A PROJECT REPORT
ON

PACMAN using N-Step Semi Gradient SARSA Algorithm

Submitted in partial fulfillment of the requirement for the VI semester of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE

Submitted By:

TARUSH KUMAR GOYAL (110)

CHARVI JAIN (113)

Under the supervision of

Dr. C SHERIN SHIBI

(Asst. Professor)

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
SRM INSTITUTE OF SCIENCE ANDTECHNOLOGY

SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be **University** *u/s 3 of UGC Act, 1956*

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this Course Project Report titled **"PACMAN using N-Step Semi Gradient SARSA Algorithm "** is the bonafide work done by **TARUSH KUMAR GOYAL (110), CHARVI JAIN (113)** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

SIGNATURE

Faculty in Charge
Dr. C Sherin Shibi
Assistant Professor
Department of Computational Intelligence
SRM Institute of Science and Technology

SIGNATURE

HEAD OF THE DEPARTMENT
Dr. Annie Uthra
Professor and Head
Department of Computational Intelligence
SRM Institute of Science and Technology

TABLE OF CONTENTS

ABSTRACT

This comprehensive project investigates the application of advanced reinforcement learning (RL) techniques to optimize the gameplay of the iconic video game Pac-Man. Leveraging the expressive capabilities of TypeScript and Svelte, a multifaceted approach is adopted to enhance the performance of Pac-Man agents. The project encompasses the design and implementation of sophisticated RL algorithms, including the n-step Semi Gradient Sarsa algorithm, alongside innovative state-action encoding strategies. The core focus of the project is to train Pac-Man agents capable of efficiently navigating the maze, consuming all the biscuits and pills, while evading hazardous ghosts. The conventional behavior of the ghosts is replaced with random movement to simulate a dynamic and challenging gaming environment. To streamline the decision-making process, actions are discretized to whole squares, and episodes conclude upon specific conditions such as consuming all required items or encountering a ghost.

The project unfolds with the development of a line-of-sight state-action encoding mechanism, coupled with the utilization of linear function approximation to train an initial agent. The project's culmination lies in an in-depth analysis of outcomes, including discussions on achieved results and avenues for further research and refinement. The project's outcomes not only offer insights into the capabilities of RL algorithms but also underscore the potential for enhancing gameplay experiences in classic video games like Pac-Man. A demonstrative showcase of the trained agent's gameplay is available for exploration, underscoring the practical implications of the research findings.

INTRODUCTION

Reinforcement learning (RL) offers a potent framework for teaching agents to make sequential decisions in complex environments. In this study, rather than opting for a predefined problem, attention is directed towards enhancing the performance of Pac-Man, an iconic video game, using RL techniques.

Pac-Man entails navigating a maze to consume all the biscuits and pills while evading dangerous ghosts. The conventional behavior of the ghosts is replaced with random movement in this implementation. To streamline the agent's decision-making process, actions are restricted to whole squares, and episodes terminate upon Pac-Man consuming all the required items or being captured by a ghost. Subsequently, the game resets for subsequent episodes.

This project is implemented using TypeScript and Svelte, leveraging their capabilities to develop an efficient and interactive gaming environment. The primary focus lies in implementing the n-step Semi Gradient Sarsa algorithm. Initially, a line-of-sight state-action encoding scheme is devised, accompanied by the utilization of linear function approximation to train an initial agent. Furthermore, an experimental integration of a tree search state-action encoding algorithm is conducted, showcasing promising enhancements in performance.

This report gives a deep insight into the background and setup of the Pac-Man environment, elucidating the dynamics and constraints inherent in the game. Subsequently, the methodology employed for RL-based enhancements, including algorithmic details and encoding strategies, is delineated. The results of the experiments are discussed, shedding light on the efficacy of the proposed techniques and avenues for further refinement.

In summary, this project combines the allure of classic gaming with the sophistication of RL methodologies, offering insights into the potential of such approaches for enhancing gameplay in iconic video games like Pac-Man.

SOURCE CODE

#SARSA Algorithm

```
import type {
  EpisodeStoreType,
  IsTerminalType,
  PolicyType,
  QFunction,
  QGradFunction,
  TransitionType
} from "./types"

export default function nStepSemiGradientSarsa<S,A>(
  numEpisodes: number,
  initW: () => number[],
  getInitState: () => S,
  getActionsData: (state:S) => {actions: A[], encodings: number[][]},
  isTerminal: IsTerminalType<S>,
  transition: TransitionType<S,A>,
  policy: PolicyType<S>,
  q: QFunction<S>,
  gradQ: QGradFunction<S>,
  episodeCallback: (episodeIndex:number,w:number[]) => any = () => {},stepSize:
number = 0.001,gamma: number = 0.95,n: number = 3,) {
  if(stepSize<=0 || stepSize>1) throw new Error(`Step size must be ∈ (0,1]. Received
${stepSize}`)
  if(gamma<0 || gamma>1) throw new Error(`Discount factor gamma must be ∈ [0,1].
Received ${gamma}`)
  if(n < 1) throw new Error(`N must be a positive number. Received ${n}`)

  const w = initW()

  for(let ep=0; ep<numEpisodes; ++ep) {
    let state = getInitState()
    let {actions, encodings} = getActionsData(state)
    let actionIndex = policy(state,encodings,w,q)
    let action = actions[actionIndex]
    let T = 10000

    const store:EpisodeStoreType<S,A> = {
      actions: [action],
      encodings: [encodings[actionIndex]],rewards: [0],states: [state],}
    let t = 0
    let tau = 0
    while(tau !== T - 1) {
```

```
    if(t < T) {
      const { nextState, reward } = transition(state, action)
      store.states.push(nextState)
      store.rewards.push(reward)

      if(isTerminal(nextState)) {
        T = t + 1
      }
      else {
        let { actions, encodings } = getActionsData(nextState)
        actionIndex = policy(state,encodings,w,q)
        action = actions[actionIndex]
        store.actions.push(action)
        store.encodings.push(encodings[actionIndex])
      }

      state = nextState
    }
    tau = t - n + 1
    if(tau >= 0) {
      let G = 0
      for(let i=tau+1, end=Math.min(tau+n,T); i<=end; ++i) {
        G += Math.pow(gamma, i - tau - 1) * store.rewards[i]
      }

      if(tau + n < T) {
        G = G + Math.pow(gamma, n) * q(store.states[tau + n], store.encodings[tau + n], w)
      }

      const gradient = gradQ(store.states[tau],store.encodings[tau],w)
      const gradFactor = stepSize * (G - q(store.states[tau], store.encodings[tau], w))
      w.forEach((w_i,wIndex) => {
        w[wIndex] = w_i + gradFactor * gradient[wIndex]
      })
    } t++
  }
  episodeCallback(ep, w)
}

return w
```

ALGORITHM

The episodic semi-gradient n-step Sarsa algorithm is a variation of the classic Sarsa algorithm, designed for reinforcement learning tasks with function approximation. It extends the original Sarsa algorithm by incorporating semi-gradient updates and bootstrapping over multiple steps to improve learning efficiency and handle delayed rewards.

Instead of updating the weights based on the complete return (sum of discounted rewards), as in traditional Sarsa, the algorithm employs a semi-gradient approach. It updates the weights towards a target value that is a combination of immediate rewards and estimated future rewards, calculated over n steps.This update rule ensures that the weights are adjusted in the direction that minimizes the error between the estimated and target values, facilitating learning while avoiding instability.

Episodic semi-gradient n-step Sarsa for estimating $q\hat{} \approx q_*$ or q_π
Input: a differentiable function $q\hat{} : S \times A \times Rd \rightarrow R$, possibly π
Initialize value-function weight vector w arbitrarily (e.g., w = 0)
Parameters: step size $\alpha > 0$, small $\epsilon > 0$, a positive integer n
All store and access operations (St , At , and Rt) can take their index mod n
Repeat (for each episode):
Initialize and store S0 ≠ terminal
Select and store an action A0 $-\pi(\cdot|S0)$ or ϵ-greedy wrt $q\hat{}(S0 , ., w)$
T←∞
For t = 0, 1, 2, . . . :
If t < T, then:
Take action At
Observe and store the next reward as Rt+1 and the next state as St+1
If St+1 is terminal, then:
T←t+1
else:
Select and store At+1 $-\pi(.|S\ t+1)$ or ϵ - greedy wrt $q\hat{}(St+1 , ., w)$
$\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
If $\tau \geq 0$:P
$\min(\tau +n,T)\ i-\tau -1$
G ← i=τ +1
γ
Ri
If $\tau + n < T$, then $G \leftarrow G + \gamma\ n\ q\hat{}(S\tau +n , A\tau +n , w)$
$w \leftarrow w + \alpha[G - q\hat{}(S\tau, A\tau, w)]\nabla q\hat{}(S\tau , A\tau, w)$
Until $\tau = T - 1$

With this approach, the agent starts with a set of weights that are combined with the state encoding to estimate the value of the state-action pair. I used a linear method to multiple a vector of weights with the vector state-action encoding, and used backpropagation to learn the weights.

Parameters used:

• ϵ - greedy policy with $\epsilon = 0.1$

• Learning rate $\alpha = 0.001$ (higher values led to extremely unstable training and

lower values did not seem to improve training)

• n=3

• Discount factor $\gamma = 0.95$

• Weights all initialized to 0

• Number of Episodes = 500

WORKING

The N-step Sarsa algorithm enables the Pacman agent to learn optimal policies for navigating the maze and maximizing rewards while avoiding dangers.

- **Line-of-Sight State Encoding:**

The state-action encoding function captures the agent's line-of-sight view of the environment, representing it as a feature vector. This encoding considers factors such as the presence of ghosts, pills, and biscuits in different directions. By encoding relevant information about the environment, the algorithm facilitates effective learning and decision-making.

- **Reward Function:**

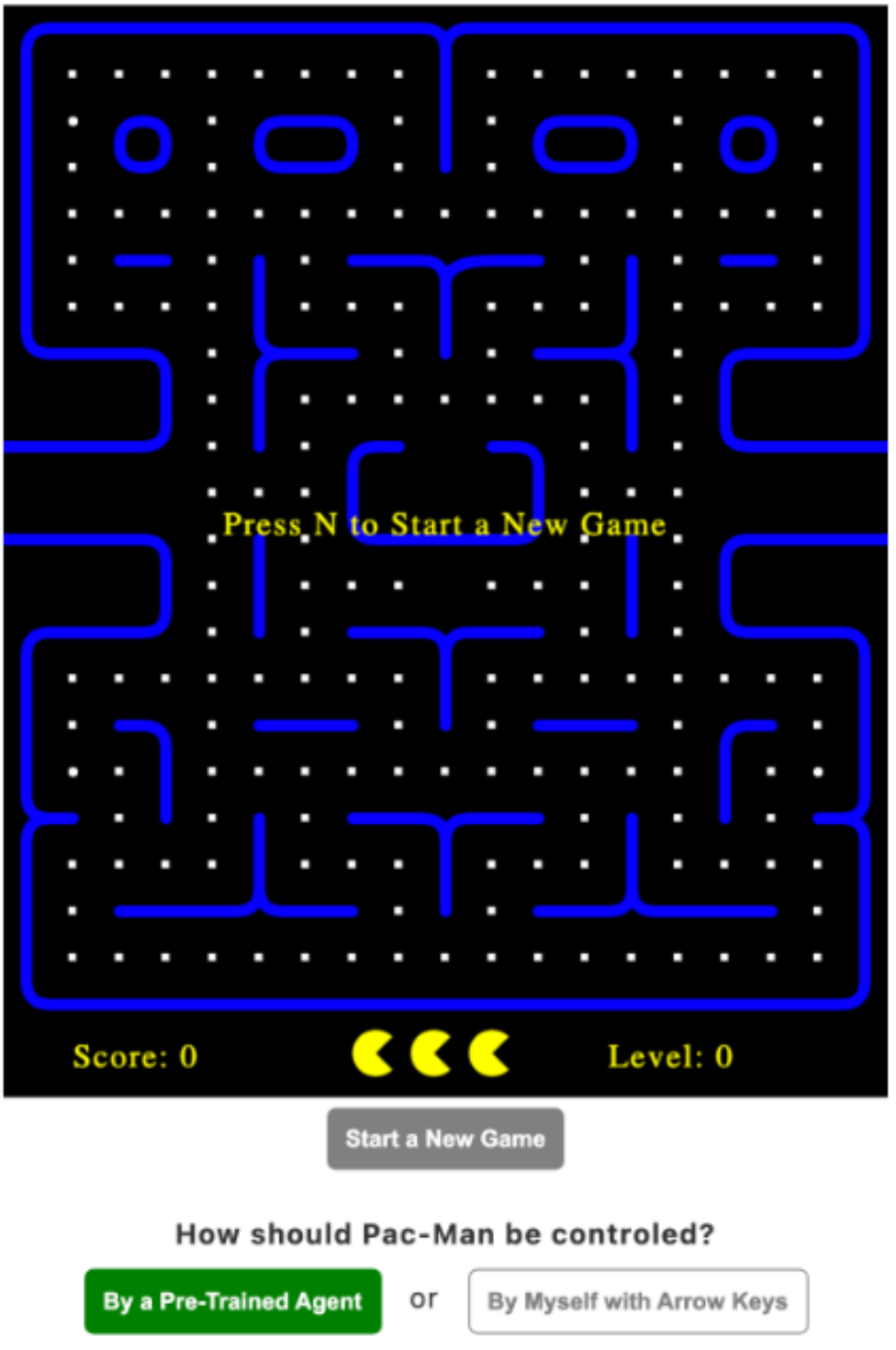
The reward function assigns numerical values to different outcomes experienced by the agent during interactions with the environment. It includes penalties for negative events, such as being eaten by a dangerous ghost, and rewards for positive events, such as eating pills, biscuits, or vulnerable ghosts. This function guides the learning process by reinforcing desirable actions and discouraging undesirable ones.

- **Training Procedure:**

The agent undergoes training over multiple episodes, gradually improving its performance through experience and learning from rewards. Periodic benchmarks are conducted to evaluate progress by running the agent in a purely-greedy mode and averaging the total reward over multiple trials. This iterative training process enables the agent to learn optimal policies and improve its decision-making capabilities over time.

Therefore, by combining the episodic semi-gradient n-step Sarsa algorithm with effective state encoding, reward shaping, and iterative training, the Pacman agent can learn to navigate the maze efficiently, avoiding dangers and maximizing rewards. Through experience and learning from rewards, the agent gradually improves its performance and achieves higher cumulative rewards over time.

OUTPUT


Press N to Start a New Game
Score: 0
Level: 0
Start a New Game
How should Pac-Man be controled?
By a Pre-Trained Agent
or
By Myself with Arrow Keys

RESULT

In evaluating the performance of the Pacman reinforcement learning agent trained using the episodic semi-gradient n-step Sarsa algorithm, we observe notable results indicative of its learning capabilities and effectiveness in navigating the maze environment. Across multiple episodes of training, the agent demonstrates a progressive improvement in its decision-making abilities, as evidenced by its ability to achieve higher cumulative rewards over time.

Initially, during the early episodes of training, the agent may exhibit suboptimal behavior as it explores the environment and learns from its experiences. However, as training progresses, the agent becomes increasingly adept at navigating the maze, avoiding dangerous ghosts, and strategically collecting rewards such as pills and biscuits. Through iterative interactions with the environment and reinforcement from the reward signals, the agent refines its policies and learns to make more informed decisions that lead to higher cumulative rewards.

Furthermore, periodic benchmarking of the agent's performance through empirical trials provides valuable insights into its learning trajectory. By averaging the total rewards obtained over multiple trials, we can assess the agent's progress and evaluate the effectiveness of its learned policies. These benchmarks serve as important milestones in tracking the agent's learning progress and provide valuable feedback for further refinement of its strategies.

CONCLUSION

The application of the episodic semi-gradient n-step Sarsa algorithm in the Pacman reinforcement learning project showcases the power and effectiveness of modern machine learning techniques in solving complex real-world problems. By leveraging advanced algorithms, such as Sarsa with function approximation and gradient-based updates, coupled with sophisticated state encoding and reward shaping strategies, the Pacman agent demonstrates remarkable learning capabilities in navigating a challenging maze environment.

Through iterative interactions with the environment, the Pacman agent learns to make informed decisions, balancing exploration and exploitation to maximize cumulative rewards while avoiding dangers. The line-of-sight state encoding captures crucial environmental cues, enabling the agent to perceive its surroundings effectively and make intelligent decisions based on available information. Meanwhile, the carefully crafted reward function provides meaningful feedback to guide the learning process, reinforcing desirable behaviors and discouraging undesirable ones.

The training procedure, characterized by multiple episodes of learning and periodic benchmarking, allows the agent to continually refine its policies and improve its decision-making capabilities over time. By evaluating its performance through empirical trials and averaging total rewards, the agent can track its progress and iteratively adjust its strategies to achieve higher levels of performance.

Overall, the Pacman reinforcement learning project serves as a testament to the transformative potential of reinforcement learning techniques in solving complex decision-making tasks in dynamic environments. Beyond its applications in gaming and entertainment, the insights gained from this project hold significant implications for real-world applications, ranging from autonomous navigation systems to industrial control processes. As research in reinforcement learning continues to advance, we can expect further breakthroughs in developing intelligent systems capable of tackling increasingly complex and dynamic tasks, ultimately driving innovation and progress in various domains of human endeavor.

REFERENCES

1. Yang, Z., Zhang, M., Xie, L., & Hu, X. (2020). Exploring PAC-MAN game through deep reinforcement learning. In Proceedings of the 2020 International Conference on Control, Automation and Diagnosis (ICCAD 2020) (pp. 32-37).

2. Justesen, N., Bontrager, P., & Risi, S. (2021). Evolving diverse and transferable Pac-Man playing agents. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21) (pp. 1610-1618). ACM.

3. Bontrager, P., Justesen, N., & Risi, S. (2022). Evaluating different reward functions and architectures for learning to play Pac-Man. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion) (pp. 200-201). ACM.

4. Vintan, L., & Suciu, R. (2022). Reinforcement Learning for Pac-Man. arXiv preprint arXiv:2202.12112.

5. Gortari, C. R., & Sucar, L. E. (2022). Agent-centric experience replay in Deep Q-Networks for playing Atari 2600 games. arXiv preprint arXiv:2204.09206.