FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
SCHOOL OF ENGINEERING

Deep Reinforcement Learning

A. Maier, K. Breininger, L. Mill, N. Ravikumar, T. Würfl
Pattern Recognition Lab, Friedrich-Alexander-Universität Erlangen-Nürnberg
June 13, 2018

Outline

Sequential Decision Making

Reinforcement Learning

- Markov Decision Processes
- Policy Iteration
- Other Solution Methods

Deep Reinforcement Learning

- Deep Q Learning
- AlphaGo
- AlphaGo Zero

Sequential Decision Making

Sequential decision making: Multi-armed bandit problem

Action Formalize choosing a machine as **action** a at time t from a set A

Reward Action a_t has a **different**[1] **unknown pdf** $p(r|a)$ generating **reward** r_t

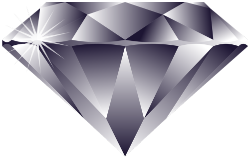
[1] This is not how gambling works

Sequential decision making: Multi-armed bandit problem

Action	Formalize choosing a machine as **action** a at time t from a set A
Reward	Action a_t has a **different**[1] **unknown pdf** $p(r \mid a)$ generating **reward** r_t
Policy	Formalize choosing an action a as pdf $\pi(a)$ which we call a **policy**

[1] This is not how gambling works

Evaluative Feedback

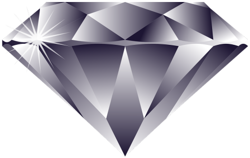


- Find action *a* producing the **maximum expected reward over time** *t*:

$$\max_a \mathbb{E}\, [p(r|a)]$$

- Difference to supervised learning: **No** feedback on **what** action to choose

Evaluative Feedback

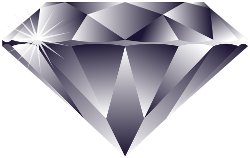


- Find action a producing the **maximum expected reward over time** t:

$$\max_a \mathbb{E}\left[p(r|a)\right]$$

- Difference to supervised learning: **No** feedback on **what** action to choose
- $\mathbb{E}\left[p(r|a)\right]$ is **not known in advance**

Evaluative Feedback

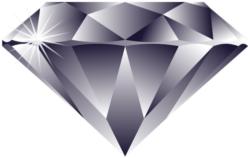


- Find action *a* producing the **maximum expected reward over time** *t*:

$$\max_a \mathbb{E}\left[p(r|a)\right]$$

- Difference to supervised learning: **No** feedback on **what** action to choose
- $\mathbb{E}\left[p(r|a)\right]$ is **not known in advance**
- We can form a one-hot encoded vector **r** which reflects which action from **a** caused the reward

Evaluative Feedback



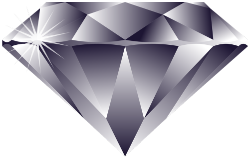
- Find action *a* producing the **maximum expected reward over time** *t*:

$$\max_a \mathbb{E}\left[p(r|a)\right]$$

- Difference to supervised learning: **No** feedback on **what** action to choose
- $\mathbb{E}\left[p(r|a)\right]$ is **not known in advance**
- We can form a one-hot encoded vector $\mathbf{r}$ which reflects which action from $\mathbf{a}$ caused the reward

➔ Estimate the joint pdf online as $\frac{1}{t}\sum_{i=1}^{t}\mathbf{r}_i := Q_t(\mathbf{a})$

Evaluative Feedback



- Find action a producing the **maximum expected reward over time** t:

$$\max_a \mathbb{E}\left[p(r|a)\right]$$

- Difference to supervised learning: **No** feedback on **what** action to choose
- $\mathbb{E}\left[p(r|a)\right]$ is **not known in advance**
- We can form a one-hot encoded vector $\mathbf{r}$ which reflects which action from $\mathbf{a}$ caused the reward

➔ Estimate the joint pdf online as $\frac{1}{t}\sum_{i=1}^{t}\mathbf{r}_i := Q_t(\mathbf{a})$

- We call $Q_t(\mathbf{a})$ the action-value function, which changes with every new information

Incremental update of $Q_t(\mathbf{a})$

$$
\begin{aligned}
Q_{t+1}(\mathbf{a}) &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{r}_i \\
&= \frac{1}{t}\left(\mathbf{r}_t + \sum_{i=1}^{t-1}\mathbf{r}_i\right) \\
&= \frac{1}{t}\left(\mathbf{r}_t + (t-1)\frac{1}{t-1}\sum_{i=1}^{t-1}\mathbf{r}_i\right) \\
&= \frac{1}{t}\left(\mathbf{r}_t + (t-1)Q_t(\mathbf{a})\right) \\
&= \frac{1}{t}\left(\mathbf{r}_t + t\,Q_t(\mathbf{a}) - Q_t(\mathbf{a})\right) \\
&= Q_t(\mathbf{a}) + \frac{1}{t}\left(\mathbf{r}_t - Q_t(\mathbf{a})\right)
\end{aligned}
$$

Exploitation

- Reward is maximized by a policy $\pi(a)$ choosing $\max_a Q_t(a)$
- We exploit a known good action
- This is a **deterministic**[1] policy called **greedy action selection**

[1] If Q_t is equal for two a, the tie has to be broken e.g. randomly

Exploitation

- Reward is maximized by a policy $\pi(a)$ choosing $\max_a Q_t(a)$
- We exploit a known good action
- This is a **deterministic**[1] policy called **greedy action selection**
- However we **need** to obtain **samples** r_a

[1] If Q_t is equal for two a, the tie has to be broken e.g. randomly

Exploitation

- Reward is maximized by a policy $\pi(a)$ choosing $\max_a Q_t(a)$
- We exploit a known good action
- This is a **deterministic**[1] policy called **greedy action selection**
- However we **need** to obtain **samples** r_a

➔ This means we **cannot follow** the greedy action selection policy for learning

[1] If Q_t is equal for two a, the tie has to be broken e.g. randomly

Exploitation

Exploration

- Reward is maximized by a policy $\pi(a)$ choosing $\max_a Q_t(a)$
- We exploit a known good action
- This is a **deterministic**[1] policy called **greedy action selection**
- However we **need** to obtain **samples** r_a

→ This means we **cannot follow** the greedy action selection policy for learning

→ Sometimes explore by selecting other moves which could potentially be better

[1] If Q_t is equal for two a, the tie has to be broken e.g. randomly

We sample discrete actions a from $\pi(a)$, but what distributions can we use?

We sample discrete actions a from $\pi(a)$, but what distributions can we use?

Uniform random

$$\pi(a) = \frac{1}{|A|}$$

- $|A|$ is the cardinality of the set of different actions A

We sample discrete actions a from $\pi(a)$, but what distributions can we use?

Uniform random

$$\pi(a) = \frac{1}{|A|}$$

- $|A|$ is the cardinality of the set of different actions A

Epsilon Greedy

$$\pi(a) = \begin{cases} 1 - \epsilon & \text{if } a = \max\limits_{a} Q_t(a) \\ \epsilon/(n-1) & \text{else} \end{cases}$$

We sample discrete actions a from $\pi(a)$, but what distributions can we use?

Uniform random

$$\pi(a) = \frac{1}{|A|}$$

- $|A|$ is the cardinality of the set of different actions A

Epsilon Greedy

$$\pi(a) = \begin{cases} 1 - \epsilon & \text{if } a = \max_a Q_t(a) \\ \epsilon/(n-1) & \text{else} \end{cases}$$

Softmax

$$\pi(a) = \frac{e^{Q_t(a)/\tau_t}}{\sum_{n=1}^{|A|} e^{Q_t(a_n)/\tau_t}}$$

- τ_t is called **temperature** and used to decrease exploration over time

Summary

So far we ...

- considered **sequential decision making** in a setting known as multi-armed bandits

Summary

So far we ...

- considered **sequential decision making** in a setting known as multi-armed bandits
- found out that **estimating a function** $Q(a)$ and the **greedy action selection** policy $\pi(a) = \max_a Q(a)$ maximized our reward

Summary

So far we ...

- considered **sequential decision making** in a setting known as multi-armed bandits
- found out that **estimating a function** $Q(a)$ and the **greedy action selection** policy $\pi(a) = \max_a Q(a)$ maximized our reward
- learned that **exploration** of different actions is necessary

Summary

So far we ...

- considered **sequential decision making** in a setting known as multi-armed bandits
- found out that **estimating a function** $Q(a)$ and the **greedy action selection** policy $\pi(a) = \max_a Q(a)$ maximized our reward
- learned that **exploration** of different actions is necessary
- assumed rewards **didn't depend** on a **state** of the world

Summary

So far we ...

- considered **sequential decision making** in a setting known as multi-armed bandits
- found out that **estimating a function** $Q(a)$ and the **greedy action selection** policy $\pi(a) = \max_a Q(a)$ maximized our reward
- learned that **exploration** of different actions is necessary
- assumed rewards **didn't depend** on a **state** of the world
- and our action at time t **doesn't influence** the **rewards** from a at $t+1$

Reinforcement Learning

Associativity

We extend the multi-armed bandits problem:

Associativity

We extend the multi-armed bandits problem:

- We introduces a state of the world at any time t: s_t
- Rewards now additionally **depend on** the **state** s_t:

$$p(r_t|s_t, a_t)$$

Associativity

We extend the multi-armed bandits problem:

- We introduces a state of the world at any time t: s_t
- Rewards now additionally **depend on** the **state** s_t:

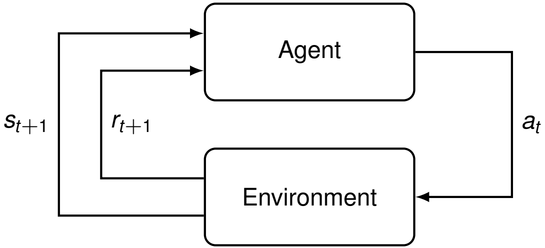
$$p(r_t | s_t, a_t)$$

- However this setting is known as **contextual bandit**
- In the full **reinforcement learning problem**, actions influence the state:

$$p(s_{t+1} | s_t, a_t)$$

Markov Decision Processes

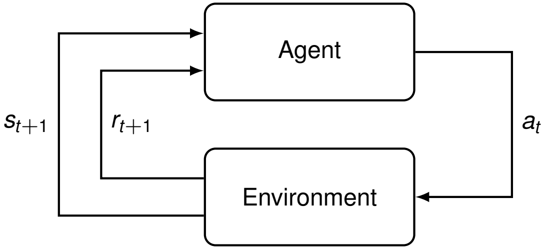
Markov Decision Process



Action An **action** a_t at time t from a set A

State A **state** s_t from a set S

Markov Decision Process

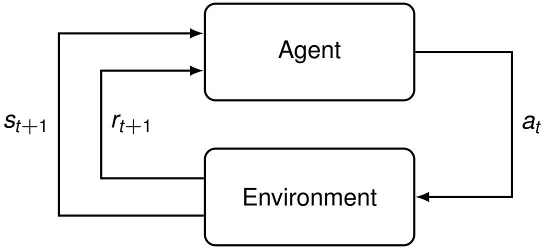


Action An **action** a_t at time t from a set A

State A **state** s_t from a set S

A **state transition pdf** $p(s_{t+1}|s_t, a_t)$

Markov Decision Process



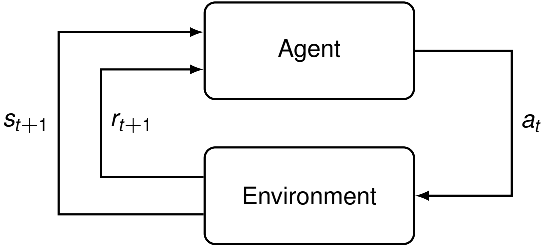
Action An **action** a_t at time t from a set A

State A **state** s_t from a set S

A **state transition pdf** $p(s_{t+1}|s_t, a_t)$

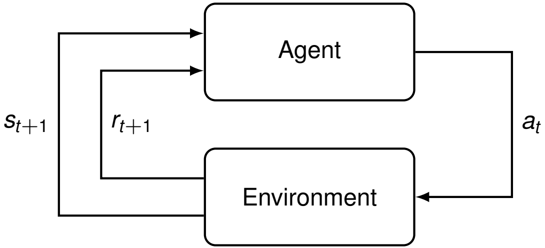
Reward Transition produces reward $r_{t+1} \in R \subset \mathbb{R}$ according to $p(r_{t+1}|s_t, a_t)$

Markov Decision Process



Action	An **action** a_t at time t from a set A
State	A **state** s_t from a set S
	A **state transition pdf** $p(s_{t+1} \mid s_t, a_t)$
Reward	Transition produces reward $r_{t+1} \in R \subset \mathbb{R}$ according to $p(r_{t+1} \mid s_t, a_t)$
Policy	Agents choose actions a_t by a policy $\pi(a \mid s)$

Markov Decision Process



Action An **action** a_t at time t from a set A

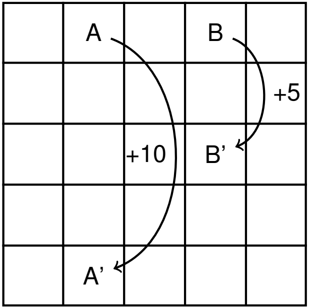
State A **state** s_t from a set S

A **state transition pdf** $p(s_{t+1}|s_t, a_t)$

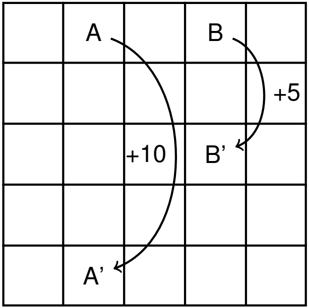
Reward Transition produces reward $r_{t+1} \in R \subset \mathbb{R}$ according to $p(r_{t+1}|s_t, a_t)$

Policy Agents choose actions a_t by a policy $\pi(a|s)$

If all those sets are **finite** we call this a **finite MDP**

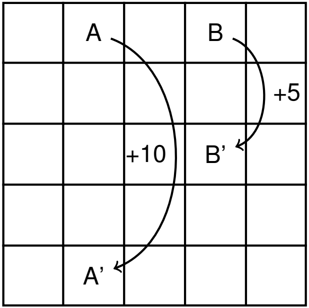


- ➔ Here s is the field we are currently on.
- The agent can **move** in all four directions
- Any action which would leave the grid has $p(s_{t+1}|a_t, s_t)$ equal to a δ distribution on $s_{t+1} = s_t$ and a similarly deterministic $r_t = -1$



→ Here s is the field we are currently on.

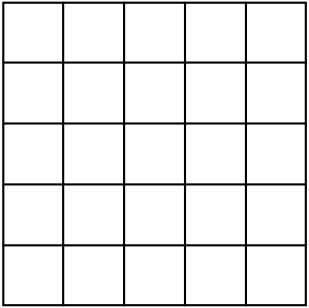
- The agent can **move** in all four directions
- Any action which would leave the grid has $p(s_{t+1}|a_t, s_t)$ equal to a δ distribution on $s_{t+1} = s_t$ and a similarly deterministic $r_t = -1$
- Every state we reach other than tile A' and B' deterministically causes $r_t = 0$



➔ Here s is the field we are currently on.

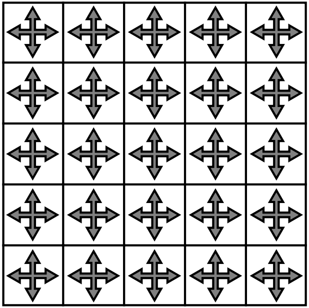
- The agent can **move** in all four directions
- Any action which would leave the grid has $p(s_{t+1}|a_t, s_t)$ equal to a δ distribution on $s_{t+1} = s_t$ and a similarly deterministic $r_t = -1$
- Every state we reach other than tile A' and B' deterministically causes $r_t = 0$
- On A or B **any** action will take us to A' or B' respectively

Example policy



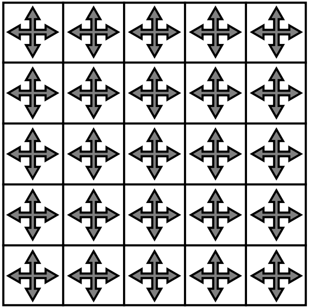
- Policies now depend on s_t
- We can extend the **uniform random policy** to be independent from s_t

Example policy



- Policies now depend on s_t
- We can extend the **uniform random policy** to be independent from s_t
- However there’s no reason to believe that this policy is any good

Example policy



- Policies now depend on s_t
- We can extend the **uniform random policy** to be independent from s_t
- However there's no reason to believe that this policy is any good
- How can we **estimate good policies**?

What is a good policy?

→ We have to be precise about good

- Preliminary we have to state **two kinds** of tasks
  1. **Episodic** tasks which have an **end**
  2. **Continuing** tasks which are **infinitely** long
- **Unify them** using a **terminal state** in **episodic tasks** which only transition to themselves with deterministic $r_t = 0$
- Goal is to **maximize the future return**

$$\max_{\pi(s_t, a_t)} g_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$$

- γ is a **discount** reducing influence of rewards **far** in the future
- $\gamma \in (0, 1]$ meaning that $\gamma = 1$ is allowed as long as $T \neq \infty$

Policy Iteration

- Before we used the action-value function $Q(a)$

- Before we used the action-value function $Q(a)$
- Now a_t has to depend on s_t

- Before we used the action-value function $Q(a)$
- Now a_t has to depend on s_t

→ Use an oracle predicting the future reward g_t following $\pi(s_t, a_t)$ from s_t

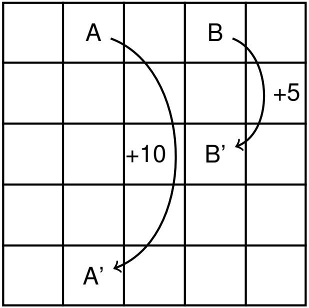
- Before we used the action-value function $Q(a)$
- Now a_t has to depend on s_t

➔ Use an oracle predicting the future reward g_t following $\pi(s_t, a_t)$ from s_t

- We introduce the **state-value function** $V_\pi(s)$

$$V_\pi(s) = \mathbb{E}_\pi\left[g_t | s_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k | s_t\right]$$

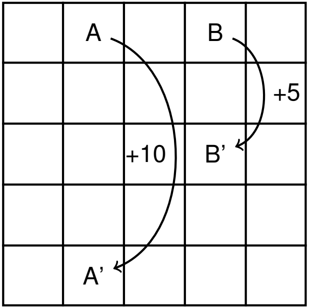
State-value Function Example



The definition of the gridworld

- Recall our grid example

State-value Function Example



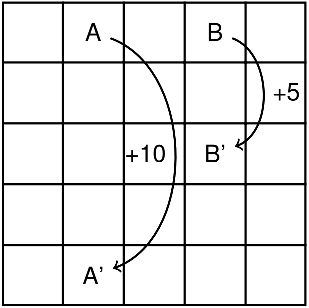
The definition of the gridworld

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$V_\pi(s)$ for the uniform random policy

- Recall our grid example
- Some **edge tiles** are negative since the policy **can't control the move**

State-value Function Example



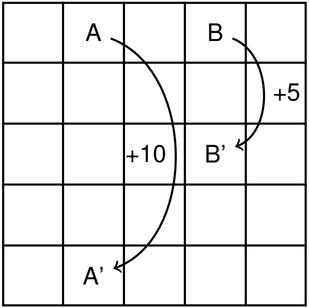
The definition of the gridworld

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$V_\pi(s)$ for the uniform random policy

- Recall our grid example
- Some **edge tiles** are negative since the policy **can’t control the move**
- What if we use the **greedy action selection** policy on this $V_\pi(s)$?

State-value Function Example



The definition of the gridworld

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$V_\pi(s)$ for the uniform random policy

- Recall our grid example
- Some **edge tiles** are negative since the policy **can’t control the move**
- What if we use the **greedy action selection** policy on this $V_\pi(s)$?
- We get a **better policy**!

Action-value function

- Before we used the action-value function $Q(a)$
- Now we introduced $V_\pi(s)$ filling a similar role

Action-value function

- Before we used the action-value function $Q(a)$
- Now we introduced $V_\pi(s)$ filling a similar role
- We can also introduce the **action-value function** $Q_\pi(s, a)$
- Basically this accounts for the **transition probabilities**

$$Q_\pi(s, a) = \mathbb{E}_\pi\left[g_t | s_t, a_t\right] = \mathbb{E}_\pi\left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k | s_t, a_t\right]$$

Are Value Functions Created Equal?

[1] in a finite MDP

Are Value Functions Created Equal?

- No.
- There can only be one[1] **optimal** $V^*(s)$

[1] in a finite MDP

Are Value Functions Created Equal?

- No.
- There can only be one[1] **optimal** $V^*(s)$
- We can state its existence without referring to a specific policy:

$$V^*(s) = \max_{\pi} V_{\pi}(s) \quad (1)$$

[1] in a finite MDP

Are Value Functions Created Equal?

- No.
- There can only be one[1] **optimal** $V^*(s)$
- We can state its existence without referring to a specific policy:

$$V^*(s) = \max_{\pi} V_{\pi}(s) \tag{1}$$

- $Q^*(s, a)$ can also be defined and is related to $V^*(s_t)$by:

$$Q^*(s, a) = \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1})\right] \tag{2}$$

[1] in a finite MDP

Optimal Value-function Example

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$V_\pi(s)$ for the uniform random policy

Optimal Value-function Example

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

$V_\pi(s)$ for the uniform random policy

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

V^*

- Observe that V^* is strictly positive since it's deterministic

Optimal Policies

- Policies can now be ordered: $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$

Optimal Policies

- Policies can now be ordered: $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$
- **Any** policy π with $V_\pi = V^*$ is an optimal policy π^*

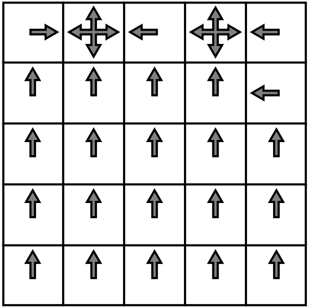
Optimal Policies

- Policies can now be ordered: $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$
- **Any** policy π with $V_\pi = V^*$ is an optimal policy π^*
- This implies there might be **more than one** optimal policy

Optimal Policies

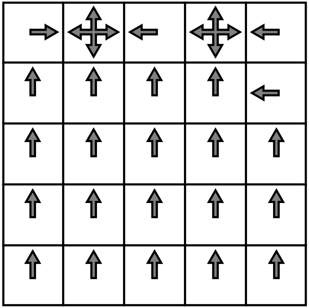
- Policies can now be ordered: $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s), \forall s \in S$
- **Any** policy π with $V_\pi = V^*$ is an optimal policy π^*
- This implies there might be **more than one** optimal policy
- Given either V^* or Q^* an optimal policy is directly obtained by **greedy action selection**

Greedy Action Selection on $V^*(s)$ or $Q^*(s, a)$

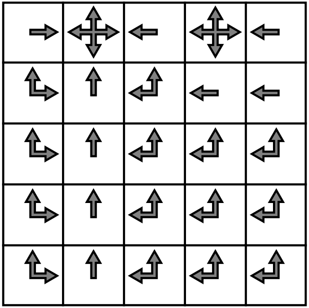


$\pi'(s, a) =$ Greedy Action Selection on $V_\pi(s)$ with $\pi(s, a)$ being uniform random

Greedy Action Selection on $V^*(s)$ or $Q^*(s, a)$



$\pi'(s, a) =$ Greedy Action Selection on $V_\pi(s)$ with $\pi(s, a)$ being uniform random



$\pi^*(s, a) =$ Greedy Action Selection on $V^*(s)$

A Tool to Compute Optimal Value-functions

- We **still need** to compute $V^*(s)$ and $Q^*(s, a)$

A Tool to Compute Optimal Value-functions

- We **still need** to compute $V^*(s)$ and $Q^*(s, a)$
- For this the **Bellman equations** can be utilized
- They are **consistency conditions** for the value functions

Bellman equation for $V_\pi(s)$

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s_{t+1}, r} p(s_{t+1}, r|s, a) \left[r + \gamma V_\pi(s_{t+1})\right]$$

Policy Evaluation

- The Bellman equations form a **system of linear equations** which can be solved for **small** problems
- Better: **Iteratively solve**, by turning the Bellman equations into **update rules**:

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s_{t+1},r} p(s_{t+1}, r|s, a)\left[r + \gamma V_k(s_{t+1})\right]$$

For all $s \in S$

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies
- Another necessary step is to **update the policy**

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies
- Another necessary step is to **update the policy**
- However if we use **greedy action selection** an update of $V_\pi(s)$ is **simultaneously** an update of $\pi(s)$

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies
- Another necessary step is to **update the policy**
- However if we use **greedy action selection** an update of $V_\pi(s)$ is **simultaneously** an update of $\pi(s)$
- Now iterate **evaluation** of the **greedy policy** on $V_\pi(s)$

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies
- Another necessary step is to **update the policy**
- However if we use **greedy action selection** an update of $V_\pi(s)$ is **simultaneously** an update of $\pi(s)$
- Now iterate **evaluation** of the **greedy policy** on $V_\pi(s)$
- Stop iterating if the **policy stops changing**

Policy Improvement

- $V_\pi(s)$ is used to **guide our search** for good policies
- Another necessary step is to **update the policy**
- However if we use **greedy action selection** an update of $V_\pi(s)$ is **simultaneously** an update of $\pi(s)$
- Now iterate **evaluation** of the **greedy policy** on $V_\pi(s)$
- Stop iterating if the **policy stops changing**
- But is this **guaranteed** to work?

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s)\,, \forall s \in S \implies \pi' \geq \pi$$

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s), \forall s \in S \implies \pi' \geq \pi$$

- This also implies:

$$V_{\pi'}(s) \geq V_\pi(s)$$

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s)\,, \forall s \in S \implies \pi' \geq \pi$$

- This also implies:

$$V_{\pi'}(s) \geq V_\pi(s)$$

- Because we **only select greedy** we have $Q_\pi(s, a) > V_\pi(s)$ before convergence

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s)\,, \forall s \in S \implies \pi' \geq \pi$$

- This also implies:

$$V_{\pi'}(s) \geq V_\pi(s)$$

- Because we **only select greedy** we have $Q_\pi(s, a) > V_\pi(s)$ before convergence
- So iteratively updating $V_\pi(s)$ and using **greedy action selection** is guaranteed to work here

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s), \forall s \in S \implies \pi' \geq \pi$$

- This also implies:

$$V_{\pi'}(s) \geq V_\pi(s)$$

- Because we **only select greedy** we have $Q_\pi(s, a) > V_\pi(s)$ before convergence
- So iteratively updating $V_\pi(s)$ and using **greedy action selection** is guaranteed to work here
- We terminate if the policy no longer changes

Policy Improvement Theorem

- We consider changing a single action a_t in state s_t but following π
- In general if

$$Q_\pi(s, \pi'(s)) \geq V_\pi(s), \forall s \in S \implies \pi' \geq \pi$$

- This also implies:

$$V_{\pi'}(s) \geq V_\pi(s)$$

- Because we **only select greedy** we have $Q_\pi(s, a) > V_\pi(s)$ before convergence
- So iteratively updating $V_\pi(s)$ and using **greedy action selection** is guaranteed to work here
- We terminate if the policy no longer changes
- Last remark: If we don't loop over all $s \in S$ for policy evaluation, but update the policy directly this algorithm is called **Value iteration**

Other Solution Methods

Limitations

- Both policy iteration and value iteration **require** using the **updated policies** during learning to obtain better approximations to $V^*(s)$

Limitations

- Both policy iteration and value iteration **require** using the **updated policies** during learning to obtain better approximations to $V^*(s)$
- For this reason we call them **on-policy** algorithms

Limitations

- Both policy iteration and value iteration **require** using the **updated policies** during learning to obtain better approximations to $V^*(s)$
- For this reason we call them **on-policy** algorithms
- Additionally we assumed the **state-transition** pdf and **reward** pdf are known

Limitations

- Both policy iteration and value iteration **require** using the **updated policies** during learning to obtain better approximations to $V^*(s)$
- For this reason we call them **on-policy** algorithms
- Additionally we assumed the **state-transition** pdf and **reward** pdf are known
- Can we relax this?

Limitations

- Both policy iteration and value iteration **require** using the **updated policies** during learning to obtain better approximations to $V^*(s)$
- For this reason we call them **on-policy** algorithms
- Additionally we assumed the **state-transition** pdf and **reward** pdf are known
- Can we relax this?
- Yes. The methods differ mostly **how they perform policy evaluation**

Monte Carlo Techniques

Properties

- Only for **episodic** tasks
- Off-policy - learns $V^*(s)$ by following any **arbitrary** $\pi(s, a)$
- Does **not need** information about dynamics of the environment

Monte Carlo Techniques

Properties

- Only for **episodic** tasks
- Off-policy - learns $V^*(s)$ by following any **arbitrary** $\pi(s, a)$
- Does **not need** information about dynamics of the environment

Scheme

- **Generate** an **episode** by using some policy
- Loop **backwards** over the episode accumulating the expected future reward $g_t = g_{t+1} + r_{t+1}$
- If a state was **not yet** visited append g_t to a list $returns(s_t)$
- Update $V_{s_t} = \frac{1}{N} \sum_{n=1}^{N} returns_n(s_t)$

Temporal Difference Learning

Properties

- On-policy
- Does **not need** information about dynamics of the environment

Temporal Difference Learning

Properties

- On-policy
- Does **not need** information about dynamics of the environment

Scheme

- Loop and follow $\pi(s_t, a_t)$
- Use a from $\pi(s_t, a_t)$, observe r_t, s_{t+1}
- Update: $V_{t+1}(s) = V_t(s) + \alpha \left[r_t + \gamma V_t(s_{t+1}) - V_t(s_t)\right]$

Temporal Difference Learning

Properties

- On-policy
- Does **not need** information about dynamics of the environment

Scheme

- Loop and follow $\pi(s_t, a_t)$
- Use a from $\pi(s_t, a_t)$, observe r_t, s_{t+1}
- Update: $V_{t+1}(s) = V_t(s) + \alpha\left[r_t + \gamma V_t(s_{t+1}) - V_t(s_t)\right]$

- **Converges to the optimal solution**
- A variant of this estimates $Q_{(s,a)}$ and is known as SARSA

Q Learning

Properties

- Off-policy
- Temporal difference type of method
- Does **not need** information about dynamics of the environment

Q Learning

Properties

- Off-policy
- Temporal difference type of method
- Does **not need** information about dynamics of the environment

Scheme

- Loop and follow $\pi(s_t, a_t)$ derived from $Q_t(s, a)$ e.g. ϵ-greedy
- Use a from $\pi(s_t, a_t)$, observe r_t, s_{t+1}
- Update: $Q_{t+1}(s, a) = Q_t(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q_t(s_{t+1}, a_t) - Q_t(s_t, a_t) \right]$

If you have Universal Function Approximators

- What about just parametrizing $\pi(s_t, a_t, \mathbf{w})$ by weights $\mathbf{w}$ and use some loss-function L?

If you have Universal Function Approximators

- What about just parametrizing $\pi(s_t, a_t, \mathbf{w})$ by weights **w** and use some loss-function L?

→ Known as **policy gradient** and this instance is called REINFORCE

If you have Universal Function Approximators

- What about just parametrizing $\pi(s_t, a_t, \mathbf{w})$ by weights $\mathbf{w}$ and use some loss-function L?

➔ Known as **policy gradient** and this instance is called REINFORCE

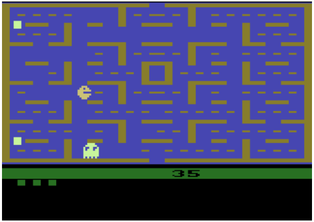
- Generate an episode using $\pi(s_t, a_t, \mathbf{w})$
- Go forwards in the episode: $t = 0, \ldots, T-1$
- $\mathbf{w} = \mathbf{w} + \eta\gamma^t g_t \nabla_{\mathbf{w}} \ln\left(\pi(a_t|s_t, \mathbf{w})\right)$

Deep Reinforcement Learning

Deep Q Learning

Atari Games: Human-level control through deep reinforcement learning [4]

- Volodymyr Mnih et al. (Google DeepMind) 2013/2015
- Idea: Let a neural network play Atari games!
- Input: Current and three subsequent video frames from game
- Processed by network trained with reinforcement learning
- Goal: learn best controller movements

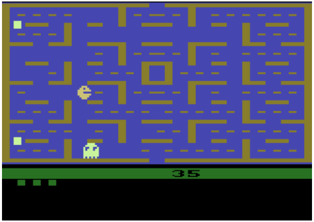


Atari Pac-Man

Source: Human-level control through deep reinforcement learning [4]

Atari Games: Human-level control through deep reinforcement learning [4]

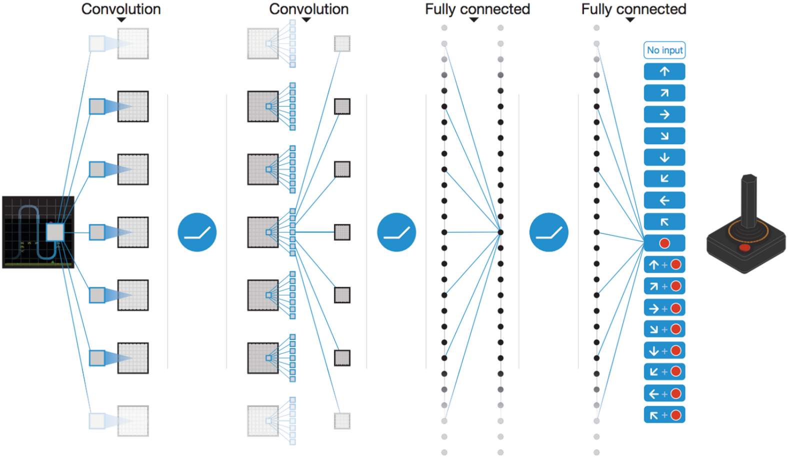
- Volodymyr Mnih et al. (Google DeepMind) 2013/2015
- Idea: Let a neural network play Atari games!
- Input: Current and three subsequent video frames from game
- Processed by network trained with reinforcement learning
- Goal: learn best controller movements
- Convolutional layers for frame processing, fully-connected for final decision making



Atari Pac-Man

Source: Human-level control through deep reinforcement learning [4]

Learning Atari Games



Source: Human-level control through deep reinforcement learning [4]

Learning Atari Games

- **Deep Q-network**: Deep network that applies Q-learning
- State s_t of the game: current + 3 previous frames (image stack)
- 18 outputs associated with an action

→ Each output estimates optimal action value for "its" action given the input

- Instead of label & cost function, update to maximize reward
- Reward: +1/-1 when game score increased/decreased, 0 otherwise
- ϵ-greedy policy with ϵ decreasing to a low value during training
- Semi-gradient form of Q-learning to update network weights $\mathbf{w}$
- Uses mini-batches to accumulate weight updates

Target Network

- Weight update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}_t) - \hat{q}(s_t, a_t, \mathbf{w}_t) \right] \cdot \nabla \mathbf{w}_t \hat{q}(s_t, a_t, \mathbf{w}_t)$$

- Problem: The target $\gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}_t)$ is a function of $\mathbf{w}_t$.

➔ Target changes simultaneously with the weights we want to learn!

➔ Training can oscillate or diverge

Target Network

- Weight update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[r_{t+1} + \gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}_t) - \hat{q}(s_t, a_t, \mathbf{w}_t) \right] \cdot \nabla \mathbf{w}_t \hat{q}(s_t, a_t, \mathbf{w}_t)$$

- Problem: The target $\gamma \max_a \hat{q}(s_{t+1}, a, \mathbf{w}_t)$ is a function of $\mathbf{w}_t$.

➔ Target changes simultaneously with the weights we want to learn!

➔ Training can oscillate or diverge

- Idea: Use a second **target network**:
- After each C steps, copy weights of action-value network to a duplicate network and keep them fixed
- Use output $\bar{q}$ of "target network" as a target to stabilize:

$$\gamma \max_a \bar{q}(s_{t+1}, a, \mathbf{w}_t)$$

Experience Replay

Goal: Reduce correlation between updates

- After performing action a_t for image stack s_t (state) and receiving reward r_t, add (s_t, a_t, r_t, s_{t+1}) to **replay memory**

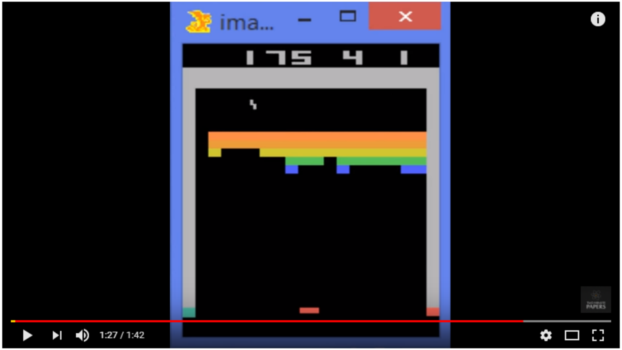
➔ Memory accumulates experiences

- To update the network, draw random samples from memory, instead of taking the most recent ones

➔ Removes dependence on current weights

➔ Increases stability

Atari Breakout Example



Video on learning Atari Breakout. Click here

AlphaGo

Mastering the game of Go with deep neural networks and tree search [1]

- Go is an ancient Chinese boardgame: Black plays against white for control over the board
- Simple rules but extremely high number of possible moves and situations
- Performance on par with professional human players thought years away

Traditional Go board

Source: https://commons.wikimedia.org/wiki/File:FloorGoban.jpg

Challenges in Go

- Go is a "perfect information" game: No hidden information and no chance
- Theoretically, we can construct a full game tree and traverse it with Minimax to find the best moves

Challenges in Go

- Go is a "perfect information" game: No hidden information and no chance
- Theoretically, we can construct a full game tree and traverse it with Minimax to find the best moves
- Problem: High number of legal moves (≈ 250 – chess ≈ 35)
- Games involve many moves (≈ 150)

➔ Exhaustive search is infeasible!

Challenges in Go (cont.)

- Search tree can be **pruned** if we have an accurate evaluation function
- For chess (DeepBlue) already extremely complex and based on massive human input
- For Go: “No simple yet reasonable evaluation function will ever be found for Go.” (Müller 2002) [5]

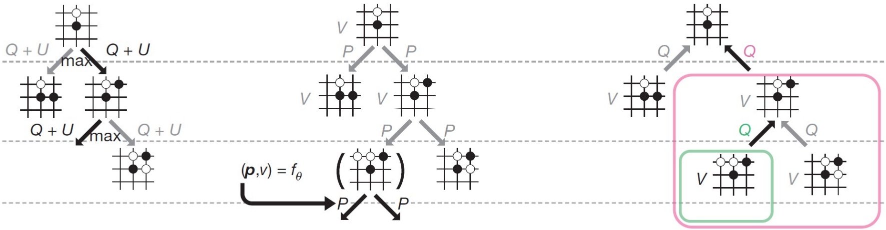
Challenges in Go (cont.)

- Search tree can be **pruned** if we have an accurate evaluation function
- For chess (DeepBlue) already extremely complex and based on massive human input
- For Go: “No simple yet reasonable evaluation function will ever be found for Go.” (Müller 2002) [5]
- Still: **AlphaGo beat Lee Sedol and Ke Jie, two of the world’s strongest players in 2016 and 2017!**

Mastering the game of Go with deep neural networks and tree search [1]

- AlphaGo was developed by Silver et al. (also Google DeepMind)
- Combination of multiple methods:
  - Deep neural networks
  - Monte Carlo tree search (MCTS)
  - Supervised learning **and**
  - Reinforcement learning
- First improvement compared to a full tree search: Monte Carlo Tree Search (MCTS)
- Networks to support efficient search through tree

Monte Carlo Tree Search



- Idea: Run many Monte Carlo simulations of episodes (=entire Go games) to select action (=where to place a stone)
- Starting from a root node representing the current state, MCTS iteratively extends the search tree

Source: Mastering the game of go without human knowledge [2]

Monte Carlo Tree Search (cont.)

Algorithm:

- **Selection**: Starting at root, traverse with tree policy to a leaf node
- **Expansion**: (Optional) add one or more child nodes to the current leaf
- **Simulation**: From the current or the child node, simulate episode with actions according to rollout policy
- **Backup**: Propagate the received reward back through the tree

- Repeat for a certain amount of time, then stop
- Then, choose action from root node according to accumulated statistics
- Start again with new root node

Monte Carlo Tree Search (cont.)

- Tree policy guides in how far successful paths are frequented more often.
- Typical exploration/exploitation trade-off.
- Problem: Estimation via MCTS not accurate enough for Go.

Monte Carlo Tree Search (cont.)

- Tree policy guides in how far successful paths are frequented more often.
- Typical exploration/exploitation trade-off.
- Problem: Estimation via MCTS not accurate enough for Go.

- Ideas in AlphaGo:
  - Control tree expansion by using a neural network to find promising actions.
  - Improve value estimation by a neural network.
- More efficient extension & evaluation of search tree ➔ better at Go!

Deep Neural Networks for Go

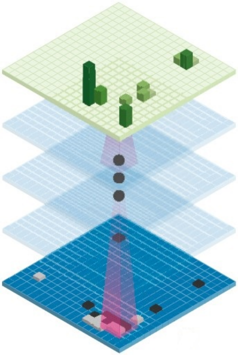
Utilization of three different networks:

- **Policy network**: Suggests the next move in leaf nodes for extension
- **Value network**: Given the current board position, get chances of winning
- **Rollout policy network**: Guide rollout action selection
- All networks are deep convolutional networks
- Input: Current board position and additional precomputed features

Source: Mastering the game of Go with deep neural networks and tree search [1]

Policy Network

- 13 conv-layers, one output for each point on the Go board.
- Huge database of expert human moves (30 mio) available.
- Start with **supervised** learning: Train network to predict the next move in **human expert plays**
- Further train network with reinforcement learning by playing against **older versions of itself**. Reward when winning the game
- Older versions avoid correlation and instability
- Training time: 3 weeks on 50 GPUs + 1 day for RL



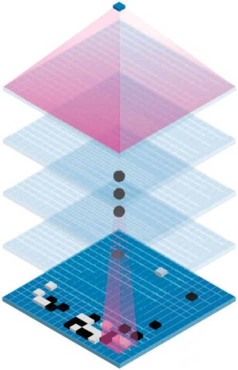
Source: Mastering the game of Go with deep neural networks and tree search [1]

Value network

- Same architecture as policy network but just one output node
- Goal: Estimate how likely the current state leads to a win
- Training utilized self-play games of reinforcement learned policy

→ Trained using Monte-Carlo policy evaluation for 30 mio positions from these games

- Training time: 1 week on 50 GPUs



Source: Mastering the game of Go with deep neural networks and tree search [1]

Rollout policy network

- AlphaGo could use policy network to select moves during roll-out
- Problem: Inference comparatively high: 5 ms
- Solution: Train simpler, linear network on subset of data that provides actions **fast**
- Speedup of $\approx$ 1000 compared to policy network ➔ more simulations possible

AlphaGo Zero

AlphaGo Zero: Do we even need humans for training?

- After minor improvements, Silver et al. proposed AlphaGo Zero:

→ **Solely** trained with reinforcement learning & playing against itself!

- Simpler MCTS, no rollout policy
- Include MCTS in self-play games
- Multi-task training: Policy and value network share initial layers
- Further extension in Dec. ’17: AlphaZero [3] – able to also play chess and shogi

NEXT TIME

ON DEEP LEARNING

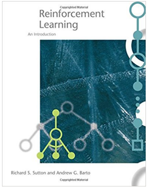
Next Time

- Algorithms to learn if we **don't even observe rewards**
- How to benefit from **adversaries**
- Extensions to perform **image processing** tasks

Comprehensive Questions

- What is a policy?
- What are value functions?
- Explain the exploitation vs exploration dilemma.
- Describe typical solutions to the dilemma.
- What is the difference of a multi armed bandit problem to the full reinforcement learning problem?
- Describe a Markov decision process.
- Is an optimal policy necessarily unique?
- What do the Bellman equations represent?
- Describe policy iteration.
- Why does policy iteration work?
- How can you beat your friends in every Atari game?
- How can one master the game of Go?

Further Reading



Reinforcement Learning

Richard Sutton

- Link - the one real reference for Reinforcement learning in its 2018 draft, including Deep Q learning and Alpha Go details

References

References I

[1] David Silver, Aja Huang, Chris J Maddison, et al. "Mastering the game of Go with deep neural networks and tree search". In: Nature 529.7587 (2016), pp. 484–489.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, et al. "Mastering the game of go without human knowledge". In: Nature 550.7676 (2017), p. 354.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: arXiv preprint arXiv:1712.01815 (2017).

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. "Human-level control through deep reinforcement learning". In: Nature 518.7540 (2015), pp. 529–533.

References II

[5] Martin Müller. "Computer Go". In: Artificial Intelligence 134.1 (2002), pp. 145–179.

[6] Richard S. Sutton and Andrew G. Barto. Introduction to Reinforcement Learning. 1st. Cambridge, MA, USA: MIT Press, 1998.