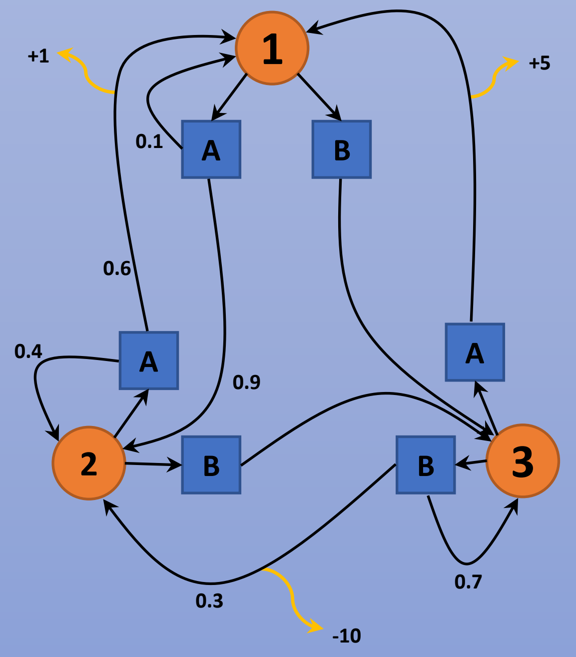
Decision Making III

Katie DC

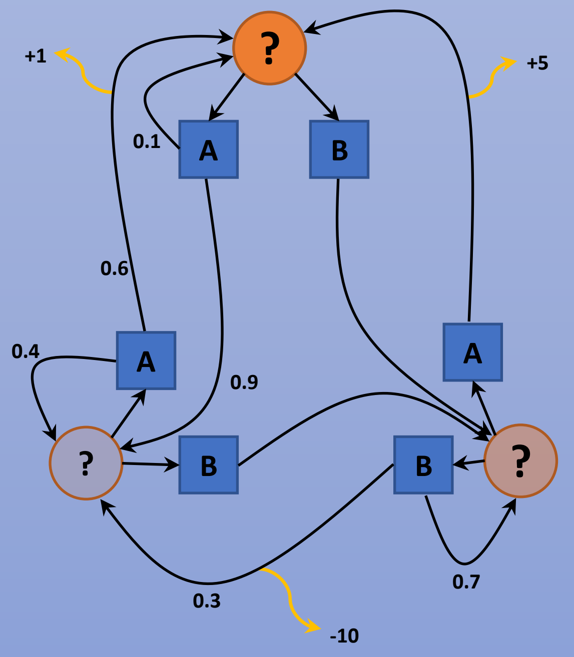
Markov Models

Markov Decision Processes (MDP)



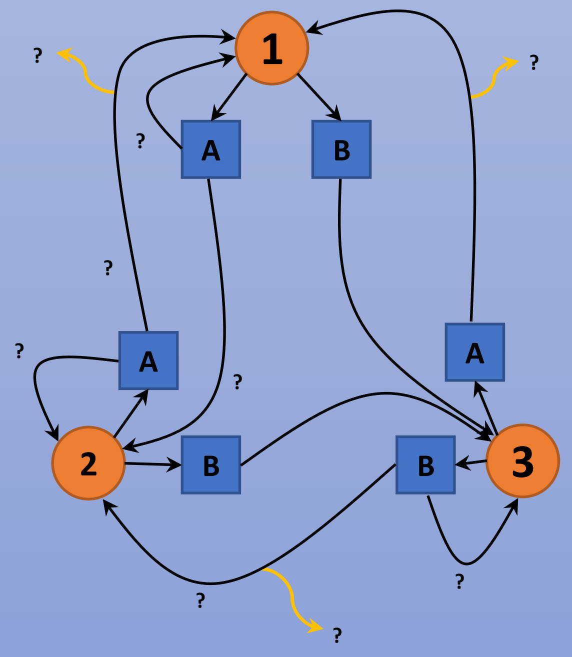
Uncertainty in effects of actions

Partially Observable MDP



Uncertainty in current state

Reinforcement Learning



Uncertainty in model

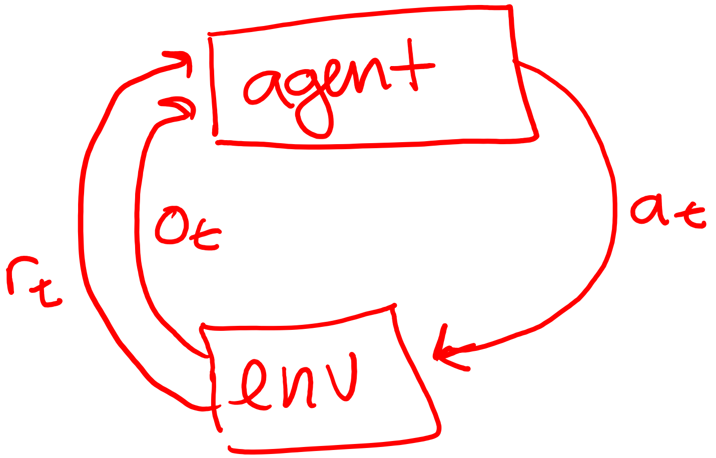
Challenges for Reinforcement Learning

1. **Exploration** of the world must be balanced with **exploitation** of the knowledge gained through previous experience
2. Reward may be received long after important choices have been made, so **credit must be assigned to earlier decisions**
3. Must **generalize** from limited experience

There are many solutions to this problem!

For a comprehensive overview, check out *Reinforcement Learning: An Introduction* by Sutton and Barto.

Reinforcement Learning



not given T or R directly
→ must learn through experience

Goal: determine optimal actions that maximize expected reward

$$\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

solution methods:

- model based
- model free

Q-learning: Model-free method: Q-Learning

- Agent gathers experience: (s, a, r, s')
- Q-function returns the expected reward of that action at that state
- *Temporal Differences* to estimate optimal value Q^* for each state
- Agent maintains Q-table of all Q values for each state s and action a

Incremental Estimation

suppose we have a random variable X

how to estimate mean given samples $x_{1:n}$?

$$\hat{x}_n = 1/n \sum_{i=1}^{n} x_i$$

we can show that

$$\hat{x}_n = \hat{x}_{n-1} + 1/n (x_n - \hat{x}_{n-1})$$

$\alpha(n) \rightarrow$ learning rate, often constant

$$\hat{x} \leftarrow \hat{x} + \alpha (x - \hat{x})$$

temporal difference error

Incremental Estimation Example

current mean estimate of $3 = \hat{x}$

new sample: $x = 7$

$$\hat{x} \leftarrow \hat{x} + \alpha(x - \hat{x})$$

if $\alpha = .1$

$$\hat{x} \leftarrow 3 + .1(7-3) = 3.4$$

if $\alpha = .5$

$$\hat{x} \leftarrow 3 + .5(7-3) = 5$$

Q-Learning (1)

key idea: apply incremental estimation to Bellman eq.

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) U(s')$$

$$= R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q(s',a')$$

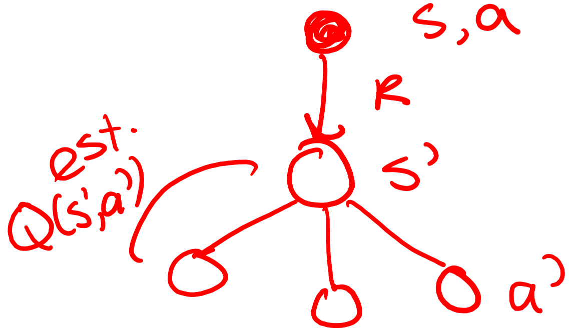
since we don't have T or R, use observed next state s' + r to estimate the Q values

use the following incremental update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Q-Learning (2)

Q-Learning Algorithm



function Qlearning

$t \leftarrow 0$

s_0 ←initial state

Initialize Q

loop

Choose action a_t based on Q and some exploration strategy

Observe new state s_{t+1} and reward r_t

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right)$$

$t \leftarrow t + 1$

s a r s' a'

Q-Learning Challenges

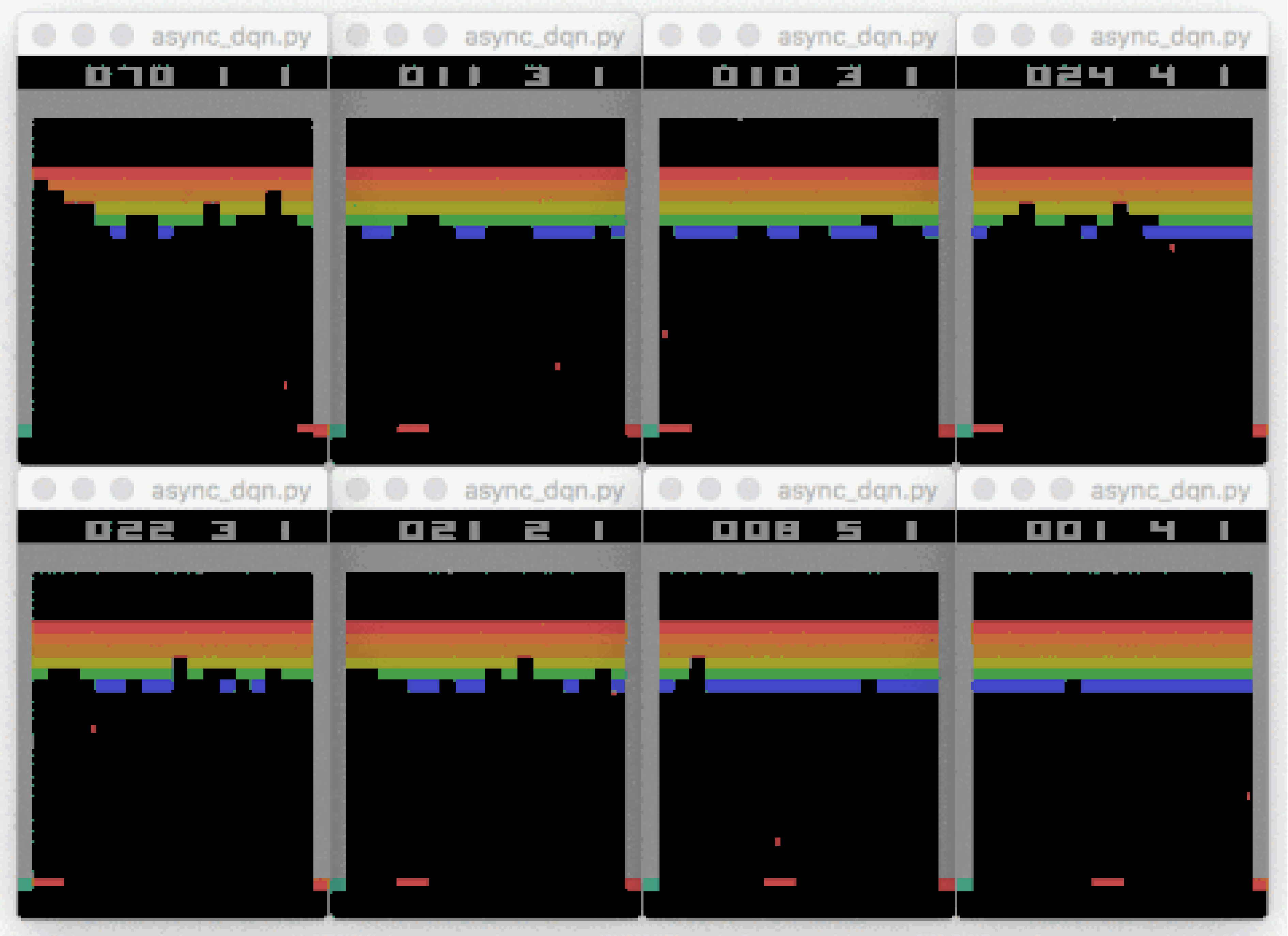
- How should an agent decide which actions to choose to explore?
- One way to define probabilistic exploration strategy, using the Boltzmann distribution:

$$P(\mathrm{a}|\mathrm{s}) = \frac{\mathrm{e}^{Q(s,a)/k}}{\sum_j e^{Q(s,a_j)/k}}$$

The k parameter (called temperature) controls probability of picking non-optimal actions. If k is large, all actions are chosen uniformly (explore), if k is small, then the best actions are chosen.

Q-Learning Challenges

- How should an agent decide which actions to choose to explore?
- The Q Table can be thought of as a cheat sheet. How many states and actions must be stored for a game of chess?
- Another issue generally in RL: How to know if reward is correct? How do we best shape the reward to get a desirable outcome? Is that okay?


async_dqn.py
070 1 1
async_dqn.py
011 3 1
async_dqn.py
010 3 1
async_dqn.py
024 4 1
async_dqn.py
022 3 1
async_dqn.py
021 2 1
async_dqn.py
008 5 1
async_dqn.py
001 4 1

DQN: approximate Q with deep network

- $\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
- $Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha[\text{target}]$
- Goal is to approximate our Q table with a deep network that will act as a Q Function

function DQN
  s_0 ←initial state
  Initialize Q_0
  for k = 1,2,…
    Choose action a_t / Observe new state s_{t+1} and reward r_t
    $\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
    $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla \mathbb{E}_{s' \sim P(s'|s,a)} [\mathrm{Q}_\theta(s, a) - \text{target}(\mathrm{s}')] \Big|_{\theta=\theta_k}$
    $s \leftarrow s'$

DQN Challenges

- Deep learning works for supervised learning under these conditions:
  - Samples are i.i.d., meaning that each batch has the same distribution and all samples are independent within the batch
  - For some input, the label is consistent across time

- In RL, these typically do not hold
  - Target is unstable!
  - Not iid: when parameters are updated, local states are also effected
  - Actions are chosen by estimated Q (we choose what to explore or exploit), this means our target output (action) is constantly changing as well

function DQN
  s_0 ←initial state
  Initialize Q_0
  for k = 1,2,…
    Choose action a_t / Observe new state s_{t+1} and reward r
    $\text{target} = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$
    $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla \mathbb{E}_{s' \sim P(s'|s,a)}[Q_\theta(s,a) - \text{target}(s')]\big|_{\theta=}$
    $s \leftarrow s'$

nonstationary

correlation within trajectories

DQN Solutions

- Experience Replay
  - Say you store 10^6 transitions and use a batch size of 32 to train the network.
  - Sampling from this buffer forms a dataset that is close to iid and therefore stable
- Target network:
  - Use two deep networks! θ^- and θ.
  - First retrieves Q values and the second updates in the training. By temporarily fixing the Q-value targets, the moving target issue is solved.
  - $L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a, ; \theta_i)\right)^2$

DQN Algorithm

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

ε-greedy

$$P(a|s) = \begin{cases} \varepsilon/m + 1 - \varepsilon, & \text{if } a^* \\ \varepsilon/m, & \text{ow} \end{cases}$$