CS231n: CNNs for Visual Recognition **Spring 2017**

Lecture 14

Course Coordinator: Prof. Fei-Fei Li *Scribes: Akash Gupta*

Reinforcement Learning - Problems involving agent interacting with the environment, which provides numeric reward signals. Goal - Learn how to take actions in order to maximize reward.

What is RL? -

- Agent and Environment.
- Environment provides agent a state.
- Agent receives the state and takes some action.
- Environment provides agent it's next state and reward for the action taken.

– > E.g. - Cart-Pole problem, Robot locomotion, Atari games, Go (Famous incident where DeepMind's AlphaGo beat no. 1 ranking player Lee Sedol)

– > Atari games -

1. Objective: Complete the game with the highest score.
2. State: Raw input pixels of the game state.
3. Action: Game controls e.g. Left, Right, Up, Down.
4. Reward: Some increase/decrease at each time step.

Markov Decision Process - Mathematical formulation of RL problem. E.g. - Grid World problem

– > Markov Property: Current state completely characterises the state of the world.
– > Defined by: (S, A, R, P, γ)

1. S: set of possible states
2. A: set of possible actions
3. R: distribution of reward given (state, action) pair
4. P: transition probability i.e. distribution over next state given (state, action) pair
5. γ: discount factor - how much value rewards coming soon vs later on.

– > Episode is the simulation of the events or a sequence of states, actions and rewards which leads to a terminal state.

– > A policy π is a function from S to A that specifies what action to take in each state.

– > Objective - find optimal policy π^* that maximizes cumulative discounted reward: $\sum_{t>0} \gamma^t r_t$. Formally,

$$\pi^* = argmax_\pi E[\sum_{t \geq 0} \gamma^t r_t | \pi]$$

with $s_0 \sim p(s_0)$, $a_t \sim \pi(.|s_t)$, $s_{t+1} \sim p(.|s_t, a_t)$

$->$ Some definitions -

- How good is a state? - Value function
- How good is a state-action pair? - Q-value function
- Bellman equation -
$$Q^*(s,a) \;=\; E_{s' \sim \epsilon}[r + \gamma \, max_{a'} Q^*(s', a')|s, a]$$
Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair.
- The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*.

Solving for optimal policy -

- Value iteration algorithm - Problem - Not scalable. Have to compute $Q(s,a)$ for every state action pair. Not possible in case of huge state space.
- Function approximator to estimate $Q(s,a) -> $ neural network $->$ also called Q-learning or deep Q-learning.
- Q-learning - Iteratively trying to make Q-value closer to a target value.
  - Define a Q-network that takes in current states and outputs corresponding Q-value for different actions taken on that state.
  - Issue with above training of Q-network -
    - Consecutive samples are correlated leading to inefficient learning.
    - Current Q-network params determines next training samples. Might introduce biases.
  - Solution - Experience Replay- Keep an updated replay memory of transitions as episodes are played $->$ Train Q-network on random minibatches of transitions from replay memory.
  - Still some issues with Q-learning - Q-function can be very complicated!
- Policy Gradients - The idea is to learn a policy directly i.e. finding a best policy from a collection of policies.
  - For each policy, define it's value:
$$J(\theta) \;=\; E[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta]$$
  - For optimal policy $\theta^* \;=\; argmax_\theta J(\theta)$
  - REINFORCE algorithm to solve for optimal policy.
  - Gradient estimator for one trajectory -
$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta log \pi_\theta(a_t|s_t)$$
- if $r(\tau)$ is high, push up the probabilities of the actions seen
- if $r(\tau)$ is low, push down the probabilities of the actions seen.
- This suffers from high variance. Use variance reduction.
- Various ways to do variance reduction -

- Push up the probabilities of an action seen, only by cumulative future reward from that state.
- Use discount factor γ to ignore delayed effects.
- Use Baseline. Choose baseline using constant moving average of rewards experienced so far or use Q-functions and Value functions.
- Learn Q and V using Q-learning.
- Combine Q-learning with Policy gradients $->$ Actor-Critic Algorithm. Actor(Policy) and Critic(Q-function). So actor decides which action to take and critic tells how good that action was using the Advantage function

REINFORCE in action: Recurrent Action Model (RAM) -

- Objective: Image Classification
- Take a sequence of "glimpses" selectively focusing on regions of the image to predict class.
- Glimpsing is a non-differentiable operation $->$ learn policy for how to take glimpse actions using REINFORCE. Given state of glimpses so far, use RNN to model the state and output next action.

AlphaGo - Mix of supervised learing, RL, Monte Carlo Search and recent ones (deep RL)

Summary: -

- Policy Gradients - Very general. Suffer from high variance so require a lot of samples. Challenge $->$ sample-efficiency. Guaranteed to converge to a ocal minima $J(\theta)$.
- Q-learning - does not always work out but when it works more sample efficient. Zero guarantee since your are approximating Bellman equation with a complicated function approximator.