Applying Machine Learning in Game AI Design

Yanzhu Du, Shisheng Cui, and Stephen Guo

We developed a reinforcement learning MDP agent and a genetic programming learning agent to play the game Super Mario Bros. Our results compare favorably with the current state of the art agents for this game. In particular, our genetic programming agent would be the top performing learning AI agent in the 2009 AI Mario competition.

Introduction

Electronic games are a pervasive part of the lives of many people around the world. In recent years, an increasing number of scientists and institutions have been devoting their time to the study of artificial intelligence in computer games.

One of the major factors for the fun and re-playability of computer games is a good artificial intelligence agent. Developing competent AI agents by hand is a difficult and time-consuming task. Although AI has been applied successfully in some games, most games do not contain true AI, as they utilize pre-defined scripts to simulate “artificial intelligence.”

We tackle the problem of developing an AI agent to play games. In particular, we consider the 2D side-scrolling platform game Super Mario Bros. The particular implementation that we use is an open source Java implementation from an annual international AI Mario competition (1).

Description

The agent’s objective is defined to be the progress through as many levels in the game as possible without dying. This is measured by a Mario game score. Lesser objectives include collecting coins and defeating enemies.

In our specific Mario platform, 24 times per second, the agent is given a 22*22 observation window of the discrete cells around the agent. Note that in the game physics engine, object coordinates have much finer resolution. Therefore, for object coordinates given to the agent, discretization is necessary.

In each of these 484 squares, the agent may observe one of the following:

- Bricks – Stationary obstacle. May release coins or power-ups if hit by large Mario.
- Borders – Stationary obstacle. Mario cannot pass through borders.
- Hills – Stationary obstacle. Mario can jump up through a hill, but cannot pass down.
- Flower pots (pipes) – Stationary obstacle. Enemy flowers appear periodically from pipes.
- Cannons (towers) – Stationary obstacle. Cannons periodically shoot out flying bullets.
- Enemies – Goombas, koopas, and their winged and spiky varieties.

In each discrete square, there are 6 terrain cell types and 4 enemy types. The state space of this problem is 10^{484}. This space is of a size such that complete search is computationally intractable.

Approach

Currently, the most successful agents for this Mario game are agents which reproduce the game physics and perform A* search through the 22*22 window. These approaches are domain-specific, utilize external information, and are not extendable to other games.

In choosing our approach, we considered multiple online and reinforcement learning techniques such as online SVMs, online decision trees, MDPs, POMDPs and genetic programming. We ultimately chose to implement a reinforcement learning MDP agent and a genetic programming agent which seemed to be the learning methods most directly applicable to our problem. MDPs provide a mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision maker. Genetic programming is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programs that perform a user-defined task.

Method 1: MDP

Features

To reduce the state space to a size computationally efficient to train, while also not giving up too much information, we decompose Mario's observation window into the following features:

- Mario Status: Can Mario jump? Is Mario on the ground? Can Mario throw a fireball?
- Two cells directly ahead and behind of Mario
- Are there gaps in front of Mario?
- Distance of enemies in front of Mario

We also simplify cell types into the following four:

1) Passable cell 2) Obstacles (bricks, borders, flower pots, cannons) 3) Half-borders 4) Enemies

Our total feature space size is $3*2^{19}$. Keep in mind that our features are chosen to be expressive as possible, while keeping the feature space size to be within the training abilities of our limited computational resources.

Actions

It is difficult to model the objective of moving forward efficiently using an MDP. Therefore, we restrict Mario's actions to always move to the right. Mario's action space contains these four actions: 1) Right, 2) Right Fast, 3) Right Jump, 4) Right Fast Jump.

Reward Function

In our MDP setup, we use a reward function of the form $R(s, a)$. A small penalty is given for jumping to discourage random jumping. A small reward is given for being away from enemies and gaps. A large penalty is given for touching the enemy.

Transition Probability Function

Though this game is mostly deterministic, due to limited visibility (22x22 window) and our feature mapping, the feature space state transition is not deterministic. Therefore, we must learn empirical transition probabilities to estimate $P_{sa}(s')$.

Value Function

In our training routine, we utilize a form of asynchronous value iteration to converge toward the optimal value function of the Bellman equation. The update equation used during training is the following:

$$V(s) \coloneqq \max_a (R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s'))$$

Note that the states we consider are now feature states/vectors in our feature space, not the original game states. To train our value function, we run the MDP agent on many randomly generated levels and collect statistics about $P_{sa}(s')$, while updating $V(s)$ through the Bellman equation.

Action Selection

$$\pi(s) = \operatorname*{argmax}_a (R(s,a) + \gamma \sum_{s'} P_{sa}(s')V(s'))$$

After training, our MDP agent uses the computed value function V to define a policy π. During game play, when the agent is presented with a game state, the state is mapped into a feature vector, which is then given to the policy π to determine the next action to take.

Method 2: Genetic Programming

Summary

Objective	Find program to control Mario's jump action
Program	Tree-based, Strong-typed
Value Type	Boolean, Integer, Float
Variable	Observation variable from Mario game (Mario status has Boolean type, Cell type has Integer type, Enemy position has Float type)
Function Set	+, −, ×, ÷ (protected division), ==, > (on integer and float), Type Casting between Integer and Float, Boolean AND, OR, NOT, If statement
Terminal Set	Variable, and Random constants
Fitness	Multi-objective: Mario game score, number of actions
Selection	Sort fitness according to Pareto dominance
Initial Pop	Random initialization, depth < 5, 50% of terminals are constants

Crossover	Typed one point crossover
Parameters	Population size 1000, 50% sub-tree crossover, 25% reproduction, 25% mutation. no tree size limits
Termination	16 Generations

Program

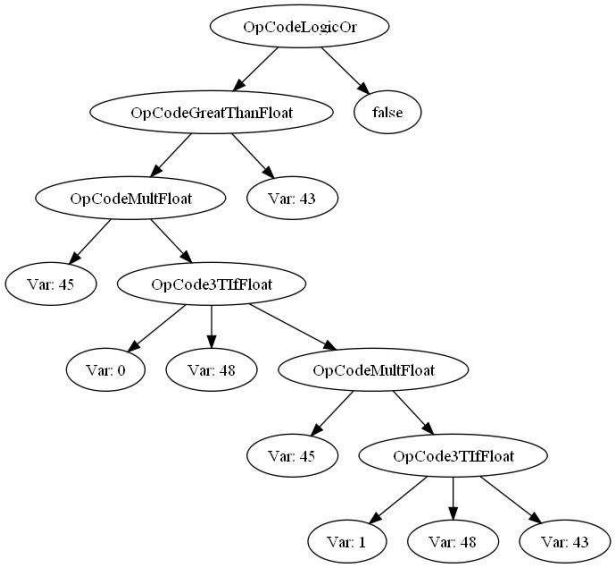
We choose to implement a strong-typed tree-based GP. Strong type may be beneficial as each type has corresponding semantics. By allowing only type-correct programs, we can limit the program space to programs that at least make semantic sense. All crossover and mutation operations preserve the type-correctness of the tree. To simplify design, we also restrict the program to control Mario's jump action only, while by default Mario will always be moving to the right.

Fitness and Selection

The fitness measure is multi-objective. It aims to maximize Mario game score while minimizing the number of actions taken. Individuals' performances are sorted according to Pareto dominance.

Genetic Programming Results

We ran 30 batches of experiments. The best individual from the genetic programming is shown below:



Overall Results

After significant amount of training, we evaluated our agents using the same environment specified in the Mario AI Competition 2009 @ CIG and compared our agents against the published results (1):

Rank	Name	Algorithm	Score	Levels	Time left	Total Kills	Mode

1	Robin Baumgarten	A*	46564.8	40	4878	373	76
...							
6	Spencer Schumann	RB,H	17010.5	8	6593	99	24
	Our GP Agent	**GP**	**13913.42**	**8**	**5458**	**58**	**32**
7	Matthew Erickson	Ev, GP	12676.3	7	6017	80	37
...							
10	Mario Pérez	SM, Lrs	12060.2	4	4497	170	23
	Our MDP Agent	**MDP**	**11104.23**	**5**	**6339**	**97**	**36**
11	Alexandru Paler	NN, A*	7358.9	3	4401	69	43

Our GP agent would be the top performing machine learning agent in the competition, as the top 6 entries are all either A* search or rule-based algorithms which reproduce game physics. Our code is available for demo upon request.

Discussion

Comparing our MDP agent to better performing agents, our agent differs in the fact it sometimes tries to kill the enemy with a fireball, rather than evade the enemy. If the fireball attack fails, Mario often will not have enough time to avoid the enemy. This suboptimal behavior comes from the fact that our MDP agent only has a coarse observation about enemies, so if it detects an enemy in the distance, but is not sure about relative position, it will continue moving forward and throw a fireball to try to remove the enemy. Ideal behavior would be to move toward the enemy and jump over them when near.

Our GP agent only controls whether Mario jumps or not. Its general behavior is to evade enemies at all times. Though it appears that the GP agent works well, the evolved logic is difficult to decipher. This is a common problem with solutions obtained by genetic programming.

The AI Mario Competition is planned to be at several AI and electronic game conferences in 2010. There may be a new “online learning” track of the competition, in which A* and rule–based agents will be at a disadvantage compared to learning agents. We are planning to submit our top agent into this competition.

References

1. **Togelius, Sergey Karakovskiy and Julian.** Mario AI Competition. [Online] In association with the IEEE Consumer Electronics Society Games Innovation Conference 2009 and with the IEEE Symposium on Computational Intelligence and Games, 9 12, 2009. [Cited: 12 9, 2009.] http://julian.togelius.com/mariocompetition2009/.

2. **Riccardo Poli, William B. Langdon,Nicholas F. McPhee.** *A Field Guide to Genetic Programming.* s.l. : http://lulu.com, March 2008. 978-1-4092-0073-4.

3. IEEE Consumer Electronics Society's Games Innovation Conference Call for Participation in the Mario Competition. [Online] 8 14, 2009. [Cited: 12 9, 2009.] http://ice-gic.ieee-cesoc.org/competitions.html.