0 Contents

1 Theory and Methods

1.1 The Inverted Pendulum (IP)

1.1.1 Dynamics

The Inverted Pendulum is an inherently unstable system with highly nonlinear dynamics and is under-actuated.

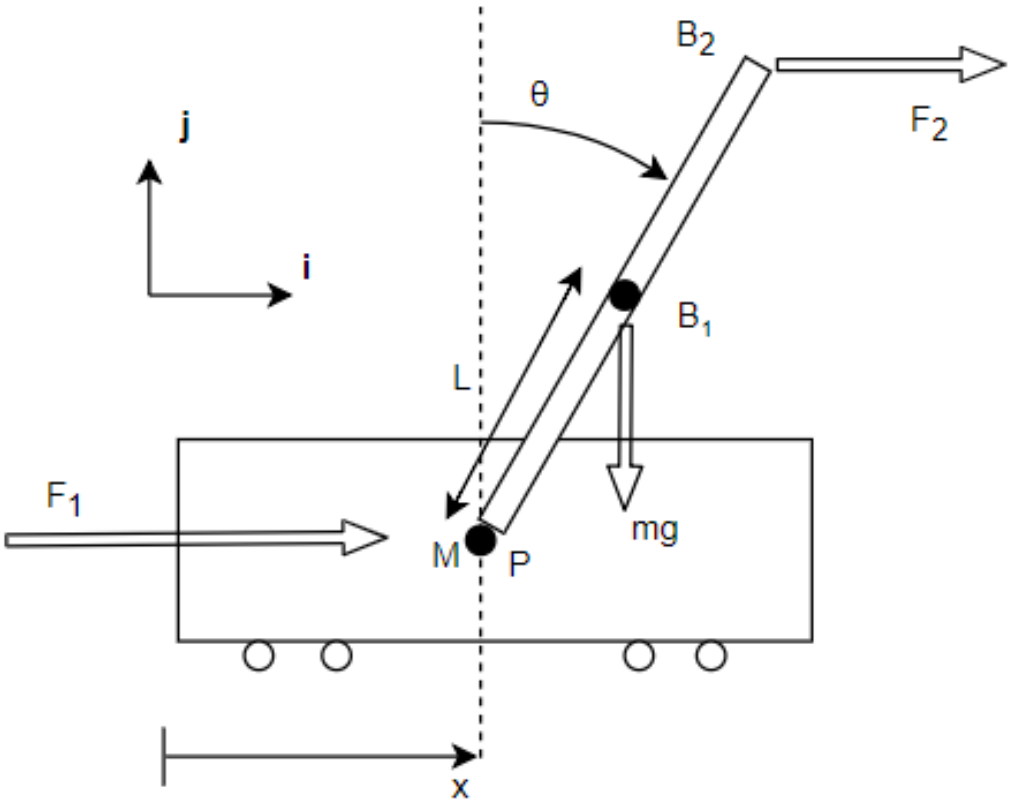

Figure 1.1: A free-body diagram of the inverted pendulum system. For the OpenAI IP the system is in discrete time and constrained by $L = 0.5$m, $m = 0.1$kg, $M = 1$kg, $F = \pm 10$N, $x_{max} = \pm 2.4$m, $\theta_{max} = \pm 12^o$.

The full state space equations for the inverted pendulum as defined in fig. 1.1 are given

by:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \frac{\left(\frac{2M+m}{m}F_2 - F_1\right)cos\theta + g(M+m)sin\theta - mL\dot{\theta}^2 sin\theta cos\theta}{(M+msin^2\theta)} \\ \dot{\theta} \\ \frac{F_1 - F_2 cos(2\theta) + msin\theta(L\dot{\theta}^2 - gcos\theta)}{L(M+msin^2\theta)} \end{bmatrix} \tag{1.1}$$

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

$$\begin{bmatrix} \delta\dot{x} \\ \delta\ddot{x} \\ \delta\dot{\theta} \\ \delta\ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m+M)}{ML}g & 0 \end{bmatrix} \begin{bmatrix} \delta x \\ \delta\dot{x} \\ \delta\theta \\ \delta\dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{1}{M} & -\frac{1}{M} \\ 0 & 0 \\ -\frac{1}{ML} & \frac{2M+m}{mML} \end{bmatrix} \begin{bmatrix} \delta F_1 \\ \delta F_2 \end{bmatrix} \tag{1.2}$$

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 $\tau = \sqrt{\frac{ML}{g(m+M)}} = 0.22s$. Note, if $M >> m, \tau \rightarrow \sqrt{\frac{L}{g}}$, which is the time constant for a simple pendulum.

OpenAI's gym is a python package that supplies an inverted pendulum environment built-in (called CartPole). This environment was wrapped to use the dynamics above and other extra functionality, whilst providing a rendering function shown in fig. 1.2.

update picture

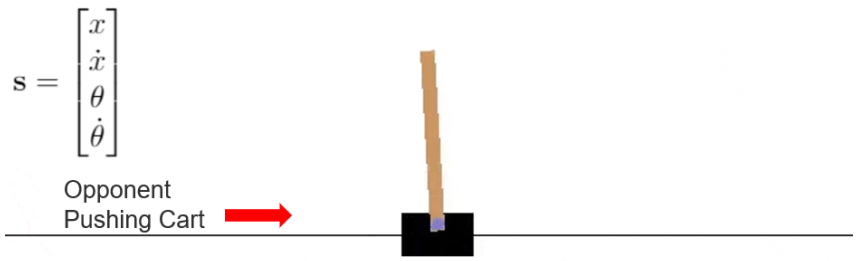

Figure 1.2: The OpenAI gym CartPole environment. The classical state representation is shown in the top left. Actions by the player and the adversary are taken as an impulse to the left or right as defined in fig. 1.1.

1.1.2 Cost and Value Function

For each step/impulse, the 2D state is calculated and a cost, is calculated as in eq. (1.3), where $\boldsymbol{w}^T = [w_1, w_2, w_3, w_4] = [0.25, 0.1, 0.7, 1]$ and $0 \geq c(x_t, u_t) \geq -1$. The weights, $\boldsymbol{w}$, were chosen through empirical measurement of the the importance of each state ***.

$$c(x_t, u_t) = -\frac{1}{\sum_i w_i} \cdot \hat{\boldsymbol{x}}^T \begin{bmatrix} w_1 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_3 & 0 \\ 0 & 0 & 0 & w_4 \end{bmatrix} \hat{\boldsymbol{x}} \tag{1.3}$$

$$\text{where, } \hat{\boldsymbol{x}}^T = \left[\frac{x_t}{x_{max}}, \frac{\dot{x}_t}{\dot{x}_{max}}, \frac{\theta_t}{\theta_{max}}, \frac{\dot{\theta}_t}{\dot{\theta}_{max}} \right]^T \tag{1.4}$$

Weighting on the inputs was set to zero, as there are only two inputs for this problem, thus the cost can be written as $c(x_t)$. The max values can be approximated experimentally (note, $x_{max} = 2.4$ and $\theta_{max} = 12^o$ are given constraints):

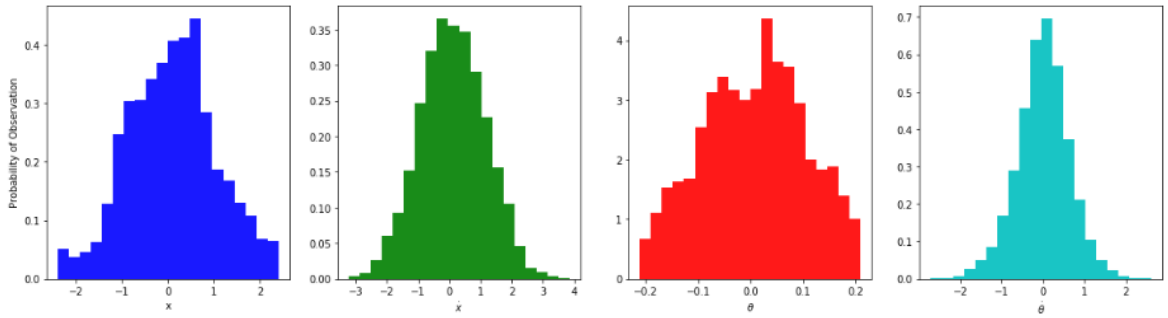

Figure 1.3: Histograms of typical state values. The frequencies greatly depend on the quality of the controller, with better controllers giving much narrower distributions. However, these are typical for a controller of medium efficacy over many episodes where the starting state is randomised (possibly to an uncontrollable state).

Suitable estimates for the the values of $\dot{x}_{max}$ and $\dot{\theta}_{max}$ are thus approximately 3 and 2 respectively. The value function is computed after the episode has completed, and is the discounted future losses at each state. A horizon constraint of $\gamma^k < \frac{1}{20}$ was chosen as it is a standard factor for insignificance. Since the steps taken after a terminal state has been reached, k, must be defined with the game dynamics (the `CartPoleWrapper` class), it is a constant. Therefore γ is calculated as $\gamma < \frac{1}{20}^{\frac{1}{k}}$. The discounted future values are calculated using a geometric series:

$$v_0 = \frac{\sum_{\tau=0}^{k} \gamma^\tau c(x_\tau)}{\sum_{\tau=0}^{k} \gamma_\tau}, \qquad \text{where } \gamma^k < \frac{1}{20} \tag{1.5}$$

Where for simplicity of notation, $v_0 = v(t)$, the state value at step t.

1.1.3 State Representations

The state can be represented in a number of ways. The simplest method would be $\boldsymbol{x} = [x, \dot{x}, \theta, \dot{\theta}]$. This has a number of advantages such as lower computational cost, greater numerical accuracy (if the process is fully observable) and simpler implementation. Conversely, a 2-dimensional (2D) representation may be used. There are several possibilities for this, all of which first require binning $\boldsymbol{x}$:

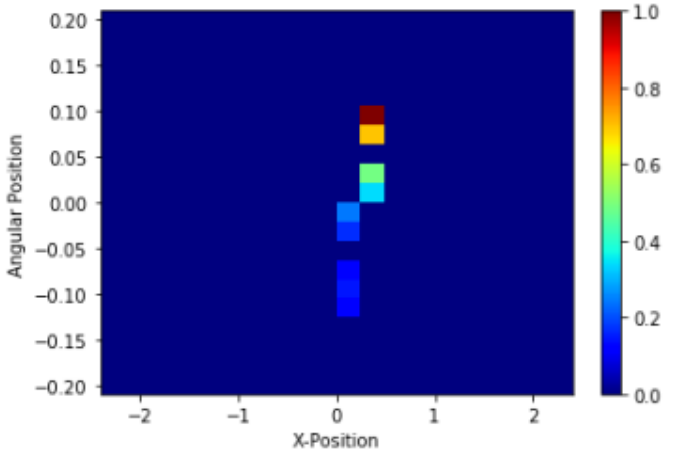

Figure 1.4: An example of a 2D state representation where there are 20 bins and 17 random actions have been taken.

(1) A matrix stack of x vs $\dot{x}$ and θ vs $\dot{\theta}$, both of which would only have one non-zero entry. This scales as b^n where b = number of bins and n = number of states.

(2) A matrix stack of x_t vs x_{t-1} for all states. Similarly this scales as b^n, however the derivative states do not need to be plotted as these can be inferred. This has the advantage that, if the derivatives are not observable, they are built into the 2D representation, however, if they are observable then this is less accurate than (1).

(3) A matrix of discounted previous states, forming a motion history image. An example of this is shown in section 1.1.3, and Algorithm 1 shows the implementation details. This was the chosen representation.

A 2D representation such as this allows us to use a convolutional neural network, which has the benefit of various transformation invariances - these are particularly useful for the inverted pendulum since it is symmetric.

Algorithm 1 Create 2D State

function GETSTATE2D($\hat{\boldsymbol{x}}_{t-1}^{(2D)}, binEdges, nBins$)
    $\hat{\boldsymbol{x}} \leftarrow$ getNormedState()
    for all $x_i \in \hat{\boldsymbol{x}}_t$ **do**
        $x_i \leftarrow \text{argmin}|binEdges - x_i|$ ▷ Get the index of the nearest bin edge.
    end for
    $HistEdges \leftarrow$linspace$(-0.5, nBins - 0.5, nBins + 1)$ ▷ Centre by shifting -0.5
    $\hat{\boldsymbol{x}}_t^{(2D)} \leftarrow$ histogram2d$(x, \theta, \text{bins} = (histEdges, histEdges))$ ▷ Inbuilt function
    return $\hat{\boldsymbol{x}}_t^{(2D)} + \lambda \hat{\boldsymbol{x}}_{t-1}^{(2D)}$
end function

The state space model for the quantisation of the linearised inverted pendulum (valid for small time steps) can be modelled as (where $\mathcal{U}$ is a uniform distribution, centred on 0):

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

$$\boldsymbol{x}_t^{(2D)} = C\boldsymbol{x}_t + \boldsymbol{V}_t \qquad \boldsymbol{V}_t \sim \mathcal{U}\left(\begin{bmatrix} \frac{1}{\delta x} \\ \frac{1}{\delta\theta} \end{bmatrix}\right) \tag{1.6}$$

$$\boldsymbol{x}_t = A\boldsymbol{x}_{t-1} + B\boldsymbol{u}_t \tag{1.7}$$

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 $\mathbf{V}$.

The initial mean squared error (MSE), and the propagation of the error (when estimated optimally) are given by eqs. (1.8) and (1.9) where $\delta x = \delta\theta$ (derivation details can be found in appendix A.2).

$$\Sigma_0 = \sigma_v^2 I = \begin{bmatrix} \frac{\delta x^2}{12} & 0 \\ 0 & \frac{\delta\theta^2}{12} \end{bmatrix} \tag{1.8}$$

$$\Sigma_{n+1} = (I - \bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\bar{\Sigma}_{n+1} \tag{1.9}$$

Where $\bar{\Sigma}_{n+1} = A\Sigma_n A^T$. When A is the linearised inverted pendulum dynamics and C is an arbitrary matrix to convert $\boldsymbol{x}_t$ into a 2D state, it can be shown (see appendix A.2) that as $t \to \infty$, the covariance of quantisation decays to zero since $\lim_{\delta x \to 0} \sigma_v^2 = 0$ and therefore this form for a 2D state becomes lossless (assuming that the neural network acts as an optimal kalman filter). For a given bin size, the MSE decreases linearly, however it decreases quadratically with bin size, therefore the MSE will always decay to zero. A fine bin size will improve the rate of decay but increase the computational load of when computing the state and through the neural network.

With a non-zero bin size the overall error can be reduced by binning more densely in regions in which the IP is expected to spend more time. Figure 1.3 shows that the state visit frequencies roughly correspond to normal distribution, therefore by transforming the bins with an inverse gaussian c.f.d. a flattened distribution can be obtained with a greater density of bins in the centre (fig. 1.5). This has the additional benefit of allowing finer control where it is needed. For example, if the pole is far from the equilibrium the optimal action more easily determined, and subject to less change with small state variations, therefore coarser binning can be used.

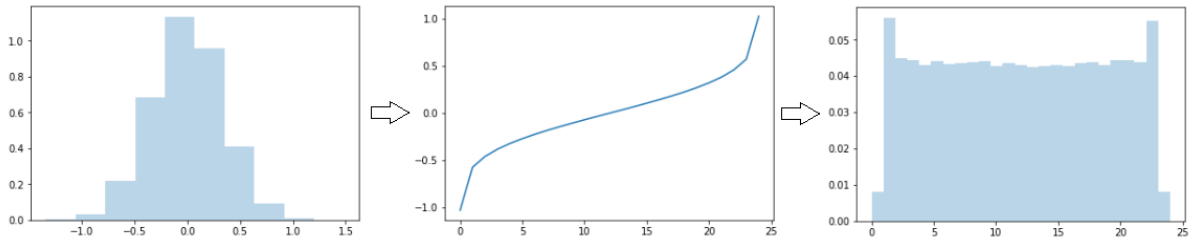

Figure 1.5: Binning of a gaussian distribution with bin edges scaled with an inverse gaussian c.f.d. For this example there are 25 bins.

1.1.4 Discrete vs Continuous Time

For a small enough discrete time step, the simulation will approximate the continuous process very well. Under a continuous action space, the sampling rate must be faster than the system dynamics to ensure controllability. When the action space is restricted to $u \in \{-1, 1\}$, there is a trade off between the sampling rate and the size of the action space. If the sampling rate is fast enough pseudo-continuous actions can be achieved via pulse-width modulation. For the IP system described, the time constant, τ, is approximately 0.022s. Moreover, the agents move alternately, therefore the sampling time step δt, must be less than 0.011s, however due to computational limitations, a time step of 0.01s was chosen.

The positional change per time step can be computed by considering the average maximum velocities: $3m/s$ and $2rad/s$. Thus the maximum positional change between an agent's actions is $\dot{x}_{i,max} \times \frac{\delta t}{x_{i,max}} = 2.5\%$ and 30% respectively. In practice a 30% change in θ would only occur around θ_{max} and would be uncontrollable. Therefore, the choice of $\delta t = 0.01$ is not detrimental to the control, but will not give pseudo-continuous actions.

1.2 Self Play and Adversaries

1.2.1 Cost and Value Functions

In AlphaZero, the adversary is working to minimise the value function whereas the player is working to maximise it. For board games, where agents are on equal footing, a value function $(-1 < v < 1)$ representing the expected outcome can be used. This has the advantage of symmetry - when the board is viewed from the other agent's perspective the value can be multiplied by -1, giving the expected outcome for that agent.

The adversary for the inverted pendulum has the additional advantage of gravity, making the play asymmetric. Given that both the state-value and cost are $-1 < v, c < 0$, multiplying by -1 would mean the adversary is maximising a value function $0 < v < 1$. State-values outside these ranges have no meaning. If a single neural network is used,

values close to equilibrium may be predicted into the wrong range. Consequently, both the adversary and the player must predict true state-values. This also has the advantage of maintaining an intuitive meaning of the state-value.

1.2.2 Choice of Action

The inverted pendulum system is symmetric in both x and θ. By taking advantage of this the number of training examples could be doubled by reflecting episodes along the axis of symmetry. However, as shown with AlphaZero [15], this provides minimal benefit and also hinders generalisation. The adversarial point of action was chosen to be at the top of the pole, acting horizontally (fig. 1.1), thus ensuring that two distinct policies must be learnt, rather than one being just inverse probabilities of the other. For simplicity $u_{adv} \in h \cdot \{-1, 1\}$ was chosen, where h is a handicap applied to the adversary.

The handicap can be approximated using the linearised dynamics eq. (1.2) by investigating what force would be needed by each agent to achieve the same change change in the state (eqs. (1.10a) and (1.10b)) (setting $\delta \boldsymbol{x} = \mathbf{0}$).

$$\delta\ddot{x} = \frac{1}{M}\delta F_1 - \frac{1}{M}\delta F_2 \quad \Longrightarrow \quad \delta F_2 \approx \delta F_1 \tag{1.10a}$$

$$\delta\ddot{\theta} = -\frac{1}{ML}\delta F_1 + \frac{2M+m}{mML}\delta F_2 \quad \Longrightarrow \quad \delta F_2 \approx \frac{m}{2M+m}\delta F_1 \tag{1.10b}$$

Empirically, the pendulum is more likely to fall rather than to exceed x_{max}. Therefore, a good estimate of h would be $\frac{m}{2M+m} = \frac{0.1}{2+0.1} \approx 0.0476$. When testing this it was found that if F_1 was set near $0.04 - 0.05$ initially, the adversary gets the upper-hand quickly and the pendulum falls very quickly, not allowing the player to gain any training experience. Therefore, for the first N iterations, the adversary force was set to 0, and increased as the number of iterations increased according to $h = (1 - 0.7^{i-N+1})$, where i is the iteration, and 0.7 was chosen such that the adversary could be implemented on the first iteration without overpowering the still-training player.

1.2.3 Agent Representation

There are two good ways of representing whether it is the adversary's turn for the neural network:

Multiply the board representation by -1 such that opponent pieces are negative. This has the disadvantages that it can only take two agents and, a network that outputs both state-values and action *p.m.f.*'s should predict the same state values for both player and adversary, but predict vastly different actions. Therefore, the values could be decoupled from the actions, which was one of the major benefits of

using a single neural network. However, a single network is simpler, and negating the board more closely follows AlphaZero's methodology.

Record the agent with each example and use agent-labeled examples to train different neural networks. Using a neural network for each agent causes half of the examples to be lost as only the relevant agent's examples are used to train each network. However, this does not suffer from the problems above and can cope with agents with a different number of possible actions more easily.

Recording the agent with each example was therefore chosen. Note, in the case of the inverted pendulum, the optimal action is the inverse of the worst action. However, this is not a general result. For example, in a system with non-linear and asymmetric dynamics it is possible to have the target perpendicular to the closest edge of the invariant set, thus for the adversary it is better to push the system into instability rather than away from equilibrium.

1.2.4 Episode Execution and Policy Iteration

Pseudocode for episode execution following the sections above is shown in Algorithm 2.

Algorithm 2 Execute Episode

```
function executeEpisode
    example ← []
    x, x^(2D), c ← resetEpisode()        ▷ Set initial x randomly and initialise the cost
    agent ← 0
    repeat
        π ←getActionProb(x, x^(2D), agent)        ▷ Perform MCTS Simulations
        example.append((x^(2D), π, c, agent))
        u ∼ π                        ▷ Sample action. Note after S steps, τ → 0
        x, c ←step(x, u)                        ▷ Take next true episode step
        x^(2D) ←getState2D(x, x^(2D))
        agent ← nextAgent(agent)
    until episodeEnded(x)
    example ← convertCostsToValues(example)
    return example
end function
```

The action, u, is sampled from $\boldsymbol{\pi}$. However, after S steps, the temperature, $\tau \to 0$, which is equivalent to $u = \text{argmax}\ \boldsymbol{\pi}$.

The overall policy iteration algorithm is given by section 1.2.4.

Algorithm 3 Policy Iteration - Training the NeuralNetwork

```
function PolicyIteration
    nnet ← NeuralNetwork                          ▷ Initialise NeuralNetwork
    examples ← [ ]
    for iter in policyIterations do
        incrementHandicap(iter)                   ▷ F_2 = 0 for first n iters
        for ep in trainingEpisodes(iter) do       ▷ More episodes for first n iters
            resetMCTS(nnet)                       ▷ Use the most current nnet
            example ← executeEpisode()
            examples.append(example)
        end for
        recents ← getMostRecentExamples(examples) ▷ Last N policys' examples
        nnet ← trainNeuralNetwork(recents, nnet)
        saveNeuralNetwork(nnet)
    end for
end function
```

1.3 Neural Networks

1.3.1 Loss Functions and Pareto

The following loss function is minimised when training the neural networks:

$$\mathcal{L} = \sum_t (v_\theta(\boldsymbol{x}_t^{(2D)}) - v_t)^2 + c_{pareto} \cdot \boldsymbol{\pi} \cdot log(\boldsymbol{p}_\theta(\boldsymbol{x})) + c||w_{nnet}||^2 \tag{1.11}$$

Where c_{pareto} is a constant. During training the agents move probabilistically throughout, rather than switching to acting greedily after 30 moves as in AlphaZero [15]. As such, mean squared error was used for the action-losses.

`is this actually a good idea? Why not use greedy?`

Due to the asymmetric nature of the problem, the value and action-losses can be of very different magnitudes. A constant factor, c_{pareto}, was used to prevent this.

2 epochs

1.3.2 Architectures

Two neural network architectures are used, one that takes the raw state, $\boldsymbol{x} = [x\ \dot{x}\ \theta\ \dot{\theta}]^T$ as input, and another that takes the 2D state, $\boldsymbol{x}^{(2D)}$. The adversary and player have separate networks, both predicting the state-value and action $p.m.f$s.

For the network with $\boldsymbol{x}^{(2D)}$ as input, there are 2 fully connected convolutional layers with max-pooling, followed by 2 feedforward layers with $ReLU$ activations. The two

"heads" split here and have 2 fully-connected layers each, with the value head outputting v_θ and the action head outputting $\boldsymbol{p}_\theta$. The network with $\boldsymbol{x}$ as input is the same, except the convolutional layers are replaced with 2 feedforward layers.

Both architectures perform training with the *Adam Optimiser*, a mini-batch size of 8, a dropout of 0.3 and batch normalisation. The networks are implemented in pytorch[1].

The neural network is evaluated once every step in the MCTS, therefore in an episode of length L and S MCTS simulations/step, the neural network is evaluated more than $L \times S$ times. The GPU used to run experiments is a single NVIDEA GeForce GTX 1050. The convolutional layers of a neural network are the most computationally expensive part and, increasing with the image size, are more than 50% of the computational time in the architectures defined above. Therefore, there is a trade off between the number of convolutional layers used and computation time. The number of layers was chosen such that running the program takes less than 12hrs.

1.4 MCTS

The general MCTS algorithm, outlined in **??**, was implemented for the inverted pendulum as in algorithms 4 and 5 .

1.4.1 Tree Nodes and Simulations

The state for the inverted pendulum is continuous ($\boldsymbol{x} \in \mathbb{R}^4$), as opposed to the discrete nature of board games. In order to be able to compare nodes, the states must be stored as unique integers. Each MCTS simulation ends with a leaf node being expanded, therefore after S simulations from a node, S states will be visited. Over the whole episode, $S \times L$ distinct states will be visited in total. Thus, the probability of revisiting a state from a different trajectory is $(S \times L)b^4$, where b is the number of bins for each dimension. Given that $L = 400$ and, if $S > 30$, the maximum recursion depth of the MCTS will be reached - limiting the maximum value of S. Multiplying the states by 10^{12} and converting to integers ensures a vanishingly small of encountering a state twice.

1.4.2 Action Selection

Action selection moving down the tree was modified to reflect the asymmetry of the state-value:

[1] www.pytorch.org

$$u^*_{player} = \underset{u \in \mathcal{U}_{player}}{\operatorname{argmax}} \left\{ Q(\boldsymbol{x}, u) + c \cdot \boldsymbol{p}_\theta(\boldsymbol{x}, u) \frac{\sqrt{\sum_b N(\boldsymbol{x}, b)}}{1 + N(\boldsymbol{x}, b)} \right\} = \underset{u \in \mathcal{U}_{player}}{\operatorname{argmax}} \; U_{player}(\boldsymbol{x}, u) \quad (1.12)$$

$$u^*_{adv} = \underset{u \in \mathcal{U}_{adv}}{\operatorname{argmax}} \left\{ -Q(\boldsymbol{x}, u) + c \cdot \boldsymbol{p}_\theta(\boldsymbol{x}, u) \frac{\sqrt{\sum_b N(\boldsymbol{x}, b)}}{1 + N(\boldsymbol{x}, b)} \right\} = \underset{u \in \mathcal{U}_{adv}}{\operatorname{argmax}} \; U_{adv}(\boldsymbol{x}, u) \quad (1.13)$$

Where c = 1 was used. Negating Q in U_{adv} causes the adversary to minimise the state-value whilst still maximising the upper confidence bound, thus ensuring exploration.

When the inverted pendulum exceeds a constraint the tree search should return -1. Note, the simulation terminates after 400 steps (the arbitrarily set "end" of the episode), but this should not return -1. Therefore, when 400 steps is reached, the neural network is evaluated and v is returned.

Algorithm 4 Get Action Probabilities (based on S.Nair's implementation [16])

```
function GETACTIONPROB(x_root, x^(2D), agent)
    for i in nMCTSSims do          ▷ Simulate nMCTSSims episodes from x_root
        resetSimulation(x_root)
        MCTSSearch(x^(2D), agent)
    end for
    N(x_root, u) ←getCounts()     ▷ Count the times each edge, (x_root, u), was visited.
    N(x_root, u) ← N(x_root, u)^τ          ▷ Control Sparsity with the temperature, τ
    return π ← norm(N(x_root, u))
end function
```

1.5 Player and Adversary Evaluation

For problems in which one agent has a distinct advantage, determining how "good" their policy is compared to other policies becomes difficult. For example, if the pendulum stays up for longer, is is difficult to determine whether the player has improved because it is controlling the system better, or the adversary is less effective, or both have improved but one improved more. The score used is the number of steps the agent stayed up for divided by the max number of steps. For the player this ranges between 0 and 1, and for the adversary $score = (200 - steps)/200$ is used.

does this make sense?

Over one policy iteration, there are 4 different policies: the two current player and adversary policies, $f^{(player)}_{\theta,\, curr}$ and $f^{(adv)}_{\theta,\, curr}$, and the challenger policies, $f^{(player)}_{\theta,\, chal}$ and $f^{(adv)}_{\theta,\, chal}$. The improvement of the player can be determined by pitting the current player against the current adversary, and then the challenging player against the current adversary; and vice-versa for the adversary improvement[2]. This gives the four equations in **??**.

[2]Note, if the ratings were correctly updated in the last policy iteration then $\mathbb{E}[\mathbb{E}[s^{agent}_{curr}] - \mu^{agent}_{curr}] = 0$

$$\mathbb{E}[s^p_{curr}] = \frac{1}{1 + 10\hat{}\left(\frac{\mu^a_{curr} - \mu^p_{curr}}{400}\right)} \qquad \mathbb{E}[s^a_{curr}] = \frac{1}{1 + 10\hat{}\left(\frac{\mu^p_{curr} - \mu^a_{curr}}{400}\right)} \tag{1.14}$$

$$\mathbb{E}[s^p_{chal}] = \frac{1}{1 + 10\hat{}\left(\frac{\mu^a_{curr} - \mu^p_{chal}}{400}\right)} \qquad \mathbb{E}[s^a_{chal}] = \frac{1}{1 + 10\hat{}\left(\frac{\mu^p_{curr} - \mu^a_{chal}}{400}\right)} \tag{1.15}$$

$$\tag{1.16}$$

where $\mathbb{E}[s^p_{curr}] = \mathbb{E}[s(f^{(player)}_{\theta,\, curr})]$ denotes the score for the current player, and $\mu^p_{curr} = \mu(f^{(player)}_{\theta,\, curr})$ is the rating of the current player. The expected scores can be estimated via monte-carlo. Rearranging gives the relation between successive agent ratings as:

$$\mu^p_{chal} = \mu^p_{curr} + 400 log_{10}\left(\frac{E[s^p_{chal}]}{1 - E[s^p_{chal}]} \cdot \frac{1 - E[s^p_{curr}]}{E[s^p_{curr}]}\right) \tag{1.17}$$

Algorithm 5 MCTS (based on S.Nair's implementation [16])

function MCTSSEARCH($\boldsymbol{x}^{(2D)}, agent$)
  if $\boldsymbol{x}$ is terminal **then**
    if fallen **then**
      return -1
    else
      $\boldsymbol{\pi}, v \leftarrow f_\theta(\boldsymbol{x}^{(2D)})$
      return v ▷ If the episode end is reached, return v_θ
    end if
  end if

  if $\boldsymbol{x} \notin Tree$ **then** ▷ Expand Leaf Node
    $\boldsymbol{\pi}, v \leftarrow f_\theta(\boldsymbol{x}^{(2D)})$
    $N(\boldsymbol{x}, \cdot) \leftarrow 0$
    $P(\boldsymbol{x}, \cdot) \leftarrow \boldsymbol{\pi}$
    return v
  end if

  if $agent = player$ **then** ▷ Get best action using UCB
    $u^* \leftarrow \underset{u \in \mathcal{U}_{player}}{\operatorname{argmax}} U_{player}(\boldsymbol{x}, u)$
  else
    $u^* \leftarrow \underset{u \in \mathcal{U}_{adv}}{\operatorname{argmax}} U_{adv}(\boldsymbol{x}, u)$
  end if
  $\boldsymbol{x}, c \leftarrow \text{step}(\boldsymbol{x}, u)$ ▷ Take next MCTS simulated step
  $\boldsymbol{x}^{(2D)} \leftarrow \text{getState2D}(\boldsymbol{x}, \boldsymbol{x}^{(2D)})$
  $agent \leftarrow \text{nextAgent}(agent)$
  $v \leftarrow \text{MCTSSearch}(\boldsymbol{x}^{(2D)}, agent)$ ▷ Recursion to next node

  $Q(\boldsymbol{x}, u^*) \leftarrow \frac{N(\boldsymbol{x},u^*)Q(\boldsymbol{x},u^*)+v}{N(\boldsymbol{x},u^*)+1}$ ▷ Backup Q-values up the tree
  $N(\boldsymbol{x}, u^*) \leftarrow N(\boldsymbol{x}, u^*) + 1$
  return v
end function

A Appendices

A.1 Inverted Pendulum Dynamics Derivation

We can find the state space equations for the Inverted Pendulum using d'Alembert forces. Firstly we 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, $\boldsymbol{h}_P = \boldsymbol{r}_{B_1/P} \times m\dot{\boldsymbol{r}}_{B_1}$:

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

We can 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 we can 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 this for our problem by substituting in constants, we can write the full state space equation:

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

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

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

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

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 6 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 6 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 $A^2 \leq 1 \implies \rho(A) \leq 1$.

For the multivariate case, it can similarly be shown (by Gelfand's Theorem, $\rho(M_1...M_n) \leq \rho(M_1)...\rho(M_n)$) that $\rho\left(\bar{\Sigma}_{n+1}C^T(\sigma_v^2 I + C\bar{\Sigma}_{n+1}C^T)^{-1}C)\right) < 1$, therefore a sufficient condition for the decay of the covariance is $\rho(A\Sigma_n A^T) \leq \rho(\Sigma_n) \implies \rho(AA^T) \leq 1$. For the linearised dynamics derived in eq. (A.5) the eigenvalues, λ are given by $\lambda^2(1-\lambda)^2 = 0 \implies \rho(AA^T) \leq 1$. Therefore, the covariance decays to zero in the linearised multivariate case and the 2D state becomes a lossless estimate of the state.

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

$$\sigma_{n+1}^2(\sigma_v^2) = A^2\sigma_n^2(1 - \frac{C^2A^2\sigma_n^2}{\sigma_v^2 + C^2A^2\sigma_n^2}) \tag{A.14}$$

$$\approx \sigma_{n+1}^2(0) + \delta\sigma_v^2\frac{\partial\sigma_{n+1}^2(\sigma_v^2)}{\partial\sigma_v^2}|_{\sigma_v^2=0} \tag{A.15}$$

$$= \frac{\delta\sigma_v^2}{(AC\sigma_n)^2} \tag{A.16}$$

As shown above, the mean square error decays with the square of the bin-size. Similarly, linearising about $\sigma_n^2 \approx 0$ gives $\sigma_{n+1}^2 = A^2$ for the linear case, which is constant.... Need to go to second order...

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