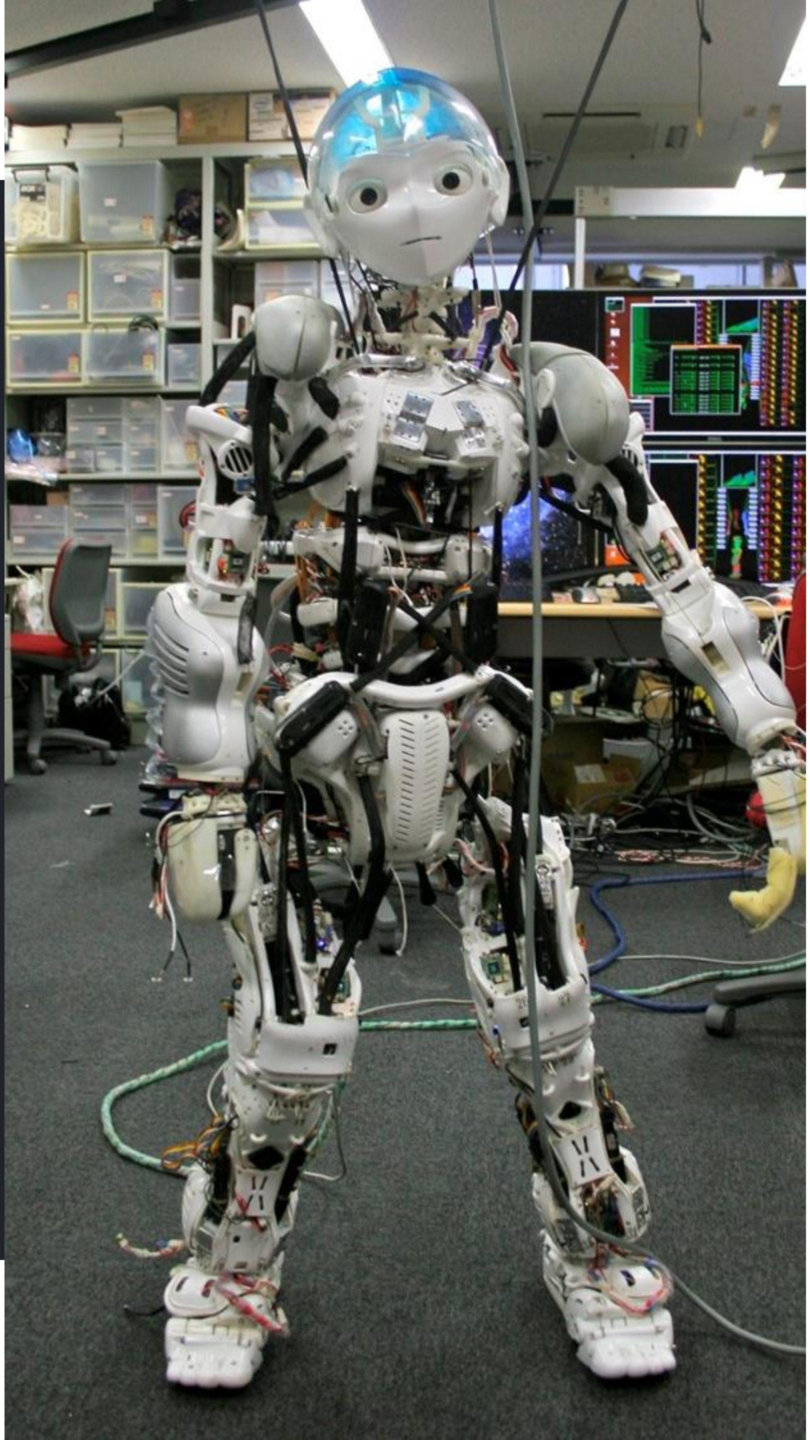
CS6482 Deep RL

G: DQN for Classic Control

J.J. Collins

Dept. of CSIS

University of Limerick



Objectives

- Understand the TD update applied to TD Gammon
- Define the CartPole Problem
- Work through a Deep Q Network Solution to

Outline

TD Gammon

OpenAI Gym: DQN for Classic Control – the CartPole problem

Challenges with DQN

Based on excerpts from:

- Chapter 16 in Sutton and Barto. Reinforcement Learning: an Introduction, 2nd Edition. The MIT Press. 2018.
- Chapter 11 in Gulli, Kapoor, and Pal. Deep Learning with TensorFlow 2 and Keras, 2nd Ed. Packt Birmingham. 2020.
- Chapter 18 in Aurelin Geron. Hands on Machine Learning (3rd Edition). O'Reilly. 2021.

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Q update for windy grid world

State	Q(Cell,North)	Q(Cell,South)	Q(Cell,East)	Q(Cell,West)
Cell 1	25	25	40	10
Cell 2	20	20	25	35

- $Q(s,a) = Q(s,a) + \alpha\left[r + \gamma max_{a'}\left(Q(s',a')\right) - Q(s,a)\right]$
- Agent is in cell 1
- Selects maximising action move east results in a transition to cell 2, and receives intermediate reward 0
- $Q(Cell1, East) = 40 + \alpha[0 + \gamma(35 - 40)]$
- Let discount = 1 and step size = 0.5
- $Q(Cell1, East) = 40 + 0.5[0 + 1(35 - 40)]$
- Q(Cell1, East) = 37.5

On Policy versus Off-Policy (Sutton and Barto, 2018)

"All learning control methods face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy? The on-policy approach in the preceding section is actually a compromise—it learns action values not for the optimal policy, but for a near-optimal policy that still explores. A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the target policy, and the policy used to generate behavior is called the behavior policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed ff-policy learning."

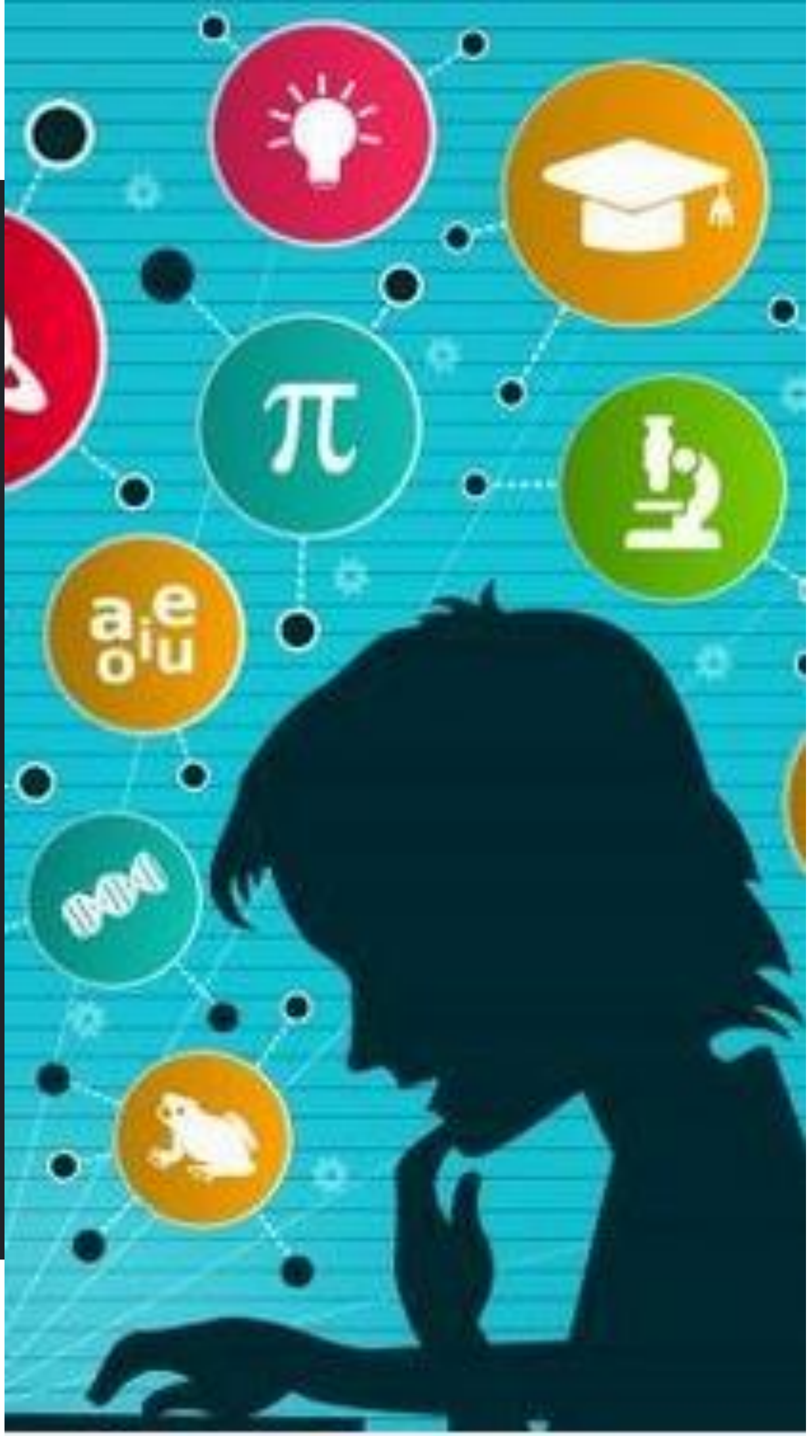
UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

On versus Off Policy

StackExchange

- The reason that Q-learning is off-policy is that it updates its Q-values using the Q-value of the next state s' and *greedy action a'*
- It estimates the *return* (total discounted future reward) for state-action pairs assuming a greedy policy were followed despite the fact that it's not following a greedy policy.
- The reason that SARSA is on-policy is that it updates its Q-values using the Q-value of the next state s' and the *current policy's* action a^n. It estimates the return for state-action pairs assuming the current policy continues to be followed.
  - https://stats.stackexchange.com/questions/184657/what-is-the-difference-between-off-policy-and-on-policy-learning

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH



On versus Off Policy

“On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy dfferent from that used to generate the data.”

Page 100, Sutton and Barto. 2018.

"An off-policy learner learns the value of the optimal policy independently of the agent's actions. Q-learning is an off-policy learner. An on-policy learner learns the value of the policy being carried out by the agent including the exploration steps.“

http://artint.info/html/ArtInt_268.html

Deep Q-Networks

- Proposed by Google's DeepMind team in NIPS 2013 paper "Playing Atari with Deep Reinforcement Learning"
- Used raw state space as input into the network
- Not handcrafted as in TD Gammon
- Separate output for each possible action
- Could use the same architecture to train on different Atari games
- Network predicts target $Q_{target} = r + \gamma \, max_a Q(s', a)$
- Loss function reduces error between predicted and target
- $loss = E_{\pi}\left[Q_{target}(s,a) - Q_{predicted}(s,w,a)\right]$
  - where w = is the training parameters

CartPole-vx

A Pole attached by a single joint to a Cart

Cart moves along frictionless track

Goal: keep pole standing upright by moving the cart left or right

Reward +1 for each time step

Game over if

- Pole more than 15 degrees from the vertical
- The cart moves more than 2.4 units from the centre

Sample Code

- https://gym.openai.com/envs/CartPole-v0/
- OpenAI Gym: game is solved if pole is in vertical position for 200 time ticks i.e. reward of 200

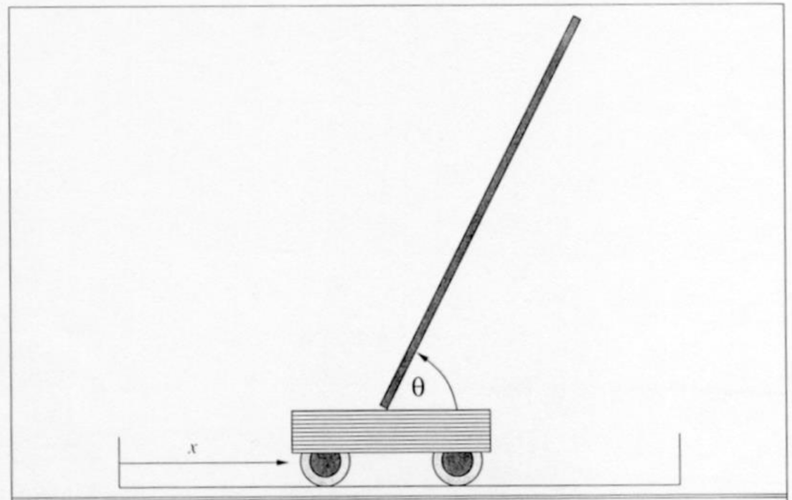


Figure 20.14 Setup for the problem of balancing a long pole on top of a moving cart. The cart can be jerked left or right by a controller that observes x, θ, $\dot{x}$, and $\dot{\theta}$.

Obs (State) in Gym environment

1. Cart Position: [-2.4, …. +2.4]

2. Cart velocity: [

3. Pole angle:

4. Pole Velocity

Deterministic Control (from Geron, 2021).

```
L1: def basic_policy(obs):
L2:     angle = obs[2]
L3:     return 0 if angle < 0 else 1
```

If pole falling left then accelerate left
Else accelerate right

```
L4: totals = []
If
L5: for episode in range(500):
L6:     episode_rewards = 0
L7:     obs, info = env.reset(seed=episode)
L8:     for step in range(200):
L9:        action = basic_policy(obs)
L10;        obs, reward, done, truncated, info = env.step(action)
L11:         episode_rewards += reward
L12:         if done or truncated:
L13:            break
L14:    totals.append(episode_rewards)

L16: import numpy as np
L17: np.mean(totals), np.std(totals), np.min(totals), np.max(totals)
```

- (41.698, 8.389445512070509, 24.0, 63.0)

Simple Neural Network Policy (from Geron 2021)

```
L1: import tensorflow as tf

L2: tf.random.set_seed(42)

L3: model = tf.keras.Sequential([
L4:    tf.keras.layers.Dense(5, activation="relu"),
L5:    tf.keras.layers.Dense(1, activation="sigmoid"),
L6: ])

L7: def pg_policy(obs):
L8:    left_proba = model.predict(obs[np.newaxis], verbose=0)[0][0]
L9:    # exploration versus exploitation in L10
L10:    return int(np.random.rand() > left_proba)

L11: np.random.seed(42)
L12: # show_one_episode is user defined method – See Geron chapter 18.
L13: show_one_episode(pg_policy)
```

Outputs the probabilities of actions.

CartPole: two possible actions (left or right), so only need one output: outputs the probability `p` of the action 0 (left), the probability of action 1 (right) will be `1 - p`.

The CartPole: One of Many Solutions (From Guilli et al 202)

```
import gym
import ……

EPOCHS = 1000
Threshold = 45 #should be 600
Monitor = False
```

CartPole-vo Solution

- Model: 1418 params
- Memory is a buffer tha stores experience <s,a,r,s'>

```
class DQN():
    def __init__(self, env_string,batch_size=64):
        self.memory = deque(maxlen=100000)
        self.env = gym.make(env_string)
        input_size = self.env.observation_space.shape[0]
        action_size = self.env.action_space.n
        self.batch_size = batch_size
        self.gamma = 1.0
        self.epsilon = 0.1
        alpha=0.01

        # Init model
        self.model = Sequential()
        self.model.add(Dense(24, input_dim=input_size, activation='tanh'))
        self.model.add(Dense(48, activation='tanh'))
        self.model.add(Dense(action_size, activation='linear'))
        self.model.compile(loss = 'mse', optimizer=Adam(lr = alpha))
```

CartPole-v0 Solution

- Methods to source random samples from memory in batches and for storing experience

```
def remember(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))
```

```
def replay(self, batch_size):
    x_batch, y_batch = [], []
    minibatch = random.sample(self.memory, min(len(self.memory), batch_size))
    for state, action, reward, next_state, done in minibatch:
        y_target = self.model.predict(state)
        y_target[0][action] = reward if done else reward + self.gamma * np.max(self.model.predict(next_state)[0])
        x_batch.append(state[0])
        y_batch.append(y_target[0])
    self.model.fit(np.array(x_batch), np.array(y_batch), batch_size=len(x_batch), verbose=0)
```

CartPole-Solution

- Action selection using $\epsilon - greedy$

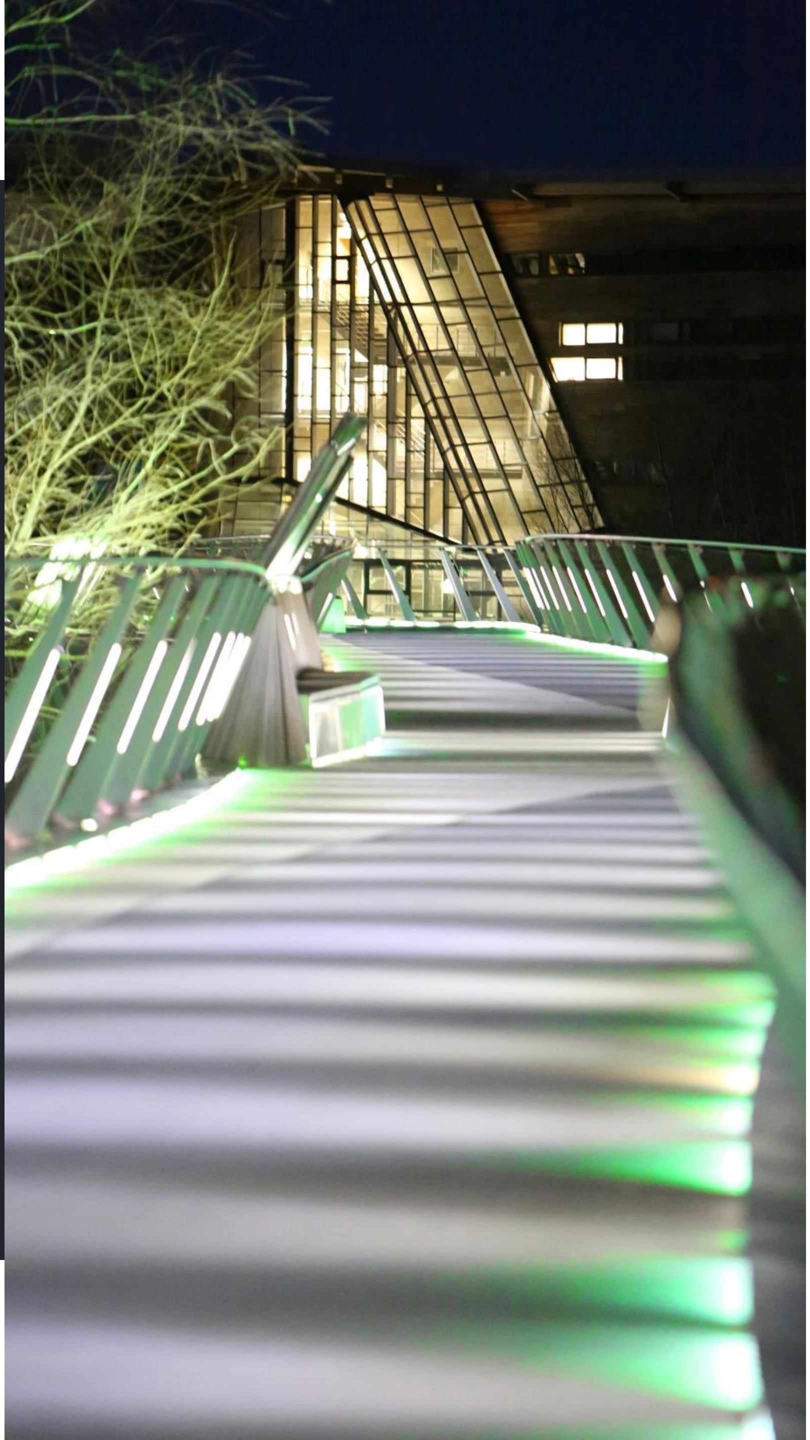
```
def choose_action(self, state, epsilon):
    if np.random.random() <= epsilon
        return seld.env.action_space.sample()
    else:
        return np.argmax(self.model.predict(state))
```

```
def preprocess_state(self, state):
    return np.reshape(state, [1, 4])
```

CartPole-v0 Solution

- Training

```
def train(self):
    scores = deque(maxlen=100)
    avg_scores = []
    for e in range(EPOCHS):
        state = self.env.reset()
        state = self.preprocess_state(state)
        done = False
        i = 0
        while not done:
            action = self.choose_action(state,self.epsilon)
            next_state, reward, done, _ = self.env.step(action)
            next_state = self.preprocess_state(next_state)
            self.remember(state, action, reward, next_state, done)
            state = next_state
            i += 1
        scores.append(i)
        mean_score = np.mean(scores)
        avg_scores.append(mean_score)
        if mean_score >= THRESHOLD and e >= 100:
            return avg_scores
        self.replay(self.batch_size)
    print('Did not solve after {} episodes'.format(e))
    return avg_scores
```



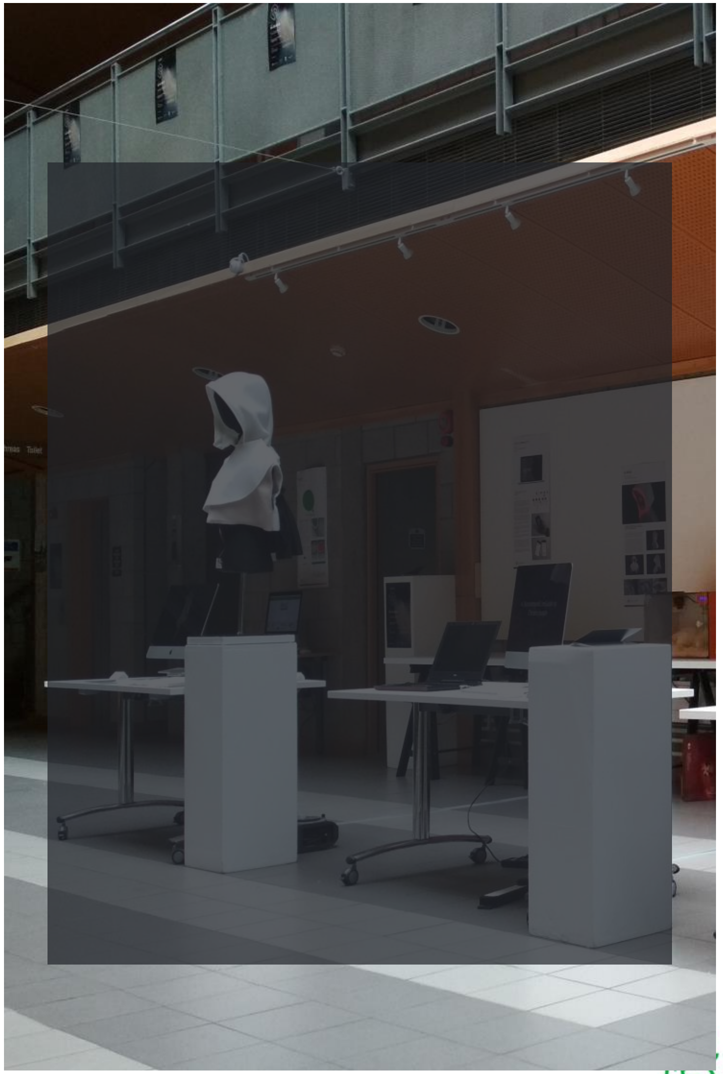
CartPole-v0 Solution

```
env_string = ‘CartPole-v0’
RL_agent = DQN(env_string)
scores = RL_agent.train()
```

- Please plot results
- Exercise:
  - Decay alpha
  - Decay epsilon
  - Make a movie of the cart

Another DQN Implementation

Code Inspection of Geron's DQN implementation for CartPole



UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

Geron: Buyer Beware

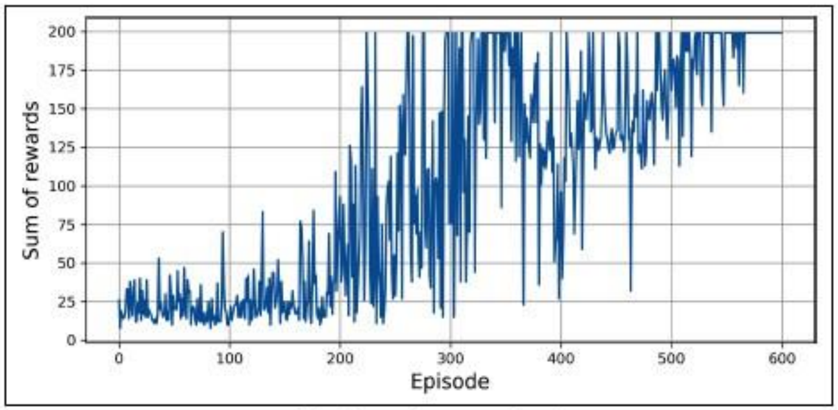


Figure 18-10. Learning curve of the deep Q-learning algorithm

As you can see, the algorithm took a while to start learning anything, in part because ε was very high at the beginning. Then its progress was erratic: it first reached the max reward around episode 220, but it immediately dropped, then bounced up and down a few times, and soon after it looked like it had finally stabilized near the max reward, at around episode 320, its score again dropped down dramatically. This is called *catastrophic forgetting*, and it is one of the big problems facing virtually all RL algorithms: as the agent explores the environment, it updates its policy, but what it learns in one part of the environment may break what it learned earlier in other parts of the environment. The experiences are quite correlated, and the learning environment keeps changing—this is not ideal for gradient descent! If you increase the size of the replay buffer, the algorithm will be less subject to this problem. Tuning the learning rate may also help. But the truth is, reinforcement learning is hard: training is often unstable, and you may need to try many hyperparameter values and random seeds before you find a combination that works well. For example, if you try changing the activation function from `"elu"` to `"relu"`, the performance will be much lower.

Reinforcement learning is notoriously difficult, largely because of the training instabilities and the huge sensitivity to the choice of hyperparameter values and random seeds.[13] As the researcher Andrej Karpathy put it, "[Supervised learning] wants to work. [...] RL must be forced to work". You will need time, patience, perseverance, and perhaps a bit of luck too. This is a major reason RL is not as widely adopted as regular deep learning (e.g., convolutional nets). But there are a few real-world applications, beyond AlphaGo and Atari games: for example, Google uses RL to optimize its datacenter costs, and it is used in some robotics applications, for hyperparameter tuning, and in recommender systems.

Geron, 2021.

Combatting Catastrophic Forgetting and Maximisation Bias and Instability

Fixed Q Targets

Vanilla DQN for Classic Control (from Geron 2021)

```python
batch_size = 32
discount_factor = 0.95
optimizer = tf.keras.optimizers.Nadam(learning_rate=1e-2)
loss_fn = tf.keras.losses.mean_squared_error

def training_step(batch_size):
    experiences = sample_experiences(batch_size)
    states, actions, rewards, next_states, dones, truncateds = experiences
    next_Q_values = model.predict(next_states, verbose=0)
    max_next_Q_values = next_Q_values.max(axis=1)
    runs = 1.0 - (dones | truncateds)  # episode is not done or truncated
    target_Q_values = rewards + runs * discount_factor * max_next_Q_values
    target_Q_values = target_Q_values.reshape(-1, 1)
    mask = tf.one_hot(actions, n_outputs)
    with tf.GradientTape() as tape:
        all_Q_values = model(states)
        Q_values = tf.reduce_sum(all_Q_values * mask, axis=1, keepdims=True)
        loss = tf.reduce_mean(loss_fn(target_Q_values, Q_values))

    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Fixed Q Targets (from Geron 2021)

In vanilla DQN, same network used for predictions and estimating targets
Feedback loop leads to instability

Use 2 DQNs
Online model learns at each step and is used to select actions
Target network used only to define the targets

```
L1: target = keras.models.clone_model(model)
L2: target.set_weights(model.get_weights))
```

Make one change in training_step():

```
L3: Next_Q_values = target.predict(next_states)
```

And copy weights from model network to target network at set intervals

```
L4: if episodes % interval = 0;
L5: target.set_weights(model.get_weights())
```

- 2013 paper, interval set to 10K

Double DQNs (from Geron 2021)

- Target network tends to overestimate Q values.
- Always select a Q value that is slightly larger than the true value
- Use the online model to select the best actions for the next states and use the target model only to estimate Q values for these best actions

```
L1: keras.backend.clear_session()
L2: tf.random.set_seed(42)
L3: np.random.seed(42)

L4: model = keras.models.Sequential([
L5:     keras.layers.Dense(32, activation="elu", input_shape=[4]),
L6:     keras.layers.Dense(32, activation="elu"),
L7:     keras.layers.Dense(n_outputs)
L8:  ])
L9:  target = keras.models.clone_model(model)
L10: target.set_weights(model.get_weights())

L11: batch_size = 32
L12: discount_rate = 0.95
L13: optimizer = keras.optimizers.Adam(learning_rate=6e-3)
L14: loss_fn = keras.losses.Huber()
```

Double DQN (From Geron 2021)

```
def training_step(batch_size):
    experiences = sample_experiences(batch_size)
    states, actions, rewards, next_states, dones = experiences
    next_Q_values = model.predict(next_states)
    best_next_actions = np.argmax(next_Q_values, axis=1)
    next_mask = tf.one_hot(best_next_actions, n_outputs).numpy()
    next_best_Q_values = (target.predict(next_states) * next_mask).sum(axis=1)
    target_Q_values = (rewards + (1 - dones) * discount_rate * next_best_Q_values)
    target_Q_values = target_Q_values.reshape(-1, 1)
    mask = tf.one_hot(actions, n_outputs)
    with tf.GradientTape() as tape:
        all_Q_values = model(states)
        Q_values = tf.reduce_sum(all_Q_values * mask, axis=1, keepdims=True)
        loss = tf.reduce_mean(loss_fn(target_Q_values, Q_values))
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

Reinforcement Learning is not Easy.

- Alex IPran. Deep Reinforcement Learning Doesn't Work Yet. 2018. https://www.alexirpan.com/2018/02/14/rl-hard.html [Accessed March 2025].

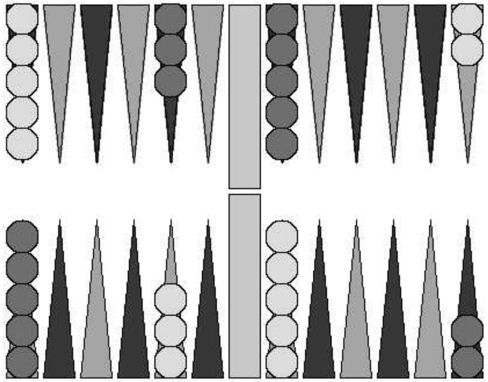
Mountain Car for classic control difficult to solve using standard methods

- The environment is solved by achieving -110 average over 100 episodes.
- https://www.reddit.com/r/reinforcementlearning/comments/cd4gk6/dqn_mountain_car/
- If using Mountain Car for Etivity3, don't fret/worry about reward. Demonstarte (a) solid understanding of the code with added value in the form of insightful plots, heatmaps, etc. (b) do a little research to source reputable material on reasons for poor performance and solutions that potentially address these issues, (c) if and only if practibable, have a go at one of the easier solutions i.e. Fixed Q Target or Double DQN but not tiling or Sarsa(Lambda) or Q(lambda).

OpenAI Gym Leaderboard:

- https://github.com/openai/gym/wiki/Leaderboard

History: TD Gammon (Tesauro, 1995)



- Reinforcement Learning for backgammon using Temporal Difference (TD) learning.
- Instead of using a look up table that had a value associated with each possible board state, used a Multi-Layered Perceptron (MLP) to map board configurations to values
- MLP learns through self-play: 1.5M games
- Plays at the level of the best human players
- Outperforms Neurogammon - neural net trained by supervised learning

History: TD Gammon (Tesauro, 1995)

TITLE	CITED BY	YEAR
Temporal difference learning and TD-Gammon G Tesauro Communications of the ACM 38 (3), 58-68	3182	1995

- Input is the encoded board position
- 20 hidden units
- 1 output unit provides an estimate of V
- Uses backprop to update the weights
- $w_{t+1} = w_t + \alpha(r + V_{t+1} - V_t)z_t$ where $z_{t=} \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_t$
- $(V_{t+1} - V_t)$ is the temporal difference between the current and previous turn's board evaluations.
- $\nabla_w V_k$ is the gradient of the network output with respect to the weights
- λ is a heuristic parameter controlling the temporal credit assignment of how an error detected at a given time step feeds back to update previous estimates.
  - λ = 0, no feedback occurs beyond the current time step, while
  - λ = 1, the error feeds back without decay arbitrarily far in time.

Summary

TD Gammon was a significant milestone in the field of RL in which function approximation was successfully used to train an agent to human-level competence.

Worked through the code for DQN for CartPole-v0

Now, your turn to complete the missing method and conduct empirical studies.

Next DQN for Atari

Homework

1. **Work through the template for the Cart Pole from chapter 18 in Geron (2021).**

2. **Implement DQN for the Mountain Car problem.**

Thank you

UNIVERSITY OF LIMERICK
OLLSCOIL LUIMNIGH

University of Limerick,
Limerick, V94 T9PX,
Ireland.

Ollscoil Luimnigh,
Luimneach,
V94 T9PX, Éire.

+353 (0) 61 202020

ul.ie