0 Contents

1 Results and Discussion

1.1 Training

In this section, the training of the player and adversary will be investigated. The first sections will discuss the results of a baseline neural network, $f_\theta(\boldsymbol{x})$, which takes the true state as input. This is followed by a comparison with those of the network that took the 2-dimensional state as input, $f_\theta(\boldsymbol{x}^{(2D)})$. The results are interpreted with a discussion of the merits and shortcomings of the techniques. For the training runs shown, the parameters used are given in table 1.1 and **??**.

Parameter	Value
Unopposed Episodes for initial iteration	100
Opposed Episodes per iteration	40
Number of MCTS Simulations per step	20
$F_2^{(max)}$	0.05
$\boldsymbol{x}^{(2D)}$ bins	40
$\boldsymbol{x}^{(2D)}$ discount	0.5

Table 1.1: Training parameters used

Iterations start at zero and are defined by two main steps: (1) A batch of training examples are generated using the current neural networks. (2) The neural networks are retrained using these examples. The full algorithm can be found in **??**.

1.1.1 MCTS

Figure 1.1 shows the structure of a typical tree search without neural network training. This is significantly better than with no tree search, which typically achieves an episode length of 14.

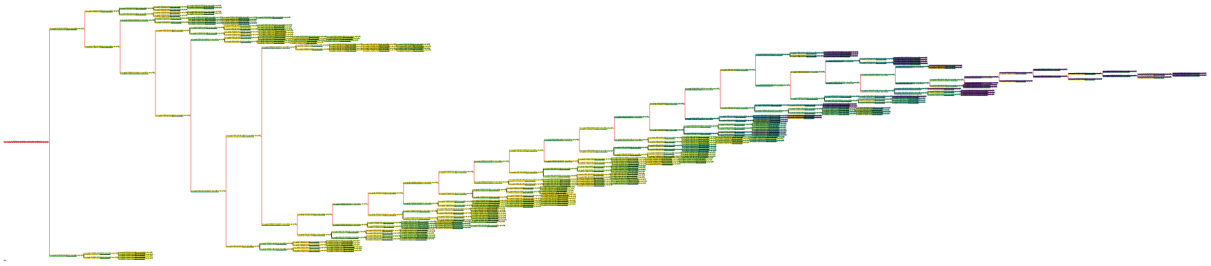


Figure 1.1: An example MCTS tree structure lasting 33 steps with 15 MCTS simulations per step and both agents are untrained. The adversary has and average power of 5% of the player ($F_2 = 0.05F_1$). Lighter (yellow) colours represent "better" states.

Figure 1.2: A zoomed in view of fig. 1.1. The three colours of each node are based on the value of $U = Q(\boldsymbol{x}, u) + ucb$ as in **??**, the number of state-action visits, $N(\boldsymbol{x}, u)$, and the state-cost, $c(\boldsymbol{x}_t)$. The red nodes follow the true trajectory. Ns ($N(\boldsymbol{x})$) is the total number of state visits, this does not equal the action-state visit count, Nsa ($N(\boldsymbol{x}, u)$), these differ along the true trajectory due to the re-running of the MCTS simulation.

Figure 1.2 shows the sequential maximisation that occurs during the tree search. Note that the action-value, $Q(\boldsymbol{x}_t, u_t)$, for the adversary is negated and Q is between -1 and 0 for the player. $N(\boldsymbol{x}, u)$ is correlated with the predicted value, V_{pred}, and U, which suggests that the neural network and tree search are working as expected. At the first branching in fig. 1.2 the ucb is 17% of U. Intuitively, this is within the correct range as a larger exploration term would cause the MCTS to branch more, diminishing the computational benefit of the structured search, whereas a smaller exploration term inhibits learning. To achieve this ratio, a value of 1 was chosen for $c_{explore}$. The action-state visit counts are 22 and 17, which are markedly higher than the number of simulations per step (10 in this example), suggesting the MCTS is performing a highly structured search. Furthermore,

the similar predicted state values, $v_\theta(\boldsymbol{x})$, reflect similar visit counts of each branch.

1.1.2 State and Action Predictions

In this subsection, the output of the baseline neural network for a specific episode (figs. 1.3 to 1.5) is investigated. From this, the suitability of using a neural network for both the adversary and the player, what the neural networks are modelling and limitations of the model are discussed.

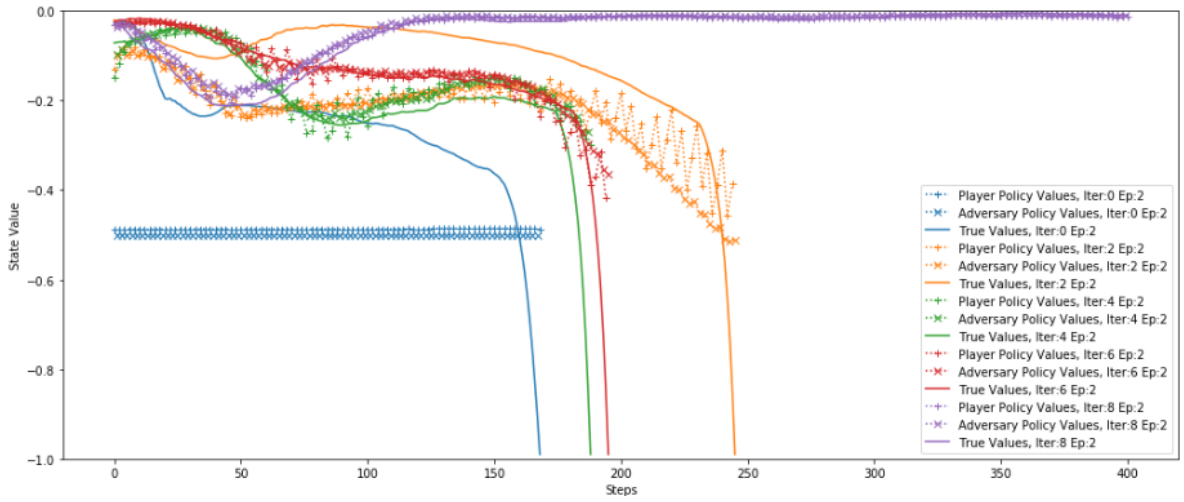


Figure 1.3: The predicted state-values (from both the adversarial and player networks) plotted against the true state-value for a number of iterations.

Figure 1.3 shows that by iteration 4 the player and adversary networks both follow the true value very closely, and by the 10th iteration the mean squared error (MSE) is almost zero (fig. A.4). This is likely due to the adversary pushing only in one direction, therefore making the value very predictable. In iteration 2, the adversary values are more "jagged", alternating between a positive and negative evaluation depending on whether the player (which roughly alternates between pushing left and right when near the equilibrium) has pushed the pendulum further over or not. I.e. by iteration 10, the adversary has learnt to predict the player's actions, which suggests that the prediction of the state-value is not impeded by the use of two neural networks.

Figures 1.4 and 1.5 show the change in policy action predictions between iteration 2 and iteration 4. By iteration 2 the adversary has already decided on the direction that it will push, however the player is still tuning itself. At iteration 4, the player 4 switches between pushing left and right at a lower frequency.

The output of the predicted values for a small slice of $\dot{x}$ is shown in fig. 1.6. Along all pairs of axes, the data is not linearly separable, furthermore principal component analysis (PCA) shows that the predicted values are not linearly separable along the principal axes either (fig. 1.7).

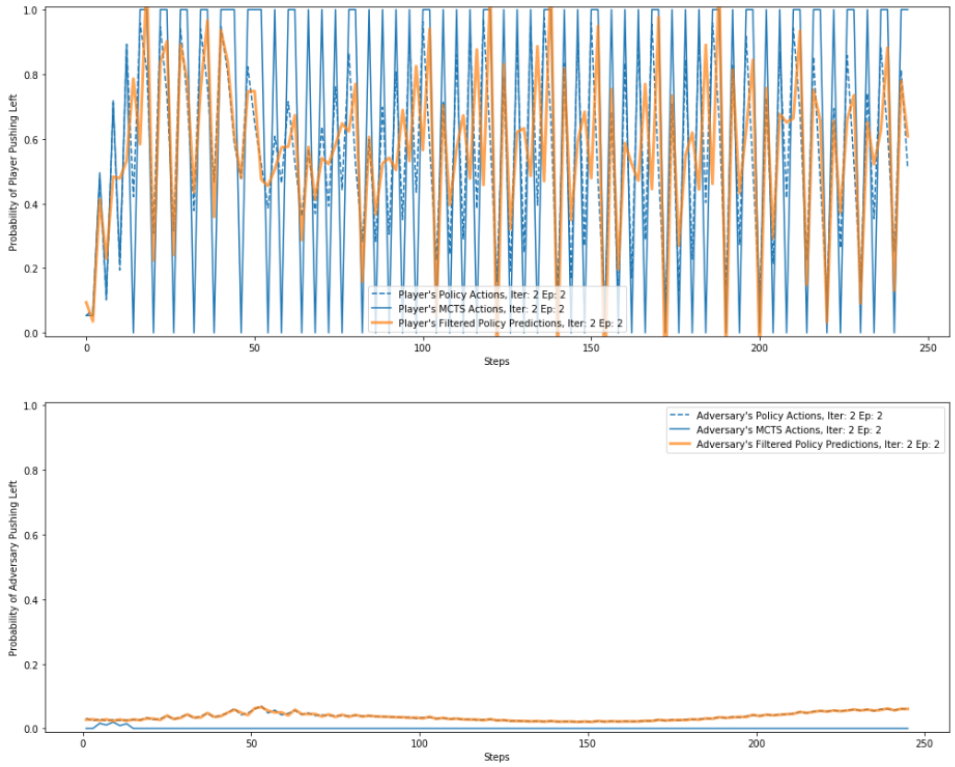


Figure 1.4: MCTS and policy predicted actions vs step for an episode in iteration 2.

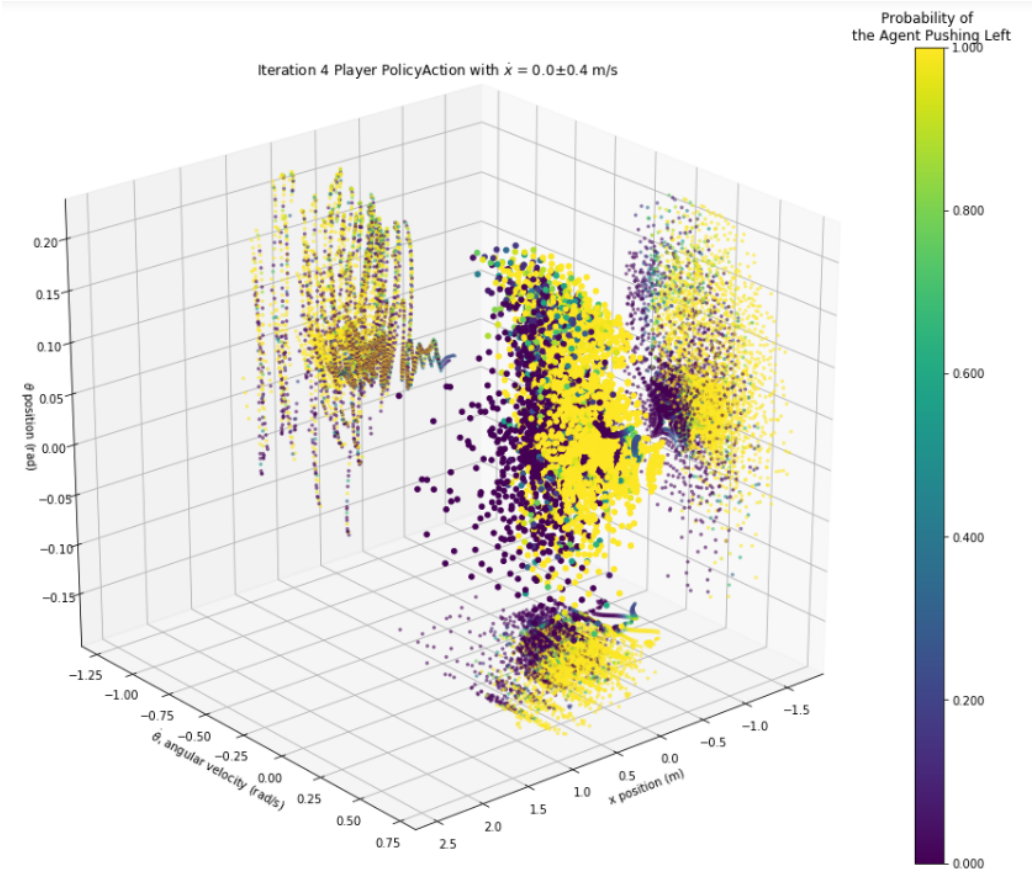


Figure 1.6: A 3D plot of policy-predicted actions during training. The central "blob" is projected onto the $(x, \dot{\theta})$, (x, θ) and $(\theta, \dot{\theta})$ axes.

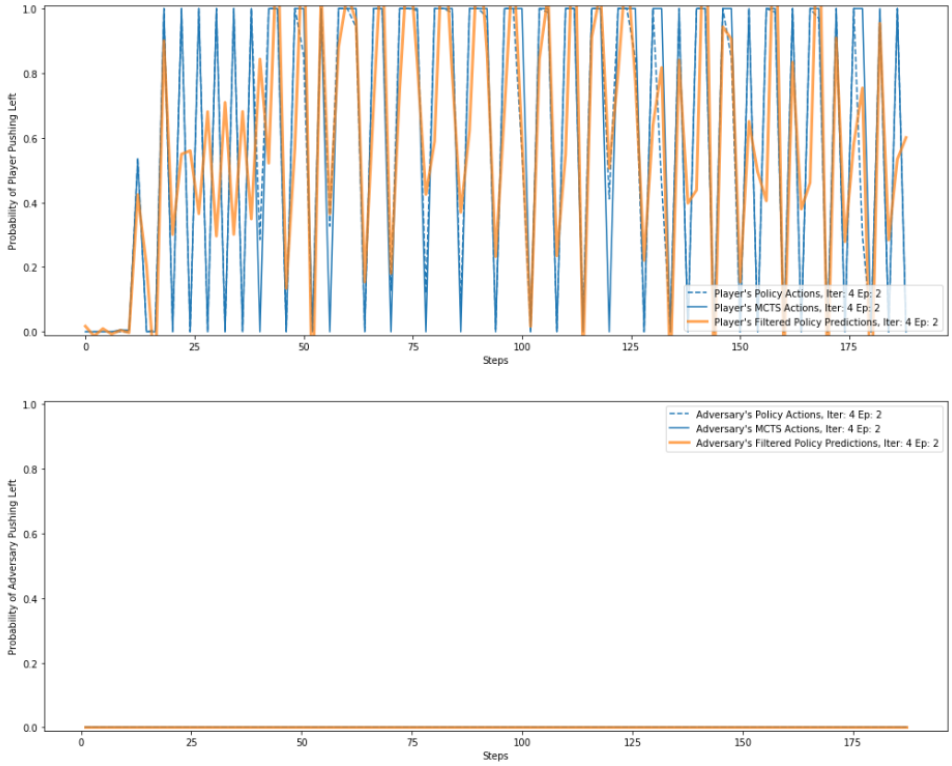


Figure 1.5: MCTS and policy predicted actions vs step for an episode in iteration 4.

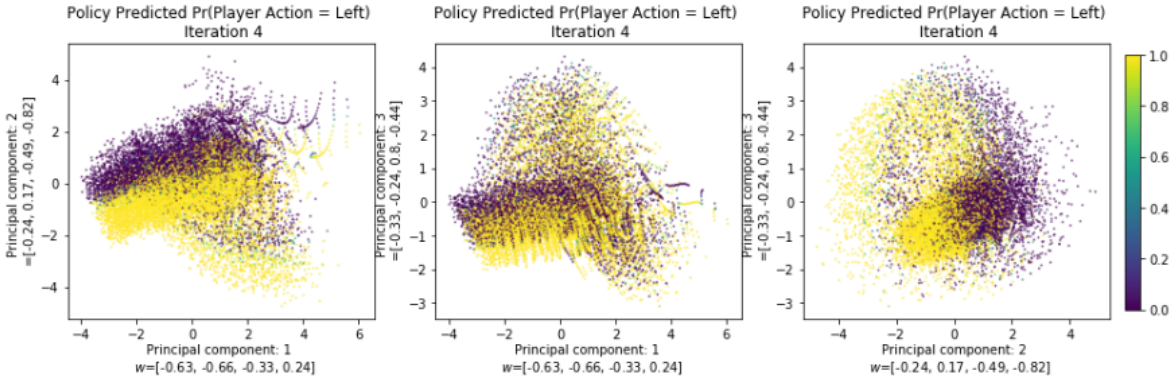


Figure 1.7: Principal Component Analysis of the data in fig. 1.6.

In the optimal control solution, a real valued force would be used to get the pendulum to the equilibrium. Due to the constraint over the actions, the policy must approximate that by repeatedly pushing left/right in order to get an average force that is equivalent to a continuous one. By iteration 4 (fig. 1.5), the neural network matches the MCTS prediction at almost every step and switches quickly between pushing left and right. This implies that a trajectory through the (x, θ) plane would look like a "wave", where the power cycle of the wave in the vicinity of the point is proportional to the instantaneous average power predicted at that point. Additionally, a higher frequency of the wave is related to a faster response time of the system. Higher frequencies requiring deeper neural

networks. Portions of this wave are shown in the (x, θ) projection of fig. 1.6.

The instantaneous predicted average power of the policy can be estimated by applying a low-pass filter to the predicted action probabilities (orange line in figs. 1.4 and 1.5). The low-pass filter has a normalised cut-off frequency of $\frac{1/\tau}{2/\delta t}$, where τ is characteristic time of the system and $\frac{2}{\delta t}$ is the Nyquist frequency. For this system, however, this only provides marginal insight as, for this implementation, the time step is almost the same as the characteristic time. Decreasing the time step requires the episode length to be increase such that the real-world episode time is constant at $dt \times S = 0.01s \times 400steps = 4s$. Therefore, decreasing dt causes a significant computational time.

Figures 1.4 and 1.5 show that the frequency decreases between iteration 2 and iteration 4, therefore, the neural network is not making use of the highest frequency that it has the capacity to use as in iteration 2. This suggests that against this adversary, a frequency higher than the natural frequency is not necessary for good control. This may be because a higher frequency is needed for more uncertain situations and, since the adversary is only pushing in one direction, does not need to oscillate as quickly.

1.1.3 Power Matching and the Policy

In this subsection, the problem of power matching the player and adversary and the efficacy of pre-training the player as a solution to this are analysed. This is followed by a discussion on why the player and adversary act as they do, with an interpretation of the policies they have learnt.

It was found that training the player unopposed indefinitely will cause the player to balance the pendulum within a few time steps from a wide range starting positions, however introducing an adversary at or close to "full power" (i.e. $F_2 \gtrapprox 0.0476F_1$, **??**) will cause the inverted pendulum to fall immediately. Therefore, an unopposed training episode and an incremental increase in adversary power were introduced. After implementing this, typically, either the adversary is too strong and the episode is over within 20 time steps, or the adversary learns to simply push in one direction only. Figure 1.8 shows this transition occuring over just one episode. Since the adversary was not trained on iteration 0, iteration 1 has MCTS predictions to push both right and left. However, during the training in iteration 2, even with approximately 10% of training examples pushing left, the network trains to push right 100% of the time, and as a result the MCTS also predicts that pushing right is always the best action.

The player does not suffer from this bias and has a far more uniform spread of policy predictions in iteration 2. In iteration 3, the policy predicts either left or right with 100% conviction, however, these are distributed in a roughly 50:50 split, with the proportion of $p(Player\ Action = left) > 0.5$ being slightly higher, to counter the adversary always pushing right. As a result of the adversary being trained to push solely in one direction,

the proportion states that are visited are biased in one direction. Visually, this is as if the system is performing the set-point tracking of a reference angle slightly off centre, and therefore slowly moves in one direction until reaching x_{max}.

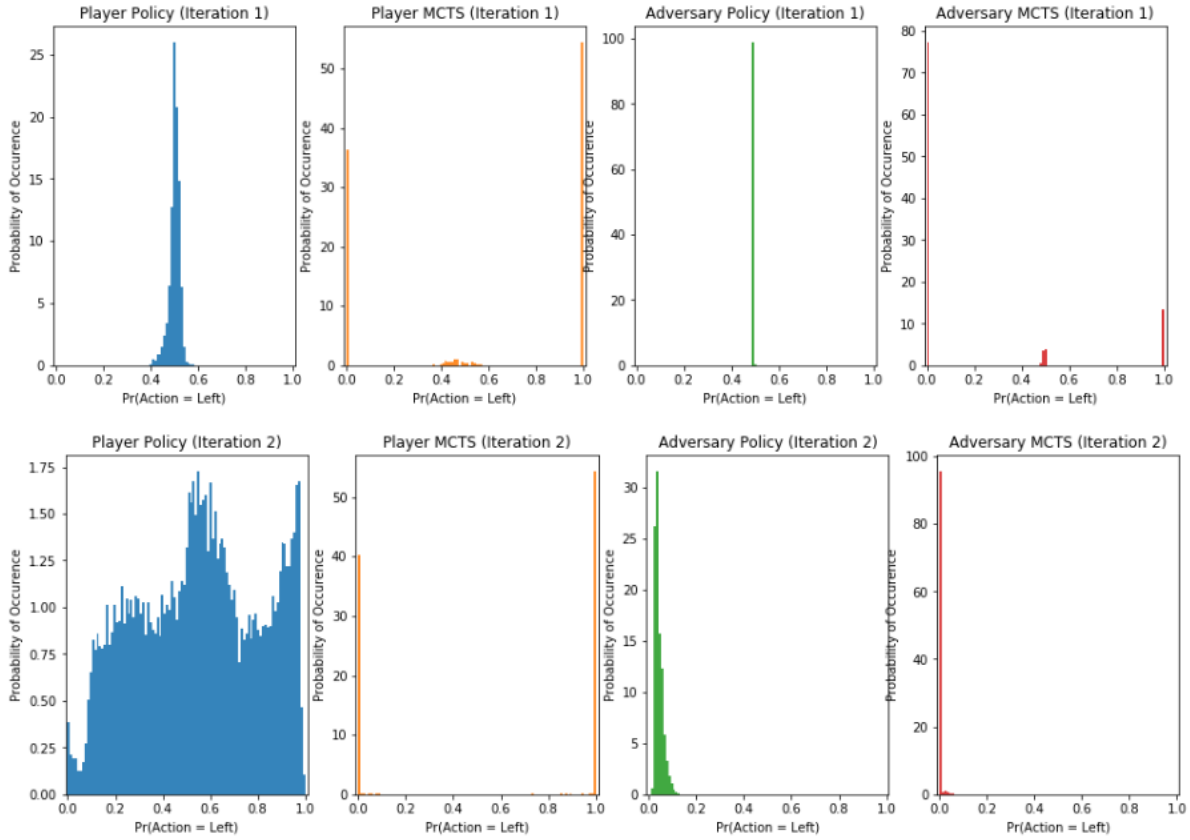


Figure 1.8: Histograms showing the distribution of predicted actions by the MCTS and pure policy for both the player and adversary.

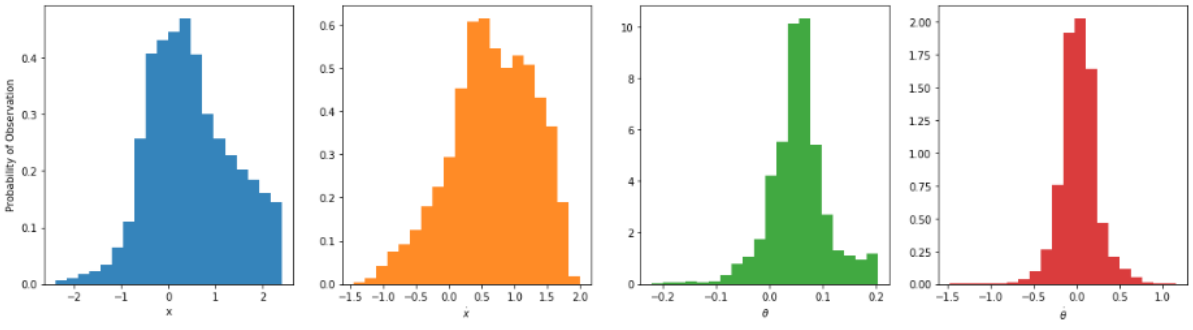


Figure 1.9: Histograms showing the proportion of time that the system was in each state on the second iteration.

The most likely cause for this bias is the differing abilities between agents due to pre-training the player too much or too little, and the power curve of the adversary. How much to train the player before introducing an adversary and how strong to make

the adversary is difficult to balance. The player can be better-trained by increasing the number of training examples it has from unopposed episodes and by increasing the number of unopposed policy iterations. Additionally, the adversary's power, and power curve, can be adjusted by varying γ and F_2 in the adversary handicap formula, $F_2(1-\gamma^{i-N+1})$. However by training the player more and under-powering the adversary, the adversary learns to push solely in one direction. Pre-training the player too little typically leads to the overtraining of the neural network on a small number of examples, which is difficult to re-train against. Alternatively, under-powering the adversary, or making it too weak for too long seems to cause the adversary to learn that slowly pushing the inverted pendulum to x_{max} is the best option, rather than attempting to push it over quickly.

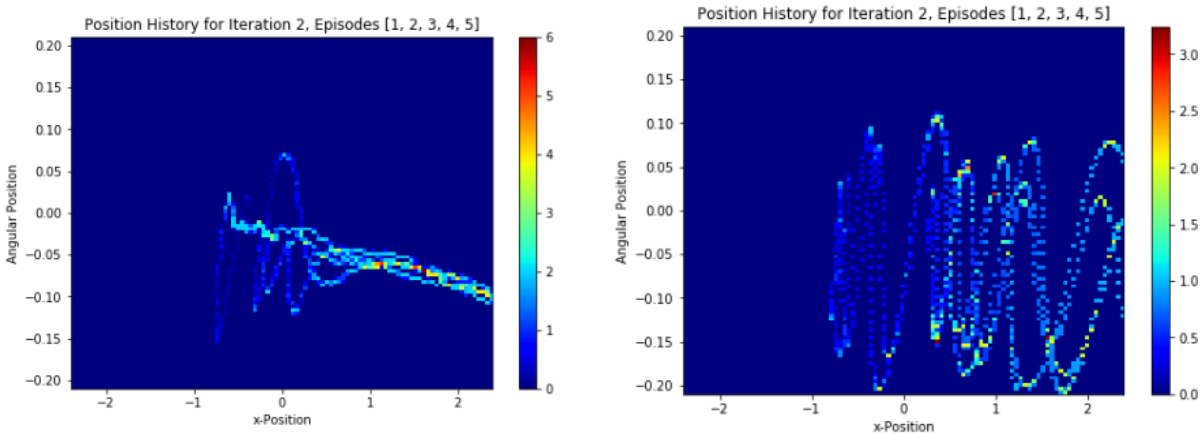
The cost function being weighted as $\boldsymbol{w}^T = [0.4, 0.1, 0.7, 1]$ may have exacerbated this behaviour. A low weight the on x-position compared to the angular position may mean that the player is not prioritising stopping the slow drift to $\boldsymbol{x}_{max}$. Yet from the adversary's perspective, since it is initially learning whilst underpowered, this is the best way to reduce the state value. A possible way in which this could have been prevented would have been to set the x-position cost to zero. In this case, if the adversary solely pushes in one direction, either the player would be able to find an equilibrium by leaning into the adversary's force - minimising the $\dot{x}$ cost, but outputting a constant θ cost; or the player would attempt to minimise the θ cost and as a result move in one direction continuously - outputting a constant $\dot{x}$ cost. Both of these scenarios do not consistently improve the adversary's state value, therefore this could force the adversary to attempt to knock the pendulum over, rather than slowly improving the state-value.

1.1.4 Comparison with $f_\theta(\boldsymbol{x}^{(2D)})$

The training of the convolutional neural network (CNN) results in training episodes with very oscillatory behaviour compared to the the baseline network (figs. 1.10a and 1.10b).

The oscillatory nature of these episodes are due to the player switching between pushing left and right much slower than the characteristic time of the system (fig. 1.11).

This is likely due to the a relatively shallow neural network being used (2 convolutional layers followed by 1 linear layer, see **??**). A deeper neural network was found to exceed the memory allocation limit of the GPU used. It is possible that this caused a more simple model of the controller to be trained, such that the data became easily linearly separable, as shown in fig. 1.12 where the (x, θ) and $(\theta, \dot{\theta})$ axes have easily separable predictions.



(a) True-State Input

(b) 2-Dimensional State Input

Figure 1.10: Motion history images showing the trajectories of 5 episodes for both neural networks. Higher valued colours represent more recent positions (or positions with multiple visits as the images are computed as a sum of discounted states).

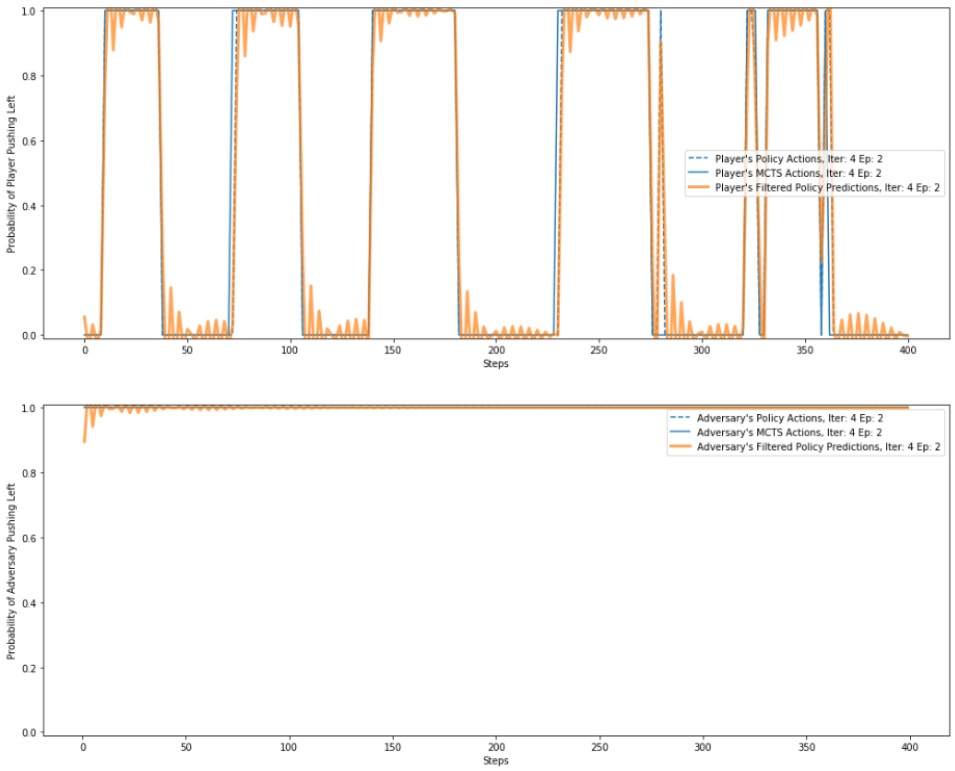


Figure 1.11: MCTS and Policy Predicted Actions vs Step for An Episode in Iteration 4.

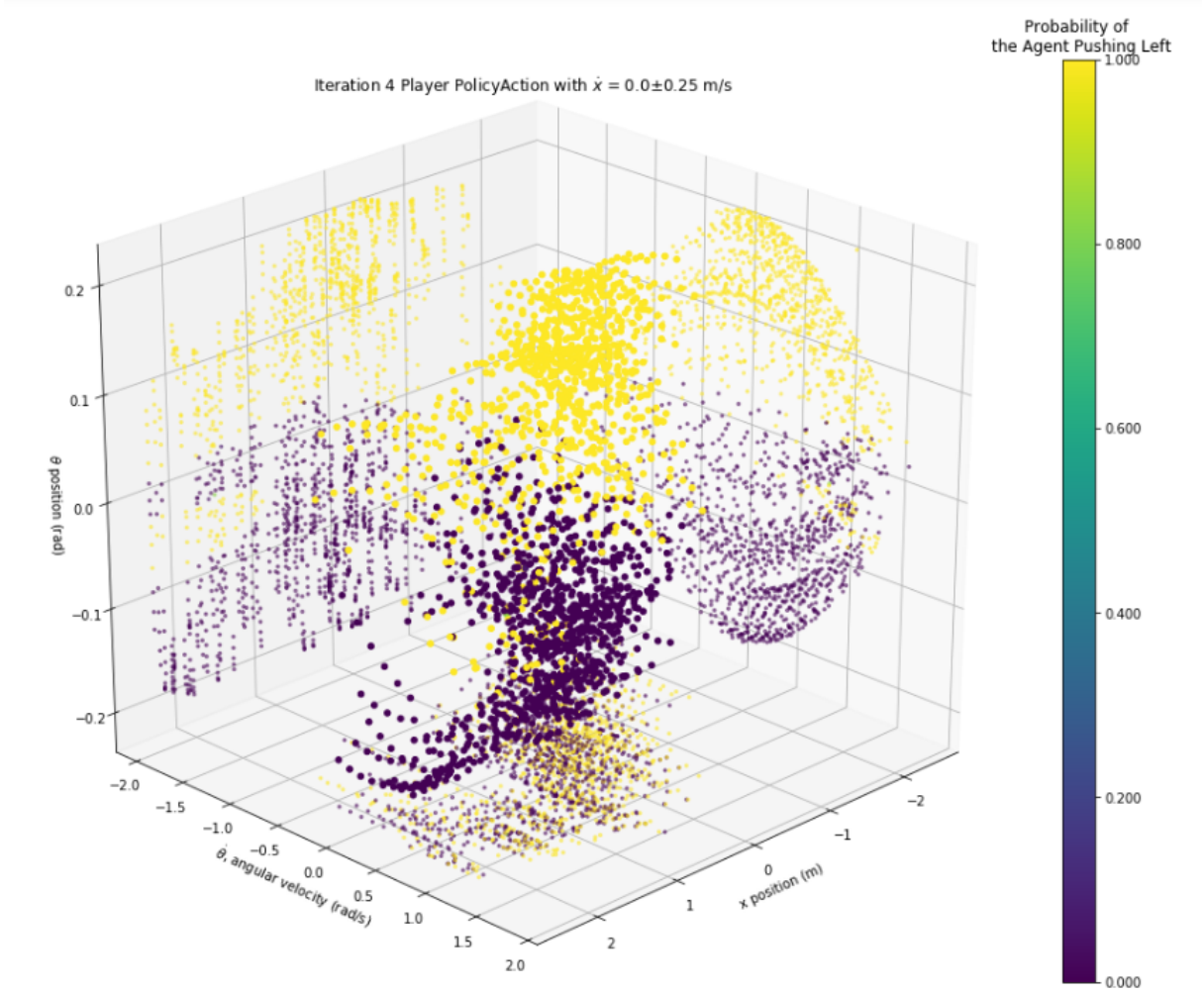


Figure 1.12: A 3D plot of policy-predicted actions after the 4th iteration of training of the CNN.

As the training iterations increased the oscillations became larger, until iteration 9 where they became large enough that the controller could not stabilise the pendulum at all.

Another possibility for the poor training of the CNN is the relatively small number of training examples used. Each iteration generates less than 16,000 examples (40 episodes each with 400steps), and at each iteration the last 5 iteration's examples are used for training, which gives a maximum of 80,000 training examples - or 40,000 for each agent. With 40 bins there are 40x40=1600 possible positions. The number of possible 2D-states increases as 1600^n, where n is the number time steps recorded, therefore, even after 2 time steps, there are 64 times more possible 2D-states than training examples.

1.2 Evaluation and Testing

In this section, the results of experiments with the player against a number of adversaries is discussed and the performance and robustness of the algorithm is evaluated.

1.2.1 Performance Against a Random Opponent

Against a random adversary with $F_2 = 0.05F_1$, the player's performance improves with the number of MCTS simulations when it has not been trained. However, the performance after training is constant, as shown in fig. 1.13.

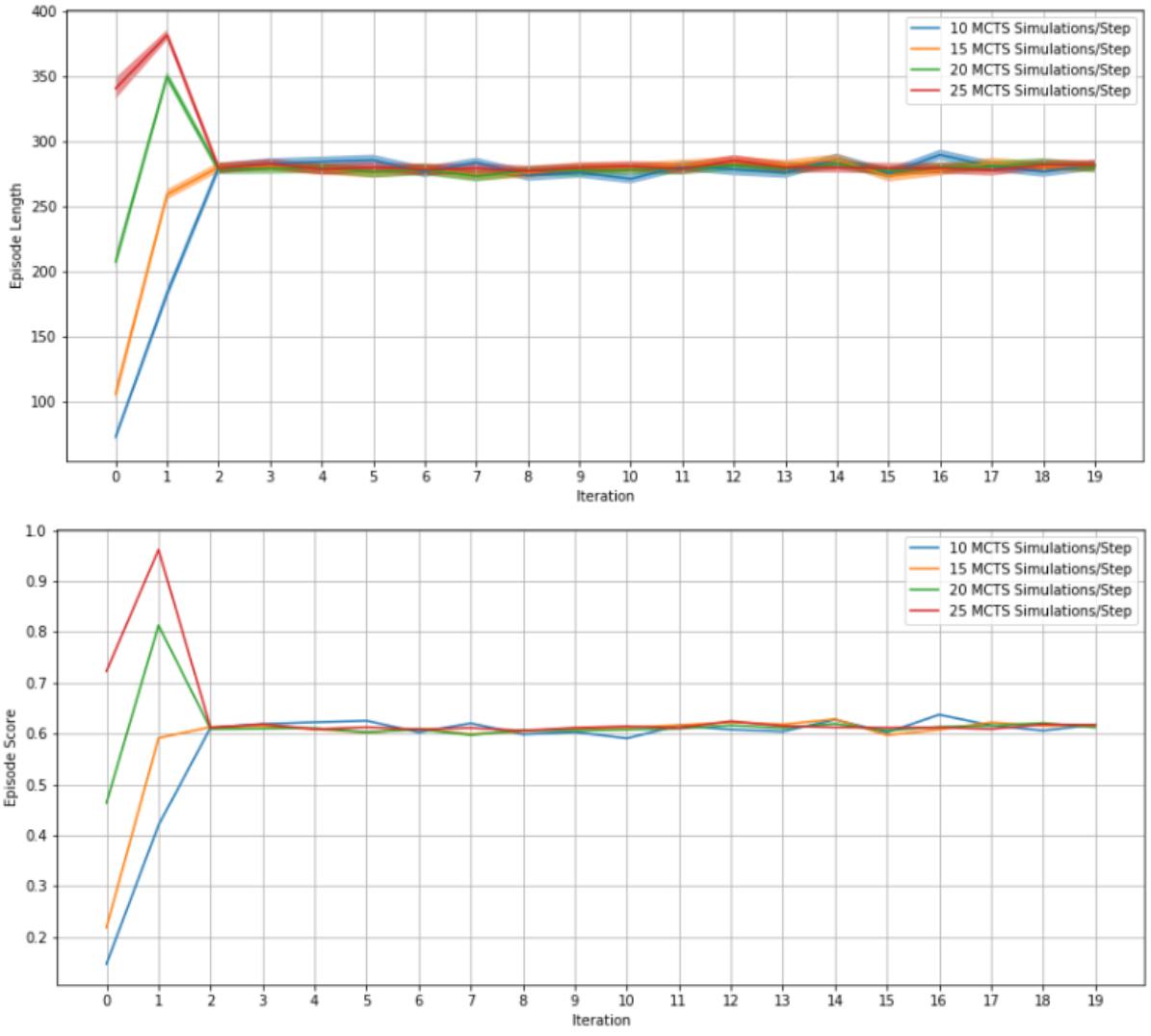


Figure 1.13: The length and length std, and mean score of a random adversary with $F_2 = 0.05F_1$.

This stagnation of performance is easily explained by the training of the adversary. The first iteration increases as the neural network is trained without an adversary, however

1.2.2 Performance Against Variable F_2

Is it robust?

explain why the trained adv vs trained player won't work - if they push they same way then great, if not then it is rubbish.

1.3 Conclusions And Future Work

could put an increasing cost on either player playing the same move too many times?

Make it more difficult for the player, e.g. add first order lag $F_t = \lambda F_{k-1} + (1 - \lambda)u_k$. This may allow exploitation of resonant frequencies by the adversary. Suggestions for fixing the reward hacking (reiterate what was said already). Think of some other ideas. Talk about how the potential benefits was/wasn't achieved.

A Appendices

A.1 Inverted Pendulum Dynamics Derivation

The state space equations for the Inverted Pendulum can be found using d'Alembert forces. Define the distance and velocity vectors to the important points:

$$\begin{aligned}
\boldsymbol{r}_P &= x\boldsymbol{i} \\
\boldsymbol{r}_{B_1/P} &= Lsin\theta\boldsymbol{i} + Lcos\theta\boldsymbol{j} \\
\boldsymbol{r}_{B_1} &= (x + Lcos\theta)\boldsymbol{i} + L\dot{\theta}sin\theta\boldsymbol{j} \\
\dot{\boldsymbol{r}}_{B_1} &= (\dot{x} + L\dot{\theta}cos\theta)\boldsymbol{i} - L\dot{\theta}sin\theta\boldsymbol{j}
\end{aligned}$$

Linear Momentum, $\boldsymbol{\rho} = \sum_i m_i \dot{\boldsymbol{r}}_{i/o} = m\dot{\boldsymbol{r}}_{B_1} + M\dot{\boldsymbol{r}}_P$:

$$\boldsymbol{\rho} = \begin{bmatrix} (M+m)\dot{x} + mL\dot{\theta}cos\theta \\ -mL\dot{\theta}sin\theta \\ 0 \end{bmatrix}$$

Moment of momentum about P, $\boldsymbol{h}_P = \boldsymbol{r}_{B_1/P} \times m\dot{\boldsymbol{r}}_{B_1}$:

$$\begin{aligned}
\boldsymbol{h}_P &= -mL(L\dot{\theta} + \dot{x}cos\theta)\boldsymbol{k} \\
\therefore \dot{\boldsymbol{h}}_P &= -mL(L\ddot{\theta} + \ddot{x}cos\theta - \dot{x}\dot{\theta}sin\theta)\boldsymbol{k}
\end{aligned}$$

Balance moments using $\dot{\boldsymbol{h}}_P + \dot{\boldsymbol{r}}_P \times \boldsymbol{\rho} = \boldsymbol{Q}_e$ and $\boldsymbol{Q}_e = \boldsymbol{r}_{B_1/P} \times -mg\boldsymbol{j} + \boldsymbol{r}_{B_2/P} \times F_2\boldsymbol{i}$:

$$\dot{\boldsymbol{h}}_P + \dot{\boldsymbol{r}}_P \times \boldsymbol{\rho} = \begin{bmatrix} 0 \\ 0 \\ -mL(\ddot{x}cos\theta + L\ddot{\theta}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -L(mgsin\theta + 2F_2cos\theta) \end{bmatrix} = \boldsymbol{Q}_e$$

And also balance linear momentum using $\boldsymbol{F}_e = \dot{\boldsymbol{\rho}}$:

$$\dot{\boldsymbol{\rho}} = \begin{bmatrix} (m+M)\ddot{x} + mL(\ddot{\theta}cos\theta - \dot{\theta}^2sin\theta) \\ -mL(\ddot{\theta}sin\theta + \dot{\theta}^2cos\theta) \\ 0 \end{bmatrix} = \begin{bmatrix} F_1 + F_2 \\ R - (M+m)g \\ 0 \end{bmatrix} = \boldsymbol{F}_e$$

Finally write the system dynamics in terms of $\ddot{\theta}$ and $\ddot{x}$:

$$\ddot{\theta}\big(M + msin^2\theta\big)L = \left(\frac{2M+m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta \quad \text{(A.1)}$$

$$\ddot{x}(M + msin^2\theta) = F_1 - F_2 cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta) \quad \text{(A.2)}$$

Simplifying by substituting in constants, the full state space equation are:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{\theta}\\ \ddot{\theta}\end{bmatrix} = \begin{bmatrix}\dot{x}\\ \frac{\left(\frac{2M+m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)}\\ \dot{\theta}\\ \frac{F_1 - F_2 cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)}\end{bmatrix} = \begin{bmatrix}f_1(\boldsymbol{x}, F_1, F_2)\\ f_2(\boldsymbol{x}, F_1, F_2)\\ f_3(\boldsymbol{x}, F_1, F_2)\\ f_4(\boldsymbol{x}, F_1, F_2)\end{bmatrix} \quad \text{(A.3)}$$

Using Lyapunov's indirect method, the linearised equations about the equilibrium, $\boldsymbol{x}_e = [x_e, \dot{x}_e, \theta_e, \dot{\theta}_e]^T = [0,0,0,0]^T$, are:

$$\begin{bmatrix}\delta\dot{x}\\ \delta\ddot{x}\\ \delta\dot{\theta}\\ \delta\ddot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{\partial f_1}{\partial x}\big|_{\boldsymbol{x}_e} & \frac{\partial f_1}{\partial \dot{x}}\big|_{\boldsymbol{x}_e} & \frac{\partial f_1}{\partial \theta}\big|_{\boldsymbol{x}_e} & \frac{\partial f_1}{\partial \dot{\theta}}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_2}{\partial x}\big|_{\boldsymbol{x}_e} & \frac{\partial f_2}{\partial \dot{x}}\big|_{\boldsymbol{x}_e} & \frac{\partial f_2}{\partial \theta}\big|_{\boldsymbol{x}_e} & \frac{\partial f_2}{\partial \dot{\theta}}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_3}{\partial x}\big|_{\boldsymbol{x}_e} & \frac{\partial f_3}{\partial \dot{x}}\big|_{\boldsymbol{x}_e} & \frac{\partial f_3}{\partial \theta}\big|_{\boldsymbol{x}_e} & \frac{\partial f_3}{\partial \dot{\theta}}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_4}{\partial x}\big|_{\boldsymbol{x}_e} & \frac{\partial f_4}{\partial \dot{x}}\big|_{\boldsymbol{x}_e} & \frac{\partial f_4}{\partial \theta}\big|_{\boldsymbol{x}_e} & \frac{\partial f_4}{\partial \dot{\theta}}\big|_{\boldsymbol{x}_e}\end{bmatrix}\begin{bmatrix}\delta x\\ \delta\dot{x}\\ \delta\theta\\ \delta\dot{\theta}\end{bmatrix} + \begin{bmatrix}\frac{\partial f_1}{\partial F_1}\big|_{\boldsymbol{x}_e} & \frac{\partial f_1}{\partial F_2}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_2}{\partial F_1}\big|_{\boldsymbol{x}_e} & \frac{\partial f_2}{\partial F_2}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_3}{\partial F_1}\big|_{\boldsymbol{x}_e} & \frac{\partial f_3}{\partial F_2}\big|_{\boldsymbol{x}_e}\\ \frac{\partial f_4}{\partial F_1}\big|_{\boldsymbol{x}_e} & \frac{\partial f_4}{\partial F_2}\big|_{\boldsymbol{x}_e}\end{bmatrix}\begin{bmatrix}\delta F_1\\ \delta F_2\end{bmatrix} \quad \text{(A.4)}$$

$$\begin{bmatrix}\delta\dot{x}\\ \delta\ddot{x}\\ \delta\dot{\theta}\\ \delta\ddot{\theta}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & \frac{mg}{M} & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & \frac{(m+M)}{ML}g & 0\end{bmatrix}\begin{bmatrix}\delta x\\ \delta\dot{x}\\ \delta\theta\\ \delta\dot{\theta}\end{bmatrix} + \begin{bmatrix}0 & 0\\ \frac{1}{M} & -\frac{1}{M}\\ 0 & 0\\ -\frac{1}{ML} & \frac{2M+m}{mML}\end{bmatrix}\begin{bmatrix}\delta F_1\\ \delta F_2\end{bmatrix} \quad \text{(A.5)}$$

The eigenvalues are given by $det(\lambda I - A) = \lambda^2(\lambda^2 - \frac{(m+M)}{ML}g) = 0$. Therefore, the system is unstable about $\boldsymbol{x}_e$ due to the right half plane pole, $\lambda = \sqrt{\frac{(m+M)}{ML}g}$. Additionally, the time constant of this unstable system is $\tau = \sqrt{\frac{ML}{g(m+M)}}$. Note, if $M >> m, \tau \to \sqrt{\frac{L}{g}}$, which is the time constant for a simple pendulum.

It can be proved that the inverted pendulum system is controllable by showing:

$$rank[\boldsymbol{B}\ \boldsymbol{AB}\ \boldsymbol{A^2B}\ \boldsymbol{A^3B}] = 4 \quad \text{(A.6)}$$

Therefore for any initial condition we can reach $\boldsymbol{x}_e$ in finite time under these linear assumptions.

A.2 Propagation of Quantisation Error

The state space model for the quantisation of the linearised inverted pendulum can be written as:

$$\boldsymbol{x}_t^{(2D)} = C\boldsymbol{x}_t + \boldsymbol{V}_t \qquad \boldsymbol{V}_t \sim \mathcal{U}\left(\begin{bmatrix} \frac{1}{\delta x} \\ \frac{1}{\delta\theta} \end{bmatrix}\right) \tag{A.7}$$

$$\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_t \tag{A.8}$$

Where A and B are the linearised system dynamics (valid for small time steps), and C is the linear transformation to a 2D state space, with quantisation noise $\mathbf{V}$.

Assuming the quantisation bin sizes, δx and $\delta\theta$, are small and that x and θ are independent within the bin, the quantisation noise can be modelled as uniform random variables with covariance, $cov(\boldsymbol{V}, \boldsymbol{V}) = \mathbb{E}[\boldsymbol{V}\boldsymbol{V}^T]$:

$$= \mathbb{E}\begin{bmatrix} x^2 & x\theta \\ \theta x & \theta^2 \end{bmatrix} = \begin{bmatrix} \int_{-\delta x/2}^{\delta x/2} x^2 \cdot \frac{1}{\delta x} dx & 0 \\ 0 & \int_{-\delta\theta/2}^{\delta\theta/2} \theta^2 \cdot \frac{1}{\delta\theta} d\theta \end{bmatrix} = \begin{bmatrix} \frac{\delta x^2}{12} & 0 \\ 0 & \frac{\delta\theta^2}{12} \end{bmatrix} \tag{A.9}$$

For simplicity, let $\delta x = \delta\theta$, and therefore, $cov(\boldsymbol{V}, \boldsymbol{V}) = \sigma_v^2 I$.

Kalman filtering can be used to find an optimal estimate $\hat{\boldsymbol{x}}_n = K[\boldsymbol{x}_n | \boldsymbol{y}_{n-k:n}]$ using the algorithm (derived in [3]).

Algorithm 1 Multivariate Kalman Filtering

1: **Given:** $\hat{\boldsymbol{x}}_n = K[\boldsymbol{x}_n|\boldsymbol{y}_{1:n}]$ and $\Sigma_n = \mathbb{E}[(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n)(\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n)^T]$

Prediction:

2: $\bar{\boldsymbol{x}}_{n+1} = K[\boldsymbol{x}_n|\boldsymbol{y}_{1:n}] = A\hat{\boldsymbol{x}}_n + B\boldsymbol{u}_{n+1}$

3: $\bar{\Sigma}_{n+1} = \mathbb{E}[(\boldsymbol{x}_{n+1} - \bar{\boldsymbol{x}}_{n+1})(\boldsymbol{x}_{n+1} - \bar{\boldsymbol{x}}_{n+1})^T] = A\Sigma_n A^T + \Sigma_w$ ▷ $\Sigma_w = \mathbf{0}$

Update:

4: $\tilde{\boldsymbol{y}}_{n+1} = \boldsymbol{y}_{n+1} - C\bar{\boldsymbol{x}}_{n+1}$ ▷ Calculate Innovation residual

5: $S_{n+1} = \Sigma_v + C\bar{\Sigma}_{n+1}C^T$ ▷ Calculate Innovation Covariance

6: $\hat{\boldsymbol{x}}_{n+1} = \bar{\boldsymbol{x}}_{n+1} + \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1}\tilde{\boldsymbol{y}}_{n+1}$ ▷ $\Sigma_v = \sigma_v^2 I$

7: $\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1}C)\bar{\Sigma}_{n+1}$

Thus we can find the optimal linear estimate of the covariance of Equation (A.7) from algorithm 1 line 7:

$$\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1}C^T S_{n+1}^{-1}C)\bar{\Sigma}_{n+1} \tag{A.10}$$

$$= (I - \bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\bar{\Sigma}_{n+1}, \qquad where\ \ \bar{\Sigma}_{n+1} = A\Sigma_n A^T \tag{A.11}$$

For the univariate case this reduces to:

$$\sigma_{n+1}^2 = \bar{\sigma}_{n+1}^2(1 - \frac{C^2\bar{\sigma}_{n+1}^2}{\sigma_v^2 + C^2\bar{\sigma}_{n+1}^2}) \tag{A.12}$$

$$= A^2\sigma_n^2(1 - \frac{C^2A^2\sigma_n^2}{\sigma_v^2 + C^2A^2\sigma_n^2}) \tag{A.13}$$

It is obvious from this that as $\sigma_v^2 \to 0$, $\sigma_{n+1}^2 \to 0$. Furthermore, if the spectral radius, $\rho\left(A^2(1 - \frac{C^2A^2\sigma_n^2}{\sigma_v^2+C^2A^2\sigma_n^2})\right) < 1$, then the mean squared error from quantisation will reduce to zero. Since $0 < ||\frac{C^2A^2\sigma_n^2}{\sigma_v^2+C^2A^2\sigma_n^2}||_2^2 < 1$ if $\sigma_v^2 > 0$, a sufficient condition is that $A^2 \leq 1 \implies \rho(A) \leq 1$.

Using Gelfand's Theorem, $\rho(M_1...M_n) \leq \rho(M_1)...\rho(M_n)$, and the eigenvalue identity, $(M + cI)v = (c + \lambda)v$. The MSE of the multivariate case can be reduced to:

$$1 > \rho\left((I - \bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\bar{\Sigma}_{n+1}\right) \tag{A.14}$$

$$> \rho(\bar{\Sigma}_{n+1})\left[1 - \rho(\bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\right] \tag{A.15}$$

Therefore, if $\rho\left(\bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\right)$ is less than one, and $\rho(\bar{\Sigma}_{n+1}) < 1$, then this is true. This is equivalent to showing that if $0 < \kappa(\Sigma_n)\kappa(A)^2\kappa(CC^T) \leq 1$ and $\rho(AA^T) < 1$, where $\kappa(\cdot)$ denotes the condition number, then the MSE will decay:

$$\begin{aligned}
\rho\left(\bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\right) &\leq \rho(\bar{\Sigma}_{n+1})\rho(CC^T)\rho((\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}) \\
&\leq \rho(\bar{\Sigma}_{n+1})\rho(CC^T)\frac{1}{|\lambda_{min}(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)|} \\
&= \rho(\bar{\Sigma}_{n+1})\rho(CC^T)\frac{1}{|\lambda_{min}(C\bar{\Sigma}_{n+1}C^T)| + \sigma_v^2} \\
&< \frac{|\lambda_{max}(\bar{\Sigma}_{n+1})\lambda_{max}(CC^T)|}{|\lambda_{min}(C\bar{\Sigma}_{n+1}C^T)|} \\
&< \frac{|\lambda_{max}(\bar{\Sigma}_{n+1})\lambda_{max}(CC^T)|}{|\lambda_{min}(\bar{\Sigma}_{n+1}CC^T)|} \\
&< \frac{|\lambda_{max}(\bar{\Sigma}_{n+1})\lambda_{max}(CC^T)|}{|\lambda_{min}(\bar{\Sigma}_{n+1})\lambda_{min}(CC^T)|} \\
&= |\kappa(\Sigma_n)|\kappa(A)^2\kappa(CC^T)
\end{aligned}$$

where the above uses the results $\lambda_i(X\Sigma X^*) = \lambda_i(\Sigma X^* X)$, where Σ is symmetric (this equality also holds for spectral radii) and $\lambda_{min}(\bar{\Sigma}_{n+1}CC^T) > \lambda_{min}(\bar{\Sigma}_{n+1})\lambda_{min}(CC^T)$. Note that $\kappa(\Sigma_n) = 1$. Therefore sufficient conditions for the decay of the covariance are:

$$(1) \qquad \rho(AA^T) \leq 1 \implies |\lambda_{max}(AA^T)| \leq 1 \tag{A.16}$$

$$(2) \qquad \kappa(A)^2 \leq 1 \implies |\lambda_{min}(A)| = |\lambda_{max}(A)| \tag{A.17}$$

$$(3) \qquad \kappa(CC^T) \leq 1 \tag{A.18}$$

For the linearised dynamics derived in eq. (A.5) the eigenvalues of AA^T, λ, are given by $\lambda^2(1-\lambda)^2 = 0$, and the eigenvalues of A are $\lambda = (0,\ 0,\ \pm\sqrt{\frac{(m+M)}{ML}g})$. Therefore (1) and (2) are true.

Therefore, the covariance decays to zero if $\rho(CC^T) \leq 1$ in the linearised multivariate case and the 2D state becomes a lossless estimate of the state. Given that C is a 2x4 projection matrix of the form

$$C = \begin{bmatrix} c_{11} & 0 & 0 & 0 \\ 0 & 0 & c_{23} & 0 \end{bmatrix} \implies CC^T = \begin{bmatrix} c_{11}^2 & 0 \\ 0 & c_{23}^2 \end{bmatrix} \tag{A.19}$$

By the choice of $\boldsymbol{x}^{(2D)}$ as square, $c_{11} = c_{23}$. By the same logic as above, $\kappa(CC^T) = 1$.

The rate of decay for the univariate case can be determined with a first order approximation about $\sigma_v^2 = 0$:

$$\sigma_{n+1}^2(\sigma_v^2) = A^2\sigma_n^2(1 - \frac{C^2A^2\sigma_n^2}{\sigma_v^2 + C^2A^2\sigma_n^2}) \tag{A.20}$$

$$\approx \sigma_{n+1}^2(0) + \delta\sigma_v^2 \frac{\partial\sigma_{n+1}^2(\sigma_v^2)}{\partial\sigma_v^2}|_{\sigma_v^2=0} \tag{A.21}$$

$$= \frac{\delta\sigma_v^2}{(AC\sigma_n)^2} \tag{A.22}$$

A.3 Neural Network Losses

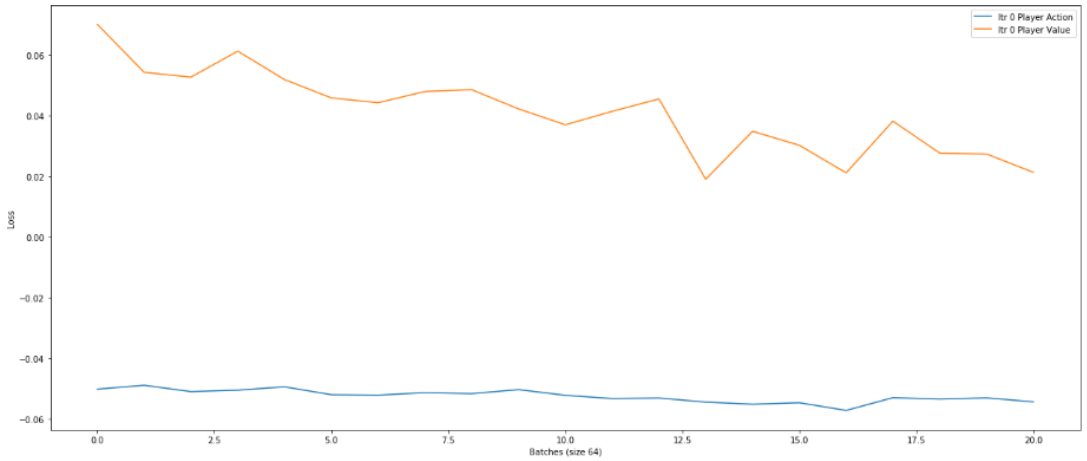


Figure A.1: The losses for both the player and adversary neural networks after one set of training examples.

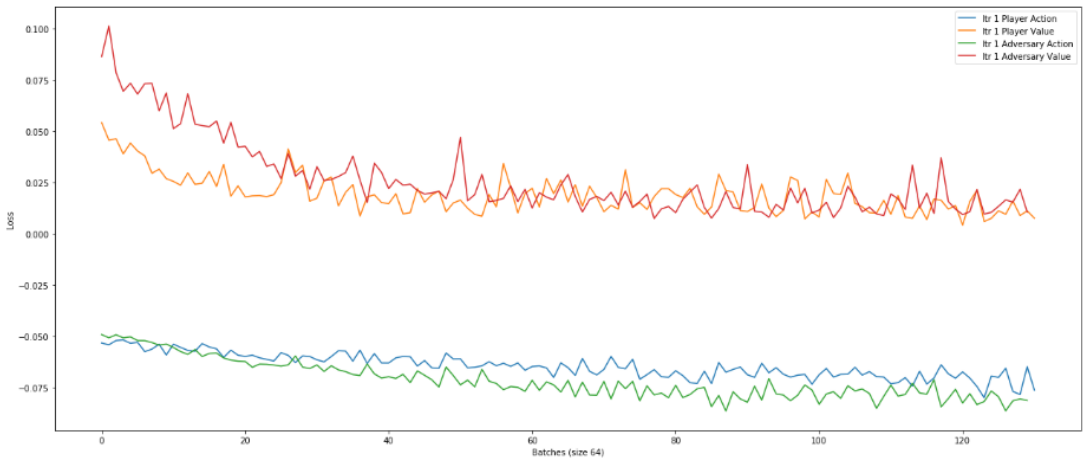


Figure A.2: The losses for both the player and adversary neural networks after two sets of training examples.

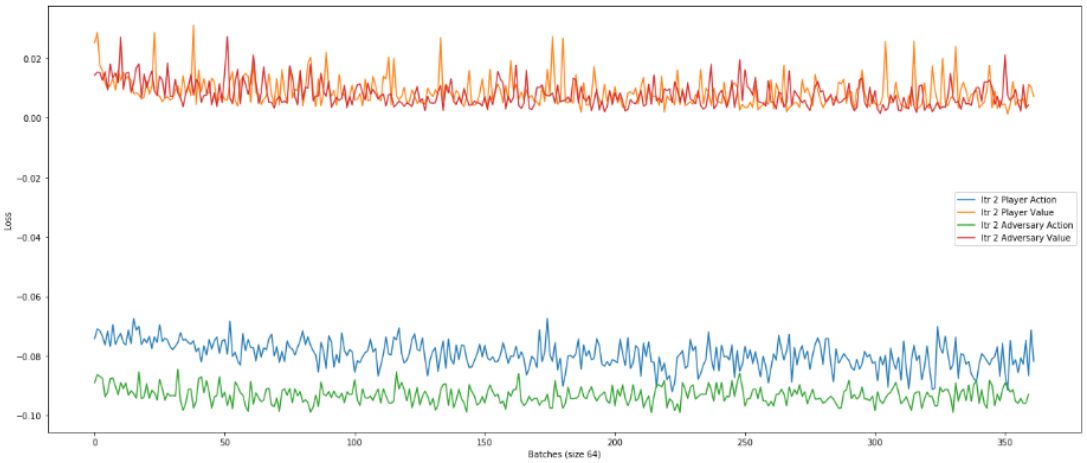


Figure A.3: The losses for both the player and adversary neural networks after three sets of training examples.

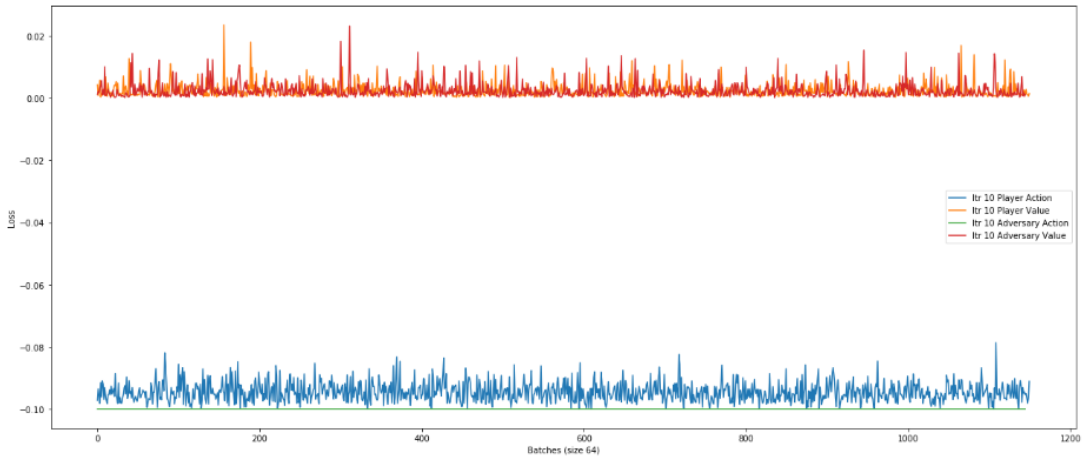


Figure A.4: The losses for both the player and adversary neural networks after ten sets of training examples.

B References

[1] M.C. Smith, I Lestas, *4F2: Robust and Non-Linear Control* Cambridge University Engineering Department, 2019

[2] G. Vinnicombe, K. Glover, F. Forni, *4F3: Optimal and Predictive Control* Cambridge University Engineering Department, 2019

[3] S. Singh, *4F7: Statistical Signal Analysis* Cambridge University Engineering Department, 2019

[4] Arthur E. Bryson Jr, *Optimal Control - 1950 to 1985*. IEEE Control Systems, 0272-1708/95 pg.26-33, 1996.

[5] I. Michael Ross, Ronald J. Proulx, and Mark Karpenko, *Unscented Optimal Control for Space Flight*. ISSFD S12-5, 2014.

[6] Zheng Jie Wang, Shijun Guo, Wei Li, *Modeling,Simulation and Optimal Control for an Aircraft of Aileron-less Folding Wing* WSEAS TRANSACTIONS on SYSTEMS and CONTROL, ISSN: 1991-8763, 10:3, 2008

[7] Giovanni Binet, Rainer Krenn and Alberto Bemporad, *Model Predictive Control Applications for Planetary Rovers*. imtlucca, 2012.

[8] Raković, Saša, *"Robust Model-Predictive Control*. Encyclopedia of Systems and Control, pg.1-11 ,2013.

[9] Russ Tedrake, *Underactuated Robotics*. MIT OpenCourseWare, Ch.3, Spring 2009.

[10] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction (2nd Edition)*. The MIT Press, Cambridge, Massachusetts, London, England. 2018.

[11] I. Carlucho, M. De Paula, S. Villar, G. Acosta . *Incremental Q-learning strategy for adaptive PID control of mobile robots*. Expert Systems with Applications. 80. 10.1016, 2017

[12] Yuxi Li, *Deep Reinforcement Learning: An Overview*. CoRR, abs/1810.06339, 2018.

[13] Sandy H. Huang, Martina Zambelli, Jackie Kay, Murilo F. Martins, Yuval Tassa, Patrick M. Pilarski, Raia Hadsell, *Learning Gentle Object Manipulation with Curiosity-Driven Deep Reinforcement Learning*. arXiv 2019.

[14] David Silver, Julian Schrittwieser, Karen Simonyan et al, *Mastering the game of Go without human knowledge*. Nature, vol. 550, pg.354–359, 2017.

[15] David Silver, Thomas Hubert, Julian Schrittwieser et al, *A general reinforcement learning algorithm that masters chess, shogi and Go through self-pla*. Science 362:6419, pg.1140-1144, 2018.

[16] S. Thakoor, S. Nair and M. Jhunjhunwala, *Learning to Play Othello Without Human Knowledge* Stanford University Press, 2018 `https://github.com/suragnair/alpha-zero-general`

[17] HowlingPixel.com, *Elo Rating System*. `https://howlingpixel.com/i-en/Elo_rating_system` acc: 11/03/2019. published 2019