Reinforcement Learning
Assignment 2: REINFORCE and basic Actor-Critic Methods

Daniël Zee (s2063131) [1] **Simon Klaver (s1140760)** [1]

Abstract

In this paper we discuss the theory behind policy-based deep reinforcement learning methods, and experiment on our own implementations of the REINFORCE, Actor-Critic and Advantage Actor-Critic algorithm on the CartPole-v1 environment. Comparing to our results from DQN, we have observed that REINFORCE and Actor-Critic provide similar learning results after hyperparameter optimisation. Advantage Actor-Critic showed drastically improved learning stability using a small bootstrapping depth for the state value estimation, ultimately converging to the optimal policy.

1. Introduction

In the first assignment we focused on the deep reinforcement learning algorithm DQN. This algorithm took the tabular Q-Learning algorithm as its starting point and introduced a neural network to approximate the state-action values $Q_\theta(s, a) \approx Q(s, a)$ for every state s and action a in an environment specified as an Markov decision process (MDP). This approach allowed the algorithm to learn the approximate state-value function for continuous state spaces. DQN is therefore categorised as a value-based method, as the policy is implicitly derived over the best state-action value in a state.

There are however also many environments imaginable where both the state space and the action space are continuous, meaning that we cannot simply compare the state-action values for every action in a given state. In these cases, we need to find the policy directly, which we again represent by a parametrised neural network $\pi_\theta(a|s)$. This network then directly gives us the probability of taking action a in state s for the policy. Using experience collected in the environment, the agent then updates the policy network using gradient learning, optimising the policy towards higher episode returns in an effort to more closely resemble the optimal policy π^*. These methods are therefore called policy-based methods.

For this assignment, we have studied and implemented three policy-based deep reinforcement learning algorithms: REINFORCE (Williams, 1992), Actor-Critic (AC) and Advantage Actor-Critic (A2C). The performance of all three algorithms was compared on the CartPole-v1 environment by Gymnasium (Towers et al., 2024) and an ablation study was performed on the hyperparameters specific to these algorithms.

2. Theory

The goal of policy-based reinforcement learning methods is to ultimately converge to the optimal policy $\pi_\theta \approx \pi^*$. To measure the quality of a policy, we define the quality of a policy by the expected state value over all possible starting states s_0, which equals to the expected discounted return over all episodes τ sampled from the policy:

$$\begin{aligned} J(\pi_\theta) &= \mathbb{E}_{s\sim s_0}[V^\pi(s)] \\ &= \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \end{aligned}$$

As we wish to maximise this value, we need to find the gradient of the objective function with respect to θ, and perform gradient ascent to maximise its value. The problem with the current objective function however is that we cannot estimate its gradient, as we cannot sample over an expected value. We therefore apply the so called "Log-derivative trick", which allows us to rewrite the probability of sampling an episode to the sum of log probabilities of taking action a_t in state s_t for every time step t in our episode. We have now changed the function to use a summation, for which we can easily compute its gradients by summing the gradients of each individual log probability. The full derivation of the policy gradient goes as follows, and makes use of the

[1]Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands. Correspondence to: Daniël Zee <d.f.zee@umail.leidenuniv.nl>, Simon Klaver <s1140760@vuw.leidenuniv.nl>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025.

fact that $\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)$, following the chain rule:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \\
&= \mathbb{E}_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)R(\tau)] \\
&= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t, s_t)R(\tau)\right]
\end{aligned}$$

This objective function, and its gradient notation, form the loss function for the policy networks in policy-based methods. In practice however, we usually take the negative of these functions. This allows is to perform gradient descent instead of ascent, which is more common in training neural networks.

2.1. REINFORCE

The most simple algorithm for optimizing the objective function we will discuss is REINFORCE, introduced by Williams in 1992 (Williams, 1992). REINFORCE is also referred to as Monte Carlo policy gradient, as it estimates the state values via Monte Carlo estimation over a single episode at a time. The discounted return for every state-action pair in the episode are then used to calculate the policy loss at every time step. The original REINFORCE algorithm then updates the policy network parameters using an online approach, where the gradient ascent update is performed at every time step individually. Because parameter updates can be expensive, we have chosen an episodic version of REINFORCE, where the loss function is defined as the sum over the policy losses at every time step. A single parameter update then takes place after a finished training episode. Algorithm 1 shows the pseudocode for episodic REINFORCE.

Algorithm 1 REINFORCE (episodic)

Input: A differentiable policy $\pi_\theta(a|s)$, learning rate $\alpha \in \mathbb{R}^+$, discount factor $0 < \gamma \leq 1$
Initialization: Initialize parameters θ
repeat
  Generate full episode $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a|s)$
  for $t \in 0, \ldots, T-1$ **do**
    $R_t = \sum_{k=t}^{T-1} \gamma^{k-1} r_k$
  end for
  $L(\theta) = \sum_{t=0}^{T-1} -R_t \log \pi_\theta(a_t|s_t)$
  $\theta \leftarrow \theta + \alpha \nabla L(\theta)$
until $\nabla J(\theta)$ converges

2.2. Actor-Critic Methods

Relying on Monte Carlo estimates for the state value estimations is a high variance approach with low sample efficiency. Actor-Critic Methods aim to address these shortcomings by combining value-based function estimation with the policy-based method. Instead of estimating the state values by the discounted return R_t in every episode step t, we now construct a state-action value estimate $\hat{Q}_n(s_t, a_t)$ based on an n-step bootstrapping target. This is similar to the temporal difference (TD) leaning we used for DQN, with the big difference being that we now have to use the Expected SARSA update rule instead of the Q-learning, as we are interested in the on-policy state values for our policy network. The 1-step TD error therefore looks like this:

$$\begin{aligned}
\delta_t &= r + \gamma \mathbb{E}_{a\sim\pi_\theta(a|s_{t+1})}[Q(s_{t+1}, a)] - Q(s_t, a_t) \\
&= r + \gamma V^{\pi_\theta}(s_{t+1}) - Q(s_t, a_t) \\
&= r + \gamma V_\psi(s_{t+1}) - V_\psi(s_t, a_t) \\
&= \hat{Q}_1(s_t, a_t) - V_\psi(s_t, a_t)
\end{aligned}$$

We can construct $\hat{Q}_n$ using the immediate reward combined with state value of a bootstrapped next state. We therefore implement a parametrized value network V_ϕ to bootstrap these values, with the intention of reducing the variance in our state-action value estimations. The current value of $Q(s_t, a_t)$ is then also estimated from our value network. $\hat{Q}_n$ then replaces R_t in the policy loss function, and the squared TD error forms the loss function for the value network. To further reduce variance, we can also increase the number of sampled episodes between updates of the policy and networks. Algorithm 2 shows the pseudocode for the Actor-Critic method.

This approach is called Actor-Critic because we now have two independent neural networks: the actor (policy network), which tries to choose the actions resulting in the highest returns, and the critic (value network), which informs the actor how good the actions are under the current policy. This loop of evaluating the state values over a given policy and using its results to further improve the policy shows similarities with policy iteration in tabular RL.

2.3. Advantage Actor-Critic

Another method to reduce variance in training the policy network is by using baseline subtraction. Our current $\hat{Q}_n(s_t, a_t)$ estimates give us a measure of the expected re-

Algorithm 2 Actor-Critic with bootstrapping

1: **Input**: A differentiable policy $\pi_\theta(a|s)$, learning rate $\alpha \in \mathbb{R}^+$, discount rate $0 < \gamma \leq 1$, update episodes $M \in \mathbb{N}^+$, estimation depth $n \in \mathbb{N}^+$
2: **Initialization**: Initialize parameters θ and ϕ
3: **repeat**
4: **for** episode = 1 to M **do**
5: Generate full episode $\tau = \{s_0, a_0, r_0, s_1, \ldots, s_T\}$ following $\pi_\theta(a|s)$
6: **for** $t \in 0, \ldots, T-1$ **do**
7: $\hat{Q}_n(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$
8: **end for**
9: **end for**
10: $L(\phi) = \frac{1}{M} \sum_{t=0}^{T-1} (Q_n(\hat{s}_t, a_t) - V_\phi(s_t))^2$
11: $\phi \leftarrow \phi + \alpha \nabla L(\phi)$
12: $L(\theta) = \frac{1}{M} \sum_{t=0}^{T-1} \hat{Q}_n(s_t, a_t) \log \pi_\theta(a_t|s_t)$
13: $\theta \leftarrow \theta + \alpha \nabla L(\theta)$
14: **until** $\nabla J(\theta)$ converges

turn of taking action a in state s, but it does not tell us if another action might have resulted in a higher return. If for example $\hat{Q}_n(s_t, 0) = 100$ and $\hat{Q}_n(s_t, 1) = 200$, we would like the policy network increase the probability if selecting action 1 compared to 0. But in our current policy loss function both these observations would result in an increase of their probabilities, but with different magnitudes, as they are both positive. Through baseline subtraction we can create negative values, indicating that actions should be discouraged if they are on average worse that other actions. A simple application of baseline extraction in REINFORCE would be to normalize the discounted returns at each episode step by subtracting the mean and dividing by the standard deviation. A more sophisticated baseline can be used in Actor-Critic methods however, using the so-called advantage function:

$$\hat{A}_n(s_t, a_t) = \hat{Q}_n(s_t, a_t) - V_\psi(s_t)$$

This function gives us an estimation of how much better a given action a is in state s compared to the value function in that state, the expectation of all possible actions. We see that this is the same function as the TD error used in our value network loss. This makes sense, as the advantage function will give larger absolute values during the early stages of training, when our value network has not been trained on many observations and does not accurately approximate the expectation over all actions. Moving the value network towards minimising $\hat{A}_n(s_t, a_t)^2$ does therefore not converge to 0, but in the limit gives us the real advantage values given by a perfect value network.

Actor-Critic methods which use advantages for baseline reduction are called Advantage Actor-Critic. To implement the baseline reduction using the advantage function in the Actor-Critic method described in algorithm 2, we only need to replace $Q_n(s_t, a_t)$ with $A_n(s_t, a_t)$ in line 12, resulting in the following loss function for the policy network:

$$L(\theta) = \frac{1}{M} \sum_{t=0}^{T-1} \hat{A}_n(s_t, a_t) \log \pi_\theta(a_t|s_t)$$

3. Experiments

We have performed experiments using implementations of all three described algorithms. Agents were run for a total of 10^6 environment steps per algorithm. After every 1000 environment steps, we evaluated the current iteration of the policy network by calculating the episode return in a fresh environment, greedily selecting the action with the highest probability for every state. Each configuration has been run for 5 repetitions, over which the average evaluation returns and confidence intervals were plotted. Both the average evaluation returns and confidence intervals have been smoothened using a Savitzky-Golay filter with window size 31. All experiments were performed with a learning rate $\alpha = 0.001$ using the Adam optimizer (Kingma & Ba, 2014), and a discount factor $\gamma = 1.0$.

3.1. Environment

All experiments were performed on the CartPole-v1 environment (Towers et al., 2024). This environment simulates a classic control problem where a pole is attached to a cart, which moves along a frictionless track. The goal is to balance the pole by applying forces in the left and right direction on the cart. The action space is binary and corresponds to actions for pushing the cart either to the left or to the right. The state space consists of 4 continuous values: the cart position, the cart velocity, the pole angle, and the pole angular velocity. A default reward of $+1$ is given for every step taken, with an episode time limit of 500 steps. An episode is terminated if either the pole angle is greater than $\pm 12°$, the cart position is greater than ± 2.4, or the episode time limit is reached.

3.2. REINFORCE

First of all, we will implement the REINFORCE algorithm as has been explained in Section 2.1. Here we have made the choice to not only implement the algorithm as is, but to also include an additional experiment: what if we would normalize the return values? As is known, REINFORCE has a high variance, meaning sometimes the returns are very high, and other times they can be very low. This holds especially true in the Cartpole-v1 environment, where the reward for each step taken is equal to 1. This means that the longer

an episode is the higher the reward becomes: an episode of 10 steps will have return value 10, and an episode of 400 steps will have a return value of 400. This is of course only talking about the returns of the environment: when implementing REINFORCE a discount factor gamma will be applied to these returns, as is shown in Algorithm 1.

In theory, normalizing the return values will allow for a more stable learning process: large differences in return values may cause great differences in gradient magnitudes during backpropagation between different episodes. The normalization will instead keep the gradients in a more consistent range, and as such prevents overshooting or vanishing updates.

In practice, as is shown in Figure 1, the normalization does help the learning process. As we can see in the graph, the REINFORCE implementation with normalization has a learning curve with an almost consistently higher reward at every given environment step than the REINFORCE implementation without normalization.

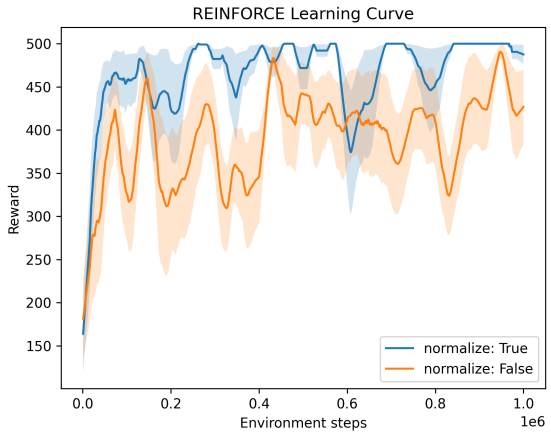


Figure 1. Comparison between REINFORCE with and without episode return normalization on the greedy evaluation performance for the agent.

3.3. Actor-Critic Method

For the implementation of the Actor-Critic Method, as has been explained in Section 2.2, our implementation is exactly as the pseudocode in Algorithm 2. Compared to REINFORCE, this algorithm introduces two new hyperparameters: the interval per which amount of episodes to update the value and policy networks called update_episodes, and the number of steps to use for multi-step bootstrapping called estim_depth.

A small study over the estim_depth hyperparameters can be seen in Figure 2. The estim_depth of 500 is included as a guideline: as the length of each episode is at most 500 long, an estim_depth of equal length prevents the bootstrapping of any next states, causing the algorithm to perform Monte-Carlo for every state. Combine this with an update_episodes value of 1 and it would effectively become REINFORCE instead of Actor-Critic. Therefore, while the curve seems very promising, it is not a suitable value when using Actor-Critic in the Cartpole-V1 environment. A more suitable value shown is an estim_depth of 50, which does allow for bootstrapping and also has a nice learning curve.

In Figure 3 a comparison between different values for the update_episodes can be seen. While the value of 50 is clearly worse than the smaller values, these smaller values all are very comparable in their learning curves. Therefore, we can draw the conclusion that while larger values should be avoided, smaller values are mostly indistinguishable and can be adapted to accommodate calculation times.

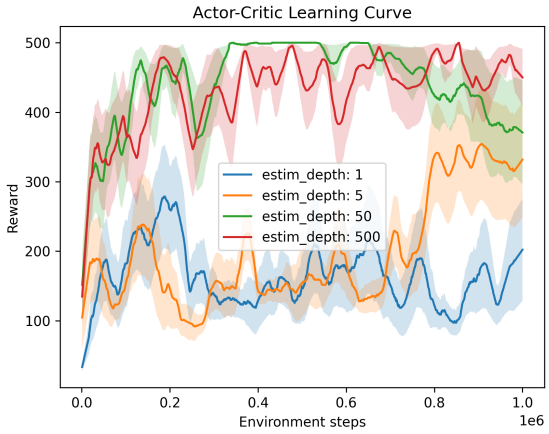


Figure 2. Comparison between different values of estim_depth on the greedy evaluation performance for the Actor-Critic agent.

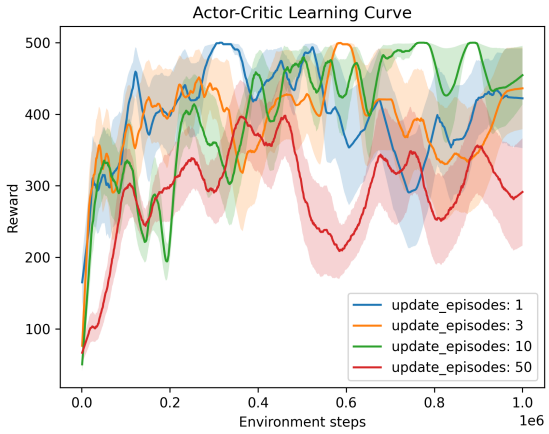


Figure 3. Comparison between different values of update_episodes on the greedy evaluation performance for the Actor-Critic agent.

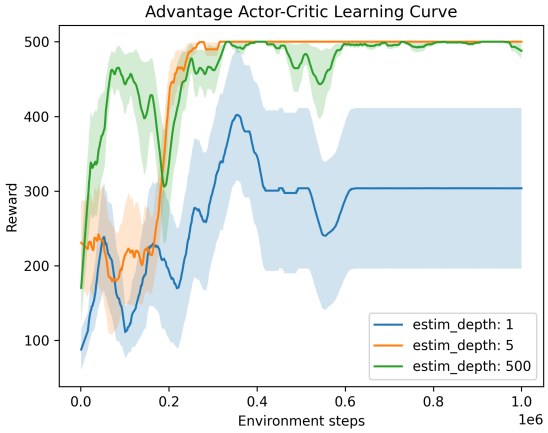
3.4. Advantage Actor-Critic

For the implementation of the Advantage Actor-Critic Method, as has been described in Section 2.3, just like the minimal change required to the pseudocode in Algorithm 2 there is only an if-statement more in the Python code. For this algorithm we did an experiment to see how much the update_episodes value would affect the learning curves, especially with regard to the estim_depth. The intuition here is that a higher update_episodes value does increase the amount of episodes run, and therefore the amount of environment steps taken, before each update. Therefore, we may see a slower learning process, and subsequently a lower learning curve.

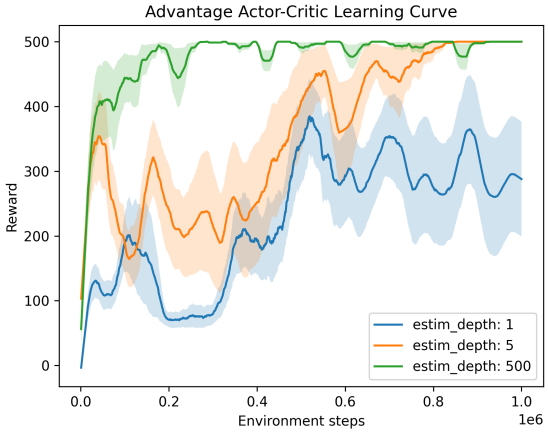
As can be seen in Figure 4, this intuition proves to be correct: while the curves for an estim_depth of 1 and 500 are seemingly unaffected by the value of update_episodes, the curve for the estim_depth of 5 clearly shows that the learning process is much slower: instead of reaching the top reward at around 200000 environment steps, it instead takes a longer climb and reaches the top reward only at around 800000 environment steps.

The reason why the values of 1 and 500 for estim_depth are unaffected by the value for update_episodes may be explained by their nature: the value of 500 causes the algorithm to perform Monte-Carlo on each state as was explained in Section 3.3, meaning it has a high variance but low bias. As it is not influenced by the value network at all, the effects of updating the value network are also non-existent.

The value of 1 on the other hand causes the algorithm to only evaluate the immediate reward and the value calculated for the next state, causing a high bias but low variance. And as the rate of updating the value network is directly relying on the value of update_episodes, this means the value of the next state will be constant for every episode in the interval, effectively diminishing the effect of update_episodes on the algorithm when the value of estim_depth is 1. The value of 5 for estim_depth instead is somewhere in between those in both bias and variance: it both sums more rewards and is affected by the value network, causing the update_episodes value to affect the algorithm in this case.



(a) update_episodes = 1



(b) update_episodes = 3

Figure 4. Comparison of values 1 and 3 for update_episodes against multiple values for estim_depth on the greedy evaluation performance for the Advantage Actor-Critic agent.

4. Discussion

Our results show that while the REINFORCE algorithm is able to periodically achieve policies that evaluate to the optimal episode return, the learning is still quite unstable. When the policy breaks down we see that in time it is able to recover again, but it can still break down again, showing no signs of convergence.

Introducing the value network in Actor-Critic aims to improve the learning stability by reducing the variance inherent to Monte Carlo estimates. We have however seen that this can result in a high level of bias in our estimated TD targets when using a low estimation depth during bootstrapping. A higher estimation depth helps, but never achieves better results than REINFORCE during our experiments. Increasing the episodes between updates also does not improve learning.

Advantage Actor-Critic shows more promising results. While performance using an estimation depth of 1 stays poor, the algorithm is significantly more performant at higher estimation depth values compared to basic Actor-Critic. At an estimation depth as low as 5 we have observed stable convergence to the optimal policy. We have also seen that with an estimation depth of 500, which results in the TD target becoming the Monte Carlo return, the use of the advantage baseline subtraction still results in more stable learning compared to REINFORCE.

Figure 5 compares the best performing configurations for all policy-based methods tested, together with the best results for DQN from assignment 1. We observe that while A2C is the slowest to learn in the early stages, it outclasses every other algorithm on this environment. It is also interesting to note that REINFORCE performs very similarly to DQN. This is noteworthy as REINFORCE is a simple algorithm with little hyperparameters compared to DQN, which required extensive augmentations like Experience Replay and Target Network in order to achieve acceptable performance. It is however important to note that these results do not generalise to all other RL environments, as there is no RL algorithm which always performs best.

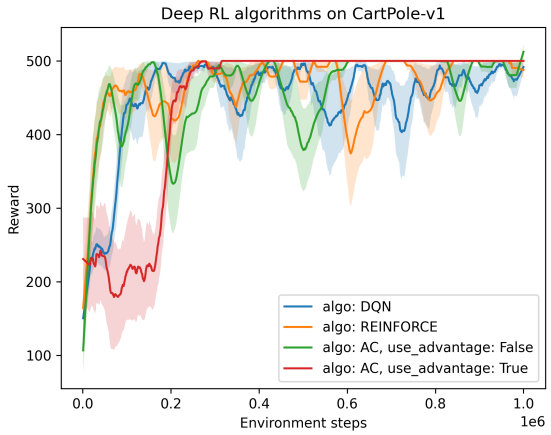


Figure 5. Comparison of the best found learning curves on CartPole-v1 for DQN, REINFORCE, AC and A2C.

5. Conclusion

For this assignment, we have studied the theory behind policy-based deep reinforcement learning methods and implemented the REINFORCE, Actor-Critic and Advantage Actor-Critic algorithms. During our experiments on the CartPole-v1 environment, we have observed the practical benefits of directly approximating the policy, allowing us to perform on-policy learning without having to design our own exploration strategies. This simplicity does however come at the cost of high learning variance due to the use of Monte Carlo estimates, which we have successfully reduced using a value network used for bootstrapping state values, together with baseline subtraction using the advantage function.

Actor Critic algorithms are a family of RL algorithms, meaning that our implementations are not the only way they can be designed. Future work could therefore include testing different Actor-Critic designs on CartPole-v1. We could for example define separate learning rates for the policy and value network, as there is an argument to be made that the value network should learn more quickly to better adapt to the changes made in the policy network. Another change could be to move to an online learning approach, where the policy and value networks are periodically updated during the episodes, which is more in line with DQN.

References

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.