Uncertainty Methods in Active Reinforcement Learning

by

Rohan Nuttall

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Some real-world deployments of deep reinforcement learning (RL) may require a human-in-the-loop. Whether to ask-for-help, obtain new demonstrations and data, or handle out-of-distribution states, many methods rely on uncertainty estimates from a neural network to determine when to solicit a human's assistance. In existing work, it is common to rely on variance from an ensemble of models as a proxy for when the agent is uncertain about taking an action, however there has been little investigation into comparing the efficacy of other methods. This thesis compares three methods for uncertainty estimation in the action-advising framework: bootstrapped ensembles, Monte Carlo dropout and variance networks. Additionally, the methods are assessed on whether they produce "calibrated" uncertainty estimates. Variance networks are proposed as being advantageous in the action-advising setting due to their advice efficiency and ability to capture uncertainty about the environment dynamics.

Preface

This thesis is an original work by 'Rohan Calum Nuttall'. No part of this thesis has been previously published.

Acknowledgements

Thank you to my supervisor, Matthew Taylor, for his guidance and support; his patience and flexibility allowed me to explore many ideas of interest and make my graduate studies a fulfilling experience. I learned much from his expertise and research style. I would like to thank my parents for their unrelenting encouragement and love, which was especially invaluable while completing a thesis during a global pandemic. I'm grateful to my fellow graduate students who made the experience a memorable one: Brad Burega, Vlad Tkachuk, Connor Stephens, Liam Peet-Pare, Alex Ayoub, David Tao, Jordan Coblin, Erfan Miahi, and Vincent Mai. I'd also like to thank Levi Lelis and Marlos Machado for sitting on my examination committee and encouraging rigour in empirical work. Thank you to Jonathan Schaeffer for inspiring me to pursue graduate studies at the University of Alberta. I have also deeply appreciated the support of Kunal, Matt, Arthur, Cam, Jay, Josh, Tom, Luca, Jenny, Boyan, Arynn, Katiana, and Jerome. Finally, I am thankful to Mia for being a source of warmth and happiness in my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a framework that enables computational agents to achieve goals from experience [1]. It achieves this by learning a policy that maps observations to actions in order to maximize a numerical reward signal from an environment. However, an agent must *explore* in order to collect data on the highest value actions to *exploit*. This exploration process can be especially costly in real-world domains where data is expensive to obtain, rewards may be sparse (i.e., feedback is only given when a task is completed, as opposed to on each step of training), or potentially harmful exploratory actions need to be avoided. Similar to how many applications of supervised machine learning benefit from (or require) interaction with human users, in order for reinforcement learning (RL) to operate successfully in the real-world, it can be desirable to ensure agents can effectively interact with human experts or non-experts. Existing work has shown success in leveraging human feedback to assist agents with learning complex, hard-to-specify goals [2], accelerate online policy learning from demonstrations [3], ensure agent alignment with user intentions [4], and enable agents to solicit advice from humans or other agents when in unsafe situations [5].

Action advising is a framework for human-in-the-loop reinforcement learning where a more capable *teacher* policy may instruct a *student* policy which action to take as it learns a task [6] with the goal of ideally outperforming the teacher. As no

assumptions need to be made about the similarity between the student and teacher, action advising is a highly flexible framework for interfacing humans and agents to collaborate in solving complex, safety-critical tasks. However, since requesting expert advice can be costly, the student may be restricted by the number of advice calls it can make. This requires algorithms that can effectively solicit help when it is needed most. Heuristics for distributing advice such as random advising, early advising, and importance advising, while placing no constraints on state representation, are limited by their inability to express an agent's uncertainty about a particular state [6] [7].

As *deep* reinforcement learning achieves state of the art performance in challenging domains such as the board game Go [8], video game play [9], robotics [10], and a variety of other non-linear control tasks [11] [12], methods for actively querying expert demonstrations and advice have increasingly leveraged measures of a model's *epistemic* uncertainty (i.e., that which can be reduced with more training data) from neural networks [13] [14]. Much existing work using deep RL relies on ensemble-based uncertainty estimates or Monte Carlo dropout as a proxy for when the agent is uncertain about taking an action [15] [16]. However, there has been little investigation into comparing the differences between these methods. Furthermore, using estimates of *aleatoric* uncertainty (i.e., that which comes from irreducible stochasticity in the environment) has not been explored in the action advising literature. As deployed agents may experience out-of-distribution states or stochastic environment dynamics, using methods that capture this form of uncertainty in the action advising setting is of critical importance for real-world applications. Beyond action advising, uncertainty estimates have been used to guide planning [17], perform online parameter tuning [18], improve sample efficiency [19], learn risk-averse policies [20], and drive exploration [21]. Well-calibrated uncertainties have also been shown to improve performance in model-based RL [22].

One particularly relevant application where uncertainty estimates in the action advising framework could be useful is to consider the problem of safely deploying a large

language model (LLM). As combining RL with LLMs begins to enable goal-directed tasks ranging from specialized customer service to legal reasoning [23] [24], human intervention will likely be required. If agents can reliably express when they are uncertain about taking a certain action, we can move closer towards robust deployments of RL-based systems. Understanding which uncertainty estimation methods are most effective at yielding this capability is an important research direction.

Key Research Question: What impact does the choice of uncertainty estimation method have on action advising performance?

As such, this thesis makes the following contributions:

1. Introduces a new method for uncertainty-aware action advising.
2. Analyzes the uncertainty estimates from three methods during training across several different tasks.
3. Proposes a new type of query threshold that removes the need for hyperparameter tuning.
4. Assesses the quality of uncertainty estimates in terms of calibration.

This thesis is structured into four further chapters. In Chapter 2, relevant background concepts in action-advising and uncertainty estimation are introduced. Chapter 3 presents the experimental design and empirical investigations into action advising with different uncertainty estimators. Chapter 4 discusses and evaluates each method according to uncertainty calibration. Finally, Chapter 5 presents key future research directions for the field.

Chapter 2

Background

In this section, we will provide an overview of some of the key concepts related to this thesis.

2.1 Reinforcement Learning

An RL agent continually interacts with an environment through the computational framework of a Markov decision process (MDP). A finite MDP is a formalization of sequential decision making problem described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. On every interaction step, the agent receives a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$. The environment transitions to a new state $s_{t+1} \in \mathcal{S}$ and the agent receives a reward $r_{t+1} \in \mathcal{R}$ according to the dynamics function $\mathcal{T}$, which describes the probability, $P(s_{t+1}, r_{t+1}|s_t, a_t)$, of the next state and reward occurring given the previous state and action. The discount rate, γ, determines the present value of future rewards. The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps states to actions so as to maximize the expected sum of discounted rewards.

The value of taking an action a in state s and following policy π thereafter is given by the action-value function:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \tag{2.1}$$

The *optimal* policy, π_*, is a policy that is better than or equal to all other policies

and share the same optimal action-value function:

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \tag{2.2}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. This function obeys the Bellman optimality equation which is an identity that expresses the value of a state-action pair according to that of its successors:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big] \tag{2.3}$$

In order to learn the optimal action value function, the Bellman equation can be used as an iterative update. The tabular Q-learning algorithm [25] uses the following update rule on every interaction with the environment:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big], \tag{2.4}$$

where α is the stepsize. However, in large state-action spaces it is impractical to use a table to represent $\mathcal{S} \times \mathcal{A}$, and a function approximator is required to estimate the action-value function.

2.2 Deep Q-learning

In deep Q-learning [26], the action-value function is approximated by a neural network with parameters θ called a deep Q-network (DQN). The DQN algorithm stores agent-environment interactions into a *replay buffer*, $\mathcal{D}$, and on each iteration, i, uses stochastic gradient descent (i.e., using the Adam optimizer [27]) on batches sampled from this buffer to minimize the mean-squared error loss function:

$$L_i^{DQN}(\theta_i) = \mathbb{E}_{(s,a,s',r)\backsim\mathcal{D}}\Big[(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i))^2\Big] \tag{2.5}$$

A separate neural network called the *target network* is used to represent $Q(s',a';\theta_{i-1})$ which remains fixed when optimizing the loss function $L_i^{DQN}(\theta_i)$. Ev-

ery N steps, the parameters from the Q-network are copied to the target network. However, DQN and other deep reinforcement learning algorithms, despite achieving human-level performance on some tasks, are known to be sample inefficient compared to humans, often taking tens of millions of environment interactions to converge [28]. In addition to seeking methods for enabling these agents to interact with humans, sample inefficiency is one of the motivations behind leveraging expert knowledge to accelerate learning.

2.3 Action Advising

When an expert (i.e., a human or other agent that is optimal or close to optimal) is available, the action advising framework allows a learning agent — also known as the student agent — to receive the action that the expert would take in a given state. Clouse (1997) showed that by integrating such an approach with learning the student agent can reduce sample complexity and eventually outperform the teacher [29]. It is only assumed that a common action set and communication channel exists, and that a limited advice budget is available. Requests for action advice can either be student-initiated or teacher-initiated. Torrey and Taylor (2013) find that the same amount of teacher-initiated advice given at different moments can result in different learning performance by evaluating four algorithms: early advising, whereby advice is given to the agent during the first n states the student encounters; importance advising, which provides advice if the difference between the maximum and minimum action-values of the teacher is above some threshold; mistake correcting, which adds an additional condition to importance advising by intervening if the student's intended action disagrees with the teacher's action; and predictive advising, whereby the teacher uses a classifier to predict the student's action and intervenes if they disagree for important states [6].

Recent work has focused more on student-initiated action advising to avoid the need for a teacher to monitor all the states a student encounters. Ilhan and Liebana (2020)

enable the student agent to assess the *novelty* of a state before asking for advice using random network distillation [30], and show this performs better than early advising methods [31]. Another approach, proposed by Da Silva et al. (2020), is to estimate the *uncertainty* of a state resulting in the Requesting Confidence-Moderated Advice (RCMP) algorithm, which outperforms early advising and importance advising both in terms of asymptotic performance and advice efficiency [14]. While RCMP uses an ensemble of neural networks, Ilhan et al. (2021) use Monte Carlo dropout [32] as a mechanism for computing uncertainty to determine when advice should be reused, though they do not address whether this uncertainty heuristic is preferred over Da Silva et al.'s method [7].

2.4 Requesting Confidence-Moderated Advice

The RCMP algorithm computes the student's uncertainty about a state by using an ensemble of action-value estimates from neural networks similar to the Bootstrapped DQN algorithm introduced by Osband et al. (2016) [21]. As the networks are randomly initiated and trained (i.e., using the DQN algorithm explained in Section 2.2) with different samples from the replay buffer, each one will output a slightly different estimate of the action values. The variance across the networks are used as an estimate of uncertainty:

$$\mu(s) = \frac{\sum_{a \in \mathcal{A}} var(\mathbf{Q}(s,a))}{|A|}, \tag{2.6}$$

where $\mathbf{Q}(s,a)$ is a vector of i action value estimates given by each network for state s and action a and A is the total number of action. The final value prediction used by the policy is the mean across all networks. If $\mu(s)$ is high and the teacher policy is available (i.e., there is budget left), then RCMP will request advice. However, it is not clear how the choice of uncertainty estimation method of a neural network prediction impacts performance in the RCMP framework.

2.5 Uncertainty Estimation

There are two primary types of uncertainties that can be modelled: aleatoric and epistemic uncertainty. Aleatoric uncertainty represents noise inherent in the data itself and cannot be reduced by collecting more samples. On the other hand, epistemic uncertainty relates to lack of knowledge about which true model the data is generated from and can be reduced with enough data and a model of sufficient capacity [33].

2.5.1 Bootstrap Ensemble

Inspired by the statistical bootstrap [34], training N randomly initialized networks on independent samples from the same replay buffer can be used to obtain a distribution over predictions. The variance over the networks' outputs forms an estimate of epistemic uncertainty [21], [35]. During training, the variance of the ensemble (Equation 2.6) will be lower where sufficient samples exist, and higher in areas of the state-action space where there is less coverage.

2.5.2 Monte Carlo Dropout

Originally proposed as a regularization technique for reducing overfitting in neural networks [32], dropout has also been used to obtain a measure of epistemic uncertainty [36]. By setting some hidden units in a neural network to zero with probability p, a neural network with dropout will result in slightly different predictions for the same input. However, in the context of RL, adding such additional variance to the agent's Q-value estimates is typically undesirable and can destabilize convergence to the optimal policy. Therefore, a separate network with dropout applied after each layer (trained in parallel) with the same architecture as the primary DQN agent is used to obtain the uncertainty estimate. For a given state, K stochastic forward passes through the dropout network are made to construct the matrix:

$$\mathbf{D} = \begin{bmatrix} Q_1(s,a_1) & Q_1(s,a_2) & \dots & Q_1(s,a_A) \\ Q_2(s,a_1) & Q_2(s,a_2) & \dots & Q_2(s,a_A) \\ \vdots & \vdots & \ddots & \vdots \\ Q_K(s,a_1) & Q_K(s,a_2) & \dots & Q_K(s,a_A) \end{bmatrix}$$

By computing the variance across each column of matrix $\mathbf{D}$, the total uncertainty for a given state can be obtained using Equation 2.6.

2.5.3 Variance Networks

Unlike dropout and ensembles, variance networks use the negative log-likelihood loss to learn the target's noise due to the stochasticity of the environment and as such serve as a measure of aleatoric uncertainty [33]. The variance network outputs two values: the mean $\mu_Q(s,a)$ and variance $\sigma_Q(s,a)$ of a Gaussian distribution corresponding to the value of a state-action pair. The parameter λ is used to balance the fact that the negative log-likelihood function down-weights labels with high variance in the optimization process [19]:

$$L^{VN}(\theta) = \mathbb{E}_{(s,a,s',r)\backsim\mathcal{D}}\left[\Big(y - \mu_{Q_\theta}(s,a)\Big)^2 + \lambda\frac{\Big(y - \mu_{Q_\theta}(s,a)\Big)^2}{\sigma^2_{Q_\theta}(s,a)} + \ln\sigma^2_{Q_\theta}(s,a)\right], \quad (2.7)$$

where $y = (r + \gamma \max_{a'} Q(s',a';\theta^-)$ is the prediction from the target network described in Equation 2.5. While dropout and ensembles are commonly used in the action advising literature as a signal for requesting advice, the use of variance networks has not been explored.

2.5.4 Calibration

It is desirable to have uncertainty estimates that are not overconfident so as to avoid scenarios where an agent should have actually asked for advice in a state, but did not due to the method underestimating the uncertainty. *Calibration* is a technique

that can be used to evaluate the quality of uncertainty estimates. In the regression setting, it involves assessing whether, say, the 90% confidence interval around a prediction contains the true outcome 90% of the time [37]. Lakshminarayanan et al. (2016) showed that dropout can produce poorly-calibrated uncertainty estimates in the stationary, classification setting [35], but no literature to our knowledge has investigated calibration in the context of action-values predicted by neural networks.

Chapter 3

Analyzing Uncertainty in Action Advising

In the previous chapter, we introduced the RCMP algorithm, discussed two popular mechanisms for quantifying uncertainty from deep neural networks, and proposed variance networks as a new method for enabling uncertainty-aware action advising. While Monte Carlo dropout and bootstrapped ensembles have been used as proxies for estimating the epistemic uncertainty of the agent [38], learning the variance from the agent's environment interactions can provide information about the aleatoric uncertainty of the environment. This chapter reports results from action advising performance as well as how the uncertainty estimates from these three methods differ during training.

We first determine the best hyperparameters for each method across a variety of advice thresholds. The base algorithm used in all experiments is the deep Q-network (DQN) [26], which is described in Section 3.1. Four environments with varying complexity are tested from OpenAI Gym: Cartpole, Acrobot, Lunar Lander, and Mountain Car [39]. Section 3.2 reports on the utility of variance networks compared with existing methods accordingly to two performance metrics: the number of advice calls required to reach a certain performance threshold (i.e., advice efficiency), as well as the number of episodes to reach a certain threshold (i.e., sample efficiency). Section 3.3 introduces a new type of query threshold that improves advice efficiency.

3.1 Deep Q-Network (DQN)

The base network architecture of the DQN implementation uses 2 ReLU-activated hidden layers with 256 units each to estimate the action value function. The exploration rate, ϵ, is initially set to 1.0 and decayed linearly to 0.01 after each episode as referenced along with the other hyperparameters in Table 3.1. On every environment step, a batch of 64 samples is drawn from a replay buffer and used to update the parameters of the Q-network (referred to as a *soft update*) with the mean squared error (MSE) loss and Adam optimizer [27]. The target network is updated every 5 episodes (referred to as a *hard update*).

The uncertainty estimates for the bootstrapped ensemble are obtained by first randomly initializing a set of N independent networks (each with its own target network) with Kaiming initialization [40]. The ensemble is trained by randomly sampling a new batch, with replacement, for each network from the replay buffer. The average variance across the network outputs is used to estimate uncertainty. For the Monte Carlo dropout method, dropout with probability p is applied to the final layer of a separate Q-network (trained in parallel) and the average variance across 10 forward passes is calculated. The variance network is implemented by using a single network with two heads which learn both the mean and variance of the action value estimate using the MSE and negative log-likelihood loss controlled by the parameter λ [33]. The uncertainty estimation parameter sweep strategy is described in the next section.

3.1.1 Parameter Sweep

The hyperparameters used across all the environments for the base DQN algorithm are reported in Table 3.1. To determine the best hyperparameters for comparing each uncertainty method, a range of uncertainty thresholds are chosen for each environment and the performance of each configuration in Table 3.2 is averaged across 15 runs

each initialized with a unique random seed. For each run, the sum of the total reward obtained over 400 episodes is recorded. The best performing parameter for each method is found by taking the Cartesian product between each method and each environment's fixed query threshold, q_F, for a total of 24 settings per method. The performance from the best performing configuration is reported every 10 episodes by plotting the mean return from 10 *evaluation* episodes (during which learning is paused) along with the standard error across 15 independent runs with different random seeds.

Table 3.1: Hyperparameters used for the base DQN algorithm.

Hyperparameter	**Value**
Discount Factor (γ)	0.995
Epsilon Decay Factor	0.05
Replay Buffer Size	10K
Hidden Units	256
Batch Size	64
Soft Update Frequency	Every environment step
Hard Update Frequency	Every 5 episodes
Optimizer	Adam
Learning Rate (∇)	0.0003

Table 3.2: Hyperparameters for each method. The best performing parameter for each method is found by taking the Cartesian product between each method and each environment's fixed query threshold, q_F, for a total of 24 settings per method.

Hyperparameter	**Value**
Dropout Probability (p)	0.1, 0.2, 0.4, 0.8
Ensemble Size (N)	2, 3, 4, 5
Loss Attenuation Factor (λ)	0.001, 0.01, 0.1, 1.0
Cartpole (q_F)	0.3, 0.5, 0.7, 0.9, 1.2, 1.5
Acrobot (q_F)	0.02, 0.05, 0.1, 0.5, 0.8, 1.0
Mountain Car (q_F)	0.2, 0.5, 0.7, 1, 2, 3
Lunar Lander (q_F)	1, 2, 4, 8, 16, 32

3.2 Fixed Uncertainty Threshold

This section evaluates the three methods using a fixed uncertainty threshold commonly used in the literature [7] [14].

3.2.1 Cartpole Results

The Cartpole task, depicted in Figure 3.1, is a dense reward (i.e., a non-zero reward is given to the agent on every step) control environment which consists of keeping a pole upright for as long as possible. The agent receives a reward of +1 for every step taken, and the episode is terminated if the pole angle from the starting position exceeds $\pm 12°$ or if the episode length exceeds 200 steps. The action space comprises two discrete actions: move cart left or right. The state vector has four dimensions: cart position, cart velocity, pole angle, and angular velocity. The fixed query threshold, q_F, for this task is set to 0.3 and the teacher agent (which is a pre-trained DQN with the same architecture as the student) used achieves a mean reward of 200.0 averaged over 50 episodes.

Figure 3.2 shows the performance of the three methods on Cartpole with a budget of 2000 advice calls. Both dropout and the ensemble perform comparably, with the

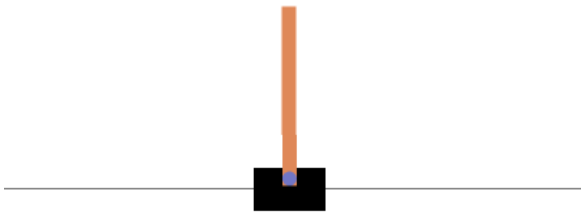


Figure 3.1: Rendering of Cartpole environment from OpenAI Gym [39].

variance network achieving the highest performance. It can be seen that dropout exhausts the advice budget within the first 25 episodes, while the ensemble only starts querying for advice after episode 30. This exposes the fact that the uncertainty distributions for each of these methods can sometimes be drastically different, as shown in Figure 3.3. This demonstrates that when a fixed uncertainty threshold is used in action advising with one of these three methods, it is important to tune q_F to both the environment *and* the method itself. This motivates the need for adaptive uncertainty thresholding, which is explored in Section 3.3.

Tables 3.3 and 3.4 report on the performance and advice efficiencies. It can be seen that the variance network requires less advice than dropout and the ensemble to reach higher performance level during training. Interestingly, the scale of the average uncertainty from dropout increases each episode while the other two methods fluctuate around the same scale. This accounts for why dropout exhausts most of the advice budget after the first 25 episodes, as the uncertainty never drops below $q_F = 0.3$ after episode 15.

This section also introduces the fact that dropout, ensemble and variance network produce very difference uncertainty estimates, resulting in advice being requested at different points during training for the same environment. On the Cartpole task with a fixed query threshold, the variance network achieves higher performance ($P \leq .001$) using less advice than dropout or the ensemble. For all environments, the Welch's

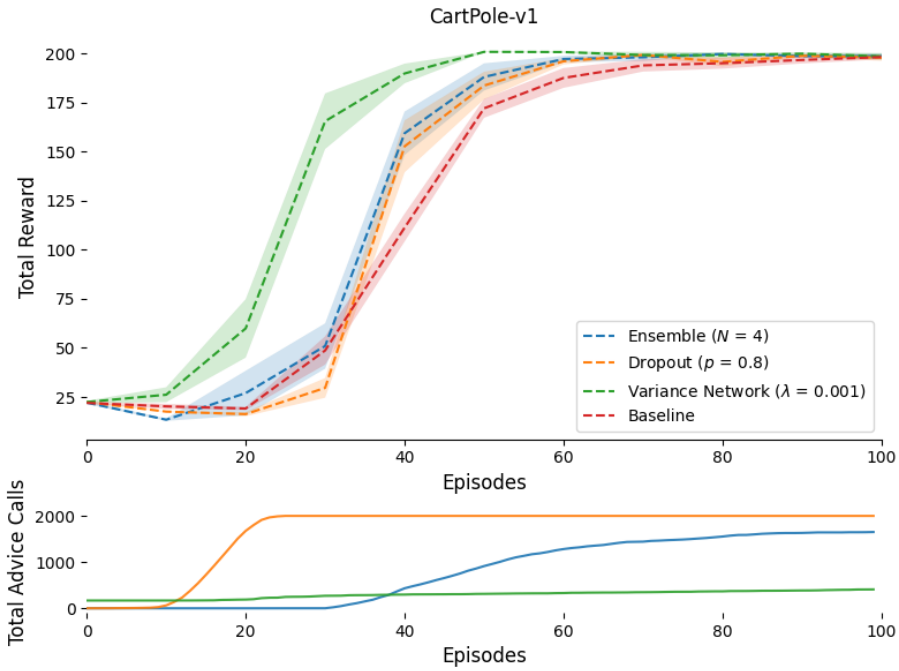


Figure 3.2: The performance of dropout ($p = 0.8$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 0.001$) on Cartpole with an advice budget of 2000. The ensemble and dropout achieve comparable performance, while the variance network performs best using the least amount of advice (as depicted in the bottom subplot). Every 10 episodes, the average reward from 10 evaluation rounds averaged across 15 independent runs is reported. Shaded regions depict the standard error across the 15 independent runs. The teacher agent used achieves a mean reward of 200.0 averaged over 50 episodes.

Table 3.3: Mean reward and standard error during training averaged over 10 evaluations rounds and 15 seeds on Cartpole. Total area under the curve (AUC) is reported in the last column along with asterisks depicting the statistical significance ($P \leq .001$) of the variance network performing better than the next best method.

Method	Episode 30	Episode 50	Episode 100	Total AUC
Ensemble	51 ± 13	188 ± 8	199 ± 2	1254 ± 33
Dropout	30 ± 6	184 ± 3	198 ± 2	1212 ± 19
Variance Network	$\mathbf{165 \pm 14}$	$\mathbf{200 \pm 2}$	$\mathbf{200 \pm 1}$	$\mathbf{1463 \pm 27}$***
Baseline	49 ± 3	172 ± 1	198 ± 1	1166 ± 27

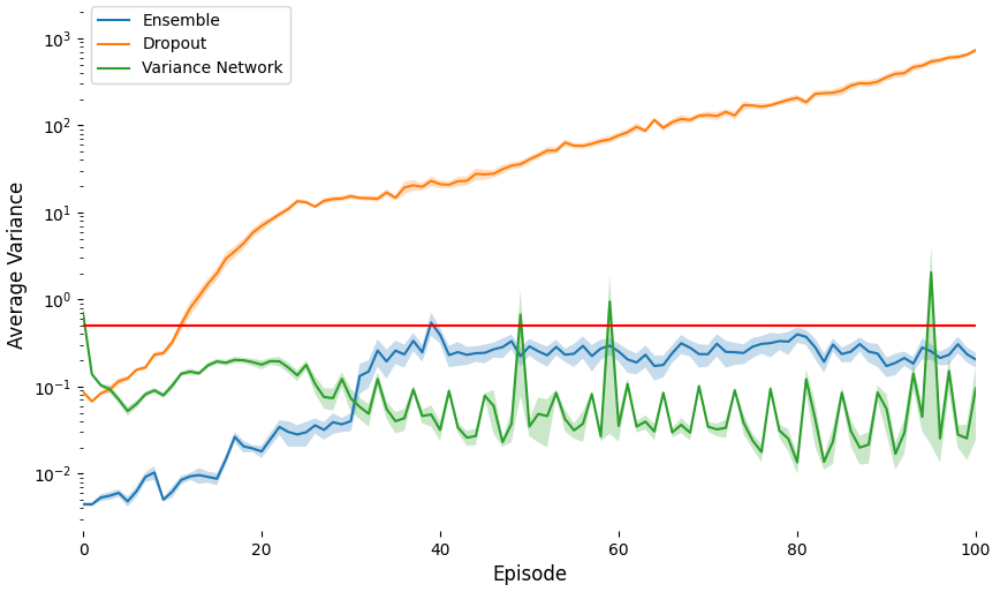


Figure 3.3: The average variance during training, averaged over 15 trials, from dropout ($p = 0.8$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 0.001$) on Cartpole. The fixed query threshold of 0.3, q_F, above which advice is queried is shown in red. It can be seen the three methods exhibit very difference uncertainty distributions both in terms of scale and behaviour.

Table 3.4: Mean advice calls with standard error during training averaged over 10 evaluations rounds and 15 seeds on Cartpole. The lowest advice used is indicated by bold, but is not statistically significant.

Method	Episode 30	Episode 50	Episode 100
Ensemble	$\mathbf{1 \pm 5}$	862 ± 72	1653 ± 61
Dropout	2000 ± 0	2000 ± 0	2000 ± 0
Variance Network	262 ± 4	$\mathbf{312 \pm 3}$	$\mathbf{409 \pm 1}$

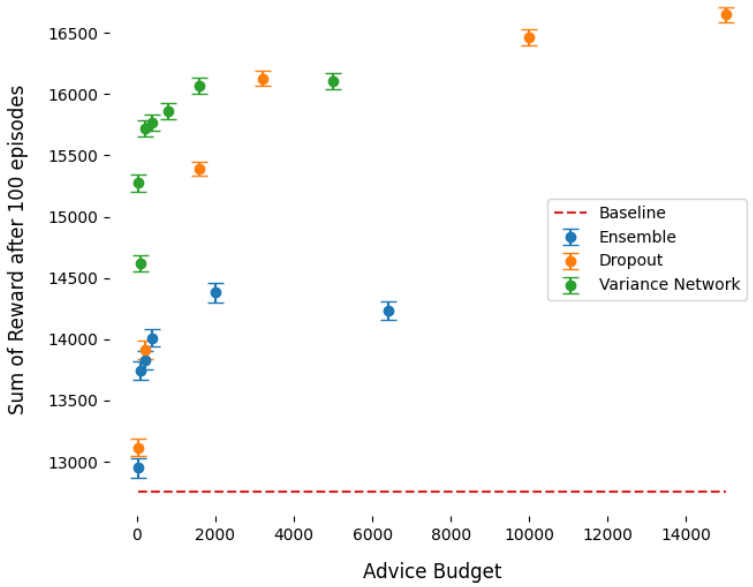


Figure 3.4: The Pareto frontier for advice budget and sum of reward after 100 episodes is depicted during learning. The mean reward and standard error after each episode across 15 seeds is summed and plotted along with the standard error across a variety of budgets.

t-test is used to test the null hypothesis that the variance network and next best performing method (in this case, the ensemble) are equivalent. For this environment, the Pareto frontier is also shown in Figure 3.4 to describe how the variance network performs better when less advice is available. However, if very large advice budgets are available then dropout appears to be the best choice.

3.2.2 Acrobot Results

The Acrobot environment consists of two links connected by two joints. The joint between the two links is actuated. An episode begins with the system hanging downwards and the goal of the agent is to swing the end of the lower link past the gray line depicted in Figure 3.5. A reward of -1 is given on all timesteps and the episode terminates either when the maximum number of steps (in this case 400) elapses, or the end of the free link reaches the target height. The discrete, 3-dimensional action space represents torque of -1, $+1$, or 0 applied to the actuated joint. The state space is 6-dimensional, consisting of: the sines and cosines of both joint angles, as well as their angular velocities. The fixed query threshold, q_F, for this task is 0.05 and the teacher agent used achieves a mean reward of -68.5 averaged over 50 episodes.

Figure 3.5: Rendering of Acrobot environment from OpenAI Gym [39].

Figure 3.6 shows the performance of the three methods on Acrobot with a budget of 2000 advice calls. While dropout and the variance network surpass the query threshold quite early on, it can be seen that the ensemble only begins querying for advice after episode 50. All methods exhaust the available advice within the first 150 episodes (see Table 3.6) and the variance network achieves best performance over the course of training. The variance network demonstrates slightly better asymptotic performance. Interestingly, the uncertainty distributions in Figure 3.7 converge to the same scale by the final episode.

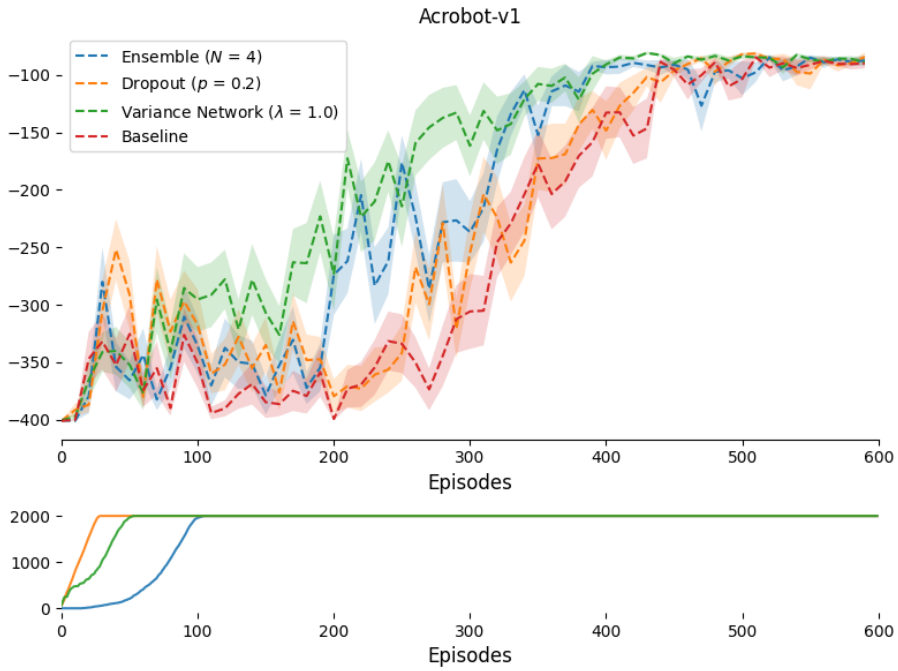


Figure 3.6: The performance of dropout ($p = 0.2$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 1.0$) on Acrobot with an advice budget of 2000. The variance network and ensemble achieve superior performance than dropout throughout training. Every 10 episodes, the average reward from 10 evaluation rounds averaged across 15 independent runs is reported. Shaded regions represent the standard error. The teacher agent used achieves a mean reward of -68.5 averaged over 50 episodes.

On the Acrobot task with a fixed query threshold, the variance network again achieves higher performance ($P \leq .001$), but does not use less advice than dropout or the ensemble.

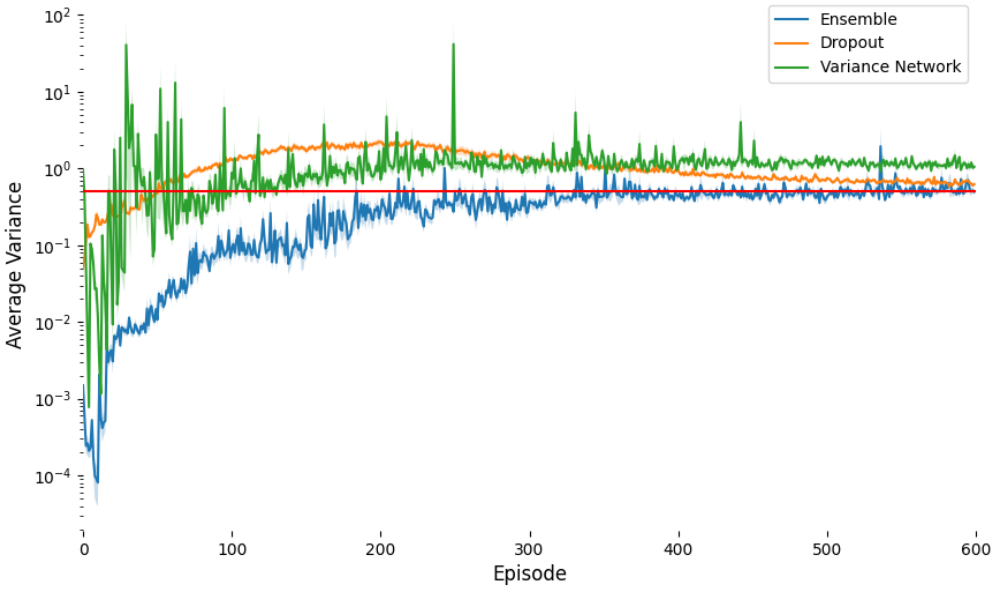


Figure 3.7: The uncertainty distributions from dropout ($p = 0.2$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 1.0$) on Acrobot. The fixed query threshold of 0.05, q_F, above which advice is queried is shown in red. The average uncertainty per episode from all three methods range significantly in scale, but the variance network exhibits comparatively higher volatility.

Table 3.5: Mean reward and standard error during training averaged over 10 evaluations rounds and 15 seeds on Acrobot. Total area under the curve (AUC) is reported in the last column along with asterisks depicting the statistical significance ($P \leq .001$) of the variance network performing better than the next best method.

Method	Episode 150	Episode 300	Episode 600	Total AUC
Ensemble	-379 ± 16	-237 ± 26	-89 ± 4	-12857 ± 389
Dropout	-335 ± 23	-255 ± 31	-90 ± 4	-13959 ± 257
Variance Network	$\mathbf{-307 \pm 28}$	$\mathbf{-162 \pm 26}$	$\mathbf{-86 \pm 3}$	$\mathbf{-11154 \pm 240}$***
Baseline	-385 ± 13	-306 ± 32	-91 ± 4	-15176 ± 341

Table 3.6: Mean advice calls with standard error during training averaged over 10 evaluations rounds and 15 seeds on Acrobot. All methods exhaust the advice budget within the first 150 episodes.

Method	Episode 150	Episode 300	Episode 600
Ensemble	2000 ± 0	2000 ± 0	2000 ± 0
Dropout	2000 ± 0	2000 ± 0	2000 ± 0
Variance Network	2000 ± 0	2000 ± 0	2000 ± 0

3.2.3 Mountain Car Results

Mountain Car is a popular classical control testing domain for reinforcement learning algorithms. It is a sparse reward exploration environment which comprises a car positioned in a valley, as indicated in Figure 3.8, that must drive up the steep mountainside. However, the car's engine alone is not powerful enough to overcome the force of gravity. Therefore, the only way to achieve the goal is to build up inertia by throttling back and forth within the valley. Eventually, the car will have enough momentum to escape the valley. On each timestep, the agent receives a reward of -1. Once the car reaches the top of the mountain and passes the goal marker, the episode ends. The actions available to the agent are: accelerate forward (+1), accelerate backwards (-1), and idle (0). The 2-dimensional state space comprises the position and velocity of the car. The query threshold, q_F, used for this environment is 0.7 and the teacher agent used achieves a mean reward of -129.3 averaged over 50 episodes.

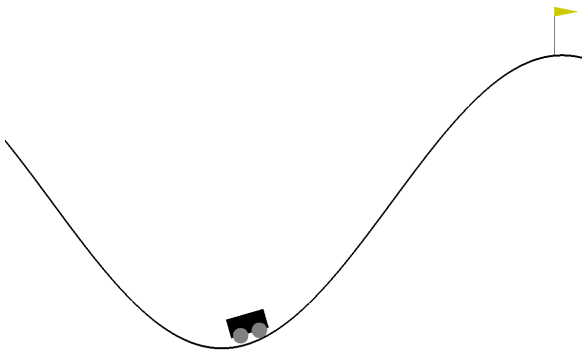


Figure 3.8: Rendering of Mountain Car environment from OpenAI Gym.

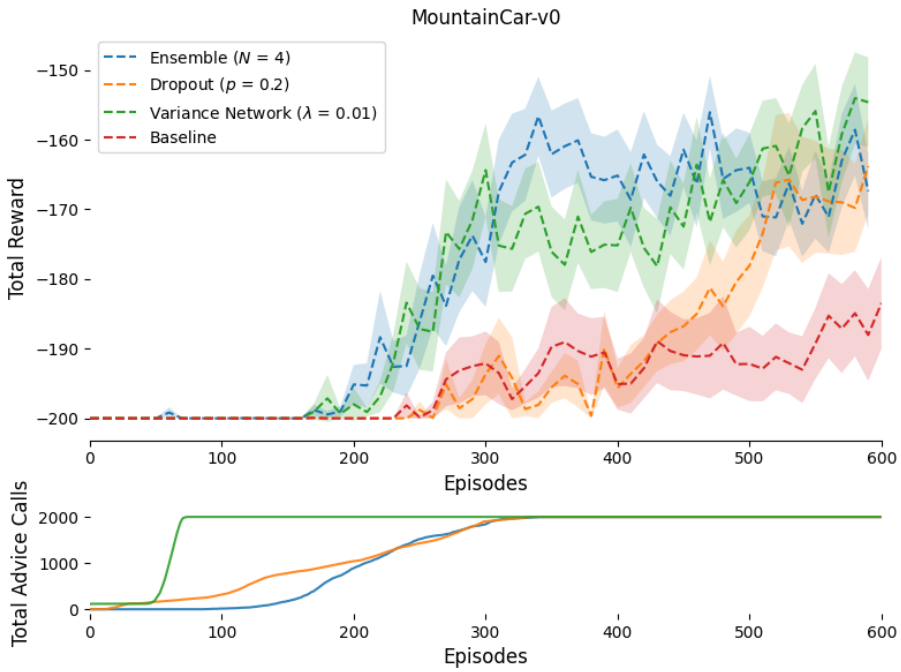


Figure 3.9: The performance of dropout ($p = 0.2$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 0.01$) on Mountain Car with an advice budget of 2000. The variance network and ensemble perform similarly, whereas dropout only begins to solve the task over 100 episodes later. Every 10 episodes, the average reward from 10 evaluation rounds averaged across 15 independent runs is reported. Shaded regions represent the standard error. The teacher agent used achieves a mean reward of -129.3 averaged over 50 episodes.

Figure 3.9 shows the performance of the three action advising methods with a budget of 2000 compared with the baseline. On this task, again the ensemble and variance network perform comparably. While the variance network achieves slightly higher performance at the middle and end of training as shown in Table 3.7, the advice it takes to reach the intermediate performance thresholds is considerably less with the ensemble as detailed in Table 3.8. It can be seen in Figure 3.10 that while the variance network's average uncertainty initially drops, it quickly increases passed the fixed query threshold around episode 50 and remains much higher than the other methods throughout training. Similar to the uncertainty distributions in Acrobot, while the scale originally differ, all three methods appear to follow a similar trend towards the end of training.

Table 3.7: Mean reward over and standard error during training averaged over 15 independent random seeds on Mountain Car. Total area under the curve (AUC) is reported in the last column. The variance network does not perform better than the ensemble in terms of total AUC. The best scores are reported in bold, but are not statistically significant.

Method	Episode 150	Episode 300	Episode 600	Total AUC
Ensemble	-200 ± 0	-178 ± 7	-167 ± 6	$\mathbf{-10818 \pm 117}$
Dropout	-200 ± 0	-194 ± 4	-164 ± 7	-11506 ± 134
Variance Network	$\mathbf{-200 \pm 2}$	$\mathbf{-164 \pm 5}$	$\mathbf{-155 \pm 5}$	-10906 ± 108
Baseline	-200 ± 2	-200 ± 4	-200 ± 6	-11712 ± 125

Table 3.8: Mean advice calls with standard error during training averaged over 10 evaluations rounds and 15 seeds on Mountain Car. The lowest advice used is indicated by bold.

Method	Episode 150	Episode 300	Episode 600
Ensemble	$\mathbf{174 \pm 17}$	$\mathbf{1834 \pm 18}$	$\mathbf{2000 \pm 0}$
Dropout	780 ± 53	1903 ± 16	2000 ± 0
Variance Network	2000 ± 0	2000 ± 0	2000 ± 0

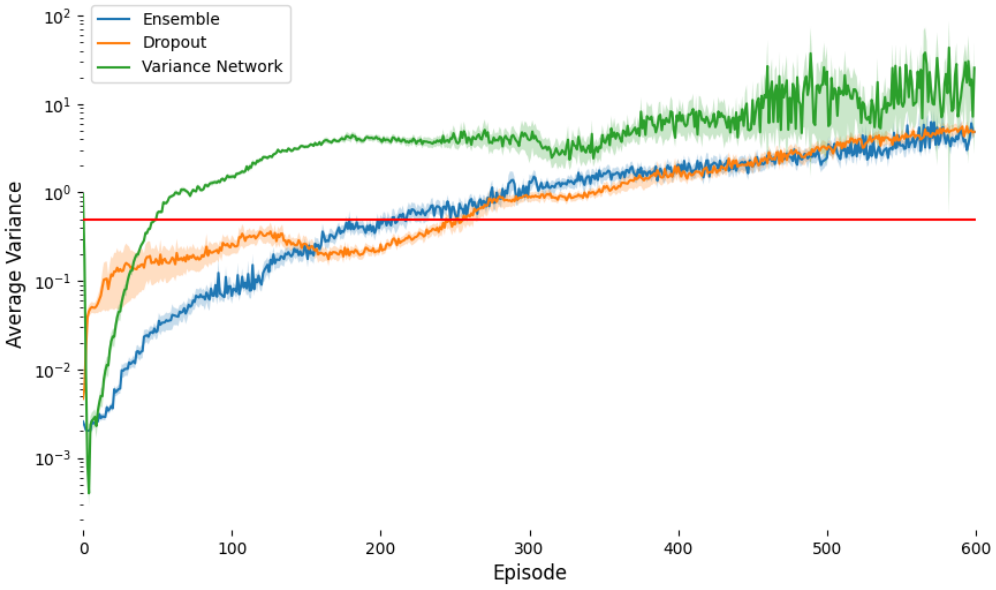


Figure 3.10: The uncertainty distributions from dropout ($p = 0.2$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 0.01$) on Mountain Car. The fixed query threshold of 16, q_F, above which advice is queried is shown in red. The average uncertainty per episode of the variance network remains significantly higher than dropout or the ensemble.

On the Mountain Car task with a fixed query threshold, the variance network does not achieve significantly better performance than the ensemble. The variance network also exhausts the advice budget within the first 150 episodes.

3.2.4 Lunar Lander Results

The goal of the Lunar Lander task is to learn a policy that safely guides the lander to the pad demarcated by the flags in Figure 3.11. The agent receives a reward of $+100$ for coming to rest, -100 for 'crashing,' and -0.3 for every step it uses the main engine. A soft landing receives a reward of 140 and the maximum reward of 200 is from a soft landing on the pad using the least amount of fuel. The four actions available to the agent are: fire left engine, right engine, main engine, or do nothing. The state-space is 8-dimensional, comprising the x and y coordinates of the lander, its linear velocities, angle, angular velocity, and two booleans indicating if its legs are in contact with the ground. An episode terminates if the lander body contacts the ground (signalling a crash), goes off-screen, or has come to a rest. The query threshold, q_T, for this task is 16 and the teacher used achieves a mean reward of 159.0 averaged over 50 episodes.

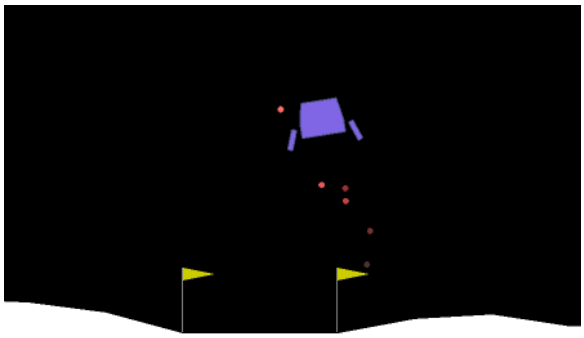


Figure 3.11: Rendering of Lunar Lander environment from OpenAI Gym.

Figure 3.12 shows the performance of the three methods with the advice budget of 2000. Early in training, none of the methods appear to perform substantially better than the baseline. The ensemble achieves the highest scores midway through training, only using about half of its available advice compared with the remaining two methods. Interestingly, its performance of the variance network drops back below that of the baseline before recovering at the final episode. Table 3.10 shows that the

ensemble uses less advice to get to comparable performance as the variance network. The average uncertainty of the variance network varies widely across training, while dropout and the ensemble remain roughly stable as shown in Figure 3.13.

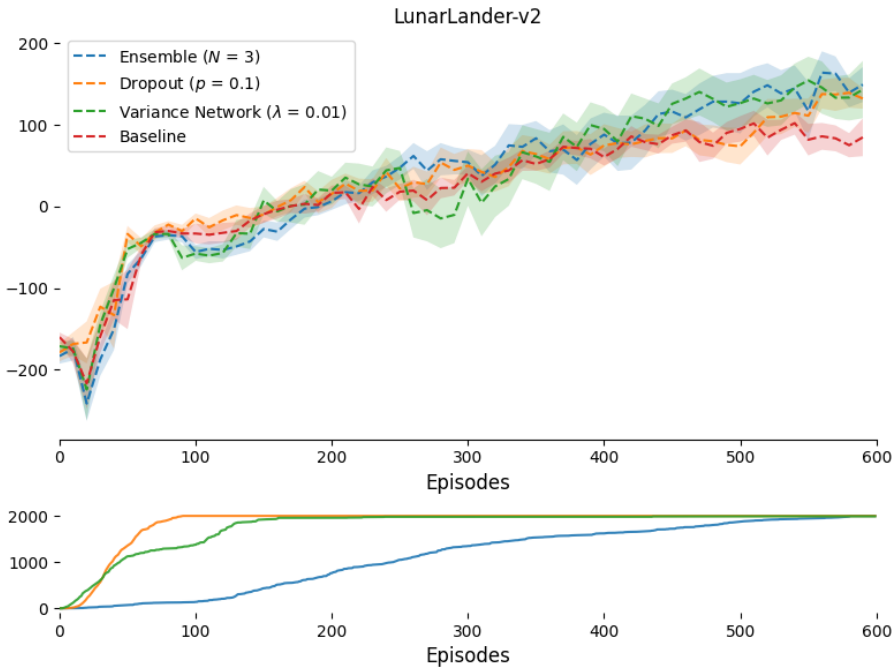


Figure 3.12: The performance of dropout ($p = 0.1$), bootstrapped ensemble ($N = 3$), and variance network ($\lambda = 0.01$) on Lunar Lander with an advice budget of 2000. All three methods perform similarly, with dropout showing slightly weaker performance at the end of training. Every 10 episodes, the average reward from 10 evaluation rounds averaged across 15 independent runs is reported. Shaded regions represent the standard error. The teacher used achieves a mean reward of 159.0 averaged over 50 episodes.

On the Lunar Lander task with a fixed query threshold, the variance network performs comparably to the ensemble, but the result is statistically non-significant. Both achieve higher scores than dropout.

Table 3.9: Mean reward and standard error during training averaged over 10 evaluations rounds and 15 seeds on Lunar Lander. Total area under the curve (AUC) is reported in the last column. The ensemble achieves higher scores than the variance network, but the results are not statistically significant. The best scores are reported in bold.

Method	Episode 150	Episode 300	Episode 600	Total AUC
Ensemble	-27 ± 10	$\mathbf{54 \pm 15}$	$\mathbf{150 \pm 21}$	$\mathbf{1992 \pm 801}$
Dropout	-10 ± 10	50 ± 20	132 ± 12	1858 ± 517
Variance Network	$\mathbf{8 \pm 16}$	34 ± 32	143 ± 34	1967 ± 950
Baseline	-9 ± 10	41 ± 9	87 ± 22	1135 ± 341

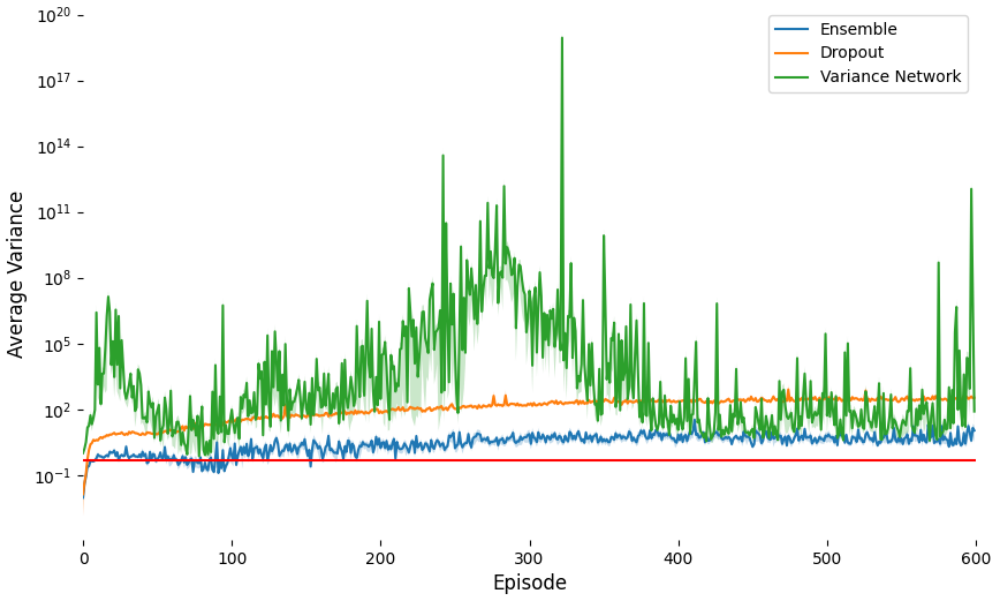


Figure 3.13: The uncertainty distributions from dropout ($p = 0.2$), bootstrapped ensemble ($N = 4$), and variance network ($\lambda = 0.01$) on Lunar Lander. The fixed query threshold of 16, q_F, above which advice is queried is shown in red. The average uncertainty per episode from all three methods range significantly in scale, but the variance network exhibits comparatively higher volatility.

Table 3.10: Total advice calls during training averaged over 10 evaluations rounds and 15 seeds on Lunar Lander. Lowest advice use is indicated by bold.

Method	Episode 150	Episode 300	Episode 600
Ensemble	$\mathbf{427 \pm 71}$	$\mathbf{1345 \pm 163}$	$\mathbf{2000 \pm 0}$
Dropout	2000 ± 0	2000 ± 0	2000 ± 0
Variance Network	1924 ± 42	1982 ± 18	1993 ± 8

In this section, three uncertainty methods on four environments are evaluated using a fixed query threshold. It is discovered that the uncertainty distributions from each method can vary widely in scale and behaviour, resulting in advice being queried at different intervals throughout training. The variance network achieves a statistically significant ($P \leq .001$) higher score on the Cartpole and Acrobot tasks than the ensemble, while performing similarly to the ensemble on the more challenging Mountain Car and Lunder Lander tasks. Dropout ranks last in terms of total area under the curve on all four tasks. One limitation of the fixed query threshold is that it has to be tuned specifically to the environment and potentially even to the method itself. The next section discusses results using a new kind of query threshold that does not require hyperparameter tuning.

3.3 Reward Adaptive Query Threshold

The previous section reported action advising performance across four different tasks using a fixed uncertainty threshold chosen specifically for each environment. As could be seen from Figures 3.3, 3.7, 3.10, and 3.13, the bootstrap ensemble, dropout and variance network produce drastically different uncertainty estimates during training. Not only does this make it harder to compare each method's performance, it introduces the need to tune the query threshold and uncertainty method hyperparameter to each method. This is a significant limitation of the RCMP framework.

To address this, we propose a new mechanism for action advising which queries for advice only when the current state's uncertainty is greater than a proportion, q_D, of the last s steps: Reward Adaptive Query Threshold (RAQT). Furthermore, q_D is dynamically incremented as a function of the total return from the last N episodes as described in Algorithm 1.

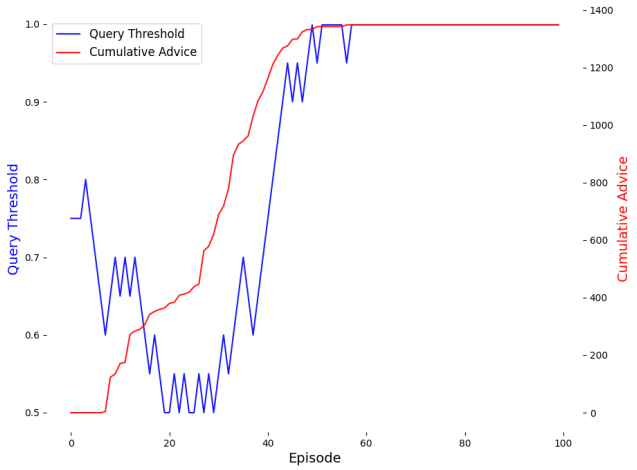


Figure 3.14: This plot depicts how the Reward Adaptive Query Threshold (RAQT) mechanism works to dynamically increase the query threshold as the agent learns. Using this reduces the number of advice calls by 33% on the Cartpole task with dropout (plotted). Blue shows the dynamic query threshold, q_D, and red shows the cumulative advice calls.

Figure 3.14 depicts how, for dropout on Cartpole, this mechanism ensures that as

the agent becomes more capable at solving a task (i.e., the reward goes up), the query threshold increases and the amount of advice requested stabilizes. This is particularly helpful at addressing the observation that in the case of the experiments on Cartpole in Section 3.2.1 where even after the agent solves the task, the uncertainty estimates can continue to increase in the case of dropout. Using RAQT, 33% less advice is used with dropout on Cartpole than the fixed query threshold as presented in Table 3.4.

Algorithm 1 Reward Adaptive Query Threshold

$B \leftarrow$ buffer of last N episodic returns
$U \leftarrow$ buffer of last s uncertainty estimates
procedure TUNER(B)
    Query threshold: $q_D \leftarrow 0.75$
    Increment factor: $f \leftarrow 0.05$
    Maximum query threshold: $q_{max} \leftarrow 0.999$
    Minimum query threshold: $q_{min} \leftarrow 0.5$
    if $B[N] - mean(B) < 0$ **then**
        if $q_D > q_{min}$ **then**
            $q_D \leftarrow q_D - f$ Reduce query threshold
        end if
    else
        if $q_D < q_{max}$ **then**
            $q_D \leftarrow q_D + f$ Increase query threshold
        end if
    end if
    return $min(q_D, q_{max})$
end procedure

procedure QUERY(U)
    $j \leftarrow 0$
    for i in U **do**
        if $i < U[s]$ **then**
            $j \leftarrow j + 1$
        end if
    end for
    if $j/length(U) >$ TUNER(B) **then**
        return $True$
    else
        return $False$
    end if
end procedure

As shown in in Figure 3.15, using RAQT allows the methods to query for advice without the need to choose a hyperparameter specific to the scale of its uncertainty estimates. In Table 3.11, we can see that the fixed and adaptive query thresholds perform similarly. The variance network on Cartpole still outperforms dropout and ensembles with less advice (Table 3.12). However, noting that the advice budget for the remaining environments is exhausted relatively quickly, in Appendix A.5 the performance using an advice budget of 8000 is displayed.

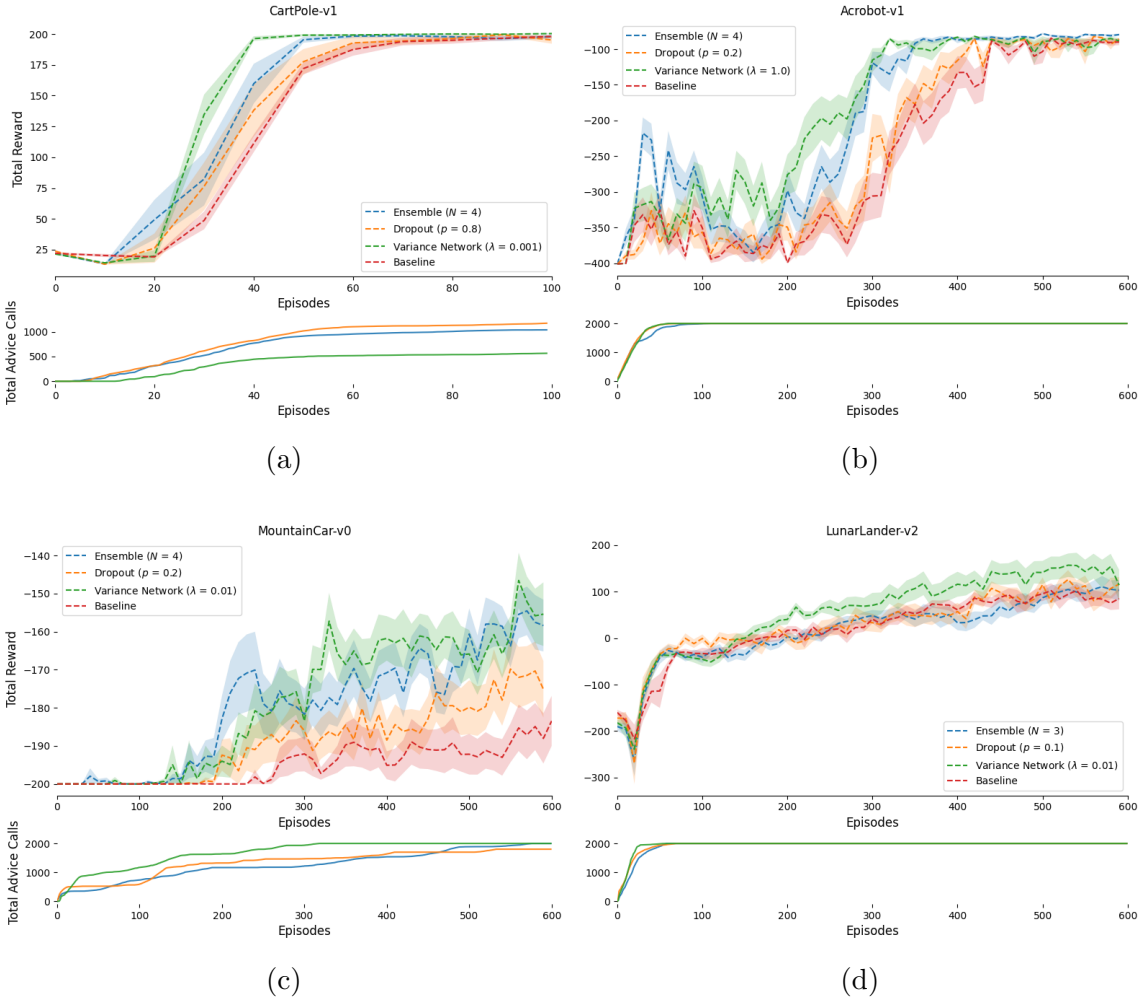


Figure 3.15: Results from RAQT strategy with a 2000 advice call budget.

Table 3.11: Total AUC for each method and environment using a fixed vs. adaptive query threshold. The highest scores are indicated by bold.

		Fixed	Adaptive
CartPole	Ensemble	1254 ± 33	$\mathbf{1313 \pm 50}$
	Dropout	1212 ± 19	$\mathbf{1238 \pm 54}$
	Variance Network	$\mathbf{1463 \pm 27}$	1385 ± 19
	Baseline	1166 ± 27	1166 ± 27
Acrobot	Ensemble	-12857 ± 389	$\mathbf{-11907 \pm 156}$
	Dropout	$\mathbf{-13959 \pm 257}$	-14414 ± 220
	Variance Network	$\mathbf{-11154 \pm 240}$	-11363 ± 266
	Baseline	-15176 ± 341	-15176 ± 341
MountainCar	Ensemble	-10818 ± 117	$\mathbf{-10796 \pm 191}$
	Dropout	-11506 ± 134	$\mathbf{-11338 \pm 203}$
	Variance Network	-10906 ± 108	$\mathbf{-10720 \pm 142}$
	Baseline	-11712 ± 125	-11712 ± 125
LunarLander	Ensemble	$\mathbf{1992 \pm 801}$	934 ± 472
	Dropout	$\mathbf{1858 \pm 517}$	1557 ± 537
	Variance Network	1967 ± 950	$\mathbf{3227 \pm 766}$
	Baseline	1135 ± 341	1135 ± 341

Table 3.12: Total advice calls to get to a given percent of the final reward by each method (budget is 2000). Lowest number of advice calls indicated by bold.

		CartPole	Acrobot	MountainCar	LunarLander
25%	Ensemble	501 ± 70	1412 ± 72	$\mathbf{1166 \pm 221}$	$\mathbf{1594 \pm 123}$
	Dropout	596 ± 55	2000 ± 0	1324 ± 171	1727 ± 102
	Variance Network	$\mathbf{279 \pm 23}$	$\mathbf{1124 \pm 34}$	1676 ± 156	1954 ± 47
50%	Ensemble	754 ± 84	$\mathbf{1412 \pm 72}$	$\mathbf{1166 \pm 221}$	$\mathbf{1889 \pm 64}$
	Dropout	812 ± 82	2000 ± 0	1464 ± 173	1934 ± 56
	Variance Network	$\mathbf{279 \pm 23}$	2000 ± 0	1785 ± 140	1985 ± 16
75%	Ensemble	754 ± 84	2000 ± 0	$\mathbf{1166 \pm 221}$	2000 ± 0
	Dropout	999 ± 95	2000 ± 0	1700 ± 149	2000 ± 0
	Variance Network	$\mathbf{430 \pm 45}$	2000 ± 0	1883 ± 82	2000 ± 0
100%	Ensemble	1105 ± 146	2000 ± 0	$\mathbf{1448 \pm 188}$	2000 ± 0
	Dropout	2000 ± 0	2000 ± 0	1803 ± 120	2000 ± 0
	Variance Network	$\mathbf{536 \pm 68}$	2000 ± 0	1971 ± 30	2000 ± 0

3.4 Discussion

In this chapter, we showed that while the variance network is able to outperform the ensemble and dropout on two of four tasks using a fixed query threshold, its advice efficiency is not always superior. We also discussed the limitation of needing to tune a fixed uncertainty threshold to each environment and method and showed that using an adaptive threshold instead results in comparable performance. The results presented in Figure 3.15 suggest that out of the three methods evaluated, dropout is consistently the inferior choice both in terms of advice efficiency and sample efficiency. Furthermore, as shown in Appendix A.2, dropout is also particularly susceptible to the choice of its hyperparameter. As the ensemble also requires more computational overhead (i.e., training N independent networks), the variance network can be advantageous compared to the two other methods evaluated.

Another observation from this chapter is that each method produces quite different

uncertainty estimates during training and across environments — both in terms of scale and behaviour. One possible explanation for the stark difference between the uncertainties (and, as such, the number of advice calls made) between the Cartpole and Lunar Lander task is the complexity of the state-action space. As Cartpole only has two actions and four state variables (compared with 4×8 for Lunar Lander), it could be that faster the agent converges to an optimal policy, the less of the state space it needs to spend time exploring (the intuition being that less explored parts of the state space will have higher uncertainties). However, this does not explain why the uncertainty estimates from dropout continues to increase even as the agent converges on Cartpole, nor why the uncertainty estimates on Mountain Car continue to diverge. As the ground truth uncertainty is not typically known, it is challenging to determine which method produces more *accurate* estimates. However, this motivates the use of *calibration* as a potential means to evaluate the quality of uncertainty estimates.

Chapter 4

Uncertainty Calibration

Recognizing that calibration is a tool that can give us a sense of the quality of an uncertainty estimate, this chapter first evaluates the three methods of this chapter in a simple regression setting. Then, we assess whether calibration can be informative for assessing the quality of uncertainty methods in the context of action advising.

Indeed, for many applications such as supply chain optimization, medical diagnostics and autonomous driving, accurate uncertainty estimates are crucial for ensuring the outputs of modern machine learning systems are reliable [41]. If the confidence interval or probability for a label predicted by a neural network does not reflect the ground truth likelihood, it will be unable to indicate when it's likely to be incorrect and is said to be mis-calibrated [42]. A well-calibrated uncertainty estimate means that a 90% confidence interval will contain the true outcome 90% of the time. Miscalibration can typically occur because of model bias: the predictor is not able to assign the correct probability to every confidence interval [37].

In the context of model-based reinforcement learning, it has been shown that using calibrated transition probabilities improves performance [22]. Whereas previous work has focused on the calibration performance in the stationary, i.i.d problem setting [35], this chapter looks at how calibrated the uncertainty estimates are for the Q-values themselves in action advising.

4.1 Simple linear regression

We begin by first demonstrating how each of the methods compare on a simple linear regression task. A dataset $\mathcal{D} = \{\mathbf{x_n}, y_n\}_{n=1}^{N}$ of size $N = 2000$ training examples is produced using the function $y = x+\epsilon(x)$, where the inputs x are drawn uniformly from the interval (-4, 4). Input-dependent noise is sampled from a Gaussian distribution $\epsilon(x) \sim \mathcal{N}(0, \sigma(x))$ where:

$$\sigma(x) = \begin{cases} 0.5, & x \in (-4.0, -1.5) \\ 1, & x \in (-1.5, 1.5) \\ 3, & x \in (1.5, 4.0) \end{cases} \tag{4.1}$$

As in the previous chapters, a neural network (f) is used to model the predictive distribution $p_\theta(y|\mathbf{x})$, where θ represents the neural network parameters. Over 400 epochs, the Adam optimizer with a step size of 0.001 is used to train the models consisting of two fully connected hidden layers of 64 ReLU units each. By using the mean and variance μ_θ, σ_θ obtained from each method, we can construct the distribution $F_n(x_n) \sim \mathcal{N}(\mu_\theta, \sigma_\theta)$.

In the regression setting, perfect calibration can be defined as:

$$\frac{\sum_{n=1}^{N} \mathbb{I}\{y_n \leq F_n^{-1}(p)\}}{N} = p \text{ for all } p \in [0, 1], \tag{4.2}$$

where $F_n^{-1}(p) = \inf\{y : p \leq F_t(y)\}$, $F_t(y)$ is the cumulative distribution function of f, and p is the expected confidence interval. In order to produce a plot that quantifies the calibration of a method's uncertainty estimates, we choose m confidence levels $0 \leq p_1 \leq \cdots \leq p_m \leq 1$. For each confidence level, we compute

$$\hat{p}_m = \frac{|\{y_n | F_n(y_n) \leq p_m, n = 1, \ldots, N|}{N} \tag{4.3}$$

to obtain the empirical frequency of points contained within the specified interval. The calibration error is defined as:

$$CE = \sum_{m=1}^{M} (p_m - \hat{p}_m)^2 \tag{4.4}$$

Figure 4.1 shows the results from plotting $\{(p_m, \hat{p}_m)\}_{m=1}^{M}$. It can be seen that the bootstrap ensemble has the worst calibration performance that $CE = 5.35$, followed by dropout with $CE = 2.71$. As the calibration curves for both of these methods are below the line of identity, they are said to be under-calibrated. That is, they consistently underestimate the true uncertainty of the dataset. However, the variance network achieves near perfect calibration ($CE = 0.003$). Given that $\sigma(x)$ is Gaussian noise, it is be expected that a variance network of sufficient capacity should fit the empirical variance.

4.2 Q-values

In the case of Q-value uncertainty, the story is quite different. As bootstrapping can be effective when the underlying distribution is unknown [18], it is perhaps not surprising that the ensemble has a low calibration error on some of the environments as shown in Figure 4.2 as compared with dropout and the variance network. The same procedure is used to compute the calibration curves as in Section 4.1, with one modification made to the dataset construction step. As the true labels are required to produce $\hat{p}_m$, every ten episodes during the evaluation round, we perform 100 rollouts from the start state s_0, compute the discounted sum of rewards and store them as $Q^{true}(s_0, a_0)$, where a_0 is the first action in the action space the environment.

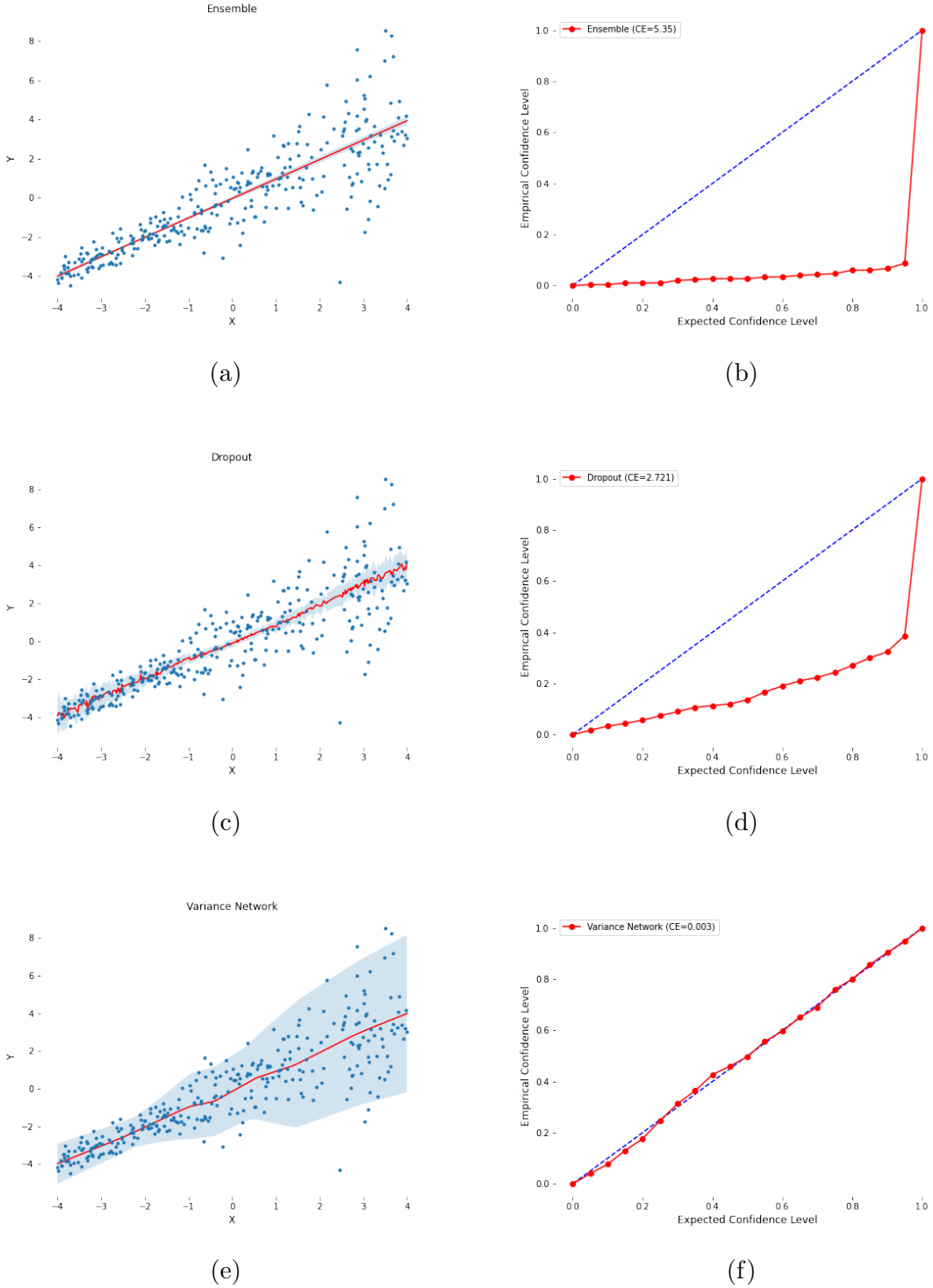


Figure 4.1: Calibration diagnostic plots for a simple linear regression task for the ensemble (b), dropout (d) and variance network (f). The variance network yields calibrated uncertainty estimates while the other two methods underestimate the true uncertainty. Perfect calibration is referenced by the blue dotted line.

It is interesting to point out that despite the variance network utilizing more advice (Figure A.5) – and having higher average uncertainty estimates than dropout and the ensemble – it still underestimates the true uncertainty on all tested environment except Cartpole. On Lunar Lander, fact that the ensemble has a $CE = 1.11$ whereas the variance network is $CE = 5.66$ suggests that calibrated uncertainty estimates may not be as important in the action advising setting.

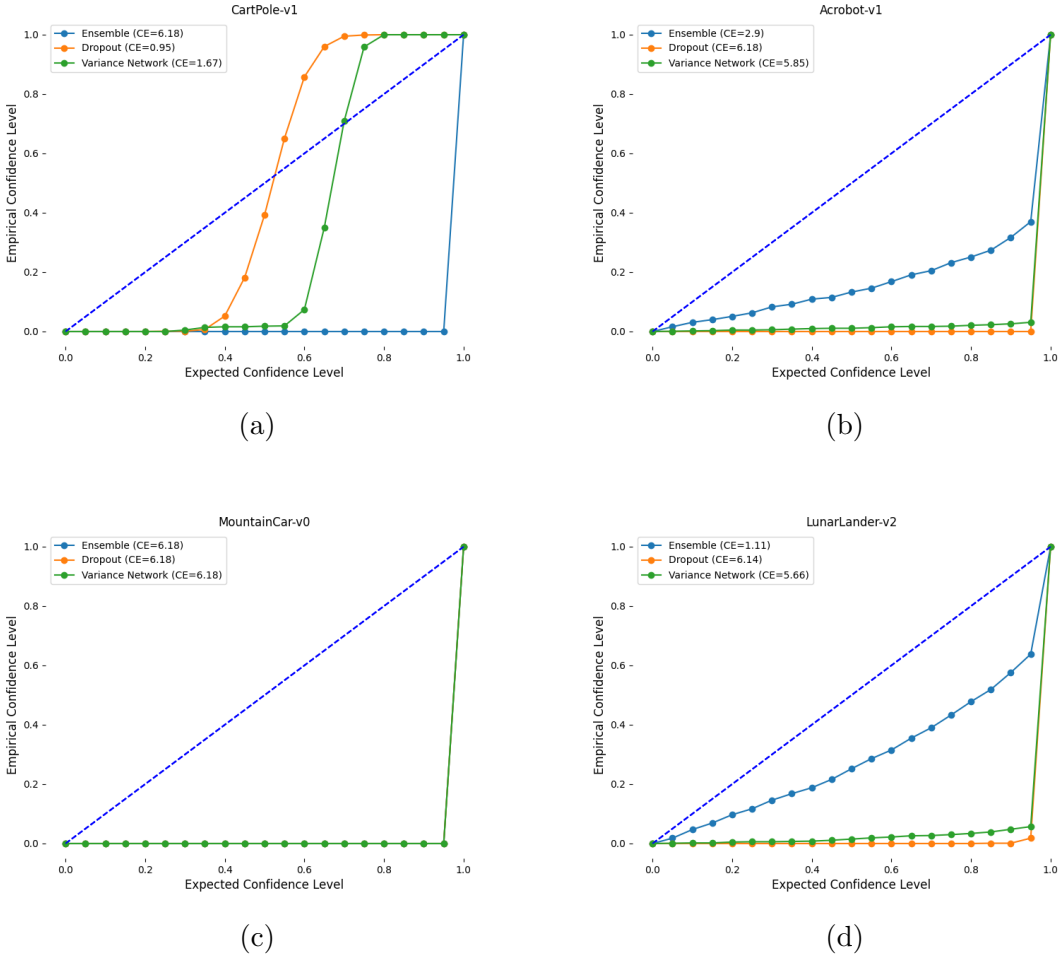


Figure 4.2: Calibration diagnostic plots obtained by comparing the expected and empirical frequency of observing Q^{true} within a given percentile interval. Plots are from last episode after convergence. None of the three methods produce well-calibrated uncertainty estimates of the Q-values, mostly underestimating the true uncertainty.

Chapter 5

Conclusion and Future Work

In this thesis, we explored how the uncertainty estimates from three different methods affect action-advising performance: bootstrap ensembles, Monte Carlo dropout and variance networks. We showed that the variance network, which captures information about aleatoric uncertainty, can outperform the other methods on two of the four discrete control environments investigated. Each method yields a different distribution of average uncertainty during training, which impacts where and when advice is queried. Observing this, we introduced a new kind of adaptive query threshold that dynamically increases the query threshold as a function of reward and argue this should be favoured in action advising over the fixed query thresholds used in literature as it does not require environment-dependent hyperparameter tuning. Finally, we conclude that none of the methods evaluated produce *calibrated* uncertainty estimates for Q-values.

There are many interesting future avenues for research:

- While we focused on discrete control environments, results with continuous control algorithms and on pixel-based environments warrant further investigation.
- We used variance as a proxy for uncertainty, but it could be interesting to contrast measures like coefficient of variation, entropy or approximations of value of information [43] [44].

- Instead of treating advice equally to the agent's own interaction data, using different advice buffer strategies (i.e. priority, weighting, etc.) could be explored.

- It is known that distributional RL [45] also yields value-based uncertainty estimates, but using these for action advising has not been explored. Furthermore, combining the three methods to capture both epistemic and aleatoric uncertainty could be useful.

- Integrating the availability of the teacher(s) within the state of the agent and devising methods to *learn when to ask for help* could be interesting – especially if the agent is exposed to the cost of requesting advice, or the budget, is available.

- Given that the variance network is calibrated in the stationary setting, calibrated state or reward model estimates could be obtained and tested as an alternative for action-advising based on Q-value uncertainty in complex domains. Especially as reward models are trained from human preferences, well-calibrated uncertainties are particularly important and with could be a particularly compelling area of research.

- A deeper study into the nature of uncertainty estimates produced by each of the methods discussed in this thesis could yield insights as to why each method produces different distributions.

Bibliography

[1] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018.

[2] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," Advances in Neural Information Processing Systems, 2017.

[3] M. E. Taylor, H. B. Suay, and S. Chernova, "Integrating reinforcement learning with human demonstrations of varying ability," International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[4] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: A research direction," vol. abs/1811.07871, ArXiv, 2018.

[5] S. Reddy, A. D. Dragan, S. Levine, S. Legg, and J. Leike, "Learning human objectives by evaluating hypothetical behavior," Internatoinal Conference on Machine Learning, 2020.

[6] L. Torrey and M. Taylor, "Teaching on a budget: Agents advising agents in reinforcement learning," International Conference on Autonomous Agents and Multiagent Systems, 2013.

[7] E. Ilhan, J. Gow, and D. P. Liebana, "Action advising with advice imitation in deep reinforcement learning," International Conference on Autonomous Agents and Multiagent Systems, 2021.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," vol. 550, pp. 354–359, 2017.

[9] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. W. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," vol. abs/1912.06680, 2019.

[10] A. R. Mahmood, D. Korenkevych, B. Komer, and J. Bergstra, "Setting up a reinforcement learning task with a real-world robot," International Conference on Intelligent Robots and Systems, 2018.

[11] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, and et al., "Magnetic control of tokamak plasmas through deep reinforcement learning," 2022.

[12] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous navigation of stratospheric balloons using reinforcement learning.," Nature, 2020.

[13] S.-A. Chen, V. Tangkaratt, H.-T. Lin, and M. Sugiyama, "Active deep q-learning with demonstration," Springer Maching Learning, 2020.

[14] F. L. Da Silva, P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "Uncertainty-aware action advising for deep reinforcement learning agents," Association for the Advancement of Artificial Intelligence, 2020.

[15] Z. Kenton, A. Filos, O. Evans, and Y. Gal, "Generalizing from a few environments in safety-critical reinforcement learning," vol. abs/1811.07871, ArXiv, 2019.

[16] B. Lütjens, M. Everett, and J. P. How, "Safe reinforcement learning with model uncertainty estimates," International Conference on Robotics and Automation, 2019.

[17] Z. Abbas, S. Sokota, E. J. Talvitie, and M. White, "Selective dyna-style planning under limited model capacity," International Conference on Machine Learning, 2020.

[18] M. White and A. M. White, "Interval estimation for reinforcement-learning algorithms in continuous-state domains," Conference on Neural Information Processing Systems, 2010.

[19] V. Mai, K. Mani, and L. Paull, "Sample efficient deep reinforcement learning via uncertainty estimation," International Conference on Learning Representations, 2022.

[20] W. R. Clements, B.-M. Robaglia, B. van Delft, R. B. Slaoui, and S. Toth, "Estimating risk and uncertainty in deep reinforcement learning," vol. abs/1905.09638, ArXiv, 2019.

[21] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, "Deep exploration via bootstrapped dqn," Conference on Neural Information Processing Systems, 2016.

[22] A. Malik, V. Kuleshov, J. Song, D. Nemer, H. Seymour, and S. Ermon, "Calibrated model-based deep reinforcement learning," vol. abs/1906.08312, ArXiv, 2019.

[23] R. Bommasani, D. A. Hudson, and E. A. et al., "On the opportunities and risks of foundation models," vol. abs/2108.07258, ArXiv, 2021.

[24] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. J. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano, "Learning to summarize from human feedback," vol. abs/2009.01325, ArXiv, 2020.

[25] C. J. C. H. Watkins and P. Dayan, "Q-learning," Springer Machine Learning, 2004.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, 2015.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," vol. abs/1412.6980, ArXiv, 2015.

[28] M. Hessel, J. Modayil, H. V. Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," Association for the Advancement of Artificial Intelligence, 2018.

[29] J. Clouse, "On integrating apprentice learning and reinforcement learning," University of Massachusetts, 1997.

[30] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," vol. abs/1810.12894, ArXiv, 2019.

[31] E. Ilhan and D. P. Liebana, "Student-initiated action advising via advice novelty," vol. abs/2010.00381, ArXiv, 2021.

[32] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," 2014.

[33] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," Conference on Neural Information Processing Systems, 2017.

[34] B. Efron and R. Tibshirani, "An introduction to the bootstrap," 1993.

[35] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," Conference on Neural Information Processing Systems, 2017.

[36] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," ser. Proceedings of Machine Learning Research, International Conference on Machine Learning, 2016.

[37] V. Kuleshov, N. Fenner, and S. Ermon, "Accurate uncertainties for deep learning using calibrated regression," International Conference on Machine Learning, 2018.

[38] A. Sedlmeier, T. Gabor, T. Phan, L. Belzner, and C. Linnhoff-Popien, "Uncertainty-based out-of-distribution detection in deep reinforcement learning," *ArXiv*, vol. abs/1901.02219, 2019.

[39] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," ArXiv, 2016.

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," International Conference on Computer Vision, 2015.

[41] J. Z. Liu, J. W. Paisley, M. Kioumourtzoglou, and B. A. Coull, "Accurate uncertainty estimation and decomposition in ensemble learning," vol. abs/1911.04061, arXiv, 2019.

[42] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," vol. abs/1706.04599, International Conference on Machine Learning, 2017.

[43] D. Arumugam and B. V. Roy, "The value of information when deciding what to learn," NeurIPS, 2021.

[44] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," Association for the Advancement of Artificial Intelligence, 2008.

[45] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," ICML, 2017.

Appendix A:

A.1 Implementation Details

In this section, the implementation of the methods used in this thesis are summarized.

Q-Network architecture: DQN [26] is the base architecture for all experiments. The architecture consists of two ReLU-activated hidden layers with 256 units each to estimate the action value function. The number of input units for the network is the dimension of an environment's state-space, and the output units are the number of actions available to the agent.

Training procedure: On every environment step, a batch of 64 samples is drawn from a replay buffer and used to update the parameters of the Q-network with the mean squared error (MSE) loss and Adam optimizer [27]. The target network, Q_t, is updated every 5 episodes. The exploration rate, ϵ, is initially set to 1.0 and decayed linearly to 0.01 after each episode as referenced along with the other hyperparameters in Table A.1.

Ensemble: The uncertainty estimates for the bootstrapped ensemble are obtained by first randomly initializing a set of N independent networks (each with its own target network). The ensemble is trained by randomly sampling a new batch, with replacement, for each network from the replay buffer. The average variance across the network outputs is used to estimate uncertainty.

Monte Carlo dropout: Dropout with probability p is applied to the final layer of a separate Q-network trained in parallel with the primary Q-network used for acting.

The average variance across 10 forward passes is calculated and used as the proxy for uncertainty.

Variance network: The variance network is implemented by using a single network with two heads which learn both the mean and variance of using the MSE and negative log-likelihood loss controlled by the parameter λ [33]:

$$L^{VN}(\theta) = \mathbb{E}_{(s,a,s',r)\backsim\mathcal{D}}\left[\left(y - \mu_{Q_\theta}(s,a)\right)^2 + \lambda\frac{\left(y - \mu_{Q_\theta}(s,a)\right)^2}{\sigma^2_{Q_\theta}(s,a)} + \ln\sigma^2_{Q_\theta}(s,a)\right],$$

Table A.1: Hyperparameters used for the base DQN algorithm.

Hyperparameter	**Value**
Discount Factor (γ)	0.995
Epsilon Decay Factor	0.05
Replay Buffer Size	10K
Hidden Units	256
Batch Size	64
Soft Update Frequency	Every environment step
Hard Update Frequency	Every 5 episodes
Optimizer	Adam
Learning Rate (∇)	0.0003

Table A.2: Hyperparameters tested for each method, environment version and query thresholds. The hyperparameter used for a given figure is included in the figure legend. All environments are run with OpenAI Gym v0.21.0 [39].

Hyperparameter	**Value**
Dropout Probability (p)	0.1, 0.2, 0.4, 0.8
Ensemble Size (N)	2, 3, 4, 5
Loss Attenuation Factor (λ)	0.001, 0.01, 0.1, 1.0
CartPole-v1 (q_F)	0.3, 0.5, 0.7, 0.9, 1.2, 1.5
Acrobot-v1 (q_F)	0.02, 0.05, 0.1, 0.5, 0.8, 1.0
MountainCar-v0 (q_F)	0.2, 0.5, 0.7, 1, 2, 3
LunarLander-v2 (q_F)	1, 2, 4, 8, 16, 32

A.2 Hyperparameter Sweeps for Fixed Query Threshold

This section presents results from the parameter sweeps performed to select the uncertainty method hyperparameter for the fixed uncertainty threshold experiments in Chapter 3.

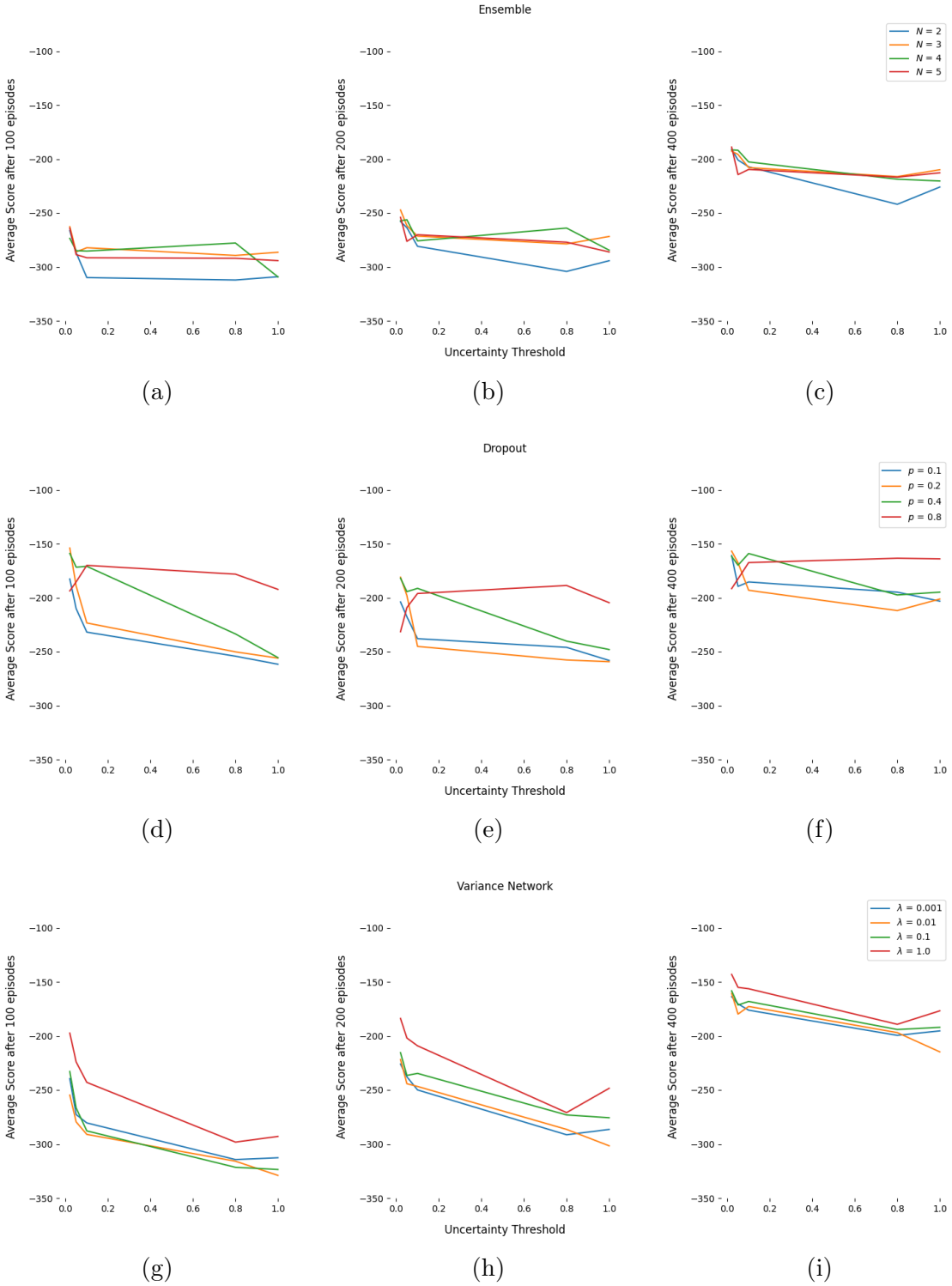


Figure A.1: Sensitivity of uncertainty hyperparameter across various fixed advice thresholds on Acrobot.

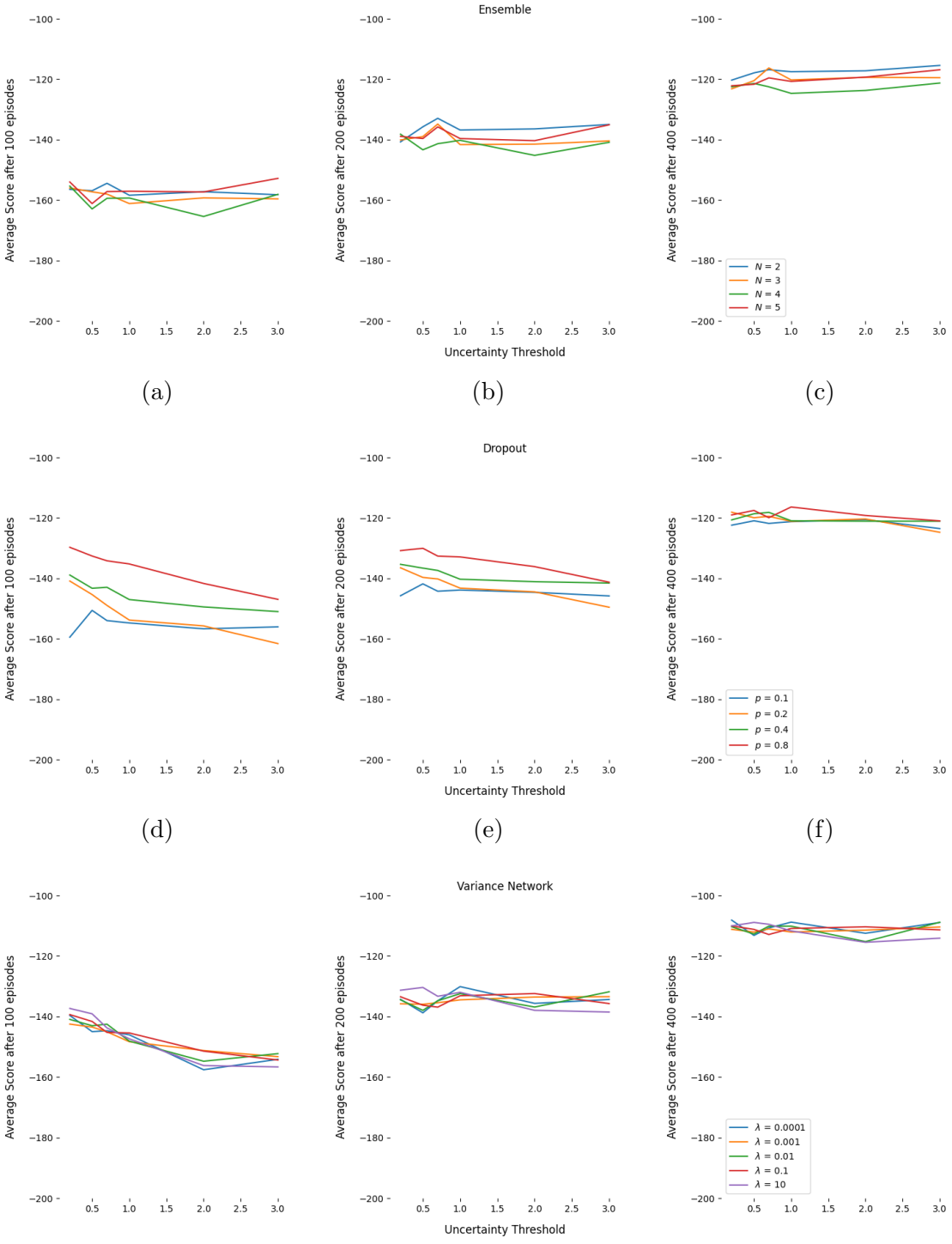
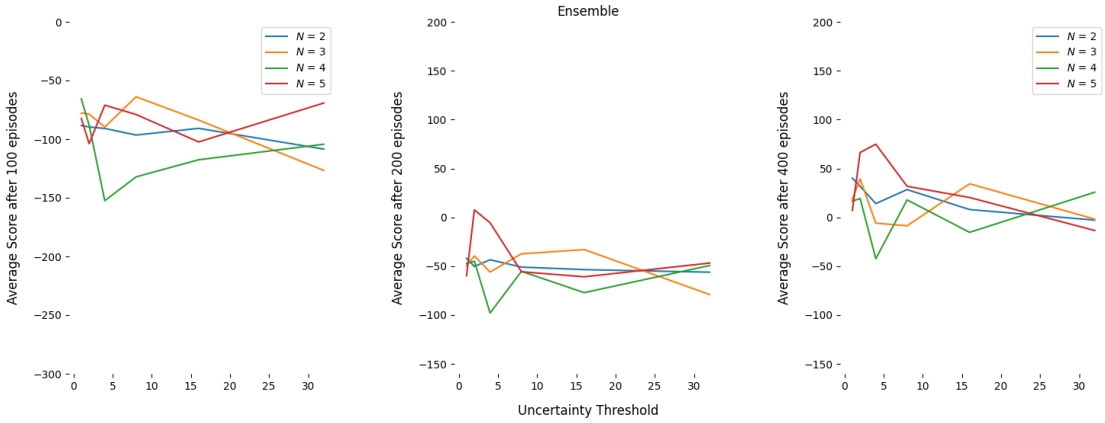
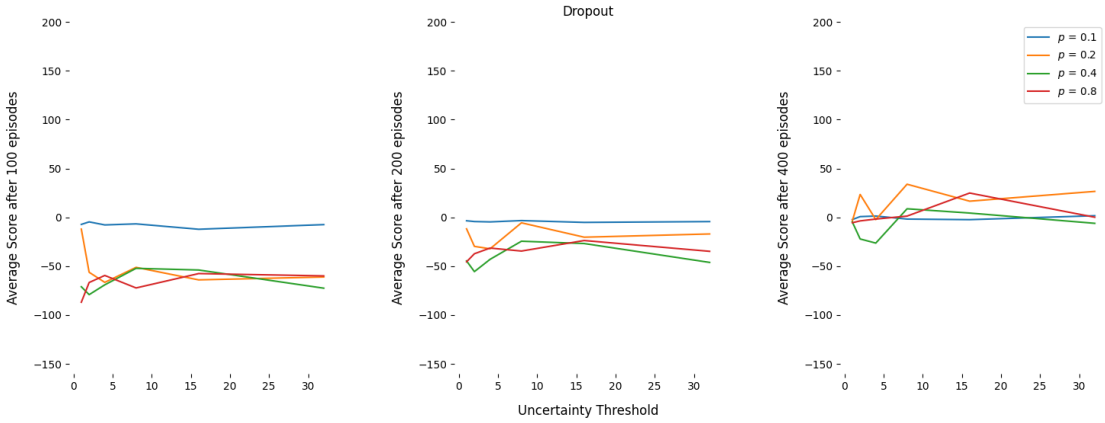


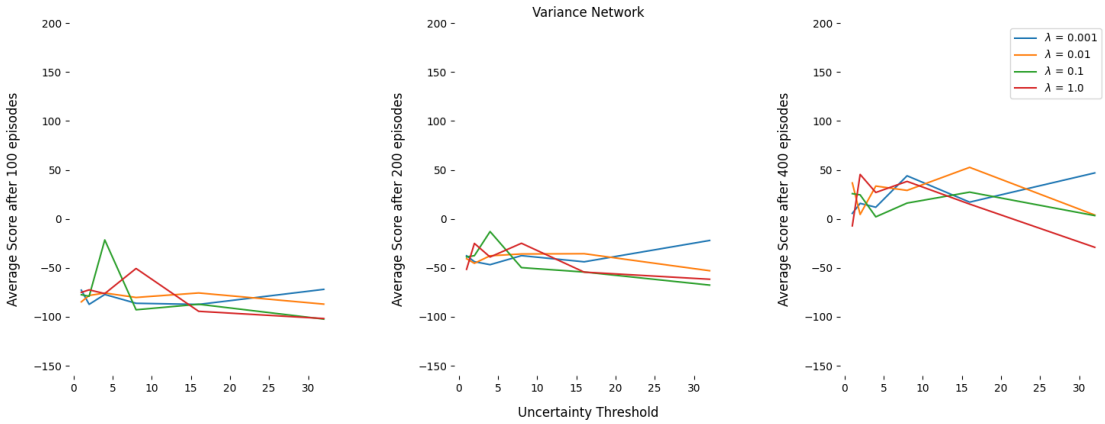
Figure A.2: Sensitivity of uncertainty hyperparameter across various fixed advice thresholds on Mountain Car.



(a) (b) (c)



(d) (e) (f)



(g) (h) (i)

Figure A.3: Sensitivity of uncertainty hyperparameter across various fixed advice thresholds on Lunar Lander.

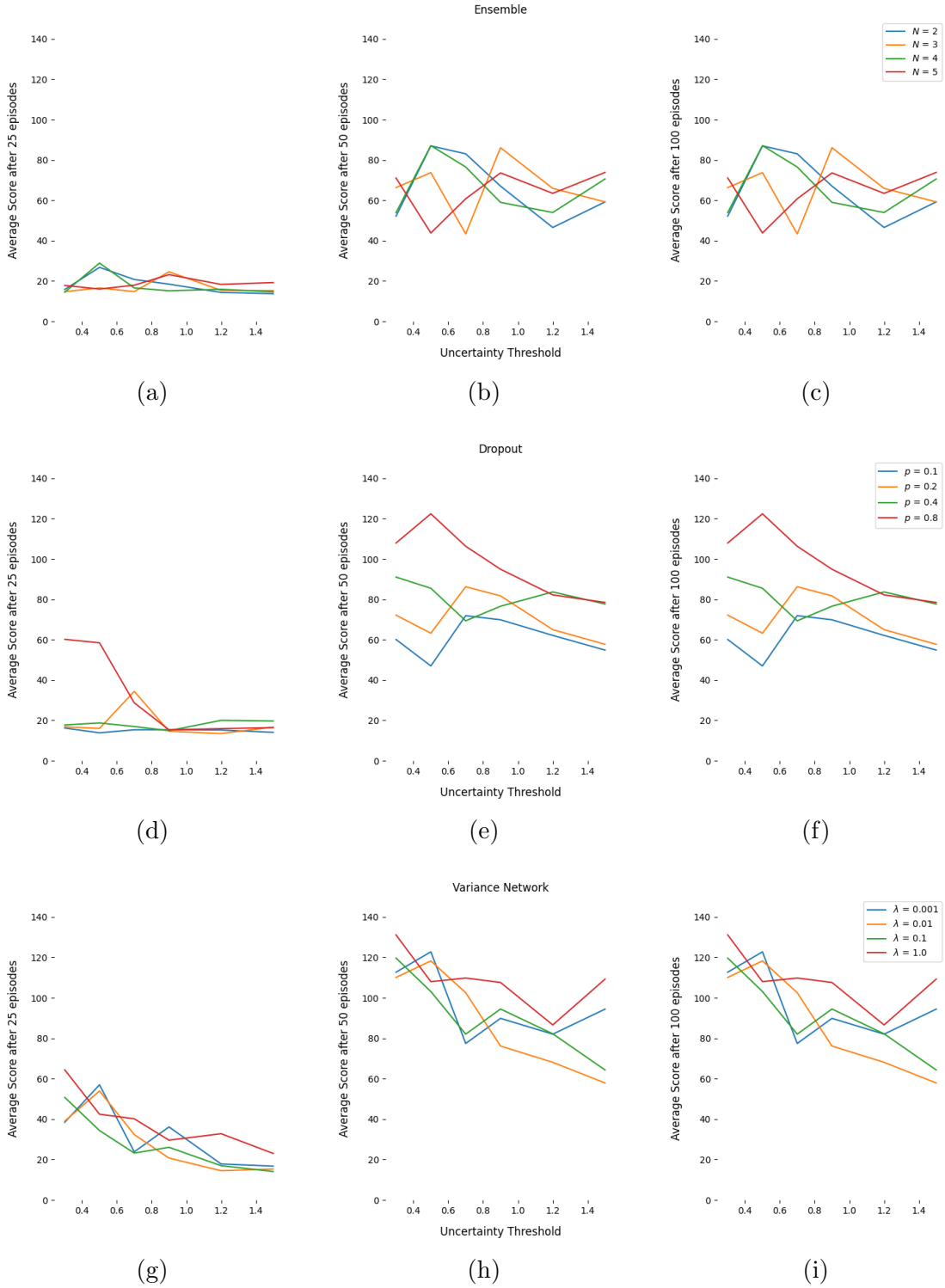


Figure A.4: Sensitivity of uncertainty hyperparameter across various fixed advice thresholds on Cartpole.

A.3 Adaptive Query Threshold with 8000 Advice Budget

The methods discussed in Chapter 3 often exhaust the advice budget. For reference, the evaluation curves using an advice budget of 8000 is shown for the reward adaptive query threshold.

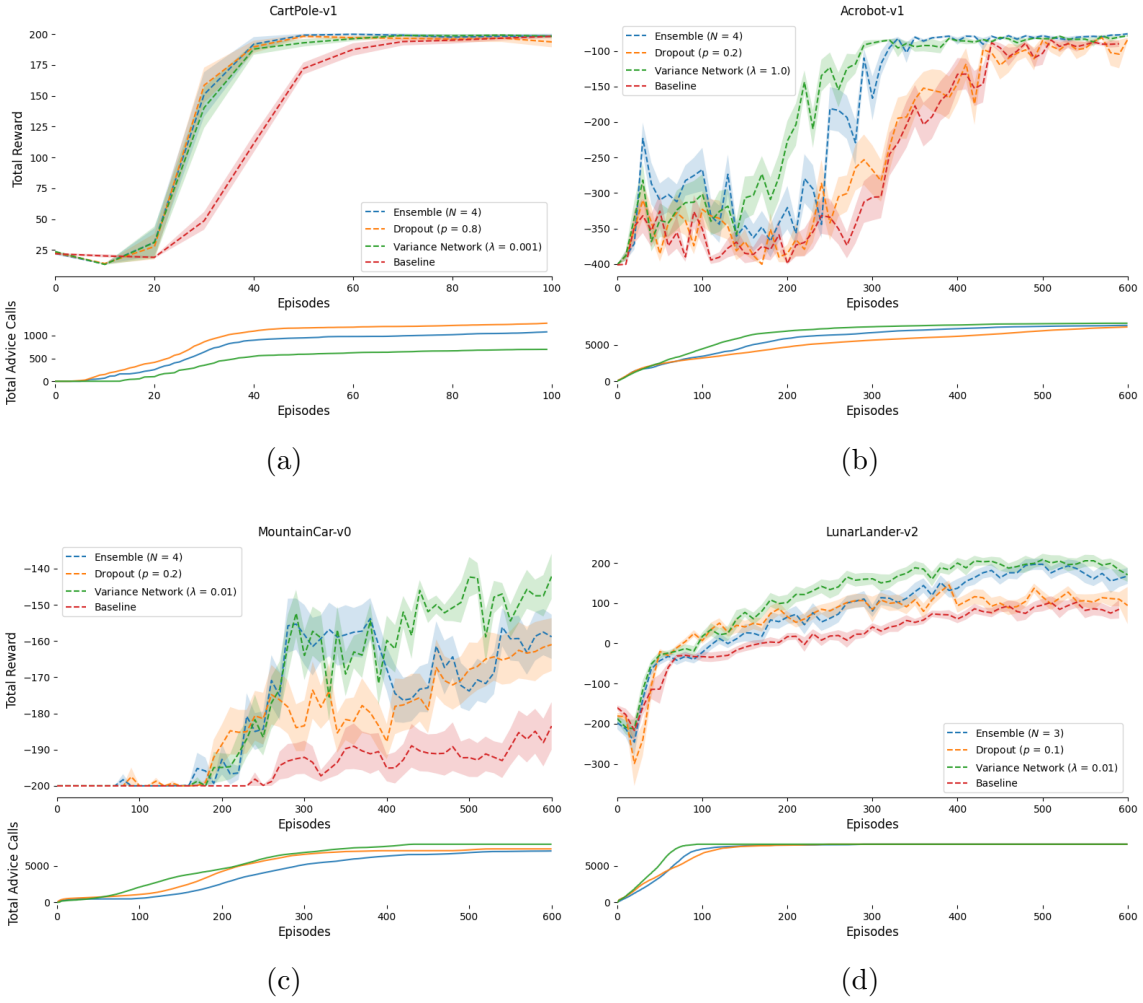


Figure A.5: Results from reward adaptive query threshold (RAQT) strategy with an advice call budget of 8000.