CSE 3013 Artificial Intelligence

Project Report

On

Atari Game Development Using Deep Reinforcement Learning

Prepared by

Sanskriti Bansal - 20BCE2634

Darshini Solanki - 20BCE2638

Under the supervision of

Prof. Mohana CM

School of Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

NOVEMBER 2023

DECLARATION

I hereby declare that the report entitled “Atari Game Development Using Deep Reinforcement Learning” submitted by me, for the CSE3013 Artificial Intelligence (EPJ) to Vellore Institute of Technology is a record of bonafide work carried out by me under the supervision of Prof. Mohana CM.

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Place : Vellore

Date : 24 November 2023

Sanskriti Bansal - 20BCE2634

Darshini Solanki - 20BCE2638

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Prof. Mohana CM, School of Computer Science and Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to the Dean of School of Computer science Engineering, VIT Vellore, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Head of the Department for her support throughout the course of this project. We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Sanskriti Bansal - 20BCE2634

Darshini Solanki - 20BCE2638

ABSTRACT

General game playing is an extremely complex challenge, since building a model that is able to learn to play any game is a task that is closely related to achieving artificial general intelligence (AGI). Video games are an appropriate test bench for general purpose agents, since the wide variety of games allows solutions to use and hone many different skills like control, strategy, long term planning and so on. Designed to provide enough of a challenge to human players, Atari games in particular are a good testbed for incipient general intelligence. In this project, we aim to develop one such Atari game named, Space Invaders using Deep Reinforcement Learning technique. DRL is a cutting-edge approach at the intersection of deep learning and reinforcement learning, aimed at training intelligent agents to make sequential decisions in complex environments. The model is run for 10 episodes and has given convincing results.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

1. AI - Artificial Intelligence
2. DRL - Deep Reinforcement Learning
3. RL - Reinforcement Learning
4. CNN - Convolutional Neural Network
5. DQN - Deep Q-Network
6. TRPO - Trust Region Policy Optimization
7. DRQN - Deep Recurrent Q-Network

SYMBOLS AND NOTATIONS

Variables and Parameters:

1. A - Agent
2. E - Environment
3. S - State
4. R - Reward
5. Q - Q-value
6. P - Policy
7. ϵ - Exploration-exploitation trade-off parameter
8. μ - Mean
9. σ - Standard Deviation
10. nb_steps - Number of steps
11. nb_episodes - Number of episodes

1. INTRODUCTION

Deep Reinforcement Learning (DRL) is a cutting-edge approach at the intersection of deep learning and reinforcement learning, aimed at training intelligent agents to make sequential decisions in complex environments. It combines the power of deep neural networks to process raw data with reinforcement learning's framework of learning through interaction and feedback.

Deep Reinforcement Learning (DRL) has emerged as a powerful approach for training agents to perform tasks in various domains, ranging from robotics to gaming. One particularly impactful application of DRL is its use in playing classic Atari 2600 games. The Atari 2600, a pioneering gaming console from the late 1970s, provides a challenging environment for testing and developing intelligent agents due to its diverse set of games with varying complexities.

The application of Deep Reinforcement Learning in Atari games serves as a testament to the evolving synergy between AI and classic gaming. Beyond its gaming applications, DRL's impact reverberates through fields like robotics, autonomous vehicles, finance, and healthcare, where decision-making in complex environments is paramount. As DRL techniques continue to advance, the journey from pixelated screens to real-world challenges promises to reshape the landscape of AI innovation.

1.1 Project objective

- Implement and adapt state-of-the-art deep reinforcement learning algorithms.
- Train agents to play a selection of Atari games using the OpenAI Gym framework.
- Optimize hyperparameters and network architectures to improve learning efficiency and performance.
- Compare and analyze the performance of different algorithms and techniques.
- Document the process, findings, and lessons learned for future reference

1.2 Motivation

Traditional reinforcement learning involves training agents to learn optimal actions in an environment by interacting with it and receiving feedback in the form of rewards. Deep Reinforcement Learning takes this approach a step further by incorporating deep neural networks, which enable the agent to learn complex patterns and representations from raw input data, such as images from game screens.

2. LITERATURE SURVEY

Paper Name	Technology Used	Result
Playing Atari Games with Deep Reinforcement Learning and Human Checkpoint Replay (2016)	This study describes a revolutionary deep reinforcement learning strategy for learning how to play the most challenging Atari 2600 games. The proposed method, known as human checkpoint replay, involves using checkpoints from human games as starting points for the learning process. To estimate the state value function, the model employs a convolutional neural network that accepts just raw pixel inputs.	The experiments reveal a significant improvement compared to all previous learning systems, as well as over a random player. Their strategy is inspired by curriculum learning and aims to compensate for the problems of current exploration tactics in finding good control policies in environments with little rewards.
Visual Rationalizations in Deep Reinforcement Learning for Atari Games (2019)	In this paper, researchers suggested making deep reinforcement learning more transparent by visualizing the facts on which the agent bases its decision. They emphasized the necessity of developing a rationale for an observed action that may be applied to a black-box decision agent.	The trials on three Atari 2600 games show that the visualizations correctly attend to the regions that lead to the action, such as the agent and the impediment. They contended that such visual rationalizations, or post-hoc explanations, are necessary to facilitate communication between users and agents.
Deep Reinforcement Learning: An Overview (2018)	This paper discusses recent achievements in deep reinforcement learning, with an emphasis on the most commonly used deep architectures, such as autoencoders, convolutional neural networks, and recurrent neural networks, that have been effectively combined with the	High-performance deep learning models can automatically extract complex data representations from high-dimensional input data, surpassing typical machine learning methods. The most commonly utilized deep architectures with RL include deep convolutional networks, autoencoders, and

	reinforcement learning framework.	recurrent networks. Additionally, deep networks specific to partially observable MDPs (POMDPs) environments have been explored.
Deep Reinforcement Learning: A Brief Survey (2017)	The deep Q-network (DQN), trust region policy optimisation (TRPO), and asynchronous advantage actor critic are among the key deep RL algorithms included in the survey. Parallel to this, the distinctive benefits of deep neural networks are emphasized, with a particular emphasis on visual comprehension via RL	Recent research using generative causal models showed improved generalization in several benchmarks over typical DRL algorithms, attained via inference from causes and effects in the environment. the development of all-purpose AI systems that can communicate with and absorb information from their surroundings. RL's ability to interact with the environment has both benefits and drawbacks. In essence, RL gives agents the capacity to conduct studies to better comprehend their environment, allowing them to select even high-level causal links.
Playing Atari with Deep Reinforcement Learning (2013)	The model is a convolutional neural network that was trained using a variation of Q-learning. It takes raw pixel input and produces a value function that predicts future rewards. Without changing the architecture or learning methodology, the technique is applied to seven Atari 2600 games from the Arcade Learning Environment.	In this research, a novel deep learning reinforcement learning model is presented, and it is shown how it can learn challenging control strategies for Atari 2600 video games utilizing just raw pixel input. To make the training of deep networks for RL easier, a kind of online Q-learning is described that blends stochastic mini batch updates with experience replay memory. In six of the seven games it was tested on, the method produces cutting-edge outcomes without changing the architecture or hyperparameters.

Playing FPS Games with Deep Reinforcement Learning (2017)	The introduction of a technique for co-training a DQN with game characteristics proved to be crucial in directing the network's convolutional layers towards enemy detection. It has been demonstrated that co-training greatly enhances the model's training efficiency and performance. Using the API created by Kempka et al., we assess our model on the two distinct tasks taken from the Visual Doom AI Competition (ViZDoom)2.	This article presents a comprehensive infrastructure for playing deathmatch games in first-person shooter games. We offer a technique for adding high-level game information to DRQN models, and we modularized our architecture to include autonomous networks in charge of various game stages. When used for challenging tasks like a deathmatch, these techniques significantly outperform the conventional DRQN model. It has been proven that the suggested model can outperform both built-in bots and human gamers, and it has also been shown that our approach can be applied to maps that are new to us.

Table 1: Literature Survey

3. TECHNICAL SPECIFICATION

3.1 TECHNIQUES USED

Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) is a cutting-edge approach at the intersection of deep learning and reinforcement learning, aimed at training intelligent agents to make sequential decisions in complex environments. It combines the power of deep neural networks to process raw data with reinforcement learning's framework of learning through interaction and feedback.

At its core, DRL revolves around an agent interacting with an environment to maximize cumulative rewards. This interaction involves a cycle of observing the environment, selecting actions based on the observed state, receiving feedback in the form of rewards, and updating the agent's strategy over time. The objective is to learn a policy—a mapping from states to actions—that enables the agent to make optimal decisions to achieve its goals.

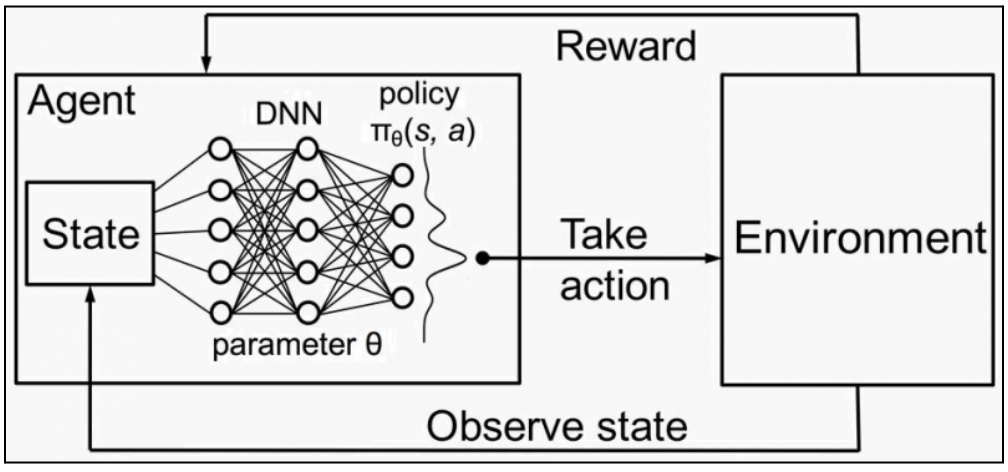

Fig1: DRL Technique

Key components of Deep Reinforcement Learning include:

1. Agent: The learner or decision-maker that interacts with the environment and learns to take actions that maximize cumulative rewards.

2. Environment: The external system with which the agent interacts. It provides feedback to the agent based on the actions it takes, influencing the agent's future decisions.

3. State: A representation of the current situation or configuration of the environment. It captures all relevant information necessary for the agent to make decisions.

4. Action: Choices made by the agent to influence the environment. The agent's goal is to learn a policy that selects actions to maximize future rewards.

5. Reward: Feedback from the environment that quantifies the desirability of an agent's action. Positive rewards encourage behaviors that lead to successful outcomes, while negative rewards discourage undesirable actions.

6. Policy: The strategy or decision-making rule that the agent uses to determine actions given states. The goal of DRL is to learn an optimal policy that maximizes expected cumulative rewards.

7. Value Function: A function that estimates the expected cumulative rewards starting from a certain state and following a certain policy. It helps the agent evaluate the desirability of different states.

DRL distinguishes itself by employing deep neural networks to approximate complex functions, such as policies and value functions. This enables DRL agents to process high-dimensional inputs like images and learn intricate patterns directly from raw data,

making it well-suited for tasks like playing video games, robotics, autonomous driving, and more.

CNN - Convolutional Neural Network

In terms of tasks requiring picture and video analysis, such as image identification, object detection, and image classification, this kind of deep learning method excels. CNNs are built to automatically and adaptively learn hierarchical characteristics from input photos and are inspired by the human visual system.

A CNN's fundamental elements are as follows:

1) Convolutional Layers: To extract different features, these layers conduct convolutions on the input picture using tiny filters (sometimes referred to as kernels). These filters build feature maps that emphasize various facets of the input image, such as edges, textures, and patterns, as they slide over the image.

2) Pooling Layers: By decreasing the spatial dimensions of the feature maps, pooling layers aid in lowering the computational complexity of the network and increase its resilience to input fluctuations. Common pooling techniques include maximum pooling and average pooling.

3) Fully Connected Layers: The final predictions or classifications based on the high-level characteristics retrieved by the convolutional and pooling layers are made using fully connected layers (layers 3). They are typical neural network layers that are present in these networks.

The performance of CNNs has proved crucial in delivering cutting-edge results on a variety of computer vision applications. Without the requirement for human feature engineering, they can learn complicated features from raw data. Their broad application

in areas like image identification, medical image analysis, self-driving automobiles, and more is a result of this capacity.

3.2 PYTHON MODULES USED

-> import gym

To access to a standardized interface for interacting with different environments. We can retrieve information about the state space, action space, and other relevant details for a particular environment.

-> import random

The random module in Python is a standard library that provides functions for generating pseudo-random numbers. In the context of reinforcement learning, it can be used for tasks such as selecting random actions during exploration or initializing weights with random values in neural networks

-> import numpy as np
-> from tensorflow.keras.models import Sequential
-> from tensorflow.keras.layers import Dense, Flatten, Convolution2D
-> from tensorflow.keras.optimizers import Adam

NumPy in the code, NumPy is likely used for numerical operations and handling array-like data structures.

TensorFlow is an open-source machine learning library, and Keras is a high-level neural networks API that runs on top of TensorFlow. In the code, `Sequential` is used to build the neural network model layer by layer.

TensorFlow Keras Layers are specific layers from the Keras module used to construct the neural network architecture. 'Dense' represents a fully connected layer, where each neuron is connected to every neuron in the previous and next layers. 'Flatten' is a layer that flattens the input,

converting multi-dimensional data into a flat vector.'Convolution2D' is a 2D convolution layer, which is crucial for processing image data.
TensorFlow Keras Optimizer 'Adam' is an optimization algorithm commonly used for training neural networks.

-> from rl.agents import DQNAgent
-> from rl.memory import SequentialMemory
-> from rl.policy import LinearAnnealedPolicy, EpsGreedyQPolicy

DQNAgent represents a Deep Q-Network (DQN) agent, which is a type of reinforcement learning algorithm used for learning policies in environments with discrete action spaces.
from rl.memory import SequentialMemory
SequentialMemory represents a sequential memory buffer used for storing and replaying experiences in reinforcement learning.
LinearAnnealedPolicy and EpsGreedyQPolicy classes define exploration policies for the reinforcement learning agent. EpsGreedyQPolicy represents an epsilon-greedy policy, and LinearAnnealedPolicy introduces a linear annealing schedule for the exploration-exploitation trade-off.
from tensorflow.keras.optimizers import Adam

4. DESIGN

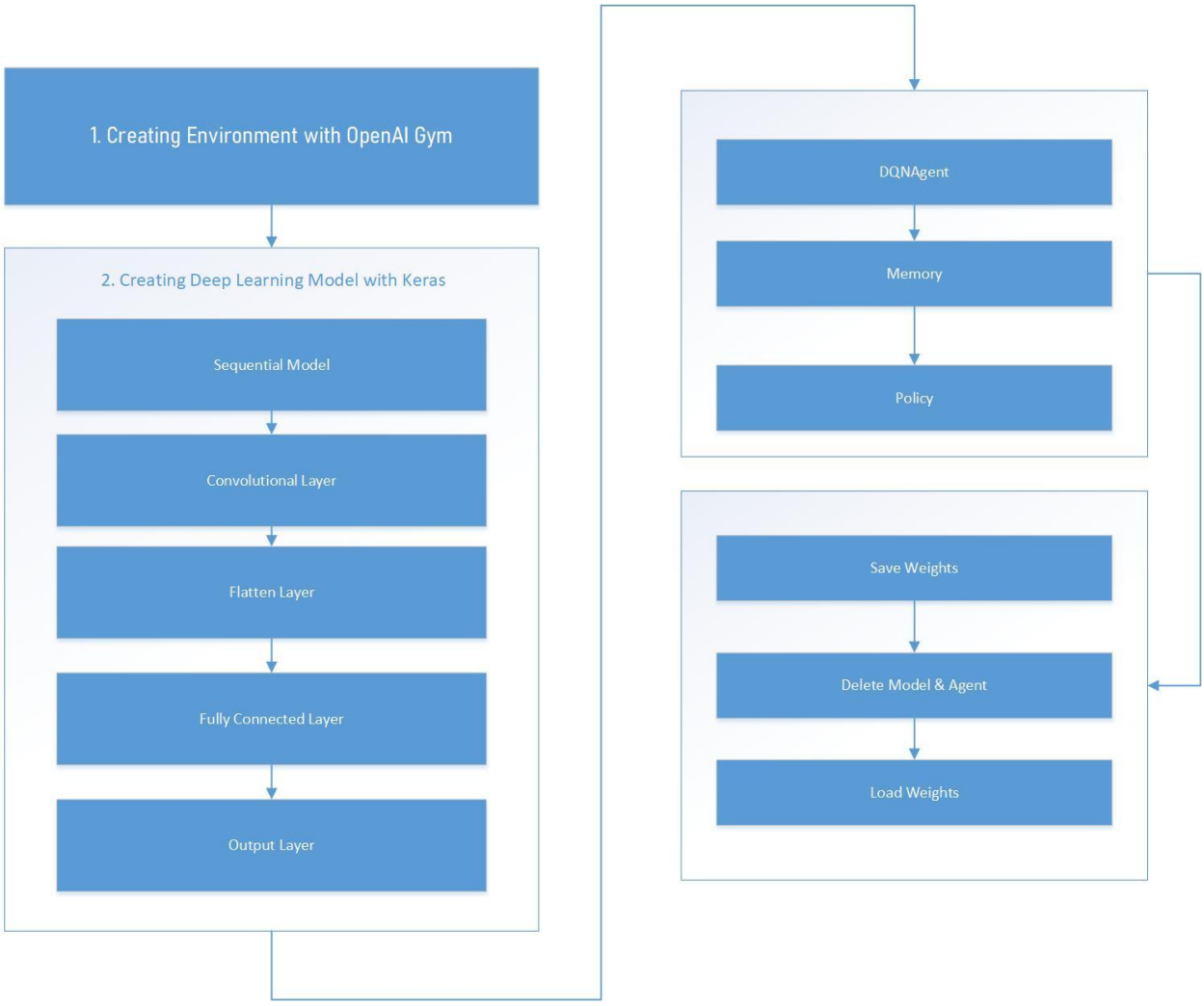

Fig2: design of proposed technique

5. PROPOSED SYSTEM

1. Creating Environment with OpenAI Gym

OpenAI Gym is an open-source Python library and toolkit designed to help developers and researchers develop and compare reinforcement learning algorithms. It provides a wide range of environments, from simple toy examples to more complex simulated environments, which allow you to test and evaluate reinforcement learning algorithms and agents. It can be easily integrated with popular reinforcement learning libraries, such as TensorFlow, PyTorch, and stable-baselines, making it convenient for training and evaluating reinforcement learning agents.

2. Creating Deep Learning Model with Keras

We have defined a neural network model for deep reinforcement learning in a convolutional neural network (CNN) architecture. This model is used for image-based input in reinforcement learning tasks, such as playing video games.

Following are the steps involved in model creation:

- model = Sequential(): This line creates a Sequential model in Keras, a popular deep learning framework. A Sequential model allows you to build a neural network layer by layer in a linear fashion.
- model.add(Convolution2D(32, (8,8), strides=(4,4), activation='relu', input_shape=(3, height, width, channels)): This adds the first convolutional layer to the model.
- strides=(4, 4): The strides control how the filter moves across the input image. In this case, it moves 4 pixels at a time in both the horizontal and vertical directions.
- activation='relu': Rectified Linear Unit (ReLU) is used as the activation function for this layer.
- input_shape=(3, height, width, channels): This specifies the shape of the input data. The '3' indicates that the input data is in RGB format (3 color channels). 'height' and 'width' represent the dimensions of the input image, and 'channels' represents the number of input channels.

- model.add(Convolution2D(64, (4,4), strides=(2,2), activation='relu'): This adds the second convolutional layer. It is similar to the first layer but has 64 filters and different filter sizes and strides.
- model.add(Convolution2D(64, (3,3), activation='relu'): This adds the third convolutional layer. It has 64 filters with a 3x3 filter size, and it does not specify any particular strides.
- model.add(Flatten()): This layer flattens the output from the previous convolutional layers into a 1D vector. This is necessary to connect the convolutional layers to the fully connected (dense) layers that follow.
- model.add(Dense(512, activation='relu'): This adds a fully connected (dense) layer with 512 units and ReLU activation.
- model.add(Dense(256, activation='relu'): This adds another fully connected layer with 256 units and ReLU activation.
- model.add(Dense(actions, activation='linear'): This is the output layer of the model. It has 'actions' units, which is typically the number of possible actions that the agent can take in a reinforcement learning task. The 'linear' activation function means that the output values are not constrained within a specific range and can take any real value.

3. Build Agent with Keras-RL

It explains an implementation of a Deep Q-Network (DQN) agent using the Keras-RL library. The agent undergoes a training process where it interacts with an environment for 10,000 steps. Throughout training, the code logs various metrics such as episode rewards, mean actions, and loss. Notably, the training duration, steps per second, and exploration rate are also tracked. After training, the agent undergoes testing on the same environment for 10 episodes, producing individual rewards for each episode and an average reward of 282.0. This approach signifies a reinforcement learning paradigm, where the agent refines its decision-making strategy based on learned experiences, ultimately exhibiting improved performance in the given task. The training and testing outcomes provide insights into the effectiveness of the DQN agent in the specified environment.

4. Reloading Agent from Memory

It demonstrates the process of saving the weights of a DQN agent after training (save_weights), deleting the original model and agent (del), and then reloading the weights into a new DQN agent (load_weights). This can be useful for checkpointing trained models, allowing for later use without having to retrain from scratch. The saved weights contain the learned knowledge of the agent, which can be applied to new tasks or further training.

RESULT

Model Summary

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 3, 51, 39, 32)     6176
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 3, 24, 18, 64)     32832
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 22, 16, 64)     36928
_________________________________________________________________
flatten_2 (Flatten)          (None, 67584)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               34603520
_________________________________________________________________
dense_10 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_11 (Dense)             (None, 6)                 1542
=================================================================
Total params: 34,812,326
Trainable params: 34,812,326
Non-trainable params: 0
```

Fig3: Proposed Model Summary

Testing Output

```
Testing for 10 episodes ...
Episode 1: reward: 90.000, steps: 499
Episode 2: reward: 220.000, steps: 965
Episode 3: reward: 230.000, steps: 1224
Episode 4: reward: 240.000, steps: 715
Episode 5: reward: 635.000, steps: 1267
Episode 6: reward: 205.000, steps: 673
Episode 7: reward: 435.000, steps: 1096
Episode 8: reward: 315.000, steps: 1166
Episode 9: reward: 170.000, steps: 679
Episode 10: reward: 280.000, steps: 871
282.0
```

Fig4: reward generated and number of steps taken for 10 episodes

CONCLUSION

In the dynamic realm of Deep Reinforcement Learning (DRL), the fusion of deep neural networks and reinforcement learning has emerged as a potent force. The prowess of DRL is vividly demonstrated through its adeptness in mastering classic Atari 2600 games, showcasing adaptability and real-world applicability.

The exploration of diverse techniques in the literature survey, from human checkpoint replay to visual rationalizations, highlights the ongoing evolution of DRL strategies. Beyond gaming, these advancements resonate in domains like robotics, finance, and healthcare, addressing nuanced decision-making challenges.

The practical implementation steps provided, from leveraging OpenAI Gym for environment creation to constructing deep learning models with Keras, serve as a practical guide for enthusiasts and researchers. The integration of Convolutional Neural Networks (CNNs) amplifies DRL's versatility in tasks like image analysis for video game play.

The presented code snippets and notations act as a conduit between theory and application, offering valuable insights for those navigating the intricacies of DRL. Python modules like NumPy and TensorFlow, coupled with the Keras-RL library, streamline the development of DRL agents, enhancing accessibility and robustness.

In summary, the intersection of cutting-edge technologies propels DRL into a transformative era. From conquering pixelated screens to addressing real-world complexities, DRL's trajectory promises to reshape the landscape of artificial intelligence. As collaborative efforts within the AI community persist, new frontiers will undoubtedly unfold, bringing us closer to the realization of intelligent agents adept at navigating the complexities of our evolving world.

REFERENCES

1. Hosu, I. A., & Rebedea, T. (2016). Playing atari games with deep reinforcement learning and human checkpoint replay. *arXiv preprint arXiv:1607.05077*.

2. Weitkamp, L., van der Pol, E., & Akata, Z. (2019). Visual rationalizations in deep reinforcement learning for atari games. In *Artificial Intelligence: 30th Benelux Conference, BNAIC 2018,'s-Hertogenbosch, The Netherlands, November 8–9, 2018, Revised Selected Papers 30* (pp. 151-165). Springer International Publishing.

3. Mousavi, S. S., Schukat, M., & Howley, E. (2018). Deep reinforcement learning: an overview. In *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2* (pp. 426-440). Springer International Publishing.

4. Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26-38.

5. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

6. Lample, G., & Chaplot, D. S. (2017, February). Playing FPS games with deep reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 31, No. 1).

APPENDIX (Code)

```
import gym
import random
env = gym.make('SpaceInvaders-v4')
height, width, channels = env.observation_space.shape
actions = env.action_space.n
```

```
env.unwrapped.get_action_meanings()
```

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Convolution2D
from tensorflow.keras.optimizers import Adam
```

```
def build_model(height, width, channels, actions):
    model = Sequential()
    model.add(Convolution2D(32, (8,8), strides=(4,4), activation='relu',
input_shape=(3,height, width, channels)))
    model.add(Convolution2D(64, (4,4), strides=(2,2), activation='relu'))
    model.add(Convolution2D(64, (3,3), activation='relu'))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(actions, activation='linear'))
    return model
```

```
model = build_model(height, width, channels, actions)
model.summary()
```

```
from rl.agents import DQNAgent
from rl.memory import SequentialMemory
from rl.policy import LinearAnnealedPolicy, EpsGreedyQPolicy
```

```
def build_agent(model, actions):
    policy = LinearAnnealedPolicy(EpsGreedyQPolicy(), attr='eps', value_max=1.,
value_min=.1, value_test=.2, nb_steps=1000)
```

```
  memory = SequentialMemory(limit=100, window_length=3)
  dqn = DQNAgent(model=model, memory=memory, policy=policy,
                 enable_dueling_network=True, dueling_type='avg',
                  nb_actions=actions, nb_steps_warmup=100
                 )
  return dqn
```

```
import sys

print(sys.getrecursionlimit())
sys.setrecursionlimit(30000)
print(sys.getrecursionlimit())
```

```
dqn = build_agent(model, actions)
dqn.compile(Adam(lr=1e-4))
```

```
dqn.fit(env, nb_steps=100, visualize=False, verbose=2)
```

```
scores = dqn.test(env, nb_episodes=10, visualize=False)
print(np.mean(scores.history['episode_reward']))
```

```
del model, dqn
```