

ゼロから作る
Deep Learning 4
機械学習編
1章 バンディット問題

O'REILLY®
オライリー・ジャパン

ゼロから作る

Deep
Learning 4

強化学習編



斎藤 康毅 著

23/11/16(木)

バンディット問題 Bandit Problem

Bandit(盗賊, 山賊) : スロットマシン

- レバーを引くとランダムに絵柄が変わる
- 絵柄によりコインが(0~n)枚もらえる



多腕バンディット問題 Multi-armed Bandit Problem

問題設定

- 1 本レバースロットマシンが複数台ある
- スロットマシン毎に絵柄の出方が異なる
- プレイヤーはスロットマシンの情報を何も知らない

(例えば)1000回
プレイ後の得た
コインの枚数を
最大化したい



多腕バンディット問題 Multi-armed Bandit Problem

用語

環境 Environment : スロットマシン

エージェント Agent : プレイヤー

行動 Auction : 1 台選んでプレイする

報酬 Reward : スロットマシンから出るコイン


多腕バンディット問題 Multi-armed Bandit Problem

良いスロットマシンとは？

- ・スロットマシン毎にもらえるコインの確率分布が異なる


良いスロットマシン \Leftrightarrow 期待値が高い

期待値の高いスロットマシンを毎回
選ばばよい



SM_a

coin	0	1	5	10
P	0.7	0.15	0.12	0.03



SM_b


coin	0	1	5	10
P	0.5	0.4	0.09	0.01

多腕バンディット問題 Multi-armed Bandit Problem

良いスロットマシンとは？

- ・スロットマシン毎にもらえるコインの確率分布が異なる

$$SM_a: \mu_a = 0 \times 0.7 + 1 \times 0.15 + 5 \times 0.12 + 10 \times 0.03 = 1.05$$




SM_a

coin	0	1	5	10
P	0.7	0.15	0.12	0.03

良いスロットマシン \Leftrightarrow 期待値が高い

期待値の高いスロットマシンを毎回
選ばばよい

$$SM_b: \mu_b = 0 \times 0.5 + 1 \times 0.4 + 5 \times 0.09 + 10 \times 0.01 = 0.95$$



SM_b


coin	0	1	5	10
P	0.5	0.4	0.09	0.01

多腕バンディット問題 Multi-armed Bandit Problem

良いスロットマシンとは？

- ・スロットマシン毎にもらえるコインの確率分布が異なる

プレイヤーは知らない




SM_a

coin	0	1	5	10
P	0.7	0.15	0.12	0.03

良いスロットマシン \Leftrightarrow 期待値が高い

- ・期待値の高いスロットマシンを毎回選ばばよい



SM_b

coin	0	1	5	10
P	0.5	0.4	0.09	0.01

多腕バンディット問題 Multi-armed Bandit Problem



SM_a

coin	0	1	5	10
P	0.7	0.15	0.12	0.03



SM_b

coin	0	1	5	10
P	0.5	0.4	0.09	0.01

報酬	Reward	: スロットマシンから出るコイン
価値	Value	: 報酬の期待値
行動価値	Action Value	: 行動に対して得られる報酬の期待値
	$R_t \in \{0, 1, 5, 10\}$: t回目 to 得られる報酬
	$A_t \in \{a, b\}$: t回目の行動 (SMの種類)

数式 記号

式	式の意味
$E[R]$	報酬Rの期待値
$E[R A]$	Aという行動をした場合のRの期待値
$E[R A = a] = E[R a]$	aという行動をした場合のRの期待値
$q(A) = E[R A]$	行動価値:Quality, 行動Aの価値
$q(A)$	真の値←エージェントは知らない
$Q(A)$	推定値

行動価値推定アルゴリズム

エージェントはスロットマシンの価値を知らない
→各スロットマシンの価値を推定する必要がある

SM	1回目	SMの価値
a	0	0
b	1	1

SM	1回目	2回目	3回目	SMの価値
a	0	1	5	2
b	1	0	0	0.333

$$Q(A = a) = 0, Q(A = b) = 1$$

$$Q(A = a) = 2, Q(A = b) = 0.333$$

この期待値 $Q(A)$ は推定値だけど SM_a の方が価値が高そう

↑ 標本平均：大数の法則より無限回のサンプリングで真の値に一致する

実装の注意点① 平均値の実装

R_1, R_2, \dots, R_n から Q_n を求める.

単に平均すると $n+1$ 回目の行動で R_{n+1} を得たとき $R_1 \sim R_{n+1}$ を再度 必要としてしまう.

→ 漸化的に求める

$$Q_{n-1} = \frac{R_1 + \dots + R_{n-1}}{n-1}$$

$$R_1 + \dots + R_{n-1} = (n-1) Q_{n-1}$$

$$Q_n = \frac{1}{n} (R_1 + \dots + R_{n-1} + R_n)$$

$$= \frac{1}{n} ((n-1) Q_{n-1} + R_n)$$

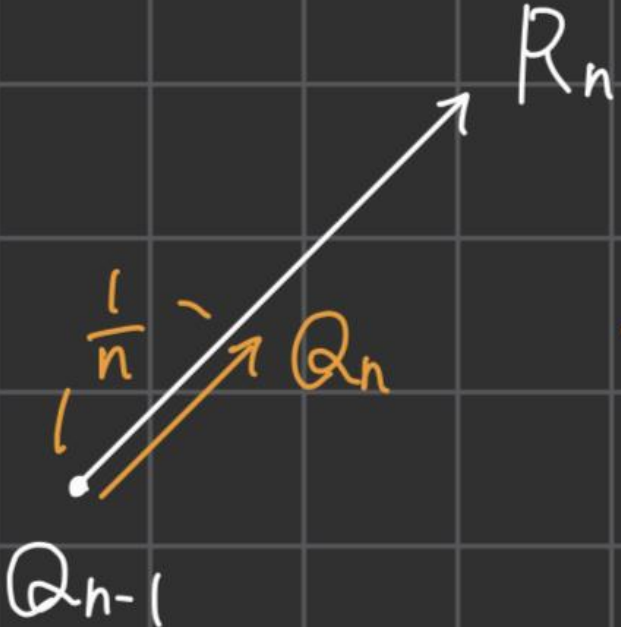
$$= \left(1 - \frac{1}{n}\right) Q_{n-1} + \frac{1}{n} R_n$$

$$= Q_{n-1} + \frac{1}{n} (R_n - Q_{n-1})$$

$$Q_n = Q_{n-1} + \frac{1}{n} (R_n - Q_{n-1})$$

α 足りる時間も小さい

実装の注意点① 平均値の実装

$$Q_n = Q_{n-1} + \frac{1}{n} (R_n - Q_{n-1})$$


$\frac{1}{n}$: 学習率としての役割

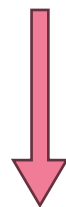
プレイヤーの戦略

- greedy(貪欲) 各SMの価値の推定値が最大のものを常に選ぶ
実験が少なく推定値の大小と
真の値の大小が一致しないかも
- 実験 SMの価値を精度よく推定するため
価値が低いものでも様々なSMを選択する

プレイヤーの戦略

活用 Exploitation

経験から最善な行動をする
(greedy)



真の最善をみのがしているかもしれないから

探索 Exploration

Greedyでない行動を試す



プレイヤーの戦略 ϵ -greedy法

強化学習は活用と探索のバランスを
以下に取るのが難しい

そこで, ϵ -greedy法

確率 ϵ で探索 (ランダムな行動)

確率 $1-\epsilon$ で活用



多腕バンディット問題 実験①

問題設定①

- 1 本レバースロットマシンが10台ある
- スロットマシン毎に絵柄の出方が異なる
- プレイヤーはスロットマシンの情報を何も知らない
- 1000回プレイ後の得たコインの枚数を最大化したい
- $\epsilon=0.1$ で探索 (ランダム)



多腕バンディット問題 実験①

問題設定①

- ・スロットマシン i は確率 $reta[i]$ でコインを1枚返し, それ以外で0枚返す
- ・ $reta[i]$ 自体も初めにランダムに決めておく



多腕バンディット問題 実装①

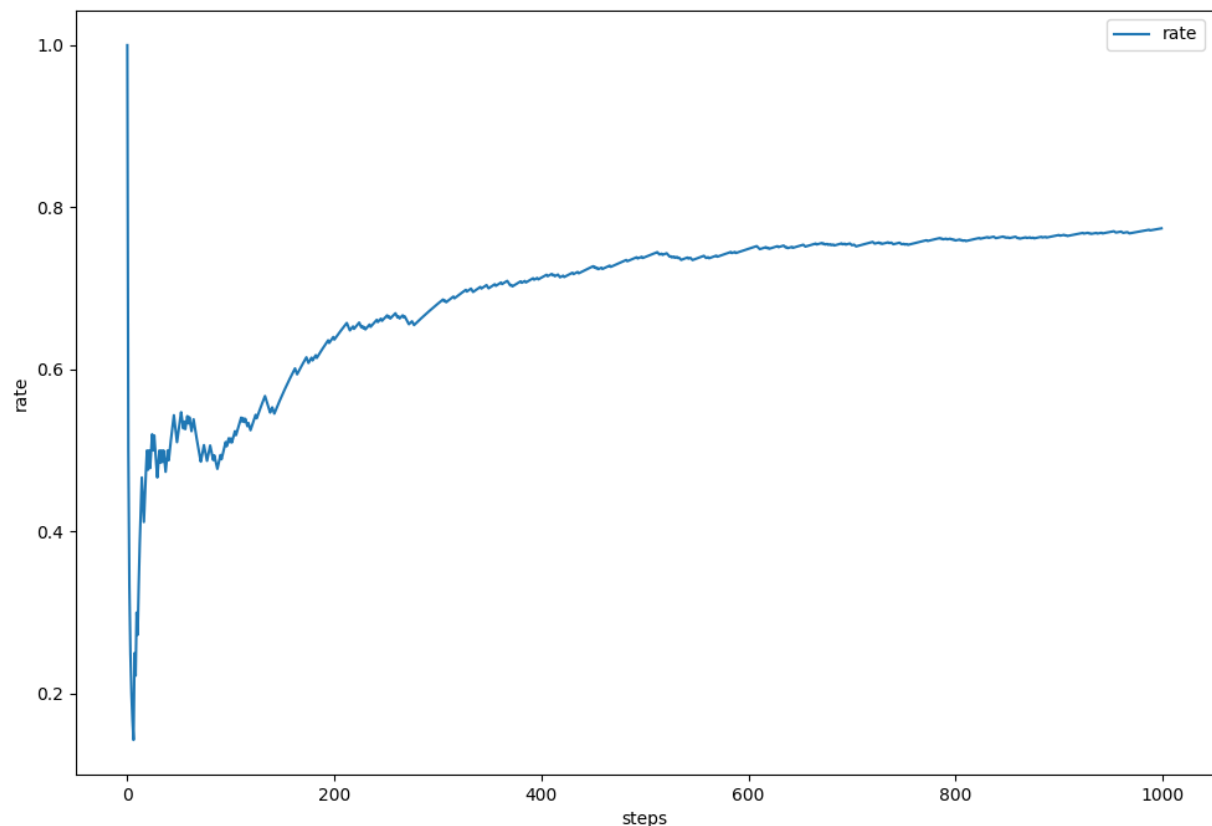
<https://github.com/cijb-7724/deep-learning-4-in-cpp/tree/main/ch01>

ch01/ch01_fig13_bandit.cpp

ch01/make_graph.py

多腕バンディット問題 実験①

結果①

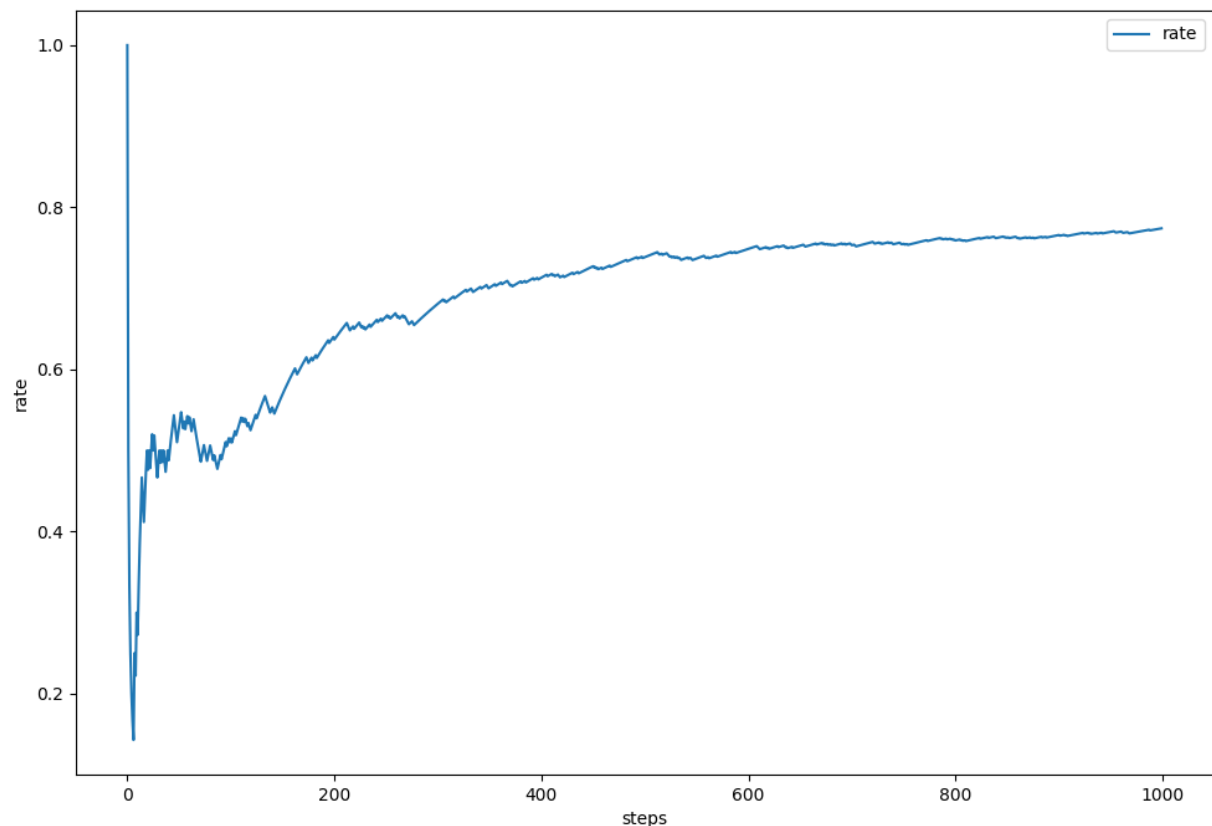


rate=0.77くらいに収束している

初めに決めたrate[i]の最大値が
0.77くらいでそこに収束してると考
えられる

多腕バンディット問題 実験①

結果①



初めに決めた $\text{rate}[i]$ の最大値が0.77くらいでそこに収束してると考えられる

```
arms 10
i = 0 0.592845
i = 1 0.844266
i = 2 0.857946
i = 3 0.847252
i = 4 0.623564
i = 5 0.384382
i = 6 0.297535
i = 7 0.056713
i = 8 0.272656
i = 9 0.477665
```

実際は0.85だから
収束しきっていない

多腕バンディット問題 実験② 平均的な性質

問題設定②

①では各SMの勝率（コインを1枚もらえる確率）は初めに1回ランダムに決めただけだった

②では①自体を200回実験し、その平均での各ステップでの勝率を計算する

```
arms 10
i = 0 0.592845
i = 1 0.844266
i = 2 0.857946
i = 3 0.847252
i = 4 0.623564
i = 5 0.384382
i = 6 0.297535
i = 7 0.056713
i = 8 0.272656
i = 9 0.477665
```

多腕バンディット問題 実験② 平均的な性質

問題設定②

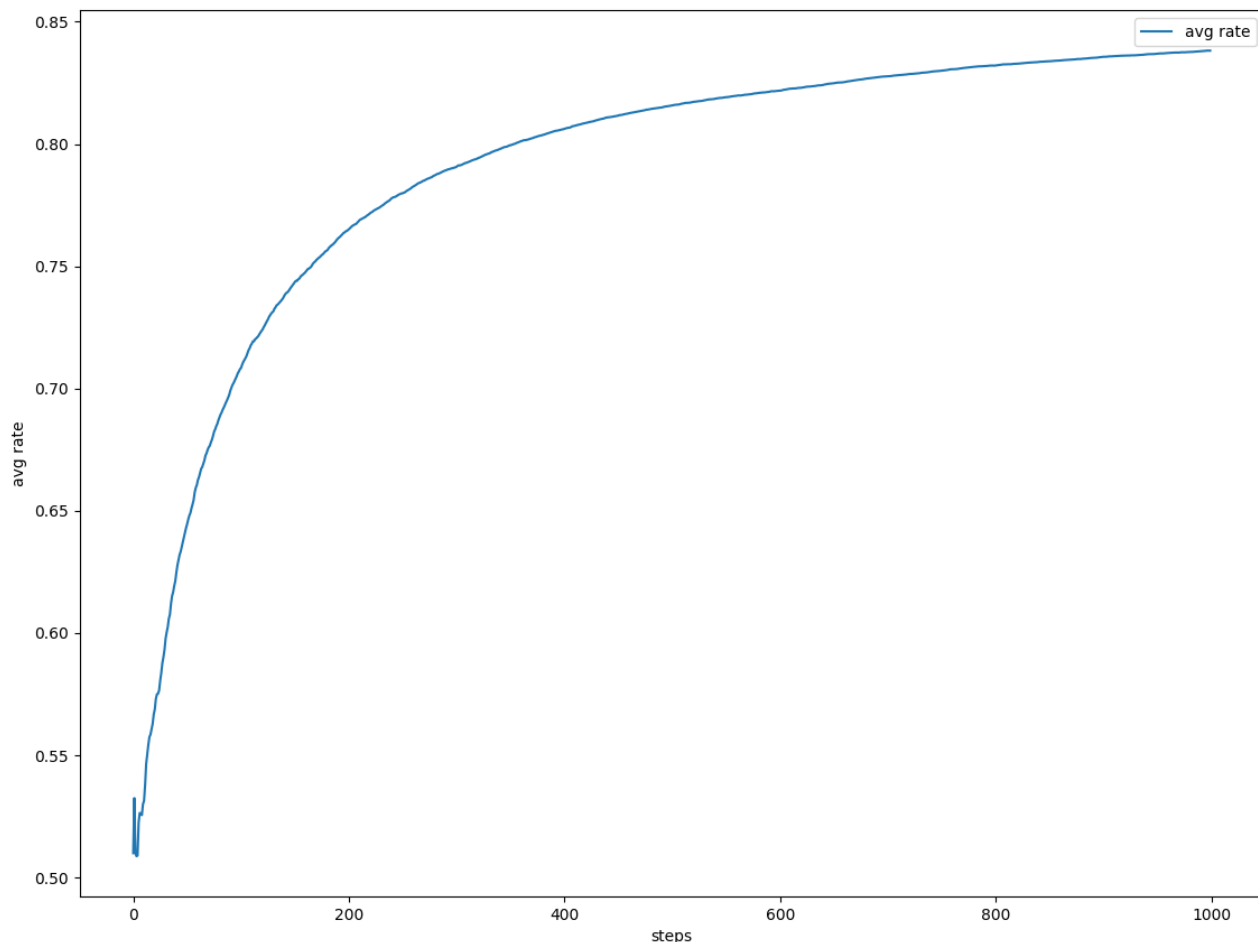
②では①自体を200回実験し、
その平均での各ステップでの勝率を計算する

	1	2	3	...	1000
1 回目の実験	1.0	0.5	0.333	...	0.913
2 回目の実験	0.0	0.0	0.0	...	0.821
...
200 回目の実験	1.0	1.0	1.0	...	0.615
平均	0.493	0.497	0.504	...	0.838

```
arms 10
i = 0 0.592845
i = 1 0.844266
i = 2 0.857946
i = 3 0.847252
i = 4 0.623564
i = 5 0.384382
i = 6 0.297535
i = 7 0.056713
i = 8 0.272656
i = 9 0.477665
```

多腕バンディット問題 実験② 平均的な性質

結果②



ステップを重ねるたびに急速に
勝率が上がってる
(600stepほどで頭打ち)

多腕バンディット問題 実験③ ϵ の比較

問題設定③

②の平均的な勝率は $\epsilon=0.1$ であった.

ϵ を変化させたとき勝率の収束の仕方を考える

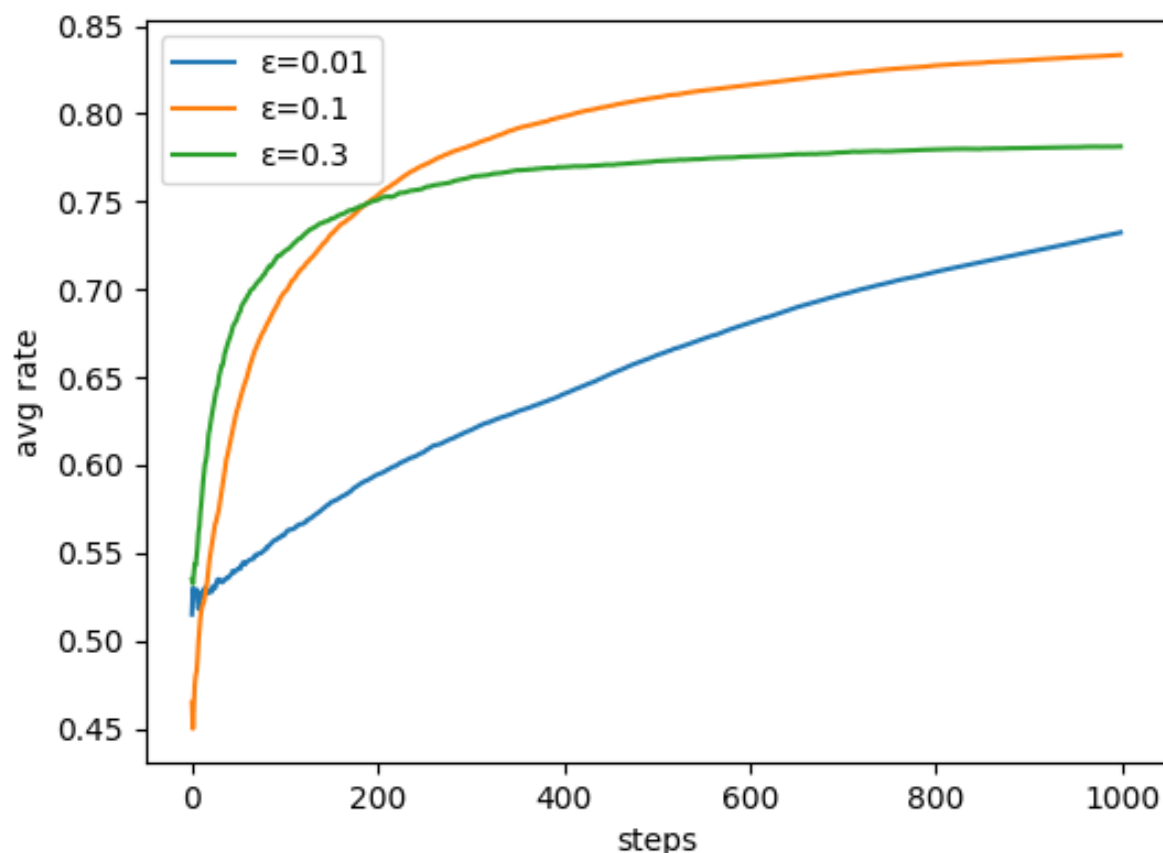
$\epsilon=0.01, 0.1, 0.3$

の3つを試す

その他の設定は①と同じ

多腕バンディット問題 実験③ ϵ の比較

結果③



$\epsilon=0.01$: 探索のしなさすぎて適切な行動を選択できていない

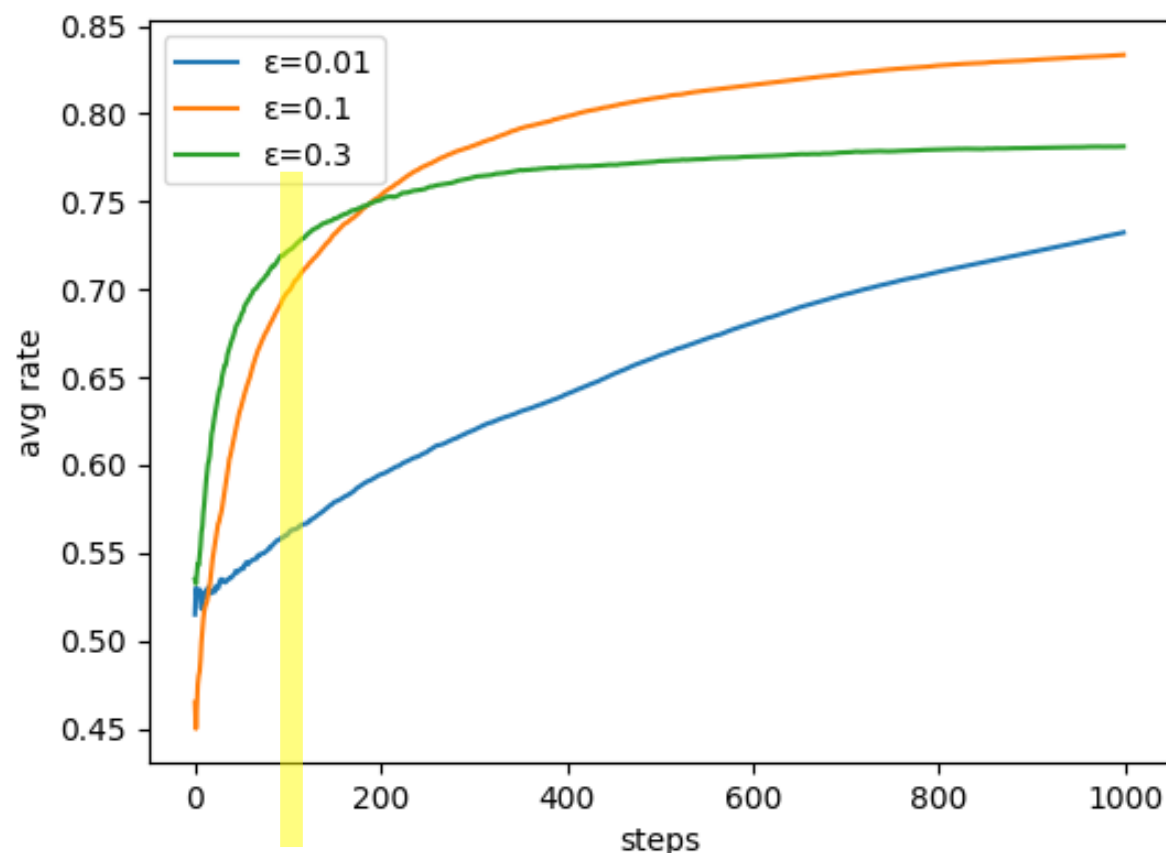
$\epsilon=0.3$: 探索のしすぎで適切な行動をとり続けていない

$\epsilon=0.1$: ちょうどよく見える

しかし！！

多腕バンディット問題 実験③ ϵ の比較

結果③



1000回プレイできるとき $\epsilon=0.1$ が適切だとわかったが,
100回しかプレイできないとき
 $\epsilon=0.3$ が適切とも捉えられる

∴問題設定によって ϵ -greedy法の
 ϵ を変化させる必要がある

非定常問題

定常問題: 報酬の確率分布が定常

1000回のプレイが終わるまでは
スロットマシンの勝率は変化しなかった

非定常問題: 報酬の確率分布が非定常

毎プレイで勝率が変化する

非定常問題

非定常問題: 報酬の確率分布が非定常
毎プレイで勝率が変化する

```
int NonStatBandit::play(int arm) {  
    double rate = this->rates[arm];  
    for (int i=0; i<this->arms; ++i) this->rates[i] += 0.1 * rand_double(-1, 1); //ノイズを追加  
    if (rate > rand_double(0, 1)) return 1; //rewardの値を返す  
    else return 0;  
}
```

非定常問題

Q_n : 価値, R_n : 報酬

$$Q_n = \frac{R_1 + \dots + R_n}{n} = \frac{1}{n} R_1 + \frac{1}{n} R_2 + \dots + \frac{1}{n} R_n$$

↖ 重み

定常

$$Q_n = Q_{n+1} + \frac{1}{n} (R_n - Q_{n-1})$$

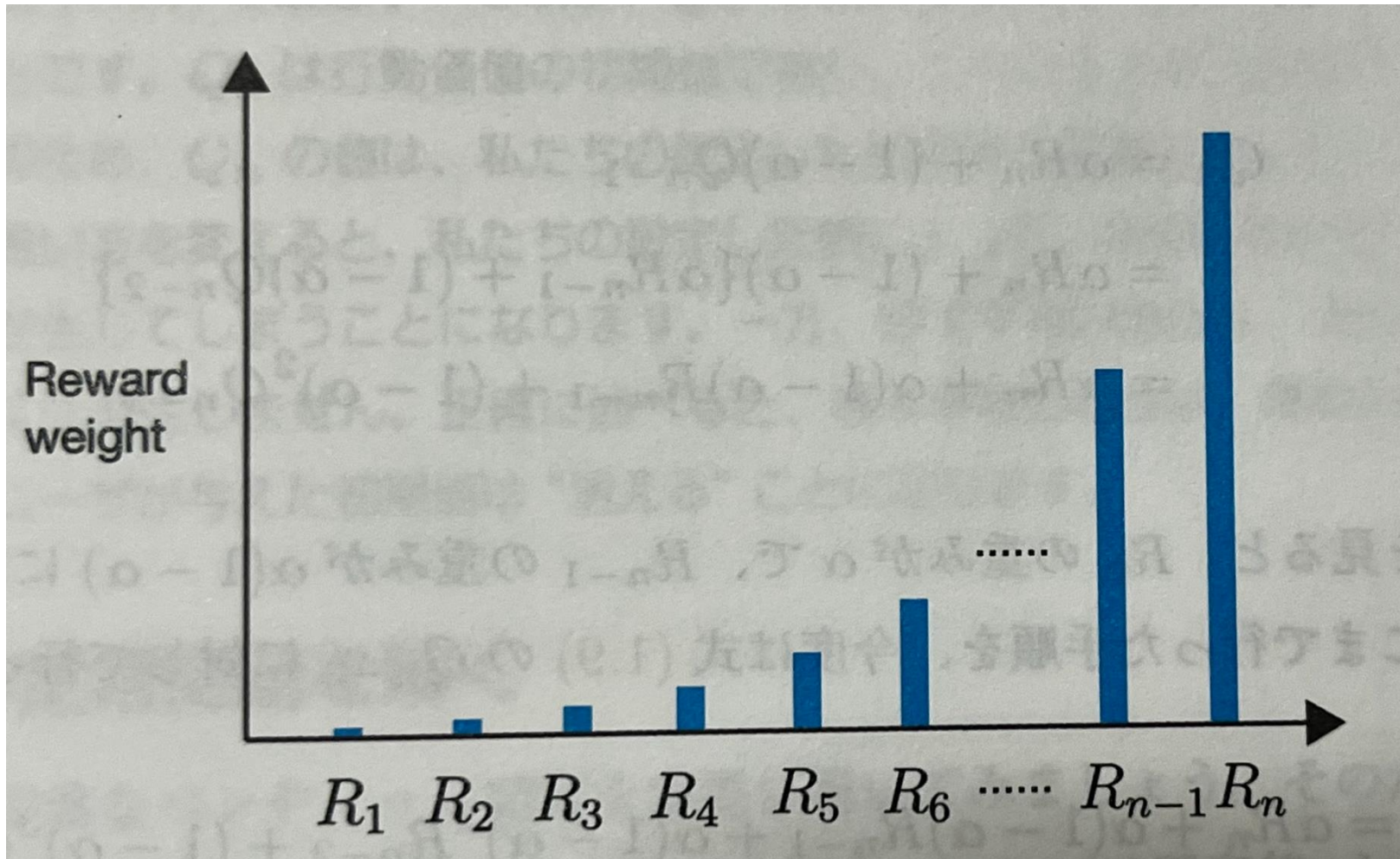
非定常

$$Q_n = Q_{n+1} + \alpha (R_n - Q_{n-1})$$

$$Q_n = \alpha R_n + \alpha(1-\alpha)R_{n-1} + \alpha(1-\alpha)^2 R_{n-2} + \dots + \alpha(1-\alpha)^{n-1} R_1 + (1-\alpha)^n Q_n$$

$R_1 \sim R_n$ の 指数 (加重) 移動平均

非定常問題



多腕バンディット問題 実験④ 非定常問題

問題設定④

②の問題設定において、毎プレイで勝率が変化する

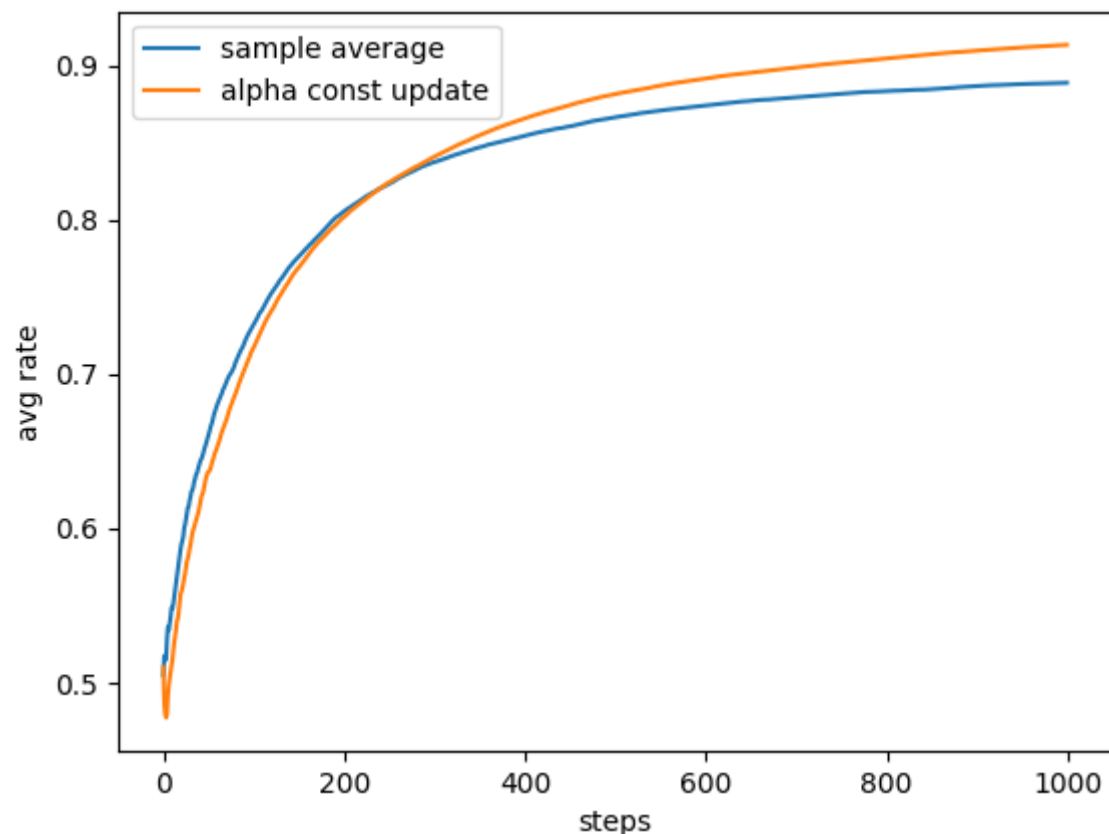
```
int NonStatBandit::play(int arm) {  
    double rate = this->rates[arm];  
    for (int i=0; i<this->arms; ++i) this->rates[i] += 0.1 * rand_double(-1, 1); //ノイズを追加  
    if (rate > rand_double(0, 1)) return 1; //rewardの値を返す  
    else return 0;  
}
```

このときエージェント目線での各スロットマシンの価値Qを以下の2つの更新式で実装し違いを観察する

定常	$Q_n = Q_{n+1} + \frac{1}{n} (R_n - Q_{n-1})$	$\alpha = 0.8$
非定常	$Q_n = Q_{n+1} + \alpha (R_n - Q_{n-1})$	

多腕バンディット問題 実験④ 非定常問題

結果④



直近の報酬を重要視して
過去の報酬を（ほぼ）無視する
ことで、非定常問題にも対応する
ことができた