horizontal line

Deep Q-learning trained model for playing classical Atari

**Members:**

Rahul Shiv Chand 2015A7PS0163P

Chetan S. Jois 2015A4PS0261P

Kushagra Jain 2015A7PS0190P

# **OVERVIEW**

Reinforcement learning is a machine learning paradigm which takes inspiration from how humans learn to take actions. Deep reinforcement learning is a relatively new branch which applies deep neural networks as functional approximators. This combination enables the concepts of RL to be scaled up to solve real life problems. This project uses deep RL to create an agent capable of achieving human level control in classic Atari games like breakout.

# **Objective**

To build a model which can achieve human level control in classic Atari games using policy gradient and Q-learning.

# **Data Description**

In reinforcement learning data is made up of **episodes.** Each episode is a set of 3-tuples given below

* **St** : State at time ‘t’ , a state is the description of environment at any given time. In case of breakout a state is made up of the last 2 frames.
* **At** : Action taken at time ‘t’. **At** is the action that out agent selected at time ‘t’. In case of breakout an action is either moving “LEFT” or “RIGHT”
* **Rt** : Discounted reward at time ‘t’. **Rt** is the discounted reward we obtain if we take action At in state St.

Each episode is equivalent to one play of the game (start of game to end of game). We obtain these episodes by playing against a computer opponent.

# **Brief description of techniques and terminology used**

* **Q (s,a) :** Q(s,a) denotes the total reward we will obtain if we take action **“a”** in state **“s”.**
* **Policy (p) :** Policy at each step is the action which will maximize our total reward.

# The aim of reinforcement learning is to find the value of **Q(s,a)** for each **‘s’** and **‘a’**, that is at each moment should the paddle of ping-pong be moved LEFT or RIGHT to win the game. Finding the optimum solution is computationally expensive, therefore the methods below our used to find approximate solutions

* **Q-learning** : Q-learning is an iterative method which approximates **Q(s,a)** as:

Q ( s , a ) = r(s, a ) + ⋎ V\*(Q(s, a ) )

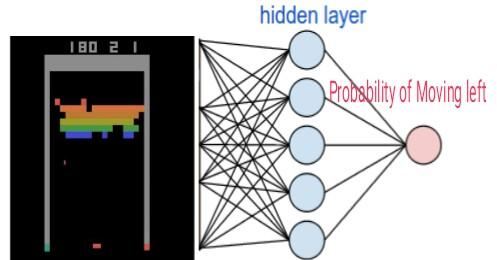
We use Q-learning to iteratively find an approximate solution

* **Policy gradient** : In policy gradient rather than finding for each ‘s’ and ‘a’ what is the reward obtained, we directly try to find at each state which action must be selected so as to maximize total reward.

**How is a state represented?**

In both the methods above we use the term “state” but how is a state in case of breakout represented?

In breakout a state is the raw pixels (a snapshot of the game at any given time), these raw pixels are used as state because they describe what is happening in the game at any moment of time. Using just the current snapshot of the game can result in ambiguity, for example just looking at the image below we cannot decide which way the ball is moving. Therefore we use the subtraction of last 2 frames as our state.



# 

# **Modelling details and Motivation behind models used**

* To map each state, action pair to a reward value we use a CNN ( Convolutional neural network). Our CNN takes the raw pixels ( snapshot of the game) as input and predicts the reward we will obtain for each action we can take.

# 

# 

# *Image taken from David Silver’s(UCL) lecture on Reinforcement learning*

* The **‘s’** in the image above is the snapshot of our game.
* We use a CNN because our state consist of a set of game snapshots(images), and CNN’s have shown great potential in tasks related to image recognition and feature extraction.
* Our CNN consists of:
  + 1 convolutional layer with:
    - Filters = 32
    - Kernel size = 8
    - Strides = 4
  + 1 convolutional layer with:
    - Filters = 64
    - Kernel size = 4
    - Strides = 2
  + 1 convolutional layer with:
    - Filters = 64
    - Kernel size = 3
    - Strides = 11

* + 1 convolutional layer with:
    - Filters = 64
    - Kernel size = 3
    - Strides = 11

* + 1 convolutional layer with:
    - Filters = 64
    - Kernel size = 3
    - Strides = 11
  + 1 fully connected layer with 512 output nodes
  + 1 final fully connected layer with 4 output values: ‘LEFT’, ‘RIGHT’, ‘FIRE’, ‘NO ACTION’

# **Results**

* The agent succeeded in achieving, on an average, 30.20 points.
* This is better than humans considering an average human player can only achieve 29.8 points per game.
* The model was trained overnight over 100,000 games with around 200 frames per game with one decision per frame making it 20,000,000 data points in total.