DISSERTATION

THE WISDOM OF THE CROWD: RELIABLE DEEP REINFORCEMENT LEARNING THROUGH ENSEMBLES OF Q-FUNCTIONS

Submitted by

Daniel L Elliott

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2018

Doctoral Committee:

Advisor: Charles W. Anderson

Bruce Draper
Michael Kirby
Edwin Chong

Copyright by Daniel L. Elliott 2018

All Rights Reserved

ABSTRACT

THE WISDOM OF THE CROWD: RELIABLE DEEP REINFORCEMENT LEARNING THROUGH ENSEMBLES OF Q-FUNCTIONS

Reinforcement learning agents learn by exploring the environment and then exploiting what they have learned. This frees the human trainers from having to know the preferred action or intrinsic value of each encountered state. The cost of this freedom is reinforcement learning can feel too slow and unstable during learning: exhibiting performance like that of a randomly initialized Q-function just a few parameter updates after solving the task. We explore the possibility that ensemble methods can remedy these shortcomings and do so by investigating a novel technique which harnesses the wisdom of the crowds by bagging Q-function approximator estimates.

Our results show that this proposed approach improves all tasks and reinforcement learning approaches attempted. We are able to demonstrate that this is a direct result of the increased stability of the action portion of the state-action-value function used by Q-learning to select actions and by policy gradient methods to train the policy. Recently developed methods attempt to solve these RL challenges at the cost of increasing the number of interactions with the environment by several orders of magnitude. On the other hand, the proposed approach has little downside for inclusion: it addresses RL challenges while reducing the number interactions with the environment.

TABLE OF CONTENTS

ABSTRACT ii
LIST OF FIGURES v

Chapter 1 Introduction 1
1.1 Motivating example 2
1.2 Recent advancements in reinforcement learning 3
1.3 The wisdom of crowds 5
1.4 Objectives 7
1.5 Document overview 8

Chapter 2 Background 10
2.1 Commonly-used neural network update methods with RL 10
2.2 Ensemble learning 16
2.3 Crowd ensembles in computer science 17
2.4 Q-learning 20
2.4.1 Q-learning using ANNs as function approximators 22
2.4.2 Actor-critic RL 23
2.4.3 Deep Q-learning 24
2.5 Deep deterministic policy gradients 27
2.6 Ensemble learning for RL 28
2.6.1 Recent ensemble-like approaches 32
2.7 Summary 34

Chapter 3 Baseline results 36
3.1 The cart-pole task 36
3.2 State of the art Q-learning solution for the low-dimension variant 38
3.3 Deep Q-learning on the cart-pole swing-up task 45
3.4 Biped walker task and the state of the art performance 54
3.5 Pendulum task 58
3.6 Experience replay 59
3.6.1 Comparison with and without experience replay 60
3.6.2 Indirect versus direct experience replay 62
3.7 DDPG actor and critic target functions 63

Chapter 4 The Crowd ensemble for Q-learning 65
4.1 Motivation 65
4.2 Crowd ensemble for discrete action Q-learning 66
4.3 Exploring crowd ensemble approach for the low-dimension cart-pole task . 69
4.3.1 A peek under the crowd ensemble hood 72
4.3.2 Q-function ensembles reduce decision instability 78
4.4 Crowd ensemble comparison with parallel Q-learning 83

4.5 Crowd ensemble for deep Q networks . . . 85
4.6 Crowd ensemble for biped walker task with actor-critic networks . . . 87
4.7 Crowd ensemble for pendulum task with actor-critic networks . . . 92

Chapter 5 Discussion . . . 94
5.1 The decision space: where the rubber meets the road . . . 94
5.1.1 Decision space volatility . . . 100
5.1.2 Ensemble decision spaces are less volatile . . . 101
5.1.3 Forgetting mostly occurs along the path to the goal . . . 107
5.2 A non-ensemble Q-function is relatively plain in the directions of the action dimensions . . . 112
5.3 Ensemble Q-functions lead to greater policy stability during training . . . 122
5.4 Shared layers: another opportunity for better collective decision-making . . . 124
5.4.1 Ensemble members are diverse . . . 125
5.4.2 Crowd ensemble prevents parameter update zig-zagging . . . 127

Chapter 6 Conclusions and future work . . . 131
6.1 Q-learning undervalues the roles of actions . . . 132
6.2 Specialization within an ensemble . . . 134
6.3 Can crowd ensembles solve tasks which are unsolvable by an individual? . . . 135
6.4 The wisdom of the crowd . . . 135
6.5 Shared layers . . . 137
6.6 Real world challenge: seed corn dryer balancing . . . 138
6.7 Are there other solutions to decision surface instability? . . . 140
6.8 Experience replay questions . . . 141
6.9 SCG versus ADAM . . . 143
6.10 Global actors *and* global critics? . . . 143

Bibliography . . . 145

LIST OF FIGURES

1.1 Five evaluations of the same agent on the cart-pole task where the agent state is represented as an image of the entire cart-pole track designed to simulate pointing a camera at the system we wish to control. Each evaluation is 2000 time steps in length. Zero position on the y-axis is the goal region and π and $-\pi$ are pointing down and are the same point in state space. 4

2.1 Example mixture of experts (MoE). Shows three or more experts, each an ANN with a single hidden layer. The mixture's output is a weighted combination of the experts' output. The weights come from the weighting network (shown as an ANN oriented perpendicularly to the experts). 18
2.2 Architecture of Q-learning. 21
2.3 Architecture of actor-critic training process. 23
2.4 Example deep Q-learning network with two convolutional layers and two fully-connected layers and three action outputs. Shown are two example input frames and two example features at each convolutional layer. The input frames and subsequent features were captured from our DQN agent during training. 25

3.1 GUI representation of the cart-pole task. 37
3.2 Three example solutions recorded while training a single agent on the low-dimension cartpole task. Each solution is described by six figures. The first four show the state positions during evaluation. The bottom two are the reward received at each time step, and the action selected at each time step. 41
3.3 Q-values of an example cart-pole task solution learned using Q-learning. Each plot shows the best Q-function value across 25×25 pole angle, a, (x-axes) and pole velocity, $\dot{a}$, (y-axes) points for a specific cart position, p, with zero cart velocity, $\dot{p}$. The grid of plots covers the range of cart positions (rows) and cart velocities (columns) where the cart position and velocity is held constant for each plot. 42
3.4 Decision surface of an example cart-pole task solution learned using Q-learning. Each plot shows the best action across 25×25 pole angle (x-axes) and pole velocity (y-axes). The grid of plots covers the range of cart positions (rows) and cart velocities (columns) where the cart position and velocity is held constant for each plot. There are three colors in these plots for the three possible actions: blue is $+1$, red is -1, and green is 0. 42
3.5 Center plots from Figures 3.4 and 3.3 where the Q-function is converted to an unfilled contour plot and overlayed on top of the action-selection plot. Red represents push left, green is no action, and blue is push right. 43
3.6 Baseline Q-learning results on the low-dimension cart-pole task. 44
3.7 Comparison of mean reward during evaluation for ADAM and SCG on 50 reruns of the low-dimension cart-pole task. 45
3.8 Two pre-processed frames which will combine to make a state for the DQN cart-pole agent. Frames are time t and $t + 1$ respectively. 48

3.9 Evaluated reward of 27 independent agents on the cart-pole swing-up task. In Figure 3.9a each agent's evaluated reward is plotted with a unique color. The dark line is the mean reward at each evaluation. In Figure 3.9b a box plot of the evaluated reward is drawn every 5×10^4 time steps The thin, yellow line running through the boxes is the median. 50
3.10 Two randomly-selected agents from Figure 3.9 plotted using red and cyan lines. The black line is the typical mean evaluated reward of the DQN approach. 51
3.11 Evaluations of two example solutions highlighting the difference in performance between a reward of 1200 and that of 1900. 51
3.12 Features of the first convolutional layer of a DQN agent on the high-dimension cart-pole task trained for 1×10^6 time steps. Features are drawn using color with red intensity representing increasingly positive values and blue intensity showing increasingly negative values. Each of the twenty features is labeled and the axis tick labels show feature size. An extra column of black pixels was added to divide the feature image according which input frame the mask is applied to. 52
3.13 Features computed by the convolutional layers of an example DQN. 53
3.14 DDPG architectures for the critic and actor ANNs. 56
3.15 Performance of our DDPG baseline implementation on the bipedal walker task. The line is the mean and the error bars are the standard error across 50 agents. 57
3.16 Frames captured from a successful agent on the bipedal walker task. 57
3.17 Graph of knee, hip, and hull state parameters during a single evaluation lasting 850 time steps. The upper-most plot is the hull angle and angular velocity. The second plot from the top is the joint positions of all four joints. The red and green lines are the two hip joints. The next two plots are the velocities of each joint. The bottom plot is the reward at each time step. 58
3.18 Results of a DDPG-trained agent on the pendulum task. The example Q-function and policy were captured at the end of training. 60
3.19 Comparison of DDPG with a large (10^6 time steps) and a small (1000 time steps) experience replay memory on the bipedal walker task. The small experience replay is an estimate for the effectiveness of this approach with no experience replay. 61
3.20 Comparison with and without experience replay with Q-learning on the low-dimension cart-pole task. 61
3.21 Comparison of direct and indirect experience replay. Indirect experience replay uses experiences generated while training a different agent. In these results Q-learning only very lightly negatively affected indirect experience replay while the DDPG agents' performance was dramatically reduced. 63
3.22 Comparison of DDPG on bipedal walker task with and without target networks. 64

4.1 Comparison of crowd ensemble for a variety of N_e values. Figures 4.1a and 4.1b show the mean, evaluated reward after the specified number of time steps. Figure 4.1c shows the median, evaluated rewards. 70
4.2 Randomly selected agents for $N_e = 1$, the single Q-learner, (top row) and $N_e = 50$. . . 71
4.3 Number of agents (out of 30) that have solved the low-dimension cart-pole task as training progresses. 72

4.4 Performance of ten runs of crowd ensemble with $N_e = 10$. The thin, black lines are the evaluated performance of the ensemble members (evaluated independently). The thick, red line is the performance of the ensemble as a whole. The thick, blue line is the mean performance of the ensemble members. 73
4.5 Mean performance ensemble members for four selected values of N_e. The red, vertical line is there for reference: it is the training length usually used for low-dimension cart-pole experiments in our experiments. 74
4.6 Mean performance of the eight $N_e = 50$ ensembles used to generate the corresponding line in Figure 4.5. Note that performance remains steady for all 1×10^6 time steps with one exception where one of the eight suffered from a moderate drop in evaluated reward. A line for $N_e = 1$ is included for reference. 75
4.7 Box plots of majority sizes during training for all selected N_e values. The red bars represents the median at each evaluation, the boxes represent the range of the lower and upper quartile values of the data, and the whiskers the entire range of the data excluding outliers. Outliers are represented with unfilled circles and are determined to be all samples lying outside two times the interquartile range. 76
4.8 Mean majority size for selected N_e values. How these values are computed is described in detail in the text. 77
4.9 Example, two-dimension decision surface of a single Q-learner across $10,000$ discrete locations for all possible cart positions and pole angles with cart and pole velocities set to zero. Blue is push right, red it push left, and green is no action. 78
4.10 View of two-dimensions of Q-function values learned a single Q-learner after 10^5 Q-learning trials. Image was made by sampling the Q-function across $10,000$ discrete locations for all possible cart positions and pole angles with cart and pole velocities set to zero. 79
4.11 View of two-dimensions of the count of decision surface changes for the final 10^4 Q-learning training samples (final 50 batches) for three randomly-selected agents. The x-axis is cart position and the y-axis is pole position. 81
4.12 Histogram of the count of decision surface changes for the final 10^4 Q-learning training samples (final 50 batches) for three randomly-selected agents. Here we see that the number of states with little to no change is higher for ensemble Q-functions and that the number of non-changing states increases as N_e increases. 82
4.13 Comparison of crowd ensemble, Fauber & Schwenker (parallel) approach, and single Q-learner. Showing all 3×10^5 time steps of the crowd ensemble training. 84
4.14 Comparison of crowd ensemble with historical experience replay, Fauber & Schwenker (parallel) approach, and single Q-learner. Showing all 5×10^6 time steps of the Fauber and Schwenker approach with $N_e = 20$. 85
4.15 Comparison of DQN results against the crowd ensemble DQN approach. 86
4.16 Comparison of DQN results from Figure 4.15 against the crowd ensemble DQN approach. Each line shows the number of times five selected agents solved ($R > 1800$) the task within the specified number of time steps. 87
4.17 Biped via DDPG crowd ensemble N_e comparison. The y-axis is the mean evaluation reward averaged over 50 agents. 90
4.18 Biped via DDPG crowd ensemble N_e comparison. The y-axis is the mean evaluation reward averaged over 50 agents. 91

4.19 Solution efficiency histograms for the non-ensemble approach and the first five N_e values. 92
4.20 Comparison of ensemble sizes against the non-ensemble baseline for the pendulum task. Results are the mean reward during evaluation episodes (every 25 episodes) over 30 agents. 93

5.1 Center plots from Figure 3.4 and 3.3 where the Q-function surface is converted to a unfilled contour plot and overlayed on top of the action-selection plot. Two of the four states are represented here: cart position and velocity are held at zero. 95
5.2 Frames of a movie showing an agent's movement through the decision space. Described in greater detail in the text, each frame is a single $(s_p(t), \dot{s}_p(t))$ value while s_a and $\dot{s}_a$ are allowed to change to make up each plot's x- and y-axes, respectively. The red dot is the current $s_a(t)$ and $\dot{s}_a(t)$ for that frame. The rows are labeled by the frame numbers. The light colors is push left, the dark is push right, and the gray is no action. 98
5.3 Each plot shows the Q-function difference across 25×25 pole angle (x-axis) and pole velocity (y-axis) points for a specific cart position with zero cart velocity. Starting in the upper-left corner and working across the rows, the plots correspond to cart position moving from -2.2 to 2.2. 99
5.4 Each plot shows the number of times the selected action changed in the 119 evaluations ($119,000$ time steps) after the agent solved the task for the first time. The most blue region is zero changes and the red is 84 changes out of a maximum possible 111 changes. Each plot is 25×31 pole angle (x-axes) and pole velocity (y-axes) pairs for a specific cart position with zero cart velocity. Starting in the upper-left corner and working across the rows, each plot shows a cart position moving from -2.2 to 2.2. . . 101
5.5 Plot of $\max_a Q(s,a) - \min_a Q(s,a)$ for a randomly selected single Q-learner ($N_e = 1$). We posit that this is a measure of decision certainty or stability. The states are the same as those shown in Figure 5.2 but only half the frames (every other) are included to save space. 102
5.6 Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected single Q-learner ($N_e = 1$). The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5. 102
5.7 Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected ensemble Q-learner ($N_e = 20$). Each ensemble member received a different training sequence. The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5. 103
5.8 Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected single Q-learner ($N_e = 20$). Each ensemble member was given the same (s,a) pair sequence with which to update which exacerbated the change count. The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5. 104

5.9 Bar plot of a raw count of the number of sampled state space locations which changed after a single application SCG with a data set generated by allowing the agent attempt to solve the task for 2000 time steps. Each bar is a different N_e value with the exception of the bar labeled "$N_e = 20$, shared" which supplied each ensemble member the same sequence of training data. Computed for the entire state space, not just the $(s_p, \dot{s}_p)$ shown in Figures 5.6, 5.7, and 5.8. Total of 1.2×10^6 state locations. 105
5.10 Plot of voting majority size as a percentage of the possible ensemble size ($N_e = 20$) for a randomly ensemble single Q-learner ($N_e = 20$). 105
5.11 Evaluated reward for all 150 parameter updates of the agent used to generate Figure 5.4. 107
5.12 Evaluation on the cart-pole task started in pole-up (leftmost plot, started at 0.1π and pushed left to give pole downward velocity) for an agent that has forgotten how to solve the swing-up task (rightmost plot). 108
5.13 Histogram of rewards on cart-pole balance task (started with pointing pole in upward with downward velocity) on agents with low reward on cart-pole task due to catastrophic forgetting. 109
5.14 Frames of a movie showing the agent's movement through the state space for an example run of a single Q-learner ($N_e = 1$). Each Frame is a single time step. The first 120 time steps are shown. State space locations are colored according to the number of times the preferred action at that location changed. The red dot in the agent's current location in state space. Color bar is scaled to match colors in Figure 5.15 110
5.15 Frames of a movie showing the agent's movement through the state space for an example run of an ensemble Q-learner ($N_e = 20$). Each Frame is a single time step. The first 120 time steps are shown. State space locations are colored according to the number of times the preferred action at that location changed. The red dot in the agent's current location in state space. Color bar is scaled to match colors in Figure 5.14 . . . 111
5.16 Surface plot of $\max_a Q(s, a)$ with each tile colored according to $\max_a Q(s, a) - \min_a Q(s, a)$. This image is included to show the magnitude of the Q-function relief across states relative to the Q-function difference across actions. 112
5.17 Three contour plots of $Q(s, a = -1)$ (blue), $Q(s, a = 0)$ (green), and $Q(s, a = 1)$ (red). 113
5.18 Contour plots of critic Q-functions on the pendulum task during training overlayed with the actor output as an unfilled contour plot. The $Q(s, a)$ values are shown for the a selected by the actor for that given s. The red and blue lines of the actor-output contour plot are the extreme values of $a = -2$ and $a = 2$. The gray lines are the more moderate action values. 114
5.19 Contour plots of critic Q-functions on the pendulum task during training overlayed with the actor output as an unfilled contour plot. The $Q(s, a)$ values are shown for the a selected by the actor for that given s. The red and blue lines of the actor-output contour plot are the extreme values of $a = -2$ and $a = 2$. The gray lines are the more moderate action values. 115
5.20 Contour plots of selected action (actor network output) in black and brown overlayed with the critic Q-functions differences between the best and worst actions as an unfilled contour plot in green and blue. 116

5.21 Contour plots of selected action (actor network output) overlayed with the combined ensemble critic Q-functions differences between the best and worst actions as an unfilled contour plot. 117
5.22 To top row shows data collected from an example, single critic agent and the second row from an ensemble critic agent. Column one shows the ratio of the Q-function values of a randomly-drawn agent for the positive direction actions over the negative direction actions over the 500 training episodes. Column two shows the average Q-function value for locations where a positive or negative direction action is preferred by the learned policy for the same agent. The plot in the third row shows the mean ratio size of the action direction with the largest ratio averaged across all five randomly-selected agents. 119
5.23 Comparison of the angle the pole is held after an agent has solved the task for the non-ensemble and ensemble approaches. The ensemble approach utilizes both actions to keep the pole pointing straight-up much of the time while the non-ensemble approach always favors one action over another and always balances the pole off-center. Counts are for five agents in both histograms. 121
5.24 Comparison of critic preferred action and actor output changes during training for ten ensemble and ten non-ensemble agents on the bipedal walker task. Y-axis limits were selected to make differences visible; the non-ensemble agents' Q-function and policy output changes are frequently outside the given range. 123
5.25 Comparison of maximum critic (Q-function) output changes during training for ten ensemble and ten non-ensemble agents on the bipedal walker task. Y-axis limits were selected to make differences visible; the non-ensemble agents' Q-function and policy output changes are frequently outside the given range. 124
5.26 Comparison of all crowd ensemble member solutions for agent with $N_e = 5$ after training for 40,000 time steps. Each sub-figure consists of four plots (listed top to bottom): cart position, cart velocity, pole angle, and pole angular velocity. 126
5.27 Mean, cumulative (computed as distance from initialized values) change in parameter values of two convolutional layers of DQN networks used in high-dimension cart-pole task. Dashed lines are the first convolutional layer (input layer) and solid lines are the second layer. Red lines are features learned by an ensemble of size $N_e = 10$ and black are features learned by a single Q-learner. 127
5.28 Mean change in parameter values of two convolutional layers of DQN networks used in high-dimension cart-pole task (magnitude of weight matrices from time $t-1$ to time t. Dashed lines are the first convolutional layer (input layer) and solid lines are the second layer. Red lines are features learned by an ensemble of size $N_e = 10$ and black are features learned by a single Q-learner. 127
5.29 Four examples of change in parameter values of two convolutional layers of DQN networks used in the high-dimension cart-pole task (magnitude of the weight matrix changes from time $t-1$ to time t) for $N_e = 1$ networks. Red lines are the first convolutional layer (input layer) and blue lines are the second layer. 128

5.30 Fraction of gradient update movement in the direction of the final parameter values over the total gradient at selected time steps during training of the high-dimension cart-pole agent for two N_e values. For reference, a horizontal line is added to each plot at $y = 0$. The two vertical lines separate the plot into the three groups of sequential weight update samples. 130

6.1 Actor-critic architecture with a critic comprised of an ensemble of Q-functions and a single actor network. All networks share the input layer. The state would input into the shared layer and the Q-functions would augment the output of the shared layer with the action information to compute $Q(s, a)$. 138

6.2 Seed corn dryer with 28 bins and four sets of fans and burners. Fans force air into upper tunnel (left). Air is pushed through roughly half of the bins into the lower tunnel and then out of the building through the remaining bins. 140

Chapter 1

Introduction

The promise of reinforcement learning (RL) is that an agent can be taught to learn some behavior by supplying it with a (potentially incomplete) representation of its current state and only a scalar-valued feedback representing the quality of its performance as a training signal. This stands in contrast to the supervised learning approach where the agent is taught from a set of sample training states providing the preferred action for every state as the training signal. RL allows an agent to learn by trial-and-error while supervised learning requires the human trainer to be able to provide the preferred action at every position in the training set.

In the RL approach the agent learns by exploring its environment and the, sometimes, many approaches to solving a given problem. It frees the human trainers from having to know the preferred action or intrinsic value of each encountered state. There is no denying that RL has a grassroots feel: it is an important form of learning in the natural world and it is only natural that machine learning practitioners would want to mimic its success.

This freedom comes at a price, however. Common complaints about RL, especially when using function approximators to learn value functions or Q-functions, include that RL is too slow and unstable during learning. Instability during learning results in agents performing just as poorly as they did at the beginning of training after solving the task in as little as a single parameter update. The problem with RL, in many circumstances, is not in finding a good solution but in keeping and finding the solution in a reasonable amount of time.

RL is too slow to find solutions. In a continuous state space, the types of which are all around us, slow convergence can be exacerbated by the size of the state space. RL algorithms must use the limited feedback to train the function approximator to find a set of features suitable to representing the state such that a useful value function can be learned.

The choice of function approximator can get in the way of finding a good solution. Local minima can trap the agent in a poor part of the solution space. Artificial neural networks (ANN) are

a popular selection for the Q-function approximator but they are rife with local minima, particularly when using the most common parameter update algorithms. As seen in later sections, this can lead to an agent with a Q-function approximated using a single ANN which never obtains even a mediocre solution. Furthermore, the function approximator may not be powerful enough to represent the optimal Q-function.

RL function approximators suffer from instability during learning. Value functions learned via function approximation are unstable. RL problems have non-stationary distributions of states and actions. Often previous regions of the state-action space are revisited later in learning, and current approaches can result in losing knowledge of value and policy by the time they are revisited.

This dissertation presents a method which attempts to directly address these three major obstacles to reinforcement learning application. The proposed approach is an ensemble learning approach to RL which trains ensemble members using the same data points and combining them all equally using voting or averaging.

1.1 Motivating example

The cart-pole task is a non-trivial, classic RL task which serves as the central experimental domain from which we draw a majority of our conclusions in this work. Figure 1.1 shows evaluated solutions from an agent while training on the high-dimension variant of the cart-pole task where the RL agent receives a state representation as an image of the cart-pole track. The agent was trained using Q-learning. The Q-function was modeled using a deep artificial neural network with two convolutional layers and two fully-connected hidden layers. The agent was trained to balance the pole by rewarding it with a $+1$ when near the up position, a -1 when near the down position, and 0 elsewhere. The solutions are viewed as a graph of the pole position over time. Every 1000 training time steps the agent was evaluated for 2000 time steps. Figure 1.1a is the initial solution after just 500 time steps of training. This initial solution does little more than allow the pole to swing, very gently, back and forth across the $\pi, -\pi$ threshold which is the down-pole position.

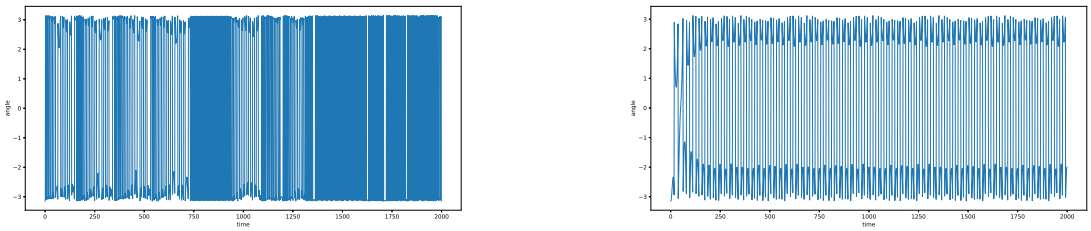
Figure 1.1b is the solution after $t = 350,000$ time steps. This solution swings the pole up a bit higher and does so by rocking the cart back and forth (cart position not shown here). It is able to achieve a higher reward during evaluation because the pole swings out of the region of the space where we give the agent a reward of -1 into the region where a reward of zero is given. Figure 1.1c is the solution after $t = 2,000,000$ time steps where the agent is getting close to solving the task with a reward of $R = 1351$ when summed over 2000 steps. The agent is now able to repeatedly swing the pole into the goal region. With every swing into the goal region, the agent receives a reward of $+1$. There is a period in the middle of the evaluation where the agent is able to balance the pole for over 100 time steps.

Figure 1.1d is the solution after $t = 3,000,000$ time steps where the agent has effectively solved the task with a reward of $R = 1896$. The simulation sample rate is 15 times per second, so 56 hours of simulated training time was required to get to this point! While balancing the pole in the upward position the cart is moving quite a bit as it counteracts the pole's shift from right of upright to left of upright.

Figure 1.1e is the solution at $t = 3,400,000$ steps where the evaluated reward has dropped precipitously to $R = -1850$. We observe that the agent has forgotten how to solve this task. We see that after some initial attempt to swing the pole up, the agent does little else to attempt to swing it back up into the goal. Instead, the pole's momentum swings it back and forth through the down-pole position swinging less high with each pass. The inconsistency of this performance will not instill a human operator with a sense of confidence in a controller. After two simulated days of training the agent is back where it started. It will take many hours of training for the agent to regain the ability it has lost. This characteristic of RL algorithms to forget good solutions is one of the motiviations for new algorithm development in this dissertation.

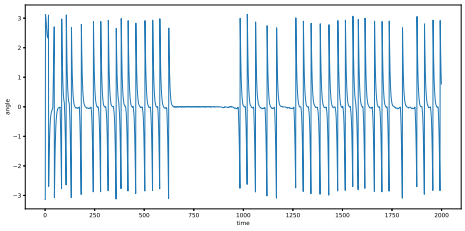
1.2 Recent advancements in reinforcement learning

There has been tremendous advancements in RL in recent years spurred by the excitement surrounding deep Q-learning (DQN) [1]. That work boldly reset the bar for the RL community and

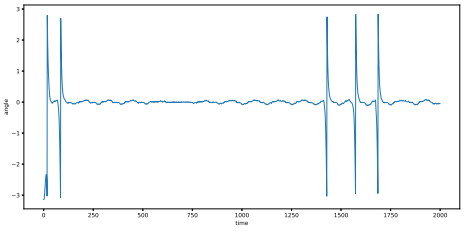


(a) $t = 0$, $R = -1967$

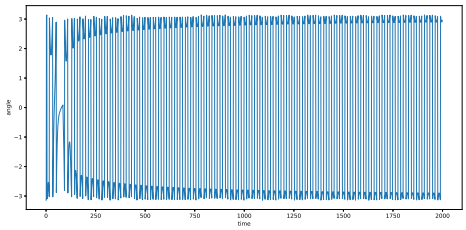
(b) $t = 3.5 \times 10^5$, $R = -1082$



(c) $t = 2 \times 10^6$, $R = 1351$



(d) $t = 3 \times 10^6$, $R = 1896$



(e) $t = 3.4 \times 10^6$, $R = -1850$

Figure 1.1: Five evaluations of the same agent on the cart-pole task where the agent state is represented as an image of the entire cart-pole track designed to simulate pointing a camera at the system we wish to control. Each evaluation is 2000 time steps in length. Zero position on the y-axis is the goal region and π and $-\pi$ are pointing down and are the same point in state space.

challenged us to attempt to solve more challenging tasks. Impressively, RL has featured prominently in published work showing super-human performance in tasks which were previously considered untouchable by state of the art RL approaches just a few years ago. These include RL agents playing chess at a grand-master level [2] and the famous AlphaGo [3].

This has been good for the community and we have found just how far we can go when applying existing RL approaches refined with DQN techniques coupled with large amounts of computing power. There has been work showing other RL approaches can be adapted to the deep Q-learning paradigm [4]. With a new set of flags planted, a large amount of effort has been de-

voted to addressing the RL obstacle of slow learning [5, 6] with an emphasis on speeding-up RL for the high-dimension inputs popularized by the DQN work – especially since DQN exacerbated this issue both in terms of the required computation and number of training time steps. Other work set about to addressing the problem of simplifying the task of finding successful meta-parameters – an issue also exacerbated by DQN's increased number of meta-parameters.

For all of these recent works mentioned here the primary emphasis is achieving improved performance on simulated tasks in the shortest amount of wall-clock time. The issues of RL instability during training and of reducing the number of interactions with the training environment have received relatively less attention. The crowd ensemble is able to address all three RL obstacles to varying degrees without sacrificing the number of interactions with the training environment which, in many real-world application, may be more expensive than the computational costs. Furthermore, the crowd ensemble method can be used alongside any of these recent advances in RL.

Methods to reduce training time using methods such as pre-training [7, 8], transfer learning [9], and learning from human demonstration [10] have also been developed recently. The crowd ensemble method can also be used with these methods for further reduction of training time.

1.3 The wisdom of crowds

Several fields, including computer science, have long recognized, and have attempted to harness, what conventional wisdom has known for a century or more: that crowds make better decisions than individuals. The extent of this enhanced ability can be surprising, however. In an early realization on the subject, Francis Galton, in 1906, observed that a large group's mean guess was able to come within one pound when guessing the weight of an Ox [11]. This was surprising because the crowd, while containing a few potential experts such as farmers and butchers, was presumably made of non-experts with no knowledge of estimating the weight of oxen.

The popular bean count estimation contest is formalized as part of an investigation into crowd wisdom in market pricing by Treynor in [12]. In these contests a large jar is filled with beans and contestants have to guess at the number in order to win a prize. In all experiments allowing

the individual guesses to be independent consistently led to mean guesses which were within the top five percent of all guesses and the one or two contestants who beat the average in a given trial changed in subsequent trials. Conversely, allowing for shared biases by giving all contestants additional information or planting biases (e. g., telling the contestants that the jar walls are thinner than the typical jar) reduced their independence and increased the error of the crowd's average. The more shared warnings, the worse the average. The same phenomenon is observed when allowing for other sources of shared error such as sharing the estimate of an expert with all contestants. Applying this insight to investing, Treynor concludes that, if a company's value is affected by 100 attributes and an investor has found an error in two of them, any informational advantage that investor may have will be overwhelmed by his or her disadvantage with respect to the crowd on the other 98. He concludes that chasing an expert is folly and a simple combination of crowd information is best.

Larrick and Soll [13] emphasize the power of averaging predictions and the little-understood caveats surrounding its application. Appropriate use of averaging in group decisions begins with understanding that averaging doesn't mean a regression to the mean nor getting something from nothing. Instead it is an error-reduction technique: the error of the average is less than the error of the average individual. Combining estimates can do no worse than the average individual.

Hastie and Kameda [14] examine two variants of majority rule. Like our RL tasks their analysis relies upon the decision-makers having the same goals: that what is good for one is equally good for all. Their measure of success is the accuracy of the prediction. They find that simple majority rule voting selects the same action decision as the condorcet majority winner selection method which is the standard by which most social choice alternatives are compared.

James Surowiecki, a journalist, synthesizes the many benefits of crowd-based decision-making and the many ways it has been built into our society in his book *The Wisdom of Crowds* [15]. Although the crowd rarely out-performs the best individual [15] no field can provide a mechanism to predict which individual this will be. Fortunately, many of the obstacles to quality crowd decision-making are a result of the shortcomings and complexities of human group dynamics [14,15]. Prob-

lems like group-think and competing objectives are not problems we will encounter when training Q-function ensembles: instead our crowd ensemble members' motivations are perfectly aligned.

Surowiecki uses the Challenger space shuttle disaster [15, p.11] as an example of the crowd's uncanny ability to predict in his summary of the work by Maloney and Mulherin [16]. In the minutes after the Challenger explosion investors immediately sold shares in the four major companies involved in the design and manufacture of the shuttle. One of them, Morton Thikol, was sent into free fall whereas the stocks for the other three contractors had stabilized and began to recover by the end of the first day. Maloney and Mulherin investigated the stock market's reactions and could find no incident of insider trading behaviors from any of the four major contractors. The crowd of individuals had correctly identified, within hours, what a panel of experts would eventually disclose six months later: the company which had manufactured the faulty component responsible for the disaster.

1.4 Objectives

The RL community has taken on new, larger challenges, has invented new supplementary techniques, and applied greater amounts of computing power to solve these tasks in a tolerable amount of time. We wish to bring RL closer to being ready for widespread use in industry, particularly for control of real world, physical systems. If RL is going to leave the laboratory and find widespread, real-world use, addressing all of the major obstacles presented earlier in this chapter is required. Furthermore, these challenges must be addressed, not only on the new, more difficult tasks, but also on the types of tasks on which we have been testing our RL approaches for years where performance improvements are harder to come by.

The crowd ensemble approach explored in this work does just that. The first objective of this work is to show that the crowd ensemble improves performance on a variety of tasks: the cart-pole (a classical RL task), a high-dimension variant of the cart-pole where the state is represented by an image of the cart-pole environment, and the bipdal walker task which which can be solved more efficiently using an actor-critic approach instead of Q-learning which is used for the cart-pole

tasks. A parallel set of objectives is to demonstrate that the crowd ensemble can address the three major challenges to RL. We pay special attention to addressing the challenge of instability during training because that particular obstacle has not been addressed in recent RL advancements and we hypothesize that it is the obstacle which the crowd ensemble is best suited to address.

Our final objective is to understand why the crowd ensemble does or does not benefit a specific class of tasks or address a particular challenge. By studying why the crowd ensemble works we hope to find additional RL training techniques or approaches which can further address these obstacles.

1.5 Document overview

This dissertation discusses the background necessary to understand the proposed approach and its place in the literature in Chapter 2. Chapter 3 explores the state of the art for all four tasks used to test the effectiveness of the crowd ensemble approach. Here we will develop an intuition about the challenges and opportunities each task presents. We show that these tasks can all be solved without utilizing a crowd ensemble but that there is room for improvement when using these baseline solutions.

In Chapter 4 we see results comparing the crowd ensemble approach against our baseline, state of the art approaches from Chapter 3. Each task utilizes a different RL approach to find a solution and applying the crowd ensemble technique to each approach requires some creativity. Here we will see that the crowd ensemble improves both the mean and median reward received by the agent in every experiment. These results highlight the flexibility of the crowd ensemble. In this chapter we also see that the crowd ensemble is able to reduce training time for these tasks, overcome training issues associated with our choice of function approximator, and improve the stability of learned solutions: all three of the major obstacles to RL application listed above.

Chapter 5 takes a deeper look at the results of Chapter 4. It begins with an examination how the crowd ensemble addresses the particular obstacle of instability during training. We find that the Q-functions learned by all the RL approaches examined here do an excellent job of identifying

good states (e. g., cart and pole positions and velocities) but have a much weaker differentiation between action quality. The crowd ensemble combination scheme dramatically improves the action discrimination capabilities thus masking this aspect of Q-learning. The chapter also discusses how the crowd ensemble can be used to speed-up RL.

The final chapter, Chapter 6, discusses the ramifications of our analysis and results, some directions for new research illuminated by these results, and several items left for future work. The wisdom of the crowds informs us that, when certain conditions are met, the best way to combine the predictions of an ensemble is to do so with naiveté . We conclude that the crowd ensemble approach meets these criterion and fits with the body of research on the subject.

Chapter 2

Background

This chapter is a presentation of the core concepts our experiments are built upon. We begin with a look at the predominant ANN-update algorithms used by RL practitioners today and how those approaches improve upon the stochastic gradient descent (SGD) method which had dominated RL/ANN research until recently. We will also discuss how artificial neural networks (ANNs) are utilized to model Q-functions and, in the case of actor-critic, policies. We then discuss ensemble methods in general and a classical example of a crowd ensemble from the computer science literature before moving on to RL foundations. Finally, we return to ensemble methods focusing on RL. RL has been a very active field in the past few years. This dissertation touches upon several emerging areas of reinforcement learning (RL). Many of the tricks of the trade that were developed in that time are discussed here.

2.1 Commonly-used neural network update methods with RL

Gradient descent is the prevalent paradigm for updating ANN parameters including when using an ANN to approximate a Q-function. Each parameter update is computed from a batch of samples each of which consists of a state/action pair, $\big((s(t_1), a(t_1)\big)$, and a target Q-value, $v(t_1)$:

$$\big((s(t_1), a(t_1)), v(t_1)\big), \big((s(t_2), a(t_2)), v(t_2)\big), \ldots, \big((s(t_D), a(t_D)), v(t_D)\big)$$

The defacto gradient descent method is SGD which computes a weight update for each training sample separately. The benefit of SGD is that taking these small steps results in a noisy gradient descent which can avoid some of the local optima of the global error surface that might otherwise be encountered when performing a single parameter update for the entire batch [17]. This is also a natural update algorithm for RL which receives its training samples incrementally and is used in

recent seminal works such as [1]. Alternatively, the parameter updates from the entire batch can be summed prior to the actual update in an approach referred to as batch gradient descent.

Utilizing an ANN with a single output is more amenable to continuous actions than utilizing a seperate output for each action and is, historically, the more common architecture for modeling a Q-function. The quantity $\frac{\partial E_d}{\partial w_j}$ is the derivative of the error with respect to the output layer weight connecting input j to the single output node for training sample d. The parameter w_j is the weight connecting input x_j to the output, o. The value a_o is the potential of the ANN output which is computed as $a_o = \sum_j w_j x_j$ and then passed through the activation function to compute the output: $o = \sigma(a_o)$. In this example we use the sigmoid activation function: $\sigma(a) = \frac{1}{1+e^{-a}}$. The output error in this example is the commonly used squared error: $E_d = \frac{1}{2}(v_d - o_d)^2$. In the derivation below, we drop the d subscript on the variables o, a_o, and x_j for compactness but these values are computed for each data point. The update for a single training sample for a single layered ANN with one output is

$$\begin{aligned}
\frac{\partial E_d}{\partial w_j} &= \frac{\partial E_d}{\partial a_j}\frac{\partial a_j}{\partial w_j} && (2.1)\\
&= \left(\frac{\partial E_d}{\partial a_j}\right)\left(\frac{\partial}{\partial w_j} w_j x_j\right)\\
&= \left(\frac{\partial E_d}{\partial a_j}\right) x_j\\
&= \left(\frac{\partial E_d}{\partial o}\frac{\partial o}{\partial a_o}\right) x_j && (2.2)\\
&= \left(\frac{1}{2}(v-o)^2\right)\left(\frac{\partial o}{\partial a_o}\right) x_j\\
&= \left(\frac{1}{2}2(v-o)\frac{\partial t - o}{\partial o}\right)\left(\frac{\partial o}{\partial a_o}\right) x_j\\
&= -(v-o)\left(\frac{\partial o}{\partial a_o}\right) x_j\\
&= -(v-o)\left(\frac{\partial \sigma(a_o)}{\partial a_o}\right) x_j\\
&= -(v-o)\big(\sigma(a_o)(1-\sigma(a_o))\big) x_j\\
&= -(v-o)o(1-o)x_j && (2.3)
\end{aligned}$$

which results in a weight parameter update of

$$\Delta w_j = \eta(v - o)o(1 - o)x_j. \tag{2.4}$$

Notice that the direction of the gradient from (2.3) is reversed in the parameter update in (2.4) to perform error minimization.

The hidden layer parameter updates pass the error of the downstream (i. e., closer to the output) layer back through its own weights and activation function to compute a similar parameter update. Letting $\delta_k = \frac{\partial E_d}{\partial a_k} = (v-o)o(1-o)$ which is the error for downstream node k derived in (2.2)–(2.3), the error gradient for the hidden layers are derived as follows.

$$\begin{aligned}
\frac{\partial E_d}{\partial w_{kj}} &= \frac{\partial E_d}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \\
&= \left(\frac{\partial E_d}{\partial a_j}\right) x_i \\
&= \sum_{k \in O^{(j)}} \frac{\partial E_d}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\
&= \sum_{k \in O^{(j)}} \delta_k \frac{\partial a_k}{\partial a_j} \\
&= \sum_{k \in O^{(j)}} \delta_k \frac{\partial a_k}{\partial o_j}\frac{\partial o_j}{\partial a_j} \\
&= \sum_{k \in O^{(j)}} \delta_k \left(\frac{\partial}{\partial o_j} w_{kj} o_j\right)\frac{\partial o_j}{\partial a_j} \\
&= \sum_{k \in O^{(j)}} \delta_k w_{kj}\frac{\partial o_j}{\partial a_j} \\
&= \sum_{k \in O^{(j)}} \delta_k w_{kj}\frac{\partial}{\partial a_j}\sigma(a_j) \\
&= \sum_{k \in O^{(j)}} \delta_k w_{kj}\sigma(a_j)(1 - \sigma(a_j)) \\
&= \sum_{k \in O^{(j)}} \delta_k w_{kj} o_j(1 - o_j) \\
&= o_j(1 - o_j) \sum_{k \in O^{(j)}} \delta_k w_{kj}
\end{aligned}$$

In this and all upstream hidden layers δ_j is defined to be

$$\delta_j = o_j(1 - o_h) \sum_{k \in K^{(k)}} w_{kj}\delta_k$$

resulting in a parameter update of

$$\Delta w_{kj} = o_j(1 - o_h) \sum_{k \in K^{(k)}} w_{kj}\delta_k.$$

Recently, parameter update methods which use heuristics to speed-up training have become the norm. In this dissertation two such methods are used: scaled conjugate gradient (SCG) and ADAM, the latter of the two holding a place of far greater popularity in the literature. SCG has not found wide-spread use in the RL community but we have experienced good results when applying it to lower-dimension tasks.

SCG is a mathematically sophisticated algorithm [18, 19] [20, p. 56]. The conjugate gradient (CG) method works by attempting to eliminate the zig-zagging motion of steepest descent where the same direction is selected multiple times on its way to an optima. This is done by recognizing that the residual, the direction and magnitude of steepest descent of the error function with respect to the current position in parameter space, is orthogonal to all previous search directions assuming the steps taken in the direction of the previous search directions were of optimal length. Conceptually conjugate gradient approaches function by selecting a direction, updating parameters in that direction, and then selecting a new, conjugate direction to prevent spoiling the gains made by updating in the new direction. Optimally the update in each direction would find the function minimum in that direction to remove as much error as possible. However this is not computationally feasible.

In practice conjugate gradient methods are imperfect and their basic assumptions are routinely violated – especially when used in conjunction with RL. For instance the Q-function is updated using only a small, unrepresentative set of training samples. Furthermore, heuristic approaches such as experience replay result in optimizations occurring on different training samples. For

this reason the RL may not always benefit from the mathematical elegance of conjugate gradient methods and a more approximate, less computationally burdensome method may work just as well.

Enter SCG. SCG speeds-up the conjugate gradient approach by avoiding the line search for a function minimum in each of the chosen directions and replaces it with an estimate of the distance in the current conjugate direction which the update algorithm should advance. The SCG algorithm contains some heuristics but they are grounded in numerical analysis and seem to work well in practice. SCG has three meta-parameters which determine when training is terminated. One controls the minimum update size of the parameters, another controls the minimum change in the objective function between parameter updates, and the third sets a maximum number of conjugate gradient steps (or updates) that will be preformed.

SCG begins with an initial set of parameters. For each conjugate gradient step (parameter update) the following steps are taken:

1. Compute a new conjugate direction.
2. Update estimate of Hessian matrix.
3. Compute a step size in direction of maximal gradient using Hessian estimate.
4. If the parameter update conceived from step size and direction of maximal gradient leads to a large enough decline in the objective function, update the parameters.

ADAM is a momentum-style heuristic to improve the SGD method for updating ANN parameters [21]. ADAM, in its definition, defines the objective function to be noisy either as a result of ADAM receiving a different, random sub-sample of the training data for each update or as a result of a noisy objective function itself. Both of these are present in state-of-the-art RL approaches.

ADAM begins with an initial set of parameters values, θ_0. ADAM maintains, across updates, an estimate of the mean and variance of the gradient. ADAM has four meta-parameters which must be set before training: ϵ, β_1, β_2, and α. The learning rate, α, controls parameter update step size just as in traditional gradient descent algorithms. The parameter β_1 controls the exponential decay of the gradient history for the mean gradient calculation. The parameter β_2 controls the exponential

decay of the gradient variance calculation. The parameter ϵ is there to ensure computational safety as shown in step five below.

ADAM proceeds as follows:

1. Compute error function's first-order gradient, g_t given current set of parameters θ_t.

2. Update the mean gradient according to $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$.

3. Update the gradient variance estimate according to $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$.

4. Compute bias-corrected gradient mean and variance estimates according to:

$$\begin{aligned} \hat{m}_t &\leftarrow \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &\leftarrow \frac{v_t}{1 - \beta_2^t}. \end{aligned}$$

5. Update parameters:

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}.$$

Steps one through five are iterated until some convergence criteria is satisfied. In a RL application this amounts to performing steps one through five some arbitrary number of times according to the observed performance of the agent. In other words, convergence of θ_t is not monitored.

The tracking of m_t and v_t over time is similar to the concept of momentum from the SGD literature [17, p. 93]. It is more complex, however, in that the mean of the gradient computed in step two above, m, is scaled by the inverse of the variance. This is a measure of uncertainty of the gradient direction [21] and serves to reduce the gradient when there is a high degree of uncertainty (disagreement) about the direction of the parameter updates. Momentum is typically thought of as a way to more quickly traverse regions of the parameter space with little gradient or to climb out of local optima or skip over ridges. Momentum can also be thought of as a regularizing technique that

prevents overfitting in the latter parts of training while further speeding-up training in the so-called transient phase of learning where rapid progress through parameter space is made [22].

Kingma and Ba express that the default settings for the ϵ, β_1, β_2, and α meta-parameters ought to be sufficient for most domains [21]. This claim is supported by the literature and by our own experience although we use non-default α values for some experiments.

2.2 Ensemble learning

Multiple, homogeneous or heterogeneous, function approximators combined to create a superior function approximator is the defining characteristic of ensemble learning. The crowd ensemble technique considered here encourages diversity in membership by presenting the members with different training samples. The crowd ensemble is very similar to bagging (or bootstrap aggregating) [23]. Bagging trains N_e models using N_e data sets sampled from a set of training samples with replacement. The predictions of the models are combined using simple averaging or voting. There is no attempt at selecting winning ensemble members or dividing the data to encourage specialization.

More sophisticated ensemble learning approaches incorporate a scheme designed to take advantage of the relative strengths of the ensemble members in addition to presenting the learners with different training samples. An early example of this sort of ensemble learning approach is boosting [24] which incrementally adds members to the ensemble and trains them using data points randomly drawn according to the difficulty of that data point for the ensemble. Furthermore, each ensemble member is weighted according to the strength of its prediction when combining their outputs.

The prototypical ensemble method for combining ANNs is commonly referred to as a mixture of experts (MoE) [25]. In the MoE approach ensemble members are combined via weighted combination. The weighted combination is determined by an additional ANN called the gating network. The gating network is trained at the same time as the experts and its weighting is used to modulate the error that is backpropagated back through the experts. Figure 2.1 shows a diagram of

a MoE. The ensemble output is computed using

$$w_c \leftarrow \frac{e^{x_c}}{\sum_{j=1}^{C} e^{x_j}}$$
$$o \leftarrow \sum_{c=1}^{N_e} w_c o_c$$

where x_c is the gating network output for expert c, w_c is the softmax of of the gating network outputs, o_c is the output of expert c, and o is the output of the ensemble. The global error for the mixture is

$$\delta \leftarrow \|t - o\|^2$$

where t is the target output. The error for each ensemble member is

$$\delta_c \leftarrow w_c \delta$$

which is the error of the combined output scaled by the weight of each ensemble member used for that prediction, w_c. The weighting network's error is

$$\delta^{(w)} \leftarrow \sum_c w_c \|t - o_c\|^2.$$

This has the effect of raising the w_c value for all experts which have errors less than the average error of the mixture [25].

2.3 Crowd ensembles in computer science

The 1990's artificial life community seized upon the concept of improved group capability emerging from non-cooperative agents. One such work was performed by Johnson [26, 27] where the possibility of emergent problem solving was explored using agents trained to solve a maze navigation task. These agents were trained in isolation with no interaction or cooperation. The population consisted of up to 100 individuals. The agents have no conception of the global maze

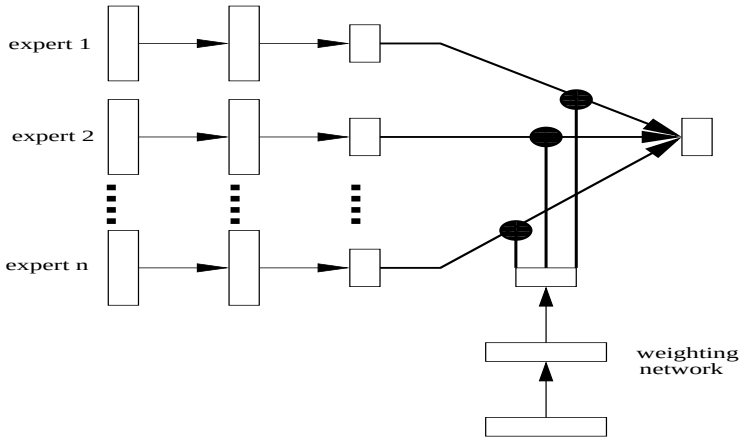


Figure 2.1: Example mixture of experts (MoE). Shows three or more experts, each an ANN with a single hidden layer. The mixture's output is a weighted combination of the experts' output. The weights come from the weighting network (shown as an ANN oriented perpendicularly to the experts).

layout or goal position. At each node in the maze the individuals developed action preferences during a single trip through the maze which began at the starting location that lasts until the goal location was found. Each maze had multiple optimal solutions. Each individual had identical capabilities and searched for the goal using a simple set of learning rules. Many sets of similarly simple rules were investigated but had no impact on the conclusions of the study.

They define P_{miq} as the preference for individual m to travel from node i to node j via edge q. During the learning phase all P_{miq} are initialized to zero and are updated according to the following rules.

1. If at goal, stop.
2. If current node has any $P_{miq} = 0$ do the following.
    - Choose link randomly from all links with $P_{miq} = 0$.
    - Set chosen link to $P_{miq} = 1$.
    - Set link $P_{mjq} = 0.1$ (the reciprocal edge from node j to i).
    - Set links with $P_{miq} = 1$ to $P_{miq} = 0$.
3. If all $P_{miq} > 0$ for a particular node

- Choose link with maximum P_{miq} value.

When exploiting (application phase) the learned P_{miq} values the following rules are followed (when at current node, i):

1. If at goal, stop
2. From set of all edges with maximum P_{miq} values, select randomly
    - If possible exclude link leading back to previous node
    - Mark selected link as "tried."
3. If all edges have been tried, select link at random.

The application rules have the effect of removing loops in paths. Note that the P_{miq} values are unchanged by the application rules.

The ensemble is created by using the applications rules to traverse the maze and combining the preferences of the individuals using:

$$\Pi_{giq} = \frac{1}{N_e} \sum_{m=1}^{N_e} P_{miq}.$$

From this simple experiment and subsequent analyses, the author comes to many conclusions. The primary conclusion is that the simple combination scheme of average individual preferences was impossible to beat and it finds the optimal solution reliably and with stability. The author also found that the resulting ensemble was robust to a variety of noise sources and that the robustness increased with N_e. An ensemble with $N_e = 20$ could withstand 70% of the preferences of the individuals to be altered without affecting performance.

In a second experiment the agents were allowed to update the preferences, the P_{miq} values, using the experience from the application phase. This led to dramatically improved performance of the individuals. Still, the ensemble performance was significantly better than the average individual.

The study also showed that limiting the ensemble membership to only those agents which achieved average or better path length through the maze didn't improve the ensemble solutions. In fact, limiting the membership according to individual training performance hurt the ensemble performance.

In a subsequent study, by alternatingly holding the capability of the individuals and maze difficulty constant the following conclusions were made: ensembles don't improve when applying highly capable individuals to simple tasks, ensemble size improves performance more slowly as maze difficulty increases, and more difficult mazes require better individuals. This last conclusion asserts that the ability of the individual puts a ceiling on the ability of the ensemble.

2.4 Q-learning

Sutton and Barto [28] describe a reinforcement learning approach as requiring three elements: a policy which defines the agent's behavior by mapping the perceived state of the agent to an action, the reward function which defines the goal, and a value function which computes the amount of reward an agent can expect to accumulate by taking a specific action from a specific state. In short, the reward function specifies the immediate value of a state while the value function specifies the long-term value of a state as the cumulative reward. Some RL approaches utilize a model of the environment. Avoiding model-based RL allows for examination of the methods free from potentially complicating factors such as the quality of the model used. Therefore, a model-free approach is utilized here.

RL has, at its roots, temporal-difference (TD) learning. TD learning bootstraps a the Q-function approximation by updating an estimate using estimates [28]. TD learning works in the context of the most common RL techniques because it assumes that, with every step toward a terminal state, information is less uncertain which means the estimate is more accurate. If traversing a maze and using TD learning to update the future value of the positions of the maze, it is logical to assume that the value estimates of the states nearest the goal or terminal states are more accurate. Therefore,

backing an estimate up to the state which was encountered immediately prior will provide a slightly more accurate estimate of that previous state's value.

TD learning can be further strengthened by adding an off-policy update mechanism. On-policy TD learning limits the TD updates to backing up values to the state seen previously. However, in many instances, a state can be reached from a plurality of previous states. Similarly, from a given state, a plurality of next states are possible. Therefore it makes sense to allow the TD learning update for a given state to back up its value from its best possible next state. This is called off-policy TD learning.

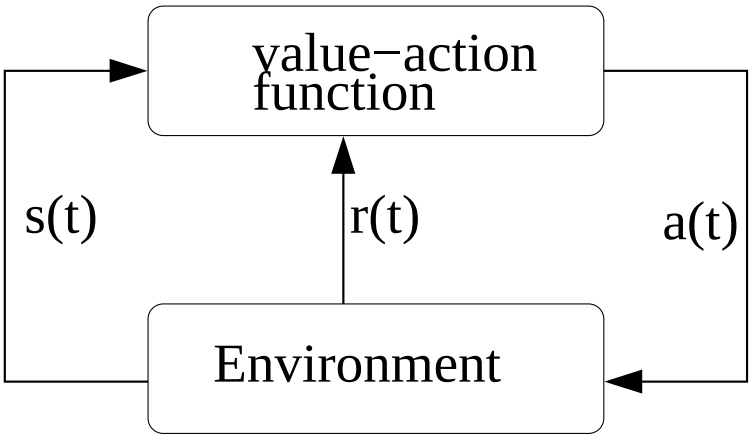


Figure 2.2: Architecture of Q-learning.

Q-learning is an off-policy, temporal-difference approach to RL [28]. In Q-learning, the value function is approximated using the Q-function, a value-action function which maps state/action pairs to an estimate of the value. Figure 2.2 shows the information flow of Q-learning. The Q-function takes the state as input from the environment and outputs an action which is sent to the environment. The Q-function is trained using reward information from the environment.

The Q-function is updated through repeated interaction with the environment according to (2.5).

$$Q(s(t),a(t)) \leftarrow Q(s(t),a(t)) + \alpha\big[r(t+1) + \gamma \max_{a'}(Q(s(t+1),a')) - Q(s(t),a(t))\big] \quad (2.5)$$

where t is the current time step, $Q(s(t), a(t))$ is the Q-function value for state/action pair $(s(t), a(t))$, $s(t+1)$ is the state which occurs when taking action $a(t)$ from state $s(t)$, $r(t)$ is the reward at time t, α is a learning rate parameter, and γ is a parameter modulating the effect of future rewards. The value $\max_{a'}(Q(s(t+1), a'))$ is the current estimate of future cumulative reward: it represents the current belief of the best possible action, a', taken at the next state, $s(t+1)$. Q-learning is a temporal-difference method because it updates $(s(t), a(t))$ Q-function estimates using the estimate of the next state, $(s(t+1), a')$. In other words, the value function is bootstrapped during learning without waiting until the agent reaches a solution. Q-learning is off-policy because the value is backed up from $(s(t+1), a')$ where a' may be different than the action $a(t+1)$ selected by the policy.

The majority of the effort required to utilize a RL approach is typically encountered in learning the value function. The approaches discussed in this dissertation utilize Q-learning to learn the value function. To deal with the continuous state spaces of our experimental domains, we use an ANN to model the value function.

2.4.1 Q-learning using ANNs as function approximators

Using a feed-forward artificial neural network (ANN) [17] with backpropagation to model a Q-function is a common function approximation method when applying Q-learning to tasks with continuous state spaces [29]. There are several ways to model a Q-function using an ANN. One method is to input the state to the ANN and output a Q-function for each possible action. A common method which is more generally applicable, especially for continuous actions, is to input the state and action into the ANN and output the corresponding Q value. For example, computing a Q-value using an ANN with a single hidden layer requires the computations in (2.6) and (2.7)

$$\begin{aligned} z &\leftarrow \sigma\left([s(t), a(t)]W_{IH}\right) \\ Q(s(t), a(t)) &= zW_{HO} \end{aligned}$$

where σ is the hidden-layer activation function, $[s, a]$ is the concatenation of the state and action vectors into a row vector, z is the vector of hidden layer activations, W_{IH} and W_{HO} are the ANN weight matrices and $Q(s(t), a(t))$ is the scalar output. The output layer has a linear (pass-through) activation function so it is not shown in (2.7).

Updating the Q-function approximation is done by backpropagating the TD-error through the ANN. This is done by setting $v = r(t) + \gamma Q(s(t+1), a')$ and $o = Q(s(t), a(t))$ in (2.2)–(2.3).

2.4.2 Actor-critic RL

Actor-critic methods use two functions to solve the RL task: an actor which learns the policy mapping states to actions and a critic mapping states and actions to values. Training of both actor and the critic is driven by the temporal-difference error of the state-action value (critic) function. The action returned by the policy function is used to advance the environment to the next state which is then passed to both functions. This relationship is shown in Figure 2.3.

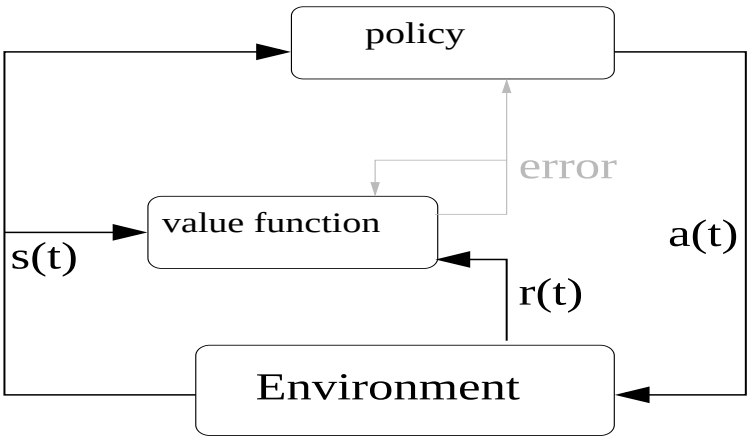


Figure 2.3: Architecture of actor-critic training process.

The critic learns a state-action value function utilizing Q-learning as discussed in Section 2.4.1. Although the policy is determined by the actor, the critic is trained off-policy because Q-learning uses the best possible action for the next state. There is some randomization applied to action selection to improve exploration.

The actor is trained from the critic's output. The actor takes only the state as input and outputs an action. The state and the action output by the actor are input into the critic to produce a Q-value (a critique of the supplied state/action pair). That Q-value is passed back through the actor as the gradient. This can be done using a simple

The actor-critic paradigm is used in the bipedal walker and pendulum experiments in this dissertation. We utilize two neural networks: one for the actor, the other for the critic.

2.4.3 Deep Q-learning

Deep Q-learning (DQN) [1, 30] instigated a sea change in RL since its conception. Although it did not constitute a new RL or ANN approach in and of itself, it was a novel combination of deep learning [31] from the supervised learning community, which was itself a modern take on work done by LeCun et. al. [32], with the base Q-learning approach using ANNs which had been in existence for decades [29]. Additionally DQN employs techniques such as experience replay which has since become a part of the Q-learning defacto approach. DQN is also frequently associated with target networks which further slow down training in exchange for increased stability during training.

When using target networks, two sets of parameter weights are kept: W_Q, the non-target network, and W_T, the target network. Q-learning progresses as described in Section 2.4 except the a' values from (2.5) are selected using $Q_T(s, a)$ which denotes that W_T is used in the Q-value computation. $Q_T(s, a)$ is also used for action selection which is sent to the environment to generate new training samples. The target network weights, W_T, are updated from W_Q by a weighted combination utilizing a mixing parameter to control the rate of the update:

$$W_T \leftarrow (1 - \tau)W_T + \tau W_Q.$$

DQN is commonly associated with using a very high dimension state representations generated from images of the state although the name and methodologies have also been applied to tasks

where non-image state representations of tasks have been used [4]. The DQN literature has also recently popularized ADAM as the preferable ANN parameter update algorithm.

Deep learning has turned heads with its ability to classify image data since its application in LeNet [32]. The LeNet approach featured convolutional layers in its deep networks which feature prominently in today's deep learning. It re-emerged to much fanfare many years later with some modifications and backed by advances in computational power [31] when it was shown to be capable of state-of-the-art (or better) performance on a number of domains using supervised learning. A deep learning approach is a neural network comprised of several layers each literally thought of as computing increasingly abstract representations of the high dimension input data. A deep learning network (or deep network) is not so much a new way of thinking about ANNs but a formalization of what ANN practitioners have always considered them to do by explicitly designing the ANN architecture to condition the network to more readily learn these representations. An example of this is the convolutional layers used in DQN where the parameters of the convolutional layers are literally interpreted as features which are being extracted from the image input as a form of pre-processing. The RL community followed suit and applied the deep learning architecture and attitude of harnessing advanced computing power to solve increasingly difficult tasks. These experiments contributed to the prestige of the Q-learning approach.

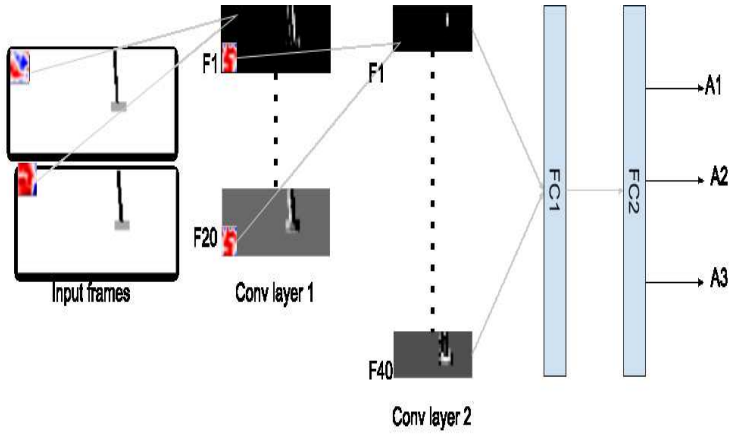


Figure 2.4: Example deep Q-learning network with two convolutional layers and two fully-connected layers and three action outputs. Shown are two example input frames and two example features at each convolutional layer. The input frames and subsequent features were captured from our DQN agent during training.

Figure 2.4 shows an example computation learned using DQN to solve the cart-pole task, one of the tasks used in this dissertation to analyze the crowd ensemble. Because only images are input to the DQN, state features such as velocities must be inferred by providing two or more image frames as the state. In Figure 2.4 we see two image frames being input. Convolutional layers are typically used as the first DQN layers when image data is involved. Here we see two convolutional layers. The first has 20 features, the second has 40. Figure 2.4 shows the output of convolving the first and last feature of each layer. Each of the twenty features of layer one is convolved across both frames to create 20 output features. Each of the forty features of layer two is convolved across all 20 output features from layer one to create 40 output features. In this example the output of the second layer is 40, 33×11 images which are vectorized into a 14520-dimension input for the first fully-connected (FC) layer. From there the data progresses through the fully-connected layers until it reaches the output layer. The standard output layer for DQN is one output per action. There are three possible actions in this example. This example is discussed in greater detail in Section 4.5 where we present results of applying the crowd ensemble to this DQN architecture.

Experience replay [33] has become a prominent part of nearly all RL approaches as a result of the popularity of DQN. With experience replay, the agent stores all training samples in a memory. When it comes time to update parameters, training samples (experiences) are randomly selected from the memory. Experience replay is advertised as breaking the covariance of the training data and making the data more independent and, therefore, better aligned with the assumptions of most model parameter update algorithms. Experience replay has other benefits as well. It allows RL to squeeze additional information from a single time step by repeatedly revisiting it throughout training whenever it is selected for replay. It may also prevent forgetting to some degree because data from many parts of the state space are presented along with data from the most recent regions of the state space. Experience replay has obvious benefits and has begun to draw greater scrutiny [34, 35] and has inspired more sophisticated techniques for sampling experience [36]. In our experiments, we use the basic experience replay approach where the memory is represented by a queue which is randomly sampled to create a data set from which to compute the parameter updates.

2.5 Deep deterministic policy gradients

We use the deep deterministic policy gradient (DDPG) [4] RL approach in our bipedal walker and pendulum experiments. DDPG is an application of many of the DQN tricks of the trade to the actor-critic framework as well as the work of Silver et al. on deterministic policy gradients (DPG) [37]. The deterministic policy gradient work is based upon the convergence proofs for stochastic policy gradient by Sutton et al. [38]. The practical contribution of DPG is that practitioners may now be freed from having to sample the Q-function for a policy gradient; instead they can simply use the observed state/action pairs of the current policy to generate the gradient. Policy gradient algorithms are a part of the policy iteration class of algorithms [28]. A policy iteration algorithm alternates between policy evaluation and policy improvement. Policy evaluation updates the estimated value of states given the current policy, $Q^{\mu}(s)$. Policy improvement, updates the policy $\mu(s)$ to maximize reward given the state/action value estimates. In DDPG $Q^{\mu}(s)$ is modeled using an ANN trained using Q-learning.

In DPG an actor function, $\mu(s|\theta^{\mu})$, specifies the agent's policy by deterministically mapping states to specific actions. Parameter updates are performed first for the critic and then for the actor. The critic is learned using Q-learning. DPG and DDPG actor updates are performed by following the gradient of the reward:

$$\theta^{\mu}(t+1) \leftarrow \theta^{\mu}(t) + \alpha E_{s\sim p^{\mu(t)}}[\nabla_{\theta^{\mu}} Q^{\mu}(s, \mu(s|\theta^{\mu}))]$$

where the $E_{s\sim p^{\mu(t)}}$ notation denotes an expectation over the states visited while obeying the policy $\mu(t)$ which is the policy (actor network output) at the current time step. The α parameter is the learning rate. The value $Q^{\mu}(s, \mu(s|\theta^{\mu}))$ can be computed by passing the observed state, s, and action, $\mu(s|\theta^{\mu})$, through the critic and taking the output. The value $\nabla_{\theta^{\mu}} Q^{\mu}(s, \mu(s|\theta^{\mu}))$ is then found by back-propagating this value through the actor [4, 37, 38].

2.6 Ensemble learning for RL

Ensembles have been proposed for use with RL although none have found widespread use. In this section we begin with MoE-style RL ensembles which have mechanisms to encourage ensemble members (experts) to specialize in specific state space regions and to intelligently select a single expert or to mix the expert outputs. We then discuss work similar to the crowd ensemble where ensemble members are trained in parallel. Finally we look at recent RL approaches which are not presented as ensemble methods, e. g. asynchronous Q-learning and double Q-learning, but have elements of a crowd ensemble and whose benefits may be explained by the same analysis we do here for the crowd ensemble.

A Gaussian mixture model approach is utilized as the ensemble mechanism by Agostini and Celaya [39]. Their approach is a form of fitted value iteration which means training is performed using a database of stored experiences. This gives each ensemble member a region of expertise defined by a position in the state space and a variance which determines the size of the members' regions. Their approach relies on heuristics to add and remove experts during training. At each time step the active member is determined by each experts' relevance function. They test their approach on the pendulum swing-up and cart-pole balancing tasks and compare against a non-ensemble version of their approach. They find that a collection of approximators is better than a single approximator with respect to accumulated reward.

A couple MoE-like approaches are compared in a limited set of experiments on the pole-balancing task where the goal is to keep the pole from dropping out of the goal position [40]. The ensemble members are ANNs trained using fitted Q-iteration which is a batch update of Q-learning very similar method used to training the individual ensemble members here [41]. They conclude that a hard combination of ensemble members, that is only one expert is active at a time, is superior to the soft-combination of experts they initially attempted.

A straight-forward mixture of experts similar to that shown in Figure 2.1 is utilized on the pole-balancing task in Anderson and Hong [42]. They found that allowing the MoE to learn its

own partitioning of the state space led to improved results than either using a single ANN as the function approximator or performing the partitioning by hand in advance of learning.

The multiple model reinforcement learning (MMRL) [43] approach is a part of a line of research on continuous RL. In MMRL each expert has a forward model and a Q-function. The forward model predicts the change in state, $\dot{s}(t+1) = s(t-1) - s(t)$, given the current state/action pair. MMRL is a MoE approach where each expert receives a soft weight based upon the accuracy of the experts' forward models. The weights are called responsibility values and are computed using a Gaussian softmax of the experts' forward model errors:

$$\lambda_i(t) = \frac{e^{-\frac{1}{2\sigma^2}\|\dot{s}(t)-\hat{\dot{s}}_i(t)\|^2}}{\sum_{j=1}^{N_e} e^{-\frac{1}{2\sigma^2}\|\dot{s}(t)-\hat{\dot{s}}_j(t)\|^2}} \tag{2.8}$$

where $\hat{\dot{s}}_i(t)$ is expert i's prediction of the state space change from time $t-1$ to t and σ is a configurable parameter determining the variance of the Gaussian.

Not only do the responsibility values determine how the experts' value functions are combined but they also drive how experts' parameters are updated. This encourages specialization of the experts to separate regions of the state space. The ensemble forward model prediction error, $\delta^{(f)}$, and the TD-error, $\delta^{(TD)}$ are weighted according to λ_i values:

$$\delta_i(t) = \lambda_i(t)\delta(t)$$

where $\delta(t)$ is the error value (either $\delta^{(TD)}$ or δ^{f}).

The action selection is a weighted combination of the experts' outputs. In the implementation used here, the Q-value outputs are combined using:

$$Q(s(t), a) = \sum_{c=1}^{N_e} \lambda_c(t) Q_c(s(t), a)$$

and the action is selected in the usual way:

$$a(t) \leftarrow \arg\max_{a \in \mathscr{A}} Q(s(t), a).$$

In [43] MMRL is tested on a predator-prey chase simulation (discrete state-action space) and the non-stationary pendulum swing-up (pendulum length and weight change every trial, continuous state-action space) tasks. For these experiments they also introduce a reward model to provide additional, goal-related information to the agent which is a necessary requirement of the parameter-update scheme they chose to use. The dynamics are modeled using a linear model and the reward and value functions are modeled using quadratic functions.

In these experiments two heuristics were added which act as priors which are multiplied against (2.8). One is the temporal continuity prior which smoothes-out the responsibility values between time steps. The second is the spatial locality prior which further localizes the experts to a particular region of the state space.

In the discrete task, MMRL is compared to a single Q-learner and is found to catch the prey more quickly than a non-ensemble agent. In the pendulum task MMRL is compared against an actor-critic agent and they show that MMRL with $N_e \in \{2, 4, 8\}$ solves the problem more quickly. They also show that MMRL successfully switches between experts as the dynamics are switched during evaluation.

In [44] a concept very similar to the crowd ensemble method explored in this dissertation is proposed. Their approach has an important difference: the ensemble members are trained in parallel with each other, each with a different action selection policy. This results in N_e separate training sessions and a factor of N_e more interactions with the training environment to train the ensemble. For example, in their majority voting policy, actions are selected during training according to:

$$\pi_m(s_t) = \arg\max_{a \in \mathscr{A}} \sum_{c=1}^{N_e} \hat{w}_m(c) N_c(s_t, a_t)$$

where $\hat{w}_m(l = m) = \lceil \frac{N_e - 1}{2} \rceil + 0.01$ and $\hat{w}_m(l \neq m) = 1$ during training. N_C is the action selected by ensemble member c. This heuristic policy is a weighted combination designed to favor the action selected by expert m while still taking into account the votes of the other $N_e - 1$

ensemble members. Their results show that their approach with $N_e > 3$ performs better than a single Q-learner on a maze navigation task and a simplified tetris task (SZ-tetris) with a reduced set of pieces. They did not test $N_e > 10$, presumably because of the high computational cost of adding each ensemble member. The approach presented in [44] is compared to the crowd ensemble approach and the results are presented in Section 4.4.

In [45] the work in [44] is extended to introduce the concept of selective neural network ensembles for RL is introduced. Selective neural network ensembles for RL are designed to take the *creme de la creme* of the ensemble members to squeeze more performance out of the ensemble. This is done by training an ensemble like in [44] but adding an additional training step which optimizes a quadratic function which determines the weighted combination of only a subset of all N_e ensemble members. They compare the selected ANN ensemble approach against the method developed in [44] on the same maze navigation and SZ-tetris tasks and find that the selective ensemble method performed better. They also examined larger values of N_e in [45] and found that selective ensembles with much fewer members out-performed larger crowd ensembles from [44] (e. g. $N_e = 50$ compared to $N_e \in [15, 30]$).

In [46], a very brief paper, Huang et al. introduce the actor-critic ensemble (ACE). Their approach won second place at the NIPS 2018 learning to run competition. The task is to get a skeleton to travel as far as possible in 1000 time steps.

The authors report that they were motivated by the observation that the actors were frequently leading to states with exceedingly high probabilities of failure. On the other hand, they observed that the critic Q-functions typically accurately identified such actions.

The ACE algorithm trains N_e DDPG actor-critic agents separately. During evaluation all N_e action decisions are computed from the ensemble member policies and a critic network is used to score the actions using its Q-function. The evaluation critic is the mean of an ensemble of Q-functions. It is unclear from their description if the critic ensemble members are the critics used during training or if they are generated through some other process. The authors state that ACE

with ten actors and ten critics (other results for one actor and no critic and ten actors and one critic ensembles were also provided) results in better performance.

In [47] a number of ensemble approaches were compared which combine multiple RL approaches with the goal of improving learning speed and evaluated performance. They combine the ensemble members using majority voting, a rank voting scheme which allows each ensemble member to weight actions, and a couple other more complex methods based on each ensemble member assigning Boltzmann probabilities to actions. The candidate RL approaches available to the ensemble members are actor-critic and two variants of the actor-critic approach: QV-learning [48] and ACLA [49]. The voting and Boltzmann combination techniques are applicable to discrete action tasks with action sets small enough that ensemble members can choose the same actions. They apply the investigated approaches to various maze environments including dynamic obstacles and goals and partially observable states. They observe that the Boltzmann and voting combination schemes significantly outperform the other schemes and that the ensemble methods always do as well or better than non-ensemble methods. They also find that an ensemble consisting of multiple RL approaches is better than an ensemble consisting of only the RL approach which performed best when applied in isolation.

2.6.1 Recent ensemble-like approaches

Recently Q-learning approaches have been developed which, despite the use of two or more Q-functions, resemble ensemble methods and may share some of the same benefits. These approaches are not ensembles because they do not have a mechanism which combines ensemble member outputs.

Mnih et al. [5] and Nair et al. [50] present a RL approach which uses multiple, simultaneous simulations to speed-up DQN. In [5] this method is called asynchronous DQN. They begin with a shared Q-function parameterized by θ and a shared target Q-function parameterized by θ^{-}. True to its name, the θ updates are not synchronized allowing each simulation to periodically update θ based upon its accumulated gradient over several time steps. They are motivated by the hypoth-

esis that multiple simulations will explore multiple points of the state space simultaneously. In favor of this they do not use experience replay. They add further variety to the experiences of the individual simulations by applying a different $\epsilon -$ greedy ϵ value to each ensemble member by occasionally randomly sampling ϵ for each simulation. Just as with DQN the shared target network is periodically updated from the shared θ.

Asynchronous DQN is tested using a large number of arcade learning environment [51] tasks, a racing simulator with continuous actions, and a maze task. They compare the performance of their approach to several recent Q-learning approaches sing an average of five runs and plotting the performance in terms of hours of wall-clock time. They find that the asynchronous updates learn faster. A brief analysis of the performance of asynchronous DQN indicates that this method, once good learning rates are found, learn reliably with little issues with non-learning agents (local optima from poor initial values) or divergence after a long period of learning. We find similar benefits with the crowd ensemble approach. They do not share a view of their results plotted against the number of total simulated time steps which is the emphasis of the comparison in this dissertation.

Double Q-learning is designed to address the issue of over estimation of Q-values due to the use of the max operator in (2.5) [52]. This is done by using two Q-functions which we will label A and B. The update for Q-function A becomes:

$$Q^A(s(t), a(t)) \leftarrow \alpha[r(t) + \gamma Q^B(s(t+1), \arg\max_a Q^A(s(t+1), a)) - Q^A(s(t), a(t))]$$

and visa versa for Q-function B.

They show that any error in state observation or action estimate will result in an over-estimation of the Q-function [52]. They show that these sort of errors accumulate into severe overestimation on the DQN-style arcade learning environment tasks [51] where states are represented using pixel data. They are able to show that double Q-learning reduces the over-estimations and results in improved performance when allowed to run for many millions of time steps. During training one Q-function is used to determine the policy while the other is use to determine its value. Each

experience is randomly assigned to either network A or B. During evaluation, one of the two networks is always used for action selection. The double Q-learning work does, however, address a potential benefit of ensemble methods like the one presented here where the actions are partially decoupled from the parameter update mechanism. In a later work the double Q-learning concept is adapted to the DQN framework [53].

Duryea et al. take the double Q-learning concept a step further by investigating using any number of Q-learners which they call multi Q-learning [54]. This is done by randomly selecting one of N_e Q-functions during training to drive greedy action selection and then update the function according to:

$$Q^A(s(t),a(t)) \leftarrow \alpha[r(t)+\gamma\frac{1}{n-1}\sum_{B=1,B\neq A}^{N_e} Q^B(s(t+1),\arg\max_a Q^A(s(t+1),a))-Q^A(s(t),a(t))]$$

where Q^A is the selected Q-function. This updates $Q^A(s(t),a(t))$ using the average, maximum Q-value of the next state of the other Q-functions.

Multi Q-learning was tested on a grid world problem using an ANN to model the Q-functions and they show that using a larger (up to eight was tested) number of Q-functions yielded better average rewards compared to Q-learning and double Q-learning. Furthermore they found that multi Q-learning was less susceptible to a poor choice in ANN architecture: for their task Q-learning and double Q-learning suffered when the ANNs were unnecessarily increased from two to three hidden layers while multi Q-learning performance was not degraded.

2.7 Summary

Reinforcement learning has seen a tremendous number of new approaches and new test environments where it has been shown to achieve human-level (or better) performance on many challenging tasks. Principal among these is deep Q-learning (DQN) which has led to a number of variants such as asynchronous Q-learning and DDPG. DQN kindled this flurry of activity by showing that Q-learning using ANNs as function approximators is capable of solving challenging

tasks at human-level performance. It did this by training for a very large number of time steps and incorporating experience replay to maximize the update information from each training sample. It used specialized ANNs which have structures which are specialized for high-dimension input. DQN also includes target networks to stabilize the parameter updates allowing for steady, long-term progress.

DDPG is one such DQN-inspired modification of a pre-existing algorithm. DDPG extends the DQN approach to tasks with continuous actions utilizing an actor-critic approach. DDPG has been shown to be able to solve a number of tasks but its performance has not been deeply examined to see which aspects of DQN-style approaches are beneficial or if DDPG also suffers from catastrophic forgetting.

There has been little focus on addressing the instability of Q-learning agents during training nor has there been concern about the infeasibility of these approaches for real-world tasks where collecting millions of training samples or data from parallel simulations is unavailable. The DQN literature is focused on the best performance achieved during training with little concern for forgetting. The advance most similar to the crowd ensemble, asynchronous Q-learning, seeks to speed-up training in a simulated environment but it does so by multiplying the number of training samples needed making it infeasible for tasks with computationally intensive simulations and real-world systems. We seek to capitalize upon the ideas popularized or invented in the DQN literature but to do so with an emphasis on mitigating forgetting, reducing the number of training samples necessary, and looking at mean and median performance across a number of agents instead of the best performance possible.

Chapter 3

Baseline results

Ensemble methods were investigated using four problem domains. The cart-pole task, a classic control problem, is a personal implementation while the pendulum and biped problems were implemented as part of the OpenAI gym [55]. The cart-pole task has low-dimension and high-dimension variants; the second of which allows us to examine the effects of crowd ensembles using what is commonly referred to as deep Q-learning (DQN). The biped task is of moderate state and action dimension which strikes a nice balance between the two extreme state representations of the cart-pole tasks. The biped task uses continuously-valued, multi-dimension actions which provides us with the opportunity to examine the performance benefit of using a crowd ensemble when using an actor-critic RL approach. Because the biped task is difficult to analyze, the pendulum task is also included which is a lower-dimension task which also has continuously-valued actions. In this chapter these tasks will be investigated along with an examination of the state-of-the-art performance for these tasks which will serve as our baseline of comparison. We will gain some intuition about the challenges these tasks present to the RL agents.

3.1 The cart-pole task

The cart-pole task can be described specifically as the swing-up-and-balance cart-pole task. The purpose of this task is to start a trial with the agent's pole in the down position and allowing the agent to move the cart back and forth along a 2-D track in order to swing the pole up. Once the pole is up, the agent must then balance the pole. The track is of finite length and each end of the track has a wall with which the cart interacts via elastic collisions.

The cartpole task has four state variables: cart position, cart velocity, pole angle, and pole angular velocity: $s = [s_p, \dot{s}_p, s_a, \dot{s}_a]$. Possible cart positions are $s_p \in [-2.2, 2.2]$. The pole angle is defined as zero being pointing straight up and π (or $-\pi$) as pointing straight down with $s_a \in [-\pi, \pi]$. The actions are discrete with $a \in -1, 0, 1$ which translate to push left, no push, and push

right. The agent is rewarded a -1 when $s_a \in [\frac{-3\pi}{4}, -\pi] \cup [\frac{3\pi}{4}, \pi]$ (pointing downward), a $+1$ when $s_a \in [\frac{-\pi}{4}, \frac{\pi}{4}]$ (pointing upward), and zero elsewhere. Figure 3.1 shows a GUI representation of the task which allows a human to interact with the cart-pole environment.

During training two environments are used – each with the same parameters and dynamics. One environment is used to generate training data while the other is used for evaluation only. The training environment is initialized at the beginning of training and then runs continuously: it is never reset to the starting position. The evaluation environment, on the other hand, is reset to the initial position at the beginning of each evaluation episode.

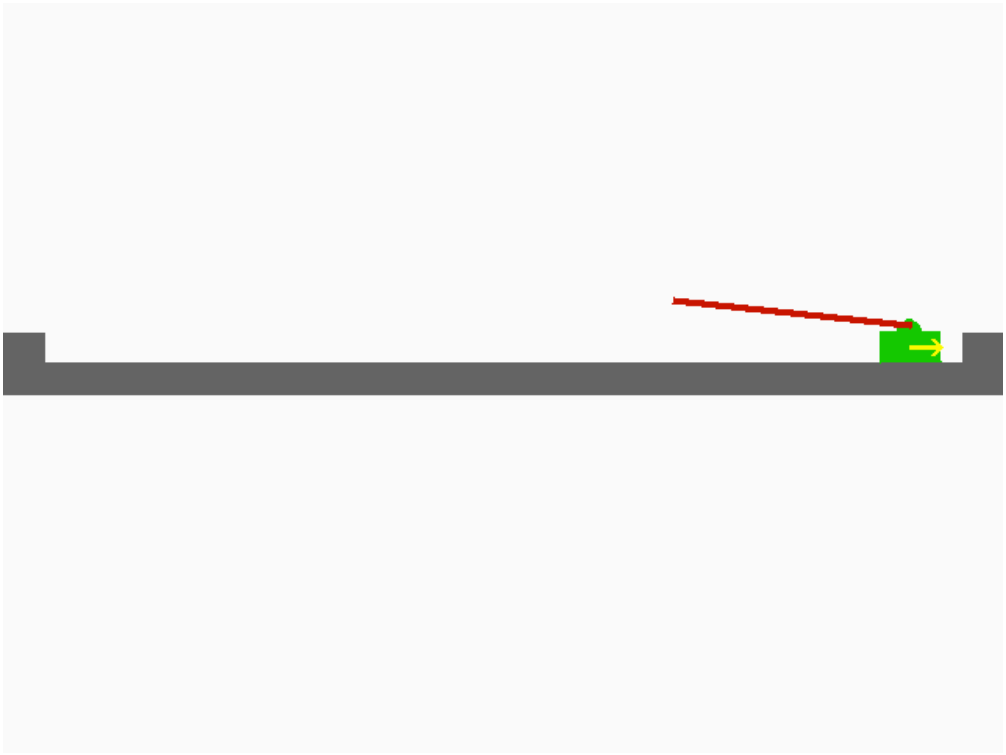


Figure 3.1: GUI representation of the cart-pole task.

3.2 State of the art Q-learning solution for the low-dimension variant

A modified experience replay is used in these experiments where all training episodes are stored in memory. During the parameter update phase of the algorithm, when experiences are drawn from memory to be replayed, entire training episodes are drawn instead of drawing randomly-selected time steps of randomly-selected training episodes as is typically done in experience replay. We do not feel that this difference is important to the results although it does weaken what is widely thought to be one of the core tenants of experience replay: decorrelation of training samples. We compared the performance of Q-learning on this task using both experience replay sampling choices and found no performance difference leading us to conclude that the most important aspect of experience replay, for this task, is the repeated exposure to a wide variety of state/action pairs.

Q-learning is the RL approach we apply to solve this task. ANNs with two hidden layers are used to model the Q-function. The Q-function ANN takes the four-dimension state and one-dimension action as input and outputs the corresponding $Q(s, a)$ value. The algorithm of the implementation is found in Algorithm listing 1.

Step 3 is the outer loop of training which counts the number of training time steps the agent has experienced while lines 6–17 perform the creation of a batch of training experiences. A value of $T = 100,000$, the total number of training time steps, is typically large enough to solve the task. We found $N_B = 1000$, the size of each training batch, to be a good value in practice.

Step 7 is the execution of the action selected in the previous iteration by executing that step and running the simulation two time steps into the future and then returning the new state. Lines 9–14 implement ϵ–greedy [28, p. 28] to encourage state exploration. At step 19 all N_B $(s(t_B), r(t_B), a(t_B))$ tuples are stored in memory for later experience replay.

Step 22 is the recomputation of the target value for the entire randomly-selected sequence selected in the previous step where step 23 is the TD-error used to compute the updates via back-propagation. Step 24 is where the parameter updates occur. SCG has a parameter determining

Algorithm 1: Pseudo-code of the baseline low-dimension cart-pole Q-learning implementation using a two-layer ANN to model the Q-function.

1 Initialize ANN weights;
2 $t \leftarrow 0$;
3 **while** $t < T$ **do**
4 Obtain initial $s(t_B = 0)$, $a(t_B = 0)$, $r(t_B = 0)$;
5 $t_B \leftarrow 1$;
6 **while** $t_B \leq N_B$ **do**
7 $s(t_B) \leftarrow$ `Act(`$a(t_B - 1)$`)`;
8 $r(t_B) \leftarrow$ `Reward(`$s(t_B)$`)`;
9 **if** `Random(0,1)` $> \epsilon$ **then**
10 Set $a(t_B)$ to random action;
11 **end**
12 **else**
13 $a(t_B) \leftarrow \arg\max_{a'} Q(s(t_B), a')$;
14 **end**
15 $S_b(t) \leftarrow \big(s(t_B), r(t_B), a(t_B)\big)$;
16 $t_B \leftarrow t_B + 1$;
17 $t \leftarrow t + 1$;
18 **end**
19 Store training sequence, S_b, of length N_B in memory;
20 **for** $t_R \in \{1 \ldots N_R\}$ **do**
21 Randomly draw a training episode sequence, S_u, from memory;
22 $T_V(t_B) \leftarrow r(t_B) + \gamma \max_a Q(s(t_B), a), \forall t_B \in \{1, \ldots, |S_u|\}$;
23 $\delta_o \leftarrow \sum_{t_B=1}^{|S_u|} T_V(t_B) - Q(s(t_B), a(t_B))$;
24 Perform 5 updates of ANN weights using SCG;
25 **end**
26 **if** $t \,\%\, evalFrequency = 0$ **then**
27 Evaluate by performing lines 6–17 with $\epsilon \leftarrow 0$ for 2000 steps.
28 **end**
29 **end**

the maximum number of parameter updates to apply which we set to five so SCG performs five updates for each of the N_R selected sequences.

The implementation can reach a cumulative reward of $R > 1950$ in as few as 20,000 time steps although many agents never find a solution with a reward that high. The cumulative reward is computed by summing the rewards across all 2000 time steps of an evaluation episode. Evaluation episodes are not counted against the number of training episodes and always start with the pole

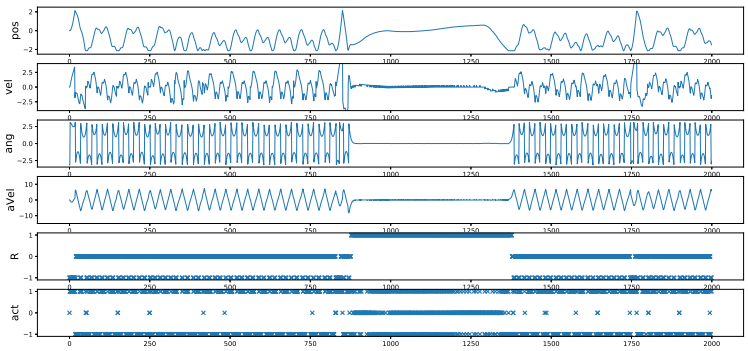
pointing down and the cart in the center of the track with zero cart and pole velocity which sets the ceiling on the maximum cumulative reward at less than 2000. Those unsuccessful agents typically get stuck in a very poor local optima early in training and receive evaluation rewards around $R = -1800$ for the entirety of training – even after $1,000,000$ time steps have past. Figure 3.2 shows three example solutions of the task collected while training a single agent.

The agent whose result is shown in Figure 3.2a is capable of swinging the pole up for nearly 400 time steps around $t = 1000$. The repeated pattern seen in the pole angle plot (third from the top) is the action of the pole passing through the pointed-down position where it transitions from π to $-\pi$. The parabolic-looking action of the pole angle plot is the pole swinging up (somewhat slowly), slowing down, and eventually reversing direction. Also evident in the cart position and cart velocity plots of Figure 3.2a is that, prior to briefly balancing the pole, the cart is slammed against a wall which caused the pole to swing up. From there the agent is able to balance the pole by quickly alternating -1, 0, and $+1$ actions.

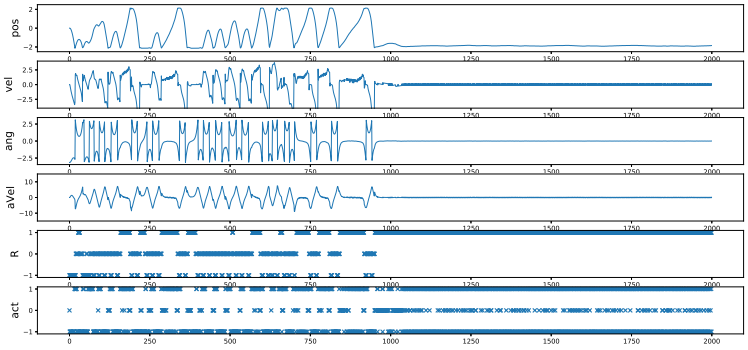
The evaluation shown in Figure 3.2b shows that the agent is able to swing-up the pole into goal state and keep it there for the remainder of the evaluation which was over 1000 time steps. The agent was able to swing the pole up into the goal position several times before balancing it as can be seen by the larger parabolic shapes of the pole angle plot but it was unable to keep the pole there.

The evaluation shown in Figure 3.2c shows an agent which is able to expertly swing the pole up quickly and hold it there for the remainder of the evaluation. The agent used the wall to quickly swing the pole up into the goal position but acted immediately to hold the pole there and stop the angular velocity. It then carefully maneuvers itself toward the center of the track where it has more room to react to changes in pole state which is the highest Q-value region of the space.

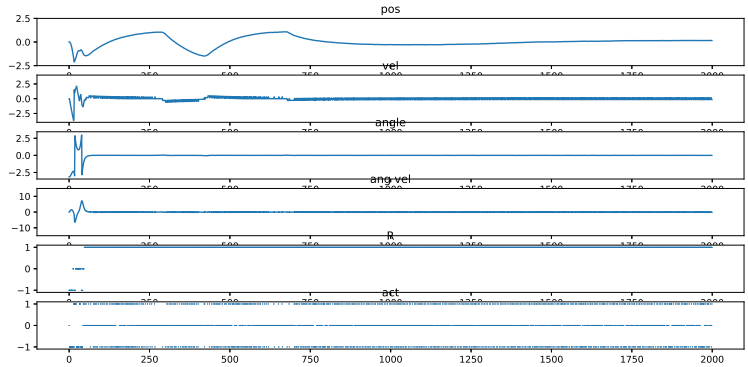
Figure 3.3 shows the Q-values across all four dimensions of the state space. This figure was created by sampling the four-dimension state space at the cross product of five cart positions and five cart velocities. At each $(s_p, \dot{s}_p)$ pair the space is sampled at 25 locations in each of the s_a and $\dot{s}_a$ dimensions. For each of the sampled $s = (s_p, \dot{s}_p, s_a, \dot{s}_a)$ locations the Q-function value



(a) $R = 111$



(b) $R = 1190$



(c) $R = 1932$

Figure 3.2: Three example solutions recorded while training a single agent on the low-dimension cartpole task. Each solution is described by six figures. The first four show the state positions during evaluation. The bottom two are the reward received at each time step, and the action selected at each time step.

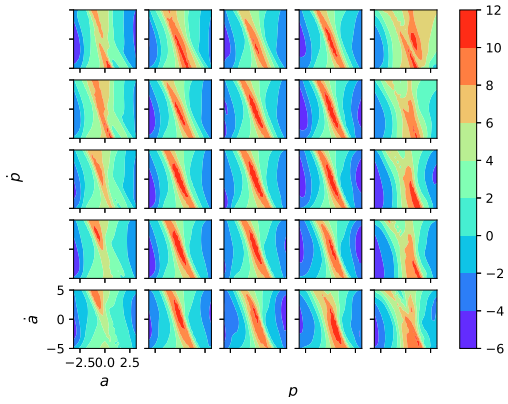


Figure 3.3: Q-values of an example cart-pole task solution learned using Q-learning. Each plot shows the best Q-function value across 25×25 pole angle, a, (x-axes) and pole velocity, $\dot{a}$, (y-axes) points for a specific cart position, p, with zero cart velocity, $\dot{p}$. The grid of plots covers the range of cart positions (rows) and cart velocities (columns) where the cart position and velocity is held constant for each plot.

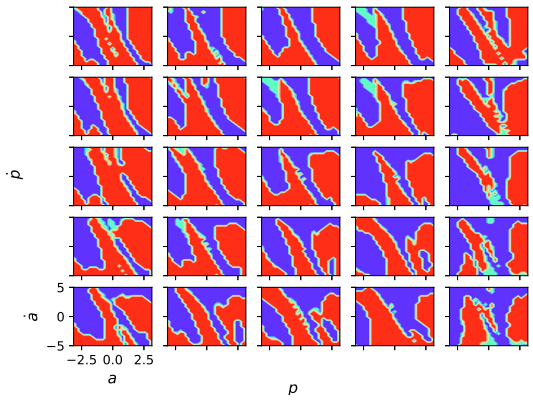


Figure 3.4: Decision surface of an example cart-pole task solution learned using Q-learning. Each plot shows the best action across 25×25 pole angle (x-axes) and pole velocity (y-axes). The grid of plots covers the range of cart positions (rows) and cart velocities (columns) where the cart position and velocity is held constant for each plot. There are three colors in these plots for the three possible actions: blue is $+1$, red is -1, and green is 0.

associated with the selected action is computed using $\max_{a \in \mathscr{A}} Q(s, a)$. Figure 3.4 is computed similarly using the same sampled state space locations but displaying $\arg\max_{a \in \mathscr{A}} Q(s, a)$: the action with the highest Q-function value at location s.

Looking at the center plot of Figure 3.3 which corresponds to cart position in the center of the track with no velocity, we see a uni-modal-looking function which, as expected, has a maximum centered over the goal region. This maximum is rotated to give preference to positive pole rotations

when the pole is $s_a < 0$ and to negative pole rotations when the pole is $s_a > 0$. This distinctive rotated region of elevated Q-function values disintegrates near the boundaries of the track because of the effects of contact with the wall.

Figure 3.4 shows the action selection at each of the selected state space points. There is a decision boundary that runs through the center of the region of high Q-function values. Figure 3.5 shows the center plots from Figures 3.4 and 3.3 where the Q-function surface is converted to a transparent contour plot and overlayed on top of the action-selection plot. This plot shows how the

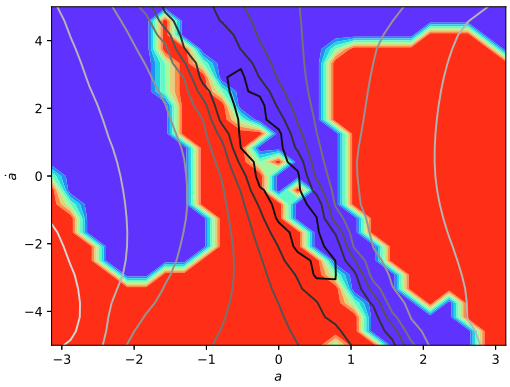
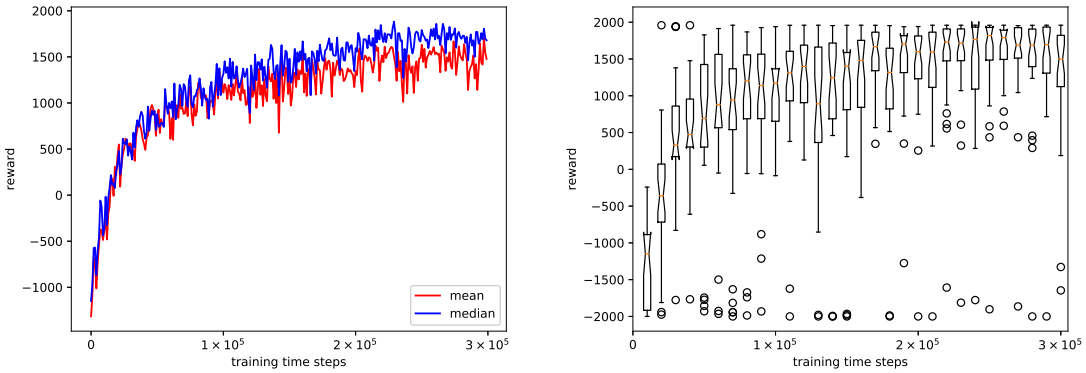


Figure 3.5: Center plots from Figures 3.4 and 3.3 where the Q-function is converted to an unfilled contour plot and overlayed on top of the action-selection plot. Red represents push left, green is no action, and blue is push right.

Q-function and policy combine to keep the pole in the high Q-value regions of the state space.

Figure 3.6 shows the results for the baseline Q-learning approach. Thirty agents were trained on this task for $300,000$ time steps and were evaluated every 1000 time steps which is the length of each training episode. Each evaluation was 2000 time steps in duration. Figure 3.6a shows the mean and median evaluated rewards. The maximum reward received during evaluation was 1960 and the smallest was -2000 and 29 of the 30 agents solved the task where solved was defined as achieving an evaluation reward of $R > 1950$ at any point during training. The median reward can be thought of as a typical agent's evaluation at a given moment in training. It does not reflect the volatility of the solutions which is better captured in the mean reward value. Although there are



(a) Mean and median rewards.

(b) Box plot of all rewards.

Figure 3.6: Baseline Q-learning results on the low-dimension cart-pole task.

outliers, the nearness of the mean and median indicates that the outliers have not seriously dragged the mean reward down.

Figure 3.6b shows a box-plot of the evaluation rewards during training. One out of every ten evaluations are shown to make room for the box plot boxes and whiskers. The boxes represent the interquartile range of the evaluations at that particular point in training. The whiskers show the range of the data that lie within two times the interquartile range. The yellow line near the middle of each box is the median reward at that point. The unfilled circles are the data points which have been identified as outliers. The size of the whiskers indicate that there is a large amount of variance in the performance of the agents at any given point in training. After $30,000$ time steps at least one of the agents has solved the task at each evaluation according to our definition. Likewise, for most of the evaluations shown here, one or more agents has suffered from catastrophic forgetting and received an evaluation nearing -2000.

When exploring implementation options for this task (e. g., meta-parameters and ANN architectures) we also attempted to utilize ADAM to update the ANN parameters. We performed a grid search over the number of experience replays during optimization, the size of the experience replay batches, and the ADAM learning rate. Figure 3.7 shows the performance using the best meta-parameter settings for ADAM against the SCG performance from Figure 3.6. SCG is better than ADAM for this task. This is notable because the best parameter combination for ADAM on

this task used batches of size 100 which meant ADAM updates were performed 10 times more often than the SCG updates. Despite the additional parameter updates, utilizing ADAM results in a mean performance worse than when using SCG.

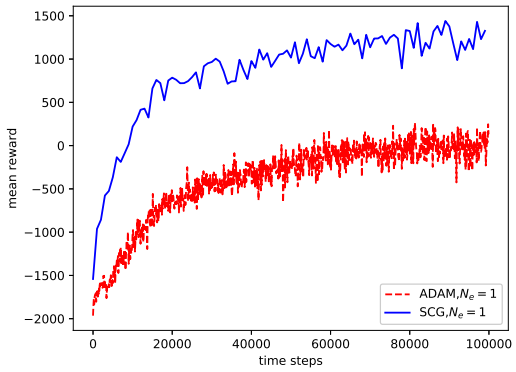


Figure 3.7: Comparison of mean reward during evaluation for ADAM and SCG on 50 reruns of the low-dimension cart-pole task.

3.3 Deep Q-learning on the cart-pole swing-up task

Most work on a cart-pole system when using an image-based state representation utilize a simpler form of the task where the pole begins in an upward position and the task is to balance the pole. Furthermore the image-based state representation provided to the agent is typically pre-processed by cropping the image by putting the cart and pole in the center and excluding the rest of the track. Our experiments avoided assisting the agent by cropping the image. Initially, most of the DQN literature has focused on tasks like the Atari benchmark problems discussed previously. By comparison, the cart-pole swing up task presents some unique challenges. The foremost among these is that the problem is unstable: relatively small deviations from a successful strategy can lead the agent into much different parts of the state space which require a much different strategy to escape.

The meta-parameters associated with a DQN are the number of layers, the number of parameters at each layer, the size of the convolutional layers' input windows, the convolutional layers'

window stride values, the activation functions, the size of the experience replay buffer, the learning rate, and the size and number of batches recalled during the parameter update phases of training. In the convolutional layers the individual nodes are referred to and thought of, literally, as features.

In these experiments we use a DQN with two convolutional hidden layers followed by two fully-connected layers. ReLU activation functions were used in the convolutional layers and tanh was used in the fully-connected layers. Batch normalization after the convolutional layers was not used. Experiments with DQN often uses target networks to stabilize learning. We found no learning speed-up on this task when using target networks so we do not include them in our implementation.

DQN was initially proposed to use an ANN with only states as input and an output for each action and this has become the defacto method for DQN. We use this architecture in these experiments. These ANN architecture decisions were made using a manual search and were found to perform well.

Algorithm 2 shows the pseudo-code for the DQN implementation used to solve this task. Algorithm 2 is very similar to Algorithm 1. The biggest difference is step 22 which passes the error back through the correct output while setting the other output errors to zero.

Prior to being input into the DQN, the state image is pre-processed. The original size of the image is 640×480 which is reduced to a 140×52 image by cropping off the ends of the image that extend out further than the maximum pole position on each side. The image is not scaled but is simply re-drawn with this smaller resolution. Also, the track is not drawn leaving only the cart and pole in the image. The background was changed from off-white to white so its grayscale value will correspond to zero. The cart is drawn as grey and the pole is drawn as black. The reduced state image resolution made the pole difficult to see at certain angles so the pole thickness was doubled when drawn. Finally, this image is vectorized and concatenated with a pre-processed image from the previous frame for an input of size 7280×2. Figure 3.8 shows two example frames after pre-processing which combine to make a state. The visual difference of these two frames is small when the cart and pole velocities are small. The cart and pole velocities are slower in these images

Algorithm 2: Pseudo-code of the baseline DQN implementation.

1 Initialize DQN weights;
2 Initialize cart-pole environment;
3 $t \leftarrow 0$;
4 **while** $t < T$ **do**
5 $t_B \leftarrow 0$;
6 **while** $t_B \leq N_B$ **do**
7 $s(t_B) \leftarrow \texttt{Act}\,(a(t_B - 1))$;
8 $r(t_B) \leftarrow \texttt{Reward}\,(s(t_B))$;
9 **if** `Random(0,1)` $> \epsilon$ **then**
10 Set $a(t_B)$ to random action;
11 **end**
12 **else**
13 $a(t_B) \leftarrow \arg\max_{a'} Q(s(t_B), a')$;
14 **end**
15 Store $s(t_B)$, $a(t_B)$, and $r(t_B)$ in memory;
16 $t_B \leftarrow t_B + 1$;
17 $t \leftarrow t + 1$;
18 **end**
19 **for** $t_R \in \{1 \dots N_R\}$ **do**
20 Sample N_B states, actions, and rewards from memory and store in S_u;
21 $T_V(t_B) \leftarrow r(t_B) + \gamma \times \max_a Q(s(t_B), a) \forall t_B \in \{1, \dots, |S_u|\}$;
22 $\delta_{a,o} \leftarrow \begin{cases} \sum_{t_B=1}^{|S_u|} T_V(t_B) - Q(s(t_B), a(t_B)) & \text{if } a = a(t_B) \\ 0 & \text{otherwise} \end{cases}$;
23 Update all DQN weights using ADAM;
24 **end**
25 **if** $t \,\%\, evalFrequency$ **then**
26 Evaluate by performing lines 5–18 and setting $\epsilon \leftarrow 0$.
27 **end**
28 **end**

because the cart is attempting to keep the pole in balance. The cart in the image is 5×15 pixels in size. The pole is 26 pixels in length.

Because of the intense computational requirements to perform deep Q-learning, a limited, grid search of the parameter space was done to find the parameters which would serve as representative of baseline DQN performance. Although several parameter combinations were discovered which led to nearly-solved solutions, an architecture with N_{F_1} = 20 features in the first convolutional

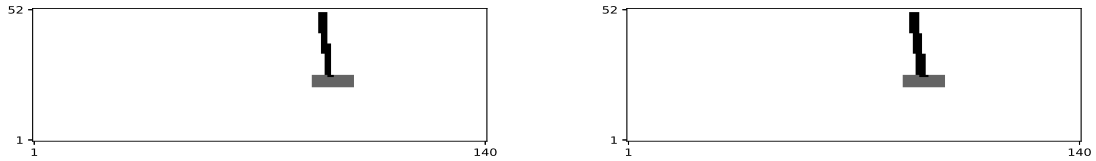


Figure 3.8: Two pre-processed frames which will combine to make a state for the DQN cart-pole agent. Frames are time t and $t + 1$ respectively.

layer and N_{F_2} = 40 features in the second convolutional layer was found to do so with regularity. The window size of the first convolutional layer is 6×6 and the second is 4×4. The strides for both layers is 2×2 meaning a new window starts every two pixels in both directions leading to overlap of the windows. The fully-connected layers are of size 100 and 20, respectively. During parameter updates the parameters are updated 25 times each using 100 sampled experience replay memories. This is implemented in lines 19–24 of Algorithm 2 with $N_R = 25$ and $N_B = 100$.

We were unable to find the same degree of success using SCG on the high-dimension variant of the cart-pole task as we did on the low-dimension version. Therefore we turned to ADAM to update the DQN parameters. ADAM [21] has become the defacto parameter update algorithm for DQN. The default ADAM parameters of $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 1 \times 10^{-8}$, and decay=0 were used during training. The learning rate was set lower than the default value with $\alpha = 0.0001$.

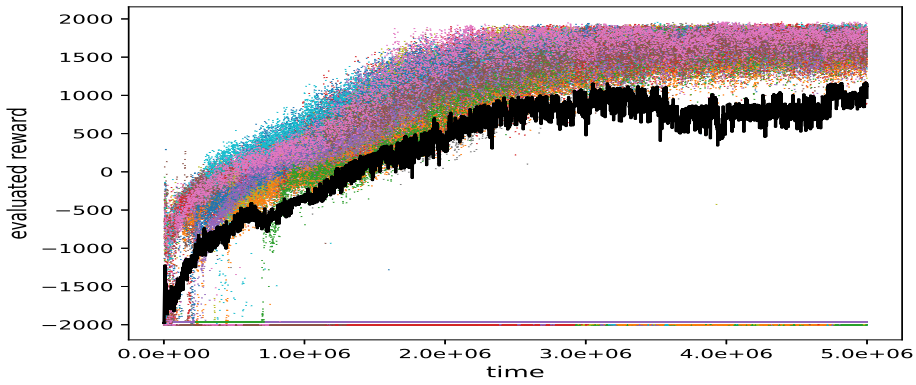
Figure 3.9 shows the evaluated reward of 25 agents on the cart-pole swing-up task using this DQN. A plot of all evaluations is shown in Figure 3.9a and Figure 3.9b summarizes those results using box plots. Each agent was allowed to train for 5×10^6 time steps. The maximum observed evaluation reward was 1960, the exact same as the low-dimension cart-pole task, and the minimum was -2000.

The black line in Figure 3.9a is the mean reward at each time step which is computed by averaging the evaluation reward over all 27 runs of the algorithm. The mean is pulled down by the instability of the learned solutions. Across the x-axis, along the $R = -2000$ line, the worst

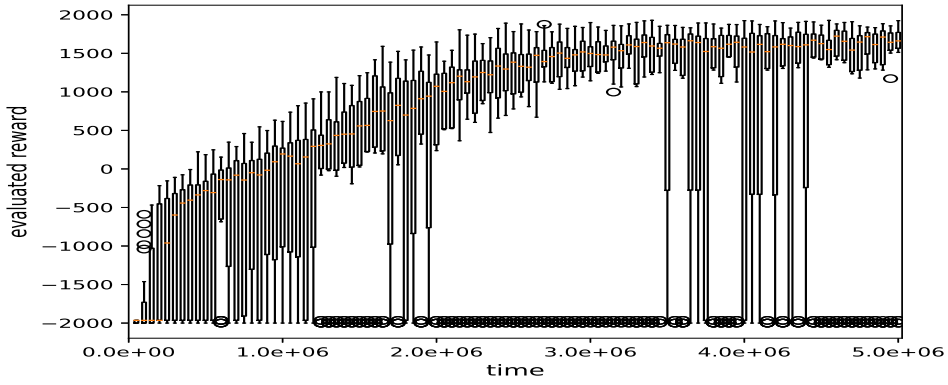
possible evaluation, a steady line of occasional very poor evaluations is visible. These poorly performing evaluations are spread across the 27 runs. There are a few evaluations between the two dense regions but they are difficult to make out due to the small size of the markers. As learning progresses, the $R = -2000$ evaluations become outliers, represented by circles in the box plot, except for around the $t = 4 \times 10^6$ mark when they become numerous enough to be included in the whiskers of the box plot. The boxes in the box plot represent the interquartile range which is the range of the data within the first and third quartiles. The whiskers show the range of the minimum and maximum values which are within three times the interquartile range. The circles indicate outliers. The cause of this dip is unknown and is probably random.

Figure 3.10 shows the evaluated reward for two randomly-selected agents from Figure 3.9. One of the agents, cyan line, shows forgetting but none of the catastrophic variety while the red line shows numerous instances of catastrophic forgetting. The cyan line's volatility is significantly smaller than the red plot but it still varies between $[1203, 1917]$ in the last 1×10^6 time steps. The red plot varies between $[-2000, 1779]$ during that same period. Although the variation of the cyan line seems small, Figure 3.11 shows the difference between the two solutions with those evaluated rewards. A solution of $R = 1203$ spends significantly less time in the goal region and is a much different solution. Although not a catastrophically bad solution, the loss of performance is large enough to be disappointing.

According to the DQN literature, the convolutional layers are supposed to play a role similar to that of the human visual cortex [1]. They are intended to learn low-level visual features useful as inputs to the fully-connected layers. Figure 3.12 shows the visual features of the first convolutional layer of the agent's DQN after 5×10^6 time steps of training. The features are drawn in color to highlight the range of positive and negative values of each feature. The colors are not normalized across images. Each feature of the first convolutional layer consists of two 6×6 masks: one for each input frame that make up the agent's state. Both masks are plotted for each feature with a black line separating the two masks.



(a) All evaluations



(b) Box plot of selected evaluations

Figure 3.9: Evaluated reward of 27 independent agents on the cart-pole swing-up task. In Figure 3.9a each agent's evaluated reward is plotted with a unique color. The dark line is the mean reward at each evaluation. In Figure 3.9b a box plot of the evaluated reward is drawn every 5×10^4 time steps The thin, yellow line running through the boxes is the median.

These features are not immediately interpretable but their output is helpful toward this end. Figures 3.13a and 3.13b show the output of two of the features of the first convolutional layer computed from the two frames shown in Figure 3.8. In these frames the pole is rotating clockwise. Feature one appears to highlight the direction of the cart and pole which are both rightward in this example. The bright white pixels of feature one of layer one appear to be the leading edge of the cart and pole indicating the direction. Feature 20 of the first layer may encode the opposite information.

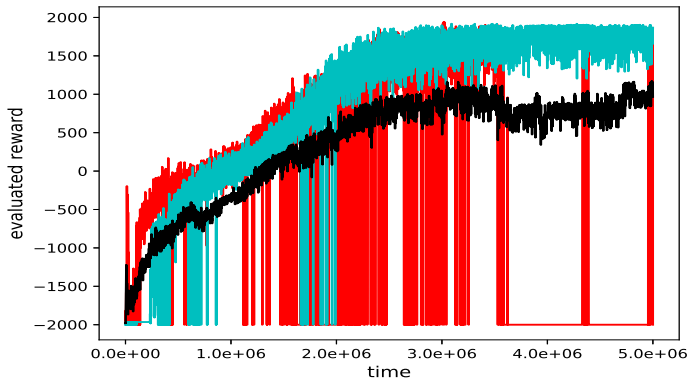
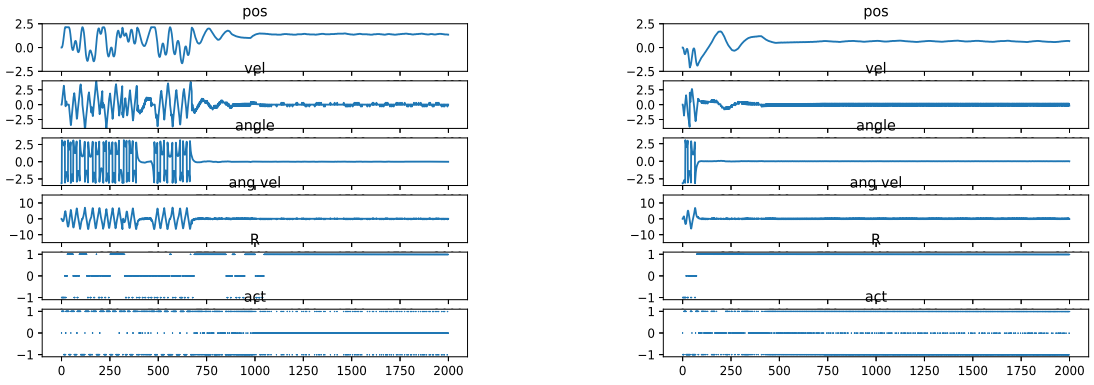


Figure 3.10: Two randomly-selected agents from Figure 3.9 plotted using red and cyan lines. The black line is the typical mean evaluated reward of the DQN approach.



(a) $R = 1200$ **(b)** $R = 1900$

Figure 3.11: Evaluations of two example solutions highlighting the difference in performance between a reward of 1200 and that of 1900.

Figures 3.13c and 3.13d show the outputs from two of the forty features of the second convolutional layer which were computed from the twenty features output by the first convolutional layer. Feature one appears to encode the location of the pole while feature 40 may encode the location of the cart.

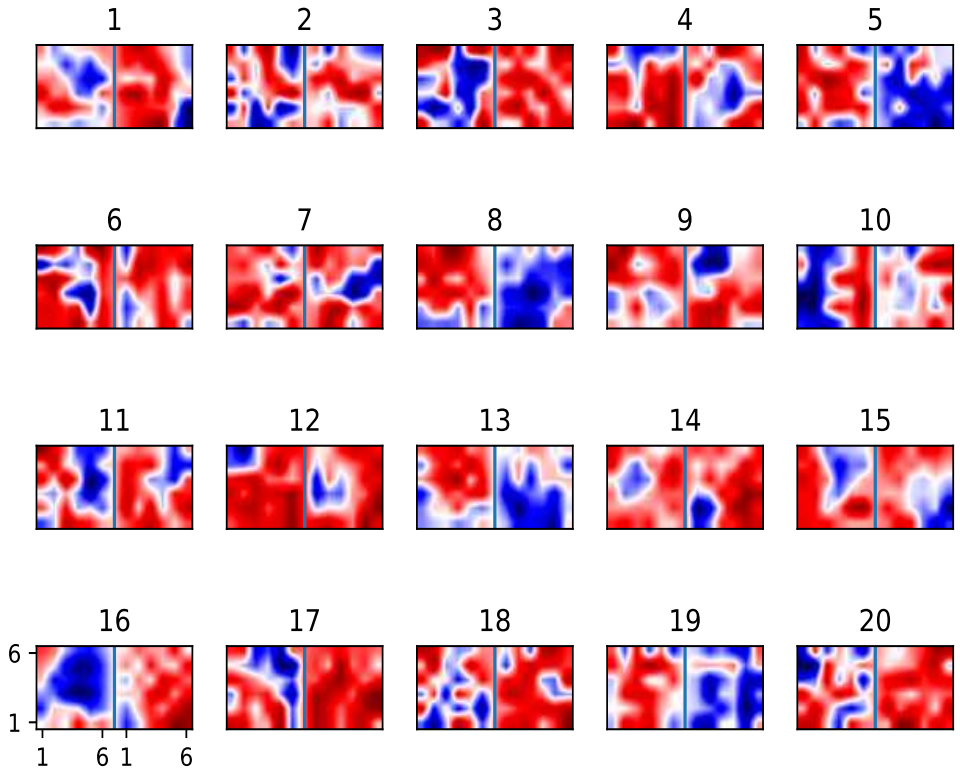
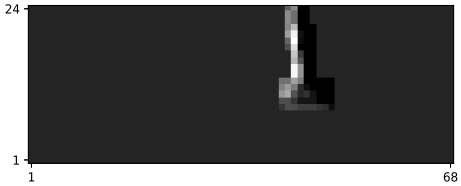


Figure 3.12: Features of the first convolutional layer of a DQN agent on the high-dimension cart-pole task trained for 1×10^6 time steps. Features are drawn using color with red intensity representing increasingly positive values and blue intensity showing increasingly negative values. Each of the twenty features is labeled and the axis tick labels show feature size. An extra column of black pixels was added to divide the feature image according which input frame the mask is applied to.

(a) Layer 1, feature 1



(b) Layer 1, feature 20

(c) Layer 2, feature 1

(d) Layer 2, feature 40

Figure 3.13: Features computed by the convolutional layers of an example DQN.

3.4 Biped walker task and the state of the art performance

The bipedal walker task objective is to train an agent to move a simple bipedal robot across a two-dimension, set-width, gently sloping, plane. We use the OpenAI [55] implementation of this task in these experiments. The action space is continuous in four dimensions: the actuations for a hip and knee in each leg of the robot. The robot consists of two legs and an oblong hull which sits on top of the legs. The state space is represented in 24 dimensions: hull angle, hull angular velocity, hull x velocity, hull y velocity, leg one hip angle, leg one hip speed, leg one knee angle, leg one knee velocity, leg one ground contact indicator (boolean), leg two hip angle, leg two hip speed, leg two knee angle, leg two knee velocity, leg two ground contact indicator (boolean), and ten lidar measurements measuring the distance to the ground from the center of the hull from ten different angles. The physics of the simulation were built using the box2D library [56].

The bipedal walker task, as defined by OpenAI, utilizes a reward function designed to encourage forward motion with limited wasted effort and a well-balanced hull. The reward at each time step whose sum is to be maximized is

$$r(t) = 130\frac{s_p(t)}{30} + -0.5s_h(t) + \sum_{a_i} -0.028|a_i(t)|,$$

where a_i is the ith action dimension, s_h is the hull angle, and s_p is the hull position along the plane ($s_p \in [0, 88.5]$). This encourages the agent to keep the hull from pointing downward which would make the biped less stable while also penalizing actuation of the hip and knee joint motors. Furthermore 100 is subtracted from $r(t)$ if the hull touches the ground indicating that the bipedal walker has fallen.

Algorithm listing 3 shows the baseline deep deterministic policy gradient (DDPG) algorithm implementation used in these experiments. This algorithm has much in common with Algorithms 1 and 2.

One noticeable difference is step 11 where the action is drawn from the policy ANN. During evaluation purely exploitive actions are drawn but during the other episodes exploration actions are

Algorithm 3: Pseudo-code of the baseline DDPG algorithm.

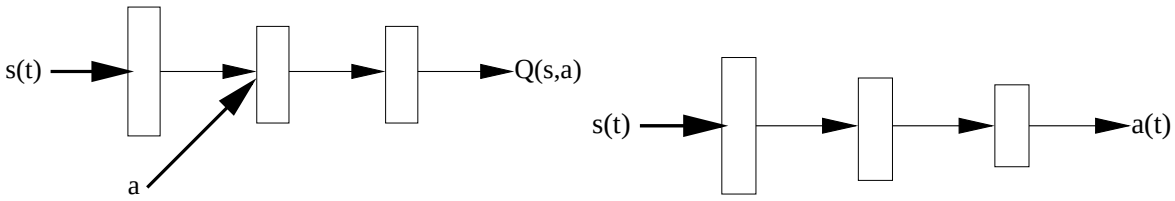
1 Initialize actor (θ^{μ}) and critic (θ^{Q}) parameters ;
2 Initialize target actor ($\theta^{\mu'}$) and target critic ($\theta^{Q'}$) parameters ;
3 Create an empty replay memory;
4 $t \leftarrow 0$;
5 **while** $t < T$ **do**
6 Obtain initial $s(t_B = 0)$, $a(t_B = 0)$, $r(t_B = 0)$;
7 $t_B \leftarrow 0$;
8 **while** $t_B \leq N_B$ **do**
9 $s(t_B) \leftarrow \texttt{Act}\,(a(t_B - 1))$;
10 $r(t_B) \leftarrow \texttt{Reward}\,(s(t_B))$;
11 $a(t_B) \leftarrow \begin{cases} \mu(s(t_B)|\theta^{\mu'}) & \text{if evaluation episode} \\ \mu(s(t_B)|\theta^{\mu'}) + \mathcal{N}(t, \mu = 0, \theta = 0.15, \sigma = 0.2) & \text{otherwise} \end{cases}$;
12 $s(t_B + 1) \leftarrow \texttt{Act}\,(a(t_B))$;
13 $r(t_B) \leftarrow \texttt{Reward}\,(s(t_B))$;
14 Store $s(t_B)$, $a(t_B)$, $r(t_B)$, and $s(t_B + 1)$ in memory;
15 $t_B \leftarrow t_B + 1$;
16 $t \leftarrow t + 1$;
17 **end**
18 Randomly draw a set of transitions, $(s(t_B), a(t_B), r(t_B), s(t_B + 1))$ from memory and store in S_u;
19 $T_V(t_B) \leftarrow r(t_B) + \gamma \times \max_a Q(s(t_B), a|\theta^{Q'}), \forall t_B \in \{1, \ldots, |S_u|\}$;
20 $\delta_o^Q \leftarrow \sum_{t_B=1}^{|S_u|} T_V(t_B) - Q(s(t_B), a(t_B)|\theta^{Q})$;
21 Update critic using δ_o^Q as the loss function via gradient descent using ADAM;
22 Update actor using $-Q(s(t_B), a(t_B)|\theta^{Q'})$ as the loss function via gradient descent using ADAM;
23 Update critic target: $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1 - \tau)\theta^{Q'}$;
24 Update actor target: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'}$;
25 **end**

selected by adding noise to the action returned by the policy. This noise is drawn from a Ornstein-Uhlenbeck process [57]. This method of generating additive noise is favored by the authors of [4].

Lines 23 and 24 are where the target networks are updated using the current actor and critic values. The purpose of the target parameters is to add additional stability during learning by slowing learning down. Presumably this will allow the agent to keep more of what it has learned and prevent falling into local optima. Target networks have found some popularity in the deep Q-learning literature for their stabilizing effects [1]. Unlike the high-dimension cart-pole task,

we were unable to solve this task without using target networks. We use a mixing parameter of $\tau = 0.001$ in these experiments. This is a fairly slow leaking of parameter values from the actual critic and actor to the target networks. Note that in step 19 the target critic network is used to compute the target values so this means the changes in the Q-function will be particularly slow.

The implementation used in these experiments is the same as described by Lillicrap et al [4] with the exception of the ADAM learning rate of 0.001 which was used for both the critic and actor networks. The other ADAM parameters were kept at their default values. The RL parameter controlling the influence of future rewards in the current state's Q-value, γ, is set to 0.99 in these experiments. The critic network has three hidden layers of size 256, 128, and 128. Just as was done in [4], the action inputs to the critic are input directly to the second hidden layer of the network which means the first hidden layer only receives the state inputs. The actor network has three hidden layers of size 256, 128, and 64. Figure 3.14 shows the ANN architecture used in these experiments.



(a) Critic network. **(b)** Actor network

Figure 3.14: DDPG architectures for the critic and actor ANNs.

Figure 3.15 shows the baseline results for our implementation on this task. A cumulative reward of 300 or better on this task is considered solved. The cumulative reward is calculated over 1200 time steps or until the biped's hull either makes contact with the ground or crosses the goal. An efficient solution typically reaches the goal in approximately 900 time steps.

The best-performing solutions seen in the experiments performed here involve moving the hip joints back and forth in a true bipedal motion alternatingly placing the right and left legs in front of the agent's bulky hull. The agent doesn't move the knee joints nearly as much: just enough to allow the front and rear legs to make contact with the terrain. These solutions allowed the agent to

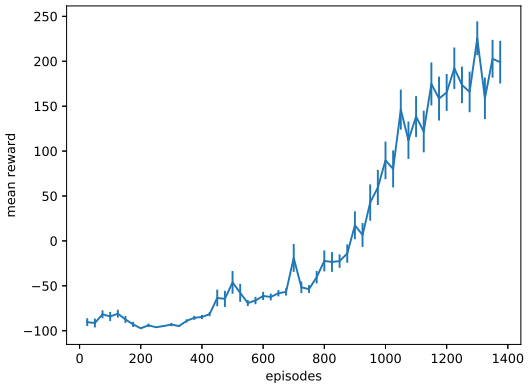
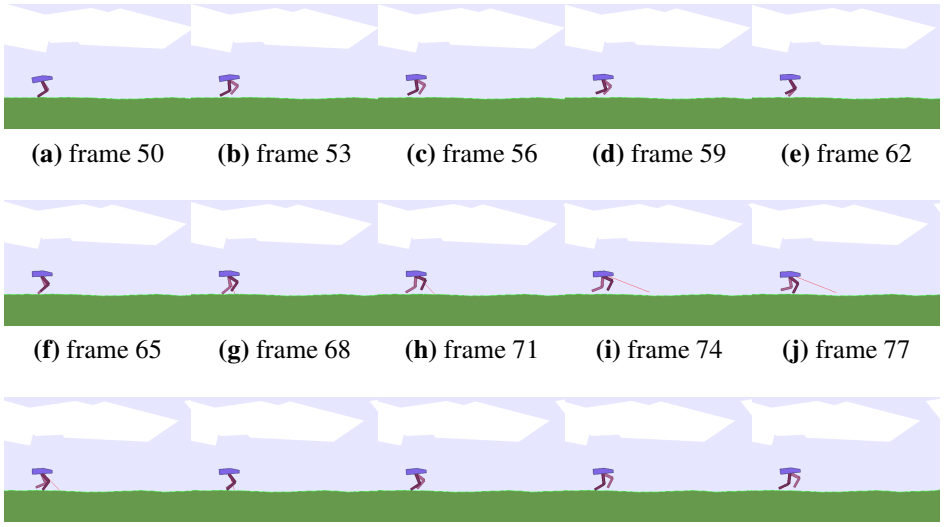


Figure 3.15: Performance of our DDPG baseline implementation on the bipedal walker task. The line is the mean and the error bars are the standard error across 50 agents.

traverse the entire width of the test terrain well within the allotted time of 1600 time steps (typically around 850 time steps). Figure 3.16 shows 15 frames from the OpenAI biped task GUI showing this gait. Figure 3.17 shows a graph of the hip and knee joint and hull positions and velocities of an evaluation which exhibits this behavior. This agent does not solve the task as can be seen in the uppermost plot where the hull falls forward and makes contact with the ground.



(a) frame 50 **(b)** frame 53 **(c)** frame 56 **(d)** frame 59 **(e)** frame 62

(f) frame 65 **(g)** frame 68 **(h)** frame 71 **(i)** frame 74 **(j)** frame 77

(k) frame 80 **(l)** frame 83 **(m)** frame 86 **(n)** frame 89 **(o)** frame 92

Figure 3.16: Frames captured from a successful agent on the bipedal walker task.

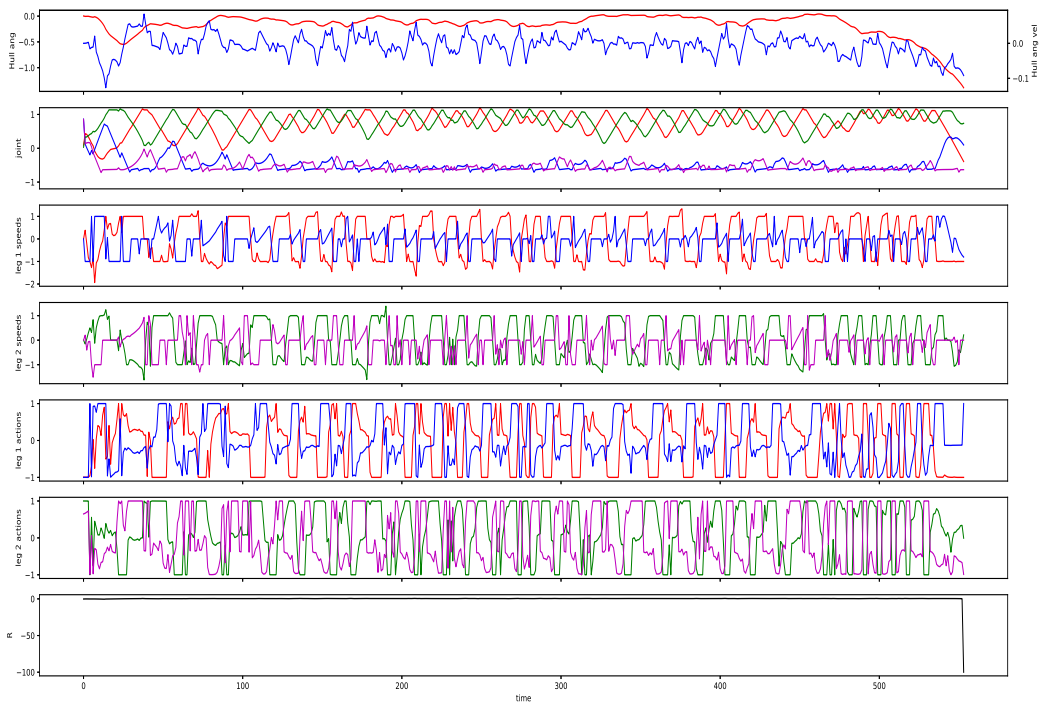


Figure 3.17: Graph of knee, hip, and hull state parameters during a single evaluation lasting 850 time steps. The upper-most plot is the hull angle and angular velocity. The second plot from the top is the joint positions of all four joints. The red and green lines are the two hip joints. The next two plots are the velocities of each joint. The bottom plot is the reward at each time step.

3.5 Pendulum task

The fourth task is the simplest. The purpose for adding this task is to aid in analyzing our results on the bipedal walker task.

We use the OpenAI implementation of the pendulum task where the objective is to swing-up and hold a pendulum into its inverted position (pointed up). Like the bipedal walker task, this is a continuous action task but it has only one action dimension: push left or right.

The pendulum task gives only negatively valued rewards. At each time step, the reward, whose sum is to be maximized, is:

$$r(t) = s_\theta(t)^2 + \frac{1}{10} s_{\dot{\theta}}(t)^2 + \frac{1}{1000} a(t)^2$$

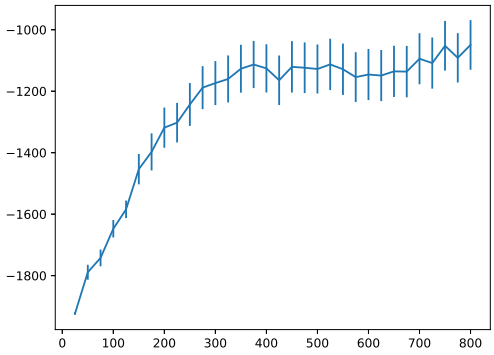
where s_θ is the pole angle, $\dot{s}_\theta$ its velocity, and $a(t)$ is the selected action. This reward mechanism primarily penalizes based upon the current pole angle but also favors slower-moving poles. Through the third term it favors smaller actions. Unlike the cart-pole task, the pole angle state, $s_\theta(t)$, representation is broken into two dimensions $[\cos(\theta), \sin(\theta)]$ for a total of three state dimensions. This avoids the discontinuity in the pole-down position and it simplifies Q-learning by providing the trigonometric function of the angle

The same algorithm was used to solve this task as the bipedal walker task. The only significant difference is the sizes of the actor and critic networks which were shrunk to a more appropriate number of parameters to match its reduced complexity. The actor network has three hidden layers of 20 nodes each with a single output node. The critic network has four layers of size 20, 20, 20, and 10, respectively, with a single output node. This ANN architecture was found using manual exploration.

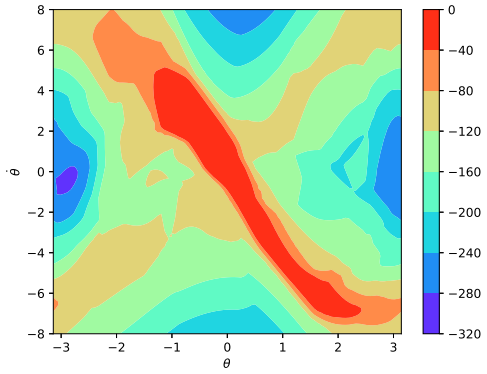
Figure 3.18 shows the performance of this baseline DDPG approach. Unlike the bipedal walker, DDPG does not show a decrease in performance during the latter half of training. The two entire state and decision space of the pendulum task can be plotted in three dimensions by showing the pole angle instead of its $\sin$ and $\cos$ components. Figure 3.18 also shows the maximum Q-function value for all possible pole positions and velocities for the final set of weights of one of the agents. The Q-function has a very similar shape to the cart-pole task when plotting only the pole position and velocity dimensions.

3.6 Experience replay

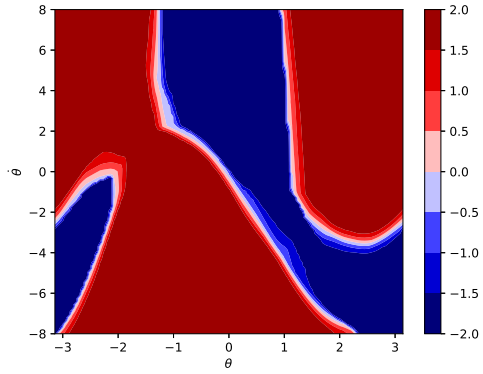
Much of the recent RL literature has focused on optimizing reward. As a result, few results regarding the necessity of the experience replay are shared. We briefly explore experience replay by comparing the DDPG and Q-learning approaches with and without experience replay. We then compare the DDPG and Q-learning approaches utilizing what we call indirect experience replay which is designed to test what we view as an underlying assumption of experience replay: that RL agents can learn from data generated from a policy much different than its current policy.



(a) Baseline pendulum performance. Error bars represent the standard error.



(b) Example Q-function.



(c) Example policy.

Figure 3.18: Results of a DDPG-trained agent on the pendulum task. The example Q-function and policy were captured at the end of training.

3.6.1 Comparison with and without experience replay

As discussed experience replay has become a defacto part of modern RL implementations. To get a feel for its affect on performance we compare the baseline RL approach for the bipedal walker task described in Section 3.4 with and without experience replay. The results are shown in Figure 3.19. We ran both versions fifteen times and averaged the rewards. To keep the code changes to a minimum, the no experience replay option was simulated by reducing the experience replay memory size to less than the maximum episode size or about 1.5 times the size of the average episode. We observed that no experience replay results in an agent that does not learn to solve the task.

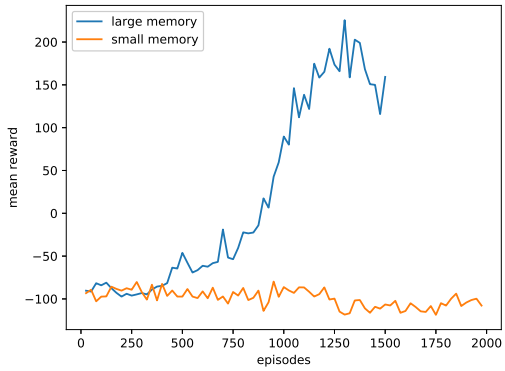
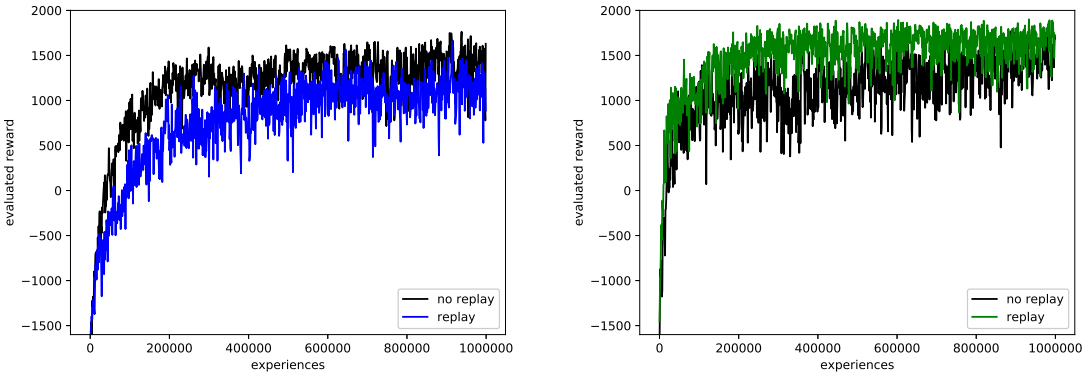


Figure 3.19: Comparison of DDPG with a large (10^6 time steps) and a small (1000 time steps) experience replay memory on the bipedal walker task. The small experience replay is an estimate for the effectiveness of this approach with no experience replay.

Comparing with and without experience replay on Q-learning for the low-dimension cart-pole task tells a slightly more complicated story. We tried two different Q-learning approaches: one where we only update the parameters once between training episodes and another where we update five times between training episodes which is the approach described in Section 3.2. Figure 3.20a shows the results when only allowing a single parameter update between episodes while Figure 3.20b shows the results when performing multiple updates between episodes. Each version of Q-learning was run 15 times.



(a) Single update between episodes.

(b) Five updates between episodes.

Figure 3.20: Comparison with and without experience replay with Q-learning on the low-dimension cart-pole task.

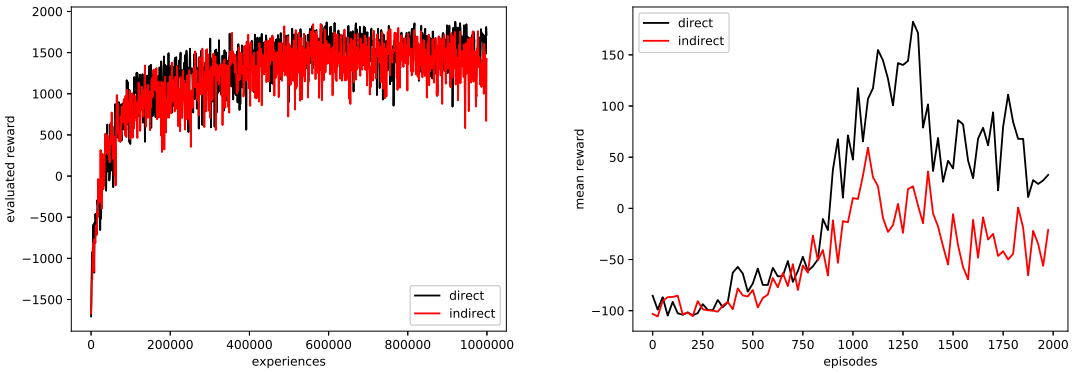
When only performing a single parameter update, no experience replay performs better than experience replay. When performing five parameter updates (five different samples from the experience replay memory), experience replay is superior. The black lines in Figure 3.20 are very similar, meaning that, when not using experience replay, there is very little difference in performance when increasing the number of updates when not doing experience replay. These results speak to the benefit of experience replay from the perspective of providing more opportunity for parameter updates: without those additional updates training on the latest batch of data is more effective.

3.6.2 Indirect versus direct experience replay

As training continues, the policy changes. We view experience replay as making the assumption that a RL agent can be trained from samples generated by a different policy. It is arguable that policies separated by thousands of experiences (or even just a few in the case of forgetting) are completely different policies. Experience replay when carried to an extreme is an indirect form of RL where the training experiences stored in the replay memory were generated by a different agent. For the purposes of comparison, we refer to this as indirect experience replay and the traditional form of experience replay, where the experience are generated by the agent during training, as direct experience replay. We test this concept on both the low-dimension cart-pole task with a non-ensemble Q-learning agent and a non-ensemble DDPG agent on the bipedal walker task. The results are shown in Figure 3.21.

These results show that there is a very small difference between indirect and direct experience replay for the Q-learner on the cart-pole task. On the other hand the DDPG agent is negatively affected by indirect RL.

We suspect that the negative effect of indirect experience replay on the DDPG agent is a symptom of it being a policy gradient approach and, therefore, requires data generated by the policy (or a similar policy). Compared to Q-learning, it is possible that policy gradient approaches may be less compatible with any form of experience replay. We return to this very briefly in Section 6.8.



(a) Q-learning agent on low-dimension cart-pole task.

(b) DDPG agent on bipedal walker task.

Figure 3.21: Comparison of direct and indirect experience replay. Indirect experience replay uses experiences generated while training a different agent. In these results Q-learning only very lightly negatively affected indirect experience replay while the DDPG agents' performance was dramatically reduced.

Since we are following the work of Lillicrap et al. [4] in our experiments with DDPG we continue to use an experience replay of size 1×10^6 and leave this to future work.

3.7 DDPG actor and critic target functions

Deep deterministic policy gradient (DDPG) RL approach is an application of many aspects of the deep Q-learning (DQN) such as experience replay and target functions to the deterministic policy gradient approach (DPG). As discussed in Section 3.6 experience replay may not be as natural a fit for DDPG as it is for Q-learning. Target functions are another RL innovation which has been popularized by the DQN literature. We did not use target functions in the Q-learning implementations for the cart-pole tasks but they are used by Lillicrap et. al. [4] whose implementation we are following here.

We wish to better understand the effect of the target functions on DDPG. We compare DDPG performance with and without target functions including individually removing target functions for the critic or the actor. The results are shown in Figure 3.22.

These results indicate that both target networks provide better results. The critic target network is necessary while the actor target network leads to improved performance but the DDPG approach

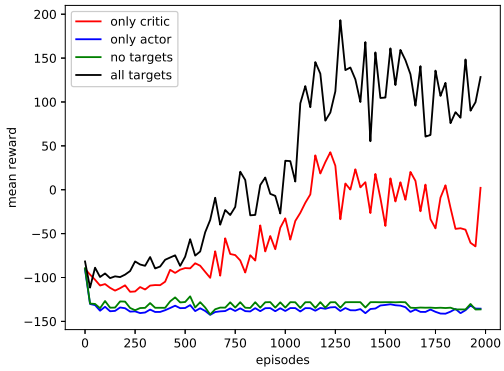


Figure 3.22: Comparison of DDPG on bipedal walker task with and without target networks.

is not crippled by its removal. This indicates that DDPG is heavily dependent upon the stability that the target network provides the Q-function. This presages the benefit we will observe later when modeling the Q-function using an ensemble which also provides stability to the Q-function estimates used in the training of the policy.

Chapter 4

The Crowd ensemble for Q-learning

In this chapter the concept of the crowd ensemble is presented along with experiments showing that the crowd ensemble out-performs the base Q-learning approach. We begin with the motivation behind the crowd ensemble approach. We then move to testing it on the tasks described in Chapter 3. For each task a modified algorithm is provided.

We also provide results showing performance competitive with the approach described by Fauber and Schwenker [44] while being applicable to real-world problems, unlike the Fauber and Schwenker approach. This is followed by experiments showing improved performance of the low-dimension cart-pole task. Then results are presented which show benefits for the high-dimension variant of the cart-pole task where state is represented by an image of the cart-pole and the Q-function is approximated using a DQN. This is followed by results on a continuous action-space task: the bipedal walker which is trained using an actor-critic approach. We see that the crowd ensemble is beneficial in all of these tasks and RL approaches. Finally, we briefly discuss results for the pendulum task.

4.1 Motivation

As discussed above, expert-based ensemble methods have not found wide-spread acceptance in practice even if the concept has found wide-spread acceptance. More recently, whenever an ensemble-like approach is employed, the literature has trended toward a crowd ensemble approach where much less effort is made to guide the ensemble members toward specialization.

A crowd ensemble RL approach is characterized by equal access to all training data for all members and a simplistic scheme to combine their knowledge. In the crowd ensemble approach to Q-learning utilized here, each ensemble member learns its own Q-function and they are only combined during action selection via simple majority voting. The action selected during training is the only thing holding the ensemble together – this provides them with a shared experience.

This allows the crowd ensemble to learn in parallel from a single simulation or real-world system which means that it can be used in problem domains where multiple simulations are not possible (which includes all real-world problems). When used with an actor-critic, the crowd ensemble is applied to the critic only. The critic output used to update the actor's selected actions is the mean or median combination of the ensemble's Q-values.

Ensemble methods are a logical approach to RL and have been attempted in several forms. RL tasks are difficult and ensemble methods are designed to provide additional representational power without adding complexity to the individual models. Intuitively, RL problems are thought to be decomposable into sub-tasks and an ensemble approach, for example, could automatically and organically decompose a task by state-space region. Finally it is thought that an ensemble approach to RL could avoid forgetting by allowing each expert to specialize in a region of the state space thereby avoiding forgetting good value functions when exploring other regions of the state space.

A crowd ensemble approach is more straight-forward to implement than the other ensemble RL methods. Implementation of a crowd ensemble with N_e members amounts to training N_e separate learners.

4.2 Crowd ensemble for discrete action Q-learning

When discrete actions are involved, the action decision can be simply decided by a majority vote. More specifically each ensemble member produces a desired action at each time step, $a_c(t)$. The action selected by the ensemble is the action with the greatest number of votes, $v_a(t)$.

Discrete action selection begins with each ensemble member selecting an action independently:

$$a_c(t) \leftarrow \arg\max_a Q_c(s(t), a)$$

where Q_c is the Q-function modeled by ensemble member c. The vote accumulation is defined as:

$$v_a(t) \leftarrow \sum_{c \in \{1 \ldots N_e\}} \delta_{a_c(t),a} \tag{4.1}$$

where $\delta_{a_c(t),a}$ is the Kronecker delta function which is one when $a_c(t) = a$ and zero otherwise. Equation (4.1) is the tally of all votes for action a. Finally, the ensemble's choice for the action at time t is:

$$a(t) \leftarrow \arg\max_{a \in \mathscr{A}} v_a(t).$$

In a departure from a conventional ensemble approach where an ensemble error is computed and then passed back to the members according to some measure of responsibility, the members of the crowd ensemble are updated using independent TD-error values:

$$\delta_c^{(TD)}(t) \leftarrow r(t) + \gamma \arg\max_{a \in \mathscr{A}} Q_c(s(t+1), a) - Q_c(s(t), a(t))$$

where the c subscript emphasizes that each member computes an independent TD-error.

Prior to learning, the Q-function approximator parameters for each expert should be randomly initialized. Learning in a crowd ensemble, as defined here, entails the following steps.

1. Vote: select an action for each expert.
2. Tally: determine which action is selected by the most members.
3. Act: take the action selected by the ensemble.
4. Compute errors: compute a TD-error for each member.
5. Update: update Q-function approximations using each member's TD-error.

An implementation of a crowd ensemble is shown in Algorithm 4. This algorithm should be compared to Algorithm 1.

Lines 13 and 14 are the action voting and action selection steps. Lines 22–27 are repeated for each ensemble member meaning each ensemble member draws a unique set of training sequences

Algorithm 4: Pseudo-code of the crowd ensemble for the low-dimension cart-pole Q-learning implementation using a two-layer ANN to model the Q-function.

```
1  Initialize ANN weights;
2  t ← 0;
3  while t < T do
4      Obtain initial s(t_B = 0), a(t_B = 0), r(t_B = 0);
5      t_B ← 1;
6      while t_B ≤ N_B do
7          s(t_B) ← Act(a(t_B − 1));
8          r(t_B) ← Reward(s(t_B));
9          if Random(0,1) > ε then
10             Set a(t_B) to random action;
11         end
12         else
13             v_a(t) ← Σ_{c=1}^{N_e} δ( arg max_{a'} Q^(c)(s(t), a'), a);
14             a(t) ← arg max_a v_a(t);
15         end
16         S_b(t) ← (s(t_B), r(t_B), a(t_B));
17         t_B ← t_B + 1;
18         t ← t + 1;
19     end
20     Store training sequence, S_b, of length N_B in memory;
21     for c ∈ {1 ... N_e} do
22         for t_R ∈ {1 ... N_R} do
23             Randomly draw a training episode sequence, S_u, from memory;
24             T_V^(c)(t_B) ← r(t_B) + γ * max_a Q^(c)(s(t_B), a) ∀t_B ∈ {1, ..., |S_u|};
25             δ_o^(c) ← Σ_{t_B=1}^{N_B} T_V^(c)(t_B) − Q^(c)(s(t_B), a(t_B));
26             Perform 5 updates of ANN weights using SCG;
27         end
28     end
29     if t % evalFrequency then
30         Evaluate by performing lines 6–18 and setting ε ← 0.
31     end
32 end
```

to replay during training. Lines 24 and 25 are computations of the member-specific Q-function target values and error function for use during parameter updates.

For the cart-pole swing-up task, training is done in a long, continuous trial of T time steps which starts with a zero cart position and velocity and pole in down position with no velocity. Evaluation of the solution is performed every 1×10^4 time steps by starting a separate simulation at the pole down position with the cart in the center of the track with no velocities.

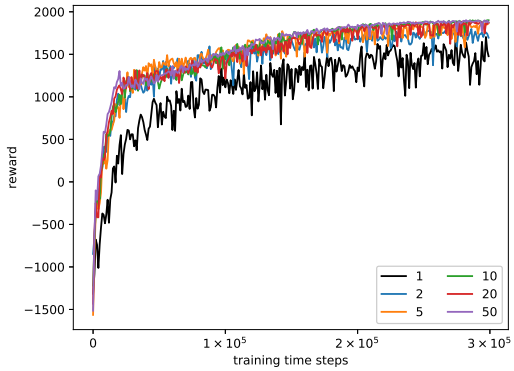
4.3 Exploring crowd ensemble approach for the low-dimension cart-pole task

The cart-pole tasks benefits from the crowd ensemble approach as shown in Figure 4.1 which shows that the mean reward during evaluation is improved when using any size ensemble with no significant improvement after $N_e = 5$. Figure 4.1a shows results for an extended set of N_e values. Here we can see that even an ensemble of size two makes a difference in performance. Also evident from the figure is the relative stability of $N_e = 50$: the evaluated reward has less variation than the smaller N_e values which have less than the non-ensemble Q-functions.

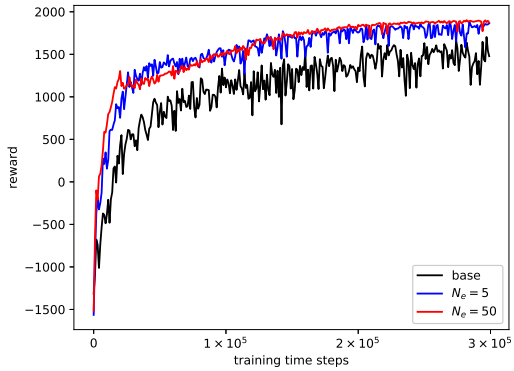
These results use experience replay which improves the stability of the RL agent during training by preventing forgetting. The increased volatility of the black line ($N_e = 1$) is a result of the performance variability caused by forgetting. The variability of the lines decreases as N_e increases. This is especially evident when comparing the $N_e = 5$ and $N_e = 50$ lines in Figure 4.1b.

A look at the median values for $N_e = 5$, $N_e = 50$, and $N_e = 1$ shows a decreased improvement but an improvement nonetheless. The median reward can be viewed as the performance of a typical agent for each approach and is a measure which is less susceptible to outliers. This indicates that the performance improvement is, in part, a factor of improved stability rather than taking the Q-learning approach to evaluated reward levels unattainable without the ensemble. The improved stability is evident in Figure 4.2 which shows eight randomly-selected runs of the ensemble with $N_e = 50$ and non-ensemble approaches.

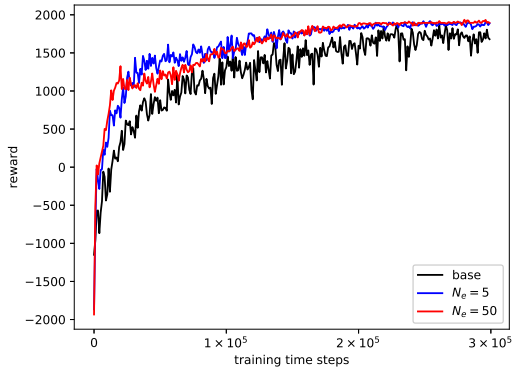
All plots for the non-ensemble runs show catastrophic forgetting with performance as low as $R = -2000$ not long after solving the problem with $R > 1950$. Each of the randomly selected



(a) All N_e values.



(b) Comparison of select N_e values against non-ensemble approach – mean reward.



(c) Comparison of select N_e values against non-ensemble approach – median reward.

Figure 4.1: Comparison of crowd ensemble for a variety of N_e values. Figures 4.1a and 4.1b show the mean, evaluated reward after the specified number of time steps. Figure 4.1c shows the median, evaluated rewards.

single Q-learners shown in Figure 4.2 is an example of this. In contrast, none of the plots in the second row of Figure 4.2 show this level of volatility.

How the large crowd ensemble ($N_e = 50$) ameliorated the instability is also evident when comparing the median and mean evaluated rewards in Figures 4.1c and 4.1b, respectively. The mean and median of the two plots is nearly identical. This indicates that the mean is unaffected by outliers and that the means is representative of truly moderate value. This could come about only if there was not catastrophic forgetting.

However, improved evaluated reward does not tell the entire story of why the crowd ensemble is beneficial. As described earlier we consider an evaluation with cumulative reward greater than

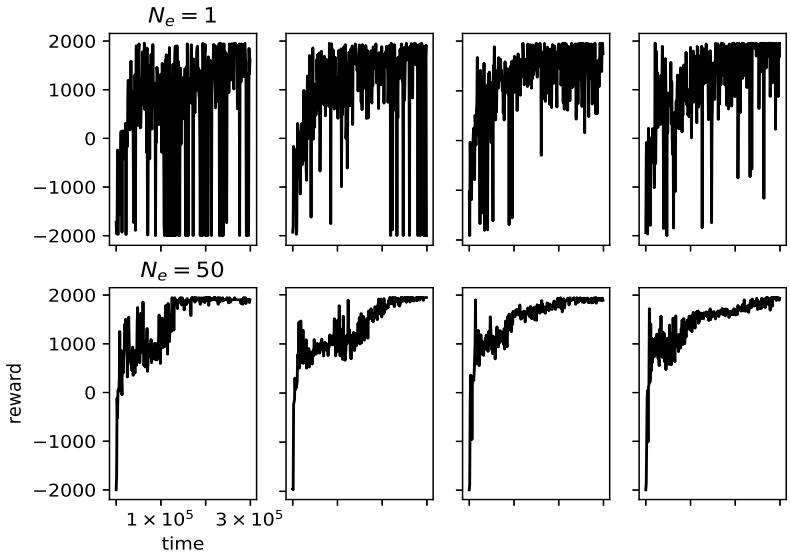


Figure 4.2: Randomly selected agents for $N_e = 1$, the single Q-learner, (top row) and $N_e = 50$.

1950 as solved. Figure 4.3 shows that the ensemble approaches solve the task earlier and more often than the non-ensemble approach. In this figure, all crowd ensemble agents solve the task within 1.7×10^5 time steps while 29 of 30 base Q-learners solve the task by 3×10^5. Furthermore, the crowd ensemble agents begin solving the task at $t = 9 \times 10^4$ with 80% solving the task by $t = 1 \times 10^5$ compared to 50% for the single Q-learner. This indicates that the crowd ensemble can do something the base Q-learning approach cannot: it can solve tasks more quickly and reliably.

There are diminishing returns when it comes to adding ensemble members. It is not clear from these results why this is. One obvious limitation, however, is that $N_e = 5$ is already reaching the practical maximum mean reward for this task so only improvements in solution stability may be visible after this point without using very large sample sizes. It is not detrimental to include additional ensemble members but it is not guaranteed to improve mean performance. In these experiments increasing the number of ensemble members will not detrimentally affect performance. Instead, the effect will be felt only in terms of computational resources which is fairly negligible when dealing with small ensemble sizes which sample from the same memory from replay. This is especially true when considering how this approach is easily parallelizable. Once implemented there is little risk in adding additional members to the ensemble.

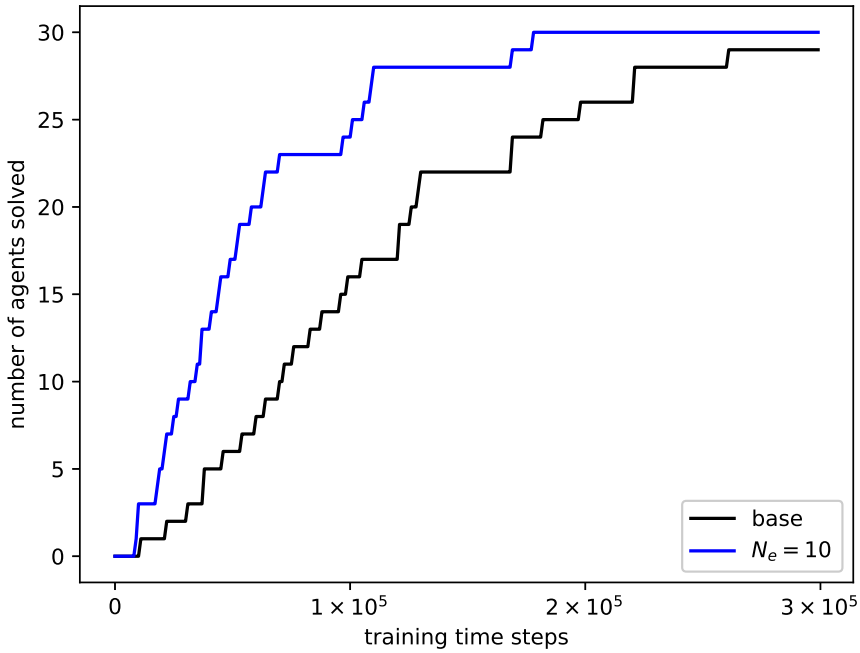


Figure 4.3: Number of agents (out of 30) that have solved the low-dimension cart-pole task as training progresses.

4.3.1 A peek under the crowd ensemble hood

Each ensemble member is trained to solve the entire task in isolation from the other members, therefore a member can be evaluated independently which will give an accurate picture of it's performance at that point in training. Figure 4.4 shows the evaluated reward during training for the ensemble members evaluated independently (black lines) along with when evaluated as an ensemble (red line) for ten randomly selected agents. There are $N_e = 10$ black lines plotted in each plot. Also included is the mean performance of the ensemble members' independent evaluations (blue line).

The mean ensemble member reward (blue line) is lower than the reward achieved by the ensemble (red line). This implies that there is a synergistic relationship between the ensemble members when trained together and combined using voting which allows the ensemble to avoid the pitfalls of forgetting that drag the ensemble average down. The ensemble performance is always in the top quartile of the collective performance of its members with just a few obvious exceptions when

ensemble performance drops catastrophically. Although, at any given time step, a particular ensemble member may out-perform the ensemble as a whole, on average an ensemble member will perform worse.

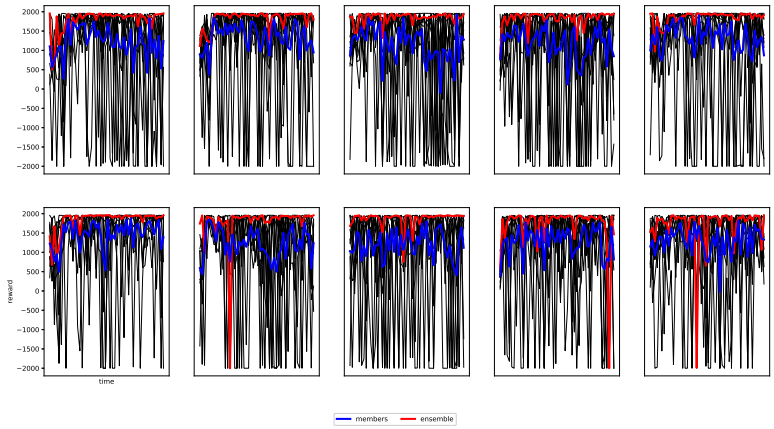


Figure 4.4: Performance of ten runs of crowd ensemble with $N_e = 10$. The thin, black lines are the evaluated performance of the ensemble members (evaluated independently). The thick, red line is the performance of the ensemble as a whole. The thick, blue line is the mean performance of the ensemble members.

Figure 4.5 compares the performance of ensemble members, evaluated independently, for ensembles of sizes $N_e = 5$, $N_e = 10$, and $N_e = 50$ across 10 randomly-selected agents (e. g., the line associated with $N_e = 10$ is the mean of the black lines in Figure 4.4). During training the mean ensemble member evaluation decreases although the mean evaluation and the stability of the ensembles do not decrease. The base Q-learning line, $N_e = 1$, is computed using 40 independently-trained Q-learners. The higher volatility of the smaller N_e values is caused, in part, by the smaller sample size.

The vertical line in Figure 4.5 is the maximum length of the training used in the low-dimension cart-pole experimentation reported earlier in this chapter. At this point, the mean ensemble member performance has yet to decrease for any ensemble size. The training time was extended to 1×10^6 to determine whether or not the diversity of ensemble member performance would hold up under continuous training. This figure implies that the ensemble members are becoming more diverse as training progresses.

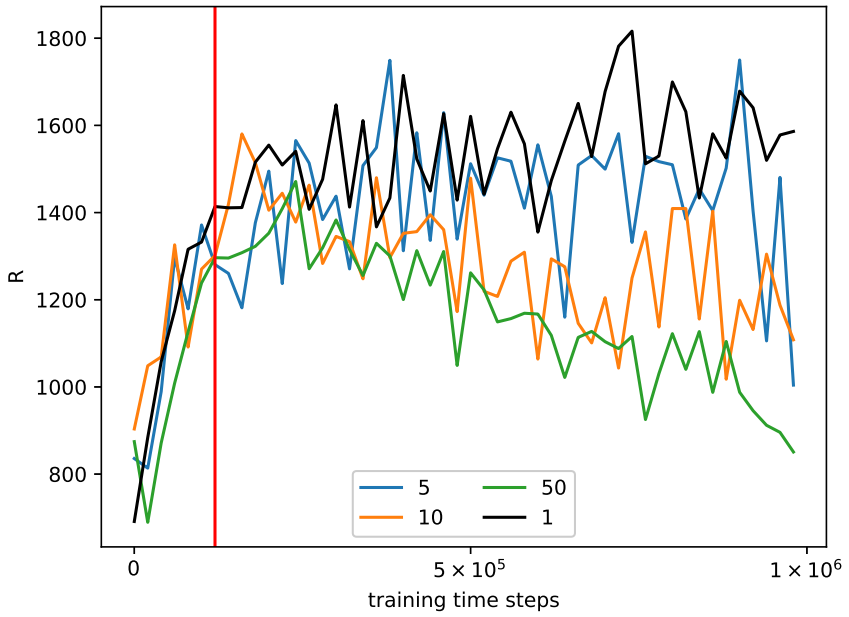


Figure 4.5: Mean performance ensemble members for four selected values of N_e. The red, vertical line is there for reference: it is the training length usually used for low-dimension cart-pole experiments in our experiments.

Figure 4.6 shows that this dramatic decrease in the mean evaluation reward of ensemble members does not affect the overall performance on the ensemble. Only eight agents were used in this plot so the mean reward has larger variance when compared to the results of Figure 4.1.

As N_e increases the training data which comprises the experience replay memory is further removed from the actual training data that would be generated by each of the individual ensemble members if they were generating their own data. This could partially explain the lower performance for the ensemble members for larger N_e.

Another factor is the reduced variation of the solutions learned by the ensemble and placed in the experience replay memory. An ensemble of size $N_e = 50$ reaches a quality solution early in training so once $t > 1.5 \times 10^5$ the experience replay memory is primarily filled with states from the goal region. This is especially true when considering that the training environment is never reset during training.

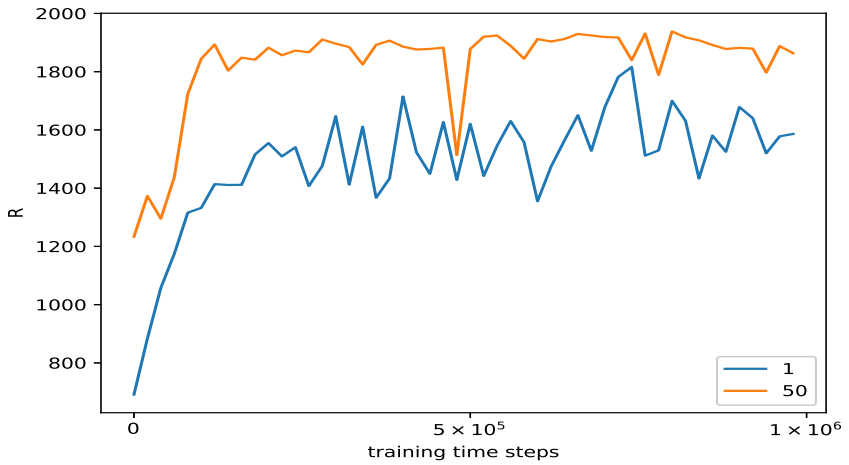


Figure 4.6: Mean performance of the eight $N_e = 50$ ensembles used to generate the corresponding line in Figure 4.5. Note that performance remains steady for all 1×10^6 time steps with one exception where one of the eight suffered from a moderate drop in evaluated reward. A line for $N_e = 1$ is included for reference.

This lack of variability and high correlations of training data pulled from similar regions of the state space is thought to be a significant contributor to forgetting. This is what is happening here. Impressively, even when allowed to run for 1×10^6 time steps, the ensemble is robust enough to withstand the increased volatility of its members. Ironically, this feeds back into the ensemble as increased diversity of ensemble member solutions.

At first blush one might think this phenomenon is an example of how ensembles are thought to be better off as a combination of weak learners. However this is not a good analogy because that reasoning applies to allowing the ensemble members to specialize in a mixture of experts setting to prevent them from attempting to model the entire state space. In the case of crowd ensembles, no such attempt is made.

The evaluated reward of the ensemble members isn't the only thing that decreases during training. The size of the voting majority during training also decreases. The cart-pole task solution is evaluated regularly after a specified number of trials has elapsed. The evaluation is 2000 time steps in duration. This is done around line 13 of Algorithm 4. At each time step the ensemble members cast their votes for their preferred action. The action with the most votes is selected and

the number of votes that action received is referred to as the majority size. Figure 4.7 shows a box plot of the mean majority size across the 2000 times steps of each evaluation.

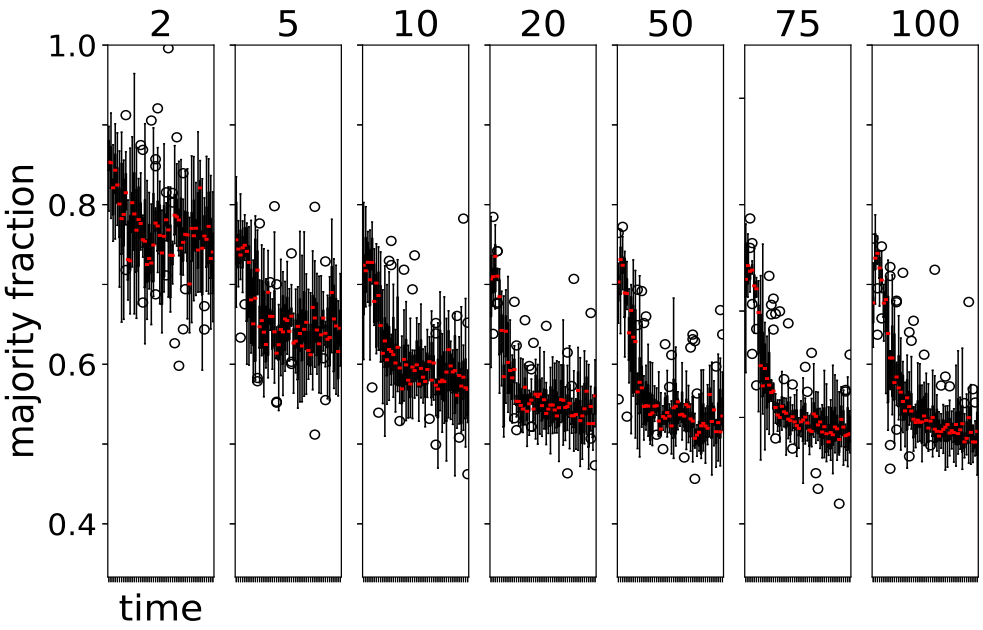


Figure 4.7: Box plots of majority sizes during training for all selected N_e values. The red bars represents the median at each evaluation, the boxes represent the range of the lower and upper quartile values of the data, and the whiskers the entire range of the data excluding outliers. Outliers are represented with unfilled circles and are determined to be all samples lying outside two times the interquartile range.

As can be seen, the majority size of the selected action is decreasing as training progresses. As discussed above this does not affect the ensemble performance. Figure 4.8 shows the means of the majority size for the same N_e values on the same plot for easier comparison. The y-axis is the percentage of the ensemble members voting with the majority. Is is clear that the majority size decreases during training as well as decreases with N_e.

Another expectation is that the ensemble members will eventually, given enough training time, converge onto similar Q-functions causing the ensemble to then show the same instability as the non-ensemble. If this is occurring one would expect that the votes of the ensemble members would become more homogeneous. Figures 4.7 and 4.8 show this is not the case. It appears that the opposite is happening with less uniformity in the selected actions as training progresses.

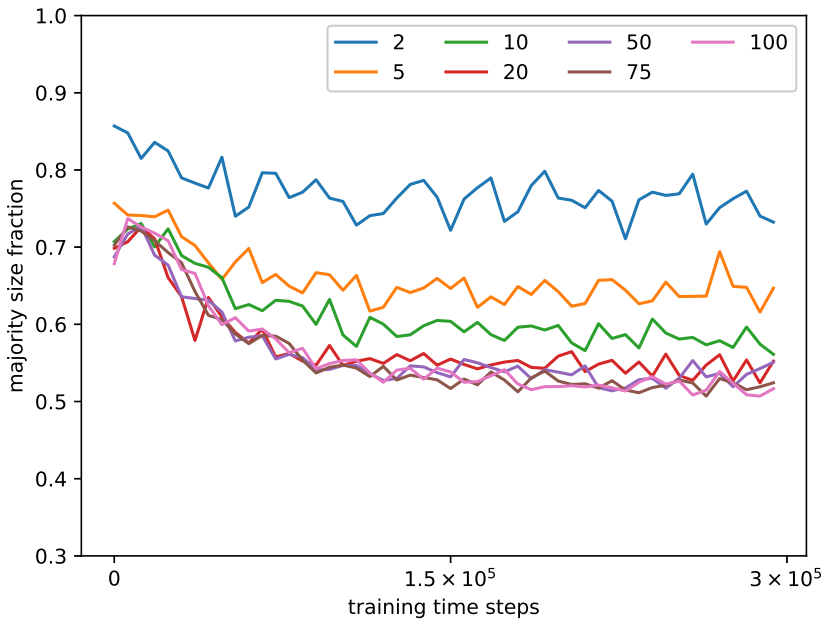


Figure 4.8: Mean majority size for selected N_e values. How these values are computed is described in detail in the text.

Is the decreasing voting majority and the mean performance of the ensemble majority connected somehow? If the mean performance of the ensemble members is decreasing one would expect the diversity of their selected actions to increase. Indeed this is what we are seeing here and this may be a partial explanation of why the majority sizes are decreasing. This was done in an effort to see if this decreasing majority size would eventually overtake the ensemble and eventually lead to worse-performing Q-functions. However it appears that, although decreasing, the majority size ratio is converging at a value still significantly higher than the minimum majority size represented by the minimal y-axis values of the two figures.

Another explanation for the decreased voter majority sizes is that the best solutions spend the vast majority of their evaluation in regions of the sate space which have the smallest voter majority. It turns out, as we will discover later, that the center of the goal region of this task lies in a region of the smallest voter majority because this is where the preferred action transitions from $a = 1$ to $a = -1$.

4.3.2 Q-function ensembles reduce decision instability

The Q-values are used to compute an action decision at each state position encountered. For every state location there is an associated, preferred action or decision. This creates a decision surface which can be plotted. Figure 4.9 shows an example decision surface for a single Q-learner on the cart-pole swing-up and balance task. This figure was made by fixing the cart and pole angle velocities at zero and computing the best action (highest $Q(s, a)$ value) at 100 discrete locations across all possible cart and pole positions. As expected the pole's zero angle position is the dividing line between states where the left and right push actions are preferred. For pole angles greater than zero (tilting right) with the cart on the left half of the track, the preferred action is to push right in an attempt to move the pole to the balanced position. In other cart positions when the pole is tilted to the right the preferred action is the push left which would accelerate the pole's descent presumably to swing the pole $360°$ back around to the balanced position. When the pole is straight up (zero angle), given its zero velocity state, the best action is do nothing.

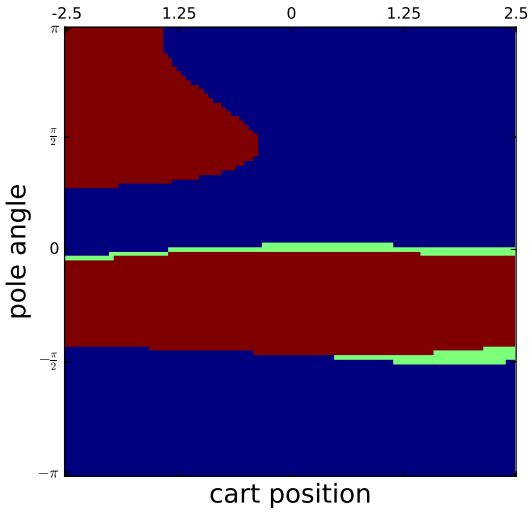
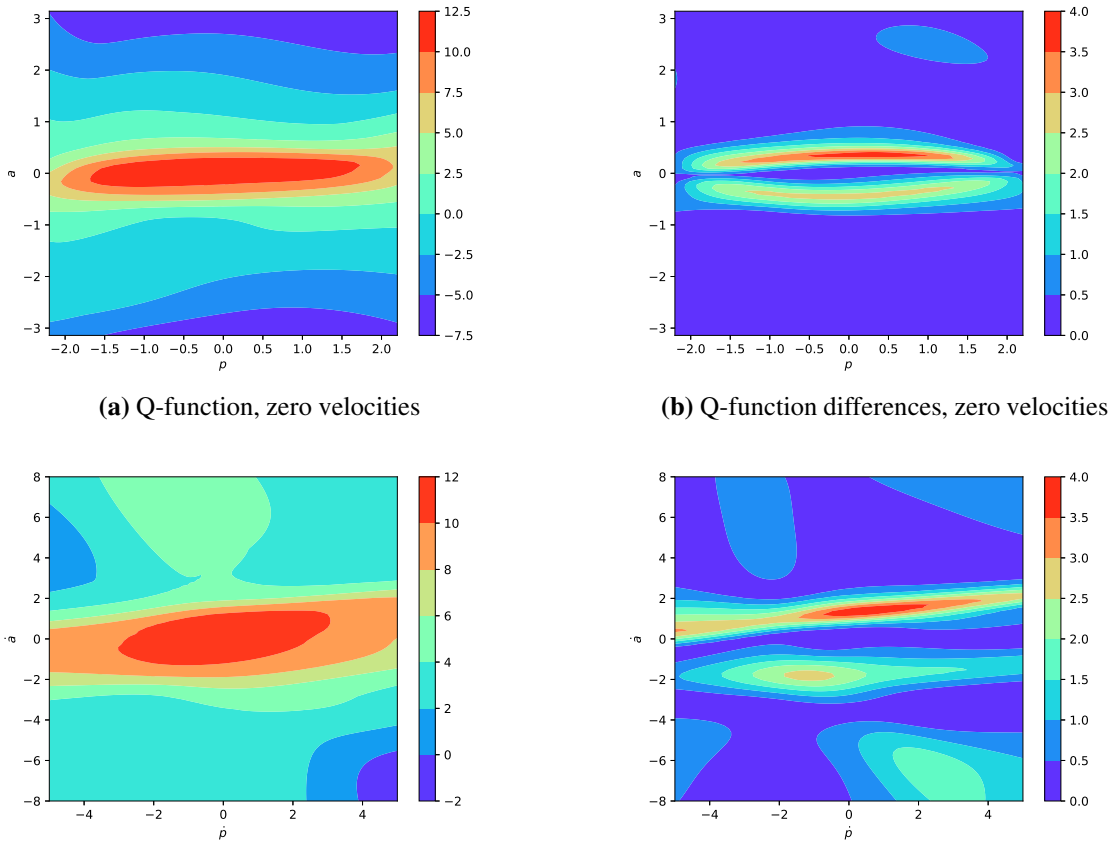


Figure 4.9: Example, two-dimension decision surface of a single Q-learner across $10,000$ discrete locations for all possible cart positions and pole angles with cart and pole velocities set to zero. Blue is push right, red it push left, and green is no action.

It is expected that, in regions adjacent to states where the preferred action changes, the Q-function values across the three actions would be most similar for a particular Q-learner in either an ensemble or by itself as a single Q-learner. This is the case in Figure 4.10b which plots the



(a) Q-function, zero velocities

(b) Q-function differences, zero velocities

(c) Q-function, zero positions

(d) Q-function differences, zero positions

Figure 4.10: View of two-dimensions of Q-function values learned a single Q-learner after 10^5 Q-learning trials. Image was made by sampling the Q-function across $10,000$ discrete locations for all possible cart positions and pole angles with cart and pole velocities set to zero.

difference between the highest and lowest Q-function values across actions on a discrete selection of cart and pole positions with zero cart or pole velocities. This is computed as $\max_{a \in \mathscr{A}} Q(s,a) - \min_{a \in \mathscr{A}} Q(s,a)$. Figure 4.10a shows the associated Q-function which clearly prefers the cart to be centered on the track with a zero pole angle. This value is computed using $\max_{a \in \mathscr{A}} Q(s,a)$. Similarly, Figure 4.10c shows the example Q-function as having a strong preference for zero pole

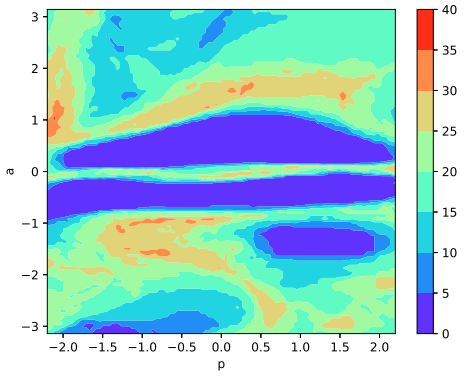
velocity. The Q-values are much less strict about the preferred cart velocity which makes some sense given that the reward function does not include information about the cart state.

Comparing Figures 4.10b and 4.9 it can be seen that the region where the decision surface transitions between actions has low $\max_{a \in \mathscr{A}} Q(s,a) - \min_{a \in \mathscr{A}} Q(s,a)$ values (the Q-fuction difference). It also stands to reason that these border states are the ones with the most volatility, in terms of action selection, during learning.

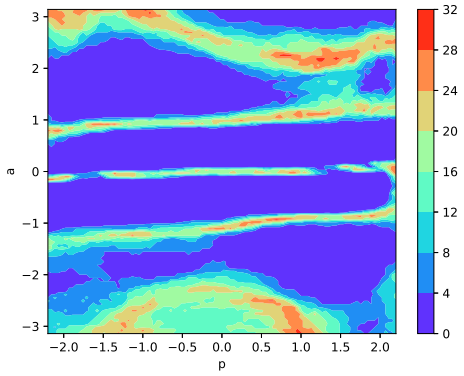
Figure 4.11 shows change count maps for three N_e values. Each image was made by counting the number of changes at 25^4 locations located across all state space dimensions (cart/pole position, cart/pole velocity) after each batch. Then these values are summed across the cart/pole velocities to reduce to two dimensions. The images are normalized with the extreme values from $N_e = 1$ (Figure 4.11a).

This relationship is supported, partially, by the band of high decision volatility running horizontally through the center of Figure 4.11a. However the regions of high volatility expand far beyond the boundary between the actions to the extent that action decision volatility appears to be the norm, not the exception. The regions of high decision volatility in Figure 4.11a are a negative image of the regions of high Q-difference values in Figure 4.10b. From Figure 4.11 we also observe a decline in the number of decision changes as N_e increases.

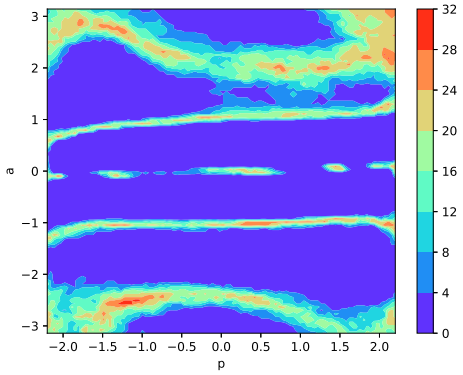
Figure 4.12 shows histograms of the change counts for the same N_e values used to create Figure 4.11. As in Figure 4.11, Figure 4.12 shows a decreased number of decision surface changes as N_e increases. These two figures tell a very clear story. After the agent has solved the task late in learning when a sufficient amount of exploration has occurred and we would prefer that the agents do not stray too far from their winning solutions, the ensemble agents change less. They change a lot less and the amount of change decreases as the ensemble size increases.



(a) $N_e = 1$

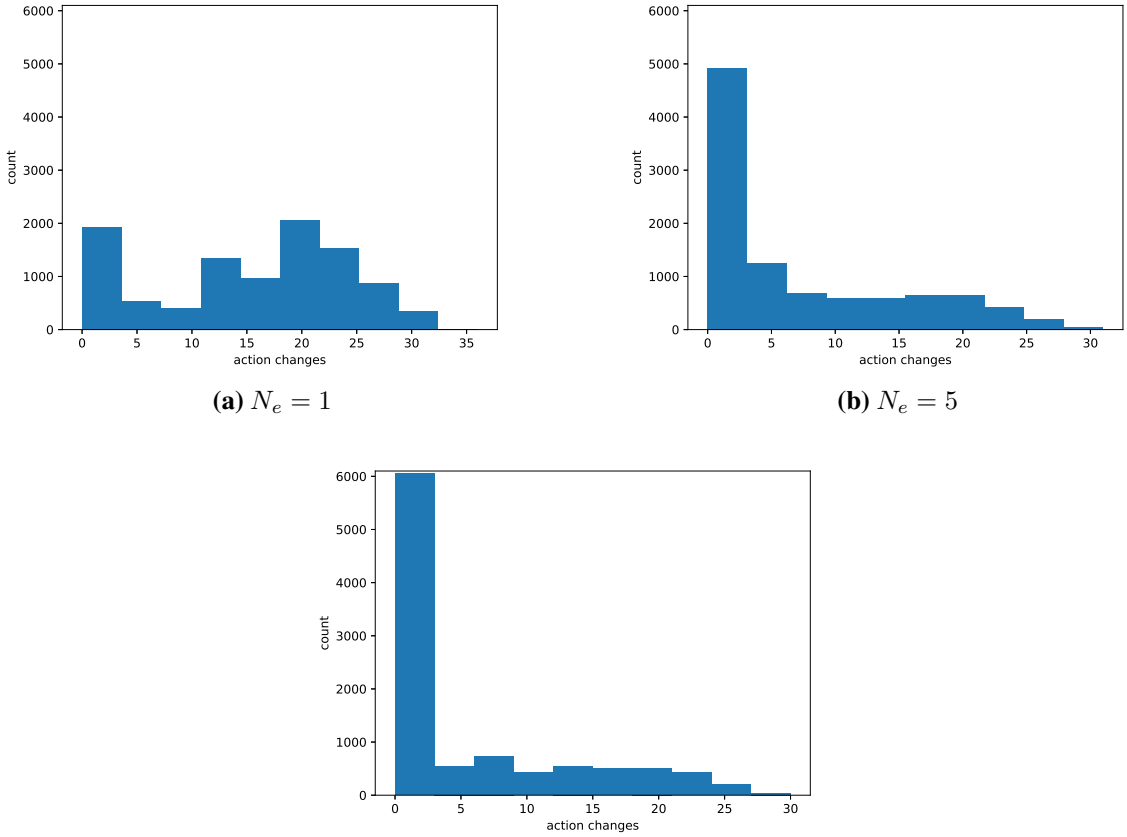


(b) $N_e = 5$



(c) $N_e = 10$

Figure 4.11: View of two-dimensions of the count of decision surface changes for the final 10^4 Q-learning training samples (final 50 batches) for three randomly-selected agents. The x-axis is cart position and the y-axis is pole position.



(a) $N_e = 1$

(b) $N_e = 5$

(c) $N_e = 10$

Figure 4.12: Histogram of the count of decision surface changes for the final 10^4 Q-learning training samples (final 50 batches) for three randomly-selected agents. Here we see that the number of states with little to no change is higher for ensemble Q-functions and that the number of non-changing states increases as N_e increases.

4.4 Crowd ensemble comparison with parallel Q-learning

Asynchronous and parallel RL approaches utilize multiple simulations to improve or speed-up training. The crowd ensemble was compared with a parallel ensemble RL approach. The parallel RL approach most similar to the crowd ensemble was proposed by Fauber and Schwenker [44]. A disadvantage of parallel RL approaches is that they are unusable when collecting data from real-world systems where data collection cannot be done in parallel. Furthermore real world data is expensive to collect both in terms of time and resources. Asynchronous and parallel RL trades reduced computation during training for a dramatically increased data collection effort. The tasks used to evaluate parallel RL methods are simple simulations where this is a worthwhile trade-off. However when considering a full simulation of a complex physical system, let alone an actual physical system, this trade-off may no longer be beneficial.

What is the potential advantage of the Fauber and Schwenker approach, specifically, and parallel ensembles in general when compared to the crowd ensemble approach? The difference between the two approaches is in how the training data is generated. In the Fauber and Schwenker approach, each ensemble member has a unique training experience as a result of its separate simulations whereas, in the crowd ensemble approach, the members have a shared experience from a single simulation. This results in a more robust, diverse ensemble of Q-learners for the Fauber and Schwenker approach.

In the parallel approach the ensemble members do not draw from the same experience replay memory during training but the decision space of the ensemble members do influence each other. In the approach proposed by Fauber and Schwenker (FS) the action decisions for a particular ensemble member were made using a weighted combination of the member's action choice and choices of the other ensemble members. The FS approach to RL is discussed in Section 2.6. We implemented the FS approach and compared its performance against the crowd ensemble approach on the cart-pole task. The results are shown in Figures 4.13 and 4.14. The FS approach results are labelled as parallel.

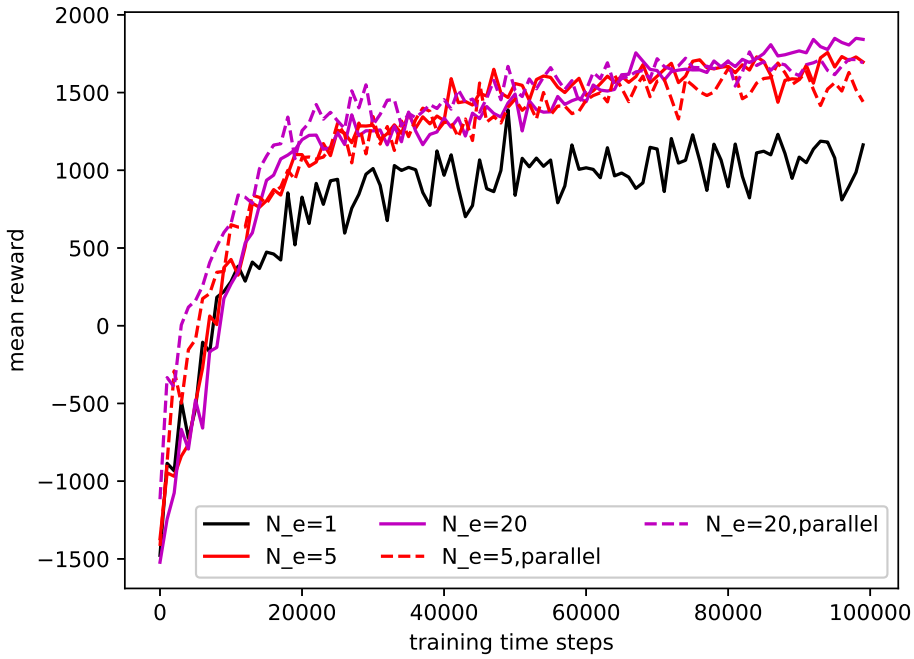


Figure 4.13: Comparison of crowd ensemble, Fauber & Schwenker (parallel) approach, and single Q-learner. Showing all 3×10^5 time steps of the crowd ensemble training.

The results shown are the average reward for the combination of two tasks: the balance task where the pole begins near the goal position with a random action applied to it to start the pole's descent and the swing-up task where the pole start in the down position. These test the agents' ability to keep the pole in the goal state and to find the goal states and then remain there.

Figure 4.13 shows competitive performance between the two approaches. The crowd ensemble provides the same benefits without the need for simultaneous simulations. If this is re-plotted to show the actual number of environment interaction steps experienced across all FS members from their parallel simulations, the crowd ensemble appears to learn much more quickly. Figure 4.14 shows a redrawing of Figure 4.13 where the total number of environment interaction steps experienced by the parallel approach is counted instead of the number of training time steps for each model).

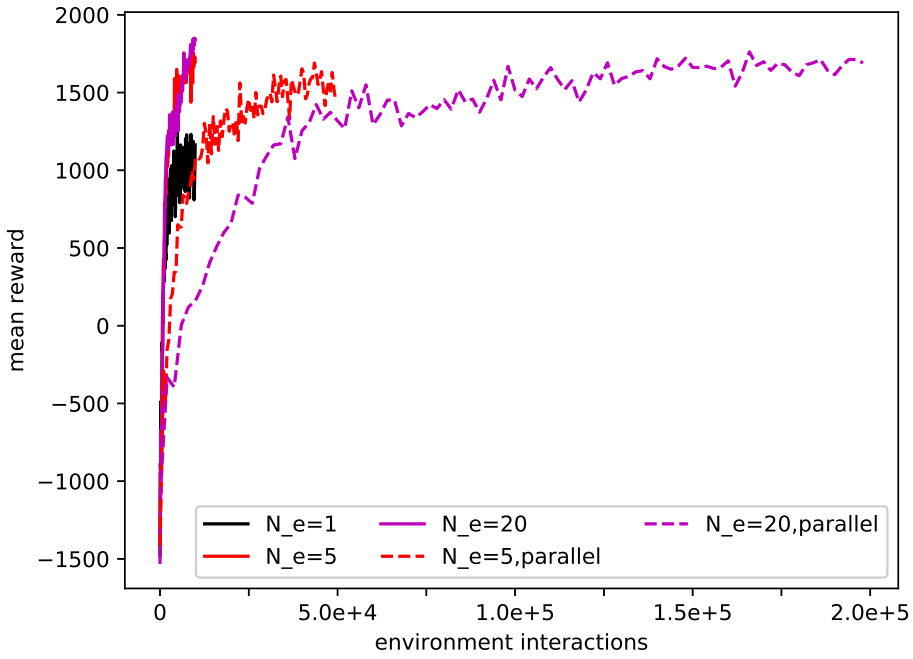


Figure 4.14: Comparison of crowd ensemble with historical experience replay, Fauber & Schwenker (parallel) approach, and single Q-learner. Showing all 5×10^6 time steps of the Fauber and Schwenker approach with $N_e = 20$.

4.5 Crowd ensemble for deep Q networks

The crowd ensemble approach can also be applied to the cartpole swing-up from pixels task via the DQN approach. Some creativity is required to avoid exacerbating by orders of magnitude the already high computational requirements. The first step is acknowledging that the convolutional layers of the network are intended to learn to represent visual features. Therefore we share the features across ensemble members.

This is done by accumulating the gradients across the N_e ensembles for the fully-connected layers and then backpropagating them through the convolutional layers. In addition to dramatically reducing the number of parameters, sharing layers has potential for the convolutional layers to learn features more quickly with the additional gradients.

Figure 4.15 shows a comparison of the mean evaluated rewards for the crowd ensemble and single DQN approaches. It appears that the mean reward of the single DQN method will never

achieve a mean performance equal to a small crowd ensemble size e. g. $N_e \geq 5$. Three things are clear from these results:

1. the crowd ensembles, on average, perform significantly better than the single DQN approach,
2. the crowd ensembles learn more quickly, and
3. the magnitude of the improvement increases as the ensemble size increases.

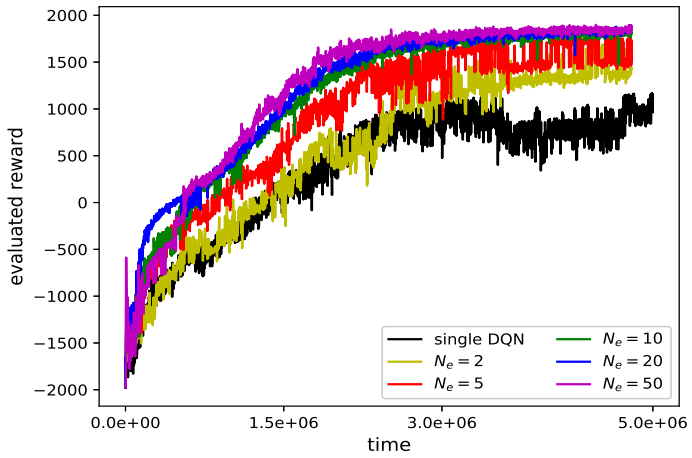


Figure 4.15: Comparison of DQN results against the crowd ensemble DQN approach.

The ability of the respective approaches to solve the task can be evaluated using Figure 4.16. Every 2000 time steps the number of agents that have solved the task at any point during training are counted all the way up to 5×10^6 for the non-ensemble approach and up to 1.5×10^6 for the ensemble approach. In this case, an evaluated reward greater than 1800 is considered solved. The crowd ensemble with $N_e = 50$ is the only one that solves the task each of the five times. As with the mean rewards shown in Figure 4.15 the number of times solved follows a similar pattern where the performance increases as N_e increases. The base DQN approach solves the task within 5×10^6 time steps three of the five times.

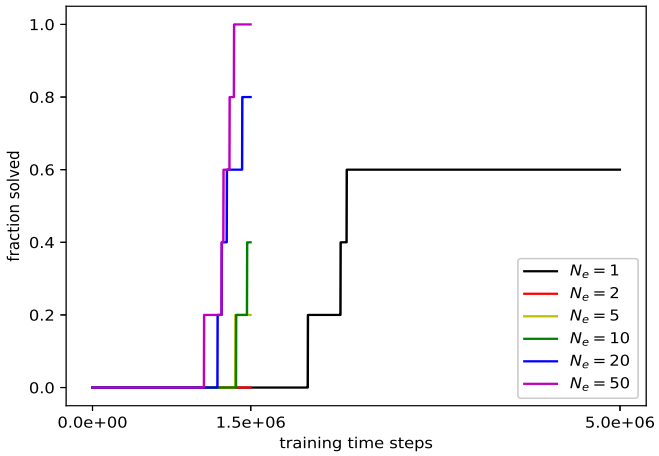


Figure 4.16: Comparison of DQN results from Figure 4.15 against the crowd ensemble DQN approach. Each line shows the number of times five selected agents solved ($R > 1800$) the task within the specified number of time steps.

4.6 Crowd ensemble for biped walker task with actor-critic networks

The crowd ensemble is an effective approach to stabilize and even speed-up Q-learning on the two versions of the cart-pole task. Here we present results designed to highlight how generally applicable this simple approach can be by applying it to the bipedal walker task (introduced in Section 3.4) using an actor-critic RL approach and continuous actions. Furthermore, the ANN used to model the critic's Q-function is larger than the ANN used in the low-dimension cart-pole task. The actor and critic networks are trained using the DDPG algorithm.

The parameters which define the baseline results used for comparison were taken from the supplementary material of Lillicrap et al. [4]. Utilizing an ensemble approach is not as simple as with Q-learning with discrete actions where a simple voting mechanism was utilized to combine the knowledge of the ensemble members. Typically, in an ensemble where the members output continuous values, those values are combined using simple averaging. DDPG, however, views the actor outputs as defining the mean of a Normal distribution with unit variance describing the optimal action at that state. Therefore combining the output of multiple actors in such a naive

way is not a viable option: averaging the outputs of the actors will, most likely, result in a location of exceedingly small probability. A combination of DDPG actors will be multi-modal in four dimensions which will result in a challengingly large number of modes. Finally, there is no straight-forward method to find the highest-probability location in such a mixture so this method will be avoided.

Instead the crowd ensemble approach is applied to DDPG by training a single actor from the combined output of an ensemble of critics. The modified algorithm is based upon the implementation described in Algorithm 3 and is shown in Algorithm 5. The modification is straight-forward: the critics are combined using a mean (or, alternatively, a median) operation. This has the effect of providing the actor with a gradient to follow which is the mean Q-function of the critics.

The algorithm is modified to train each critic Q-function independently just as was done for the crowd ensemble Q-learning implementation. Also notice that each critic has both a critic network and a critic target network.

Figure 4.17 shows the mean evaluation reward for a variety of N_e values. There is an obvious performance increase when increasing N_e from one to two and then two to five. After $N_e = 5$, however, the performance increases are more difficult to come by and the returns have diminished to nearly zero after reaching $N_e = 5$ at the highest. Figure 4.17b shows results for $N_e \in \{1, 2, 5, 20\}$ for clarity. On average the crowd ensembles show faster training and much improved rewards.

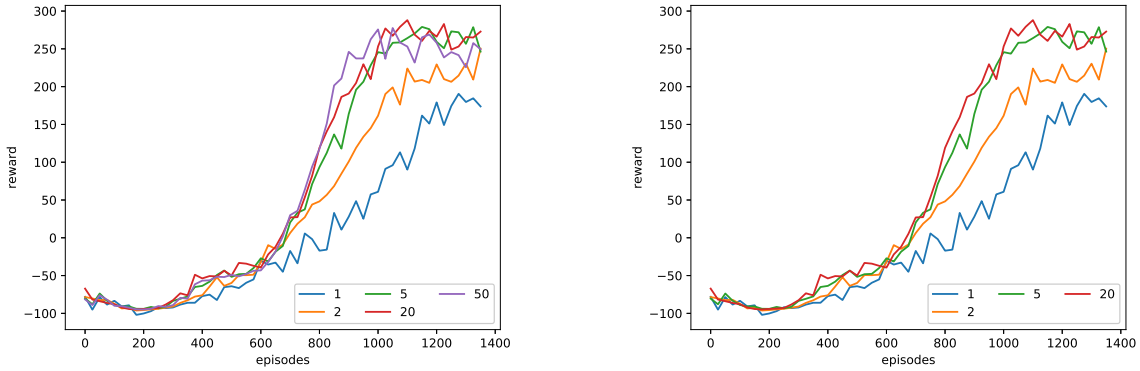
Figure 4.18a shows the median evaluated reward for all N_e values. Again, there is a significant increase in the typical reward experienced by an agent moving from $N_e = 2$ to $N_e = 5$ and then smaller increases for larger N_e values. We can observe that there is a small monotonic increase in the evaluation reward as N_e increases. The median rewards reveal that, even for $N_e = 1$, the typical agent performs better than average but the typical agent does not solve the task most of the time but will occasionally find a solution and then forget. On the other hand, for $N_e \geq 5$ the typical agent solves the task contiguously for nearly 1000 episodes.

Figure 4.18b shows another view of N_e performance. This figure shows the number of agents which have solved the task at least once during training. This shows how early training could be

Algorithm 5: Pseudo-code of the crowd ensemble DDPG algorithm.

Initialize actor (θ^{μ}) and the N_e sets of critic ($\theta^{(Q,c)}$) parameters ;
Initialize target actor ($\theta^{\mu'}$) and target critic ($\theta^{(Q',c)}$) parameters ;
Create an empty replay memory;
$t \leftarrow 0$;
while $t < T$ **do**
    Obtain initial $s(t_B = 0)$, $a(t_B = 0)$, $r(t_B = 0)$;
    $t_B \leftarrow 0$;
    while $t_B \leq N_B$ **do**
        $s(t_B) \leftarrow \texttt{Act}\,(a(t_B - 1))$;
        $r(t_B) \leftarrow \texttt{Reward}\,(s(t_B))$;
        $a(t_B) \leftarrow \begin{cases} \mu(s(t_B)|\theta^{\mu'}) & \text{if evaluation episode} \\ \mu(s(t_B)|\theta^{\mu'}) + \mathcal{N}(t, \mu = 0, \theta = 0.15, \sigma = 0.2) & \text{otherwise} \end{cases}$;
        $s(t_B + 1) \leftarrow \texttt{Act}\,(a(t_B))$;
        $r(t_B) \leftarrow \texttt{Reward}\,(s(t_B))$;
        Store $s(t_B)$, $a(t_B)$, $r(t_B)$, and $s(t_B + 1)$ in memory;
        $t_B \leftarrow t_B + 1$;
        $t \leftarrow t + 1$;
    end
    for $c \in \{1 \dots N_e\}$ **do**
        Randomly draw a set of transitions, $(s(t_B), a(t_B), r(t_B), s(t_B + 1))$ from memory and store in S_u;
        $T_V^{(c)}(t_B) \leftarrow r(t_B) + \gamma * \max_a Q(s(t_B), a|\theta^{(Q',c)}), \forall t_B \in \{1, \dots, |S_u|\}$;
        $\delta_o^{(Q,c)} \leftarrow \sum_{t_B=1}^{|S_u|} T_V^{(c)}(t_B) - Q(s(t_B), a(t_B)|\theta^{(Q,c)})$;
        Update critic using $\delta_o^{(Q,c)}$ as the loss function via gradient descent using ADAM;
    end
    Randomly draw a set of transitions, $(s(t_B), a(t_B), r(t_B), s(t_B + 1))$ from memory and store in S_u ;
    Update actor using $-\frac{1}{N_e}\left(\sum_{c=1}^{N_e} Q(s(t_B), a(t_B)|\theta^{(Q',c)})\right)$ as the loss function via gradient descent using ADAM ;
    Update critic targets: $\theta^{(Q',c)} \leftarrow \tau\theta^{(Q,c)} + (1 - \tau)\theta^{(Q',c)}\ \forall c \in \{1, \dots, N_e\}$;
    Update actor target: $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1 - \tau)\theta^{\mu'}$;
end

stopped if the purpose was to train 50 agents to solve the task. Figure 4.18b shows that $N_e = 1$ never reaches 100% ratio while all $N_e > 1$ agents solve the task. The amount of time it takes



(a) All N_e values.

(b) Selected N_e values.

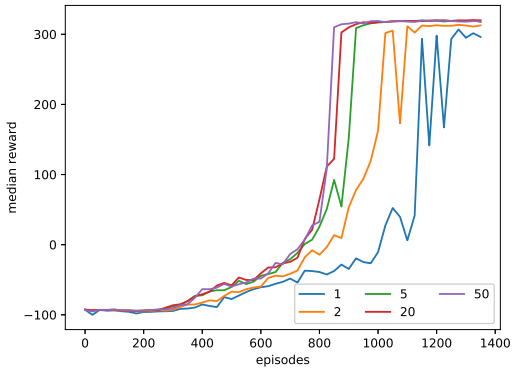
Figure 4.17: Biped via DDPG crowd ensemble N_e comparison. The y-axis is the mean evaluation reward averaged over 50 agents.

for all agents of a given ensemble size decreases with the size of the ensemble with the rate of improvement decreasing as N_e increases.

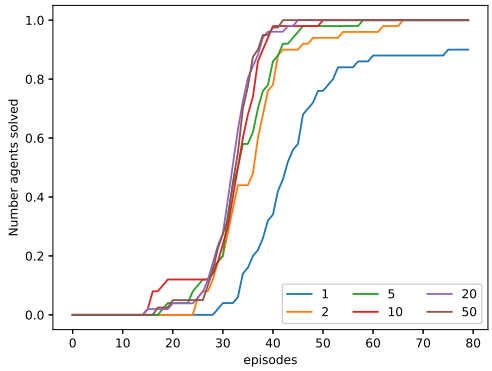
Figure 4.18c shows another view of N_e performance. This figure shows the number of agents that have solved the task peaks at about 88% across all values of N_e. There is a significant improvement from $N_e = 1$ to $N_e = 2$ and then to $N_e \geq 5$.

Although the OpenAI bipedal walker task has recieved attention in blog posts it is found only in a few published results many of which are not comparable to the crowd ensemble. One comparable example is the unplished work Zhang and Zalane where they reached a similar mean reward around $5,000,000$ time steps of training which is 5.5 times more time steps than the approach proposed here.

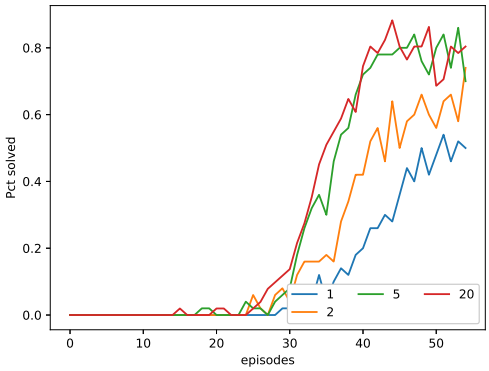
Every 25 episodes exploration is turned off and the agent is evaluated. The task is considered solved if the agent can reach $R > 300$ within 1000 time steps. The larger N_e values solve the task earlier and more frequently and they all perform better than the non-ensemble. This indicates that the ensemble methods provide better solutions and that these solutions are better as N_e increases. All solutions are not created equal, however. The reward computation also take the joint motor actuations into account so more efficient solutions are preferred. Although the ability to solve the task is a significant reason behind improved average reward as N_e increases the larger N_e values



(a) Median rewards



(b) Agents solved.



(c) Percentage of agents solving task at each evaluation

Figure 4.18: Biped via DDPG crowd ensemble N_e comparison. The y-axis is the mean evaluation reward averaged over 50 agents.

also are able to find more efficient solutions as well. Figure 4.19 shows the reward per time step for only those evaluations that were able to traverse the entire plain without falling. This is a measure of the efficiency of the gait of the agent. As N_e increases the typical efficiency shifts to a larger number. The efficiency is computed by dividing the total reward by the number of steps it took to traverse the plain.

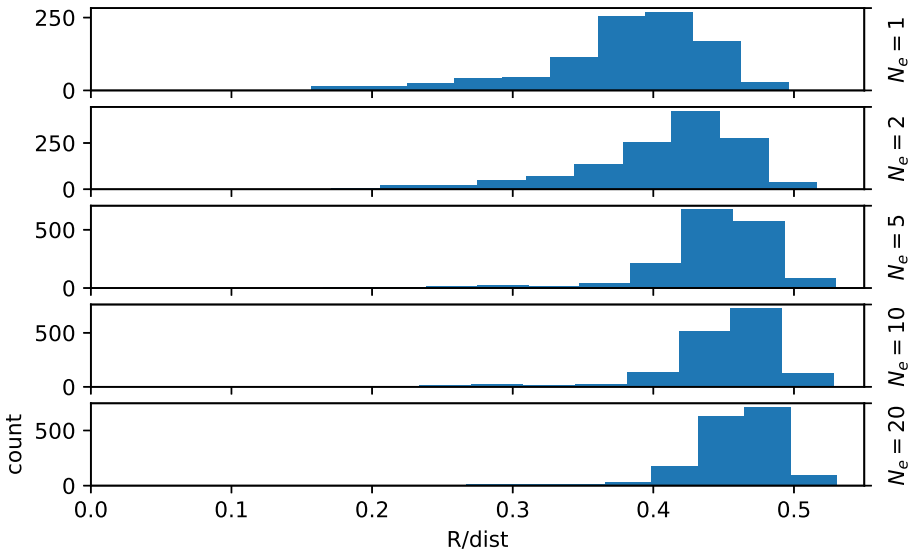


Figure 4.19: Solution efficiency histograms for the non-ensemble approach and the first five N_e values.

4.7 Crowd ensemble for pendulum task with actor-critic networks

The bipedal walker does not lend itself well to analysis due to having 24 state dimensions and four dimensions of continuous actions. Therefore, we apply the same approach described for the bipedal walker to the pendulum task. The pendulum task is a long-standing, well-known RL experimental domain where an inverted pendulum hangs from a single point and the agent's goal is to swing the pole into the goal position using push left and push right actions. The actions are continuous allowing the for variation in the force applied to the pendulum.

The differences between our approaches to the bipedal walker and pendulum tasks are that we train for a maximum fewer episodes because the task is easier to solve, we use shorter episodes, and we use smaller ANNs for the critic and actor. The actor has three hidden layers of size 10, 10, and 5 and the critic has four hidden layers of size 10, 10, 10, and 5.

We initially hypothesized that an ensemble approach would be less beneficial for the pendulum task compared to the others because of the task simplicity. This turns out not to be the case. Figure 4.20 shows a comparison of averaged across 30 runs for each N_e value. We see a very clear benefit to using a critic ensemble for the pendulum task.

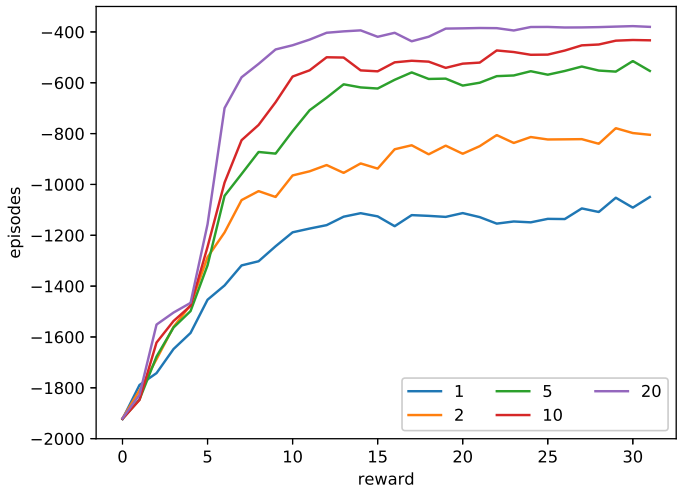


Figure 4.20: Comparison of ensemble sizes against the non-ensemble baseline for the pendulum task. Results are the mean reward during evaluation episodes (every 25 episodes) over 30 agents.

Chapter 5

Discussion

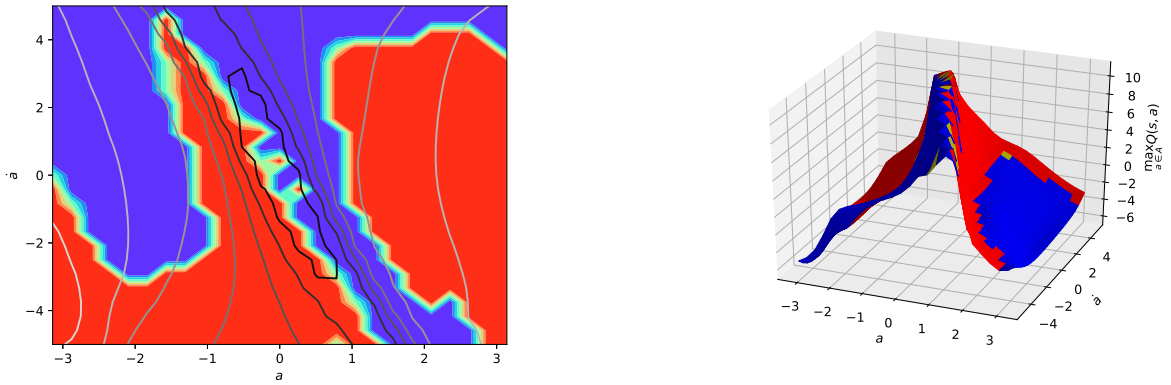
In this chapter we further explore the the reasons behind the results presented in Chapter 4. Chapter 4 revealed that, in all four tasks, crowd ensemble Q-functions lead to better performance: better mean performance, faster learning, 100% of agents solving the task, and more efficient solutions. Bagging Q-functions is a powerful ensemble technique for RL but why should such a simple combination be so generally beneficial?

Much has been made of the challenges of using function approximators to learn Q-functions. We begin there by searching for an explanation for why crowd ensemble Q-functions improve Q-learning in the low-dimension cart-pole task. We find that the crowd ensemble Q-function reduces the number of decision space changes in crucial portions of the space. From there we seek an explanation of how the crowd ensemble Q-functions improves performance when coupled with a policy gradient RL approach. The higher dimension of the bipedal walker task makes analysis difficult but we are able to see indicators that a critic of crowd ensemble Q-functions provides the actor with stability and balance between the actions.

Finally we turn our attention to the crowd DQN used with the high-dimension cart-pole task. Here we see that the ensemble members are diverse and remain that way throughout the lengthy learning process and that this diversity allows us to combine the gradients for faster feature learning.

5.1 The decision space: where the rubber meets the road

Q-functions approximated using ANNs can be fickle – their interconnectedness means that updates from one region of the state space will affect other regions of the state space. If a state is not visited often enough to correct any $Q(s, a)$ errors which have been learned while fitting the Q-function to other states, forgetting will occur. We begin by looking at how crowd ensemble Q-functions affect the decisions made by the Q-learner during training.



(a) $\arg\max_{a \in \mathscr{A}} Q(s,a)$ overlayed with $\max_{a \in \mathscr{A}} Q(s,a)$

(b) 3D view of Q-function with tiles colored according to the decision space

Figure 5.1: Center plots from Figure 3.4 and 3.3 where the Q-function surface is converted to a unfilled contour plot and overlayed on top of the action-selection plot. Two of the four states are represented here: cart position and velocity are held at zero.

We define the decision space as the selected action at each position in the state space. It is half of the function $Q(s,a)$ and it is action selection that is the motivating force behind Q-learning. The decision space is defined as:

$$a(s) = \arg\max_{a \in \mathscr{A}} Q(s,a). \tag{5.1}$$

When applying Q-learning with a discrete set of actions, (5.1) is computed directly given the current state, $s(t)$. In continuous action Q-learning, $Q(s,a)$ can be sampled to determine an estimate for (5.1). In the case of DPG, which we analyze via the biped walker and pendulum tasks, a separate function called an actor is trained to learn this value. In DPG, however, the critic still learns a Q-function from which the actor is trained so the quality of the learned Q-function is critical.

Recalling the example solutions to the four-dimension cart-pole task in Section 3.2, there is an action decision boundary that runs through the region with the highest Q-function values which is more-or-less centered over the goal region. A figure from that section is repeated here in Figure 5.1a. This was taken during training of a non-ensemble Q-learner on the cart-pole task. More of the state space Q-function values and the decision surface can be seen in Figures 3.3 and 3.4.

The Q-function in Figure 5.1a is computed as $\max_{a \in \mathscr{A}} Q(s,a) \forall_{s \in S}$. This is the Q-value of the best action at each state. This Q-function is uni-modal with a rotated, elongated region of highest Q-values which decreases almost monotonically as you get further from the goal. Plotted beneath the Q-function is the decision space (or policy) for the states. There we see a small region where $a = 0$ separates the large, contiguous regions where $a = -1$ or $a = 1$ are preferred meet.

This view of $Q(s,a)$ and its policy fit together nicely and jibe with intuition about the task. Figure 5.1b shows a three-dimension view of the same two functions with the regions of $\max_{a \in \mathscr{A}} Q(s,a)$ colored according to the decision space. Figure 5.1b features prominently the high Q-value region that lies in the goal region. This view emphasizes this exemplar Q-function's strong preference toward states in or adjacent to the goal region.

We see that $Q(s,a)$ differentiates between favorable and unfavorable states well but how does it do when discriminating between actions? Identification of good states is important when learning to solve a task but it is the policy, determined by the relative advantage of actions, which is paramount.

Figure 5.2 shows how the agent moves through decision space for the first 120 time steps of an evaluation. The agent location is the red ellipse. The three grayscale colors represent the selected action at each sampled location of the state space as it travels toward the goal. The decision space changes in the plots as the cart changes position and velocity – each plot is a two-dimension view of the four-dimension space where cart position, s_p, and cart velocity, $\dot{s}_p$, are held constant. The two dimensions of each plot are pole angle (x-axis) and pole velocity (y-axis). The cart position and velocity for each plot is the same as the agent's state at that time. The discontinuity in the pole angle representation where $-\pi$ and π are adjacent to each other means that the right and left ends of the plots are adjacent to each other. Frame 1 is the starting location for all evaluations: cart at the center of the track with zero cart velocity and pole pointing down ($s_a = \pi$ or $s_a = -\pi$) with zero pole velocity. Frames 2-16 show the pole moving upward slightly as the cart drives left into the wall which pushes the pole upward in the opposite direction. At Frame 17 the cart collides with the wall causing a sudden increase in pole velocity (y-axis) in the positive direction causing the pole

to quickly swing back through the down position. The sudden increase in pole velocity ($\dot{s}_a$) puts the agent in $\arg\max_{a \in \mathscr{A}} Q(s, a) = 1$ (push right) territory causing the cart to move underneath the pole and slow its velocity in Frames 18–26. The pole is precariously balanced through Frame 57. In Frames 58–85 the pole swings in a complete circle in the positive direction. The agent balances the pole for the remaining 1915 time steps. While balanced the agent rides along the decision boundary bouncing back and forth between $a = -1$ and $a = 1$ regions. It is kept in the goal state by the large regions of homogeneous action selection on either side of the ridge. These regions effectively form a canyon which the agent will, likely, not escape.

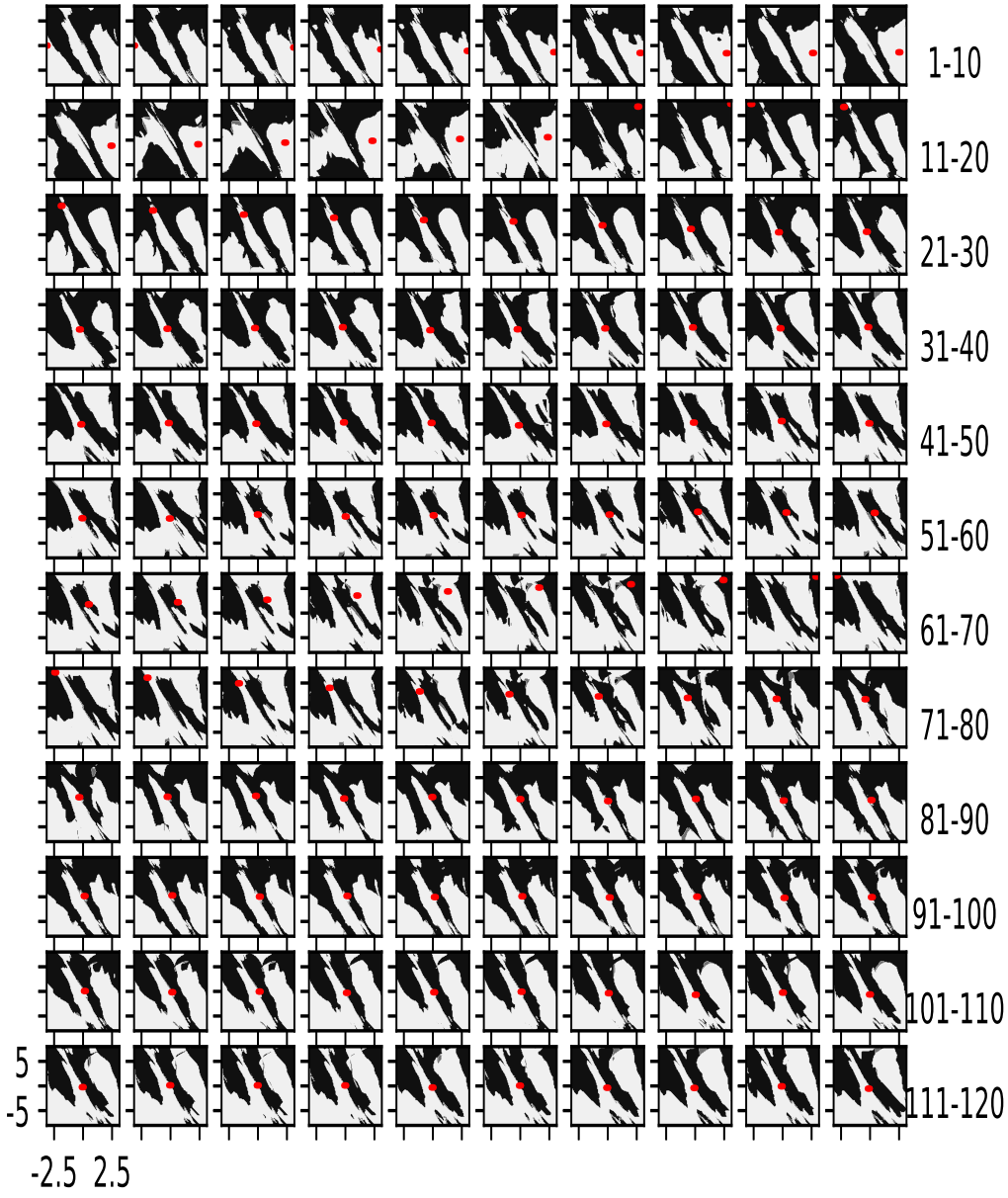


Figure 5.2: Frames of a movie showing an agent's movement through the decision space. Described in greater detail in the text, each frame is a single $(s_p(t), \dot{s}_p(t))$ value while s_a and $\dot{s}_a$ are allowed to change to make up each plot's x- and y-axes, respectively. The red dot is the current $s_a(t)$ and $\dot{s}_a(t)$ for that frame. The rows are labeled by the frame numbers. The light colors is push left, the dark is push right, and the gray is no action.

How stable is this decision surface? Figure 5.3 shows the difference in Q-values across three of the four state-space dimensions. Each plot is a different cart position with zero cart velocity starting from the leftmost and moving to the rightmost position as you view the grid left to right, top to bottom. The difference in Q-functions is defined as $\max_a Q(s,a) - \min_a Q(s,a)$ for each of the sampled state space locations.

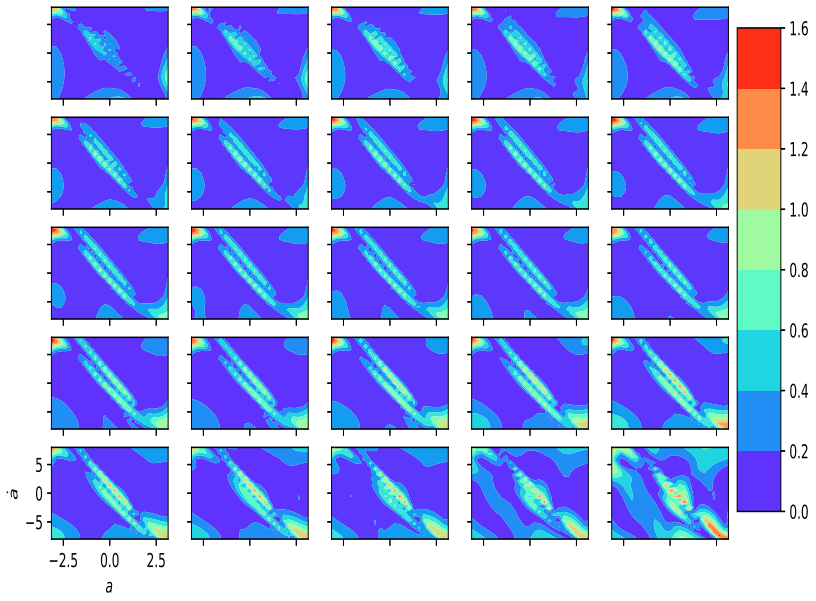


Figure 5.3: Each plot shows the Q-function difference across 25×25 pole angle (x-axis) and pole velocity (y-axis) points for a specific cart position with zero cart velocity. Starting in the upper-left corner and working across the rows, the plots correspond to cart position moving from -2.2 to 2.2.

The state space locations with the highest Q-function differences are those on either side of the decision space boundary which runs through the region with the highest Q-function values. In the center of this region, where the preferred action transitions from -1 to $+1$, the difference between the minimum and maximum Q-function value is small. The remainder of the state space appears relatively featureless with the majority of the spacing having very small $\max_a Q(s,a) - \min_a Q(s,a)$ values. These differences are especially small when compared to

the relief of $\max_{a \in \mathscr{A}} Q(s, a)$ shown in Figure 5.1b. This implies that the selected action at most locations of the decision surface are tenuous and possibly subject to frequent change.

5.1.1 Decision space volatility

Decision space (policy) changes during training are necessary: no changes, no learning. Once a reasonable solution has been learned, however, a reduction in decision space changes will help preserve what has been learned.

At issue here is that the learned Q-functions do not show large differences in $Q(s, a)$ values across $a \in \mathscr{A}$. These small differences are not necessarily incorrect – they are sensible given that, in most regions of the state space, an incorrect action will not prevent the agent from solving the task. Furthermore an agent has some ability to overcome noise in the decision surface; it requires some plurality of incorrect actions, not necessarily contiguous, to significantly impact performance.

That $Q(s, a)$ should change modestly in most state regions across all actions is the cause of much of the forgetting in Q-learning both early in training and once a solution has been found. Small changes in $Q(s, a)$ will upset the delicate balance between the actions at many state space locations.

Figure 5.4 shows a count of the number of times the selected action changed during training *after* the agent had already solved the task with a reward of $R = 1959$. Ideally policy changes should slow down at this point and reinforce this excellent solution. The darkest color corresponds to zero changes while the most red color is 84 changes which corresponds to a change in the preferred action for 76% of the parameter updates. In the goal region there is a line that runs through that transition portion of the state space with a high amount of selected action volatility. This is surrounded on both sides by regions with no change in preferred action The majority of the remainder of the plotted state space locations have high to very high numbers of changes of the decision space.

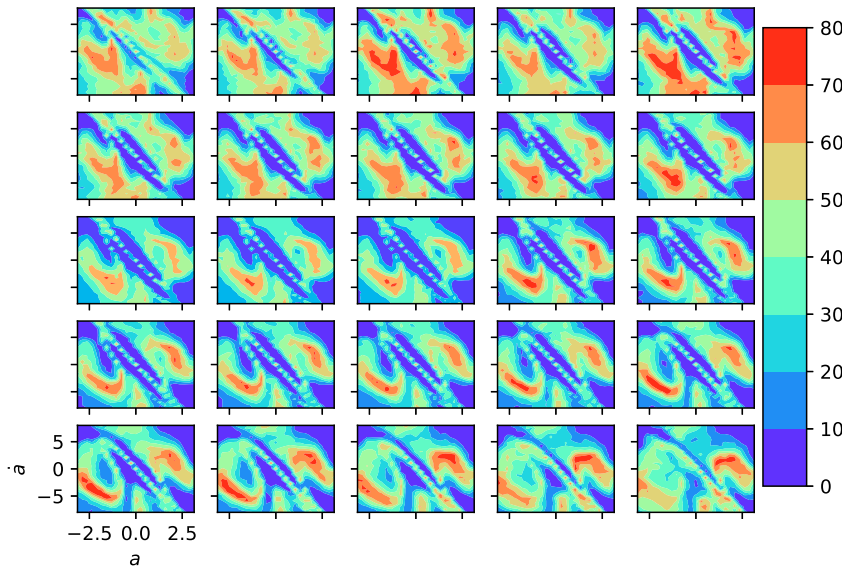


Figure 5.4: Each plot shows the number of times the selected action changed in the 119 evaluations ($119,000$ time steps) after the agent solved the task for the first time. The most blue region is zero changes and the red is 84 changes out of a maximum possible 111 changes. Each plot is 25×31 pole angle (x-axes) and pole velocity (y-axes) pairs for a specific cart position with zero cart velocity. Starting in the upper-left corner and working across the rows, each plot shows a cart position moving from -2.2 to 2.2.

The counts of the changes to the decision space is somewhat of a negative image of the Q-function value difference seen in Figure 5.3 which shows the difference between the maximum and minimum Q-function values. A small difference in Q-function values between the "best" and "worst" actions at a given state presages that the selected action for that state is likely to change. Indeed, these two plots support that intuition. Conversely the important, high Q-value regions in the goal area have larger Q-function value differences and, therefore, do not change frequently. In fact, they didn't change at all in the $119,000$ time steps which is 119×10 applications of scaled conjugate gradient on a randomly sampled subsets of the experience replay memory.

5.1.2 Ensemble decision spaces are less volatile

The difference in volatility between a single Q-learner and an ensemble is evident even after a single parameter update. Figure 5.5 shows the Q-difference values for the cart position and velocity state locations used to generate Figure 5.2 but sampled at half the rate to save space. These states

are particularly important because they are the state space locations that an agent must pass through on its way to the goal.

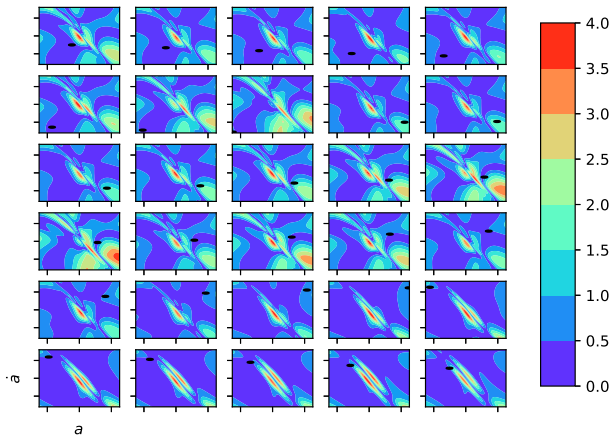


Figure 5.5: Plot of $\max_a Q(s,a) - \min_a Q(s,a)$ for a randomly selected single Q-learner ($N_e = 1$). We posit that this is a measure of decision certainty or stability. The states are the same as those shown in Figure 5.2 but only half the frames (every other) are included to save space.

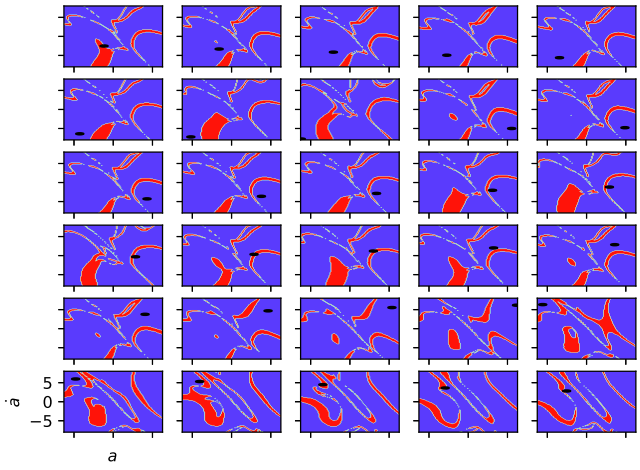


Figure 5.6: Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected single Q-learner ($N_e = 1$). The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5.

Figure 5.6 shows which of the state locations change their preferred action after a single parameter update when trained using a single sequence of 1000 experiences. The state locations shown here are the same as those shown in Figure 5.5. The training samples used to calculate the parameter update were generated by evaluating the agent from the pole down position with zero cart and pole velocity and allowing it to run for 1000 time steps. This was done to get a realistic parameter update scenario instead of applying randomly generated states that may not be experienced when exploiting the agent's knowledge. The updates occurred by performing a single loop of steps 22–24 of Algorithm 1.

As expected when using an ANN to model a Q-function, the changed state locations are not limited to the state locations featured in the training set. Comparing the image to Figure 5.5 shows that the changes occurred where the Q-function difference for the three actions is the smallest.

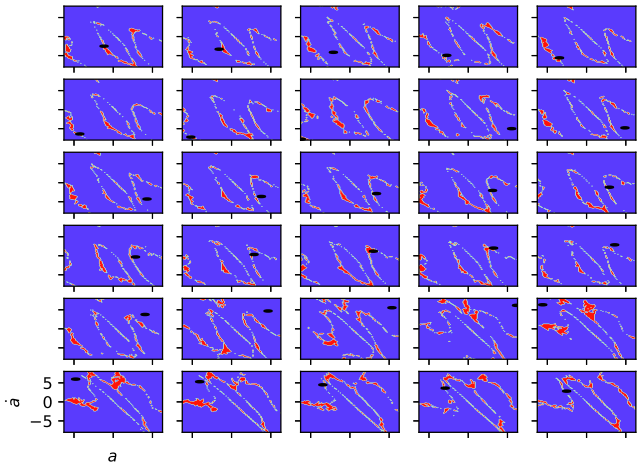


Figure 5.7: Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected ensemble Q-learner ($N_e = 20$). Each ensemble member received a different training sequence. The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5.

Figures 5.7 and 5.8 show the same state space slices with the changes plotted for an ensemble of size $N_e = 20$. Both figures were updated with 20 sequences (one for each ensemble member) of data compared to the single sequence for the single Q-learner. The difference is that Figure 5.7

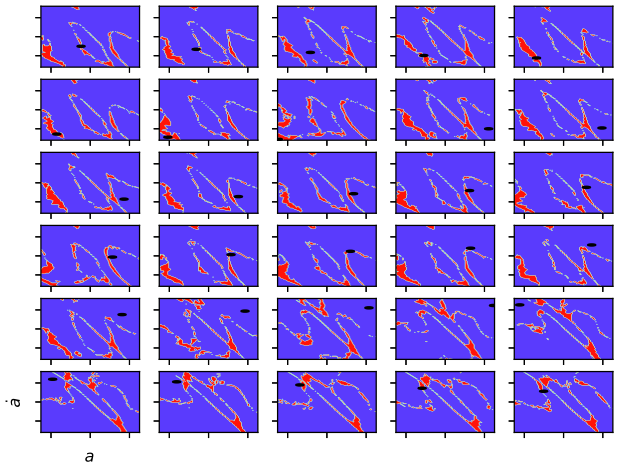


Figure 5.8: Selected states along the path to solution highlighting where the selected action changed after a single parameter update for a randomly selected single Q-learner ($N_e = 20$). Each ensemble member was given the same (s, a) pair sequence with which to update which exacerbated the change count. The blue locations have no change, the red locations had a change. States correspond to those shown in Figure 5.5.

was updated using 20 different sequences while Figure 5.8 was updated using the same sequence 20 times. The 20 different sequences were generated by evaluating each ensemble member on the swing-up task and recording the state history. This gave each ensemble member a unique data set with which to compute the parameter updates which is much more similar to how the ensembles are trained in Algorithm 4. The shared training sequence used by the agent to create Figure 5.8, on the other hand, was generated by evaluating the ensemble only and recording that sequence. For Figures 5.6, 5.7, and 5.8 the weights were selected from agents which had solved the task by achieving $R > 1950$ during evaluation after $t > 1 \times 10^5$ time steps had elapsed to ensure that weights from a well-established and explored solution were used.

Comparing the three figures it can be seen that the ensemble reduces the number of updates for both update schemes. Comparing 5.7 and 5.8 it can be seen that an additional reduction in decsion-space instability is realized when giving each ensemble member a different set of training samples which is an indicator of the value of ensemble member diversity.

The total number of state space changes across 1.2×10^6 positions in state space for the three computations is shown in a bar graph in Figure 5.9. The bar graph supports what Figures 5.6, 5.7,

and 5.8 show us: that the number of decision space changes decrease with the use of an ensemble. Furthermore, allowing the ensemble members to sample, independently, from the experience replay strengthens the benefit.

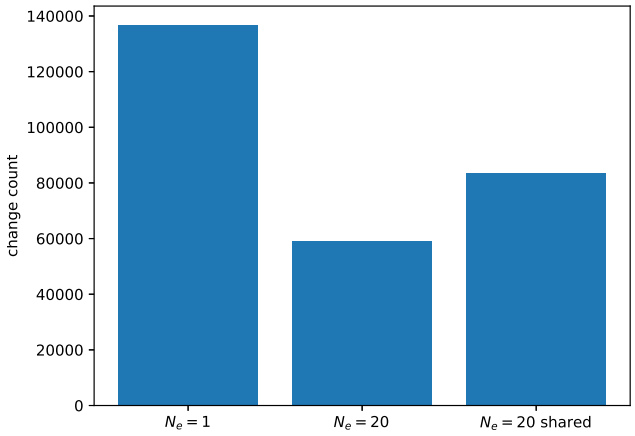


Figure 5.9: Bar plot of a raw count of the number of sampled state space locations which changed after a single application SCG with a data set generated by allowing the agent attempt to solve the task for 2000 time steps. Each bar is a different N_e value with the exception of the bar labeled "$N_e = 20$, shared" which supplied each ensemble member the same sequence of training data. Computed for the entire state space, not just the $(s_p, \dot{s}_p)$ shown in Figures 5.6, 5.7, and 5.8. Total of 1.2×10^6 state locations.

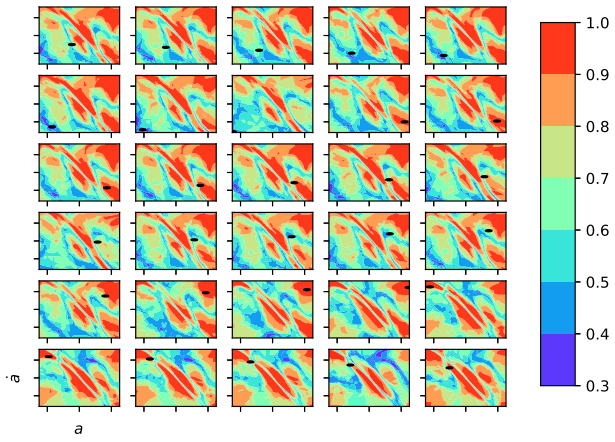


Figure 5.10: Plot of voting majority size as a percentage of the possible ensemble size ($N_e = 20$) for a randomly ensemble single Q-learner ($N_e = 20$).

Figure 5.10 shows the majority size of the ensemble just prior to the parameter update. Just like the relationship between Figures 5.6 and 5.5 where the low Q-value differences between actions coincided with regions most likely to change the selected action, the state space regions where changes occur follow the regions with the smallest majority size. The reason for decreased instability when using an ensemble is that the states with small voting majorities are fewer in number than the states with small Q-function difference: low certainty in the preferred action is still a significant driver of solution instability for both approaches.

The randomly selected ensemble used in this analysis is a terrific example of the power of treating ML ensembles as a crowd. The weights utilized in this analysis had an evaluated reward of $R = 1955$ when operating as an ensemble. At that moment, this reward is higher than all but one of the individual ensemble members when evaluated separately. That ensemble member was the only one which had a reward of $R \geq 1950$. Three of the ensemble members had negative scores. The evaluated reward of the ensemble members were

$$\{1939, 1755, 1959, -737, 1597, 987, 1616, 1919, 1658, 578,$$
$$1690, 1710, 1521, 1080, 1814, 1896, 883, -1349, -1965, 994\}.$$

The policy derived from the Q-function (the decision space) is where the rubber meets the road. Q-learning quickly finds and maintains a quality representation of the relative value of states but it does not differentiate as well between the potential actions available at each state location. The dominating influence of state on $Q(s, a)$ results in a noisy decision space.

Utilizing a vote-based ensemble combination scheme moves the decision space from a weaker $\max_{a \in \mathscr{A}} Q(s, a) - \max_{(2), a \in \mathscr{A}} Q(s, a)$, where $\max_{(2)}$ denotes the second highest maximum value, scheme to a stronger one with larger, more stable decisions. Furthermore, combining using voting alleviates issues associated with independent Q-functions with varying Q-values where one member's Q-function might dominate or a poorly-performing member might skew the average.

5.1.3 Forgetting mostly occurs along the path to the goal

The evaluated reward for a given single Q-function agent varies considerably even after it has already achieved excellent results multiple times during training. Figure 5.11 shows the evaluation history of the entire run for an example agent.

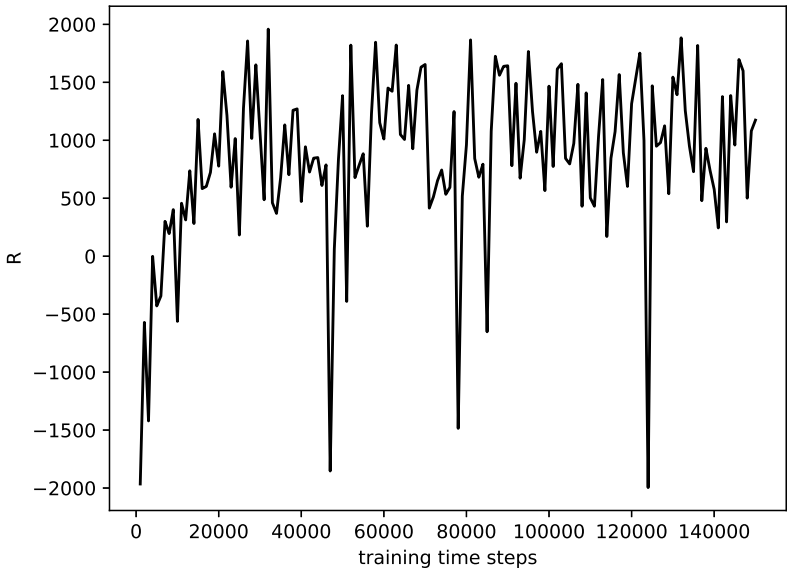
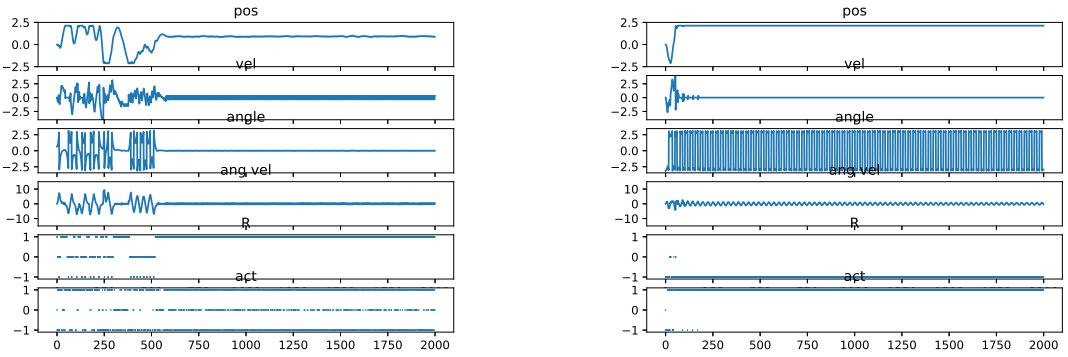


Figure 5.11: Evaluated reward for all 150 parameter updates of the agent used to generate Figure 5.4.

Several instances of forgetting are visible in the 1.5×10^5 time steps. At $t = 122,000$ a particularly bad evaluation occurs. Figure 5.12 shows the performance of that agent on the regular cart-pole task and a similar pole-balancing task where the pole is started in the upward position, $s_a(t = 0) \leftarrow 0.1\pi$, and evaluation is started with two pushes to the right to give the pole a downward velocity. Despite the poor evaluation when started in the usual pole down position, the agent handles the balancing scenario well and is eventually able to stabilize the pole in the balanced position. Making the task even easier by starting the pole at $s_a(t = 0) \leftarrow 0$ shows perfect performance. Why the disparity? After all, in Figure 5.12a, the pole has to pass through the downward position many times before balancing.



(a) Balancing task

(b) Regular cart-pole task

Figure 5.12: Evaluation on the cart-pole task started in pole-up (leftmost plot, started at 0.1π and pushed left to give pole downward velocity) for an agent that has forgotten how to solve the swing-up task (rightmost plot).

The disparity in performance based upon the starting pole position is a result of the decision space volatility in regions that lie along the path to the goal regions. The agent hasn't forgotten how to keep the pole in balance nor has it forgotten how to move the pole into high Q-value states when the pole is already swinging – especially when the pole has a favorable velocity and is heading toward the goal. The issue is that the decision space has been distorted in such a way to prevent the agent from progressing along a path to the goal.

The ability to solve the task when the pole is started in the upward position when achieving very poor performance when started in the down position as a result of forgetting is not uncommon. Thirty seven non-ensemble Q-learners were identified which experienced catastrophic forgetting in the last 2×10^4 time steps of training after having solved the task with a score of $R > 1900$. Catastrophic forgetting was defined as scoring $R < -500$ for at least one evaluation after training for at least 1.3×10^5 time steps after having solved the task at least once. Of these 37, 32 received evaluations of $R < -1600$ or lower when started in the pole down position. Of these 37, 18 of the agents scored $R > 1950$ when started in the upward pole position which means those agents could re-balance the pole after just a single swing around the pole-down position. A histogram of the pole-balancing rewards is shown in Figure 5.13.

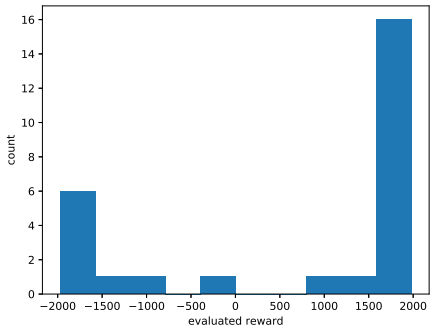


Figure 5.13: Histogram of rewards on cart-pole balance task (started with pointing pole in upward with downward velocity) on agents with low reward on cart-pole task due to catastrophic forgetting.

This is a result of the decision space volatility in the regions surrounding the goal regions in Figure 5.4; the regions which contain the most volatile state space locations. These state spaces must be successfully navigated if the agent is to reach the high Q-value value states.

This intuition extends across multiple runs of Q-learning (ensemble and non-ensemble) on this task. Figure 5.14 shows the same states as Figure 5.2 where the decision space volatility of that location is drawn instead of the selected action or Q-value. Figure 5.14 was created by computing the decision surfaces changes across 20 non-ensemble Q-learners once each had solved the task and continuing until training stopped. The figure shows the fraction of those parameter updates which resulted in a decision space change at that state location. Here we see that an agent must pass through several regions of high volatility on its way to the goal. Once the goal is reached, however, it is safely located between two relatively unchanging regions. These figures show that, in the cart-pole swing-up task, the greatest cause of catastrophic forgetting where the agent appears to have forgotten much of what it has learned is a result of volatility in the regions of the state space that must be visited on the way to the goal.

Figure 5.15 shows the same states as Figure 5.14 but the decision space changes are computed from an ensemble of size $N_e = 20$. The plots are scaled to keep the colors consistent between figures. In fact, the maximum fraction of time steps that a particular location in state space changes its selected action is nearly identical for the $N_e = 20$ and $N_e = 1$ case: just over 47% of the parameter updates.

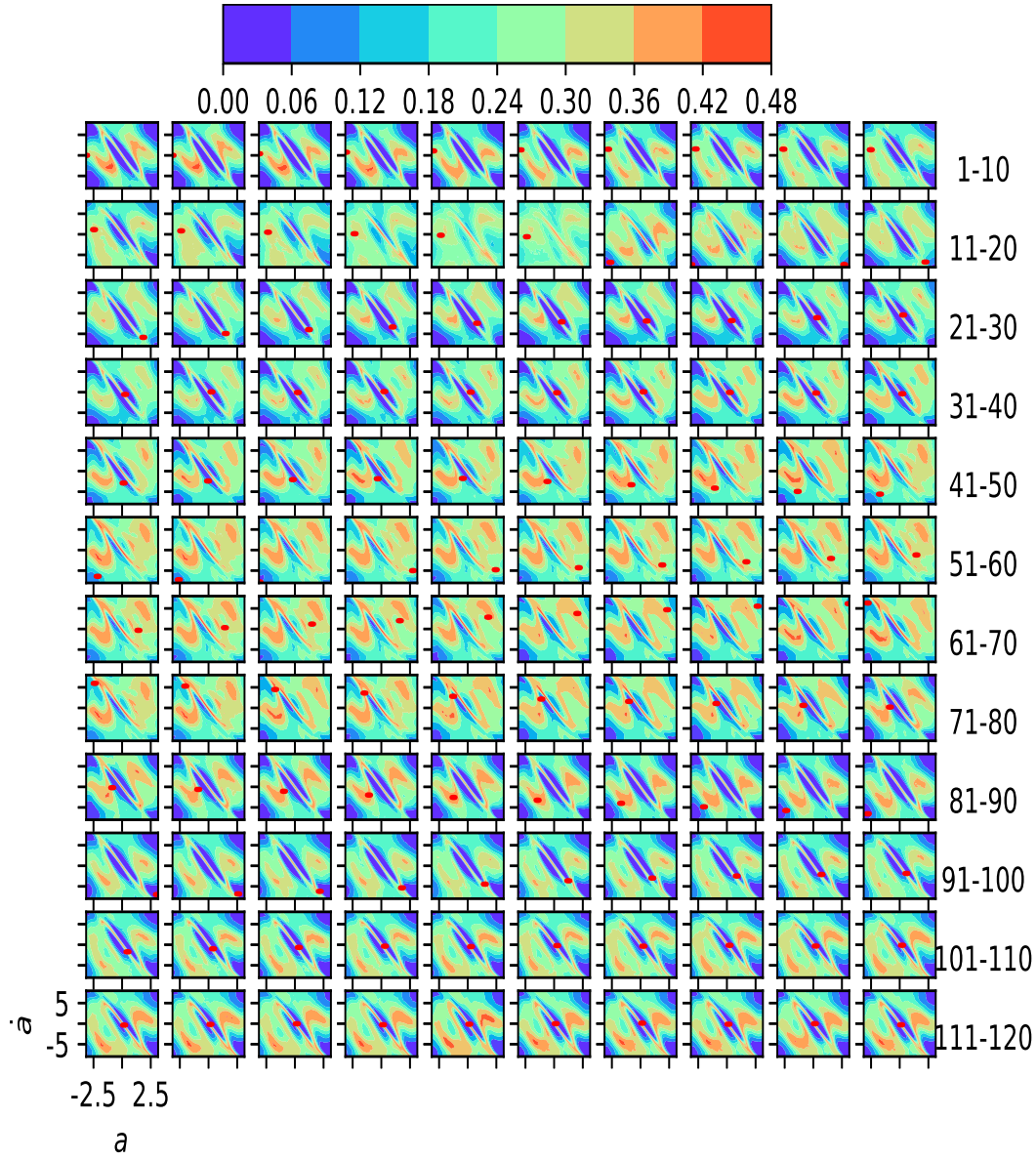


Figure 5.14: Frames of a movie showing the agent's movement through the state space for an example run of a single Q-learner ($N_e = 1$). Each Frame is a single time step. The first 120 time steps are shown. State space locations are colored according to the number of times the preferred action at that location changed. The red dot in the agent's current location in state space. Color bar is scaled to match colors in Figure 5.15

The difference in state space volatility is striking! The ensemble has not removed volatility but it has mitigated it considerably. Furthermore, the regions of Figure 5.15 which have the most instability are the regions where it matters least: namely the decision surface boundary in and around the goal region which is surrounded by regions of low decision space volatility. The other regions of high decision space volatility for the ensemble is when the pole is pointed downward

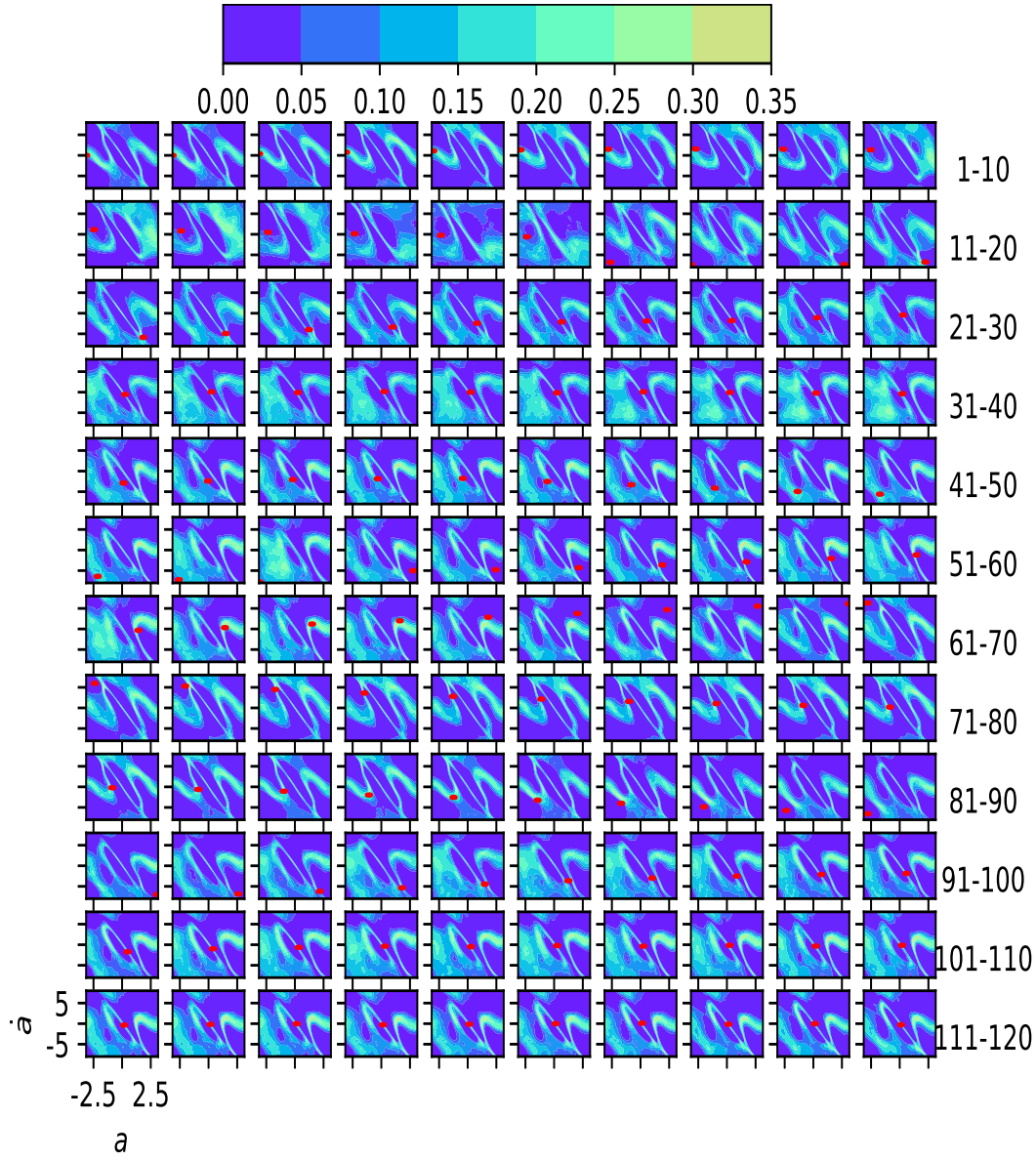


Figure 5.15: Frames of a movie showing the agent's movement through the state space for an example run of an ensemble Q-learner ($N_e = 20$). Each Frame is a single time step. The first 120 time steps are shown. State space locations are colored according to the number of times the preferred action at that location changed. The red dot in the agent's current location in state space. Color bar is scaled to match colors in Figure 5.14

with little angular velocity where the action decision had little impact. This explains why, as the number of ensemble members increases, the majority size decreases in Figure 4.8: large ensembles hold the agent in this small region of volatility for a larger fraction of the episode.

5.2 A non-ensemble Q-function is relatively plain in the directions of the action dimensions

The Q-functions are susceptible to change in decision surface because the actions do not make a large enough impact. The Q-functions are dominated by the state. The effect on $Q(s, a)$ by the actions, on the other hand, is fairly bland.

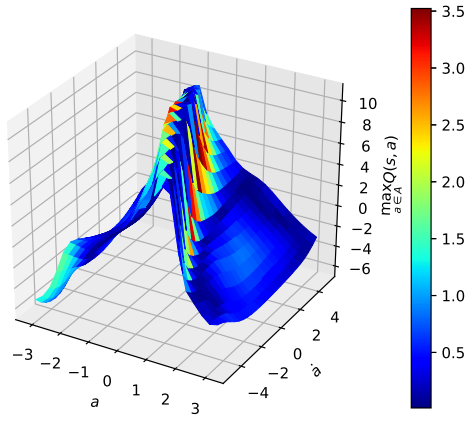


Figure 5.16: Surface plot of $\max_a Q(s, a)$ with each tile colored according to $\max_a Q(s, a) - \min_a Q(s, a)$. This image is included to show the magnitude of the Q-function relief across states relative to the Q-function difference across actions.

Figure 5.16 shows a three-dimension view of the Q-function colored by the Q-function difference at that location for that cart-pole task. Small differences in Q-function values across actions rule most of the space with the exception of either side of the ridge of high Q-values. This figure shows the small differences in Q-values of competing actions even while the amount of change in the Q-function is relatively large.

The reason behind this is that the Q-function values for each action are similar but are shifted with respect to each other. Figure 5.17 shows three contour plots, one for each $Q(s, a = -1)$, $Q(s, a = 0)$, and $Q(s, a = 1)$. The three surfaces are so similar they cannot be viewed in three dimensions. This gives the Q-function regions with high Q-value relief larger $\max_a Q(s, a) -$

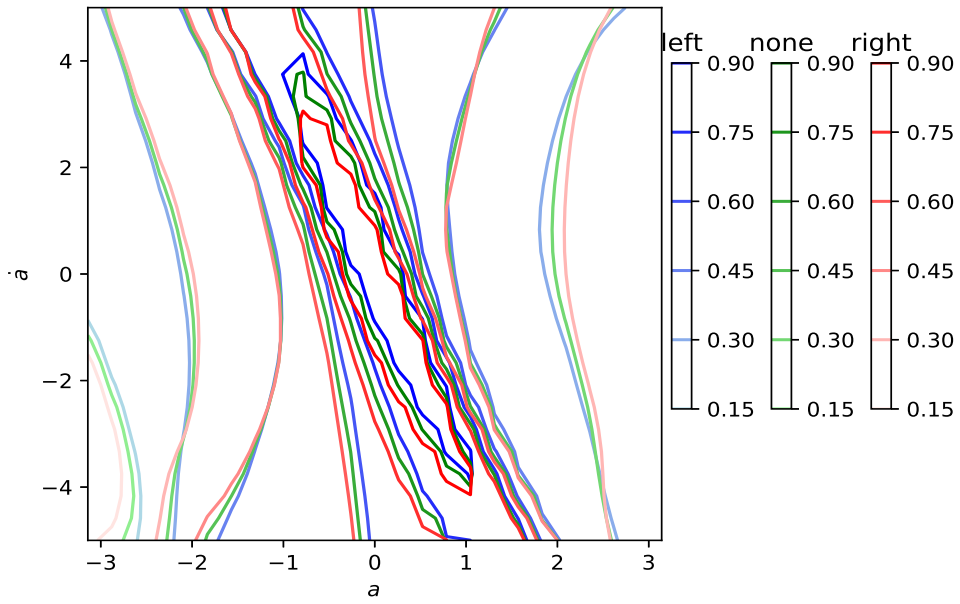


Figure 5.17: Three contour plots of $Q(s, a = -1)$ (blue), $Q(s, a = 0)$ (green), and $Q(s, a = 1)$ (red).

$\min_a Q(s, a)$ values and regions with little Q-function relief $\max_a Q(s, a) - \min_a Q(s, a)$ values near zero.

The small Q-function differences between actions gives the agent very little ability to cope with the Q-function deformations that will occur during training as well as the noisy Q-function changes that will occur as a result of imperfect experience replay sampling. As discussed above the combining crowd ensemble members using voting overcomes this decision-space noise by moving the action selection mechanism to a much more stable space.

This also affects deterministic policy gradients on the continuous action pendulum task. The pendulum task's two state dimensions are the pole angle and its angular velocity. Compared to the bipedal task it is relatively simple: typically capable of being solved within 200 episodes The DPG/actor-critic approach to solving this task is discussed in Sections 3.5 and 4.7. The approach used to solve this task is very similar to the approach used for bipedal walker to keep the analysis as relevant as possible. Recalling Figure 4.20, the ensemble size had a profound affect on the mean and median performance of the agent despite the task's relative simplicity. These results did

not shed light on the way the crowd ensembles were better or why increasing the ensemble size had such a clear-cut improvement on the task. However it is clear that the motivation for using a crowd ensemble cannot be explained by the reasons discovered when analyzing the low-dimension cart-pole task's results because the ensemble does not extend to the actions and there is no voting mechanism. The crowd ensemble is only used to model the critic's Q-function value and they are combined using a mean.

It is reasonable to expect that improved performance must be a result of superior Q-functions learned by the critic ensemble. First, let's take a look at the typical journey of an actor-critic without an critic ensemble. Figure 5.18 shows the learned Q-functions during training for a non-ensemble critic with each Q-function overlayed with the decision-space showing the output of the actor. Because the critic Q-functions take an action as an input, it is plotted for the action selected by the actor (unlike the cart-pole ANN which can output $Q(s, a)$ for all actions).

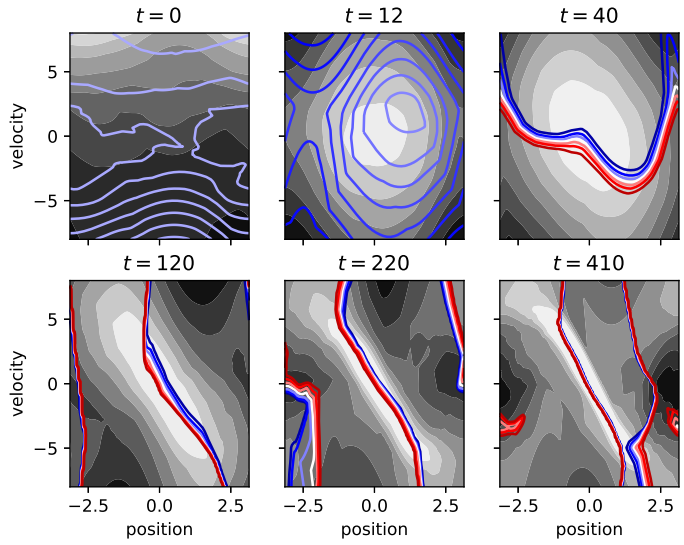


Figure 5.18: Contour plots of critic Q-functions on the pendulum task during training overlayed with the actor output as an unfilled contour plot. The $Q(s, a)$ values are shown for the a selected by the actor for that given s. The red and blue lines of the actor-output contour plot are the extreme values of $a = -2$ and $a = 2$. The gray lines are the more moderate action values.

The critic finds the goal region almost immediately and the contours of the actor quickly follow. It doesn't take long for the critic to find the typical, tilted goal region and the region is refined up through image $t = 220$. The actor has relatively steep decision boundary running through the center of this region where it quickly transitions from pushing to the left to pushing to the right. The machinations of the actor decision space from $t = 220$ through $t = 410$ show a lot of change but those changes do not improve performance.

Figure 5.19 shows the ensemble critic Q-function in the same way as Figure 5.18 and it tells a very similar story. Here the critic is comprised of $N_e = 10$ ensemble members. The critic finds the

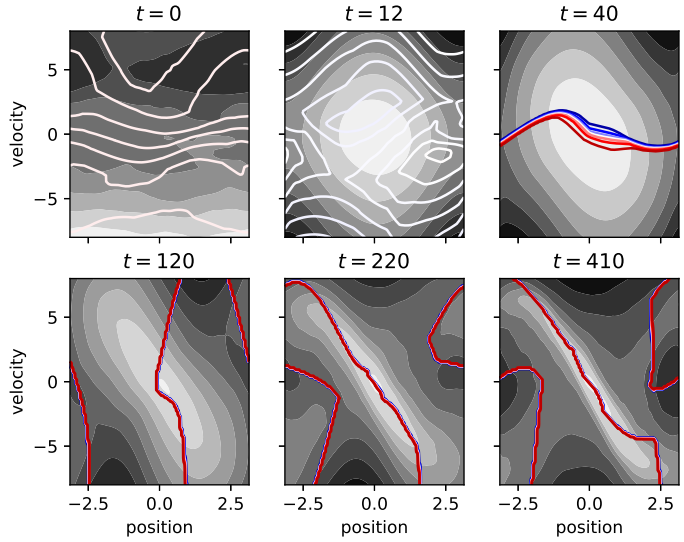


Figure 5.19: Contour plots of critic Q-functions on the pendulum task during training overlayed with the actor output as an unfilled contour plot. The $Q(s, a)$ values are shown for the a selected by the actor for that given s. The red and blue lines of the actor-output contour plot are the extreme values of $a = -2$ and $a = 2$. The gray lines are the more moderate action values.

goal region and eventually shapes the Q-functions to cover the state space regions of high Q-value. The only noticeable difference between these plots other than the differences you encounter when training two independent agents on the task is that the decision space boundary learned by the single actor is even steeper than those learned by the non-ensemble critic version. What, then, is

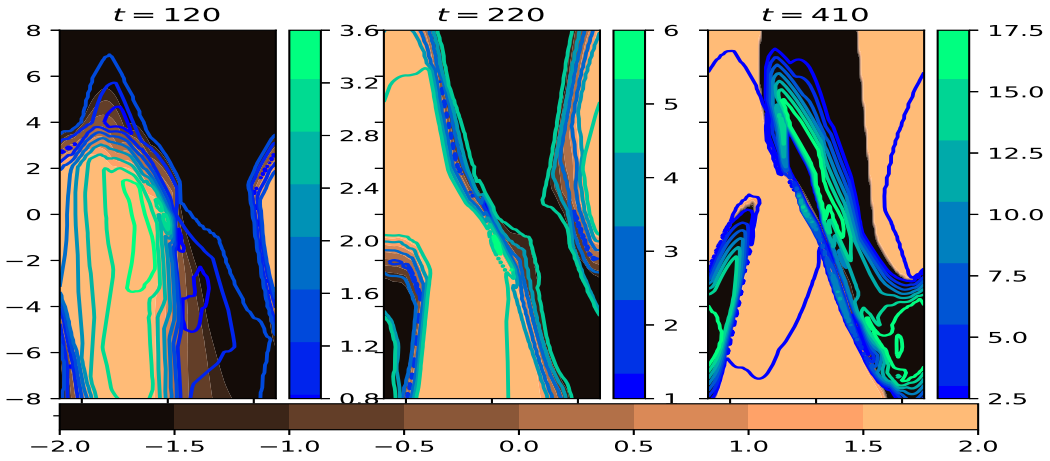


Figure 5.20: Contour plots of selected action (actor network output) in black and brown overlayed with the critic Q-functions differences between the best and worst actions as an unfilled contour plot in green and blue.

the difference between what these two approaches have learned that gives the ensemble method an advantage?

DPG learns a policy by passing the gradient of the critic to the actor to update parameters. In Figures 5.18 and 5.19 we see that the actor is reacting to the critic's $Q(s, a)$ output. As with the cart-pole task and its discrete actions, the advantage is in how the critic ensemble models the effects of the actions in $Q(s, a)$.

Figure 5.20 summarizes the typical behavior in three selected time steps during training of a single agent for a non-ensemble critic. The plots display the decision space of the actor overlayed with the maximum difference in the critic Q-function at that location. The maximum difference is computed by discretizing the action space into 11 discrete actions equally spaced in $[-2, 2]$ and storing those as set $\mathscr{A}$. The maximum difference is computed as:

$$Q^{diff}(s, a) = \max_{a \in \mathscr{A}} Q(s, a) - \min_{a \in \mathscr{A}} Q(s, a).$$

Very early in training one of the actions dominates the regions of highest $Q^{diff}(s, a)$. As training progresses one of the actions still dominates all the way through $t = 200$. This domination

is important because these differences affect the preference of the actor as they are the portion of the critic's Q-function gradient that the actor uses to learn a policy. For a short period around $t = 220$ both actions are equally represented in terms of high $Q^{diff}(s, a)$ regions. However this condition is short-lived. By $t = 250$ the preferred action is switched and $a = -2$ now occupies the regions with the highest $Q^{diff}(s, a)$ values. Action $a = -2$ dominates until training terminates after 500 episodes. In the image for $t = 410$ the $a = -2$ regions are preferred with extraordinarily steep Q-value differences of an order of magnitude of 15 times larger than the surrounding regions associated with $a = 2$. In some other examples the domination of regions with high Q^{diff} stays with the same, extreme action for the duration of training. None of these show balance between actions, however.

Compare this to the DDPG/actor-critic approach to this task when using an ensemble of critics to learn the Q-function. Figure 5.21 shows a randomly-selected, yet representational Q-difference plot (similar to Figure 5.20) for an ensemble of size $N_e = 10$. The ensemble critic balances the

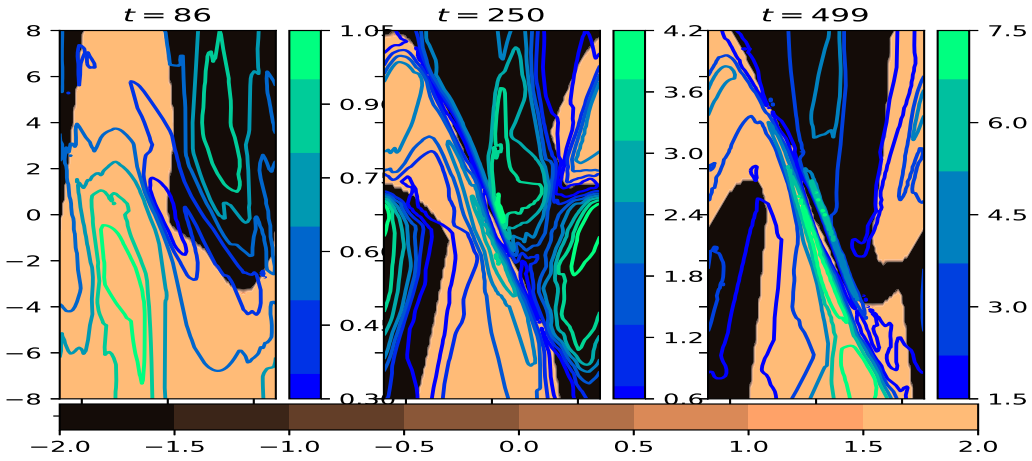


Figure 5.21: Contour plots of selected action (actor network output) overlayed with the combined ensemble critic Q-functions differences between the best and worst actions as an unfilled contour plot.

Q-function between the positive and negative direction actions for all three time steps drawn. This balance is important for learning a policy which properly values the equality of the available actions and isn't skewed toward one action or another. Also, the shift from preferring one action direction

to the other will send the actor confusing signals which will result in degraded performance or even catastrophic forgetting.

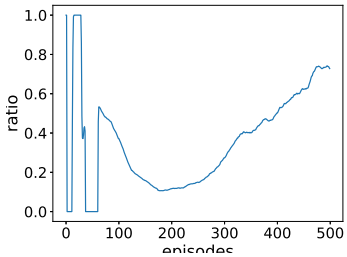
In a task such as the pendulum task one would expect the Q-function values of the positive and negative actions to be roughly equal when considering their relative advantage across the entire state space. In the critic ensemble approach this is what is found. On the other hand, the single critic approach does not balance the Q-function values between the two potential action direction.

Figure 5.22 attempts to summarize this observation across ten randomly drawn agents: five ensemble and five non-ensemble critics. Here we attempt to quantify the imbalance between the actions with respect to the gradients returned by the critic to the actor. This is done by computing something we call a Qdiff fraction which is the sum of the $\max_a Q(s,a) - \min_a Q(s,a)$ values for all states where $\mu(s) > 0$. This value is divided by the $\max_a Q(s,a) - \min_a Q(s,a)$ value across all states. This value is shown in Figure 5.22a for a non-ensemble example and Figure 5.22c for an ensemble example. A value of 0.5 is perfect balance between the two action directions.

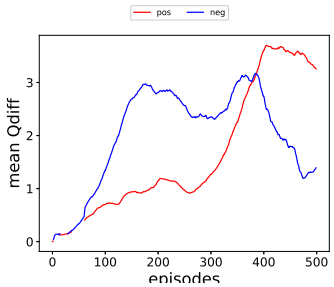
Comparing Figures 5.22a and 5.22c what stands out is how the ensemble critic settles into a ratio of around 50% meaning that the Q-function is equally representing the two action directions. Meanwhile the non-ensemble critic swings from preferring the negative action direction to preferring the positive.

Figures 5.22b and 5.22d show a similar measurement: the mean $Q(s,a)$ value for all states with $\mu(s) < 0$ (negative actions) plotted with the same value for all states with $\mu(s) > 0$. Given the actions are equal one would expect the values to be similar throughout training. Comparing Figures 5.22b and 5.22d what stands out is how the mean Q-function for the ensemble, shown in Figure 5.22d, values for the two directions are generally increasing and doing so at a similar rate. Figure 5.22b, on the other hand, shows the actions making large swings in their mean Q-function values as the agent's critic changes from preferring one action direction to the other. Figures 5.22a and 5.22b show what we saw in Figure 5.20 – the critic shifts between preferring one action direction over the other while only temporarily achieving balance when transitioning between the two. Figures 5.22c and 5.22d show that the ensemble critic example achieves balance between

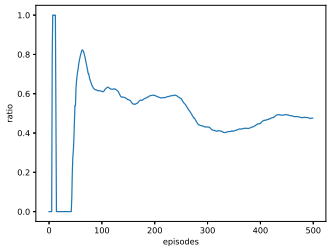
the two action directions. Figure 5.22e plots the Qdiff fraction for the larger of the two directions averaged across five ensemble and non-ensemble agents. The lowest possible value is 0.5 since it always uses the larger of the two fractions. A value closer to 0.5 is more balance between the two actions. The mean Qdiff fraction is close to 0.5 for the ensemble critic while the single critic ratio is considerably higher.



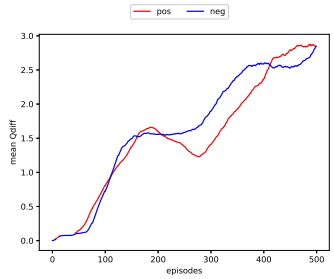
(a) $N_e = 1$ Qdiff mass ratio



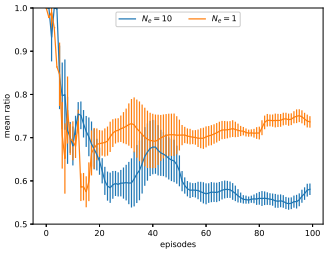
(b) $N_e = 1$ mean Qdiff



(c) $N_e = 10$ Qdiff mass ratio



(d) $N_e = 10$ mean Qdiff



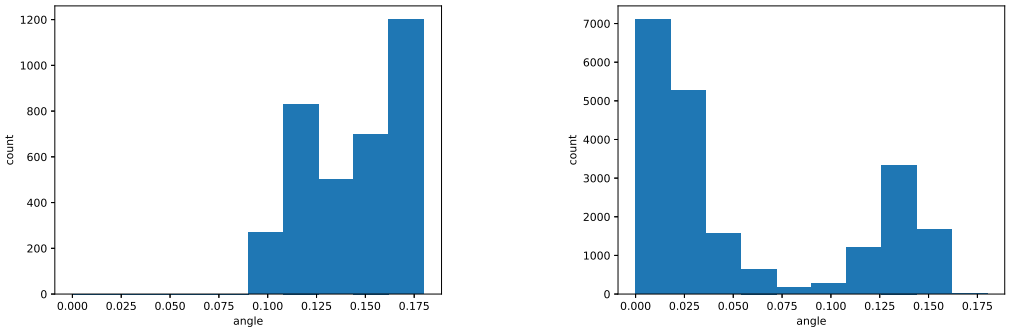
(e) Qdiff mass ratios for both N_e values averaged across ten agents.

Figure 5.22: To top row shows data collected from an example, single critic agent and the second row from an ensemble critic agent. Column one shows the ratio of the Q-function values of a randomly-drawn agent for the positive direction actions over the negative direction actions over the 500 training episodes. Column two shows the average Q-function value for locations where a positive or negative direction action is preferred by the learned policy for the same agent. The plot in the third row shows the mean ratio size of the action direction with the largest ratio averaged across all five randomly-selected agents.

Similarly to what was observed in the Q-learning approach to the cart-pole task, a DDPG/actor-critic approach to RL does a good job of finding states with high value but its representation of the action component of the $Q(s, a)$ function is less than ideal. In the discrete-action cart-pole task the action component of the Q-function was prone to frequent change due to the relatively small difference in Q-function values for the available actions. In the continuous-action pendulum task the policy can show a lot of variation in the decision space. Instead the issue is that one action is generally preferred over the other by the critic's Q-function causing the actor to learn a policy which is suboptimally slanted toward that same action. Furthermore, the action preferred by the Q-function can flip during training causing the policy updates to be contradictory and confusing which will hinder the training process.

The effect of this imbalance is evident in the quality of the solutions. The non-ensemble never learns to solve the task by balancing the pole at angle zero (pole pointing straight up) instead settling for an off-balance position leaning in one direction and applying the same action repeatedly to keep upright. By contrast the ensemble learns to apply both actions to keep the pole in a balanced position. Figure 5.23 shows histograms of the angle positions for five randomly-selected ensemble and non-ensemble agents. The histograms show counts of angle positions as an absolute distance from zero angle for the final 50 time steps of evaluation after the pole has been swung up and balanced for all agents receiving $R > -400$ after 400 training episodes.

A non-ensemble critic does not model the $Q(s, a)$ function in a way that the actions are represented equally. Instead, the critic does a good job modeling the effect of the states on the Q-function while neglecting the complexity the action dimensions bring to that function. Although the ensemble members also prefer one action direction over the other, the Q-function averaging is enough to counteract these individual biases and create a Q-function that is more robust in the action dimensions, not just the state dimensions.



(a) Non-ensemble.

(b) Ensemble.

Figure 5.23: Comparison of the angle the pole is held after an agent has solved the task for the non-ensemble and ensemble approaches. The ensemble approach utilizes both actions to keep the pole pointing straight-up much of the time while the non-ensemble approach always favors one action over another and always balances the pole off-center. Counts are for five agents in both histograms.

5.3 Ensemble Q-functions lead to greater policy stability during training

In the bipedal walker task actions are continuous, unlike the cart-pole task, and in four dimensions instead of a single dimension like the pendulum task. Furthermore, the bipedal walker task requires that the learned policy is capable of learning a sequence of actions which will move the biped forward while maintaining stability. This makes the analyses of the decision space volatility and policy bias described in Sections 5.1 and 5.2 infeasible for this task. However it is still possible to see the stabilizing effects of a Q-function ensemble by looking at the action vectors of the preferred actions of the Q-function (critic) and the policy function output (actor).

We performed an analysis of the policy output volatility during training for ensemble and non-ensemble DDPG agents on the bipdal walker task. Recalling that the policy is trained to respond to the approximated Q-function (or mean Q-function in the case of an ensemble) which takes a state and action as input and outputs a Q-value, the volatility of the Q-function was measured by quantizing the action space into 625 discrete actions (five points along each action dimension) and referring to the action with the highest Q-value as the preferred action. After each parameter update the magnitude of the change in the Q-function is computed using the L_2 vector norm of the difference between the current and previous preferred actions. Similarly the magnitude of the change in the policy is computed as the L_2 vector norm of the difference between the current and previous policy outputs.

The state space is also continuous so the Q-function and policy changes are computed at discrete locations as well. The state space is 24 dimensions which is not amenable to quantization at any resolution and, even if it were, would include a large number of locations which are either not relevant, unrealistic, or never seen during any reasonable behavior. Therefore, similarly to what was done for some of the cart-pole task analysis, we restrict this analysis to the states visited by three different agents. One of the solutions has a nearly maximum score of greater than 300, the

other 200, and the last below 100 to get a variety of paths through the state space with a total of 1500 samples.

We performed this analysis for ten ensemble agents with $N_e = 10$ and ten non-ensemble agents. We allowed each agent to train for 1500 episodes.

Figure 5.24 shows plots of the Q-function and policy changes after each parameter update for all twenty example agents. The first two columns are the non-ensemble agents' Q-function and policy output changes, respectively. The rightmost columns are the ensemble agents. The y-axis ranges were selected to best show the difference in magnitudes between the ensemble and non-ensemble approaches. It should be noted that the non-ensemble changes exceed the y-axis range for many measurements.

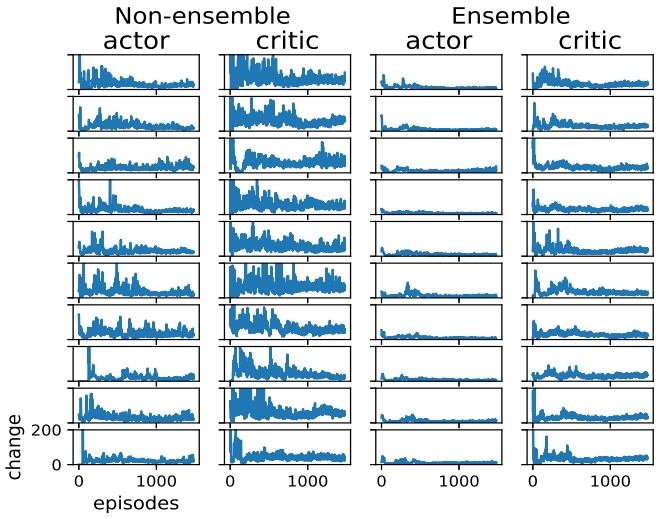


Figure 5.24: Comparison of critic preferred action and actor output changes during training for ten ensemble and ten non-ensemble agents on the bipedal walker task. Y-axis limits were selected to make differences visible; the non-ensemble agents' Q-function and policy output changes are frequently outside the given range.

In Figure 5.24 the Q-function preferred action change of the ensemble agents is smaller than those of the non-ensemble agents. The same goes for the policy outputs. We observe that the Q-function instability of the critic is adversely affecting the actor.

The difference in Q-function instability is visible in Figure 5.25. Here we plot the maximum change in Q-function across all of the sampled states and actions for all twenty agents placing the plots for the non-ensemble agents in the first column. Here we see that the maximum Q-function values for the ensemble agents are monotonically increasing which is evidenced by the downward trend of the maximum Q-function change. This is the expected behavior for Q-learning which indicates that the rate of Q-function change is slowing down as training converges. Conversely, the non-ensemble agents the maximum Q-function changes are not stabilizing and appear to be growing.

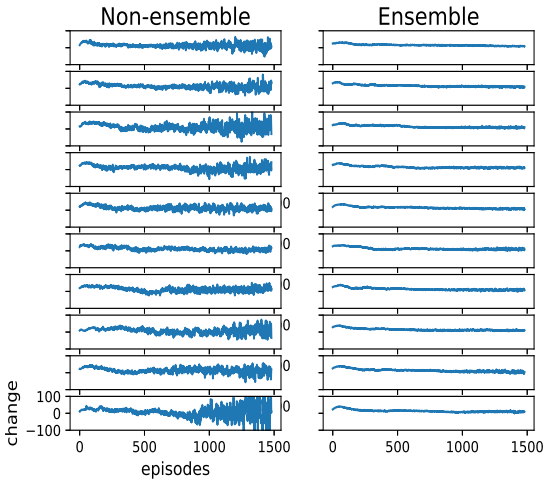


Figure 5.25: Comparison of maximum critic (Q-function) output changes during training for ten ensemble and ten non-ensemble agents on the bipedal walker task. Y-axis limits were selected to make differences visible; the non-ensemble agents' Q-function and policy output changes are frequently outside the given range.

5.4 Shared layers: another opportunity for better collective decision-making

The crowd ensemble approach has a strikingly positive effect on the performance of Q-learning on the high-dimension cart-pole swing up task. The results in Figure 4.15 show a benefit to increasing N_e for all N_e values considered in that experiment. The median reward values show that

the crowd ensemble isn't just better on average, but very quickly finds solutions which are superior to those found with the state-of-the-art DQN approach to high dimension tasks.

5.4.1 Ensemble members are diverse

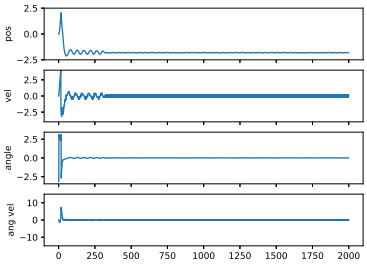
It is reasonable to assume that the crowd ensemble advantage is partially a result of the same decision space stability which aided the low-dimension version. The ANN architecture used in the high-dimension version of the cart-pole task benefits from another aspect of the crowd ensemble which is specific to our implementation of the ensemble with the DQN approach: sharing the convolutional layers across ensemble members. Figure 5.26 shows five solutions from five members of an ensemble of size $N_e = 10$. The ensemble members have been trained using the same data set and, given the length of the training required to solve the high-dimension version of this task, each ensemble member has certainly seen the same data points many times. Despite this, at any given moment during training, the ensemble shows a great deal of variation in the solutions learned by each.

The input layer(s) of an ANN can be thought of as feature transformations on the input data. These features serve as the inputs for later features so learning quality input features is important when using an ANN to model the Q-function. Although this is true for any ANN, the feature-learning aspect of the input layers is formalized when using DQN because it uses convolutional layers as its input layers.

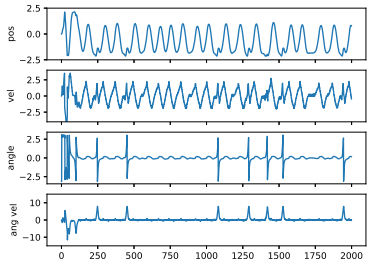
The benefits of sharing input layers is two-fold when using a crowd ensemble. One is that the crowd ensemble trades a reduction in the number of training time steps for an increase in computation time by scaling the number of parameters by a factor of N_e. Sharing the convolutional input layers removes the increase in parameters for the layers with the largest number of parameters. Two, each crowd ensemble member provides gradient information to the shared layer which provides the opportunity for better update directions and larger steps in these directions.

The surprising diversity of the ensemble members is an opportunity to harness additional training information in the form of additional gradients which can be used to speed-up the training time

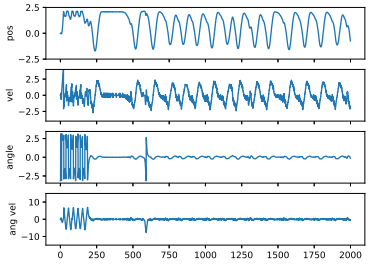
of the convolutional layers' many parameters. In the DQN architecture used in these experiments, the first two layers have 14,300 parameters. Learning a quality state representation is job one for a DQN approach where the state space is raw pixel values.



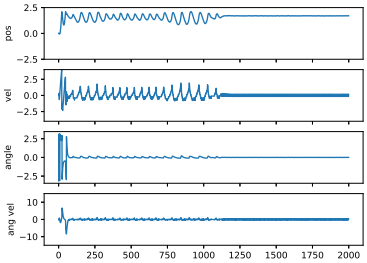
(a) Member one, R=1960.



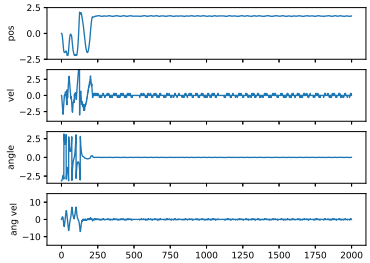
(b) Member two, R=1785.



(c) Member three, R=1726.



(d) Member four, R=1932.



(e) Member five, R=1850.

Figure 5.26: Comparison of all crowd ensemble member solutions for agent with $N_e = 5$ after training for 40,000 time steps. Each sub-figure consists of four plots (listed top to bottom): cart position, cart velocity, pole angle, and pole angular velocity.

5.4.2 Crowd ensemble prevents parameter update zig-zagging

Figure 5.27 shows the magnitude of the distance of the change in the convolutional layers of the DQNs during training. The magnitude of the distance of the parameter changes were computed using the magnitude (as a vector L2 norm) of the difference between parameters at time t and the initialized parameter values. The DQN with $N_e = 10$ has faster and larger changes in the convolutional layers' parameters. Not only do the convolutional layer features of the $N_e = 10$ ensemble get off to a faster start than the single Q-learner, but the divide between the two curves is increasing as training progresses throughout all 1.5×10^6 time steps. The values shown here were averaged over five runs of both N_e values.

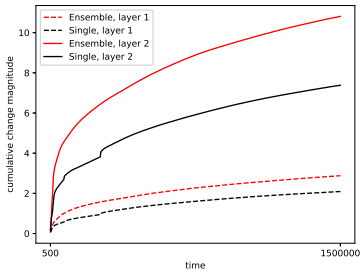


Figure 5.27: Mean, cumulative (computed as distance from initialized values) change in parameter values of two convolutional layers of DQN networks used in high-dimension cart-pole task. Dashed lines are the first convolutional layer (input layer) and solid lines are the second layer. Red lines are features learned by an ensemble of size $N_e = 10$ and black are features learned by a single Q-learner.

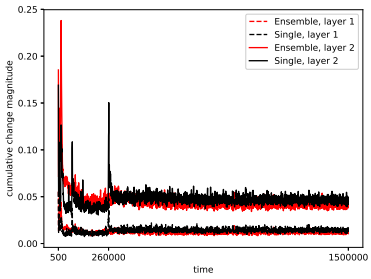


Figure 5.28: Mean change in parameter values of two convolutional layers of DQN networks used in high-dimension cart-pole task (magnitude of weight matrices from time $t-1$ to time t. Dashed lines are the first convolutional layer (input layer) and solid lines are the second layer. Red lines are features learned by an ensemble of size $N_e = 10$ and black are features learned by a single Q-learner.

Figure 5.28 shows the magnitude of parameter changes in the convolutional layers of the DQNs during training. The magnitude of the parameter changes were computed using magnitude (as a vector L2 norm) of the difference between parameters at time t and $t-1$. Unlike the magnitude of the distance from the initial values, the magnitude of the changes of the single Q-leaner are larger than the magnitude of the changes for the ensemble. For the first, approximately, 2.6×10^4 time steps $N_e = 10$ has an average parameter update magnitude which is greater. This is a result of the unpredictability of the single Q-learner approach where some individual Q-learners fail to make progress toward mastering the task until a few hundred thousand time steps have passed. This problem is side-stepped by using ensembles because the likelihood of this occurring to multiple ensemble members is small.

Figure 5.29 shows the parameter change magnitudes for four example runs with $N_e = 1$. The lower-left plot is an example of a Q-learner that struggles to begin learning for over 1×10^5 time steps. Results such as these are responsible for the flip in average weight change magnitude shown in Figure 5.28.

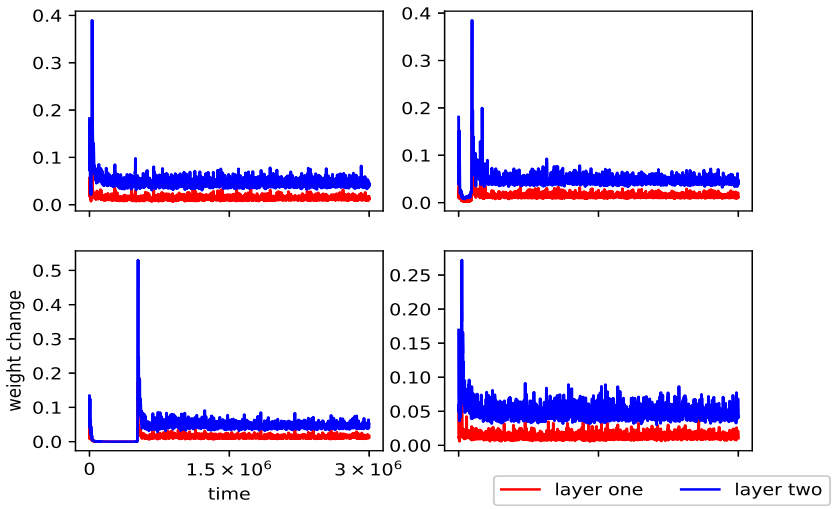


Figure 5.29: Four examples of change in parameter values of two convolutional layers of DQN networks used in the high-dimension cart-pole task (magnitude of the weight matrix changes from time $t-1$ to time t) for $N_e = 1$ networks. Red lines are the first convolutional layer (input layer) and blue lines are the second layer.

Returning to Figures 5.27 and 5.28, the ensembles have smaller convolutional layer parameter updates than the single Q-leaner but have larger cumulative changes. Why might this be? Despite the use of experience replay to pull training samples from multiple regions of the state space and the use of multiple mini-batches during training to include even more experiences, backtracking during training is a drag on the training speed for non-ensemble Q-learning. Backtracking is a sizable fraction of the parameter updates for the single Q-learner updates which results in a smaller cumulative change.

The shared layers of the crowd ensemble DQN have the additional benefit of combining the gradient information from the fully-connected layers prior to backpropagating them to the convolutional layers. The summation of these gradients cancels conflicting gradients directions and compounds beneficial gradient directions resulting in a faster learning of features which are beneficial to Q-learning.

The zig-zagging motion of the single Q-learner and the relatively more direct trajectory of the crowd ensemble Q-learners can be seen by analyzing the weights during training and comparing the changes to a vector starting at the initialized feature values of the two convolutional layers running through their values after 3×10^6 training time steps. This allows us to measure the agent's progress along this vector. Figure 5.30 shows this analysis. Here we can see that the ensemble does much less zig-zagging during training. Instead, it takes a much more direct path to the goal.

Figure 5.30 shows the results for six randomly-selected individuals (three ensemble and three non-ensemble). The weights were saved every 500 time steps in three contiguous sequences during training. Each sequence is 5×10^4 time steps in duration giving each sequence 100 weight updates to compare. The first sequence captures weight updates in $t \in [0, 5 \times 10^4]$, the second in $t \in [5 \times 10^5, 5.5 \times 10^5]$, and the third in $t \in [5.5 \times 10^6, 5.55 \times 10^6]$. For each parameter update the progress along the vector was computed and divided by the total size of the gradient for that update to compute a fraction of the gradient update which travels along the vector. Two ensemble sizes are compared: $N_e = 1$ and $N_e = 10$. The fraction is in the range $[-1, 1]$. A fraction above zero is movement toward the final feature values and below zero is movement away.

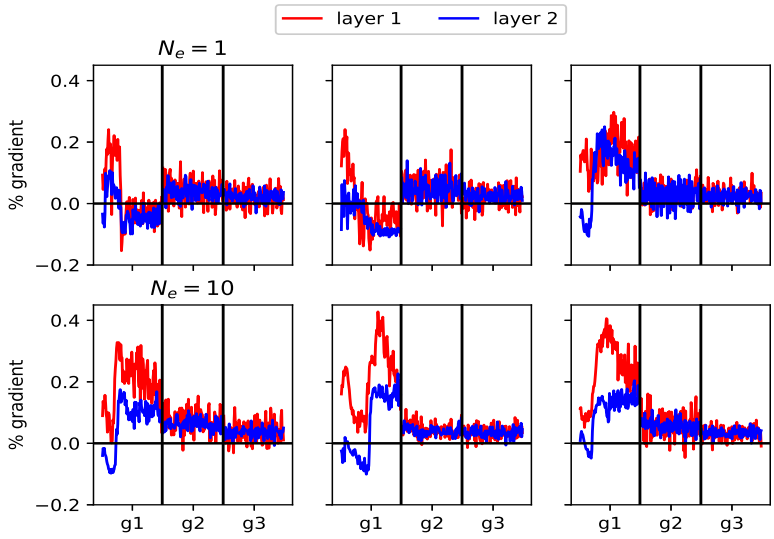


Figure 5.30: Fraction of gradient update movement in the direction of the final parameter values over the total gradient at selected time steps during training of the high-dimension cart-pole agent for two N_e values. For reference, a horizontal line is added to each plot at $y = 0$. The two vertical lines separate the plot into the three groups of sequential weight update samples.

The analysis of Figures 5.30 and 5.27 support each other well. The flurry of progress of the first few updates made by the ensemble Q-learners is visible in both graphs. In Figure 5.27 this is visible by the relatively steep increase in the ensemble (red lines) curves. In Figure 5.30 this is evident from the large positive values in the fraction of movement along the vector compared to total movement of the weights. When discussing Figure 5.27 it was noted that the gap in feature development between the ensemble and non-ensemble approach was still slowly increasing after 1.5×10^6 training time steps. This is supported by Figure 5.30 where the movements along the direction of the vector are very similar in size for both approaches but the changes for the ensemble members are consistently in the direction of the final parameter values whereas the movements for the single Q-learner frequently move away from the final parameter values.

For the first group of samples, in addition to the ratios being higher, the overall size of each update is much larger for the ensemble. For groups two and three the total size of the weight updates in the direction of the final parameter values is the same for the two methods. This means that, after the first 7.5×10^5 training time steps, the significant difference between the two approaches is the less wasted parameter update movements.

Chapter 6

Conclusions and future work

This dissertation set out to formalize and explore a simple crowd ensemble approach to reinforcement learning: bagging Q-functions. We call this approach a crowd ensemble because of its simple combination scheme and lack of specialization of the ensemble members to specific parts of the state space. In particular we wanted to know if this ensemble approach could improve performance by accelerating learning and reducing instability. The crowd ensemble RL technique improves every task and RL approach tested as a part of this dissertation. Although the explanations behind the improvements are different for DDPG and Q-learning, they emanate from the same source: the differences between actions in the state-action value function (i. e., the Q-function) are not well represented. In this final chapter we briefly review the conclusions we glean from the experimental results and analyses presented here. Because this work has opened a handful of doors each with potential for future work, potential directions for future work are discussed along side the conclusions.

We begin by summarizing the crux of this work: that actions play too small a role in the modeled Q-functions and how this can lead to a host of issues. We then discuss future work which could elucidate when a less naive form of ensemble might be useful. This leads to an unanswered question from this work: can a crowd ensemble approach solve tasks which cannot be solved using a single Q-learner. In the experiments explored here we utilize tasks which can be solved by a single Q-learner albeit less reliably or not as well.

We then reach outside of computer science into an interdisciplinary realization connected to our work known as "the wisdom of the crowds" in Section 6.4. We acknowledge that the crowd ensemble approach fits in well with established criteria for a wise crowd and propose experiments which test crowd ensemble performance with respect to the fidelity to these criteria.

In order to reduce the computational complexity added by the crowd ensemble approach on the high-dimension cart-pole task we shared the convoluational layers across ensemble members.

This led to improved performance as it did with the low-dimensional cart-pole task without shared layers. Can we utilize shared layers to improve lower-dimension tasks as well?

We feel that increased reliability and training stability provided by the crowd ensemble approach to Q-function approximation is a big step toward readiness for collaborative control alongside human operators. We discuss a real-world application that we feel is a good fit for this approach.

Next, we discuss some smaller conclusions and questions left unanswered in Sections 6.7, 6.8, and 6.9. Here we ask if there are adjustments we could make to the environments which would increase the importance of actions and come back to unresolved questions raised by our experiments regarding experience replay and a comparison of ADAM and SCG.

We conclude by briefly touching upon questions about the role of actors in recent literature in light of our work. We also question the necessity of training an actor capable of selecting an action globally when the actor-critic approach has already invested in learning a Q-function over the entire state-action space.

6.1 Q-learning undervalues the roles of actions

As a Q-function landscape evolves, those change dwarf the differences between actions. This is especially evident when comparing the relative values of $\max_{a \in \mathscr{A}} Q(s, a)$ for nearby states compared to the difference between actions, $\max_{a \in \mathscr{A}} Q(s, a) - \min_{a \in \mathscr{A}} Q(s, a)$. This causes noise in the action-selection process which results in the agent taking significant steps backward with respect to evaluated reward even when the agent has already learned to solve the task. Once the Q-function landscape with respect to states has been established and is relatively unchanging, there is still churn at the action-selection level. This causes the appearance of catastrophic forgetting because the agent can no longer find its way back to the path leading to goal states.

Experience replay has found success when using Q-learning and neural networks because its sampling scheme allows Q-learning to visit states multiple times which allows the algorithm to squeeze as much benefit from a single training sample as possible. Also, experience replay brings

the Q-learning parameter update algorithms much closer to the assumption of independently-drawn data samples. Experience replay is also beneficial because revisiting old states, presumably, can aid in the prevention of catastrophic forgetting by forcing the agent to use training samples from much earlier in the training process. These samples are valuable because the agent almost certainly visited state space locations far into the past that are different from those visited by the current Q-function. Despite this, catastrophic forgetting still occurs in the presence of experience replay.

Even in a Q-learning approach where the actions are discrete and small in number such as the cart-pole task, the Q-function value differences between actions are too small to hold up under the deformations to the Q-function which occur during training. When coupled with Q-learning with discrete actions, the crowd ensemble reduces decision space volatility leaving the paths the agent takes on its way to the highest Q-value states unmolested. The improved stability allows the agent to build upon what it learns which means it finds better solutions more quickly instead of spending training time re-establishing the paths through the state space which lead to better performance.

Even in the simple pendulum task the Q-function would favor one action or another which would send confusing signals to the policy and lead to sub-optimal solutions where the pole would be in an off-center position by repeatedly applying actions in the same direction instead of using both actions to actually balance the pole requiring fewer actions. When applied to DDPG on the pendulum task the resulting Q-function is more expressive in terms of action preference. On the pendulum task one action is preferred over the other by the critic when using a single Q-function model. This results in the gradient used to update the policy being stronger in the direction of the one action type (i. e., push left) over the other. When the critic is comprised of an ensemble of Q-functions one action is no longer preferred over the other. Instead we observed that the ensemble agents found better solutions because they utilize both action directions equally.

We were able to verify that the instability of the Q-function's preferred action causes problems when training the actor-critic on the bipedal walker task as well. In that instance we see that the policy becomes increasingly unstable: making wild swings in the direction of the preferred action vector that follow wild swings in the preferred action of the Q-function (critic). The crowd ensem-

ble mostly rectifies this issue by providing stability in the Q-function which, in turn, stabilizes the policy.

We conclude that the Q-function emphasizes states over actions in the four tasks examined in this dissertation and that our simple combination scheme for our ensemble Q-functions corrects this issue. When training an agent which uses Q-learning to control a process where minimizing the number of interactions with the environment is a crucial aspect of the task, an ensemble approach to modeling the Q-function appears to be an obvious part of the solution. When the number of interactions with the environment is less of a concern, our results show that crowd ensemble Q-functions are not disadvantageous when compared to parallel Q-learning approaches and is a simple way to get the same benefit.

6.2 Specialization within an ensemble

Future work should begin with an attempt to allow specialization of ensemble members under a mixture of experts framework. We hypothesize that the universal benefits of the crowd ensemble approach will not be fully realized on these tasks when there is an attempt to impose a specialization scheme on the ensemble members. On the other side of the same coin, we would like to see the crowd ensemble approach be applied to tasks where a more hierarchical or mixture approach is obviously beneficial. An example of this could be a cart-pole task where the pole and/or cart mass is variable or where the center of mass of the hull of the bipedal walker is changing. Presumably these changes in mass would result in changes to the best policy and, therefore, would favor a mixture of experts ensemble over a crowd ensemble. Alternatively, the crowd ensemble may be able to absorb this additional complexity. Testing the crowd ensemble's ability to cope with these more modular or hierarchical tasks and comparing it against the more traditional mixture of experts ensemble approach is a high priority.

6.3 Can crowd ensembles solve tasks which are unsolvable by an individual?

A related question to whether allowing ensemble members to specialize in regions of the state space can improve performance, is whether a crowd ensemble can solve tasks which are unsolvable by a non-ensemble Q-function. The tasks solved in our investigation of the crowd ensemble approach were all solvable using a single Q-function. We were able to show that the ensembles are, in some instances, capable of finding better solutions and/or are capable of doing so more reliably and more efficiently. The next frontier is applying the crowd ensemble to tasks that appear to be just out of the reach of a single Q-learner. The challenge here is finding tasks that can be solved using a crowd ensemble but not a non-ensemble learner. This sort of task could be found by increasing the difficulty of the tasks utilized here by adding state factors such as wind or adding obstacles to the bipedal walker task. Another approach to this sort of experiment could involve reducing the capability of the function approximators to the point that they are no longer capable of solving the task then applying the crowd ensemble approach.

6.4 The wisdom of the crowd

In Section 1.3 several examples of crowds performing better than an individual expert were discussed. We've witnessed some compelling works where the most basic ensemble combination mechanisms have shown improved performance over the individual even in the absence of combination mechanisms which intelligently combine ensemble members or even encourage cooperation. Although the experiments reported in this dissertation have somewhat demystified the benefit of simple averaging or voting as a method of combining individual estimates by revealing that the relatively poor differentiation between actions in the state-action value function is the culprit, they have also added another example of the wisdom of the crowd.

Surowiecki [15] gleans four attributes of a wise crowd from his survey of the subject.

- *Diversity of opinion*: each individual has access to a unique viewpoint.

- *Independence*: individual opinions are not influenced by others.
- *Decentralization*: individuals can draw upon local knowledge.
- *Aggregation*: mechanism exists to combine opinions.

The crowd ensemble embodies all of these attributes. The size of the voting majorities is evidence of a diversity of opinion and, as we saw, a shrinking majority had no ill-effects on ensemble performance. The need for diversity of opinion transforms the instability of Q-learning from a liability into a feature. Instability ensures a diversity of opinion as ensemble members shift their decision surfaces as they confront randomly-selected training samples from the experience replay memory. The crowd ensemble members are independent of each other. The only thing holding them together is a shared set of observations which is a result of their consensus actions. The size and directions of the parameter updates of the shared convolutional layers is another indicator of crowd ensemble member independence. Much of the gradient from the fully-connected layers of each ensemble member is canceled out in the summation. The crowd ensemble has a simple yet effective aggregation mechanism. Unlike human ensembles, the bagged Q-functions have objectives which are perfectly aligned which greatly simplifies combination and amplifies the power of simple averaging or majority rule. The decentralization attribute with respect to each individual having an independent knowledge may appear less clear with crowd ensembles. Each individual has its own neural network (or at least its own layers if some layers are shared) which represents a local knowledge base upon which their estimates are based. Also each network has its own set of parameters. Forgetting is a somewhat convenient issue with the crowd ensemble because it contributes to the locality of the knowledge. Considering decentralization as a laissez-faire approach to knowledge distribution, the crowd ensemble definitely fits the description.

Small tweaks could be made to the crowd ensemble to allow further embodiment the attributes of a wise crowd. The individuals could be further decentralized by allowing them to generate their own training data or allowing ensemble members to have private data while keeping their aggregation strategy the same. The aggregation strategy could be tweaked to allow the agents

to convey additional information about their choices when voting by allowing them to include a measure of confidence in the superiority of an action. Or we could go the other way and take steps toward the more traditional methods (e. g., mixture of experts) of combining estimations. Or we could add diversity to the ensemble by varying meta-parameters such as γ, learning rates, or network layer sizes. Another avenue for future work is to go to the other way and show how gradual eroding of the crowd ensemble's adherence to these four principles affects performance. We expect that greater adherence to these attributes will lead to better performance. RL agents provide a compelling experimental platform from which to investigate these questions which are of interest to many fields.

Recently Navajas et. al. [58] suggests a way to take the crowd ensemble approach a step further by naively combining consensus decisions from multiple mixture of experts ensembles. Social dynamics are widely thought to inhibit the beneficial power of crowd decision making. However they find a surprising result when combining results from an ensemble of ensembles where the group dynamics within the sub-ensembles are encouraged. This notion could revive the mixture of experts style concept by allowing for an ensemble of mixtures of experts where the ensemble members are allowed to influence one another. This approach could allow the ensemble to solve tasks that are more difficult than could be solved, even once, by a single Q-learner.

6.5 Shared layers

The results of Chapter 5.4 show us another way to take advantage of the instability of Q-learning to average gradients from all ensemble members which will reduce the zig-zagging action which naturally occurs during RL. Zig-zagging of parameter updates occurs whether training samples are drawn using experience replay or we allow the agent to learn online by giving the agent only the latest set of training samples. As discussed, learning a representation conducive to Q-learning in the lower layers of the ANN is vital. This approach could also be applied to the smaller ANNs in the low-dimension cart-pole task or the bipedal walker task. Would this lead to faster, more direct updates of the lower layers on these ANNs as well? We showed that the instability

of Q-learning proved beneficial in this context because the ensemble members passed gradient information which combined to take more direct steps toward the set of features associated with good performance instead of taking a zig zagging path through parameter space. Sharing layers with multiple outputs representing the ensemble could be directly compared to the asynchronous Q-learning [5] approach.

With respect to the actor-critic approach used for the bipedal walker task, further weight sharing could be attempted using something like the architecture shown in Figure 6.1. The depth at which the layer sharing ends would be the object of testing.

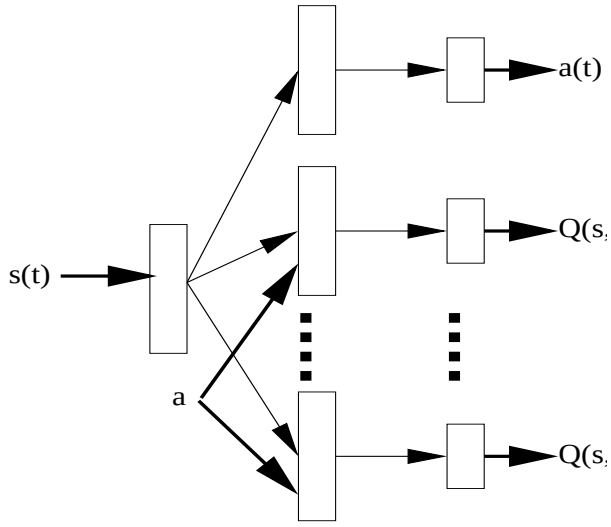


Figure 6.1: Actor-critic architecture with a critic comprised of an ensemble of Q-functions and a single actor network. All networks share the input layer. The state would input into the shared layer and the Q-functions would augment the output of the shared layer with the action information to compute $Q(s, a)$.

6.6 Real world challenge: seed corn dryer balancing

Agriculture has undergone a slow (but accelerating) technical revolution since the decline of serfdom. Today tedious, labor-intensive agriculture tasks can be automated using reinforcement learning approaches but the agent performance must be data efficient and stable.

Data efficiency in RL has been improved using techniques such as pre-training [7, 8] and transfer learning [59]. These techniques are compatible with crowd ensembles. Other work on improv-

ing RL stability has shown that providing truly stable RL learning is not feasible because of the infinitesimal leaning rate necessary to guarantee stability [60]. When designing a RL agent to work along side a human operator, sudden collapses in performance like those seen with catastrophic forgetting will destroy the operators' faith in the agent while small fluctuations in performance during training can be forgiven. The crowd ensemble is a significant step toward an agent that could serve as this sort of reliable tool during training.

In agriculture and manufacturing there remain many pieces of low-hanging fruit ripe for, at a minimum, partial automation allowing for performance unattainable by a human operator alone. One such example is the seed corn dryer [61, 62]. Seed corn must be dried prior to processing and improper conditioning will reduce the germination rate. The corn is dried while still on the ear and the bins are filled and emptied several times during a single harvest making them very hostile places for sensors. The dryer has strict constraints limiting the temperature of the air pushed through the bins as well as a goal of even air flow through each bin. Figure 6.2 shows an example dryer with 28 bins and four sets of burners and fans. Air enters the dryer via the upper-tunnel and passes through roughly half of the bins into the lower-tunnel (called down-air bins). The air exits the dryer through the opened external doors of the other half of the bins (referred to as up-air). The example dryer also has 14 bypass doors allowing air to pass from the upper-tunnel to the lower-tunnel without first passing through a bin.

The moisture content of the ear corn is different in each bin. The desired inlet temperature of each bin is determined by its current moisture content, especially for the up-air bins when the seed corn is highly sensitive to excessive heat. The air is cooled when passing through bins by way of evaporative cooling. Each dryer is different in terms of the size, number, and configuration of the bins and fans.

We plan to apply these state-of-the-art RL data efficiency methods along with the crowd ensemble to the problem of controlling the seed corn dryer fan and burner settings to match the target temperature and air flow values while the reacting to the human operators' dryer configuration decisions and changes to ambient conditions. If successful this would simultaneously reduce the

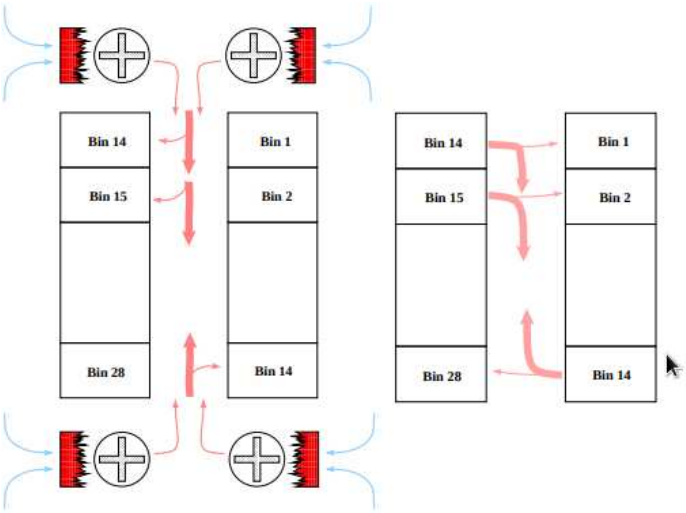


Figure 6.2: Seed corn dryer with 28 bins and four sets of fans and burners. Fans force air into upper tunnel (left). Air is pushed through roughly half of the bins into the lower tunnel and then out of the building through the remaining bins.

amount of work necessary to run the dryer while also keeping the dryer running at peak efficiency thus increasing dryer capacity, reducing total harvest duration, protecting germination, and aiding the environment.

6.7 Are there other solutions to decision surface instability?

The work in this dissertation arose first from a hypothesis that an ensemble would mitigate the catastrophic forgetting observed when training Q-learning agents. Beginning with continuous state, discrete action tasks, we observed that it did improve performance and reduce forgetting. The instability of the decision space of individual Q-functions leads to a breakdown in the solution: the agent has to pass through states with frequently changing decision spaces in order to reach the relatively stable states in and around the goal region. Further investigation revealed that this instability was caused by the relatively small Q-value differences between actions relative to the Q-value differences between nearby state locations. Work on the continuous action tasks revealed that the Q-function representation for actions was still inadequate but, this time, the Q-function was simply biased toward one action or another. We found that the crowd ensemble goes a long way toward rectifying this situation.

The experiments presented here began by finding meta-parameters which were capable of performing well for each task and then asking if and how ensembles were able to improve upon that performance. There may be other parameters which could be modified to make the actions a more important part of the Q-function. One such example is the RL parameter γ which determines how large of a role future rewards play in the current state-action value function. This was not pursued because the two sets of tasks, the two cart-pole variants and the continuous action tasks, used γ values which were at the two ends of the spectrum of commonly used γ values. In the cart-pole task we used $\gamma = 0.9$ while the continuous actions tasks used $\gamma = 0.99$.

Another option could be reducing the frequency of the action selection. We considered the frequency of action selection to be an attribute of the simulations and, therefore, not tuneable. Initially we experimented with two different action frequencies in the cart-pole task and found little difference in the results. The results we share here are for the longer of the two action frequencies which was twice as long as the other. We did find that the ability of the agent to solve the task was affected by the frequency of the actions it could apply. The ability to change actions more frequently allows the agent to use a smaller range of actions and allows for greater fine-tuning of performance. Other tasks may have more slowly changing dynamics which would allow for (or even require) longer intervals between actions. However, more slowly changing dynamics and accompanying slower actions would presumably result in similar issues where actions are not recognized as having a large effect on performance because each individual action will still only change the state a relatively small amount. For this reason this was not pursued. We don't anticipate that this will lead to a fruitful area of research but the possibility should be acknowledged.

6.8 Experience replay questions

As it stands the most commonly utilized form of experience replay is a haphazard approach to collecting training samples which results in a very large database of training samples Whether the replay memory is of limited size and training samples are stored as a FIFO queue or the memory

stores all experiences, the memory will be dominated by the most frequently visited states of the most common way or ways to solve the task.

The analysis here showing that volatility of the decision-space is not spread equally throughout the state space indicates that, perhaps, some regions could benefit from additional training more than others. A future study highlighting a relationship between experience replay memory constituency and volatility could bolster this case. This could lead to approaches which probabilistically retain experiences in the replay memory based upon the decision volatility of that region. This could allow for more efficient data collection and a more balanced memory which emphasizes regions which are in the greatest need for additional training.

RL is a bootstrapped form of learning. Some work in RL has emphasized learning by demonstration or learning by observing. This concept is compatible with the bootstrapped view of RL and the Q-learning notion of backing reward information up from more terminal, less uncertain estimates of state/action value. On the other hand RL is also dependent upon the policy used to obtain the training samples. Experience replay is an indirect challenge to the latter notion. As the Q-function changes, the training samples in the experience replay are, arguably, from a different policy. The further back in the replay memory the sampled experiences reach, the more divergent the policy from which the data was generated. If experience replay works as described, does this imply there is nothing sacrosanct about training from data generated from the same agent? Doesn't the agent benefit from training samples which address the needs of the agent by correcting its undesirable behaviors and reinforcing the desirable ones?

In Section 3.6 the concept of indirect experience replay was presented and contradictory results were presented. The low-dimension cart-pole task was solved using an experience replay memory from another agent and the results were competitive with the state-of-the-art approach. The bipedal walker task, on the other hand, was not competitive when training from an experience replay memory form another agent. Why the discrepancy? As shown in that same section, clearly DDPG benefits tremendously from experience replay. From the results of Section 3.6, we hypothesize that experience replay's primary benefit lies mostly in allowing for additional parameter updates.

This would explain its inexpendable position in the deep RL literature where the approaches used, in the case of DDPG, or the high-dimension of the function approximators requires a large number of parameter updates.

6.9 SCG versus ADAM

Can SCG work on bipedal walker or cart-pole task from pixels? To our knowledge SCG is not used in the RL literature outside of the cart-pole experiments presented here and some of our previous work [7]. The ADAM and SGD algorithms are the current ANN update methods of choice. We were unsuccessful in finding parameter settings which allowed us to use SCG on the high dimension cart-pole task. Was this a result of insufficient parameter exploration or is SCG not an appropriate parameter update method for this task? If SCG is unfit, why?

In head-to-head comparison ADAM was found to be inferior to SCG on the low-dimension cart-pole task – even when allowing ADAM to perform many more parameter updates than SCG. Would SCG also be better suited for the bipedal walker task? SCG has not been explored by the RL community but out experience with it indicates it may deserve more attention.

6.10 Global actors *and* global critics?

In the crowd ensemble DDPG approach an ensemble of Q-functions is used to compute the critic outputs. There is no attempt to create an ensemble of actors because their combination would result in actions which do not reside in the regions of the space that any of the ensemble members would select and voting is not an option in a continuous action space. Combining actor outputs in this way also violates the theoretical meaning of the actor outputs as centers of highest probability from which a stochastic policy could sample.

The critic outputs the action-state value function for every state action pair. It, alone, represents the agent's knowledge of the task. Why then should the actor be trained to output the desired action for the entire state space if this information is already represented by the critic?

This represents an area of increasing interest for us. The additional representational power and improved stability of the critic's ensemble of Q-functions could be used to train regional actors. Not only would the actors be able to be represented using much simpler function approximation schemes, but these actors may be able to better tease apart and magnify the action preferences of the critic in these smaller regions. Regional actors could be less affected by disruptions to the global Q-function that cause erroneous changes in the the preferred action.

We hypothesize that the state-value function, when computed from the Q-function using $\max_{a \in \mathscr{A}} Q(s, a)$ is relatively stable compared to decision space and our graphs serve as anecdotal evidence. We further hypothesize that a model of system's dynamics would also be more stable than the decision space since it is trained using supervised learning. This could be taken further with a single actor modeled using a very simple function approximator that is updated with Q-function gradient information using samples from the replay memory as it travels through state space alleviating the need for training an actor. This would convert the role of the policy from learning how to maximize the action-value function estimates to learning how to greedily climb the action-value function. This could utilize a planning-style approach such as Dyna [63].

Bibliography

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[2] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. Master's thesis, Imperial College London, 2015.

[3] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[4] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[6] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

[7] Charles W Anderson, Minwoo Lee, and Daniel L Elliott. Faster reinforcement learning after pretraining deep networks to predict state dynamics. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[8] D. L. Elliott and C. Anderson. Using supervised training signals of observable state dynamics to speed-up and improve reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–8, Dec 2014.

[9] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, David Silver, and Hado P van Hasselt. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4058–4068, 2017.

[10] Gabriel Victor de la Cruz, Yunshu Du, and Matthew E. Taylor. Pre-training neural networks with human demonstrations for deep reinforcement learning. *CoRR*, abs/1709.04083, 2017.

[11] Francis Galton. Vox populi. *Nature*, 75:450–451, March 1907.

[12] Jack L Treynor. Market efficiency and the bean jar experiment. *Financial Analysts Journal*, 43(3):50–53, 1987.

[13] Richard P. Larrick and Jack B. Soll. Intuitions about combining opinions: Misappreciation of the averaging principle. *Management Science*, 52(1):111–127, 2006.

[14] Reid Hastie and Tatsuya Kameda. The robust beauty of majority rules in group decisions. *Psychological review*, 112(2):494, 2005.

[15] David Surowiecki. *The Wisdom of Crowds*. Anchor Books, 2005.

[16] Michael T Maloney and J Harold Mulherin. The complexity of price discovery in an efficient market: the stock market reaction to the challenger crash. *Journal of Corporate Finance*, 9(4):453–479, 2003.

[17] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[18] Martin Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525 – 533, 1993.

[19] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Pittsburgh, PA, USA, 1994.

[20] Ian T. Nabney. *Netlab: Algorithms for Pattern Recognition*. Springer, 2002.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[22] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[23] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[24] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[25] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[26] Norman L Johnson and Kerstin Dautenhahn. Collective problem solving: Functionality beyond the individual. *Los Alamos National Laboratory technical report: LA-UR-98-2227*, 1998.

[27] Norman L. Johnson. Diversity in decentralized systems: Enabling self-organizing solutions. In *Decentralization II Conference*, pages 123–140. UCLA, 1999.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[29] Charles W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems*. PhD thesis, University of Massachusetts, 1986.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[31] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, (8):293–321, 1992.

[34] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

[35] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2015.

[36] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058, 2017.

[37] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[38] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

[39] A. Agostini and E. Celaya. A competitive strategy for function approximation in q-learning. In *2011 International Joint Conference on Artificial Intelligence*, pages 1146–1151, 2011.

[40] A. Hans and S. Udluft. Ensembles of neural networks for robust reinforcement learning. In *Ninth International Conference on Machine Learning and Applications (ICMLA)*, pages 401–406, Dec 2010.

[41] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[42] Charles W. Anderson and Zhaohui Hong. Reinforcement learning with modular neural networks for control. In *IEEE International Workshop on Neural Networks Application to Control and Image Processing*, pages 90–93, 1994.

[43] Kenji Doya and Kazuyuki Samejima. Multiple model-based reinforcement learning. *Neural Computation*, 14:1347–1369, 2002.

[44] Stefan Faußer and Friedhelm Schwenker. Neural network ensembles in reinforcement learning. *Neural Processing Letters*, 41(1):55–69, 2015.

[45] Stefan Faußer and Friedhelm Schwenker. Selective neural network ensembles in reinforcement learning: Taking the advantage of many agents. *Neurocomputing*, 169:350–357, 2015.

[46] Zhewei Huang, Shuchang Zhou, BoEr Zhuang, and Xinyu Zhou. Learning to run with actor-critic ensemble. *arXiv preprint arXiv:1712.08987*, 2017.

[47] Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

[48] Marco A Wiering. Qv (lambda)-learning: A new on-policy reinforcement learning algrithm. In *Proceedings of the 7th European Workshop on Reinforcement Learning*, pages 17–18, 2005.

[49] Marco A Wiering and Hado Van Hasselt. Two novel on-policy reinforcement learning algorithms based on td (λ)-methods. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 280–287. IEEE, 2007.

[50] Srinivasan Praveen Blackwell Sam Alcicek Cagdas Fearon Rory Maria Alessandro De Panneershelvam Vedavyas Suleyman Mustafa Beattie Charles Petersen Stig Legg Shane Mnih Volodymyr Kavukcuoglu Koray Nair, Arun and David Silver. Massively parallel methods for deep reinforcement learning. In *International Conference on Machine Learning Deep Learning Workshop*, 2015.

[51] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, June 2013.

[52] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[53] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[54] Ganger M. Hu W. Duryea, E. Exploring deep reinforcement learning with multi q-learning. *Intelligent Control and Automation*, (7):129–144, 2016.

[55] Open AI team. Openai gym. `http://gym.openai.com/`, 2016.

[56] Erin Catto. Box2d v2.3.0 user manual. Available at box2d.org, 2013.

[57] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical Review*, 36:823–841, Sep 1930.

[58] Joaquin Navajas, Tamara Niella, Gerry Garbulsky, Bahador Bahrami, and Mariano Sigman. Aggregated knowledge from a small number of debates outperforms the wisdom of large crowds. *Nature Human Behaviour 2*, pages 126–132, 2018.

[59] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10.

[60] James Nate Knight and Charles Anderson. Stable reinforcement learning with recurrent neural networks. *Journal of Control Theory and Applications*, 9(3):410–420, 2011.

[61] Daniel L Elliott and Russell E Valentine. Recurrent neural networks for moisture content prediction in seed corn dryer buildings. In *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 934–935. IEEE, 2011.

[62] MT Islam, BP Marks, and FW Bakker-Arkema. Optimization of commercial ear-corn dryers. *Agricultural Engineering International: CIGR Journal*, 2004.

[63] Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael H. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In David A. McAllester and Petri Myllymäki, editors, *UAI 2008. Proceedings of the 24th conference in uncertainty in artificial intelligence.*, pages 528–536, Helsinki, 2008. AUAI Proceedings.