Reinforcement Learning Basics

mishima.syk #12
iwatobipen

自己紹介

@iwatobipen
Job : 某製薬企業で働いています。
Like : Chemoinformatics
Language : Japanese, Python, Javascript, R, english
Twitter : https://twitter.com/iwatobipen
Blog : https://wordpress.com/view/iwatobipen.wordpress.com

Reinforcement Learning(強化学習)

ある環境内におけるエージェントが、現在の状態を観測し、とるべき行動を決定する問題を扱う機械学習の一種

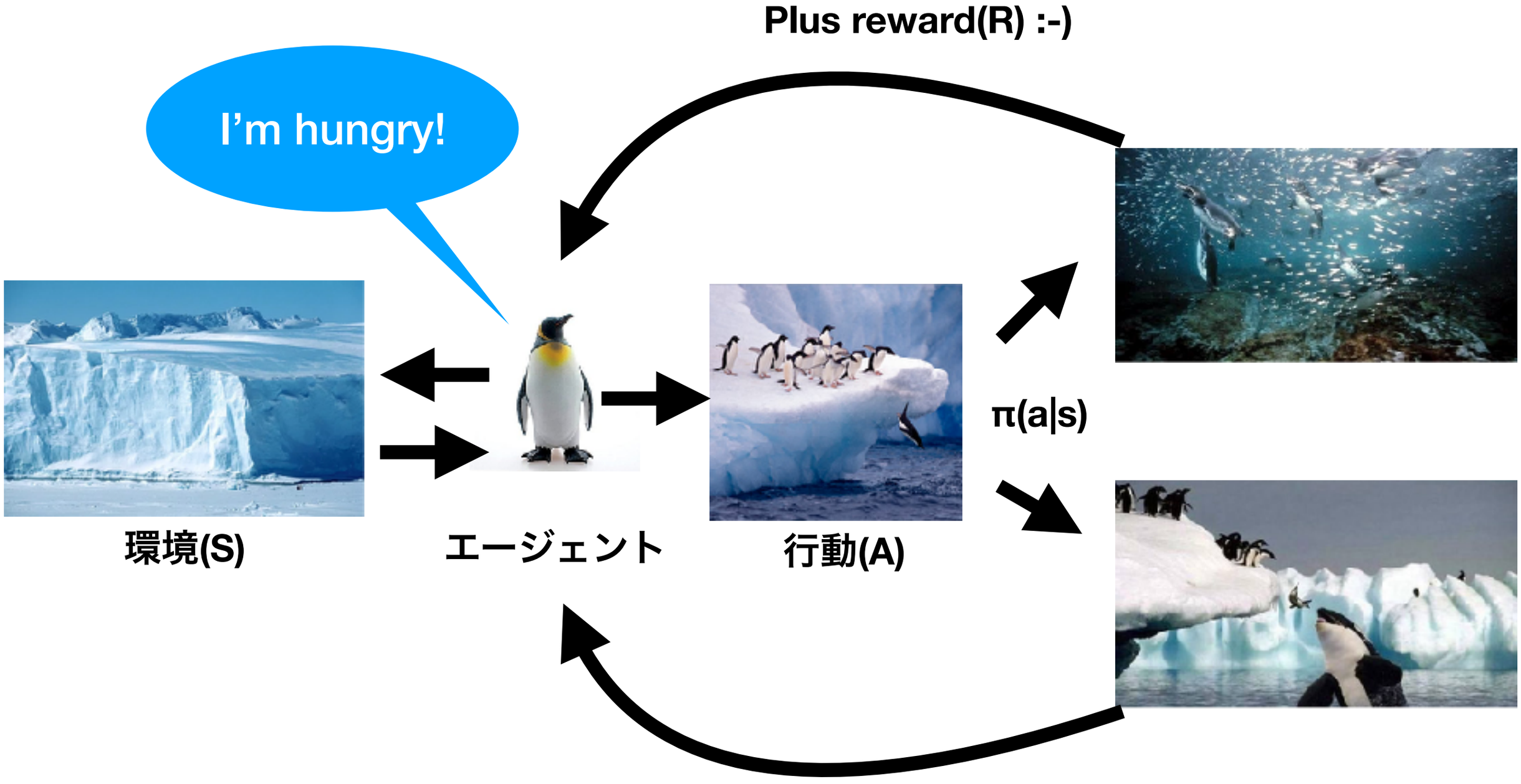
目的 : 与えられた環境における価値を最大化するようにエージェントを学習させる

今日はQ学習と、SARSAを紹介します。

参考文献)

https://drive.google.com/file/d/1xeUDVGWGUUv1-ccUMAZHJLej2C7aAFWY/view

For the first penguin…

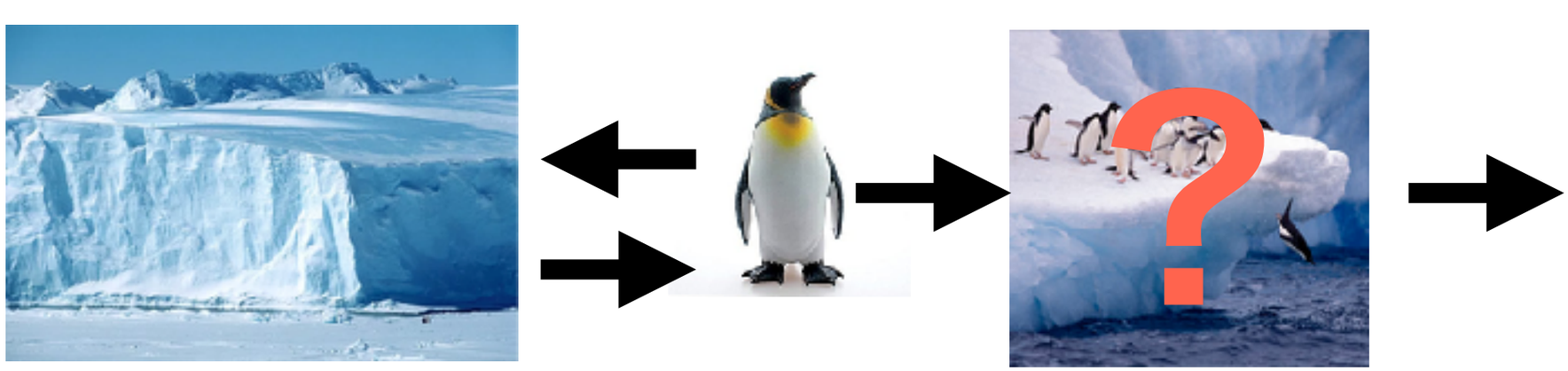


さて問題は、、、

最適な行動選択とは何か

飛び込まない＝＞鯱リスク↓空腹感↑

飛び込む！＝＞鯱リスク↑おさかなゲット確率↑



環境(S)　　**エージェント**

ある状態(S)で行動(A)をして、次の状態(S')になった時の価値(V)がわかれば次の行動を決められるジャーン！と言える。

（ただしマルコフ決定過程MDPの元という過程下）

Markov Decision Process (MDP)

環境　$S = \{s_1, s_2, \cdots, s_n\}$

行動　$A = \{a_1, a_2, \cdots, a_n\}$

ある環境sで行動aをした際にs'に遷移する確率を

$$Pr = \{S_{t+1} = s' | S_t = s, A_t = a\}$$

とする。

状態s'への遷移がその時の状態sと行動aのみに依存する。

＝＞マルコフ性という。


あんだって〜
耳が
遠くて
聞こえねーよ
!!!
!!?

簡単にまとめると今の状態が大事

エージェントのモチベーション＝利益はどう考えるか

ある時間tで実行した行動の価値が未来にどの程度貢献したのかを考えたい。

rtをある時刻における報酬と定義した場合、割引率γ（0<γ<1）を導入して利益下記のように表現する。

いま！　　みらい＞＞＞＞＞！

$$V_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

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

ある状態s、方策πの元での利益と行動価値を考える

Pは状態遷移確率。rは報酬です。

状態Stで行動aが決定される確率はπ(a|St)となります。

以上の前提を置くと

$$V^{\pi}(s) = \sum_{s' \in S} \sum_{a \in A(s)} P(S_{t+1} = s', A_t = a | S_t = s) r(s, a, s')$$

$$= \sum_{s' \in S} \sum_{a \in A(s)} P(S_{t+1} = s' | A_t = a, S_t = s) \pi(a|s) r(s, a, s')$$

遷移状態確率の事前条件を変えて方策πが入って来ました。

もう少し条件を与えて考えます。

今までの議論はある状態sに関してでした。実際に行動決定をするには行動も条件に入れた方が良いので条件に入れます。

＝＞行動価値（Q）関数。状態関数（V)と比較して見ましょう。

$$V(s) = E^{\pi}[G_{t+1}|S_t = s]$$

$$Q(s,a) = E^{\pi}[G_{t+1}|S_t = s, A_t = a]$$

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(a|s) Q^{\pi}(s,a)$$

難しい感じだけどQ関数はテーブル！

このQテーブルを更新することで最適な行動を学習します！

$$Q(s,a) = E^{\pi}[G_{t+1}|S_t = s, A_t = a]$$

状態	飛び込む	寝る
満腹	0.1	0.9
普通	0.5	0.5
空腹	0.8	0.2

方策の考え方
（よく用いられるものε-Greedy）

次の行動は一番良さげ（報酬が多い）物を選ぶのがいい。
でも試行錯誤した結果それがベストという保証はない。
＝＞良さげなの選ぶけどたまに他の可能性も探りたい
＝＞ε-greedy; 一定確率でランダムに行動する


```
if np.random.uniform() > EPSILON:
    action = state_actions.random()
else:
    action = state_actions.max()
```

state_actions: ある状態で取り得る行動の集合

ここまでのまとめ＋α

1,MDP条件下、ある状態で一番最適な行動を探します。

2,割引率を導入して利益を定式化しました。

3,ε-greedyという方策で行動することにしました。

4,3の戦略に従い試行錯誤してQ関数（テーブル）更新しながら学ぶ

5,導出は省きますが下式をベルマン方程式といい、学習に使います

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s' \mid s,a)[r(s,a,s') + \sum_{a \in A(s')} \gamma P(a' \mid s') Q^{\pi}(s,,a)]$$

Q-学習とSARSAってのを次に紹介するよ

Q-Learn & SARSA

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s'|s,a)[r(s,a,s') + \sum_{a \in A(s')} \gamma P(a'|s') Q^{\pi}(s,,a)$$

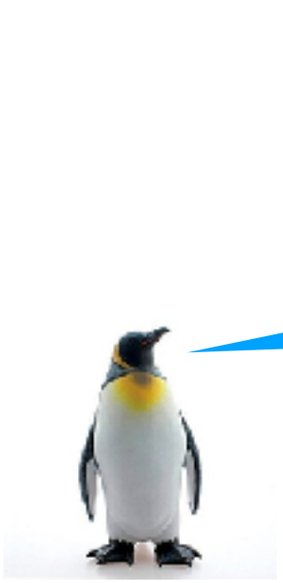
Q-Learn: 行動価値を以下の式で更新します（方策オフ）

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max Q(S_{t+1}, a'))$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max Q(S_{t+1}, a') - Q(S_t, A_t))$$

SARSA: 行動価値を以下の式で更新します（方策オン）

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}))$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

TD誤差



αは学習率だよ
Q学習とSARSAは最大となる行動をとるか、とりあえず行動するかがちがうね。

シンプルな例を考えましょう！

１次元の住人になります！　＝＞行動は右か左だけ！

右XXXstep先にお宝があります！

お宝ゲット＝＞報酬あり

それ以外＝＞報酬なし

方策＝＞ε-greedy

コードの一部

s－－－－－－T
－: step
s: start
T: treasure

```
def build_Q_table(n_states, actions):
    table = pd.DataFrame(np.zeros((n_states, len(actions))), columns=actions)
    return table

def choose_action(state, q_table):
    state_actions = q_table.iloc[state,:]
    if (np.random.uniform() > EPSILON) or (state_actions.all()==0):
        action_name = np.random.choice(ACTIONS)
    else:
        action_name = state_actions.argmax()
   return action_name
```

＜＝Q-table
　　初期化

＜＝ε-greedy

Pandas
　loc 行ラベル、列ラベル
　iloc 行のindex, 列のindex

Q-Learn & SARSA in code

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max Q(S_{t+1}, a') - Q(S_t, A_t))$$

```
def rlQlearn():
    q_table = build_Q_table(N_STATES, ACTIONS)
    for episode in range(MAX_EPISODES):
        step_counter = 0
        S = 0
        is_terminated = False
        update_env(S, episode, step_counter)
        while not is_terminated:
            A = choose_action(S, q_table)
            S_, R = get_env_feedback(S, A)
            q_predict = q_table.loc[S, A]
            if S_ != ‘terminal':

                q_target = R + GAMMA * q_table.iloc[S_, :].max()
            else:
                q_target = R
                is_terminated = True
            q_table.loc[S, A] += ALPHA * (q_target - q_predict)
            S = S_
            update_env(S, episode, step_counter + 1)
    return q_table
```

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

```
def rlSARSA():
    q_table = build_Q_table(N_STATES, ACTIONS)
    for episode in range(MAX_EPISODES):
        step_counter = 0
        S = 0
        is_terminated = False
        update_env(S, episode, step_counter)
        while not is_terminated:
            A = choose_action(S, q_table)
            S_, R = get_env_feedback(S, A)
            q_predict = q_table.loc[S, A]
            if S_ != 'terminal':
                A_ = choose_action(S, q_table)
                q_target = R + GAMMA * q_table.loc[S_, A_]
            else:
                q_target = R
                is_terminated = True
            q_table.loc[S, A] += ALPHA * (q_target - q_predict)
            S = S_
            update_env(S, episode, step_counter + 1)
    return q_table
```

SARSAはQテーブル更新

Qテーブルを計算してみる(Q-Learning)

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma maxQ(S_{t+1}, a') - Q(S_t, A_t))$$

α＝0.1, γ=0.9とした場合

S - - - - - T　　S,0,0,0,0,0,1　　初期

S - - - - - T　　S,0,0,0,0,0.9,1　　Tに到達できた場合

Q = Q + 0.1 x (0 + 0.9 x MaxQ - Q)
= 0.9

S - - - - - T　　S,0,0,0,0.081,0.9,1　　次にTに到達できた場合

Q = 0 + 0.1 x (0 + 0.9 x 0.9 - Q)
= 0.081

Githubにあげたコードで遊ぶ

Let’s Enjoy !

もう一個やってみよう！

2次元の住人になります！　＝＞行動は上下左右　ドラクエみたい

４x４の世界。左上からスタート右下がゴールです

途中に壁があって当たると痛いっす。＝＞負の報酬

それ以外＝＞報酬なし

方策＝＞ε-greedy

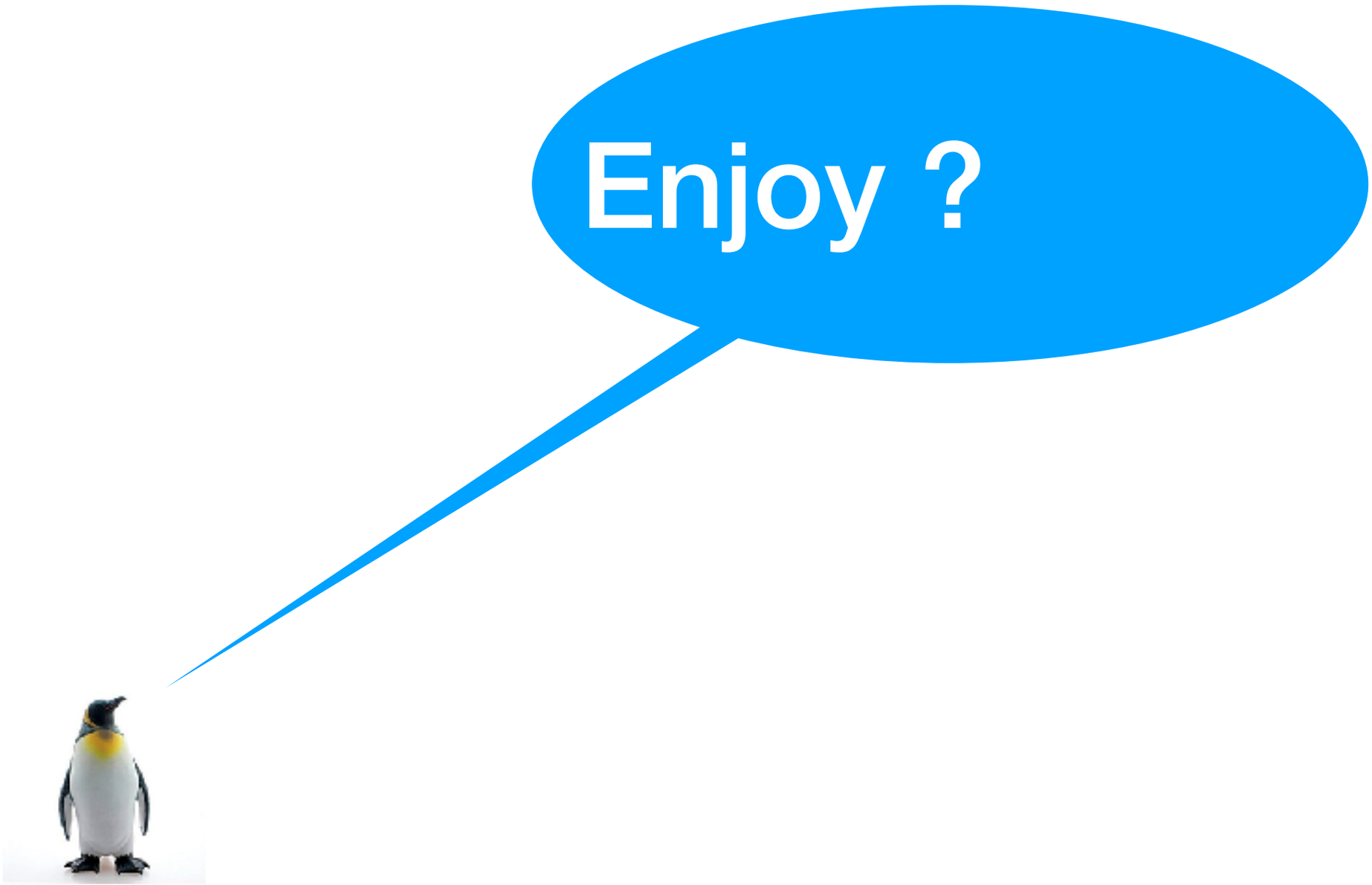
[[‘P’, 'F', 'F', 'W'],
 ['F', 'W', 'F', 'F'],
 ['F', 'F', 'W', 'W'],
 ['F', 'F', 'F', 'T']]

ベースは変わらないよ。


```
def get_env_feedback(S, A):
    # The argent can get rewards when Agent could arrive at Terminal.
    x,y = STATE_index[S]
    if A == 'u':
        x_, y_ = x - 1, y + 0  # UP
        if MAZE_LIST[x][y] == "T":
            S_ = 'terminal'
            R = 1
        elif move_check(x_, y_):
            S_ = STATE_index.index([x,y])
            R = -1
        elif MAZE_LIST[x_][y_] == "W": # OUCH
            S_ = STATE_index.index([x,y])
            R = -1
        else:
            S_ = STATE_index.index([x_,y_])
            R = 0
        return S_, R

    if A == 'd':
        x_, y_ = x + 1, y + 0  # DOWN
        if MAZE_LIST[x][y] == "T":
            S_ = 'terminal'
            ……snip…….
```

Githubにあげたコードで遊ぶ



DQNとはなにか？

DQN could refer to:

- DQN (*Dokyūn*), a slang term used in 2channel for someone who is extremely foolish
- Du Quoin (Amtrak station), Amtrak station code DQN
- Station code for Dhanera station, Gujarat, India - see List of railway stations in India
- An abbreviation for "Deep Q-Network", a variant of the Q-learning algorithm for machine learning

今日の流れ的にDQNとは

- An abbreviation for "Deep Q-Network", a variant of the Q-learning algorithm for machine learning

DQNの概要

- model(DNN)を定義　f(state) => action
- ε -greedy などで行動してmemoryに保存 #memory
- memory からランダムにサンプリングしモデルを更新 #reply
- Lossは下記のようにすればMSEとして解ける

$$L = \frac{1}{2}E[(r + \gamma max_{a'}Q_\theta(s', a') - Q_\theta(s, a))^2]$$

教師信号　　最適化対象

多分時間がないので、MountainCarのでもを

Gym使って書いて提供したので遊んで見てね

- Requirements
  - Open AI gym
  - Keras-rl ＝＞使いやすいけどすべてWrappされているので中身はわかりにくい

最後に

最近流行りのDeep Q-LearningやOpen AIは今回話せませんでした。

創薬にも活用したいですね！