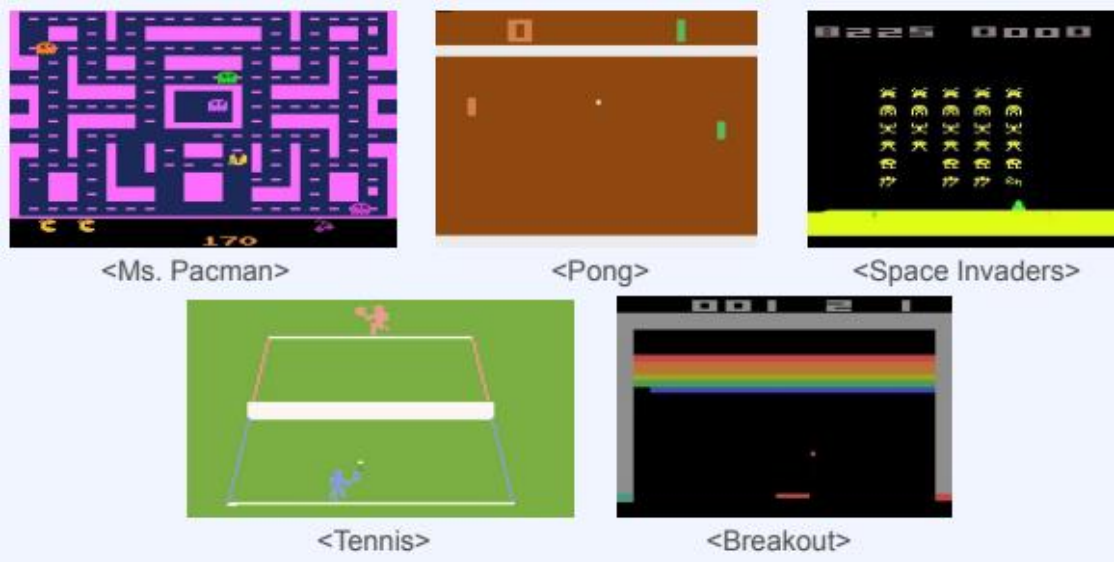
8강. DQN

Contents

- DQN (Minh et al., 2015)
- Pybullet Environment
- Kuka Robotic Arm

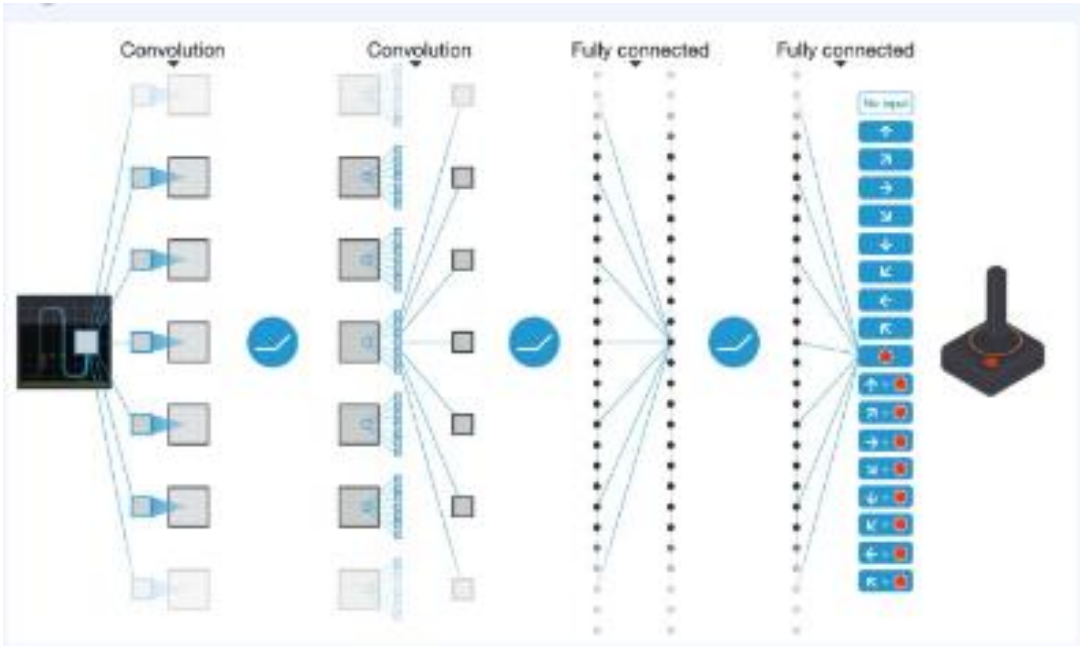
DQN

- Human-level control through deep RL(Minh et al., 2015)
- 1st success case using DNN (inspired by AlexNet)
- Solved complex env. and high dimensional input (image)
- No hand-crafted feature
- In Atari 2600 game,
  the agent is better
  than an human expert



DQN

- DQN = Deep Neural Network + Q-learning
- Used CNN architecture for input images
- Proposed solutions for 2 problems of previous approach to using deep neural network in Q-learning



DQN

- Episode data is time-correlated data:
  - Correlation between data makes learning harder (NOT i.i.d)

  ➔ Experience Replay: Uniform sample from the transition data buffer.

  ➔ Q-learning is off-policy. So, using old data from other distribution is okay
- Target oscillation problem:
  - Note in $r + \gamma \max_{a'} Q(s', a')$, $Q(s', a')$ is sharing parameters with $Q(s, a)$

  So, the target is changing while learning $Q(s, a)$

  ➔ Target Network: Manage a separate neural network to fix the target, while updating the target once in a while.

DQN

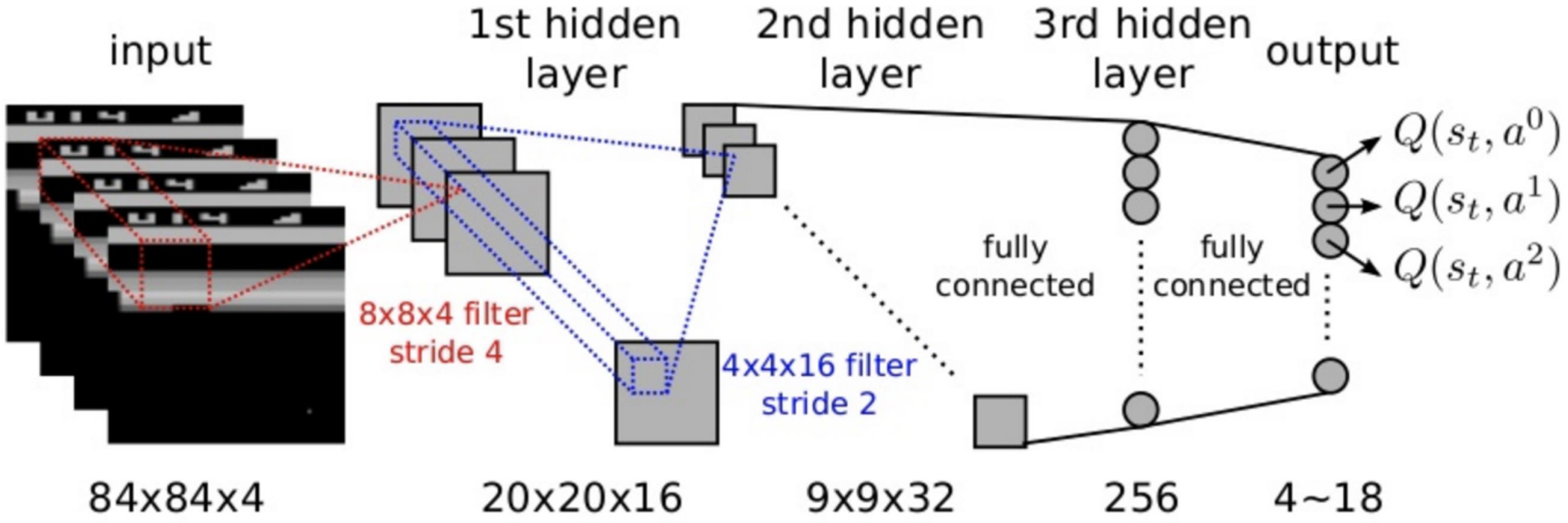
$$L_i(\theta_i) = E_{(s,\, a,\, r,\, s') \sim U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^{-}) - Q(s, a; \theta_i))^2]$$

- θ_i : parameters to learn in i-th iteration
- θ_i^{-}: parameters to calculate target value at i-th iteration
- U(D) : replay memory to store transitions(a collection of data by policies, $\pi_0\, \pi_1$ … π_i .
- Note that the data distribution is different from the latest policy π_i , but q-learning is off-policy learning, so it's okay

DQN

- Original image 210x160 (1~127 RGB)
- Preprocessing to reduce computation and memory
- 'Flickering' is a problem that makes pixel value fluctuation
  - ➔ To solve it, take max. pixel RGB value btw. t-1 pixel and t pixel (e.g. (57,34,72),(88,34,21) ➔ (88,34,72)
- Transform the RGB into Black and white, resizing into 84x84
- Stack recent 4 frames after the preprocessing
  - Note the agent could not predict the direction of moving object with 1 image. (Env. is not 1st order MDP)

DQN


input
1st hidden layer
2nd hidden layer
3rd hidden layer
output
$Q(s_t, a^0)$
$Q(s_t, a^1)$
$Q(s_t, a^2)$
fully connected
fully connected
8x8x4 filter
stride 4
4x4x16 filter
stride 2
84x84x4
20x20x16
9x9x32
256
4~18

DQN

- Reward shaping: +1 ~ -1
- Optimizer: RMSProp
- Batch size: 32
- ε-greedy: annealing from 1.0 - 0.1
- Total 50M frames (38 days game play time)
- Replay memory store (recent 1M transition)
- Frame-skipping: agent choose an action every k frame to reduce computation (same actions within k frames)
- Hyper-parameter search using hand-tunning (too much computation to use grid search, etc.,)

DQN

s_t : a sequence of state (from 1 to t)
x_t: image at t

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

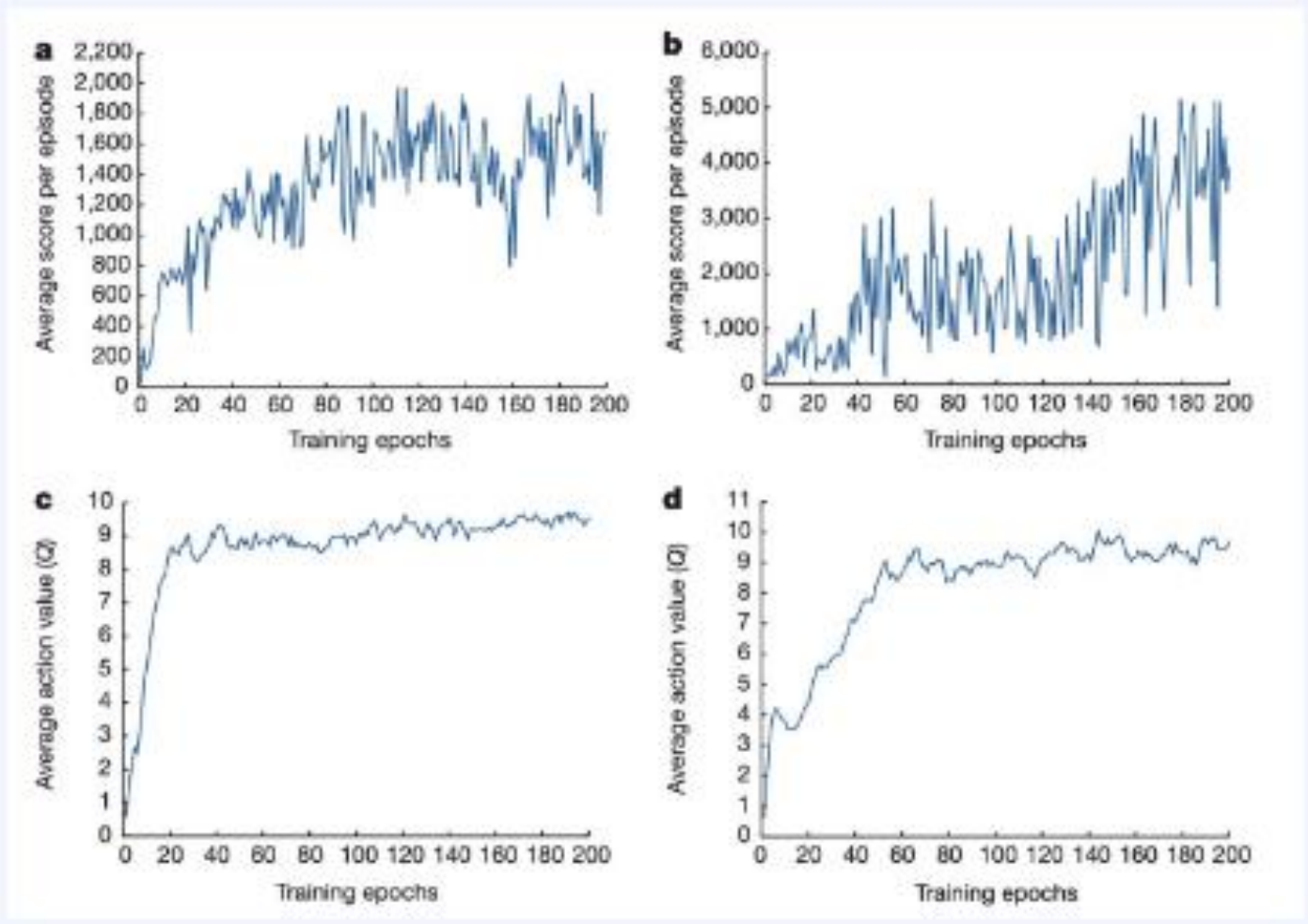
    Every C steps reset $\hat{Q} = Q$

  End For

End For

Preprocessed sequence is a stacked 4 images of size 84x84

DQN



GD every 4 transition

8 update per batch

Avg. action value is the shaped reward value (Not the same as avg. return)

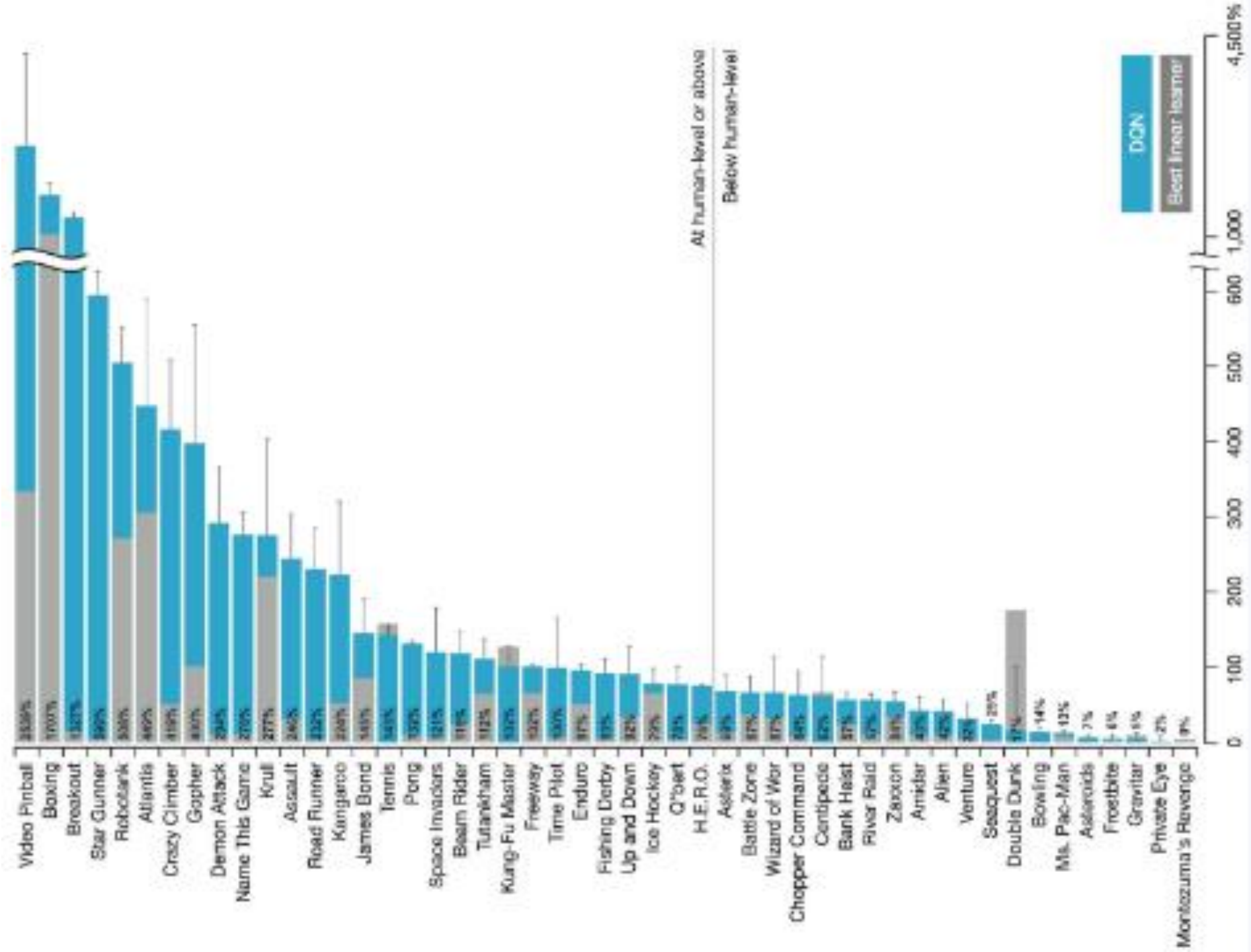
In evaluation, epsilon is 0.05

DQN

Previous SOTA for Atari 2600 is Best linear learner

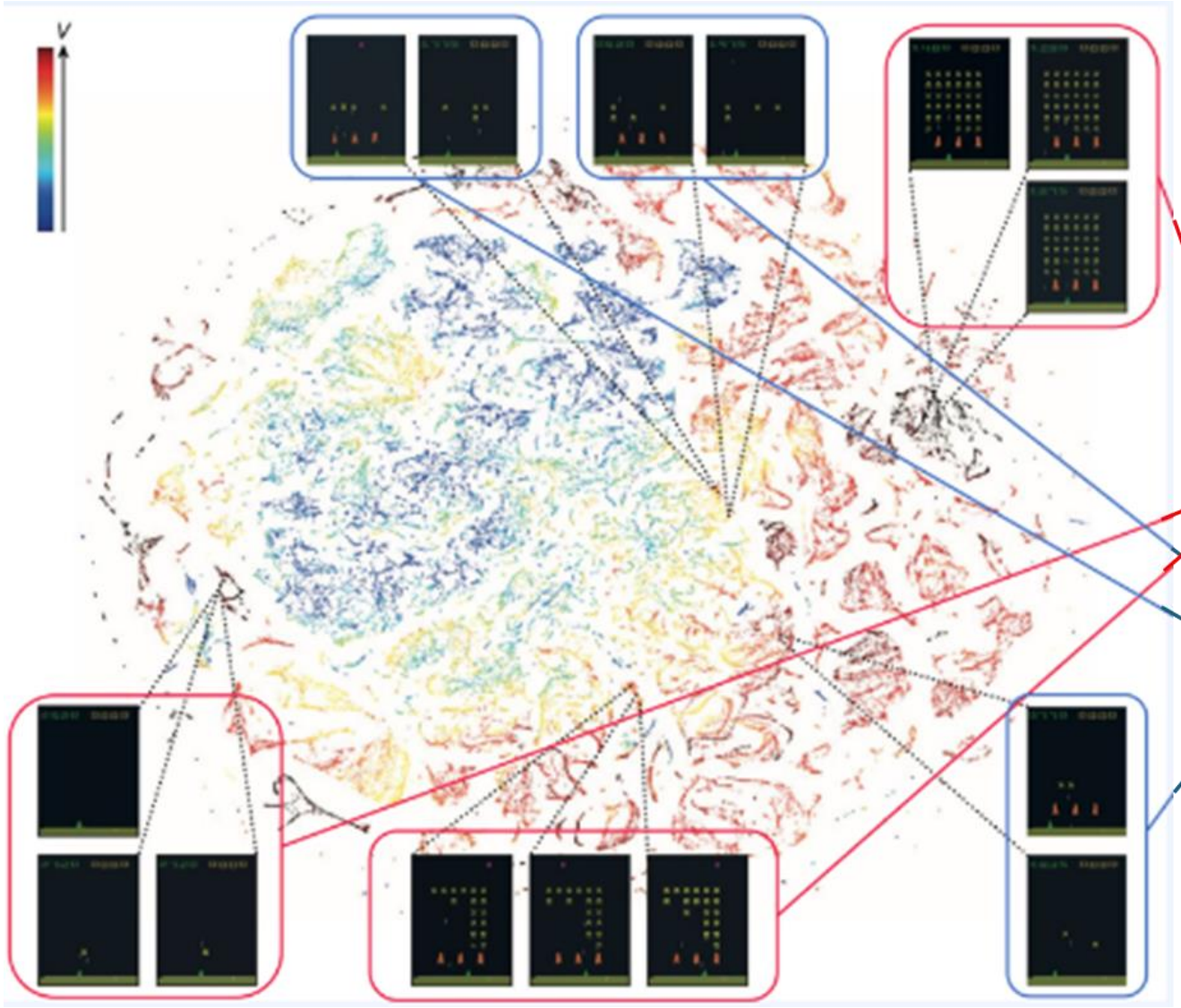
DQN is better than Best linear leaner in 43 / 49

Assuming human-level is 75% of an expert, DQN is better than humans for 29 games (For example, DQN is 25.4 times better in playing Video Pinball)



DQN

T-SNE analysis of the last hidden layer of Space Invaders network
(t-SNE: 2 dim. Visualization of vector data)



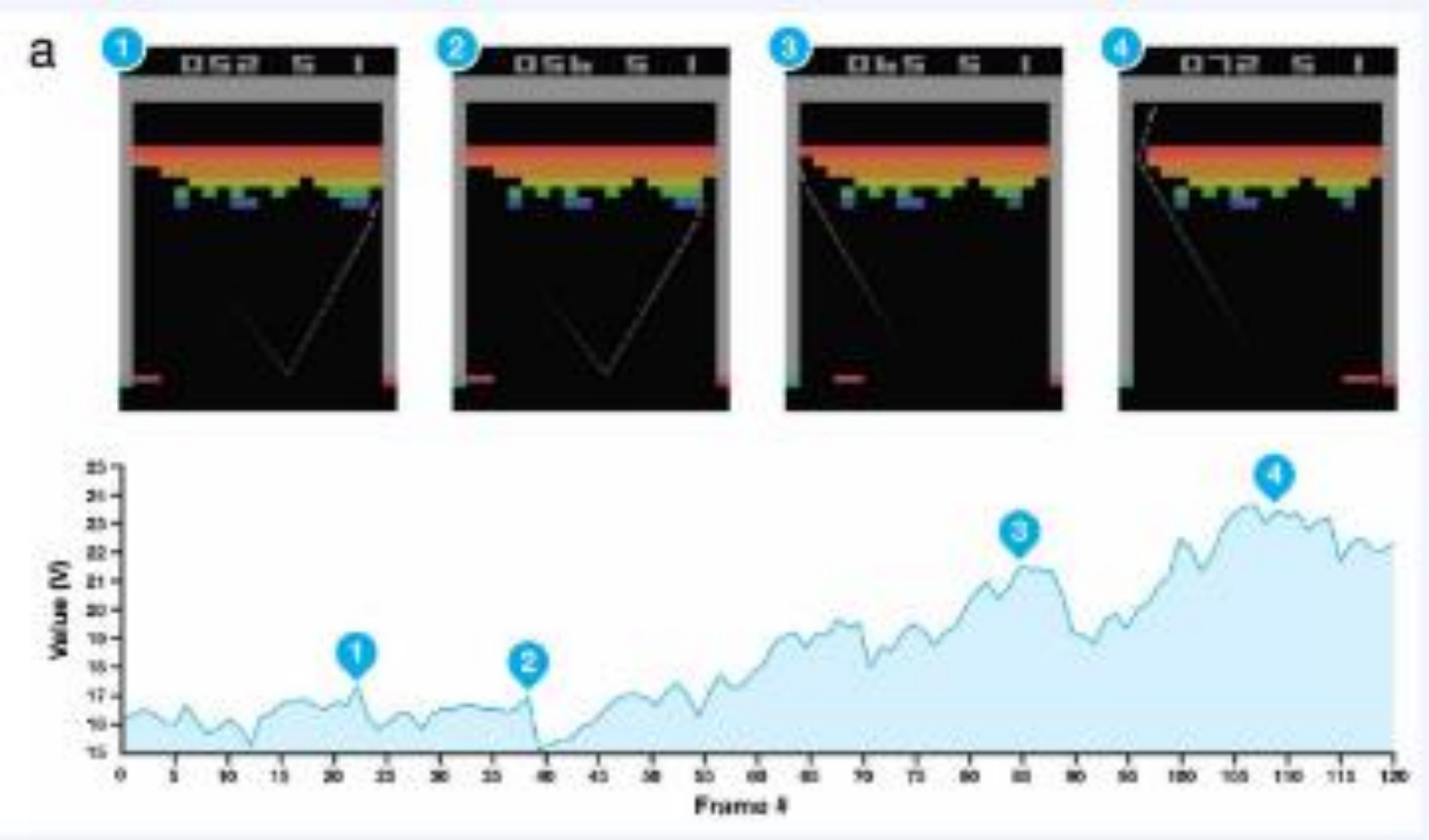
Similar states has points at similar locations

Although different visually, the states with similar expected reward(action-value) have similar locations

DQN

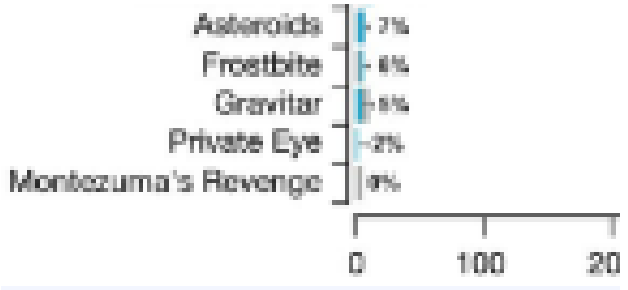
Agent's behavior shows that it learned a long-term strategy

E.g., In break out, the agent learned that the state number 4 has much higher value



DQN

- But DQN suffers in a game that requires a longer-term strategy



- Sparse reward problem
- Deceptive reward

Pybullet

Pybullet

- Anaconda Install
- 가상환경: conda create –n [이름] python=3.8
- Pytorch 설치: go https://pytorch.kr/get-started/locally/ ,
- Gym Package Install:
  - conda install -c conda-forge box2d-py
  - pip install gym==0.23.1
  - **Pip install pybullet**
  - Pip install numpy==1.23.1
  - Pip install matplotlib

Pybullet

- A suite of RL Gym Environments are installed during

"pip install pybullet“

- This includes PyBullet versions of the OpenAI Gym environments such as ant, hopper, humanoid and walker.
- There are also environments that apply in simulation as well as on real robots, such as the Ghost Robotics Minitaur quadruped, the MIT racecar and the KUKA robot arm grasping environments.

Pybullet

- The source code of pybullet, pybullet_envs, pybullet_data and the examples are here: https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym

Pybullet

- You can train the environments with RL training algorithms such as DQN, PPO, TRPO and DDPG.
- Several pre-trained examples are available, you can enjoy them like this:

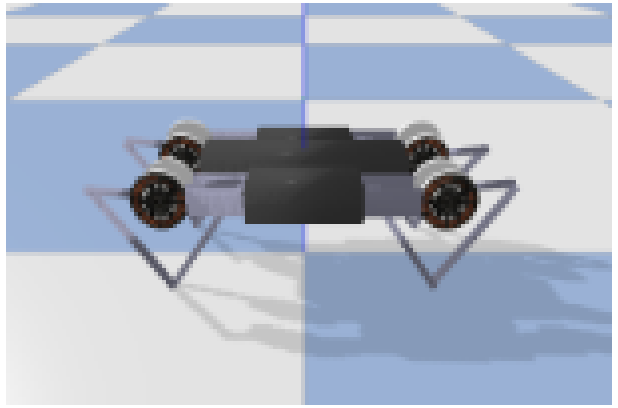
- pip install gym, pybullet, tensorflow, torch

- python -m pybullet_envs.examples.kukaGymEnvTest

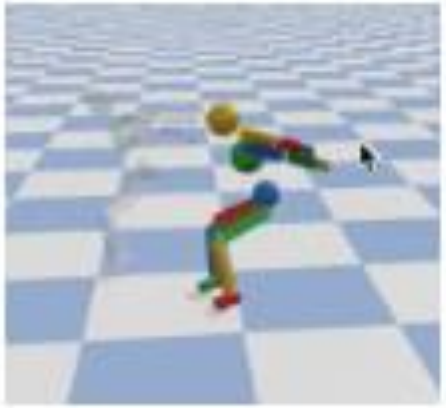
- python -m pybullet_envs.examples.minitaur_gym_env_example

Pybullet

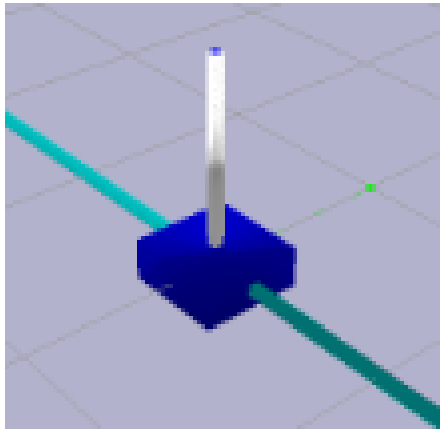
MinitaurBulletEnv-v0



HumanoidDeepMimic*BulletEnv-v1



CartPoleContinuousBulletEnv-v0

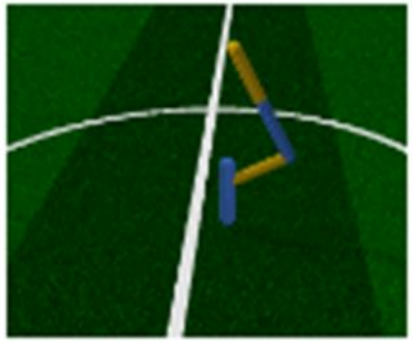


Pybullet

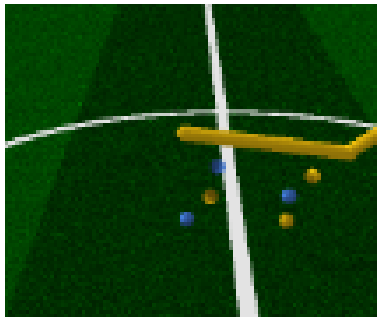
HumanoidBulletEnv-v0

AntBulletEnv-v0

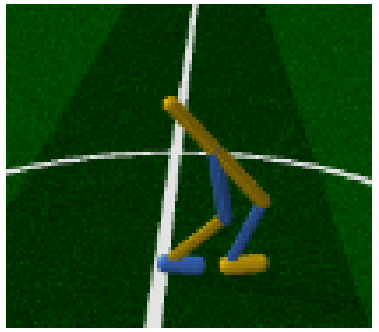
HopperBulletEnv-v0



HalfCheetahBulletEnv-v0

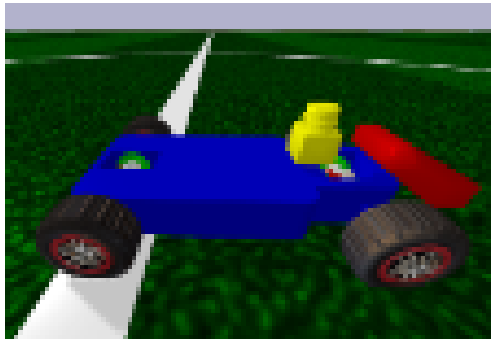


Walker2DBulletEnv-v0



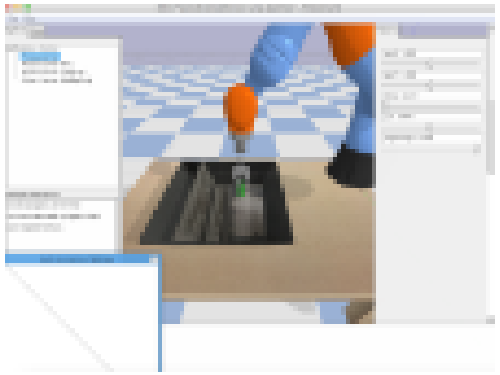
Pybullet

RacecarBulletEnv-v0

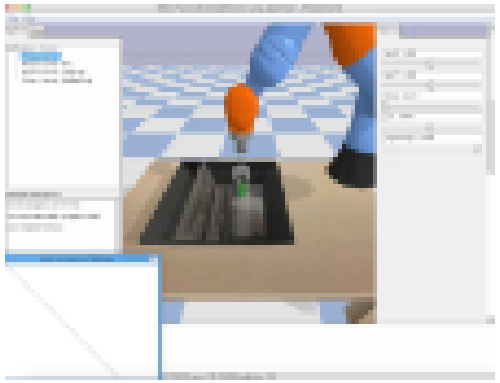


RacecarZedBulletEnv-v0

KukaBulletEnv-v0



KukaCamBulletEnv-v0



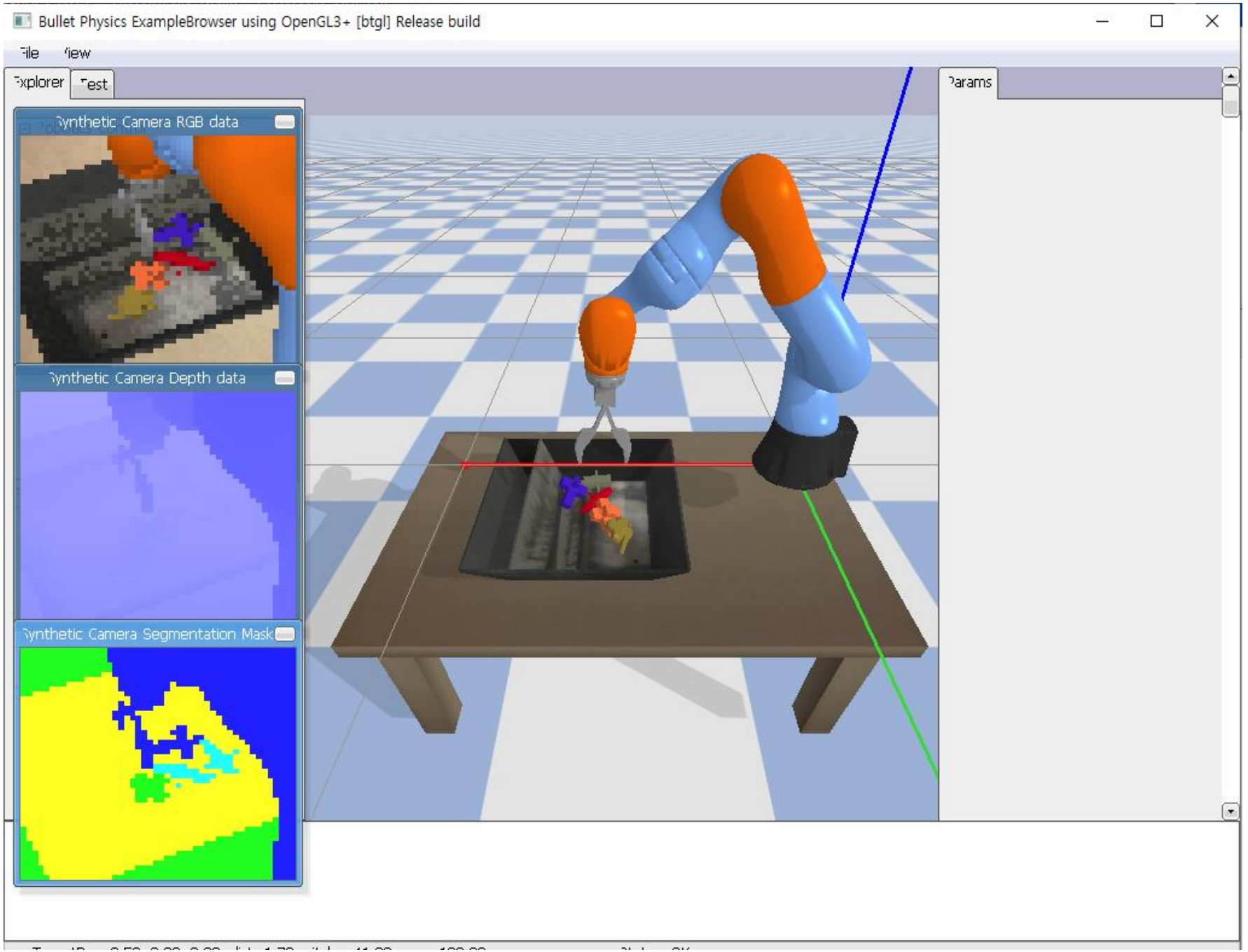
Kuka Robotic Arm

Kuka Robotic Arm

Reward

- The reward is binary and provided only at the last step.
- 1 if one of the objects is above height .2 (successful grasp)
- 0 for a failed grasp
- The arm has a fixed number of timesteps (T = 15) to find a good grasp, at which the episode ends

Kuka Pybullet



Import libraries

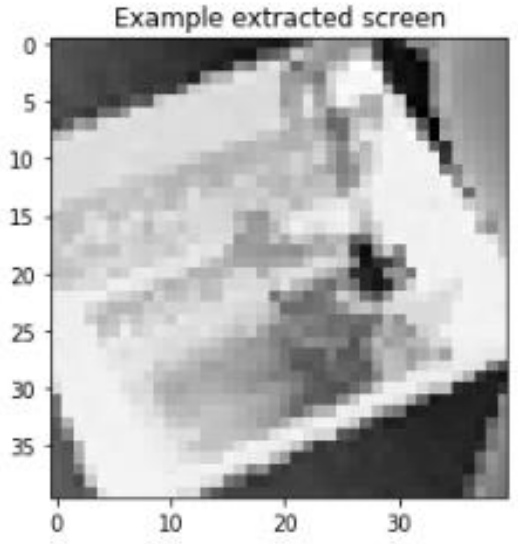
```
!pip install pybullet
!pip install tensorboardX
import gym
import math
import random
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from collections import namedtuple
import collections
from itertools import count
import timeit
from datetime import timedelta
from PIL import Image
from tensorboardX import SummaryWriter

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms as T
```

```
from pybullet_envs.bullet.kuka_diverse_object_gym_env import KukaDiverseObjectEn
from gym import spaces
import pybullet as p

env = KukaDiverseObjectEnv(renders=False, isDiscrete=True, removeHeightHack=Fals
env.cid = p.connect(p.DIRECT)

# set up matplotlib
is_ipython = 'inline' in matplotlib.get_backend()
if is_ipython:
    from IPython import display

plt.ion()

# if gpu is to be used
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

States

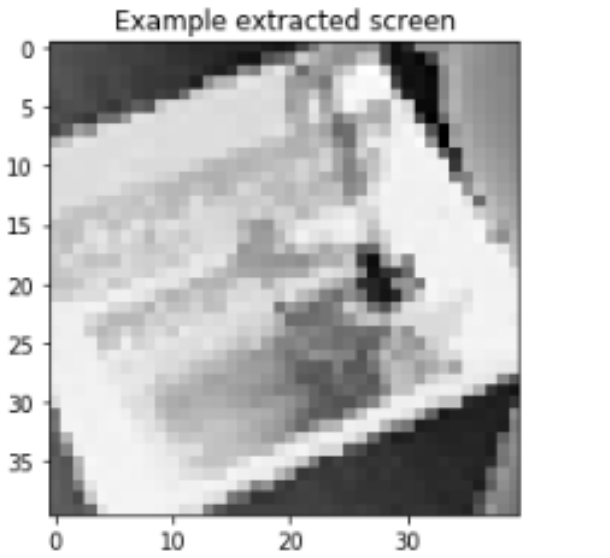
- The input image signal is (48, 48, 3) RGB image and timestep t
  - TS t is included in the state, since the policy must know how many steps remain in the episode to decide whether it must immediately move into a good grasping position.

RGB from the viewpoint of the robot's camera



- NN can solve the task purely by looking at the scene.
  - So, we'll use a stack of consecutive screens as an input. In this way, we are hoping to capture the dynamics of the environment.

Input extraction




```
preprocess = T.Compose([T.ToPILImage(),
                    T.Grayscale(num_output_channels=1),
                    T.Resize(40, interpolation=Image.CUBIC),
                    T.ToTensor()])

def get_screen():
    global stacked_screens
    # Returned screen requested by gym is 400x600x3, but is sometimes larger
    # such as 800x1200x3. Transpose it into torch order (CHW).
    screen = env._get_observation().transpose((2, 0, 1))
    # Convert to float, rescale, convert to torch tensor
    # (this doesn't require a copy)

    screen = np.ascontiguousarray(screen, dtype=np.float32) / 255
    screen = torch.from_numpy(screen)
    # Resize, and add a batch dimension (BCHW)
    return preprocess(screen).unsqueeze(0).to(device)



env.reset()
plt.figure()
plt.imshow(get_screen().cpu().squeeze(0)[-1].numpy(),cmap='Greys',
           interpolation='none')
plt.title('Example extracted screen')
plt.show()
```

Action

- The agent has to decide between 7 actions (2+2+2+1):
  1. Manipulator moves in x (2) or y (2) direction
     - Actions correspond to changes in gripper pose (displacement)
     - Assumes that the velocity for each directions equal($\pm$0.06)
     - Gripper automatically move down for each action (z: -0.06)
  2. Vertical angle offset (2) for the gripper :
     - The arm moves via position control of vertically- oriented gripper
     - We assume that the vertical angle offset for the gripper($\pm$0.25)
     - Gripper automatically closes if it moves below a fixed height threshold
  3. Not moving at all (1) : So that the manipulator can grasp an object

Neural Net.

s_t

(48, 48, 3)
5장

CNN:
32 filters, 8x8,
stride 4,
ReLU

CNN:
64 filters, 4x4,
stride 2,
ReLU

CNN:
64 filters, 3x3,
stride 1,
ReLU

FC:
(d, 512),
ReLU

FC:
(512, |A|)

$Q(s, d_x = d_v)$,
$Q(s, d_x = -d_v)$,
$Q(s, d_y = d_v)$,
$Q(s, d_y = -d_v)$,
$Q(s, d_a = 0.25)$,
$Q(s, d_a = -0.25)$

$d_v = 0.06$ is the velocity in PyBullet,
d_a is vertical angle offset for the gripper,

Q-network

```
class DQN(nn.Module):
    def __init__(self, h, w, outputs):
        super(DQN, self).__init__()
        self.conv1 = nn.Conv2d(STACK_SIZE, 32, kernel_size=8, stride=4)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3, stride=1)

        # Number of Linear input connections depends on output of conv2d layers
        # and therefore the input image size, so compute it.
        def conv2d_size_out(size, kernel_size = 5, stride = 2):
            return (size - (kernel_size - 1) - 1) // stride  + 1
        convw = conv2d_size_out(conv2d_size_out(conv2d_size_out(w,8,4),4,2),3,1)
        convh = conv2d_size_out(conv2d_size_out(conv2d_size_out(h,8,4),4,2),3,1)
        linear_input_size = convw * convh * 64
        self.linear = nn.Linear(linear_input_size, 512)
        self.head = nn.Linear(512, outputs)

    # Called with either one element to determine next action, or a batch
    # during optimization. Returns tensor([[leftOexp,rightOexp]...]).
    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = F.relu(self.bn2(self.conv2(x)))
        x = F.relu(self.conv3(x))
        x = F.relu(self.linear(x.view(x.size(0), -1)))
        return self.head(x)
```

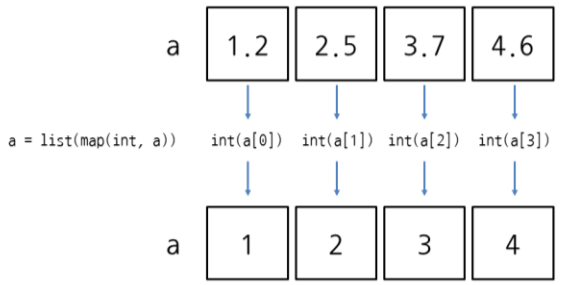
```
BATCH_SIZE = 32
GAMMA = 0.99
EPS_START = 0.9
EPS_END = 0.1
EPS_DECAY = 200
EPS_DECAY_LAST_FRAME = 10**4
TARGET_UPDATE = 1000
LEARNING_RATE = 1e-4
```

```
# Get screen size so that we can initialize layers correctly based on shape
# returned from pybullet (48, 48, 3).
init_screen = get_screen()
_, _, screen_height, screen_width = init_screen.shape

# Get number of actions from gym action space
n_actions = env.action_space.n

policy_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net = DQN(screen_height, screen_width, n_actions).to(device)
target_net.load_state_dict(policy_net.state_dict())
target_net.eval()

optimizer = optim.Adam(policy_net.parameters(), lr=LEARNING_RATE)
memory = ReplayMemory(10000)


eps_threshold = 0

def select_action(state, i_episode):
    global steps_done
    global eps_threshold
    sample = random.random()
    eps_threshold = max(EPS_END, EPS_START - i_episode / EPS_DECAY_LAST_FRAME)
    if sample > eps_threshold:
        with torch.no_grad():
            # t.max(1) will return largest column value of each row.
            # second column on max result is index of where max element was
            # found, so we pick action with the larger expected reward.
            return policy_net(state).max(1)[1].view(1, 1)
    else:
        return torch.tensor([[random.randrange(n_actions)]], device=device, dtype=torch.long)
```

train

*transitions: (s,a,s',r), (s,a,s',r),..
*zip: (s,s,..),(a,a,...),(s',s;,..),(r,r,...)
batch is Transition:
(s=(s,s..), a=(a,a,...),s'=(s',s;,..), r=(r,r,...))
batch.state = (s,s,...) # (batch, state_dim)
batch.action = (a,a,..) # (batch, action_dim)

list(map(function, list)), or
tuple(map(function, tuple)




```
def optimize_model():
    if len(memory) < BATCH_SIZE:
        return
    transitions = memory.sample(BATCH_SIZE)

    batch = Transition(*zip(*transitions))

    non_final_mask = torch.tensor(tuple(map(lambda s: s is not None,
                                            batch.next_state)), device=device, dtype=torch.bool)
    non_final_next_states = torch.cat([s for s in batch.next_state
                                                if s is not None])
    state_batch = torch.cat(batch.state)
    action_batch = torch.cat(batch.action)
    reward_batch = torch.cat(batch.reward)

    state_action_values = policy_net(state_batch).gather(1, action_batch)

    next_state_values = torch.zeros(BATCH_SIZE, device=device)
    next_state_values[non_final_mask] = target_net(non_final_next_states).max(1)[0].detach()
    # Compute the expected Q values
    expected_state_action_values = (next_state_values * GAMMA) + reward_batch

    # Compute Huber loss
    loss = F.smooth_l1_loss(state_action_values, expected_state_action_values.unsqueeze(1))

    # Optimize the model
    optimizer.zero_grad()
    loss.backward()
```

train

```
num_episodes = 10000000
writer = SummaryWriter()
total_rewards = []
ten_rewards = 0
best_mean_reward = None
start_time = timeit.default_timer()
for i_episode in range(num_episodes):
    # Initialize the environment and state
    env.reset()
    state = get_screen()
    stacked_states = collections.deque(STACK_SIZE*[state],maxlen=STACK_SIZE)
    for t in count():
        stacked_states_t =  torch.cat(tuple(stacked_states),dim=1)
        # Select and perform an action
        action = select_action(stacked_states_t, i_episode)
        _, reward, done, _ = env.step(action.item())
        reward = torch.tensor([reward], device=device)

        # Observe new state
        next_state = get_screen()
        if not done:
            next_stacked_states = stacked_states
            next_stacked_states.append(next_state)
            next_stacked_states_t =  torch.cat(tuple(next_stacked_states),dim=1)
        else:
            next_stacked_states_t = None

        # Store the transition in memory
        memory.push(stacked_states_t, action, next_stacked_states_t, reward)
```

train

```
        # Move to the next state
        stacked_states = next_stacked_states

        # Perform one step of the optimization (on the target network)
        optimize_model()
        if done:
            reward = reward.cpu().numpy().item()
            ten_rewards += reward
            total_rewards.append(reward)
            mean_reward = np.mean(total_rewards[-100:])*100
            writer.add_scalar("epsilon", eps_threshold, i_episode)
            if (best_mean_reward is None or best_mean_reward < mean_reward) and i_episode > 100:
                # For saving the model and possibly resuming training
                torch.save({
                        'policy_net_state_dict': policy_net.state_dict(),
                        'target_net_state_dict': target_net.state_dict(),
                        'optimizer_policy_net_state_dict': optimizer.state_dict()
                        }, PATH)
                if best_mean_reward is not None:
                    print("Best mean reward updated %.1f -> %.1f, model saved" % (best_mean_reward, mean_reward))
                best_mean_reward = mean_reward
            break

    if i_episode%10 == 0:
            writer.add_scalar('ten episodes average rewards', ten_rewards/10.0, i_episode)
            ten_rewards = 0

    # Update the target network, copying all weights and biases in DQN
    if i_episode % TARGET_UPDATE == 0:
        target_net.load_state_dict(policy_net.state_dict())
    if i_episode>=200 and mean_reward>50:
        print('Environment solved in {:d} episodes!\tAverage Score: {:.2f}'.format(i_episode+1, mean_reward))
        break
```

train

```
python .₩bullet_kuka_train.py
```

```
Best mean reward updated 36.0 -> 37.0, model saved
Best mean reward updated 37.0 -> 38.0, model saved
Best mean reward updated 38.0 -> 39.0, model saved
Best mean reward updated 39.0 -> 40.0, model saved
Best mean reward updated 40.0 -> 41.0, model saved
Best mean reward updated 41.0 -> 42.0, model saved
Best mean reward updated 42.0 -> 43.0, model saved
Best mean reward updated 43.0 -> 44.0, model saved
Best mean reward updated 44.0 -> 45.0, model saved
Best mean reward updated 45.0 -> 46.0, model saved
Best mean reward updated 46.0 -> 47.0, model saved
Best mean reward updated 47.0 -> 48.0, model saved
Best mean reward updated 48.0 -> 49.0, model saved
Best mean reward updated 49.0 -> 50.0, model saved
Best mean reward updated 50.0 -> 51.0, model saved
Environment solved in 24033 episodes!   Average Score: 51.00
Average Score: 51.00
Elapsed time: 17:43:47.831820
```

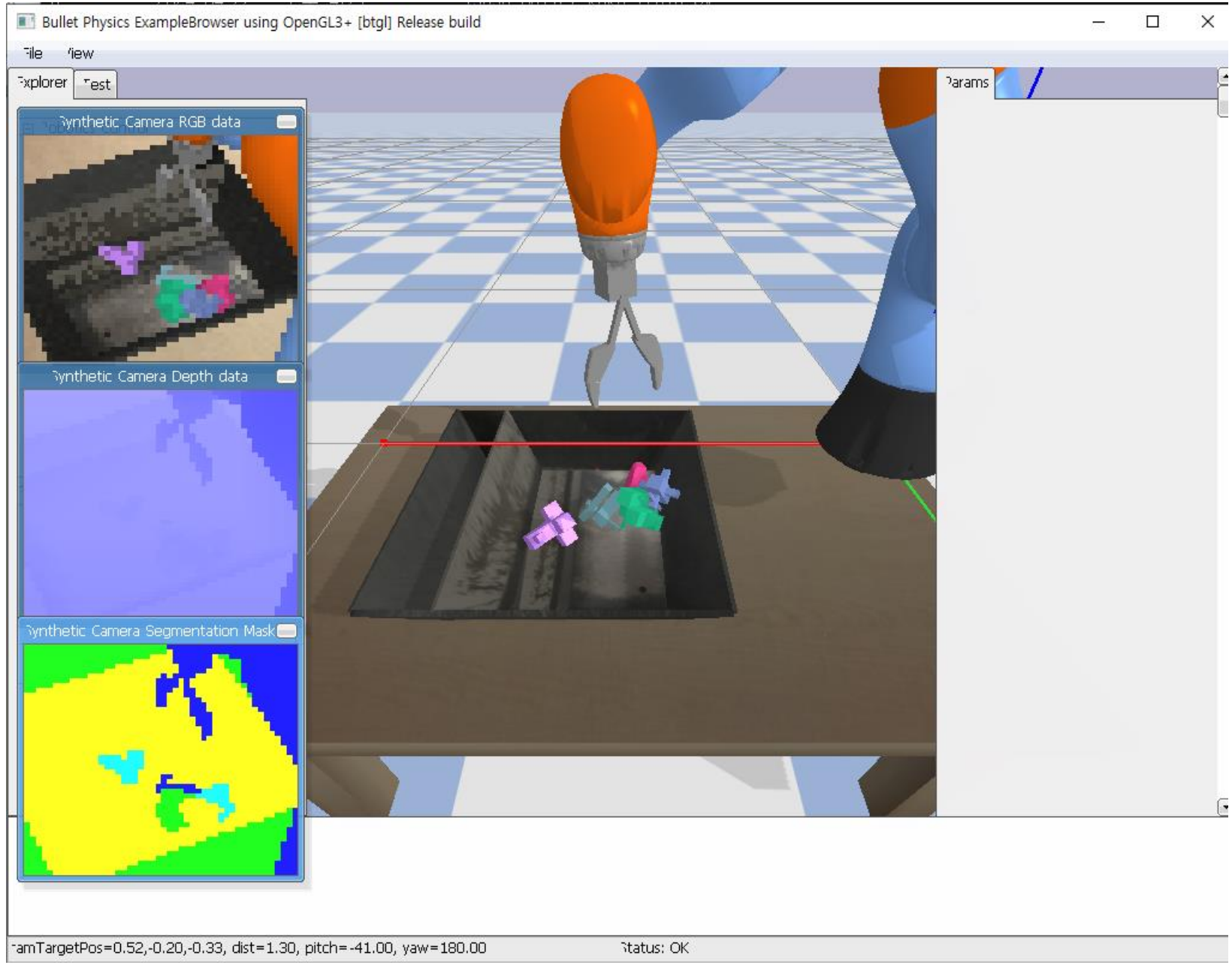
eval

```
episode = 10
scores_window = collections.deque(maxlen=100)  # last 100 scores
env = KukaDiverseObjectEnv(renders=False, isDiscrete=True, removeHeightHack=False, maxSteps=20, isTest=True)
env.cid = p.connect(p.DIRECT)
# load the model
checkpoint = torch.load(PATH)
policy_net.load_state_dict(checkpoint['policy_net_state_dict'])

# evaluate the model
for i_episode in range(episode):
    env.reset()
    state = get_screen()
    stacked_states = collections.deque(STACK_SIZE*[state],maxlen=STACK_SIZE)
    for t in count():
        stacked_states_t =  torch.cat(tuple(stacked_states),dim=1)
        # Select and perform an action
        action = policy_net(stacked_states_t).max(1)[1].view(1, 1)
        _, reward, done, _ = env.step(action.item())
        # Observe new state
        next_state = get_screen()
        stacked_states.append(next_state)
        if done:
            break
    print("Episode: {0:d}, reward: {1}".format(i_episode+1, reward), end="₩n")
```

evaluation

```
python bullet_kuka_eval.py
```

```
Version = 4.6.0 NVIDIA 536.23
Vendor = NVIDIA Corporation
Renderer = NVIDIA GeForce RTX 3090/PCIe/SSE2
b3Printf: Selected demo: Physics Server
starting thread 0
started MotionThreads thread 0 with threadHandle 00000000000005E8
MotionThreadFunc thread started
Episode: 1, reward: 0
Episode: 2, reward: 0
Episode: 3, reward: 0
Episode: 4, reward: 1
Episode: 5, reward: 1
Episode: 6, reward: 1
Episode: 7, reward: 1
Episode: 8, reward: 1
Episode: 9, reward: 1
Episode: 10, reward: 1
numActiveThreads = 0
stopping threads
```


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params
amTargetPos=0.52,-0.20,-0.33, dist=1.30, pitch=-41.00, yaw=180.00
Status: OK

Setup Code Env.

- Git download: https://git-scm.com/downloads
- Package설치:
  - 가상환경: Create: conda create –n [이름] python=3.8
  - Pytorch설치: https://pytorch.kr/get-started/locally/ 명령어라인 복사
  - `pip install pybullet`
  - `pip install tensorboardX`
  - pip install gym==0.23.1
  - Pip install numpy==1.23.1
  - Pip install matplotlib

- Train with ‘bullet_kuka_train.py’
  - Training produce ‘policy_dqn.pt’ 파일 생성

- Evaluate with ‘bullet_kuka_eval.py’