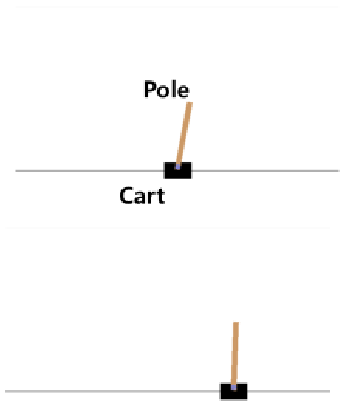
Deep Q-Learning

Programming Project

Implementing a Deep Q-learning

- Your Goal is to solve the Cartpole environment, **using neural network as a Q-value generator.**

CartPole-v1



- The Environment
  - moveable cart
  - a pole attached to the cart
- Rule
  - the episode ends if
    - if the cart leaves (-2.4, 2.4) range
    - if the pole angle in not in ±12°
- Possible Actions
  - Move left
  - Move Right
- Goal
  - maintain balance as long as possible

CartPole documentation: https://www.gymlibrary.dev/environments/classic_control/cart_pole/

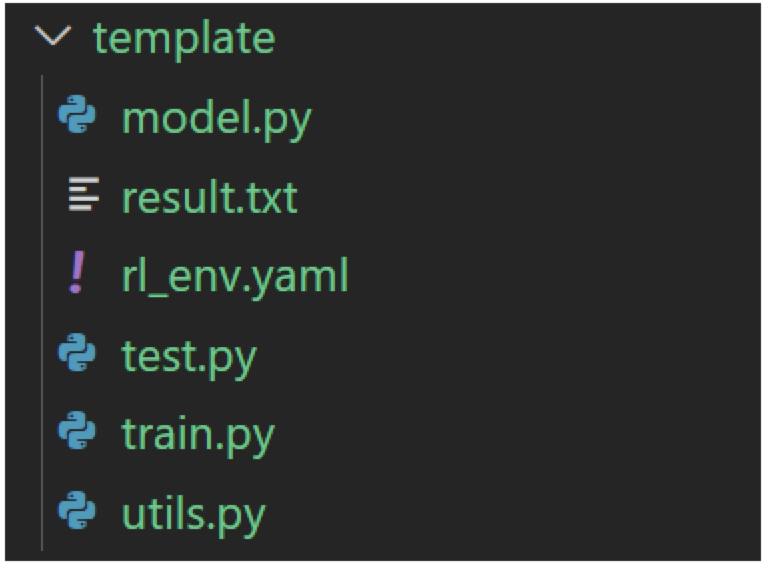
Implementing a Deep Q-learning

- Your Goal is to solve **the Cartpole** environment, **using neural network as a Q-value generator.**

- Refer to the document of the previous homework for additional info on the problem setting.

Skeleton code

- A Skeleton code file will be given. the folder structure is as follows:



- you are free to modify any file in any way, EXCEPT the test.py module.

Skeleton code

- **rl_env.yaml**
  - Your final Model will be evaluated on the TA's computer.
  - Please use gym=0.21.0 and pyglet=1.5.27

```
nlee# conda install -c conda-forge gym=0.21.0
```

```
nlee# conda install -c conda-forge pyglet=1.5.27
```

  - but if you want the exact same anaconda environment as TA, use this .yaml file.

```
# conda env create --file rl_env.yaml
```

Skeleton code

- **model.py**
  - Implement your neural network model here

```python
import torch.nn as nn
class Q_net(nn.Module):
    def __init__(self):
        super().__init__()
        '''
        ######################################################
        TODO: Implement Your Model
        ######################################################
        '''
    def forward(self, x):
        Q_values = self.layers(x)
        return Q_values
```

Skeleton code

- **train.py**
  - the purpose of this module is to train the neural network.
  - Implement your train code here. just writing down these two blocks are recommended.

```python
            # Execute the action then collect outputs
            state, reward, done, _ = env.step(action)

            #Update the memory
            '''
                ####################################################
                TODO:Implement your memory
                ####################################################
            '''

            # Update the Q-net parameters
            replay(model, memory, args, criterion, optimizer, iteration = 1)

            state_0 = state
```

```python
    if len(memory) < args.batchsize:
        return None
    d_loader = DataLoader(memory, args.batchsize ,shuffle = True, drop_last= True)


    for i, batch in enumerate(d_loader):
        '''
            ####################################################
            TODO: Implement your replay function.
            ####################################################
        '''


if __name__ == '__main__':
```

Skeleton code

- **train.py(continued)**
  - Use torch.save to save your model's trained parameter

```
#  when the agent 'solves' the environment: steak ove
if num_streaks > args.streak_to_end:
    print("The Environment is solved")
    torch.save(model.state_dict(), 'modelparam.pt')
    break
```

  - the parameter will be loaded on the test.py module, to test your model.

```
abspath = os.path.dirname(os.path.abspath(__file__))+'/modelparam_final.pt'
THE_model.load_state_dict(torch.load(abspath))
```

Skeleton code

- **test.py**
  - Your work will be evaluated using this exact code.
  - in other files, you are free to modify them freely, even ignoring the TODO blocks
  - **BUT the TA will evaluate your code using this original test.py module.**
  - So, test your code running this module before submitting

```python
def validate(model:nn.Module, iteration):
    iteration = iteration
    with torch.no_grad():
        total_t = 0
        for episode in tqdm(range(iteration)):
            done = False
            state_0 = env.reset()
            while done == False:
                total_t += 1
                actiontable = model(torch.Tensor(state_0).unsqueeze(0))
                action = np.argmax(actiontable).item()
                state, _, done, _ = env.step(action)
                state_0 = state
    return total_t/iteration

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='2022 AI Project')
    parser.add_argument('--iteration', type = int, default= 1000)

    args = parser.parse_args()

    THE_model = Q_net()
    abspath = os.path.dirname(os.path.abspath(__file__))+'/modelparam_final.pt'
    THE_model.load_state_dict(torch.load(abspath))
    env = gym.make('CartPole-v1')
    score = str(validate(THE_model, args.iteration))
    with open('result.txt', 'w') as f:
        f.writelines(score)
```

Recommended algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

from Playing Atari with Deep Reinforcement Learning(1312.5602.pdf (arxiv.org)), page 5

Project: Implementing Deep Q-learning

- Due: 2022/12/17 23:59
- Submission: to HY-ON LMS
- Submit following files
  - short report(<5 pages, .pdf)
    - explain your code structure
    - include the training result(plot/figure is recommended, but just text output is also fine)
  - Source code(.zip)
    - change the name of the 'template' folder to your student ID
    - **Must include Parameter file(< 100MB)**
    - **Must generate 'result.txt' on executing "python test.py"**

Project: Implementing Deep Q-learning

- Please submit your code and parameter even if you could not solve(= surviving 120 consecutive episodes), because the evaluation score is average of timestep survived

- Contact TA: foldtwice@hanyang.ac.kr

Grading

- 100 Points total
- Performance: 50 points
  - 50 – 0.5*(your percentile)
    - your percentile $= 100 * \left(1 - \frac{\#\text{students below you}}{\#\text{total students} - 1}\right)$
- Report: 50 points
  - Analysis 30 points
  - Idea 20 points