6 December, 2016

Neural Networks

Course 9: Reinforcement Learning

Overview

- Reinforcement Learning
- Off Policy vs On Policy
- Gradient Policy Algorithm
- Questions

Reinforcement Learning

Reinforcement Learning: concepts

- Reinforcement learning is learning what to do--how to map situations to actions--so as to maximize a numerical reward signal

- Reinforcement learning has 3 basic components:
  - Agent
  - Environment
  - Actions

Reinforcement Learning: concepts

- An agent has 4 basic components:
  - A policy

    The most important. Is the decision making function. Defines what should the agent do in any of the situations it encounters
  - A reward function

    Defines the goal of the RL agent. It maps the state of the environment to a single number, a reward, indicating the intrinsic desirability of the state
  - A value function

    Defines what is good in the long run. More exactly, the total amount of reward the agent can expect starting from that state
  - A model of the environment

    Defines the states as well as the way to transition from one state to another (through actions)

Reinforcement Learning: concepts

- The value function can be defined in two ways:

$$V^{\pi}(s) = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$

$$Q(s, a) = R(s, a, s') + \gamma \max_{a} Q(s', a')$$

Where γ is used in order to discount immediate rewards.

- Reinforcement learning is about learning a policy (π) that will tell us what action to take in which state: $\pi: S \rightarrow A$ in order to maximize the sum of rewards over the lifetime of the agent

Off Policy vs On Policy

Off Policy vs On Policy

- Off policy learning:

  An off-policy leaner learns the value of the optimal policy independently of the agent's actions. All it needs is to explore enough.

  Example: Q-learning

$$Q\ \ (s,a) = Q\ \ (s,a) + \alpha\left(R(s,a,s') + \gamma \max_a Q\ \ (s',a') - Q\ \ (s,a)\right) \text{ where}$$

$$\alpha = the\ learning\ rate$$
$$\gamma = the\ discount\ factor$$

  Observe that no matter what the current policy is, on the long term the Q values will be the same.

Off Policy vs On Policy

- On policy learning:

  An on-policy learner learns the value of the policy being carried out by the agent, including the exploration steps

  Example: SARSA

  $Q\ \ (s,a) = Q\ \ (s,a) + \alpha\left(R(s,a,s') + \gamma\, Q\ \ (s',a') - Q\ \ (s,a)\right)$ where

  $$\alpha = the\ learning\ rate$$
  $$\gamma = the\ discount\ factor$$

  Observe that the policy being learned is actual the current followed policy. What actually happens is that the agent learns to better evaluate the current value function.

Off Policy vs On Policy

- On policy vs Off policy

$$Q_{learning}: Q\ (s,a) = Q\ (s,a) + \alpha\left(R(s,a,s') + \gamma \max_a Q\ (s',a') - Q\ (s,a)\right)$$

$$SARSA\ : Q\ (s,a) = Q\ (s,a) + \alpha\left(R(s,a,s') + \gamma\, Q\ (s',a') - Q\ (s,a)\right)$$

Qlearning: Q-value is updated using the Q-values of the next state s' and the greedy action a' which can be different from the action dictated by the current policy

Sarsa: Q-value is updated using the Q-values of the next state and the current policy action a'. It estimates the returns, assuming the current policy is followed

Off Policy vs On Policy

- On policy vs Off policy
  - Off policy algorithms can update the estimated value functions using hypothetical actions, those which have not actually been tried
  - On policy algorithms update the value function strictly based on experience

  - The on-policy methods can take into account actions that might bring a very high penalty and adapt accordingly
  - The on-policy methods can get stuck in local minimum

Off Policy vs On Policy

- A classic example:

  A mouse has to find the path to its house. It can move N, W, S, E

  Between the mouse and its house there is a cliff.

  If the mouse gets to the cliff, a big reward(penalty) will be received: -100

  For every move he makes he gets a reward (penalty) of -1

  With a probability of 0.1, the mouse makes a random move (in order to explore the environment)

Off Policy vs On Policy

- A classic example:
  - The agent using Q-learning finds the optimal policy: it moves along the cliff and gets to the reward.

    However, from time to time (due to exploration) the mouse falls into the cliff

  - The agent using SARSA learning understands that from time to time it performs some random actions and tries to adapt accordingly.

    For this reason, the agent goes several steps away from the cliff and then starts orienting to the target

Gradient Policy Algorithm

Gradient Policy Algorithm

- Until now, we've been using an algorithm (Q-learning or SARSA) to decide what are actions will be
- We've used neuronal networks to approximate the value function

However, we can use neural networks to directly learn the right actions the agent needs to take in order to increase the total reward

We'll use the car example from the previous class

Example

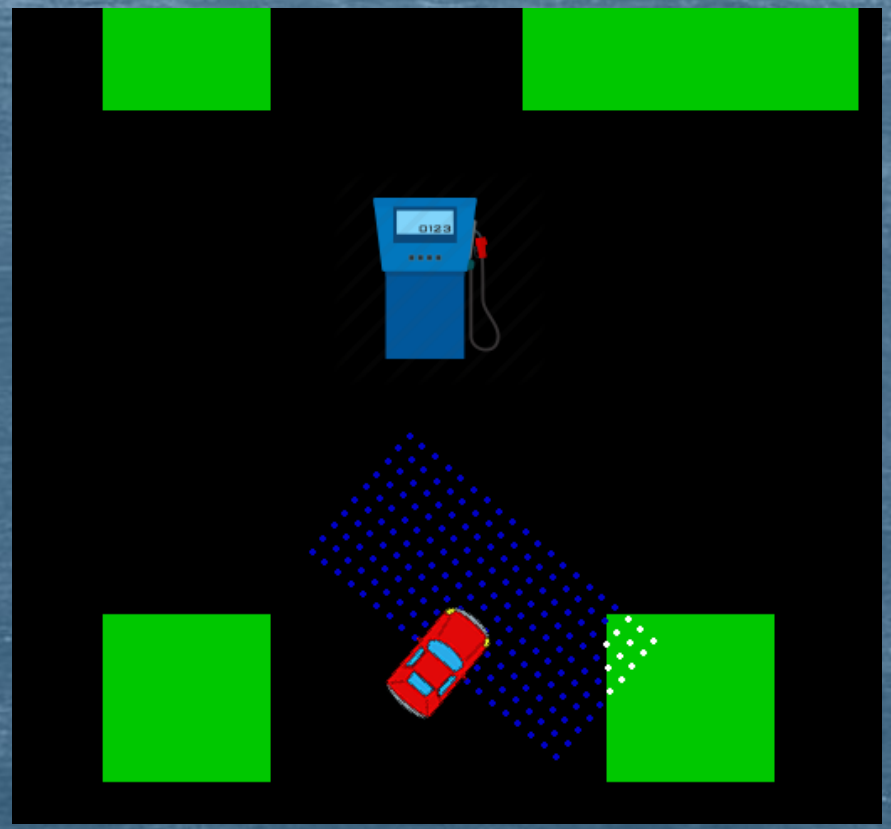
Example: Train a car to avoid obstacles and reach to the gas station

States: the car only knows what is in front and on its side.

It does that through 20 sensors that each can detect an object that is at most 10 units away

(it does not know where the gas station is) or how the map looks like

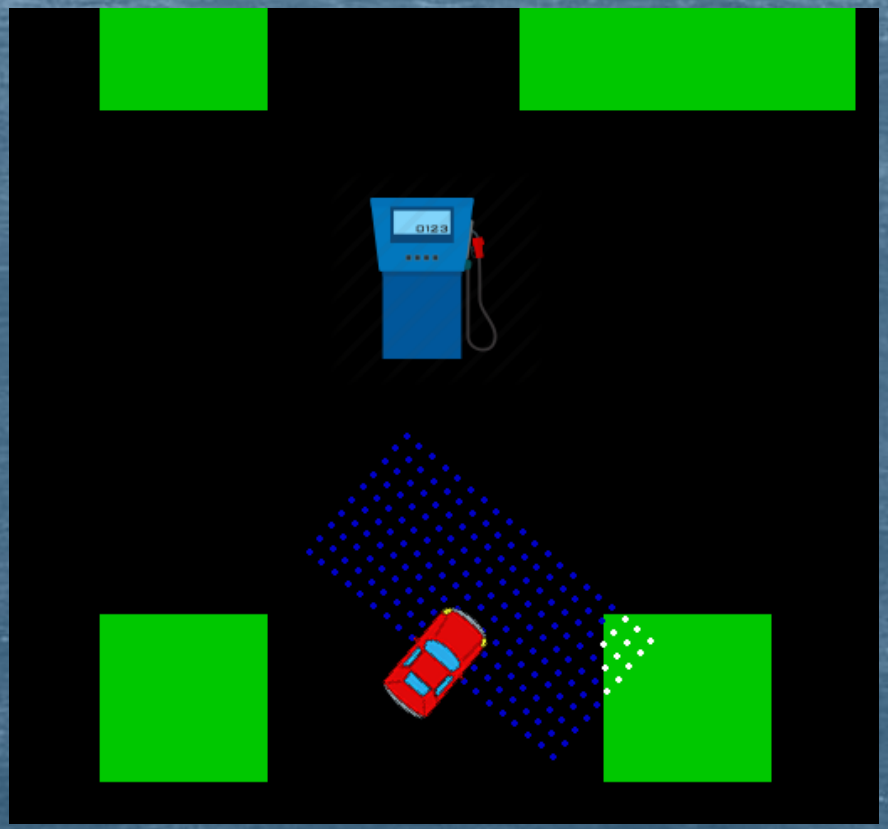
(This is more realistic)



Example

Actions:

The car always moves. It can, however, decide to move left , right or do nothing (move forward)



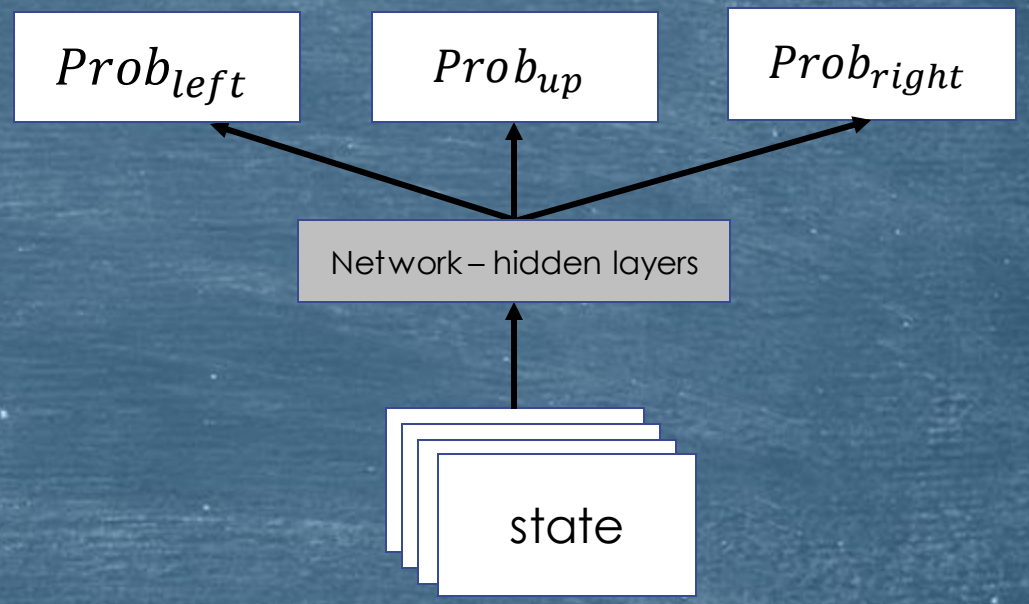
Gradient Policy Algorithm

We'll define a policy network that defines the agent (car)

The network will take the state of the agent (10 sensors) and will decide whether to move left, right or do nothing (up).

We'll use a stochastic policy. Meaning that the outputs will be probabilities

(in fact we'll use log probabilities, since the math is easier)



Gradient Policy Algorithm

Like in the neural network used for q learning, the problem is of how to choose the right label.

We want to increase : $E_{x\sim p(x|\theta)}[f(x)]$ which means:

The estimation of the function $f(x)$ (which is the reward function) under some probability distribution $p(x;\theta)$ ($p(x)$ is actually our policy function).

We are interested in how to adjust θ such that the expectation will increase (f(x) will increase)

Gradient Policy Algorithm

Like in the neural network used for q learning, the problem is of how to choose the right label.

We need to compute the gradient of the estimation

$$\nabla_\theta E_x[f(x)] = \nabla_\theta \sum_x p(x) f(x) \qquad \text{(definition of expectation)}$$

$$= \sum_x \nabla_\theta p(x) f(x)$$

$$= \sum_x p(x) \frac{\nabla_\theta p(x)}{p(x)} f(x) \qquad \text{(multiply and divide by p(x))}$$

$$= \sum_x p(x) \nabla_\theta log p(x) f(x) \qquad \text{(use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z$$

$$= E_x[f(x) \nabla_\theta \log p(x)] \qquad \text{(definition of expectation)}$$

Gradient Policy Algorithm

Like in the neural network used for q learning, the problem is of how to choose the right label.

$$\nabla_\theta E_x[f(x)] = E_x[f(x)\nabla_\theta \log p(x)]$$

This tells us that in order to increase the expectation of the reward function we have to :

1. Get some samples x
2. Compute the gradient $\nabla_\theta log p(x)$
3. Multiply this direction by $f(x)$

- More exactly, in order to change the network parameters, we have to

  Take some actions using a probability, evaluate the actions (get the reward), multiply the reward by its gradient and add the actions together

Gradient Policy Algorithm

Computing the reward

As in Q-learning, the reward should be discounted in the future.
Thus, as in Q learning, the reward should be:

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

where γ is the discount factor

r is the reward obtained after perfuming the action at time $t + k$

Gradient Policy Algorithm

Computing the reward

The problem is that the reward is only computed after an episode finished.

If the episode length is very long, than the reward will have a very high variance.

One way we could solve this is to standardize the results (i.e. z-score). This way we'll increase the probability of better than average results and decrease the probability of the others.

$$z = \frac{x - \mu}{\sigma}$$

Gradient Policy Algorithm

Training procedure detailed:

We'll first initialize the weights with some random values (as in normal neuronal networks)

We'll then generate some episodes using the forward pass of the neural networks (we'll predict the actions the network thinks are right)

On some of the episodes, the car will hit an obstacle. On others, the car will get to the gas station

Gradient Policy Algorithm

Training procedure:

We'll compute the discounted rewards for every action that was taken.

$$R_{i-1} = R_i * \gamma + r_{i-1}$$

We'll standardize the rewards. (compute how better it is than average).

Compute the gradient. If for example, the action that was taken was left, then we'll use 1 for the target value of this action , and leave the other intact. The gradient we'll be (1-probability of the action taken)

Gradient Policy Algorithm

Training procedure:

For each action taken, we'll multiply the gradient with the reward

We'll back propagate and compute the gradients for every parameter of the network.

We'll add this gradients together

We'll use this gradient to adjust the network

Questions & Discussion

References

- http://www2.econ.iastate.edu/tesfatsi/RLUsersGuide.ICAC2005.pdf
- https://www.nervanasys.com/demystifying-deep-reinforcement-learning/
- https://gym.openai.com
- http://karpathy.github.io/2016/05/31/rl/