Deep Q-Learning for High-Dimensional Control Tasks

Yoav Sabag

Yftach Gil

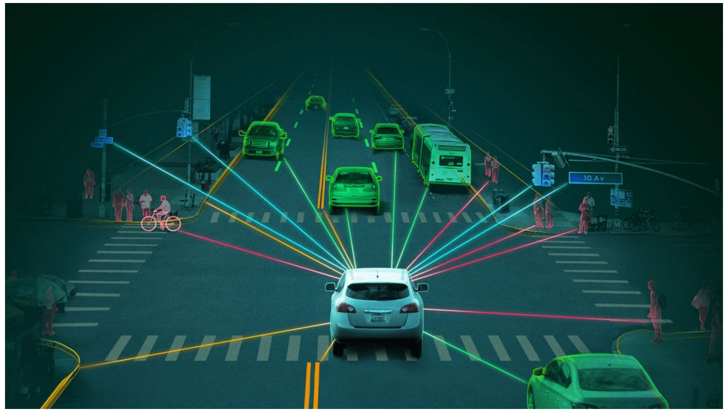
Agenda

- Introduction
- Methods
- Results
- Summary
- Conclusion

Introduction

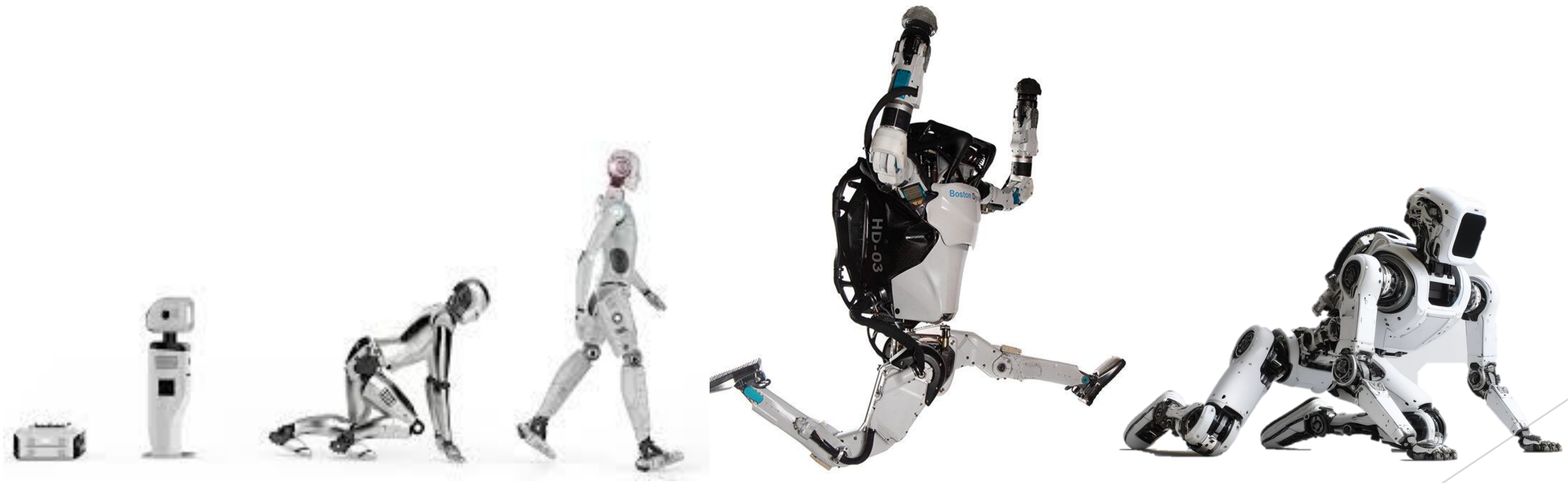
The need for controlling devices

- Cameras are highly common high-dimension sensor
- Control application: Robots, Autonomous vehicles, Traffic optimization

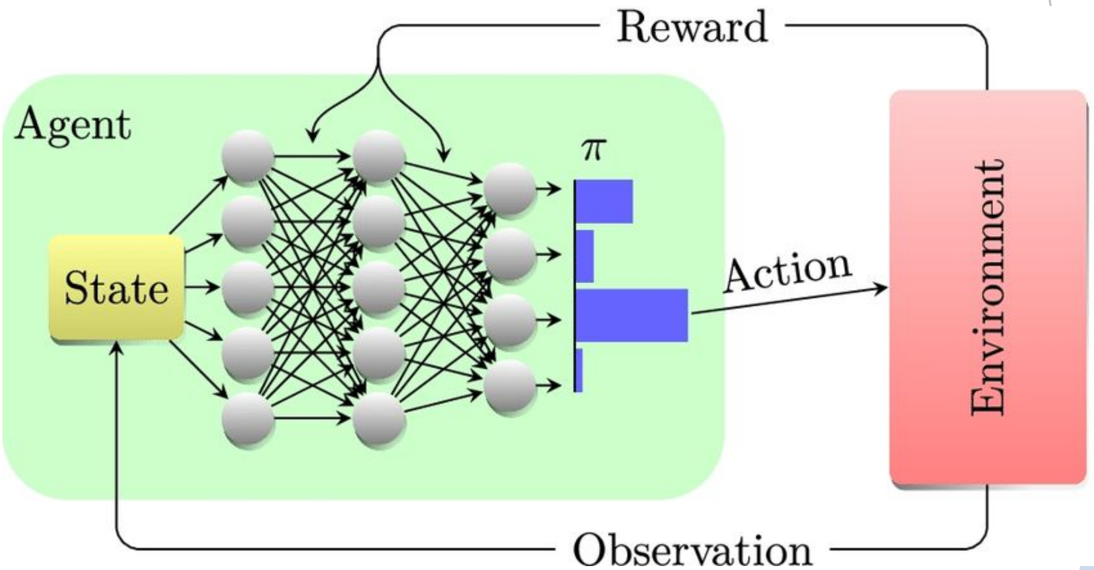


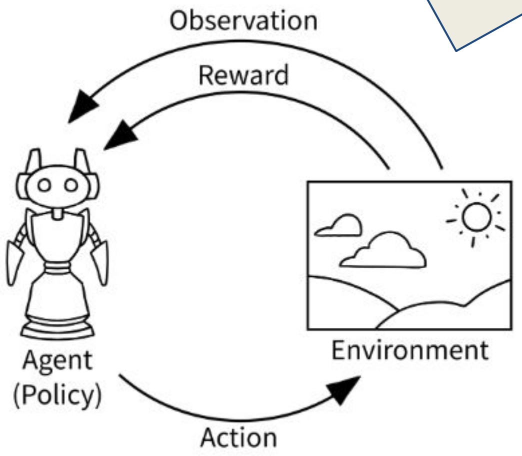
The problem: how to perform control in a high-dimensional environment?

- Input: image (video) observation
- Output: action from a set of possible action
- Target: Human level or higher control for a task

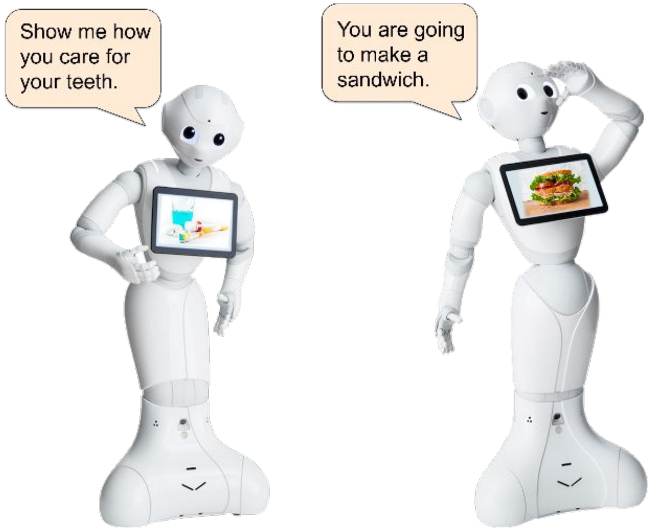


Solution - Reinforcement learning



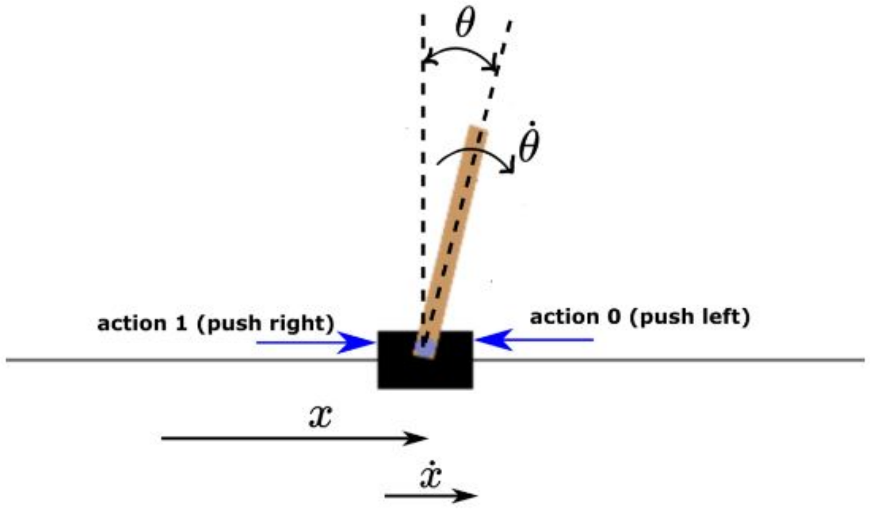


Methods


Show me how you care for your teeth.
You are going to make a sandwich.

Environment - Cart Pole

- Action: 0=Push Left, 1=Push Right
- Goal: keep the pole upright for as long as possible
- Reward: +1 for each step
- The Observation Space is [Position, Velocity, Angle, Angular Velocity]
- Episode end conditions:
  - Pole Angle is greater than ±12°
  - Cart Position is in the range ±2.4
  - Time is “long enough”: steps reach 500 (for v1) or 200 (for v0)



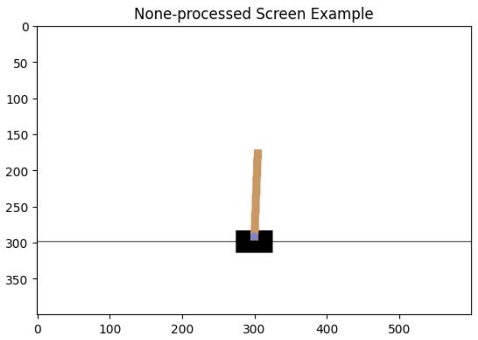
Environment

- Cart Pole v1 from GYM
- https://www.youtube.com/watch?v=B4E9tGmONn0
- Observation space is converted from vector of size 4 to image (frame) of 90x40

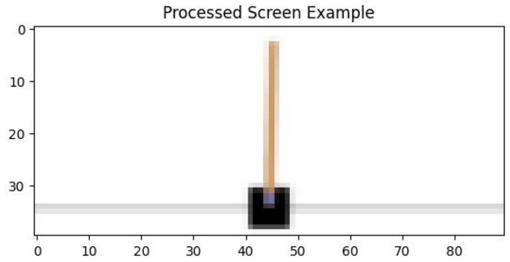
shape (4)

Num	Observation
0	Cart Position
1	Cart Velocity
2	Pole Angle
3	Pole Angular Velocity

600x400x3



90x40x3

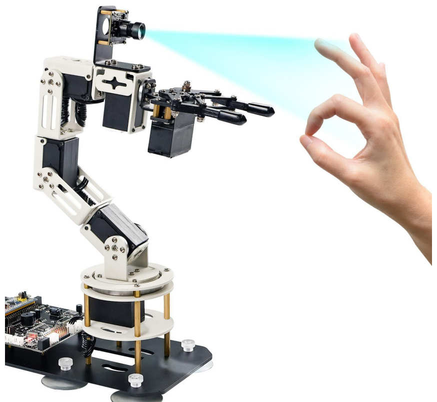


Metrics and evaluation

Average reward: The average score obtained by the agent over <u>100</u> of episodes

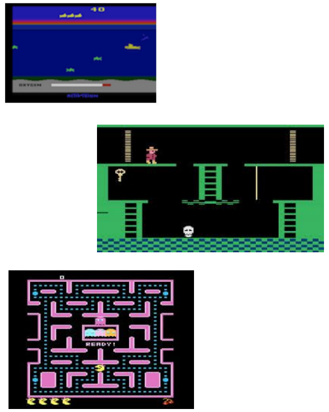
Episode length: the duration of each game episode (number of steps until end)

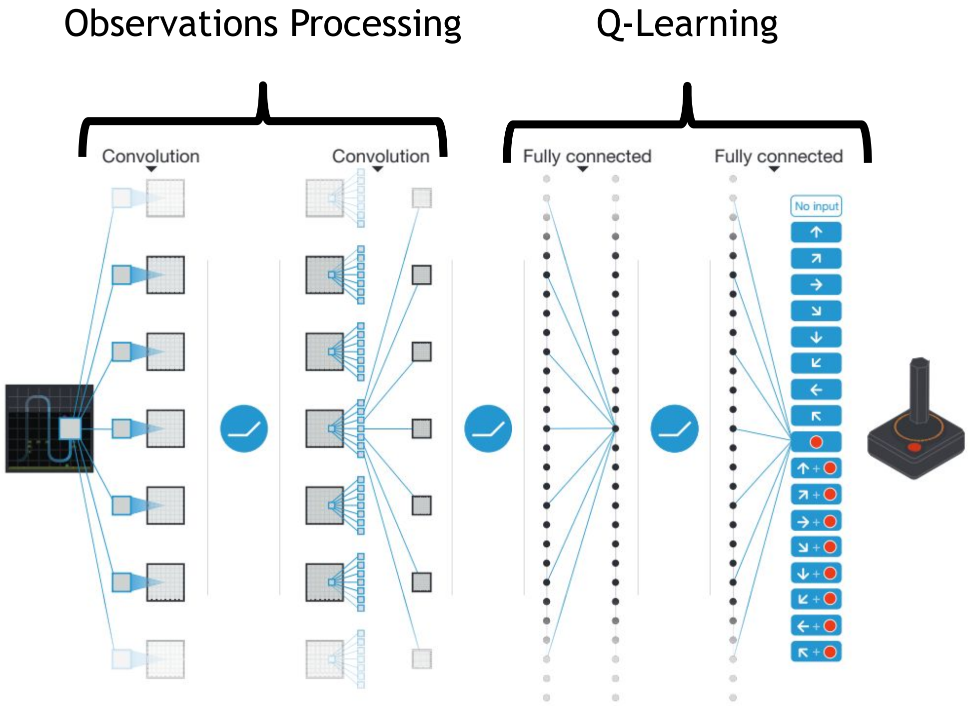
Training Episodes: the number of training episodes required for the agent to achieve defined level of performance.



Referenced Literature

- “Human-level control through deep reinforcement learning”
- RL model to master dozens of games with same model and hyperparameters
- Excel human level control over tasks
- Deal with high-dimensional sensory input





https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

Why Deep Q-Learning

- **Efficient Learns** successful policies directly from high dimensional inputs
- **Handling Non-linear Policies:**masters complex nonlinear strategies
- **Robustness to Different Domains:** as demonstrated in its application to multiple Atari games
- **Integration of Reward and Perception:** like the human brain
- **Continuous Improvement:** keep learning through interactions

METHODS - Model parts

- Agent
  - in charge of getting the frames, performs the actions determined by the policy
- DQN:

```
(conv1): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4), padding=(1, 1))
(conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
(conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(fc1): Linear(in_features=2816, out_features=32, bias=True)
(fc2): Linear(in_features=32, out_features=64, bias=True)
(fc3): Linear(in_features=64, out_features=128, bias=True)
(out): Linear(in_features=128, out_features=2, bias=True)
```

- Experience
  - stores each step's state, action, next_state and reward
- Replay memory
  - During training replay memories are queried from a batch to "replay" the agent's experience. Using replay memory the agent "remembers" best action in a defined number of episodes. This increases robustness (prevents degradation).
- Policies:
  - policy_net and target_net policies are identical nets
  - the target_net is fed with "next_states" and than the loss is computed between policy_net current state and next state.

METHODS - Model Parts

- initialize env, create policy_net and target_net
- take action → reward, state
- store experience(=state, action, next_states, reward) in replay buffer
- sample random batch from memory
- get q-values for current-policy next-target
- calculate Q-values using Bellman Optimality Equation):

$$q\_values = reward + \gamma \cdot next\_q\_values$$

- optimal action-value function:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots | s_t = s,\ a_t = a,\ \pi\right]$$

- Loss:

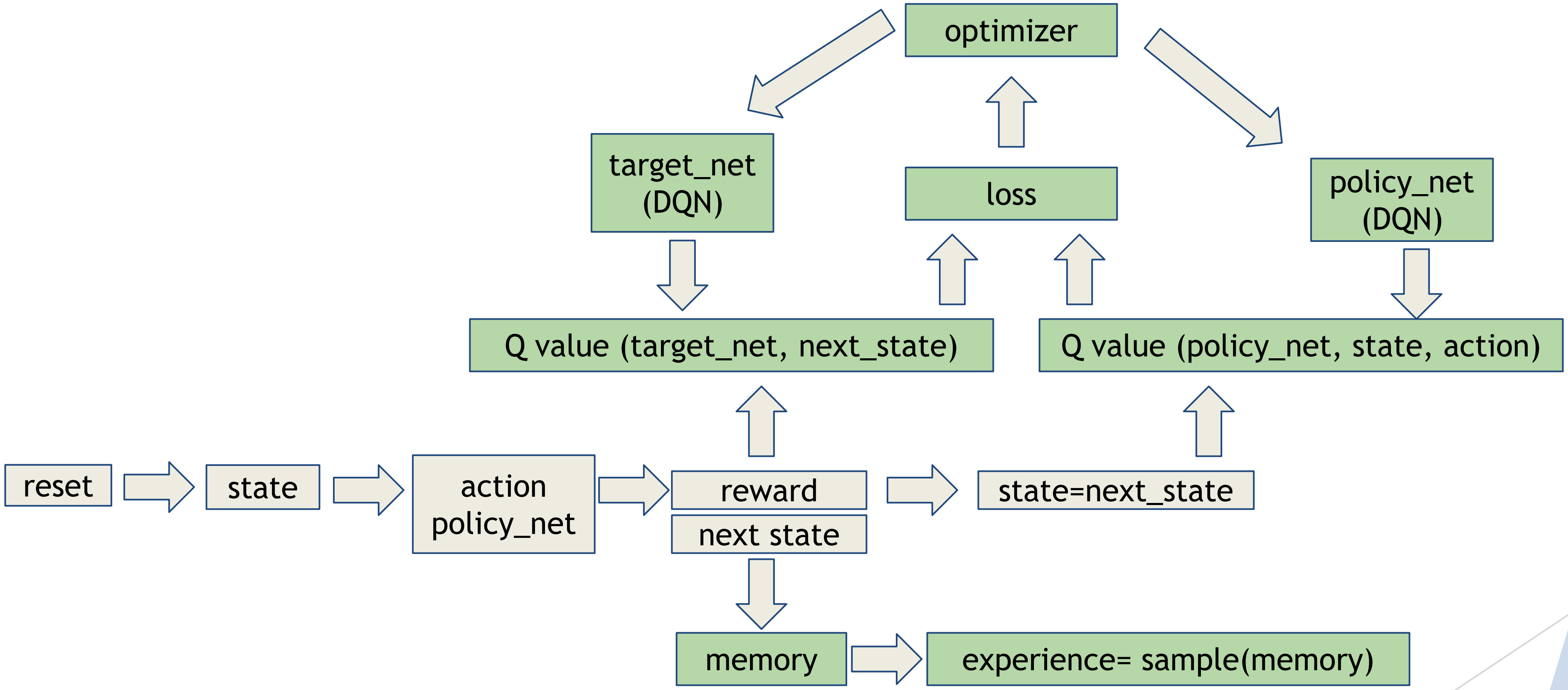
$$MSE = \frac{1}{n}\sum(current\_q\_value - target\_q\_value)^2$$

- optimal loss function:

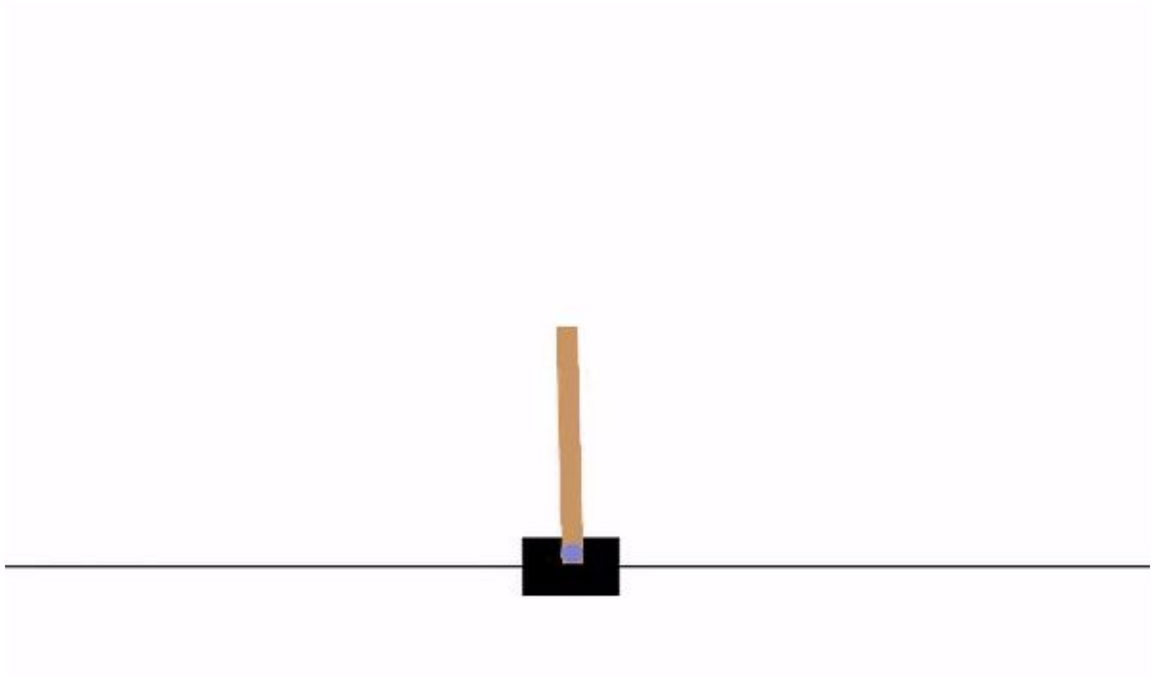
$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

- optimizer: update the network weights

METHODS - Architecture Scheme

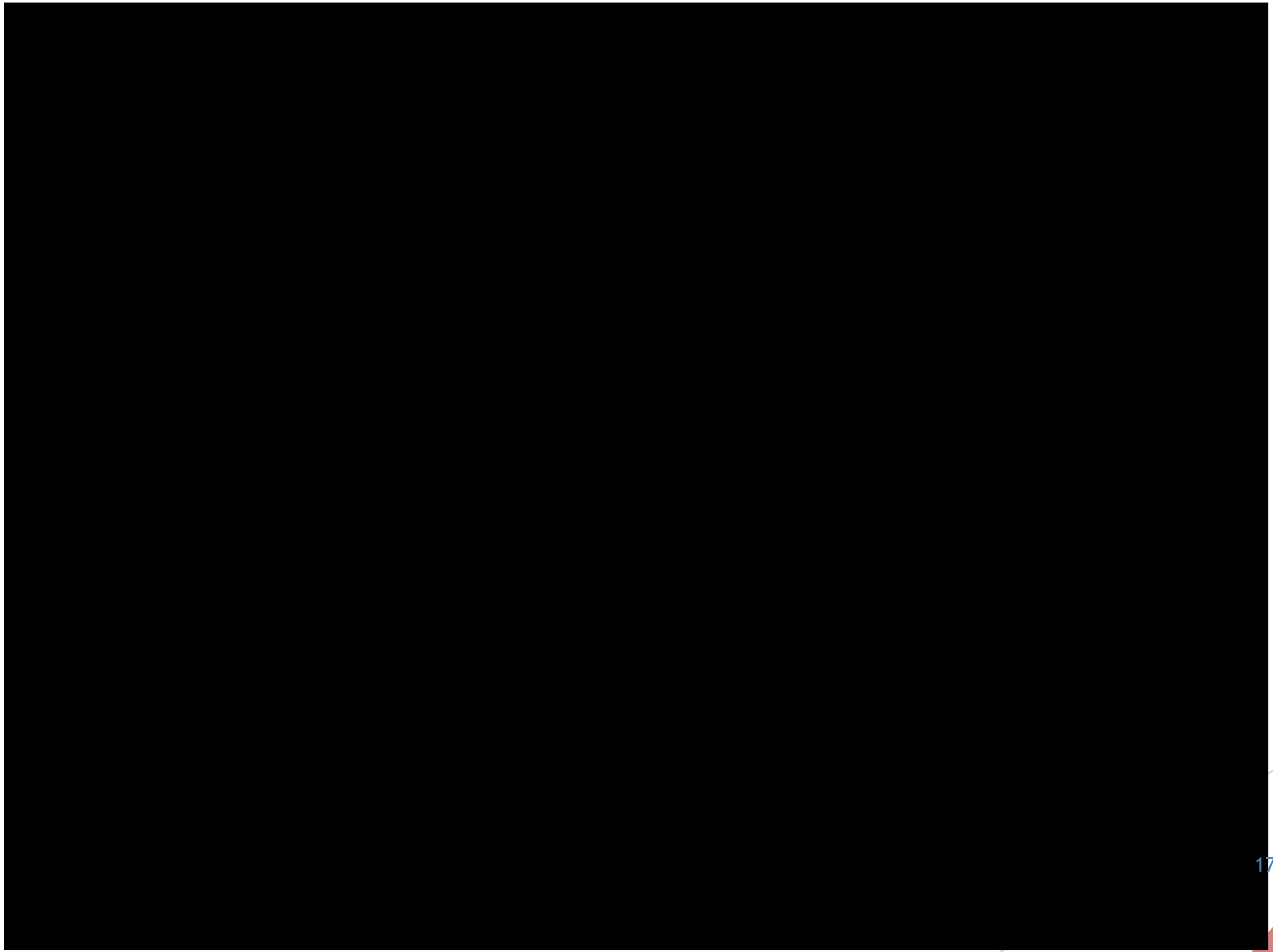


Results



Link to play: https://jeffjar.me/cartpole.html

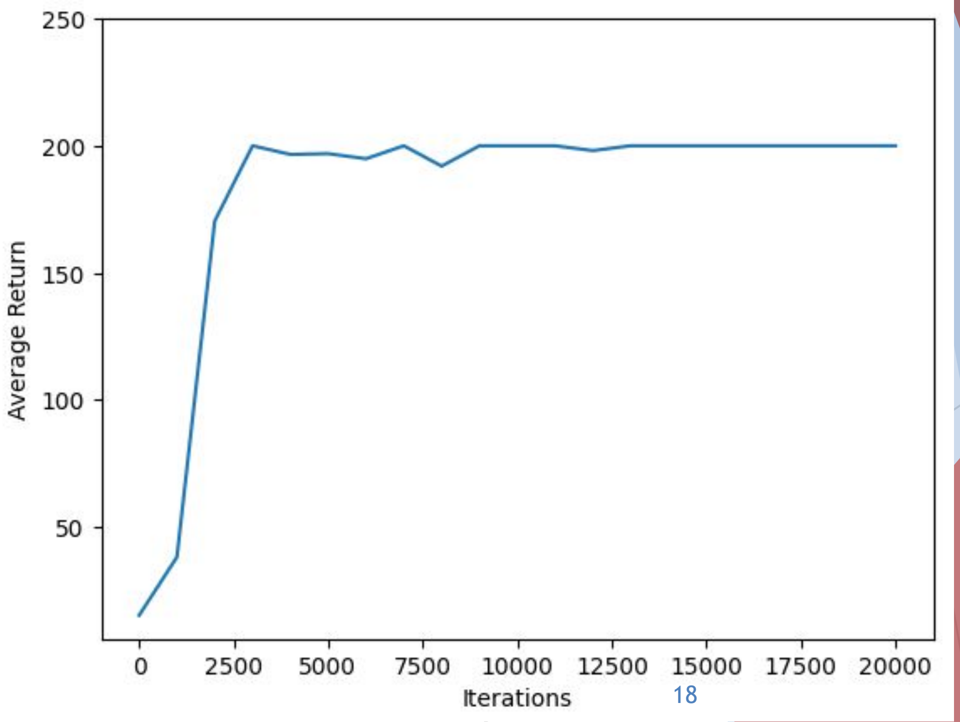
Training episode example of 167 steps



Experiment 1 Results (Reference 1)

DQN for cart-pole by TensorFlow

- observations as 4 dimensional vector [Position, Velocity, Angle, Angular Velocity]
- trained for 20,000 episodes (iterations)
- maximum reward is reached after 2500 episodes and stable after 10,000
- iteration = episode (200 steps maximum)
- number of steps = duration
- reward (=return), +1 for each step

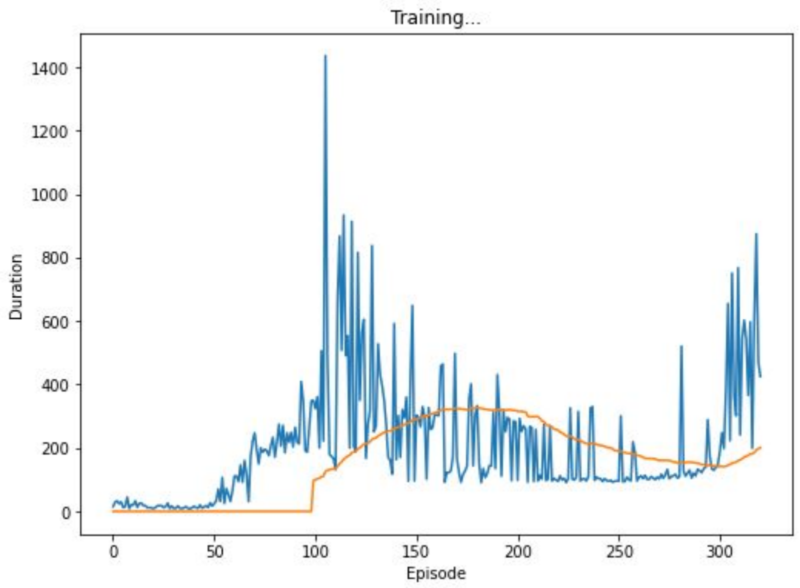


https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

Experiment 2 Results (Reference 2)

DQN - OpenAI Gym CartPole with PyTorch Kaggle notebook by dsxavier

- observations as vector (m=4, not a frame)
- trained for few hundreds episode 321
- average reward of 195 is reached 120 episodes



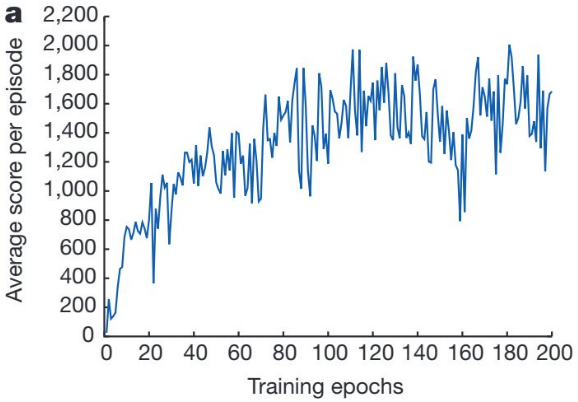

```
- Episodes: 321
- 100 episodes moving avg: 200.6199951171875
```


https://www.kaggle.com/code/dsxavier/dqn-openai-gym-cartpole-with-pytorch

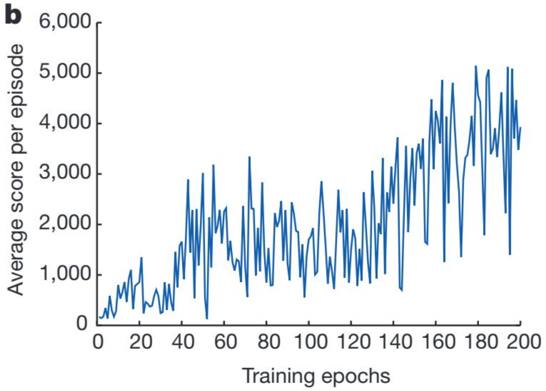
Main Paper (Reference 3)

“Human-level control through deep reinforcement learning”

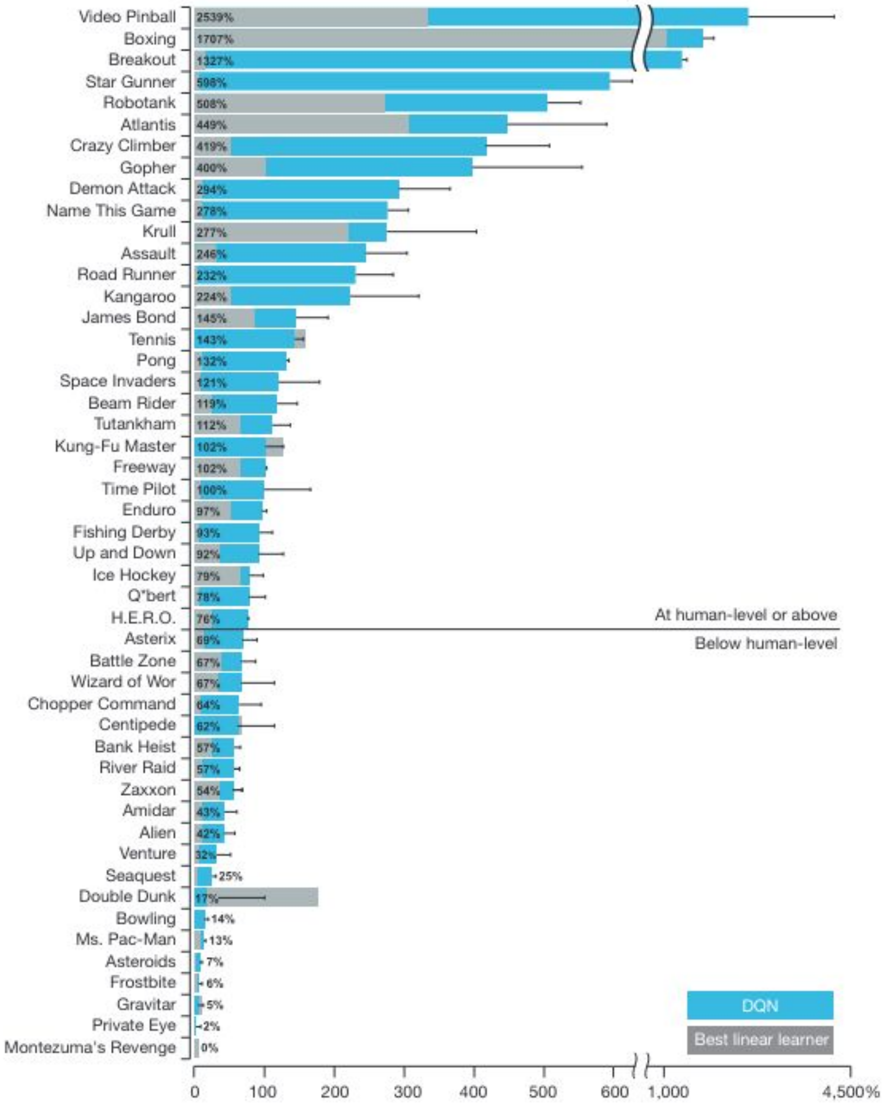
- Control through deep reinforcement learning
- Metric: Q-Value, game score
  - Normalized performance of DQN score:
    Score= 100 * (DQN score - random play score)/(human score -random play score)
  - Training score: Game score



Each point is the average score achieved per episode after the agent is run with e-greedy policy (e 5 0.05) for 520 k frames on Space Invaders game

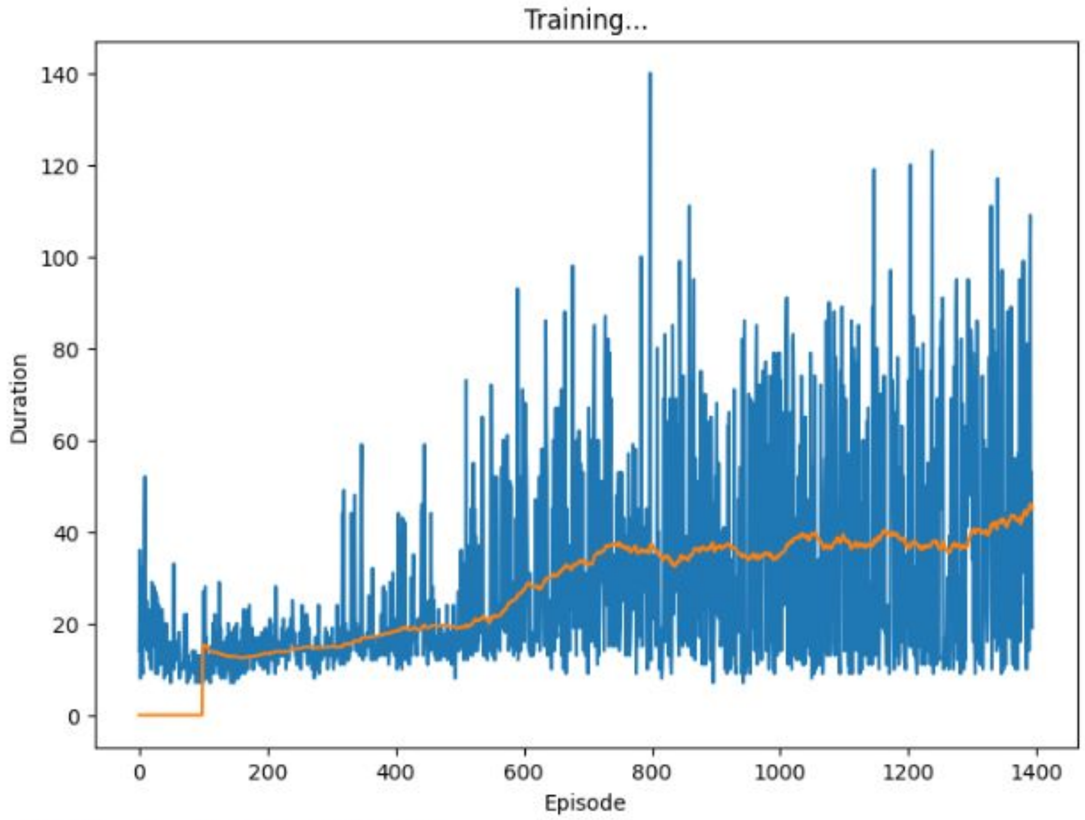


Average score achieved per episode for Seaquest game



Experiment 3

- The average results achieved so far are 22.5% of the maximum possible

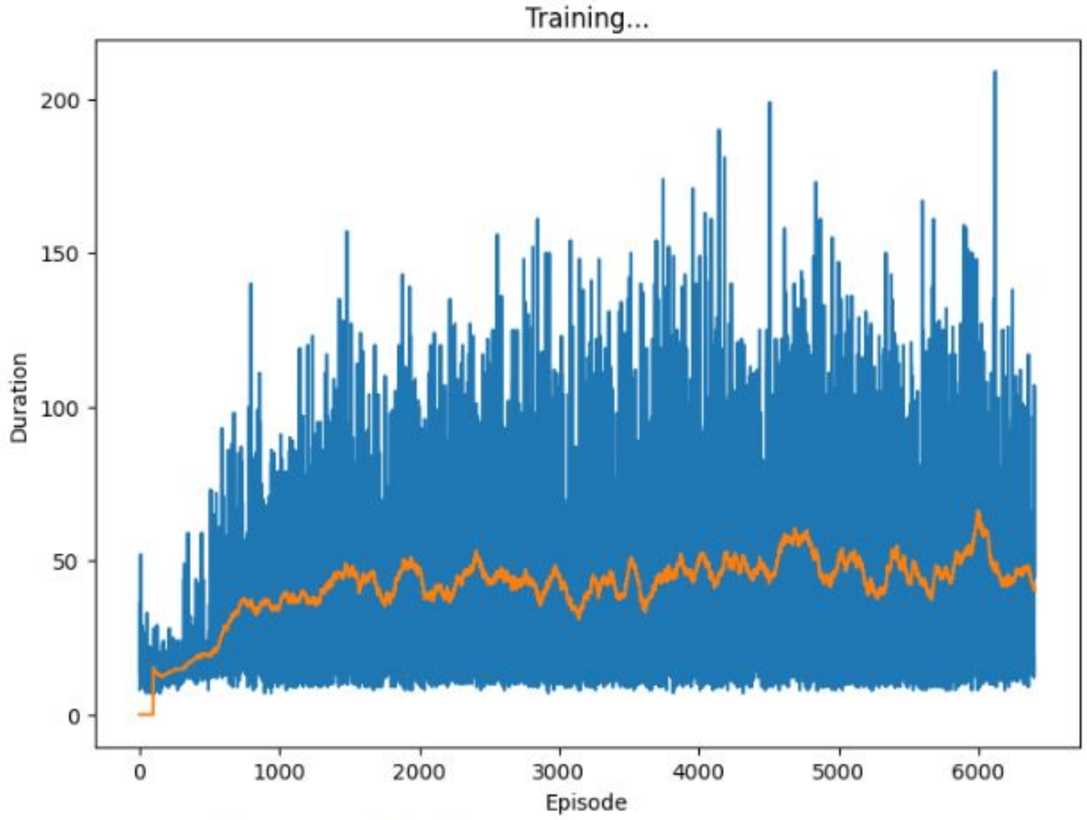



```
- Episode: 1394, Duration [steps]: 19
- 100 episodes moving avg: 44.97999954223633
- epsilon_rate: 0.01
```

```
DQN(
  (conv1): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=2816, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=2, bias=True)
)
batch_size:     256
gamma:          0.999
eps_start:      1
eps_end:        0.01
eps_decay:      0.001
target_update:  10
memory_size:    100000
lr:             0.001
num_episodes:   768

Duration of training: 11:28
```

Experiment 3




```
- Episode: 6400, Duration [steps]: 45
- 100 episodes moving avg: 40.5
- epsilon_rate: 0.01
```

```
DQN(
  (conv1): Conv2d(3, 32, kernel_size=(8, 8), stride=(4, 4), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
  (conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=2816, out_features=32, bias=True)
  (fc2): Linear(in_features=32, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=128, bias=True)
  (out): Linear(in_features=128, out_features=2, bias=True)
)
batch_size:     64
gamma:          0.999
eps_start:      1
eps_end:        0.01
eps_decay:      0.001
target_update:  10
memory_size:    100000
lr:             0.001
num_episodes:   6400

Duration of training: 113:26
```

Summary

- **Customisation of the DQN Methodology:** Originally developed for Atari game environments, the Deep Q-Network (DQN) was converted for the cart-pole control task.
- **High-Dimensional Input:** The environment was modified to output video frames of 40x90 resolution
- **Agent:** Successfully received video frames and Convolution layers were added to the DQN in order to process the high dimension input.
- **Comparative Performance Analysis:** Despite these enhancements, the convolutional DQN achieved a maximum average reward of 44.9, which is 22.5% of the potential maximum for the Cart-Pole-v1 environment.
- **Benchmarking Against Reference Models:** Reference models demonstrated superior performance, achieving higher rewards and requiring fewer training episodes to converge.

Conclusion

- **Environment adaptation:** In this work a system designed to play atari games was modified for solving control task.
- **Challenges with High-Dimensional Complexity:** The solution for the control task achieved results obtained was lower than simpler methods based on lower dimensional environments.

Further improvements:

- The high dimensional complexity was challenging for the system and it didn't excel in comparison to the references of DQN without convolution.
- Hyper parameter fine tuning and Architecture improvement are needed
- More experiments are needed to be done on other control environments to determined if the model is robust across a range of control tasks.

Referenced and Related Work

1. Reference(1): tensorflow implementation of DQN for “cart pole” control environment https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial
2. Reference(2): dsxvier library for DQN https://www.kaggle.com/code/dsxavier/dqn-openai-gym-cartpole-with-pytorch
3. Reference(3): main paper for the work https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf
4. Reference(4): environment documentation https://www.gymlibrary.dev/environments/classic_control/cart_pole/
5. Reference(5): DQN with high dimensionality input over space invader environment https://www.kaggle.com/code/yaaryan/space-invaders-game-using-deep-q-networks


THANK YOU