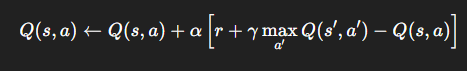
REPORT

Reinforcement Learning and Optimizer Comparison

In **Reinforcement Learning (RL)**, an agent learns to make decisions by interacting with an environment to maximize a reward signal. The main components:

* **Agent**: The decision-maker.
* **Environment**: Where the agent operates.
* **State (s)**: The current situation of the agent.
* **Action (a)**: What the agent can do.
* **Reward (r)**: Feedback from the environment.
* **Policy (π)**: A strategy the agent follows to choose actions.

Q-learning is a **model-free** RL algorithm. It aims to learn the **optimal action-value function**:

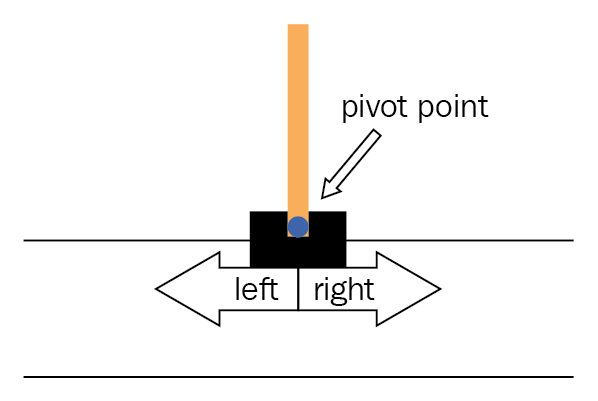


The core **Q-learning update rule** is:

Where:

* alphaα is the **learning rate**
* gamma is the **discount factor**
* r is the reward received
* s′ is the next state

## Applications

* **Atari Games**: DQN was famously used by DeepMind to beat human-level performance on Atari 2600 games.
* **Robotics**: Motion planning, grasping.
* **Finance**: Trading strategies.
* **Autonomous Driving**: Lane-keeping, navigation
* **Observation Space (State)**: A vector of 4 values:
  1. Cart Position
  2. Cart Velocity
  3. Pole Angle
  4. Pole Angular Velocity
* **Action Space** (Discrete):
  1. 0: Move cart to the **left**
  2. 1: Move cart to the **right**
* **Reward**:
  1. +1 reward for every timestep the pole remains upright (i.e., the episode continues)
  2. Goal is to keep it upright as long as possible (max score: 500)
* **Termination Conditions**:
  1. Pole angle is too large (more than ~12 degrees from vertical)
  2. Cart position is too far from center (more than ~2.4 units from center)
  3. Episode length reaches 500 steps
* **Agent's Task**

Use a **policy** to decide whether to move left or right to **maximize cumulative reward**—i.e., keep the pole balanced for as long as possible.

# Task 1: Deep Q-Learning using Neural Networks for CartPole

In this task, we trained an agent using Deep Q-Learning to solve the CartPole-v1 environment. The model architecture included two hidden layers with ReLU activation functions and a linear output layer for action values.

The training was done for 100 episodes using the following hyperparameters:  
- GAMMA = 0.95  
- MEMORY\_SIZE = 2000  
- BATCH\_SIZE = 32  
- EPSILON\_START = 1.0  
- EPSILON\_MIN = 0.1  
- EPSILON\_DECAY = 0.995  
- TARGET\_UPDATE = 10  
- LEARNING\_RATE = 0.001

## Observations

1. Epsilon decreases exponentially with each episode until it reaches the minimum value of 0.1. This was achieved at Episode 40.  
2. Initially, the agent explores a lot (with high ε), hence lower rewards. As ε reduces, the agent starts exploiting more, improving its performance.  
3. After ε reaches its minimum (0.1), performance becomes more stable, and high scores are observed, such as 232, 241, and 283.

## Answer to Questions

- Epsilon decays exponentially and promotes exploitation as it lowers. The agent's performance stabilizes and improves post ε-min.  
- For EPSILON\_MIN = 0.1, it is reached by Episode 40. If EPSILON\_MIN is set to a smaller value (e.g., 0.01), it would take longer (more episodes) to decay to that value.  
- Changing hyperparameters such as learning rate or batch size affects the agent's convergence speed and score variance.

## RESULT:

model = Sequential()

model.add(Dense(24, input\_dim=self.state\_size, activation='relu'))

model.add(Dense(24, activation='relu'))

model.add(Dense(self.action\_size, activation='linear'))

model.compile(loss='mse', optimizer=Adam(learning\_rate=LEARNING\_RATE))

Episode: 1/100, Score: 17.0, Epsilon: 1.00

Episode: 2/100, Score: 17.0, Epsilon: 0.99

Episode: 3/100, Score: 41.0, Epsilon: 0.81

Episode: 4/100, Score: 15.0, Epsilon: 0.76

Episode: 5/100, Score: 16.0, Epsilon: 0.70

Episode: 6/100, Score: 12.0, Epsilon: 0.67

Episode: 7/100, Score: 12.0, Epsilon: 0.63

Episode: 8/100, Score: 12.0, Epsilon: 0.60

Episode: 9/100, Score: 11.0, Epsilon: 0.57

Episode: 10/100, Score: 13.0, Epsilon: 0.53

Episode: 11/100, Score: 13.0, Epsilon: 0.50

Episode: 12/100, Score: 12.0, Epsilon: 0.48

Episode: 13/100, Score: 14.0, Epsilon: 0.45

Episode: 14/100, Score: 10.0, Epsilon: 0.43

Episode: 15/100, Score: 13.0, Epsilon: 0.40

Episode: 16/100, Score: 16.0, Epsilon: 0.37

Episode: 17/100, Score: 13.0, Epsilon: 0.35

Episode: 18/100, Score: 11.0, Epsilon: 0.33

Episode: 19/100, Score: 9.0, Epsilon: 0.32

Episode: 20/100, Score: 10.0, Epsilon: 0.31

Episode: 21/100, Score: 13.0, Epsilon: 0.29

Episode: 22/100, Score: 10.0, Epsilon: 0.28

Episode: 23/100, Score: 9.0, Epsilon: 0.26

Episode: 24/100, Score: 10.0, Epsilon: 0.25

Episode: 25/100, Score: 11.0, Epsilon: 0.24

Episode: 26/100, Score: 16.0, Epsilon: 0.22

Episode: 27/100, Score: 10.0, Epsilon: 0.21

Episode: 28/100, Score: 11.0, Epsilon: 0.20

Episode: 29/100, Score: 10.0, Epsilon: 0.19

Episode: 30/100, Score: 10.0, Epsilon: 0.19

Episode: 31/100, Score: 9.0, Epsilon: 0.18

Episode: 32/100, Score: 10.0, Epsilon: 0.17

Episode: 33/100, Score: 33.0, Epsilon: 0.15

Episode: 34/100, Score: 10.0, Epsilon: 0.14

Episode: 35/100, Score: 11.0, Epsilon: 0.13

Episode: 36/100, Score: 11.0, Epsilon: 0.13

Episode: 37/100, Score: 10.0, Epsilon: 0.12

Episode: 38/100, Score: 10.0, Epsilon: 0.11

Episode: 39/100, Score: 10.0, Epsilon: 0.11

Episode: 40/100, Score: 13.0, Epsilon: 0.10

Episode: 41/100, Score: 10.0, Epsilon: 0.10

Episode: 42/100, Score: 11.0, Epsilon: 0.10

Episode: 43/100, Score: 10.0, Epsilon: 0.10

Episode: 44/100, Score: 12.0, Epsilon: 0.10

Episode: 45/100, Score: 11.0, Epsilon: 0.10

Episode: 46/100, Score: 11.0, Epsilon: 0.10

Episode: 47/100, Score: 13.0, Epsilon: 0.10

Episode: 48/100, Score: 14.0, Epsilon: 0.10

Episode: 49/100, Score: 9.0, Epsilon: 0.10

Episode: 50/100, Score: 13.0, Epsilon: 0.10

Episode: 51/100, Score: 19.0, Epsilon: 0.10

Episode: 52/100, Score: 24.0, Epsilon: 0.10

Episode: 53/100, Score: 11.0, Epsilon: 0.10

Episode: 54/100, Score: 14.0, Epsilon: 0.10

Episode: 55/100, Score: 39.0, Epsilon: 0.10

Episode: 56/100, Score: 15.0, Epsilon: 0.10

Episode: 57/100, Score: 12.0, Epsilon: 0.10

Episode: 58/100, Score: 15.0, Epsilon: 0.10

Episode: 59/100, Score: 38.0, Epsilon: 0.10

Episode: 60/100, Score: 15.0, Epsilon: 0.10

Episode: 61/100, Score: 101.0, Epsilon: 0.10

Episode: 62/100, Score: 125.0, Epsilon: 0.10

Episode: 63/100, Score: 96.0, Epsilon: 0.10

Episode: 64/100, Score: 185.0, Epsilon: 0.10

Episode: 65/100, Score: 104.0, Epsilon: 0.10

Episode: 66/100, Score: 162.0, Epsilon: 0.10

Episode: 67/100, Score: 218.0, Epsilon: 0.10

Episode: 68/100, Score: 126.0, Epsilon: 0.10

Episode: 69/100, Score: 169.0, Epsilon: 0.10

Episode: 70/100, Score: 82.0, Epsilon: 0.10

Episode: 71/100, Score: 171.0, Epsilon: 0.10

Episode: 72/100, Score: 118.0, Epsilon: 0.10

Episode: 73/100, Score: 146.0, Epsilon: 0.10

Episode: 74/100, Score: 113.0, Epsilon: 0.10

Episode: 75/100, Score: 130.0, Epsilon: 0.10

Episode: 76/100, Score: 190.0, Epsilon: 0.10

Episode: 77/100, Score: 272.0, Epsilon: 0.10

Episode: 78/100, Score: 283.0, Epsilon: 0.10

Episode: 79/100, Score: 125.0, Epsilon: 0.10

Episode: 80/100, Score: 187.0, Epsilon: 0.10

Episode: 81/100, Score: 132.0, Epsilon: 0.10

Episode: 82/100, Score: 118.0, Epsilon: 0.10

Episode: 83/100, Score: 118.0, Epsilon: 0.10

Episode: 84/100, Score: 18.0, Epsilon: 0.10

Episode: 85/100, Score: 176.0, Epsilon: 0.10

Episode: 86/100, Score: 123.0, Epsilon: 0.10

Episode: 87/100, Score: 150.0, Epsilon: 0.10

Episode: 88/100, Score: 218.0, Epsilon: 0.10

Episode: 89/100, Score: 172.0, Epsilon: 0.10

Episode: 90/100, Score: 163.0, Epsilon: 0.10

Episode: 91/100, Score: 241.0, Epsilon: 0.10

Episode: 92/100, Score: 152.0, Epsilon: 0.10

Episode: 93/100, Score: 192.0, Epsilon: 0.10

Episode: 94/100, Score: 89.0, Epsilon: 0.10

Episode: 95/100, Score: 138.0, Epsilon: 0.10

Episode: 96/100, Score: 77.0, Epsilon: 0.10

Episode: 97/100, Score: 116.0, Epsilon: 0.10

Episode: 98/100, Score: 226.0, Epsilon: 0.10

Episode: 99/100, Score: 151.0, Epsilon: 0.10

Episode: 100/100, Score: 105.0, Epsilon: 0.10

MAX SCORE :283 , EPSILON:0.10 ,EPISODE 78/100

# Task 2: Optimizer Comparison - Momentum vs RMSProp vs Adam

This task involved implementing three optimizers from scratch on a simple quadratic function to analyze their convergence speed and loss minimization.

The loss was plotted against iterations for each optimizer. The following number of iterations were needed to converge:  
Momentum converged in 157 iterations

RMSProp converged in 22 iterations

Adam converged in 160 iterations

RMSProp converged the fastest but with a more aggressive initial drop. Momentum had a smooth and stable convergence. Adam took more iterations due to fine-tuned control using both first and second moments.

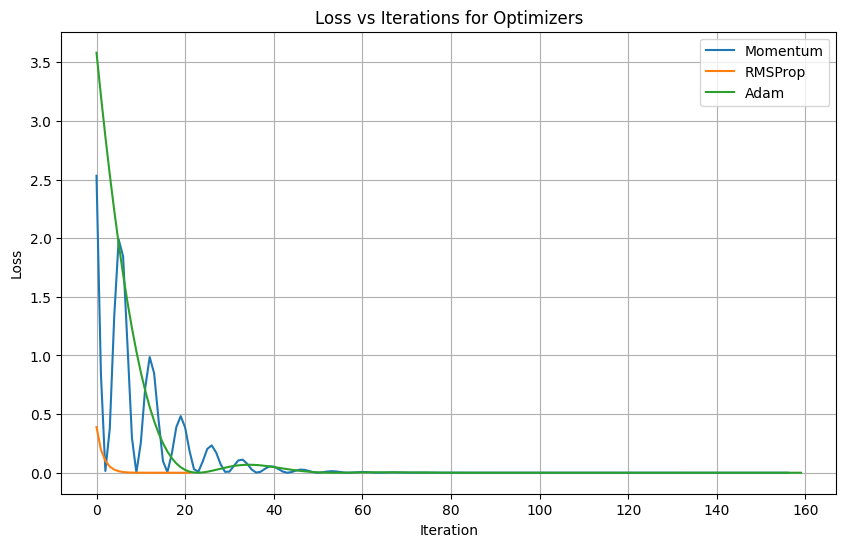
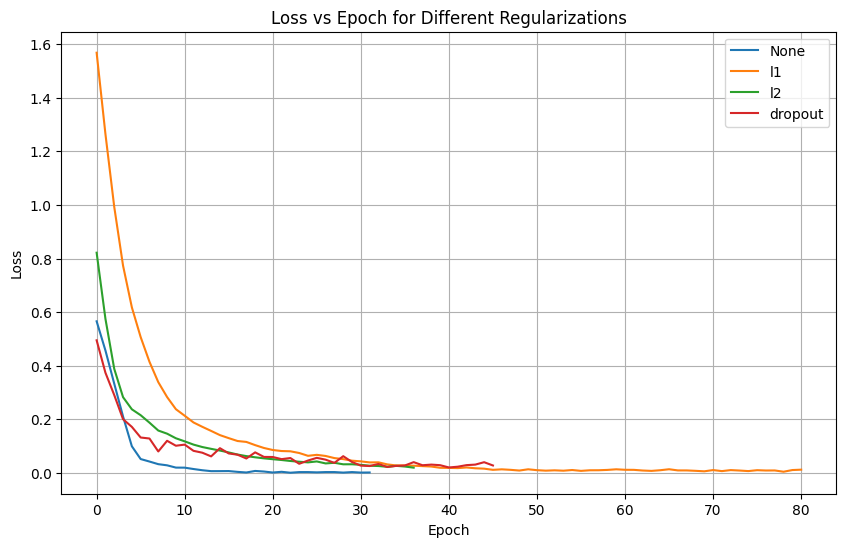


Figure: Loss vs Iterations for Momentum, RMSProp, and Adam Optimizers.

RESULTS:

Training with regularization: None

Reg: None, Ep: 1/5, Score: 11.0, Epsilon: 1.00

Reg: None, Ep: 2/5, Score: 14.0, Epsilon: 1.00

Reg: None, Ep: 3/5, Score: 8.0, Epsilon: 1.00

Reg: None, Ep: 4/5, Score: 19.0, Epsilon: 0.91

Reg: None, Ep: 5/5, Score: 15.0, Epsilon: 0.85

Training with regularization: l1

Reg: l1, Ep: 1/5, Score: 15.0, Epsilon: 1.00

Reg: l1, Ep: 2/5, Score: 18.0, Epsilon: 1.00

Reg: l1, Ep: 3/5, Score: 44.0, Epsilon: 0.81

Reg: l1, Ep: 4/5, Score: 21.0, Epsilon: 0.73

Reg: l1, Ep: 5/5, Score: 19.0, Epsilon: 0.67

Training with regularization: l2

Reg: l2, Ep: 1/5, Score: 10.0, Epsilon: 1.00

Reg: l2, Ep: 2/5, Score: 15.0, Epsilon: 1.00

Reg: l2, Ep: 3/5, Score: 13.0, Epsilon: 0.98

Reg: l2, Ep: 4/5, Score: 22.0, Epsilon: 0.88

Reg: l2, Ep: 5/5, Score: 12.0, Epsilon: 0.83

Training with regularization: dropout

Reg: dropout, Ep: 1/5, Score: 17.0, Epsilon: 1.00

Reg: dropout, Ep: 2/5, Score: 14.0, Epsilon: 1.00

Reg: dropout, Ep: 3/5, Score: 18.0, Epsilon: 0.92

Reg: dropout, Ep: 4/5, Score: 12.0, Epsilon: 0.87

Reg: dropout, Ep: 5/5, Score: 20.0, Epsilon: 0.79

THANK YOU

SARGAM TYAGI