Universe

una plataforma de software para medir y entrenar la inteligencia general de una IA a través del suministro mundial de juegos, sitios web y otras aplicaciones.

Universe permite que un agente de IA use una computadora como lo hace un humano: mirando los píxeles de la pantalla y operando un teclado y mouse virtuales. Debemos capacitar a los sistemas de inteligencia artificial en la gama completa de tareas que esperamos que resuelvan, y Universe nos permite capacitar a un solo agente en cualquier tarea que un humano pueda completar con una computadora.

Infraestructura del universe

Universe expone una amplia gama de entornos a través de una interfaz común: el agente opera un escritorio remoto al observar los píxeles de una pantalla y producir comandos de teclado y mouse. El entorno expone un servidor VNC y la universebiblioteca convierte al agente en un cliente VNC.

Ambientes

Ya hemos integrado una gran cantidad de entornos en Universe, y los vemos como solo el comienzo. Cada entorno está empaquetado como una imagen Docker y aloja dos servidores que se comunican con el mundo exterior: el servidor VNC que envía píxeles y recibe comandos de teclado / mouse, y un servidor WebSocket que envía la señal de recompensa para las tareas de aprendizaje de refuerzo (así como cualquier información auxiliar como texto o diagnósticos) y acepta mensajes de control (como el ID de entorno específico para ejecutar).

Integraciones futuras

Esta infraestructura es de uso general: podemos integrar cualquier juego, sitio web o aplicación que pueda ejecutarse en un contenedor Docker (más conveniente) o una máquina virtual de Windows (menos conveniente).

Gym

es juego de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo (RL). Con Universe, cualquier programa puede convertirse en un entorno de gimnasio. Universe funciona al iniciar automáticamente el programa detrás de un escritorio remoto VNC : no necesita acceso especial a los componentes internos del programa, el código fuente o las API de bot.

Observaciones

Si alguna vez queremos hacerlo mejor que tomar acciones aleatorias en cada paso, probablemente sería bueno saber realmente qué están haciendo nuestras acciones para el medio ambiente.

La stepfunción del entorno devuelve exactamente lo que necesitamos. De hecho, stepdevuelve cuatro valores. Estos son:

observation(objeto)

un objeto específico del entorno que representa su observación del entorno. Por ejemplo, datos de píxeles de una cámara, ángulos y velocidades conjuntas de un robot, o el estado del tablero en un juego de tablero.

reward(flotante)

cantidad de recompensa lograda por la acción anterior. La escala varía entre entornos, pero el objetivo siempre es aumentar su recompensa total.

done(booleano)

si es hora de resetvolver al medio ambiente. La mayoría de las tareas (pero no todas) se dividen en episodios bien definidos y donese Trueindica que el episodio ha finalizado. (Por ejemplo, tal vez el poste se inclinó demasiado, o perdió su última vida).

info(dict)

información de diagnóstico útil para la depuración. A veces puede ser útil para el aprendizaje (por ejemplo, puede contener las probabilidades en bruto detrás del último cambio de estado del entorno). Sin embargo, las evaluaciones oficiales de su agente no pueden usar esto para aprender. Esto es solo una implementación del clásico "ciclo agente-entorno". Cada paso de tiempo, el agente elige un actiony el entorno devuelve un observationy a reward.

Esto es solo una implementación del clásico "ciclo agente-entorno". Cada paso de tiempo, el agente elige un actiony el entorno devuelve un observationy a reward.

Ambientes Disponibles

Gym viene con un conjunto diverso de entornos que van de fácil a difícil e involucran muchos tipos diferentes de datos. Vea la lista completa de entornos para obtener una vista panorámica.

Control clásico y texto de juguete: completar tareas a pequeña escala, principalmente de la literatura RL. Están aquí para ayudarlo a comenzar. Algorítmico : realice cálculos como agregar números de varios dígitos y revertir secuencias. Uno podría objetar que estas tareas son fáciles para una computadora. El desafío es aprender estos algoritmos únicamente a partir de ejemplos. Estas tareas tienen la buena propiedad de que es fácil variar la dificultad variando la longitud de la secuencia. Atari : juega juegos clásicos de Atari. Hemos integrado el Arcade Learning Environment (que ha tenido un gran impacto en la investigación de aprendizaje por refuerzo) en un formulario fácil de instalar . Robots 2D y 3D :4 controlan un robot en simulación. Estas tareas utilizan el motor de física MuJoCo , que fue diseñado para una simulación de robot rápida y precisa. Se incluyen algunos entornos de un punto de referencia reciente realizado por investigadores de UC Berkeley (que por cierto se unirán a nosotros este verano). MuJoCo es un software propietario, pero ofrece licencias de prueba gratuitas .

Ejemplo

```
:
import universe # register Universe environments into Gym
import numpy as np
import os
import gym
from gym import error, spaces
from gym import utils
from gym.utils import seeding
```

```
: env = gym.make('Breakout-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
```

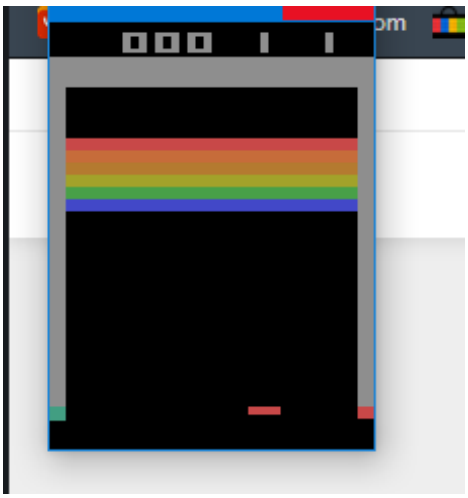
```
: try:
    import atari_py
except ImportError as e:
    raise error.DependencyNotInstalled(
            "{}. (HINT: you can install Atari dependencies by running "
            "'pip install gym[atari]'.)".format(e))


def to_ram(ale):
    ram_size = ale.getRAMSize()
    ram = np.zeros((ram_size), dtype=np.uint8)
    ale.getRAM(ram)
    return ram


class AtariEnv(gym.Env, utils.EzPickle):
    metadata = {'render.modes': ['human', 'rgb_array']}

    def __init__(
            self,
            game='pong',
            mode=None,
            difficulty=None,
            obs_type='ram',
            frameskip=(2, 5),
            repeat_action_probability=0.,
            full_action_space=False):
        """Frameskip should be either a tuple (indicating a random range to
        choose from, with the top value exclude), or an int."""

        utils.EzPickle.__init__(
                self,
                game,
                mode,
                difficulty,
                obs_type,
                frameskip,
                repeat_action_probability)
        assert obs_type in ('ram', 'image')

        self.game = game
```

```
            repeat_action_probability)
    assert obs_type in ('ram', 'image')

    self.game = game
    self.game_path = atari_py.get_game_path(game)
    self.game_mode = mode
    self.game_difficulty = difficulty

    if not os.path.exists(self.game_path):
        msg = 'You asked for game %s but path %s does not exist'
        raise IOError(msg % (game, self.game_path))
    self._obs_type = obs_type
    self.frameskip = frameskip
    self.ale = atari_py.ALEInterface()
    self.viewer = None

    # Tune (or disable) ALE's action repeat:
    # https://github.com/openai/gym/issues/349
    assert isinstance(repeat_action_probability, (float, int)), \
        "Invalid repeat_action_probability: {!r}".format(repeat_action_probability)
    self.ale.setFloat(
            'repeat_action_probability'.encode('utf-8'),
            repeat_action_probability)

    self.seed()

    self._action_set = (self.ale.getLegalActionSet() if full_action_space
                        else self.ale.getMinimalActionSet())
    self.action_space = spaces.Discrete(len(self._action_set))

    (screen_width, screen_height) = self.ale.getScreenDims()
    if self._obs_type == 'ram':
        self.observation_space = spaces.Box(low=0, high=255, dtype=np.uint8, shape=(128,))
    elif self._obs_type == 'image':
        self.observation_space = spaces.Box(low=0, high=255, shape=(screen_height, screen_width, 3), dtype=np.uint8)
    else:
        raise error.Error('Unrecognized observation type: {}'.format(self._obs_type))

def seed(self, seed=None):
    self.np_random, seed1 = seeding.np_random(seed)
    # Derive a random seed. This gets passed as a uint, but gets
    # checked as an int elsewhere, so we need to keep it below
    # 2**31.
    seed2 = seeding.hash_seed(seed1 + 1) % 2**31
    # Empirically, we need to seed before loading the ROM.
    self.ale.setInt(b'random_seed', seed2)
    self.ale.loadROM(self.game_path)

    if self.game_mode is not None:
        modes = self.ale.getAvailableModes()

        assert self.game_mode in modes, (
            "Invalid game mode \"{}\" for game {}.\nAvailable modes are: {}"
        ).format(self.game_mode, self.game, modes)
        self.ale.setMode(self.game_mode)

    if self.game_difficulty is not None:
        difficulties = self.ale.getAvailableDifficulties()

        assert self.game_difficulty in difficulties, (
            "Invalid game difficulty \"{}\" for game {}.\nAvailable difficulties are: {}"
        ).format(self.game_difficulty, self.game, difficulties)
        self.ale.setDifficulty(self.game_difficulty)

    return [seed1, seed2]
```

```
        return [seed1, seed2]

    def step(self, a):
        reward = 0.0
        action = self._action_set[a]

        if isinstance(self.frameskip, int):
            num_steps = self.frameskip
        else:
            num_steps = self.np_random.randint(self.frameskip[0], self.frameskip[1])
        for _ in range(num_steps):
            reward += self.ale.act(action)
        ob = self._get_obs()

        return ob, reward, self.ale.game_over(), {"ale.lives": self.ale.lives()}

    def _get_image(self):
        return self.ale.getScreenRGB2()

    def _get_ram(self):
        return to_ram(self.ale)

    @property
    def _n_actions(self):
        return len(self._action_set)

    def _get_obs(self):
        if self._obs_type == 'ram':
            return self._get_ram()
        elif self._obs_type == 'image':
            img = self._get_image()
        return img

    # return: (states, observations)
    def reset(self):
        self.ale.reset_game()
        return self._get_obs()

    def render(self, mode='human'):
        img = self._get_image()
        if mode == 'rgb_array':
            return img
        elif mode == 'human':
            from gym.envs.classic_control import rendering
            if self.viewer is None:
                self.viewer = rendering.SimpleImageViewer()
            self.viewer.imshow(img)
            return self.viewer.isopen

    def close(self):
        if self.viewer is not None:
            self.viewer.close()
            self.viewer = None

    def get_action_meanings(self):
        return [ACTION_MEANING[i] for i in self._action_set]

    def get_keys_to_action(self):
        KEYWORD_TO_KEY = {
            'UP':      ord('w'),
            'DOWN':    ord('s'),
```

```
        for action_id, action_meaning in enumerate(self.get_action_meanings()):
            keys = []
            for keyword, key in KEYWORD_TO_KEY.items():
                if keyword in action_meaning:
                    keys.append(key)
            keys = tuple(sorted(keys))

            assert keys not in keys_to_action
            keys_to_action[keys] = action_id

        return keys_to_action

    def clone_state(self):
        """clone emulator state w/o system state. Restoring this state will
        *not* give an identical environment. For complete cloning and restoring
        of the full state, see `{clone,restore}_full_state()`."""
        state_ref = self.ale.cloneState()
        state = self.ale.encodeState(state_ref)
        self.ale.deleteState(state_ref)
        return state

    def restore_state(self, state):
        """Restore emulator state w/o system state."""
        state_ref = self.ale.decodeState(state)
        self.ale.restoreState(state_ref)
        self.ale.deleteState(state_ref)

    def clone_full_state(self):
        """clone emulator state w/ system state including pseudorandomness.
        Restoring this state will give an identical environment."""
        state_ref = self.ale.cloneSystemState()
        state = self.ale.encodeState(state_ref)
        self.ale.deleteState(state_ref)
        return state

    def restore_full_state(self, state):
        """Restore emulator state w/ system state including pseudorandomness."""
        state_ref = self.ale.decodeState(state)
        self.ale.restoreSystemState(state_ref)
        self.ale.deleteState(state_ref)


ACTION_MEANING = {
    0: "NOOP",
    1: "FIRE",
    2: "UP",
    3: "RIGHT",
    4: "LEFT",
    5: "DOWN",
    6: "UPRIGHT",
    7: "UPLEFT",
    8: "DOWNRIGHT",
    9: "DOWNLEFT",
    10: "UPFIRE",
    11: "RIGHTFIRE",
    12: "LEFTFIRE",
    13: "DOWNFIRE",
    14: "UPRIGHTFIRE",
    15: "UPLEFTFIRE",
    16: "DOWNRIGHTFIRE",
    17: "DOWNLEFTFIRE",
}
```

Resultados



Referencias

https://gym.openai.com/docs/

Juego repositorio https://gym.openai.com/envs/Breakout-v0/