Learning playing a game using Deep Reinforcement Learning: Dangerous Dave

Mansi P Shah
University of Southern California
Los Angeles, California
mansipra@usc.edu

Akashkumar Patel
University of Southern California
Los Angeles, California
akashkum@usc.edu

Prabhat Mishra
University of Southern California
Los Angeles, California
prabhatm@usc.edu

Yash Hareshbhai Chanchad
University of Southern California
Los Angeles, California
chanchad@usc.edu

Harima Patel
University of Southern California
Los Angeles, California
harimapa@usc.edu

Jainam Divyesh Shah
University of Southern California
Los Angeles, California
jainamdi@usc.edu

ABSTRACT

This paper aims to sum up our findings while training our agent for an existing game named "Dangerous Dave". Dangerous Dave game is an Atari game where the agent explores the game environment to collect gems and a trophy to advance to a new level. We study how we can train our agent to automatically play by using a Reinforcement Learning methods like Deep Q-Learning and Actor-Critic Network. We learn about the environment using feature extraction and decide best moves in a given state using q-values. We also use experience-replay and epsilon greedy method to enable our agent to make its own decisions. The future scope of this project is to learn about the effectiveness of this approach for various other games similar to Dangerous Dave.

1 INTRODUCTION

Our goal is to train an AI agent that is supposed to collect as many diamonds and as many gems as possible along with dodging the obstacles like monsters, fire, trap, etc that come in his way to collect the trophy. Once he collects the trophy, the door to the next level opens up for him. The moves possible are left, right, jump, jump left, jump right.

A lot of AI agents exist for predictable and predefined game environments. We want to create an agent for a game that has infinite state space and can be used for other "platform games" as well. "Platform games" are the games which involve moving from one level to another by running and jumping while collecting rewards and avoiding obstacles with small changes.

Feature extraction would be used to get the current state of the environment. The game state is required to encapsulate spatial and temporal features. The spatial features include player, obstacles, gems and trophy location. The temporal features include score and remaining lives. Computer vision libraries are used to extract the score from the screen that would work as rewards. We used the Reinforcement learning method in which the environment gives the agent a state, the agent gives an action and an environment gives back a reward depending on how good the action was based on the state. We used Deep Q - Learning where Q-values (probability of success given a state and action) are estimated using Deep neural networks. Experience replay can be used to prevent Q-values from oscillating and diverging.

2 BACKGROUND/RELATED WORK

There is a lot of work done earlier for solving this problem but we drew inspiration for making our agent learn dangerous dave from [1]. Volodymyr Mnih, Et al., suggested that just by using game scores and pixels from images a deep Q network agent can surpass the performance of majority algorithms using a standard network architecture and tuning hyperparameters. We explain this section by giving you a background about the game we are trying to learn and what sort of work has been done earlier. DeepMind Technologies trained AI agent to play Atari games without changing algorithms and architecture. It used reinforcement learning to learn policies from high-dimensional sensory input like vision and speech. The model used was a convolutional neural network whose input is raw pixels and output is a value function estimating future rewards.[2]

2.1 Dangerous Dave

Since its debut in 1988, 'Dangerous Dave' has been one of the most popular games that allow the user to explore the deserted pirate's hideout and collect golden trophies that are hidden by Dave's rival Clyde Cooper, who stole them from Dave's clubhouse. For more fun, Dave can use the gun and jetpack, which can help him tackle the hurdles. This game was written for all Apple IIs in Apple IIe using 6502 & GraBASIC libraries having Hi-Res graphics and was published by UpTime. A graphic technique called "page-flipping" was used in Dangerous Dave to make the animation flicker-free. Nowadays page-flipping is handled transparently using DirectX and OpenGL. Dangerous Dave was the author's first game developed in ProDOS. It is interesting that out of 89KB which was the game's total size, 33 kB was just because of full screens for Help, Story, hints, etc. Different sounds of the explosion were created using different delays between clicking. Climbing and jumping were implemented using a small graphic trick which used lo-bit (purple, green, white0, black0) and high-bit (red, blue, white1, black1) colors. Figure 1.

2.2 Details and Definitions

Reinforcement Learning - In reinforcement learning the agent learns to achieve a goal in an uncertain environment by using trial

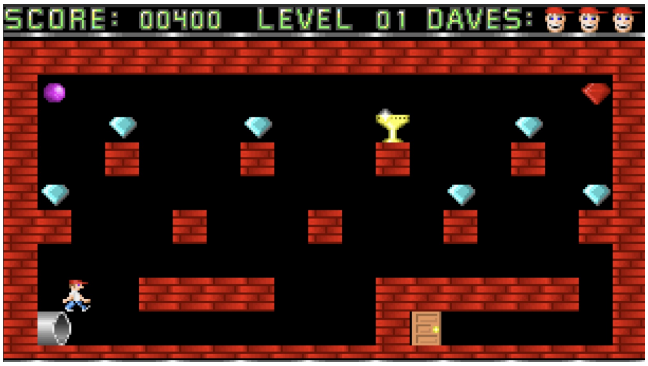

Figure 1: A still from Dangerous Dave, *Level 1.*

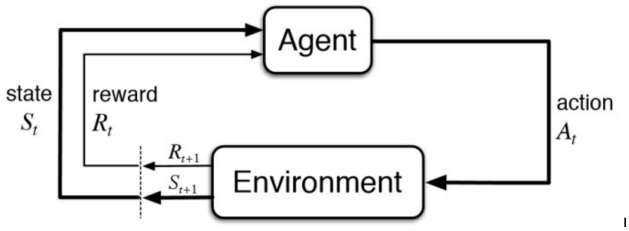

Figure 2: Reinforcement Learning Flow

and error methods. The AI agent gets rewards or penalties for its actions and the goal is to maximize the reward. The AI agent is given no hints on how to solve the problem. It differs from supervised learning whose model is trained with correct actions for given states.It follows the Markov Decision Process.Figure 2 [3]

Markov's Decision Process - The Markov property states that the future depends only on the present and not on the past. Almost all Reinforcement Learning problems can be modeled as MDP. MDP can be represented by the following elements.

(1) Set of states
(2) Set of actions
(3) Transition probability - probability of going from one state to another using some action
(4) Reward probability - probability of reward acquired by agent for moving from one state to another by taking some action.

Here policy function is mapping from state to action and q value indicates the goodness of an action in a given state.[4]

Deep Q-Learning - The Q-table has reward/probability of success for given state and action. In Q-Learning, we initialize the Q-Table, choose an action for each state and update the Q-Table using the bellman equation.

$$Q^{(k+1)}(s_t, a_t) \leftarrow (1-\alpha_t)Q^{(k)}(s_t, a_t)+\alpha_t r(s_t, a_t) + \gamma \max_{a'} Q^{(k)}(s_{t+1}, a') \quad (1)$$

In equation (1), action is a_t at state s_t at time t, r is the reward,t is time-step and α is learning rate.γ is a discount factor which causes rewards to lose their values over time so more immediate rewards are valued more.[5] Deep Q-Learning replaces Q-Table with a neural network, which gives all possible actions and q-values as output for a given state. The network weights are updated using the Bellman equation.[5]

Epsilon-Greedy Approach - At every step a random value is generated, if it is less than epsilon, any random action is chosen, otherwise, an action output by the trained Deep Q-Network is chosen. Initially, epsilon is 1 (every action is taken randomly), and as time progresses and the model is trained, epsilon decreases, so the probability of choosing the best move according to the model increases and the probability of choosing a random move decreases.[5]

Experience Replay-When the Agent interacts with the environment, the sequence of experience tuples can be highly correlated. The naive Q-learning algorithm that learns from each of these experiences tuples in sequential order runs the risk of getting swayed by the effects of this correlation.[6]

After every action [initial state, action, reward, final state] are stored in memory. To prevent q-values from oscillating or diverging, training is done on a random minibatch from experience, instead of using the latest experience. This method is called Experience Replay. It helps in breaking harmful correlations, learn some tuples multiple times and recall rare occurrences. [6]

Feature Extraction from Image -Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones (and then discarding the original features) without losing any information. CNN is a popular method used for feature extraction.[7]

Convolutional Neural Network -A Convolutional Neural Network can successfully capture Spatial and Temporal dependencies in the grabbed image and reduce images that can be processed easily without losing any features that are critical to defining the game environment. The kernel extracts high-level features by performing convolution operations. The average pooling layer reduces the spatial size of convolved features to reduce the processing power required.[8]

When we input an image into a ConvNet, each of its layers generates several activation maps. Activation maps highlight the relevant features of the image. Each of the neurons takes a batch of pixels as input, multiplies their color values by their weights, sums them up, and runs them through the activation function.[9] This is highlighted in Figure 3.

Actor-Critic - The idea is to split the model in two: one for computing an action based on a state and another one to produce the Q values of the action.
The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy. The critic evaluates the action by computing the value function. Those two models get better in their own roles as time passes. This works better than both value iteration and policy iteration, by converging faster.[10] This is highlighted in Figure 4.

Baseline Actor-Critic networks by OpenAI

Advantage Actor-Critic (A2C) - Q values can, in fact, be decomposed into two pieces: the state Value function V(s) and the advantage value A(s, a): Advantage function captures how better an action is compared to the others at a given state, the value function captures how good it is to be at this state. Instead of having the critic learn the Q values, we make him learn the Advantage values. The advantage of the advantage function is that it reduces the high variance of policy networks and stabilizes the model. [10]

Asynchronous Advantage Actor-Critic (A3C) - The simplicity, robustness, speed, and the achievement of higher scores in standard RL tasks of A3C made policy gradients and DQN obsolete. The key difference between A3C and A2C is the Asynchronous part. A3C consists of multiple independent networks with their own weights, who interact with a different copy of the environment in parallel. Thus, they can explore a bigger part of the state-action space in much less time. The agents are trained in parallel and update periodically a global network, which holds shared parameters. The main drawback of A3C is that some agents will be playing with an older version of the parameters. [10]

Actor-Critic Kronecker-Factored Trust Region (ACKTR) - ACKTR is a Policy Gradient method with trust region optimization. We apply the Kronecker-factored approximation to optimize both actor and critic. ACKTR is more sample-efficient, scalable, and computationally efficient. [11] It takes a step in the natural gradient direction which gives us the direction in parameter space that achieves the largest instantaneous improvement in the objective per unit of change in the output distribution of the network. [12]

Proximal Policy Optimization (PPO) - Proximal policy optimization (PPO) is a model-free, online, on-policy, policy gradient reinforcement learning method. It involves collecting a small batch of experiences interacting with the environment and using that batch to update its decision-making policy. Once the policy is updated with this batch, the experiences are thrown away and a newer batch is collected with the newly updated policy. The key contribution of PPO is ensuring that a new update of the policy does not change it too much from the previous policy. [13]

Deep Deterministic Policy Gradient (DDPG) - It is an off-policy reinforcement learning technique that combines both Q-learning and Policy gradients. It works well for continuous action settings. The actor is a policy network that takes the state as input and outputs the exact action, instead of a probability distribution over actions. The critic is a Q-value network that takes in state and action as input and outputs the Q-value. [14]
To estimate the policy and value function, a DDPG agent maintains four-function approximators:

(1) Actor μ (S) — The actor takes observation S and returns the corresponding action that maximizes the long-term reward.

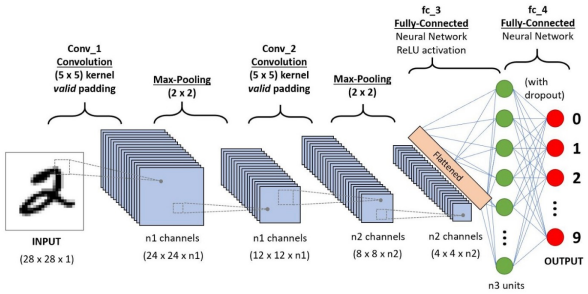

Figure 3: General Convolutional Neural Network

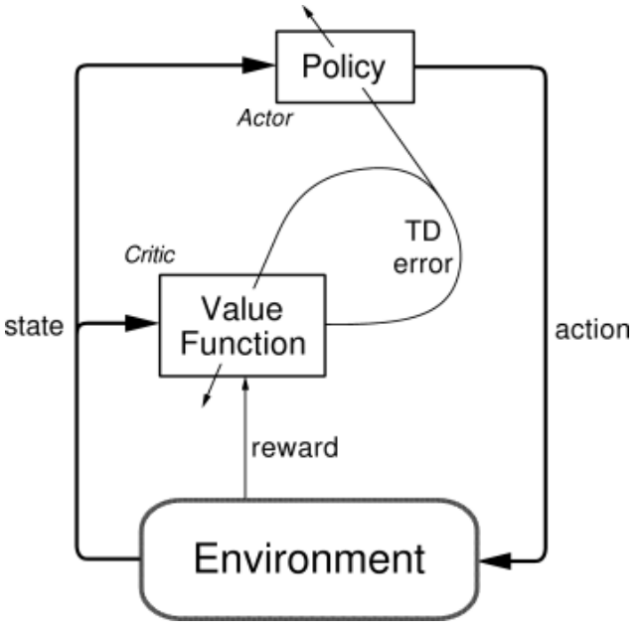

Figure 4: Actor Critic

(2) Target actor μ'(S) — To improve the stability of the optimization, the agent periodically updates the target actor based on the latest actor parameter values.
(3) Critic Q(S, A) — The critic takes observation S and action A as inputs and returns the corresponding expectation of the long-term reward.
(4) Target critic Q'(S, A) — To improve the stability of the optimization, the agent periodically updates the target critic based on the latest critic parameter values. [15]

3 ARCHITECTURE

We came up with two architectures. Both architectures include component object detection using optical character recognition(OCR), transfer learning from TensorFlow-based pre-trained models. In first architecture we have a deep neural network that helps to learn Q values for policy gradient Figure 5 while in the second architecture we have two neural networks - an actor network and a critic

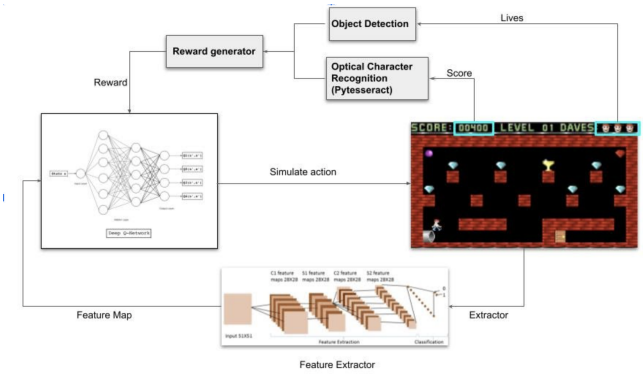

Figure 5: Architecture Model using DQN

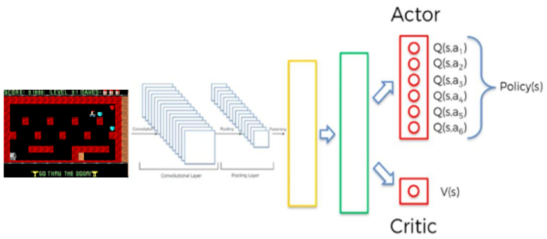

Figure 6: Architecture Model using Actor-Critic.

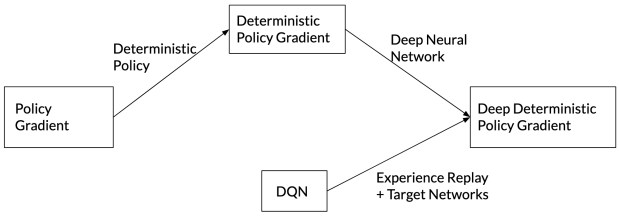

Figure 7: Architecture of DDPG.

network that reuses one neural network from previous architecture Figure 6. The second architecture implements Actor-Critic using Deep deterministic policy gradient. This just adds a neural network to previous DQN to select an action based on policy gradient. The flow of this algorithm is explained in Figure 7. Both of these use various open-source libraries like Tensorflow, Keras and Tessaract OCR. We also use open-source binary for Dangerous Dave to start the game from a python program. The project runs on a laptop with an i3 5th gen processor and 16 GB RAM. The python version used is 3.6.8.

The game environment is captured and we use an object recognition module to read scores from the game. We also combine this with object detection to detect lives and door and send it to the reward generation module to generate rewards for the action. The extractor observes the game environment using a pre-trained model(MobileNetV2) from tensor-flow to capture the game state using the screen. Both of these inputs are fed to the deep-q network which generates action for each state captured based on Q values.

4 METHODOLOGY

4.1 DQN

Initially, the DOS game Dangerous Dave will be called using Python code. Once the game starts in full screen, i.e. in the 1366 x 768 window, the Python program captures the game environment using ImageGrab. The image grabbed will then be fed to MobileNet Feature Extractor, a Convolutional Neural Network, to generate the Feature map which is used to represent the state of the game environment that would help in recognizing the location of the Agent (Dave), gems and obstacles.

The action to perform is selected using the Epsilon-Greedy approach, that is it is either the action predicted by Deep-Q-Network or any random action. With time epsilon is adjusted so that the probability of exploration (choosing the random move) decreases and exploitation (choosing move with maximum q-values) increases as the training progresses. The action can be one of the following: <jump right, jump left, move left, move right>. The initial model to predict possible actions and respective q-values is created using the Sequential model by Keras, adding dense layers with RELU activation function, and using SGD for optimization.

The action is then performed and observed in the actual Dangerous Dave DOS game environment and the resulting score is read from the top-left corner of the screen by converting color image to black and white image, resizing it and converting the image to text using pytesseract.

Based on the score, we assign rewards to that particular action and save the initial state, action, final state, reward for future training. We can prevent q-values from oscillating or diverging by training on the random batch size from experience, instead of using the latest experience using Experience Replay. On every action played, we check the number of lives remaining and also check whether the trophy has been collected by the agent or not. The complete experience is stored and fed back to Deep-Q-network to adjust its policy and learn different actions using Reinforcement Learning. The training occurs for around 1000 games with 60 moves for each game. We can see statistics of training with different batch sizes and number of moves in Table 1. Results from the loss analysis

Table 1: Statistics of training model

Batch-Size	No. of moves	Training time per game(seconds)
32	40	843.35
64	60	1211.75
128	80	2412.43

shows that larger the batch size for experience replay lower is the

Figure 8: Model complexity vs Prediction error

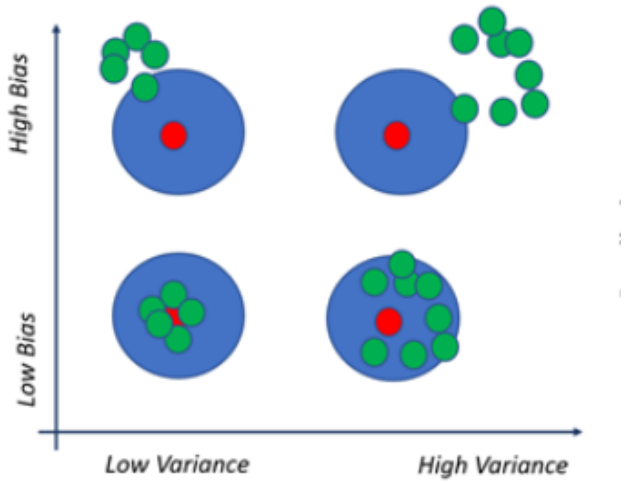

Figure 9: Variance vs Bias

training loss. Moreover, with larger batch size the rate of decrease in training loss is slow.The minimum training loss for higher batch size takes more epoch.This is depicted in following table.Table 2

Table 2: Batch size and training loss

Batch-Size	Min Training Loss
32	0.192
64	0.233
128	0.252

4.2 Actor-critic

The problem after applying DQN was that the number of moves after learning the environment was significantly high. Also, the agent was not able to determine the direction of the door after learning the environment. We can improve Q values by seeing more into future rewards. The future rewards include the reward of the door too.

Deep Deterministic Policy Gradient is a solution using Deep Q Learning in continuous action space. DDPG uses a deep neural network to estimate optimal policy directly from a given state and outputs a probability distribution. The previous method (DQN) tends to have high variance and low bias. The estimate is compounded resulting in wrong q values and hence the agent is not able to detect the last goal (door) which is required to advance to the next level. Policy-based methods have high variance and low bias. See Figure 8.

The DDPG algorithm exploits the experience replay and target network of the Deep Q neural network. The amount of work after implementing DQN reduces significantly because we just need to

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 10: DDPG Algorithm [16]

implement an actor neural network. The steps are updated using the following algorithm in the actor-critic environment.

(1) Create a runner object that handles the different environment steps.
(2) When the AI runner agent takes a step, each environment is updated with a corresponding action/step.
(3) The batch is generated from different experiences from the environment.
(4) Made a baseline model for DDPG and use batch experience for the training model.
(5) The resulting batch of experiences is passed for gradient calculation.
(6) The weights are updated and training is driven configuring various hyper parameters. The below is the algorithm of the process.

5 CHALLENGES

(1) Environment setup - Decide the version and install Tensorflow(2.4.0) an open-source library, Tesseract Optical Character recognition(5.00), Python(3.6.8), Pywin32 a windows extension to simulate random moves(300).

(2) Running DOS game from Python code - As we couldn't directly invoke the DOS game environment from the Python project, we extracted the required DOS executable files for the game and copied them to our project directory.

(3) Simulating random moves - We had to simulate the key press through Python code to move in the DOS game environment. The game has 5 moves - jump, jump left, jump right, move left, move right. For example, to simulate jump left, we press the up arrow key for 0.3 sec, release it and then press the left arrow key for 0.5 sec.

(4) Fetch 3 parameters from the game environment - To get the player's score from the top left, we crop that part from a screenshot of the game environment, convert the color image to black and white, resize the image and extract the score using pytesseract(5.00). To get the number of lives remaining, we divide the cropped image into 3 parts, each representing one life and compares it with a blank black image.

(5) Improving the CNN for feature extraction by tuning kernel size and number of floating-point operations in the neural network.

(6) Making the agent take a different move after reaching a dead end. The agent was taking the same move when it reached a dead end and was stuck in the same state because it selected the move with max q-value at a given state, so until the difference between the current and next state after an action is low, we select the next best action (action with next highest q value).

(7) Path detection to reach the door - The motivation behind this approach was to reduce the number of moves to reach the door after the player collects all the gems and a trophy. The next move is decided based on 2 values:

(a) Q-value from DQN
(b) The value returned by A*'s heuristic function which is the manhattan distance between player and door is considered as the heuristic function.

This implementation reduces the number of moves to reach the door after collecting all gems but it does not reduce the total number of steps of the entire game as it explores the right side of the game environment first then the left side of it and then finds the door.

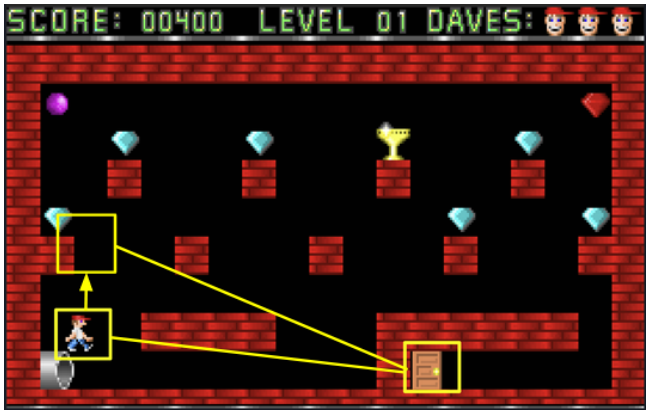

Figure 11: Path detection to reach door

(8) Solve the loop problem - The DQN Agent is used to select the best move (the move with the highest q value). The problem was that at times the agent would keep repeating the same moves forever and stay in the same region. Eg. jump right, jump right, jump left, jump left, jump right, jump right, jump left, jump left. We remember the state of the game environment that is output by the feature map and if the same sequence of states is repeated, it means there is a loop. This method does not detect loops if the gem is collected during the 1st iteration, as states would be different in the 1st and 2nd iteration. It would be able to detect the loop after the 3rd iteration.

(9) Deciding which Actor-Critic Network to use - We also explored A2C, A3C, ACKTR, and PPO (baseline networks) , but according to research training should be done on environments generated by openAI Gym. So we decided to use DDPG.

(10) Tuning hyperparameters (batch size, epoch, reward values, epsilon, decay for epsilon greedy) see Table 4

6 TRAINING STATISTICS AND RESULTS

(1) The epsilon greedy method is used to train the agent and to update the epsilon for each step using following equation.

$$\epsilon(t) = 0.01 + \exp^{(-currentGameCount * 0.001)} . \quad (2)$$

ϵ-decay in equation is 0.001.ϵ controls whether the chosen strategy is exploration or exploitation. If ϵ is high that means the agent moves randomly to explore and if ϵ is low then the

Table 3: Performance metric using A* algorithm

Methods	DQN	DQN with A* implementation
Average number of steps required to finish the game for 100 games	98	78

agent greedily performs high reward moves obtained from deep q network. This result matches with our experiment depicted in Figure 12. The initial value of epsilon is 1, that is all moves would be taken randomly and epsilon goes decreasing with time, so that moves suggested by DQN and DDPG would be selected with higher probability.

(2) We trained our agent for 1000 games with fixed number of steps. We ran the games with 40, 60, and 80 steps per game while training for DQN and DDPG. As time progresses, the agent gets trained better and makes better moves. The result is shown in Figure 13.

(3) The results of DQN are impressive indicating that the agent is able to evaluate action in a state and able to learn the environment but was not able to reach the door. After implementing Deep Deterministic Policy Gradient(DDPG) the number of moves in one game and number of episodes to learn the environment both reduced as expected.

The number of games/episodes required for training decreased from 632 in DQN to 321 in DDPG. See Figure 14.

The number of steps required to collect gems and reach the door decreased from average 100 steps for DQN to average 60 steps for DDPG. See Figure 15.

(4) The training loss of the actor network shows similarity with the DQN neural network but critical network loss looks quite unusual. This anomaly could be possible because of change in experiences present in the replay buffer. Following graphs depict the loss observed during the entire training for both the networks. The Actor and Critic networks are running concurrently and hence results may change slightly if run on different infrastructure or computing configuration Figure 16.

7 LIMITATIONS

(1) The environment in which the agent plays is static. Therefore, the approach suggested in this design might not work

Table 4: Hyperparameters for model.

Parameter	Value(DQN)	Value(DDPG)
Batch Size	32	128
Number of Steps	40, 60, 80	40, 60, 80
Number of Epochs	632	321
eps_start	1.0	1.0
eps_decay	0.001	0.001
Rewards	-1<=r<=1	-1<=r<=1
Actor Learning rate	NA	0.001
Critic Learning rate	0.001	0.001
Tau	NA	0.125

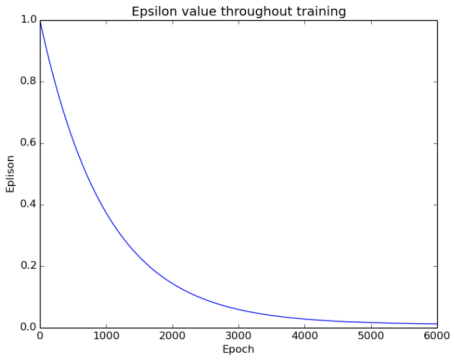

Figure 12: Epsilon value throughput training

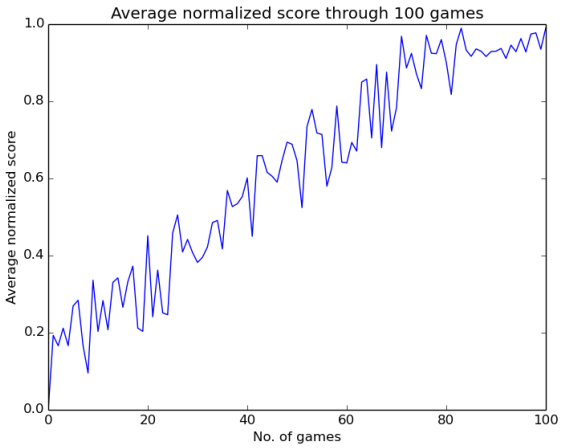

Figure 13: Average normalized score

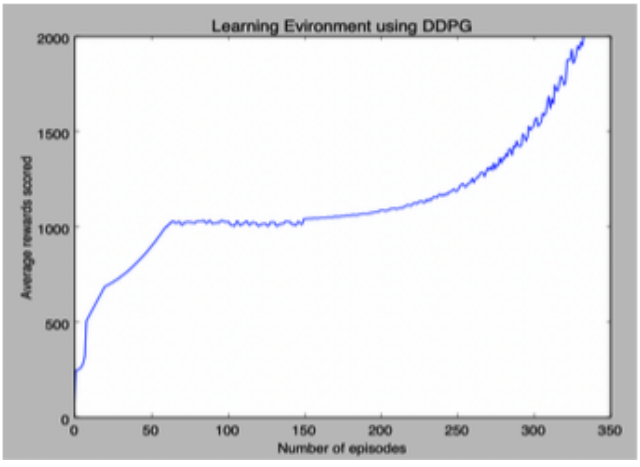

Figure 14: Learning Environment using DDPG

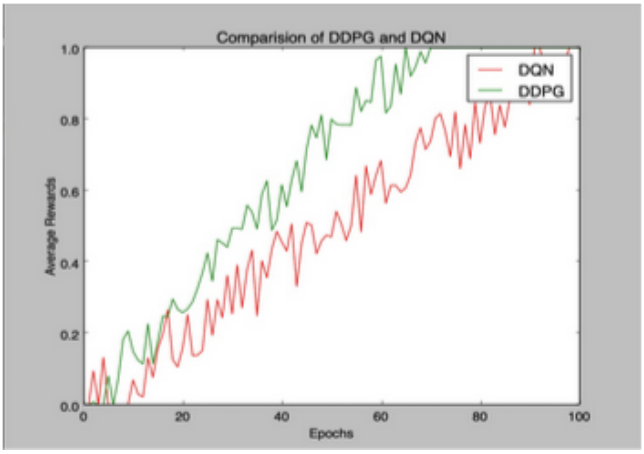

Figure 15: Comparison of DDPG vs DQN

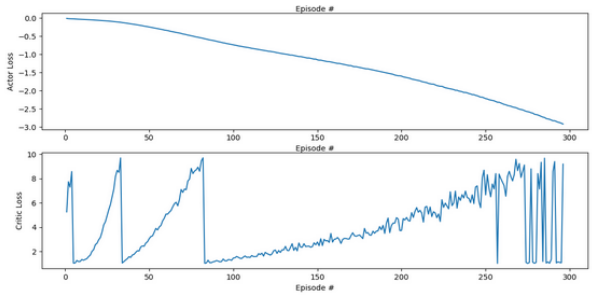

Figure 16: Training Loss in Actor Critic(DDPG)

for a dynamic environment. Dangerous Dave game has levels that have obstacles like gun-fire, fires, bullet and water which not only makes the environment adversarial but dynamic too. We have not trained our agent to play on these levels. We have to change reward system including negative rewards for actions that leads to reduction in life.

(2) The agent still doesn't play as good as human because it does not sense future outcomes in environment, rather just looks at short term goals. Also, we did not explore running the model on GPU based run time where the training time can be significantly reduced.

(3) Other complex Machine Learning techniques which would help agent to perform as nearly as human have not been explored.

(4) If environment can be represented in much more simpler way rather than using raw pixels then we could use baseline models tested and verified by OPEN AI community like A2C, A3C, ACKTR, PPO.

8 CONCLUSION

Current work involves a design where the agent has no prior knowledge of the game and it learns to play it through image pixels, scores, and the number of lives the agent has in the game. This neural network architecture can be tuned for this game and any other game by just configuring various hyper parameters. The experience replay algorithm and target network enhanced the experience of the agent and led to a stable learning network.This concludes that we can train even a low resolution game like dangerous dave using various techniques of image resolution and applying machine learning principles to train the agent to play decently and learn the environment. DQN can be used to learn an Atari game whose reward system is accurate and flawless. However, DDPG performs well when strategy is known for the game and it doesn't change over time with reward system used in DQN. The current implementation of DDPG uses minibatch taken uniformly from the replay buffer and enhances the experience of agent if sampling happens correctly.

9 FUTURE SCOPE

(1) See if this approach will work for other Atari games similar to dangerous dave where we need to avoid obstacles and collect rewards to move to the next level.
(2) Tune hyperparameters to get better results.
(3) Try using prioritized replay buffer.
(4) Improve the fetching of score and number of lives remaining from game environment.
(5) Train the agent for next levels of Dangerous Dave game.
(6) Try more algorithms such as Trust Region Policy Optimization (TRPO), Proximal Policy Optimization, Distributed Deterministic Policy Gradients, and Truncated Natural Policy Gradient (TNPG) to see if we attain better performance for this game.

REFERENCES

[1] Human-level control through deep reinforcement learning, https://www.nature.com/articles/nature14236
[2] Playing Atari with Deep Reinforcement Learning, https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf
[3] What is Reinforcement Learning, a complete guide, https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/
[4] Markov chains and Markov Decision Process, https://medium.com/@sanchittanwar75/markov-chains-and-markov-decision-process-e91cda7fa8f2
[5] Deep Q-Learning Tutorial: minDQN, https://towardsdatascience.com/deep-q-network-dqn-ii-b6bf911b6b2c
[6] Deep Q-Network (DQN)-II, https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc
[7] Feature Extraction Techniques, https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be
[8] A Comprehensive Guide to Convolutional Neural Network - the ELI5 way, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[9] What are Convolutional Neural Networks (CNN) ? https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/
[10] The idea behind Actor-Critics and how A2C and A3C improve them https://theaisummer.com/Actor$_c$$ritics$/
[11] RL — Actor-Critic using Kronecker-Factored Trust Region (ACKTR) Explained https://jonathan-hui.medium.com/rl-actor-critic-using-kronecker-factored-trust-region-acktr-explained-670777ec65ce

[12] OpenAI Baselines:ACKTR A2C
https://openai.com/blog/baselines-acktr-a2c/
[13] Top highlight Proximal Policy Optimization Tutorial (Part 1/2: Actor-Critic Method)
https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6
[14] Deep Deterministic Policy Gradient (DDPG): Theory and Implementation
https://towardsdatascience.com/deep-deterministic-policy-gradient-ddpg-theory-and-implementation-747a3010e82f
[15] Deep Deterministic Policy Gradient Agents
https://www.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html
[16] DDPG Algorithm
https://arxiv.org/pdf/1509.02971.pdf