Learning Agents: Introduction

Liliana Mamani Sanchez
lmamanis@tcd.ie

November 2, 2015
Learning in agent architectures

Agent
Learning in agent architectures

Agent

Perception

Action

Actuators

Perception

Learner

Changes

Performance standard

Critic

Representation

Rewards/Instruction

Goals

Interaction planner

Action policy
Learning in agent architectures
Learning in agent architectures

Agent

perception

action

representation

rewards/instruction

Goals

Interaction planner

Critic

Learner

Changes

Performance standard
Learning in agent architectures

Agent

Perception

Actuators

perception

action

Learner

Changes

Performance standard

Critic

representation

Rewards/instruction

Goals

Interaction planner
Learning in agent architectures

Agent

- Perception
- Actuators

perception

action

Perception

Actuators

representation

rewards/instruction

Goals

Interaction planner

Critic

Performance standard

Learner

changes
Learning in agent architectures

Agent

Learner

Perception

Actuators

perception

action

Performance standard

Critic

representation

rewards/instruction

Goals

Interaction planner
Learning in agent architectures

Agent

Learner

Perception

Actuators

Changes

perception

action

Performance standard

Critic

representation

rewards/instruction

Goals

Interaction planner
Learning in agent architectures

Performance standard

Agent

Learner

Perception

Actuators

Changes

Perception

Action

Goals

Interaction planner

Critic

Representation

Rewards/instruction

Performance standard
Learning in agent architectures

- Critic
- Perception
- Learner
- Actuators

Performance standard

Agent

actions

perception
Learning in agent architectures

Agent

Critic

Perception

Learner

Actuators

Performance standard

representation

rewards/instruction

changes

perception

action

Goals

Interaction planner

2
Learning in agent architectures

Agent

Performance standard

Critic

Perception

Learner

Actuators

goals

representation

rewards/instruction

changes

interaction planner

action

perception
Learning in agent architectures

Performance standard

Critic

representation

Perception

rewards/instruction

Learner

changes

Actuators

Goals

Interaction planner

Agent

perception

action

Goals

Interaction planner

Actuators

Perception

rewards/instruction

Learner

changes

Actuators

Performance standard
Learning in agent architectures

Agent

- Critic
  - Learner
    - Goals
      - Interaction planner
    - Action
      - Action policy
  - Perception
    - Representation
      - Rewards/instruction
      - Changes
    - Action
      - Perception

Performance standard

Goals

Interaction planner

Actuators

Learner

Perception

Critic

Rewards/instruction

Changes

Performance standard
Machine Learning for Games

▶ Reasons to use Machine Learning for Games:
  ▶ Play against, and beat human players (as in board games, DeepBlue etc)
Reasons to use Machine Learning for Games:

- Play against, and beat human players (as in board games, DeepBlue etc)
- Minimise development effort (when developing AI components); avoid the knowledge engineering bottleneck
Machine Learning for Games

- Reasons to use Machine Learning for Games:
  - Play against, and beat human players (as in board games, DeepBlue etc)
  - Minimise development effort (when developing AI components); avoid the knowledge engineering bottleneck
  - Improve the user experience by adding variability, realism, a sense that artificial characters evolve, etc.
Some questions

- What is (Machine) Learning?
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
Some questions

- What is (Machine) Learning?
- What *can* Machine Learning really *do* for us?
- What *kinds of techniques* are there?
- How do we *design* machine learning systems?
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
- How do we design machine learning systems?
- What’s different about reinforcement learning?
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
- How do we design machine learning systems?
- What’s different about reinforcement learning?
- Could you give us some examples?
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
- How do we design machine learning systems?
- What’s different about reinforcement learning?
- Could you give us some examples?
  - YES:
    - Draughts (checkers)
    - Noughts & crosses (tic-tac-toe)
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
- How do we design machine learning systems?
- What’s different about reinforcement learning?
- Could you give us some examples?
  - YES:
    - Draughts (checkers)
Some questions

- What is (Machine) Learning?
- What can Machine Learning really do for us?
- What kinds of techniques are there?
- How do we design machine learning systems?
- What’s different about reinforcement learning?
- Could you give us some examples?
  - YES:
  - Draughts (checkers)
  - Noughts & crosses (tic-tac-toe)
Defining “learning”

- ML has been studied from various perspectives (AI, control theory, statistics, information theory, ...)
- From an AI perspective, the general definition is formulated in terms of agents and tasks. E.g.:

  [An agent] is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with $E$.

  [Mitchell, 1997, p. 2]
- Statistics, model-fitting, ...
Some examples

- Problems too **difficult to program** by hand

(ALVINN [Pomerleau, 1994])
if Name = Corners & Energy < 25
then
    turn(91 - (Bearing - const))
    fire(3)
User interface agents

- Recommendation services,
- Bayes spam filtering
- JIT information retrieval
Designing a machine learning system

Main design decisions:
- **Training experience:** How will the system access and use data?
- **Target function:** What exactly should be learned?
- **Hypothesis representation:** How will we represent the concepts to be learnt?
- **Inductive inference:** What specific algorithm should be used to learn the target concepts?
Types of machine learning

- How will the system be exposed to its training experience?
  - **Direct** or indirect access:
    - indirect access: record of past experiences, databases, corpora
    - direct access: situated agents \(\rightarrow\) reinforcement learning
  - Source of feedback ("teacher"):
    - supervised learning
    - unsupervised learning
    - mixed: semi-supervised ("transductive"), active learning, ....
The hypothesis space

- The data used in the induction process need to be represented uniformly. E.g.:
  - representation of the opponent’s behaviour as feature vectors
- The choice of representation constrains the space of available hypotheses (inductive bias).
- Examples of inductive bias:
  - assume that positive and negative instances can be separated by a (hyper) plane
  - assume that feature co-occurrence does not matter (conditional independence assumption by Naïve Bayes classifiers)
  - assume that the current state of the environment summarises environment history (Markov property)
Determining the target function

- The goal of the learning algorithm is to induce an approximation $\hat{f}$ of a target function $f$
- In supervised learning, the target function is assumed to be specified through annotation of training data or some form of feedback.
- Examples:
  - a collection of texts categorised by subject $f : D \times S \rightarrow \{0, 1\}$
  - a database of past games
  - user or expert feedback
- In reinforcement learning the agent will learn an action selection policy (as in $action : S \rightarrow A$)
Deduction and Induction

- Deduction: from general premises to a conclusion. E.g.:
  - \( \{A \rightarrow B, A\} \vdash B \)
- Induction: from instances to generalisations
- Machine learning algorithms produce models that generalise from instances presented to the algorithm
- But all (useful) learners have some form of inductive bias:
  - In terms of representation, as mentioned above,
  - But also in terms of their preferences in generalisation procedures. E.g:
    - prefer simpler hypotheses, or
    - prefer shorter hypotheses, or
    - incorporate domain (expert) knowledge, etc etc
Choosing an algorithm

- Induction task as search for a hypothesis (or model) that fits the data and sample of the target function available to the learner, in a large space of hypotheses
- The choice of learning algorithm is conditioned to the choice of representation
- Since the target function is not completely accessible to the learner, the algorithm needs to operate under the inductive learning assumption that:
  
  an approximation that performs well over a sufficiently large set of instances will perform well on unseen data

- Computational Learning Theory addresses this question.
Two Games: examples of learning


► Reinforcement learning: noughts and crosses [Sutton and Barto, 1998]

► Task? (target function, data representation) Training experience? Performance measure?
A target for a draughts learner

- Learn.... $f : Board \rightarrow Action$ or $f : Board \rightarrow \mathbb{R}$
A target for a draughts learner

- Learn.... \( f : \text{Board} \rightarrow \text{Action} \) or \( f : \text{Board} \rightarrow \mathbb{R} \)
A target for a draughts learner

- Learn.... \( f : \text{Board} \rightarrow \text{Action} \) or \( f : \text{Board} \rightarrow \mathbb{R} \)

- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if \( b \) is a final board state that is won, then \( f(b) = 100 \)
  - if \( b \) is a final board state that is lost, then \( f(b) = -100 \)
  - if \( b \) is a final board state that is drawn, then \( f(b) = 0 \)
  - if \( b \) is a not a final state in the game, then \( f(b) = f(b') \), where \( b' \) is the best final board state that can be achieved starting from \( b \) and playing optimally until the end of the game.

- How feasible would it be to implement these strategies?
- Hmmmm... Not feasible...
A target for a draughts learner

- Learn.... $f : \text{Board} \to \text{Action}$ or $f : \text{Board} \to \mathbb{R}$
- But how do we label (evaluate) the training experience?
A target for a draughts learner

- Learn.... \( f : \text{Board} \rightarrow \text{Action} \) or \( f : \text{Board} \rightarrow \mathbb{R} \)
- But how do we label (evaluate) the training experience?
- Ask an expert?
A target for a draughts learner

- Learn.... $f : Board \rightarrow Action$ or $f : Board \rightarrow \mathbb{R}$
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:

  - if $b$ is a final board state that is won, then $f(b) = 100$
  - if $b$ is a final board state that is lost, then $f(b) = -100$
  - if $b$ is a final board state that is drawn, then $f(b) = 0$
  - if $b$ is a not a final state in the game, then $f(b) = f(b')$, where $b'$ is the best final board state that can be achieved starting from $b$ and playing optimally until the end of the game.
A target for a draughts learner

- Learn…. \( f : Board \rightarrow Action \) or \( f : Board \rightarrow \mathbb{R} \)
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if \( b \) is a final board state that is won, then \( f(b) = 100 \)
  - if \( b \) is a final board state that is lost, then \( f(b) = -100 \)
  - if \( b \) is a final board state that is drawn, then \( f(b) = 0 \)
  - if \( b \) is a not a final state in the game, then \( f(b) = f(b') \), where \( b' \) is the best final board state that can be achieved starting from \( b \) and playing optimally until the end of the game.
A target for a draughts learner

- Learn.... \( f : \text{Board} \rightarrow \text{Action} \) or \( f : \text{Board} \rightarrow \mathbb{R} \)
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if \( b \) is a final board state that is won, then \( f(b) = 100 \)
  - if \( b \) is a final board state that is lost, then \( f(b) = -100 \)
A target for a draughts learner

- Learn.... $f : \text{Board} \to \text{Action}$ or $f : \text{Board} \to \mathbb{R}$
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if $b$ is a final board state that is won, then $f(b) = 100$
  - if $b$ is a final board state that is lost, then $f(b) = -100$
  - if $b$ is a final board state that is drawn, then $f(b) = 0$
  - if $b$ is a not a final state in the game, then $f(b) = f(b')$, where $b'$ is the best final board state that can be achieved starting from $b$ and playing optimally until the end of the game.
A target for a draughts learner

- Learn…. $f: \text{Board} \rightarrow \text{Action}$ or $f: \text{Board} \rightarrow \mathbb{R}$
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if $b$ is a final board state that is won, then $f(b) = 100$
  - if $b$ is a final board state that is lost, then $f(b) = -100$
  - if $b$ is a final board state that is drawn, then $f(b) = 0$
  - if $b$ is a not a final state in the game, then $f(b) = f(b')$, where $b'$ is the best final board state that can be achieved starting from $b$ and playing optimally until the end of the game.
A target for a draughts learner

- Learn.... \( f : \text{Board} \rightarrow \text{Action} \) or \( f : \text{Board} \rightarrow \mathbb{R} \)
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if \( b \) is a final board state that is won, then \( f(b) = 100 \)
  - if \( b \) is a final board state that is lost, then \( f(b) = -100 \)
  - if \( b \) is a final board state that is drawn, then \( f(b) = 0 \)
  - if \( b \) is a not a final state in the game, then \( f(b) = f(b') \), where \( b' \) is the best final board state that can be achieved starting from \( b \) and playing optimally until the end of the game.
- How feasible would it be to implement these strategies?
A target for a draughts learner

- Learn.... $f : \text{Board} \rightarrow \text{Action}$ or $f : \text{Board} \rightarrow \mathbb{R}$
- But how do we label (evaluate) the training experience?
- Ask an expert?
- Derive values from a rational strategy:
  - if $b$ is a final board state that is won, then $f(b) = 100$
  - if $b$ is a final board state that is lost, then $f(b) = -100$
  - if $b$ is a final board state that is drawn, then $f(b) = 0$
  - if $b$ is a not a final state in the game, then $f(b) = f(b')$, where $b'$ is the best final board state that can be achieved starting from $b$ and playing optimally until the end of the game.
- How feasible would it be to implement these strategies?
  - Hmmm... Not feasible...
Hypotheses and Representation

- The choice of representation (e.g. logical formulae, decision tree, neural net architecture) constrains the hypothesis search space.
Hypotheses and Representation

- The choice of representation (e.g. logical formulae, decision tree, neural net architecture) constrains the hypothesis search space.
- A representation scheme: linear combination of board features:

\[
\hat{f}(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)
\]
Hypotheses and Representation

- The choice of representation (e.g. logical formulae, decision tree, neural net architecture) constrains the hypothesis search space.

- A representation scheme: linear combination of board features:

\[
\hat{f}(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) \\
+ w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)
\]

- where:
  - \(bp(b)\): number of black pieces on board \(b\)
  - \(rp(b)\): number of red pieces on \(b\)
  - \(bk(b)\): number of black kings on \(b\)
  - \(rk(b)\): number of red kings on \(b\)
  - \(bt(b)\): number of red pieces threatened by black
  - \(rt(b)\): number of black pieces threatened by red
Hypotheses and Representation

▶ The choice of representation (e.g. logical formulae, decision tree, neural net architecture) constrains the hypothesis search space.

▶ A representation scheme: linear combination of board features:

\[
\hat{f}(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)
\]

▶ where:

▶ \(bp(b)\): number of black pieces on board \(b\)
▶ \(rp(b)\): number of red pieces on \(b\)
▶ \(bk(b)\): number of black kings on \(b\)
▶ \(rk(b)\): number of red kings on \(b\)
▶ \(rk(b)\): number of red pieces threatened by black
▶ \(rt(b)\): number of black pieces threatened by red
Some notation and distinctions to keep in mind:

- $f(b)$: the true target function
- $\hat{f}(b)$: the learnt function
- $f_{\text{train}}(b)$: the training value (obtained, for instance, from a training set containing instances and its corresponding training values)

Problem: How do we obtain training values?
Training Experience

- Some notation and distinctions to keep in mind:
  - $f(b)$: the true target function
  - $\hat{f}(b)$: the learnt function
  - $f_{\text{train}}(b)$: the training value (obtained, for instance, from a training set containing instances and its corresponding training values)

- Problem: How do we obtain training values?
- A simple rule for obtaining (estimating) training values:
  - $f_{\text{train}}(b) \leftarrow \hat{f}(\text{Successor}(b))$
How do we learn the weights?

Algorithm 1: Least Means Square

LMS(c: learning rate)
for each training instance < b, f_{train}(b) >
  do
    compute error(b) for current approximation (i.e. using current weights):
      error(b) = f_{train}(b) − ̂f(b)
    for each board feature t_i ∈ {bp(b), rp(b), ...},
      do
        update weight w_i:
        w_i ← w_i + c × t_i × error(b)
    done
  done

How do we learn the weights?

Algorithm 1: Least Means Square

LMS (\(c: \text{learning rate}\))

for each training instance \(<b, f_{\text{train}}(b)\>

do

compute \text{error}(b) \text{ for current approximation (i.e. using current weights)}:

\[\text{error}(b) = f_{\text{train}}(b) - \hat{f}(b)\]

for each board feature \(t_i \in \{bp(b), rp(b), \ldots\}\),
do

update weight \(w_i:\)

\[w_i \leftarrow w_i + c \times t_i \times \text{error}(b)\]
done
done

LMS minimises the squared error between training data and current approx.:\[E \equiv \sum_{\langle b, f_{\text{train}}(b)\rangle \in D} (f_{\text{train}}(b) - \hat{f}(b))^2\]
Design choices: summary

(from [Mitchell, 1997])
These are some of the decisions involved in ML design. A number of other practical factors, such as evaluation, avoidance of “overfitting”, feature engineering, etc. See [Domingos, 2012] for a useful introduction, and some machine learning “folk wisdom”.

(from [Mitchell, 1997])
The Architecture instantiated

Performance standard

Critic

representation

rewards/instruction

Learner

Goals

Perception

changes

Actuators

Interaction planner

action policy

Agent

perception

action

f_{train}(b) := \hat{f}(\text{successor}(b, f_{train}(b), ...))

\pi^* = \arg \max_{\pi} \hat{f}(s), \forall s
The Architecture instantiated

\[
f_{\text{train}}(b) := \hat{f}(\text{successor}(b), f_{\text{train}}(b), ...)\]

\[
\pi^* = \arg \max_\pi \hat{f}(s), \forall s
\]
The Architecture instantiated

\[ f_{\text{train}}(b) := \hat{f}(\text{successor}(b)) \]

Perception representation

Agent

Critic

Learner

Goals

Interaction planner

Actuators

Performance standard

\[ \pi^* = \arg \max_{\pi} \hat{f}(s), \quad \forall s \]

\( f_{\text{train}}(b) \)

(rewards/instruction)

(changes)

(action policy)

(perception)

\( (bp(b), rp(b), \ldots) \)
The Architecture instantiated

Performance standard

\[ f_{\text{train}}(b) := \hat{f}(\text{successor}(b)) \]

Agent

Critic

Learner

Goals

Interaction planner

Actuators

Perception

changes

action policy

perception

\[ (bp(b), rp(b), \ldots) \]

\[ (b, f_{\text{train}}(b), \ldots) \]

\[ \pi^* = \arg \max \pi \hat{f}(s), \forall s \]
The Architecture instantiated

\[ f_{\text{train}}(b) := \hat{f}(\text{successor}(b)) \]

\[ f(b) := \hat{f}(\text{successor}(b)) \]

\[ \pi^* = \arg \max_{\pi} \hat{f}(s), \forall s \]
The Architecture instantiated

\[
f_{\text{train}}(b) := \hat{f}(\text{successor}(b))
\]

\[
f_{\text{train}}(b) := \hat{f}(\text{successor}(b))
\]

\[
\pi^* = \arg \max \pi \hat{f}(s), \forall s
\]
The Architecture instantiated

\[ f_{\text{train}}(b) := \hat{f}(\text{successor}(b)) \]

\[ \hat{f}(b, f_{\text{train}}(b), ...), ... \]

\[ \pi^* = \arg \max_{\pi} \hat{f}(s), \forall s \]

\[ \text{Performance standard} \]

\[ \text{Agent} \]

\[ \text{Critic} \]

\[ \text{Learner} \]

\[ \text{Goals} \]

\[ \text{Interaction planner} \]

\[ \text{Actuators} \]

\[ \text{Perception} \]

\[ \text{Initial board} \]
Reinforcement Learning

- What is different about reinforcement learning:
  - Training experience (data) obtained through direct interaction with the environment;
  - Influencing the environment;
  - Goal-driven learning;
  - Learning of an action policy (as a first-class concept);
  - Trial and error approach to search:
Reinforcement Learning

What is different about reinforcement learning:

- Training experience (data) obtained through direct interaction with the environment;
- Influencing the environment;
- Goal-driven learning;
- Learning of an action policy (as a first-class concept);
- Trial and error approach to search:
  - Exploration and Exploitation
Basic concepts of Reinforcement Learning

- The **policy**: defines the learning agent’s way of behaving at a given time:
  \[ \pi : S \rightarrow A \]

- The **(immediate) reward function**: defines the goal in a reinforcement learning problem:
  \[ r : S \rightarrow \mathbb{R} \]
  often identified with timesteps: \( r_0, \ldots, r_n \in \mathbb{R} \)

- The **(long term) value function**: the total amount of reward an agent can expect to accumulate over the future:
  \[ V : S \rightarrow \mathbb{R} \]

- A **model** of the environment
Theoretical background

- **Engineering**: “optimal control” (dating back to the 50’s)
  - Markov Decision Processes (MDPs)
  - Dynamic programming

- **Psychology**: learning by trial and error, animal learning.
  - Law of effect:
    - learning is selectional (genetic methods, for instance, are selectional, but not associative)
    - associative (supervised learning is associative, but not selectional)

- **AI**: TD learning, Q-learning
Theoretical background

- **Engineering**: “optimal control” (dating back to the 50’s)
  - Markov Decision Processes (MDPs)
  - Dynamic programming
- **Psychology**: learning by trial and error, animal learning. Law of effect:
  - learning is **selectional** (genetic methods, for instance, are selectional, but not associative) and
  - **associative** (supervised learning is associative, but not selectional)
Theoretical background

- **Engineering**: “optimal control” (dating back to the 50’s)
  - Markov Decision Processes (MDPs)
  - Dynamic programming
- **Psychology**: learning by trial and error, animal learning. Law of effect:
  - learning is *selectional* (genetic methods, for instance, are selectional, but not associative) and
  - associative (supervised learning is associative, but not selectional)
- **AI**: TD learning, Q-learning
Example: Noughts and crosses

Possible solutions:
- Minimax (assume a perfect opponent),
- Supervised learning (directly search the space of policies, as in the previous example),
- Reinforcement learning (our next example).
Example: Noughts and crosses

Possible solutions:
Example: Noughts and crosses

Possible solutions: \textit{minimax} (assume a perfect opponent),
Example: Noughts and crosses

Possible solutions: minimax (assume a perfect opponent), supervised learning (directly search the space of policies, as in the previous example),
Example: Noughts and crosses

Possible solutions: \textit{minimax} (assume a perfect opponent), \textit{supervised learning} (directly search the space of policies, as in the previous example), \textit{reinforcement learning} (our next example).
A Reinforcement Learning strategy

- Assign **values to each possible game state** (e.g. the probability of winning from that state):

<table>
<thead>
<tr>
<th>state</th>
<th>$V(s)$</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0 = \begin{array}{c} x \ H \ 0 \end{array}$</td>
<td>0.5</td>
<td>??</td>
</tr>
<tr>
<td>$s_1 = \begin{array}{c} x \ 0 \end{array}$</td>
<td>0.5</td>
<td>??</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$0$</td>
<td>loss</td>
</tr>
<tr>
<td>$s_i = \begin{array}{c} x \ x \ 0 \end{array}$</td>
<td>$0$</td>
<td>loss</td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s_n = \begin{array}{c} x \ x \ x \ 0 \end{array}$</td>
<td>$1$</td>
<td>win</td>
</tr>
</tbody>
</table>

Algorithm 2: TD Learning

While learning

select move by

looking ahead 1 state

choose next state $s$

if $\neq$ exploring

pick $s$ at random

else

$s = \text{arg max}_s V(s)$

N.B.: **exploring** could mean, for instance, pick a random next state 10% of the time.
How to update state values

$s_0$
How to update state values

$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$ (TD learning)

An update rule:

Step-size parameter (learning rate)
How to update state values

An update rule:

$$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$$

(TD learning)

Step-size parameter (learning rate)
How to update state values

\[ V(s) \leftarrow V(s) + \alpha \left[ V(s') - V(s) \right] \]  

(TD learning)
How to update state values

An update rule:

$$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$$  \hspace{1cm} \text{(TD learning)}

**Opponent's move**: $s_0$

**Our (greedy) move**: $s_1^*$, $s_i^*$
How to update state values

- Opponent's move
- Our (greedy) move
- An exploratory move

Update rule:

$$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$$

(TD learning)
How to update state values

An exploratory move

An update rule:
$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$ (TD learning)
How to update state values

\[ V(s) \leftarrow V(s) + \alpha [V(s') - V(s)] \]  
(TD learning)
How to update state values

Our (greedy) move

An exploratory move

An update rule:

\[ V(s) \leftarrow V(s) + \alpha [V(s') - V(s)] \]

TD learning
How to update state values

An update rule:
(TD learning)

\[ V(s) \leftarrow V(s) + \alpha [V(s') - V(s)] \]

step-size parameter (learning rate)
Some nice properties of this RL algorithm

▶ For a fixed opponent, if the parameter that controls learning rate ($\alpha$) is reduced properly over time, converges to the true probabilities of winning from each state (yielding an optimal policy)

▶ If $\alpha$ isn't allowed to reach zero, the system will play well against opponents that alter their game (slowly)

▶ Takes into account what happens during the game (unlike supervised approaches)
Some nice properties of this RL algorithm

- For a **fixed opponent**, if the parameter that controls learning rate ($\alpha$) is reduced properly over time, converges to the true probabilities of winning from each state (yielding an optimal policy)
Some nice properties of this RL algorithm

- For a fixed opponent, if the parameter that controls learning rate ($\alpha$) is reduced properly over time, converges to the true probabilities of winning from each state (yielding an optimal policy).
- If $\alpha$ isn’t allowed to reach zero, the system will play well against opponents that alter their game (slowly).
Some nice properties of this RL algorithm

- For a fixed opponent, if the parameter that controls learning rate ($\alpha$) is reduced properly over time, converges to the true probabilities of winning from each state (yielding an optimal policy)
- If $\alpha$ isn’t allowed to reach zero, the system will play well against opponents that alter their game (slowly)
- Takes into account what happens during the game (unlike supervised approaches)
What was not illustrated

- **RL also applies to situations where there isn’t a clearly defined adversary** ("games against nature")
- **RL also applies to non-episodic problems** (i.e. rewards can be received at any time not only at the end of an episode such as a finished game)
- **RL scales up well** to games where the search space is (unlike our example) truly vast.
  - See [Tesauro, 1994], for instance.
- **Prior knowledge** can also be incorporated
- **Look-ahead** isn’t always required
References

A few useful things to know about machine learning.

*Machine Learning*.
McGraw-Hill.

*Neural Network Perception for Mobile Robot Guidance*.

*Reinforcement Learning: An Introduction*.
MIT Press, Cambridge, MA.

TD-gammon, a self-teaching backgammon program, achieves master-level play.