ADAPTIVE GRADIENT METHODS FOR DEEP Q REINFORCEMENT LEARNING

Steven Laverty, Muhammad Khan, Adrian Gillerman

ABSTRACT

In this paper we explore the performance of popular Adaptive Gradient methods (ADAM, RMSProp, Rectified ADAM, Adamax) utilised in training the Deep Q Neural network architechure implmented on the standard cartpole environment. Further information to be added as we proceed with the results.

1 REINFORCEMENT LEARNING - BASICS

1.1 REINFORCEMENT LEARNING COMPARED WITH TRADITIONAL ML APPROACHES

Reinforcement learning can be seen as the science of interacting with the environment and finding optimal ways to make decisions (you, 2015b). When compared with the more traditional approaches in machine learning e.g. supervised or unsupervised learning, reinforcement learning differs from the former in the sense that rather than learning from available data at hand it involves learning from experience; undertaking actions and adapting future behaviour based off on the consequent rewards or punishments (Khandelwal, 2022) . In the case of reinforcement learning one can not tell what is the best thing to do as is the case with the aforementioned traditional approaches. Another manner in which RL differs from its counterparts is the fact that feedback in the formers case could be delayed and may not be instanteneous as is the case with the latter- one can reap rewards in the future for an action which might not seem to yield much benefit shortly after it was undertaken(Morales & Escalante, 2022). Furthermore, wherin the standard assumption, in the case of supervised and unsupervised learning, entails treating each observation as having been drawn from an independent and identical distribution reinforcement learning involves dealing with a dynamic system wherein the aim is to move around the environment and processes information at each timestep with the information at time t+1 being closely correlated with that at time t (François-Lavet et al., 2018).

1.2 INTEGRAL COMPONENTS OF REINFORCEMENT LEARNING

The critical components of RL involve two entities: the agent and the environment. The agent is the entity that processes observations from the enviroment and performs an action in response to the observation. The environment is the entity that provides the agent with the observation and the impact of the action that the agent takes in terms of the reward. Once the action is taken, the agent in turn influences the environment in terms of the consequences of the action (you, 2015b). The main goal of RL is to train a policy to maximise rewards based on previous trail and error experience. Only current state influences the next state.

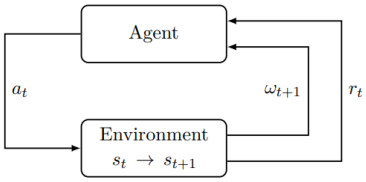


Figure 1: Reinforcement Learning Loop (François-Lavet et al., 2018)

1.3 Markov Decision Process

In general, problems with a finite number of states which have a memoryless property i.e $P(s_{t+1}|S_{1:t}) = P(s_{t+1}|s_t)$ can be modelled by Markov Decision Process (François-Lavet et al., 2018). More generally, a markov decision process (MDP) can be defined as a discrete time stochastic control process with 5 essential components: State space (S), Action space (A), Transition function (probability of observing state s_{t+1} given state s_t and action a_t), the reward function (quantifies the reward corresponding to the action taken by the agent) and the discount factor γ in [0,1) (used for assigning importance to immediate and future rewards)(François-Lavet et al., 2018) .

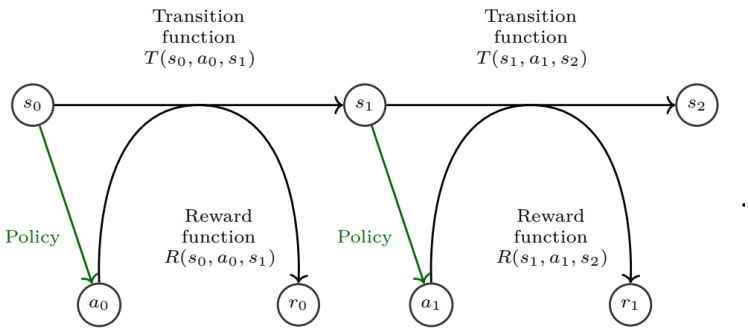


Figure 2: Markov Decision Process At each step, an action taken by the agent corresponds to a change in state of the environment and a consequent reward being generated (François-Lavet et al., 2018)

Another key component of the MDP is the policy which can be interpreted as the basis on which an agent picks an action (agent behaviour function). It can be seen as the way an agent processes an observation from the environment and maps it to the action it needs to undertake (François-Lavet et al., 2018). Policy can thus be defined well in terms of the following map:

$$\pi : State \longrightarrow Action$$

$$s \longrightarrow \pi(s)$$

why is there so much whitespace around all these equations?

There are two types of policies (you, 2015b):

i). Deterministic policy:

$$a = \pi(s)$$

i). Stochastic policy:

$$\pi(a|s) = P(A = a|S = s)$$

where $\pi(a|s)$ denotes the probability of action a being chosen given state s.

An additional component of importance is the value function which can be interpreted as the prediction of the expected future reward; a measure of how good each action/state pair is. The two types of value functions which are frequently considered are:

i). V-Value function

The V-value function denotes the expected reward corresponding to state s at time t and policy π being followed consequently.

$$V^{\pi}(s) : S \longrightarrow R$$

$$V^{\pi}(s) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s, \pi] \text{ (François-Lavet et al., 2018)}$$

ii). Q-value function

The Q-value function denotes the expected reward corresponding to action a being taken in response to state s at time t and policy π being followed consequently.

$$Q^{\pi}(s, a) : SxA \longrightarrow R$$

$$Q^{\pi}(s, a) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s, a_t = a, \pi] \text{(François-Lavet et al., 2018)}$$

The task at hand entails finding a policy so as to maximise the value function (maximise total expected future reward).

$$max_{\pi \in \Pi} V^{\pi}(s) \quad \text{or} \quad max_{\pi \in \Pi} Q^{\pi}(s, a)$$

The relation between the two value functions is as follows:

$$V^{\pi}(s) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} | s_t = s, \pi] = \sum_{a} \pi(a|s) Q^{\pi}(s, a) \text{ (François-Lavet et al., 2018)}$$

1.4 Bellman Equation & Deep Q Learning

what is this? you don't label it

Via the Q-value function, we have:

$$Q^{\pi}(s, a) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k} \, | s_t = s, a_t = a, \pi]$$

$$Q^{\pi}(s, a) = E_{\pi}[r_t + \sum_{k=1}^{\infty} \gamma^{k} r_{t+k} \, | s_t = s, a_t = a, \pi]$$

$$Q^{\pi}(s, a) = E_{\pi}[r_t + \gamma \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \, | s_t = s, a_t = a, \pi]$$

$$Q^{\pi}(s, a) = E_{\pi}[r_t + \gamma \sum_{n=0}^{\infty} \gamma^{n} r_{(t+1)+n} \, | s_t = s, a_t = a, \pi]$$

Now, via the law of iterated expectations, we have:

$$E[E(X|Y, Z)|Y] = E(X|Y)$$

$$E_{\pi}[E_{\pi}(\sum_{n=0}^{\infty} \gamma^{n} r_{(t+1)+n} | s_t = s, a_t = a, \pi, s_{t+1} = s^{'}) | (s_t = s, a_t = a, \pi)] = E_{\pi}(\sum_{n=0}^{\infty} \gamma^{n} r_{(t+1)+n} | s_t = s, a_t = a, \pi)$$

It follows that

$$Q^{\pi}(s,a) = E_{\pi}[r_t + \gamma E_{\pi}(\sum_{n=0}^{\infty} \gamma^n r_{(t+1)+n} | s_t = s, a_t = a, \pi, s_{t+1} = s^{'}) \mid s_t = s, a_t = a, \pi]$$

$$Q^{\pi}(s,a) = E_{\pi}[r_t + \gamma E_{\pi}(\sum_{n=0}^{\infty} \gamma^n r_{(t+1)+n} | s_t = s, a_t = a, \pi, s_{t+1} = s^{'}) \mid s_t = s, a_t = a, \pi]$$

Explain in words how did you get here? What is R? Before it was the reward set. Should it be r(s,a)?

$$Q^{\pi}(s,a) = \sum_{s'} P(s^{'}|s,a) \left[R(s,a,s^{'}) + \gamma Q^{\pi}(s^{'}, \pi(s^{'}))\right]$$

$$Q^{\pi}(s,a) = \sum_{s'} T(s,a,s^{'}) \left[R(s,a,s^{'}) + \gamma Q^{\pi}(s^{'}, \pi(s^{'}))\right]$$ (François-Lavet et al., 2018)

define

The optimal policy is given by:

deterministic

$$\pi^*(s) = argmax_{a \in A} Q^*(s,a)$$

while the corresponding optimal Q-value function is given as follows:

$$Q^*(s,a) = max_{\pi \in \Pi} Q^{\pi}(s,a)$$

$$Q^*(s,a) = max_{\pi \in \Pi} E_{\pi}[r_t + \gamma E_{\pi}(\sum_{n=0}^{\infty} \gamma^n r_{(t+1)+n} | s_t = s, a_t = a, s_{t+1} = s^{'}) \mid s_t = s, a_t = a]$$

$$Q^*(s,a) = E_{\pi^*}[r_t + \gamma E_{\pi^*}(\sum_{n=0}^{\infty} \gamma^n r_{(t+1)+n} | s_t = s, a_t = a, s_{t+1} = s^{'}) \mid s_t = s, a_t = a]$$

$$Q^*(s,a) = E_{\pi^*}[r_t + \gamma max_{a'} Q^*(s^{'}, a^{'}) \mid s_t = s, a_t = a]$$ (François-Lavet et al., 2018)

Learning the Q-value function is an integral part of solving the RL problem, however, this might not be ~~f~~easible in most cases using traditional approaches primarily due to computational inefficiency typically arising from the need to evaluate all possible action-state pair values (Van Hasselt et al., 2016). Thus, one of the ways to cater to this problem is to try and obtain ~~and~~ approximate the Q-value function. One of the approaches to estimate the Q-value function is to use Q-learning. The algorithm in this case involves learning a parameterized Q value function $Q(s_t, a_t; \theta_t)$ with the approximation at the kth iteration updated towards a parameterized target $Y_t^Q = r_t + \gamma max_{a'} Q^*(s^{'}, a^{'}, \theta_t)$ (François-Lavet et al., 2018).

to

what does this mean?

what multiple networks?

Deep Q learning is an approach that utilises a multi layered neural network which churns out action, Q-value pairs for a given input state. For an n dimensional state space and corresponding m dimensional action space the neural network is a map from R^n to R^m (Van Hasselt et al., 2016). Although the structure of the networks is the same, the weights of the target neural network are copied from the online neural network every γ steps. The problem of approximating the Q value function in this case thus amounts to updating the neural network parameters by minimizing the following loss:

what does this mean

$$L_{DQN} = (Y_t^{DQN} - Q(s_t, a_t; \theta_t))^2$$ (Van Hasselt et al., 2016)

where $Y^{DQN} = r_t + \gamma max_{a'} Q^*(s^{'}, a^{'}, \theta_t^-)$

what is this? you don't know this, so how can you form this target?

The update for the neural network parameters is as follows:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^{DQN} - Q(s_t, a_t; \theta_t)) \nabla_{\theta_t} Q(s_t, a_t; \theta_k)$$ (Van Hasselt et al., 2016)

2 ADAPTIVE GRADIENT METHODS

2.1 ADAM

An improvement on the existing framework of utilising plain stochastic gradient descent, ADAM is a optimization algorithm that utilises just the first order gradient information with minimum memory requirement (Kingma & Ba, 2014). The method entails adaptively choosing learning rates for each parameter by making use of the first(mean) and second moments(variance) of the gradient. The method starts off with computing exponentially moving averages of the first and second moments respectively (Kingma & Ba, 2014):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

where $\beta_i \in (0, 1]$ controls the decay rate of the moving averages. The moments are usually initialized as 0's which leads to a potential problem, since for β close to 1 we might not see significant change in the moments. This problem is what is referred to as initialization bias. However, making use of the relation between the expected value of the exponentially averaged i^{th} moment and the true moment we correct the initialization bias problem (Kingma & Ba, 2014) by normalizing the moments as follows:

$$\hat{m}_t = \frac{m_t}{1-\beta_t^1}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_t^2}$$

The final iteration entails:

$$w_{t+1} = w_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

2.2 RMSPROP

Need to add stuff on this algortihm.

3 CARTPOLE ENVIRONMENT

We finally move to the problem at hand. The goal of this paper is to compare the performance of the different adaptive gradient methods for deep Q learning using the cartpole setup. The cartpole environment consists of a pole that moves on a frictionless surface. The system entials the application of a force of +1 or -1, depending on the direction, with the aim to keep the pole in upright position and prevent it from falling over alongside keeping it in the center of the screen. The state space in this case incorporates 4 elements (cart position, cart velocity, pole angle and the velocity at the tip of the pole). The action space involves two elements (moving left or moving right). There are two possible rewards in this scenario (+1 in case the angle the cart makes with the vertical axis is less than 5^o at each timestep). The animation stops in case the angle is greater than 15^0 or the cart moves 2.5 m from the starting position. In addition episodes are truncated after a maximum of 500 steps (Kurban, 2022).

We implemented the deep Q learning algorithm on the Cart-pole environment Barto et al. (1983) using PyTorchPaszke (2013) and Farama Gymnasium. The neural network is a simple feed-forward

network with a configurable hidden dimension and number of hidden layers. Every layer other than the final output layer uses ReLU activation. The current implementation uses a hidden dimension of 256 nodes, and one hidden layer.

To collect training data for the model, we use an epsilon-greedy policyMnih et al. (2013) where with probability ϵ, a random is chosen. Otherwise, the action with the highest Q-value is chosen. ϵ is decreased exponentially from 0.9 to 0.05 with a decay constant of 0.999 per step.

Deep Q learning is implemented using an experience replay bufferMnih et al. (2013) of the past 10,000 state transitions. During training, we uniformly sample a minibatch from this replay buffer. We also train two networks simultaneously: the policy and target networks**?**. The Bellman equation for Q-values becomes:

$$Q_{\text{policy}}(s,a) = R + \gamma(\max_{a'} Q_{\text{target}}(s,a'))$$

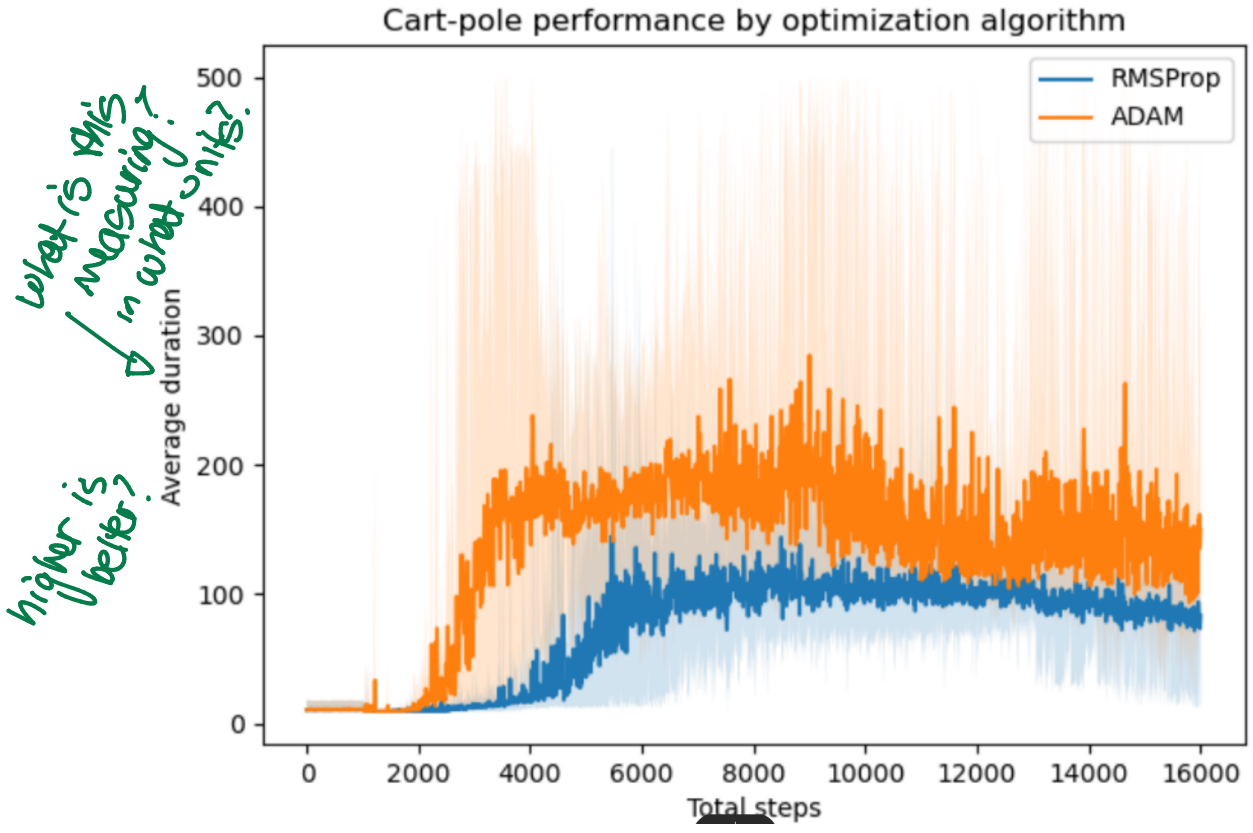
Furthermore, after every training step, our target network is updated with a soft update parameter rule**?**:

$$W_{\text{target}} \leftarrow (1-\tau)W_{\text{target}} + \tau W_{\text{policy}}$$

where $\tau = 0.01$

This helps to stabilize training by keeping the target network from changing too rapidly with the policy network.

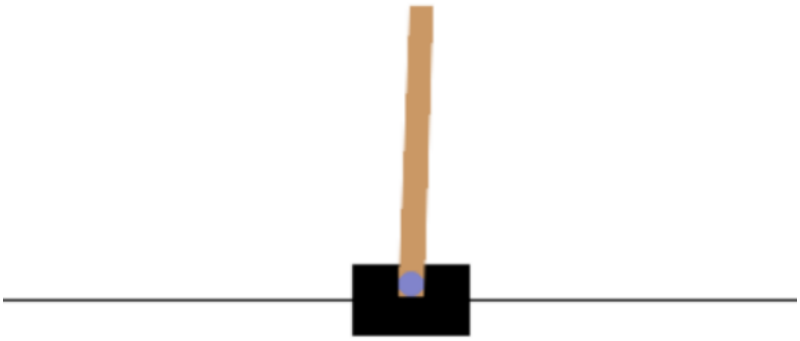
3.1 Preliminary Results



how can we tell from the plot?

In this figure, we see the averaged results of 10 parallel training simulations. For each simulation, the initial model parameters are identical; the only difference is the optimization algorithm used during training. We see that ADAM converges to a local optimum policy faster than RMSProp; possibly due to ADAM's basic model of curvature. Both optimization algorithms fail to reach an optimal policy given the hyperparameters specified above, which may indicate a more powerful model is needed.

As far as the hyperparameters are concerned for ADAM, we used a learning rate of 1e-3, no weight regularization, and beta values of 0.9 and 0.999. For RMSProp, we used a learning rate of 1e-3 and an RMS alpha value of 0.99.



Cartpole-Environment

Github: https://github.com/s-laverty/rl-optim-compare

For future:

Implement couple of different adaptive gradient methods

1. Adamax
2. Rectified adam
3. Adam W
4. Add background for the above adaptive gradient methods
5. Compare performance with different hyperparameters

REFERENCES

Rl course by david silver - lecture 2: Markov decision process, May 2015a. URL https://www.youtube.com.pk/watch?v=lfHX2hHRMVQ&%3Blist=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&%3Bindex=2&gl=PK.

Rl course by david silver - lecture 1: Introduction to reinforcement learning, May 2015b. URL https://www.youtube.com.pk/watch?v=2pWv7GOvuf0&list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&index=1.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018. URL http://arxiv.org/abs/1811.12560.

Iclr. Iclr/master-template: Template and style files for iclr. URL https://github.com/ICLR/Master-Template.

Renu Khandelwal. Supervised, unsupervised, and reinforcement learning, Jul 2022. URL https://arshren.medium.com/supervised-unsupervised-and-reinforcement-learning-245b59709f68.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Rita Kurban. Deep q learning for the cartpole, Dec 2022. URL https://towardsdatascience.com/deep-q-learning-for-the-cartpole-44d761085c2f.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, and et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Eduardo F Morales and Hugo Jair Escalante. A brief introduction to supervised, unsupervised, and reinforcement learning. In *Biosignal Processing and Classification Using Computational Learning and Intelligence*, pp. 111–129. Elsevier, 2022.

Adam Paszke. Reinforcement learning (dqn) tutorial. https://github.com/pytorch/tutorials/blob/main/intermediate_source/reinforcement_q_learning.py, 2013.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Mike Wang. Deep q-learning tutorial: Mindqn, Oct 2021. URL https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.