Model Citizens: Understanding the Art of Jaywalking with Deep Policy Gradient Algorithms

[1]**Vatsal Mehta**, [2]**Karen Zhang**
Northeastern University
360 Huntington Ave, Boston, MA 02115
[1]mehta.vats@northeastern.edu [2]zhang.yuchen@northeastern.edu

Abstract

We explore training reinforcement learning (RL) agents to jaywalk safely and efficiently across multi-lane intersections while investigating explainable deep learning within the RL context. The stochastic nature of this environment poses challenges to learning the optimal policy. To address this, we develop a new simulation environment and compare two widely used policy gradient algorithms, Proximal Policy Optimization (PPO) and Actor-Critic with Experience Replay (ACER), across three difficulty levels. We perform input layer weight and integrated gradient attribution analysis, represented through heatmap visualizations. Through the heatmaps, we analyze the learning and decision-making process of the algorithms and explain trends observed during the training process. In this work, we seek to contribute a new simulation environment, explore different policy gradient algorithms, and provide insights into improving explainability for RL algorithms.

Code — https://github.com/VatsalMehta27/jaywalker-rl

1 Introduction

With Northeastern University's central location in Boston, traffic affects more than just the cars. Students frequently navigate busy intersections to reach key campus locations such as classrooms, dormitories, gyms, and dining halls. For students with tightly packed schedules and short windows between classes or activities, efficient travel is necessary to remain on time. Unfortunately, crossing streets like Huntington Ave is often the primary bottleneck, where students are at the mercy of traffic lights and vehicles, factors beyond their control. Naturally, the only feasible solution to reduce crossing time is to jaywalk.

Motivated by this real-world problem, we propose training a reinforcement learning (RL) agent to jaywalk optimally across multi-lane intersections. The objective is to minimize the waiting and crossing time while ensuring the agent's safety, replicating the decision-making and risk assessment of a real jaywalker. To achieve this, we implement and compare two policy gradient algorithms: Proximal Policy Optimization (PPO) and Actor-Critic with Experience Replay (ACER). Our goal is to understand which algorithm provides the best balance between performance and safety in a simulated environment. In doing so, we seek to contribute a new simulation environment and provide insights into improving explainability for RL algorithms.

2 Background

2.1 Markov Decision Processes (MDP)

Markov Decision Processes (MDPs) are the foundational formalization of sequential decision-making problems. In an MDP, an agent interacts with an environment over a series of time steps, taking actions that influence future outcomes. This problem formulation relies on the Markov assumption, which states that the future state depends only on the current state and action, and is independent of all previous states and actions. MDPs are formally defined as a tuple (S, A, T, R, γ) where:

- S: the set of possible states representing the environment,
- A: the set of actions available,
- $T(s_{t+1} \mid s_t, a_t)$: the transition function, specifying the probability of moving to state s_{t+1} after taking action a_t in state s_t,
- $R(s_t, a_t)$: the reward function, assigning a numerical reward for a state transition,
- γ: the discount factor, which scales future rewards to prioritize immediate outcomes.

At each time step t, the agent selects an action a_t based on the current state s_t, and receives a reward r_{t+1} after the transition to the next state s_{t+1}. The objective of the agent is to determine the optimal policy $\pi^*(a \mid s)$, such that the agent maximizes the expected future cumulative discount rewards:

$$E_\pi \left[\sum_{t=0}^{H-1} \gamma^t r_t \right].$$

2.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) (Schulman et al. 2017b) is a popular actor-critic-based reinforcement learning algorithm designed to improve the stability and sample efficiency of policy updates. It optimizes a clipped surrogate objective function:

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

This clipping approximates the KL-divergence constraint proposed in Trust Region Policy Optimization (TRPO) (Schulman et al. 2017a), which prevents large deviations in the policy to ensure stability in the training process. Large positive rewards are clipped in their update, while negative rewards can result in arbitrarily large negative updates, giving rise to its conservative nature. PPO is typically parametrized by a neural network and is effective in both discrete and continuous action spaces using different types of policy distributions.

PPO makes online updates to the network weights, performing multiple epochs of optimization for each mini-batch to enhance sample efficiency. As an actor-critic-based method, the algorithm incorporates an advantage estimation function to approximate the relative value of actions in a given state. While PPO primarily optimizes the policy distribution, it also updates a separate value function network (critic) which is used for the advantage calculation. The critic is typically optimized using the mean-squared error loss (MSE) between the discounted trajectory rewards-to-go and the value estimation:

$$\text{MSE} = \frac{1}{N}\sum_{t=1}^{N}(V(s_t) - G_t)^2$$

2.3 Actor-Critic with Experience Replay (ACER)

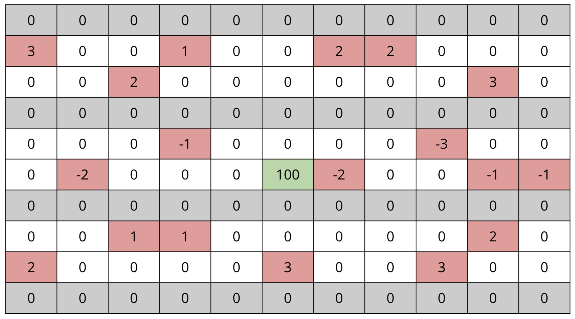
Actor-Critic with Experience Replay (ACER) (Wang et al. 2017) is an advanced reinforcement learning algorithm that extends the actor-critic framework by incorporating experience replay to improve sample efficiency. The actor learns the policy to maximize the expected return, and the critic evaluates the policy by estimating the value function for a given state $V(s)$. These are both typically parametrized by neural networks. ACER makes use of a replay buffer to store past experiences and sample them during the training process, similar to Deep Q-Networks (Mnih et al. 2013). For the purpose of this paper, we focus on the discrete action space version of ACER.

Unlike most other policy gradient methods, ACER supports both on- and off-policy learning, by introducing truncated importance sampling with bias correction (equation below is adapated from (DI-Engine 2024)),

$$\begin{aligned} \hat{g}_t &= \rho_t \nabla_\theta \log \pi_\theta(a_t|x_t) \left[Q^{\text{ret}}(x_t, a_t) - V_{\theta_v}(x_t) \right] \\ &+ E_{a\sim\pi_\theta} \left[\left(\frac{\rho_t(a) - c}{\rho_t(a)} \right)_+ \cdot \right. \\ &\nabla_\theta \log \pi_\theta(a|x_t) \left[Q_{\theta_v}(x_t, a) - V_{\theta_v}(x_t) \right]] \end{aligned}$$

where the importance sampling ratio, ρ_t, is defined as,

$$\rho_t(a) = \frac{\pi_t(a|x_t)}{\pi(a|x_t)}$$



0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	2	2	0	0	0
0	0	2	0	0	0	0	0	0	3	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	0	-3	0	0
0	-2	0	0	0	100	-2	0	0	-1	-1
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	2	0
2	0	0	0	0	3	0	0	3	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure 1: *Sample visualization of world grid*

and Q^{ret} is the recursively estimated return using the value function:

$$Q^{\text{ret}}(x_t, a_t) = r_t + \gamma\bar{\rho}_{t+1} \left[Q^{\text{ret}}(x_{t+1}, a_{t+1}) \right] + \gamma V(x_{t+1})$$

The clipping (truncation) mechanism, inspired by TRPO, balances the trade-off between bias and variance in policy updates.

3 Project description

3.1 MDP Definition

States Each state consists of:

- **World grid** - The world grid records the position and velocity of cars on the road and the agent. This is an $n \times m$ matrix grid representation of the world, where $n, m \in \mathbf{N}$. Each cell can contain either the agent, a car, or nothing. They are represented by the following:
    1. Agent = 100
    2. Car = its velocity (e.g. 2, -1). A positive number means the car is moving left to right and a negative number means the car is moving right to left.
    3. Empty = 0

  Figure 1 shows a sample visualization of the world grid. The red cells contain a car and the value is the velocity the car. The green cell with value 100 represents the agent. The gray cells represent the sidewalks, where cars can't spawn.
- **Traffic Light** - The traffic light represents the state of the light at the cross walk and is defined as: traffic_light $\in \{"RED", "YELLOW", "GREEN"\}$

Actions There are three actions that the agent can take at each timestep, each represented as an integer, such that $a \in \{0, 1, 2\}$.

1. Move backward = 0
2. Wait / Stay Still = 1
3. Move forward = 2

Rewards The agent can receive rewards in 3 ways:

1. Being alive = -1.

- For each time step in which the agent is alive (not in the goal state or run over by a car), the agent receives a reward of -1 to encourage faster solutions.

2. Reach goal state = 100.
3. Death = -50

Transitions At each time step:

1. The agent moves deterministically based on action taken.
2. The environment is stochastic, as cars spawn in a random lane with a probability p_spawn and the cars' velocity is sampled from a distribution. More details below.
3. The cars move v squares with velocity v. (Positive values mean vehicles move from left to right and negative values mean to move from right to left)
4. If the car is going through the crosswalk while the agent is present, it will stop with a probability of p_stop.
5. If there is another car directly in front of a car, the car in the back will match the speed of the car in front, so the cars won't crash into each other.
6. Cars spawn with probability p_spawn on any of the road lanes in the grid. The velocity of the car spawned is sampled from a normal distribution with mean of 1.5 and std of 0.5.
7. Cars will stop at the crosswalk when the traffic light is read. If the traffic light is yellow, the cars will stop at the crosswalk with p_stop.

Discount Factor (γ) We used a discount factor $\gamma = 0.99$ for all experiments to ensure the agent has a long-term outlook on the rewards.

3.2 Algorithms

In this project, we applied Proximal Policy Optimization (PPO) and Actor-Critic with Experience Replay (ACER) to a discrete action setting. Additionally, we chose Deep Q-Network (DQN) and REINFORCE with Baseline (Sutton and Barto 2018) to be the baseline models for comparison.

Proximal Policy Optimization (PPO) As discussed above, PPO is an extremely robust algorithm that is applicable on many different types of domains. In this domain, we parametrized the learned policy as a categorical distribution. The actor-critic framework is implemented as a neural network to efficiently learn the weights. The architecture is described in detail in below:

- **State Input** - The environment state representation as described in section 3.1 is transformed into a 1D array:
  - The $n \times m$ world grid matrix is flattened into a 1D array of length $n \cdot m$.
  - The traffic light is one-hot encoded, such that the index of the array with the value 1 represents the active light. For example, a red light would be encoded as $[1, 0, 0]$, yellow would be $[0, 1, 0]$, and green would be $[0, 0, 1]$.

  Both the flattened grid and the one-hot encoded traffic light representation are concatenated to create a 1D array of size $n \cdot m + 3$, which is then fed as input to the network.
- **Base Network** - The base network of both the actor and the critic consists of:
  - **Input Layer** - Input dimension is $n \cdot m + 3$, and output is the number of hidden dimensions.
  - **Hidden Layers** - The input of the hidden layer has size 64, and the output is also of size 64.
  - **Activation** - The ReLU (Rectified Linear Unit) activation function is used throughout the network to introduce non-linearity.
- **Actor (Policy)** - The actor network applies a softmax activation over the output of the dense layers to generate a probability distribution over the available actions, yielding three probabilities representing each action.
- **Critic (Value Function)** - The critic network outputs a single value, representing the state value $V(s)$ for the given input state. This helps guide the training process by providing a baseline for the advantage estimation in PPO.

An extended version of the PPO pseudocode is provided below (Algorithm 1) (adapted from (OpenAI 2024)).

Algorithm 1: Proximal Policy Optimization (PPO)

```
1: Input: Initial policy parameters $\theta_0$, Initial value function parameters $\phi_0$
2:
3: for $k = 0, 1, 2, ...$ do
4:     Collect set of trajectories $D_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
5:     Compute rewards-to-go $R_t$.
6:     Compute advantage estimates, $\hat{A}_t$, based on the current value function $V_k$.
7:     Update the policy by maximizing the clip objective:
           $$\theta_{k+1} = \arg\max_\theta \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} L^{CLIP} A_t^\pi(s_t, a_t)$$
       typically via stochastic gradient ascent with Adam.
8:     Update value function by optimizing MSE:
           $$\phi_{k+1} = \arg\min_\phi \frac{1}{|D_k|} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - R_t)^2$$
       via some gradient descent algorithm, e.g. Adam.
9: end for
```

Actor-Critic with Experience Replay (ACER) As discussed above, ACER is an efficient off-policy reinforcement learning algorithm that combines actor-critic methods with experience replay to improve sample efficiency. Similar to the PPO implementation, the learned policy is parametrized by a categorical distribution. The actor-critic framework in ACER is implemented using a neural network to learn the policy and value function. The architecture is described below:

- **State Input** - The state is transformed in the same manner as described for PPO above.
- **Base Network** - The base network is shared by both the policy (actor) and the value (critic) networks. It is designed as follows:
  - Input Layer - The input dimension is $n \cdot m + 3$, which corresponds to the concatenated state representation.
  - Hidden Layers - The network contains one hidden layer, each with 64 units, and ReLU activation functions to introduce non-linearity.
- **Actor (Policy)** - The actor network produces a probability distribution over actions using a softmax activation function. This distribution represents the policy $\pi(a|s)$, which is used to select actions during both on-policy and off-policy updates.
- **Critic (Value Function)** - The critic network outputs a single scalar value, $V(s)$, representing the state value for the current input state.
- **Importance Sampling and Retrace** - To handle off-policy corrections during training, ACER uses truncated importance sampling as expressed by the equation in section 2.3 to adjust the actor's policy updates. Additionally, the retrace operator ensures stable learning by clipping importance weights and incorporating multi-step returns.
- **Experience Replay** - The algorithm maintains a replay buffer which stores experienced transitions $(s, a, r, s', done, \pi(\cdot \mid s))$. During training, mini-batches are sampled from this buffer to compute updates for both the actor and critic for off-policy learning.

An extended version of the ACER pseudocode is provided in Algorithm 2 (adapted from (Stanford Vision and Learning Lab 2024)).

Deep Q-Network (DQN) Deep Q-Network (DQN) is a baseline algorithm chosen to help better analyze PPO and ACER. DQN combines Q-learning with deep neural networks, designed to handle complex and high dimensional environments. DQN estimates the Q-values $Q(s, a)$ using a neural network. The algorithm optimizes the Q-network with the Bellman equation:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a'),$$

where s and s' are the current and next states, a is the action, r is the reward, and γ is the discount factor.

The neural network used to estimate the Q-values consists of an input, 2 hidden, and an output layer.

DQN also consists of the following components, which stabilizes its training and encourages exploration:

Algorithm 2: ACER for discrete actions

```
 1: Reset gradients $d_{\theta'} \leftarrow 0$ and $d_{\theta_v} \leftarrow 0$.
 2: Initialize parameters $\theta' \leftarrow \theta_v$ and $\theta_v \leftarrow \theta_v$.
 3: if Off-Policy then
 4:     Sample the trajectory from replay memory    ▷ Change $\mu(s_0)$ to be log prob?
 5:     $\{(s_0, a_0, r_0, \mu(s_0)), \ldots, (s_k, a_k, r_k, \mu(s_k))\}$
 6: else
 7:     Get initial state $s_0$
 8: end if
 9: for $i \in \{0, \ldots, k\}$ do
10:     Compute $f_{\theta'}(s_i)$, $Q_{\theta_v}(s_i)$, and $f_{\theta_v}(x_i)$.
11:     if On-Policy then
12:         Perform action $a_i$ according to $f_{\theta'} s x_i)$.
13:         Receive reward $r_i$ and new state $s_{i+1}$.
14:         $\mu(s_i) \leftarrow f_{\theta'}(s_i)$.
15:     end if
16:     $\rho_i \leftarrow \min\{1, \frac{f_{\theta'}(a_i|s_i)}{\mu(a_i|s_i)}\}$.
17: end for
18: $Q^{\text{ret}} \leftarrow \begin{cases} 0 & \text{for terminal } s_k \\ \sum_a Q_{\theta_v}(s_k, a) f_{\theta_v}(a|s_k) & \text{otherwise} \end{cases}$
19: for $i \in \{k-1, \ldots, 0\}$ do
20:     $Q^{\text{ret}} \leftarrow r_i + \gamma Q^{\text{ret}} + \gamma V_{\theta_v}(s_{i+1})$.
21:
22:     Trust region update:
        $$\begin{aligned} g \leftarrow & \min\{c, \rho_i\} \nabla_{\theta'} \log f_{\theta'}(a_i|s_i)(Q^{\text{ret}} - V_{\theta_v}(s_i)) \\ & + \sum_a \frac{1}{c} \rho_i(a) f_{\theta_v}(a|s_i) \nabla_{\theta'} \log f_{\theta'}(a|s_i) \\ & \quad \cdot (Q_{\theta_v}(s_i, a) - V_{\theta_v}(s_i)) \\ & + \nabla_{\theta'} V_{\theta_v}(s_i) \text{DKL}[f_{\theta'}(s_i) || f_{\theta_v}(s_i)] \end{aligned}$$
23:     Accumulate gradients wrt $\theta'$
24:     $d_{\theta'} \leftarrow d_{\theta'} + g$.
25:     Accumulate gradients wrt $\theta_v$
26:     $d_{\theta_v} \leftarrow d_{\theta_v} + \nabla_{\theta_v}(Q^{\text{ret}} - Q_{\theta_v}(s_i, a_i))^2$.
27:     Update Retrace target:
28:     $Q^{\text{ret}} \leftarrow \rho_i(Q^{\text{ret}} - Q_{\theta_v}(s_i, a_i)) + V_{\theta_v}(s_i)$.
29: end for
30: $\theta' \leftarrow \theta' + \alpha d_{\theta'}$
31: $\theta_v \leftarrow \theta_v + \beta d_{\theta_v}$
32: Updating the average policy network: $\theta_v \leftarrow \alpha\theta' + (1 - \alpha)\theta_v$.
```

- **Replay buffer:** Steps in the environment are stored as experiences (s, a, r, s') in a replay buffer. It is randomly sampled from to improve stability and learning.
- **Target Network:** A secondary network that is updated periodically. It gives more stable target Q-values that will limit the update made at each iteration, stabilizing training.
- **ϵ-Greedy Exploration:** DQN exploits or explores based on ϵ probability. Exploiting means selecting the action that gives the highest Q-value and exploration means selecting an action at random.

REINFORCE with Baseline REINFORCE with Baseline is a simple Monte Carlo variant of policy gradient algorithms that relies on returns from episodes to update the policy. It is an on-policy algorithm, meaning the policy being updated is the same as the policy generating the data for training.

The gradient to update policy parameters can be computed by:

$$\nabla_\theta J(\theta) = \alpha\gamma(G - \hat{v}(S_t, \mathbf{w}))\nabla_\theta \ln \pi_\theta(A_t|S_t, \theta).$$

where

$$\hat{v}(S_t, \mathbf{w})$$

is the baseline state-value function parametrization. This updates the policy parameters θ in the direction that improved the likelihood of actions chosen from the policy that yields high rewards.

REINFORCE architecture utilizes Monte Carlo Sampling. It samples trajectories to estimate returns G, which is used in policy update. REINFORCE also uses a baseline that is subtracted from the returns G to stabilize training.

3.3 Explainability for Deep Reinforcement Learning

In order to explore the inner workings of the algorithms, we chose to apply popular explainability methods for general deep learning models to our deep implementations of the actor network in PPO and ACER.

- **Input Layer Weight Visualization** - We visualized the sum of the absolute values of the weights connected to each input feature in the input layer. Through this approach, we are able to visualize the relative importance of different features in the state representation. Cells in the world grid with greater sum of absolute weights will be given more importance in deciding the most optimal action and vice versa. This was visualized as a heatmap on the world grid. (Voss et al. 2021)
- **Integrated Gradients** - Integrated Gradients (Sundararajan, Taly, and Yan 2017) is an attribution method that quantifies the contribution of input features to the model's output by integrating gradients along the path from a baseline input to the actual input. Formally, it computes:

  $$\text{IG}_i(X) = (X_i - X'_i) \cdot \int_{\alpha=0}^{1} \frac{\partial F(X' + \alpha \cdot (X - X'))}{\partial X_i} \, d\alpha$$

  where, X' is the baseline, F is the model function, and α is a scaling factor.

  The method adheres to two key axioms:

  - **Sensitivity** - Features with zero contribution have zero attribution.
  - **Implementation Invariance** - Attributions are consistent for functionally equivalent models.

  In our work, Integrated Gradients was applied using Captum to analyze the input features most influential to action probabilities in the actor network. This is because we were most interested in understanding the decision-making process of the agent.

	Easy	Medium	Hard
Number of groups of lanes	2	3	4
Maximum number of vehicles	10	30	50
Prob. car spawning in lanes	0.4	0.8	0.9
Prob. car stopping for agent	0.8	0.5	0.3

Table 1: *Parameters that differ across the three environments*

4 Experiments

4.1 Experiment Setup

Three environments of varying levels of complexity were used to evaluate each algorithm. All three environments have the basic functionalities including traffic lights and the same reward functions. The agent is trying to achieve the same goal no matter the environment.

The three environments have different number of groups of lanes, different probability of a car spawning at each timestep, different probability of cars stopping for the agent, and different maximum number of vehicles allowed on the grid at a given time. Table 1 illustrates the four parameters that are different across the three environments and the values of the parameters for each environment.

The baseline models, PPO and ACER are trained in each of the three environments.

4.2 Training Results

We plotted the returns per episodes or training steps, timesteps per episode, and loss. We are primarily focused on the returns curve and timesteps curve. The returns curve informs us if the agent learns to cross the street without dying and the timesteps curve informs us if the agent learns to cross the street in as little timesteps as possible, achieving the goals of being a jaywalker.

The details of each experiment can be found in the notebooks folder of the GitHub repository.

DQN Deep Q-Network (DQN) is trained for baseline comparison with PPO and ACER.

During hyperparameter tuning, we tuned the learning rate and replay buffer size and found the best performing model out of 6 combinations of them. The identified hyperparameters were used later on for training the DQN agent on the different environments.

Looking at Figure 2, DQN performs really well on all three environments. As expected the agent has a slightly lower return and higher timesteps taken for the more difficult environments due it their complexities. But overall, the agent was able to learn and converge to an optimal policy even on the more difficult environments. It is also very stable, the training curve is smooth and without large perturbations. One note here is that the metrics are recorded for

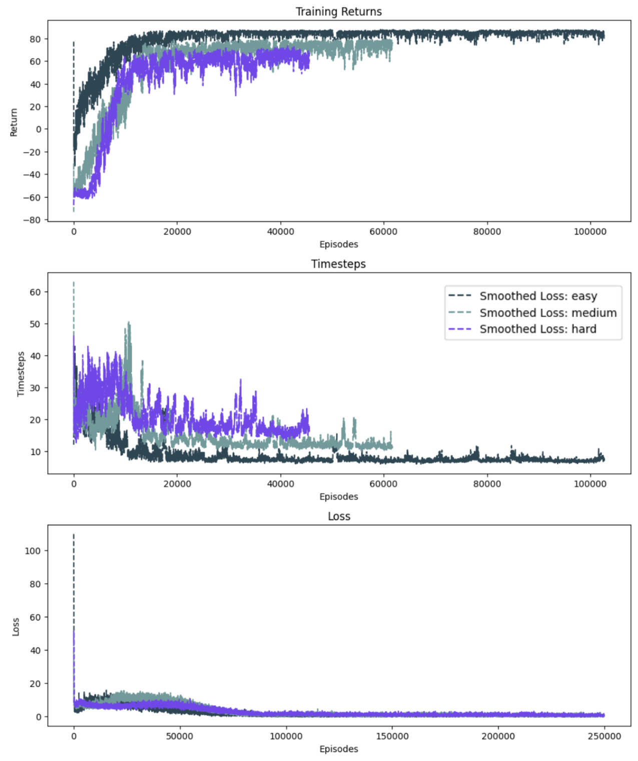


Figure 2: *DQN agent performance on each of the three environments*

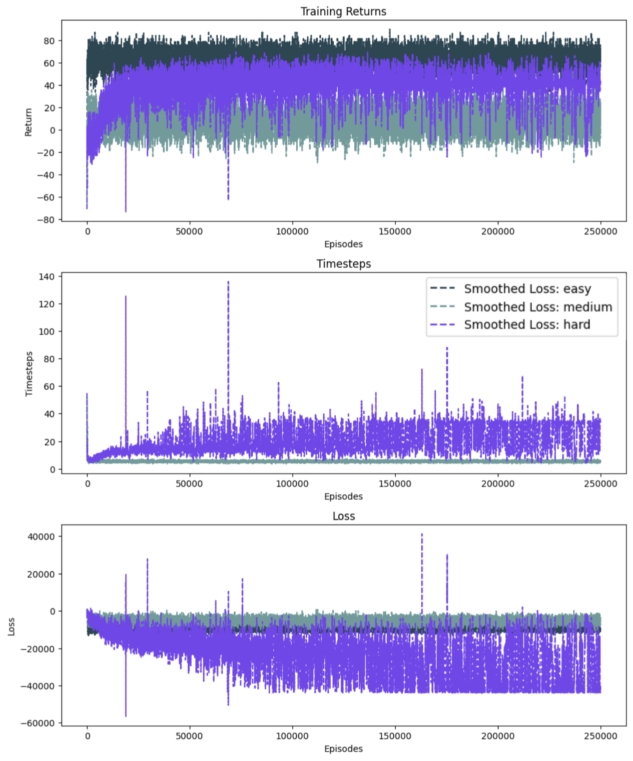


Figure 3: *REINFORCE agent performance on each of the three environments*

every completed episode, so the hard environment has fewer datapoints to plot, where the agent doesn't always terminate (timeout is not considered for the plotting purpose).

REINFORCE with Baseline REINFORCE is trained for baseline comparison with PPO and ACER.

During hyperparameter tuning, we tuned the learning rate to get the best performing model. The learning rate was used later on for training the REINFORCE agent on the three environments.

Referring to Figure 3, REINFORCE was able to learn the optimal policy and converge to a high return on both the easy and difficult environment, despite being a relatively simple algorithm. Surprisingly, it doesn't perform as well on the medium model, only converging to returns of between 0 and 20. This could be because the agent experienced more negative transitions and was hit by the cars, resulting in lower returns overall.

PPO PPO is the first of the two algorithms we wanted to explore in detail.

During hyperparameter tuning, we tuned the batch size, learning rate, and epochs per iteration to find the best performing model out of 27 combinations. The chosen hyperparameters were used later on for training the PPO agent on the three environments.

As seen in Figure 4, PPO agent performs very well on both the easy and medium environments, converging to almost an average return of 100, the highest returns achieved

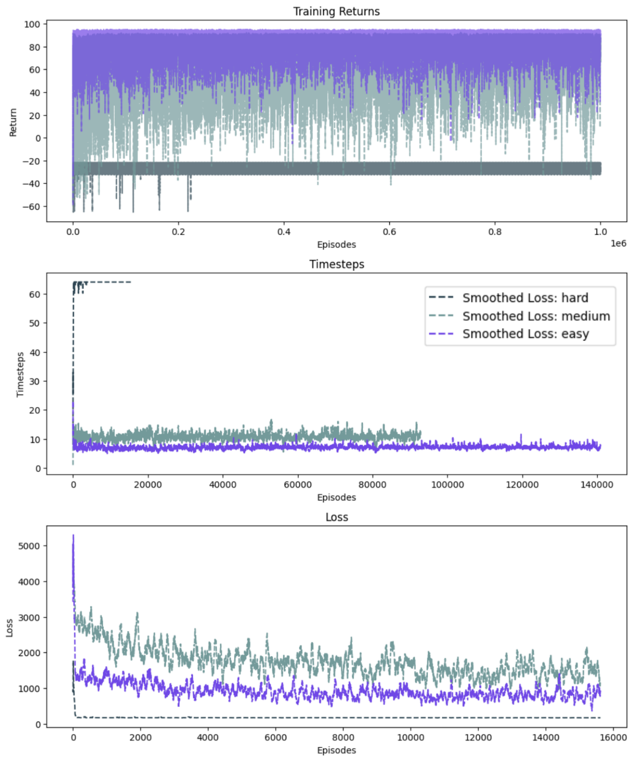


Figure 4: *PPO agent on each of the three environments*

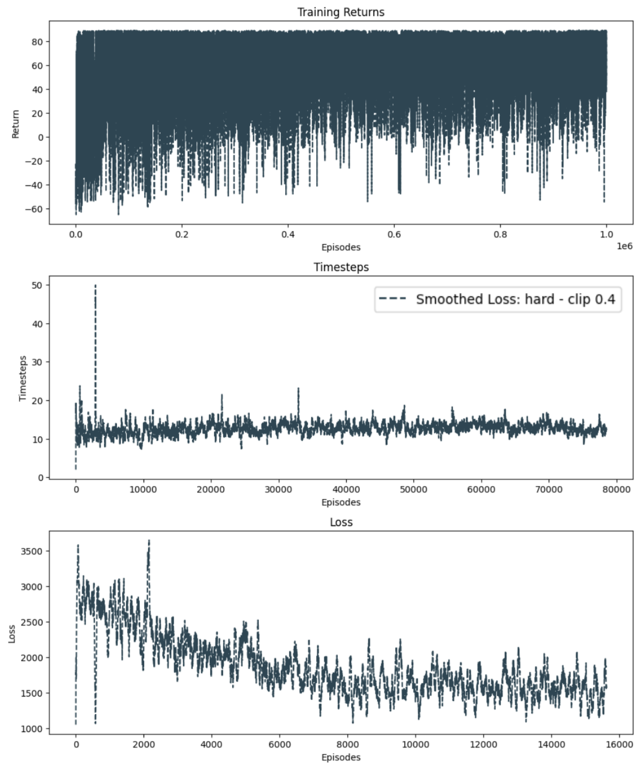


Figure 5: *PPO agent with clip of 0.4 on the hard environment*

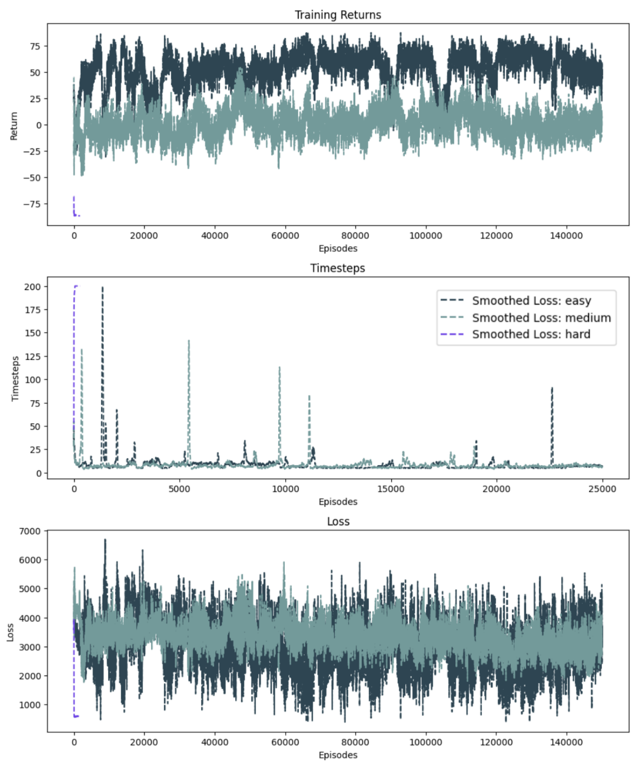


Figure 6: *ACER agent on each of the three environments*

among all the algorithms. However, the learning curve is not as stable as that of DQN.

Despite its success on the easier environments, PPO agent was not able to learn in the hard environment. It plateaus at an return of around -20 and the timesteps taken per episode is very large, meaning the agent did not learn to properly cross the street. We hypothesize that this is due to the PPO agent being ”too careful”, resulting in the agent doesn’t end up crossing the street. PPO clips the influence a positive reward has on the update but does not clip the negative reward. Hence, when the agent reaches the goal and received a positive reward, the update is not large enough to overcome the updates from the large updates negative rewards from dying. Since it is much easier to die in the difficult environment, the negative updates are disproportionally large, and we end up with an agent that is ”too cautious”.

To confirm our hypothesis, we tried increasing the clipping from 0.2 to 0.4 such that the positive rewards will have a greater update on the policy. As shown in Figure 5, the retrained agent was able to properly learn to cross the street, converging to a return of around 80 and timesteps per episode of 15. This confirmed our theory that the clipping and disproportionate amount of negative rewards caused the PPO agent to learn poorly before.

ACER ACER is the second of the two algorithms we wanted to explore in detail.

During hyperparameter tuning, we tuned the learning rate and replay ratio to find the best performing model out of 6 combinations of them. The chosen hyperparameters were used later on for training the ACER agent on the three environments.

As seen in Figure 6, ACER doesn’t learn as well as the previous algorithms. It only converged to returns of approximately 50 to 75 and 0 to 25 on the easy and medium environments respectively. The training is very unstable and the curve reflects these fluctuations. The ACER agent stopped training early on in ther hard environment, as the agent was stuck on receiving the same very negative returns on every iteration, meaning it did not learn a proper policy.

ACER, like PPO, also clips the update and uses KL Divergence constraint during optimization. Hence, ACER could face similar issues as PPO regarding the clip limiting the agents, and making it more cautious and not performing as well. Trying different clip values and KL Divergence could affect how the ACER agent performs on the environments.

While experimenting, we found that ACER was also relatively more susceptible to different seeds, as the performance changed significantly when we changed the seeds for the model.

4.3 Interpreting Model Decisions

In this section, we detail the results obtained from our explainability analysis, focusing on the input layer weight visualization for the neural network and the Captum integrated gradients visualization for the starting state of each environment.

Input Layer Weight Visualization The heatmap of the sum of absolute values of the input layer weights reveals which features were considered most influential in the decision-making process of the agent. The analysis shows how the distance of cars to the sidewalk and the traffic lights affect the agent's decision to move or stay. Notably, the weights for the vehicles approaching closest to the crosswalk were given the greatest total weight. This value scales inversely as the distance of the vehicle increases from the agent - vehicles in the last lane group have lower weights. This is expected, as the approaching vehicles pose the most danger to a jaywalker. It's also interesting to see most of the agents learned to put minimal weight on the sidewalks, resulting in the world grid pattern emerging in the heatmap. Almost all the agents placed the most importance on the value of the green light, again suggesting an importance of safety and ability to cross.

As discussed in the training results, REINFORCE provides surprising results in the medium environment, in that the sidewalks did not show the lowest weight (Figure 8). The weights were more evenly distributed, though the greatest emphasis was still on the vehicles near the crosswalk.

These trends are illustrated in the input weight plots for the 4 algorithms analyzed. Figures 7 to 10 show the medium difficulty environment results, with additional plots available on our GitHub repository.

The PPO results for the hard environment were particularly insightful to explore due to the unexpectedly poor returns. The input weight visualizations showed a greater emphasis on vehicle cells across the entire lanes, indicating that PPO was more cautious in its decision-making (Figure 12). In comparison, DQN consistently focused on the vehicle cells immediately next to the crosswalk, suggesting that DQN was more likely to move forward despite potential danger (Figure 11). We hypothesized that this difference is due to PPO's conservative clipping update mechanism, which is absent in DQN, causing DQN to prioritize immediate obstacles more strongly.

The input weight visualization for PPO trained with an increased clip of 0.4 seems to also support this hypothesis (Figure 13). The weight attribution reduces the broader focus in the harder environment that the initial PPO had with a clip of 0.2. The few vehicle cells closest to the crosswalk showed the greatest weights, which is closer to the visualization for DQN.

Integrated Gradients The integrated gradients visualization provided a more granular understanding how the agent evaluates the initial state of the environment. We generated a heatmap of the attributions calculated for each input state feature and compared with the human-readable render of the environment, illustrated in Figures 14 to 17. The heatmaps showed that the vehicles closest to the agent had the greatest positive attribution, which vehicles further away had negative attribution. This observation is consistent with the analysis of the input weights, where closer vehicles are given greater importance. In the starting state, the agent doesn't place much, if any, emphasis on the light color, thought this could change as the agent progresses in the environment.

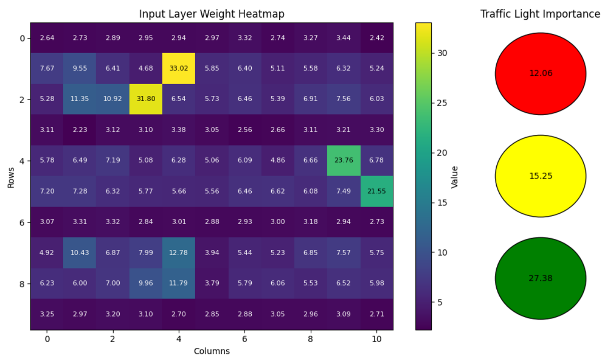


Figure 7: *PPO medium env. input weight visualization*

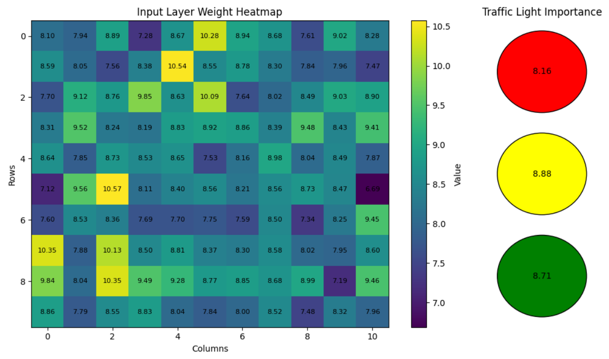


Figure 8: *REINFORCE medium env. input weight visualization*

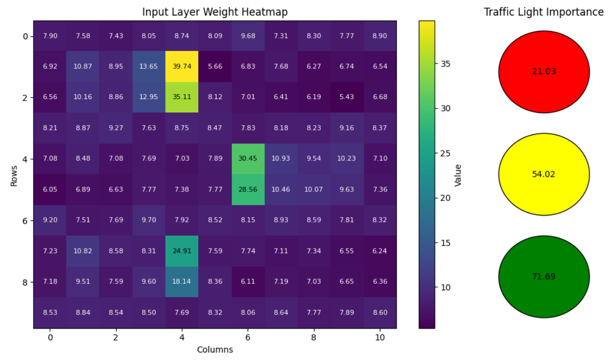


Figure 9: *DQN medium env. input weight visualization*

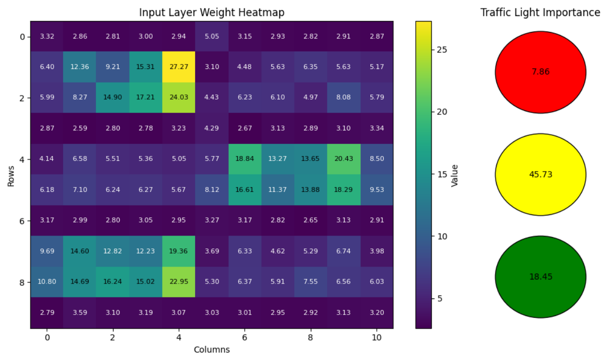


Figure 10: *ACER medium env. input weight visualization*

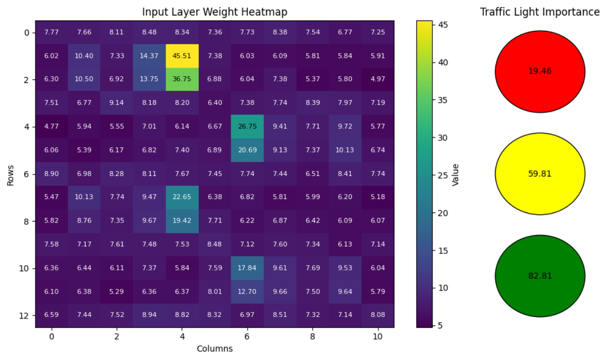


Figure 11: *DQN hard env. input weight visualization*

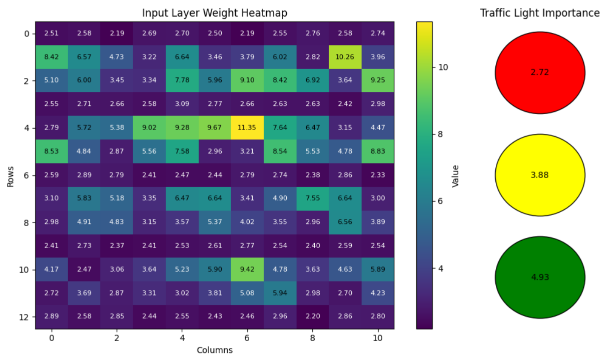


Figure 12: *PPO hard env. input weight visualization*

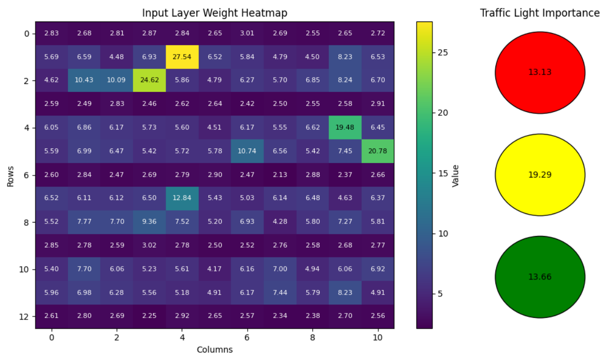


Figure 13: *PPO* $c = 0.4$ *hard env. input weight visualization*

Interestingly, ACER did not clearly show the same level of attribution to the vehicles as PPO, which had the clearest attribution visualizations of the 4 algorithms. This could explain why the medium environment results for ACER were poor, since it didn't effectively learn the importance of the vehicle positions in determining safety. Additional training for all the algorithms could help in making these trends clearer. We provide further visualizations of the attributions in an evaluation rollout on our GitHub to further provide clarity into how the network decisions evolve.

5 Further work

5.1 Expanding the Environment

There are several ways to expand the relatively simple version of the environment used in this paper.

- **Experimenting with Reward Shaping** - Since some of the agents tend to be safe, it would be interesting to explore how added a first-time bonus to reaching a sidewalk would affect the short-term vs. long-term importance that the algorithms give.
- **Introduce Continuous State and Action Space** - For the purpose of an introductory environment, we developed the intersection under the assumption of grid-like world, with discrete and fixed movements defined. In the real world, a jaywalker can translate any real number distance across a sidewalk (with some constraints), and the same applies for vehicles. Introducing a version of the environment that utilizes a continuous plane could be more interesting, and present a better simulation of the problem.
- **Multi-Agent Problem Formulation** - The jaywalking domain can be extended into a multi-agent problem, where there are several jaywalker agents aiming to cross the intersection in the shortest amount of time. This could involve collaboration or competition between the agents. Like in real life, the number of agents crossing at the same time could influence the probability of vehicles stopping, and therefore the success rate of crossing.

5.2 Alternative Policy Gradient Methods

Discrete Soft Actor-Critic (SAC) The discrete version of Soft Actor-Critic (SAC) (Haarnoja et al. 2018) is an alternative off-policy gradient algorithm that could be applied to this domain. It is based on the actor-critic framework, which learns a parameterized stochastic policy represented as a categorical distribution over a discrete action space. The policy is optimized using a reparameterization trick and incorporates entropy regularization to encourage exploration while maximizing expected rewards. Additionally, the algorithm employs two Q-functions as the critic and leverages the clipped double-Q trick to mitigate overestimation bias and achieve stable off-policy learning.

It would be interesting to compare discrete SAC with the off-policy performance of ACER, especially given the instability encountered during ACER training in our experiments. The inclusion of entropy regularization in SAC may present an opportunity to study balancing exploration and safety

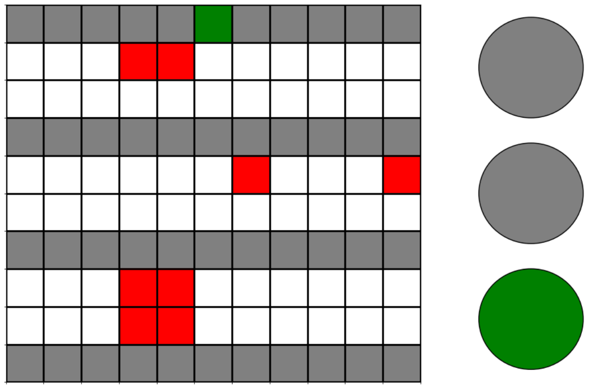


Figure 14: *PPO medium env. starting state*

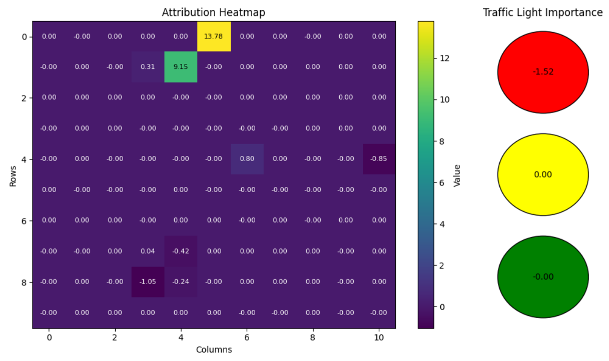


Figure 15: *PPO medium env. attribution visualization*

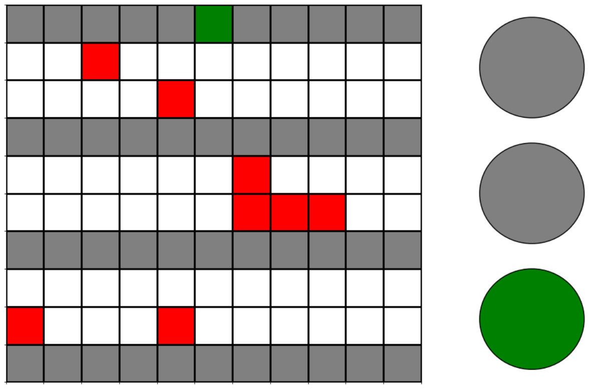


Figure 16: *ACER medium env. starting state*

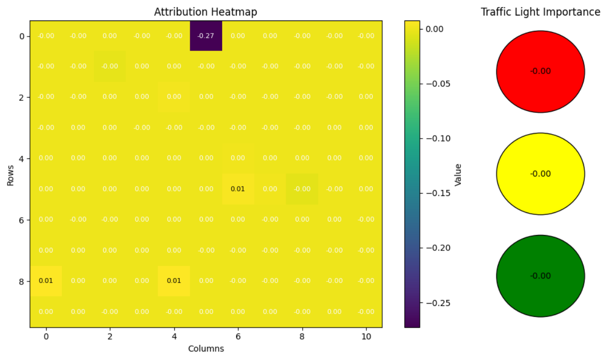


Figure 17: *ACER medium env. attribution visualization*

within this domain. Furthermore, such a comparison with PPO, with an additional entropy term included in the loss function, could shed light on how algorithm performance is influenced by the focus on exploration given when designing the algorithm.

Hindsight Experience Replay Hindsight Experience Replay (HER) (Andrychowicz et al. 2018) is a powerful off-policy technique that enhances learning in environments with sparse rewards. By re-labeling experiences with alternative goals, HER allows agents to learn from episodes that would otherwise be ignored. It stores trajectories in a replay buffer and modifies them to treat unsuccessful attempts as useful training data by setting new goals based on the final state.

Although this environment doesn't strictly have sparse rewards, it could help with learning as the size of the intersection increases. We could use also use experience re-labeling technique to emphasize the importance of safety as an alternative goal during the training process.

5.3 Alternative Explainability Methods

SHapely Additive exPlanations (SHAP) This technique calculates feature importance using a unique game theory-based approach by evaluating different combinations of features. (Lundberg and Lee 2017) This could provide further insight into the importance of each vehicle cell in the world and how the traffic lights influence the actions.

Reward Decomposition This method allows us to analyze how different rewards in the environment influence the overall reward received by an agent. (Juozapaitis et al. 2019) In the context of this environment, we could explore how the agent balances the waiting, death, and completion rewards, and how changing those environment values can modify the agent's behavior.

6 Conclusion

This project demonstrated the effectiveness of policy gradient methods, particuarly PPO, in optimizing jaywalking across multi-lane intersections. Of the algorithms tested – Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Actor-Critic with Experience Replay (ACER), and REINFORCE – DQN shows the most consistent and highest performances across all 3 difficulties of environments. PPO achieved strong results in the simpler environments but failed on the more complex ones, requiring adjustments of the clip hyperparameter for good performance. ACER struggled with training instability and failed to converge to an optimal policy in the medium and difficult environments, indicating limitations in its applicability to stochastic environments.

The explainability analysis provided key insights into the decision-making of the agents, highlighting the importance of vehicle proximity to the crosswalk in determining whether to move forward. This paper lays the groundwork for future research into explainability with policy gradient methods, which could reveal useful insights into the inner workings of reinforcement learning.

6.1 Code Availability

The complete code written for this paper can be found at https://github.com/VatsalMehta27/jaywalker-rl.

This includes the Jaywalker Gymnasium environment, implementations of all the algorithms discussed in the paper from scratch, the interpretability code, and all the plots presented.

References

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2018. Hindsight Experience Replay. arXiv:1707.01495.

DI-Engine. 2024. ACER. Accessed: 2024-12-11.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290.

Juozapaitis, Z.; Koul, A.; Fern, A.; Erwig, M.; and Doshi-Velez, F. 2019. Explainable Reinforcement Learning via Reward Decomposition.

Lundberg, S.; and Lee, S.-I. 2017. A Unified Approach to Interpreting Model Predictions. arXiv:1705.07874.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602.

OpenAI. 2024. Proximal Policy Optimization (PPO) - Spinning Up Documentation. Accessed: 2024-12-11.

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I.; and Abbeel, P. 2017a. Trust Region Policy Optimization. arXiv:1502.05477.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017b. Proximal Policy Optimization Algorithms. arXiv:1707.06347.

Stanford Vision and Learning Lab. 2024. Reinforcement Learning Pseudocode. Accessed: 2024-12-11.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic Attribution for Deep Networks. arXiv:1703.01365.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Voss, C.; Cammarata, N.; Goh, G.; Petrov, M.; Schubert, L.; Egan, B.; Lim, S. K.; and Olah, C. 2021. Visualizing Weights. *Distill*. Https://distill.pub/2020/circuits/visualizing-weights.

Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; and de Freitas, N. 2017. Sample Efficient Actor-Critic with Experience Replay. arXiv:1611.01224.