Lecture Notes for

Deep Learning and Artificial Intelligence

Winter Term 2018/2019

Policy Gradients and Actor Critic Learning

DBS

Policy-Based Reinforcement Learning

So far: We approximated the utility or the state-value function using parameters θ:

$$\mathrm{f}(s, \theta) \approx U(S)$$

$$\text{or } \mathrm{f}(s, a, \theta) \approx Q(S, A)$$

Policy was generated by taking the action with the highest value. (e.g. ε-greedy)

Idea: Directly learn what to do, i.e., learn a policy:

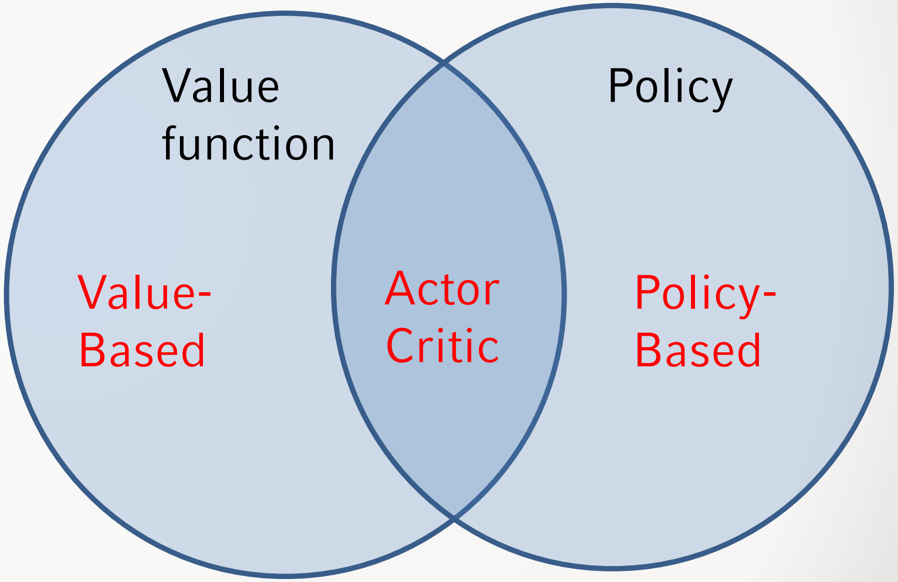
$$\pi_\theta(s, a) = \mathbb{P}[a|s, \theta]$$

where θ optimizes some performance measure $\mathrm{J}(\theta)$

(again we still work on model-free reinforcement learning)

Value-Based and Policy Based RL

- Value Based
  - Learn value function
  - Implicit policy (e.g. ε-greedy)
- Policy-Based
  - no value function
  - Learnt policy
- Actor-Critic
  - Learnt value function
  - Learnt policy



Advantages and Disadvantages

- Advantages of Policy Gradient methods
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
  - Constraints on parameters allow to include prior knowledge
- Disadvantages
  - Often converge to local rather than global optimum
  - Evaluating a policy is inefficient and high variance

Stochastic Policies

Example: Rock-Scissors-Paper

Rules: Each players picks one.

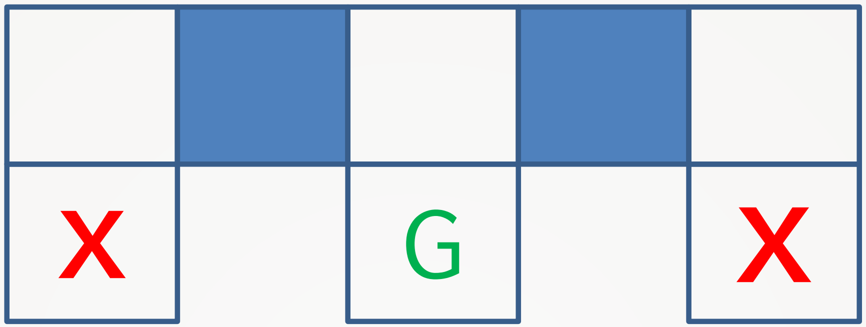
- Rock shatters scissors
- Paper covers rock
- Scissors cut paper
- Both players picking the same => draw

$\Rightarrow$ Playing a deterministic policy makes you predictable

$\Rightarrow$ Given that your opponents adapts to your actions, the best policy is pick a random action with a uniform distribution (Nash equilibrium)

(What is the optimal policy if you add a fourth action (e.g. well with scissors and rock fall into the well, but paper covers the well)

Example: Aliased Gridworld



- Partial observability: features describe whether there is a wall in N,E,S,W. => agent does not know in which blue state it is.
- A deterministic policy would either:

  always go right or ***always go left***

  $\Rightarrow$ depending on the start state the agent might get stuck
- a stochastic policy sometimes would take the other direction

$\Rightarrow$ at some time it would escape and go back to the middle

(What would ε–greedy do?)

Advantages of stochastic policies

- Allows for better convergence
  - If optimal policy is deterministic, the stochastic policy may converge against it.
  - Whereas greedy policies may jump due to the max functions, stochastic policies learnt with policy gradient smoothly converge into each other.
  - Stochastic policies naturally explore all options until you derive a zero likelihood for some (s,a)
- If the same state description should not always we handled in the same way:
  - antagonistic environments
  - partial observability: if the right action for an observation is ambivalent, switching actions might be better

How to train Policy Functions

- $\pi_\theta(s, a)$ describes a policy, i.e., all describable policies depend on θ
- To find the best parameters θ^*, we need a way to compare policies
- **Idea**: We can still measure the overall return of following a policy.

$\Rightarrow$ Define objective function $J(\theta)$ measuring the expected return of following $\pi_\theta(S, A)$.

Depends on the type of Environment:

- Episodic - Use the expected return of the **start state** s_1:
$$J_1(\theta) = U^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[U(s_1)]$$
- Continuing - Use **average return**:
$$J_{avU}(\theta) = \sum_s \mu^{\pi_\theta}(s)\, U^{\pi_\theta}(s)$$

or **average reward per time-step**:
$$J_{avU}(\theta) = \sum_s \mu^{\pi_\theta}(s) \sum_a \pi_\theta\,(s, a) R_s^a$$

where $\mu^{\pi_\theta}(s)$ is the stationary distribution of the Markov chain for π_θ.

Optimizing the policy

- Finding the best θ is the optimization problem maximizing $\mathrm{J}(\theta)$.

$\Rightarrow$ Any optimization method is applicable

Non-gradient methods:

- Hill climbing
- Simplex algorithms
- Genetic algorithms

Gradient based methods:

- Gradient descent /ascent
- Conjugate gradient
- Quasi-Newton

We will stick to gradient descent/ascent for the lecture.

Score Function

- Assume that the policy function $\pi_\theta(s, a)$ is differentiable whenever it is non-zero and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$.
- e.g. $\pi_\theta(s, a)$ is a neural network
- We will exploit the following reformulation to define likelihood ratios:

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta log \pi_\theta(s, a)\end{aligned}$$

- $\nabla_\theta log \pi_\theta(s, a)$ is called score function

Softmax Policy

- If we have a discrete action set (e.g. Atari), we need to predict a multinomial distribution over the actions

$\Rightarrow$ We can use a softmax function over |A| outputs.

- We weight the action using a linear combination $\phi(s,a)^T\theta$

- The probability of action a is proportional to the exponential weight:

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^T\theta}$$

- The score function is

$$\nabla_\theta log\pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

Gaussian Policy

- For continuous action spaces, we can employ a Gaussian policy
- Assuming the mean is a linear combination of state features

$$\mu(s) = \phi(s)^T \theta$$

- Variance σ^2can be assumed as either fixed or also a weighted combination as well
- A Gaussian Policy samples action a from a Gaussian over the action space:

$$a \sim \mathcal{N}(\mu(s), \sigma^2)$$

- The corresponding score function looks like:

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

Motivation: One-Step MDP

Before we will generally optimize policies, we examine an easier case:

- Consider a one-step MDP where:
  - Starting state is samples from $s \sim \mu(s)$
  - Episode ends after taking on step with reward $r = R_{s,a}$
- Use the likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$
$$= \sum_{s \in S} \mu(s) \sum_{a \in A} \pi_\theta(s,a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} \mu(s) \sum_{a \in A} \pi_\theta(s,a) \nabla_\theta log \pi_\theta(s,a) R_{s,a}$$
$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) r]$$

Policy Gradient Theorem

- We now turn to the general multi-step MDPs
- The policy gradient theorem generalizes our observation on the one-step case to multiple steps

$\Rightarrow$ We have to replace the immediate reward $R_{s,a}$ with action-value complete return $Q^{\pi}(s,a)$

- The policy gradient theorem holds for all three cases:

start state objective, average reward, average reward per time step

Theorem:

For any differentiable policy $\pi_\theta(s,a)$ and for any of the policy objective function $J = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avU}$ the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) Q^{\pi}(s,a)]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Update parameters by stochastic gradient ascent using the policy gradient theorem
- Using return G_t as an unbiased sample of $Q^{\pi}(s,a)$

$$\Delta\theta_t = \alpha \nabla_\theta log \pi_\theta(s,a) G_t$$

function REINFORCE
  Initialise θ
    for each episode$\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
      for $t = 1$ to $T - 1$ **do**
          $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta (s_t, a_t) G_t$
      end for
    end for
    return θ
end function

Discussion Monte-Carlo Policy Gradient

- Tackles some short-comings of value function approximation:
  - Learns stochastic policies
  - Can cope with continouos action spaces
- But suffers from the common problems of MC methods:
  - have to see the return before updating
  - might suffer from high variance
  - slow convergence

Reducing Variance using a Baseline

Idea: Do not compute the expected return of following π_θ but the change compared to a baseline utility?

Example: Consider stochastic routing and the direct path as the baseline.

⇒ gradient increases with the distance to the target because G_t is multiplied

⇒ a choice reducing the distance to the target by 100m generated different gradients depending on the distance to the target

⇒ subtracting a measure for the remaining distance to the target reduces the variance of the gradients for difference states

- Formally: Compare performance relative to a baseline performance B(s) which is not dependent on any action a.

$$\nabla J(\theta) \propto \sum_s \mu \sum_a \big(q_\pi(s,a) - b(s)\big) \nabla\pi(a|s,\theta)$$

- The equations remain valid because the subtracted quantity is zero:

$$\sum_a b(s)\nabla\pi(a|s,\theta) = b(s)\sum_a \nabla\pi(a|s,\theta) = b(s)\nabla 1 = 0$$

General Baseline Function

- the baseline B(s) is an estimate of the U(s)
- In most applications there is not natural baseline

Idea: We can employ a value-function approximation as in the last lecture to learn a baseline.

- Since we have episodic experience in REINFORCE, we can integrate MC Control to learn a baseline during policy optimization
- For every episode we can improve the baseline simultanously with the episode
- Errors in the baseline do not add to the policy because it only influence the impact of the gradient but not its direction (see previous slide)

REINFORCE with Baseline

for an episodic MDPs, a differentiable policy function $\pi(a|s,\theta)$, a differentiable state-value function U(s,ω) and learning rates α_θ and α_ϑ

function REINFORCEwithBaseline
  Initialise θ, ϑ
    for each episode$\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
      for $t = 1$ to $T - 1$ **do**

$$\boldsymbol{G \leftarrow \sum_{j=t}^{T} \gamma^{j-t} R_j}$$

$$\delta \leftarrow G - \mathrm{U}(s, \vartheta)$$

$$\omega \leftarrow \omega + \alpha_\vartheta \nabla_\vartheta \mathrm{U}(s_t, \vartheta)$$

$$\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \log \pi_\theta (s_t, a_t)\delta$$

      end for
    end for
    return θ
end function

Actor-Critic Learning

Can we avoid MC learning and employ Boostraping?

- We need an estimate of the future to bootstrap
- Directly learning the policy does not provide such an estimate

Idea: Use value function approximation to learn such an estimate.
The value function estimate is called Critic: $Q_\omega(s,a) \approx Q^{\pi_\theta}(s,a)$

Actor-critic algorithms maintain two sets of parameters:

critic: learn an action-value function to predict future rewards. (we will name critic parameters ω)

actor: learns a stochastic policy, based on the estimates of the critic. (we will name actor parameters θ)

Actor-critic algorithm follow the approximate policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log\pi_\theta(s,a) Q_\omega(s,a)]$$

$$\Delta\theta = \alpha \nabla_\theta log\pi_\theta(s,a) Q_\omega(s,a)$$

Action-Value Function approximation

- To evaluate θ we can use the methods for value-function approximation from the last lecture:
  - Monte-Carlo policy evaluation
  - Temporal-Difference Learning
  - TD(λ)
- Optimization of the critic, e.g. with least squares policy evaluation

- What makes Actor-Critic algorithm different from value-function approximation?
  - No greedy/ε-greedy policy
  - Instead policy is learned based on the action-value function approximation

Action-Value Actor-Critic

- Basic actor-critic algorithm based on action-value critic
- Uses linear value function approx. $Q_\omega(s,a) = \phi(s,a)^T\omega$

  Critic: Updates ω by linear TD(0)

  Actor: Updates θ by policy gradient

```
Function QAC
   init s,θ
   sample a~π_θ
   for each step do
      sample reward r = R_s^a; sample transition s'~P_s^a.
      sample action a' = π_θ(s',a')
      δ = r + γQ_ω(s',a')-Q_ω(s,a)
      θ = θ + α∇_θ logπ_θ(s,a)Q_ω(s,a)
      ω ← ω + βδϕ(s,a)
      a ← a',s ← s'
   end for
end function
```

Discussion Actor-Critic

- Using approximations to score return reduces variance but increases bias
- A biased policy gradient might not find a good policy

$\Rightarrow$ Value function has to be chosen carefully

$\Rightarrow$ Bias can be avoid, i.e., we follow the exact gradient

$\Rightarrow$ Absolute value of the gradient is again increasing the gradient for states with larger expectations

Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions hold,

1. Value function approximator is compatible to the policy

$$\nabla_\omega Q_\omega(s,a) = \nabla_\theta \log \pi_\theta\,(s,a)$$

2. Value function parameters minimize the mean-squared error

$$\epsilon = \mathbb{E}\left[\left(Q^{\pi_\theta}(s,a) - Q_\omega(s,a)\right)^2\right]$$

then the policy gradient is exact,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\,\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\omega(s,a)]$$

Proof Compatibility Theorem

If ω is chosen to minimize the mean-squared error ϵ then the gradient of ϵ, $\nabla_\omega \epsilon = 0$.

$$\nabla_\omega \epsilon = 0$$

$$\nabla_\omega \mathbb{E}\left[\left(Q^\theta(s,a) - Q_\omega(s,a)\right)^2\right] = 0 \qquad \text{(cond.2)}$$

$$\mathbb{E}\left[\left(Q^\theta(s,a) - Q_\omega(s,a)\right) \nabla_\omega Q_\omega(s,a)\right] = 0$$

$$\mathbb{E}\left[\left(Q^\theta\right)(s,a) - Q_\omega(s,a) \nabla_\theta \log \pi_\theta(s,a)\right] = 0 \text{ (cond.1)}$$

$$\mathbb{E}\left[Q^\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)\right] = \mathbb{E}[Q_\omega(s,a) \nabla_\theta \log \pi_\theta(s,a)]$$

Thus, in this case the gradient of J(θ) is exactly:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_\omega(s,a)]$$

Baseline Functions and Actor-Critic

- in reinforce with baseline we tried to minimize the impact of a state to the relative improvement of an action
- We learned a state-value function to find a generic baseline

Can we integrate the baseline into the actor critic framework?

- We learn a state-value function $U^{\pi_\theta}(s)$
- in addition to the critic $Q^{\pi_\theta}(s,a)$
- Which allows us to define the advantage function:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - U^{\pi_\theta}(s)$$

- And optimize :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Optimizing the Advantage function

- Using an advantage function can significantly reduce the variance in the policy gradient
- So the critic should rather learn the advantage function
- In a basic case, we estimate both $Q^{\pi_\theta}(s,a)$ and $U^{\pi_\theta}(s)$
- By two function approximators and with different parameter sets

$$U_\vartheta(s) \approx U^{\pi_\theta}(s)$$

$$Q_\omega(s,a) \approx Q^{\pi_\theta}(s,a)$$

$$A_{\vartheta,\omega}(s,a) = Q_\omega(s,a) - U_\vartheta(s)$$

- To optimize, we can build the gradients on both and learn based on ,e.g. , TD learning

TD Targets for estimating A(s,a)

- Given the true utility $U^{\pi_\theta}(s)$, the TD error:

$$\delta^{\pi_\theta} = r + \gamma U^{\pi_\theta}(s')\text{-}U^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|s,a] &= \mathbb{E}_{\pi_\theta}[r + \gamma U^{\pi_\theta}(s')|s,a] - U^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - U^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

$\Rightarrow$ We can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\delta^{\pi_\theta}]$$

Without knowing $U^{\pi_\theta}(s)$ we have to approximate the TD error by

$$\delta_\vartheta = r + \gamma U_\vartheta(s')\text{-}U_\vartheta(s)$$

$\Rightarrow$ We do not need to explicitely learn $Q_\omega(s,a)$ and only need to optmize δ_ϑ by SGD on ϑ

Critics at Different Time-Scales

- Critic can estimate value functions $U^{\pi_\theta}(s)$ based on various target functions with different time scales..
  - For MC, the target is the return G_t

$$\Delta\theta = \alpha\big(G_t - U^{\pi_\theta}(s)\big)\phi(s)$$

  - For TD(0), the target is the TD target r+$U^{\pi_\theta}(s')$

$$\Delta\theta = \alpha\big(r + U^{\pi_\theta}(s') - U^{\pi_\theta}(s)\big)\phi(s)$$

  - For the forward view of TD(λ), the target is the λ–return

$$\Delta\theta = \alpha\left(G_t^\lambda - U^{\pi_\theta}(s)\right)\phi(s)$$

  - For the backward view of TD(), we can employ eligibility traces

$$\delta_t = r_{t+1} + \gamma U^{\pi_\theta}(s_{t+1}) - U^{\pi_\theta}(s_t)$$

$$e_t = \gamma\lambda e_{t-1} + \phi(s)$$

$$\Delta\theta = \alpha\delta_t e_t$$

Actors at Different Time-Scale

- The policy gradient can also be estimated at many time-scales

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

- MC policy gradient uses the error from the complete return

$$\Delta\theta = \alpha\big(G_t - U^{\pi_\theta}(s_t)\big)\nabla_\theta \log \pi_\theta(s_t,a_t)$$

- Actor-critic policy gradient uses the one-step TD-error

$$\Delta\theta = \alpha\big(r + U^{\pi_\theta}(s_{t+1}) - U^{\pi_\theta}(s_t)\big)\nabla_\theta \log \pi_\theta(s_t,a_t)$$

- Using the forward view of TD(λ), is straight forward

$$\Delta\theta = \alpha\left(G_t^\lambda - U^{\pi_\theta}(s_t)\right)\nabla_\theta \log \pi_\theta(s_t,a_t)$$

Policy Gradient with Eligibility Traces

- As before we want to use eligibility traces to implement the backward view.

- By equivalence with TD(λ), we can substitute $\phi(s) = \nabla_\theta \log \pi_\theta (s, a)$

$$\delta = r_{t+1} + \gamma U^{\pi_\theta}(s_{t+1}) - U^{\pi_\theta}(s_t)$$
$$e_{t+1} = \lambda e_t + \nabla_\theta \log \pi_\theta (s, a)$$
$$\Delta\theta = \alpha \delta e_t$$

- Applying this update can be done online and on incomplete sequences

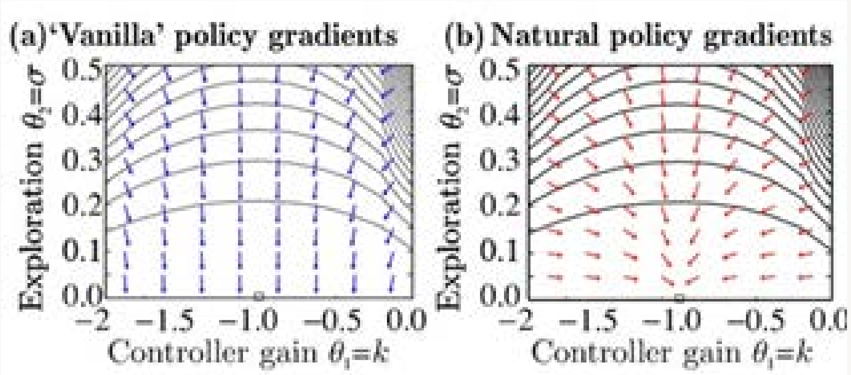
Alternative Policy Gradient Directions

- Gradient ascent can follow any ascending direction
- Finding a “good” direction can significantly speed up training (c.f. GD with individual learning rates)
- A policy might be reparametrized without changing the action probabilities. (multiple θ’s might lead to the same vector of π_θ)

  (e.g. increasing all values by the same amount in softmax policy)

$\Rightarrow$ The vanilla gradient is sensitive to this reparametrisations

Natural Policy Gradient



Lecture notes D. Silver: Introduction to Reinforcement Learning (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)

Lecture 7: Policy Gradient (Slide 36)

Idea: To make the gradient more parameter independent, we examine the relative size of gradients to each other.

⇒ The Fisher information matrix is the covariance of the score-function: $G_\theta = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta (s,a) \nabla_\theta \log \pi_\theta (s,a)^T]$

⇒ i.e. it measures the uncertainty of our estimate and can be used to decorrelate the parameter space:

$$\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \log \pi_\theta (s,a)$$

Natural Actor-Critic

Using compatible function approximations,

$$\nabla_\omega A_\omega(s,a) = \nabla_\theta \log \pi_\theta (s,a)$$

the natural policy gradient simplifies to,

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta (s,a) A_\omega(s,a)] \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta (s,a) \nabla_\theta \log \pi_\theta (s,a)^T \omega] \\ &= G_\theta \ \omega\end{aligned}$$

$$\Rightarrow \quad \nabla_\theta^{nat} J(\theta) = \omega$$

$\Rightarrow$ In this case we only need to update the actor parameters in direction of the critic parameters

Summary Policy Gradient Methods

- Policy gradient methods learns a function π_θ which provides a stochastic policy

$\Rightarrow$ π_θ can be continuously differentiable in θ in and thus, shows better convergence than ε-greedy policies

- to optimize π_θ we maximize the objective function J(θ)
- J(θ) can be directed by observed rewards (REINFORCE)
- to apply the idea of bootstraping, we need am estimate of the remaining future

$\Rightarrow$ Learn a critic using a method of value function approximation like MC,TD, TD(λ) to guide the actor π_θ

$\Rightarrow$ to assure convergence actor and critics must be compatible (Compatible Function Approximation Theorem)

Further Directions in RL

Reinforcement learning is an currently active field of research because many of the applications of AI required the machine to act on its own.

Topics not covered in this lecture:

- integrating learning and planning
- inverse reinforcement learning
- imitation learning

Integrating Learning and Planning

General setting in RL applications:

- often no queryable environment available or physical interaction to slow or dangerous.

$\Rightarrow$ build simulation based on real observation

- What is the difference between the MDP and a simulation?

$\Rightarrow$ we can learn an MDP based on available experience and "physical" background knowledge

$\Rightarrow$ use the MDP to simulate new experience

$\Rightarrow$ learn the MDP and use planning to optimize the policy

Integrating Learning and Planning

Methods in this direction include:

- DynaQ: Combines sampled and real experience
  - learns Q(S,A) and a MDP Model (s,a)
  - updates Q-values and the model based on real experience
  - additionally, generate samples from Model and further optimize Q-values ("imaginary experience")
- Monte-Carlo Tree Search:
  - generates a forward search tree for the current state
  - applies different policies to generated complete episodes
  - tries to find good estimates for the next actions to take
  - particularly useful for games with high branching factors such as GO

Inverse Reinforcement Learning

- in the setting so far, we observe an “suboptimal” policy and the reward the agent receives
- Inverse Reinforcement Learning: We have assume some optimality in the agents actions but do not see the reward. Thus, we want to estimate the reward function for the observed agent.
- only usable if the agent acts somewhat consistent to optimize a certain reward
- directed into understanding real agents instead of optimizing a known goal

Imitation Learning

- often we do not want to model an optimal agent but an agent behaving as human as possible

$\Rightarrow$ reward is based on acting like an observed agent

- applications: Sports-Games build agents acting like particular players, build more human like chat-bots

- Task can be considered supervised learning predicting the next action of the teacher
- interesting in combination to inverse reinforcement learning but not the same.

$\Rightarrow$ If I can predict the next action based on the observed state observation, I can reason about the the variables being maximized

Literature

- Lecture notes D. Silver: Introduction to Reinforcement Learning (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)
- S. Russel, P. Norvig: Artificial Intelligence: A modern Approach, Pearson, 3rd edition, 2016
- R. S. Sutton, A. G. Barto: **Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning),** The MIT Press; Auflage: 2., 2018