Adaptive Gradient Methods for Deep Q Learning

Steven Laverty Muhammad Khan Adrian Gillerman

Rensselaer Polytechnic Institute

ML & Optimization
April 23, 2023

Objectives

- 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
- Provide empirical evidence for the relatively better performance of the AdamW when compared with the other algorithms
- Verify the statistical significance of the difference in performance by comparing the means of the episode length using a one factor ANOVA test

Overview

Traditional ML compared to RL

- Reinforcement learning can be seen as the science of interacting with the environment and finding optimal ways to make decisions (sil, 2015b)

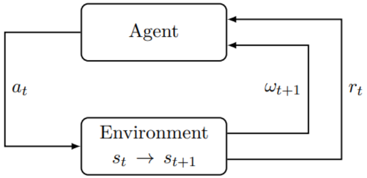
- Feedback in the case of supervised or unsupervised learning is instantaneous whereas it could be delayed in the case of RL

- Traditional ML approaches involve learning from available data at hand however RL involves learning from experience

- Data in reinforcement learning cannot be assumed to be independent and identically distributed as is the case with conventional supervised and unsupervised learning (François-Lavet et al., 2018)

Integral components of RL

- Agent: Entity that processes observations from the enviroment and performs an action in response to the observation
- Environment: 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



(François-Lavet et al., 2018)

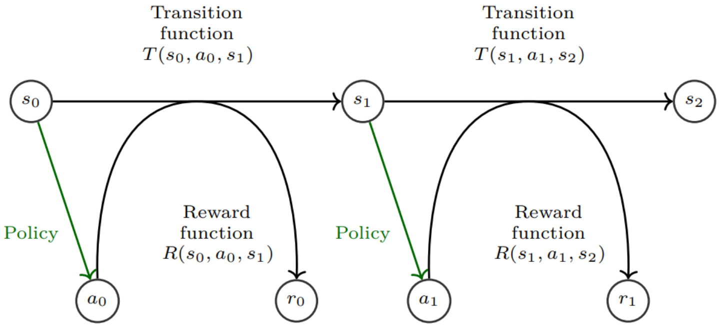
Markov Decision Process

- 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

Markov decision process (MDP) can be defined as a discrete time stochastic control process with 5 essential components (François-Lavet et al., 2018):

- State space (S): Set of all possible environment states
- Action space (A): Set of all possible actions agent can undertake
- 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. $P(s_{t+1}|s_t, a_t)$
- 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}
- Discount factor γ in [0,1): Used for assigning importance to immediate and future rewards

Markov Decision Process (Contd.)


Transition function $T(s_0, a_0, s_1)$
Transition function $T(s_1, a_1, s_2)$
s_0
s_1
s_2
Policy
Reward function $R(s_0, a_0, s_1)$
Policy
Reward function $R(s_1, a_1, s_2)$
a_0
r_0
a_1
r_1

(François-Lavet et al., 2018)

- Policy is the basis on which an agent picks an action (agent behaviour function); 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)

Markov Decision Process (Contd.)

Policy can 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):

- Deterministic policy:

$$a = \pi(s)$$

- Stochastic policy:

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

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

- The main goal of RL is to train a policy to maximise rewards based on previous trial and error experience

Value Function

Value function is 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:

- V-Value function: Denotes the expected reward corresponding to state s at time t and policy π being followed consequently

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

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

- 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

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

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

- The task at hand basically entails finding a policy so as to maximise the value function

Bellman Equation

Via the Q-value function, we have:

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

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

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

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

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

It follows that:

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

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

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

Bellman Equation (Contd.)

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

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 $\pi(s^{'})$

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

Bellman Equation (Contd.)

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)

Bellman Equation (Contd.)

The optimal Q-value function is given as follows:

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

The corresponding optimal deterministic policy is given by:

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

Now,

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

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

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

Deep Q Learning

- Learning the Q-value function is an integral part of solving the RL problem and Q learning is one possible solution. However, in most cases using this approach is infaesible, especially in the case of high dimensional state spaces, primarily due to computational inefficiency arising from the need to evaluate all possible action-state pair values (Van Hasselt et al., 2016)
- This is where Deep Q learning comes into play
- 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)

Deep Q Learning (Contd.)

- 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
- Target neural network is used to approximate the future discounted rewards ($Q_{target}(s_t, a_t; \theta^{(t)}_{target})$)
- Policy neural network on the other hand involves learning a parameterized Q value function $Q_{policy}(s_t, a_t; \theta^{(t)}_{policy})$
- The loss function utilised is as follows:

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

  $Y_t^{DQN} = r_{t+1} + \gamma \max_a Q_{target}(s_{t+1}, a, \theta_t)$ (Van Hasselt et al., 2016)
- We utilise Huber loss function (Seif, 2022)

Deep Q Learning (Contd.)

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

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

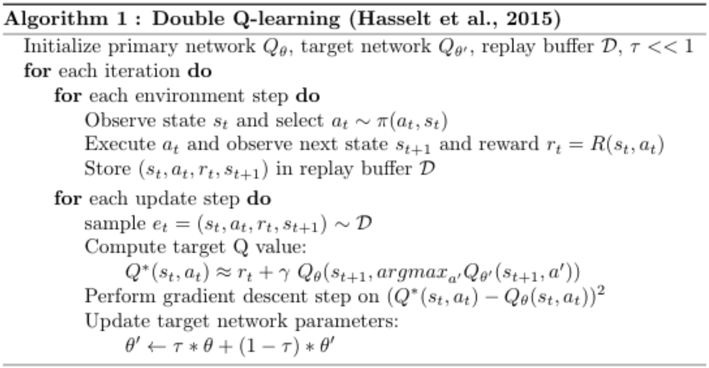
- After the policy network update, the parameters of the target network are updated according to the soft target rule (Lillicrap et al., 2019)

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

- 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)

Deep Q Learning (Contd.)

The pseudocode below has been adopted from (van Hasselt et al., 2015)



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

```
Initialize primary network $Q_\theta$, 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:
            $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
```

RMS Prop

- 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
- When the momentum parameter γ is chosen well, RMSProp will converge faster than SGD

RMS Prop (Contd.)

- The following equation updates the the approximation to the second moment of the gradient.

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

(Ruder, 2016)

- The next equation updates weight parameters using the root mean square.

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

- The momentum parameter is γ, for which a good choice is 0.9
- The learning rate parameter is η and a possible choice is 0.001
- The optimization algorithm RMSProp - Root Mean Square Propagation - builds on prior algorithms by adapting the learning rate using the gradient and the second moment

Adam

- 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 (Kingma & Ba, 2014)

- 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

Adaptive Gradient Methods - Adam (Contd.)

- The moments are usually initialized as 0's which leads to a potential problem referred to as initialization bias (Kingma & Ba, 2014). 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 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:

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

AdamW

- AdamW involves decopuling weight decay from the optimization steps taken with respect to the loss function in Adam (Loshchilov & Hutter, 2017). 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:

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

AdamW (Contd.)

Let us look at the effective stepsize in greater detail:

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

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

- $\frac{w_{t-1}}{(\sqrt{\hat{v}_t}+\epsilon)} \implies$ parameters with high magnitude of the gradient are regularized much less compared to ones with low magnitude
- AdamW involves decopuling weight decay from the computation of the gradient and first moment with the consequent parameter update as follows:

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

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

Rectified Adam

- RAdam, or rectified Adam, 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.
- From the ICLR paper “On the Variance of the Adaptive Learning Rate and Beyond,” the authors found that “Thus, to reduce such variance, it is better to use smaller learning rates in the first few epochs of training, which justifies the warmup heuristic.”

Rectified Adam (Contd.)

- 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

$$l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$$

$$r_t \leftarrow \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \hat{m}_t l_t$$

Else update parameters with un-adapted momentum

- 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.

Cartpole Environment

- The cartpole environment consists of a pole that moves on a frictionless surface
- 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
- State space in this case incorporates 4 elements (cart position, cart velocity, pole angle and the velocity at the tip of the pole)
- Action space involves two elements (moving left or moving right). In case angle of the cart with the vertical axis is ¡ 5^o the reward is +1
- 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 (Kurban, 2022)

Experimental setup

The deep Q learning algorithm is implemented in Python (Paszke 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

Experimental setup (Contd.)

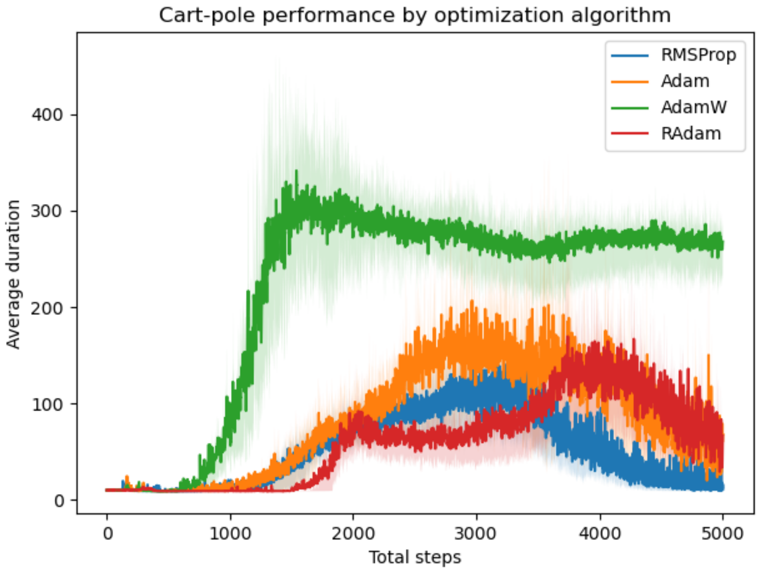
In order to directly compare the performance of different optimizers, the following training hyperparameters are held constant:

- 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
- 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

Experimental setup (Contd.)

- The quality of the policy cannot be directly inferred from training episode length since deep Q learning is an off policy algorithm
- After each time step, a separate evaluation environment runs 16 full cart-pole episodes from start to finish using the deterministic (greedy) policy
- The average total reward determines the quality of the deterministic policy
- Same randomly-initialized model is trained using each optimizer in turn
- Cart-pole environment used for training is re-initialized with the same random seed for every optimizer
- Entire training and evaluation procedure is repeated with 16 different randomly-initialized models to ensure a robust comparison
- 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.

Results and Observations



Results and Observations (Contd.)

- The graph illustrates the total number of iterations run for the respective algorithm plotted against the duration of the episode
- AdamW appears to perform realtively well compared to the other three algorithms
- Significantly less number of training steps are required for AdamW compared to the rest of the three
- Decoupling weight decay, the basis of AdamW, appears to have a beneficial impact on the performance when compared to Adam
- Poor performance of RAdam and RMSProp 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)

Results and Observations (Contd.)

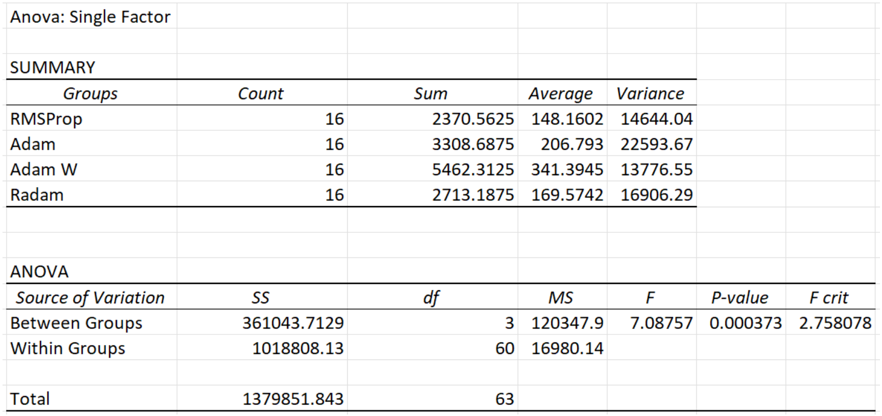
The cartpole environment corresponding to the best performing models for each optimization algorithm

- Untrained: `https://raw.githubusercontent.com/s-laverty/rl-optim-compare/main/img/8_4_init.gif`
- RMSProp: `https://raw.githubusercontent.com/s-laverty/rl-optim-compare/main/img/8_4_rmsprop_best.gif`
- Adam: `https://raw.githubusercontent.com/s-laverty/rl-optim-compare/main/img/8_4_adam_best.gif`
- AdamW: `https://raw.githubusercontent.com/s-laverty/rl-optim-compare/main/img/8_4_adamw_best.gif`
- RAdam: `https://raw.githubusercontent.com/s-laverty/rl-optim-compare/main/img/8_4_radam_best.gif`

Results and Observations (Contd.)

- Ran a one factor Anova test to test the statistical significance of the difference in the performance of the 4 algorithms
- 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

Results and Observations (Contd.)



Anova: Single Factor

SUMMARY

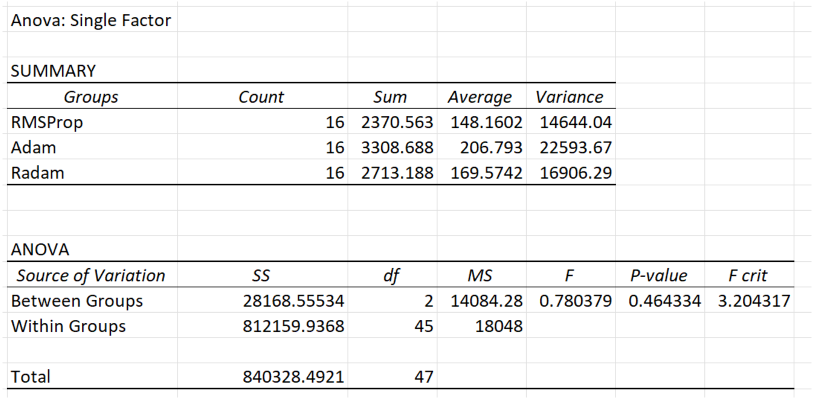
Groups	*Count*	*Sum*	*Average*	*Variance*
RMSProp	16	2370.5625	148.1602	14644.04
Adam	16	3308.6875	206.793	22593.67
Adam W	16	5462.3125	341.3945	13776.55
Radam	16	2713.1875	169.5742	16906.29

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				

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.

Results and Observations (Contd.)



Anova: Single Factor

SUMMARY

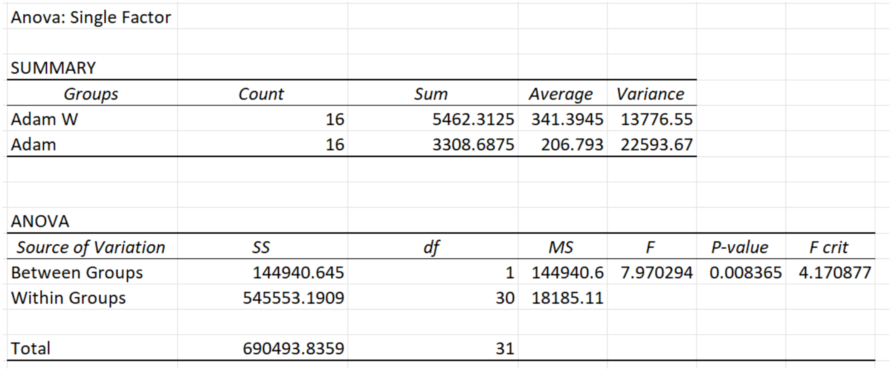
Groups	*Count*	*Sum*	*Average*	*Variance*
RMSProp	16	2370.563	148.1602	14644.04
Adam	16	3308.688	206.793	22593.67
Radam	16	2713.188	169.5742	16906.29

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				

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.

Results and Observations (Contd.)



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				

At 1% significance level we fail to we reject the null hypothesis $\Longrightarrow$ there does exist a statistically significant difference between the performance of Adam and AdamW as we hypothesized.

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
- 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
- For 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 I

- 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&amp;%3Blist=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&amp;%3Bindex=2&amp;gl=PK.
- Rl course by david silver - lecture 1: Introduction to reinforcement learning, May 2015b.
  URL https://www.youtube.com.pk/watch?v=2pWv7GOvuf0&amp;list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ&amp;index=1.

References II

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`.

References III

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-245b5

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

References IV

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.

References V

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-442c1e830`

References VI

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-mach

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

References VII

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.

References VIII

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-

The End

Questions? Comments?