A Expanded Calculations

A.1 Minimum Variance Model Gradient

Recall that under our assumptions the minimum variance trajectory distribution is:

$$q^*(\tau) = \frac{R(\tau)p(\tau)}{\mathbb{E}_{p(\tau)}\left[R(\tau)\right]} \tag{11}$$

The trajectory distribution under our learned model is $q(\tau)$, and the objective is to minimize $\mathrm{KL}(q^\star(\tau)||q(\tau))$ by gradient based optimization. Then the gradient of the objective with respect to q is:

$$\begin{aligned}
&\nabla_q \mathrm{KL}(q^*(\tau) \mid\mid q(\tau)) && (12)\\
&\quad = \nabla_q \int q^*(\tau) \log \frac{q^*(\tau)}{q(\tau)} d\tau && (13)\\
&\quad = -\int q^*(\tau) \nabla_q \log q(\tau) d\tau && (14)\\
&\quad = -\int p(\tau) \frac{R(\tau)}{\mathbb{E}_{p(\tau)}\left[R(\tau)\right]} \nabla_q \log q(\tau) && (15)\\
&\quad = -\mathbb{E}_{p(\tau)}\left[\frac{R(\tau)}{\mathbb{E}_{p(\tau)}\left[R(\tau)\right]} \nabla_q \log q(\tau)\right] && (16)\\
&\quad \propto -\mathbb{E}_p\left[R(\tau)\nabla_q \log q(\tau)\right] && (17)
\end{aligned}$$

The proportionality is because $\mathbb{E}_{p(\tau)}\left[R(\tau)\right]$ is a constant which can be be absorbed into the learning rate . Now using the definition of a trajectory distribution (Eq. 1) we see:

$$\nabla_q \log q(\tau) = \sum_{t=1}^{T-1} \nabla_q \log q(s_{t+1}|s_t, a_t) \tag{18}$$

Giving us the result:

$$\nabla_q \mathrm{KL}(q^*(\tau) \mid\mid q(\tau)) \propto -\sum_{t=1}^{T-1} \mathbb{E}_p\left[R(\tau)\nabla_q \log q(s_{t+1}|s_t, a_t)\right] \tag{19}$$

B Planning and training loop

Here we expand on how model learning, value estimation, and planning fit together in a model-based RL loop. We can plug in EstimateValue (Alg. 1) as a subroutine that produces unbiased return estimates into many planning algorithms. For example, Alg. 3 defines RSPlanner, a simple random shooting planner. At each timestep, RSPlanner samples K distinct sequences of actions, with each sequence containing H (the planning horizon) sampled actions. Each individual action is sampled independently from a simple distribution π over the action space, such as a uniform or Gaussian distribution. RSPlanner then uses EstimateValue to estimate a return for each action sequence, and selects the sequence with the highest estimated return. The planner then returns the first action in that sequence as the action for the current timestep. The planning process is repeated at the next timestep, and so on until the end of the episode.

Current approaches in model-based reinforcement learning tend to integrate planning and model training into an iterative loop: after training the model, they execute the planner in the real environment to collect some more data, and use this data to retrain the model. The pseudocode for MBRL in Alg. 3 shows how to do this with our framework. As MBRL uses RSPlanner to rollout more trajectories, it stores the observed transitions and retrains the model (TrainModel) and discriminator (TrainDiscriminator) on the new data. We define those model and discriminator training subroutines in Alg. 2. Crucially, the objective in TrainModel is return weighted as per Eq. 10, which trains the model to minimize value estimate variance in planning.

```
Function TrainModel
    input: Trajectory dataset $\mathcal{T} = \{\tau\}$
    Initialize $q(\cdot|s,a)$ with random parameters;
    while not converged do
        $\tau = \{s_t, a_t\}_{t=1}^{T} \sim \mathcal{T}$;
        /* See Eq. 10 */
        $L = -\sum_t R(\tau) \log q(s_{t+1}|s_t, a_t)$;
        Update $q$ by $\nabla L$
    end
    Result: Trained model $q$
Function TrainDiscriminator
    input: Learned model $q(\cdot|s,a)$
    input: Transition dataset $\mathcal{T} = \{(s,a,s')\}$
    Initialize $\mathcal{D}$ with random parameters;
    while not converged do
        $(s,a,s') \sim \mathcal{T}$ ;            // Sample transition
        $\tilde{s}' \sim q(\cdot|s,a)$ ;         // Fake next state
        /* Classification loss */
        $L = -\log(\mathcal{D}(s,a,s')) - \log(1 - \mathcal{D}(s,a,\tilde{s}'))$ Update $\mathcal{D}$ by $\nabla L$
    end
    Result: Trained discriminator $\mathcal{D}$ (logits $\tilde{\mathcal{D}}$)
```

Algorithm 2: Training dynamics model and discriminator

```
Function RSPlanner
    input: Current state $s_t$
    input: Plan horizon $H$
    input: Action sampling distribution $\pi$
    for $i = 1$ to $K$ do
        /* Sample action sequences */
        $\mathbf{a}^{(i)} \leftarrow \{a_1^{(i)}, \cdots, a_H^{(i)}\} \overset{i.i.d}{\sim} \pi$;
    end
    /* Select best action sequence */
    $\mathbf{a} \leftarrow \arg\max_{\mathbf{a}^{(i)}} \mathrm{EstimateValue}(s_t, \mathbf{a}^{(i)})$;
    $a_t \leftarrow a_1$;
    Result: $a_t$
Function MBRL
    input: Rollout$(\cdot)$: Executes given planner in real environment, produces trajectory
    $\mathcal{T} \leftarrow \{\}$;     // Trajectory dataset
    for $j = 1$ to $J$ do
        $\tau \leftarrow$ Rollout(RSPlanner);
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{\tau\}$;
        $q(\cdot|s,a) \leftarrow$ TrainModel$(\mathcal{T})$;
        $\mathcal{D} \leftarrow$ TrainDiscriminator$(q(\cdot|s,a), \mathcal{T})$;
    end
```

Algorithm 3: Planning and MBRL Loop

C Environments

IcyRoad: The agent drives a car down a straight but icy road with extends in the $+x$ direction. If the car exceeds a threshold x-velocity ($\dot{x} > 2$), there is a high chance of swerving off the road in either the $+y$ or $-y$ directions, incurring a negative reward. The four dimensional observations give the car's position and velocity $s = (x, y, \dot{x}, \dot{y})$, and there are 3 discrete actions for accelerating (increase $\dot{x}$ by 1), decelerating (decrease $\dot{x}$ by 1) and cruising ($\dot{x}$ stays the same). The velocities and states are updated using simple Euler integration, and the reward is $\dot{x}$ with bonuses for higher speeds and penalties for swerving:

$$r(s,a) = \begin{cases} \dot{x} - 6, & |y| > 1 \text{ (off road)} \\ \dot{x} + 8, & |y| \leq 1, \dot{x} > 2 \text{ (high speed bonus)} \\ \dot{x} & \text{otherwise} \end{cases} \tag{20}$$

The agent's car is always initialized with $(x, y, \dot{x}, \dot{y}) = (0, 0, 1, 0)$, and each episode proceeds for 5 timesteps before terminating.

Intersection: Here the agent again controls a car, this time approaching an intersection with an oncoming car entering the intersection at the same time (Fig. 2). Depending on the episode the oncoming car will randomly turn left or right with 50% chance. If it turns left, the agent's car must slow down to avoid collision. The observation is 10 dimensional and contains the positions and velocities of both cars as well as their heading (angle). The agent's car moves in the $+y$ direction, and has 3 discrete actions for accelerating, decelerating, or neither along the y-axis. The reward is:

$$r(s,a) = \dot{y}_{\text{agent}} - 10 \cdot \delta[\text{collision}] \tag{21}$$

where δ[condition] is 1 if the condition is true and 0 otherwise. The cars are initialized in fixed positions entering the intersection from opposite directions, with a velocity of 5 (for the agent) or -5 (for the oncoming car), and each episode is 25 timesteps.