Reinforcement Learning

A Quick Introduction

Nir Ben-Zvi

June 11, 2017

Outline

Outline

Why is RL Cool?

- RL has been getting a lot of wins in academy in recent years
- Like many other branches of ML which got the 'deep' treatment
- We'll later see why the two fuse together nicely

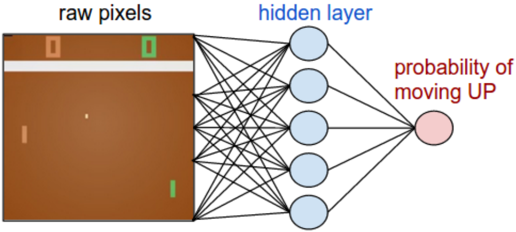
What Makes It Different?

- Unformally, RL is about processes (or actions)
- It's not strictly supervised or unsupervised learning
- Instead, it's about learning how to interact with a certain process
- Games are one of the famous examples

Examples

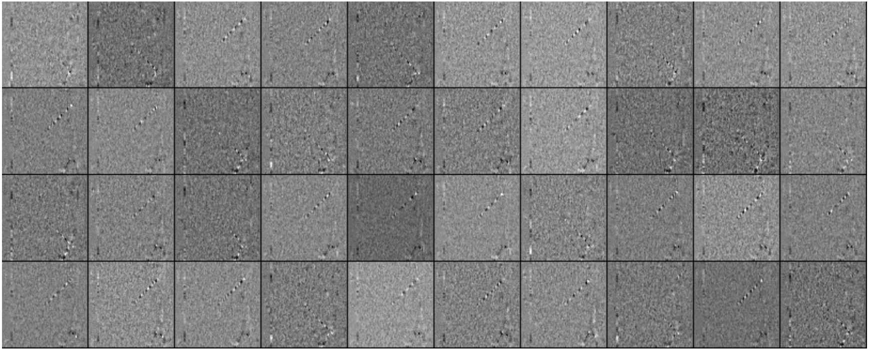
- Atari Games (DeepMind)
- Pong (From Karpathy's Blogpost)

Example: Learning Pong



taken from Karpathy's Blogpost

Example: Learning Pong



taken from Karpathy's Blogpost

Other Non-Game Uses

- Visual Attention Models (Mnih et Al, now DeepMind)
- StarCraft (DeepMind)
- Robotics
- Neural Turing Machines (DeepMind) and followup works
- Alpha Go and games

Outline

Agent Environment Interaction

Reinforcement Learning usually includes an enviroment and an agent

- For the purpose of this intro lecture assume a single agent
- Environment/agent combos can be a casino/gambler, maze/robot etc.

Agent Environment Interaction

Reinforcement Learning usually includes an enviroment and an agent

- For the purpose of this intro lecture assume a single agent
- Environment/agent combos can be a casino/gambler, maze/robot etc.
- RL is about learning how to interact with the environment with accorance to timesteps
- Many approaches exist, and we'll present a few today

Agent Environment Interaction, cont.

- The agent learns what to do so as to maximize a numerical reward
- The learner is not told which actions to take, but instead must discover which actions yield high rewards from experience
- Rewards are not immediate
- These two characteristics - *trial-and-error search* and *delayed reward* are the two most important distinguishing features of RL

Agent Environment Interaction, cont.

- The agent learns what to do so as to maximize a numerical reward
- The learner is not told which actions to take, but instead must discover which actions yield high rewards from experience
- Rewards are not immediate
- These two characteristics - *trial-and-error search* and *delayed reward* are the two most important distinguishing features of RL

Outline

Reinforcement Learning Approaches

An RL agent usually includes one or more of the following:

- Model: agent's understanding of the true environment
- Value Function: how good is each state and/or action
- Policy: the agent's behavior function

Outline

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model
- We won't discuss this today

Outline

Background; Markov Chains

- A Markov Chain (or Markov Process) is a memoryless random process - i.e. a sequence of random states $s_1, s_2, \ldots$ which abide the *Makov Property*

Background; Markov Chains

- A Markov Chain (or Markov Process) is a memoryless random process - i.e. a sequence of random states $s_1, s_2, \ldots$ which abide the *Makov Property*

Makov Property

A state s_t is *Markov* if and only if:

$$\mathbb{P}\left[s_{t+1} \mid s_t\right] = \mathbb{P}\left[s_{t+1} \mid s_1, \ldots, s_t\right]$$

Background; Markov Chains

- A Markov Chain (or Markov Process) is a memoryless random process - i.e. a sequence of random states $s_1, s_2, \ldots$ which abide the *Makov Property*

Makov Property

A state s_t is *Markov* if and only if:

$$\mathbb{P}\left[s_{t+1} \mid s_t\right] = \mathbb{P}\left[s_{t+1} \mid s_1, \ldots, s_t\right]$$

- In other words; the state captures all relevant information about the past
- When a state is known, previous events are meaningless

State Transition Matrix

- The Markov State transition matrix defines a Markov Process;

$$\mathcal{P}_{ss'} = \mathbb{P}\left[s_{t+1} = s' \mid s_t = s\right]$$

- The State Transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s':

$$\mathbb{P} = \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \vdots \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

- ...such that rows and columns sum to one
- Also called *doubly stochastic*

Markov Process, Revisited

We'll now define a Markov Process more rigorously:

Markov Process, Revisited

We'll now define a Markov Process more rigorously:

Makov Property

A Markov Process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix

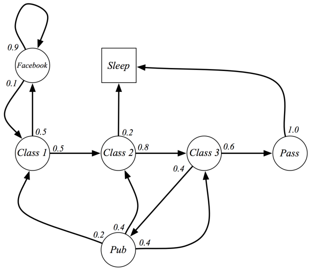
Markov Process, Revisited

We'll now define a Markov Process more rigorously:

Makov Property

A Markov Process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix



Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system
- $a \in \mathcal{A}$ – agent's action

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system
- $a \in \mathcal{A}$ – agent's action
- $\mathcal{P} = p(s'|s, a)$ – the dynamics of the system

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system
- $a \in \mathcal{A}$ – agent's action
- $\mathcal{P} = p(s'|s,a)$ – the dynamics of the system
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – the reward function (possibly stochastic)

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system
- $a \in \mathcal{A}$ – agent's action
- $\mathcal{P} = p(s'|s,a)$ – the dynamics of the system
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – the reward function (possibly stochastic)

Definition

MDP is the tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$

Reinforcement Learning Framework

Agent/Environment interaction in Reinforcement Learning is usually modeled as a *Markov Decision Process* (MDP from now on)

- $s \in \mathcal{S}$ - state of the system
- $a \in \mathcal{A}$ – agent's action
- $\mathcal{P} = p(s'|s,a)$ – the dynamics of the system
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ – the reward function (possibly stochastic)

Definition

MDP is the tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{A}, \mathcal{R} \rangle$

- $\pi(a|s)$ - the agent's policy
- A discount function/factor γ also usually exists

Why Discount?

- Avoids infinite returns in cyclic Markov Chains
- In a lot of real life scenarios, immediate rewards may be of more interest than delayed awards
- ...as is common human/animal bahavior

Reinforcement Learning Framework

- It's common to break agent interaction into *episodes*
- Each episode beigns with state s_0, drawn from some distribution $\mu(s_0)$ and ends at some terminal state
- An action a_t will be chosen by the agent with accordance to the policy $\boldsymbol{\pi}$
- The next state is sampled according to $\mathcal{P}$ presented earlier; $\mathbb{P}(s_{t+1} \mid s_t, a_t)$
- Taking rewards into account, this becomes:

$$\mathbb{P}(s_{t+1}, r_t \mid s_t, a_t)$$

- The process continues until a terminal state is reached

Trajectories and Episodes

- It's also common to introduce *trajectories*
- A trajectory τ is simply a series of states ending at some timestep T

$$\tau = \{(a_t, s_t)\}_{t=1}^{T}$$

- A trajectory can potentially contain multiple episodes

Trajectories and Episodes

- It's also common to introduce *trajectories*
- A trajectory τ is simply a series of states ending at some timestep T

$$\tau = \{(a_t, s_t)\}_{t=1}^{T}$$

- A trajectory can potentially contain multiple episodes
- Those terms get confused a lot in the literature (and for iterative tasks are often interchangeable)

Outline

The Value Function

- A value function is a prediction of future award
- Sometimes called *state-value function* - we ask "how much reward will I get from state s_t onwards?"
- The un-discounted value function for a policy π:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t) \mid s_0 = s\right]$$

- Where T denotes the *horizon*

The Value Function

- A value function is a prediction of future award
- Sometimes called *state-value function* - we ask "how much reward will I get from state s_t onwards?"
- The un-discounted value function for a policy π:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{T} r(s_t, a_t) \mid s_0 = s\right]$$

- Where T denotes the *horizon*
- The dicsounted value function is then:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma(t) r(s_t, a_t) \mid s_0 = s\right]$$

- Where typically γ is simply a scalar $0 < \gamma < 1$ so that $\gamma(t) = \gamma^t$

The Value Function - Goal

- Our goal is to maximize our value function; performed by finding the policy π^* which maximizes it $\forall s \in \mathcal{S}$:

$$V^*(s) = V^{\pi^*} = \max_{\pi} V^{\pi}(s)$$

Outline

The Bellman Equation

- The value function can be written recursively:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \mid s_0 = s\right]$$
$$= \mathop{\mathbb{E}}_{\substack{a\sim\pi(\cdot|s) \\ s'\sim p(\cdot|s,a)}} \left[r(s,a) + \gamma V^\pi(s')\right]$$

- The optimal value satisfies the Bellman equation:

$$V^*(s) = \max_\pi \mathop{\mathbb{E}}_{\substack{a\sim\pi(\cdot|s) \\ s'\sim p(\cdot|s,a)}} \left[r(s,a) + \gamma V^\pi(s')\right]$$

- Proof exists in literature

Q-Function

- The Q-function gives the expected total reward of
  - from state s_t and action a_t
  - under policy π
  - with discount factor γ

$$Q(s,a) = r(s,a) + \gamma \underset{s' \sim p(\cdot s,a)}{\mathbb{E}} \left[V(s') \right]$$

Q-Function

- The Q-function gives the expected total reward of
  - from state s_t and action a_t
  - under policy π
  - with discount factor γ

$$Q(s,a) = r(s,a) + \gamma \underset{s' \sim p(\cdot s,a)}{\mathbb{E}} \left[V(s') \right]$$

- If we know V^*, the optimal policy is to deterministically progress:

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- This implies a very important relationship between V and Q:

$$V^*(s) = \max_a Q^*(s,a)$$

Optimal Value Functions

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a)$$

- ...at *all* states:

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} + \dots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Again, we get the Bellman equation:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Outline

Fully Observeable Setting

- The framework described above pertain to the *fully-observable setting* - where the agent can observe the full dynamics of the system
- A more powerful framework exists, in which the agent only has access to an observation o_t at a given timestep

Fully Observeable Setting

- The framework described above pertain to the *fully-observable setting* - where the agent can observe the full dynamics of the system
- A more powerful framework exists, in which the agent only has access to an observation o_t at a given timestep
- In this instance, the true state s_t becomes a summary of experience:

$$s_t = f(o_1, r_1, a_1, \dots, a_{t-1}, o_t, r_t)$$

- If $f(o_t) = s_t$, this reduces back to the observable setting
- The more powerful framework is called a *Partially Observable Markov Decision Process* (POMDP)

POMDPs

- Solving POMDPs hard!

POMDPs

- Solving POMDPs hard!
- POMDP can be viewed as a natural extension of HMMs similarly to defining MDPs though Markov Chains
- One way of reducing a POMDP to an MDP is by defining a timeframe of last k steps as the agents current state - denoted *history* in the literature
- This results in:

$$a_{t+1} \sim \pi(a_t \mid \mathbf{h}_t)$$

- One common way of modeling $\mathbf{h}_t$ in practice is to use an RNN
- We'll stick to MDPs today

Outline

Policy

- The Policy is the agent's behavior
- Usually represented by a mapping from current state to action - $\boldsymbol{\pi}$
  - Deterministic: $a_t = \pi(s_t)$
  - ...or Stochastic: $\pi(a \mid s) = \mathbb{P}(a \mid s)$
- Policy-based RL is about searching directly for an optimal policy π^*
- This is the policy which achieves the maximum future reward

Outline

RL Methods by Type

RL is exploding in current research, but most methods used are pretty old. We will focus on two popular methods;

- Q-Learning - learn a value function which satisfies the Bellman equations
- Policy Gradient - attempt to directly learn a policy maximizing rewards (using an analytic iterative solution, hence the "gradient")

RL Methods by Type

RL is exploding in current research, but most methods used are pretty old. We will focus on two popular methods;

- Q-Learning - learn a value function which satisfies the Bellman equations
- Policy Gradient - attempt to directly learn a policy maximizing rewards (using an analytic iterative solution, hence the "gradient")
- These are all *model-free* methods

Model-Free Methods

- For the Atari example of DQN, states are 64×64 RGB images and there are 4 actions

Model-Free Methods

- For the Atari example of DQN, states are 64×64 RGB images and there are 4 actions
- The transition dynamics $\mathcal{P}$ are of size:

$$|S \times S \times A| \approx (4.68 \times 10^{1954}) \times (4.68 \times 10^{1954}) \times 4$$

Model-Free Methods

- For the Atari example of DQN, states are 64×64 RGB images and there are 4 actions
- The transition dynamics $\mathcal{P}$ are of size:

$$|S \times S \times A| \approx (4.68 \times 10^{1954}) \times (4.68 \times 10^{1954}) \times 4$$

- Model-free implies we make no effort to learn the underlying dynamics of the environment
- Instead, we estimate the policy/value-function directly by interacting with the environment

RL Methods by Type

Many other methods exist:

- Value Based- try to learn a value function satisfying Bellman Equation
  - Q-Learning, Double-Q-Learning
  - Temporal Difference (TD) Learning
  - SARSA
- Policy Search- attempt to directly learn a policy maximizing rewards
  - Gradient Based
    - Policy Gradient methods (Natural Policy Gradients, REINFORCE)
    - Actor-Critic algorithms (AC3)
  - Gradient-Free
    - Simulated Annealing
    - Cross-Entropy Search

Outline

Learning Optimal Q^*

- If the agent knows the dynamics p and the reward function r, it can find Q^* by dynamic programming
  - Many methods exist!
  - ...but they are useless if we don't know the dynamics, or if the state-space is huge
- Otherwise, it needs to estimate Q^* from its experience
  - The *experience* of an agent is a sequence $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$
  - The n-th time-step is (s_n, a_n, r_n, s_{n+1})

Simulation-Based Q-Value Iteration

- Lets interact with the environment for T timesteps and obtain a trajectory $(s_0, a_0, \ldots, s_T)$ according to policy $\pi^{(n-1)}$
  - $\pi^{(n-1)}$ can be based on Q_{n-1}
- Now update $Q_n(s,a)$ to better represent our sample

Off-Policy vs. On-Policy

- The update can use some a' drawn from *some* policy
- I.e., it doesn't depend on what action we actually took at state s_{t+1}
- This allows us to sample our environment w.r.t. any policy - not only the one we are learning (e.g. $\pi^{(n)}$)
- This is called an *off-policy* approach

Moving to an Online Algorithm

- The general algorithm works for batches/entire tracjectories
- What if we go with $n = 1$?

Moving to an Online Algorithm

- The general algorithm works for batches/entire tracjectories
- What if we go with $n = 1$?
- We'll receive an *online* algorithm instead!
- Lets take an estimate of the environment based only on current state and move forward
- This results in the Q-Learning algorithm [WatkinsDayan92]

Q^*-Learning Algorithm

The full Q-Learning algorithm:

Q-Learning

input: learning rate $\alpha \in (0, 1)$
initialize: $Q_0(s, a)$ for all $a \in \mathcal{A}$ and $s \in \mathcal{S}$
for each time-step n **do**
  observe the current state s_n
  select and execute an action a_n
  receive reward r_n
  observe the next sate s_{n+1}
  let:

$$Q_n(s_n, a_n) \leftarrow (1 - \alpha_n) Q_{n-1}(s_n, a_n) + \alpha_n \left(r_n + \gamma \max_a Q_{n-1}(s_{n+1}, a) \right)$$

end for

Update Step Explained

Lets disect the update step:

Q-Learning

$$Q(s_n,a_n) \leftarrow \underbrace{(1-\alpha_n)Q_{n-1}(s_n,a_n)}_{\text{old value}} + \underbrace{\alpha_n}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_n}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q_{n-1}(s_{n+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} \right)$$

Why Not V-Learning?

- One might as why we don't see V-Learning in the literature
- Q-values make actions explicit
- Thus, they work when the transition function is not available
- As is usually the case for most interesting problems

Exploration vs. Exploitation

- How should the agent gain experience in order to learn?
  - If it explores too much it might only run into bad options
  - If it exploites learned Q function too early (or too frequently) it might not learn about better ones!
- This is the RL *Exploration vs. Exploitation* problem
- ϵ-greedy policy - try a new action regardless of current learned policy with probability ϵ

Credit Assignment Problem

- How does one know which action contributed to a high award for a given episode?
  - An action can have an effect in the far future
- This is the RL *Credit Assignment* problem
- Different solutions exist; decaying reward function is one
- The hand-wavvy reasoning behind this is that repeating the same task hundreds of thousands of times eliminates the issue

Q-Learning - Wrap Up

- Q-Learning algorithm has been proven to find an optimal result for finite MDPs [Watkins and Dayan, 92]
- Model free - only uses Q-function and not system dynamics $\mathbb{P}(s_{t+1}|s_t, a_t)$
- Difficult to adapt to non-Markovian settings
  - Model as a POMDP instead?
  - Partial observability makes the learning problem much harder and often intractable
- Also problematic for continuous or very large action-state spaces
  - Function approximators assist there
- Fuses beautifully with NNs (what would be the loss?)

Outline

Learning the Policy Directly

- Optimize a policy end-to-end by computing an estimate of the *gradient of the expected reward of the policy*
- Assume a stochastic policy $\mu(a_t \mid s_t)$ - giving a prob. distribution over actions

Learning the Policy Directly

- Optimize a policy end-to-end by computing an estimate of the *gradient of the expected reward of the policy*
- Assume a stochastic policy $\mu(a_t \mid s_t)$ - giving a prob. distribution over actions
- Optimally, examples with high rewards for *good actions* and low rewards for *bad actions* would result in increasing probability of good action
- Vanilla PG runs into a lot of problems and is rarely used in practice
    - REINFORCE is a popular solution

Learning the Policy Directly

- Broadly speaking, if our policy is chosen w.r.t to parameter-set θ, we want $\max_\theta \mathbb{E}[R \mid \theta]$
  - Where R is the total reward of en episode
  - ...and actions are chosen from $\pi(a_t \mid s_t; \theta)$

Learning the Policy Directly

- Broadly speaking, if our policy is chosen w.r.t to parameter-set θ, we want $\max_\theta \mathbb{E}[R \mid \theta]$
  - Where R is the total reward of en episode
  - ...and actions are chosen from $\pi(a_t \mid s_t; \theta)$
- Find gradient of policy w.r.t. current parameter-set θ
- Optimize using SGD

Credit Assignment Problem, Again

- We should all be familiar with MLE optimization of this form:

$$\max_{\theta} \sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

Credit Assignment Problem, Again

- We should all be familiar with MLE optimization of this form:

$$\max_{\theta} \sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

- Where we maximize the log-probability of outputs y_n for inputs x_n

Credit Assignment Problem, Again

- We should all be familiar with MLE optimization of this form:

$$\max_{\theta} \sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

- Where we maximize the log-probability of outputs y_n for inputs x_n
- Here we instead wish to optimize action a_t for states s_t
- Had we known a^* - the optimal action for each state; we could simply optimize:

$$\max_{\theta} \sum_{n=1}^{N} \log p(a_n^* \mid s_n; \theta)$$

Credit Assignment Problem, Again

- We should all be familiar with MLE optimization of this form:

$$\max_{\theta} \sum_{n=1}^{N} \log p(y_n \mid x_n; \theta)$$

- Where we maximize the log-probability of outputs y_n for inputs x_n
- Here we instead wish to optimize action a_t for states s_t
- Had we known a^* - the optimal action for each state; we could simply optimize:

$$\max_{\theta} \sum_{n=1}^{N} \log p(a_n^* \mid s_n; \theta)$$

- Alas, we don't

Outline

Approximating Good vs Bad Actions

- For a given trajectory
  $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- Make a 'guess' at which actions were good and which weren't
- Increase probability of good actions repeating

Approximating Good vs Bad Actions

- For a given trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- Make a 'guess' at which actions were good and which weren't
- Increase probability of good actions repeating
- Let $R = \sum_{t=0}^{T-1} r_t$ the sum of rewards

Approximating Good vs Bad Actions

- For a given trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$
- Make a 'guess' at which actions were good and which weren't
- Increase probability of good actions repeating
- Let $R = \sum_{t=0}^{T-1} r_t$ the sum of rewards
- Assuming we can get $\nabla_\theta \mathbb{E}[R]$ we are done!

Getting a Derivative of the Policy

- We need to get a derivative for the policy function w.r.t. to some weights θ
- We don't actually know which actions resulted in outcomes
- We will use some math trickery to get a gradient-descent step which updates actions w.r.t. their 'usefulness'
- Lets name this mythical gradient $\hat{g}$

Training in Practice

- Assume the policy is some parametric model whose parameter set is θ
- Interact with environment 100 times, with 200 steps for each such interaction
- In total, we have 20,000 environment interactions
- Update every interaction which resulted in a 'bad' outcome as 'bad' and vice-versa
- This doesn't make sense for a short period, but does for a long series of such interactions

Formal Explanation

- Idea based on *score function gradient estimator*
- Used for expressions of the form $\mathbb{E}_{x \sim p(x|\theta)}[f(x)]$

Formal Explanation

- Idea based on *score function gradient estimator*
- Used for expressions of the form $\mathbb{E}_{x \sim p(x|\theta)}[f(x)]$
    - Expectation of scalar valued *score function* f
    - For x drawn from p w.r.t θ
    - In our case, f is our reward function (was R earlier)
    - ...and p is the policy $\pi(a \mid s; \theta)$

Formal Explanation

- Idea based on *score function gradient estimator*
- Used for expressions of the form $\mathbb{E}_{x \sim p(x|\theta)}[f(x)]$
  - Expectation of scalar valued *score function* f
  - For x drawn from p w.r.t θ
  - In our case, f is our reward function (was R earlier)
  - ...and p is the policy $\pi(a \mid s; \theta)$
- Math!

Likelihood Ratio Trick

Likelihood Ratio Trick

$$
\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_x[f(x)] &= \nabla_\theta \sum_x p(x;\theta) f(x) && \text{def. of expectation} \\
&= \sum_x \nabla_\theta p(x;\theta) f(x) && \text{insert gradient} \\
&= \sum_x p(x;\theta) \frac{\nabla_\theta p(x;\theta)}{p(x;\theta)} f(x) && \text{multiply and divide by } p(x;\theta) \\
&= \sum_x p(x;\theta) \nabla_\theta \log p(x;\theta) f(x) && \text{note } \nabla_\theta \log(z) = \frac{1}{z}\nabla_\theta z \\
&= \mathop{\mathbb{E}}_x[\nabla_\theta \log p(x;\theta) f(x)] && \text{def. of expectation}
\end{aligned}
$$

Likelihood Ratio Trick

Likelihood Ratio Trick

$$
\begin{aligned}
\nabla_\theta \mathop{\mathbb{E}}_x[f(x)] &= \nabla_\theta \sum_x p(x;\theta) f(x) && \text{def. of expectation} \\
&= \sum_x \nabla_\theta p(x;\theta) f(x) && \text{insert gradient} \\
&= \sum_x p(x;\theta) \frac{\nabla_\theta p(x;\theta)}{p(x;\theta)} f(x) && \text{multiply and divide by } p(x;\theta) \\
&= \sum_x p(x;\theta) \nabla_\theta \log p(x;\theta) f(x) && \text{note } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= \mathop{\mathbb{E}}_x[\nabla_\theta \log p(x;\theta) f(x)] && \text{def. of expectation}
\end{aligned}
$$

$\nabla_\theta \log p(x;\theta)$ is called the *score function*.

Formal Explanation, Cont.

- Using the equality $\nabla_\theta \mathbb{E}_x[f(x)] = \mathbb{E}_x[\nabla_\theta \log p(x;\theta) f(x)]$
- We can sample values $x \sim p(x;\theta)$ and compute the LHS (over N samples)
- Hence; $\hat{g} = \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \log p(x_i;\theta) f(x_i)$
- Full derivation in the literature!

Problems

- The estimator $\hat{g}$ is generally very noisy
- Instead of increasing the probability of *good episodes* - we would like to increase the probability of *good actions*
- This is called *gradient bias* in the literature
- Various solutions exist
- PG usually converges to a local maxima (unlike Q-Learning's guaranteed global)

Outline

REINFORCE

- REINFORCE $\rightarrow$ REward Increment $=$ Nonnegative Factor times Offset Reinforcement times Characteristic Eligibility
- It turns out that we can improve the above formula by lowering the variance of gradient-estimates

REINFORCE

- REINFORCE $\rightarrow$ REward Increment $=$ Nonnegative Factor times Offset Reinforcement times Characteristic Eligibility
- It turns out that we can improve the above formula by lowering the variance of gradient-estimates
- Add stability to the training process
- In practice, REINFORCE often converges where vanilla PG won't

REINFORCE Algorithm

REINFORCE

initialize: θ at random
for each trajectory $\tau = \{s_1, a_1, r_1, \ldots, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
  for $t = 1 \ldots T-1$ **do**
    let: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot v_t$
  end for
end for

Note that v_t represents a stabilized reward and substitutes R in the vanilla PG

Why Is 'Stabilization' Important?

- A simple example would be to normalize (e.g. subtract mean and divide by variance) of the computed reward R_t
- This would guarantee that the number of 'bad' and 'good' actions (in terms of gradient computations) would be equal

Outline

Q-Learning vs. Policy Gradients

	Q-Learning	REINFORCE
Learning objective	Value function	Policy parameters via gradient
Policy Stochasticity	Convergence to essentially deterministic policy	Explicitly stochastic
Model	Model free	Model free
Markovity Assumption	MDP\POMDP	Problem specific, not inherent
Support for continuous\large state space	Possible via function approximation, difficult to train.	Yes
Main Practical Successes	DeepMind Atari playing net	Visual Attention, robot control

- **Q-Learning better adapted to classic observable Markovian setting, PG-based learning more relevant the further problem is from that setting (hidden and continuous states).**

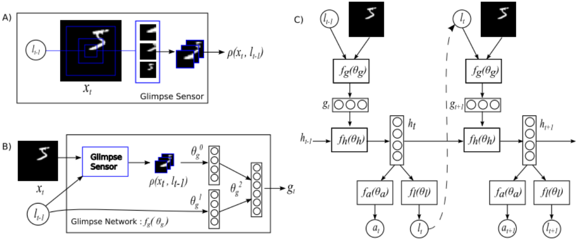
Outline

Policy Gradients in Neural Networks

- Policy Gradients are best paired with function approximators - particularly Neural Networks
  - ...this is also true for Q-Learning (e.g. *DQN*)
- For example, consider an RNN in which output represents action probabilities
  - Softmax output $a_t \in [k]$ for k actions
  - Hidden units of RNN θ_t are policy parameterization.
  - Weights frozen for duration of episode and updated at the end
  - Naturally compatible with backpropagation- $\nabla_\theta \log \pi_\theta (a_t|s_t)$ is the gradient of the corresponding RNN evaluated at timestep t

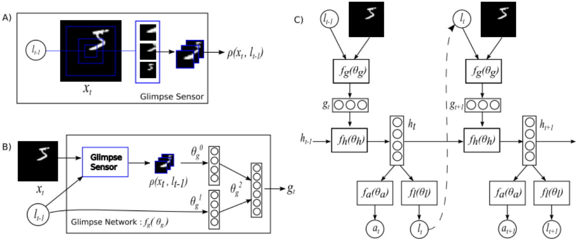
Outline

Recurrent Attention Model



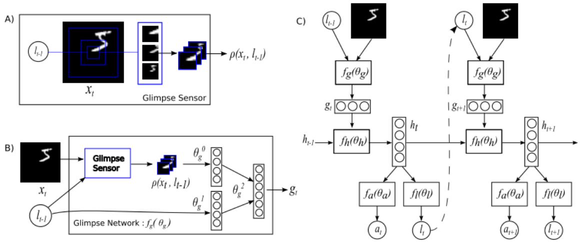
- Original paper by (Mnih et al., 2015)
- Policy parameterized by RNN
- At each step 2 types of actions (l_t glimpse location and a_t classification) controlled by 2 sub-networks
- Goal is to learn stochastic policy $\pi\left((l_t, a_t)\,|s_{1:t};\theta\right)$ maximizing rewards

Example: Recurrent Attention Model (Mnih et al., 2015)

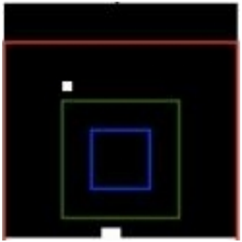


- Trajectory given by $s_{1:t} = x_1, l_1, a_1, ..., x_{t-1}, l_{t-1}, a_{t-1}, x_t$
- At each step 2 types of actions (l_t glimpse location and a_t classification) controlled by 2 sub-networks
- Reward $R = \sum_{t=1}^{T} r_t$ where $r_T = 1$ for correct classification and 0 otherwise

Example: Recurrent Attention Model (Mnih et al., 2015)



Example: Recurrent Attention Model (Mnih et al., 2015) - Dynamic Environment



http://www.cs.toronto.edu/~vmnih/docs/attention.mov

- Same approach used to train agent to play simple game in dynamic environment

Outline

Outline

Further Reading/Where I Stole This From

- Sutton, R. S., Mcallester, D., Singh, S., Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation.
- Karpathy, A. (2016). Deep Reinforcement Learning: Pong from Pixels (blogpost)
- Sutton, R. Barto, A. (1998). Reinforcement Learning: An Introduction (Book)
- Watkins, C.J.H., Dayan, P. (1992) Technical Note: Q-Learning
- OpenAI Gym RL Introduction
- Mnih. V et Al. (2014). Playing Atari with Deep Reinforcement Learning
- Grondman. I et Al. (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients

Further Reading/Where I Stole This From

...I also blatantly stole slides from:

- David Silver's RL Course from UCL
- David Silver's ICML2016 RL Workshop slides
- David Silver's NIPS2016 RL Workshop slides
- Generally anything by David Silver
- REINFORCE presentation by Ronen Tamary of HUJI
- Q-Learning presentation by Noga Zaslavsky of HUJI
  - Both presented at HUJI's own DL seminar - good stuff there!

Outline

Thank You