Human-in-the-loop Shared Control

Peter Du peterdu2
Aamir Hasan aamirh2
Abhi Kamboj akamboj2

1. Introduction

Learning based approaches for control and decision making have rapidly made their way into various domains as advances in data collection, compute, and algorithmic breakthroughs have allowed machines to perform at levels comparable to (or better) than human experts. Despite these advances however, there are still domains where a fully autonomous agent with no human supervision or backup faces hurdles from both a regulatory and public trust perspective. In particular, safety critical applications of autonomy without human oversight remains a contested approach. The most prominent example of this can be seen in the autonomous driving domain where companies continue to train and require "safety drivers" to monitor and be ready to take over vehicle control at any given moment. Even in fully autonomous products such as Waymo One's autonomous taxi service in Phoenix, safety drivers are often used during inclement weather or dispatched to take control when the vehicle senses difficulty.

The goal of our project is to look at different ways of characterising uncertainty in an autonomous policy and use them to develop control that is shared between the human and the autonomous agent. To this end, we look at how this information can be obtained from reinforcement learning (RL) based policies and imitation learning based policies. In the RL policies, we first consider a Deep Q-Learning (DQN) policy without modification and use the Q-values provided by the network. We then retrain the policy using Bayesian Deep Learning to more explicitly characterise uncertainty. We then directly try to learn a Bayesian Neural Network policy through behaviour cloning. During runtime, the policies are preempted when various uncertainty thresholds are reached and a human is asked to provide the next action. We compare the performance of these shared policies among each other as well as a fully autonomous policy to see which methods of requesting human input has the best overall performance.

2. Approach

Our original proposal highlighted the following steps for the project:

1. Setup environments on OpenAI Gym
2. Build baseline models for all environments
3. Build Bayesian models for all environments
4. Compare performance of all models for shared control.

We used the variance in Q-values as our baseline approach and discuss it in Sec. 2.2. We explored two Bayesian models, namely, Bayesian Deep Q-Learning and Behavior Cloning with Bayesian DNN, which are discussed in Sec. 2.3 and Sec. 2.4 respectively.

Unfortunately, during the course of the project, our approach changed to counter unexpected subtleties in the environments and models. The changes in the planned approach are discussed below.

2.1. Environments

We planned on testing our models on OpenAI environments of varying complexities, namely: Breakout and Highway-env [3, 2]. However, as we ran experiments on the environments we found that, even for the baseline methods, achieving good performance was incredibly hard. Hence, as we were forced to lower the complexity of our environments, we decided to begin testing on the discrete space LunarLander environment.

Lunar Lander The LunarLander-v2 environment, shown in Figure 1, is a discrete space, discrete action environment. In this environment the agent is tasked with landing a Lander at a designated (constant) area on the surface of the moon, given control of thrusters located at the left, right, and bottom of the lander. The state of the environment at any time is a 8 dimensional vector and the action the agent can perform is a 4 dimensional vector.

Breakout The Breakout-ram-v0 environment, shown in Figure 2, is also a discrete space, discrete action environment. The environment emulates the popular Atari game, Breakout, where the objective of the agent is to clear all the colored areas on the screen by controlling a paddle, with the ability to fire 5 balls. The state of the environment at any time is a 128 dimensional vector and the action the agent can perform is a 4 dimensional vector.

As discussed in Sec. 4, we were unable to successfully achieve a good performance on the baseline for the Breakout environment and have hence omitted the discussion of the models' performance on it. Similarly, we tested the more complex HighwayEnv and were also unable to achieve sufficient performance results.

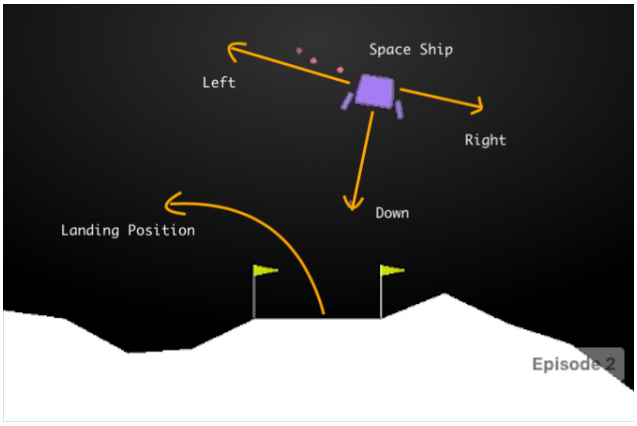

Figure 1. The Lunar Lander environment [4]

Figure 2. The Breakout environment [3]

2.2. Vanilla Deep Q-Learning

Our first and baseline approach uses a Deep Q-Learning network trained using the standard approach for deep Q-learning. The q-value network is trained on the Gym environment using a replay buffer and target network with the following parameters:

- Number of updates: 5000
- Rollout length: 500
- Replay buffer size: 100,000
- Target copy interval: 50 updates
- Optimizer: Adam w/ learning rate 0.005
- Epsilon decay: $\epsilon_{init} = 0.9$, decay = 0.9995, $\epsilon_{final} = 0.1$

Once trained, the network is ran on the environment. However, instead of directly querying the optimal action at each timestep, we first get the predicted Q-value corresponding to each observation. The variance of Q-values are calculated and used to determine if the state should elicit human control. Here we make a slight modification to the notion of "critical states" that were originally proposed in [1]. In that work, critical states are defined by:

$$S_{crit} = \{x | \big(\max_a Q(s,a) - \frac{1}{A} \sum_a Q(s,a) \big) > l\}$$

where A is the size of the action space and l is a predefined constant. In other words, the set of states where the user should intervene are ones where the policy is suggesting that a particular action has significantly higher expected future reward than the average of other actions.

In our case, we take the variance:

$$V_x = \frac{\sum_a (Q(s,a) - \bar{Q})^2}{A}$$

and approximate uncertainty when the variance is below a predefined threshold. The main difference is that here, we attempt to find a proxy for policy uncertainty versus state criticality. When the variance among actions is low, we prompt the user to provide the next action under the assumption that the policy lacks confidence of the following action. Of course this does introduce the scenario where the expected future return is simply similar for all actions. Under these scenarios, we don't expect reduced performance as the human should still provide the best action, however we do expect increased human intervention requested by the shared control scheme.

The approach of using vanilla Deep Q-Learning acts as a baseline and "first effort" attempt at a shared control policy as it does not require modification to existing policies. As a result, the proxy for uncertainty is not ideal. In the following cases, we attempt to approximate uncertainty of the policy more directly.

2.3. Bayesian Deep Q-Learning

Our next approach stays within the realm of reinforcement learning policies and Deep Q-Learning. However, we now replace the vanilla Deep Q-Learning deterministic network with a Bayesian neural network. The network input and output dimensions remain the same as before. At a high level, the use of Bayesian layers allows us to more directly characterise uncertainty in the network predictions by sampling a distribution of outputs for the state. Here, the variance of the distribution can be used to determine the uncertainty associated with the output.

However, the use of Bayesian layers required significantly more tuning to learn a reasonably good agent policy. We initially tried to use the Bayesian neural network for both target and model networks and train using the standard Deep Q-Learning approach. The model network was used to explore the action space and populate the replay buffer while the target network was used to estimate the expected future return using reward signals collected from the environment. As the Bayesian policy now outputs a distribution of Q-values, we took a sample of n outputs at each step and used the mean of the samples as the predicted value. However, with this approach, we noticed poor training behaviour and significant training overhead, primarily caused by the need to sample at each step. If the number of samples n was kept small, the resulting network has faster training iterations but poor approximations of the actual networks output. While the opposite resulted in extremely slow training iterations. As a result, we replaced the target network with a deterministic Q-value network which did not require sampling. The Bayesian model was still used to explore the state space. The network is shown in Figure 3.

Other training parameters are given as follows:

- Number of updates: 10000
- Rollout length: 500
- Replay buffer size: 100,000
- Optimizer: Adam w/ learning rate 0.001
- Epsilon decay: $\epsilon_{init} = 0.9$, decay=0.9996, $\epsilon_{final} = 0.2$
- Samples taken for loss computation: 5

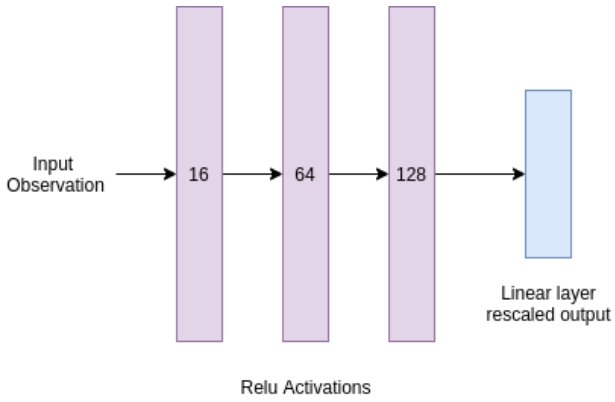

Figure 3. Bayesian neural network architectures

During runtime, the Bayesian Deep Q-Learning policy generates Q-value distributions for each state or observation. At each step, we sample $N = 50$ outputs from the policy to approximate the distribution. The variance along each action dimension is calculated from the samples and the total variance is given by the sum of each action dimension variance. Similar to the prior vanilla Deep Q-Learning shared control, we set a predefined variance threshold l. However, now that we use the variance of the output distribution to directly approximate uncertainty, if the total variance *exceeds* l, then the user is prompted to give the next action.

2.4. Behaviour Cloning with Bayesian DNN

In contrast with the above policies, we now try to see if directly learning a decision making policy using Bayesian neural networks can yield better stability and performance when paired with shared human control. Unlike the Bayesian Deep Q-Learning approach, we no longer learn a model to output expected future rewards by training with reinforcement learning (i.e. letting the agent explore on its own). Instead, we first train a deterministic expert policy using RL and then use it to collect data for a supervised learning task.

As mentioned in class, purely using the expert to provide data samples may result in poor performance as the model is shown a limited set of scenarios and is likely to enter a state which it has not been trained on. To alleviate some of this issue, we use an epsilon greedy exploration approach where the training data rollouts are obtained by using actions from the model policy with epsilon probability. During dataset collection, this causes the agent to explore a greater number of "random" states which may not yield high reward. We note that by the nature of the shared control policy, it should be able to better handle this (lack-of) exploration problem. Unlike a standard DNN policy trained via behaviour cloning (BC), the Bayesian Neural Network policy should be able to encode the uncertainty of its outputs. The states where the

policy has seen little to no training data should yield higher variance in the output distribution, which are the very states a human will be asked to intervene.

As with Bayesian Deep Q-Learning, we found that training a Bayesian BC policy was also slightly tricky and requires more tuning that initially expected. The number of samples taken when calculating the loss for each training data point was made to be double that of when training Bayesian Deep Q-Learning. The overall network architecture is shared and shown in Figure 3. The setup of the shared controller was kept the same. During runtime, we sample 50 outputs from the policy at each timestep and use the samples to calculate the variance along each action dimension. The total variance is given by the sum along each dimension and the human is prompted for the next action if this exceeds a predefined threshold l.

- Number of updates: 5000
- Rollout length: 500
- Optimizer: Adam w/ learning rate 0.005
- Epsilon decay: $\epsilon_{init} = 0.9$, decay = 0.9995, $\epsilon_{final} = 0.2$
- Loss: Cross entropy loss
- Samples taken for loss computation: 10

3. Results

In this section we present the experimental results from using the various forms of uncertainty characterization in order to prompt human intervention. An important detail to note for these results regards the variation in the policies themselves. For example, we noticed that the use of Bayesian layers caused more instability during training and required more tuning to yield reasonable performance. This was something we did not expect to play as large a roll as it did during our initial proposals and updates. As a result, we will compare both the performance of each uncertainty characterization among each other, and the relative performance differences between each policy when operating with and without shared control. We hope that the latter will help to be more representative of the impact of the uncertainty metric on shared control without relying heavily upon the performance of the base policy.

3.1. Setup

In order to minimize human bias in the environments, we first allow the human to manually control the environments and practice achieving high reward at the task. This process took around 30 mins and once the human was familiar with the controls and behaviour of the environments, we had them get accustomed to the shared control scheme. In these practice runs, the controllers from the prior section were used, however, the human was prompted for input at random timesteps instead of the system using an uncertainty metric. The goal here was to get the human user to feel comfortable with providing input at a given notice. Once this was complete, we collected data from the full shared control scheme where the human was asked for input when an appropriate uncertainty threshold was reached. For each control scheme, we run two sets of 8 trials and record the rewards obtained. The first set uses only the trained policy while the second set uses shared control.

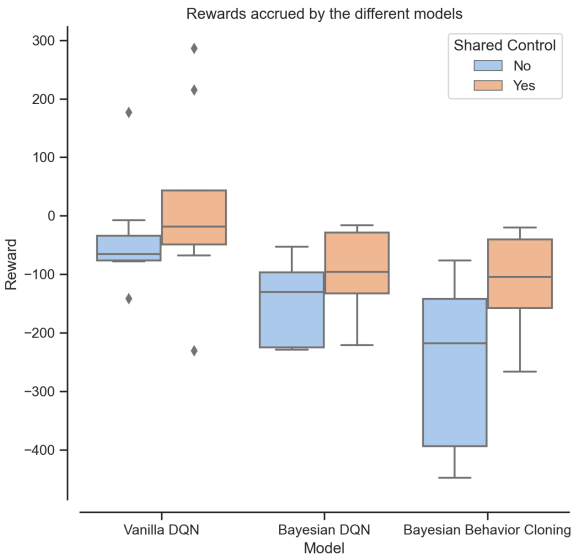

Figure 4. The rewards accrued by each model

3.2. Comparison of Policies with Shared Control

Figures 4 and 5 shown below illustrate the results of the experiments. Figure 4 shows a box plot of the rewards by each model with and without shared control. Figure 5 shows the number of interventions by each model. The results are further analyzed and discussed in section 4.2 and 4.3.

4. Discussion

4.1. Challenges with Testing Environments

As mentioned earlier in Sec. 2, we were unable to successfully train the baseline model on the Breakout environment. We tried training a DQN model on the environment while varying the hyperparameters such as learning rate, replay buffer size, epsilon decay, target network update fre-

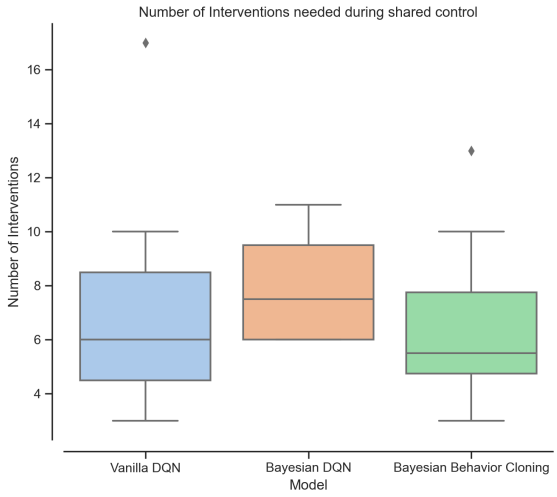

Figure 5. The number of interventions needed by each model during shared control

quency, Q network update batch size, number of updates, and the rollout length over a wide range of values. We also consulted several online blogs as well as Deepmind's original implementation and made changes such as counting the loss of a life (the need to fire multiple balls). Even after spending multiple weeks of compute with a powerful GPU on training, we were still unable to get the model to get decent reward on the environment. We suspect that the more complex the environments become, the fine-tuning required to train models to achieve good performance on them.

4.2. Rewards

Figure 4 show the rewards obtained by the various control schemes in each of the 8 trials in the lunar lander environment. One shared characteristic among the trained controllers is the tendency to land with higher velocity and a harsher impact. As a result, the controllers on their own tend to achieve a moderate negative overall reward (penalized by the harsh landing velocity). Instances where the controller fails occur when the lander is far away from the designated landing region or is unable to land in an upright position. These scenarios result in negative rewards of significantly larger magnitude.

When comparing absolute performance, we see that the vanilla DQN with Q-value variance results in the highest reward when using shared control. In these trials, there are also two instances where the system prompts the user for input near the end of the landing sequence and the user is able to reduce landing velocity significantly, resulting in high positive reward. However, we note that the performance of the vanilla DQN controller was also better than those of the Bayesian variety. When comparing relative performance increase, adding shared control to behaviour cloning with Bayesian NN saw the highest increase. Indeed, we see that Bayesian BC on its own has significant variance in its performance and far more "outright" failures than the other two approaches. When shared control is enabled, the policy has high uncertainty in regions where it has not explored sufficiently and the human is able to compensate in these scenarios.

4.3. Intervention Frequency

In Figure 5 we track the number of human interventions each scheme required during the trials. The number of interventions can significantly skew performance and usability depending on when and how often they occur. For example, a large number of rapidly occurring intervention requests will bias towards the human's role in the final performance. Across all three schemes, the number of prompts given are similar and averaged around 6-8 interventions required for each trial. A similar rate of prompts helps ensure the reward obtained using shared control with the varying policies are comparable and not being artificially inflated by trivially requiring the human to control most of the trials. The intervention frequency can be balanced and traded off with the overall reward by changing the uncertainty thresholds. In general, an increase in reward is seen when the threshold is decreased as the human is given more control.

5. Conclusion

Overall, our project attempts to determine the best way to develop shared control between an autonomous agent and a human. We investigated shared control by looking at different ways to characterize uncertainty in 3 different neural networks: Vanilla Deep Q-Learning, Bayesian Q-Learning, and Bayesian Behavior Cloning. We implement the shared control policy and test it in 8 user trials per policy, then compare the results with their corresponding baseline fully autonomous policies. The results show that Vanilla DQN has the best performance with the baseline and shared control policy when compared with Bayesian DQN and Bayesian BC. However, Bayesian BC has the largest improvement from its own baseline by implementing our shared control policy. Our study implicates that higher variance policies such as Bayesian BC can be significantly improved on with shared autonomy mechanism as presented in this paper. Overall, our study demonstrates how shared autonomy mechanisms can improve performance of an autonomous task, especially for high variance models. This improvement is important to consider when designing autonomous systems such as autonomous vehicles, mobile robots, industrial manipulators, or many other systems that may require human intervention.

6. Statement of Individual Contribution

All members contributed equally in all aspects of the project, which were, running and designing the experiments, writing and compiling the report, and preparing the in-class presentation.

References

[1] Sandy H. Huang, Kush Bhatia, Pieter Abbeel, and Anca D. Dragan. Establishing appropriate trust via critical states. *CoRR*, abs/1810.08174, 2018.

[2] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[3] OpenAI. A toolkit for developing and comparing reinforcement learning algorithms.

[4] Shiva Verma. Train your lunar-lander: Reinforcement learning, Oct 2021.