Large Language Models and RL from Human Feedback

Instructor: Daniel Brown

[Some slides adapted from Ana Marasovic, SpinningUp in Deep RL, and others]
Course feedback is open

• Extra credit if class response rate is 70% or higher
  • Sliding scale if we reach 70%:
    • Extra credit points = response_rate_percentage / 10

• Currently at 23%
Why not just imitate behavior? (Behavioral Cloning)

What would the human do?

Action

Observation

Policy $\pi$

Action

Observation
Reward Learning
(Inverse Reinforcement Learning)

Why? What is the human’s reward function?
What if I can’t demonstrate something?
Reinforcement Learning from Human Feedback (RLHF)

Prior approaches to Inverse RL

1. Typically couldn’t do much better than the demonstrator.
   Find a reward function that explains the ranking, allowing for extrapolation.
2. Were hard to scale to complex problems.
   Reward learning becomes a supervised learning problem.
RLHF

• Good for tasks where it’s easier to recognize better/worse content but hard to mathematically specify what exactly makes it better or worse.

• Natural Language Interactions are a good example!
You
Write me a short poem about salt lake city utah

ChatGPT
In Salt Lake City, where mountains touch the sky,
And the Great Salt Lake’s waters stretch far and wide,
Where urban bustle meets nature’s serene grace,
A city of contrasts, a captivating place.

Amongst the peaks, the cityscape gleams bright,
With buildings tall, reaching for the light.
Yet in the valleys, a tranquility resides,
Where echoes of the past softly abide.

Message ChatGPT...
High-Level Recipe for ChatGPT

1. Unsupervised pre-training
2. Supervised finetuning (behavioral cloning) from human demonstrations
3. Collect preference rankings over outputs to train a reward function
4. Perform policy gradient updates using RL with learned reward
Preliminaries: Language Models

• Models that assign probabilities to sequences of words are called language models or LMs

• Language modeling: The task of predicting the next word in a sequence given the sequence of preceding words.
Neural language modeling

[BOS] → Sylvester
Neural language modeling
Neural language modeling

[BOS] Sylvester Stallone → has
Neural language modeling

[BOS] Sylvester Stallone has → made
the number of tokens in the vocabulary

the size of the vector representation up to and including the current token

representation(current token) \times output matrix

“+” softmax

i-th dimension \sim the “probability” [not really] that the next token is the i-th token in the vocabulary

select the token with the high(est) “probability” as a token to display (generate)

[BOS] Sylvester Stallone has

the logits vector

Read about other sampling strategies here: https://huggingface.co/blog/how-to-generate
Neural language modeling

Problems:
• How do we deal with different length inputs?
• How do we model long-range dependencies?
Large Language Models

Input
recite the first law $
<s> The

1 2 3 4 5 6 7 8 ... 1024
Positional encoding for token #1 + Token embedding of <s>

<s> 1 2 ... 1024
<s> a robot must obey the orders given it ...
Perform dot product between query and all keys to get a raw score for each previous word (including current word).
Normalize these scores via a softmax to get a probability distribution. Then return a weighted sum of the values.
<table>
<thead>
<tr>
<th>Word</th>
<th>Value vector</th>
<th>Score</th>
<th>Value X Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;s&gt;</td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>robot</td>
<td></td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>must</td>
<td></td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>obey</td>
<td></td>
<td>0.001</td>
<td></td>
</tr>
<tr>
<td>the</td>
<td></td>
<td>0.0003</td>
<td></td>
</tr>
<tr>
<td>orders</td>
<td></td>
<td>0.005</td>
<td></td>
</tr>
<tr>
<td>given</td>
<td></td>
<td>0.002</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td></td>
<td>0.19</td>
<td></td>
</tr>
<tr>
<td>Sum:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Decoder #12, Position #1
output vector

Token Embeddings

output token probabilities (logits)

0.19850038  aardvark
0.70898034  aarhus
0.46333563  aaron

Pick an output token based on its probability (sample)

The

DECODER

...
High-Level Recipe for ChatGPT

1. Unsupervised pre-training
2. Supervised finetuning (behavioral cloning) from human demonstrations
3. Collect preference rankings over outputs to train a reward function
4. Perform policy gradient updates using RL with learned reward
Learning a language model by reading the internet!

Table 1
Commonly used corpora information.

<table>
<thead>
<tr>
<th>Corpora</th>
<th>Type</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4 [68]</td>
<td>CommonCrawl</td>
<td><a href="https://www.tensorflow.org/datasets/catalog/c4">https://www.tensorflow.org/datasets/catalog/c4</a></td>
</tr>
<tr>
<td>CC-Stories [69]</td>
<td>CommonCrawl</td>
<td>Not open source yet</td>
</tr>
<tr>
<td>CC-News [70]</td>
<td>CommonCrawl</td>
<td><a href="https://commoncrawl.org/blog/news-dataset-available">https://commoncrawl.org/blog/news-dataset-available</a></td>
</tr>
<tr>
<td>RealNews [71]</td>
<td>CommonCrawl</td>
<td><a href="https://github.com/rowanz/grover/tree/master/realnews">https://github.com/rowanz/grover/tree/master/realnews</a></td>
</tr>
<tr>
<td>WebText</td>
<td>Reddit Link</td>
<td>Not open source yet</td>
</tr>
<tr>
<td>OpenWebText [73]</td>
<td>Reddit Link</td>
<td><a href="https://skylion007.github.io/OpenWebTextCorpus/">https://skylion007.github.io/OpenWebTextCorpus/</a></td>
</tr>
<tr>
<td>PushShift.io [74]</td>
<td>Reddit Link</td>
<td><a href="https://pushshift.io/">https://pushshift.io/</a></td>
</tr>
<tr>
<td>BigQuery [76]</td>
<td>Code</td>
<td><a href="https://cloud.google.com/bigquery">https://cloud.google.com/bigquery</a></td>
</tr>
<tr>
<td>CodeParrot</td>
<td>Code</td>
<td><a href="https://huggingface.co/codeparrot">https://huggingface.co/codeparrot</a></td>
</tr>
<tr>
<td>the Pile [77]</td>
<td>Other</td>
<td><a href="https://github.com/EleutherAI/the-pile">https://github.com/EleutherAI/the-pile</a></td>
</tr>
<tr>
<td>ROOTS [78]</td>
<td>Other</td>
<td><a href="https://huggingface.co/bigscience-data">https://huggingface.co/bigscience-data</a></td>
</tr>
</tbody>
</table>

Learning a language model by reading the internet!

• Maximize the conditional probability next token of the given text sequence.

\[
L_{LM} = \frac{1}{T} \sum_{t=1}^{T} -\log P(w_t | w_1, w_2, ..., w_{t-1})
\]

What’s the problem?

**Prompt:** “Define behavioral cloning”

**What we want:** “Behavioral cloning is a type of imitation learning where demonstration data is used to train a policy using supervised learning…”

**What we might get:** “Define reinforcement learning. Define imitation learning. Define inverse reinforcement learning. Define Q-learning ….”
Solution #1: Few-shot prompting

Prompt:
“Question: Define reinforcement learning.
Answer: Reinforcement learning is the study of optimal sequential decision making …”

Question: Define inverse reinforcement learning.
Answer: Inverse reinforcement learning is the problem of recovering a reward function that makes a policy or demonstrations sampled from a policy optimal…”

Question: Define behavioral cloning
Response:
Answer: Behavioral cloning is a type of imitation learning where…"
Other forms of useful prompting

• “Let’s think step by step.”
  • 17% to 78% improvement on some problems!
  • “Large Language Models are Zero-Shot Reasoners”

• “You are an extremely helpful expert in reinforcement learning and sequential decision making …”

• Chain-of-thought prompting
  • “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”
**Standard Prompting**

**Model Input**

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

**Model Output**

A: The answer is 27. ✗

---

**Chain-of-Thought Prompting**

**Model Input**

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

**Model Output**

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓
High-Level Recipe for ChatGPT

1. Unsupervised pre-training
2. Supervised finetuning (behavioral cloning) from human demonstrations
3. Collect preference rankings over outputs to train a reward function
4. Perform policy gradient updates using RL with learned reward
Give specific demonstrations of what we want.
Give specific demonstrations of what we want.

- Same loss function as pretraining. Cross entropy loss (classification)

\[ L_{LM} = \frac{1}{T} \sum_{t=1}^{T} -\log P(w_t | w_1, w_2, ..., w_{t-1}) \]
High-Level Recipe for ChatGPT

1. Unsupervised pre-training
2. Supervised finetuning (behavioral cloning) from human demonstrations
3. Collect preference rankings over outputs to train a reward function
4. Perform policy gradient updates using RL with learned reward
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-3.5 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 the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

The policy generates an output.
How to model as an MDP?

- X: set of possible tokens (words or pieces of words)
- State space: all possible sequences of tokens (X*).
- Initial state: task specific prompt $s_0 = (x_0, \ldots, x_m)$
- Action space: all possible tokens X
- Transitions: Deterministic. Just append action token to state to get next state. $s_{t+1} = (x_0, \ldots, x_m, a_0, \ldots, a_t, a_{t+1})$
- Reward: $r: S \times A \rightarrow \mathbb{Reals}$
Learn transformer-based reward model from preferences

$\tau_1 < \tau_2$

$\sum_{s \in \tau_1} R_\theta(s) < \sum_{s \in \tau_2} R_\theta(s)$

Bradley-Terry pairwise ranking loss

$\mathcal{L}(\theta) = - \sum_{\tau_i < \tau_j} \frac{\exp \sum_{s \in \tau_i} R_\theta(s)}{\exp \sum_{s \in \tau_i} R_\theta(s) + \exp \sum_{s \in \tau_j} R_\theta(s)}$
Reward shaping

• We don’t want the learned policy to deviate too much based on RL.
• Add a divergence term (KL divergence) to reward

\[
\hat{r}(s, a) = r(s, a) - \beta \text{KL}(\pi(\cdot|s)||\pi_0(\cdot|s)) \\
= r(s, a) - \beta (\log \pi(\cdot|s) - \log \pi_0(\cdot|s))
\]

Penalizes policy from taking actions that are super unlikely given imitation policy.

For discrete probability distributions \( P \) and \( Q \) defined on the same sample space, \( \mathcal{X} \), the relative entropy from \( Q \) to \( P \) is defined\(^{[11]} \) to be

\[
D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left( \frac{P(x)}{Q(x)} \right),
\]
Flashback: Vanilla Policy Gradient

1. Start with random policy parameters $\theta_0$

2. Run the policy in the environment to collect $N$ rollouts (episodes) of length $T$ and save returns of each trajectory.
   
   \[
   a_t \sim \pi_\theta(\cdot | s_t) \Rightarrow (s_0, a_0, r_0, s_1, a_1, r_1, ..., r_T, s_{T+1})
   
   D = \{\tau_1, ..., \tau_N\}, \quad R = \{R(\tau_1), ..., R(\tau_N)\}
   \]

3. Compute policy gradient

4. Update policy parameters
   
   \[
   \nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t) \cdot R(\tau) \right]
   \]

5. Repeat (Go to 2)
   
   \[
   \theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \bigg|_{\theta_k}
   \]
Actor (Policy) | Critic (Value Function)

I rotate the piece | Really bad action
Advantage Actor Critic (A2C)

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function

\[
\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t \right]
\]

\[
\Phi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)
\]

Policy gradient update

\[
\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \bigg|_{\theta_k}
\]

TD-Learning update

\[
w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_w V(s, a; w)
\]

TD error \( \delta_t = r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \)
Proximal Policy Optimization (PPO)

• One of the most popular deep RL algorithms
• Used to train ChatGPT and other LLMs

Motivation:
• Many Policy Gradient algorithms have stability problems.
• This can be avoided if we avoid making too big of a policy update.

https://huggingface.co/blog/deep-rl-ppo
Proximal Policy Iteration (PPO)

- Measure how much current policy changed compared with old version using a ratio:
  \[ r_t(\theta) = \frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \]

- Clip policy gradient update based on this ratio:
  \[ L_{CLIP}^{CLIP}(\theta) = \mathbb{E}_t \left[ \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right] \]
Proximal Policy Iteration (PPO)

• Simpler way to write clip objective:

\[
L(s, a, \theta_k, \theta) = \min \left( \pi_\theta(a|s) \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_\theta_k}(s, a), \ g(\epsilon, A^{\pi_\theta_k}(s, a)) \right)
\]

where

\[
g(\epsilon, A) = \begin{cases} 
(1 + \epsilon)A & A \geq 0 \\
(1 - \epsilon)A & A < 0 
\end{cases}
\]

Proximal Policy Iteration (PPO)

- Simpler way to write clip objective:

\[
L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \ g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)
\]

where

\[
g(\epsilon, A) = \begin{cases} 
(1 + \epsilon)A & A \geq 0 \\
(1 - \epsilon)A & A < 0
\end{cases}
\]

What if the advantage is positive?

\[
L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, (1 + \epsilon) \right) A^{\pi_{\theta_k}}(s, a)
\]

We want to increase \(\pi_\theta(a|s)\), but not too much!

Once \(\pi_\theta(a|s) > (1 + \epsilon)\pi_{\theta_k}(a|s)\) the min kicks in and limits our policy update.

Proximal Policy Iteration (PPO)

• Simpler way to write clip objective:

\[
L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \ g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right)
\]

where

\[
g(\epsilon, A) = \begin{cases} 
(1 + \epsilon)A & A \geq 0 \\
(1 - \epsilon)A & A < 0 
\end{cases}
\]

What if the advantage is negative?

\[
L(s, a, \theta_k, \theta) = \max \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} (1 - \epsilon), (1 - \epsilon) \right) A^{\pi_{\theta_k}}(s, a)
\]

We want to decrease \(\pi_\theta(a|s)\), but not too much!
Once \(\pi_\theta(a|s) < (1 + \epsilon)\pi_{\theta_k}(a|s)\) the max kicks in and limits our policy update.

Algorithm 1 PPO-Clip

1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, \ldots$ do
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A_{\pi_{\theta_k}}(s_t, a_t), \ g(\epsilon, A_{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left( V_\phi(s_t) - \hat{R}_t \right)^2 ,$$

typically via some gradient descent algorithm.
8: end for
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-3.5 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 the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.
Recap with alternative notations

• Start with pretrained transformer model (SFT) for policy.
• Take same model and add a linear layer that outputs a scalar for the reward model.
• Train reward model.

\[
\text{loss}(r_\theta) = -E_{(x, y_0, y_1, i) \sim D}[\log(\sigma(r_\theta(x, y_i) - r_\theta(x, y_{1-i})))]
\]

• Train PPO policy with reward and train value function.

\[
\text{objective (}\phi) = E_{(x, y) \sim D_{\pi_{\phi}^{RL}}} [r_\theta(x, y) - \beta \log(\pi_{\phi}^{RL}(y \mid x)/\pi_{\text{SFT}}(y \mid x))] + \\
\gamma E_{x \sim D_{\text{pretrain}}} [\log(\pi_{\phi}^{RL}(x))]
\]
Voila!

** We don’t know exactly what they did to train ChatGPT, but based on tech reports and prior work, this is probably very close.
You
Write a limerick
about the end of an AI class at the University of Utah

ChatGPT
In Utah, at the class's close,
AI minds buzzed, creativity rose.
With neurons a-flare,
They bid code "adieu" there,
Leaving algorithms in repose.
We made it! Next time: Review