Machine Learning

Reinforcement Learning

Prof. Dr. Martin Riedmiller
AG Maschinelles Lernen und Natürlichsprachliche Systeme
Institut für Informatik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

riedmiller@informatik.uni-freiburg.de
Motivation

Can a software agent learn to play Backgammon by itself?

Learning from success or failure

Neuro-Backgammon:
playing at worldchampion level
(Tesauro, 1992)
Can a software agent learn to **balance a pole** by itself?

Learning from success or failure

**Neural RL controllers:** noisy, unknown, nonlinear (Riedmiller et.al.)
Can a software agent learn to cooperate with others by itself?

Learning from success or failure

COoperative RL agents:
complex, multi-agent, cooperative
(Riedmiller et.al.)
Reinforcement Learning

has biological roots: reward and punishment

no teacher, but:

actions + goal \( \rightarrow \) algorithm/ policy

‘Happy Programming’
Actor-Critic Scheme (Barto, Sutton, 1983)

Actor-Critic Scheme:
- Critic maps external, delayed reward in internal training signal
- Actor represents policy
Overview

1 Reinforcement Learning - Basics
A First Example

Repeat

Choose: Action $a \in \{\rightarrow, \leftarrow, \uparrow\}$

Until Goal is reached
The 'Temporal Credit Assignment' Problem

Which action(s) in the sequence has to be changed?
⇒ Temporal Credit Assignment Problem
Sequential Decision Making

Examples:
Chess, Checkers (Samuel, 1959), Backgammon (Tesauro, 92)
Cart-Pole-Balancing (AHC/ ACE (Barto, Sutton, Anderson, 1983)), Robotics and control, . . .
Three Steps

⇒ Describe environment as a Markov Decision Process (MDP)

⇒ Formulate learning task as a dynamic optimization problem

⇒ Solve dynamic optimization problem by dynamic programming methods
1. Description of the environment

\( S \): (finite) set of states
\( A \): (finite) set of actions

Behaviour of the environment 'model'

\[ p : S \times S \times A \rightarrow [0, 1] \]

\( p(s', s, a) \) Probability distribution of transition

For simplicity, we will first assume a deterministic environment. There, the model can be described by a transition function

\[ f : S \times A \rightarrow S, \quad s' = f(s, a) \]

'Markov' property: Transition only depends on current state and action

\[ Pr(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2}, \ldots) \]
2. Formulation of the learning task

every transition emits transition costs, 'immediate costs', \( c : S \times A \to \mathbb{R} \) 
(sometimes also called 'immediate reward', \( r \))

Now, an agent policy \( \pi : S \to A \) can be evaluated (and judged):
Consider pathcosts:
\[
J^\pi(s) = \sum_t c(s_t, \pi(s_t)), s_0 = s
\]

Wanted: optimal policy \( \pi^* : S \to A \)
where \( J^{\pi^*}(s) = \min_{\pi} \{ \sum_t c(s_t, \pi(s_t)) | s_0 = s \} \)

\( \Rightarrow \) Additive (path-)costs allow to consider all events
\( \Rightarrow \) Does this solve the temporal credit assignment problem? YES!
Choice of immediate cost function $c(\cdot)$ specifies policy to be learned

Example:

$$
c(s) = \begin{cases} 
0 & \text{if } s \text{ success } (s \in \text{Goal}) \\
1000 & \text{if } s \text{ failure } (s \in \text{Failure}) \\
1 & \text{else}
\end{cases}
$$

$J^\pi(s_{\text{start}}) = 12$

$J^\pi(s_{\text{start}}) = 1004$

⇒ specification of requested policy by $c(\cdot)$ is simple!
3. Solving the optimization problem

For the optimal path costs it is known that

\[ J^*(s) = \min_a \{ c(s, a) + J^*(f(s, a)) \} \]

(Principle of Optimality (Bellman, 1959))

⇒ Can we compute \( J^* \) (we will see why, soon)?
Computing $J^*$: the value iteration (VI) algorithm

Start with arbitrary $J_0(s)$
for all states $s$: $J_{k+1}(s) := \min_{a \in A} \{ c(s, a) + J_k(f(s, a)) \}$

![Diagram]

Start with arbitrary $J_0(s)$
for all states $s$: $J_{k+1}(s) := \min_{a \in A} \{ c(s, a) + J_k(f(s, a)) \}$

![Diagram]
Convergence of value iteration

Value iteration converges under certain assumptions, i.e. we have
\[ \lim_{k \to \infty} J_k = J^* \]

\[ \Rightarrow \] Discounted problems:  
\[ J^\pi*(s) = \min_\pi\{\sum_t \gamma^t c(s_t, \pi(s_t)) \mid s_0 = s\} \]
where \( 0 \leq \gamma < 1 \) (contraction mapping)

\[ \Rightarrow \] Stochastic shortest path problems:

- there exists an absorbing terminal state with zero costs
- there exists a 'proper' policy (a policy that has a non-zero chance to finally reach the terminal state)
- every non-proper policy has infinite path costs for at least one state
Ok, now we have $J^*$

$\Rightarrow$ when $J^*$ is known, then we also know an optimal policy:

$$\pi^*(s) \in \arg \min_{a \in A} \{c(s, a) + J^*(f(s, a))\}$$

![Diagram of a simple graph with nodes labeled 1, 2, 5, and 7, showing transitions between states with weights of 1.]

The diagram shows transitions between states labeled 2, 5, and 7, with weights of 1.
Back to our maze
Overview of the approach so far

- Description of the learning task as an MDP $S, A, T, f, c$
  $c$ specifies requested behaviour / policy

- Iterative computation of optimal path costs $J^*$:
  $\forall s \in S : J_{k+1}(s) = \min_{a \in A} \{ c(s, a) + J_k(f(s, a)) \}$

- Computation of an optimal policy from $J^*$
  $\pi^*(s) \in \arg \min_{a \in A} \{ c(s, a) + J^*(f(s, a)) \}$

- Value function ('costs-to-go') can be stored in a table
Overview of the approach: **Stochastic Domains**

- **value iteration** in stochastic environments:
  \[ \forall s \in S : J_{k+1}(s) = \min_{a \in A} \left\{ \sum_{s' \in S} p(s, s', a) \left( c(s, a) + J_k(s') \right) \right\} \]

- **Computation of an optimal policy from** \( J^* \)
  \[ \pi^*(s) \in \arg \min_{a \in A} \left\{ \sum_{s' \in S} p(s, s', a) \left( c(s, a) + J_k(s') \right) \right\} \]

- **value function** \( J \) ('costs-to-go') can be stored in a table
Reinforcement Learning

Problems of Value Iteration:

for all \( s \in S \): \( J_{k+1}(s) = \min_{a \in A} \{ c(s, a) + J_k(f(s, a)) \} \)

problems:

- Size of \( S \) (Chess, robotics, . . . ) \( \Rightarrow \) learning time, storage?
- 'model' (transition behaviour) \( f(s, a) \) or \( p(s', s, a) \) must be known!

Reinforcement Learning is dynamic programming for very large state spaces and/or model-free tasks
Important contributions - Overview

- Real Time Dynamic Programming
  (Barto, Sutton, Watkins, 1989)

- Model-free learning (Q-Learning, (Watkins, 1989))

- neural representation of value function (or alternative function approximators)
Real Time Dynamic Programming (Barto, Sutton, Watkins, 1989)

Idea:

instead For all \( s \in S \) now For some \( s \in S \) . . .

\[ \Rightarrow \] learning based on trajectories (experiences)
Q-Learning

Idea (Watkins, Diss, 1989):
In every state store for every action the expected costs-to-go. \( Q_\pi(s, a) \) denotes the expected future pathcosts for applying action \( a \) in state \( s \) (and continuing according to policy \( \pi \)):

\[
Q_\pi(s, a) := \sum_{s' \in S} p(s', s, a)(c(s, a) + J_\pi(s'))
\]

where \( J_\pi(s') \) expected pathcosts when starting from \( s' \) and acting according to \( \pi \)
Q-learning: Action selection

is now possible without a model:

Original VI: state evaluation
Action selection:

\[ \pi^*(s) \in \arg \min \{ c(s, a) + J^*(f(s, a)) \} \]

Q: state-action evaluation
Action selection:

\[ \pi^*(s) = \arg \min Q^*(s, a) \]
Learning an optimal Q-Function

To find $Q^*$, a value iteration algorithm can be applied

$$Q_{k+1}(s, u) := \sum_{s' \in S} p(s', s, a)(c(s, a) + J_k(s'))$$

where $J_k(s) = \min_{a' \in A(s)} Q_k(s, a')$

◊ Furthermore, learning a Q-function without a model, by experience of transition tuples $(s, a) \rightarrow s'$ only is possible:

**Q-LEARNING** (Q-Value Iteration + Robbins-Monro stochastic approximation)

$$Q_{k+1}(s, a) := (1 - \alpha) Q_k(s, a) + \alpha (c(s, a) + \min_{a' \in A(s')} Q_k(s', a'))$$
Summary Q-learning

Q-learning is a variant of value iteration when no model is available. It is based on two major ingredients:

- uses a representation of costs-to-go for state/ action-pairs $Q(s, a)$
- uses a stochastic approximation scheme to incrementally compute expectation values on the basis of observed transitions $(s, a) \rightarrow s'$

◊ converges under the same assumption as value iteration + 'every state/ action pair has to be visited infinitely often' + conditions for stochastic approximation
Q-Learning algorithm

**Repeat**

start in arbitrary initial state $s_0$; $t = 0$

**Repeat**

choose action greedily $u_t := \arg \min_{a \in A} Q_k(s_t, a)$
or $u_t$ according to an exploration scheme
apply $u_t$ in the environment: $s_{t+1} = f(s_t, u_t, w_t)$
learn $Q$-value:

$$Q_{k+1}(s_t, u_t) := (1 - \alpha)Q_k(s_t, u_t) + \alpha(c(s_t, u_t) + J_k(s_{t+1}))$$

where $J_k(s_{t+1}) := \min_{a \in A} Q_k(s_{t+1}, a)$

**Until** Terminal state reached

**Until** policy is optimal ('enough')
Representation of the path-costs in a function approximator

Idea: neural representation of value function (or alternative function approximators) (Neuro Dynamic Programming (Bertsekas, 1987))

⇒ few parameters (here: weights) specify value function for a large state space
⇒ learning by gradient descent: \[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial (J(s') - c(s,a) - J(s))^2}{\partial w_{ij}} \]
Example: learning to intercept in robotic soccer

- as fast as possible (anticipation of intercept position)
- random noise in ball and player movement → need for corrections
- sequence of \( \text{TURN}(\theta) \) and \( \text{DASH}(v) \)-commands required

⇒ handcoding a routine is a lot of work, many parameters to tune!
Reinforcement learning of intercept

Goal: Ball is in kickrange of player

- state space: $S^{work} =$ positions on pitch
- $S^+$: Ball in kickrange
- $S^-$: e.g. collision with opponent

$$c(s) = \begin{cases} 
0 &, s \in S^+ \\
1 &, s \in S^- \\
0.01 & , \text{ else} 
\end{cases}$$

- Actions: `TURN(10°), TURN(20°), \ldots TURN(360°), \ldots DASH(10), DASH(20), \ldots`
- neural value function (6-20-1-architecture)
Learning curves

Percentage of successes

Costs (time to intercept)