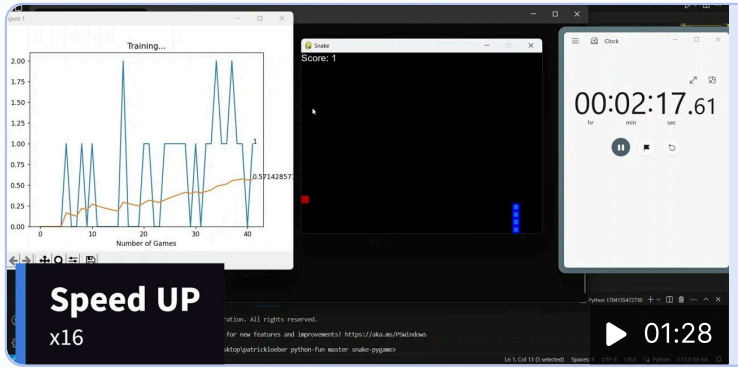
AI Mastery in Snake: A Reinforcement Learning Approach

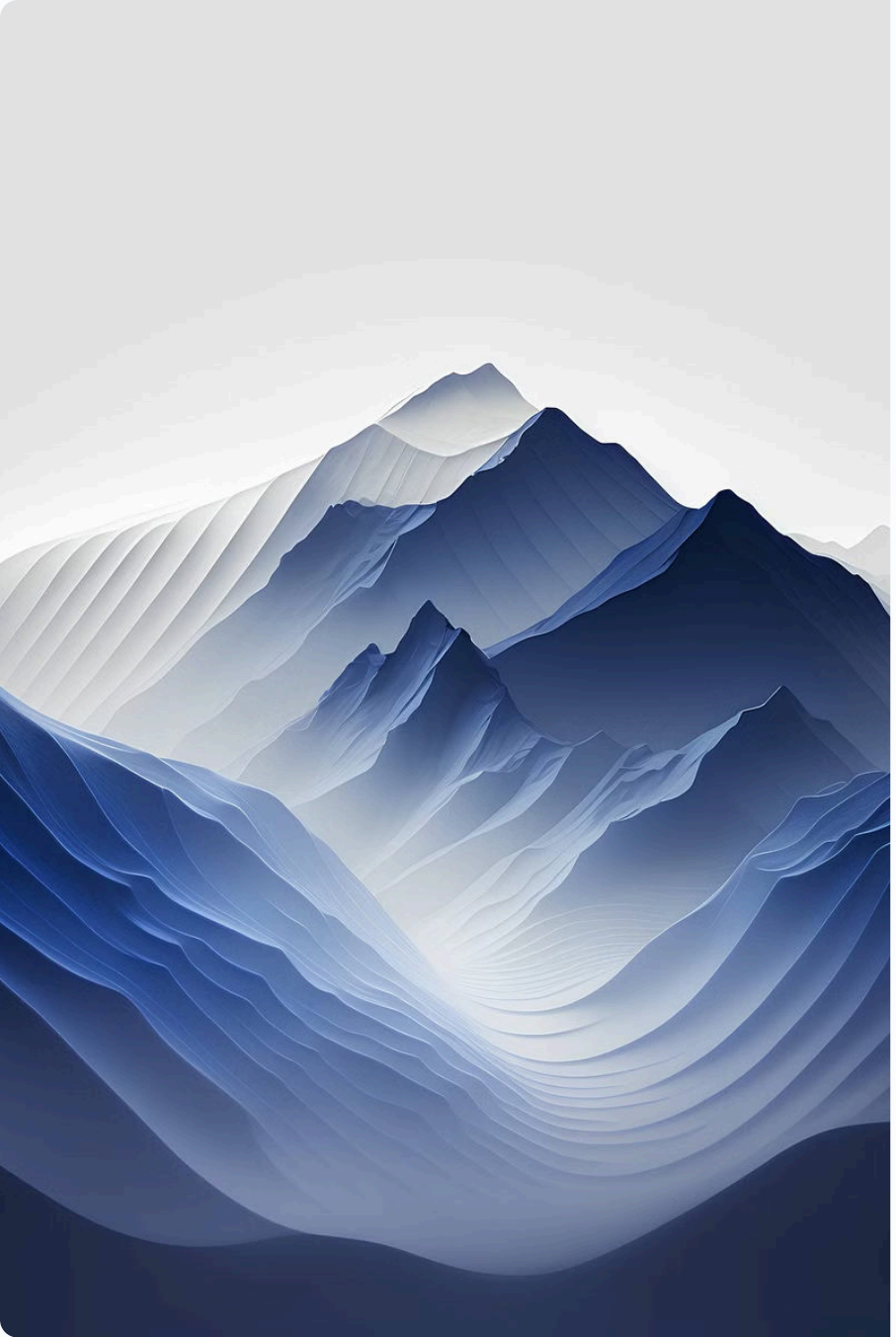
This report details the development of an AI that learns to play Snake from scratch. Leveraging Pygame for game development and PyTorch for AI training, we apply reinforcement learning techniques to evolve our AI from basic trial-and-error to strategic gameplay. The results demonstrate the AI's progression and the efficacy of deep learning in real-time problem-solving.



YouTube

AI Mastery in Snake: A Reinforcement Learning Approach

1- Reinforcement Learning Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take...



Outline

1. Part 1: Theory
2. Part 2: Implement the game (environment)
3. Part 3: Implement the agent
4. Part 4: Implement the model

Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

Or:

RL is teaching a software agent how to behave in an environment by telling it how good it's doing.

Deep Q Learning

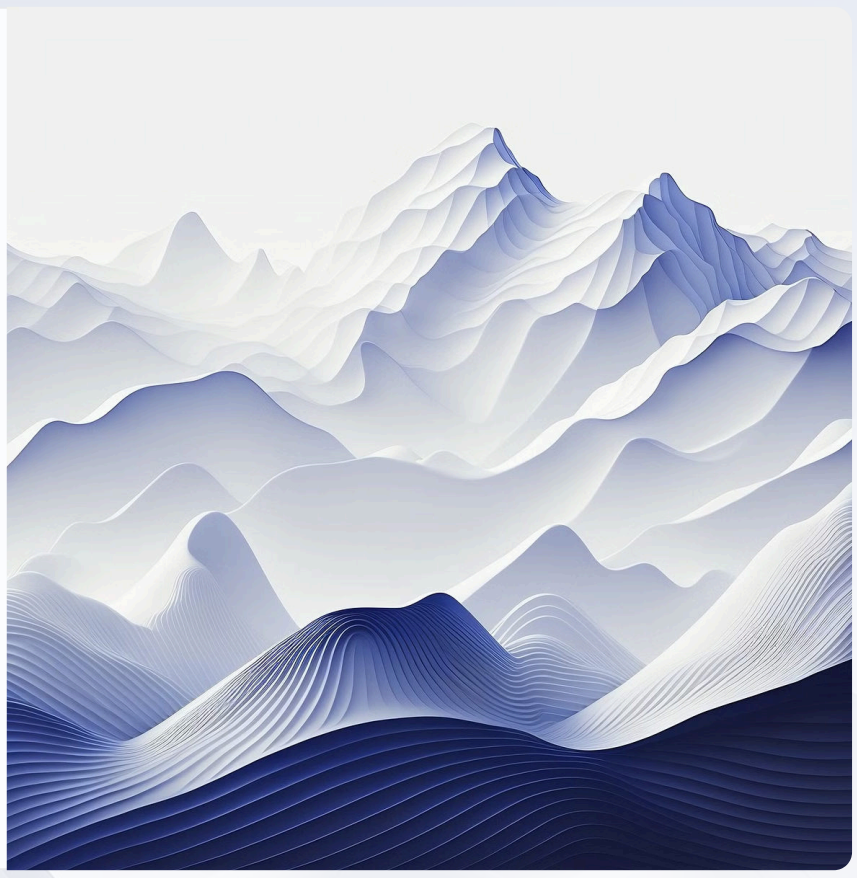
This approach extends reinforcement learning by using a deep neural network to predict the actions.

Code Organization Overview

Game (PyGame)

- play_step(action)

  → reward, gameover, score



Agent

- game

- model

Training:

- State = get_state(game)

- Action = get_move(state)

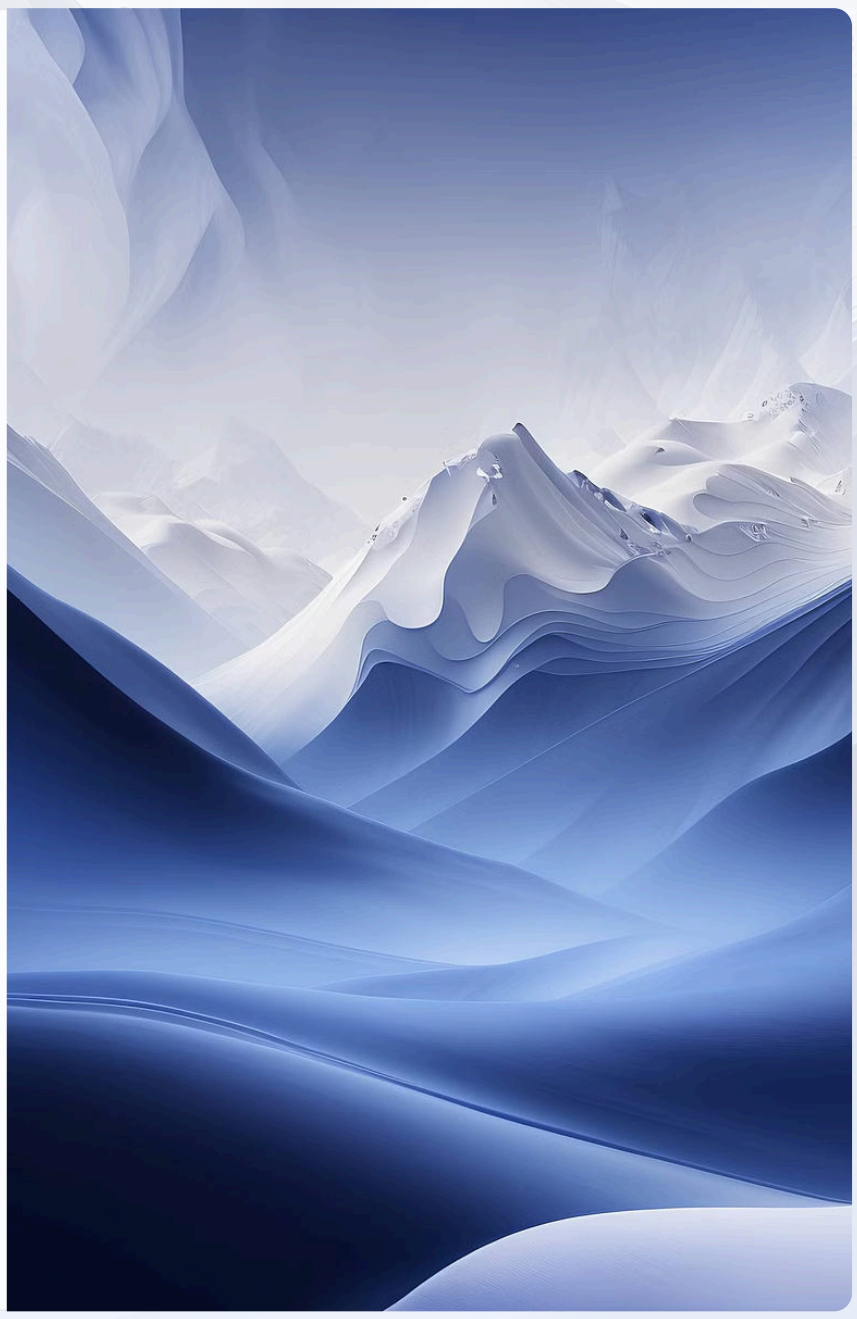
  - model.predict()

- reward , game_over, score = game.play_step(action)

- New_state = get_state (game)

- Remember

- model.train()



Model (PyTorch)

Linear _QNet (DQN)

- model.predict(state)

→ action

Reward

- Eat Food : +10

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

- Game Over : -10

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

- Else : 0

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Action

- [1, 0, 0] → Straight

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

- [0, 1, 0] → Right Turn

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

- [0, 0, 1] → Left Turn

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

State (11 Values)

[danger straight, danger right, danger left,

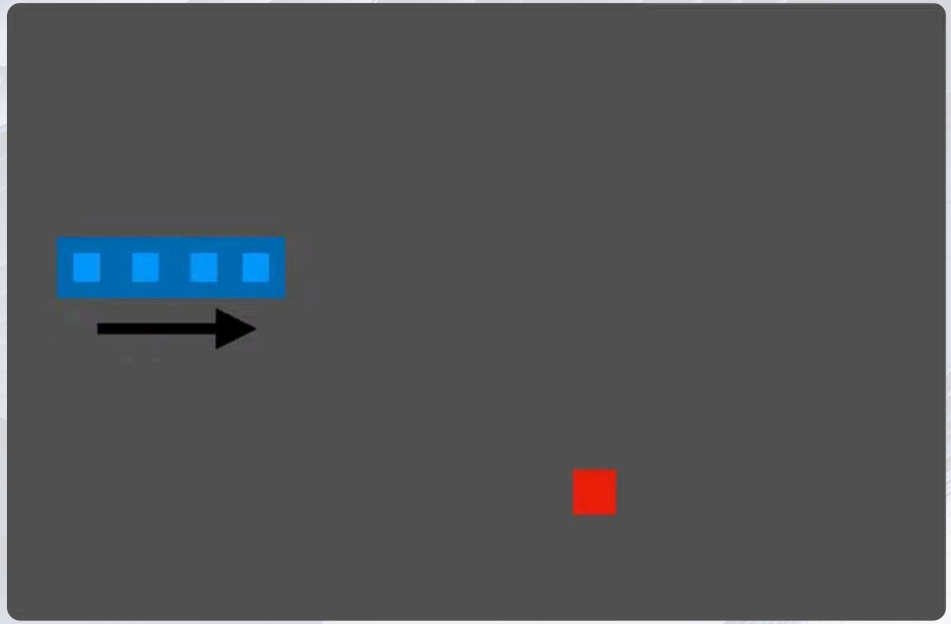
direction left, direction right, direction up, direction down,

food left, food right, food up, food down]

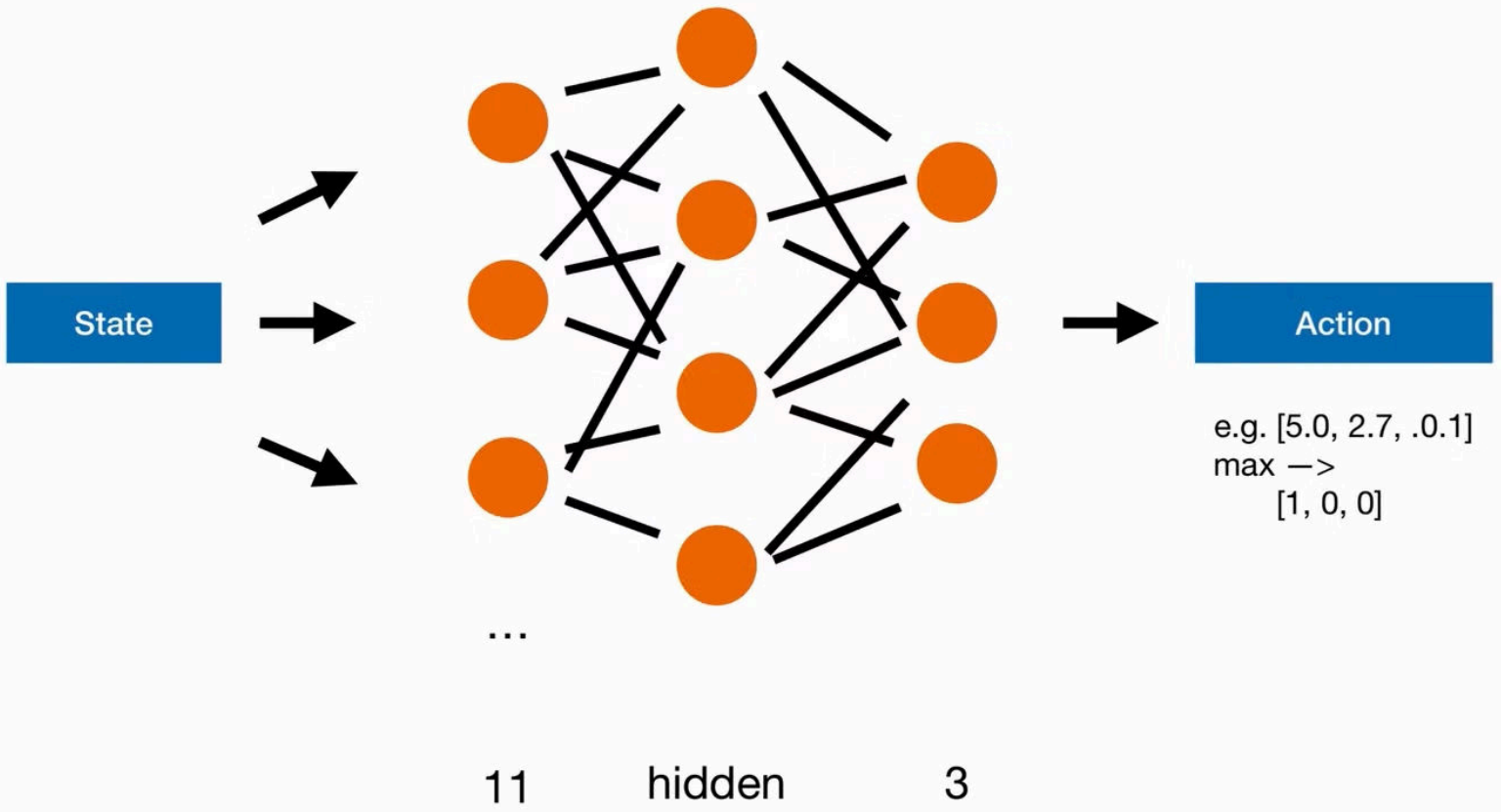
[0, 0, 0,

0, 1, 0, 0,

0, 1, 0, 1]



Model



(Deep) Q Learning

Q Value = Quality of action

1. Init Q Value (= init model)
2. Choose action (model.predict(state))

(or random move)

3. Perform action
4. Measure reward
5. Update Q value (+ train model)

Bellman Equation

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

New Q value for that state and that action

Current Q value

Learning Rate

Reward for taking that action at that state

Discount rate

Maximum expected future reward **given the new s' and all possible actions at that new state**

Q Update Rule Simplified:

$$Q = \text{model.predict}(\text{state}_0)$$

$$Q_{\text{new}} = R + \gamma \cdot \max(Q(\text{state}_1))$$

Loss function:

$$loss = (Q_{\text{new}} - Q)^2$$

→ Mean Squared Error