# Introducere în Reinforcement Learning

# Cursul #7

Ștefan Iordache, Cătălina Iordache, Ciprian Păduraru

# Cuprins

# 01

# Q-Learning

# De ce să folosim Q-Learning?

- **Cel mai simplu algoritm, de înțeles și aplicat!**
- **Off-Policy & Model-Free!**
- **Idee simplă: Învățăm o politică care maximizează recompensa totală.**

# De ce se numește Q-Learning?

**Q = Quality** (cât de utilă este o acțiune pentru rezultate viitoare?)

# Formula magică!

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} - \underbrace{Q(s_t, a_t)}_{\text{current value}}\Big)}^{\text{temporal difference}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

# Schelet Q-Learning

- **Inițializarea tabelului Q cu valoare zero (peste tot), urmând să ajustăm valorile în pașii de antrenare.**

```python
import numpy as np

# state_size -> retine numarul de stari
# action_size -> retine numarul de actiuni posibile
Q = np.zeros((state_size, action_size))
```

# Schelet Q-Learning

- **Explorare & Exploatare**

```python
import random

epsilon = 0.3

if random.uniform(0, 1) < epsilon:
    """
    EXPLORARE: selectăm o acțiune aleatorie
    """
else:
    """
    EXPLOATARE: selectăm acțiunea cu valoare maximă
    Valoare maximă = recompensa cea mai mare
    """
```

# Schelet Q-Learning

- **Actualizare valori Q**

```
current_value = Q[state, action]
new_value = reward + gamma * np.max(Q[new_state, :])
temporal_difference = new_value - current_value

Q[state, action] = (1 - lr) * current_value + lr * temporal_difference
```

# Antrenare

```
%%time
"""Antrenare agent"""

import random
from IPython.display import clear_output

# Hiperparametri
lr = 0.1
gamma = 0.6
epsilon = 0.1

# Colectii epoci si penalizari
all_epochs = []
all_penalties = []

for i in range(1, 100001):
    state = env.reset()

    epochs, penalties, reward, = 0, 0, 0
    done = False

    while not done:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample() # Explorare
        else:
            action = np.argmax(q_table[state]) # Exploatare

        next_state, reward, done, info = env.step(action)

        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])

        new_value = (1 - lr) * old_value + lr * (reward + gamma * next_max)
        q_table[state, action] = new_value

        if reward == -10:
            penalties += 1

        state = next_state
        epochs += 1

    if i % 100 == 0:
        clear_output(wait=True)
        print(f"Episodul: {i}")

print("Antrenare finalizata.\n")
```

# Evaluare

```python
"""Evaluarea performanței Q-learning (după antrenare)"""

total_epochs, total_penalties = 0, 0
episodes = 100

for _ in range(episodes):
    state = env.reset()
    epochs, penalties, reward = 0, 0, 0

    done = False

    while not done:
        action = np.argmax(q_table[state])
        state, reward, done, info = env.step(action)

        if reward == -10:
            penalties += 1

        epochs += 1

    total_penalties += penalties
    total_epochs += epochs

print(f"Număr mediu de pași per episod: {total_epochs / episodes}")
print(f"Număr mediu de penalizări per episod: {total_penalties / episodes}")
```

# Hiperparametri - Descriere

- **Alpha** (**α**) – Learning rate. Trend descrescător pe parcursul învățării.
- **Gamma** (**γ**) – Valoare cu trend descrescător. Către finalul episoadelor preferăm să luăm în calcul tot mai mult recompensa imediat următoare în locul celor pe termen lung.
- **Epsilon** (**ε**) – Spre finalul etapei de învățare nu mai avem nevoie de explorare, ci de exploatare. Experimentarea repetată conduce către o valoare cu trend descrescător.

# Hiperparametri - Tuning

- **Grid search** – Foarte utilă la începutul experimentării cu orice algoritm de inteligență artificială.
- **Random search**
- ***Ray.tune: Hyperparameter Optimization Framework***

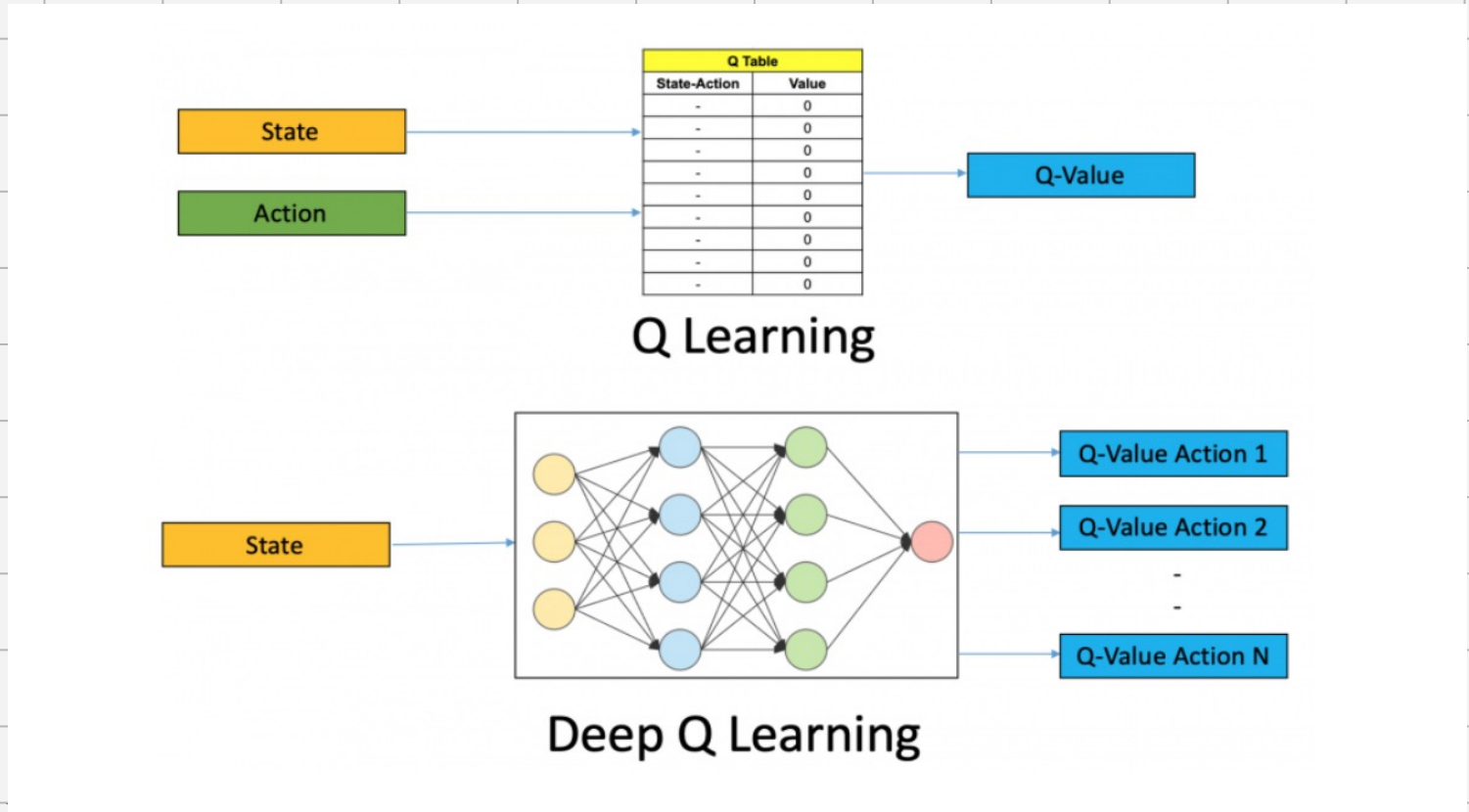
# 02

# Deep Q-Networks

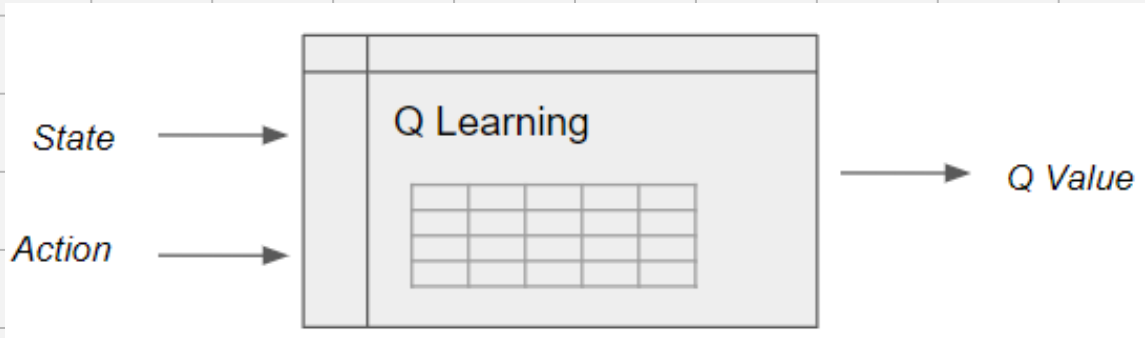
# De ce avem nevoie de mai mult?

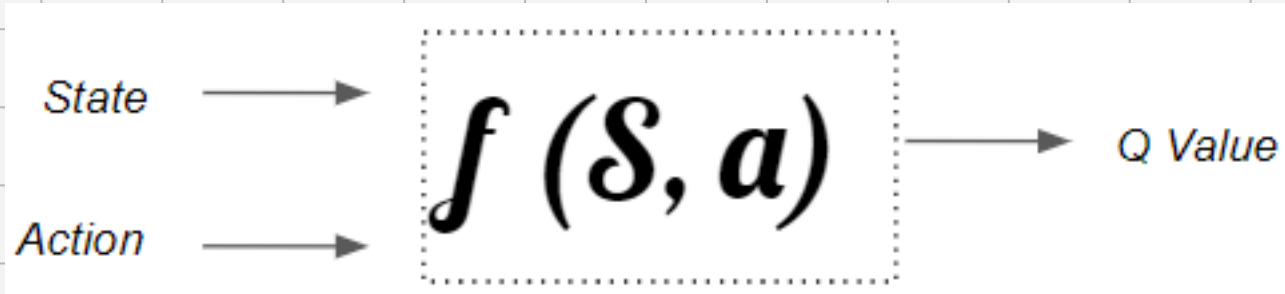
- **Mediile foarte mari sunt problematice!**
  - Necesarul de memorie pentru salvarea și actualizarea tabelului este foarte mare.
  - Timpul de explorare este nerealist.

# Cum funcționează DQN/DQL?



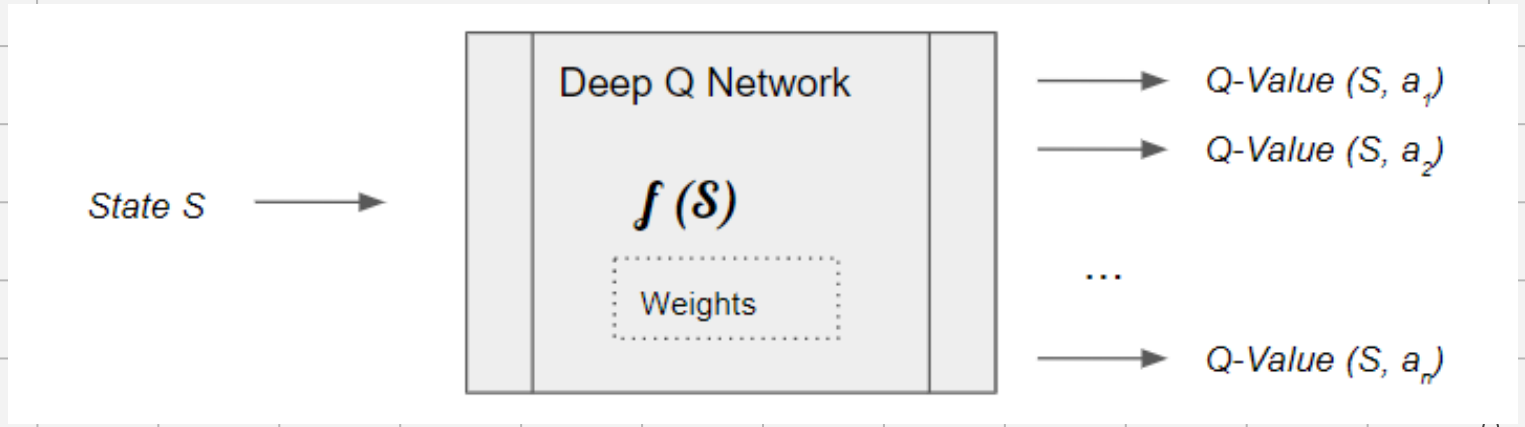
# Cum funcționează DQN/DQL?



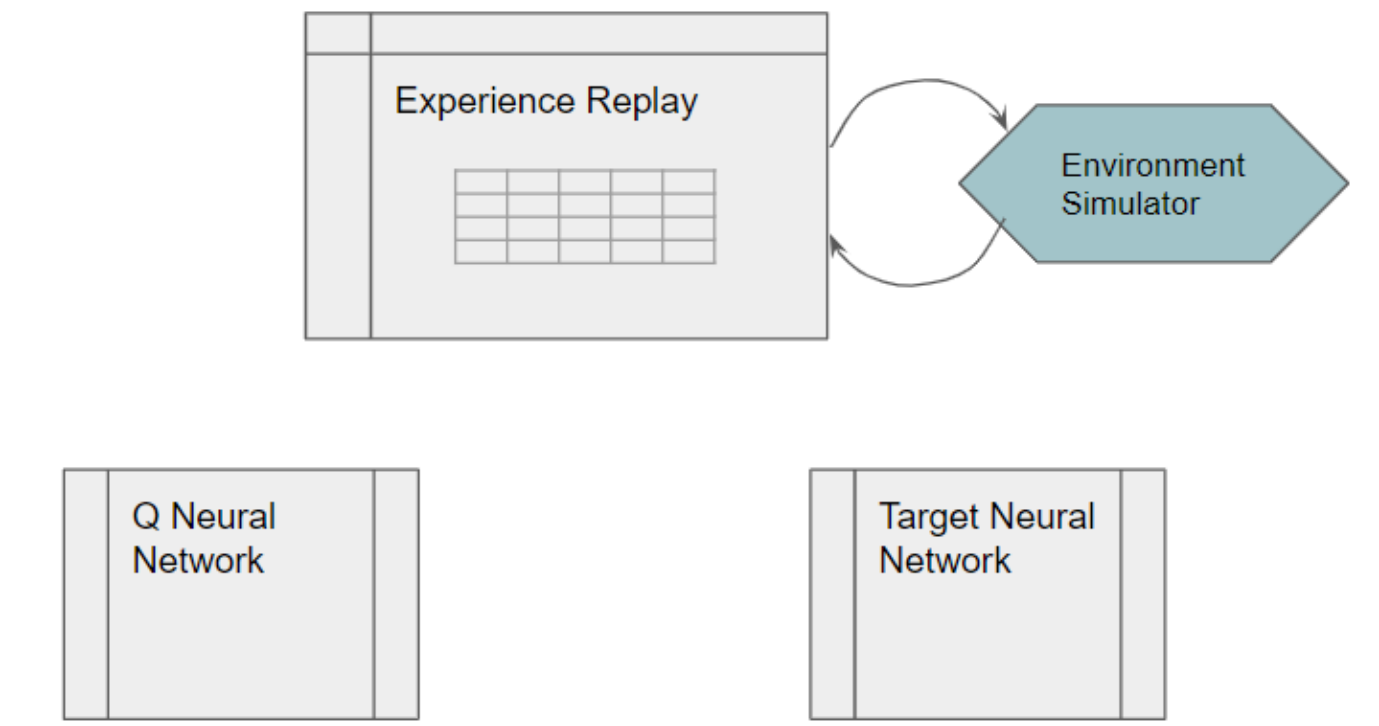



# Neural Networks

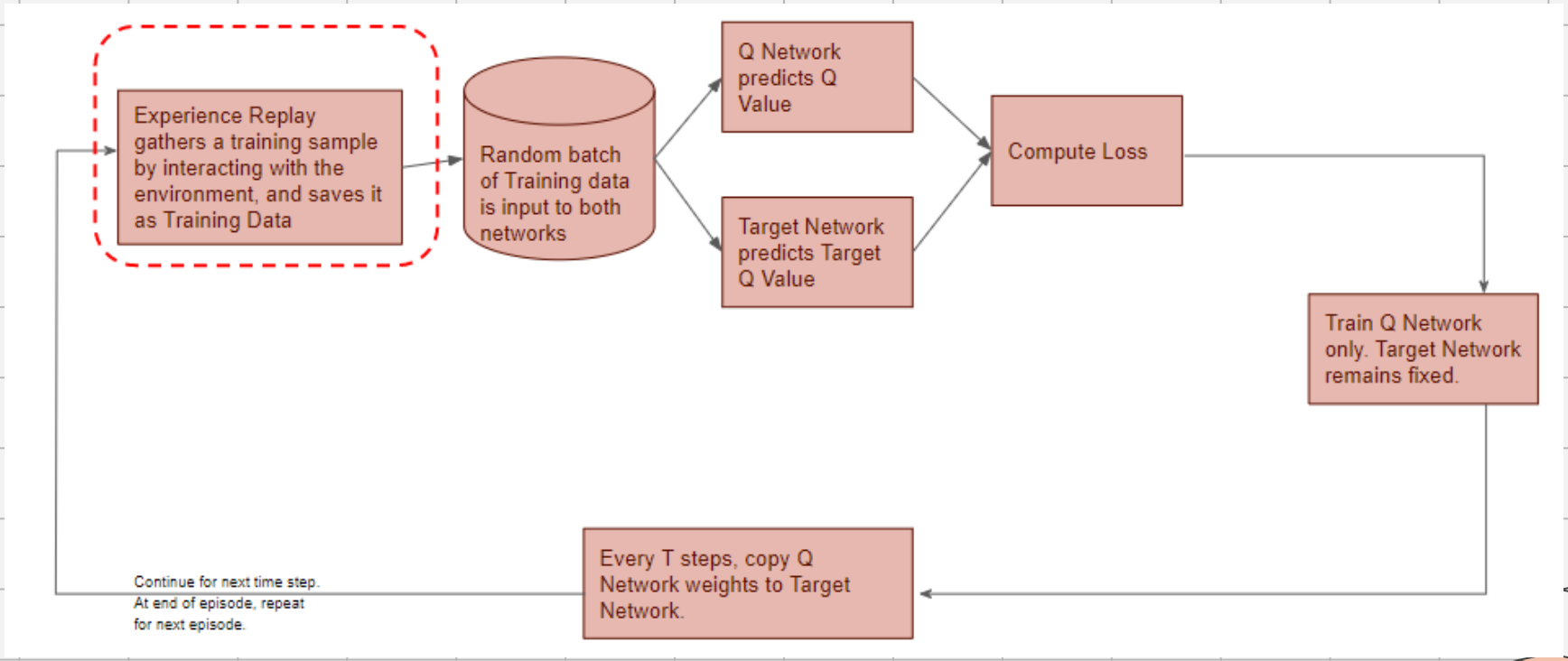
**Cea mai bună metodă de aproximare a funcțiilor complexe!**



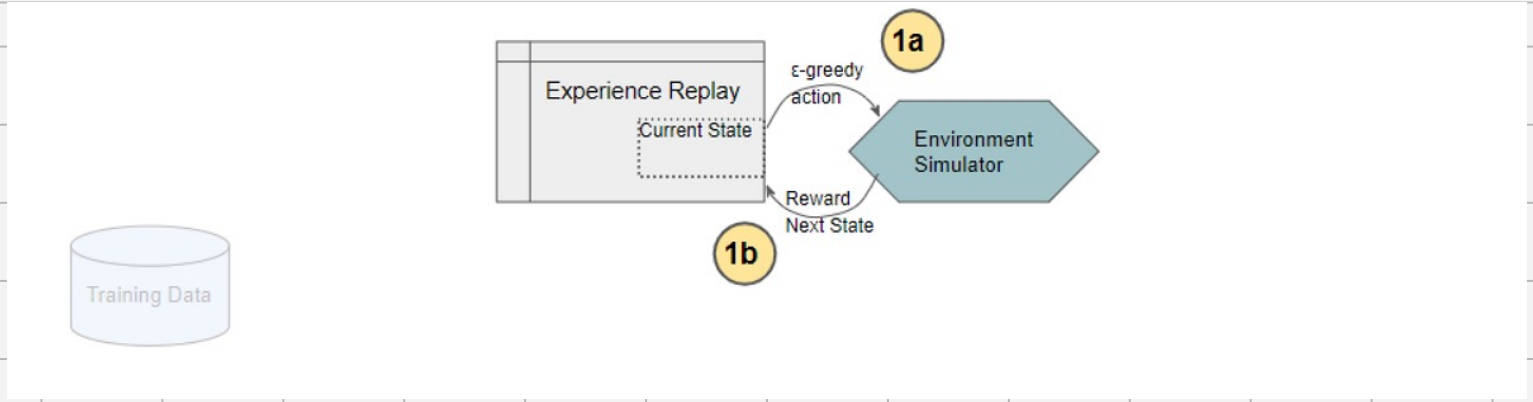
# Arhitectura DQN/DQL

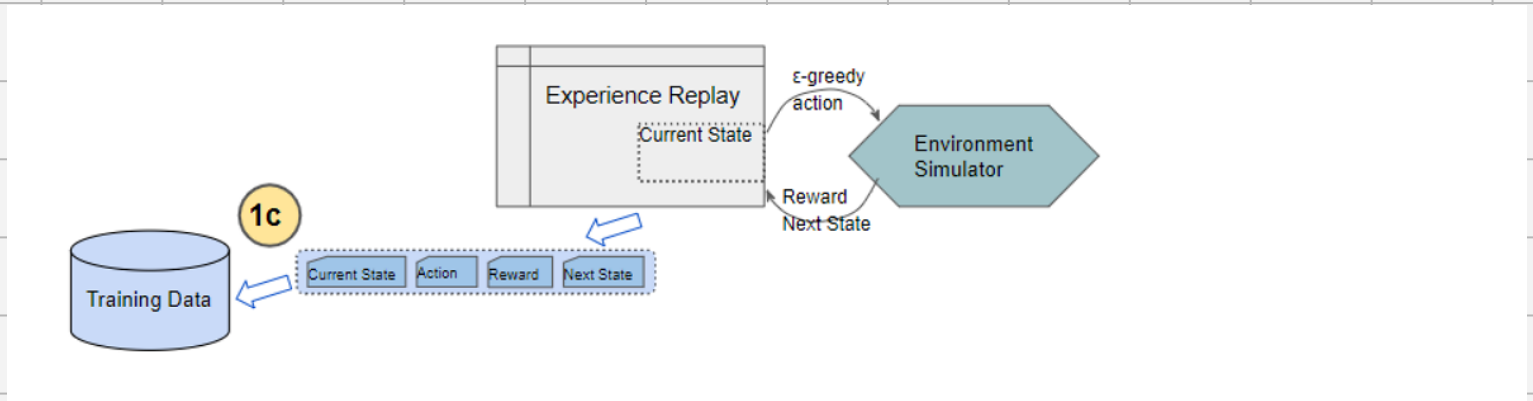


# Extragerea datelor de antrenare

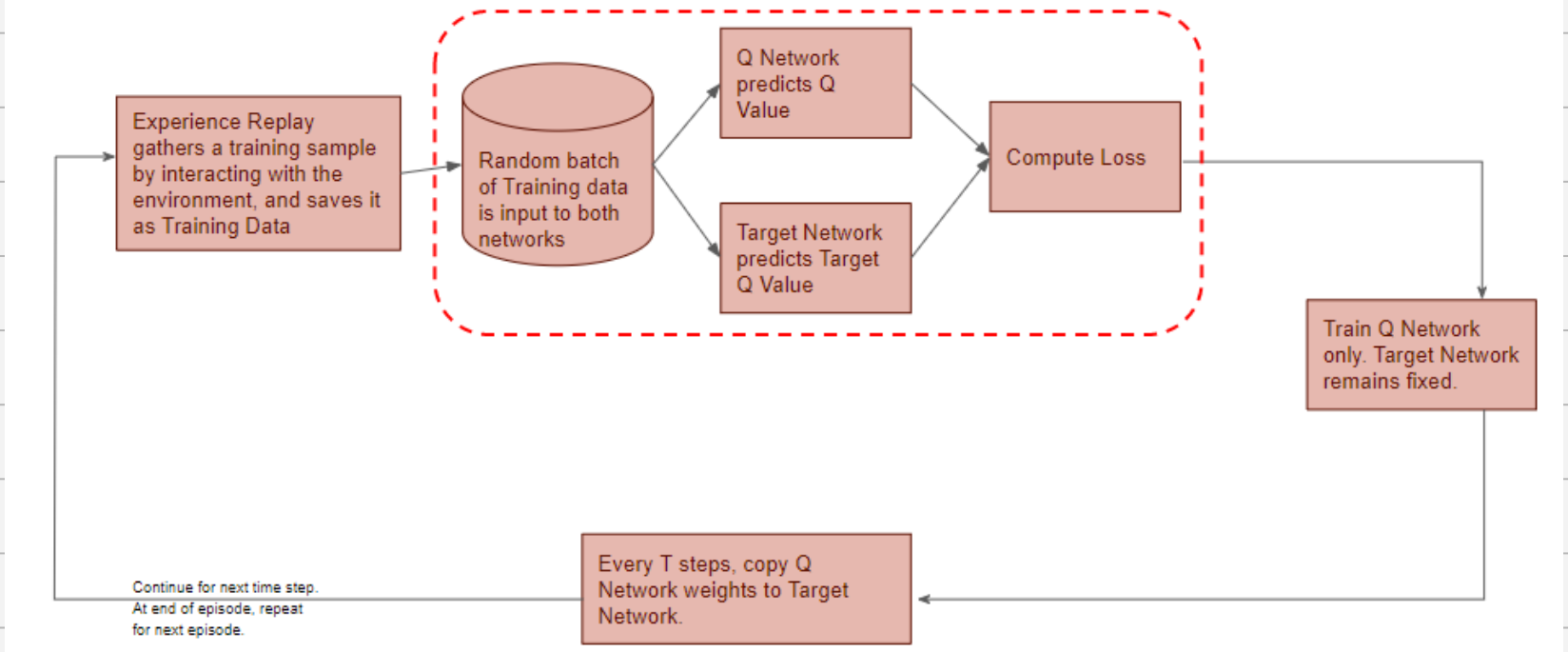


# Extragerea datelor de antrenare

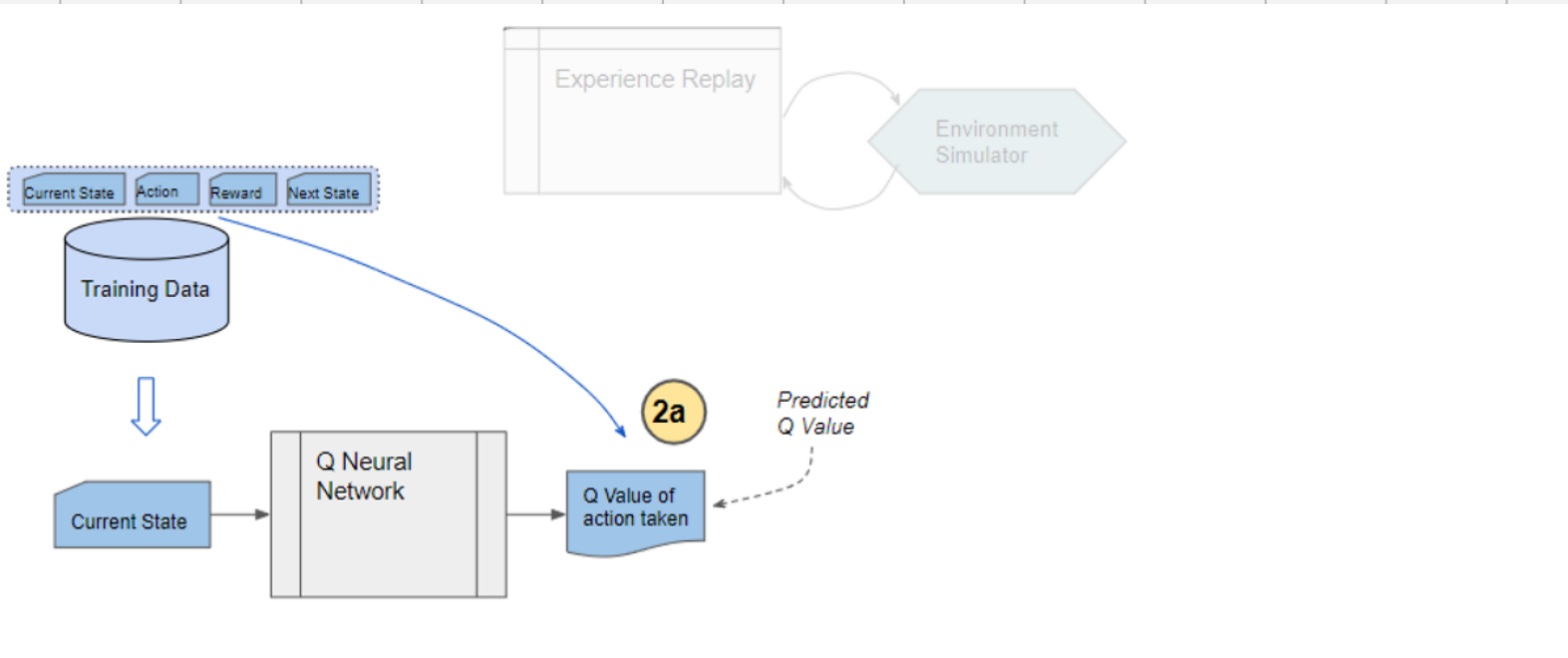


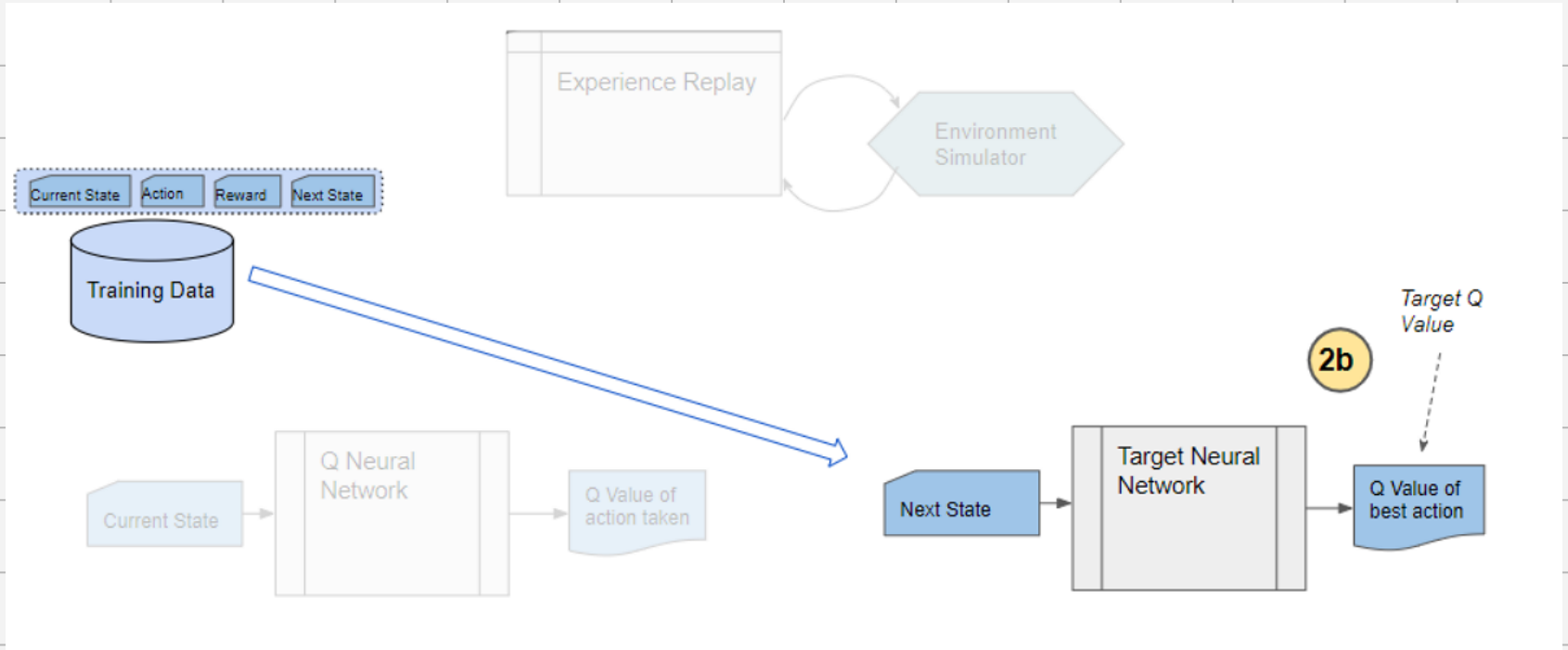
# Predicții


Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data
Random batch of Training data is input to both networks
Q Network predicts Q Value
Target Network predicts Target Q Value
Compute Loss
Train Q Network only. Target Network remains fixed.
Every T steps, copy Q Network weights to Target Network.
Continue for next time step.
At end of episode, repeat for next episode.

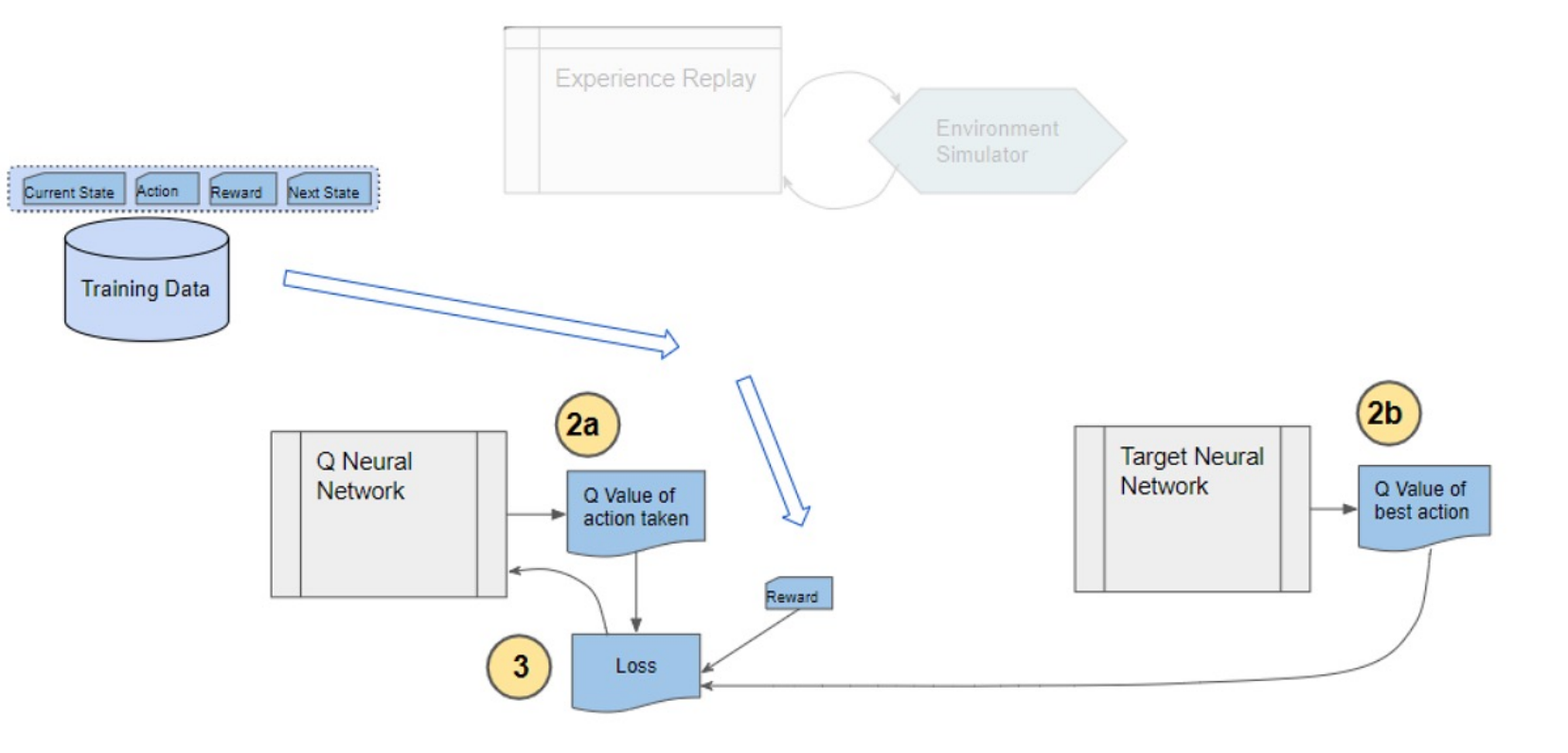
# Predicții – Q-value



# Predicții – Target Q-value



# Loss & Q-Network Train



# Thanks!

Este timpul pentru întrebări!!!

stefan.iordache10@s.unibuc.ro
catalina.patilea@s.unibuc.ro
ciprian.paduraru@fmi.unibuc.ro

+40 7.. ... ...