Formal Approaches to Decision-Making under Uncertainty

Lecture 4-1: Dealing with Large MDPs

Arnd Hartmanns
Formal Methods and Tools
UNIVERSITY OF TWENTE

Very Large MDPs

Example from ML: Atari games

States: all possible screen images (+ state of RAM)

Actions: joystick movement, button

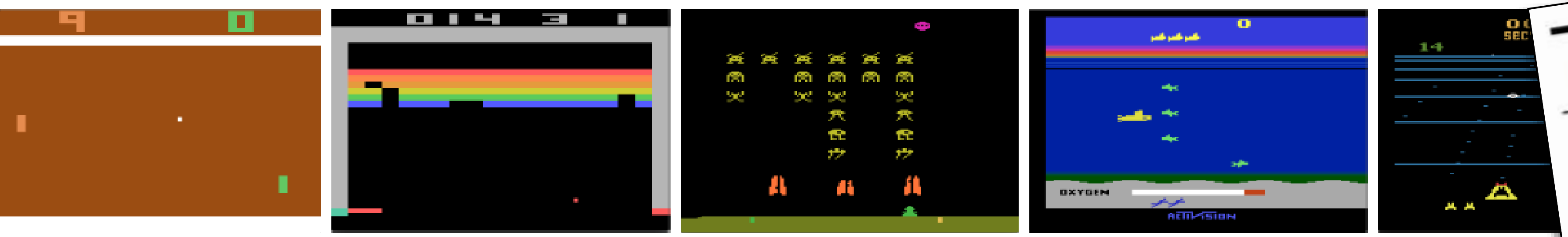
Reward: winning game points

state space explosion

$128 \text{ colors}^{(210 \times 160 \text{ pixels})} =$

$> 10^{70802}$ states

reinforcement learning instead of model checking
using deep neural network as function approximators
→ get good strategy, but lose all optimality guarantees




Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies

Q-Learning

Large MDP given implicitly, e.g. via interface

– $init()$ → initial state

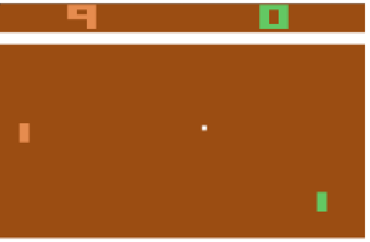
– $acts(s)$ → actions of state s

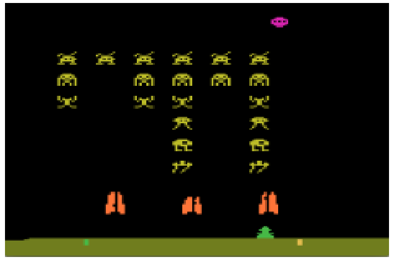
– $sample(s, a)$ → randomly select successor s' and get reward r

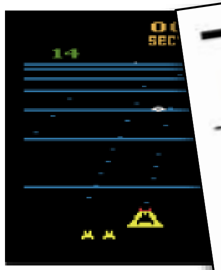
– $term(s)$ → is s a goal or S_0 state?

but not

– $distr(s, a)$ → get full distribution info for action a







Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

Q-Learning

Model checking algorithms:

maintain vectors $x[s]$ or $x_i[s]$

Q-learning:

maintain function $Q: S \times A \to \mathbb{R}$

with $Q(s, a)$ indicating the "quality" of action a from s
– i.e. an approx. of the goal probability or expected reward

do simulation runs up to goal or S_0 state,
updating the Q-function with
the newly found rewards as you go

Q-Learning

$R(\square G)$ $r_1 + r_2 + r_3 + \ldots + r_n$ ↙ goal

discounted: $r_1 + \gamma(r_2 + \gamma(r_3 + \ldots + \gamma r_n))$

Algorithm for one *episode* (simulation run):

1. $s := init()$ "ε-greedy strategy"; often start with ε near 1, decrease over more episodes
2. with probability ϵ, select uniformly random a from $acts(s)$; "exploration"

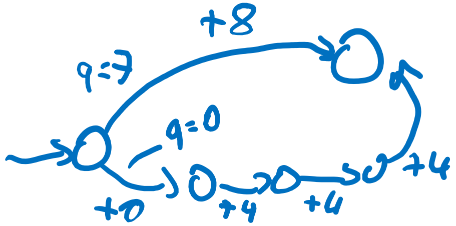
   with probability $1 - \epsilon$, select $a := \arg\max_{a' \in acts(s)} Q(s, a')$, "exploitation"
3. $r, s' := sample(s, a)$

learning rate — "value" of state s' ($x[s']$)

4. $Q(s,a) := Q(s,a) + \alpha\left(r + \underbrace{\gamma}_{\text{discounting factor}} \cdot \max_{a' \in acts(s')} Q(s', a') - Q(s,a)\right)$

$(1-\alpha)\, Q(s,a) + \alpha($ ______ " ______ $)$

5. $s := s'$
6. if $\neg term(s')$ then go to 2



Q-Learning

Q-learning algorithm:

1. Perform n learning episodes
2. Return $\max_{a \in acts(s_I)} Q(s_I, a)$ as approx. for $\mathrm{R}_{\max}(\lozenge G)$ if $\gamma = 1$

   and $\mathcal{S}_{\max} = \{ s \mapsto \arg\max_{a \in acts(s)} Q(s, a) \}$ as optimal scheduler

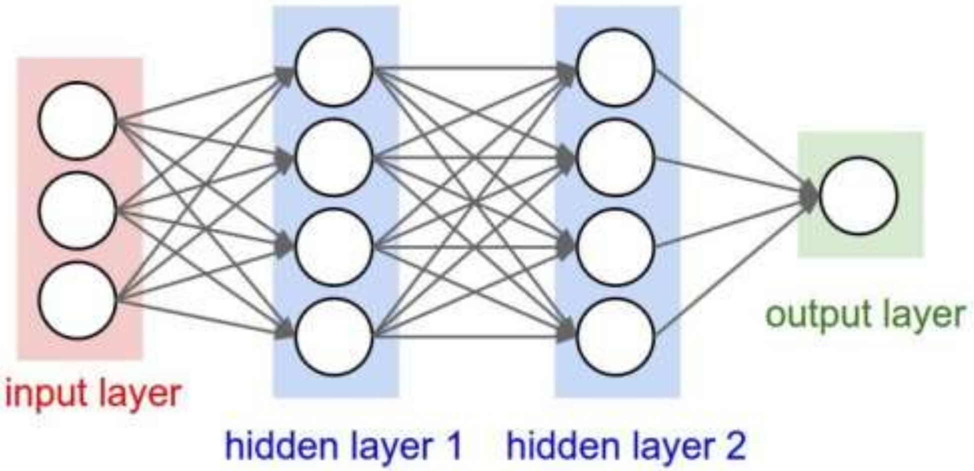
Fact: $\lim_{n \to \infty} \max_{a \in acts(s_I)} Q(s_I, a) = \mathrm{R}_{\max}(\lozenge G)$ if we play every action infinitely often

But when to stop?

SMC vs. PMC vs. Learning

	Memory usage	Runtime
Statistical Model Checking	constant (in size of DTMC) for DTMC only	depends on desired error & confidence
Probabilistic Model Checking	O(size of DTMC/MDP) to store MDP/DTMC + iteration vectors $O(\lvert S\rvert)$	depends on size of DTMC/MDP and probabilities & cycles
Reinforcement Learning (via Q-learning)	+ do not need DTMC/MDP fully in memory, just sample (like in SMC) – Q-function in $O(\lvert S\rvert \cdot \lvert A\rvert)$ but only for states we actually visit	? depends on #episodes (runs) (how to determine?)

Neural networks
are function approximators.



store Q-function
in (deep) neural network

+ fixed memory usage independent of MDP size

– no more $\lim_{n\to\infty}$ convergence,
learning behaviour very unpredictable

SMC for MDPs

DTMCs: $P(\Diamond G)=?$
MDP: $P_{\underline{max}}(\Diamond G)=?$

What about SMC for MDPs?
→ LSS: lightweight scheduler sampling

nondeterminism: must optimise, not estimate

Perform SMC for M randomly chosen schedulers, return max/min

Identify scheduler by single integer σ

step counter.
(for step-bounded props.)

$a \coloneqq (\mathcal{H}(\sigma.s) \bmod |acts(s)|)$-th element of $acts(s)$

1
$P_{max}(\Diamond G)$
??? ↕
best sampled scheduler
worst –"–
$P_{min}(G)$
0

(O(M))
+ O(1) memory usage like original SMC
– distance from best scheduler found to optimal scheduler unknown