Adversarial Methods for the alignment of CodeLLMs

End of semester presentation

Ilyas Oulkadda Julien Perez

July 2024

Table of Contents

Table of Contents

Introduction

Goal

Improve CodeLLMs alignment (Ouyang et al., 2022)

How ?

- Adversarial (OpenAI et al., 2021)
- Self-play (Sukhbaatar et al., 2018)
- Curriculum learning (Sukhbaatar et al., 2018)
- At scale (Bowman et al., 2022)

Current open models

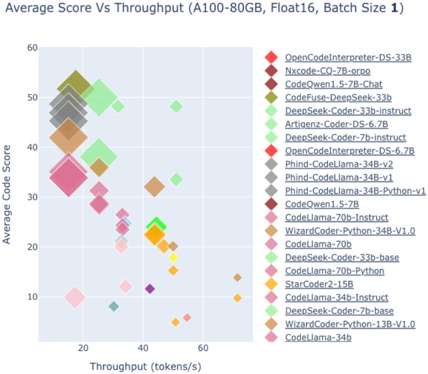

Figure: The open code LLM benchmark (Ben Allal, Muennighoff, et al., 2022)

Table of Contents

Why ?

- For the planet
- We have limited resources

Low Rank Adaptation

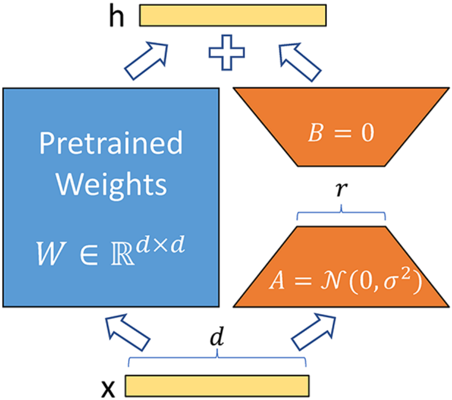

Figure: LoRA adds new adapters, the weight matrix is a product of 2 lower rank matrices (Hu et al., 2021)

Quantization

Table: Different quantization for Phi 1.5 (Gunasekar et al., 2023), 1.3B parameters

dtype	Model	Grad. Calc.	Backward Pass	Opt. Step
Float32	4.9 GB	4.9 GB	9.81 GB	19.62 GB
Float16/BF16	4.9 GB	7.36 GB	9.81 GB	9.81 GB

Example with PHI-1.5

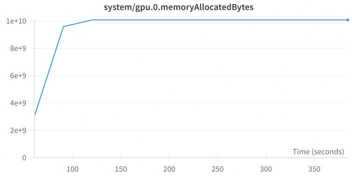

Figure: Memory usage while training Phi 1.5 on HumanEval (Chen et al., 2021)

Table of Contents

Proximal Policy Optimization (Schulman et al., 2017)

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{1}$$

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{2}$$

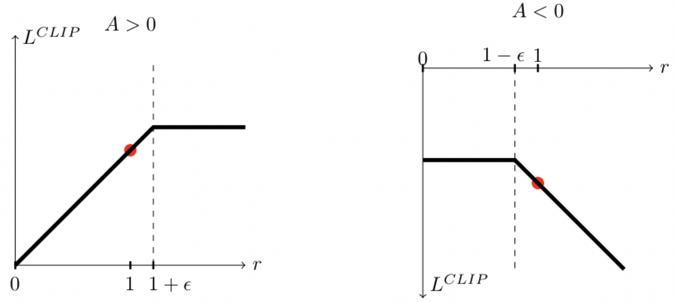

Figure: Objective function for both negative advantages and positive advantages.

Instability of PPO during RLHF (Ouyang et al., 2022)

Reward Model

The reward model is an approximation of the human preference.

Solution

Use unit tests as our reward model

Dataset generation

Main challenge

Generate a diverse dataset

Method (Ben Allal, Lozhkov, et al., 2024) (Gunasekar et al., 2023) (Eldan and Li, 2023)

- List of coding categories
- Generate sub-categories
- Generate multiple exercises
- Random profession per exercises
- Generate the unit tests
- JSON format

Sample

```python
def split_constellations(name):
    """
    Given a string representing a comma-separated list of constellations, split it into a list.
    """


def test_split_constellations():
    # Testing different scenarios based on the function description
    assert split_constellations("Andromeda, Perseus, Triangulum") == [
        "Andromeda",
        "Perseus",
        "Triangulum",
    ]
    assert split_constellations("The Big Dipper, Ursa Major") == [
        "The Big Dipper",
        "Ursa Major",
    ]
    assert split_constellations("The Southern Cross") == ["The Southern Cross"]
    assert split_constellations("") == []
    assert split_constellations(", , ,") == []


test_split_constellations()
```

Figure: Sample from our generated dataset, composed by a prompt (function definition, arguments and docstring), a test suite and a the name of the test suite function

Sample


```python
def gaussian_elimination(A):
    """
    Find the determinant and inverse of a given 3x3 matrix A using Gaussian elimination.

    Args:
        A (list of lists): A 3x3 matrix represented as a list of lists.

    Returns:
        tuple: A tuple containing the determinant and the inverse of the matrix A.
    """
    # Placeholder for the Gaussian elimination implementation
    pass

def test_gaussian_elimination():
    # Test 1: Identity matrix
    A = [[1, 0, 0],
         [0, 1, 0],
         [0, 0, 1]]
    assert gaussian_elimination(A) == (1, A)

    # Test 2: Simple 3x3 matrix
    A = [[2, 3, 4],
         [1, 5, 7],
         [6, 8, 10]]
    assert gaussian_elimination(A) == (30, [[-3.5, 2, 1], [1, -0.5, -1], [1.5, -1, 0]])

    # Test 3: Zero matrix
    A = [[0, 0, 0],
         [0, 0, 0],
         [0, 0, 0]]
    assert gaussian_elimination(A) == (0, A)

    # Test 4: Matrix with a determinant of zero
    A = [[1, 2, 3],
         [4, 5, 6],
         [7, 8, 9]]
    assert gaussian_elimination(A) == (0, None)

    # Test 5: Matrix with complex numbers
    A = [[1 + 2j, 3, 4 - 1j],
         [2, 5, 6 + 4j],
         [7, 8 - 3j, 9]]
    assert gaussian_elimination(A) == (4 + 13j,
                                       [[(4 - 13j) / 25, -3 / 5, (11 + 24j) / 25],
                                        [2, -5, 6],
                                        [(11 - 12j) / 25, (4 + 3j) / 5, (-4 + 3j) / 25]])

test_gaussian_elimination()
```

Figure: Same here but a little bit more complex

Reward function

Our reward model is actually just a function that compiles the code and runs the tests.

Rewards

- **Timeout**: -2
- **Syntax error**: -1
- **Runtime error**: -0.6
- **Assertion error (test fails)**: -0.3
- **Pass all unit tests**: 4

Training

Table: Deepseeker-Coder-1b-base (Guo et al., 2024) performances (Ben Allal, Muennighoff, et al., 2022)

Win Rate	humaneval-python	java	javascript	cpp
16	32.13	27.16	28.46	27.96

Main issues

- Missing libraries
- Indentation
- Timeouts and inputs
- Wrong tests

Results

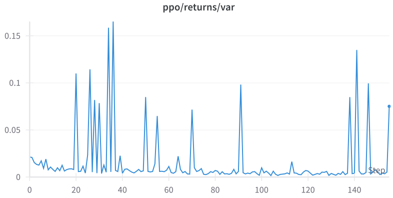

Figure: The observed high variance in rewards may be attributed to the varying difficulty levels of coding exercises.

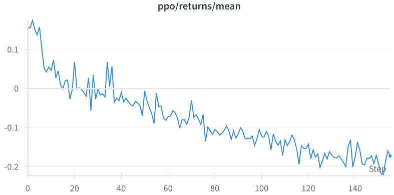

Figure: The mean reward does not show a consistent increase, indicating that the current PPO setup may not be maximizing expected rewards efficiently.

Limitations

- Requires a high quality test suite
- Larger models
- Only tests functionality
- Limited to a restrained type of exercises

Limitations

```python
def tax_reform_simulation(initial_gdp, tax_rates, years):
    """
    Docstring explaining the exercise:
    Create a function to simulate the economic impact of a tax reform over a certain number of years.

    Args:
        initial_gdp (float): initial Gross Domestic Product (GDP) of the economy
        tax_rates (list): a list of percentage tax rates for each income level
        years (int): number of years to simulate the tax reform

    Returns:
        None, prints out the GDP for each year after the tax reform
    """
    # Placeholder for function implementation
    pass

def test_tax_reform_simulation():
    # Simple case: constant tax rate, positive GDP growth
    assert tax_reform_simulation(1000, [20], 3) == [1020.0, 1040.4, 1061.204]

    # Simple case: no tax, positive GDP growth
    assert tax_reform_simulation(1000, [0], 2) == [1020.0, 1040.4]

    # Simple case: negative tax rate (subsidy), positive GDP growth
    assert tax_reform_simulation(1000, [-10], 1) == [1030.0]

    # Complex case: varying tax rates, positive and negative GDP growth
    assert tax_reform_simulation(1000, [10, 15, 5, -5], 5) == [1015.0, 1030.25, 1045.7625, 1033.47625, 1065.147625]

    # Edge case: zero initial GDP, varying tax rates
    assert tax_reform_simulation(0, [20, 15, 5], 3) == [0, 0, 0]

if __name__ == "__main__":
    test_tax_reform_simulation()
```

Figure: Function returns nothing while test suite expects a result

Table of Contents

Direct Preference Optimization (Rafailov et al., 2023)

$$-\mathbb{E}_{(x,y_u,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_u\mid x)}{\pi_{\text{ref}}(y_u\mid x)}-\beta\log\frac{\pi_\theta(y_l\mid x)}{\pi_{\text{ref}}(y_l\mid x)}\right)\right] \quad (3)$$

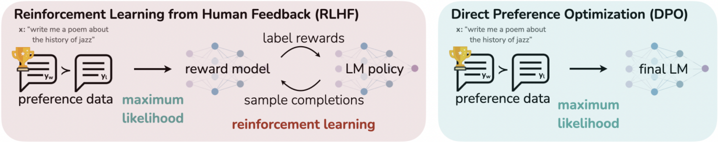

Figure: PPO RLHF VS DPO (Rafailov et al., 2023)

Adversarial game

Goal

Transfer coding ability from a larger model to a smaller one

Setup

- An oracle model
- A student model
- Oracle generates exercises and solutions
- Student also generates it's own solution

Datasets

Requirements

For DPO, datasets must have a Prompt, Chosen and Rejected

- Llama 3 70B on Groq as an oracle (AI@Meta, 2024)
- Mistral 7B as a student (Jiang et al., 2024)
- Same method as for PPO, categories, sub-categories...

Adversarial dataset

At the end of each training, we retrieve the hardest exercise to use as templates for more samples. (Sukhbaatar et al., 2018)

Training setup

- Trained on H100 80GB
- HuggingFace DPO
- LoRA
- Float16 Quantization
- Single adapter

Results

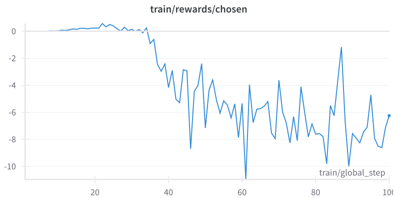

Figure: Chosen rewards during our training, the higher the reward the more our model converge to the chosen distribution

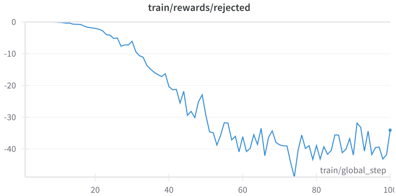

Figure: Rejected rewards during our training, the lower the reward the more our model is diverging from the rejected distribution

Benchmark

After 2 iterations, we were able to test the model on the HumanEval benchmark showing a promising 1% increase.

Results

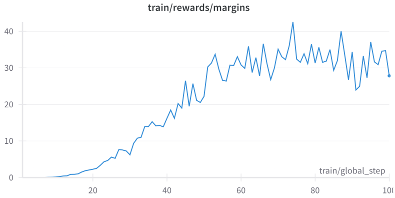

Figure: Increasing reward margin means our model is actually converging toward the Oracle's distribution and diverging from the initial distributions

Figure: Accuracies plot shows the frequency of choosing the preferred answer. The trained model quickly reaches a perfect accuracy score, which is a good sign but could also mean that the difference between preferred and rejected answers is too obvious.

Conclusion

Advantages

- Learning efficiently
- Easier than PPO unit tests
- Covers large coding aspect
- Scalable oversight

Limitations

- Dependent on the Oracle
- Monolithic Oracle

Table of Contents

What's next

- Run on Jean Zay or future LRE GPUs
- Larger and better dataset
- Data decontamination
- Train Oracles
- Improve Adversarial sampling

Table of Contents

Conclusion

- Promising results for adversarial DPO
- Limitations during PPO
- PPO failed but we have interesting behavior results
- Dataset generation can be easily scaled now for future use

Bibliography I

AI@Meta (2024). "Llama 3 Model Card". In: URL: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Ben Allal, Loubna, Anton Lozhkov, et al. (2024). *Cosmopedia*. URL: https://huggingface.co/datasets/HuggingFaceTB/cosmopedia.

Ben Allal, Loubna, Niklas Muennighoff, et al. (2022). *A framework for the evaluation of code generation models*. https://github.com/bigcode-project/bigcode-evaluation-harness.

Bowman, Samuel R. et al. (2022). *Measuring Progress on Scalable Oversight for Large Language Models*. arXiv: 2211.03540 [cs.CL].

Chen, Mark et al. (2021). *Evaluating Large Language Models Trained on Code*. arXiv: 2107.03374 [cs.LG].

Eldan, Ronen and Yuanzhi Li (2023). *TinyStories: How Small Can Language Models Be and Still Speak Coherent English?* arXiv: 2305.07759 [cs.CL].

Gunasekar, Suriya et al. (2023). *Textbooks Are All You Need*. arXiv: 2306.11644 [cs.CL].

Bibliography II

- Guo, Daya et al. (2024). *DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence*. arXiv: 2401.14196 [cs.SE]. URL: https://arxiv.org/abs/2401.14196.
- Hu, Edward J. et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: 2106.09685 [cs.CL].
- Jiang, Albert Q. et al. (2024). *Mixtral of Experts*. arXiv: 2401.04088 [cs.CL].
- OpenAI, OpenAI et al. (2021). *Asymmetric self-play for automatic goal discovery in robotic manipulation*. arXiv: 2101.04882 [cs.LG].
- Ouyang, Long et al. (2022). *Training language models to follow instructions with human feedback*. arXiv: 2203.02155 [cs.CL].
- Rafailov, Rafael et al. (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv: 2305.18290 [cs.CL].
- Schulman, John et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.CL].

Bibliography III

Sukhbaatar, Sainbayar et al. (2018). *Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play*. ICLR 2018. arXiv: 1703.05407 [cs.LG].