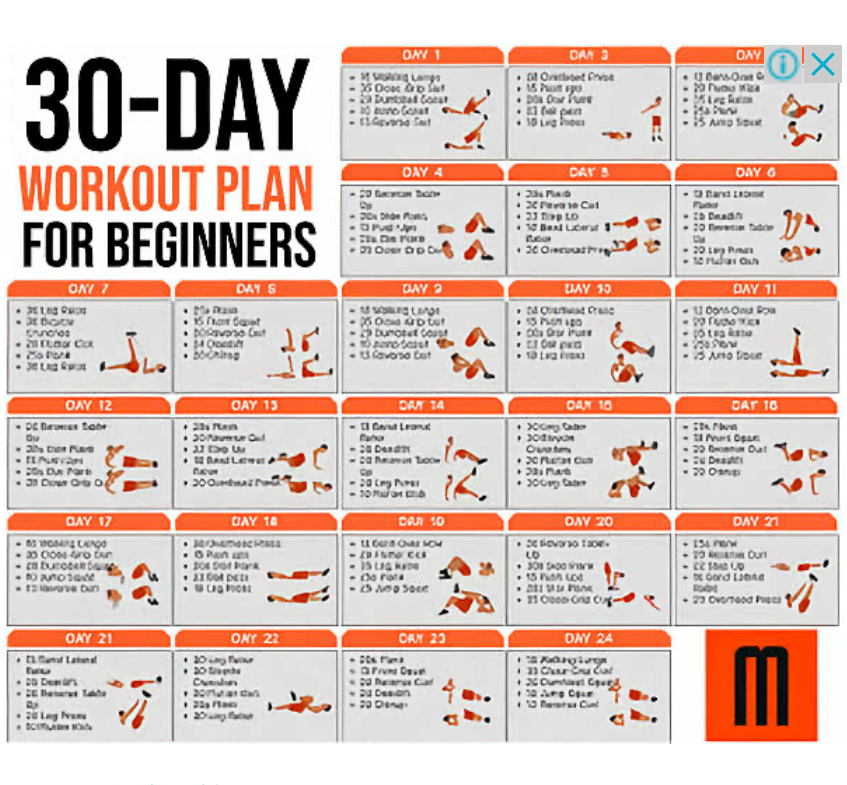
化学系エンジニアがAIを学ぶ

PyTorchでディープラーニング、強化学習を学び、主に化学工学の問題に取り組みます



05
04
2019

深層強化学習(Deep Q Network, DQN)の簡単な例

ディープラーニング 強化学習 PyTorch

はじめに

DQNを学ぼうとして色々と調べたが、どこもかしこもOpenAI Gymを使っていて、まずそれの扱いから考えないといけないのでつらい。ここではOpenAI Gymを使わず、非常に簡単な問題を対象にDQNを適用してみることとする。Python、PyTorchを用いる。

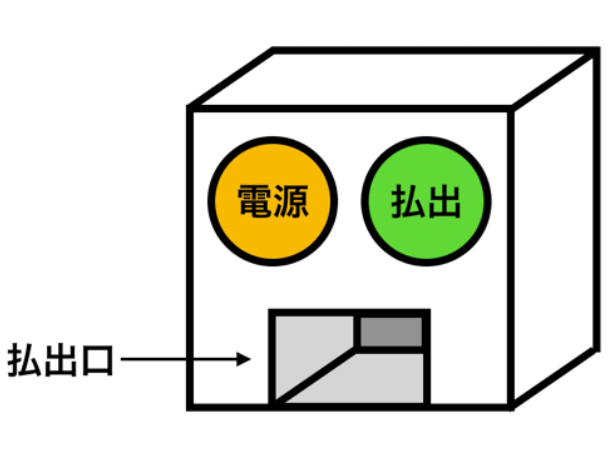
また、DQNをやろうとすると出てくる"Experience Replay"。これも入れようとすると考えるのがつらいので、入れない。

下記記事にDQN全般やExperience Replayについてちょっと解説がある。

Deep Q Network (DQN) - DeepLearningを勉強する人

対象とする問題

下図のような報酬払出装置を考える。装置には「電源」ボタンと「払出」ボタンが設置されている。「電源」ボタンを押すと装置電源のON/OFFが切り替わる。電源ONのときに「払出」ボタンを押すと報酬が払い出される。電源OFFのときに「払出」ボタンを押しても報酬は払い出されない。



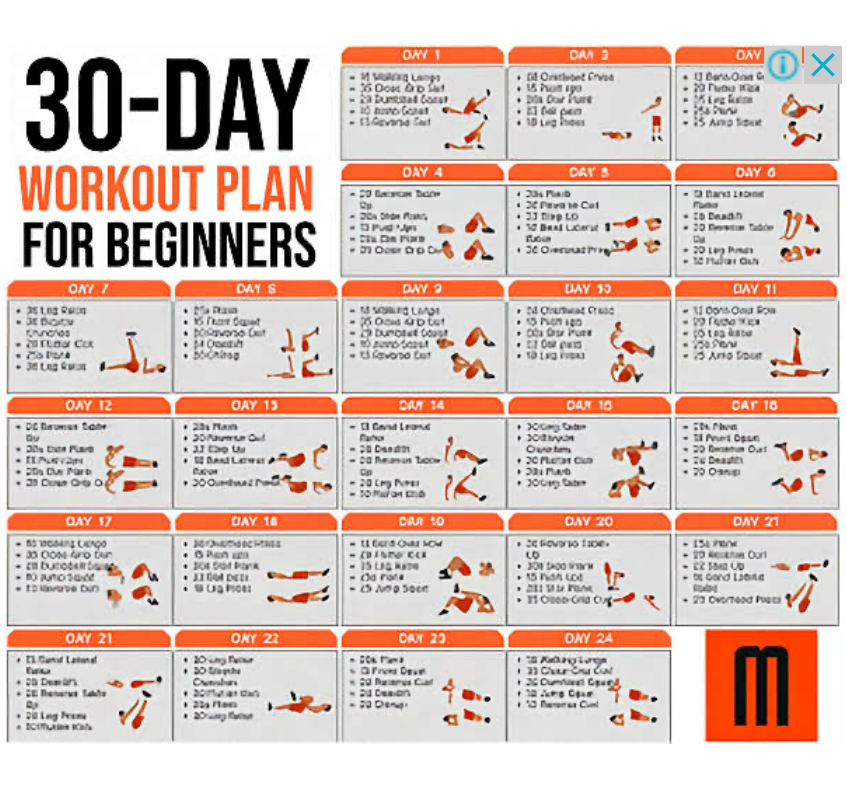
1回の行動で、いずれかのボタンを1度押すことができるものとする。5回の行動を行う場合に、報酬を最大化する手順を学習することができるか？

なお、この問題の内容に関して下記書籍を参考にした。

- 牧野浩二、西崎博光，『Pythonによる深層強化学習入門』, オーム社 (2018)

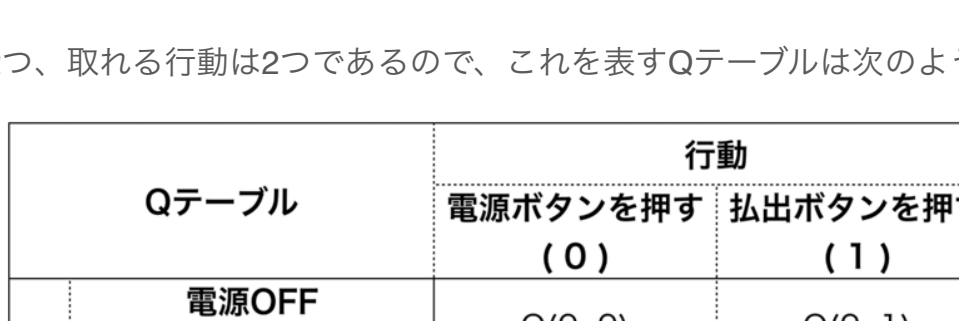
この払出装置(Dispenser)をプログラムにすると次の通りとなる。

```
class Dispenser(object):
    def __init__(self, init_state):
        """
        初期のON/OFF状態を設定する
        init_state: 0->電源OFF、1->電源ON
        """
        self.state  = init_state

    def powerbutton(self):
        """
        電源ボタンを押し、ON/OFFを切り替える
        """
        if self.state == 0:
            self.state = 1
        else:
            self.state = 0

    def step(self, action):
        """
        払出機を操作する
        action: 0->電源ボタンを押す 1->払出ボタンを押す
        状態と報酬が返る
        """
        if action == 0: # 電源ボタンを押した場合
            self.powerbutton() # 電源ON/OFF切り替え
            reward = 0 # 報酬はない
        else:           # 払出ボタンを押した場合
            if self.state == 1:
                reward = 1 # 電源ONのときのみ報酬あり
            else:
                reward = 0
        return self.state, reward
```



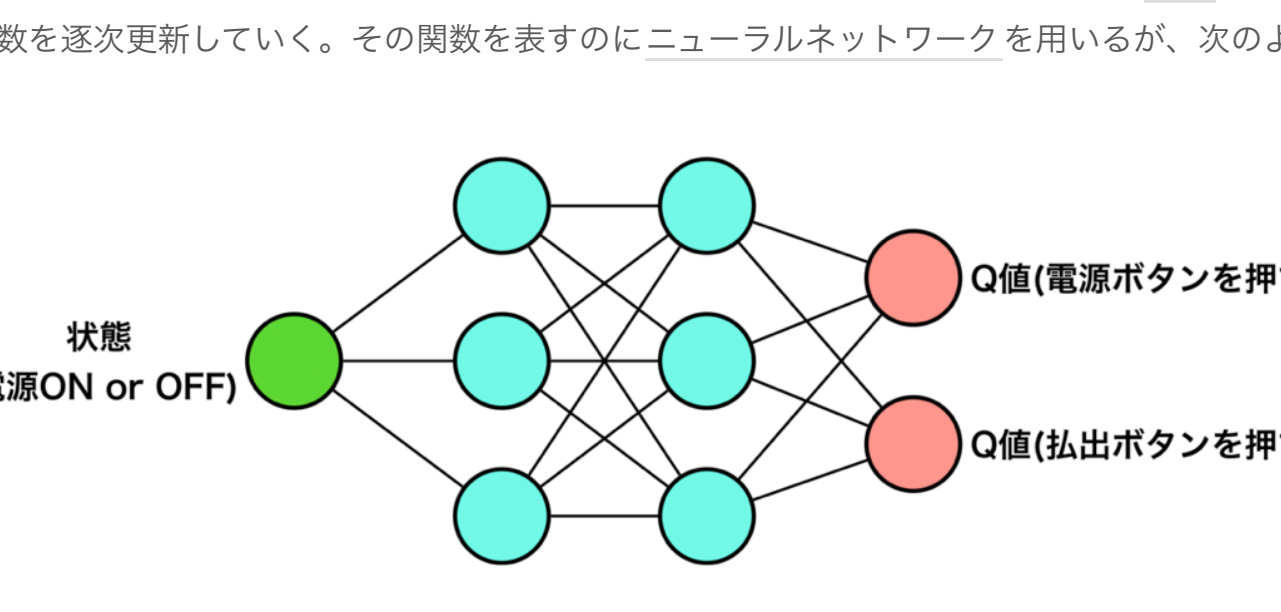
Deep Q Network (DQN)

報酬払出機の状態は2つ、取れる行動は2つであるので、これを表すQテーブルは次のようになる。



Qテーブル		行動	
		電源ボタンを押す（0）	払出ボタンを押す（1）
状態	電源OFF（0）	Q(0, 0)	Q(0, 1)
	電源ON（1）	Q(1, 0)	Q(1, 1)

Q学習ではこのQテーブルのQ値（Q(0,0), Q(0,1), Q(1,0), Q(1,1)）を逐次更新していくが、 DQNではこれらのQ値を得る関数を逐次更新していく。その関数を表すのにニューラルネットワークを用いるが、次のようなイメージとなる。



状態を入力として、Q値を出力として得る形である。ここでは上図の通りのニューラルネットワークを用いることとする。（あまりディープではないが・・・）

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class DQN(nn.Module):
    def __init__(self):
        super(DQN, self).__init__()
        self.l1 = nn.Linear(1, 3)
        self.l2 = nn.Linear(3, 3)
        self.l3 = nn.Linear(3, 2)

    def forward(self, x):
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = self.l3(x)
        return x

dqn = DQN()
optimizer = torch.optim.SGD(dqn.parameters(), lr=0.01)
criterion = nn.MSELoss()
```

DQNの学習方法

では、どのような情報を使ってこのニューラルネットワークを学習させていくのか、ということになるが、これにはQ学習のQ値の更新式を用いる。

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)]$$

この式は今のQ値（$Q(s_t, a_t)$）を、今の推定の目標値（$r_{t+1} + \gamma \max Q$）に近づくよう更新するものである。すなわち、

$$Q(s_t, a_t) \fallingdotseq r_{t+1} + \gamma \max Q(s_{t+1}, a)$$

となることを目指している。よって$Q(s_t, a_t)$と$r_{t+1} + \gamma \max Q$の誤差を小さくするようにニューラルネットワークを学習させるという形にすればよいことになる。いずれの値も現在の状態、行動、次の状態、報酬が分かれば算出可能であり、DQNの学習をプログラムにすると次のようになる。

```
def update_dqn(state, action, next_state, reward):
    ## 各変数をtensorに変換
    state = torch.FloatTensor([state])
    action = torch.LongTensor([action]) # indexとして使うのでLong型
    next_state = torch.FloatTensor([next_state])

    ## Q値の算出
    q_now = dqn(state).gather(-1, action) # 今の状態のQ値
    max_q_next = dqn(next_state).max(-1)[0].detach() # 状態移行後の最大のQ値
    gamma = 0.9
    q_target = reward + gamma * max_q_next # 目標のQ値

    # DQNパラメータ更新
    optimizer.zero_grad()
    loss = criterion(q_now, q_target) # 今のQ値と目標のQ値で誤差を取る
    loss.backward()
    optimizer.step()

    return loss.item() # lossの確認のために返す
```

DQNからの行動の決定

DQNは入力された状態に対し、取りうるすべての行動のQ値のtensorを返す（1次元tensor)。そこから最大のQ値のindexを取り出して(0または1)、それをそのまま行動としている。

```
import numpy as np

EPS_START = 0.9
EPS_END = 0.0
EPS_DECAY = 200

def decide_action(state, episode):
    state = torch.FloatTensor([state]) # 状態を1次元tensorに変換
    ## ε-グリーディー法
    eps = EPS_END + (EPS_START - EPS_END) * np.exp(-episode / EPS_DECAY)
    if eps <= np.random.uniform(0, 1):
        with torch.no_grad():
            action = dqn(state).max(-1)[1] # Q値が最大のindexが得られる
            action = action.item() # 0次元tensorを通常の数値に変換
    else:
        num_actions = len(dqn(state)) # action数取得
        action = np.random.choice(np.arange(num_actions)) # ランダム行動
    return action
```

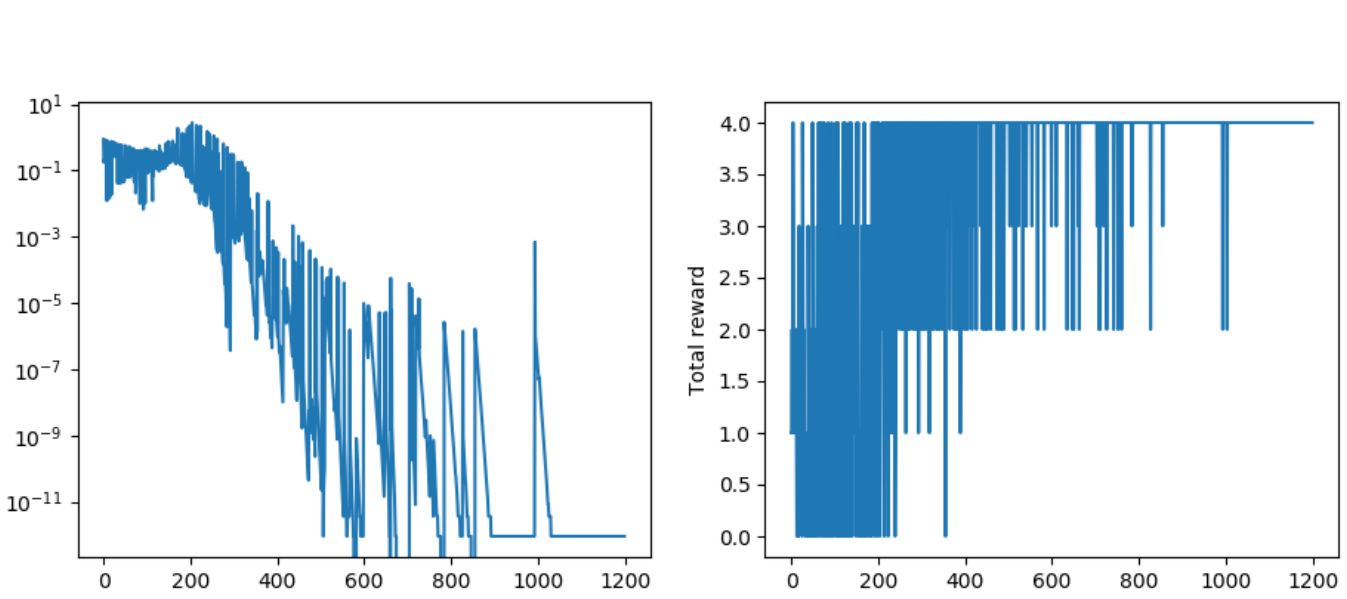
学習する

現在の状態からDQNより行動を決めて、環境から次の状態・報酬を取得し、これらの結果を用いてDQNを更新する、を繰り返す。エピソードの数や上記のoptimizerの学習率、ε-グリーディー法のepsの変化のさせ方はうまく学習が進むように調節して決めている。（適度な値を決めるのにかなり苦労した。）

```
import matplotlib.pyplot as plt

NUM_EPISODES = 1200
NUM_STEPS = 5

log = [] # 結果のプロット用

for episode in range(NUM_EPISODES):
    env = Dispenser(0)
    total_reward = 0 # 1エピソードでの報酬の合計を保持する

    for s in range(NUM_STEPS):
        ## 現在の状態を確認
        state = env.state
        ## 行動を決める
        action = decide_action(state, episode)
        ## 決めた行動に従いステップを進める。次の状態、報酬を得る
        next_state, reward = env.step(action)
        ## DQNを更新
        loss = update_dqn(state, action, next_state, reward)
        total_reward += reward

    log.append([total_reward, loss])

## 結果表示
r, l = np.array(log).T
fig = plt.figure(figsize=(11,4))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_xlabel("Episode")
ax1.set_ylabel("Loss")
ax1.set_yscale("log")
ax2.set_xlabel("Episode")
ax2.set_ylabel("Total reward")
ax1.plot(l)
ax2.plot(r)
plt.show()
```

学習結果

下図に学習の推移を示す。左は誤差の推移、右は1エピソードにおけるトータル報酬の推移である。払出機は初め電源OFFのため、5回の行動で得られる最大のトータル報酬は4であるが、右下のグラフから最終的にトータル報酬4が得られる行動を適切に学習できていることを確認できる。



学習後のDQNから各状態におけるQ値を確認できるが、今回の学習での値は下記である。

```
>>> dqn(torch.FloatTensor([0]))
tensor([9.0000, 8.1200], grad_fn=<AddBackward0>)
>>> dqn(torch.FloatTensor([1]))
tensor([8.1060, 10.0000], grad_fn=<AddBackward0>)
```

計算の詳細は示さないが、今回と同一の問題にQ学習を適用するとこれらとほぼ同じQ値が得られ、DQNによってQ値を適切に表現する関数を構築できていることを確認できる。

さいごに

DQNを適用するのが全く意味がないような簡単な問題にDQNを適用した。"Experience Replay"も考えなかった（状態×行動の組み合わせが4パターンしかない問題なのであまり効果はないのかも）。簡単すぎる問題であるがDQNを適用する上での基礎的なポイントは掴めたのではないかと思う。

コード

ここで示した全コードおよびQ学習を適用した場合のコードはこちら:

chemical-engineer-learns-ai/20190504_simple_example_of_DQN at master · nakamura-13/chemical-engineer-learns-ai · GitHub

参考

・Reinforcement Learning (DQN) Tutorial — PyTorch Tutorials 1.1.0 documentation

・牧野浩二、西崎博光，『Pythonによる深層強化学習入門』, オーム社 (2018)

schemer1341 5年前 読者になる

0 B!ブックマーク 0 シェアする

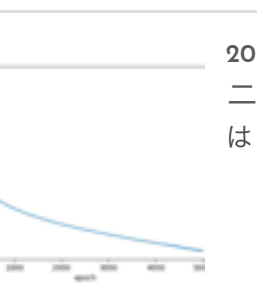
関連記事


2019-09-11
深層強化学習の簡単な例 ～Double DQN適用～
はじめに 以前の記事で簡単な問題を対象にDQN、Experience Repl...

2019-04-30
反応器計算におけるQ学習2
はじめに 以前の記事で管型反応器において所望の反応率を得ると...

2019-04-27
反応器計算におけるQ学習
はじめに 簡単な例として、管型反応器において反応温度を調節し...

2019-01-20
スキナー箱のQ学習
はじめに ネズミがボタンを押して餌を得るという問題について強...


2019-01-13
ニューラルネットワークによる手書き数字の認識
はじめに scikit-learnライブラリに含まれる手書き数字データを...

コメントを書く

« 深層強化学習(DEEP Q NETWORK, DQN)の簡単... 反応器計算におけるQ学習2 »

プロフィール

schemer1341

読者になる 5

このブログについて

検索

記事を検索

リンク

はてなブログ
ブログをはじめる
週刊はてなブログ
はてなブログPro

最新記事

メモ: Python defaultdictの使い方
OpenAI Gym 基本的な使い方のメモ
メモ: Miniconda インストール on macOS
メモ: ベイズ推定 MCMC法のためのPyStanの使い方
CNN(Convolutional Neural Network)を用いた画像識別の簡単な例

月別アーカイブ

▼ 2021 (3)
  2021 / 2 (1)
  2021 / 1 (2)
▶ 2020 (2)
▶ 2019 (18)