**Learning Atari Game Strategies Using Deep**

**Reinforcement Learning**

Machine Learning Techniques

**Project Report**

Our aim was to create an AI agent which learns to play a number of Atari games well using the same set of hyper parameters. Our primary motivation behind choosing this topic was the vast potential of deep reinforcement learning to learn agents that could perform some specific set of tasks in vastly different environments, looking at the state provided and the rewards accumulated. Another important reason was that this was a topic not covered in class, so it gave us a first hand experience of exploring a topic on our own, and marveling at the beauty of machine learning (in this case, no separate dataset is required, and this differs substantially from the supervised and unsupervised settings that we are used to).

**Background**

**Reinforcement Learning** is a machine learning paradigm based on learning by doing, concerned

with how agents ought to take actions in an environment so as to maximize some notion of

cumulative reward. The following definitions are loosely based on David Silver’s lecture series

on RL in University College London.

A basic reinforcement learning model consists of:

• A set of environment states S

• A set of actions A

• Rules of transitioning between states

• Rules that determine the scalar immediate reward of a transition

• Rules that describe what the agent observes.

In reinforcement learning, at each time step t the agent:

• Executes action At

• Receives observation Ot

• Receives scalar reward Rt

And the environment:

• Receives action At

• Emits observation Ot+1

• Emits scalar reward Rt+1

**Reward**: A reward Rt is a scalar feedback signal which indicates how well agent is doing at step t. The agent’s job is to maximize expected cumulative reward.

**History**: History is the sequence of observations, actions, rewards Ht = O1,R1,A1, ...., At−1,Ot,Rt

**State:** State is the information used to determine what happens next. Formally, state is a function of history: St = f(Ht)

**Policy**: A policy(π) is the agent’s behavior or in other words, a distribution over actions given states.π(a|s) = P[At = a|St = s]

**Value Function:** Value function is a prediction of future reward. vπ(s)=Eπ[Rt+1 + γRt+2 + γ2Rt+3 + ....|S − t = s]

**Model:** A model predicts what the environment will do next: P predicts the next state: Pass = P[St+1 = s |St = s, At = a]

**R predicts the next reward:** Ras = E[Rt+1|St = s, At = a]Markov Property: A state St is Markov if and only if P[St+1|S1, ...., St]. The state captures all relevant information from the history.

**State Transition Matrix:** For a markov state s and successor state s’, the state transition probability is defined by:P(s, s ) = P[St+1 = s |St = s]

**Return**: The return Gt is the total discounted reward from time-step t. Gt = Rt+1 + γRt+2 + ... = ∑∞k=0 γk Rt+k+1 where γ ∈ [0,1]

**Markov Decision Process:** A Markov Decision Process (or Markov Chain) is a tuple < S,A,P,R,γ > where:

• S is a finite set of states.

• A is a finite set of actions.

• P is a state transition probability matrix.

• R is a reward function, Ras = E[Rt+1|St = s, At = a]

**State-Value Function:** The state-value function vπ(s) of an MDP is the expected return startingfrom state s, and then following policy π: vπ(s)=Eπ[Gt|St = s]Action-Value Function: The action-value function qπ(s, a) is the expected return starting fromstate s, taking action a and then following policy π: qπ(s, a)=Eπ[Gt|St = s, At = a]

**Bellman Equation for optimal Q function:** Q∗(s, a) = E[r + γmaxa’ Q(s’ ,a’ )|s, a] After covering these definitions, lets describe some learning methods and algorithms:

**Monte-Carlo Learning**

After an episode S1,A1,R2, ...., ST, for each state St with return Gt:

• Increment counter N(s) = N(s)+1

• Update V (s) incrementally as

V(St) = V(St) + 1/N(St)(Gt − V(St))

**Temporal-Difference(TD) Learning**

In TD learning, after an episode S1,A1,R2, ...., ST, for each state St with return Gt: Update value V (St) towards estimated return Rt+1 + γV (St+1) as: V(St) = V (St) + α(Rt+1 + γV (St+1) − V (St)) Rt+1 + γV (St+1) is called the TD target and δtRt+1 + γV(St+1) − V(St) is called the TD-error

Action Value Function Approximation by Stochastic Gradient Descent

For solving large MDPs, estimate value function with function approximation ˆq(s, a, w) ≈ qπ(s, a)So we will adjust w in direction of negative gradient to minimize mean-squared error between approximate-value function ˆq(s, w, a) and true value function qπ(s, a) In TD learning, the TD target is Rt+1 + γQ(St+1,At+1)δw = α(Rt+1 + γˆq(St+1,At+1,w) − ˆq(St,At,w))∇wˆq(St,At,w)

**What We Tried?**

We tested our algorithms on ’Pong’ - A two-dimensional game similar to table tennis.

Using the difference of two consecutive frames as states to capture the motion, our agent uses Policy gradients to learn an optimal policy by moving the policy in the gradient(direction) indicated by the rewards.

We broadly used two methods -

1.Q learning

2.Policy Gradients.

Q learning approximates the maximum expected return for performing an action at a given state using an action-value (Q) function.

Policy gradient methods rely upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) by gradient descent.

Image frame:  210x160x3 (100,800 sounds kind of impossible)

Action:  move the paddle UP or DOWN (i.e. a binary choice).

Reward:  +1 reward if ball hit, -1 reward if ball missed, or 0 otherwise.

Goal: To move the paddle so that we get maximum cumulative reward

**Training protocol**

For Pong, we initialized the policy network with some W1, W2 and played 100 games of Pong.

Let’s us assume that each game is made up of 200 frames.

So, in total we’ve made 20,000 decisions for going UP or DOWN

For example, Suppose Won 12 games and lost 88.

So,

200\*12 = 2400 Decisions we made in the winning games and do a positive update.

200\*88 = 17600 Decisions we made in the losing games and do a negative update.

The network will now become slightly more likely to repeat actions that worked, and slightly less likely to repeat actions that didn’t work.

**Things That We Learned**

This project introduced us to a new paradigm in machine learning - Reinforcement Learning.

The following is a broad list of concepts and some tools that we learned:

• Basic terminology in reinforcement learning like rewards, history, environment, markov states, policy, value function, model and discount factor.

• Bellman Expectation and Optimality Equations for value function and action value function.

• Model Free Prediction - Monte Carlo Learning and Temporal-Difference Learning

• Policy Gradient Methods for RL

• Experience Replay

• Exploration and Exploitation

• Open AI gym