
3	$0.09,\ 0.10,\ 0.11,\ 0.12]^\top$

Why position embedding?

Tell the model where a token is. So “cat sat” and “sat cat” are different.

GPT Transformer: Step 1

Token and Position embedding

Word	Token	Embedding
The	101	$[0.2,\ -1.1,\ 0.7,\ 0.4]^\top$
Cat	572	$[-0.5,\ 0.3,\ 1.4,\ -0.8]^\top$
Sat	913	$[1.0,\ 0.1,\ -0.6,\ 0.9]^\top$

$+$

Position	Position Embedding
1	$[0.01,\ 0.02,\ 0.03,\ 0.04]^\top$
2	$[0.05,\ 0.06,\ 0.07,\ 0.08]^\top$
3	$0.09,\ 0.10,\ 0.11,\ 0.12]^\top$

GPT Transformer: Step 1

Token and Position embedding

Word	Token	Embedding
The	101	$[0.2,\ -1.1,\ 0.7,\ 0.4]^\top$
Cat	572	$[-0.5,\ 0.3,\ 1.4,\ -0.8]^\top$
Sat	913	$[1.0,\ 0.1,\ -0.6,\ 0.9]^\top$

$+$

Position	Position Embedding
1	$[0.01,\ 0.02,\ 0.03,\ 0.04]^\top$
2	$[0.05,\ 0.06,\ 0.07,\ 0.08]^\top$
3	$0.09,\ 0.10,\ 0.11,\ 0.12]^\top$

Word (x_i)	Embedding h_i
The	$[0.21,\ -1.08,\ 0.73,\ 0.44]^\top$
Cat	$[-0.45,\ 0.36,\ 1.47,\ -0.72]^\top$
Sat	$[1.09,\ 0.2,\ -0.49,\ 1.02]^\top$

GPT Transformer: Step 2

Self Attention (Query, Key, Value)

- Query: information I am looking for.

$$Q = HW_Q = \begin{pmatrix} 0.21 & -1.08 & 0.73 & 0.44 \\ -0.45 & 0.36 & 1.47 & -0.72 \\ 1.09 & 0.2 & -0.49 & 1.02 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1.73 & -0.14 \\ -1.53 & 1.38 \\ 1.91 & 0.80 \end{pmatrix}$$

- Key: information I have

$$K = HW_K = \begin{pmatrix} 0.21 & -1.08 & 0.73 & 0.44 \\ -0.45 & 0.36 & 1.47 & -0.72 \\ 1.09 & 0.2 & -0.49 & 1.02 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -2 & 1 \\ 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 3.54 & -0.79 \\ -0.42 & 2.55 \\ 1.22 & -1.31 \end{pmatrix}$$

GPT Transformer: Step 2

Score Matrix

$$S = \frac{QK^\top}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}} H W_Q W_K^\top H^\top$$

where $S(i, j)$ is the relevance of token j to token i.

Attention-Weight Matrix and Causal Masking

$A = [A_{ij}]$ such that

$$A_{ij} = \frac{e^{S_{ij}}}{\sum_{m \le i} e^{S_{im}}} \quad j \le i; \quad A_{ij} = 0 \quad j > i.$$

Token i can only attend to earlier tokens.

GPT Transformer: Step 2

Attention-Weight Matrix and Causal Masking: Example

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0.3 & 0.7 & 0 \\ 0.2 & 0.5 & 0.3 \end{pmatrix}$$

Output of Attention

Value $V = HW_V$ is the information to be retrieved. The output of attention is

$$Z = AVW_O,$$

where W_O map the output a $n \times d$ matrix (the same as H).

Let $H \leftarrow H + AVW_O$ and pass it to the next attention block.

GPT Transformer: Step 3

Linear layer and next-token distribution

Input the representation of the last token $h_n^{(L)}$ to a a feed-forward network. The output (logit) is

$$o_n = h_n^{(L)} W_{FF} + b,$$

where the dimension of o_n is the vocabulary size. For example,

$$o_n = [1.2,\ 0.4,\ 3.8,\ -0.7]^\top.$$

Next token distribution

$$p_n = \text{softmax}(o_n) = \left[\frac{e^{o_{n,i}}}{\sum_j e^{o_{n,j}}} \right]^\top$$

GPT Transformer: Multi-head attention

Each attention block has multiple heads. Each head, say head m, has its own $W_Q^{(m)}$, $W_K^{(m)}$, and $W_V^{(m)}$. The outputs are concatenated and projected

$$\tilde{Z} = [A_1 V_1, \cdots, A_M V_M] \tilde{W}_O.$$

GPT Transformer: Training

Use the cross-entropy loss for next-token prediction. Given a training sequence $(x_1, x_2, \cdots, x_N)$. The loss is

$$\mathcal{L} = \frac{1}{N} \sum_t - \log P(x_{t+1} | x_1, \cdots, x_t)$$

GPT Transformer: Training

Use the cross-entropy loss for next-token prediction. Given a training sequence $(x_1, x_2, \cdots, x_N)$. The loss is

$$\mathcal{L} = \frac{1}{N} \sum_t -\log P(x_{t+1}|x_1, \cdots, x_t)$$

Base Model

Next token prediction with all Internet data.

SFT

Next token prediction with human demonstrations (prompt, ideal response). Each demonstration becomes a sequence $(x_1, x_2, \cdots, x_m)$.

Reference

- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention is all you need.* Advances in neural information processing systems 30 (2017).
- Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan et al. *Language models are few-shot learners.* Advances in neural information processing systems 33 (2020): 1877-1901.
- Ouyang, Long, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang et al. *Training language models to follow instructions with human feedback.* Advances in neural information processing systems 35 (2022): 27730-27744.
- GPT 5.4 Thinking