Artificial Intelligence

Deep Reinforcement Learning : Deep Q-Net (2)

한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

인공지능학과
Department of Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

Fall 2022

주요일정

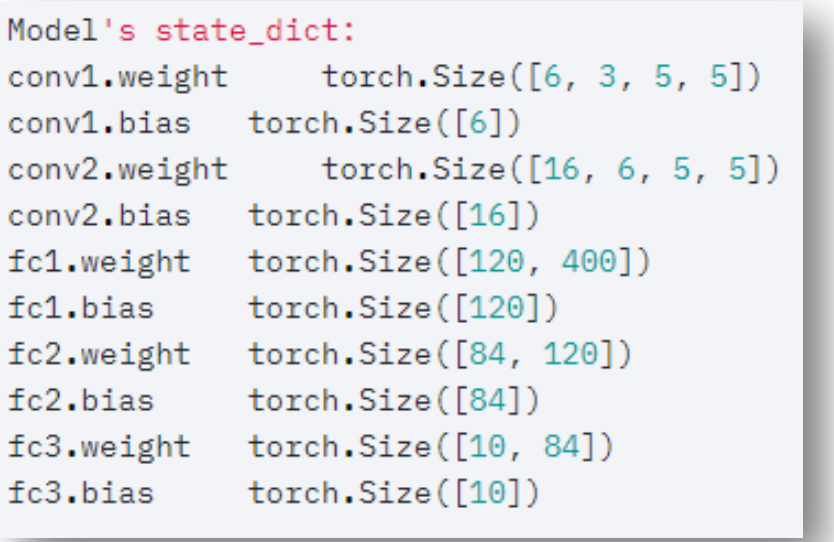
- 12월 6일 휴강 (서울캠퍼스 출장)
- 12월 15일 기말고사 (수업시간)
- 12월 17일 기말프로젝트 마감

Outline

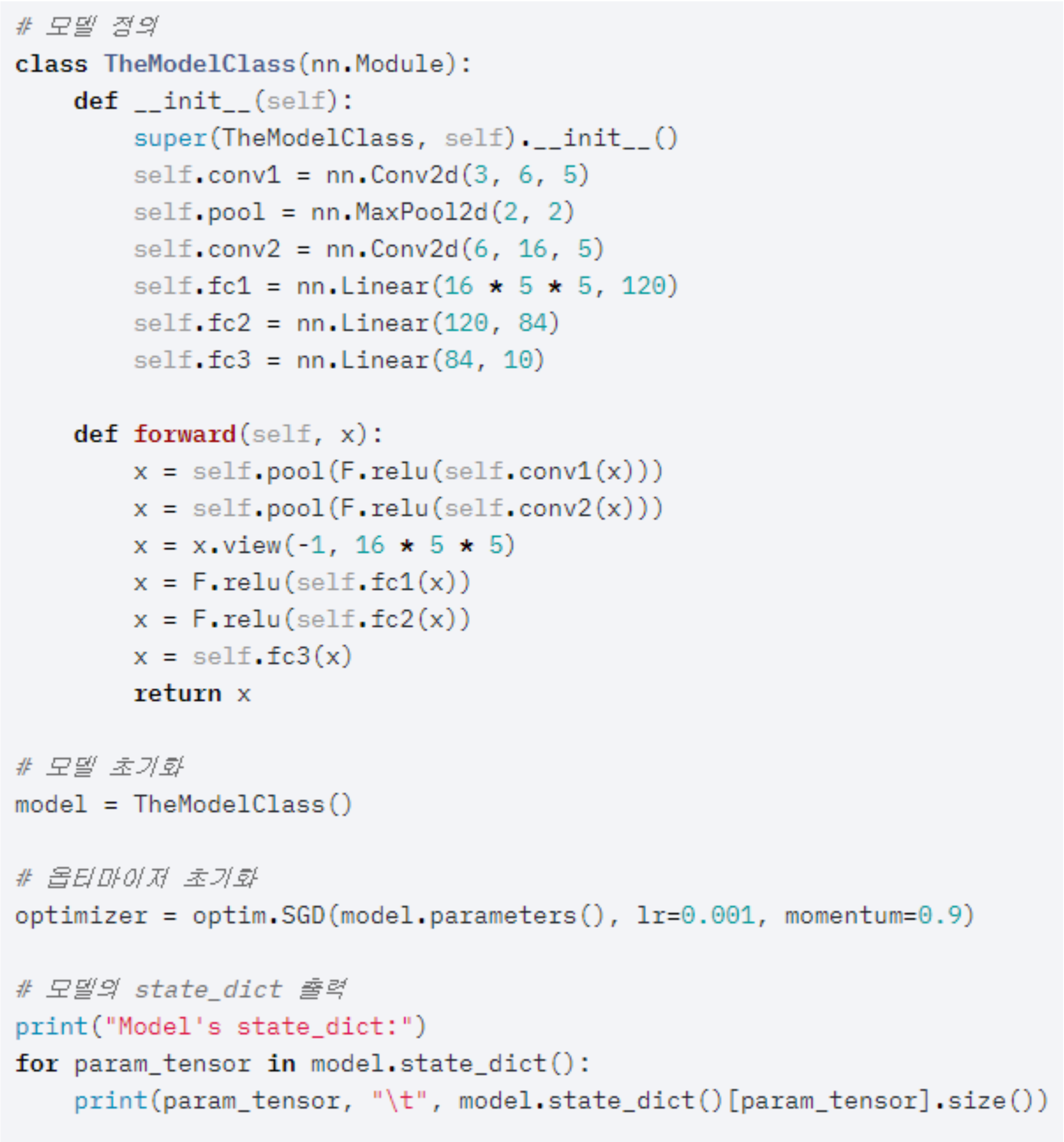
- PyTorch
    - Save and load models
    - Tensorboard
- 프로젝트 설명
- Review DQN
- Implementing a DQN

Save and Load Models

State_dict



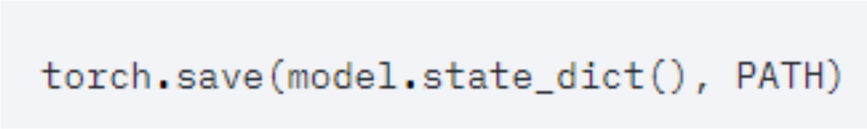
```
Model's state_dict:
conv1.weight     torch.Size([6, 3, 5, 5])
conv1.bias   torch.Size([6])
conv2.weight     torch.Size([16, 6, 5, 5])
conv2.bias   torch.Size([16])
fc1.weight   torch.Size([120, 400])
fc1.bias     torch.Size([120])
fc2.weight   torch.Size([84, 120])
fc2.bias     torch.Size([84])
fc3.weight   torch.Size([10, 84])
fc3.bias     torch.Size([10])
```



```
# 모델 정의
class TheModelClass(nn.Module):
    def __init__(self):
        super(TheModelClass, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# 모델 초기화
model = TheModelClass()

# 옵티마이저 초기화
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# 모델의 state_dict 출력
print("Model's state_dict:")
for param_tensor in model.state_dict():
    print(param_tensor, "\t", model.state_dict()[param_tensor].size())
```

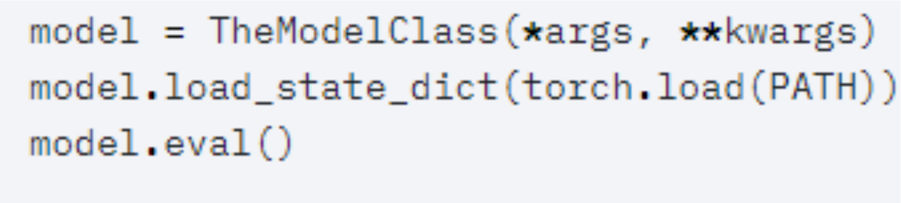
Save and Load Models

- Save



```
torch.save(model.state_dict(), PATH)
```

- Load

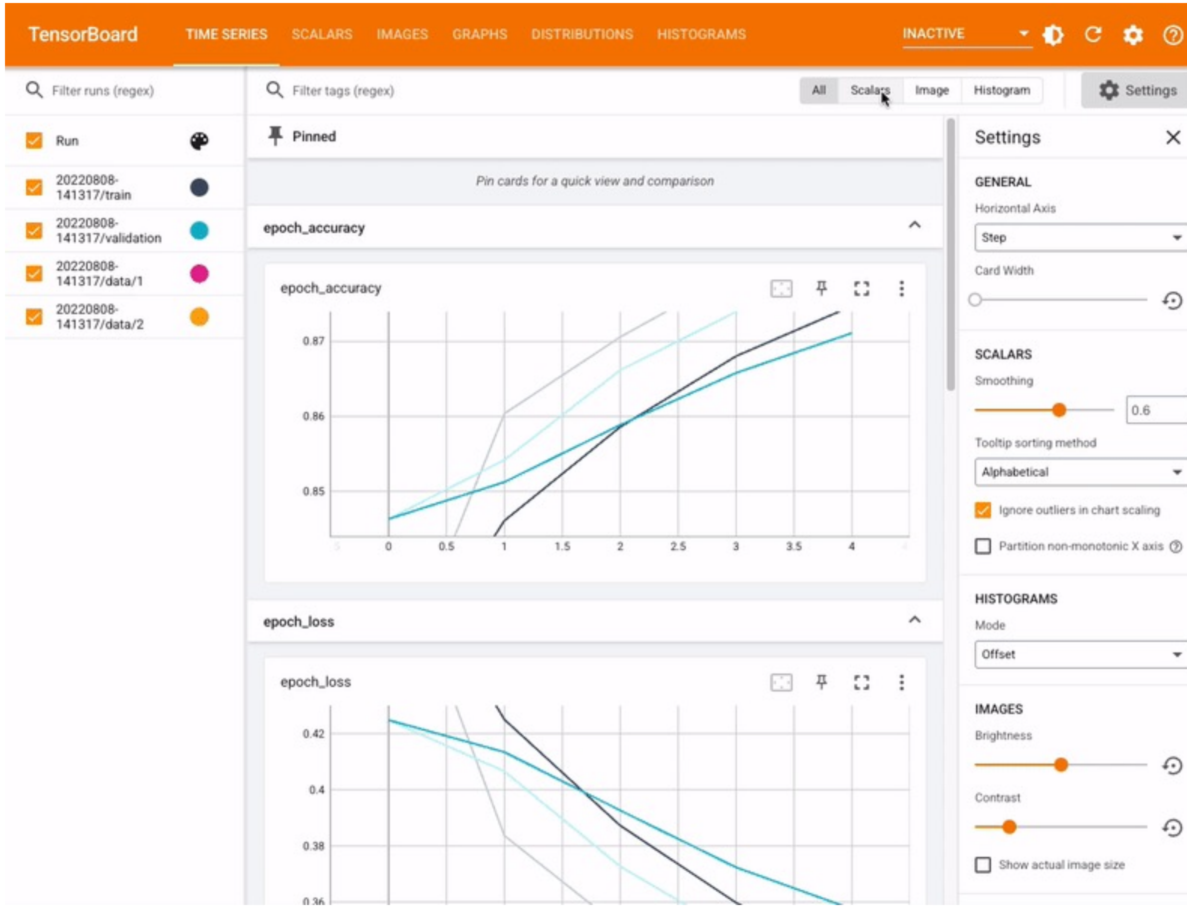


```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

Tensorboard

TensorBoard: TensorFlow의 시각화 툴킷

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the computation graph
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data



☑ Run
☑ 20220808-141317/train
☑ 20220808-141317/validation
☑ 20220808-141317/data/1
☑ 20220808-141317/data/2

Settings

GENERAL

Horizontal Axis: Step

Card Width

SCALARS

Smoothing: 0.6

Tooltip sorting method: Alphabetical

☑ Ignore outliers in chart scaling

□ Partition non-monotonic X axis

HISTOGRAMS

Mode: Offset

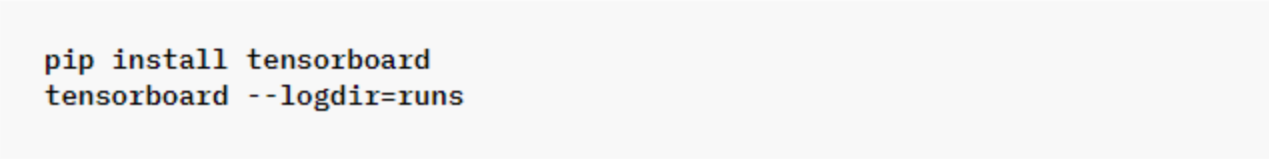
IMAGES

Brightness

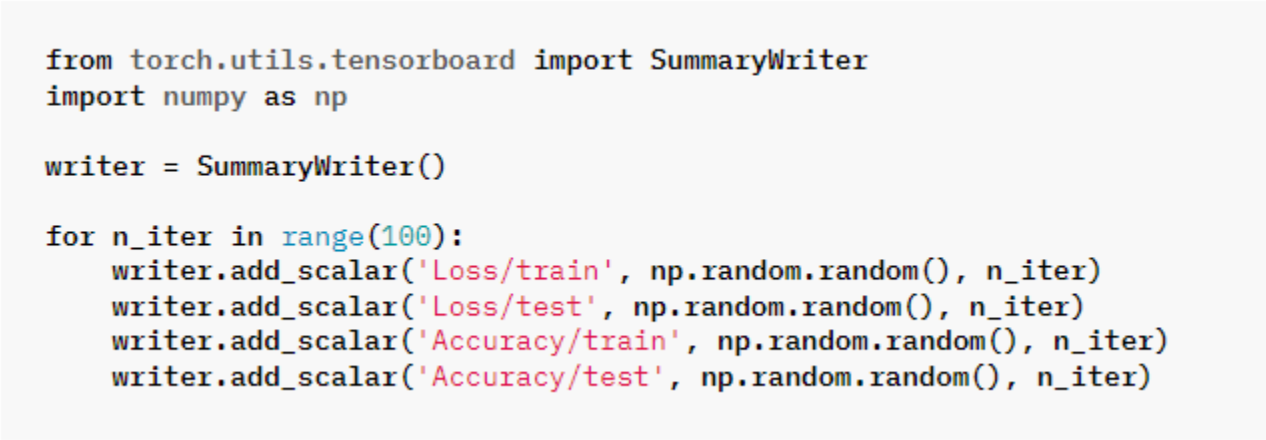
Contrast

□ Show actual image size

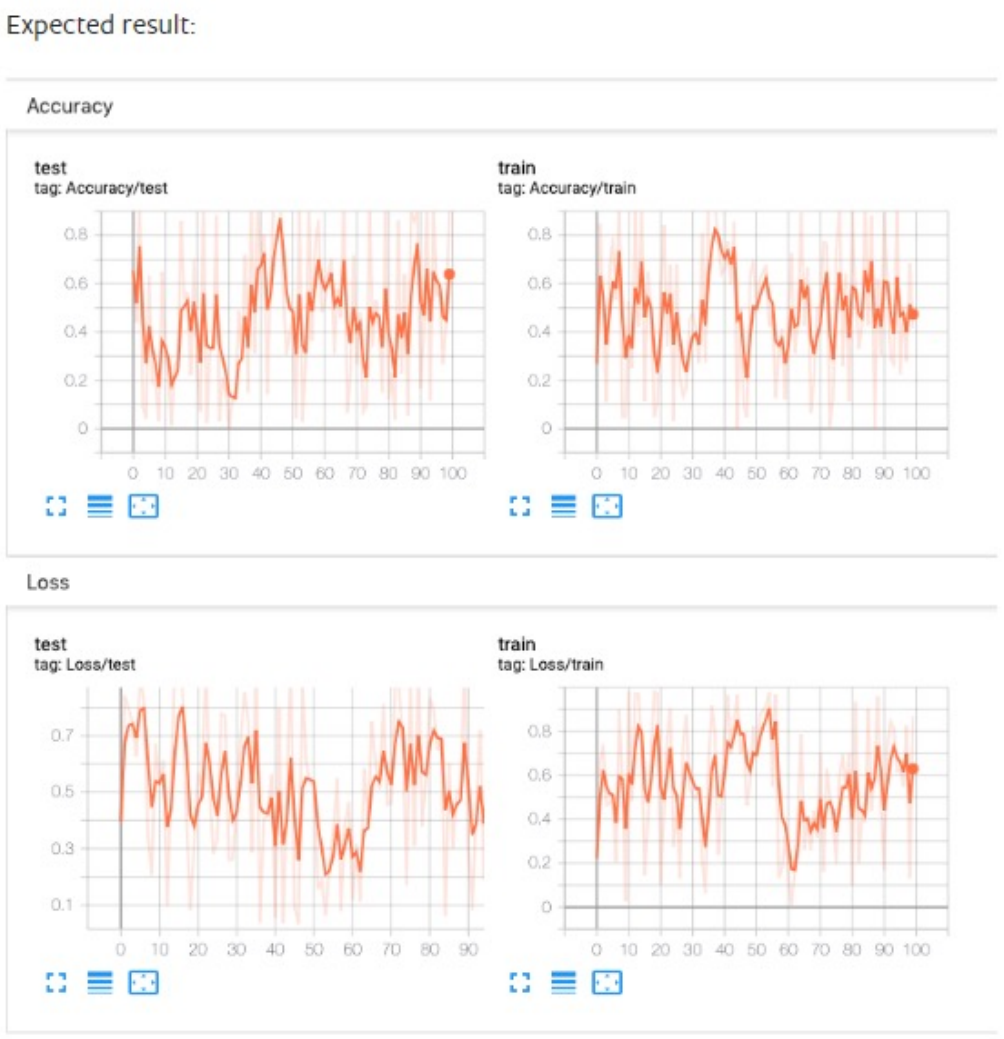
Tensorboard



```
pip install tensorboard
tensorboard --logdir=runs
```



```
from torch.utils.tensorboard import SummaryWriter
import numpy as np

writer = SummaryWriter()

for n_iter in range(100):
    writer.add_scalar('Loss/train', np.random.random(), n_iter)
    writer.add_scalar('Loss/test', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/train', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/test', np.random.random(), n_iter)
```

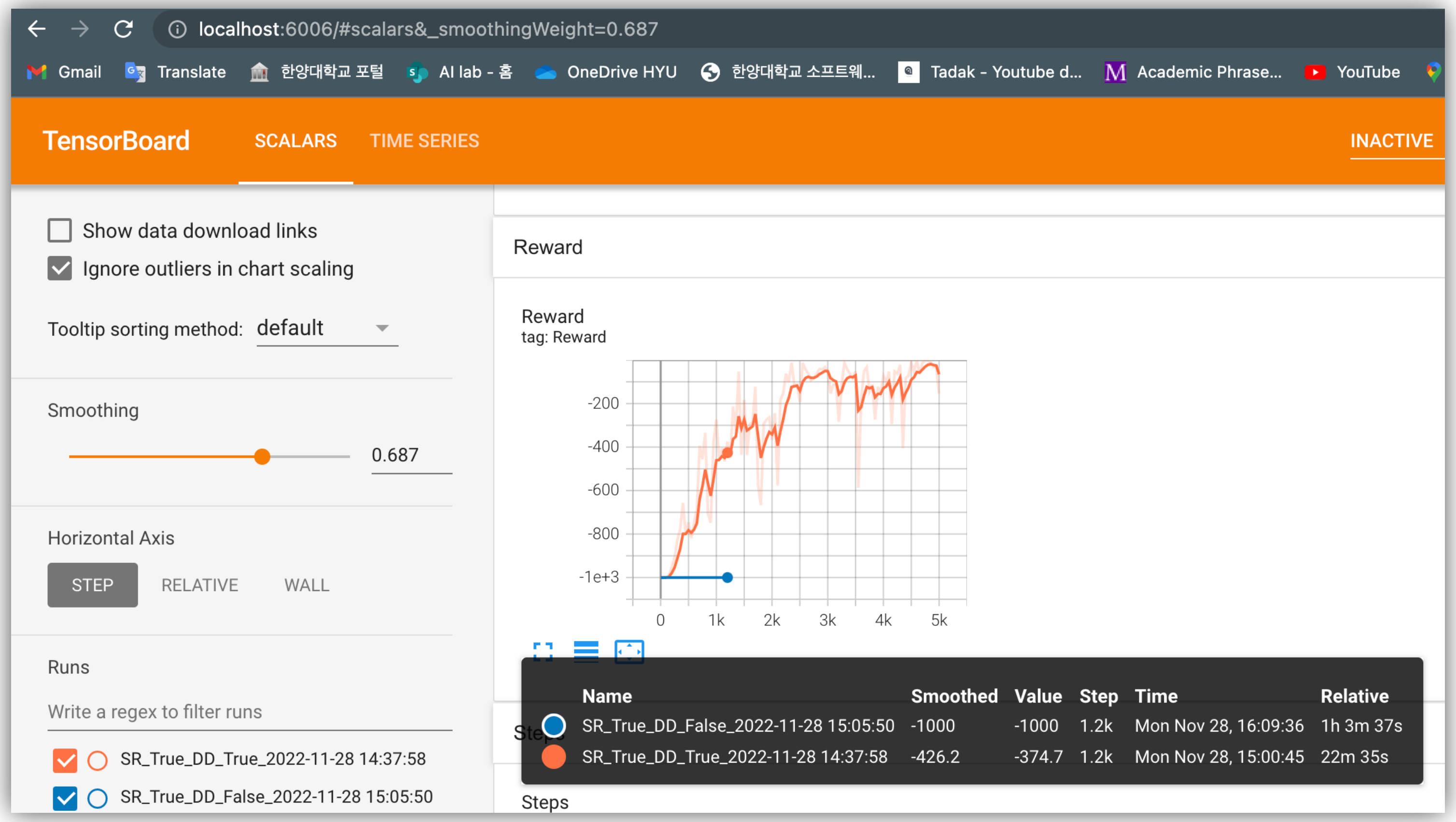


Expected result:

$ tensorboard –-logdir=runs

Tensorboard

```
$  tensorboard –-logdir=runs
```



□ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.687

Horizontal Axis: [✓ STEP] RELATIVE WALL

Runs

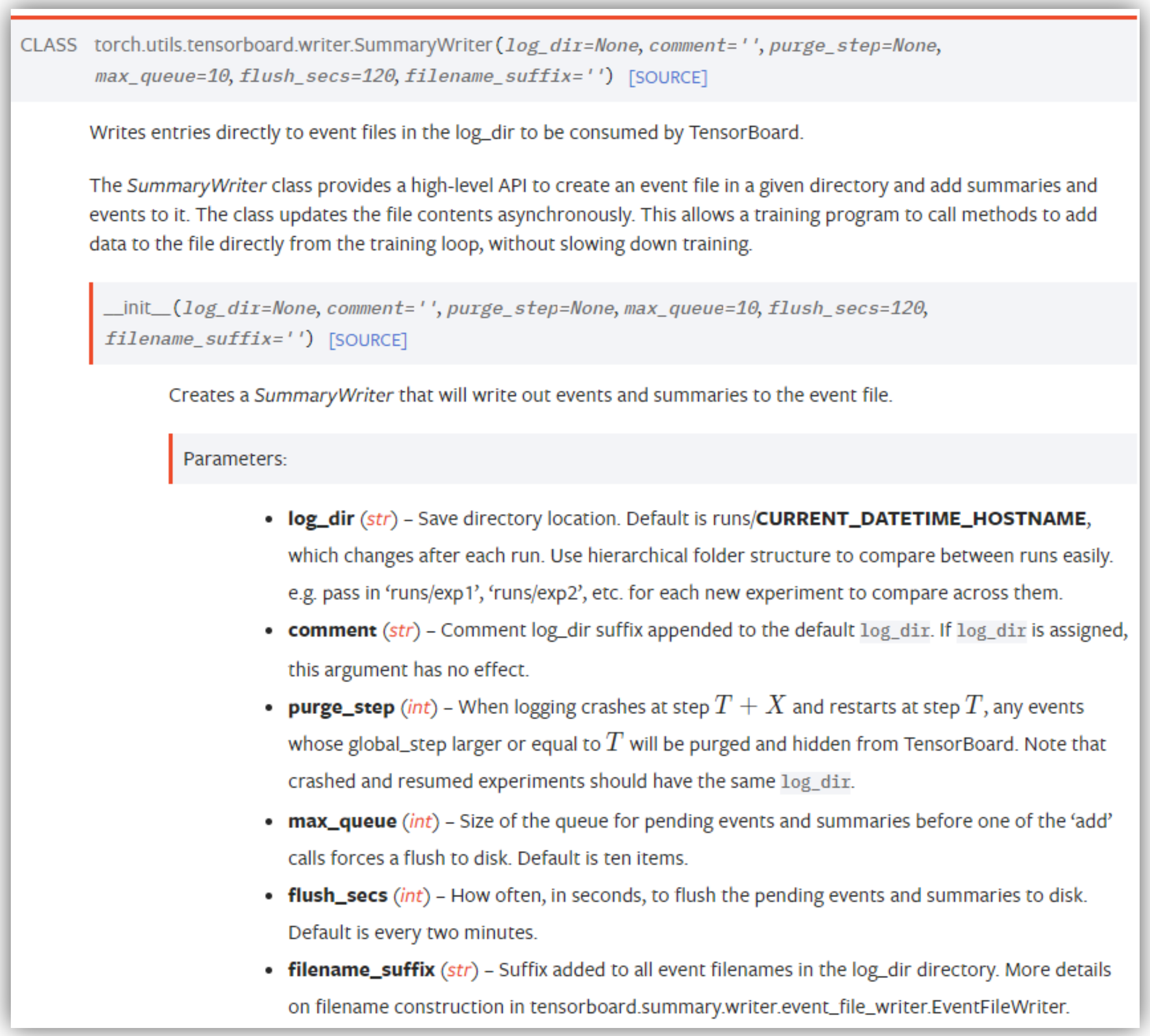
Write a regex to filter runs

☑ ○ SR_True_DD_True_2022-11-28 14:37:58

☑ ○ SR_True_DD_False_2022-11-28 15:05:50

	Name	Smoothed	Value	Step	Time	Relative
●	SR_True_DD_False_2022-11-28 15:05:50	-1000	-1000	1.2k	Mon Nov 28, 16:09:36	1h 3m 37s
●	SR_True_DD_True_2022-11-28 14:37:58	-426.2	-374.7	1.2k	Mon Nov 28, 15:00:45	22m 35s

SummaryWriter



CLASS torch.utils.tensorboard.writer.SummaryWriter(*log_dir=None*, *comment=''*, *purge_step=None*, *max_queue=10*, *flush_secs=120*, *filename_suffix=''*) [SOURCE]

Writes entries directly to event files in the log_dir to be consumed by TensorBoard.

The *SummaryWriter* class provides a high-level API to create an event file in a given directory and add summaries and events to it. The class updates the file contents asynchronously. This allows a training program to call methods to add data to the file directly from the training loop, without slowing down training.

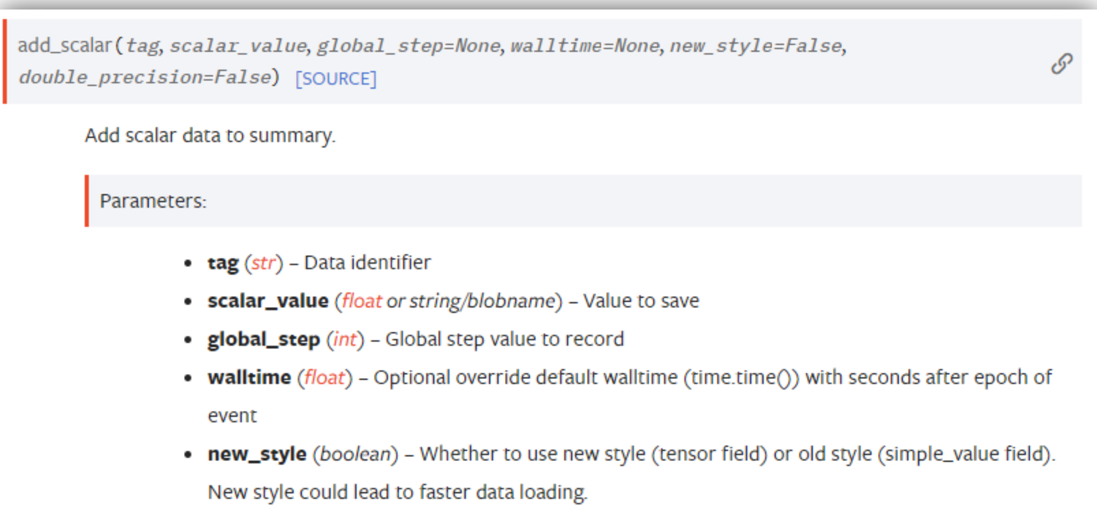
__init__(*log_dir=None*, *comment=''*, *purge_step=None*, *max_queue=10*, *flush_secs=120*, *filename_suffix=''*) [SOURCE]

Creates a *SummaryWriter* that will write out events and summaries to the event file.

Parameters:

- **log_dir** (*str*) – Save directory location. Default is runs/**CURRENT_DATETIME_HOSTNAME**, which changes after each run. Use hierarchical folder structure to compare between runs easily. e.g. pass in 'runs/exp1', 'runs/exp2', etc. for each new experiment to compare across them.
- **comment** (*str*) – Comment log_dir suffix appended to the default `log_dir`. If `log_dir` is assigned, this argument has no effect.
- **purge_step** (*int*) – When logging crashes at step $T + X$ and restarts at step T, any events whose global_step larger or equal to T will be purged and hidden from TensorBoard. Note that crashed and resumed experiments should have the same `log_dir`.
- **max_queue** (*int*) – Size of the queue for pending events and summaries before one of the 'add' calls forces a flush to disk. Default is ten items.
- **flush_secs** (*int*) – How often, in seconds, to flush the pending events and summaries to disk. Default is every two minutes.
- **filename_suffix** (*str*) – Suffix added to all event filenames in the log_dir directory. More details on filename construction in tensorboard.summary.writer.event_file_writer.EventFileWriter.

Add scalar data to summary

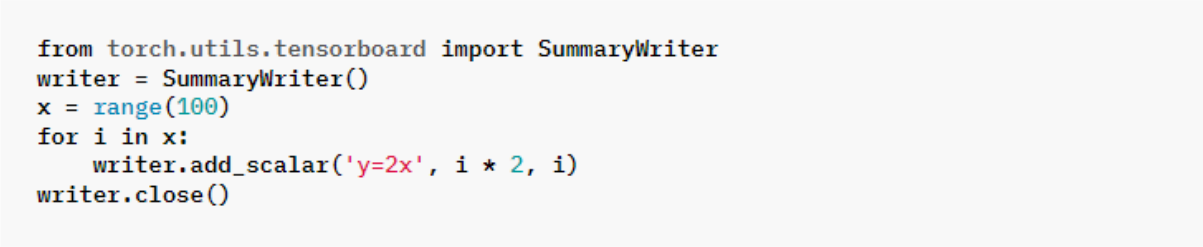


add_scalar(*tag*, *scalar_value*, *global_step=None*, *walltime=None*, *new_style=False*, *double_precision=False*) [SOURCE]

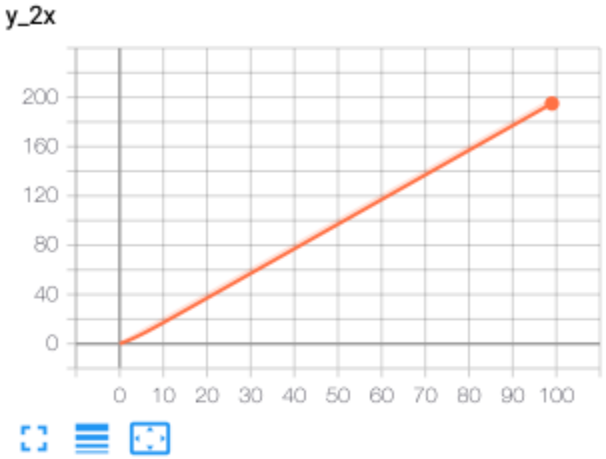
Add scalar data to summary.

Parameters:

- **tag** (*str*) – Data identifier
- **scalar_value** (*float or string/blobname*) – Value to save
- **global_step** (*int*) – Global step value to record
- **walltime** (*float*) – Optional override default walltime (time.time()) with seconds after epoch of event
- **new_style** (*boolean*) – Whether to use new style (tensor field) or old style (simple_value field). New style could lead to faster data loading.

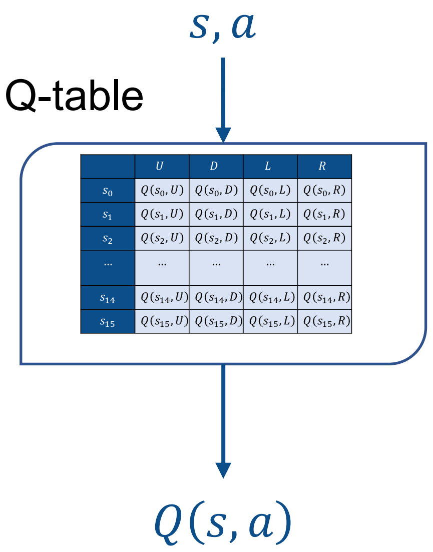


```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
x = range(100)
for i in x:
    writer.add_scalar('y=2x', i * 2, i)
writer.close()
```



Review

Q-learning (Table vs NN)



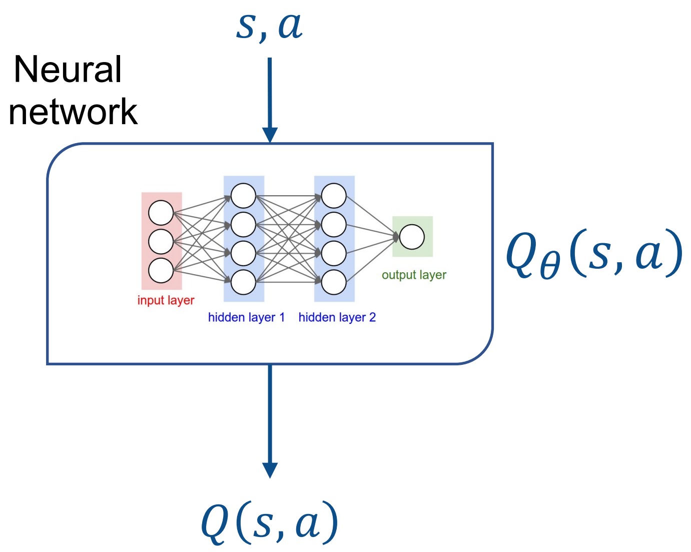
s, a

Q-table

	U	D	L	R
s_0	$Q(s_0, U)$	$Q(s_0, D)$	$Q(s_0, L)$	$Q(s_0, R)$
s_1	$Q(s_1, U)$	$Q(s_1, D)$	$Q(s_1, L)$	$Q(s_1, R)$
s_2	$Q(s_2, U)$	$Q(s_2, D)$	$Q(s_2, L)$	$Q(s_2, R)$
…	…	…	…	…
s_{14}	$Q(s_{14}, U)$	$Q(s_{14}, D)$	$Q(s_{14}, L)$	$Q(s_{14}, R)$
s_{15}	$Q(s_{15}, U)$	$Q(s_{15}, D)$	$Q(s_{15}, L)$	$Q(s_{15}, R)$

$Q(s, a)$

Q-learning (Table)



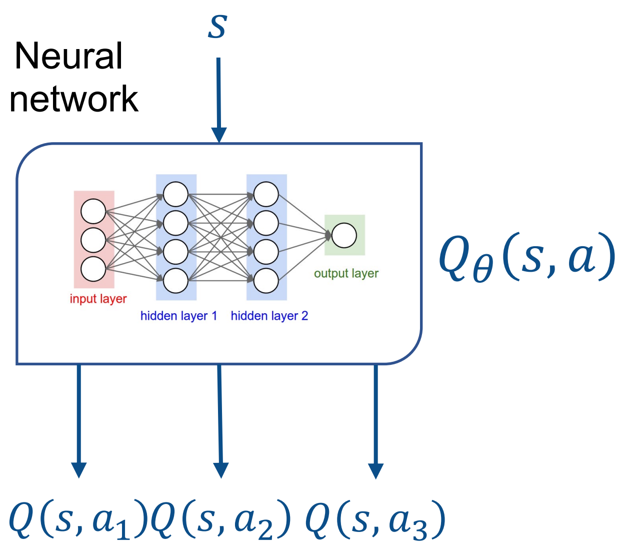
s, a

Neural network

$Q_\theta(s, a)$

$Q(s, a)$

DQN (action-in)



s

Neural network

$Q_\theta(s, a)$

$Q(s, a_1) Q(s, a_2)\ Q(s, a_3)$

DQN (action-out)

Naïve DQN

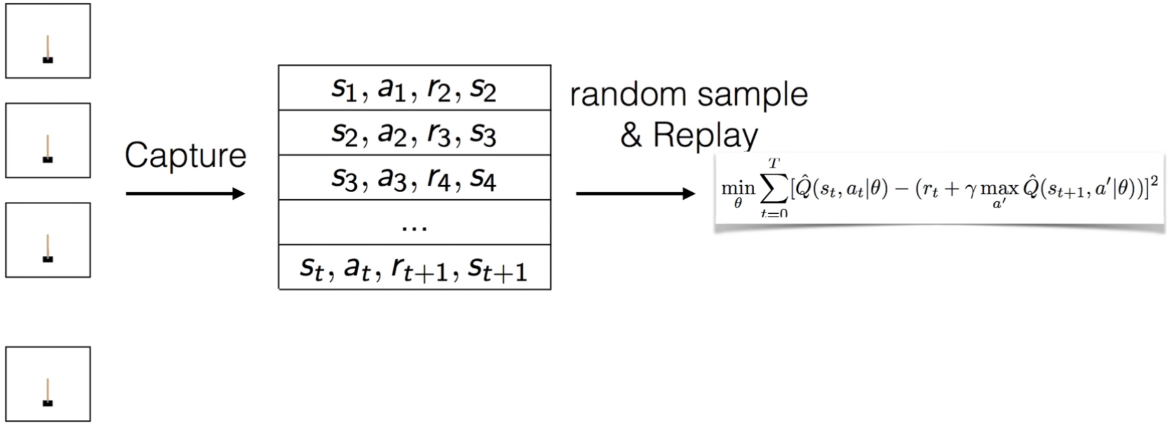
Model: $Q_\theta(s_t, a_t)$

Training data: $\langle s_t, a_t, r_t, s_{t+1} \rangle$

Loss function: $\mathcal{L}(\theta) = ||y_t - Q_\theta(s_t, a_t)||_2^2$ where $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$

Problems & Solutions

- Naïve Q-learning: Q-table를 Neural Network로 대체
  - Non-stationary target, Correlated samples문제가 있음
- Solutions
  - Capture and replay



s_1, a_1, r_2, s_2

s_2, a_2, r_3, s_3

s_3, a_3, r_4, s_4

$\dots$

$s_t, a_t, r_{t+1}, s_{t+1}$

$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

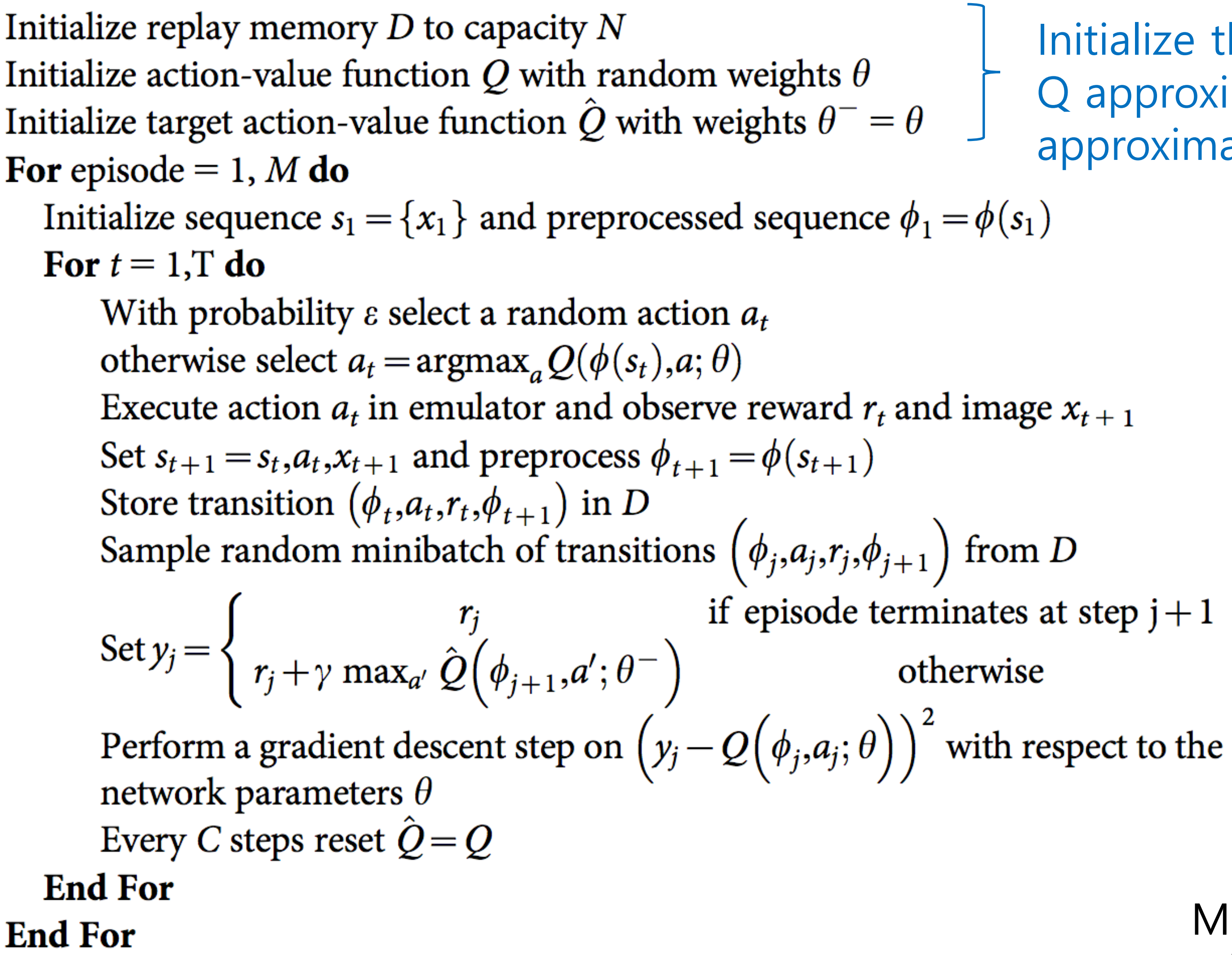
  - Separate target network

$$\text{minimize } L(\theta) = \left\| \left(r + \gamma \max_{a\prime} Q_{\theta}(s', a') \right) - Q_{\theta}(s, a) \right\|_2^2$$

$$\Downarrow$$

$$\text{minimize } L(\theta) = \left\| \left(r + \gamma \max_{a\prime} Q_{\theta'}(s', a') \right) - Q_{\theta}(s, a) \right\|_2^2$$

Q-learning with experience replay



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ select a random action $a_t$
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**
```

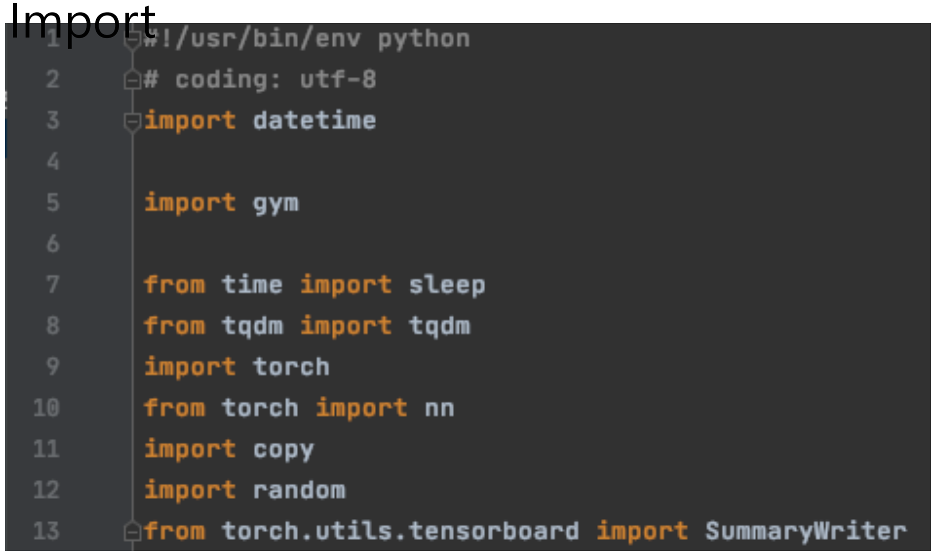
Initialize the replay memory and two identical Q approximators (DNN). $\hat{Q}$ is our target approximator.

Mnih et al., Human-level control through deep reinforcement learning, Nature 2015

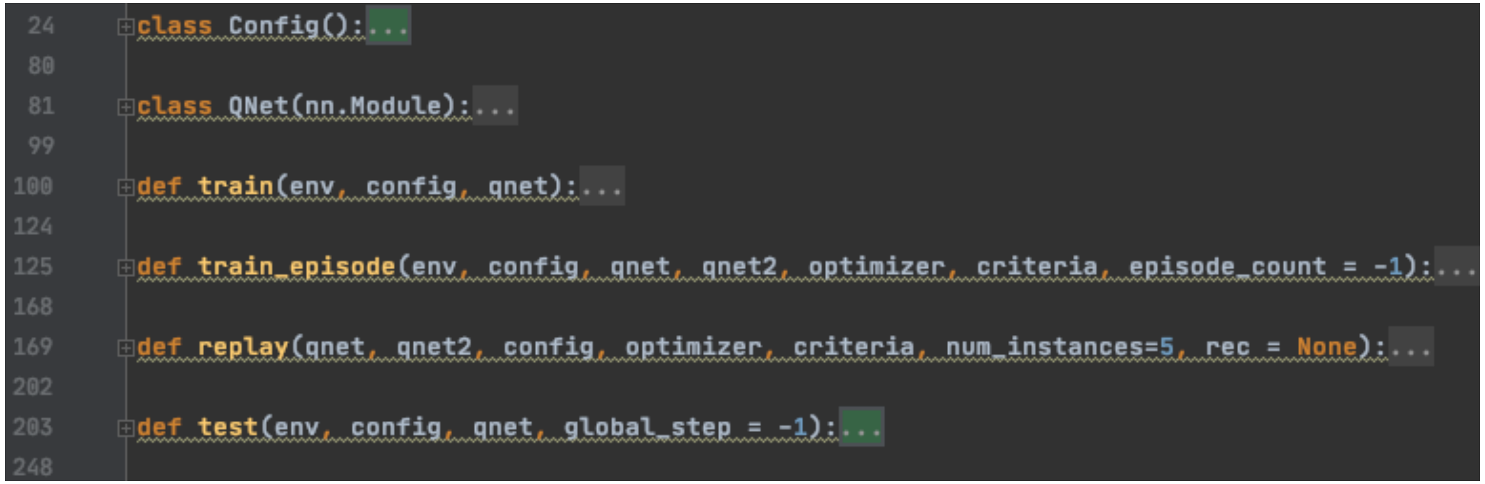
Deep Q-net 구현

Taxi

Import



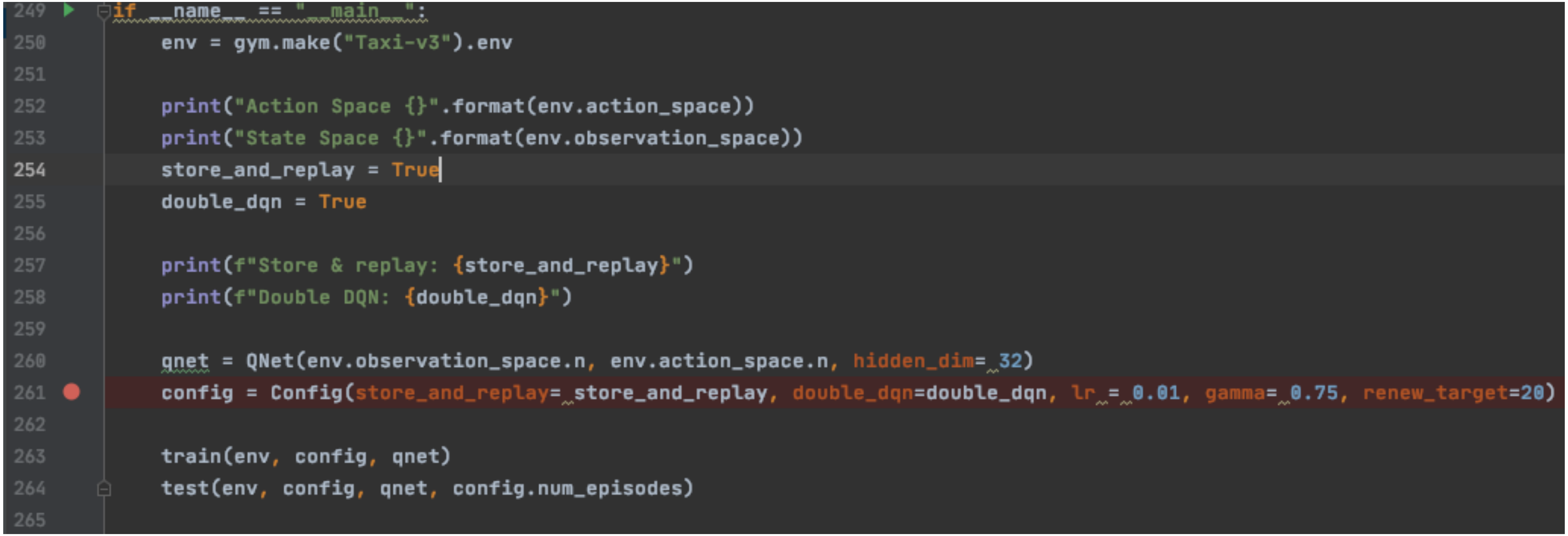
```python
#!/usr/bin/env python
# coding: utf-8
import datetime

import gym

from time import sleep
from tqdm import tqdm
import torch
from torch import nn
import copy
import random
from torch.utils.tensorboard import SummaryWriter
```



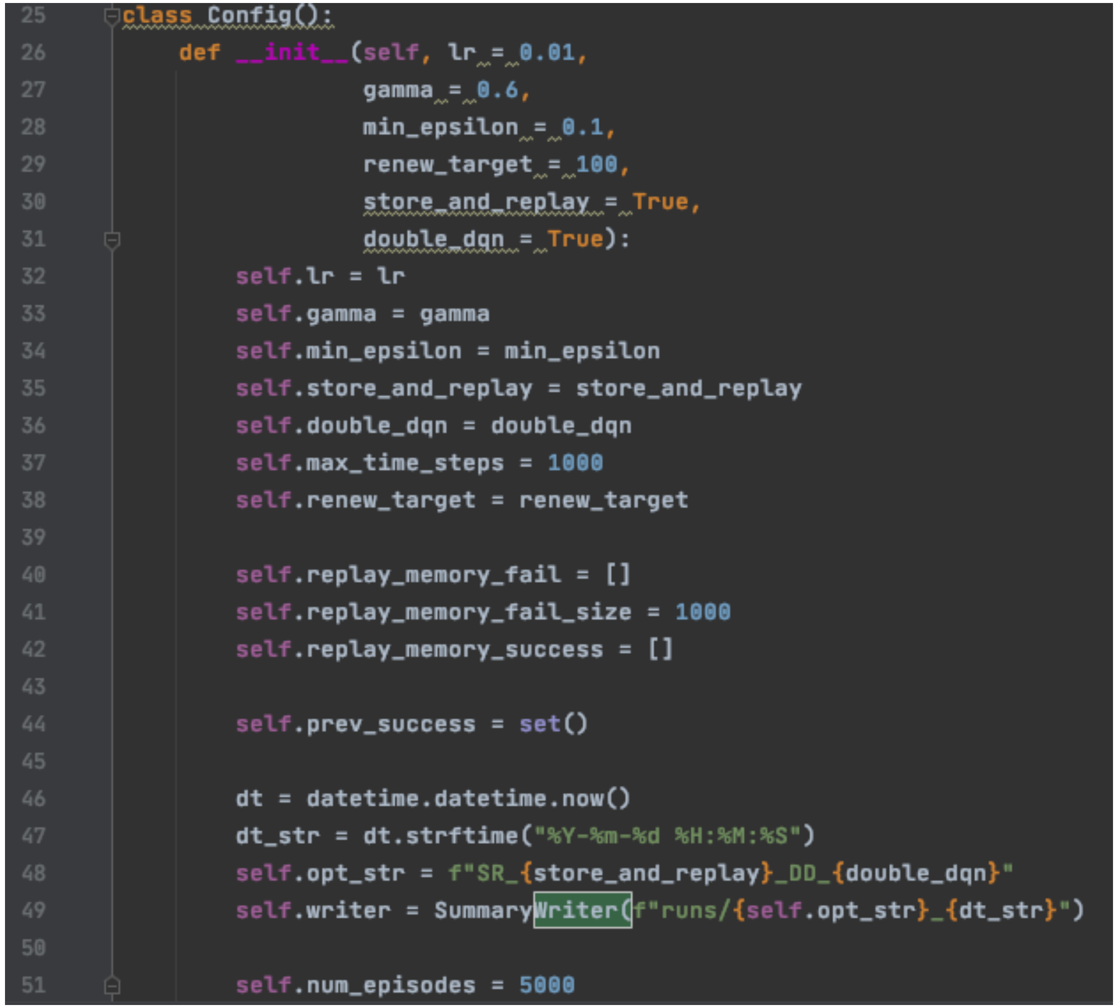
```python
class Config():...

class QNet(nn.Module):...

def train(env, config, qnet):...

def train_episode(env, config, qnet, qnet2, optimizer, criteria, episode_count = -1):...

def replay(qnet, qnet2, config, optimizer, criteria, num_instances=5, rec = None):...

def test(env, config, qnet, global_step = -1):...
```

- class Config()
  - 학습에 필요한 각종 세팅, replay memory 등을 저장 및 관리
- class QNet(nn.Module)
  - DQN class
- def train(env, config, qnet):
  - 모델 학습 (전체 에피소드)
- def train_episode(env, config, qnet, …):
  - 모델 학습 (에피소드 1개)
- def replay(qnet, qnet2, config, …):
  - 모델 학습 (에피소드 1개)
- def test(env, config, qnet,…):
  - 테스트

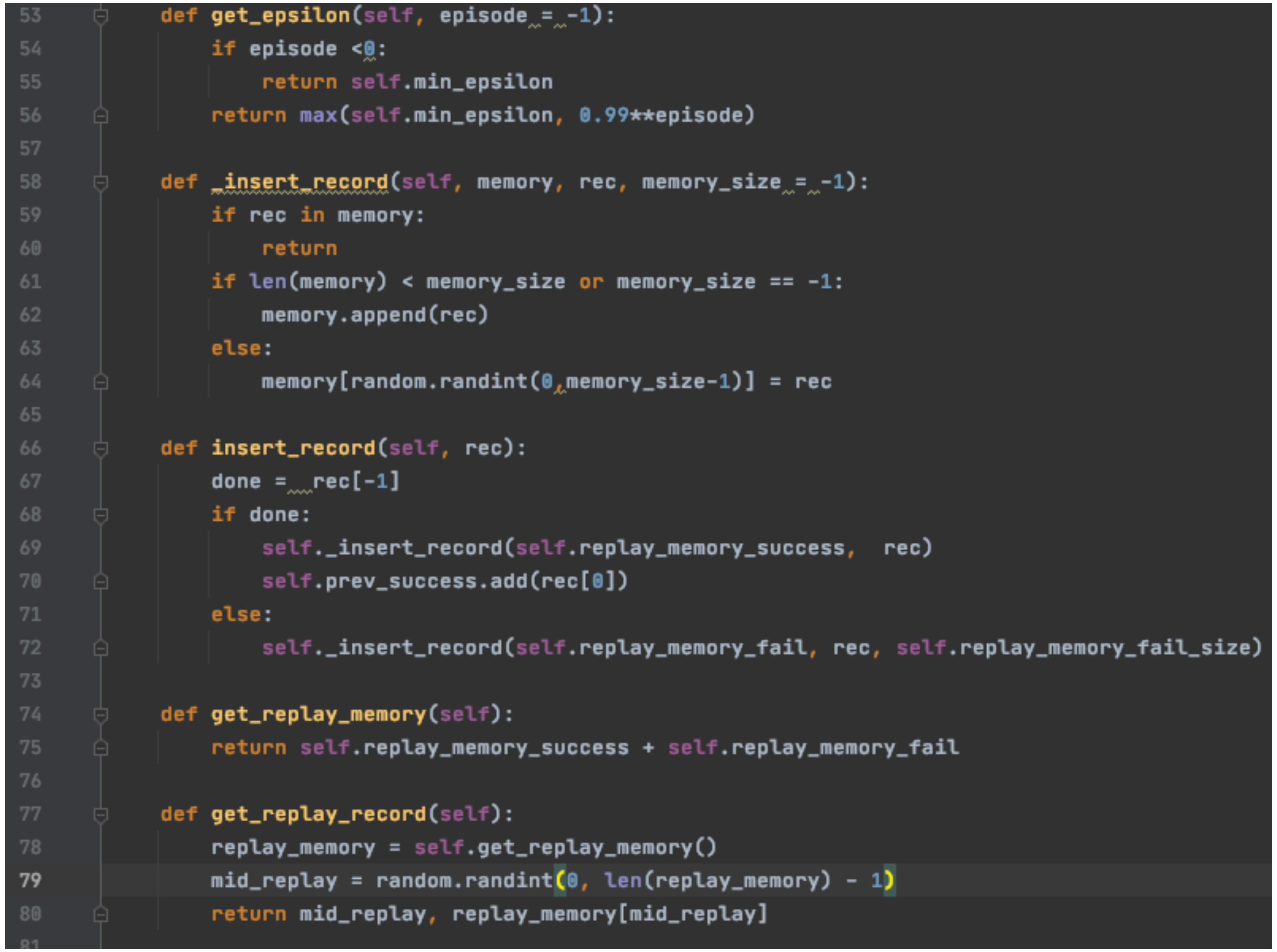
Main



```python
if __name__ == "__main__":
    env = gym.make("Taxi-v3").env

    print("Action Space {}".format(env.action_space))
    print("State Space {}".format(env.observation_space))
    store_and_replay = True
    double_dqn = True

    print(f"Store & replay: {store_and_replay}")
    print(f"Double DQN: {double_dqn}")

    qnet = QNet(env.observation_space.n, env.action_space.n, hidden_dim= 32)
    config = Config(store_and_replay= store_and_replay, double_dqn=double_dqn, lr = 0.01, gamma= 0.75, renew_target=20)

    train(env, config, qnet)
    test(env, config, qnet, config.num_episodes)

```



```python
class Config():
    def __init__(self, lr = 0.01,
                 gamma = 0.6,
                 min_epsilon = 0.1,
                 renew_target = 100,
                 store_and_replay = True,
                 double_dqn = True):
        self.lr = lr
        self.gamma = gamma
        self.min_epsilon = min_epsilon
        self.store_and_replay = store_and_replay
        self.double_dqn = double_dqn
        self.max_time_steps = 1000
        self.renew_target = renew_target

        self.replay_memory_fail = []
        self.replay_memory_fail_size = 1000
        self.replay_memory_success = []

        self.prev_success = set()

        dt = datetime.datetime.now()
        dt_str = dt.strftime("%Y-%m-%d %H:%M:%S")
        self.opt_str = f"SR_{store_and_replay}_DD_{double_dqn}"
        self.writer = SummaryWriter(f"runs/{self.opt_str}_{dt_str}")

        self.num_episodes = 5000
```



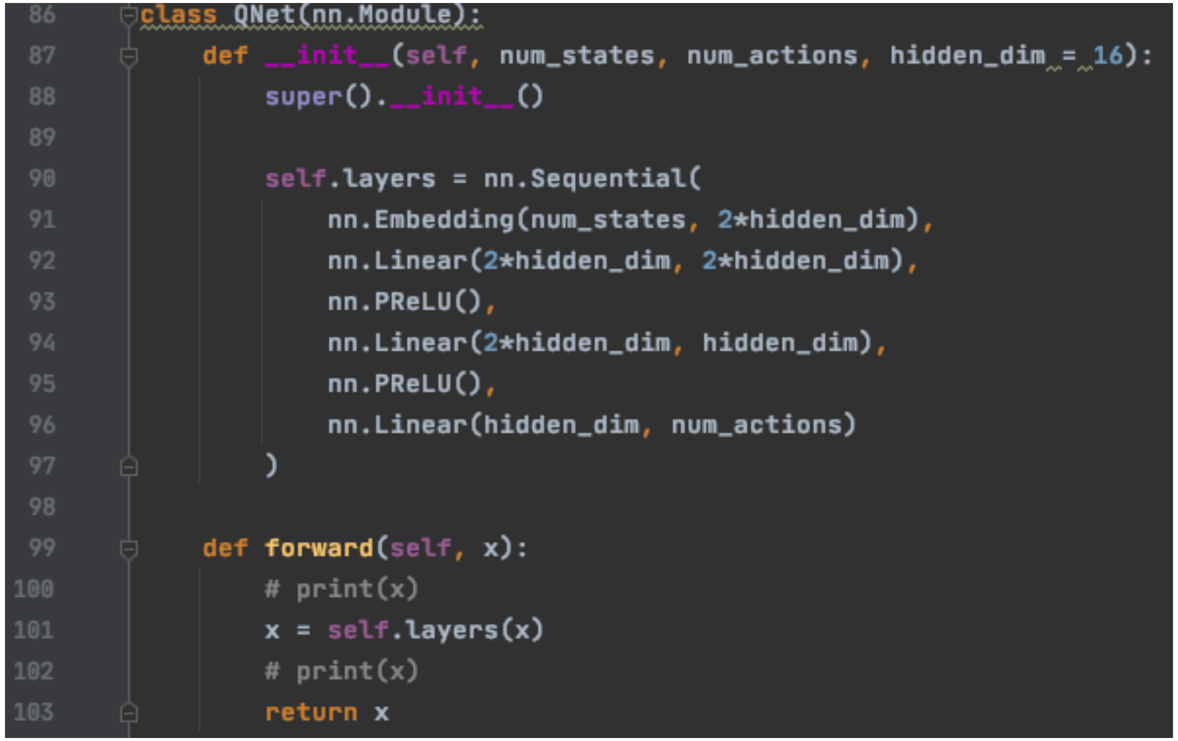
```python
    def get_epsilon(self, episode = -1):
        if episode <0:
            return self.min_epsilon
        return max(self.min_epsilon, 0.99**episode)

    def _insert_record(self, memory, rec, memory_size = -1):
        if rec in memory:
            return
        if len(memory) < memory_size or memory_size == -1:
            memory.append(rec)
        else:
            memory[random.randint(0,memory_size-1)] = rec

    def insert_record(self, rec):
        done =  rec[-1]
        if done:
            self._insert_record(self.replay_memory_success,  rec)
            self.prev_success.add(rec[0])
        else:
            self._insert_record(self.replay_memory_fail, rec, self.replay_memory_fail_size)

    def get_replay_memory(self):
        return self.replay_memory_success + self.replay_memory_fail

    def get_replay_record(self):
        replay_memory = self.get_replay_memory()
        mid_replay = random.randint(0, len(replay_memory) - 1)
        return mid_replay, replay_memory[mid_replay]
```

```python
class Config():
```

학습에 필요한 각종 세팅, replay memory 등을 저장 및 관리

class QNet(nn.Module)

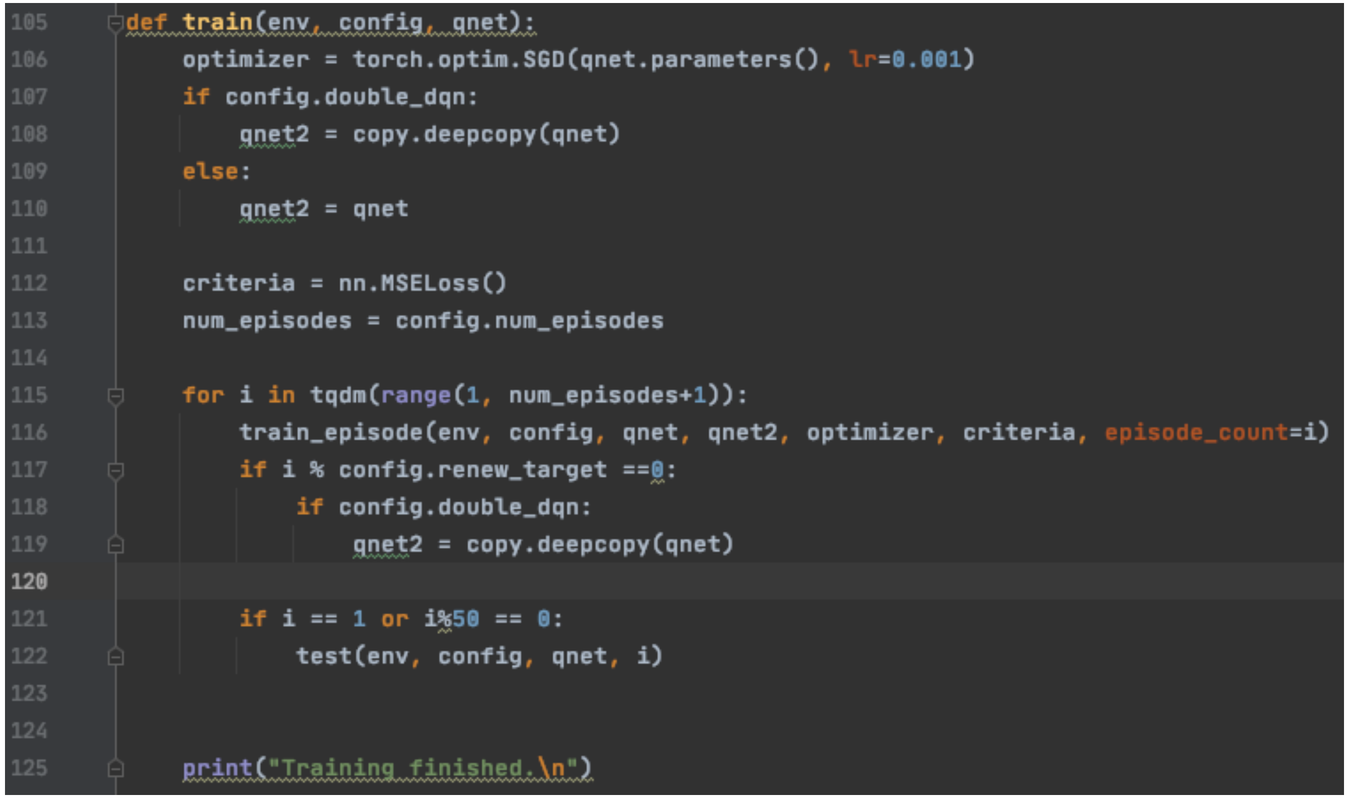
DQN class



```python
class QNet(nn.Module):
    def __init__(self, num_states, num_actions, hidden_dim = 16):
        super().__init__()

        self.layers = nn.Sequential(
            nn.Embedding(num_states, 2*hidden_dim),
            nn.Linear(2*hidden_dim, 2*hidden_dim),
            nn.PReLU(),
            nn.Linear(2*hidden_dim, hidden_dim),
            nn.PReLU(),
            nn.Linear(hidden_dim, num_actions)
        )

    def forward(self, x):
        # print(x)
        x = self.layers(x)
        # print(x)
        return x
```

def train(env, config, qnet):

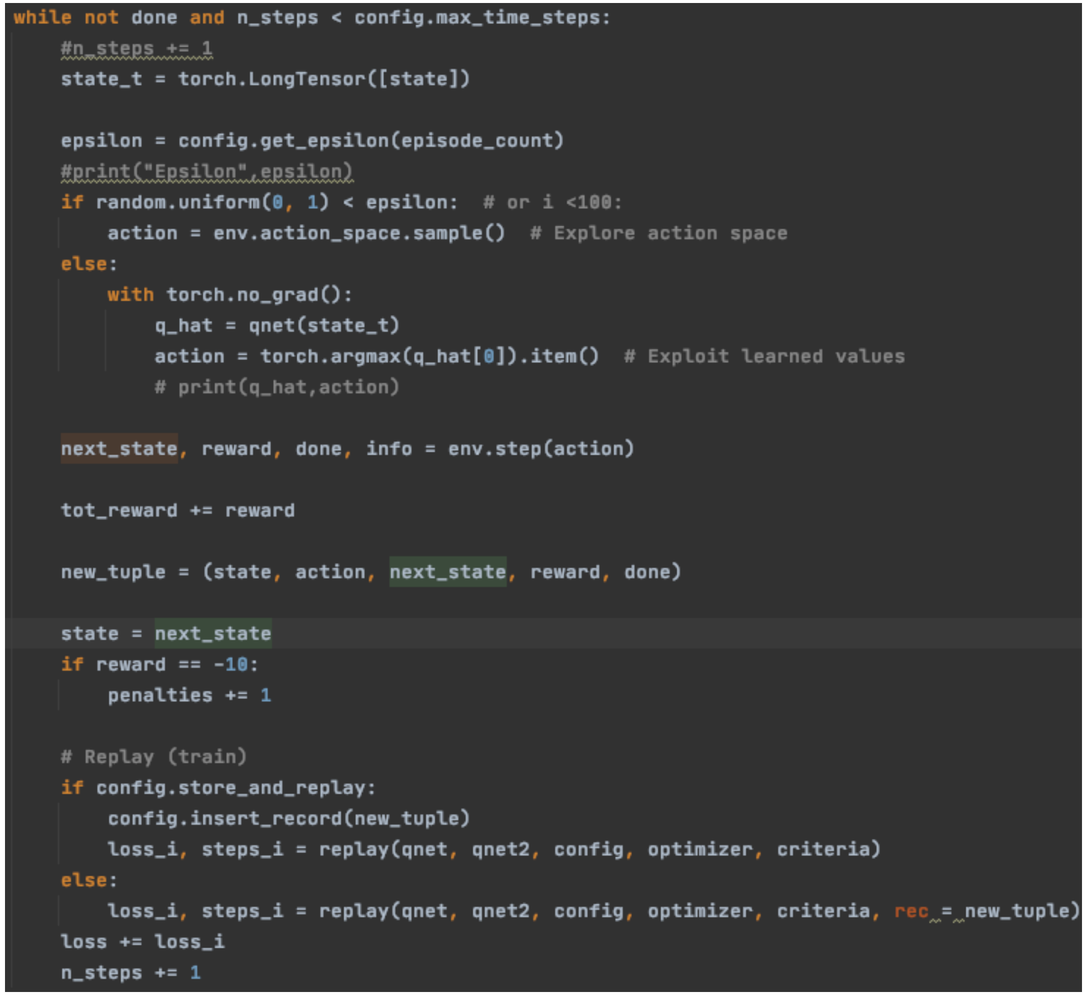
모델 학습 (전체 에피소드)



```python
def train(env, config, qnet):
    optimizer = torch.optim.SGD(qnet.parameters(), lr=0.001)
    if config.double_dqn:
        qnet2 = copy.deepcopy(qnet)
    else:
        qnet2 = qnet

    criteria = nn.MSELoss()
    num_episodes = config.num_episodes

    for i in tqdm(range(1, num_episodes+1)):
        train_episode(env, config, qnet, qnet2, optimizer, criteria, episode_count=i)
        if i % config.renew_target ==0:
            if config.double_dqn:
                qnet2 = copy.deepcopy(qnet)

        if i == 1 or i%50 == 0:
            test(env, config, qnet, i)


    print("Training finished.\n")
```

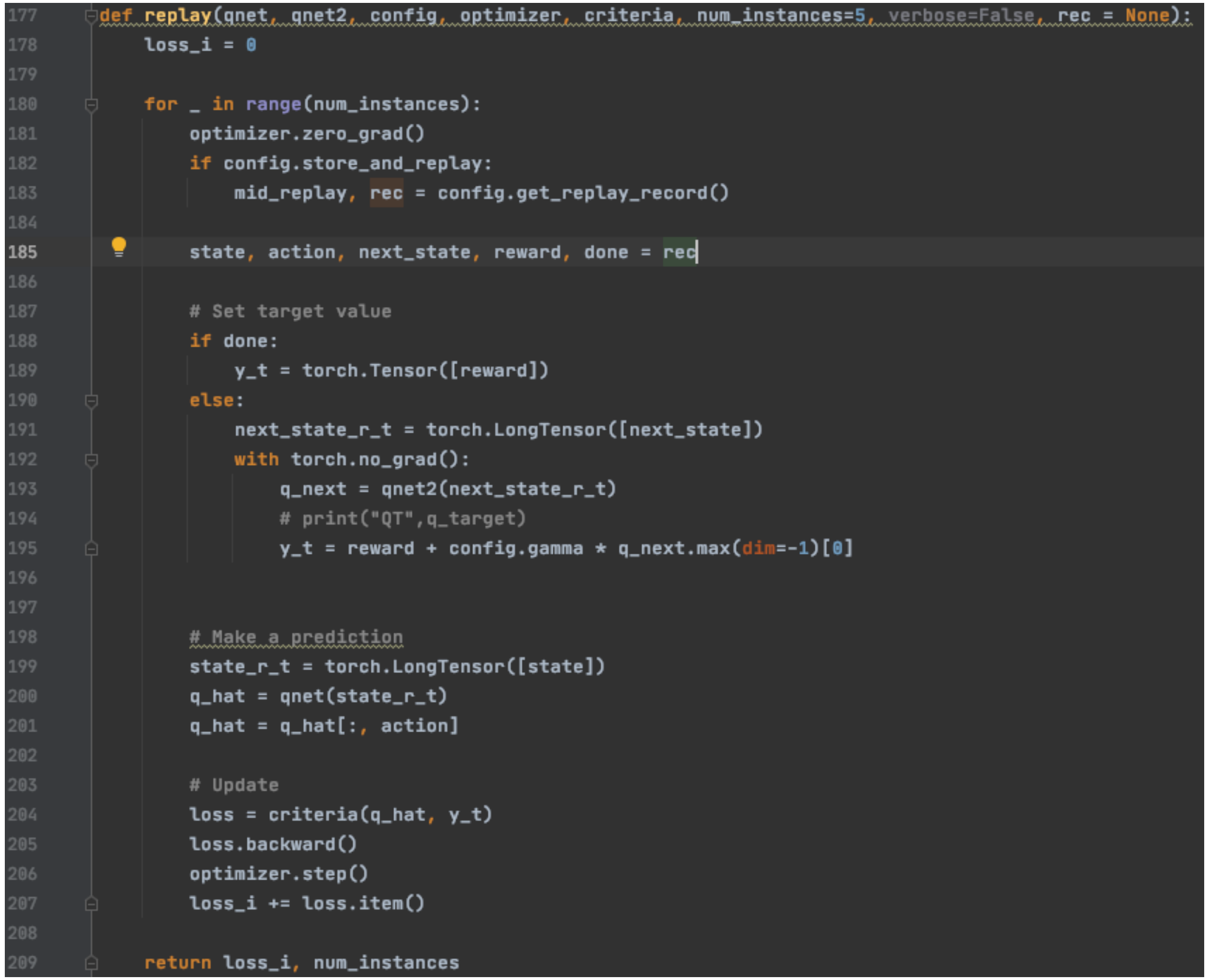
`def train_episode(env, config, qnet, …):`

모델 학습 (에피소드 1개)



```
while not done and n_steps < config.max_time_steps:
    #n_steps += 1
    state_t = torch.LongTensor([state])

    epsilon = config.get_epsilon(episode_count)
    #print("Epsilon",epsilon)
    if random.uniform(0, 1) < epsilon:  # or i <100:
        action = env.action_space.sample()  # Explore action space
    else:
        with torch.no_grad():
            q_hat = qnet(state_t)
            action = torch.argmax(q_hat[0]).item()  # Exploit learned values
            # print(q_hat,action)

    next_state, reward, done, info = env.step(action)

    tot_reward += reward

    new_tuple = (state, action, next_state, reward, done)

    state = next_state
    if reward == -10:
        penalties += 1

    # Replay (train)
    if config.store_and_replay:
        config.insert_record(new_tuple)
        loss_i, steps_i = replay(qnet, qnet2, config, optimizer, criteria)
    else:
        loss_i, steps_i = replay(qnet, qnet2, config, optimizer, criteria, rec = new_tuple)
    loss += loss_i
    n_steps += 1
```

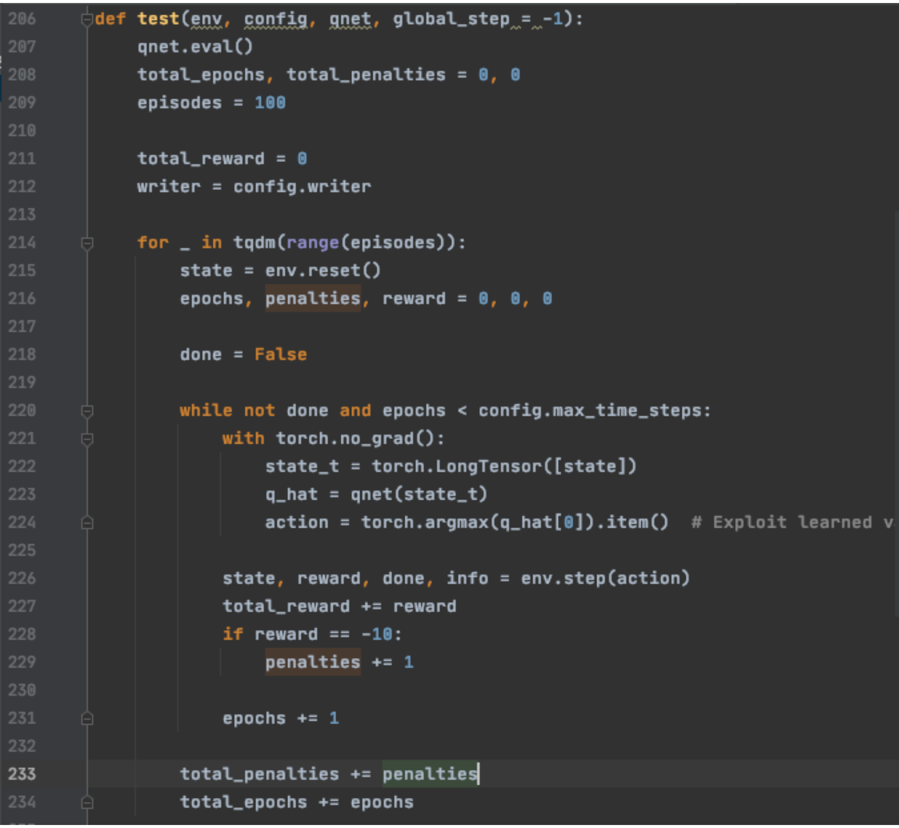
def replay(qnet, qnet2, config, …):
모델 학습 (에피소드 1개)



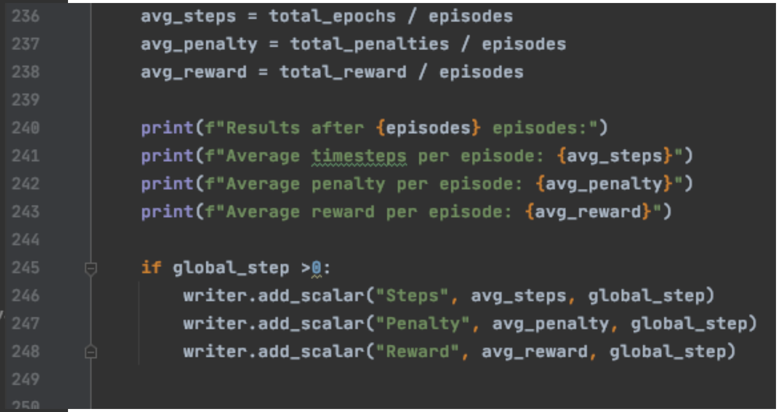
```python
def replay(qnet, qnet2, config, optimizer, criteria, num_instances=5, verbose=False, rec = None):
    loss_i = 0

    for _ in range(num_instances):
        optimizer.zero_grad()
        if config.store_and_replay:
            mid_replay, rec = config.get_replay_record()

        state, action, next_state, reward, done = rec

        # Set target value
        if done:
            y_t = torch.Tensor([reward])
        else:
            next_state_r_t = torch.LongTensor([next_state])
            with torch.no_grad():
                q_next = qnet2(next_state_r_t)
                # print("QT",q_target)
                y_t = reward + config.gamma * q_next.max(dim=-1)[0]


        # Make a prediction
        state_r_t = torch.LongTensor([state])
        q_hat = qnet(state_r_t)
        q_hat = q_hat[:, action]

        # Update
        loss = criteria(q_hat, y_t)
        loss.backward()
        optimizer.step()
        loss_i += loss.item()

    return loss_i, num_instances
```

def test(env, config, qnet,…):

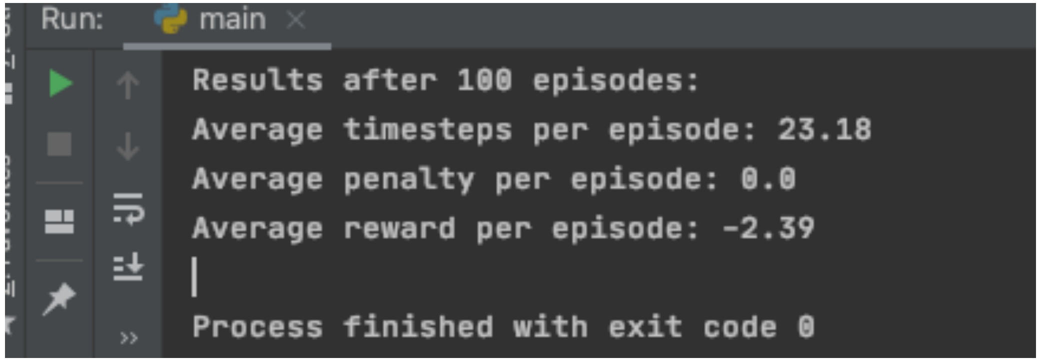
테스트



```
def test(env, config, qnet, global_step = -1):
    qnet.eval()
    total_epochs, total_penalties = 0, 0
    episodes = 100

    total_reward = 0
    writer = config.writer

    for _ in tqdm(range(episodes)):
        state = env.reset()
        epochs, penalties, reward = 0, 0, 0

        done = False

        while not done and epochs < config.max_time_steps:
            with torch.no_grad():
                state_t = torch.LongTensor([state])
                q_hat = qnet(state_t)
                action = torch.argmax(q_hat[0]).item()  # Exploit learned v

            state, reward, done, info = env.step(action)
            total_reward += reward
            if reward == -10:
                penalties += 1

            epochs += 1

        total_penalties += penalties
        total_epochs += epochs
```



```
    avg_steps = total_epochs / episodes
    avg_penalty = total_penalties / episodes
    avg_reward = total_reward / episodes

    print(f"Results after {episodes} episodes:")
    print(f"Average timesteps per episode: {avg_steps}")
    print(f"Average penalty per episode: {avg_penalty}")
    print(f"Average reward per episode: {avg_reward}")

    if global_step >0:
        writer.add_scalar("Steps", avg_steps, global_step)
        writer.add_scalar("Penalty", avg_penalty, global_step)
        writer.add_scalar("Reward", avg_reward, global_step)
```



```
Results after 100 episodes:
Average timesteps per episode: 23.18
Average penalty per episode: 0.0
Average reward per episode: -2.39

Process finished with exit code 0
```

References

- Deep Reinforcement Learning, UW CSE Deep Learning – Felix Leeb
- CSCE-689 Reinforcement Learning, Guni Sharon
- Lecture 7: DQN (Sung Kim) https://youtu.be/S1Y9eys2bdg
- Mnih et al., Human-level control through deep reinforcement learning, Nature 2015
- Mnih et al., Playing Atari with Deep Reinforcement Learning, Arxiv 2013