# DQN(Deep Q-Network)

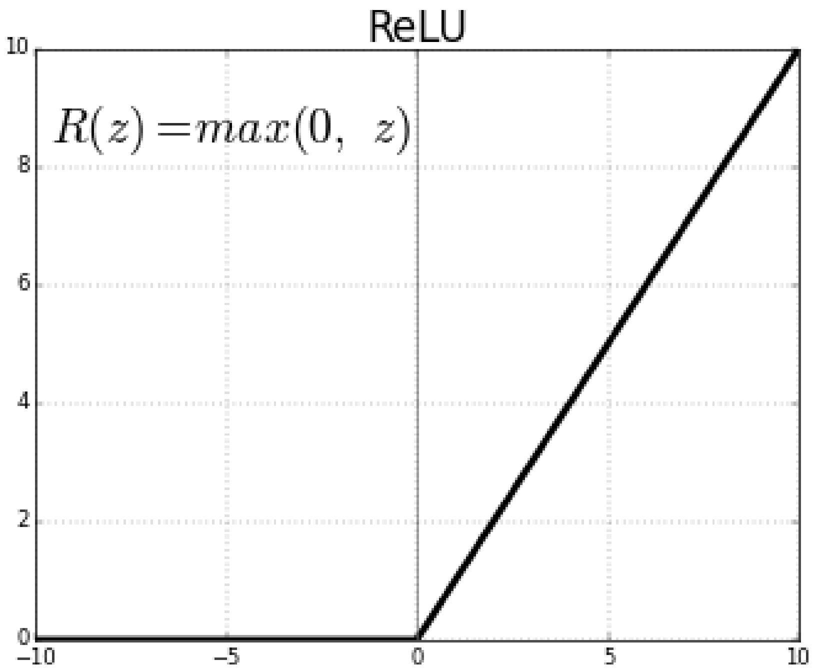
## Cartpole-v1

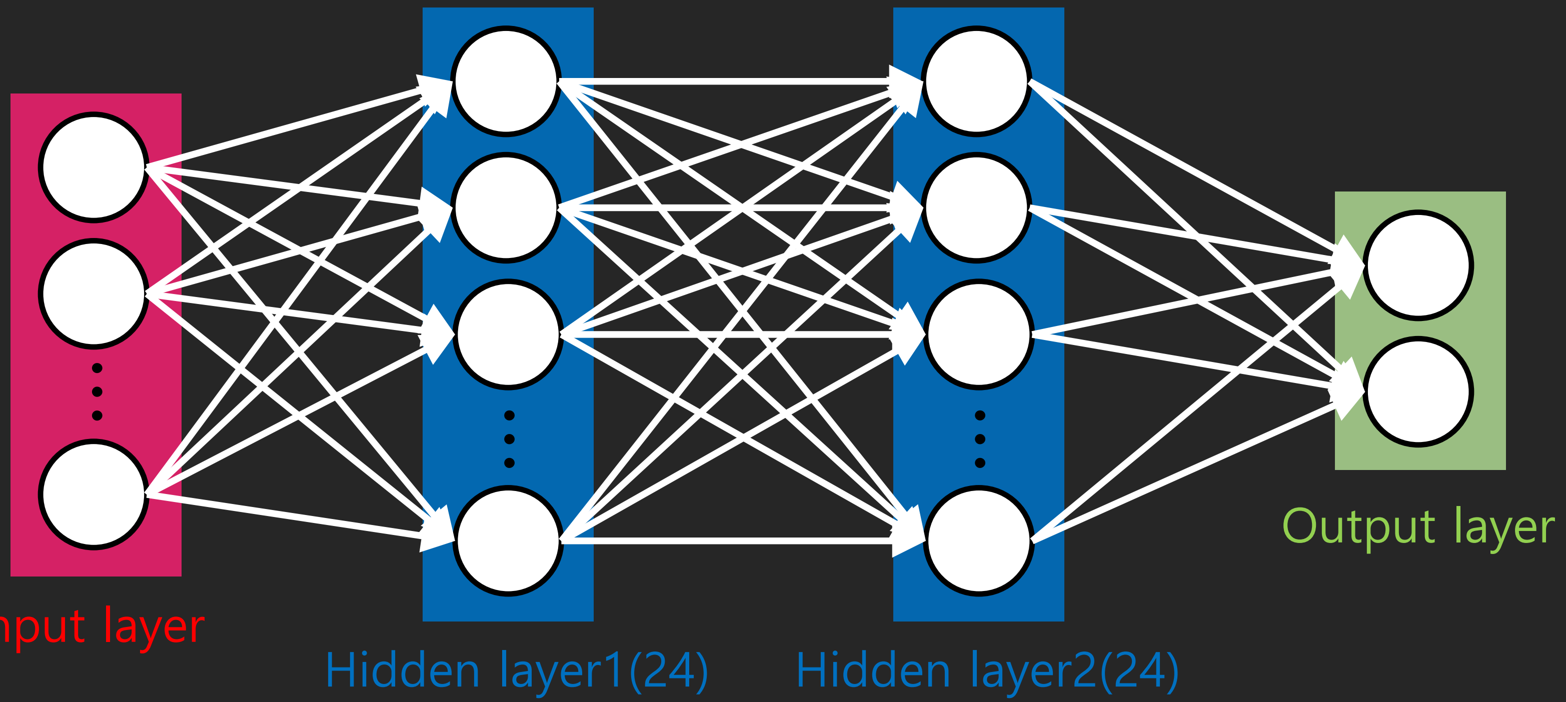
민세웅

한양대학교지능통신연구실
Intelligent Communication Systems Lab

한양대학교지능통신연구실
Intelligent Communication Systems Lab

# Network Construction




```python
# 상태가 입력, 큐함수가 출력인 인공신경망 생성
def build_model(self):
    model = Sequential()
    model.add(Dense(24, input_dim=self.state_size, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(24, activation='relu',
                    kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='linear',
                    kernel_initializer='he_uniform'))
    model.summary()
    model.compile(loss='mse', optimizer=Adam(lr=self.learning_rate))
    return model
```

OpenAI 한양대학교 iCSL 한양대학교 지능통신연구실 Intelligent Communication Systems Lab

# Linear Regression

$$cost(W) = (Ws - y)^2$$

states target

$$y = r + \gamma \max Q(s')$$

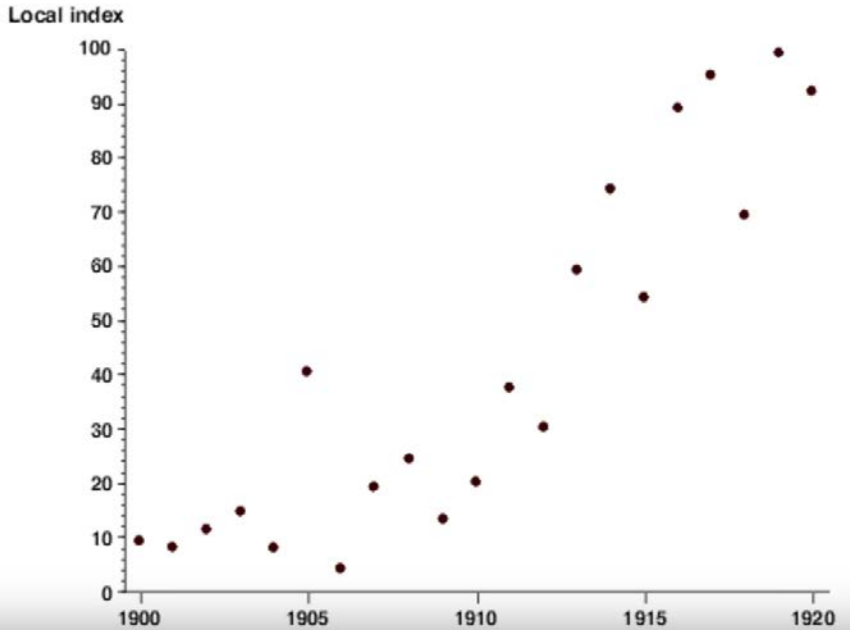
```
# 현재 상태에 대한 모델의 큐함수
# 다음 상태에 대한 타깃 모델의 큐함수
target = self.model.predict(states)
target_val = self.target_model.predict(next_states)

self.model.fit(states, target, batch_size=self.batch_size,
               epochs=1, verbose=0)
```

```
# 벨만 최적 방정식을 이용한 업데이트 타깃
for i in range(self.batch_size):
    if dones[i]:
        target[i][actions[i]] = rewards[i]
    else:
        target[i][actions[i]] = rewards[i] + self.discount_factor * (
            np.amax(target_val[i]))
```

# Reinforcement Neural Net

- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

# Correlation Between Samples

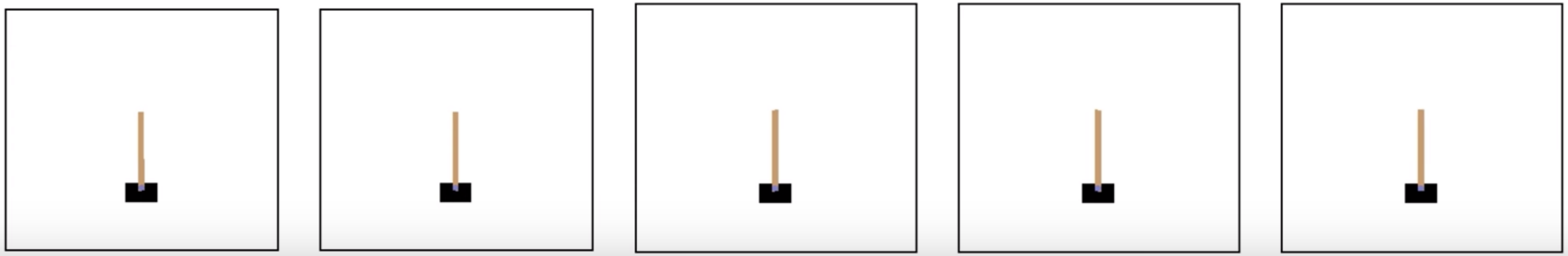



**Algorithm 1** Deep Q-learning

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 $\epsilon$ 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 $\phi_{t+1} = \phi(s_{t+1})$

    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**

OpenAI



# Non-stationary Targets

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

$$\hat{Y} = \hat{Q}(s_t, a_t|\theta)$$

$$Y = r_t + \gamma \max_{a'} \hat{Q}_\theta(s_{t+1}, a'|\theta)$$

OpenAI

한양대학교지능통신연구실 Intelligent Communication Systems Lab

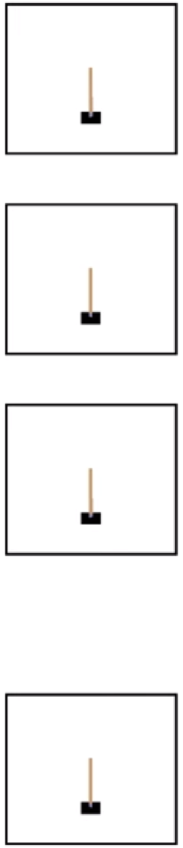
# DQN's three Solutions

1. Go deep
2. Capture and replay
   - Correlations between samples
3. Separate networks: create a target network
   - Non-stationary targets

OpenAI

한 양 대 학 교 지 능 통 신 연 구 실
Intelligent Communication Systems Lab

# Experience Replay



Capture →

| |
|---|
| $s_1, a_1, r_2, s_2$ |
| $s_2, a_2, r_3, s_3$ |
| $s_3, a_3, r_4, s_4$ |
| $\dots$ |
| $s_t, a_t, r_{t+1}, s_{t+1}$ |

random sample & Replay →

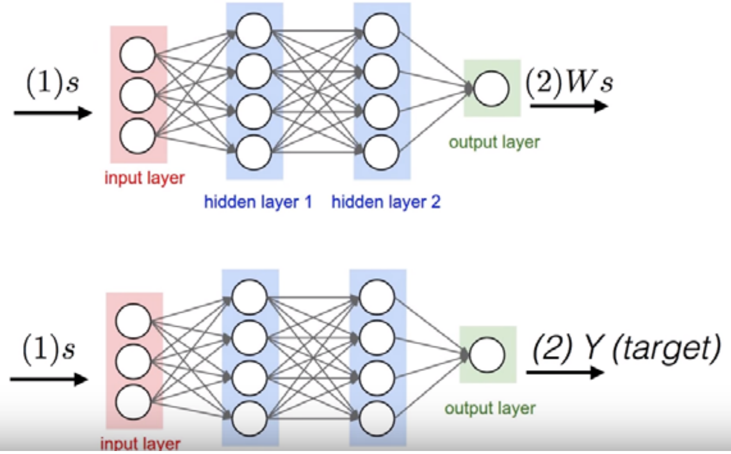
$$\min_{\theta} \sum_{t=0}^{T} [\hat{Q}(s_t, a_t|\theta) - (r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta))]^2$$

# Experience Replay

```python
# 리플레이 메모리, 최대 크기 2000
self.memory = deque(maxlen=2000)

self.batch_size = 64

# 샘플 <s, a, r, s'>을 리플레이 메모리에 저장
def append_sample(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))

# 메모리에서 배치 크기만큼 무작위로 샘플 추출
mini_batch = random.sample(self.memory, self.batch_size)

states = np.zeros((self.batch_size, self.state_size))
next_states = np.zeros((self.batch_size, self.state_size))
actions, rewards, dones = [], [], []

for i in range(self.batch_size):
    states[i] = mini_batch[i][0]
    actions.append(mini_batch[i][1])
    rewards.append(mini_batch[i][2])
    next_states[i] = mini_batch[i][3]
    dones.append(mini_batch[i][4])
```

# Separate target Network

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




```
# 모델과 타깃 모델 생성
self.model = self.build_model()
self.target_model = self.build_model()
# 타깃 모델 초기화
self.update_target_model()
# 타깃 모델을 모델의 가중치로 업데이트
def update_target_model(self):
    self.target_model.set_weights(self.model.get_weights())
# 현재 상태에 대한 모델의 큐함수
# 다음 상태에 대한 타깃 모델의 큐함수
target = self.model.predict(states)
target_val = self.target_model.predict(next_states)

# 벨만 최적 방정식을 이용한 업데이트 타깃
for i in range(self.batch_size):
    if dones[i]:
        target[i][actions[i]] = rewards[i]
    else:
        target[i][actions[i]] = rewards[i] + self.discount_factor * (
            np.amax(target_val[i]))
if done:
    # 각 에피소드마다 타깃 모델을 모델의 가중치로 업데이트
    agent.update_target_model()
```

OpenAI

한양대학교 지능통신연구실 Intelligent Communication Systems Lab