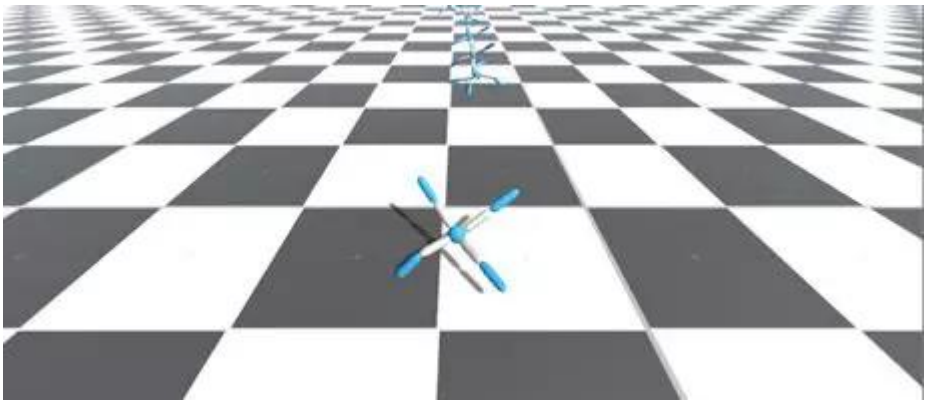
人工智能丨使用 Python* 的英特尔® 分发版实现 Unity* 机器学习入门

原创：Manisha B.英特尔开发人员专区8 月 31 日



本文将向游戏开发人员介绍如何使用强化学习创建更好的人工智能（AI）行为。使用 Python* 的英特尔® 分发版 — 常用面向对象的高级编程语言的进阶版 — 读者可收集关于如何训练预先存在的机器语言（ML）代理学习和适应的信息。在此场景下，我们将使用英特尔® Optimization for TensorFlow* 在本地化环境中运行 Unity * ML-Agents。

简介

Unity ML-Agents 能够很好地支持游戏开发人员在常用 Unity 引擎中创建场景时，了解如何应用强化学习概念。我们使用 ML-Agents 插件创建了一个模拟环境。然后配置严格的训练，通过可用于 Unity 中创建的场景的 TensorFlow 生成输出文件，并改进此次模拟。

基本步骤如下所示：

1.

首先从介绍强化学习开始。

2.
3. 通过安装 TensorFlow 1.4 的 "requirements.txt" 文件和其他相关性进行设置。

4.
5. 训练预先存在的 ML-Agents。

6. **系统配置**

所使用的配置如下所示：

1. 标准华硕笔记本电脑

2.
3. 第四代英特尔® 酷睿™ i7 处理器

4.
5. 8 GB RAM

6.
7. Windows® 10 企业版

8. **什么是强化学习？**

强化学习是一种“训练”智能程序（称为代理）以在已知或未知的环境中不断适应和学习的方法。系统基于所得分数（或正（奖励）或负（惩罚））不断进步。然后我们根据代理与环境之间的交互，暗示需要采取哪些行动。

关于强化学习的一些要点：

- 它与普通的机器学习不同，因为我们不查看训练数据集。
-
- 它不适用于数据，而是适用于环境，我们可以通过它描绘真实场景。
-
- 它以环境为基础，因此许多参数都会发挥作用，因为 “RL” 需要大量信息来相应地学习和适应。
-
- 它使用潜在的大规模环境，即真实的场景；可能是 2D 或 3D 环境、模拟世界或基于游戏的场景。
-
- 它依靠学习目标来实现目标。
-
- 它从所提供的环境中获得奖励。
-

强化学习周期如下所示。

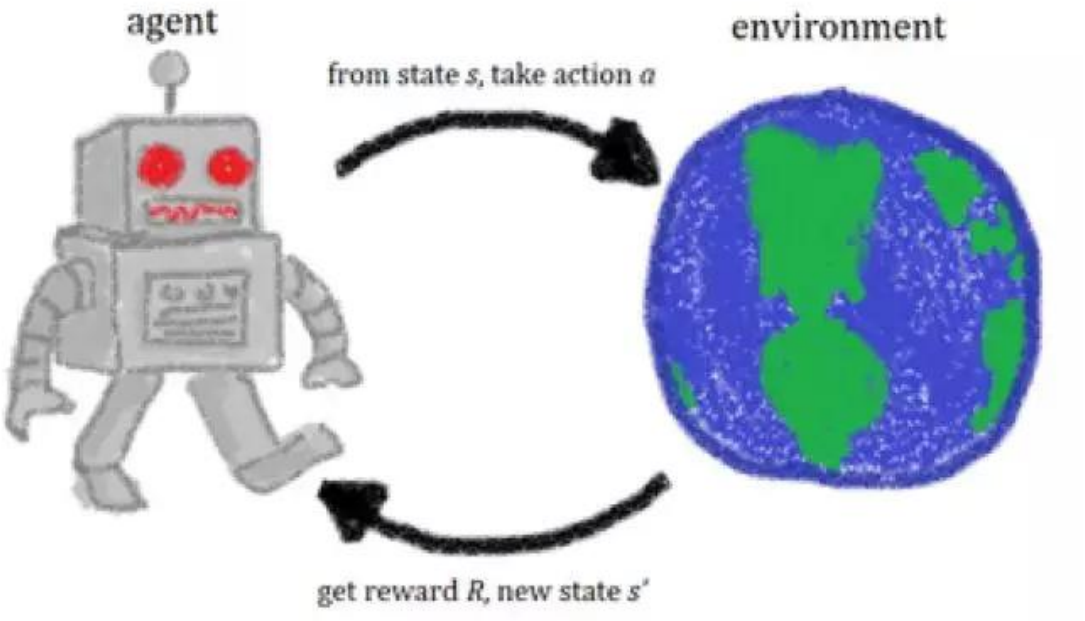


图 1.强化学习周期。

奖励制度如何运作

奖励主要通过当单个或多个代理与环境交互时状态发生转换时提供分数来实现。这些分数被称为奖励。训练越多，得到的奖励越多，系统会因此而越来越准确。环境可以具有许多不同的特性，如下所述。

代理

代理是制定明智决策的软件例程。代理应该能够感知到周围环境所发生的一切。代理能够根据制定导致动作的决策来感知环境。代理执行的动作必须是最佳操作。软件代理可以是自动的，也可以与其他代理或人合作。

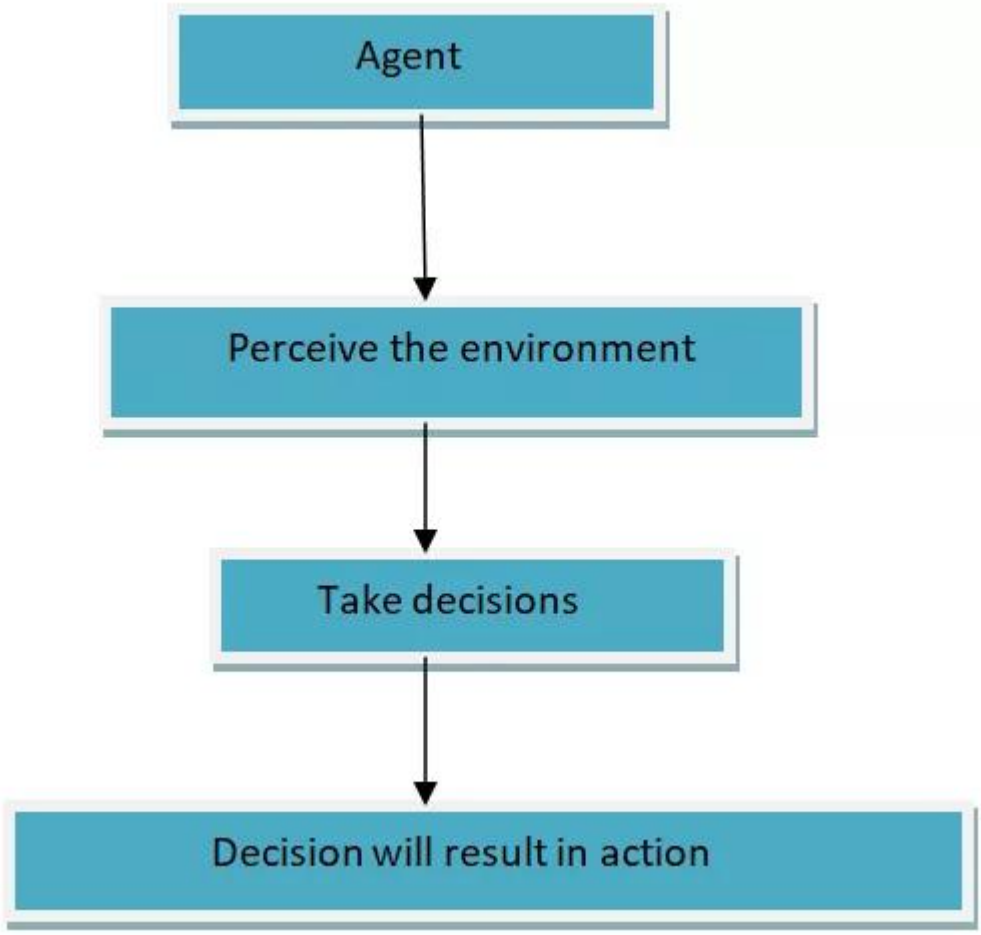


图 2.展示环境工作流程的流程图。

环境

环境决定代理与其世界交互的参数。代理必须适应环境因素，才能进行学习。环境可以是 2D 或 3D 世界，也可以是网格。

环境的部分重要特性包括：

a) 确定性

b) 可观察

c) 独立或连续

d) 单代理或多代理

下面逐一解释这些特性。

确定性

如果我们可以根据输入和动作从逻辑上推断和预测环境中将会发生的情况，那么这种情况就是确定的。由于具有确定性，因此 AI 可以预测所发生的变化，并且强化学习问题变得更简单，因为一切都是已知的。

确定型有穷自动机（DFA)

在自动机理论中，如果系统的每一次转换由原始状态和输入符号所决定，那么它将被称为 "DFA"。每次状态状态都需要读取输入符号。系统完成有穷数量的步骤，并且针对一种状态仅执行一个动作。

不确定型有穷自动机（NDFA)

如果我们在不确定机器将进入哪种状态的场景中，那么将其称之为"NDFA。” 仍然完成有穷数量的步骤，但转换不是唯一的。

可观察

如果我们说周围的环境完全可以观察得到，那么该环境适合实施强化学习。如果将其视作下象棋，那么环境是可预测的，潜在移动步数也是有限的。相反，扑克牌游戏并不是完全可观察到的，因为下一张牌是未知的。

独立或连续

我们继续说象棋/扑克场景，当下一次移动或出牌选择有限时，就属于独立状态。如果有多种可能状态，我们称之为连续状态。

单代理或多代理

强化学习中的解决方案可以使用单代理或多代理。处理非确定性问题时，我们使用多代理强化学习。了解强化学习的关键是如何使用学习技巧。在多代理解决方案中，不同环境之间的代理交互非常多。关键是了解哪种类型的信息是通常可用的。

单代理无法处理收敛问题，因此当强化学习中存在部分收敛时，由动态环境中的多代理进行处理。在多代理模式下，每个代理的目标和动作都会影响环境。

下图展示了单代理和多代理模式之间的差异。

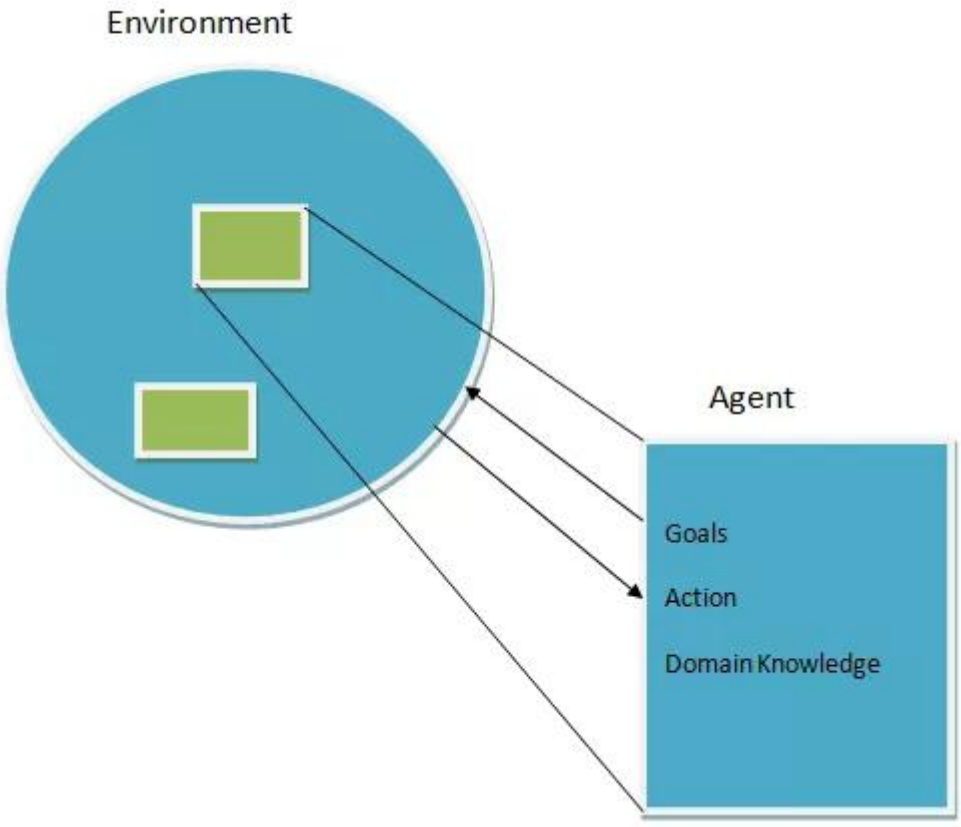


图 3.单代理系统。

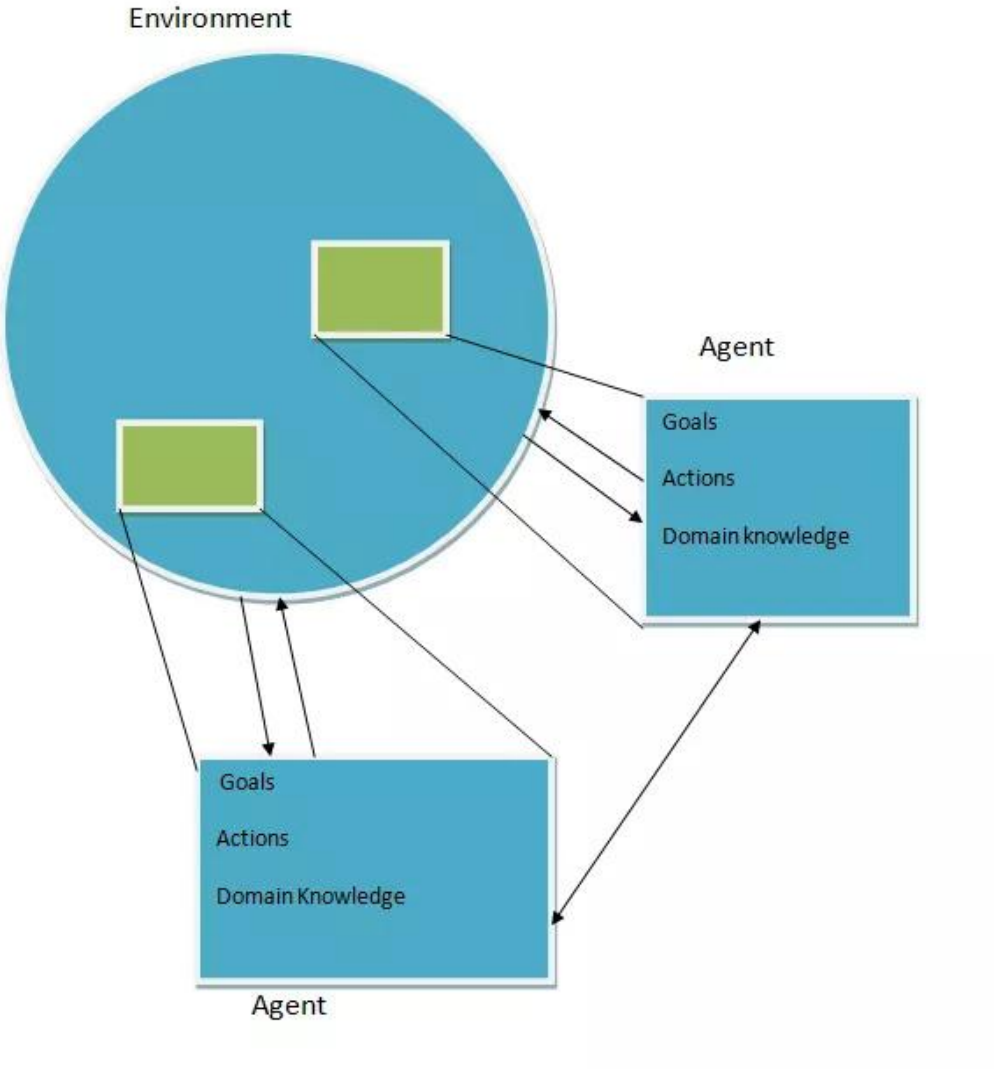


图 4.多代理系统。

入门

我们将使用 Unity 集成开发引擎（IDE）演示基于游戏的模拟环境中的强化学习。我们从头开始创建模拟，之后使用 Unity ML-Agents 展示如何在创建的项目中实施强化学习，并观察结果的准确性。

第 1 步：创建环境

首先，我们创建面向 Python 的英特尔分发版的环境。

前提条件

确保您已安装 Anaconda* IDE。Anaconda 是 Python 编程语言的免费开源分发版，用于与数据科学和机器学习相关的应用。通过 Anaconda，我们可以安装不同的 Python 库，这些库对机器学习非常有用。

下载链接为：

https://www.anaconda.com/download/。

使用英特尔 biuld 创建新环境的命令如下所示。

conda create -n idp intelpython3_core python=3

安装完所有相关性后，我们继续执行第 2 步。

第 2 步：激活环境

现在我们要激活环境。使用的命令如下所示。

source activate idp

第 3 步：检查环境

激活环境后，我们检查 Python 版本。（应该反映英特尔的那个。）

(idp) C:\Users\abhic>python

Python 3.6.3 |Intel Corporation| (default, Oct 17 2017, 23:26:12) [MSC v.1900 64 bit (AMD64)] on win32

输入**"help"，"copyright"，"credits"**或

"license"，了解更多信息。

Python 的英特尔® 分发版由英特尔公司提供。

请查看：

https://software.intel.com/zh-cn/python-distribution

第 4 步：克隆 GitHub* 库

我们需要在激活的英特尔优化环境（例如名为 idp）内部通过 GitHub * 链接克隆或拷贝 Unity ML 库。克隆库时，我们使用以下命令：

(idp) C:\Users\abhic\Desktop>git clone

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

第 5 步：安装要求

克隆期间 ,我们需要安装某些要求。Requirements.txt 位于 Python 子目录中。

(idp) C:\Users\abhic\Desktop\ml-agents\python>pip install -r requirements.txt

这样可以安装强制性相关性。

第 6 步：创建 build

build 在 Unity IDE 中创建，并生成可执行文件。爬虫可执行文件如下所示。

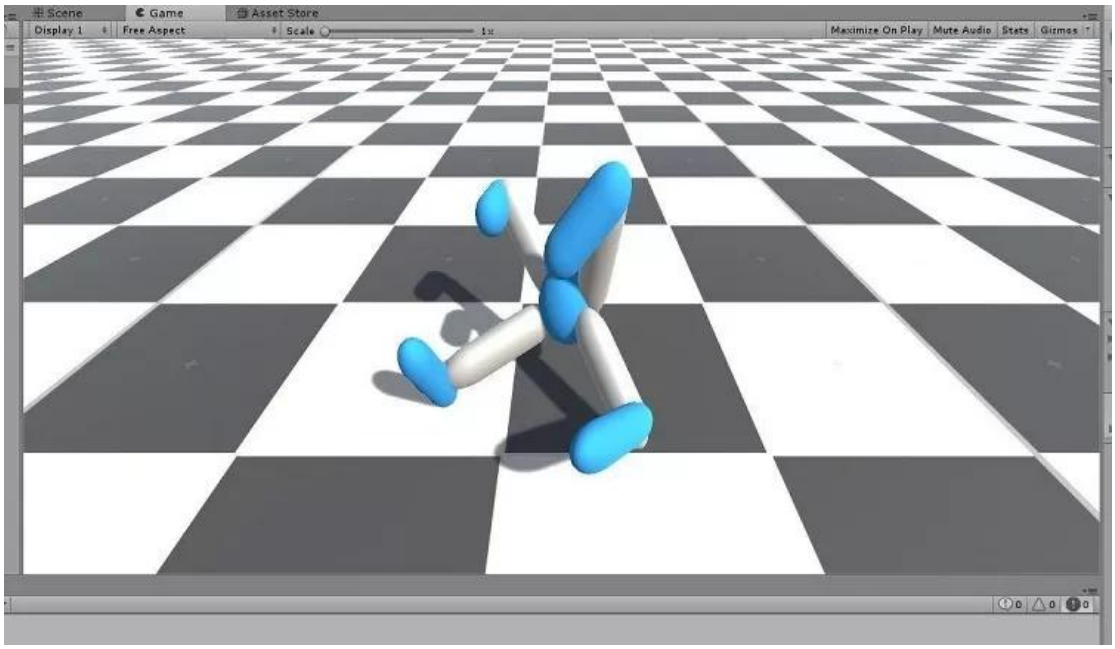


图 5.训练前的爬虫可执行文件

第 7 步：优化 build

如果要使用 Python 的英特尔分发版加快训练的速度，可通过 Python 子目录发布以下命令：

(idp) C:\Users\abhic\Desktop\ml-agents\python>python learn.py manisha.exe --run-id=manisha –train

训练完成一次完整的运行之后，我们获取在大脑内部使用的字节文件，它在 Academy 的子对象中：

INFO: unityagents:Saved Model

INFO: unityagents:Ball3DBrain:Step:50000.Mean Reward:100.000.STD of Reward:0.000.

INFO: unityagents:Saved Model

INFO: unityagents:Saved Model

INFO: unityagents:List of nodes to export:

INFO: unityagents: action

INFO: unityagents: value_estimate

INFO: unityagents: action_probs

INFO: tensorflow:Restoring parameters from ./models/manisha\model-50000.cptk

INFO: tensorflow:Restoring parameters from ./models/manisha\model-50000.cptk

INFO: tensorflow:Froze 15 variables.

INFO: tensorflow:Froze 15 variables.

Converted 15 variables to const ops.

我们生成的字节文件用于模拟与机器学习的协作。

使用 Python 的

英特尔分发版* 的优势

Python 不是针对多线程而设计的。它在一个线程上运行，虽然开发人员可以在其他线程上运行代码，但这些线程不能轻易访问任何 Python 对象。Python 的英特尔分发版* 包含支持线程的库，因此可以将英特尔® 线程构建模块（英特尔® TBB）视为潜在的多线程工具。

将 Python 的英特尔分发版用于 Unity ML-Agents 具有以下优势：

-

训练过程更快。

-

- TensorFlow 的 CPU 版本开销更少。

-

- 使用英特尔优化管道处理多代理更轻松、更快。

- **Unity* ML-Agents v 0.3**

  Unity ML-Agents 不断发展，并根据社区反馈不断更新。ML-Agents 基于模仿学习，与强化学习不同。最常见的模仿学习方法是“行为克隆”。行为克隆是一种应用于神经网络（特别是卷积神经网络或 CNN）以复制行为的方法，比如复制自动驾驶汽车环境，其中系统的目标是像人类一样驾驶汽车。

  模仿学习

  一般来说，当我们谈论“模仿学习”时，通常指的是通过示范进行学习。这种示范是基于我们在分析并向代理生成学习信号时获取的学习行为模式。下表列出了模仿学习和强化学习之间的差异。

  模仿学习对比强化学习。

  模仿学习

  学习过程通过示范进行。

  不需要这种奖惩机制。奖赏不是必需的。

  通常演变为实时交互。

  训练后，代理人在执行任务时变得“像人一样”。

  强化学习

涉及从奖励和惩罚中学习。

基于试错法。

特别适用于快速模拟方法。

训练后，代理在执行任务时变得“最佳”。

TensorFlowSharp

在本节中，我们将介绍有关 TensorFlowSharp 的一些基本知识。TensorFlow 于 2015 年首次发布，是谷歌用于数据流编程和深度学习框架的开源库。TensorFlow 不提供 C＃本机 API，而且支持本地用 C ＃ 编写的内部大脑。为了使内部大脑能够进行机器学习，我们需要使用 TensorFlowSharp，它是一个第三方库，其特定目的是将 .NET 框架绑定至 TensorFlow。

运行示例

我们现在通过 Unity ML-Agents 项目的一个示例来实施模仿学习。具体过程涉及以下步骤：

1. 包含 TensorFlowSharp Unity 插件。
2.
3. 启动 Unity IDE。
4.
5. 查找 Assets 内部的示例文件夹。ML-Agents 项目文件夹中有一个名为 “Examples” 的子文件夹。我们将使用名为 Crawler 的示例。所有更改都将发生在该文件夹中。
6.

我们创建训练环境时，必须将代理使用的大脑设为 External。这样有利于代理在尝试制定决策时与外部训练流程进行通信。

我们正在探索示例项目 Crawler。该装置是一只长有四肢的生物，四肢端部各延伸出一个较短的肢体或前臂（见上图 5）。为确保该场景成功，我们设定了以下目标：代理必须沿 x 轴移动身体，期间不掉下来。

我们必须设置一些参数来对示例进行微调。环境包含三个与单个大脑链接的代理。在 Academy 中，在 Inspector 窗口中找到子对象 "CrawlerBrain"。将大脑类型设为 External。

接下来打开 Player Settings：

1. 前往 Menu > Edit > ProjectSetting> Player。
2.
3. 前往 Options Resolution and Presentation。
4.
5. 检查 "Run in Background" 是否选中。然后选择分辨率对话框是否设为 "disabled"。最后单击 "Build"。保存在 Python 文件夹中。将其命名为 "Manisha1" 然后保存。

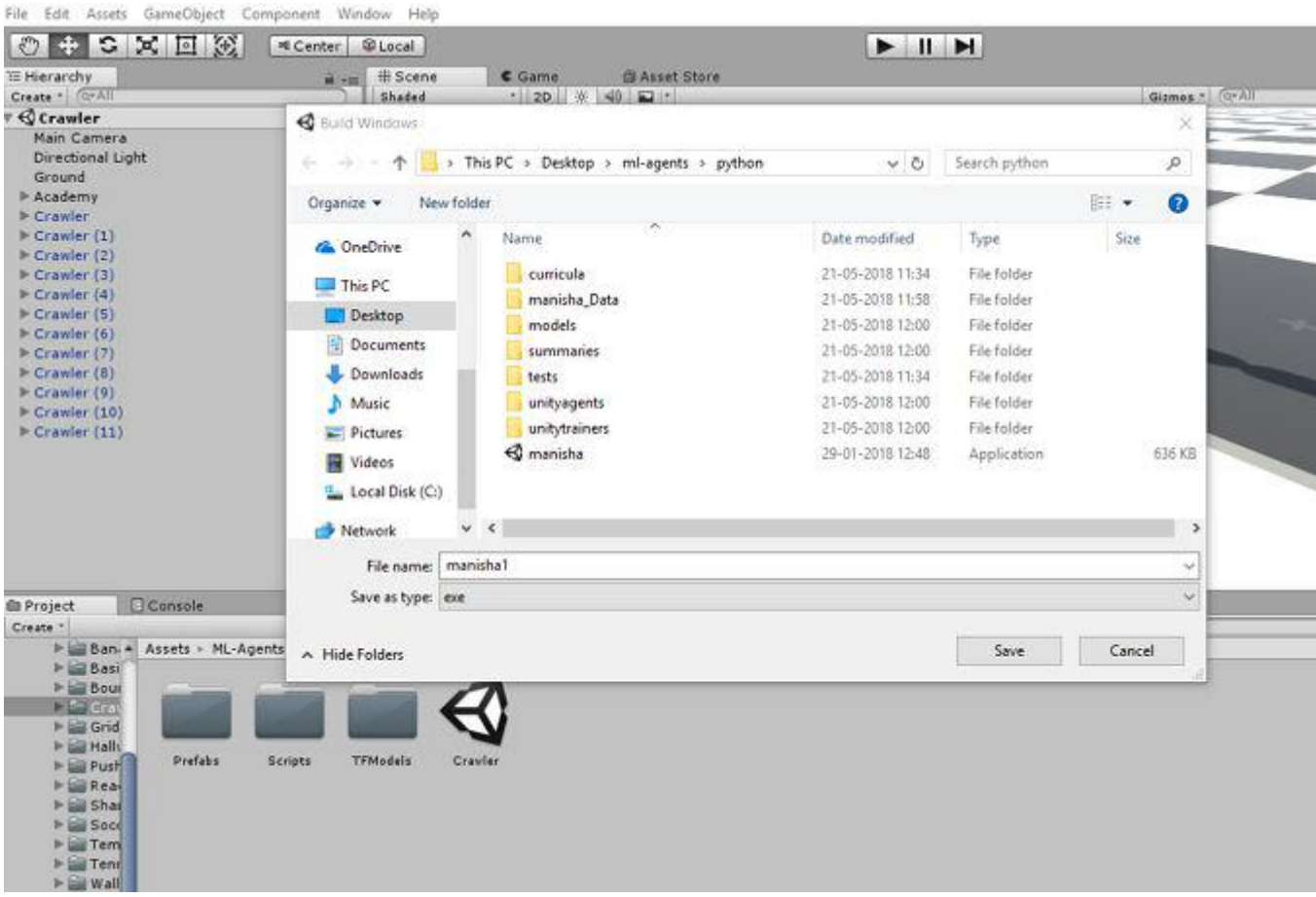


图 6.将 build 保存在 Python 文件夹中。*

训练大脑

现在我们开始训练大脑。如果要打开 Anaconda 提示窗口，可使用 Windows 中的搜索选项并输入 Anaconda。Anaconda 提示窗口将打开。进入 Anaconda 提示窗口后，我们要找出可用的环境。

(base) C:\Users\abhic>conda info --envs

conda environments:

#

base * C:\ProgramData\Anaconda3

idp C:\Users\abhic\AppData\Local\conda\conda\envs\idp

tensorflow-gpu C:\Users\abhic\AppData\Local\conda\conda\envs\tensorflow-gpu

tf-gpu C:\Users\abhic\AppData\Local\conda\conda\envs\tf-gpu

tf-gpu1 C:\Users\abhic\AppData\Local\conda\conda\envs\tf-gpu1

tf1 C:\Users\abhic\AppData\Local\conda\conda\envs\tf1

我们传递以下命令：

(base) C:\Users\abhic>activate idp

Python 的英特尔分发版和英特尔 Optimization for TensorFlow 均在环境 idp 中安装。接下来我们打开桌面上克隆的文件夹，激活 idp。

(idp) C:\Users\abhic\Desktop\ml-agents>

由于我们将 .exe 文件保存在 Python 子目录中，因此我们可以在那儿找到该文件。

(idp) C:\Users\abhic\Desktop\ml-agents>cd python

使用目录命令 **dir** 我们可以列出 Python 子文件夹中的条目：

我们显示文件夹中的内容，以便更轻松地识别 Python 子文件夹中的文件。由于主要更改或支持代码都在此子文件夹中，因此更改我们将在子文件夹中训练机器学习代理的方式更加轻松有效。python 子文件夹非

常重要，因为默认代码和其他支持库都位于此子文件夹。在为 Unity 场景创建 build 时，我们看到生成了一个自动执行文件，以及名为“manisha1.exe” 和 “manisha1_Data” 的数据文件夹。

目录 C:\Users\abhic\Desktop\ml-agents\python

28-05-2018 06:28

.28-05-2018 06:28 ..21-05-2018 11:34 6,635 Basics.ipynb 21-05-2018 11:34 curricula 21-05-2018 11:34 2,685 learn.py 29-01-2018 13:48 650,752 manisha.exe 29-01-2018 13:24 650,752 manisha1.exe 28-05-2018 06:28 manisha1_Data 21-05-2018 11:58 manisha_Data 21-05-2018 12:00 models 21-05-2018 11:34 101 requirements.txt 21-05-2018 11:34 896 setup.py 21-05-2018 12:00 summaries 21-05-2018 11:34 tests 21-05-2018 11:34 3,207 trainer_config.yaml 21-05-2018 12:00 24 unity-environment.log 21-05-2018 12:00 unityagents 29-01-2018 13:55 36,095,936 UnityPlayer.dll 21-05-2018 12:00 unitytrainers 18-01-2018 04:44 42,704 WinPixEventRuntime.dll 10 File(s) 37,453,692 bytes 10 Dir(s) 1,774,955,646,976 bytes free

查看子目录内部，找到可执行文件 “manisha1”。现在我们准备使用 Python 的英特尔分发版和英特尔 Optimization for TensorFlow 来训练模型。至于训练，我们将使用 learn.py 文件。使用英特尔优化的 Python 的命令如下所示。

(idp) C:\Users\abhic\Desktop\ml-agents\python>python learn.py manisha1.exe --run-id=manisha1 –train

(idp) C:\Users\abhic\Desktop\ml-agents\python>python learn.py manisha1.exe --run-id=manisha1 --train

INFO:unityagents:{'--curriculum':'None',

'--docker-target-name':'Empty',

'--help':False,

'--keep-checkpoints':'5',

'--lesson':'0',

'--load':False,

'--run-id':'manisha1',

'--save-freq':'50000',

'--seed':'-1',

'--slow':False,

'--train':True,

'--worker-id':'0',

'':'manisha1.exe'}

INFO:unityagents:

'Academy' started successfully!

Unity Academy name:Academy

Number of Brains:1

Number of External Brains:1

Lesson number:0

Reset Parameters:

Unity brain name:CrawlerBrain

Number of Visual Observations (per agent):0

Vector Observation space type: continuous

Vector Observation space size (per agent):117

Number of stacked Vector Observation:1

Vector Action space type: continuous

Vector Action space size (per agent):12

Vector Action descriptions:, , , , , , , , , , ,

2018-05-28 06:57:56.872734:I C:\tf_jenkins\home\workspace\rel-win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_guard.cc:137] Your CPU supports instructions that this TensorFlow binary was not compiled to use:AVX AVX2

C:\Users\abhic\AppData\Local\conda\conda\envs\idp\lib\site-packages\tensorflow\python\ops\gradients_impl.py:96:UserWarning:Converting sparse IndexedSlices to a dense Tensor of unknown shape.This may consume a large amount of memory.

"Converting sparse IndexedSlices to a dense Tensor of unknown shape."

INFO: unityagents:Hyperparameters for the PPO Trainer of brain

CrawlerBrain:

batch_size: 2024

beta: 0.005

buffer_size: 20240

epsilon: 0.2

gamma: 0.995

hidden_units: 128

lambd: 0.95

learning_rate: 0.0003

max_steps: 1e6

normalize: True

num_epoch: 3

num_layers: 2

time_horizon: 1000

sequence_length: 64

summary_freq: 3000

use_recurrent: False

graph_scope:

summary_path: ./summaries/manisha1

memory_size: 256

INFO:unityagents:CrawlerBrain:Step:3000.Mean Reward:-5.349.Std of Reward:3.430.

INFO:unityagents:CrawlerBrain:Step:6000.Mean Reward:-4.651.Std of Reward:4.235.

The parameters above set up the training process.After the training process is complete (it can be lengthy) we get the following details in the console:

INFO: unityagents:Saved Model

INFO: unityagents:CrawlerBrain:Step:951000.Mean Reward:2104.477.Std of Reward:614.015.

INFO: unityagents:CrawlerBrain:Step:954000.Mean Reward:2203.703.Std of Reward:445.340.

INFO:unityagents:CrawlerBrain:Step:957000.Mean Reward:2205.529.Std of Reward:531.324.

INFO:unityagents:CrawlerBrain:Step:960000.Mean Reward:2247.108.Std of Reward:472.395.

INFO:unityagents:CrawlerBrain:Step:963000.Mean Reward:2204.579.Std of Reward:554.639.

INFO:unityagents:CrawlerBrain:Step:966000.Mean Reward:2171.968.Std of Reward:547.745.

INFO:unityagents:CrawlerBrain:Step:969000.Mean Reward:2154.843.Std of Reward:581.117.

INFO:unityagents:CrawlerBrain:Step:972000.Mean Reward:2268.717.Std of Reward:484.157.

INFO:unityagents:CrawlerBrain:Step:975000.Mean Reward:2244.491.Std of Reward:434.925.

INFO:unityagents:CrawlerBrain:Step:978000.Mean Reward:2182.568.Std of Reward:564.878.

INFO:unityagents:CrawlerBrain:Step:981000.Mean Reward:2315.219.Std of Reward:478.237.

INFO:unityagents:CrawlerBrain:Step:984000.Mean Reward:2156.906.Std of Reward:651.962.

INFO:unityagents:CrawlerBrain:Step:987000.Mean Reward:2253.490.Std of Reward:573.727.

INFO:unityagents:CrawlerBrain:Step:990000.Mean Reward:2241.219.Std of Reward:728.114.

INFO:unityagents:CrawlerBrain:Step:993000.Mean Reward:2264.340.Std of Reward:473.863.

INFO:unityagents:CrawlerBrain:Step:996000.Mean Reward:2279.487.Std of Reward:475.624.

INFO:unityagents:CrawlerBrain:Step:999000.Mean Reward:2338.135.Std of Reward:443.513.

INFO:unityagents:Saved Model

INFO:unityagents:Saved Model

INFO:unityagents:Saved Model

INFO:unityagents:List of nodes to export :

INFO:unityagents: action

INFO:unityagents: value_estimate

INFO:unityagents: action_probs

INFO:tensorflow:Restoring parameters from ./models/manisha1\model-1000000.cptk

INFO:tensorflow:Restoring parameters from ./models/manisha1\model-1000000.cptk

INFO:tensorflow:Froze 15 variables.

INFO:tensorflow:Froze 15 variables.

Converted 15 variables to const ops.

集成训练大脑与 Unity 环境

之所以使用 Python 的英特尔分发版，是因为我们想提高训练过程的准确性。由于步骤繁多，部分示例需要更多时间来完成训练过程。

由于 TensorFlowSharp 仍处于试验阶段，因此默认状态下禁用。如果要启用 TensorFlow 并使内部大脑可用，请按照下列步骤操作：

1. 确保 TensorFlow 插件在 Assets 文件夹中。在 “Project” 选项卡中，导航到此路径：Assets->ML-Agents->Plugins->Computer。
2.
3. 打开 Edit->projectSettings->Player，启用 TensorFlow 和 .NET 支持。选择 Scripting Runtime Version to Experimental(.net 4.6)。
4.

5. 打开脚本定义的符号并添加以下文本：ENABLE_TENSORFLOW。
6.
7. 按下 Enter 键并保存项目。
8. **将训练过的模型整合到 Unity 中**

   训练过程结束后，TensorFlow 框架为该项目创建一个字节文件。在 crawler / models / manisha1 下方找到在训练过程中创建的模型。

   在面向爬虫场景的 build 中生成的可执行文件将是我们用于生成下一个文件的名称。训练完成时文档名称将是包含字节文件扩展名的环境的名称。

   如果 “manisha1.exe” 是可执行文件，则生成的字节文件将是 “manisha1.bytes”，它遵循规则 <env-name>.bytes。

1. 将生成的字节文件从模型文件夹拷贝至 TF Models 子文件夹。
2.
3. 打开 Unity IDE 并选择爬虫场景。
4.
5. 从场景层级结构中选择大脑。
6.
7. 将大脑的类型改为内部。
8.
9.

将 .bytes 文件从项目文件夹拖到大脑检查器中的图形模型占位符，然后点击播放运行它。

10.

输出应类似于下面的屏幕截图。

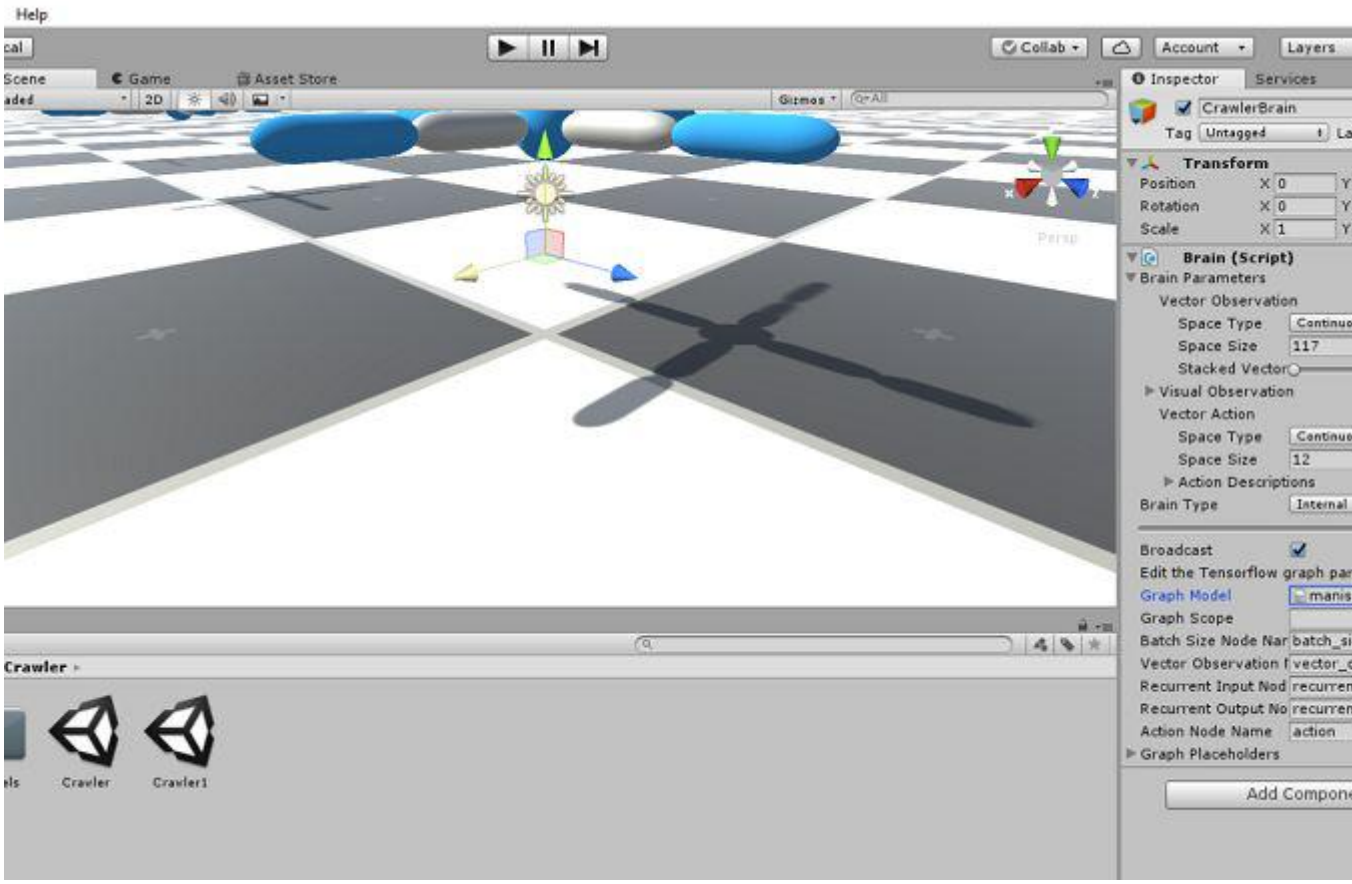


图 7.未激活内部大脑时创建的可执行文件。

然后我们用内部大脑构建项目。生成可执行文件。

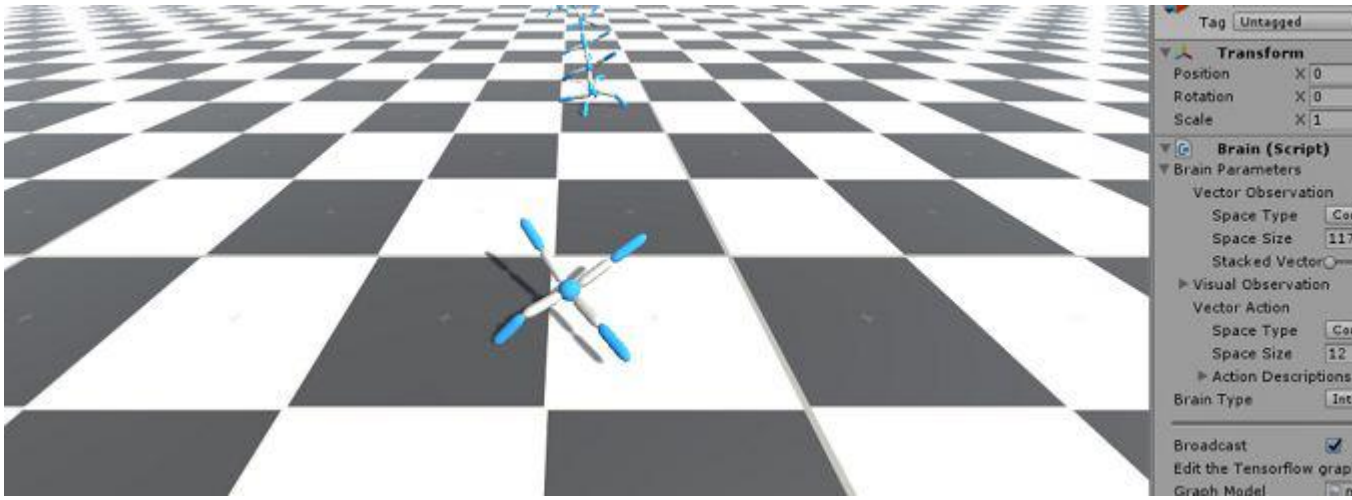


图 8.使用内部大脑构建项目后的可执行文件。

总结

Unity 和英特尔携手降低游戏开发人员的入门门槛，支持他们用令人惊叹的 AI 行为增强沉浸感。智能代理能够执行动态且极具吸引力的行为，为打造更真实、更好的用户体验带来了希望。游戏开发领域使用强化学习仍处于早期阶段，但它有可能成为一项颠覆性技术。使用本文列出的技术和资源开始创建您自己的高级游戏，感受它们的精彩之处。