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, AdamW, RMSProp, Rectified Adam) utilised in training the Deep Q Neural network architechure implmented on the standard cartpole environment. We first provide a brief background on reinforcement learning and introduce the adaptive gradient methods which are to be utilised. A consice introduction to the cartpole environment is followed by a description of the architecture and implementation of the Deep Q network. As for the results we provide empirical evidence for the relatively better performance of the AdamW when compared with the other algorithms and verify the statistical significance of the difference in performance by comparing the means of the episode length using a one factor ANOVA test.

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 (sil, 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, it is not immediately obvious what behavior is desired (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 former's 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, unlike conventional supervised and unsupervised learning, data in reinforcement learning cannot be assumed to be independent and identically distributed. Reinforcement learning involves a dynamic system in which the aim is to move around the environment and process information one step at a time. Thus, data from time step t will be highly correlated with data from time step $t + 1$ (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 (sil, 2015b). The main goal of RL is to train a policy to maximise rewards based on previous trial and error experience. Only current state influences the next state.

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 (François-Lavet et al., 2018) :

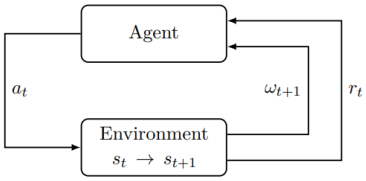


Figure 1: Reinforcement Learning Loop (François-Lavet et al., 2018)

i). State space (S): Set of all possible environment states

ii). Action space (A): Set of all possible actions agent can undertake

iii). Transition function T(s_{t+1}, s_t, a_t) : Probability of observing state s_{t+1} given state s_t and action a_ti.e. $P(s_{t+1}|s_t, a_t)$

iv). Reward function R(s_{t+1}, s_t, a_t): Quantifies the reward corresponding to the action a_t taken by the agent in response to state s_t and consequently observing state s_{t+1}

v). Discount factor γ in [0,1): Used for assigning importance to immediate and future rewards

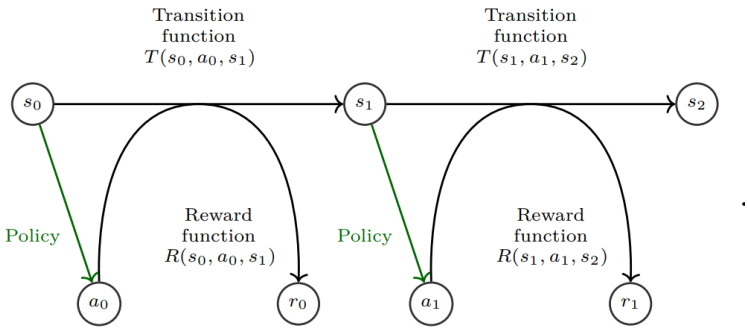


Figure 2: Markov Decision Process At each step, an action taken by the agent corresponds to a change in state of the environment and a consequent reward being generated (François-Lavet et al., 2018)

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:

$$\pi : State \longrightarrow Action$$

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

There are two types of policies (sil, 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.

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

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

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.

$$Q^{\pi}(s, a) : S\text{x}A \longrightarrow R$$

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

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

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

The relation between the two value functions is as follows:

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

1.4 Bellman Equation & Deep Q Learning

Via the Q-value function, we have:

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

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

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

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

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

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

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

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

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

It follows that:

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

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

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

Now

$E(\sum_{n=0}^{\infty} \gamma^n r_{(t+1)+n}|s_t = s, a_t = a, s_{t+1} = s^{'}, \pi)) :=$ Reward corresponding to state $s^{'}$ and performing action $\pi(s^{'})$

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

It follows that:

$$Q^\pi(s,a) = E[r_t + \gamma Q^\pi(s^{'}, \pi(s^{'})) \,|s_t = s, a_t = a, \pi]$$

$$Q^\pi(s,a) = E[r_t|s_t = s, a_t = a, \pi] + E[\gamma Q^\pi(s^{'}, \pi(s^{'}))|s_t = s, a_t = a, \pi]$$

Since $r_t = R(s, a, s^{'})$ it follows that:

$$Q^\pi(s,a) = E[R(s,a,s^{'})|s_t = s, a_t = a, \pi] + E[\gamma Q^\pi(s^{'}, \pi(s^{'}))|s_t = s, a_t = a, \pi]$$

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

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

This leads to the Bellman equation:

$$Q^{\pi}(s,a) = \sum_{s'} \ T(s,a,s^{'}) \ [R(s,a,s^{'}) + \gamma Q^{\pi}(s^{'},\pi(s^{'}))]$$

(François-Lavet et al., 2018)

The optimal Q-value function is given as follows:

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

The corresponding optimal deterministic policy is given by:

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

Now,

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

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

$$Q^{*}(s,a) = E_{\pi^{*}}[r_t + \gamma \underset{a^{'} \in A}{max} Q^{*}(s^{'},a^{'}) \ | s_t = s, a_t = a] \ \text{(François-Lavet et al., 2018)}$$

Learning the Q-value function is an integral part of solving the RL problem, however, this might not be faesible 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). Although one might initially think about utilising Q-learning which involves storing expected rewards corresponding to each state action pairs in a table and updating them iteratively, this approach is not faesible for problems where the state space is infinite (as is the case in our setting- more to be discussed later). For the purposes of our problem, hence, we are unable to use simple Q learning and need to use deep Q learning. 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).

The deep Q network, utilised in deep Q learning, involves training two neural networks with the same architecture simultaneously - the target and policy neural network respectively. Now, via the universal approximation theorem, we can approximate any function upto desired accuracy using a neural network. For the purposes of our problem, we utilise a target neural network to approximate the future discounted rewards ($Q_{target}(s_t, a_t; \theta^{(t)}_{target})$). The policy neural network on the other hand involves learning a parameterized Q value function $Q_{policy}(s_t, a_t; \theta^{(t)}_{policy})$ with the approximation at the kth iteration updated towards the expected future reward i.e. the loss function used in training the policy neural network is:

$$L_{DQN} = f(Y_t^{DQN}, Q_{policy}(s_t, a_t; \theta^{(t)}_{policy})) \text{ (Van Hasselt et al., 2016)}$$

where $Y_t^{DQN} = r_{t+1} + \gamma \max_a Q_{target}(s_{t+1}, a, \theta_{target}^{(t)})$ (Van Hasselt et al., 2016)

Rather than utilising the conventional mean square error, the loss function that we use is the Huber loss function:

$$\implies f(Y_t^{DQN}, Q_{policy}(s_t, a_t; \theta_{policy}^{(t)})) = H(Y_t^{DQN} - Q_{policy}(s_t, a_t; \theta_{policy}^{(t)}))$$

where:

$$H(a) = \begin{cases} \frac{1}{2}a^2 & ||a||_1 \leq \delta \\ \delta||a||_1 - \frac{1}{2}a^2 & ||a||_1 > \delta \end{cases}$$

As can be observed, when the magnitude of the residual is low we compute the regular mean square error while when the residual's magnitude is high we compute the mean absolute error. This loss function allows us to ensure we do not puch too much weight on outliers and helps with overall performance of the model (Seif, 2022).

Since the parameter update step is different for each of the four optimization algorithms we intend to use, it will be discussed in the following section.

Furthermore, After the policy network update, the parameters of the target network are updated according to the soft target rule(Lillicrap et al., 2019):

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

Now, as discussed earlier, training data in the case of RL is in the form of the experience obtained by the agent as it traverses through the environment. This collection of agent's experiences is referred to as experience replay buffer (TORRES.AI, 2021) and in the process of training we sample experiences from this collection in order to aid the learning process.

The pseudocode below highlights the implementation of the Deep Q network architecture (van Hasselt et al., 2015):

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\, Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

2 Adaptive Gradient Methods

2.1 RMSProp

RMSProp, or root mean square propagation, is an unpublished adaptive optimization technique produced by Geoff Hinton in the Coursera course "Neural Network for Machine Learning." (Huang, 2022) This algorithm keeps track of the gradient as well as the second moment to update the relevant

parameters. Moreover, RMSProp extends SGD by using momentum for the gradient and the second moment, to keep track of past values in predicting future values.

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

A good choice for the momentum parameter γ is 0.9, and the learning rate η may be set to 0.001. (Ruder, 2016)

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

When the momentum parameter is chosen well, RMSProp will converge faster than SGD. However, if this parameter is too small, too many steps are required before the algorithm converges to an appropriate solution. On the other hand, if the momentum parameter is too large, oscillations can result, prohibiting the error from reaching a sufficiently low value. The optimization algorithm RMSProp - Root Mean Square Propagation - builds on prior algorithms by adapting the learning rate using the gradient and the second moment.

2.2 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:

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

2.3 ADAMW

An improvement on the existing framework corresponding to the Adam algorithm, AdamW involves decoupling weight decay from the optimization steps taken with respect to the loss function in Adam (Loshchilov & Hutter, 2017). In order to illustrate the difference, let us consider the loss function involving an L_2 regularizer i.e:

$$L = f(x; w) + \frac{\lambda}{2}||w||_2^2$$

Following the standard Adam algorithm, the first moment of the gradient at t^{th} tiimestep is as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)[\nabla_w f_t(x_t; w_{t-1}) + \lambda w_{t-1}]$$

$$\hat{m_t} = \frac{\beta_1 m_{t-1} + (1-\beta_1)[\nabla_w f_t(x_t; w_{t-1}) + \lambda w_{t-1}]}{1-\beta_t^1}$$

The update for the parameters is as follows:

$$w_t = w_{t-1} - \alpha \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon}$$

Let us look at the effective stepsize in greater detail:

$$\alpha \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} = \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)[\nabla_w f_t(x_t; w_{t-1}) + \lambda w_{t-1}]}{(1-\beta_t^1)(\sqrt{\hat{v_t}} + \epsilon)}$$

$$\alpha \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} = \alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)[\nabla_w f_t(x_t; w_{t-1})]}{(1-\beta_t^1)(\sqrt{\hat{v_t}} + \epsilon)} + \frac{\lambda \alpha}{1-\beta_t^1} \frac{w_{t-1}}{(\sqrt{\hat{v_t}} + \epsilon)}$$

Now, $\frac{w_{t-1}}{(\sqrt{\hat{v_t}} + \epsilon)}$ leads to parameters with high magnitude of the gradient being regularized much less compared to ones with low magnitude of the gradient. This imbalance pertaining to regularization leads to a problem especially in the case of vanishing or exploding gradients. In an attempt to counter this issue, AdamW involves decopuling the weight decay from the computation of the gradient and first moment and is consequently introducted in the final step corresponding to the parameter update as follows:

$$w_t = w_{t-1} - \alpha \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} - \lambda w_{t-1} \text{ (Loshchilov \& Hutter, 2017)}$$

The addition of weight decay in the above step ensures that weights are regularized more or less evenly thereby having the potential to improve the performance of the model.

2.4 RECTIFIED ADAM

Rectified Adam (RAdam), was introduced in 2020 and incorporates an expression to amend the variance of the adaptive learning rate. (Wright, 2019) The goal of RAdam is to achieve better convergence than Adam by handling the initially large variance in the beginning of training the model. This problem with Adam results from the reduced number of training samples available at the onset. We include only the relevant part of RAdam in the following equations from the published paper (Liu et al., 2019):

If the variance is tractable, i.e. $\rho_t > 4$, then

$$l_t \leftarrow \sqrt{(1 - \beta_2^t)/v_t} \quad \text{(Compute adaptive learning rate)}$$

$$r_t \leftarrow \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}} \quad \text{(Compute the variance rectification term)}$$

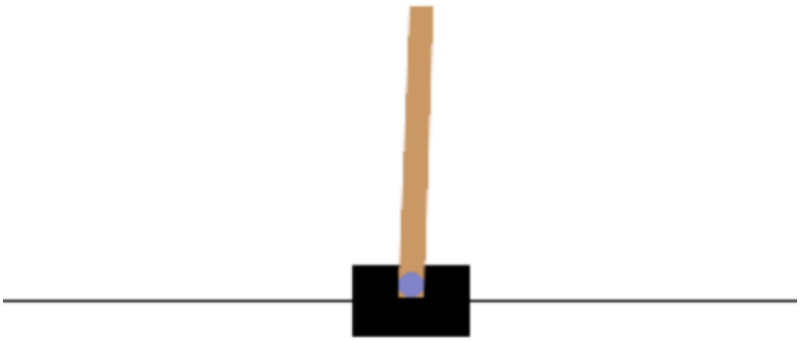
$$\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \hat{m}_t l_t \quad \text{(Update parameters with adaptive momentum)}$$

Else update parameters with un-adapted momentum

Therefore, decreasing the learning rate in the beginning training epochs reduces the variance. In other words, RAdam considers the increased uncertainty at the initial epochs by including a rectifier term that decreases the learning rate and makes the variance consistent. Rectified Adam decreases the variance of the adaptive learning rate by including a rectification term.

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 is one possible reward 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 12^o or the cart moves 2.4 m from the starting position. An episode, which corresponds to the cartpole moving along the track without violating the conditions mentioned above, is terminated after a maximum of 500 steps (Kurban, 2022). Therefore, the maximum possible total reward from a single episode is 500.



Cartpole-Environment

4 EXPERIMENTAL SETUP

The deep Q learning algorithm is implemented in PythonPaszke (2013). The Q-value function is approximated by an artificial neural network implemented in PyTorch with the following hyperparameters:

- 4-dimensional input (cart position, cart velocity, pole angle, pole angular velocity)
- 3 128-dimensional hidden layers, each with ReLU activation
- 2-dimensional output layer with linear activation (Q action-values)

Both the policy network and the target network are initialized with the same random starting weights. We utilize the Farama Gymnasium implementation of the cart-pole environment to train and evaluate the policy network. In order to directly compare the performance of different optimizers, the following training hyperparameters are held constant:

- Q-learning discount factor (γ) is 0.9

- Epsilon greedy training policy with $\epsilon(t) = 0.05 + 0.85(0.999^t)$
- Soft-target network parameter update constant (τ) is 0.01
- Training minibatch size of 128
- Experience replay buffer of the 2500 most recent state transitions
- Learning rate of 1×10^{-4}
- Weight decay (L2) regularization of 1×10^{-2} for all network parameters *except* the output layer bias vector

During training, every state transition is saved to the replay buffer (a python deque) as a 4-tuple of (s_t, a_t, r_t, s_{t+1}). If the next state is terminal, it will be recorded as `None` to indicate no future rewards. After each state transition, the policy network is trained using one minibatch of state transitions with the minibatch being randomly sampled without replacement from the experience replay buffer. This is followed by an update of the parameters of the target neural network. When the environment reaches a terminal state or truncates after 500 steps, it is reset to a randomized initial state and training continues without interruption.

Since deep Q learning is an off-policy reinforcement learning algorithm, the quality of the policy cannot be directly inferred from training episode length. Thus, after each time step, a separate evaluation environment runs 16 full cart-pole episodes from start to finish using the deterministic (greedy) policy determined by the latest policy network iteration. The average total reward determines the quality of the deterministic policy.

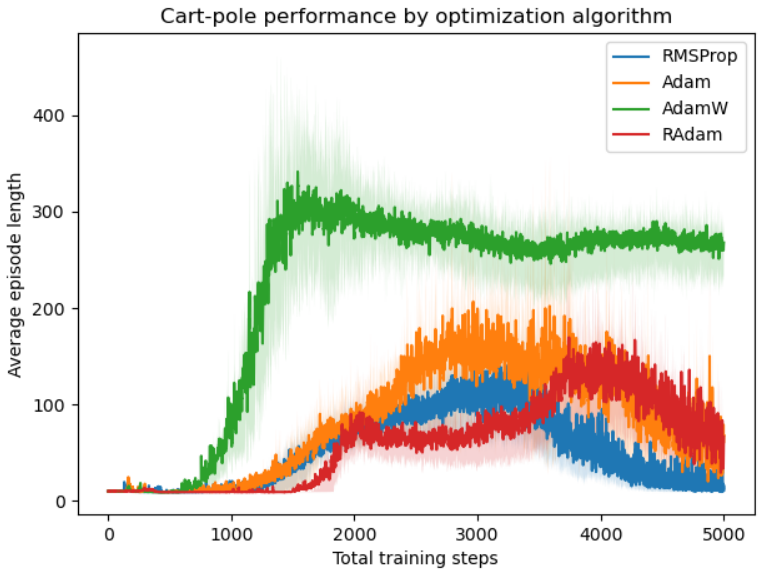
In order to directly compare the performance of the different optimizers, the same randomly-initialized model is trained using each optimizer in turn. The cart-pole environment used for training is re-initialized with the same random seed for every optimizer. The entire training and evaluation procedure is repeated with 16 different randomly-initialized models to ensure a robust comparison of performance between the different optimizers.

For the RMSProp optimizer, the α parameter is set to 0.99.

For Adam, AdamW, and RAdam, the β parameters are set to $\beta_1 = 0.9 \quad \beta_2 = 0.999$.

All hyperparameters were selected by hand-tuning.

5 Results and Observations



In this figure, we see the averaged results of 16 parallel training simulations. For each simulation, the initial model parameters are identical; the only difference is the optimization algorithm used during training. Let us delve a bit deeper into the graph at hand. The total training steps (x-axis) denotes the number of steps for which the algorithm was run. The average episode duration length (y axis) denotes the average duration of the episode length for the 16 simulations. As can be observed, AdamW appears to perform significantly better than the rest of the three adaptive gradient methods. The peak of the graph, for each optimization algorithm, highlights the average episode length corresponding to the optimal policy. As can be seen, significantly less number of training steps are required for AdamW compared to the rest of the three. A possible reason for the better performance of AdamW, as discussed in the section on the algorithm itself, lies in the fact that weight decoupling allows us to regularize the weights in a better manner as compared to other algorithms especially the nearest competitor Adam.

Prior to discussing the potential causes of poor performance of both RMSProp and RAdam, it would be beneficial to introduce the notion of sparse rewards. A sparse reward task is said to occur when a large number of the states in the state space do not produce feedback. This can happen when an agent attempts to achieve an objective, but receives feedback only when the agent is in close proximity to the target (Daaboul, 2020). The poor performance of RAdam can be attributed to the fact that the reward signal in this case might be too sparse. (Nota, 2019) Alternatively, it might be the case that the choice of epsilon might play a role (Nota, 2019). As per (Nota, 2019), a different algorithm not relevant to the context of this paper, A2C , had trouble learning Pong when this parameter was too small 1e-8, but succeeded when it was increased to 1e-3.

In order to test the statistical significance of the performance difference, we ran a one factor Anova test utilising the observations corresponding to the best average episode length for each of the algorithms respectively. The null and alternate hypothesis for Anova are as follows:

H_0: Mean episode duration is same for each optimization algorithm

H_1: Mean episode duration is not same. There exists at least one optimization algorithm whose mean episode length is different from the rest

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RMSProp	16	2370.5625	148.1602	14644.04
Adam	16	3308.6875	206.793	22593.67
Adam W	16	5462.3125	341.3945	13776.55
Radam	16	2713.1875	169.5742	16906.29

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	361043.7129	3	120347.9	7.08757	0.000373	2.758078
Within Groups	1018808.13	60	16980.14			
Total	1379851.843	63				

The test above compares the means of the episode length corresponding to 4 gradient methods. As can be observed, the p value is extremely low and even at 1% significance level we reject the null hypothesis $\implies$ there exists at least one optimization algorithm whose mean episode length is different from the rest.

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
RMSProp	16	2370.563	148.1602	14644.04
Adam	16	3308.688	206.793	22593.67
Radam	16	2713.188	169.5742	16906.29

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	28168.55534	2	14084.28	0.780379	0.464334	3.204317
Within Groups	812159.9368	45	18048			
Total	840328.4921	47				

The test above compares the means of the episode lengths corresponding to 3 gradient method except for AdamW. As can be observed, the p value is high and at 1% significance level we fail to reject the null hypothesis $\implies$ that we are unable to deduce that there exists a difference in mean episode length for the three algorithms.

Anova: Single Factor

SUMMARY

Groups	*Count*	*Sum*	*Average*	*Variance*
Adam W	16	5462.3125	341.3945	13776.55
Adam	16	3308.6875	206.793	22593.67

ANOVA

Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	144940.645	1	144940.6	7.970294	0.008365	4.170877
Within Groups	545553.1909	30	18185.11			
Total	690493.8359	31				

The test above compares the means of the episode length corresponding to AdamW and its nearest competitor. As can be observed, the p value is extremely low and at 1% significance level we fail to we reject the null hypothesis $\implies$ there does exist a statistically significant difference between the performance of Adam and AdamW as we hypothesized.

6 Conclusions and Future Work

In this paper we compared the performance of the four adaptive gradient methods (Adam, AdamW, RMSProp, RAdam) in the cartpole environment setting. We obtained a statistically significant difference in the performance of AdamW and the rest of the three algorithms when gauged through the lens of comparing the means of the episode lengths. As far as future work is concerned, this comparison can be extended to the other simple agent reinforcement learning environments such as pendulum, mountain-car, mujoco (bas).

References

Gymnasium documentation. URL https://gymnasium.farama.org/content/basic_usage/.

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.

Karam Daaboul. Reinforcement learning: Dealing with sparse reward environments, Aug 2020. URL https://medium.com/@m.k.daaboul/dealing-with-sparse-reward-environments-38c0489c844d.

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.

Jason Huang. Rmsprop, Aug 2022. URL https://optimization.cbe.cornell.edu/index.php?title=RMSProp#:~:text=RMSProp%2C%20root%20mean%20square%20propagation,lecture%20six%20by%20Geoff%20Hinton.

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.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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.

Chris Nota. Radam: A new state-of-the-art optimizer for rl?, Aug 2019. URL https://medium.com/autonomous-learning-library/radam-a-new-state-of-the-art-optimizer-for-rl-442c1e830564.

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

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

George Seif. Understanding the 3 most common loss functions for machine learning regression, Feb 2022. URL https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e

Jordi TORRES.AI. Deep q-network (dqn)-ii, May 2021. URL https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

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.

Steve Williams. Rmsprop, Aug 2020. URL https://issuu.com/stevewilliams2104/docs/optimization-algorithms-for-machine-learning/s/10920058.

Less Wright. New state of the art ai optimizer: Rectified adam (radam)., Aug 2019. URL https://lessw.medium.com/new-state-of-the-art-ai-optimizer-rectified-adam-radam-5d854730807b.