Research Internship Presentation

Yash Bansal
Indian Institute of Technology, Delhi

Topic:- Training Neural Action Policies for Planning Benchmarks in Jani

Supervisor:- Songtuan Lin

Overview

- **Objective**
  - Train neural action policies using reinforcement learning for planning benchmarks modelled in JANI.
- **Motivation**
  - Existing PLAJA framework had limited support for experimenting with complex architectures
  - Needed a more flexible and reliable pipeline for RL experimentation
- **Approach**
  - Converted JANI models into a **Gymnasium-compatible environment** to integrate smoothly with **Stable-Baselines3**
  - Used the Gym interface to easily train and evaluate different RL algorithms
  - Leveraged PyTorch's nn.Module to define and experiment with custom **DQL network architectures**, ranging from shallow to deep models
  - Tuned hyperparameters and evaluated performance across benchmarks

Understanding JANI and PLAJA

- Familiarised myself with JANI benchmarks
- Studied and understood the PLAJA codebase
- Understood the implementation of Deep Q-Learning, including policy structures, architectures, and reward functions
- Attempted to train neural action policies using PLAJA for feedforward architectures, but encountered multiple compile-time and runtime issues
- Switched to Stable-Baselines3 to build a new training framework

Model Loading and Analysis using Stormpy

- Parsed JANI benchmark files using Stormpy to extract the model and goal property specification
- Analyzed model structure: automata count, number of states, initial state, action space, and transition matrix
- Verified correct retrieval of the goal property
- Explored the Stormpy model class, inspecting available functions and attributes to support Gymnasium environment construction

Gymnasium-Compatible Environment

- Implemented a custom Gymnasium environment compatible with Stable-Baselines3
- Defined state space, action space, and transition dictionary
- Implemented key environment methods:
  - **reset():** resets to the initial state
  - **step(action):** applies an action and returns next state, reward, and done status
  - **render():** visualizes the current state, goal status, and available actions
  - **get_valid_actions():** returns valid actions based on current state and constraints
- **Reward function:**
  - Invalid action → **−1**
  - Goal reached → **+100**
  - Valid step (non-goal) → **−0.1**

Reinforcement Learning using Stable-Baselines3

- Used Stable-Baselines3 to train a Deep Q-Learning (DQL) model on the custom Gymnasium environment
- Ensured compatibility between the environment and the RL pipeline
- Configured custom neural network architectures using PyTorch; experimented with 2 to 7 layers
- Tuned key hyperparameters including learning rate, exploration fraction, number of training episodes, and learning start threshold
- Evaluated the model's goal-reaching ability through multiple simulations under both deterministic and probabilistic transitions

Experiments and Analysis

- Main experiments conducted on **blocksworld 5** and **elevators3-3** benchmarks.
- For **elevators3-3**:
  - Initial 2-layer architectures performed poorly.
  - Deeper 5-layer networks significantly improved results.
- For **blocksworld 5**:
  - Both shallow and deep architectures worked well.
  - However, deeper networks required substantially more training episodes than smaller networks.

Elevators 3-3 Analysis

- Approximately 1000 states with 27 goal states.
- Training with max 2000 steps per episode led to poor performance as the agent rarely reached goals early on.
- Increasing max steps to 3000 allowed better exploration and goal reachability.
- Low exploration fraction resulted in faster initial learning but left some states unexplored.
- During testing, the agent could get stuck in unexplored states, causing poor success rates.
- Increasing exploration fraction increased training duration but improved testing success rate from 20% to 50%.

Reward Function Analysis

- Assigning similar negative rewards to invalid actions and non-goal steps caused the model to avoid valid steps and prefer invalid actions.
- Using a high negative reward for invalid actions, a small negative reward for non-goal steps, and a large positive reward for reaching the goal improved learning.
- The model quickly learned to avoid invalid actions within the first 10–15 episodes and then consistently reached the goal thereafter.

Blocksworld_5 Analysis

- Approximately 1100 states with a single goal state.
- A simple 2-layer network was sufficient to learn this environment.
- Due to the single goal, a high exploration fraction and more training episodes were necessary to find the goal initially.
- After training, the model reached the goal in about 15–20 steps, compared to 200–250 steps for elevators3-3.

Conclusion

- Developed a custom Gymnasium environment and implemented RL algorithms using Stable-Baselines3.
- Designed a configurable neural network architecture based on PyTorch.
- Experimented with various architectures and fine-tuned training parameters.
- Gained insights into the impact of model structure, reward design, and exploration strategies on reinforcement learning performance.

Thank You