Machine Learning for Playtesting

➢ Batu Aytemiz
➢ Oleksandra “Saya” Keehl
➢ David Sungho Park
➢ Sara Sholes
Outline

1. Problem Statement
2. Tech Stack
3. Implementation Decisions
4. Showcase
Create a Game Playing Agent using Different Methods

1. Reinforcement Learning using Unity Machine Learning Agents

2. Our implementation using MCTS and Neural Networks
**Why is this important?**

1. Can get really good at playing the games!
2. Streamline the pipeline for creating AIs.
3. Applications in Robotics and Self Driving Cars.
4. Playtesting.
The Design Cycle

http://www.genomecompiler.com/new-design-cycle/
Machine Playtesting as a Design-Assistance Tool

Score: 435

Latest Monster Stats:
Size: 8
Base Score: 80
Size Bonus: 105
Color Bonus: 40
Total Score: 225

Blue: Waiting until the very last moment - lazy.
Red: Collect monsters as soon as they are alive - greedy.
Monte Carlo Tree Search
Monte Carlo Tree Search

Run continuously in the allotted time

- Selection
- Expansion
- Simulation
- Back-propagation

Our NN
Unity 3D

https://unity3d.com/
2D Roguelike

https://unity3d.com/learn/tutorials/s/2d-roguelike-tutorial
MCTS + NN

For each game:

1. Initialize NN weights to random
2. Run a bunch of Monte Carlo simulations (using NN weights to influence choice of action in simulation stage), collecting info about the action probabilities and reward for each state it encounters.
3. Train NN via gradient descent, where the loss measures the difference between MTCS predicted action and reward probabilities
4. Repeat steps 2 and 3 to refine the NN model
Our MCTS + NN Model

Input data: feature X is an input state

Labels: Label is the action and reward probabilities are the from the previous iteration of MCTS simulations

NN could be fully connected network, on if game state is fed in as the board layout, could use convolutional network
Reinforcement Learning

Environment

Agent

State & Reward

State Transition

Action
Reinforcement Learning

Environment

Agent

State & Reward

State Transition

Action
Actions
State

1) Own Position
2) Food Position
3) Enemy Position
4) Positions of breakable Walls
5) Level
6) Food Amount
7) Distance to exit
Reward Function
Reward Function
Reward Function ++

1) Every time you get food ++
2) You moved closer to the exit ++
3) You take damage from a zombie --
4) You take a No Action move --
5) When you die, punishment based on how far you are --
Reward Function - Learning About Food

Brain name: RougeLikeBrain
Mode: External
Frame Count: 127
Reward: 109.00
Maximum Reward: 124.00
State: 1.00 7.00 2.00 1.00 6.00 1.00 6.00 0.00 0.00 0.00 0.00 0.00 2.00 5.00 2.00 2.00 6.00 1.00 2.00 4.00 0.00 0.00 0.00 0.00
Action: 2
A Lot of Tuning
Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms other online policy gradient methods, and overall strikes a favorable balance between sample complexity, simplicity, and wall-time.

1 Introduction

In recent years, several different approaches have been proposed for reinforcement learning with neural network function approximators. The leading contenders are deep Q-learning [Mni+15],
Further Work - Update State Representation

Make the state definition identical in both representations.

Roll in previous states.

Change state representation to take Pixel Data.
Further Work - Implement other RL algorithms

Policy Approximation

Value Approximation

Q-Learning
Further Work - Connect the Pipeline

Integrate trained Neural Networks with the MCTS framework.

Experiment with different Neural Networks.
Further Work - Analyze Results

Run Comparisons between different implementations
Live Demo
Monster Carlo architecture

Design experiment

Persistent storage

Visualizations

Experiment parameters

Result tree and log data

Launch several instances

C# support

Game

MCTS

Python support

Selected action

Score

Request micro-decision

Executable

Play trace and score

Experiment parameters

Action prefix