Reinforcement Learning

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

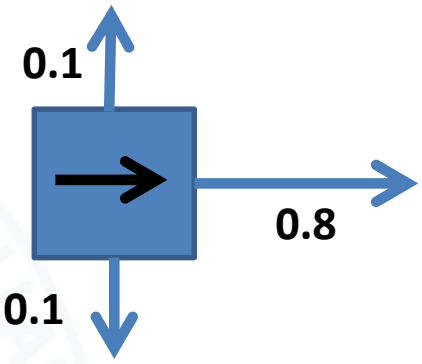
Basics

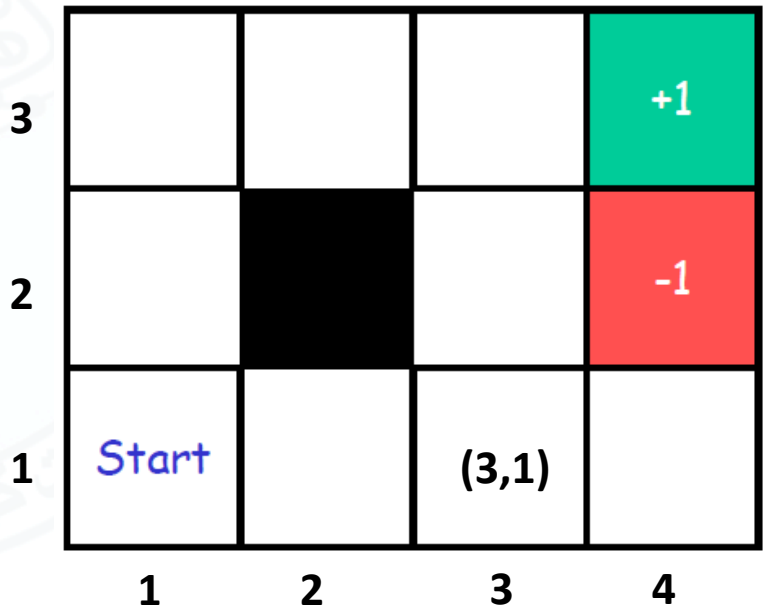
- A school may give you rewards as follows:
  - 1000 if you stood first
  - 500 if you stood second
  - 250 if you stood third
  - 100 if you take any other position
- Your chances of getting the positions are as follows:
  - 50% for first position
  - 30% for second
  - 10% for third
  - 10% for any other position
- What is the expected reward?

- V(1) = 1000
- V(2) = 500
- V(3) = 250
- V(>3) = 100
- P(1) = 0.50
- P(2) = 0.30
- P(3) = 0.10
- P(>3) = 0.10
- E = V(1)P(1)+V(2)P(2)+V(3)P(3)+V(>3)P(>3)=500+150+75+10 = 735

Decisions of Agents

- A decision is an action by an agent
  - In what kind of environment will it be easy to make decisions?
  - Example
    - Consider an agent that can move UDLR in a grid
    - However, due to sensor/actuator errors, it ends up in its intended next square 80% of the time
    - 10% of the time it ends up at a right angle from the intended target
- To maximize the reward, what will be the sequence of decisions of the agent?
  - Each state gives a reward: the two terminal states (red and green) have rewards +1 and -1 respectively. All other states have a reward of -0.04.

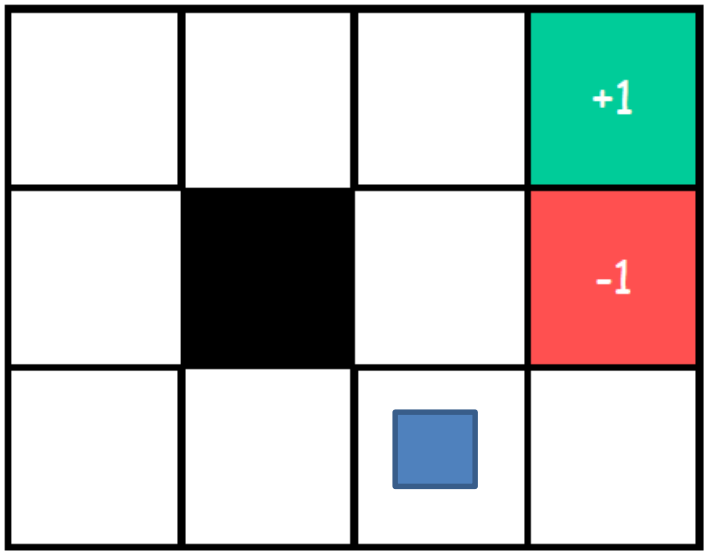




Uncertainty in effects of actions

- In a non-deterministic environment
  - Assume the current state is "s"
  - An action "a" is performed in this state
  - The probability that the result of this action produces state s' is:

$$P(Result(a, s) = s'|a) = \mathrm{P}(\mathrm{s}'|\mathrm{s}, \mathrm{a})$$



Assume that the agent executes an action "U" in state (3,1)

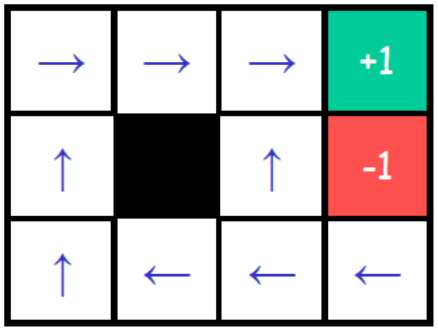
$$P(Result(U, (3,1)) = (3,2)|U) = 0.8$$
$$P(Result(U, (3,1)) = (2,1)|U) = 0.1$$
$$P(Result(U, (3,1)) = (4,1)|U) = 0.1$$

Markov Decision Processes

- Agent in a Markov Decision Process
  - A set of states S
  - A set of actions in each state A(s)
    - An action causes a transition
  - A transition probability model (may be unknown!)
    - P(s'|s,a) is the probability of reaching state s' from s by action a
  - Each state has a reward $R(s)$
    - The probability of reaching a state is dependent only on the previous state and the action taken in that state

Reward, Policy and Utility

- Reward: Reward of -0.04 in each state except the terminal states
- What action should be taken in each state to maximize the overall reward?
- The solution is represented in terms of a policy
  - Tells the action to be taken in that state: $\pi(s)$
- For a given policy, each state has a utility
  - Equal to the reward of that state and discounted rewards for future states (i.e., expected utility of the next state)



Bellman equations

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] = \mathrm{E}\left[R(s) + \sum_{t=1}^{\infty} \gamma^t R(S_t)\right] = \mathrm{E}\left[R(s) + \gamma \sum_{t=0}^{\infty} \gamma^t R(S_{t+1})\right] = R(s) + \gamma U^{\pi}(next)$$

$$U^{\pi}(next) = \sum_{s' \in S} P\big(s' | s, \pi(s)\big) U(s') \qquad U(next) = max_{a \in A(s)} \sum_{s' \in S} P(s' | s, a) U(s')$$

Bellman Equations

- Utility of a state s for a policy is the sum of rewards of all the states to come beginning in the initial state

$$U^{\pi}(s) = \sum_{t=0}^{\infty} R(S_t)$$

- Since the effect of an action is not deterministic, therefore, we take the expected value of the rewards, or:

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} R(S_t)\right]$$

- We reduce the effects of future rewards in states (because, we aren't sure about them!)

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

Bellman Equations

- The equation can be expanded as:

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] = \mathrm{E}\left[R(s) + \sum_{t=1}^{\infty} \gamma^t R(S_t)\right]$$

$$= \mathrm{E}\left[R(s) + \gamma \sum_{t=0}^{\infty} \gamma^t R(S_{t+1})\right] = R(s) + \gamma U^{\pi}(next)$$

- Thus: $U^{\pi}(s) = R(s) + \gamma U^{\pi}(next)$
- The action in the current state is determined by the policy, therefore:
  - The probability that the effect of action a $= \pi(s)$ in state s will be state s' is $P\big(s' | s, \pi(s)\big)$
  - The utility of being in s' is $U(s')$
  - Therefore, the expected utility of the next state under a given policy is:

$$U^{\pi}(next) = \sum_{s' \in S} P\big(s' | s, \pi(s)\big) U(s')$$

If the policy is not known…

- If we do not know the policy, then the principle of maximum expected utility (MEU) states that, we should perform an action which maximizes our expected utility
  - The probability of moving from state s to state s' due to action a is $P(s'|s,a)$
  - The utility of state s' is $U(s')$
  - Therefore, the expected utility due to action a is:

$$EU(a) = \sum_{s' \in S} P(s'|s,a)U(s')$$

  - Therefore the utility under MEU, the utility of the next state is

$$U(next) = max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)U(s')$$

Q-Learning

- In active learning we must also obtain what the optimal action is. So instead of utilities, which are specific to states, we use an action-utility representation $Q(s, a)$ which is directly related to the utility of the state by

$$U(s) = max_a Q(s, a)$$

Q-values vs. Utilities

- Utility is for a state, Q-value is for a state-action pair
- Utility tells us the desirability of a state, Q-value tells us the desirability of an action in a state
- Similar to utility values

$$U(s) = R(s) + \gamma max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a) U(s')$$

- Bellman Equations can also be written for Q-values

$$Q(s,a) = R(s) + \gamma \sum_{s' \in S} P(s'|s,a) max_{a' \in A(s')} Q(s',a')$$

- Notice that: $U(s) = max_a Q(s,a)$

Q-Learning

- Given initial estimates of $Q(s,a)$, we can update the value as a weighted combination of the previous and "new" values

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

  - At each step, pick the action that you think is the best based on the Q value estimates
  - α is the learning rate and it should decrease as go through the iterations
- The optimal action in a state s is: $\text{arg}max_a$ $Q(s,a)$

Pseudo code

Function Q-Learning-Agent(percept) **returns** action to be performed by agent

Inputs: percept, a percept indicating the current state s' and reward signal r'

Persistent: Q, a table of values for state-actions, initially zero

N_{sa}, a table of frequencies for state-action pairs, initially zero

s,a,r, the previous state, action and reward, initially null

If Terminal?(s) **then** $Q(s', a') \leftarrow r'$

If s is not null **then**

increment $N_{sa}(s, a)$

Update: $Q(s, a) \leftarrow Q(s, a) + \alpha\big(N_{sa}(s, a)\big)\big(R(s) + \gamma max_{a'} Q(s', a') - Q(s, a)\big)$

s,a,r $\leftarrow$ s', $\text{arg}max_{a'} f\big(Q(s', a'), N_{sa}(s', a')\big), r'$

Return a

The exploration function

- The exploration function allows an otherwise suboptimal action to be optimal if it has not been sampled enough times by returning an optimistic utility in such cases
  - $N(s,a)$ is the frequency of state-action pair (s,a) and $f(u,n)$ is a function called the exploration function. It controls the tradeoff between greed and curiosity. It should be increasing in u and decreasing in n.
- Example

$$f(u,N) = \begin{cases} R^+ & if\ n < N_e \\ u & else \end{cases}$$

Q-Function Approximation

- Instead of using a table, we can also use a function approximation scheme such as a Neural network to return Q(s,a) given a representation of s and a

Policy Search

- Directly search for a policy that maximizes the utility/Q-value

Deep Q-Learning

To remove correlations, build data-set from agent's own experience

- Take action a_t according to ε-greedy policy

  (Choose "best" action with probability 1- ε, and selects a random action with probability ε)

- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$ (Huge data base to store historical samples)
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimize MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i)}_{\text{target}} - Q(s, a; \theta_i) \right)^2 \right]$$

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters θ_i^-

$$r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$$

- Optimize MSE between Q-network and Q-learning targets

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

- Periodically update fixed parameters $\theta_i^- \leftarrow \theta_i$

DQN clips the reward to [-1, +1]

This prevents Q-values from becoming too large

Ensures gradients are well-conditioned

- Loss function :

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$

- Differentiating the loss function w.r.t. the weights we arrive at following gradient :

$$\nabla_{\theta_i} \mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

Do gradient descent:

$$\theta_{i+1} = \theta_i + \alpha \cdot \nabla_{\theta_i} L_i(\theta_i)$$

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

End For

During Training

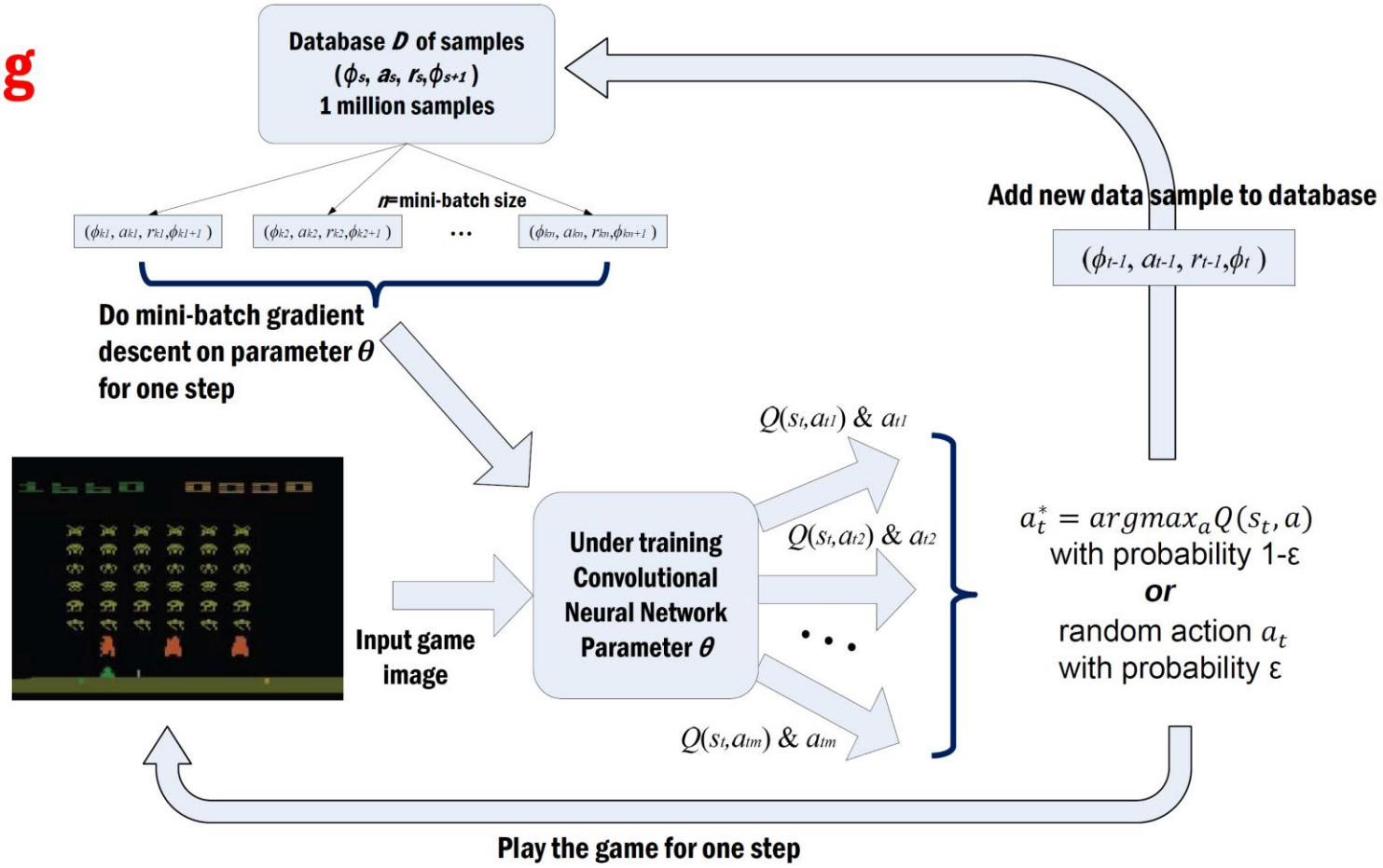


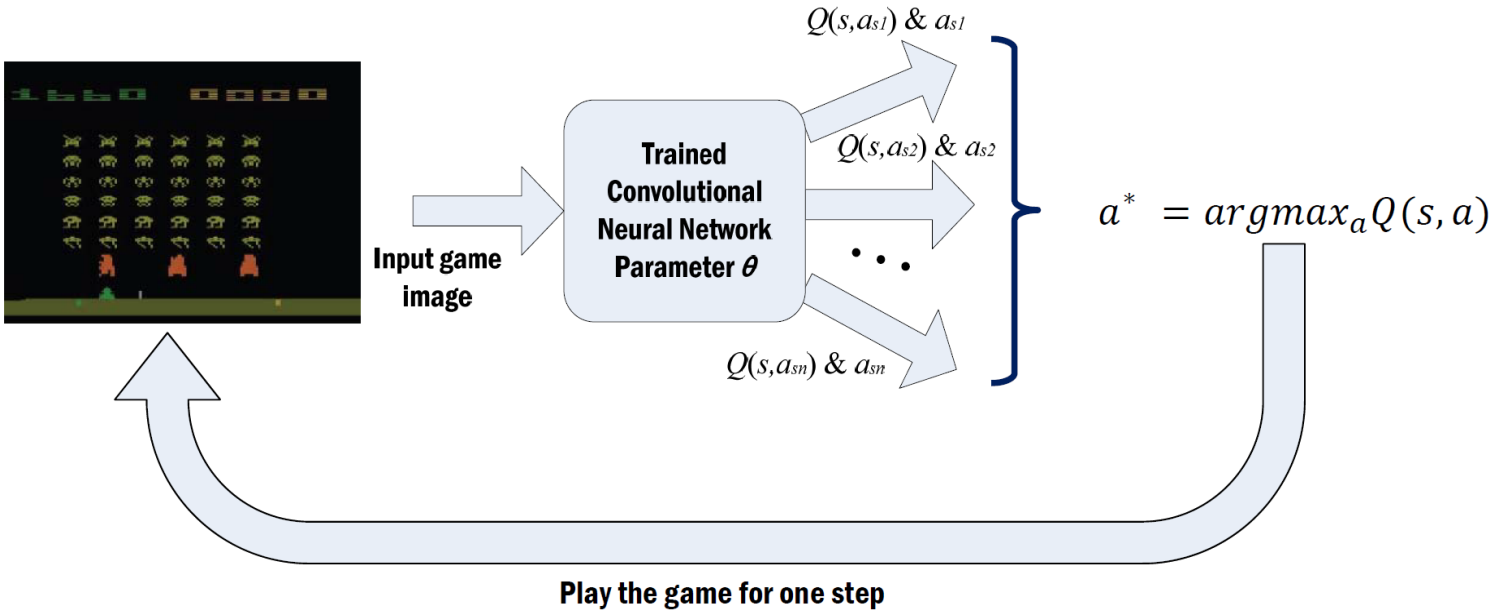
image at time t : x_t

$s_t = s_{t-1}, a_{t-1}, x_t$

preprocessed sequence

$\phi_t = \phi(s_t)$

After Training



Coding

- Coding
  - https://github.com/aimacode/aima-python/blob/master/rl.ipynb

Applications

- Robot Walking
- Game Play
  - Algorithm plays against itself to learn what moves to take in different states
  - Uses Neural Networks
- Web document searches
  - Minimizes the number of web-crawls
  - Only crawl when needed

Dynamic Marble Control

Useful Links

- Videos
- Learn from interaction with the environment
  - Just like humans
    - Learning from examples
    - Learning from experience
- Demystifying
  - https://www.intelnervana.com/demystifying-deep-reinforcement-learning/
- Flappy bird code
  - https://yanpanlau.github.io/2016/07/10/FlappyBird-Keras.html
- http://edersantana.github.io/articles/keras_rl/
- https://oshearesearch.com/index.php/2016/06/14/kerlym-a-deep-reinforcement-learning-toolbox-in-keras/
- Siraj
  - https://www.youtube.com/watch?v=79pmNdyxEGo&t=43s
  - https://www.youtube.com/watch?v=A5eihauRQvo
  - CAR
  - https://www.youtube.com/watch?v=EaY5QiZwSP4

End of Lecture

Humans Are the World's Best Pattern-Recognition Machines, But for How Long?

http://bigthink.com/endless-innovation/humans-are-the-worlds-best-pattern-recognition-machines-but-for-how-long