Research Notes

Mustafa Omar[1]
[1] *Physics and Astronomy, University of Nottingham*
(Dated: June 7, 2022)

In this document all that is present are notes taken as the research progressed, the notes are structures in order of progression and are unfiltered raw thoughts and steps. with no respect to grammar.

WEEK 0 & 1: MULTI-AGENT RL BASELINE

- Reading the provided tutorial

Reading the provided tutorial in [1] to obtain an initial idea of problem at hand. The tutorial teaches the basic API of petting-zoo [2] to enable the creation of a customisable environment (a game), this environment is further refined using super-suit [3], which is a wrapper that allows the environment to be further customised and controlled. Finally the tutorial shows how to use a stable-baselinesv3 [4] Cnn model to learn a policy for the given environment.

- Reproducing the tutorial for understanding

I've copied the tutorial [1] verbatim in order to 'reproduce it', this has simply exposed me to the aforementioned APIs, the only question is how would I proceed from here ?, currently the model is training on a CPU alone, utilising only two cores, this is expected to take at least a couple of hours, I have no clear reference of how to speed things up here (unfamiliar with what exactly PPO is doing).

- Reading ch13 of Richard S.sutton

Seems like the only way forward is to read ch13 of the provided book Richard S.sutton [5] , this chapter introduces me to multiagent reinforcement learning, the problem is, once ive learned this chapter how will that knowledge be used to work with tensor networks ?

- Notation

I was given an alert that the notation when researching/(writing the report) may be an issue, so this should be a primary conscious concern from the beginning.

1) notation will match the given notation in [5] (Section:summary of notation), for reinforcement learning. also shown in Appendix A

2) Notation for Tensornetworks will be...

- Coding my tensor network layer

At some point im expecting to have to write my own tensor network in jax [6], at the point of writing this I have no image of how this is going to look or work, and how I'm then going to get it to work with a regular network and for back-propagation to work,

- dig into cnn architecture

The model saved after following the tutorial in [1] is saved as a 'PPO' object, ppo appears to be a wrapper around the CnnPolicy network that manages hyper-parameters (works for many policies of course), using ppo we can obtain the parametrs of the network but I will have to find a way to obtain the actual structure of the CnnPolicy itself, as in layer details. this is possible to reconstruct by observing the structure of the parameters and making a few educated guesses, but to definitely know the exact structure, including activation, pooling and etc, I will have to find that information thought the documentation code. Fastforward, i have found the network architecture, and the assumption that i couldve found the architecture by guessing from the parameters is a horrible one, this is because things like the stride of the CNN, such parameters when accumulated are impassible to 'guess'; this was a naive assumption.

- How the stable base line Cnn policy is structured

There appears to be four parts, the NatureCnn [7], the shared network, the policy network and the value network,

in this case the shared network is empty and the NatureCnn is dubbed the 'feature extractor', the policy and value networks are the agent and critic respectively, in terms of information flow in the code in the tutorial [1] it looks like this:

NatureCnn -) policy

NatureCnn -) value

Where the NatureCnn is shared. In the documentation for PPO, stable baselines3.common.policies.ActorCriticCnnPolicy, takes net arch (network architecture as input), as well as a feature extractor, deeper into the documentation, an MLP extractor is used. Depnding on if the the feature extractor is used, the network architecture is left empty or a hardcoed archetecture is used. both net arch and the feature extractor are then used to build the neural netowrk.

- Building my own nature CNN in jax

The code displayed below is my own version of the NatureCnn architecture displayed in the documnetation of stablebaselines, Haiku infers the input dimention when initialising the network, hence the missing initial variable when compared to the source.

```
self.cnn = nn.Sequential(
        nn.Conv2d(n_input_channels, 32, kernel_size=8, stride=4, padding=0),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=0),
        nn.ReLU(),
        nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=0),
        nn.ReLU(),
        nn.Flatten(),
)
```

FIG. 1: NatureCnn architecture stable baseline

```
def feedforward(x):
  x = hk.Conv2D(
      32, 8,
      stride=4,
      padding='VALID'...
  )(x)
  x = jax.nn.relu(x)
  x = hk.Conv2D(
      64, 4,
      stride=2,
      padding='VALID', ...
  )(x)
  x = jax.nn.relu(x)
  x = hk.Conv2D(
      64, 3,
      stride=2,
      padding='VALID', ...
  )(x)
  x = jax.nn.relu(x)
  x = hk.Flatten(preserve_dims=0)(x)
  return x
```

FIG. 2: Haiku NatureCnn [7]

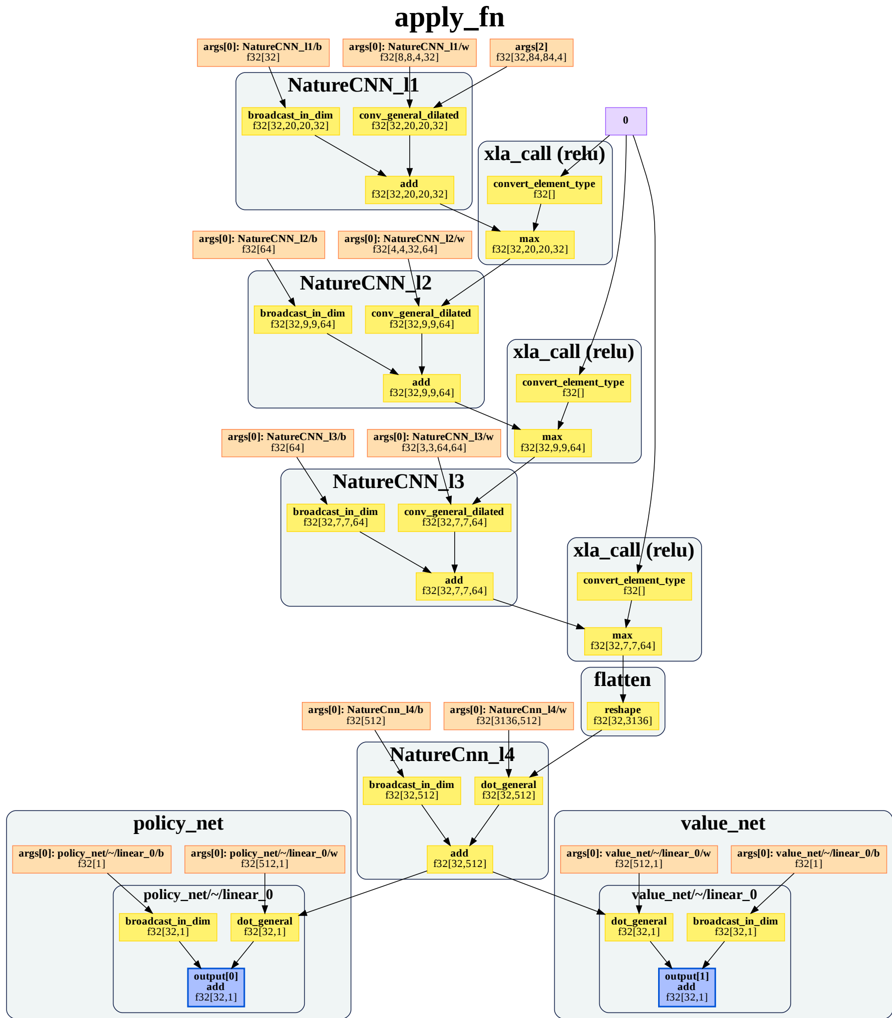

FIG. 3: Haiku full actor critic model dot graph

- copy weights from trained model into my own architecture

The next task is to take the parameters of the model generated by the PPO algorithm and use those within my own architecture. In the parameters of the PPO model, there appears to be a 'log std' layer (first layer), this is a scalar value and i have no idea what it is. apart from that, the network I created matches the strucutre of the PPo model exactly, the only thing left is to reformat the parameters so they're transferable. (what will i do with the 'log std' scaler value ?). Now I have successfully transferred the parameters from a PPO model to my own Haiku model and successfully run a forward pass.

- Confession

So far the 'trained parameters' aren't really trained (training takes a few hours + im lazy), they're initial PPO model params, and i have not yet successfully observed the 'untrained' PPO model to be running on the environment (difficulty with colab visualisations) i have the code for everything but too impatient to wait, this is a step that was skippable until I have a clear idea of the whole process. After completing the next step i will proceed by training the PPO model, then taking those params and using them on my haiku architecture before moving on with the research.

WEEK 2: VALIDATION, SETUP COMPLETION AND RL METHOD EXPLORATION

- Github for code

I begin by moving the code base away from colab, and would resort to only training models on the it. Furthermore I proceed to adapt my code such that it follows the test first paradigm, where unit tests are created to test functionality before the different functionalities are used together to obtain an output. The code is open sourced and can be viewed here https://github.com/Mo379/TensorNets

- validate jax model transfer play game, check game

The next step is to check the validity of the model that utilises the transferred parameters by comparing the ouputs of the PPO model with the haiku model, immediately two conditions of the test created for this task fail, firstly, the output of the saved PPO model gives a value of none for one of the models (actor or critic, not sure which), and secondly the outputs do not match. Ive discovered two possible reasons for this, firstly I was missing a preprocessing step namely pixel normalisation such that $pixel_{value} \in [0.0, 1.0]$ this was easily added at the top of the code in FIG 2, secondly, the log std (distribution) needs to be added at the correct place, the following figure (FIG 4) shows the first issue solved and the second issue partially solved because I'm uncertain about where the log std is supposed to be applied (Mislead by the jax.treemap(... x.shape...) function, as the scan is not ordered {facepalm} in addition to the pytorch ordered dict for parameters operating in a backwards manner), and the exact mathematical implementation.

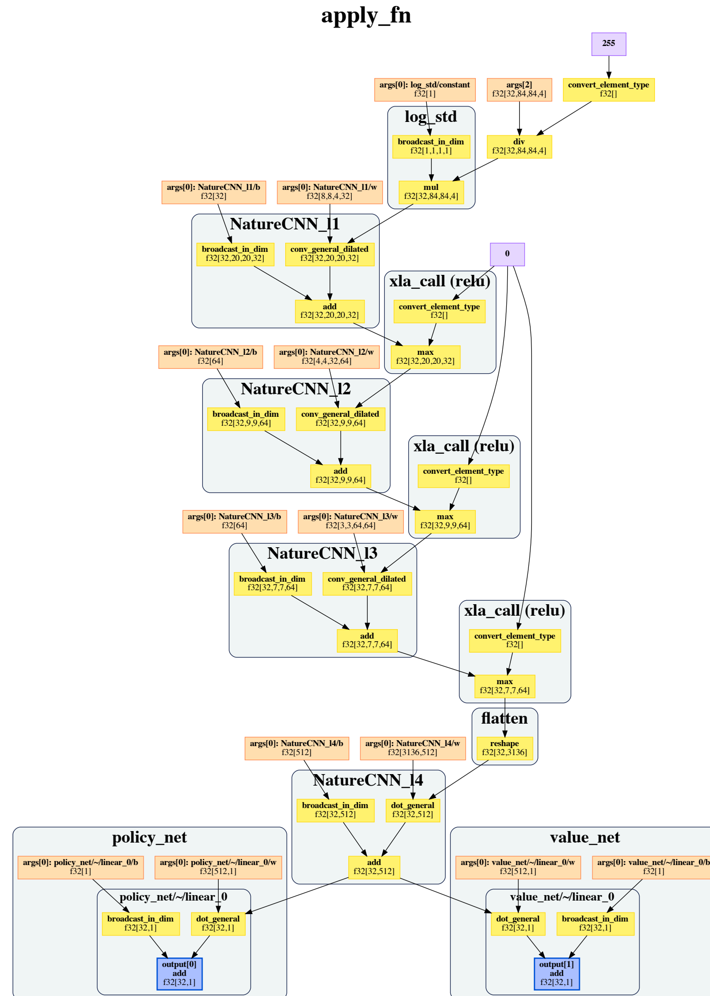

FIG. 4: Haiku full actor critic model dot graph updated week2

- figureing out log std

The actions of the agent have to be translated into a discrete distribution instead of using the continus value outputted by the policy network, therefore stablebaselines applied a distribution that disretises the output of the policy network, hence the position where the log std layed is being applied in FIG 4 is incorrect, the following figure (FIG) shows the correct solution. The only thing now (Hopefully), is to figure out the implementation of the distribution and then to test again to see if the model transfer is valid.

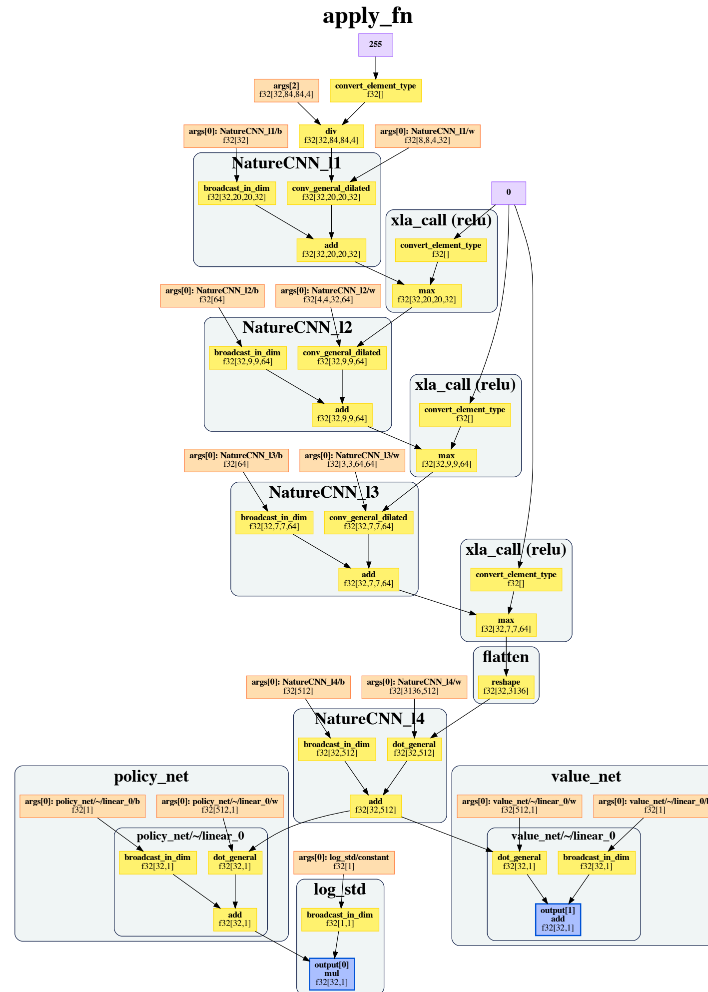

FIG. 5: Haiku full actor critic model dot graph updated week2 2

- Unforseen Issue

The PPO model, when loaded twice on the same scrip and run on the same data ouputs different results, leading to the concolusion that the PPO model infact uses a stochastic policy (Confirmed to be true, there is a deterministic flag that can be set and this problem is resolved). Therefore i cannot expect an output that is consistent and i cannot expect to be able to confirm that the model is successfully transferred without fully training it (If the transfer is successful). Even though the

- run model on game without training

The task now is to deploy the haiku model on the environment and see it run, im expecting this to take some time but to be easy at the same time.

- Train the PPO model params
- Find Jax RL algorithms
- Create a supervised learning methodology to train your own network

[1] J. Terry, Multi-agent deep reinforcement learning in 13 lines of code using pettingzoo, (2022).
[2] J. K. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sulivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. L. Williams, Y. Lokesh, R. Sullivan, and P. Ravi, Pettingzoo: Gym for multi-agent reinforcement learning, arXiv preprint arXiv:2009.14471 (2020).
[3] J. K. Terry, B. Black, and A. Hari, Supersuit: Simple microwrappers for reinforcement learning environments (2020), arXiv:2008.08932 [cs.LG].
[4] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, Stable baselines, https://github.com/hill-a/stable-baselines (2018).
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (A Bradford Book, Cambridge, MA, USA, 2018).
[6] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, JAX: composable transformations of Python+NumPy programs (2018).
[7] K. K. S. D. e. a. Mnih, V., Human-level control through deep reinforcement learning, Nature **518**, 529 (2015).

APPENDIX A

$\doteq$	equality relationship that is true by definition
$\approx$	approximately equal
$\propto$	proportional to
$\Pr\{X=x\}$	probability that a random variable X takes on the value x
$X \sim p$	random variable X selected from distribution $p(x) \doteq \Pr\{X=x\}$
$\mathbb{E}[X]$	expectation of a random variable X, i.e., $\mathbb{E}[X] \doteq \sum_x p(x)x$
$\arg\max_a f(a)$	a value of a at which $f(a)$ takes its maximal value
$\ln x$	natural logarithm of x
e^x	the base of the natural logarithm, $e \approx 2.71828$, carried to power x; $e^{\ln x} = x$
$\mathbb{R}$	set of real numbers
$f : \mathcal{X} \to \mathcal{Y}$	function f from elements of set $\mathcal{X}$ to elements of set $\mathcal{Y}$
$\leftarrow$	assignment
$(a, b]$	the real interval between a and b including b but not including a
ε	probability of taking a random action in an ε-greedy policy
α, β	step-size parameters
γ	discount-rate parameter
λ	decay-rate parameter for eligibility traces
$\mathbb{1}_{predicate}$	indicator function ($\mathbb{1}_{predicate} \doteq 1$ if the *predicate* is true, else 0)

FIG. 6: Mathematics and probability

ε	probability of taking a random action in an ε-greedy policy
α, β	step-size parameters
γ	discount-rate parameter
λ	decay-rate parameter for eligibility traces
$\mathbb{1}_{predicate}$	indicator function ($\mathbb{1}_{predicate} \doteq 1$ if the *predicate* is true, else 0)

FIG. 7: Miscellaneous

In a multi-arm bandit problem:

k	number of actions (arms)
t	discrete time step or play number
$q_*(a)$	true value (expected reward) of action a
$Q_t(a)$	estimate at time t of $q_*(a)$
$N_t(a)$	number of times action a has been selected up prior to time t
$H_t(a)$	learned preference for selecting action a at time t
$\pi_t(a)$	probability of selecting action a at time t
$\bar{R}_t$	estimate at time t of the expected reward given π_t

FIG. 8: General

In a Markov Decision Process:

s, s'	states
a	an action
r	a reward
$\mathcal{S}$	set of all nonterminal states
$\mathcal{S}^+$	set of all states, including the terminal state
$\mathcal{A}(s)$	set of all actions available in state s
$\mathcal{R}$	set of all possible rewards, a finite subset of $\mathbb{R}$
$\subset$	subset of; e.g., $\mathcal{R} \subset \mathbb{R}$
$\in$	is an element of; e.g., $s \in \mathcal{S}$, $r \in \mathcal{R}$
$\lvert\mathcal{S}\rvert$	number of elements in set $\mathcal{S}$

FIG. 9: Environment

t	discrete time step
$T, T(t)$	final time step of an episode, or of the episode including time step t
A_t	action at time t
S_t	state at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
R_t	reward at time t, typically due, stochastically, to S_{t-1} and A_{t-1}
π	policy (decision-making rule)
$\pi(s)$	action taken in state s under *deterministic* policy π
$\pi(a\lvert s)$	probability of taking action a in state s under *stochastic* policy π

FIG. 10: Agent

G_t	return following time t
h	horizon, the time step one looks up to in a forward view
$G_{t:t+n}, G_{t:h}$	n-step return from $t+1$ to $t+n$, or to h (discounted and corrected)
$\bar{G}_{t:h}$	flat return (undiscounted and uncorrected) from $t+1$ to h (Section 5.8)
G_t^λ	λ-return (Section 12.1)
$G_{t:h}^\lambda$	truncated, corrected λ-return (Section 12.3)
$G_t^{\lambda s}, G_t^{\lambda a}$	λ-return, corrected by estimated state, or action, values (Section 12.8)

FIG. 11: Return; n-step, float and λ return

$p(s', r \lvert s, a)$	probability of transition to state s' with reward r, from state s and action a
$p(s' \lvert s, a)$	probability of transition to state s', from state s taking action a
$r(s, a)$	expected immediate reward from state s after action a
$r(s, a, s')$	expected immediate reward on transition from s to s' under action a

FIG. 12: Transition probability and expected reward

$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy

FIG. 13: State Value

V, V_t	array estimates of state-value function v_π or v_*
Q, Q_t	array estimates of action-value function q_π or q_*
$\bar{V}_t(s)$	expected approximate action value, e.g., $\bar{V}_t(s) \doteq \sum_a \pi(a\|s)Q_t(s,a)$
U_t	target for estimate at time t

FIG. 14: Array estimates and expected approximates

δ_t	temporal-difference (TD) error at t (a random variable) (Section 6.1)
δ_t^s, δ_t^a	state- and action-specific forms of the TD error (Section 12.9)
n	in n-step methods, n is the number of steps of bootstrapping

FIG. 15: Temporal difference

d	dimensionality—the number of components of $\mathbf{w}$
d'	alternate dimensionality—the number of components of $\boldsymbol{\theta}$
$\mathbf{w}, \mathbf{w}_t$	d-vector of weights underlying an approximate value function
$w_i, w_{t,i}$	ith component of learnable weight vector
$\hat{v}(s,\mathbf{w})$	approximate value of state s given weight vector $\mathbf{w}$
$v_{\mathbf{w}}(s)$	alternate notation for $\hat{v}(s,\mathbf{w})$
$\hat{q}(s,a,\mathbf{w})$	approximate value of state–action pair s, a given weight vector $\mathbf{w}$
$\nabla\hat{v}(s,\mathbf{w})$	column vector of partial derivatives of $\hat{v}(s,\mathbf{w})$ with respect to $\mathbf{w}$
$\nabla\hat{q}(s,a,\mathbf{w})$	column vector of partial derivatives of $\hat{q}(s,a,\mathbf{w})$ with respect to $\mathbf{w}$

FIG. 16: The weight vector

$\mathbf{x}(s)$	vector of features visible when in state s
$\mathbf{x}(s,a)$	vector of features visible when in state s taking action a
$x_i(s), x_i(s,a)$	ith component of vector $\mathbf{x}(s)$ or $\mathbf{x}(s,a)$
$\mathbf{x}_t$	shorthand for $\mathbf{x}(S_t)$ or $\mathbf{x}(S_t, A_t)$
$\mathbf{w}^\top\mathbf{x}$	inner product of vectors, $\mathbf{w}^\top\mathbf{x} \doteq \sum_i w_i x_i$; e.g., $\hat{v}(s,\mathbf{w}) \doteq \mathbf{w}^\top\mathbf{x}(s)$
$\mathbf{v}, \mathbf{v}_t$	secondary d-vector of weights, used to learn $\mathbf{w}$ (Chapter 11)
$\mathbf{z}_t$	d-vector of eligibility traces at time t (Chapter 12)

FIG. 17: Visible feature vectors

$\boldsymbol{\theta}, \boldsymbol{\theta}_t$	parameter vector of target policy (Chapter 13)
$\pi(a\|s,\boldsymbol{\theta})$	probability of taking action a in state s given parameter vector $\boldsymbol{\theta}$
$\pi_{\boldsymbol{\theta}}$	policy corresponding to parameter $\boldsymbol{\theta}$
$\nabla\pi(a\|s,\boldsymbol{\theta})$	column vector of partial derivatives of $\pi(a\|s,\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$
$J(\boldsymbol{\theta})$	performance measure for the policy $\pi_{\boldsymbol{\theta}}$
$\nabla J(\boldsymbol{\theta})$	column vector of partial derivatives of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$
$h(s,a,\boldsymbol{\theta})$	preference for selecting action a in state s based on $\boldsymbol{\theta}$

FIG. 18: Parameter vector: policy loss action preference

$b(a\|s)$	behavior policy used to select actions while learning about target policy π
$b(s)$	a baseline function $b : \mathcal{S} \mapsto \mathbb{R}$ for policy-gradient methods
b	branching factor for an MDP or search tree
$\rho_{t:h}$	importance sampling ratio for time t through time h (Section 5.5)
ρ_t	importance sampling ratio for time t alone, $\rho_t \doteq \rho_{t:t}$
$r(\pi)$	average reward (reward rate) for policy π (Section 10.3)
$\bar{R}_t$	estimate of $r(\pi)$ at time t

FIG. 19: Behaviour policy importance sampling

$\mu(s)$	on-policy distribution over states (Section 9.2)
$\boldsymbol{\mu}$	$\|\mathcal{S}\|$-vector of the $\mu(s)$ for all $s \in \mathcal{S}$
$\\|v\\|_\mu^2$	μ-weighted squared norm of value function v, i.e., $\\|v\\|_\mu^2 \doteq \sum_{s\in\mathcal{S}} \mu(s)v(s)^2$
$\eta(s)$	expected number of visits to state s per episode (page 199)
Π	projection operator for value functions (page 268)
B_π	Bellman operator for value functions (Section 11.4)

FIG. 20: Miscellaneous 2

$\mathbf{A}$	$d \times d$ matrix $\mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t\big(\mathbf{x}_t - \gamma\mathbf{x}_{t+1}\big)^\top\right]$
$\mathbf{b}$	d-dimensional vector $\mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t]$
$\mathbf{w}_{\mathrm{TD}}$	TD fixed point $\mathbf{w}_{\mathrm{TD}} \doteq \mathbf{A}^{-1}\mathbf{b}$ (a d-vector, Section 9.4)
$\mathbf{I}$	identity matrix
$\mathbf{P}$	$\|\mathcal{S}\| \times \|\mathcal{S}\|$ matrix of state-transition probabilities under π
$\mathbf{D}$	$\|\mathcal{S}\| \times \|\mathcal{S}\|$ diagonal matrix with $\boldsymbol{\mu}$ on its diagonal
$\mathbf{X}$	$\|\mathcal{S}\| \times d$ matrix with the $\mathbf{x}(s)$ as its rows

FIG. 21: Special matrices

$\bar{\delta}_{\mathbf{w}}(s)$	Bellman error (expected TD error) for $v_{\mathbf{w}}$ at state s (Section 11.4)
$\bar{\delta}_{\mathbf{w}}$, BE	Bellman error vector, with components $\bar{\delta}_{\mathbf{w}}(s)$
$\overline{\mathrm{VE}}(\mathbf{w})$	mean square value error $\overline{\mathrm{VE}}(\mathbf{w}) \doteq \\|v_{\mathbf{w}} - v_\pi\\|_\mu^2$ (Section 9.2)
$\overline{\mathrm{BE}}(\mathbf{w})$	mean square Bellman error $\overline{\mathrm{BE}}(\mathbf{w}) \doteq \big\\|\bar{\delta}_{\mathbf{w}}\big\\|_\mu^2$
$\overline{\mathrm{PBE}}(\mathbf{w})$	mean square projected Bellman error $\overline{\mathrm{PBE}}(\mathbf{w}) \doteq \big\\|\Pi\bar{\delta}_{\mathbf{w}}\big\\|_\mu^2$
$\overline{\mathrm{TDE}}(\mathbf{w})$	mean square temporal-difference error $\overline{\mathrm{TDE}}(\mathbf{w}) \doteq \mathbb{E}_b\big[\rho_t\delta_t^2\big]$ (Section 11.5)
$\overline{\mathrm{RE}}(\mathbf{w})$	mean square return error (Section 11.6)

FIG. 22: Bellman error