DQN Reinforcement Learning

Advancing Deep Q-Networks for Optimal Navigation in Frozen Lake:
Evaluating Adaptability and Performance in Dynamic Environments

Aviv Salomon
Ben Gornizky

TABLE OF CONTENTS

01

Introduction

History

- 1992 Q-learning algorithm introduced by Watkins
- 2013 DQN introduced by DeepMind in the paper 'Playing Atari with Deep Reinforcement Learning' at NIPS
- 2015 Comprehensive DQN paper 'Human-level control through deep reinforcement learning' published in Nature
- 2018 Rainbow DQN algorithm introduced, combining several improvements

DeepMind

Google DeepMind is a research company owned by Google that focuses on artificial intelligence (AI) and machine learning.

DQN In A Nutshell

DQN is a reinforcement learning algorithm that combines Q-Learning with deep neural networks to learn how to act in an environment.

- **Key Concepts:**
  - States: Environment configurations.
  - Actions: Possible moves.
  - Rewards: Feedback signals.
- **Applications:** Video games (Atari), robotics, finance (trading strategies).
- **Advantages:** Handles high-dimensional state spaces, effective for complex policy learning.
- **Challenges:** Requires extensive training data, tuning for stability.
- **Purpose:** It aims to enable an agent to make decisions that maximize cumulative rewards over time.

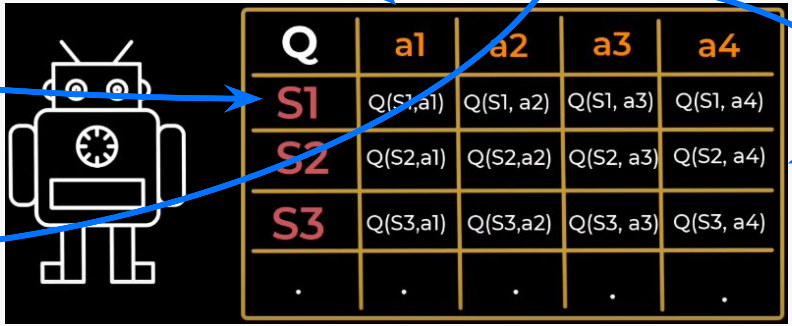
02

Architecture

Architecture

<u>RL agents policy (π):</u>

In each **State** the Agent know which **Action** to do, based on **Q-Table** to achieve the maximum **Reward**



The DQN replace the Q-table with a neural network instead

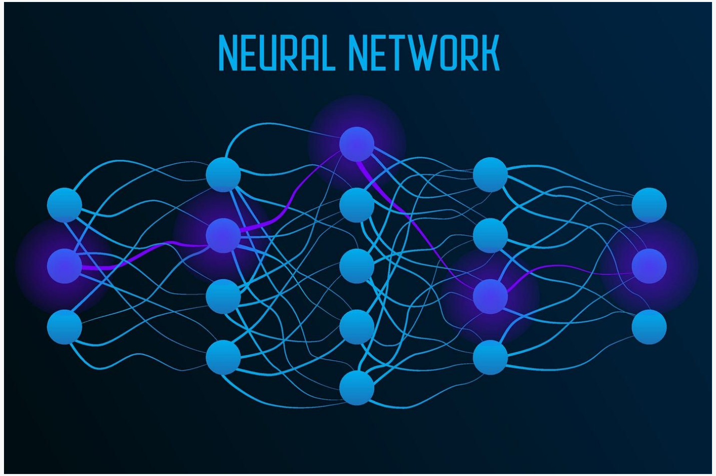
Architecture

DQN:

The DQN Replace Q–Table with NN prediction.
Performing with more robust and scalable solution rather than using a big action–state table

State

Action



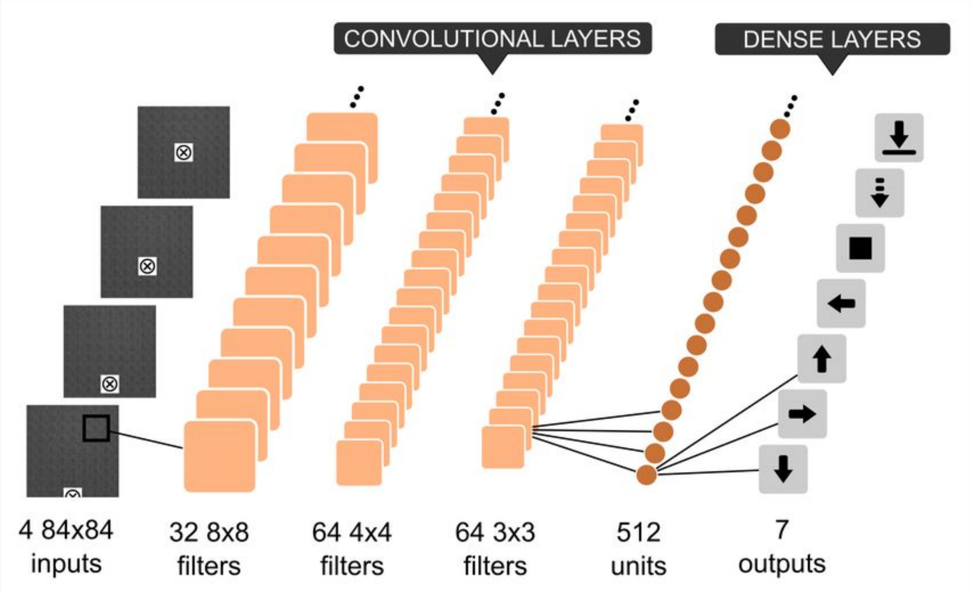
Q–Value

```
self.agent.replay_memory.store(state, action, next_state, reward)
```

Architecture

Agent & NN

1. **Input Layer:**
   - State representation (game screen pixels)
2. **Convolutional Neural Networks (CNNs):**
   - Extract spatial features from inputs
3. **Fully Connected Layers:**
   - Combine features to predict Q-values
   - ReLU Activation function
4. **Output Layer:**
   - Q-values for each possible action



Architecture

Optimal Q action-value function

s – state (observation)
a – action
r – reward
t – index for step
γ – discount
π – policy P(action|state)

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

The data for the NN

The data is the states and actions that were happened in the environment

e – experience
t – index for step
Dt – collection of e, data set

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

$$D_t = \{e_1, \ldots, e_t\}$$

Architecture

Loss function

i – index for iteration
θ – weights
U(D) – unified the data set

s′ – state, s – previous state
a′ – action, a – previous action
r – reward
γ – discount
Q – action-value function

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

Optimization algorithm

SGD – stochastic gradient descent

03

Articles

Articles

Human-level control through deep reinforcement learning

LETTER

doi:10.1038/nature14236

Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Playing Atari with Deep Reinforcement Learning

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih   Koray Kavukcuoglu   David Silver   Alex Graves   Ioannis Antonoglou
Daan Wierstra   Martin Riedmiller
DeepMind Technologies

04

Environment

Froze Lake

1. **Game Description:**
   - Frozen Lake is a grid-based game where an agent must navigate from a start point to a goal point on a slippery surface, avoiding holes.

2. **Environment Setup:**
   - Grid Layout: Typically an $n \times n$ grid (4x4 or 8x8)
     - S: Start point
     - G: Goal point
     - H: Holes (fall into one, and the game is over)
     - F: Frozen surface (safe to walk on)

3. **Borders:**
   - The grid is bounded, meaning the agent cannot move outside the grid.
   - Actions attempting to move out of bounds result in no movement.

Froze Lake

4. States and Actions:

- States (S): Each cell in the grid represents a unique state.
- Actions (A): Four possible actions – left, right, up, down.
- Rewards (R):
  - +1 for reaching the goal.
  - 0 for all other transitions (including falling into a hole).

5. Slippery Surface Dynamics:

- **Slippery Nature:** Due to the slippery surface, actions do not always result in moving to the intended adjacent cell. Instead, there is a chance the agent will slide to an unintended cell, adding randomness to the game and increasing the complexity of navigation.

05

METHODOLOGY

Project Goal

Train an AI agent using Deep Q-Networks (DQN) to navigate the Frozen Lake environment and successfully reach the goal state while avoiding falling into holes.

Topic	OBJECTIVE
Problem Definition	• Train DQN agent to navigate Frozen Lake, reaching goal while avoiding holes.
Environment Setup	• Define Frozen Lake grid with start, goal, holes, and frozen surfaces.
State and Action Representation	• States: Agent's grid position; Actions: Left, right, up, down movements.
DQN Implementation	• Use neural network for Q-function approximation.
Training Process	• Implement DQN model
Evaluation and Testing	• Assess agent's performance: average reward, success rate.
Hyperparameter Tuning	• Optimize parameters like learning rate, discount factor.

06

Project Plans

Project Stages

Implement RL:
DQN model and Environment

Stage 1

Train and Test RL Model

Stage 2

Collect and Analyze
Results

Stage 3

Thanks

Aviv Salomon
avivsalo@gmail.com

Ben Gornizky
bengornizky@gmail.com