RLHF

MUSTAFA HAJIJ

Policy gradient

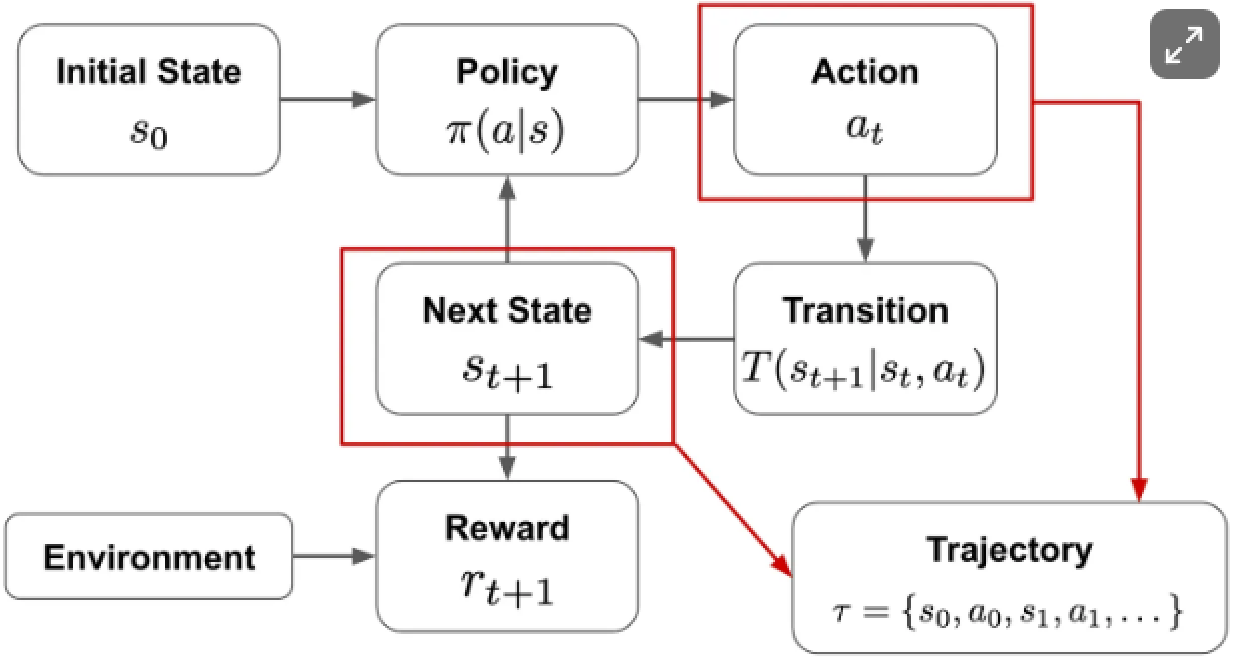
$$s \in \mathcal{S} \quad \text{(State)} \qquad a \in \mathcal{A} \quad \text{(Action)} \qquad r_s \in \mathbb{R} \quad \text{(Reward)}$$

$$T(s_{t+1}|s_t, a_t) \quad \text{(Transition)} \qquad \pi(a|s) \quad \text{(Policy)}$$

$$\pi_\theta(a|s) \quad \text{(Policy)}$$

Policy has parameters θ

Policy gradient



Policy gradient

- **Objective**: Maximize the expected return of a stochastic, parameterized policy, π_w.
- **Expected Return**:

$$J(\pi_w) = \mathbb{E}_{\tau \sim \pi_w}[R(\tau)]$$

Where $R(\tau)$ is the total rewrad.

Policy gradient

- **Optimizing the Policy by Gradient Ascent**:

$$w_{k+1} = w_k + \alpha \nabla_w J(\pi_w)$$

- The gradient, $\nabla_w J(\pi_w)$, is the **policy gradient**(Vanilla Policy Gradien).

Policy gradient

$$\tau = \{s_0, a_0, s_1, a_1, \ldots, s_t, a_t\} \quad \text{(Trajectory)}$$

$$R(\tau) = \sum \gamma^t r_{s_t} \quad \text{(Return)}$$

Policy gradient

1. **Probability of a Trajectory**: Given a trajectory $\tau = (s_0, a_0, \ldots, s_{T+1})$ with actions from π_w:

$$P(\tau|w) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_w(a_t|s_t)$$

2. **The Log-Derivative Trick**: The derivative of $\log(u)$ is $\frac{\nabla u}{u}$. By rearrangement $\nabla u = u \nabla log(u)$:

$$\nabla_w P(\tau|w) = P(\tau|w)\nabla_w \log P(\tau|w)$$

Policy gradient

3. **Log-Probability of a Trajectory**:

$$\log P(\tau|w) = \log \rho_0(s_0) + \sum_{t=0}^{T} (\log P(s_{t+1}|s_t, a_t) + \log \pi_w(a_t|s_t))$$

4. **Gradients of Environment Functions**: The environment has no dependence on w, so gradients of $\rho_0(s_0)$, $P(s_{t+1}|s_t, a_t)$, and $R(\tau)$ are zero.

Policy gradient

- **Grad-Log-Prob of a Trajectory**:
  The gradient of the log-prob of a trajectory is:

$$\nabla_w \log P(\tau|w) = \cancelto{0}{\nabla_w \log \rho_0(s_0)} +$$

$$\sum_{t=0}^{T} \left(\cancelto{0}{\nabla_w \log P(s_{t+1}|s_t, a_t)} + \nabla_w \log \pi_w(a_t|s_t) \right)$$

  Simplifying, we get:

$$\nabla_w \log P(\tau|w) = \sum_{t=0}^{T} \nabla_w \log \pi_w(a_t|s_t)$$

Policy gradient

$$
\begin{aligned}
\nabla_w J(\pi_w) &= \nabla_w \underset{\tau \sim \pi_w}{\mathrm{E}}[R(\tau)] \\
&= \nabla_w \int_\tau P(\tau \mid w) R(\tau) \\
&= \int_\tau \nabla_w P(\tau \mid w) R(\tau) \\
&= \int_\tau P(\tau \mid w) \nabla_w \log P(\tau \mid w) R(\tau) \\
&= \underset{\tau \sim \pi_w}{\mathbf{E}}[\nabla_w \log P(\tau \mid w) R(\tau)] \\
\therefore \nabla_w J(\pi_w) &= \underset{\tau \sim \pi_w}{\mathbf{E}}\left[\sum_{t=0}^{T} \nabla_w \log \pi_w(a_t \mid s_t) R(\tau)\right]
\end{aligned}
$$

Policy gradient

- The policy gradient is an expectation, which can be estimated via sample mean.
- Using trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$ from the policy π_w, we get:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_w \log \pi_w(a_t|s_t) R(\tau)$$

- $|\mathcal{D}|$ represents the number of trajectories (N in this case).
- This expression is our desired computable form.
- With a policy that allows $\nabla_w \log \pi_w(a|s)$ calculations and by collecting trajectory datasets, we can compute the gradient and update.

Addressing GPT Challenges - Training Strategy Overview

- Supervised Fine-Tuning (SFT):

  Refines a pre-trained GPT-3 model's responses for specific tasks or guidelines, enhancing its understanding and output relevance.

- Training a Reward Model (RM):

  Develops a system that assesses the quality of text generated by the model, guiding it towards human-preferred responses

- Reinforcement Learning from Human Feedback (RLHF)

Supervised Fine-Tuning (SFT)

Source: OpenAI API with InstructGPT.

Human-in-the-loop: Integral part of the process to ensure quality and diversity.

Goal: Capture a broad spectrum of language patterns and ensure diversity in prompts.

Supervised Fine-Tuning (SFT)

Objective: Train the model to generate desired responses to given prompts.

Method:

- Concatenate the prompt and the desired response.
- Use this concatenated text as input for the model.
- Train the model to predict the next token in the sequence.

Training a Reward Model (RM)

1. Generating Responses: For a given prompt, produce multiple responses using the SFT model.

2. Pairing & Ranking: Create pairs from these responses and have a human rank the better response in each pair.
    - Prompt: "What is ice?"
    - Responses:
        - A. "It's the solid form of water."
        - B. "Frozen water that's cold to touch."
        - C. "Water that has turned solid due to low temperatures."
        - D. "A crystalline substance formed when water freezes."

Training a Reward Model (RM)

Create Response Pairs: Pair the responses with each other.

- Total Pairs: C(4,2) = 6 pairs
- Example Pairs:
  1. (A, B)
     A: It's the solid form of water.
     B: Frozen water that's cold to touch.
  2. (A, C)
  3. (A, D)
  4. (B, C)
  5. (B, D)
  6. (C, D)

Human Ranking: Ask a human to rank responses within each pair.

- For pair (A, B), the human might prefer response A over B.

Training a Reward Model (RM)

Total Prompts Used: 33,000

- Responses per Prompt: Between 4 to 9

- Sample Calculation:

- For 4 responses per prompt: $33{,}000 \times C(4,2) = 198{,}000$ pairs
- For 9 responses per prompt: $33{,}000 \times C(9,2) = 1{,}188{,}000$ pairs

Training a Reward Model (RM)

- Objective: Align model generation with human preference.
- Method: Rate model responses based on human preference.
- Outcome: Train a model (Reward Model) to simulate these ratings.

Reward Model Architecture

Starting Point: Use the SFT model from Phase 1.

Modifications:

- Remove the last linear layer (unembedding layer).
- Add a randomly initialized linear layer.
- The model now outputs a scalar value, effectively becoming a regressor.

Training for Reward Model Architecture

Pair Selection:

- Losing Pair: "What is Ice? It's the solid form of water."
- Winning Pair: "What is ice? Water that has turned solid due to low temperatures."

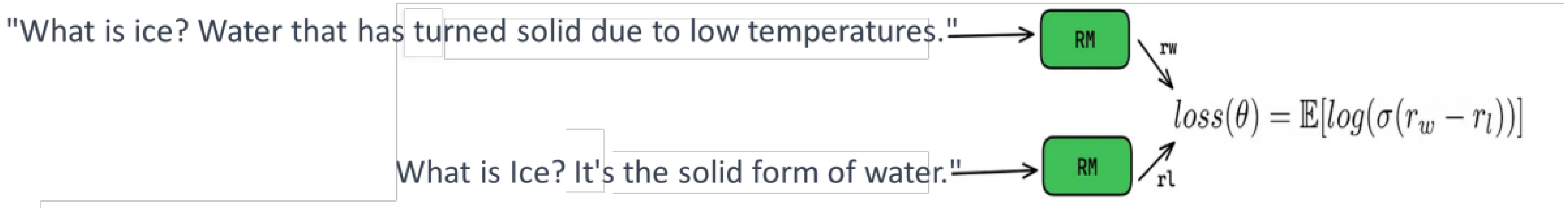
Training for Reward Model Architecture

Pair Selection:

- Losing Pair: "What is Ice? It's the solid form of water."
- Winning Pair: "What is ice? Water that has turned solid due to low temperatures."

Training for Reward Model Architecture

1. Winning Reward: Pass the winning prompt-response to the reward model.
2. Losing Reward: Pass the losing prompt-response to the same model.
3. Difference: Calculate the difference between the two rewards.



Introduction to Reinforcement Learning Model (RL)

Objective: Train a model that generates text aligning with human preferences.

- Components:
    - SFT Model: Generates responses but may not align with human preferences.
    - Reward Model: Provides a scalar reward but doesn't generate text.
- Goal of RL: Combine the capabilities of the above models to produce text that maximizes the reward.

Aligning Generation with Human Preference

- The RL model's aim is to maximize the reward.
- This can be formulated mathematically as an objective maximization problem.
- Objective:

$$J(w) = \mathbb{E}_{x \sim p_w}[r(x)]$$

  where:
  - $J(w)$ is the objective function.
  - x is the generated response.
  - $r(x)$ is the reward for response x.

Training the RL Model

- Use gradient ascent to maximize the objective function.
- Gradient Ascent Update:

$$w \leftarrow w + \alpha \nabla_w J(w)$$

where:

  - α is the learning rate.
  - $\nabla_w J(w)$ is the gradient of the objective function.

Training the RL Model

With some mathematical manipulations, the gradient expression becomes:

$$\nabla_w J(w) = \mathbb{E}_{x \sim p_w} [r(x) \nabla_w \log p_w(x)]$$

$$= \frac{1}{D} \sum_{D} \sum_{t=0}^{T} \nabla_w r(x) \log p_w(x_t | x_{t-1}, x_{t-2}, \ldots)$$

where:

- D refers to all the prompt-response pairs.

Training the RL Model

1. The model generates a token x_t based on the previous tokens $x_0, x_1, \ldots, x_{t-1}$.
2. It continues to generate tokens until the sequence is complete.
3. After generating the entire sequence, the Reward Model (RM) evaluates it and assigns a reward $r(x)$.
4. This reward informs the gradient for the policy gradient update.

Note: During sequence generation, the model is unaware of the sequence's reward. It learns the reward only post-generation, using it to refine its parameters for better future sequences.

Training the RL Model

1. Feed a prompt to the model.
2. The model generates a corresponding response.
3. Concatenate the prompt and response.
4. Pass this concatenated pair to the reward model to obtain a reward score.
5. Use the reward and probabilities in the gradient equation.
6. Update the model parameters using this gradient.

Challenge with Direct Model Updates

- Using the initial training method can make the model unpredictable.
- These updates can push the model to produce text that doesn't make sense or is off-topic

Solution: KL Divergence

Ensure updates don't deviate too much from the SFT trained in phase 1

- Use KL divergence to measure the "distance" between the SFT and the RL model.
- KL divergence compares two distributions.

Training Process

1. For a given prompt, the RL model generates a response.

2. At each generation step, a probability distribution over the vocabulary is produced.

3. Feed the same prompt-response to the SFT model to get its probability distribution over the vocabulary.

4. Calculate the KL divergence between the distributions from the RL model and the SFT model.

5. Subtract this divergence value from the reward for the prompt response pair

Training Process

$$J(p_w) = \mathop{\mathbb{E}}_{x \sim p_w} \left[r(x) - \beta \mathrm{KL}\left(p_w(x), p_{SFT}(x)\right)\right]$$

- $p_w(x)$ is the probability distribution over the vocabulary produced by the RL model for the sequence x.
- $p_{SFT}(x)$ is the probability distribution over the vocabulary produced by the original SFT model for the sequence x.
- β is a hyperparamete.

Training Process

Problem: • The model's performance on some datasets was inferior to the original pretrained model. Solution:

• Introduce an additional term in the cost function to keep the RL model close to the original pretrained base.

Training Process

$$J(p_w) = \mathbb{E}_{x \sim p_w}\left[r(x) - \beta \mathrm{KL}\left(p_w(x), p_{SFT}(x)\right)\right] + \gamma \mathbb{E}_{x \sim \text{pretrain}}\left[\log\left(p_w(x)\right)\right]$$

Where:

- p_w is the RL model with parameters w.
- γ is a hyperparameter, ensuring the RL model aligns with the original pretrained base.

Refs:

The Story of RLHF: Origins, Motivations, Techniques, and Modern Applications (substack.com)

Policy Gradients: The Foundation of RLHF (substack.com)

b1efde53be364a73914f58805a001731-Paper-Conference.pdf (neurips.cc)

RLHF: Reinforcement Learning from Human Feedback (huyenchip.com)

Reinforcement learning from human feedback - Wikipedia