STATS-413-HW6

Problem 1

(1) Suppose we want $\pi_\theta(y|x)$ to learn from $\pi_{teacher(y|x)}$. Our objective function is

$$J_{MLE}(\theta) = \mathbb{E}p(x)\mathbb{E}\pi_{teacher}(y|x)[\log \pi_\theta(y|x)]$$

Calculate the gradient $J'_{MLE}(\theta)$. Describe the stochastic gradient algorithm for maximizing J, where we replace expectations by sampling from p(x) and $\pi(y|x)$, thus "stochastic", where expectations are approximated by average over time. This is called imitation learning or behavior cloning.

1. **MLP**:

   we learning how the logistic regression becomes a multi layer perceptron (MLP):

   $$s_i = x_i^T w + b, \quad p_i = \sigma(s_i) = \frac{1}{1+e^{-s_i}}$$

   In a transformer-based student model $\pi_\theta(y|x)$:

   - **Input**: question/instruction x (encoded as tokens)
   - **Hidden layers**: residual stream accumulating context
   - **Output logits**: $s_y = W_{\text{unembed}} h_{\text{final}}$ for each possible token y
   - **Probability**: $\pi_\theta(y|x) = \text{softmax}(s_y)$

2. Loss function - Maximum likelihood Estimation: we know the log-likelihood loss for classification is:

$$J = \sum_{c=1}^{C} y_c s_c - \log\left(\sum_{j=1}^{C} e^{s_j}\right)$$

where y_c is the one-hot encoded true label (*what the teacher model outputs*)

3. Gradient Calculation:

   For imitation learning, we want to maximize:

$$J_{MLE}(\theta) = \mathbb{E}p(x)\mathbb{E}\pi_{teacher}(y|x)[\log \pi_\theta(y|x)]$$

Why is this "imitation"?

From Chapter 2.9 (Word Embedding), we learned that embeddings capture distributed semantic representations. When we maximize $\log \pi_\theta(y|x)$, we're pulling the model's hidden representations toward those that align with the teacher's outputs. The residual stream accumulates features batch by batch, and the MLP and attention blocks refine these representations to match the teacher's policy.

Stochastic Gradient Algorithm:

Algorithm 1: Behavior Cloning (from Chapter 2 optimization perspective)

Input: p(x), π_teacher(y|x), learning rate (η)
Initialize: θ with pre-trained weights (Chapter 5.10 initialization)

Repeat:

1. **Sample** $x \sim p(x)$
2. **Sample** $y \sim \pi_{teacher}(y|x)$ [*teacher generates ideal response*]
3. **Forward pass:** $h^{(L)} = Transformer(x)$ [*residual stream*]
4. **Logits:** $s = W_{unembed} \cdot h^{(L)}$[*unembedding*]
5. **Probability:** $p = softmax(s)$ [*classification*]
6. **Loss:** $L = -logp_y$ [cross-entropy]
7. **Backprop:** $\partial L/\partial \theta$ via chain rule [*backpropagation*]
8. Update: $\theta \leftarrow \theta + \eta \cdot \nabla_\theta log\pi_\theta(y|x)$ [*gradient ascent*]

Until convergence

---

backpropagation computes:

$$\frac{\partial J}{\partial W^{(l)}} = \frac{\partial J}{\partial s^{(l)}} (h^{(l-1)})^T$$

For the final unembed layer (Chapter 2.9), if y is the true token:

$$\frac{\partial J}{\partial W_{\text{unembed}}} = (p - \mathbf{1}_y) h^T$$

where $p - \mathbf{1}_y$ is the "error signal" that measures how far the predicted distribution is from the target. We ascend this gradient to push probability mass toward the teacher's token.

The gradient is [*using chain rule of expectations*]:

$$\boxed{J'_{MLE}(\theta) = \mathbb{E}p(x)\mathbb{E}\pi teacher(y|x) \left[\frac{\partial}{\partial \theta} \log \pi_\theta(y|x) \right]}$$

(2) Suppose we want $\pi_\theta(y|x)$ to learn by itself based on a given reward model $r(x, y)$. Our objective function is

$$J_{RL}(\theta) = \mathbb{E}_{p(x)} \mathbb{E}_{\pi_\theta(y|x)} [r(x, y)]$$

Prove

$$J'_{RL}(\theta) = \mathbb{E}_{p(x)} \mathbb{E}_{\pi_\theta(y|x)} [r(x, y) \frac{\partial}{\partial \theta} log \pi_\theta(y|x)].$$

RLHF (Reinforcement Learning from Human Feedback):

Starting with:

$$J_{RL}(\theta) = \mathbb{E}p(x)\mathbb{E}\pi_\theta(y|x)[r(x, y)]$$

where

- x = state (question)
- y = action (completion)
- $r(x, y)$ = reward (fixed)
- $\pi_\theta(y|x)$ = policy (probability of action y given state x)
- $J_{RL}(\theta)$ = expected reward = $\sum (\text{probability of action}) \text{ x } (\text{reward for the action})$
- $\mathbb{E}\pi\theta(y|x)$ = average over model's action
- $\frac{\partial}{\partial \theta} \log \pi_\theta(y|x)$ = direction to increase log-prob

Taking the gradient w.r.t θ:

$$\begin{aligned}\frac{\partial J_{RL}}{\partial \theta} &= \mathbb{E}p(x)\mathbb{E}\pi_\theta(y|x)\left[\frac{\partial}{\partial \theta} r(x, y)\right] \\ &= \frac{\partial}{\partial \theta}\mathbb{E}\pi\theta(y|x)[r(x, y)]\end{aligned}$$

Since the reward $r(x, y)$ doesn't depend on θ, we use only differentiate the policy:

- the reward $r(x, y)$ is fixed term
- only $\pi_\theta(y|x)$ depends on θ

$$\begin{aligned}&= \frac{\partial}{\partial \theta} \int r(x, y)\pi_\theta(y|x)dy \\ &= \int r(x, y)\frac{\partial}{\partial \theta}\pi_\theta(y|x)dy\end{aligned}$$

Converting back to expectation notation:

$$= \mathbb{E}\pi\theta(y|x)\left[r(x, y)\frac{\partial \pi_\theta(y|x)}{\partial \theta}\right]$$

$$\frac{\partial \pi_\theta(y|x)}{\partial \theta} = \pi_\theta(y|x)\frac{\partial}{\partial \theta}\log \pi_\theta(y|x)$$

But r is independent of θ, so:

$$= \mathbb{E}\pi\theta(y|x) \left[r(x,y) \frac{\partial}{\partial \theta} 1 \right]$$
$$+ \mathbb{E}y|x \left[\frac{\partial \pi\theta(y|x)}{\partial \theta} r(x,y) \right]$$

Using the identity:

$$\frac{\partial \pi_\theta(y|x)}{\partial \theta} = \pi_\theta(y|x) \frac{\partial}{\partial \theta} \log \pi_\theta(y|x)$$

If $f = e^g$, then:

$$\frac{df}{d\theta} = e^g \frac{dg}{d\theta} = f \frac{d \log f}{d\theta}$$

Therefore,

$$\boxed{J'_{RL}(\theta) = \mathbb{E}p(x)\mathbb{E}\pi\theta(y|x) \left[r(x,y) \frac{\partial}{\partial \theta} \log \pi_\theta(y|x) \right]}$$

Describe the stochastic gradient algorithm for maximizing J.

Explain that J' remains the same if we change $r(x, y)$ to $r(x, y) - b(x)$ for a baseline $b(x)$ that only depends on x. If $b(x) = V(x) = \mathrm{E}_{\pi(y|x)}[r(x, y)]$ for a policy π, then $V(x)$ is called the value function under π, and $A(x, y) = r(x, y) - V(x)$ is called advantage of action y at state x.

Also explain that

$$\mathrm{E}_{\pi_\theta(y|x)} \left[\frac{\partial}{\partial \theta} \log \pi_\theta(y|x) \right] = 0$$

by setting $r(x, y) = 1$ in J_{RL} and J'_{RL} above.

Stochastic Gradient Algorithm & Baseline Subtraction

- Part 1: Stochastic Gradient Algorithm for RL

The Gradient (from previous derivation):

$$J'RL(\theta) = \mathbb{E}p(x)\mathbb{E}\pi\theta(y|x) \left[r(x,y) \frac{\partial}{\partial \theta} \log \pi_\theta(y|x) \right]$$

This is the expected gradient. But we can't compute expectations exactly—we need to approximate them with samples.

Algorithm: Stochastic Gradient Ascent (REINFORCE)

Algorithm: Policy Gradient / REINFORCE

Input: Learning rate η, number of iterations N
Initialize: θ with pre-trained weights

For t = 1 to N:

1. SAMPLE question: $x \sim p(x)$
2. GENERATE completion: $y \sim \pi_\theta(y|x)$
3. EVALUATE reward: $r \leftarrow r(x, y)$
4. COMPUTE gradient: $g_t = r(x, y) \cdot \partial / \partial \theta log \pi_\theta(y|x)$
5. UPDATE policy: $\theta \leftarrow \theta + \eta \cdot g_t$

End For

Baseline Subtraction

We can replace $r(x, y)$ with $r(x, y) - b(x)$without changing the expected gradient:

$$
\begin{aligned}
J'_{RL,new} &= J'_{RL,old} - \int_x p(x) \int_y \frac{\partial}{\partial \theta} [\pi_\theta(y|x)] b(x) dy dx \\
&= J'_{RL,old} - \int_x p(x) b(x) \int_y \frac{\partial}{\partial \theta} [\pi \theta(y|x)] dy dx \\
&= J'_{RL,old} - \int_x p(x) b(x) \frac{\partial}{\partial \theta} \int_y \pi \theta(y|x) dy dx \\
&= J'_{RL,old} - \int_x p(x) b(x) \frac{\partial}{\partial \theta} [1] dx \\
&= J'_{RL,old}
\end{aligned}
$$

$$J'_{RL,new} = E_p(x)E_{\pi_\theta}(y|x)[(r(x,y) - b(x)) \cdot \frac{\partial}{\partial\theta} log\pi_\theta(y|x)]$$
$$= E_p(x)E_{\pi_\theta}(y|x)[r(x,y) \cdot \frac{\partial}{\partial\theta} log\pi_\theta(y|x)] - E_p(x)E_{\pi_\theta}(y|x)[b(x) \cdot \frac{\partial}{\partial\theta} log\pi_\theta(y|x)]$$
$$= J'_{RL,old} - E_p(x)b(x)E_{\pi_\theta}(y|x)[\frac{\partial}{\partial\theta} log\pi\theta(y|x)]$$

The second term vanishes because:

$$E_{\pi_\theta}(y|x)[\frac{\partial}{\partial\theta} log\pi_\theta(y|x)] = 0$$

This is proven by noting that $\pi_\theta(y|x)$ must sum to 1:

$$\frac{\partial}{\partial\theta}\int_y \pi_\theta(y|x)dy = \frac{\partial}{\partial\theta}(1) = 0$$

Therefore:

$$\int_y \frac{\partial}{\partial\theta}\pi_\theta(y|x)dy = 0$$
$$\int_y \pi_\theta(y|x) \cdot \frac{\partial}{\partial\theta} log\pi_\theta(y|x)dy = 0$$

(3)

Similarity & Differences Between MLE and RL

Similarities:

- Both sample (x_i, y_i) from a joint distribution to estimate $J'(\theta)$
- Both use gradient ascent:$\theta \leftarrow \theta + \eta \cdot \nabla_\theta J'(\theta)$
- Both approximate expectations with empirical samples
- Both follow the policy gradient structure

Key Differences:

	Imitation Learning - (1)	Reinforcement Learning - (2)
Data distribution	$\pi_{teacher}(y\|x)$ = [expert]	$\pi_\theta(y\|x)$ = [self-generated]

Gradient weighting	No reward factor	Weighted by $r(x, y)$
Supervision	Dense (every action label)	Sparse (only reward signal)
Non-stationarity	Fixed teacher policy	Policy changes each iteration
Sample efficiency	High (expert data)	Lower (exploration needed)
Scalability	Limited by expert availability	Scales with compute

Problem 2

Play with the PyTorch code provided by the following webpage:

Write a brief explanation of the code and show your results.

Key Components

1. Text Generation - Uses transformers pipeline to generate multiple responses to prompts
2. Dataset Creation - Creates pairwise comparison data for questions about:
   - Stock market news
   - Economic state
   - Technology developments
   - Political situations
3. Label Studio Integration - Exports data for human preference labeling
4. File I/O - Saves generated examples to JSON files

Workflow Diagram

```
graph TD
DATA["1. Data Preparation"]⟶ REWARD["  2.Reward Model Training"]
REWARD ⟶ POLICY["3. Policy Gradient Fine-tuning"]
POLICY ⟶ EVAL["4. Evaluation"]
```

Output

Please see the attached `RLHF_Custom_Datasets.ipynb` for the details.

Main Issues Found and fixed:

- Issues
    1. CUDA Compatibility Warning ⚠️
        - `GPU (NVIDIA GB10)` has CUDA capability 12.1
        - PyTorch version only supports up to 12.0
        - Impact: May cause stability issues or unexpected behavior
    2. Deprecated Functions ⚠️
        - `torch.utils._pytree._register_pytree_node` is deprecated
        - `resume_download parameter` is deprecated
        - Impact: Code will break in future library versions
    3. Multiple `pad_token_id` Warnings ⚠️
        - Setting `pad_token_id` to `eos_token_id` for generation
        - Impact: May affect text generation quality
    4. Path Hardcoding
        - Paths changed from `/content/` (Google Colab) to `/home/ohsono/jupyterlab/stats413/`
        - Recommendation: Use relative paths for portability
- Fix
    1. CUDA Compatibility Issue
        - Upgrade PyTorch to a version supporting CUDA 12.1
        - Or add error handling/warnings in the notebook
        -
    2. Deprecated Functions
        - Update code to use `torch.utils._pytree.register_pytree_node` (new API)
        - Remove resume_download parameter from `huggingface_hub` calls
    3. `pad_token_id` Warnings
        - Explicitly set `pad_token` in tokenizer configuration
        - Add proper padding token handling
    4. Path Hardcoding
        - Replace hardcoded paths with relative paths
        - Use `os.path.join()` for cross-platform compatibility

Similarity & Differences Between MLE and RL

Similarities:

- Both sample (x_i, y_i) from a joint distribution to estimate $J'(\theta)$
- Both use gradient ascent: $\theta \leftarrow \theta + \eta \cdot \nabla_\theta J'(\theta)$
- Both approximate expectations with empirical samples
- Both follow the policy gradient structure