RL Tutorial

FIT2017

MDP

- 1. S 表示状态集 (states)；
- 2. A 表示动作集 (Action)；
- 3. P^{s'}_{s,a} 表示状态 s 下采取动作 a 之后转移到 s' 状态的概率；
- 4. R_{s,a} 表示状态 s 下采取动作 a 获得的奖励；
- 5. γ 是衰减因子。

Reinforcement Learning

- 学习一个策略，输入当前的状态s，输出采取动作a的概率π(s,a)
- 只选择当前奖励最大的动作显然不（一定）是最优的：

$$E_\pi[\sum_{k=0}^{\infty}\gamma^k R_k] = E_\pi[R_0 + \gamma R_1 + \ldots]$$

两种套路

- 1.学习价值函数，根据价值函数导出策略
- 2.直接学习策略
- DQN属于前者

Reinforcement Learning

- 价值：按照当前策略获得的递减奖励期望

$$v(s) = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_k]$$

- 拓展到状态－动作对上：

$$q(s,a) = R_{s,a} + E_\pi[\sum_{k=1}^{\infty} \gamma^k R_k]$$

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy* π_* *that is better than or equal to all other policies,* $\pi_* \geq \pi, \forall \pi$
- *All optimal policies achieve the optimal value function,* $v_{\pi_*}(s) = v_*(s)$
- *All optimal policies achieve the optimal action-value function,* $q_{\pi_*}(s, a) = q_*(s, a)$

最优策略存在性

贝尔曼等式

$$
\begin{aligned}
&v(s) \\
= &\sum_{a\in A}\pi(s,a)q(s,a) \\
= &\sum_{a\in A}\pi(s,a)(R_{s,a}+\gamma\sum_{s'\in S}T_{s,a}^{s'}v(s'))
\end{aligned}
$$

$$
\begin{aligned}
&q(s,a) \\
&= R_{s,a}+\gamma\sum_{s'\in S}T_{s,a}^{s'}v(s') \\
&= R_{s,a}+\gamma\sum_{s'\in S}T_{s,a}^{s'}\sum_{a'\in A}\pi(s',a')q(s',a')
\end{aligned}
$$

Q-Learning

- 根据当前的价值探索，每一步更新状态价值
- “梯度下降”

$$q(s,a) = q(s,a) + \alpha\{r + max_{a'}\{\gamma q(s', a')\} - q(s,a)\}$$

- 使用ϵ-贪婪策略

$$\pi_{\epsilon-greedy}(s,a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|} & a = argmax_a q(s,a) \\ \frac{\epsilon}{|A|} & a \neq argmax_a q(s,a) \end{cases}$$

初始化$Q(s,a),\forall s\in S, a\in A(s)$,任意的数值，并且$Q(terminal-state,\cdot)=0$

重复（对每一节episode）：

初始化 状态S

重复（对episode中的每一步）：

使用某一个policy比如（$\epsilon-greedy$)根据状态S选取一个动作执行

执行完动作后，观察reward和新的状态S'

$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha(R_{t+1}+\lambda \max_a Q(S_{t+1},a)-Q(S_t,A_t))$

$S\leftarrow S'$

循环直到S终止

Q-Learning

其他策略学习方法

- 蒙卡，SARSA
- Q-Learning保证收敛到最优策略的状态-动作价值

价值函数近似

- 刚才的式子只是数学上的
- 实际应用中，我们用NN学习一个函数f(s, a)来拟合Q(s, a)
- 更简单的，我们可以让NN做到：输入状态s，输出所有动作对应的Q值，称为Q网络

DQN Algorithm

https://arxiv.org/abs/1312.5602

- 训练样本？Q-Learning探索产生
- Q-Learning中Q值的更新：
  - target就是 $r + max_{a'}\{\gamma q(s',a')\}$
- 每次将计算出的target作为label，当前状态（并不一定只是一帧画面）作为input喂进网络

$$L(w) = \mathbb{E}[(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{Target} - Q(s, a, w))^2]$$

Experience Replay

- NN作为有监督学习模型，要求数据满足独立同分布，但Q-Learning算法得到的样本前后是有关系的
- 将系统探索环境得到的数据储存起来，然后随机采样样本喂给NN

TensorFlow Basic Usage

- Install: https://www.tensorflow.org/install/
- Example: MNIST

PyTorch Basic Usage

- Install: http://pytorch.org/
- Example: MNIST again
- torchvision: vision related datasets and models

Implement DQN

- Data source?
- OpenAI gym: https://gym.openai.com/
  - provide games and APIs
- pygame module
  - write games and APIs yourself
- From other games…
  - https://arxiv.org/pdf/1608.02192v1.pdf
  - that’s cool :)

OpenAI gym

- Make environment
  - env = gym.make('CartPole-v0')
- (Re)start game
  - state = env.reset()
- Act, get reward and the next state
  - balabala = env.step(action)
- Render scene
  - env.render()

DQN with TensorFlow

DQN with PyTorch

Train on the server

- Don’t forget: **CUDA_VISIBLE_DEVICE**
- Render?
  - X-Server:
    - Send event to X-Client
    - X-Client do computation and request back
    - X-Server show the results
    - Slow…

Train on the server

- What if a local X-Server?
- Virtual X-Server: xvfb
- Already installed on our server
- Usage:
  - xvfb-run -a -s "-screen 0 1400x900x24 +extension RANDR" -- python dqn.py

Next time

- 基于策略的Policy gradient:
  - Actor-critic
  - …
- Advanced Q-Networks
  - Double Q Nets https://arxiv.org/abs/1509.06461
  - Dueling Nets https://arxiv.org/abs/1511.06581
  - …
- Robotics?

Thanks!

Q&A?

References

- http://www.algorithmdog.com/series/rl-series
- https://zhuanlan.zhihu.com/intelligentunit
- http://pytorch.org/
- https://www.tensorflow.org/