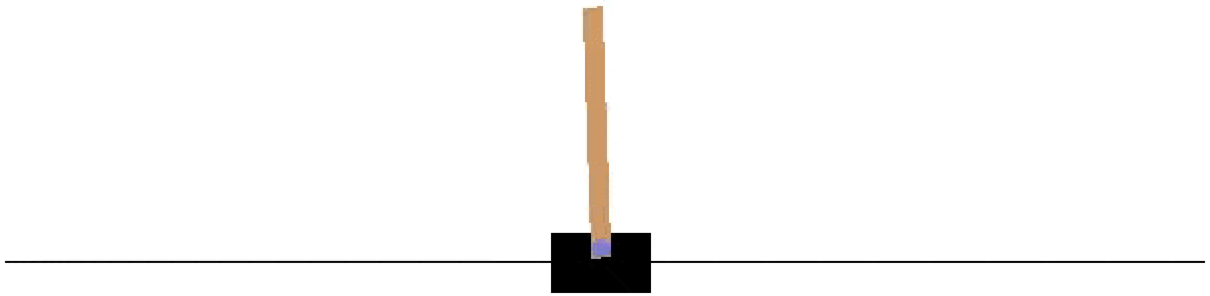
DQN Hands-on

Prof. Seungchul Lee
Industrial AI Lab.

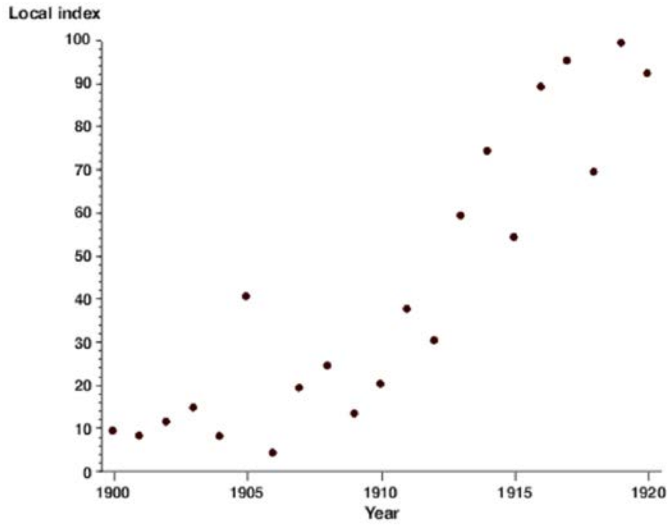
CartPole-v0

- A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track.
- The system is controlled by applying a force of +1 or -1 to the cart.
- The pendulum starts upright, and the goal is to prevent it from falling over.
- A reward of +1 is provided for every timestep that the pole remains upright.
- The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.



Experience Replay

- Learning from batches of consecutive samples is problematic:
  - Samples are correlated → inefficient learning
  - Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) → can lead to bad feedback loops



- Address these problems using experience replay
  - Continually update a replay memory table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
  - Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

- It has been shown that this greatly stabilizes and improves the DQN training procedure.

Experience Replay

```
done = False
state = env.reset()

while not done:

    Q = sess.run(Qpred, feed_dict = {x: np.reshape(state, [1, n_input])})

    if np.random.uniform() < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q)

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

    if done:
        reward = -200

    replay_buffer.append([state, action, reward, next_state, done])

    if len(replay_buffer) > n_buffer:
        replay_buffer.pop(0)

    state = next_state
```

Experience Replay

```
if episode % n_batch == 1:

    for _ in range(50):

        minibatch = random.sample(replay_buffer, n_batch)

        x_stack = np.empty(0).reshape(0, n_input)
        y_stack = np.empty(0).reshape(0, n_output)

        for state, action, reward, next_state, done in minibatch:

            Q = sess.run(Qpred, feed_dict = {x: np.reshape(state, [1, n_input])})

            if done:
                Q[0][action] = reward
            else:
                Q_next = sess.run(Qpred, feed_dict = {x: np.reshape(next_state, [1, n_input])})
                Q[0][action] = reward + gamma*np.max(Q_next)

            x_stack = np.vstack([x_stack, state])
            y_stack = np.vstack([y_stack, Q])

        sess.run(optm, feed_dict = {x: x_stack, y: y_stack})
```

Learned Optimal Policy

```python
state = env.reset()

done = False

while not done:

    env.render()

    s = map_discrete_state(state)
    action = np.argmax(Q_table[s,:])

    next_state, _, done, _ = env.step(action)
    state = next_state

env.close()
```