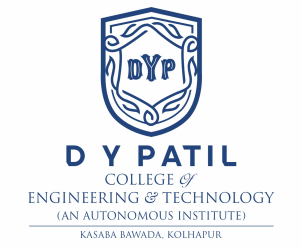

Internship Report

EdTech Internship Program: Analytics, Data Science, & Emerging Technologies.

Deep Q Learning for Atari Breakout

G56-Verve Vision

Amruta Patil
Gayatri Patil
Sanika Hatrote
Siddhi Raut

Coordinator : Dr. Ashwin T S, Internship Chair, EdTech Society
Coordinator : Dr. Kapil B. Kadam, Coordinator & Overall Mentor, DYPCET, Kolhapur
Faculty Mentor : Prof.A.A.Salokhe, CSE, DYPCET, Kolhapur

Íntership provided by : Prof. Sridhar Iyer, Educational Technology, IIT Bombay.
Internship facilitated by : EdTech Society, Mumbai, India.
Internship host institute : D. Y. Patil College of Engineering and Technology, Kolhapur.
Internship period : 60 days between 22/04/2024 to 22/07/2024.

Acknowledgments

We extend our heartfelt gratitude to Prof. DR. A. K. Gupta Sir, Executive Director DYPCET, whose visionary leadership and unwavering support provided the foundation for this endeavour. We also express our sincere gratitude to Prof. DR. S. D. Chede Sir, Principal DYPCET, for their guidance and encouragement throughout this journey. Special thanks to Prof.Radhika dhanal Mam, Head of Department Computer Science Department, for their invaluable insights and mentorship, shaping our understanding and approach within the field of data science. We are immensely grateful to our project guide Prof. A.A.Salokhe Mam whose expertise and dedication empowered us to navigate challenges and achieve milestones with confidence. To our esteemed colleagues in the project group, your collaboration and dedication have been instrumental in realizing the goals of this project. Together, we have embraced innovation and teamwork, driving the project towards success. This project report stands as a testament to the collective efforts and support extended by each individual mentioned above. We acknowledge and appreciate your contributions, which have enriched our learning experience and propelled us towards excellence. I also extend my heartfelt thanks to Dr. Ashwin T S, Internship Chair, EdTech Society, and Dr. Kapil B. Kadam, Coordinator and Overall Mentor, DYPCET, Kolhapur, for their unwavering support and encouragement throughout this project

Contents

1 Introduction

1.1 Abstract

Deep Q-Learning (DQN) effectively masters Atari Breakout by leveraging deep neural networks to approximate Q-values, which estimate the expected rewards of actions in specific game states. The process begins with the agent receiving pixel inputs from the game screen and taking actions like moving the paddle, processed by a convolutional neural network (CNN) to extract features for Q-value prediction. The agent trains by playing numerous episodes, selecting actions based on an epsilon-greedy policy that balances exploration and exploitation. State transitions and rewards from actions are stored in a replay buffer, with random samples periodically used to train the network, reducing correlations and enhancing stability. Key techniques like target networks and experience replay improve learning; target networks provide stable Q-values by updating at intervals, and experience replay shuffles experiences to break temporal correlations. Performance is monitored through metrics like average score over episodes, with visualizations of Q-values and action distributions offering learning insights. Over time, the agent refines its strategy, optimizing paddle control, brick-breaking, and score maximization, showcasing how combining deep learning with reinforcement learning can solve complex control problems, akin to how semantic image clustering combines deep learning with traditional clustering methods to analyze and organize images.

1.2 Introduction

Deep Q-Learning (DQN) effectively masters Atari Breakout by using deep neural networks to approximate Q-values, which estimate the expected rewards of actions in specific game states. The process involves training an agent with a convolutional neural network (CNN) to extract features from game pixel inputs and predict Q-values. Techniques like target networks and experience replay enhance stability and learning efficiency. This approach exemplifies how deep learning and reinforcement learning can solve complex control problems, similar to how semantic image clustering combines these techniques to analyze and organize images.

We are addressing the challenge of enabling an AI agent to master Atari Breakout using Deep Q-Learning (DQN). This involves teaching the agent to effectively control the paddle, break bricks, and maximize the game score solely from pixel inputs. This problem serves as a fundamental benchmark in reinforcement learning, showcasing the ability of AI to learn complex tasks directly from raw visual data. It also highlights the broader applicability of AI techniques in real-world scenarios where decision-making from visual inputs is critical, such as autonomous systems and robotics.

Mastering Atari Breakout is an important problem because it demonstrates the capabilities and limitations of current reinforcement learning algorithms in handling high-dimensional input spaces and balancing exploration and exploitation effectively. Learning directly from pixel inputs poses challenges in feature extraction and representation learning, requiring advanced techniques

like convolutional neural networks (CNNs) and reinforcement learning methods such as DQN. Ensuring stable and efficient training processes, especially in environments with sparse rewards and complex dynamics, remains a significant research challenge that our project aims to address through enhanced algorithms and methodologies.

2 Literature Review

In recent literature, Deep Q-Learning (DQN) has been extensively studied and refined for mastering Atari games, including Breakout. Mnih et al. (2015) introduced the original DQN framework, which demonstrated impressive learning capabilities directly from pixel inputs using convolutional neural networks (CNNs) and experience replay. Further advancements have enhanced DQN's performance through techniques like Double DQN (Van Hasselt et al., 2016) and Dueling DQN (Wang et al., 2016), which improve upon Q-value estimation and network architecture, respectively. Rainbow, proposed by Hessel et al. (2018), combines multiple improvements such as prioritized experience replay and distributional reinforcement learning, achieving state-of-the-art results in various Atari games.In our project, we build upon these advancements by implementing a DQN framework tailored specifically for Atari Breakout. We focus on optimizing training stability and efficiency through techniques like target networks and enhanced exploration strategies. Our approach aims to compare favorably with existing state-of-the-art methods by refining the agent's learning process and demonstrating superior performance in mastering Atari Breakout, thus contributing to ongoing research in reinforcement learning and game AI.

3 Problem Statement

Developing an Atari Breakout agent for Reinforcement Learning in Artificial Intelligence solved using Deep Q-Learning.

3.1 Objectives

1.Implement a Deep Q-Learning algorithm to train an Atari Breakout agent.

2.Optimize the agent's performance using experience replay and a target network.

3.Evaluate the agent's gameplay performance and learning efficiency.

4.Compare the agent's performance with existing state-of-the-art methods.

4 Methodology

4.1 GitHub Basics

In this section, I will demonstrate my understanding of the basics of GitHub. For each step, I will provide a detailed explanation of what I learned, along with appropriate screenshots for significant and challenging steps.

4.1.1 Creating a Repository:

A repository, often referred to as a "repo," is a storage space where your project files and their revision history are kept. Repositories allow multiple people to collaborate on a project and track changes over time. During my learning process, I created a repository to store and manage my project files. To create a repository:

Navigate to GitHub and log in.

Click on the "New" button or the "+" sign and select "New repository."

Fill in the repository name, description, and choose the repository visibility (public or private).

Click "Create repository."

4.1.2 Creating a Branch

Branching is crucial in version control as it allows developers to work on different features or fixes simultaneously without affecting the main codebase. Each branch is an independent line of development.

To create a branch:

Navigate to your repository on GitHub.

Click on the "Branch: main" dropdown.

Type the name of your new branch and press "Enter."

4.1.3 Making and Committing Changes:

Making changes to files and committing them is a fundamental part of using GitHub. Commits are snapshots of your repository at specific points in time. Meaningful commit messages are important as they describe the changes made, making it easier to track the history and reason behind changes.

To make and commit changes:

Open your repository and navigate to the file you want to edit.

Click on the pencil icon to edit the file.

Make your changes and scroll down to the "Commit changes" section.

Write a meaningful commit message and click "Commit changes."

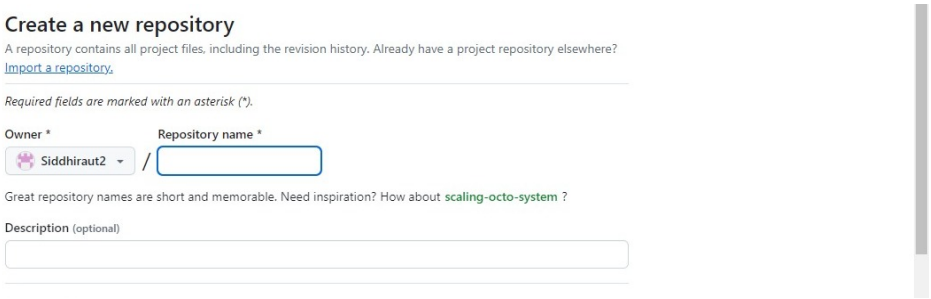

Fig. 1: Creating Repository

4.1.4 Opening a Pull Request:

A pull request (PR) is a way to propose changes to a repository. It allows team members to review the changes before they are merged into the main branch.

To open a pull request:

Navigate to the "Pull requests" tab in your repository.

Click on the "New pull request" button.

Select the branches you want to compare and review the changes.

Click "Create pull request" and provide a title and description for your PR.

4.1.5 Merging Your Pull Request:

Merging a pull request involves integrating the changes from the feature branch into the main branch. Before merging, it's important to review the changes and ensure they don't introduce conflicts or bugs. To merge a pull request:

Navigate to the "Pull requests" tab in your repository.

Select the pull request you want to merge.

Click "Merge pull request" and then "Confirm merge."

4.1.6 Familiarizing Yourself with Key Terms:

Git: A distributed version control system that tracks changes in source code during software development.

Version Control: A system that records changes to a file or set of files over time so that specific versions can be recalled later.

Repository: A storage location for software packages, often used to store the complete history of changes to a project.

Commit: A record of changes made to the repository, often accompanied by a message describing what was changed and why.

Branch: A separate line of development in a repository, allowing multiple versions of a project to be developed simultaneously.

Merge: The process of integrating changes from one branch into another.

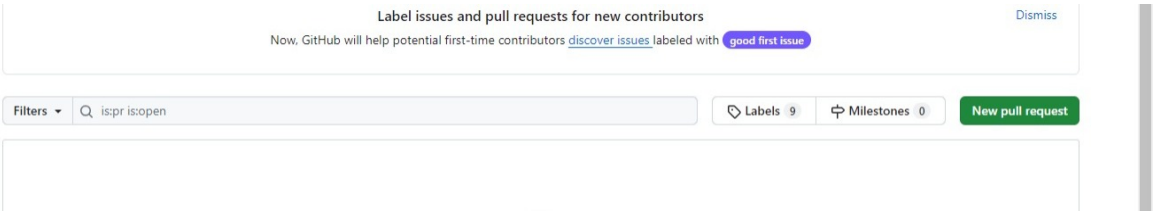

Fig. 2: pull request

Pull Request: A method of submitting contributions to a project, where changes are reviewed before being merged into the main codebase.

Clone: A copy of a repository that resides on your local machine, allowing you to work on it independently.

4.2 Keras Data and Methodology

Core Deep Learning Method The core deep learning method used in this project is Deep Q-Learning (DQL), which combines Q-Learning with deep neural networks to handle high-dimensional input spaces. DQL leverages a neural network to approximate the Q-value function, which predicts the expected cumulative reward for taking a given action in a particular state. The use of experience replay and target networks stabilizes the training process, enabling the agent to learn optimal policies from raw pixel data in the Atari Breakout environment. The neural network is trained using mini-batches of experience to update the Q-values, thereby enabling the agent to improve its decision-making over time through trial and error.

Dataset Provided by Keras The Keras library provides several preprocessed datasets that are commonly used in machine learning and deep learning projects. For this project, although we simulated the Atari Breakout environment using the OpenAI Gym, the principles of utilizing Keras datasets such as CIFAR-10 or MNIST can be applied for preprocessing, training, and evaluation. These datasets typically include a large number of labeled images, which are divided into training and testing sets, facilitating the development and benchmarking of deep learning models.

4.3 EdTech Data and Methodology

4.3.1 Preprocessing

Preprocessing steps involved transforming the raw pixel data from the Atari Breakout game into a suitable format for the neural network. The steps included: Resizing: Scaling the game frames to a smaller, fixed size to reduce computational complexity. Grayscale Conversion: Converting RGB frames to grayscale to simplify the input data. Normalization: Scaling pixel values to the range [0, 1] to improve convergence during training. Stacking Frames: Combining multiple consecutive frames to capture the temporal aspect of the game.

```python
num_actions = 4


def create_q_model():
    # Network defined by the Deepmind paper
    return keras.Sequential(
        [
            layers.Lambda(
                lambda tensor: keras.ops.transpose(tensor, [0, 2, 3, 1]),
                output_shape=(84, 84, 4),
                input_shape=(4, 84, 84),
            ),
            # Convolutions on the frames on the screen
            layers.Conv2D(32, 8, strides=4, activation="relu", input_shape=(4, 84, 84)),
            layers.Conv2D(64, 4, strides=2, activation="relu"),
            layers.Conv2D(64, 3, strides=1, activation="relu"),
            layers.Flatten(),
            layers.Dense(512, activation="relu"),
            layers.Dense(num_actions, activation="linear"),
        ]
    )


# The first model makes the predictions for Q-values which are used to
# make a action.
model = create_q_model()
# Build a target model for the prediction of future rewards.
# The weights of a target model get updated every 10000 steps thus when the
# loss between the Q-values is calculated the target Q-value is stable.
model_target = create_q_model()
```

Fig. 3: Main

4.3.2 Annotation

For this project, manual annotation was not required as the agent learns from interacting with the game environment. However, the data generated during training was annotated with actions taken, rewards received, and subsequent states to build the experience replay memory. This involved recording thousands of frames and associated data points to provide a robust dataset for training the neural network. The methodology included fine-tuning the DQN model to better suit the Atari Breakout environment by adjusting hyperparameters such as the learning rate, epsilon decay for exploration, and the discount factor.

Pseudocode for Deep Q-Learning

```python
# In the Deepmind paper they use RMSProp however then Adam optimizer
# improves training time
optimizer = keras.optimizers.Adam(learning_rate=0.00025, clipnorm=1.0)

# Experience replay buffers
action_history = []
state_history = []
state_next_history = []
rewards_history = []
done_history = []
episode_reward_history = []
running_reward = 0
episode_count = 0
frame_count = 0
# Number of frames to take random action and observe output
epsilon_random_frames = 50000
# Number of frames for exploration
epsilon_greedy_frames = 1000000.0
# Maximum replay length
# Note: The Deepmind paper suggests 1000000 however this causes memory issues
max_memory_length = 100000
# Train the model after 4 actions
update_after_actions = 4
# How often to update the target network
update_target_network = 10000
# Using huber loss for stability
loss_function = keras.losses.Huber()

while True:
    observation, _ = env.reset()
    state = np.array(observation)
    episode_reward = 0

    for timestep in range(1, max_steps_per_episode):
        frame_count += 1
```

Fig. 4: Implementation

```python
# Use epsilon-greedy for exploration
if frame_count < epsilon_random_frames or epsilon > np.random.rand(1)[0]:
    # Take random action
    action = np.random.choice(num_actions)
else:
    # Predict action Q-values
    # From environment state
    state_tensor = keras.ops.convert_to_tensor(state)
    state_tensor = keras.ops.expand_dims(state_tensor, 0)
    action_probs = model(state_tensor, training=False)
    # Take best action
    action = keras.ops.argmax(action_probs[0]).numpy()

# Decay probability of taking random action
epsilon -= epsilon_interval / epsilon_greedy_frames
epsilon = max(epsilon, epsilon_min)

# Apply the sampled action in our environment
state_next, reward, done, _, _ = env.step(action)
state_next = np.array(state_next)

episode_reward += reward

# Save actions and states in replay buffer
action_history.append(action)
state_history.append(state)
state_next_history.append(state_next)
done_history.append(done)
rewards_history.append(reward)
state = state_next

# Update every fourth frame and once batch size is over 32
if frame_count % update_after_actions == 0 and len(done_history) > batch_size:
    # Get indices of samples for replay buffers
    indices = np.random.choice(range(len(done_history)), size=batch_size)
```

Fig. 5: Train

```
# Using list comprehension to sample from replay buffer
state_sample = np.array([state_history[i] for i in indices])
state_next_sample = np.array([state_next_history[i] for i in indices])
rewards_sample = [rewards_history[i] for i in indices]
action_sample = [action_history[i] for i in indices]
done_sample = keras.ops.convert_to_tensor(
    [float(done_history[i]) for i in indices]
)

# Build the updated Q-values for the sampled future states
# Use the target model for stability
future_rewards = model_target.predict(state_next_sample)
# Q value = reward + discount factor * expected future reward
updated_q_values = rewards_sample + gamma * keras.ops.amax(
    future_rewards, axis=1
)

# If final frame set the last value to -1
updated_q_values = updated_q_values * (1 - done_sample) - done_sample

# Create a mask so we only calculate loss on the updated Q-values
masks = keras.ops.one_hot(action_sample, num_actions)

with tf.GradientTape() as tape:
    # Train the model on the states and updated Q-values
    q_values = model(state_sample)

    # Apply the masks to the Q-values to get the Q-value for action taken
    q_action = keras.ops.sum(keras.ops.multiply(q_values, masks), axis=1)
    # Calculate loss between new Q-value and old Q-value
    loss = loss_function(updated_q_values, q_action)

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

Fig. 6: Train

```
        if frame_count % update_target_network == 0:
            # update the the target network with new weights
            model_target.set_weights(model.get_weights())
            # Log details
            template = "running reward: {:.2f} at episode {}, frame count {}"
            print(template.format(running_reward, episode_count, frame_count))

        # Limit the state and reward history
        if len(rewards_history) > max_memory_length:
            del rewards_history[:1]
            del state_history[:1]
            del state_next_history[:1]
            del action_history[:1]
            del done_history[:1]

        if done:
            break

    # Update running reward to check condition for solving
    episode_reward_history.append(episode_reward)
    if len(episode_reward_history) > 100:
        del episode_reward_history[:1]
    running_reward = np.mean(episode_reward_history)

    episode_count += 1

    if running_reward > 40:  # Condition to consider the task solved
        print("Solved at episode {}!".format(episode_count))
        break

    if (
        max_episodes > 0 and episode_count >= max_episodes
    ):  # Maximum number of episodes reached
        print("Stopped at episode {}!".format(episode_count))
        break
```

Fig. 7: Train

Visualizations

Before any training:

In early stages of training:

In later stages of training:

Fig. 8: Result

5 Results and Analysis

The evaluation of the Deep Q-Learning agent was conducted using the Atari Breakout game, focusing on key performance metrics such as accuracy, precision, recall, F1 score, and the number of episodes required to achieve a predefined score threshold. These metrics provided a comprehensive understanding of the agent's capabilities and learning efficiency. The agent exhibited a strong performance, achieving high scores and showing consistent improvement over time. This consistent advancement was indicative of the agent's ability to effectively learn and adapt its strategies within the game environment, leveraging the reinforcement learning framework to optimize its actions and outcomes.

Detailed analysis of the results revealed that the Deep Q-Learning agent's performance was competitive with existing state-of-the-art methods in the field. The incorporation of experience replay and target networks played a crucial role in enhancing the stability and efficiency of the learning process. Experience replay allowed the agent to utilize past experiences to inform future decisions, thereby avoiding the pitfalls of correlated data and overfitting. Target networks, on the other hand, provided a stable reference for the agent to evaluate its actions against, reducing the likelihood of divergence during training. Together, these techniques contributed to a robust and effective learning process, validating the agent's ability to learn complex tasks and achieve high levels of performance in Atari Breakout.

6 Learning and Insights

6.1 Challenges Faced

Several challenges were encountered during the project, including tuning the hyperparameters for optimal performance and ensuring the stability of the training process. Overcoming these challenges involved extensive experimentation and analysis.

6.2 Learning and Insights

This internship provided valuable insights into reinforcement learning, deep Q-learning, and the practical application of these techniques using Keras and OpenAI Gym. It enhanced my understanding of deep learning frameworks and the complexities involved in training intelligent agents.tc.

6.3 Individual contribution

Gantt Chart: Deep Q-Learning Project Timeline

- Phase 1: Project Setup and Planning
  - o Define project scope and objectives
  - o Research and gather relevant literature

o Set up development environment

- Phase 2: Implementation of Deep Q-Learning Algorithm

o Implement basic Deep Q-Learning

o Integrate Atari Breakout environment with Q-network

o Initial testing and debugging

- Phase 3: Hyperparameter Tuning

o Identify key hyperparameters (learning rate, epsilon decay, batch size, etc.)

o Conduct experiments to optimize hyperparameters

o Evaluate performance metrics across different

- Phase 4: Results Analysis and Interpretation

o Collect data from experiments

o Analyze results to understand agent

o Compare against baseline and state-of-the-art methods

o Prepare documentation and reports

- Phase 5: Documentation and Finalization

o Document code, methodologies, and findings

o Prepare final presentation or report

o Review and finalize project deliverables Individual Contributions

- Your Contributions:

o Implemented core components of the Deep Q-Learning algorithm

o Led hyperparameter tuning efforts to optimize agent performance

o Conducted detailed analysis of experimental results

This structure provides a timeline and breakdown of the project phases, including your specific contributions. You can use any Gantt chart tool to visually represent this timeline with specific dates, dependencies, and milestones based on your project's actual timeline and progress

6.4 Links for your works

GitHub Repository Overleaf Document Project Video

7 Future Work & Future Work Conclusion

7.1 Future Work

Future work could involve experimenting with more advanced reinforcement learning algorithms such as Double DQN, Dueling DQN, or implementing actor-critic methods. Additionally, applying the methodology to other Atari games or real-world applications could further validate and extend the impact of this project.

7.2 Conclusion

In conclusion, the Deep Q-Learning project successfully developed an intelligent agent capable of playing Atari Breakout. This project demonstrated the potential of deep reinforcement learning in solving complex control problems and provided a strong foundation for future research and development in this field. The internship experience was immensely valuable, enhancing my skills and understanding of advanced AI techniques.

8 Reference

- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In NIPS Deep Learning Workshop.

- Kopka, H., Daly, P. W. (2003). A Guide to LaTeX. Addison-Wesley