RL HW 2:
James Wilsench - s1666320

March 26, 2017

1 Actor-Critic Architecture

The actor-critic learning architecture (AC) separates the actual carrying out of the policy (the role of the actor) from that of the reward evaluation in the update step (the role of the critic) [3]. The critic evaluates the actions of the actor at time t by comparing the reward that the actor achieves when in a given state s_t (factoring in the expected value of the resulting state $V(s_{t+1})$) from an action a_t to the value that the critic has learnt to expect when transitioning from s_t, $V(s_t)$. The result of this comparison, δ_t is given by:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{1}$$

where r_{t+1} ia the reward from state s_{t+1} and $0 < \gamma < 1$ is a future discount factor.

The actor then updates its policy based by increasing its preference for action a_t when in s_t if (s_t, a_t) exceeded the critic's expectations (i.e. $\delta > 0$) and decrease this preference if it disappointed the critic (i.e. $\delta < 0$). In the simplest formulation action preference could be defined by a scalar $p(s, a) \in \mathbf{R}$, while the probability of selecting an action, the policy $\pi(s, a)$, could be given by a softmax function so that:

$$\pi(s, a) = \frac{e^{p(s_t, a_t)}}{\sum\limits_{a'_t \in A_t} p(s_t, a'_t)}$$

where A_t is the set of all possible actions from s_t. Updating the preference $p(s, a)$ is then in-line with updating the expected value according to δ_t. Note

that this is just an informative example whilst the key feature of AC is that the δ_t signal from the critic informs the actor's policy update in some way.

$$p(s,a) \leftarrow p(s,a) + \beta\delta_t \qquad V(s,a) \leftarrow V(s,a) + \eta\delta_t$$

where $\beta, \eta > 0$ are learning rates.

1.1 Application of AC: Dopamine

In a reinforcement learning model of the effects of dopamine (a neurotransmitter) on behaviour Montague et al present an AC approach in an attempt to explain so-called *incentive salience* [2]. Incentive salience, is the ability of animals to link value (e.g. a food source) with an appropriate plan of action to obtain it (a policy).

Incentive salience was first identified in animal experiments where use of dopamine receptor blockers led to previously observed actions for reward (food) acquisition to not be pursued [1]. Test subjects would however, consume food reward as expected if given it directly. In the reinforcement learning view of animal behaviour this implies a disconnect between action policy and perceived value that can be explained using the separation of actor from critic. In this interpretation dopamine receptor suppression could be said to interfere with communication between the separate neurological actor and critic mechanisms in the pre-frontal cortex of the brain.

1.2 Comparison with SARSA

Whether directly as in the case of SARSA or indirectly as in the case of AC (through the critic), both methods depend on the current expected value when making policy updates and are thus online learning methods. The difference being that in SARSA policy is directly determined by whichever action maximises $Q_t(s_t, a_t)$ (the state-action value function analogous to $V(s_t)$ in AC), whereas in AC only the updates depend on $V(s_t)$ and so policy can be initialised or otherwise formulated in any way as long as there are reasonable policy parameters to update.

2 RL with Function Approximation

In this section we explore approximation of the state-action value function $Q_t(s,a)$ by a linear combination of features $\phi^i_{s,a}$ with coefficients θ^i_t so that:

$$Q_t(s,a) = \theta^{\mathbf{T}}_{\mathbf{t}} \phi_{\mathbf{s},\mathbf{a}} = \sum_{i=1}^{N} \theta^i_t \phi^i_{s,a} \tag{2}$$

A gradient-based off-policy TD update method is used throughout this section to update the feature weights vector $\theta_{\mathbf{t}}$.

$$\theta_{\mathbf{t+1}} = \theta_{\mathbf{t}} + \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right) \nabla_{\theta_{\mathbf{t}}} Q_t(s_t, a_t) \tag{3}$$

where $\alpha > 0$ is the learning rate and $0 < \gamma < 1$ is a future discount parameter.

2.1 Special Case: Tabular State Space

In the formulation in HW1 there was in effect one parameter unique to that particular state-action combination, (s,a) being updated at a time. This can be encoded as a (large) collection of binary valued features $\phi^{(s,a)}$ where:

$$\phi^{(s,a)}(s',a') = \delta_{(s,a)}(s',a') = \begin{cases} 1 & \text{if } (s',a') = (s,a) \\ 0 & \text{otherwise} \end{cases}$$

The components of the parameter vector θ_t will then be updated with the following step.

$$\begin{aligned} \Delta\theta^{(s',a')}_{t+1} &= \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right) \delta_{(s,a)}(s',a') \\ &= \begin{cases} \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \right) & \text{if } (s',a') = (s,a) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Thus the approximation function would only be updated one parameter at a time, the parameter corresponding to the specific state-action pair.

$$Q_t(s,a) = \sum_{a' \in A} \sum_{s' \in S} \delta_{(s,a)}(s',a') \theta^{(s',a')}_t = \theta^{(s,a)}_t$$

where $A = [0,1,2,3]$ and S is the set of representations of the Enduro grid. Thus θ_t is simply Q_t from the HW1 which updates according to a Q-learning rule as in HW1.

2.2 Function Approximation Agent

We designed a linear function approximation agent based on the linear Q value approximation process described in Eq. (2) and (3) and include a description of its parameters and performance below.

2.3 Feature Description & Model Justification

Feat.	Val.	Ind.	Act.	Desc.
a_bias	Binary	: 4	: 4	Action bias
dist	Sigmoidal	4 : 6	1 : 3	Car distance centre
not_dist	Sigmoidal	6 : 8	1 : 3	Car distance centre
infront	Binary	8 : 10	[0, 3]	Cars in front
not_infront	Binary	10 : 12	[0, 3]	Cars not in front
inleft	Binary	12 : 14	1 : 3	Cars to left
not_inleft	Binary	14 : 16	1 : 3	Cars not to left
inright	Binary	16 : 18	1 : 3	Cars to right
not_inright	Binary	18 : 20	1 : 3	Cars not to right
speed0	Binary	20 : 22	[0, 3]	Speed greater than 0
not_speed0	Binary	22 : 24	[0, 3]	Speed less than 0
collider	Binary	24 : 26	[0, 3]	Has collided
not_collider	Binary	26 : 28	[0, 3]	Has not collided

Table 1: Features used in linear approximation agent with possible values, indices in features list, actions modulated (Python indexing) and description.

Features were arrived at via a process of testing and elimination. Table 1 gives an overview of the features included in the final model. Most of these features modulate a particular subset of actions through updates the θ_t weights.

Most features in Table 1 have an associated reverse feature which is active specifically when the other is not (*not_*). This is done to allow the agent to actively respond and learn in both cases. Most of the features can be clearly understood from the Table and code, however, the distance and collision features require some pre-processing. There are also action biased features which are always active for specific actions, modulation of their weights therefore informs the base level to which an action is favoured in the absence of other information and hopefully encourages increased speed and mobility.

The straight forward manoeuvring features are $inleft$ and $inright$ as well

as there associated negations. These features are active for left and right actions and are provided to give information to the agent about nearby cars to be avoided (in columns adjacent to the agent) to encourage dodging of these cars when possible. Similarly, the $infront$ and $not\_infront$ feature encourages either breaking or accelerating (tied to these actions) in cases where any or no cars are directly in front of the agent. These features were the first tested and make direct use of the same grid used in HW1 by looking forward in the grid up to a specified row number - $self.depth$.. Products of the manoeuvring features were tested but were found not to give significant advantages.

The distance of the car agent from the centre of the road is encoded in the $dist$ and not_dist features using a logistic transformation of the pixel-based displacement of the car agent to the road's centre. The use of the logistic transformation puts the distance on the same scale as the other binary features, preventing distance from becoming disproportionately influential. This was favoured over a direct binary approach to allow some room for manoeuvring. $dist$ and not_dist $(1 - \sigma(dist))$ reverse the positive direction along the x-axis so that the agent can respond to both extremes (with the feature tied to both left and right actions). It is thought that presented with this distance information, the agent will be encouraged to remain near the centre.

The collision features are based on the collision detection system coded into the agent, making use of the $agent.collision$ function. It is designed to encourage the agent to recover its speed while avoiding further collisions after one has occurred (using the break and accelerate actions). This is similar to the speed features $speed0$ and not_speed0 which are thought to also encourage speeds to be higher (by weighting towards acceleration), whilst avoiding breaking when possible (by weighting it less).

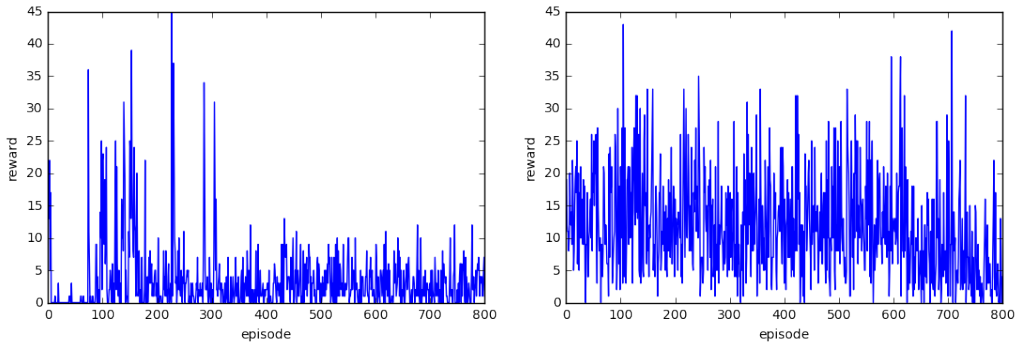
3 Model Analysis

We used the example code provided for HW1 as a comparative baseline to assess the linear approximation model. The initial Q_0 and θ_0 values (referred to as weights) were initialised uniformly with the exception of the accelerate action which was slightly up-weighted initially to decrease training time. The θ_0 values were initialised lower in an attempt to reduce the problem of exploding gradients with $\theta_0 = 1/N$ (N the number of features). Because of unstable convergence in the linear agent we reduced the learning rate to 1e-4 for both models and allowed both systems to train for 500 episodes. In

order that the two models are judged fairly we also changed the exploration method for the Q-learning agent to ϵ-greedy.

3.1 Comparative Learning Performance

Figure 1 shows the learning curve for the linear and Q-learning agents. In both cases most of the direct improvements in performance occur within the first 100 episodes with occurrence being particularly poor for the linear agent in the first 100. This could be due to the time taken to fine tune the weights. In comparison, most of the performance improvements of the Q-learning agent occur in just the first 100 episodes.

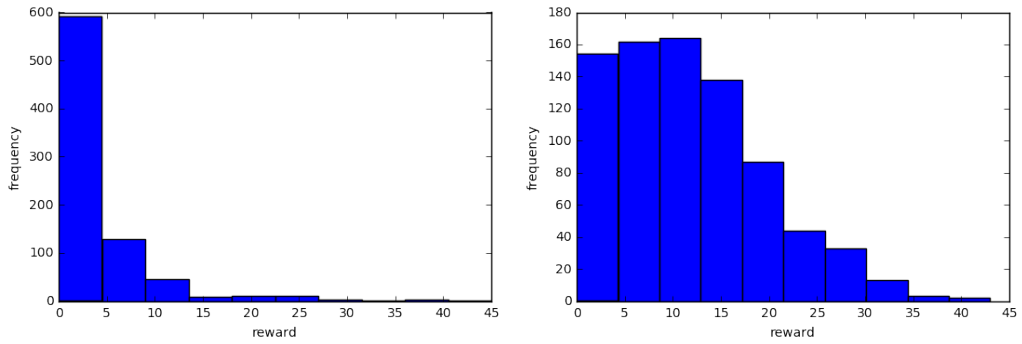


(a) (b)

Figure 1: Learning curves for the linear (left) and Q-learning (right) agents.

After the initial training run, the Q-learning model appears to attain better mean performance with fewer zero and low reward episodes. This could be due to the inflexibility of the linear agent to deal with states that require a complex representation with interacting components, since the model as is only models each factor independently with no interactions.

Figure 2 shows the distribution of rewards while training. This illustrates that rewards for the linear agent are very often zero whilst the Q-learning agent is more likely to attain a 5 or more, even whilst training. This shows possible gaps in the linear model that could be filled by choosing better features and examining the interactions between them. However, it does make sense that given enough training time the Q-learning agent should out-perform most models using only simple features.



(a) (b)

Figure 2: Distribution of rewards while learning for the linear (left) and Q-learning (right) agents over 800 episodes of training.

3.2 Feature Analysis

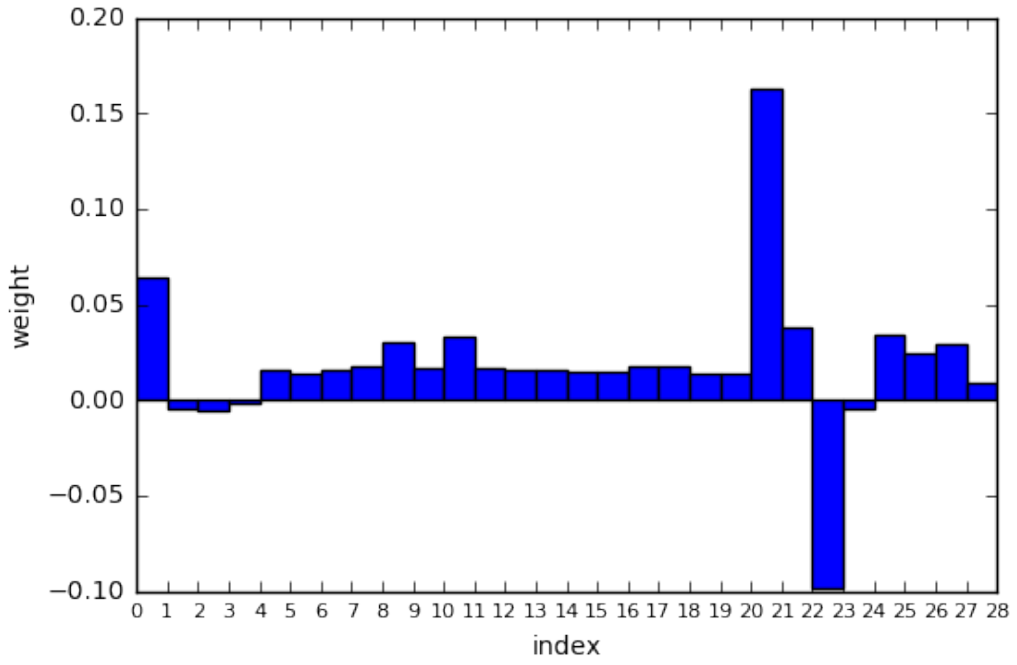


Figure 3: Bar plot of final feature weights after 800 episodes of training for the linear approximation agent (indexed according to Table 1).

Figure 3 shows the final θ weights associated with the 28 features different features (for indexing see Table 1). The bar plot shows that the two most in-

fluential features (20 and 22) are paired and both effect acceleration through speed, but in opposing cases and have oppositely signed weights. This may demonstrate that the agent performs best when speed is maintained at a moderate level or simply that after a collision (when speed is reduced), it is important to change course to avoid further collisions. Note that in general further acceleration when speed is high (greater than zero) is favoured most of all, pointing towards a positive feedback loop of behaviours that should propel the agent forward in the absence of collisions.

Other features with large weights include acceleration bias (0) which shows that the system has clearly learnt to favour acceleration over other actions (which all have slightly negative biases) given no other information. Also high are the weights for $infront$ and $not_infront$ for the accelerate action (8 and 10). All else, being equal this lead to the system accelerating in both cases, although this may lead to more collisions on average, it could also increase average speed and thus performance. All of the collisions features (24 to 27) show some weighting discrepancies from the mean with acceleration before and after collisions being the most influential, this may help to recover speed directly after a collision. Surprisingly the direction features (4 to 7) only show slight variation from one another, suggesting that they are not as influential in determining whether the agent changes direction as whether there is or is not a car in the right lane (16 and 17).

It is important to note that while a large weight does not necessitate high performance, a negligible weight does suggest that the feature has little influence in the current formulation. Almost all of the car detection features (8 to 19) have negligible weights, suggesting that linear agent has not been successful in leveraging this system to its advantage in the final model.

3.3 Model Convergence

We considered the mean of the absolute value of the time-differenced parameters to get an estimate of the first-order rate of convergence.

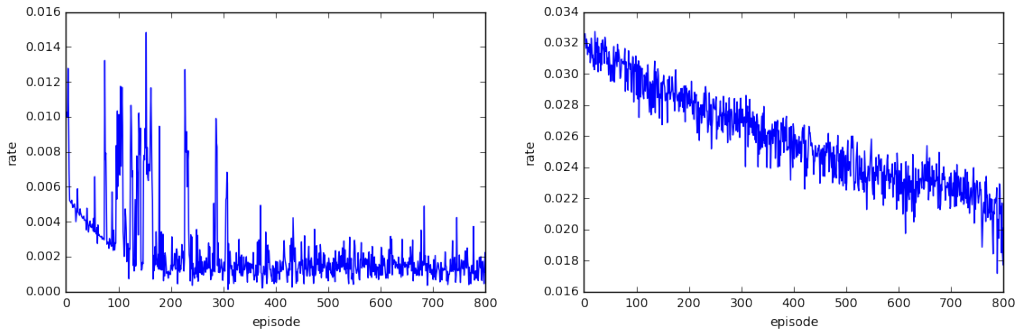
$$\hat{\theta}_t = \frac{1}{N} \sum_{n=1}^{N} |\theta_{t+1}^{(n)} - \theta_t^{(n)}| \tag{4}$$

A similar estimate $\hat{Q}_t$ was constructed for the Q-learning case using $Q_t(s, a)$ and $Q_{t+1}(s, a)$.

As can be seen in Figure 4 the linear agent certainly converges quicker but more noisily as the differences between successive θ_t values becomes stable

around 200 episodes. This is not true for the Q-learning agent which still shows some change in the gradient estimate after even 800 episodes and only slows down after 600.

This observation makes sense when we consider that the Q-learning agent has



(a) (b)

Figure 4: Distribution of rewards while learning for the linear (left) and Q-learning (right) agents over 800 episodes of training.

many more features (when one considers an analogy to the tabular case described above) to find appropriate weights for $(4 \times 10 \times 11 = 440 > 28)$. This implies greater training time as some of these states are bound to be rarer or less influential than others and so training necessarily becomes slow. This suggests a trade-off between final model performance and potential training time made when choosing between approximate and exact learning strategies.

References

[1] Satoshi Ikemoto and Jaak Panksepp. Dissociations between appetitive and consummatory responses by pharmacological manipulations of reward-relevant brain regions. *Behavioral neuroscience*, 110(2):331, 1996.

[2] P Read Montague, Steven E Hyman, and Jonathan D Cohen. Computational roles for dopamine in behavioural control. *Nature*, 431(7010):760–767, 2004.

[3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.