Game Engines and Machine Learning
Mars Geldard, Jonathon Manning, Paris Buttfield-Addison & Tim Nugent

Practical Artificial Intelligence with Swift
From Fundamental Theory to Development of AI-Driven Apps

Paris Buttfield-Addison, Jon Manning & Tim Nugent

Unity Game Development Cookbook
Essentials for Every Game

Jon Manning & Paris Buttfield-Addison

Mobile Game Development with Unity
BUILD ONCE, DEPLOY ANYWHERE
Why a game engine?
A game engine is a controlled, self-contained spatial, physical environment that can (closely) replicate (enough of) the real world (to be useful).
How Google's AlphaGo Beat a World Champion

Inside a man-versus-machine showdown

Christopher Nores | MAR 28, 2016

The South Korean professional Go player Lee Sedol reviews the match after finishing a single game of Google's artificial-intelligence program, AlphaGo.

On March 19, 2016, the strongest Go player in the world, Lee Sedol, was up against Google DeepMind's artificial-intelligence program, AlphaGo. They're at the Four Seasons Hotel in Seoul's Gangnam district, and it's a big deal: Most major South Korean television networks were carrying the game. In China, 60 million people are tuning in. For the English-speaking world, the American Go Association and DeepMind have been hyped up the match as if it were a David vs. Goliath contest.

TL;DR

DeepMind's AI is teaching itself parkour, and the results are adorable

Like watching a baby learn to limbo

By James Vincent | Jul 10, 2017, 8:13am EDT

Keeping up with the latest AI research can be an odd experience. On the one hand, you're aware that you're looking at cutting-edge experimentation, with new algorithms and methods that will probably (eventually) snowball into a technological revolution of all time. On the other hand, sometimes what you see is just unavoidably weird and funny.

Case in point: is a new paper from Google's AI subsidiary DeepMind titled "DeepMind's AI is teaching itself parkour, and the results are adorable." The paper describes a method for an AI to learn to move through a given environment, by learning to move in the opposite direction. The authors used a game called "Doom" for this purpose, and the results are indeed quite adorable, as the AI performs parkour-like movements through the game's environment.

Computer Out-Plays Humans in "Doom"

Deep Learning Key To Mastering Videogame's 3-D World

By Byron Spice / 412-268-9006 / bspice@cs.cmu.edu

An artificial intelligence agent developed by two Carnegie Mellon University computer science students has proven to be the ultimate survivor in the classic video game Doom — outplaying both the game's built-in AI agents and human players.
Cognitive

Physical

Visual
Unity: A General Platform for Intelligent Agents

Abstract

This work discusses the design, implementation, and evaluation of a general platform for intelligent agents. The platform is designed to support the development of intelligent agents that can interact with their environment and make decisions based on their observations. The platform provides a set of tools and APIs that allow developers to create and manage intelligent agents, as well as to test and evaluate their performance. The platform is designed to be extensible, allowing developers to add new features and capabilities as needed. The platform has been used to develop intelligent agents for various applications, including robotics, game AI, and autonomous vehicles. The results of the platform's evaluation show that it is effective in supporting the development of intelligent agents and that it has the potential to be widely adopted in the field of artificial intelligence.

1 Introduction

1.1 Background

Intelligent agents have been an important area of research for several decades. The concept of intelligent agents was first introduced in the mid-1980s, and since then, there has been significant progress in the field. A number of frameworks and platforms have been developed to support the creation and evaluation of intelligent agents. Among these, the most notable are the Agent-Oriented Architecture (AOA) and the JADE platform.

AOA is a framework that provides a high-level abstraction for the design and development of intelligent agents. JADE, on the other hand, is a Java-based platform that supports the creation of intelligent agents and provides tools for testing and evaluation.

The work presented in this paper is a general platform that builds on the strengths of existing frameworks and platforms. It is designed to be extensible and adaptable to a wide range of applications. The platform has been evaluated and tested in various scenarios, and the results show that it is effective in supporting the development of intelligent agents.
Learning from multiagent emergent behaviors in a simulated environment

Danny Lange (Unity Technologies)
1:00pm-1:40pm Wednesday, April 17, 2019
Machine Learning, Models and Methods
Location: Regent Parlor
Secondary topics: Data and Data Networks, Models and Methods, Reinforcement Learning
On the Road to Artificial General Intelligence

Danny Lange
VP of AI and Machine Learning, Unity
Basics of Unity
ML-Agents Fundamentals
The Process
Live Demo
So What?
Basics of Unity
ML-Agents Fundamentals
“The ML-Agents toolkit is mutually beneficial for both game developers and AI researchers as it provides a central platform where advances in AI can be evaluated on Unity’s rich environments and then made accessible to the wider research and game developer communities.”

–Unity ML-Agents Toolkit Overview

https://github.com/Unity-Technologies/ml-agents/
Academy

- Orchestrates the observations and decision making process
- Sets environment-wide parameters, like speed and rendering quality
- Talks to the external communicator
- Make sure agent(s) and brain(s) are in sync
- Coordinates everything
Brain

- Holds logic for the Agent’s decision making
- Determines which action(s) the Agent should take at each instance
- Receives observations from the Agent
- Receives rewards from the Agent
- Returns actions to the Agent
- Can be controlled by a human, a training process, or an inference process
Agent

- Attached to a Unity Game Object
- Generates observations
- Performs actions (that it’s told to do by a brain)
- Assigns rewards
- Linked to one Brain
None of these concepts are new

Some might have new names
Training Methods
Reinforcement Learning

Imitation Learning

Neuroevolution

... and many other learning methods
### Reinforcement Learning

- Signals from rewards
- Trial and error
- Simulate at high speeds
- Agent becomes optimal

### Imitation Learning

- Learning through demonstrations
- No rewards
- Simulate in real-time (mostly)
- Agent becomes human-like
Actions  Observations  Rewards
**Reinforcement Learning**

- Signals from rewards
- Trial and error
- Simulate at high speeds
- Agent becomes optimal

**Imitation Learning**

- Learning through demonstrations
- No rewards
- Simulate in real-time (mostly)
- Agent becomes human-like
Abstract
Recent advances in Deep Reinforcement Learning and Robotics have been driven by the presence of increasingly realistic and complex simulation environments. Many of the existing platforms, however, provide either unrealistic visuals, inaccurate physics, low task complexity, or a limited capacity for interaction among artificial agents. Furthermore, many platforms lack the ability to flexibly configure the simulation, hence turning the simulation environment into a black-box from the perspective of the learning system. Here we describe a new open source toolkit for creating and interacting with simulation environments using the Unity platform: Unity ML-Agents Toolkit. By taking advantage of Unity as a simulation platform, the toolkit enables the development of learning environments which are rich in sensory and physical complexity, provide compelling cognitive challenges, and support dynamic multi-agent interaction. We detail the platform design, communication protocol, set of example environments, and variety of training scenarios made possible via the toolkit.

1 Introduction
1.1 Background
In recent years, there have been significant advances in the state of Deep Reinforcement Learning research and algorithm design (Mnih et al., 2015; Schulman et al., 2017; Silver et al., 2018; Espeholt et al., 2018). Essential to this rapid development has been the presence of challenging, easy to use, and scalable simulation platforms, such as the Arcade Learning Environment (Bellemare et al., 2013), VizDoom (Kempka et al., 2016), Mujoco (Todorov et al., 2012), and others (Beattie et al., 2016; Johnson et al., 2016). The existence of the Arcade Learning Environment (ALE), for example, which contained a set of fixed environments, was essential for providing a means of benchmarking the control-from-pixels approach of the Deep Q-Network (Mnih et al., 2013). Similarly, other platforms have helped motivate research into more efficient and powerful algorithms (Oh et al., 2016; Andrychowicz et al., 2017). These simulation platforms serve not only to enable algorithmic improvements, but also as a starting point for training models which may subsequently be deployed in the real world. A prime example of this is the work being done to train autonomous robots within...
Unity: A General Platform for Intelligent Agents

Arthur Juliani
Unity Technologies
arthur@unity3d.com

Vincent-Pierre Berges
Unity Technologies
vincentpierre@unity3d.com

Esh Vckay
Unity Technologies
esh@unity3d.com

Yuan Gao
Unity Technologies
vincentg@unity3d.com

Hunter Henry
Unity Technologies
brandish@unity3d.com

Marwan Mattar
Unity Technologies
marwan@unity3d.com

Danny Lange
Unity Technologies
dlange@unity3d.com

Abstract
Recent advances in Deep Reinforcement Learning and Robotics have been driven by the presence of increasingly realistic and complex simulation environments. Many of the existing platforms, however, provide either unrealistic visuals, inaccurate physics, low task complexity, or a limited capacity for interaction among artificial agents. Furthermore, many platforms lack the ability to flexibly configure the simulation, hence turning the simulation environment into a black-box from the perspective of the learning system. Here we describe a new open source toolkit for creating and interacting with simulation environments using the Unity platform: Unity ML-Agents Toolkit. By taking advantage of Unity as a simulation platform, the toolkit enables the development of learning environments which are rich in sensory and physical complexity, provide compelling cognitive challenges, and support dynamic multi-agent interaction. We detail the platform design, communication protocol, set of example environments, and variety of training scenarios made possible via the toolkit.

1 Introduction
1.1 Background
In recent years, there have been significant advances in the state of Deep Reinforcement Learning research and algorithm design (Mnih et al., 2015; Schulman et al., 2017; Silver et al., 2018; Espeholt et al., 2018). Essential to this rapid development has been the presence of challenging, easy to use, and scalable simulation platforms, such as the Arcade Learning Environment (Bellemare et al., 2013), VizDoom (Kempka et al., 2016), Mujoco (Todorov et al., 2012), and others (Beattie et al., 2016; Johnson et al., 2016). The existence of the Arcade Learning Environment (ALE), for example, which contained a set of fixed environments, was essential for providing a means of benchmarking the control-from-pixels approach of the Deep Q-Network (Mnih et al., 2013). Similarly, other platforms have helped motivate research into more efficient and powerful algorithms (Oh et al., 2016; Andrychowicz et al., 2017). These simulation platforms serve not only to enable algorithmic improvements, but also as a starting point for training models which may subsequently be deployed in the real world. A prime example of this is the work being done to train autonomous robots within...
The Process

Imitation Learning
Step by Step

• Pick a task
• Create an environment
• Create/identify the agent
• Create an academy
• Pick a learning/training method
• Create observations, rewards, and actions
• Pick algorithms, tune, and train
## Step by Step

<table>
<thead>
<tr>
<th>A car that drives by itself</th>
<th>• Pick a task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cartoony race track</td>
<td>• Create an environment</td>
</tr>
<tr>
<td>Our self-driving car</td>
<td>• Create/identify the agent</td>
</tr>
<tr>
<td>A bog-standard Academy</td>
<td>• Create an academy</td>
</tr>
<tr>
<td>Imitation Learning</td>
<td>• Pick a learning/training method</td>
</tr>
<tr>
<td>Raycasts, Modify transform</td>
<td>• Create observations, rewards, and actions</td>
</tr>
<tr>
<td>Train!</td>
<td>• Pick algorithms, tune, and train</td>
</tr>
</tbody>
</table>
The Environment
The Environment
Imitation learning
Imitation Learning

- Learning through demonstrations
- No rewards
- Simulate in real-time (mostly)
- Agent becomes human-like
So What?
Reinforcement Learning

- Signals from rewards
- Trial and error
- Simulate at high speeds
- Agent becomes optimal

Imitation Learning

- Learning through demonstrations
- No rewards
- Simulate in real-time (mostly)
- Agent becomes human-like
Rewards in Actions

Actions

Rewards
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], "vanilla" policy gradient methods [Mni+16], and trust region / natural policy gradient methods [Sch+15b]. However, there is room for improvement in developing a method that is scalable (to large models and parallel implementations), data efficient, and robust (i.e., successful on a variety of problems without hyperparameter tuning). Q-learning (with function approximation) fails on many simple problems1 and is poorly understood, vanilla policy gradient methods have poor data efficiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks).

This paper seeks to improve the current state of affairs by introducing an algorithm that attains the data efficiency and reliable performance of TRPO, while using only first-order optimization. We propose a novel objective with clipped probability ratios, which forms a pessimistic estimate (i.e., lower bound) of the performance of the policy. To optimize policies, we alternate between sampling data from the policy and performing several epochs of optimization on the sampled data.

Our experiments compare the performance of various different versions of the surrogate objective, and find that the version with the clipped probability ratios performs best. We also compare PPO to several previous algorithms from the literature. On continuous control tasks, it performs better than the algorithms we compare against. On Atari, it performs significantly better (in terms of sample complexity) than A2C and similarly to ACER though it is much simpler.

1While DQN works well on game environments like the Arcade Learning Environment [Bel+15] with discrete action spaces, it has not been demonstrated to perform well on continuous control benchmarks such as those in OpenAI Gym [Bro+16] and described by Duan et al. [Dua+16].
TensorFlow
“That seems more useful.”

–You, probably.
Reinforcement Learning

- Signals from rewards
- Trial and error
- Simulate at high speeds
- Agent becomes optimal

Imitation Learning

- Learning through demonstrations
- No rewards
- Simulate in real-time (mostly)
- Agent becomes human-like
Demos
Actions

X-rotation
Z-rotation
Observations

AddVectorObs(gameObject.transform.rotation.z);
AddVectorObs(gameObject.transform.rotation.x);
AddVectorObs(ball.transform.position - gameObject.transform.position);
AddVectorObs(ballRb.velocity);
Rewards

if ((ball.transform.position.y - gameObject.transform.position.y) < -2f ||
    Mathf.Abs(ball.transform.position.x - gameObject.transform.position.x) > 3f ||
    Mathf.Abs(ball.transform.position.z - gameObject.transform.position.z) > 3f)
{
    Done();
    SetReward(-1f);
}
else
{
    SetReward(0.1f);
}
Useful...?

• Training behaviours, rather than coding behaviours

• Exploring or training behaviours in physical, spatial, simulated scenarios
  • Self-driving cars
  • Warehouses, factories

• Low-risk, low-cost way to test visual, physical, cognitive machine learning problems

• “Free” visualisation!
A game engine is a controlled, self-contained spatial, physical environment that can (closely) replicate (enough of) the real world (to be useful).

(but it’s also useful for non-physical problems that you might be able to make a physical representation of and observe.)
Thank you

@themartianlife  @the_mcjones  @parisba
Thank you

@themartianlife @the_mcjones @parisba @aiwithswift

At #OSCON? Join us for a half-day tutorial on Unity Machine Learning!

https://lab.to/AlConfNYC2019