オンラインセミナー「人工知能」
強化学習

奈良先端科学技術大学院大学
松原 崇充, 吉野 幸一郎

後半45分の内容

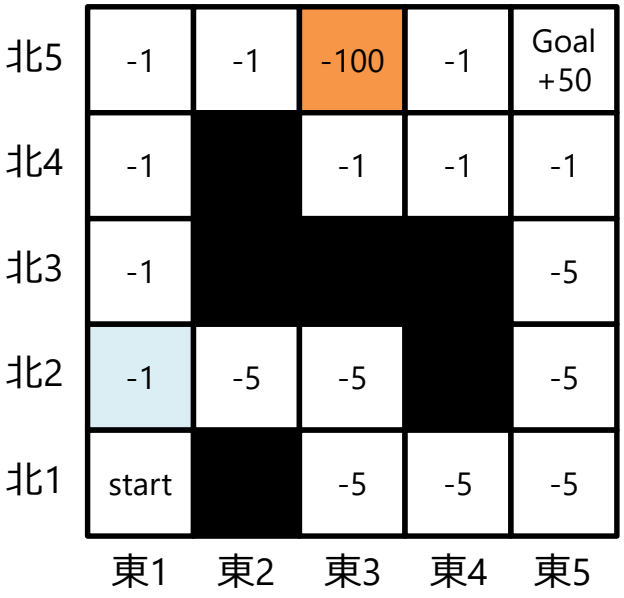
- 価値関数
- 価値関数の最大化（価値ベースの強化学習）
- 関数近似
- 深層強化学習
- 意志決定への応用
- 文生成への応用

後半45分の内容

- **価値関数**
- **価値関数の最大化（価値ベースの強化学習）**
- **関数近似**
- **深層強化学習**
- **意志決定への応用**
- **文生成への応用**

強化学習の目的のおさらい

- **将来に貰える報酬を最大化したい**
  - 青マスからは直進した方がよさそうに見えるがかなり先に渋滞がある
  - トータルのコストを見ると右折して回り道をする方が得
- **将来の報酬の期待値を見る**
  - $V^{\pi}(s) = \sum_{k=0}^{\infty} R_E(s_{t+k})$
  - 北の道は (-1)x6+(-100)+50 = -56
  - 東の道は (-1)x2+(-5)x7+50 = 13



先々のコストを見通した上で右の道を選択

報酬の定義

- $R(s, a, s')$　s の時 a を行い s' に遷移したときの報酬
- $R_E(s, a)$　s の時 a を行い得られる報酬の期待値
- $R_E(s)$　s の時可能な行動で得られる報酬の期待値

- **強化学習の目的は、将来得られる報酬の最大化**
  - 地図の例では Goal に辿り着くまでの報酬を最大化したい
  - 近そうに見える道でも実は途中で時間がかかるなどの問題をそのコストに応じて評価したい

価値関数の意味

$$V^{\pi}(s) = \sum_{k=0}^{\infty} R_E(s_{t+k})$$

- **方策 π にはどの状態 (s) でどの行動を取るかが全て記載されている**
  - 北1東1では北へ、北2東1では東へ …
- **記載された方策に従って行動をした場合に得られる報酬の和が価値関数**
  - $R_E(s^{t+k})$ は今から k 回先の状態 s^{t+k} で得られる報酬の期待値
  - 可能な限り先まで計算する

報酬

	東1	東2	東3	東4	東5
北5	-1	-1	-100	-1	G +50
北4	-1		-1	-1	-1
北3	-1				-5
北2	-1	-5	-5		-5
北1	S		-5	-5	-5

価値関数

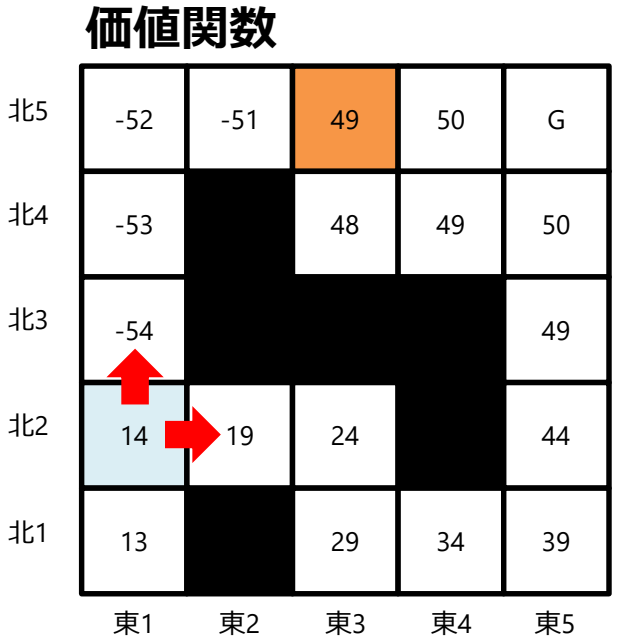
	東1	東2	東3	東4	東5
北5	-52	-51	49	50	G
北4	-53		48	49	50
北3	-54				49
北2	14	19	24		44
北1	13		29	34	39

行動価値関数

- **方策に記載されている以外の行動も検証したい**
  - 北2東1から北か東のどちらへ行くか
  - 検証のために行動 a を変数にする

$$Q^{\pi}(s,a) = \sum_{k=0}^{\infty} \gamma^{k} R_E(s_{t+k}, a_{t+k})$$

γ を 1.0 未満に設定すれば将来の報酬を割り引くことができる。ここでは γ =1.0 を仮定



- **行動ごとに価値関数を求めたものを行動価値関数という**
  - 状態が北2東1のとき
    - 北に行く行動の価値は -54 + (-1) = -55
    - 東に行く行動の価値は 19 + (-5) = 14

東へ行くのがベター

最適価値関数

- **価値関数が最適の方策に従って計算される場合を最適価値関数と呼ぶ**

最適状態価値関数: $V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s)$

最適行動価値関数: $Q^{\pi^*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$

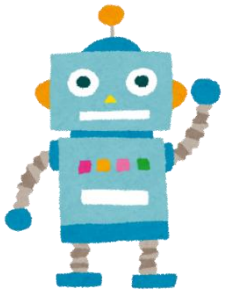
- **最適価値関数を求めれば各状態 s に最良の a が求まる**
  - ただし方策が最適かどうかはわからない…
- **以下の式で再帰的に最適価値関数を求めることができる**

$$Q^{\pi^*}(s,a) = \sum_{s^{t+1}} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q^{\pi^*}(s',a')\right)$$

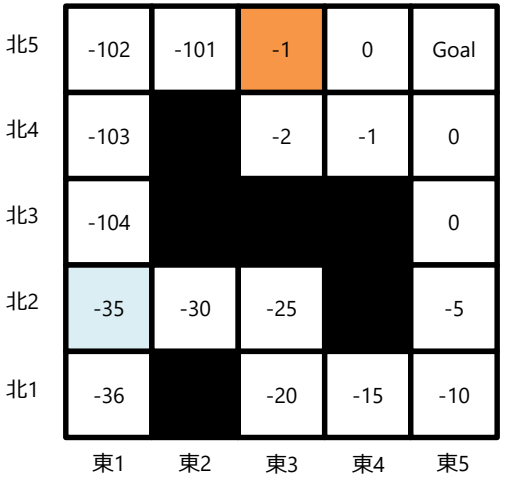
最適行動価値関数って？

$$Q^{\pi^*}(s,a) = \sum_{s^{t+1}} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a} Q^{\pi^*}(s',a')\right)$$

- **なぜこれで最適値が求まるかは本授業で解説しない**
  - 興味がある場合「ベルマン（Bellman）方程式」で検索
- **全ての s と a の組み合わせに $Q^{\pi^*}(s,a)$ を求める**
  - どの a が将来の報酬を最大化するか



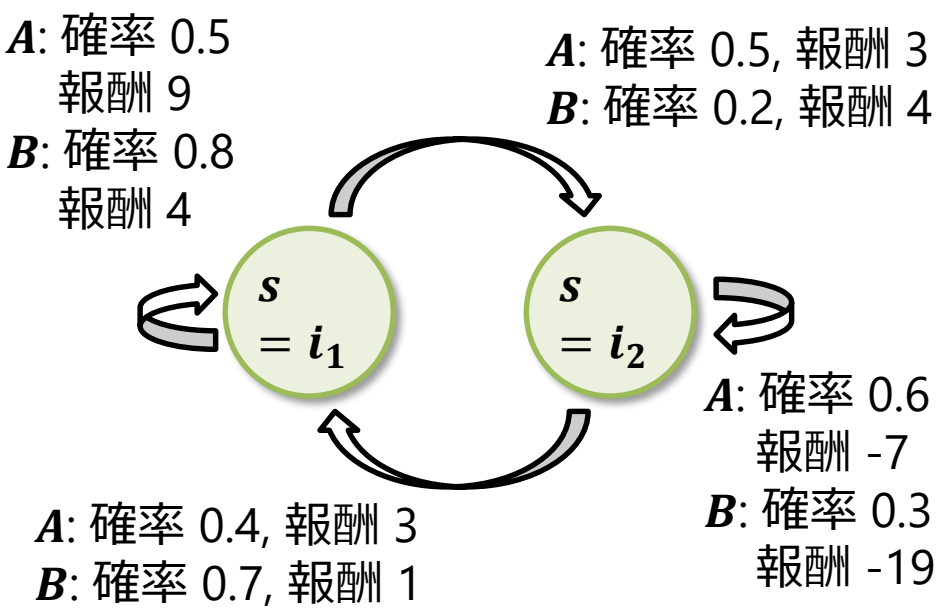
Q^{π^*} (s = 北2東1, a = 北3東1) = -104
Q^{π^*} (s = 北2東1, a = 北2東2) = -30 なので
a = 北2東2 を選択した方がよさそう！



最適行動価値関数を求める

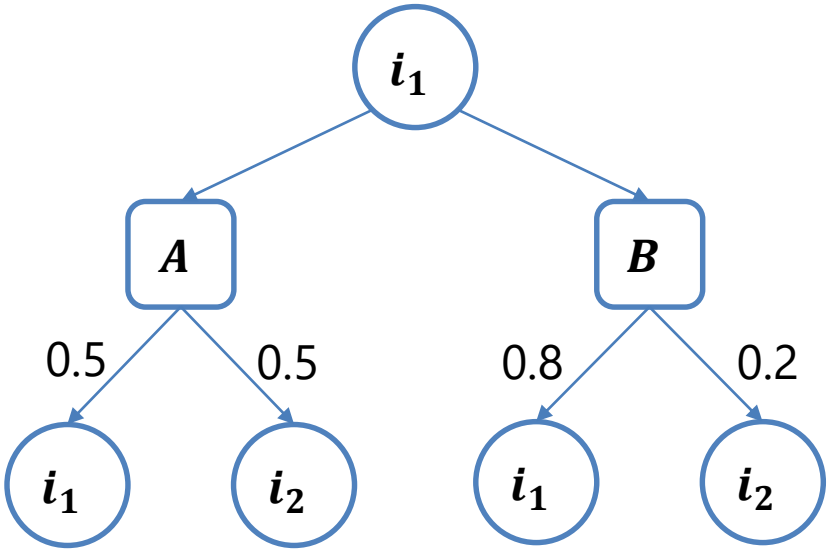
$$Q^{\pi^*}(s,a) = \sum_{s^{t+1}} \underbrace{P(s'|s,a)}_{\text{状態遷移確率}} \left(\underbrace{R(s,a,s')}_{\text{今の報酬}} + \gamma \max_{a'} \underbrace{Q^{\pi^*}(s',a')}_{\text{再帰（漸化式）}} \right)$$

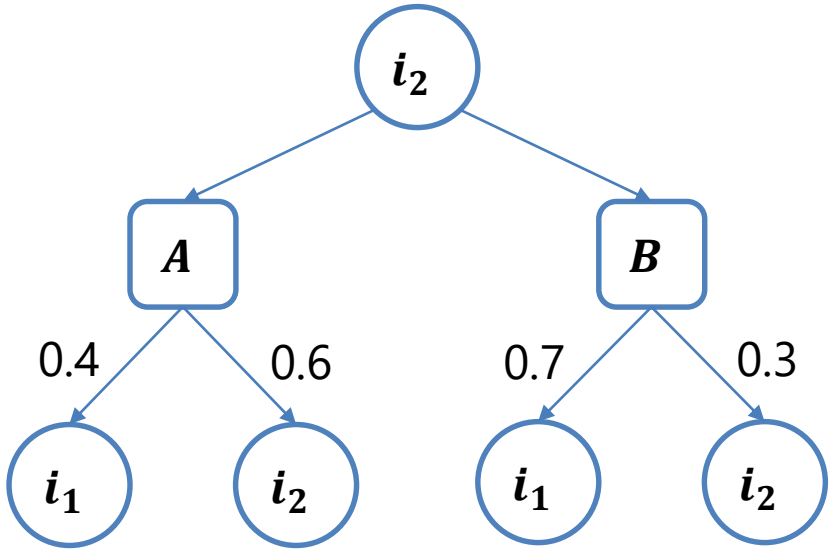
- **本来は状態遷移確率が存在する**（地図の例は確率が 1.0と 0.0）
  - 以下では状態遷移確率が存在するケースを考える
  - 状態は i_1, i_2 のみ
  - 行動は A, B のみ
  - 忘却率 $\gamma = 0.9$
- **各状態 i_1, i_2 でどちらの行動 A, B が最適か？**



行動価値関数の計算

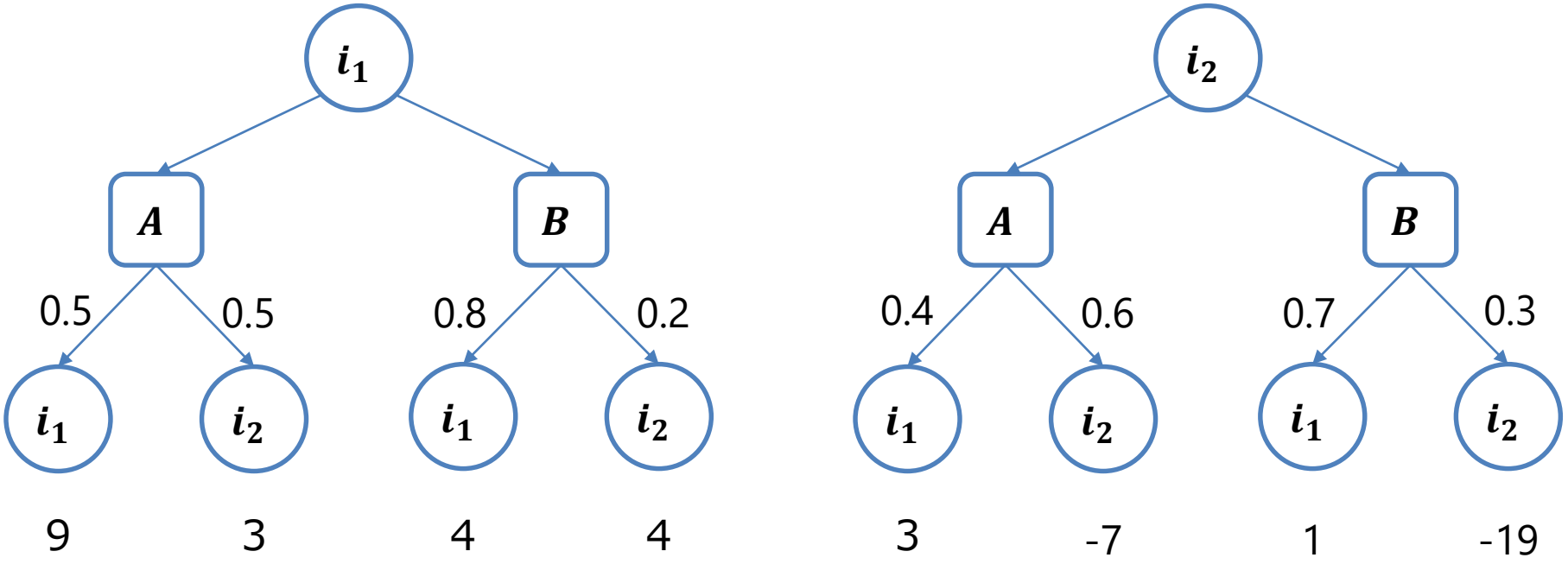
- Q関数は0で初期化
- Q関数および価値関数を計算





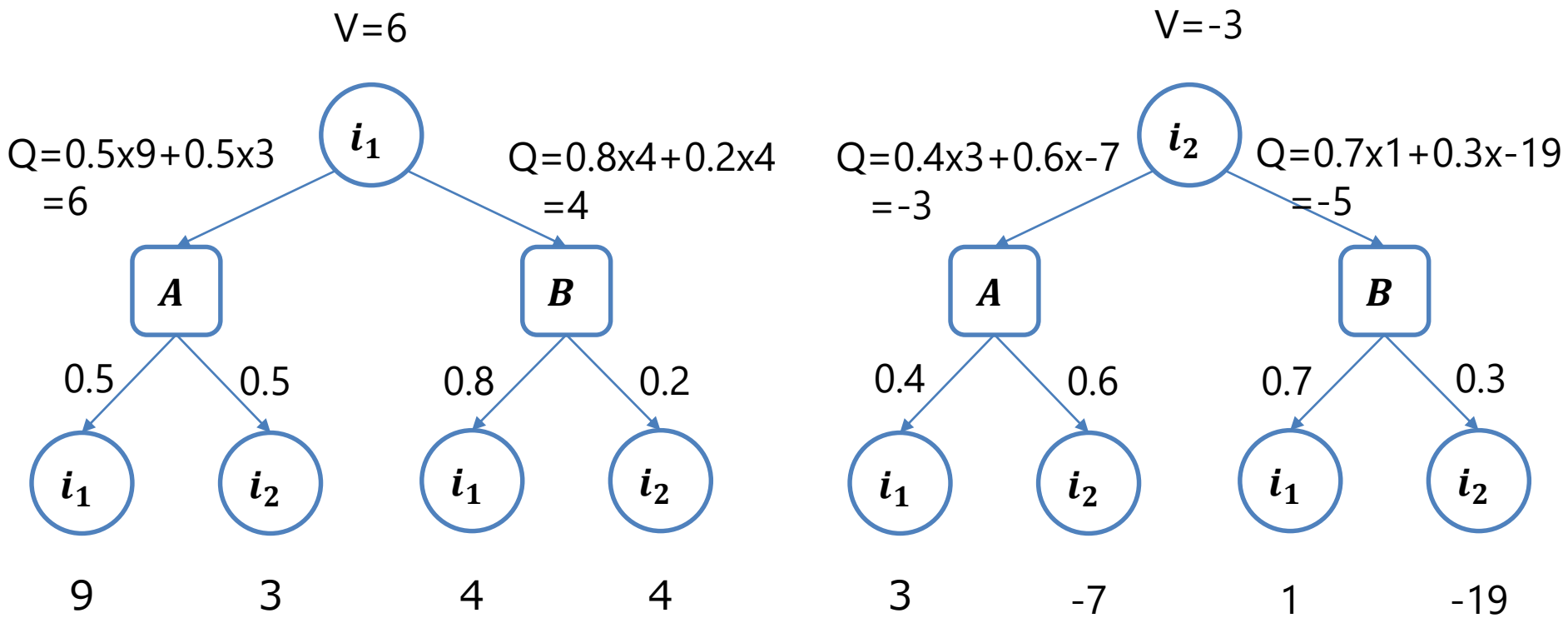
行動価値関数の計算

- 報酬を代入



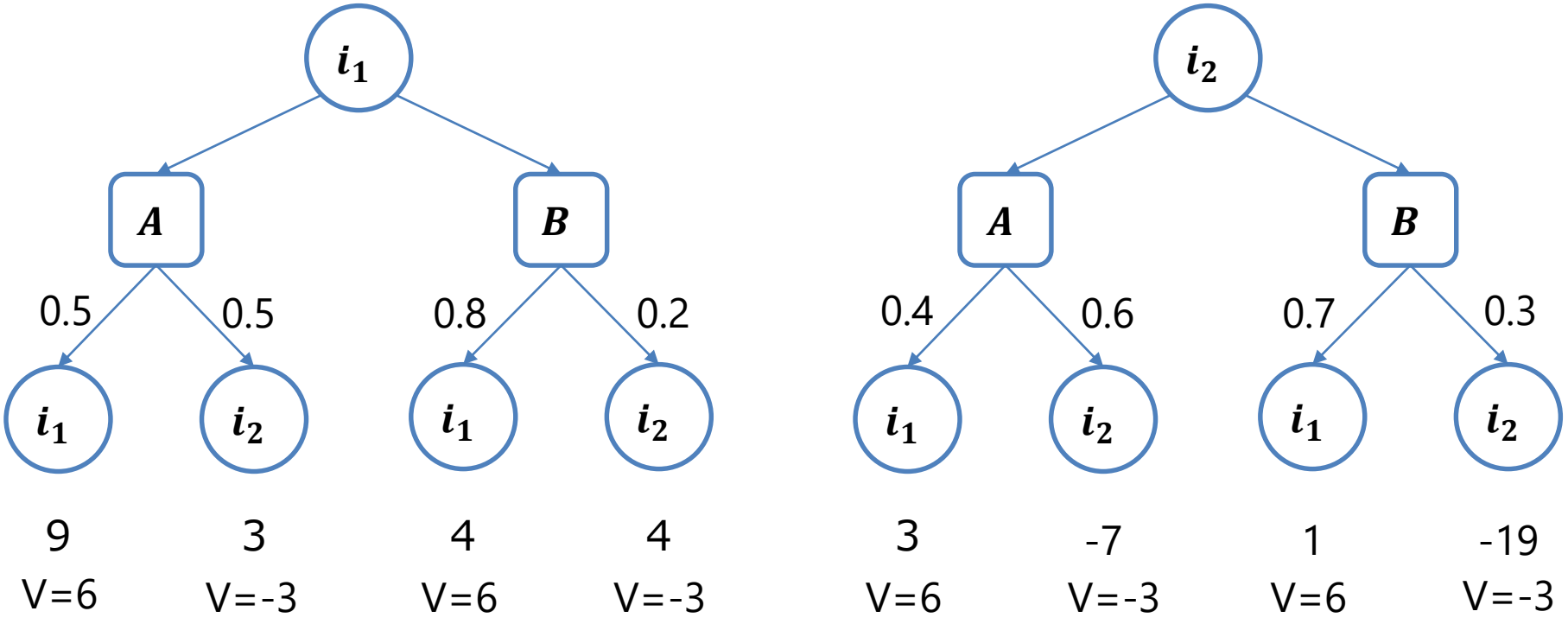
行動価値関数の計算

- **Q関数、価値関数の値を計算**



行動価値関数の計算

- 価値関数を代入してさらに値を更新

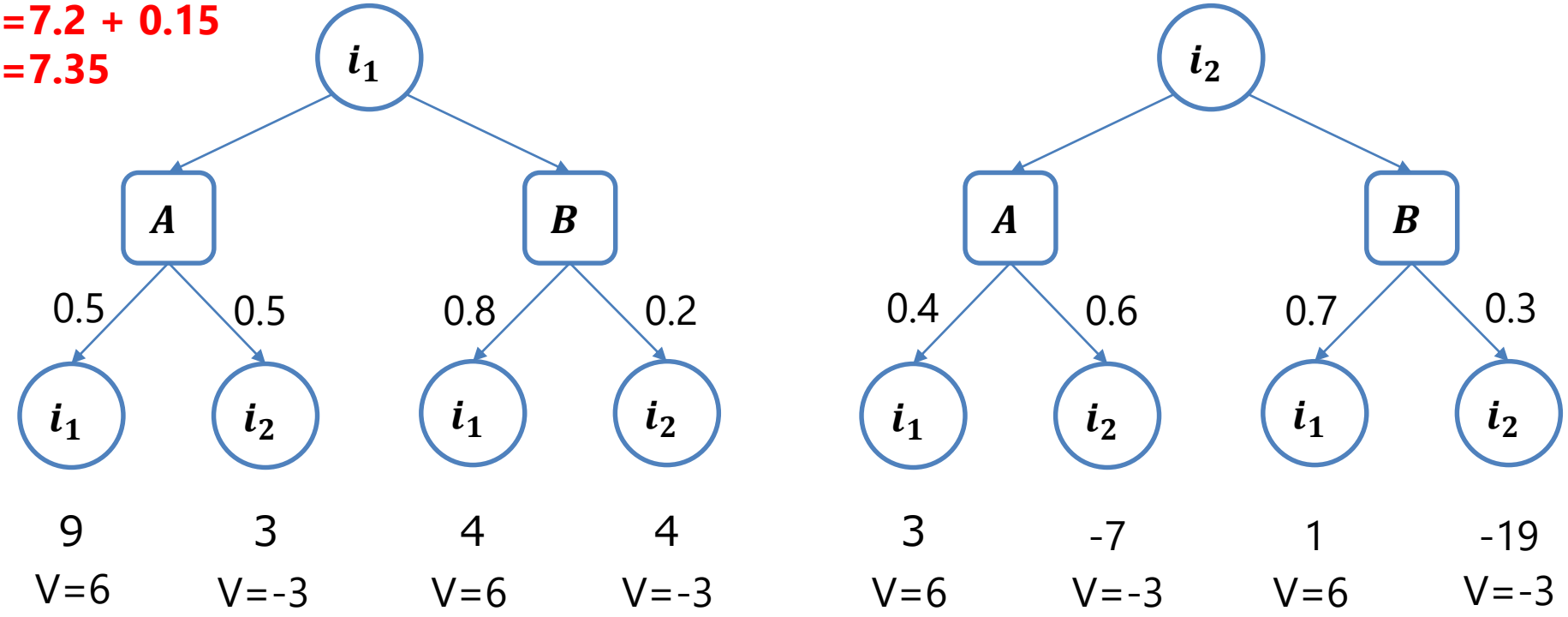


行動価値関数の計算

- 価値関数を代入してさらに値を更新

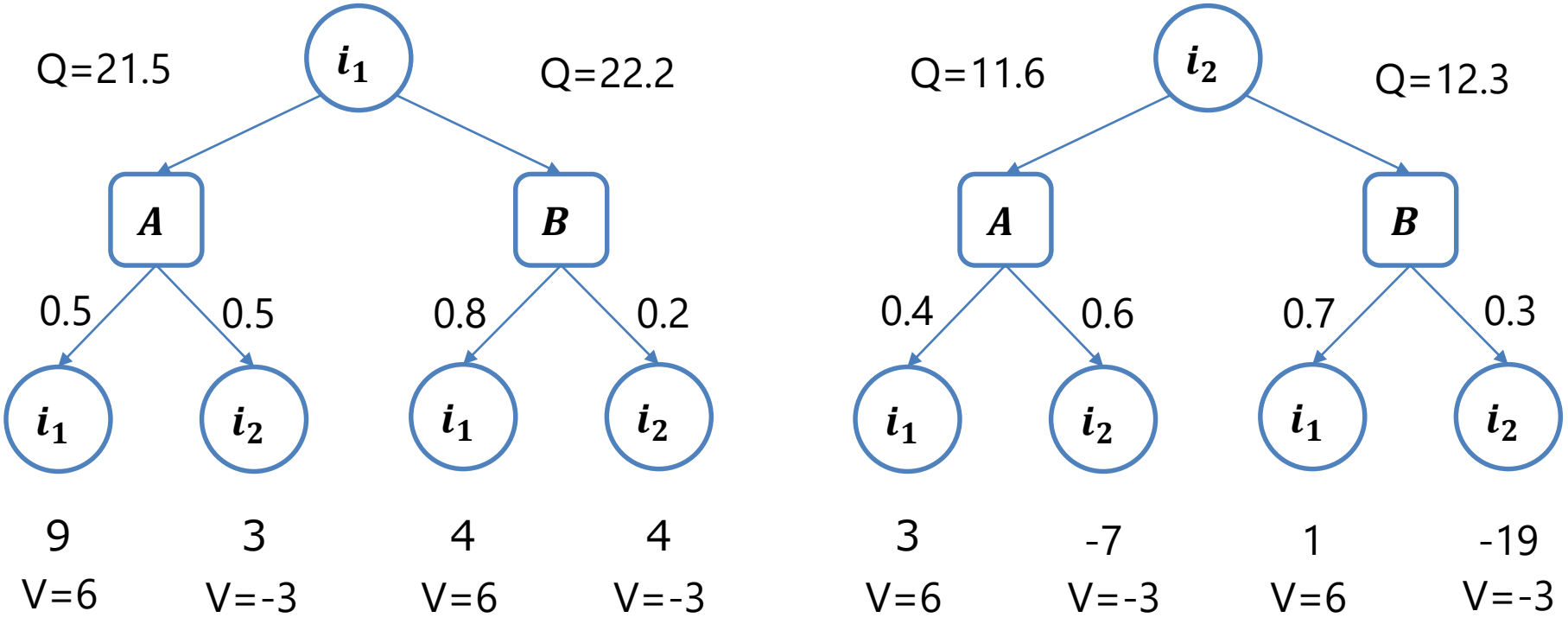
$$Q^{\pi^*}(s,a) = \sum_{s^{t+1}} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q^{\pi^*}(s',a')\right)$$

Q=0.5x(9+ 0.9x6)
+0.5x(3 + 0.9x-3)
=7.2 + 0.15
=7.35



行動価値関数の計算

- **100回くらい繰り返すと...**
  実は行動 B を取り続けることが最適になる



状態遷移のサンプリング

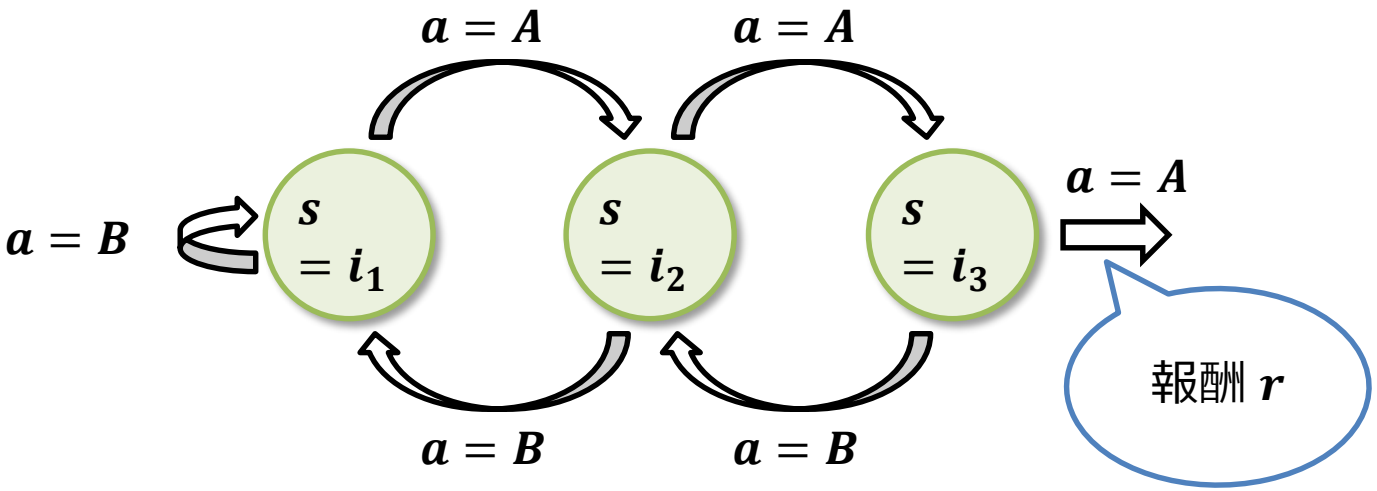
$$Q^{\pi^*}(s,a) = \sum_{s^{t+1}} \underbrace{P(s'|s,a)}_{\text{状態遷移確率}} \left(\underbrace{R(s,a,s')}_{\text{今の報酬}} + \gamma \max_{a'} \underbrace{Q^{\pi^*}(s',a')}_{\text{再帰（漸化式）}} \right)$$

- **状態遷移確率は通常未知**
  - 学習データから求める → 最尤推定、MAP推定
  - 試行を繰り返しながら求める → サンプリング
- **試行を繰り返してサンプリングする**
  - 実際の環境やシミュレーターでの試行から状態遷移確率を近似

$$Q(s,a) \xleftarrow[update]{} (1-\alpha)Q(s,a) + \alpha \left(R(s,a,s') + \gamma \max_{a'} Q(s',a') \right)$$

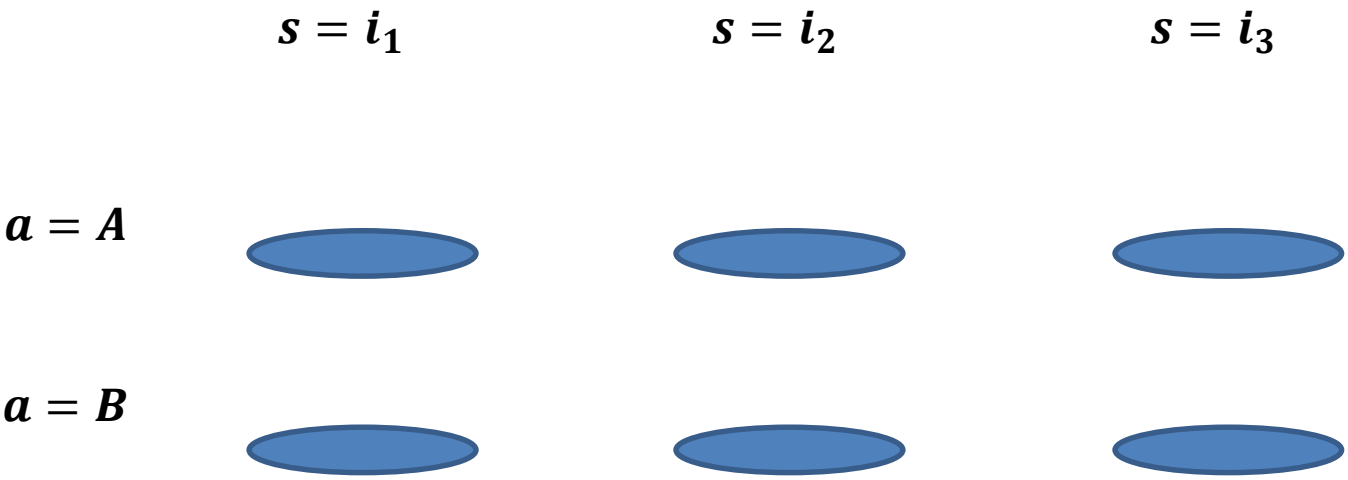
Q学習

Q学習の簡単な例



- s は3種類（i_1, i_2, i_3）、 a は2種類（A, B）しかない
- 状態3の時に行動Aを行うと報酬が貰える
- 状態遷移は全て確率1

Q学習の簡単な例

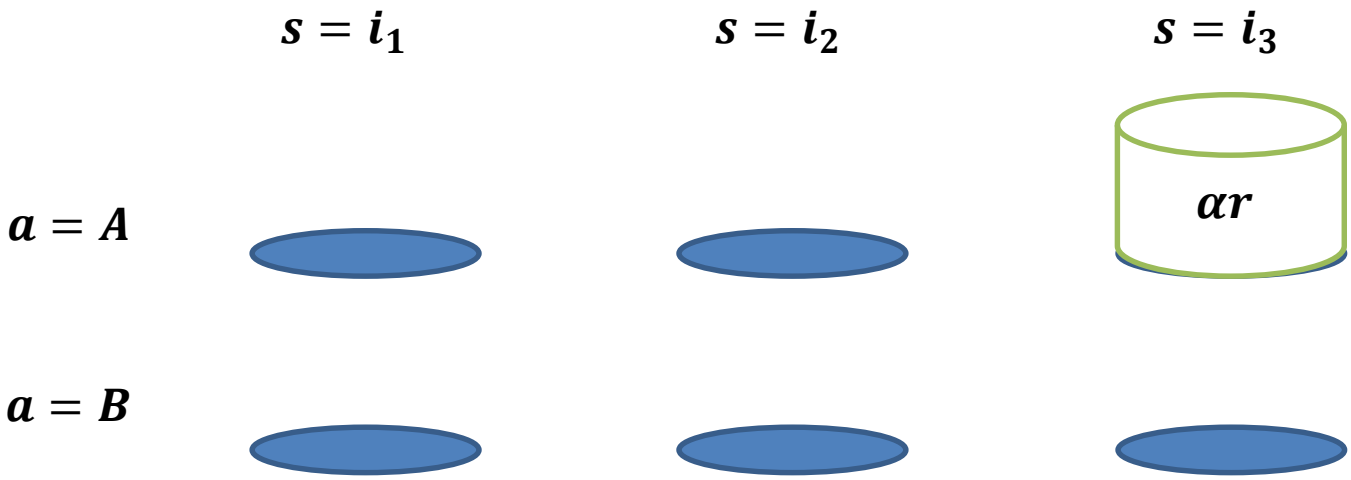


α: 学習率
γ: 忘却率

1. 全ての状態と行動の組み合わせにQ値を設定する

1. 初期値は0とする

Q学習の簡単な例



α: 学習率
γ: 忘却率

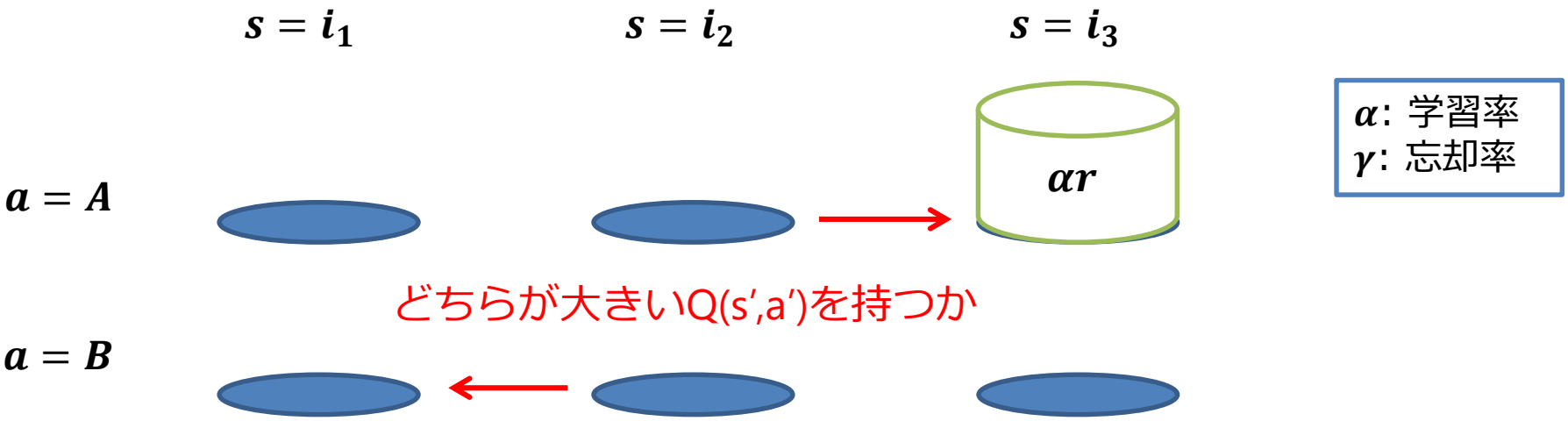
1. **全ての状態と行動の組み合わせにQ値を設定する**

   初期値は0とする

2. **ランダムに行動した結果 $s = i_3, a = A$の時に報酬 r を得る**

   報酬は学習率 α をかけて$Q(s = i_3, a = A)$ に積まれる

Q学習の簡単な例



α: 学習率
γ: 忘却率

1. **全ての状態と行動の組み合わせにQ値を設定する**

   初期値は0とする

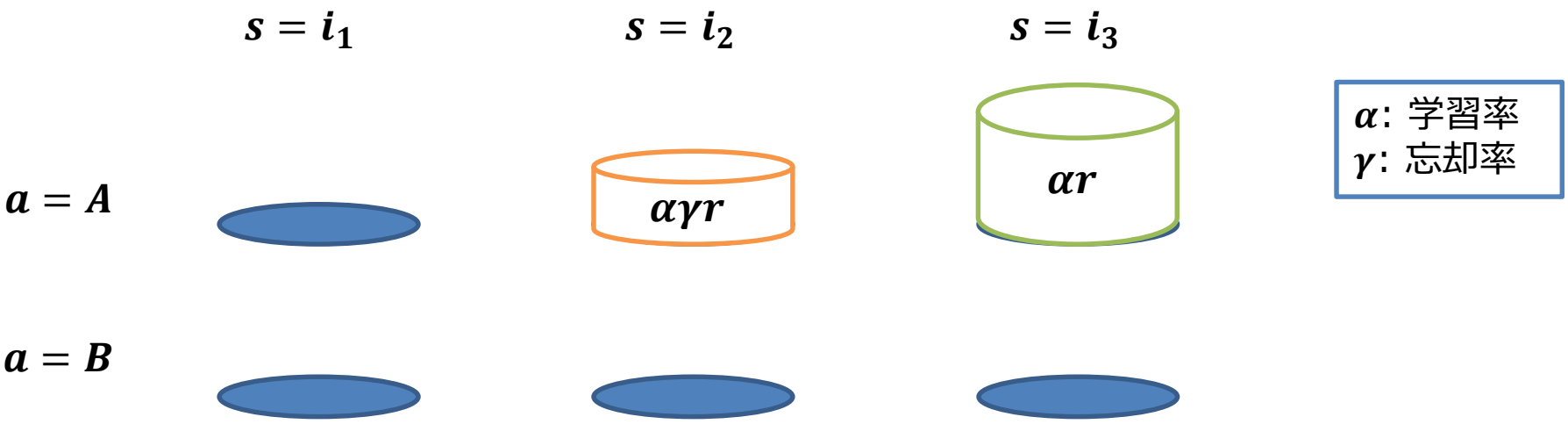
2. **ランダムに行動した結果 $s = i_3, a = A$の時に報酬 r を得る**

   報酬は学習率 α をかけて$Q(s = i_3, a = A)$ に積まれる

3. **$s = i_2$ の時それぞれの行動に対し次のターンのQ値の大きさを見る**

   $\max_{a^{t+1}} Q(s^{t+1}, a^{t+1})$ の処理（$s = i_2,\ a = A \rightarrow s' = i_3, s = i_2,\ a = B \rightarrow s' = i_1$）

Q学習の簡単な例



1. **全ての状態と行動の組み合わせにQ値を設定する**
2. **ランダムに行動した結果 $s = i_3, a = A$の時に報酬 r を得る**

   報酬は学習率 α をかけて$Q(s = i_3, a = A)$ に積まれる
3. **$s = i_2$ の時それぞれの行動に対し次のターンのQ値の大きさを見る**

   $\max_{a^{t+1}} Q(s^{t+1}, a^{t+1})$ の処理（$s = i_2,\ a = A \rightarrow s' = i_3,\ s = i_2,\ a = B \rightarrow s' = i_1$）
4. **次のターンで得られる Q値を忘却値をかけて $s = i_2, a = A$ に積む**

後半45分の内容

- 価値関数
- 価値関数の最大化（価値ベースの強化学習）
- **関数近似**
- **深層強化学習**
- **意志決定への応用**
- **文生成への応用**

Q学習の問題点

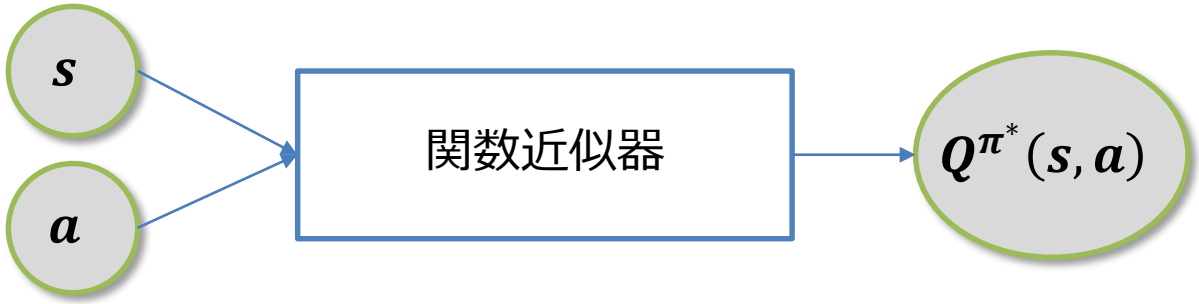
$$Q(s,a) \xleftarrow[update]{} (1-\alpha)Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a')\right)$$

- **Q学習は上式に従って無限にアップデートを繰り返せば最適行動価値関数 $Q^{\pi^*}(s,a)$ を計算可能**
  - ただし無限にサンプリングをすることは現実的でない
  - s や a の種類が多い場合全てのパターンを網羅できない

関数近似というアイディアでこの問題を解決する

関数近似（Q-network）

- **$Q^{\pi^*}(s, a)$ は状態と行動を入力して価値を出力する関数**
  - 何らかの方法で近似ができればよい



  - 関数近似器は対応が学習できれば何でもよい
  - 状態などを入力して値を出力する回帰モデルを使う
    - 教師あり学習
- **この回帰に深層学習 (Deep Learning) を使うのが Deep Q-network (DQN)**

DQN

- **DQNは以下の式に従って最適化を行う**
  - $\mathcal{L}(\theta_i) = E_{s,a,r,s'}\left[\left(y - Q(s,a)\right)^2\right]$
    学習の損失関数は予測したQ値と教師信号 y の誤差
  - $y = R(s,a,s') + \gamma \max_{a'} Q(s,a)$
    教師信号はQ学習におけるアップデート部分と同じ
    → Q学習におけるテーブルに相当する回帰モデルを学習

回帰モデルによってサンプルから良いモデルを学習可能
しかしサンプリングが必要という問題は残る

Q学習の疑似コード（ε-greedy）

$Q(s,a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)

for i in range(試行数)

　　s^t を観測

　　if rand() < ε

　　　　$\max_{a_m^t} Q(s^t,a^t)$で決まる行動 a^t を取る

　　else

　　　　ランダムに行動 a^t を決定

　　次の s^{t+1} を決定

　　報酬 $R\left(s^t,a^t,s^{t+1}\right)$ を受け取る

　　$Q(s^t,a^t)$

　　$\underset{update}{\longleftarrow} (1-\alpha)Q(s^t,a^t)+\alpha\left(R\left(s^t,a^t,s^{t+1}\right)+\gamma\max_{a^{t+1}} Q(s^{t+1},a^{t+1})\right)$

end

Q学習をDQNで行う場合 (pytorch): 0

$Q(s,a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)

for i in range(試行数)

s^t を観測

if rand() < ε

$\max_{a_m^t} Q(s^t, a^t)$で決まる行

else

ランダムに行動 a^t を決定

次の s^{t+1} を決定

報酬 $R\left(s^t, a^t, s^{t+1}\right)$ を受け取る

$Q(s^t, a^t)$

$$\xleftarrow[update]{} (1-\alpha)Q(s^t, a^t) + \alpha\left(R\left(s^t, a^t, s^{t+1}\right) + \gamma \max_{a^{t+1}} Q(s^{t+1}, a^{t+1})\right)$$

end

サンプルで使うライブラリ

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
```

基本的なアルゴリズムを理解するために用いるので、ここにあるサンプルだけでQ学習は動きません。

Q学習をDQNで行う場合 (pytorch): 1

$Q(s,a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)
for i in range(試行数)
　　s^t を観測
　　if rand() < ε
　　　　$\max_{a_m^t} Q(s$
　　else
　　　　ランダム
　　次の s^{t+1} を決定
　　報酬 $R\left(s^t, a^t, s^{t+1}\right)$
　　$Q(s^t, a^t)$
　　$\xleftarrow[update]{} (1-\alpha)Q(s^t$
end

例えば pytorch などでは

```
Q = NN()
```

として、クラスNNは

```
class NN(nn.Module):
    def __init__(self):
        super(NN, self).__init__()
        self.fc1 = nn.Linear(state_num, HIDDEN)
        self.fc2 = nn.Linear(HIDDEN, action_num)
    def __call__(self, x):
        h = F.relu(self.fc1(x))
        y = F.relu(self.fc2(h))
        return y
```

とするとQテーブル替わりの関数近似器が作れる
state_num, action_num はそれぞれ状態と行動の次元数

Q学習をDQNで行う場合 (pytorch): 2

$Q(s,a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)

for i in range(試行数)

s^t を観測（または代入）

if rand() < ε

$\max_{a_m^t} Q(s$

else

ランダム

次の s^{t+1} を決定

報酬 $R\left(s^t, a^t, s^{t+1}\right)$

$Q(s^t, a^t)$

$\xleftarrow[update]{} (1-\alpha)Q(s^t$

end

状態遷移を行う環境からSをサンプルする
エピソードの最初の場合

```
state = INITIAL_STATE # (初期状態)
```

それ以外では、

```
state = env(prev_state, act) # env は環境
```

あるいは前のステップのサンプリング結果を使う場合

```
state = next_state # next_state は最後で定義
```

Q学習をDQNで行う場合 (pytorch): 3

$Q(s,a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)

for i in range(試行数)

s^t を観測

if rand() < ε

$\max_{a_m^t} Q(s^t, a^t)$で決まる行動 a^t を取る

ランダムネスを入れて局所解に落ちないように
行動を現在の行動価値関数から決定
Q()は入力すると各行動に対応するQ値を出力する

```
state_prob = np.array(state, dtype="float32").reshape((1, state_num))
state_prob = Variable(torch.from_numpy(state_prob))
max, id = torch.max(Q(state_prob.data, 1), 1)
action = id.numpy()[0]
```

1,2行目で状態を torch.nn に入力可能なように変換し 3行目で入力、最大の行動価値関数を持つものを探す

Q学習をDQNで行う場合 (pytorch): 4

$Q(s, a)$ の全組み合わせを初期化（0を代入）

set ε (0<ε<1)

for i in range(試行数)

s^t を観測

ランダムに行動を選択

```
action = random.choice(action_list)
```

動 a^t を取る

else

ランダムに行動 a^t を決定

次の s^{t+1} を決定

報酬 $R\left(s^t, a^t, s^{t+1}\right)$ を受け取る

$Q(s^t, a^t)$

$$\xleftarrow[update]{} (1-\alpha)Q(s^t, a^t) + \alpha\left(R\left(s^t, a^t, s^{t+1}\right) + \gamma \max_{a^{t+1}} Q(s^{t+1}, a^{t+1})\right)$$

end

Q学習をDQNで行う場合 (pytorch): 5

$Q(s,$

set ε

for i

次の状態は環境から得る

```
next_state = env(state, act) # env は環境
```

報酬も環境から得る

```
reward = reward_function(state, action, next_state)
```

a_m

else

ランダムに行動 a^t を決定

次の s^{t+1} を決定

報酬 $R(s^t, a^t, s^{t+1})$ を受け取る

$Q(s^t, a^t)$

$$\xleftarrow[update]{} (1-\alpha)Q(s^t,a^t) + \alpha\left(R(s^t,a^t,s^{t+1}) + \gamma \max_{a^{t+1}} Q(s^{t+1},a^{t+1})\right)$$

end

Q学習をDQNで行う場合 (pytorch): 6

パラメータの更新を行うため、次の状態に対する最適行動を見つける

```
next_state_prob = np.array(state, dtype="float32").reshape((1, state_num))
next_state_prob = Variable(torch.from_numpy(state_prob))
max, id = torch.max(Q(next_state_prob.data, 1), 1) # max に最大のQ値が入る
```

報酬とあわせて以下の教師信号 target に対して学習を行う

```
target = reward + gamma * max
optimizer.zero_grad()
loss = nn.MSELoss()(Q(state_prob), Variable(torch.from_numpy(target)))
loss.backward
optimizer.step()
```

$$Q(s^t, a^t)$$

$$\xleftarrow[update]{} (1-\alpha)Q(s^t, a^t) + \alpha\left(R(s^t, a^t, s^{t+1}) + \gamma \max_{a^{t+1}} Q(s^{t+1}, a^{t+1})\right)$$

end

Q学習をDQNで行う場合 (pytorch): 7

最後に次のステップのために状態、行動を代入する

```
state = next_state # Q値計算のための次の状態を使いまわす場合
```

s^t を観測

if rand() < ε

$\max_{a_m^t} Q(s^t, a^t)$で決まる行動 a^t を取る

else

ランダムに行動 a^t を決定

次の s^{t+1} を決定

報酬 $R\left(s^t, a^t, s^{t+1}\right)$ を受け取る

$Q(s^t, a^t)$

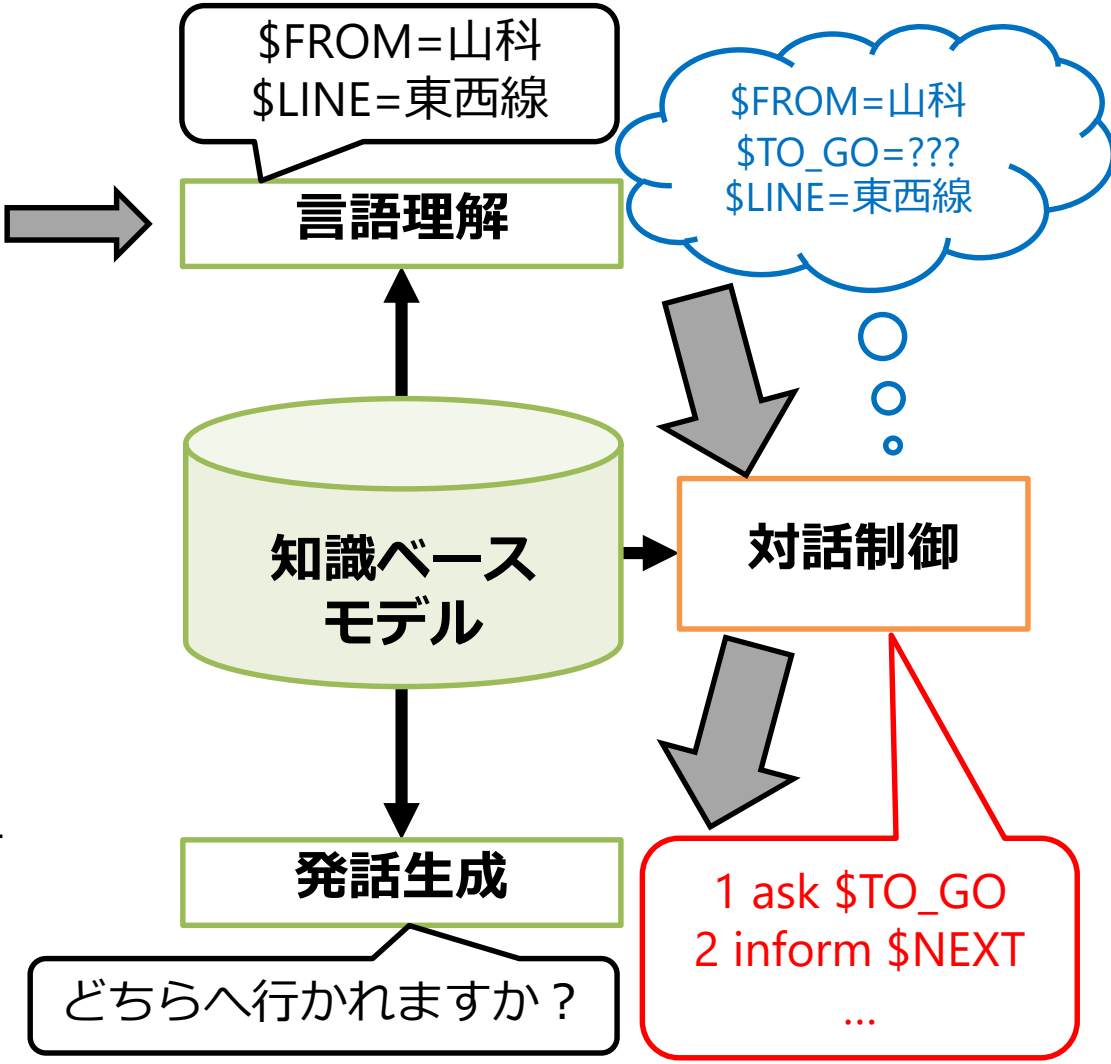
$$\xleftarrow[update]{} (1-\alpha)Q(s^t, a^t) + \alpha\left(R\left(s^t, a^t, s^{t+1}\right) + \gamma \max_{a^{t+1}} Q(s^{t+1}, a^{t+1})\right)$$

end

意志決定への応用

山科から東西線に乗りたいんだけど

- **言語理解を通じてユーザ発話を対話状態に変換**
  - 詳細は音声言語処理2
- **対話状態に応じて次のシステムの行動を決定**
  - システムの行動に応じて発話を生成



意志決定への応用

山科から東西線に乗りたいんだけど

$FROM=山科
$LINE=東西線

言語理解

$FROM=山科
$TO_GO=???
$LINE=東西線

対話制御

1 ask $TO_GO
2 inform $NEXT
…

どちらへ行かれますか？

- **現在の言語理解結果を利用して次にシステムが行うべき行動を決定する**
  - 状態=ユーザが何を行ったか
  - 行動=システムがどう返すべきか
  - 理解内容が足りない場合確認

バス案内対話システム

- **京都市バスのサービスとして実証実験 (2002—2003)**
  - 電話に掛けると自動応答システムが次のバスを教えてくれる
- **Let's Go バス案内システム (2005)**
  - ルールベースだった京都市バスシステムを米国・ピッツバーグで再開発し強化学習を適用したもの
  - この枠組みは Cambridge のレストラン案内システム、San Francisco のホテル案内システムなど様々な音声対話システムで利用・検証されている
  - 米国、中国を筆頭として様々な機関が実証実験を試みている

事態間関係を用いた論証対話

- **因果関係を論証に利用（合理的な論証を構築可能）**

被告人は車を運転していましたか？/
被告人の呼気中からアルコールは検出されましたか？

情報探索対話 (質問と応答)

はい

「被告人は車を運転していた」かつ
「被告人の呼気中からアルコールが検出された」
ので**「被告人は飲酒運転をしていた」**

合理的な論証の構築

- **深層強化学習を用いた質問の選択**
  - 効率的な論証構築のためにどのような質問を行うか

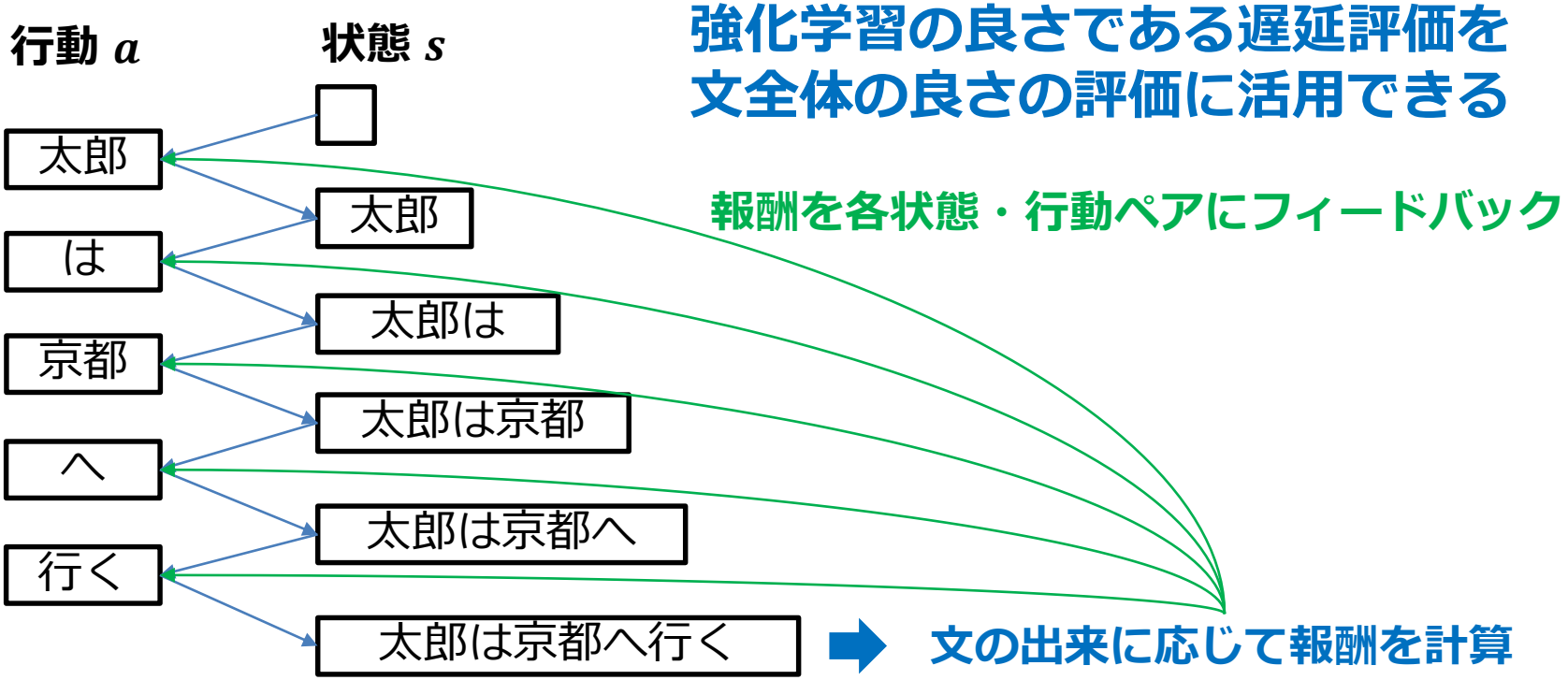
深層強化学習を用いた論証構築の効率化

- **既存の事実収集戦略よりも効率的に事実を収集可能**
  - 殺人の疑いをかけられた少年が無実である根拠を収集する対話
  - 事実を収集する対話の過程をマルコフ決定過程でモデル化

turn	話者	発話	合理性
1	質問者 回答者	通りを横切った女性は殺人事件を目撃していないのですか？ はい、その女性は殺人事件を目撃していません。	0.1
2	質問者 回答者	通りにいた老人は少年が「殺してやる」と言ったのを聞いていないのですか？ はい、その老人は少年が「殺してやる」と言ったのは聞いていません。	0.4
3	質問者 回答者	その老人は嘘をついていますか？ わかりません。	0.4
4	質問者 回答者	では少年は中腰になって背の高い男性を刺していないのでしょうか？ わかりません。	0.4
5	質問者 回答者	通りを横切った女性は少年が父を刺すのを目撃していないのですね？ わかりません。	0.4
6	質問者 回答者	では少年は凶器となったナイフを購入しましたか？ いいえ、少年は凶器となったナイフを購入していません。	0.7

言語生成への応用

- **言語の生成過程をマルコフ決定過程と見なす**
  - 行動は各単語の生成
  - 状態は単語系列で出来上がった文



後半部分のまとめ

- **強化学習の目的は将来的な期待報酬の最大化**
  - 将来の報酬の遅延評価を行うことができる
- **将来的な累積期待報酬は価値関数として定式化**
  - 価値関数あるいは行動価値関数を最大化する問題
- **行動価値関数は無限のサンプルで最適なものが求まる**
  - Q学習によって求めることができる
- **この効率化のために関数近似というテクニックを使う**
  - 状態が連続値の場合や膨大な場合はサンプリングが厳しい
  - 関数近似によって深層学習を適用することができる
- **様々な応用がある**
  - 対話システム（チャットボット）応用については**音声言語処理2**で

お勧めの教科書


これからの
強化学習
編著 牧野貴樹 澁谷長史 白川真一
著 浅田 稔 麻生英樹 荒井幸代 飯間 等
伊藤 真 大倉和博 黒江康明 杉本徳和
坪井祐太 銅谷賢治 前田新一 松井藤五郎
南 泰浩 宮崎和光 目黒豊美 森村哲郎
森本 淳 保田俊行 吉本潤一郎
森北出版


Pythonで学ぶ
強化学習 [改訂第2版]
入門から実践まで
Reinforcement Learning
by Python
久保隆宏
Machine Learning
MLS
機械学習
スタートアップ
シリーズ