Worst Cases Policy Gradients

Yichuan Charlie Tang
Apple Inc.
yichuan_tang@apple.com

Jian Zhang
Apple Inc.
jianz@apple.com

Ruslan Salakhutdinov
Apple Inc.
rsalakhutdinov@apple.com

Abstract: Recent advances in deep reinforcement learning have demonstrated the capability of learning complex control policies from many types of environment. When learning policies for safety critical applications, it is important to be sensitive to risks and avoid catastrophic events. Towards this goal, we propose an actor-critic framework which models the uncertainty of the future and simultaneously learns a policy based on that uncertainty model. Specifically, given a distribution of the future return for any state and action, we optimize policies for varying levels of conditional Value-at-Risk. The learned policy can map the same state to different actions depending on the propensity for risk. We demonstrate the effectiveness of our approach in the domain of driving simulations, where we learn maneuvers in two scenarios. Our learned controller can dynamically select actions along a continuous axis, where safe and conservative behaviors are found at one end while riskier behaviors are found at the other. Finally, when testing with very different simulation parameters, our risk-averse policies generalize significantly better compared to other reinforcement learning approaches.

Keywords: Safe, Risk-sensitive Reinforcement Learning, Autonomous Systems

1 Introduction

One of the key challenges for building intelligent systems is developing the capability to make robust and safe sequential decisions in complex environments. Towards this goal, the recent breakthroughs in deep reinforcement learning (RL) are very encouraging and have led to super human performance in various video games and board games [1, 2, 3].

As we move towards real-world applications and learning from increasingly diverse environments, we may encounter both *parametric* and *inherent* uncertainties due to the stochastic nature of the model and the environments [4]. One cause of stochasticity is the inability to fully observe the state (e.g. intentions, beliefs) of other agents in a multi-agent environment. Robust handling of uncertainties and risks are a must before we can fully leverage the power of deep RL in real world safety-critical applications such as self-driving [5]. However, standard RL optimizes the average expected return and is not risk sensitive [6].

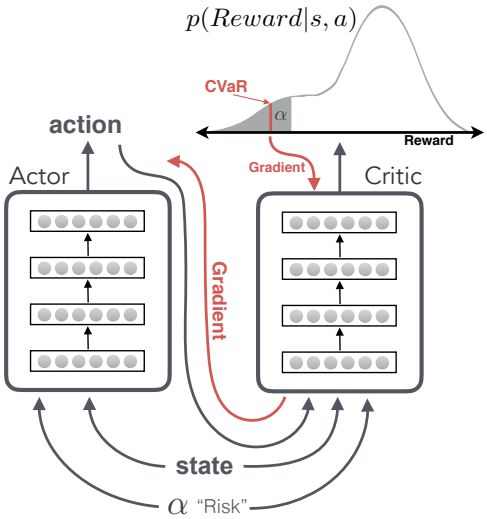


Figure 1: *Proposed architecture: the critic's estimated distribution over future returns is used to compute the conditional Value-at-Risk (CVaR) measure, which then generates gradients for learning the actor/policy network. See Sec. 3 for details.*

In this paper, we take a step towards this goal by developing a novel deep RL architecture that optimizes a risk-sensitive criterion. In contrast, the vast majority of existing deep RL techniques maximize the expected value over possible future returns [7, 8, 9, 10]. Maximizing expectation, however, is not risk-sensitive as it does not explicitly penalize rare occurrences of catastrophic events. Working under the assumption that the future return is inherently stochastic, we first start by modeling its

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

distribution. The risk of various actions can then be computed from this distribution. Specifically, we use the conditional Value-at-Risk [11] as the criterion to maximize. Our architecture is based on the actor-critic [12, 8], but we modify our critic to predict the full distribution over future returns instead of simply the expectation. The proposed framework (Fig. 1), which we call worst cases policy gradients (WCPG), learns a continuous family of policies, each optimizing for their respective level of risk: $\alpha \in [0.0, 1.0]$. Depending on the risk appetite, the learned policy can choose to act differently even from the same state.

After introducing our framework and the learning algorithm in Section 3, we demonstrate the effectiveness of WCPG on two traffic simulation scenarios, where an agent must learn to safely interact with other agents and to achieve their own goals. In Section 4, we describe the details of our environments, training procedure, and quantitative results, where we find WCPG learns risk-averse policies which are significantly more robust in test scenarios.

2 Preliminaries

We formulate our problem as a *Markov Decision Process* (MDP) which consists of a state space $\mathcal{S}$ (a compact subset of $\mathbb{R}^d$), an action space $A = \mathbb{R}^m$, and a reward function $r(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The model of the environment is $p(s'|s, a)$ which specifies the probability of transitioning to state s' from state s and executing action a. The policy function $\pi_\theta(a|s)$, parameterized by θ, is the probability of choosing action a given state s. $\rho^\pi(s)$ denotes the stationary distribution over the state space given that policy π is followed. We denote the total discounted future return as $R_t^\gamma = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0.0, 1.0]$ is the discount factor. The value function is $V^\pi(s) = \mathbb{E}[R_1^\gamma | S_1 = s, \pi]$ and the state-action value function is $Q^\pi(s, a) = \mathbb{E}[R_1^\gamma | S_1 = s, A_1 = a, \pi]$. Reinforcement learning consists of a class of algorithms which can be used to find the optimal policy for MDPs [6].

Policy gradient methods directly optimize the policy parameters θ to maximize the expected total return. Popular in continuous control, they compute the gradients with respect to the objective J:

$$J = \int_S \rho^\pi(s) \int_A \pi_\theta(a|s) Q^\pi(s, a) da\, ds \tag{1}$$

Using the *Policy Gradient Theorem* and the log-derivative trick [13, 14], the gradient of the objective can be written as:

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)], \tag{2}$$

where the gradient does not dependent on the state distribution $\rho^\pi(s)$. [15] have shown that it may be beneficial to substitute $Q^\pi(s, a)$ with other terms, where it can be one of several expressions: $Q^\pi(s, a)$, $V^\pi(s)$, or the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, leading to a lower variance of the gradients.

2.1 Deep Deterministic Policy Gradients

While Eq. 2 holds for any stochastic policy, [16] introduced the Deterministic Policy Gradient (DPG) theorem for deterministic policies, whose gradients can be estimated more efficiently. DPG states (under certain regularity conditions) that for a *deterministic* policy π:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho}[\nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)}]. \tag{3}$$

The deep deterministic policy gradient (DDPG) framework [8] extended DPG to large continuous state action space environments. DDPG works well for continuous control and is a popular variant of the *actor-critic* [14, 12] policy gradients method. A critic function learn to estimate $Q^\pi(s, a)$ using temporal difference bootstrapping and provides the training signal for the actor (policy network). Variants of DDPG have also been explored, including actors with parametrized actions [17].

3 Worst Cases Policy Gradients

Standard reinforcement learning maximizes for the expected (possibly discounted) future return [6]. However, maximizing for average return is not sensitive to the possible risks when the future return is stochastic, due to the inherent randomness of the environment not captured by the observable state. For example, when the return distribution has high variance or is heavy-tailed, finding a policy which maximizes the expectation of the distribution might not be ideal: a high variance policy (and therefore higher risk) that has higher return in expectation is preferred over low variance policies with lower expected returns. Instead, we want to learn more robust policies by minimizing long-tail risks, reducing the likelihoods of bad outcomes.

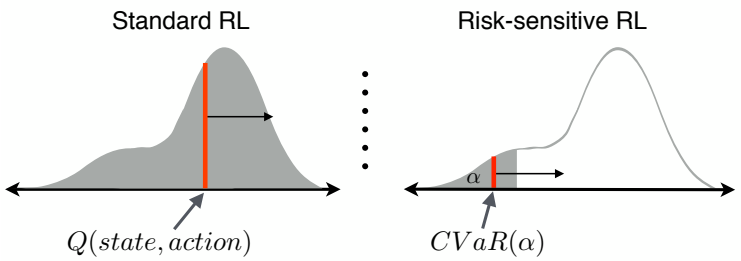


Figure 2: ***Left****: standard RL maximizes for the expected future return (red line).* ***Right****: WCPG optimizes the risk-sensitive $CVaR(\alpha)$, which is the expected value up to the α-percentile of the distribution of future return.*

We start by first modeling the distribution of expected future returns for all joint states and actions. For a single particular state-action pair $\{s, a\}$, let R be the random variable for future return, and $p^\pi(R|s, a)$ be the probability *distribution* of R given our current state and action while following policy π. We can recover the widely used state-action value function: $Q^\pi(s, a) = \mathbb{E}_{p^\pi}[R|s, a]$. In the illustration of Fig. 2, the x-axis is R and the y-axis is the probability density. In the left panel, the red line marks the expected value of the distribution or $Q^\pi(s, a)$. Typically, we optimize the policy by changing the shape of the density so as to increase $Q^\pi(s, a)$ or shift the red line to the right. However, for risk-averse learning, it is desirable to maximize the expected *worst* cases performance instead of the average-case performance. We can accomplish this by optimizing for the conditional Value-at-Risk (CVaR) criterion [11] instead. Intuitively, CVaR represents the expected return should we experience the bottom α-percentile of the possible outcomes. The right panel of Fig. 2 demonstrates that by optimizing for CVaR, we seek policies that will shift the tail-end of the distribution to the right, thus reducing the probability of experiencing a very negative event.

Formally, the criterion that we care about is the α-percentile of the distribution over returns. This is captured by the conditional Value-at-Risk:

$$CVaR_\alpha \doteq \mathbb{E}_{p^\pi}[R|R \leq pcntl(\alpha)], \tag{4}$$

where $pcntl(\alpha)$ is the α-percentile of p^π. As $\alpha \to 0$, the policy will focus on performing well in the "*worst cases*" scenarios. Computing CVaR metric directly for long time horizons (e.g. a brute-forced approach like sampling [18]) would be prohibitively expensive. As an alternative, we extend DDPG (an actor-critic architecture) so that the critic learns to model the distribution over expected total discount reward. Provided a good distributional critic can be learned, CVaR can be computed from the distribution and we can train the actor by backpropagating the gradient back through the actor or policy network.

At this point, the reader might wonder if it is possible to first train (offline) a critic for $p^\pi(R|s, a)$ and then obtain a risk-sensitive policy (online) simply by selecting actions to maximize the α-percentile of $p^\pi(R|s, a)$ at every state. However, this approach would fail as $p^\pi(R|s, a)$ is conditioned on executing policy π for all subsequent steps. Selecting the action (which modifies the original policy) that gave the best α percentile at time t is not meaningful if we revert to policy π for time $t + 1$ and beyond. In other words, multiple actions at a given state s would have similar distributions over R, as a single action (especially continuous) at time t can be nullified by future actions.

3.1 Distributional Critic

To learn the critic, we must define the equivalent Bellman operator for distributions. Let us first define $Z(s, a) \doteq p^\pi(R|s, a)$ as the distribution over total future (possibly discounted) return generated by executing policy π until reaching a terminal state. We define P^π as the transition operator:

$$P^\pi Z(s, a) \doteq Z(s', a'),\ s' \sim p(\cdot|s, a),\ a' \sim \pi(\cdot|s'), \tag{5}$$

and the distributional Bellman operator [19, 20, 21] $\mathcal{T}^\pi$ is: $\mathcal{T}^\pi Z(s, a) \doteq r(s, a) + \gamma P^\pi Z(s, a)$. Here, we approximate $Z(s, a)$ up to its 2nd-order statistics, the mean and the variance. Defining the mean as $Q^\pi(s, a) = \mathbb{E}_{p^\pi}[R|s, a]$ and the variance $\Upsilon^\pi(s, a) = \mathbb{E}_{p^\pi}[R^2|s, a] - (Q^\pi(s, a))^2$, we have $Z^\pi(s, a) \sim \mathcal{N}(Q^\pi(s, a), \Upsilon^\pi(s, a))$. Modeling Z as a Gaussian leads to a closed-form calculation of the CVaR, as we will see shortly[1].

[1]This is critical as we must compute CVaR for every update step, for every state-action tuple, and for different risk α parameters. Sampling to estimate CVaR would be prohibitively expensive computationally.

There are two sets of projection equations for the distributional Bellman operator. The projection for Q^π (mean) is the standard Bellman equation, similar as in Q-learning: $Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a)Q^\pi(s',a')$. The projection for Υ^π (variance) is:

Proposition 1. *The following equation for the variance of the future return exists.*

$$\begin{aligned}\Upsilon^\pi(s,a) =& r(s,a)^2 + 2\gamma r(s,a)\sum_{s'} p(s'|s,a)Q^\pi(s',a') \\ &+ \gamma^2 \sum_{s'} p(s'|s,a)\Upsilon^\pi(s',a') + \gamma^2 \sum_{s'} p(s'|s,a)Q^\pi(s',a')^2 - Q^\pi(s,a)^2. \end{aligned} \tag{6}$$

A straight-forward proof is given in the Appendix. Eq. 6 is the parametric distributional Bellman operator for the distributional state-action value function. In practice, due to a relatively large continuous state and action spaces, we use function approximation for representing the critic. Specifically, a neural network parameterized by ω is used for estimating the critic's mean and variance: $f_{critic}(s,a|\omega) \rightarrow \{\hat{Q}^\pi(s,a), \hat{\Upsilon}^\pi(s,a)\}$.

We learn the critic by using the Temporal Difference (TD) learning, where the TD error is obtained by assuming Gaussianity (maximum entropy distribution) and using the Wasserstein metric as the loss. While other losses such as KL-divergence could also be used, the distributional Bellman operator using the p-th Wasserstein metric has been shown to be a contraction operator for policy evaluations [22]. The p-th Wasserstein distance between two probability distributions u and v is defined as [23]:

$$W_p(u,v) \doteq \Big(\int_0^1 \big|F_u^{-1}(s) - F_v^{-1}(s)\big|^p \, ds \Big)^{1/p}, \tag{7}$$

where F^{-1} is the inverse cumulative distribution function (CDF). Assuming $u \sim \mathcal{N}(\mu_1, C_1)$ and $v \sim \mathcal{N}(\mu_2, C_2)$, the 2-Wasserstein distance simplifies to:

$$W_2(u,v) = ||\mu_1 - \mu_2||_2^2 + trace(C_1 + C_2 - 2(C_2^{1/2} C_1 C_2^{1/2})^{1/2}) \tag{8}$$

The critic will try to minimize the Wasserstein distance by backpropagating the gradient $\frac{\partial W_2}{\partial \omega}$.

3.2 CVaR Actor

For every state action $\{s,a\}$, we can use the critic's estimated $\hat{Q}^\pi(s,a)$, $\hat{\Upsilon}^\pi(s,a)$ (mean and variance of a Gaussian) to compute the $CVaR_\alpha$ measure in closed-form:

$$\Gamma^\pi(s,a,\alpha) \doteq CVaR_\alpha = Q^\pi(s,a) - (\phi(\alpha)/\Phi(\alpha))\sqrt{\Upsilon^\pi(s,a)}, \tag{9}$$

where $\phi(\cdot)$ is the standard normal distribution and $\Phi(\cdot)$ is its CDF: $\Phi(x) = \frac{1}{2}\big(1 + \text{erf}(x/\sqrt{2})\big)$. We use $\Gamma^\pi(s,a,\alpha)$ to explicitly denote the reliance of $CVaR$ on both the state, action, as well as a specific policy π. We stress that $\Gamma^\pi(s,a,\alpha)$ is a scalar term computable at every state-action pair $\{s,a\}$. In plain language, $\Gamma^\pi(s,a,\alpha)$ is the expected future CVaR when in state s and executing action a, and following policy π hereafter.

We now replace the standard objective of Eq. 1 by the risk-averse $CVaR_\alpha$ objective:

$$J_\alpha = \int_S \big[\rho^\pi(s) \int_A \pi_\theta(a|s)\Gamma^\pi(s,a,\alpha)\, da\big] ds. \tag{10}$$

Theorem 1. *Suppose that the MDP satisfied conditions A.1 and A.2 (see Appendix), then:*

$$\nabla_\theta J_\alpha = \mathbb{E}_{s\sim\rho, a\sim\pi}[\nabla_\theta \log \pi_\theta(a|s)\Gamma^\pi(s,a,\alpha)]. \tag{11}$$

Proof. The proof mainly follows the original Policy Gradient Theorem (PGT) [14], see Appendix.

Given Eq. 11, we follow DPG from Eq. 3 to derive the equivalent deterministic policy gradient, where we backpropagate the CVaR loss gradients through the critic and down to the actor/policy networks. We obtain the deterministic gradient for the actor network by following a similar derivation as in [16]:

$$\begin{aligned}\nabla_\theta J_\alpha &= \mathbb{E}_{s\sim\rho,\alpha}[\nabla_\theta \pi(a|s,\alpha)\nabla_a \Gamma^\pi(s,a,\alpha)] \\ &= \mathbb{E}_{s\sim\rho,\alpha}\big[\nabla_\theta \pi(a|s,\alpha)\nabla_a \hat{Q}^\pi(s,a,\alpha) - (\phi(\alpha)/\Phi(\alpha))\nabla_\theta \pi(a|s,\alpha)\nabla_a \sqrt{\hat{\Upsilon}^\pi(s,a,\alpha)}\big]. \end{aligned} \tag{12}$$

Note that our new objective J is dependent on "risk level" α, and the question now arises as to how to choose the percentile α, which ranges from 0.0 to 1.0. One strategy is to discretize α into N

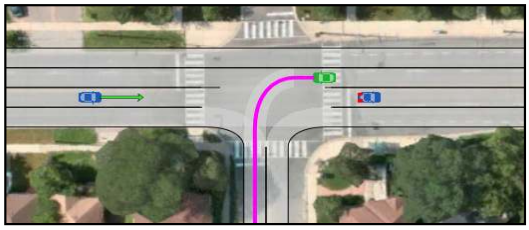


Figure 3: *Unprotected left turn environment.*

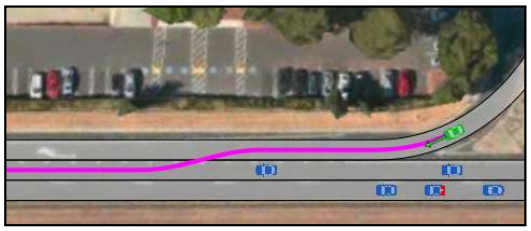


Figure 4: *Highway merge environment.*

discrete values and train N separate policy networks optimizing for N different settings. Instead of this naive approach with N times more parameters, we learn a single network which is conditioned on α as an additional input (see Fig. 1). The advantage of this approach is that we can learn a continuous family of α conditional policies $\pi_\theta(a|s, \alpha)$ with varying risk-sensitivity.

During training, a different input α will have a different loss function J_α. To train for all αs, we uniformly sample $\alpha \sim Uniform(0, 1)$ during the *start* of an episode and fix α for the entirety of that episode. During inference, π can output *different* actions given the same exact state s, conditioned on the setting of α. Intuitively, a small α leads to conservative actions while a larger α leads to more aggressive actions.

We now have the learning equations for both the actor and the critic. We employ an off-policy training algorithm by using an experience replay buffer. Algorithm 1 in the Appendix outlines the entire WCPG training procedure. Fig. 1 provides an overall illustration of the entire WCPG architecture and gradient flow during training.

4 Experimental Results

4.1 Environments

We tested our algorithm on two continuous-action 2D driving environments, focusing on two of the more critical scenarios: unprotected turns and merges (Figs 3, 4). In our environments, each agent is a vehicle, with the goal of getting from point A to point B while staying on the road and avoiding collisions. We simulate the vehicle dynamics using a discrete time kinematics bicycle model [24]. The simulation timestep is 100 milliseconds. We restrict our action space to be the acceleration of the ego vehicle[2]. The steering is determined by the Stanley controller, a non-linear closed loop feedback steering controller [25]. The physical dimension of our simulation environments is approximately 200 meters by 200 meters, where agents are randomly spawned on specific "birth" lanes with a given probability. Ego's initial velocity is randomly chosen between 5 to 20 m/s.

As a part of our multi-agent environments, non-ego agents are controlled by rule-based behaviors. Their velocity is randomly chosen and they have the ability to perform adaptive cruise control: slowing down and speeding up according to the vehicle in front of them. They can also perform safe lane changes by dynamically planning a smooth trajectory in order to merge from one lane to another. To introduce more realistic behaviors, the non-ego agents, during spawning, each samples randomly from one of three behaviors (when close to ego): *yield, ignore*, or *accelerate*, mimicking human drivers who might be conservative, distracted, or aggressive.

The reward for catastrophic failure (e.g. collision) is -50.0, while the reward for successful completion of a maneuver depends on time-to-completion: $50 \times e^{-steps/50} + 10$. A failure to complete the maneuver results in a reward of 0. We initially attempted to learn safer policies by simply scaling up the collision penalties (e.g. -1000.0), however we found that the resulting policies learned to be overly conservative and often did not even attempt the maneuvers. See Appendix for more details.

4.2 WCPG Network

Our network is an actor-critic based on DDPG [8] with two modifications. The first is that an additional α CVaR input is added to both the actor and the critic. The second is that instead of producing a scalar value for approximating $Q(s, a)$, the critic outputs parameters which govern the distribution of future returns. We use the *softplus*[3] function to guarantee that the critic's estimation of

[2] *Ego* refers to the vehicle for which we are learning a policy. We bound the acceleration and deceleration of all vehicles to $\pm 4\ m/s^2$, similar to that of a typical real vehicle.

[3] $f(x) = \log(1 + \exp(x))$

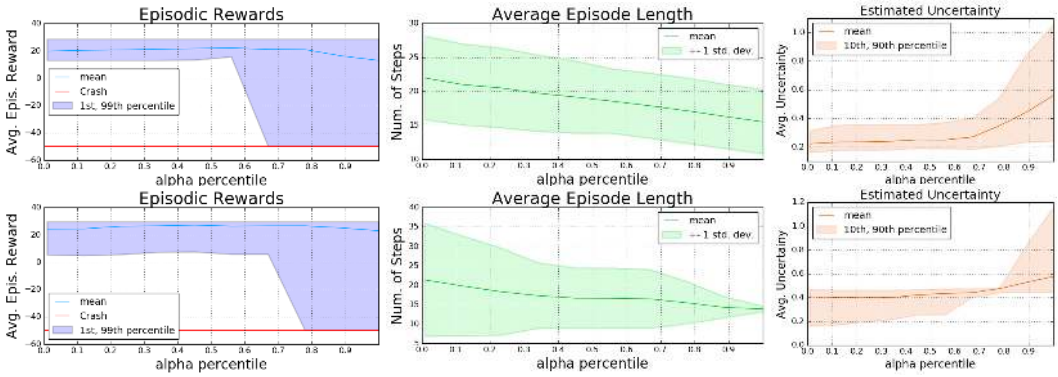


Figure 5: *Testing performance for the unprotected turn (top row) and merge (bottom row) environments.* ***Left****: we plot the distribution of episodic reward and their 1st and 99th percentile as a function of* α*. Red line indicates the reward incurred for a collision. The policy function successfully learned to be more risk-averse as* $\alpha \to 0$*.* ***Middle****: the number of steps to complete an episode increases as* α *decreases, due to the more conservative behavior of the learned policy.* ***Right****: average episodic standard deviation of future return estimated by the critic, as a function of* α*.*

variance will always be positive. See Fig. 1 for the diagram of our network model. The input to our network is 16 dimensional and consists of encoding the closest vehicles around ego (in sorted order) and their states: (x, y) position, heading, and velocity. The actor consists of 3 hidden ReLU layers with 32 hidden units each. The output of the actor is ego's acceleration. The critic network consists of 4 hidden layers with 64 hidden units each. We did not find the performance to be particularly sensitive to the choice of number of layers and layer size. See Appendix for more details.

Off-policy training is performed with an experience replay buffer of up to 10^6 $\{s, a, r, s'\}$ tuples. The learning rate for both the actor and the critic is 0.0001, and the minibatch size is 512. The same action is executed 4 times in a row for a total of 400 ms. Exploration noise is Gaussian with a standard deviation of 2.0. Input mean and variance for normalization is calculated in an online moving average fashion. α is randomly sampled from $[0.01, 1.0]$. Training is run until convergence for 5000 episodes, where each episode can last up to 30 seconds in simulation (300 timesteps).

4.3 Testing Performance

We now look at WCPG performs on test environments[4]. Results are shown for the two environments in Fig. 5. In the left panel, we can see that as α decreases, the policy becomes risk-averse and more robust, reducing the probability of collisions. In the middle, we can see that smaller αs leads to more conservative behavior (e.g. waiting for a wider gap), increasing the time-to-completion for both environments. Finally, the right panel shows that the critic's own estimate of uncertainty grows with an increasing α. This also means that the actor also has a higher risk-tolerance with an increasing α.

4.4 Extrapolation Performance

A challenging test is in the ability for policies to extrapolate, where the parameters of testing environments are "out-of-distribution" or extrapolated from the training distribution. Specifically, we significantly increase the upper-bound of non-ego agents' velocity by an additional 15 m/s and the spawn rate upwards of 8%. The total number of spawned agents is also increased (up to 4 additional agents) to make the scene denser.

Tables 1, 2 show the performance of WCPG and baselines on the extrapolation environments. We report collision rates for 100 random trials. Our results are highlighted in green. The training results are reported in the first row of each environment. We compare with 4 state-of-the-art algorithms in DDPG, proximal policy optimization (PPO) [9], C51/Rainbow [22, 10], and D4PG [26]. We used open source implementations from OpenAI and Dopamine[5]. For DDPG training, we used default parameters, but ensured that the model architecture, batch size, exploration noise, number of updates and other parameters matched that of the WCPG training parameters. PPO training was performed with minibatch size of 32, $\gamma = 0.99$, $\lambda = 0.95$, $clip = 0.2$, and learning rate of 0.0003. Training was performed until convergence.

[4]The random seeds for the testing environments are different compared to training environments.

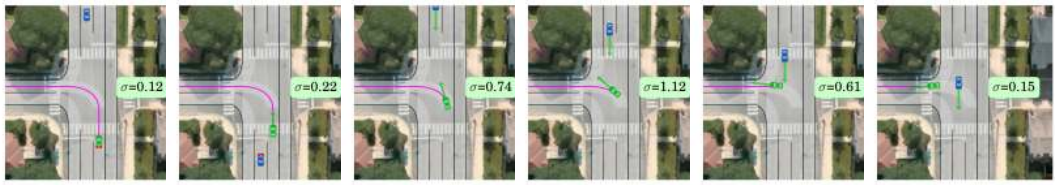
[5]https://github.com/openai/baselines, https://github.com/google/dopamine.

Table 1: *Left turn environment (extrapolation) - Collision rate % (Success rate %)*

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	0% (100%)	0% (100%)	0% (100%)	2% (98%)	15% (85%)	0% (100%)	4% (96%)	2% (98%)	2% (98%)
+5 m/s	5%	0% (100%)	0% (100%)	0% (100%)	0% (100%)	15% (85%)	13% (87%)	24% (76%)	10% (90%)	4% (96%)
+10 m/s	5%	0% (100%)	0% (100%)	0% (100%)	0% (100%)	15% (85%)	18% (82%)	40% (60%)	11% (89%)	4% (96%)
+15 m/s	5%	0% (100%)	0% (100%)	0% (100%)	3% (97%)	21% (79%)	20% (80%)	49% (51%)	14% (86%)	2% (98%)
+0 m/s	2%	0% (100%)	0% (100%)	0% (100%)	0% (100%)	19% (81%)	7% (93%)	15% (85%)	4% (96%)	5% (95%)
+0 m/s	8%	0% (100%)	0% (100%)	0% (100%)	3% (97%)	26% (74%)	6% (94%)	15% (85%)	5% (95%)	1% (99%)
+10 m/s	8%	0% (100%)	0% (100%)	0% (100%)	2% (98%)	23% (77%)	19% (81%)	40% (60%)	11% (89%)	2% (98%)

Table 2: *Merge environment (extrapolation) - Collision rate % (Success rate %)*

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	0% (100%)	0% (100%)	0% (100%)	0% (100%)	8% (92%)	0% (100%)	0% (96%)	0% (100%)	6% (94%)
+5 m/s	1%	0% (4%)	1% (5%)	0% (54%)	2% (90%)	10% (88%)	4% (96%)	31% (68%)	9% (91%)	9% (91%)
+10 m/s	1%	0% (2%)	0% (5%)	0% (50%)	0% (93%)	8% (92%)	3% (97%)	26% (73%)	5% (95%)	8% (92%)
+15 m/s	1%	0% (1%)	0% (5%)	0% (57%)	1% (94%)	9% (91%)	1% (99%)	24% (76%)	6% (94%)	9% (91%)
+0 m/s	2%	1% (7%)	0% (11%)	1% (50%)	2% (78%)	18% (76%)	5% (95%)	40% (59%)	8% (92%)	14% (86%)
+0 m/s	3%	0% (6%)	0% (10%)	0% (43%)	2% (70%)	17% (79%)	5% (95%)	39% (59%)	9% (91%)	12% (88%)
+10 m/s	3%	0% (4%)	0% (8%)	1% (43%)	2% (76%)	11% (79%)	4% (96%)	33% (67%)	5% (95%)	10% (90%)



(a) t=0.0 s (b) t=4.8 s (c) t=6.0 s (d) t=6.4 s (e) t=6.8 s (f) t=7.2 s

Figure 6: *Critic's estimation of the standard deviation of the future return. Uncertainty increases dramatically as ego goes ahead of an oncoming vehicle and reduces as ego completes the maneuver. Variance is due to the fact that the behavior of the oncoming vehicles can be random (yield/ignore/accelerate). See text for details.*

For all of the methods, ***training*** performance is quite good with almost all of them achieving 0% crash rates. However, if we use *out-of-distribution* environment parameters, PPO shows severe overfitting. DDPG is better but also starts to experience collisions. Note that C51/Rainbow and D4PG are *distributional* RL techniques (highlighted in blue), but they do not optimize for any risk-sensitive criteria. For all baseline methods, the performance significantly degrades in the presences of changing environmental parameters. In contrast, WCPG policies are more robust as we reduce α, achieving 0% crash rate for the left turn environment and between 0% and 1% for the merge environment, while keeping the success rate high. See Appendix for full results.

4.5 Uncertainty Modeling

It is also interesting to examine how the critic predicts uncertainty by looking at the critic's own estimate of the standard deviation of future return. Fig. 6 shows six intermediate steps from a test episode along with critic's estimations. As ego approaches the intersection and begins to make the left turn, the variance gradually increases and crescendos when ego is in the path of oncoming traffic. As ego completes the turn, uncertainty quickly reduces.

4.6 Carla Simulation

We tested how well our policy can transfer to similar scenarios in the CARLA simulator [27]. It currently contains seven different towns, dozens of different vehicles, and simple traffic law abiding "auto-pilot" CPU agents. For our left turn and merge environments, we found locations within CARLA that had similar geometry (See Fig. 7). We extract the position, heading, and velocity of vehicles from the CARLA simulator and fed into our previously trained WCPG policy networks[6]. We report collision and success rates for 100 random trials in Tab. 3. We can see that even in a different simulation, lower α values improved robustness dramatically.

[6]Due to a much slower simulation speed, we leave training directly on CARLA as future work.

Figure 7: *CARLA scenarios. Left: 3D view. Right: top-down view.*

Table 3: *Collision and (success rates) for different α in CARLA scenarios.*

Unprotected Left Turn: (Town05)		
$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 1.0$
0% (100%)	24% (76%)	42% (58%)

Merge: (Town04)		
$\alpha = 0.2$	$\alpha = 0.5$	$\alpha = 1.0$
2% (83%)	4% (89%)	24% (76%)

5 Related Work

Risk-sensitive, safe RL and the related robust MDPs have been extensively studied in literature [28, 29, 30, 31]. A recent comprehensive survey on the topic is provided by [4], where safe RL is categorized into two main types. The first type is based on the modification of the exploration process to avoid unsafe exploratory actions. Strategies consist of incorporating external knowledge [32] or using risk-directed explorations [33, 34]. The second type modifies the optimality criterion [35, 36, 37, 38, 20] used during training. Our work falls under this latter category, where we try to optimize our policy to strike a balance between pay-off and avoiding catastrophic events.

Different types of "safe" criteria have been proposed: exponential utility functions [35], linear combination of return and variance [39], percentile performance [40], Sharpe ratio and other variance-related criteria [41]. The worst-case minimax criterion can also be directly maximized [36, 42]. In [36], $\hat{Q}$-Learning was introduced, where the $\hat{Q}$ function is the lower bound of the Q function. However, optimizing the minimax criterion can result in overly pessimistic policies [43]. Optimizing a constrained criterion (return) subjected to a bounded variance was proposed in [44].

Conditional Value-at-Risk (CVaR) is another criterion that is gaining popularity in various fields such as engineering and finance [11, 45]. It has various desirable properties, including coherence and the easy of computation for certain underlying distributions. Combing CVaR with reinforcement learning, policy gradients for a bounded CVaR was proposed by [46], while a sampling based algorithm for CVaR was discussed in [47]. However, these approaches directly estimate the gradients to CVaR and are often computationally expensive or require extensive trajectory roll-outs. In contrast, WCPG optimizes for CVaR *indirectly* by first using *distributional* RL techniques [48, 19, 49] to estimate the distribution of return and then compute CVaR from this distribution. Recently, there have been a resurgence of interest in distributional RL [22, 50, 26]. However, the estimated distributions have not been used to minimize any risk-sensitive criteria. While implicit quantile networks [51] do optimize for risk-sensitive measures, a disadvantage to their approach is the approximation of risk and its gradients via K discrete samples, which greatly increases computation complexity.

Our proposed WCPG builds on DDPG, a popular deep RL method for continuous control. We retain DDPG's advantage of a powerful and sample efficient off-policy method that works well for large continuous state and action space. Instead of directly optimizing for CVaR, which can be difficult, we compute CVaR in closed-form from our distributional critic's estimation of future return, without resorting to sampling. In addition, both our actor and critic take risk-tolerance α as input during training, which allows the learned policy to operate with varying levels of risk after training.

6 Discussions

We have proposed a novel actor-critic framework to learn risk-sensitive policies by maximizing CVaR. Our policies can be adjusted dynamically after deployment to select risk-sensitive actions. In simulated driving environments, we perform significantly better with a smaller α, when compared to other top RL algorithms. Modeling uncertainty with heavy-tailed or mixture distributions could be explored in the future. It would also be interesting to see if the automatic selection of α would be possible for completing a maneuver while minimizing risk exposure.

Acknowledgements We thank Barry Theobald, Hanlin Goh, Nitish Srivastava, Johannes Heinrich, and the anonymous reviewers for making this a better manuscript.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[3] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, and et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[4] J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015. URL http://jmlr.org/papers/v16/garcia15a.html.

[5] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[6] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[7] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[11] R. T. Rockafellar, S. Uryasev, et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

[12] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press, 2000.

[13] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. doi:10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[14] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *In Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[15] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015. URL http://arxiv.org/abs/1506.02438.

[16] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 387–395, 2014. URL http://jmlr.org/proceedings/papers/v32/silver14.html.

[17] M. Hausknecht and P. Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[18] A. Tamar, Y. Glassner, and S. Mannor. Optimizing the CVaR via Sampling. *ArXiv e-prints*, Apr. 2014.

[19] M. Sobel. The variance of discounted markov decision processes. 19:794–802, 12 1982.

[20] T. Morimura, M. Sugiyama, H. Kashima, H. Hachiya, and T. Tanaka. Parametric return density estimation for reinforcement learning. In *UAI 2010*, pages 368–375, 2010.

[21] A. Tamar, D. D. Castro, and S. Mannor. Learning the variance of the reward-to-go. *Journal of Machine Learning Research*, 17(13):1–36, 2016. URL http://jmlr.org/papers/v17/14-335.html.

[22] M. G. Bellemare, W. Dabney, and R. Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017. URL http://arxiv.org/abs/1707.06887.

[23] I. Olkin and F. Pukelsheim. The distance between two random vectors with given dispersion matrices. *Linear Algebra and its Applications*, 48:257–263, 1982.

[24] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015.

[25] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23 (9):661–692, 2006.

[26] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, A. Muldal, N. Heess, and T. Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[28] J. Bagnell, A. Y. Ng, and J. Schneider. Solving uncertain markov decision problems. *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-01-25*, 2001.

[29] A. Nilim and L. El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Oper. Res.*, 53(5):780–798, September-October 2005.

[30] J. Morimoto and K. Doya. Robust reinforcement learning. *Neural computation*, 17(2):335–359, 2005.

[31] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. *ICML*, 2017. URL https://arxiv.org/abs/1703.02702.

[32] P. Geibel and F. Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Int. Res.*, 24(1):81–108, July 2005. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622519.1622522.

[33] C. Gehring and D. Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1037–1044. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[34] E. L. M. Law. *Risk-directed exploration in reinforcement learning*. PhD thesis, McGill University, 01 2005.

[35] R. A. Howard and J. E. Matheson. Risk-sensitive markov decision processes. *Management science*, 18 (7):356–369, 1972.

[36] M. Heger. Consideration of risk in reinforcement learning. In *Machine Learning Proceedings 1994*, pages 105–111. Elsevier, 1994.

[37] O. Mihatsch and R. Neuneier. Risk-sensitive reinforcement learning. *Machine learning*, 49(2-3):267–290, 2002.

[38] V. S. Borkar. Q-learning for risk-sensitive control. *Math. Oper. Res.*, 27(2):294–311, May 2002. ISSN 0364-765X.

[39] M. Sato, H. Kimura, and S. Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16(3):353–362, 2001. doi:10.1527/tjsai.16.353.

[40] J. A. Filar, D. Krass, and K. W. Ross. Percentile performance criteria for limiting average markov decision processes. *IEEE Transactions on Automatic Control*, 40(1):2–10, 1995.

[41] A. Tamar, D. D. Castro, and S. Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012. URL http://icml.cc/2012/papers/489.pdf.

[42] M. L. Littman and C. Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *ICML*, volume 96, pages 310–318, 1996.

[43] C. Gaskett. Reinforcement learning under circumstances beyond its control, 2003.

[44] P. L.A. and M. Ghavamzadeh. Actor-critic algorithms for risk-sensitive mdps. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 252–260. Curran Associates, Inc., 2013.

[45] Y. Chow, A. Tamar, S. Mannor, and M. Pavone. Risk-sensitive and robust decision-making: a cvar optimization approach. In *Advances in Neural Information Processing Systems*, pages 1522–1530, 2015.

[46] Y. Chow and M. Ghavamzadeh. Algorithms for cvar optimization in mdps. In *Advances in neural information processing systems*, pages 3509–3517, 2014.

[47] A. Tamar, Y. Glassner, and S. Mannor. Optimizing the cvar via sampling. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 2993–2999, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9429.

[48] S. C. Jaquette. Markov decision processes with a new optimality criterion: Discrete time. *The Annals of Statistics*, pages 496–505, 1973.

[49] D. White. Mean, variance, and probabilistic criteria in finite markov decision processes: a review. *Journal of Optimization Theory and Applications*, 56(1):1–29, 1988.

[50] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos. Distributional reinforcement learning with quantile regression. *arXiv preprint arXiv:1710.10044*, 2017.

[51] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. Implicit quantile networks for distributional reinforcement learning. In *Proceedings of ICML 2018*, pages 1096–1105. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/dabney18a.html.

APPENDIX

A Proof of Proposition 1

For convenience, we first restate the definitions here:

$$Q^\pi(s,a) = \mathbb{E}\big[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)|s_0=s,a_0=a,\pi\big]$$

$$\Upsilon^\pi(s,a) = \mathbb{E}\big[\big(\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\big)^2|s_0=s,a_0=a,\pi\big] - Q^\pi(s,a)^2$$

For conciseness, we define $Q^\pi_{sa} \doteq Q^\pi(s,a)$, $\Upsilon^\pi_{sa} \doteq \Upsilon^\pi(s,a)$, $R^t_{sa} \doteq r(s_t,a_t)$, $\mathbb{E}^\pi_{sa}[\sum_{t=0}^{\infty}\gamma^t R^t_{sa}] \doteq \mathbb{E}\big[\big(\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)\big)^2|s_0=s,a_0=a,\pi\big]$. We also use s',a' to denote the next resulting state and action obtained from taking action a in state s, and $a' \leftarrow \pi(s')$.

Proof.

$$\Upsilon^\pi_{sa} = \mathbb{E}^\pi_{sa}\Big[\big(\sum_{t=0}^{\infty}\gamma^t R^t_{sa}\big)^2\Big] - (Q^\pi_{sa})^2 \tag{13}$$

$$= \mathbb{E}^\pi_{sa}\Big[\big(R^0_{sa} + \sum_{t=1}^{\infty}\gamma^t R^t_{s'a'}\big)^2\Big] - (Q^\pi_{sa})^2 \tag{14}$$

We now take out the first term out of the summation:

$$\Upsilon^\pi_{sa} = (R^0_{sa})^2 + 2\gamma^t R^0_{sa}\sum_{s'}p(s'|s)\mathbb{E}^\pi_{s'a'}\Big[\sum_{t=1}^{\infty}\gamma^t R^t_{s'a'}\Big] + \sum_{s'}p(s'|s)\mathbb{E}^\pi_{s'a'}\Big[\big(\sum_{t=1}^{\infty}\gamma^t R^t_{s'a'}\big)^2\Big] - (Q^\pi_{sa})^2 \tag{15}$$

Now, we can reuse the definition of Q^π_{sa} and Υ^π_{sa} to arrive at the recursion:

$$\Upsilon^\pi_{sa} = (R^0_{sa})^2 + 2\gamma^t R^0_{sa}\sum_{s'}p(s'|s)Q^\pi_{s'a'} + \gamma^2\sum_{s'}p(s'|s)\Upsilon^\pi_{s'a'} + \gamma^2\sum_{s'}p(s'|s)Q^\pi_{s'a'} - (Q^\pi_{sa})^2 \tag{16}$$

□

B Proof of Theorem 2

Condition A.1

The policy function π is deterministic: $\pi(a|s) \triangleq \delta(a')$, where $a' \leftarrow \pi(s)$ and $\delta(\cdot)$ is the Dirac delta function. This is a mild condition as our proposed WCPG is based on the deterministic policy gradient (DDPG), which also assumes deterministic policy functions.

Condition A.2

Given a particular state s and action a, the environment transition is deterministic, that is: $p(s'|s,a) \triangleq \delta(s')$, where $\delta(\cdot)$ is the Dirac delta function. We note here that environment determinism for continuous control is often made by widely popular RL environments. For example, both Mujoco simulator and the DeepMind Control Suite are deterministic. The original Atari Learning Environment (ALE) also has deterministic dynamics.

For clarity and without loss of generality, we define Γ without conditioning on risk parameter α:

$$\Gamma^\pi(s,a,\alpha) = Q^\pi(s,a) - \frac{\phi(\alpha)}{\Phi(\alpha))}\sqrt{\Upsilon^\pi(s,a)} \tag{17}$$

$$\Gamma^\pi_{sa} = Q^\pi_{sa} - C\sqrt{\Upsilon^\pi_{sa}}, \tag{18}$$

where C is an α dependent constant.

Proposition 3. *Let us define Γ^π_s to be the expected conditional Value-at-Risk for policy π starting from state s, and given Condition A.1 is satisfied,*

$$\Gamma^\pi_s = \sum_a \pi(a|s)\Gamma^\pi_{sa}. \tag{19}$$

Proof. It is trivial to see that as π approach the Dirac delta function, probability mass concentrate on the chosen deterministic action a.

Proposition 4. *Provided that Condition A.2 is satisfied,*

$$\Gamma^{\pi}(s,a) = r(s,a) + \gamma \sum_{s',a'} p(s'|s,a)\Gamma^{\pi}(s',a') \tag{20}$$

Proof.

$$\Gamma^{\pi}_{sa} = Q^{\pi}_{sa} - C\sqrt{\Upsilon^{\pi}_{sa}} \tag{21}$$

$$= \big(r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'}\big) - C\sqrt{\Upsilon^{\pi}_{sa}} \tag{22}$$

From Eq. 4 in proof of Proposition 1, we substitute Υ^{π}_{sa}:

$$\Gamma^{\pi}_{sa} = \big(r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'}\big) \tag{23}$$

$$- C\sqrt{(R^0_{sa})^2 + 2\gamma R^0_{sa} \sum_{s'} p(s'|s) Q^{\pi}_{s'a'} + \gamma^2 \sum_{s'} p(s'|s) \Upsilon^{\pi}_{s'a'} + \gamma^2 \sum_{s'} p(s'|s) Q^{\pi}_{s'a'} - (Q^{\pi}_{sa})^2}. \tag{24}$$

Given Condition 2, we simplify expectations involving summations over s', and rearranging:

$$\Gamma^{\pi}_{sa} = \big(r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'}\big) - C\sqrt{(R^0_{sa})^2 + 2\gamma^t R^0_{sa} Q^{\pi}_{s'a'} + \gamma^2 \Upsilon^{\pi}_{s'a'} + \gamma^2 Q^{\pi}_{s'a'} - (Q^{\pi}_{sa})^2} \tag{25}$$

$$= \big(r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'}\big) - C\sqrt{\gamma^2 \Upsilon^{\pi}_{s'a'} + (R^0_{sa} + 2\gamma Q^{\pi}_{s'a'})^2 - (Q^{\pi}_{sa})^2} \tag{26}$$

$$= \big(r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'}\big) - C\sqrt{\gamma^2 \Upsilon^{\pi}_{s'a'} + (Q^{\pi}_{sa})^2 - (Q^{\pi}_{sa})^2} \tag{27}$$

$$= r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) Q^{\pi}_{s'a'} - \gamma C\sqrt{\Upsilon^{\pi}_{s'a'}} \tag{28}$$

$$= r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) \big(Q^{\pi}_{s'a'} - C\sqrt{\Upsilon^{\pi}_{s'a'}}\big) \tag{29}$$

$$= r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) \Gamma^{\pi}_{s'a'} \tag{30}$$

□

Proof of Theorem 2.

Proof. We mainly follow the start state formulation of the Policy Gradient Theorem of (Sutton et al. 2000), but also making use of Propositions 3 and 4.

$$\frac{\partial \Gamma^{\pi}_{s}}{\partial \theta} \doteq \frac{\partial}{\partial \theta} \sum_{a} \pi(a|s) \Gamma^{\pi}_{sa}, \quad \forall s \in \mathcal{S} \qquad \text{(Proposition 3)} \tag{31}$$

$$= \sum_{a} \left[\frac{\partial \pi(a|s)}{\partial \theta} \Gamma^{\pi}_{sa} + \pi(a|s) \frac{\partial \Gamma^{\pi}_{sa}}{\partial \theta} \right] \tag{32}$$

$$= \sum_{a} \left[\frac{\partial \pi(a|s)}{\partial \theta} \Gamma^{\pi}_{sa} + \pi(a|s) \frac{\partial}{\partial \theta} \Big[r_{sa} + \gamma \sum_{s',a'} p(s'|s,a) \Gamma^{\pi}_{s'a'} \Big] \right] \qquad \text{(Proposition 4)} \tag{33}$$

$$= \sum_{a} \left[\frac{\partial \pi(a|s)}{\partial \theta} \Gamma^{\pi}_{sa} + \pi(a|s) \gamma \sum_{s',a'} p(s'|s,a) \frac{\partial \Gamma^{\pi}_{s'a'}}{\partial \theta} \right] \tag{34}$$

$$= \sum_{x} \sum_{t}^{\infty} \gamma^t Pr(s \to x, t, \pi) \sum_{a} \frac{\partial \pi(a|x)}{\partial \theta} \Gamma^{\pi}_{x,a},$$

where we have used $Pr(s \to x, k, \pi)$ to denote the probability of going to state x from state s under policy π in k steps. This can be seen after unrolling (Eq. 22) for a few steps. It can then be seen

that:

$$\frac{\partial J}{\partial \theta} = \frac{\partial \Gamma^{\pi}_{s_0}}{\partial \theta}, \text{ for some initial state } s_0 \tag{35}$$

$$= \sum_s \sum_{t=0}^{\infty} \gamma^t Pr(s_0 \to s, t, \pi) \sum_a \frac{\partial \pi(a|s)}{\partial \theta} \Gamma^{\pi}_{s,a} \tag{36}$$

$$= \sum_s \rho(s) \sum_a \frac{\partial \pi(a|s)}{\partial \theta} \Gamma^{\pi}_{s,a} \tag{37}$$

Using the log-derivative trick we can further write the gradient as:

$$\frac{\partial J}{\partial \theta} = \sum_s \rho(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) \Gamma^{\pi}(s,a) \tag{38}$$

$$\tag{39}$$

And for conditional objective conditioned on α, we have:

$$\frac{\partial J_\alpha}{\partial \theta} = \sum_s \rho(s) \sum_a \pi_\theta(a|s,\alpha) \nabla_\theta \log \pi_\theta(a|s,\alpha) \Gamma^{\pi}(s,a,\alpha) \tag{40}$$

□

C Algorithm

Algorithm 1: WCPG training algorithm

Input: Environment $\mathcal{E}$, actor-critic net hyperparameters, M episodes, T steps/episode, replay buffer $\mathcal{B}$.
Initialize: Randomly initialize actor π and critic, $\mathcal{B} \leftarrow \emptyset$.

for *episodes* $e = 0$ *to* M **do**
  $s_0 \leftarrow \mathcal{E}$.reset(), $\alpha \sim U(0,1)$
  for *steps* $t = 0$ *to* T **do**
    action $a_t \leftarrow \pi_\alpha(s_t)$; add exploration noise.
    $\{s_{t+1}, r_t, done\} \leftarrow \mathcal{E}$.step($a_t$)
    $\mathcal{B} \leftarrow \{s_t, a_t, r_t, s_{t+1}, \alpha\}$
    Randomly sample a minibatch from $\mathcal{B}$.
    Update critic: use Eqs. 6 and 8.
    Update actor with the CVaR objective Eq. 12.
    if *done* **then** goto next episode.
  end
end

D Distributional Critic

Our distributional critic uses the Gaussian approximation as it allows us to have closed form CVaR computation, which is critical as we must compute CVaR for every update step, for every data tuple. In addition, ease of computation is important when we want to learn a policy which is conditioned on the risk α, where we have to compute CVaR for multiple αs.

While a Gaussian is unimodal and not heavy-tailed, our environmental reward is lower-bounded by the negative reward caused by a collision. In this case, Gaussian distribution's light-tail is not terribly restrictively, since we do not have unbounded loss. In addition, it is important to note that the unimodal Gaussian approximation is for a particular state-action pair. The value distribution for a particular state s (e.g. distributional $V(s)$), which marginalizes over the actions, could still be multi-modal with even potentially different variances for different modes.

E Experiments

E.1 Driving Simulation

We focused our experiments on self-driving environments as safety and risk-sensitive decision making is paramount in this application. They are also multi-agent environments, where the latent

behaviors of other agents are not always observable. Specifically, as noted in the paper, the other on-coming vehicles have a random probability of 3 behaviors: yielding, ignoring, or aggressively accelerate. This stochasticity creates inherent uncertainty in the environment and leads to the need for a distributional RL.

E.1.1 Driving simulation experiment details

See Table 4 for the details of the training environment parameters.

Table 4: Driving Environment Details

Environment	Scale (m)	Ego vel. (m/s)	Agents vel.	Spawn rate	Yield/ignore/aggressive	$R_{success}$	$R_{collision}$
Unprotected Turn	240	[0, 20]	[10, 20]	1%	0%, 80%, 20%	$50 \times e^{-steps/50} + 10$	-50.0
Merge	180	[5, 20]	[5, 15]	1%	20%, 40%, 40%	$75 \times e^{-steps/50} + 5$	-50.0

E.1.2 Network Parameters

Table 5: WCPG actor and critic networks for driving environments.

Actor Network					Critic Network				
layer	dim.	prev.	acti.	σ	layer	dim.	prev.	acti.	σ
I	16	-	linear	-	I	16	-	linear	-
I_α	1	-	linear	-	I_{act}	1	-	linear	-
h_1	32	I	ReLU	$\sqrt{2}$	I_α	1	-	linear	-
h_2	16	I_α	ReLU	$\sqrt{2}$	h_1	64	I	ReLU	0.01
h_3	32	h_1, h_2	ReLU	$\sqrt{2}$	h_2	64	I_α	ReLU	0.01
O_{act}	1	h_3	tanh	$\sqrt{2}$	h_3	64	h_1, h_2, I_{act}	ReLU	0.01
					h_4	64	h_3	ReLU	0.01
					O_{critic}	2	h_4	linear	0.01
								softplus	0.01

We show the exact network parameters for the WCPG actor and critic network used for the driving environment simulations in Table 5.

E.2 Additional Proof-of-concept Experiment

E.2.1 Fast-Slow Lane Selection

We demonstrate the concept of WCPG in a simple discrete setting where we must learn a policy to decide to drive in the fast lane or the slow lane on a highway. The problem is a highly simplified proof-of-concept MDP where we can leverage brute-force sampling to perform policy improvement and find the optimal policy. This allows us to test the core concepts behind WCPG independently from the various approximations needed in learning deep RL policies for more complex environments. Imagine a vehicle is traveling on a two lane freeway and needs to learn a policy for performing lane *selection*. The left lane is the fast lane while the right lane is the slower lane. At every timestep, the rewards are stochastic and the fast lane reward has both higher mean and higher variance than the slow lane. The stochastic policy we wish to learn is the probability of lane change at each timestep.

We formulate this as a discrete finite-horizon MDP with two states, s_{left}, s_{right} for the vehicle being in the left and right lanes, respectively. The MDP is shown in Fig. 8, where the total number of time steps is 4. The initial state at $t = 0$ is the right lane and at each timestep the policy outputs the probability of lane change: $Pr(left\ lane) = 1.0 - Pr(right\ lane)$. When the vehicle is in the left lane, it receives a reward randomly sampled from $\mathcal{N}(2.0, 4.0)$. When the vehicle in the right lane, its reward is sampled from $\mathcal{N}(1.0, 1.0)$. This roughly models the assumption of higher risk and rewards for the fast lane, while the slow lane might be safer but would take longer to get to the destination.

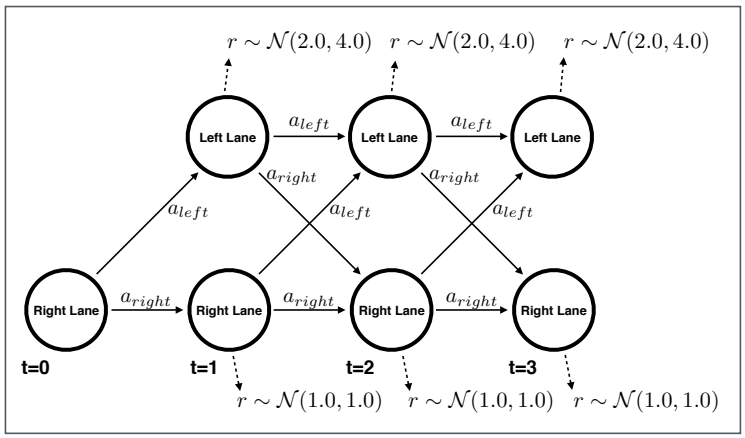


Figure 8: *MDP for the Fast-Slow Lane Selection problem. The goal is to learn a policy for choosing either the left(fast) or the right(slow) lane, see text for details. This proof-of-concept problem demonstrate the core concepts of WCPG.*

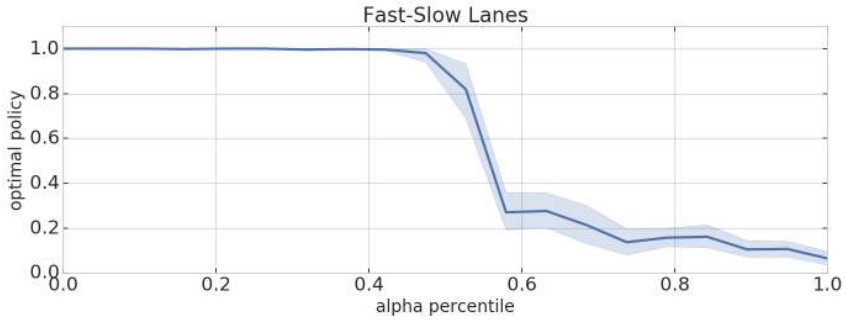


Figure 9: *Optimal policy as a function of α percentile.*

Using this environment, we are interested in learning an α conditioned optimal policy and to verify our hypothesis that different αs will lead to different policies. The MDP is simple enough such that sampling based policy improvement schemes will arrive at the optimal solution.

The Fast-Slow Lanes MDP can be solved via brute force sampling based policy improvement. For a given α, we grid the entire policy space. Specifically, we enumerate α from 0.0 to 1.0 with 32 intervals. Iterating through the policies, we numerically evaluate each policy using 1000 random trials, where the CVaR objective is computed from the statistics of the trials. The optimal policy is then found by selecting the policy that achieves the highest CVaR for any given α. We plot the optimal policies in Fig. 9.

The optimal policy is a scalar specifying the probability for left/right lane selection. When α is small, the optimal policy is to stay in the right lane as the rewards have a much smaller variance. Conversely, when α approaches 1.0, the optimal policy is to choose the left lane as we obtain higher rewards in expectation. This results verifies our hypothesis and is expected given the specified distribution of rewards. While this is a simple proof-of-concept problem, it demonstrates that when a globally optimal solution can be found, the proposed CVaR objective does indeed lead to conditional policies that varies depending on user specified risk appetite.

E.3 Extrapolation Experiments Full Results

We report the full extrapolation results for the unprotected left and merge environments in Tables 6, 7, 8, 9. The numbers are all in percentages and we also report the standard errors of the mean, over 100 trials with random environment seeds.

Table 6: *Unprotected Left Turn (extrapolation) - Collision rate* %

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	0.0±0.0	0.0±0.0	0.0±0.0	2.0±1.4	15.0±3.6	0.0±0.0	4.0±2.0	2.0±1.4	2.0±1.4
+5 m/s	5%	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	15.0±3.6	13.0±3.4	24.0±4.3	10.0±3.0	4.0±2.0
+10 m/s	5%	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	15.0±3.6	18.0±3.9	40.0±4.9	11.0±3.1	4.0±2.0
+15 m/s	5%	0.0±0.0	0.0±0.0	0.0±0.0	3.0±1.7	21.0±4.1	20.0±4.0	49.0±5.0	14.0±3.5	2.0±1.4
+0 m/s	2%	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	19.0±3.9	7.0±2.6	15.0±3.6	4.0±2.0	5.0±2.2
+0 m/s	8%	0.0±0.0	0.0±0.0	0.0±0.0	3.0±1.7	26.0±4.4	6.0±2.4	15.0±3.6	5.0±2.2	1.0±1.0
+10 m/s	8%	0.0±0.0	0.0±0.0	0.0±0.0	2.0±1.4	23.0±4.2	19.0±3.9	40.0±4.9	11.0±3.1	2.0±1.4

Table 7: *Unprotected Left Turn (extrapolation) - Success rate* %

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	100.0±0.0	100.0±0.0	100.0±0.0	98.0±1.4	85.0±3.6	100.0±0.0	96.0±2.0	98.0±1.4	98.0±1.4
+5 m/s	5%	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	85.0±3.6	87.0±3.4	76.0±4.3	90.0±3.0	96.0±2.0
+10 m/s	5%	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	85.0±3.6	82.0±3.9	60.0±4.9	89.0±3.1	96.0±2.0
+15 m/s	5%	100.0±0.0	100.0±0.0	100.0±0.0	97.0±1.7	79.0±4.1	80.0±4.0	51.0±5.0	86.0±3.5	98.0±1.4
+0 m/s	2%	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	81.0±3.9	93.0±2.6	85.0±3.6	96.0±2.0	95.0±2.2
+0 m/s	8%	100.0±0.0	100.0±0.0	100.0±0.0	97.0±1.7	74.0±4.4	94.0±2.4	85.0±3.6	95.0±2.2	99.0±1.0
+10 m/s	8%	100.0±0.0	100.0±0.0	100.0±0.0	98.0±1.4	77.0±4.2	81.0±3.9	60.0±4.9	89.0±3.1	98.0±1.4

Table 8: *Merge (extrapolation) - Collision rate* %

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	8.0±2.7	0.0±0.0	0.0±0.0	0.0±0.0	6.0±2.4
+5 m/s	1%	0.0±0.0	1.0±1.0	0.0±0.0	2.0±1.4	10.0±3.0	4.0±2.0	31.0±4.6	9.0±2.9	9.0±2.9
+10 m/s	1%	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	8.0±2.7	3.0±1.7	26.0±4.4	5.0±2.2	8.0±2.7
+15 m/s	1%	0.0±0.0	0.0±0.0	0.0±0.0	1.0±1.0	9.0±2.9	1.0±1.0	24.0±4.3	6.0±2.4	9.0±2.9
+0 m/s	2%	1.0±1.0	0.0±0.0	1.0±1.0	2.0±1.4	18.0±3.9	5.0±2.2	40.0±4.9	8.0±2.7	14.0±3.5
+0 m/s	3%	0.0±0.0	0.0±0.0	0.0±0.0	2.0±1.4	17.0±3.8	5.0±2.2	39.0±4.9	9.0±2.9	12.0±3.3
+10 m/s	3%	0.0±0.0	0.0±0.0	1.0±1.0	2.0±1.4	11.0±3.1	4.0±2.0	33.0±4.7	5.0±2.2	10.0±3.0

Table 9: *Merge (extrapolation) - Success rate* %

Params		WCPG (varying α)					State-of-the-art baselines			
vel.	spwn.	0.02	0.1	0.3	0.6	1.0	DDPG	PPO	C51	D4PG
(train)	1%	100.0±0.0	100.0±0.0	100.0±0.0	100.0±0.0	92.0±2.7	100±0.0	96.0±2.0	100.0±0.0	94.0±2.4
+5 m/s	1%	4.0±2.0	5.0±2.2	54.0±5.0	90.0±3.0	88.0±3.3	96.0±2.0	68.0±4.7	91.0±2.9	91.0±2.9
+10 m/s	1%	2.0±1.4	5.0±2.2	50.0±5.0	93.0±2.6	92.0±2.7	97.0±1.7	73.0±4.5	95.0±2.2	92.0±2.7
+15 m/s	1%	1.0±1.0	5.0±2.2	57.0±5.0	94.0±2.4	91.0±2.9	99.0±1.0	76.0±4.3	94.0±2.4	91.0±2.9
+0 m/s	2%	7.0±2.6	11.0±3.1	50.0±5.0	78.0±4.2	76.0±4.3	95.0±2.2	59.0±4.9	92.0±2.7	86.0±3.5
+0 m/s	3%	6.0±2.4	10.0±3.0	43.0±5.0	70.0±4.6	79.0±4.1	95.0±2.2	59.0±4.9	91.0±2.9	88.0±3.3
+10 m/s	3%	4.0±2.0	8.0±2.7	43.0±5.0	76.0±4.3	79.0±4.1	96.0±2.0	67.0±4.7	95.0±2.2	90.0±3.0