REINFORCEMENT LEARNING

Real-Time DQN Training for CartPole Environment with Live Performance Plottings

Reinforcement Learning

Reinforcement Learning Steps

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

OBJECTIVE	Train a DQN agent to balance a pole on a cart in the CartPole-v1 environment using reinforcement learning.
GOAL	Develop an agent that maximizes its balance time by making intelligent decisions based on observations.
APPROACH	Use DQN algorithm with experience replay and epsilon-greedy policy, implemented with TensorFlow in OpenAI Gym's CartPole environment.
KEY TECHNIQUES	DQN, experience replay, and epsilon-greedy policy for balancing exploration and exploitation.
OUTCOMES	The trained agent learns to balance the pole for extended periods, evaluated by episode rewards and fine-tuned hyperparameters.

Hyper Fine Tuning

gamma (0.95), epsilon (1.0), epsilon decay (0.995), epsilon minimum (0.01), and learning rate (0.001)

Reinforcement Learning

1. **Formulate Problems**	Task of keeping the pole upright.
2. **Create Environment**	OpenAI Gym to simulate the CartPole environment.
3. **Define Reward**	Reward for staying upright; penalty for failure.
4. **Create Agent**	Build a neural network to predict actions
5. **Train Agent**	Learn from interactions and replay experiences.
6. **Validate Agent**	Test performance in the environment.
7. **Deploy Policy**	Finalize and apply the trained policy.

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

CartPole-v0	**Balance a pole on a moving cart.**	**Quick reflexes and real-time decisions.**
MountainCar-v0	**Drive a car up a hill using momentum.**	Control acceleration and physics.
LunarLander-v2	**Land a spacecraft on a designated pad.**	Balance fuel use and landing accuracy.
Breakout-v0	**Hit a ball to break blocks in a game.**	Prediction and timing.
Atari Games	**Classic games requiring strategic play.**	Unique mechanics per game.
Roboschool	**Simulated robotic environments for control learning.**	Complex physics and precise control.

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

Environment	Reward Structure	Description
CartPole-v0	+ve for each timestep the pole is balanced.	Encourages longer balance time.
MountainCar-v0	+ve for reaching the flag, -ve for timestep.	Motivates climbing to the goal.
LunarLander-v2	+ve for landing successfully, -ve for crashes/fuel usage.	Balances careful landing and fuel efficiency.
Breakout-v0	+ve for each block destroyed, -ve for missing the ball.	Rewards aggressive play and accuracy.
Atari Games	rewards for scoring and achieving objectives.	Encourages varied gameplay strategies.
Roboschool	+ve for completing tasks, -ve for failures	Promotes effective task execution.

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

Estimate Models Used

- DDPG (Deep Deterministic Policy Gradient)

- PPO (Proximal Policy Optimization)

- DQN (Deep Q-Network)

- Because of Stable, Continuous control.

Model	Price	Training
DDPG (Deep Deterministic PG)	Varies	Yes
PPO (Proximal Policy Opt.)	Varies	Yes
DQN (Deep Q-Network)	Varies	Yes

Deliverable	Description
Trained Q-Table	A Q-table with learned values for state-action pairs.
Average Reward	The average reward achieved by the trained agent.
Policy for CartPole	A learned policy for the CartPole environment.
Exploration Decay Curve	A plot showing how the exploration rate decays over episodes.
Performance Evaluation	Statistics on how well the agent performs in the environment.

Reinforcement Learning

26.09.2023

Estimate Equations Used

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

- Bellman Equation (for value function approximation)

$V(s) = R(s) + \gamma * \Sigma P(s, s') * V(s')$

Bellman Equation (for value function approximation)

V(s): Value function
R(s): Reward function
γ: Discount factor
Σ: Summation over possible states s'
P(s, s'): Transition probability from state s to s' (s prime)

- Policy Gradient Equation (for policy optimization)

$\nabla J(\theta) = \Sigma \ \nabla\pi($a

Policy Gradient Equation (for policy optimization)

∇J(θ): Gradient of the objective function with respect to policy parameters θ
∇π(a): Gradient of the policy with respect to an action a

- DQN-Learning Equation (for action-value function)

Q(s,a;θ)=r+γa′maxQ(s′,a′;θ−)

Q-Learning Equation (for action-value function)

- θ\thetaθ: Parameters (weights) of the neural network
- θ−\theta^{-}θ−: Target network parameters

Bellman Equation Importance:
enables agents to estimate and optimize future rewards

Reinforcement Learning

Reinforcement Learning Steps

1. Formulate Problems
|
2. Create Environment
|
3. Define Reward
|
4. Create Agent
|
5. Train Agent
|
6. Validate Agent
|
7. Deploy Policy

Comparision b/w Reinforcement Learning & Symbolic AI

Aspect	Symbolic AI	Reinforcement Learning
Problem-Solving Approach	Deductive reasoning, rule-based	Trial and error, reward-based
When to Use Them	When you know the rules and facts	When you need to explore and learn
Combining Both	Can provide prior knowledge	May incorporate symbolic knowledge
Example	**Medical (Mycin)**	**Tesla Car -Auto Pilot**

Example:
Reinforcement Learning = Tesla Car -Auto Pilot
Symbolic AI = Medical (Mycin)

Neural-symbolic architectures / hybrid intelligence

Neural-Symbolic Architectures / Hybrid Intelligence	Application	How They Are Applied	Examples
Neural-Symbolic Integration	Combining neural networks and symbolic reasoning	Enhancing AI systems with the strengths of both approaches	**Doctors (for Medical Diagnosis)**
Knowledge Graph Embeddings	Linking symbolic knowledge to neural representations	Improving reasoning and inference with neural networks	**Movie Recommendations**
Explainable AI (XAI)	Making neural network decisions interpretable	Ensuring transparency and trust in AI systems	**Explanations (for AI decisions)**
Natural Language Processing with Logic	Incorporating symbolic logic into NLP tasks	Enhancing natural language understanding with logic	**Smart Chatbots (for Natural Language Understanding)**

Example:
Medical Diagnosis, Movie Recommendations, NLP Chatbot

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

Environment	Training Focus	Learning Challenge
CartPole-v1	Balance a pole on a moving cart.	Quick decision-making, stability control.
MountainCar-v0	Drive up a steep hill.	Requires long-term planning, momentum control.
LunarLander-v2	Safely land a lunar module.	Precision landing, balancing fuel and landing force.
Breakout-v0	Break bricks with a ball and paddle.	Reaction timing, aiming strategies.
Pong-v0	Play table tennis against an AI opponent.	Competitive play, fast reflexes, strategy.
BipedalWalker-v3	Make a bipedal robot walk on uneven terrain.	Complex control, stability in movement.

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

Validation Method	Description	Environments
Episode Reward	Total reward over episodes.	CartPole, MountainCar, LunarLander, Pong
Success Rate	Percent of goals achieved.	LunarLander, MountainCar, BipedalWalker
Survival Time	Time agent stays active before failure.	CartPole, Pong, BipedalWalker
Action Efficiency	Minimizing steps to complete tasks.	Breakout, LunarLander, MountainCar
Precision in Actions	Accuracy in performing tasks.	LunarLander, Breakout, Pong
Policy Consistency	Consistent actions across similar states.	All environments

Reinforcement Learning

26.09.2023

ESTIMATE AWS SAGEMAKER PRICING

1. **Formulate Problems**

|

2. **Create Environment**

|

3. **Define Reward**

|

4. **Create Agent**

|

5. **Train Agent**

|

6. **Validate Agent**

|

7. **Deploy Policy**

Let's calculate the approximate monthly cost in INR:

Monthly Cost (in USD) = $3.06 * 24 hours * 30 days = $2,206.40
Monthly Cost (in INR) ≈ $2,206.40 * 73.5 INR/USD

Monthly Cost (in USD) ≈ $2,203.20 USD

Pricing:- p3.2xlarge instance:
Approximately $3.825 Pricing/Hour

Instance Type	GPU	vCPU	Memory (RAM)	Storage	Notes
p3.2xlarge (or similar)	NVIDIA V100 GPU	8	16 GB	EBS volumes as needed for data/model storage	GPU acceleration for RL training

Reinforcement Learning

1. **Formulate Problems**
2. **Create Environment**
3. **Define Reward**
4. **Create Agent**
5. **Train Agent**
6. **Validate Agent**
7. **Deploy Policy**

AWS DeepRacer Pricing

AWS Link:-
https://aws.amazon.com/deepracer/pricing/

With AWS DeepRacer, you can create your own machine learning models (in a process called 'training') and race them (in a process called 'evaluation').

Free Tier	Service pricing
10 free hours to train or evaluate models for 30 days	Training or evaluation: $3.50 per hour
5GB of free storage during your first month.	Model storage: $0.023 per GB-month

Device pricing	Pricing
AWS DeepRacer	($399) is a fully autonomous 1/18th scale, four-wheel drive car.Using a single 4 megapixel camera with 1080p resolution to view the track
AWS DeepRacer Evo	($598) is the next generation in autonomous racing.
AWS DeepRacer Sensor Kit	($249).

Pricing Example:

AWS DeepRacer Jobs		**Hours**	**Cost per hour ($)**	**Total ($)**
Model training		**2**	**$3.50**	**$7.00**
Model evaluation		**0.083**	**$3.50**	**$0.29**
Total				**$7.29**
AWS DeepRacer Storage	**GB**	**GB-month used**	**Cost per GB-month ($)**	**Total ($)**
Model storage	**3.96**	**3.96**	**$0.02**	**$0.09**