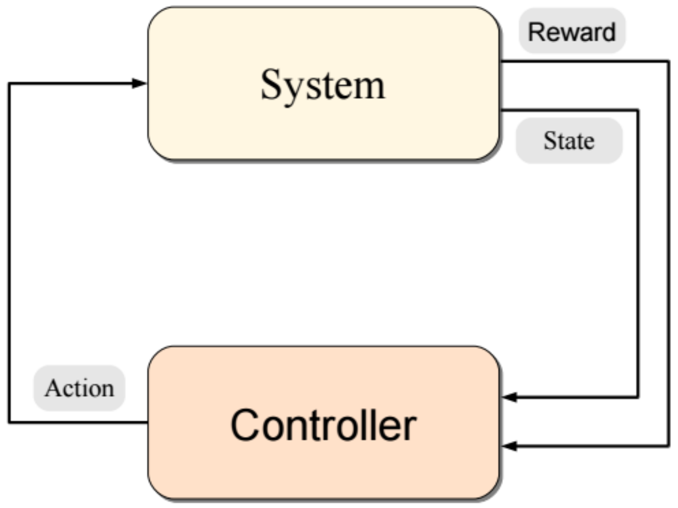
An introduction to Reinforcement Learning

Daniel Renz, Translational Neuromodeling Unit, ETHZ & UZH

12.03.2015

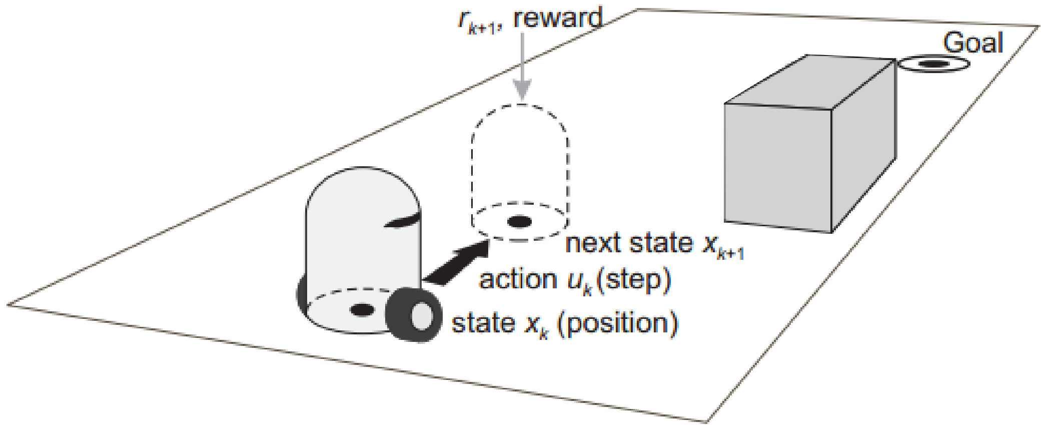
What is Reinforcement Learning (RL) ?

- Reinforcement Learning = Sequential decision-theory = Adaptive control.
- Learning by interaction: An agent learns through trial-and-error by interacting with an unfamiliar, stochastic environment.
- This means that an RL agent combines learning, planning and execution into one phase.
- Goal-directed learning: Learn to act such as to maximize expected long-term reward (*return*).
- Often sparse rewards, unlike in supervised learning.

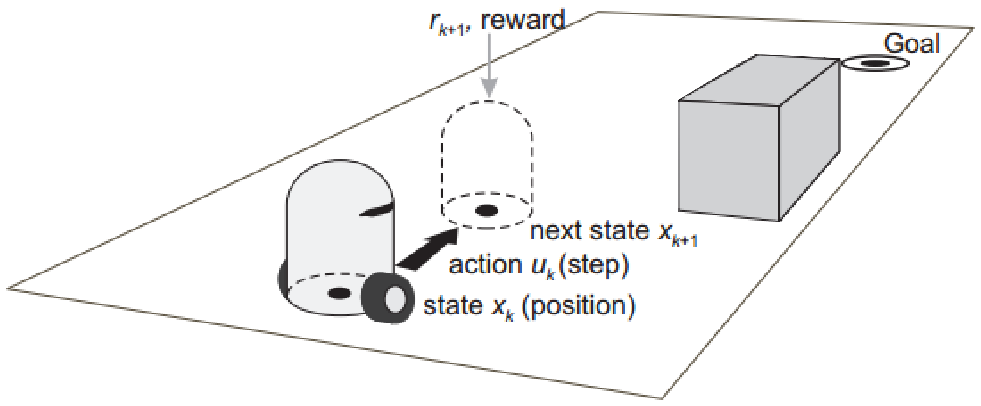


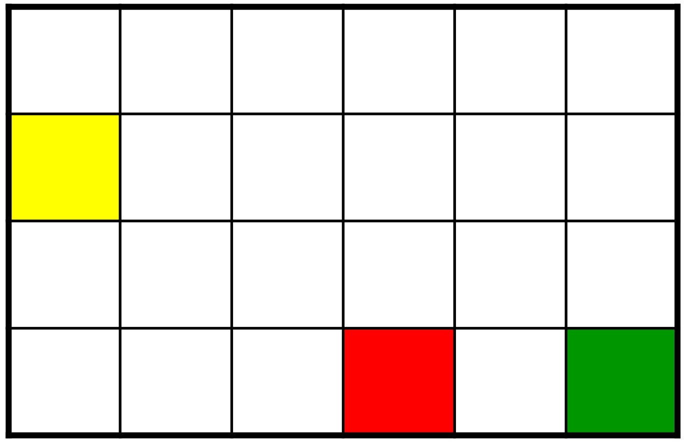
Example cleaning robot

- The agent has access to a set of *actions.*
- At each point in time, it occupies a certain *state.*
- A *reward* results from a transition (s_t, a_t, s_{t+1}).
- A *policy* $\pi(s,a) = p(a|s)$ is essentially a decision function.
- Solving the RL problem means finding the optimal policy w.r.t. expected return.



Example cleaning robot





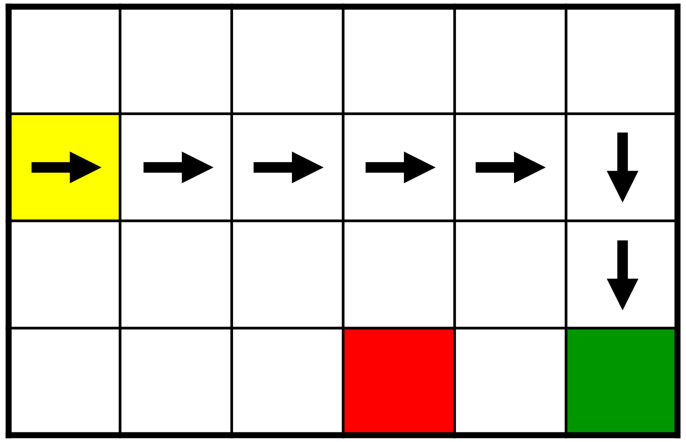
Actions: UP, DOWN, LEFT, RIGHT

Choosing UP results in

80%	move UP
10%	move LEFT
10%	move RIGHT

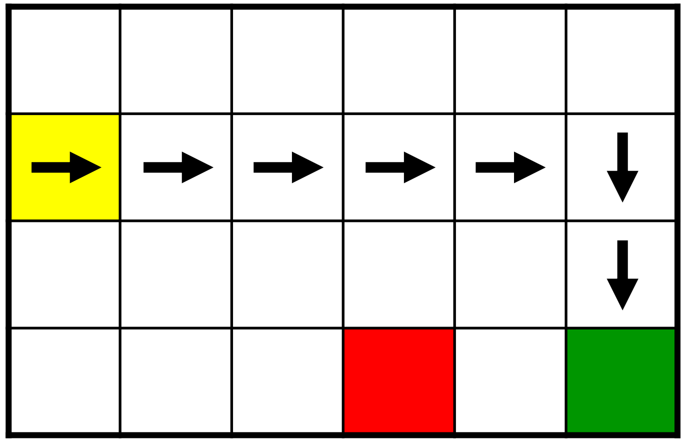
Get reward of +1 in the lower right corner.
Get reward of -0.04 for each step.
How to get max reward?

Example cleaning robot



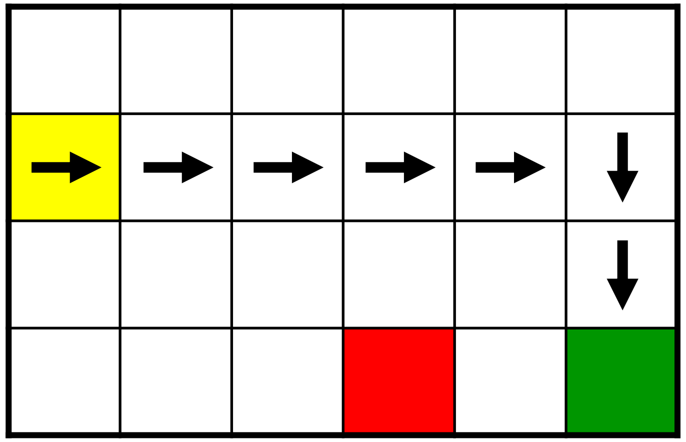
Is this a valid solution?

Example cleaning robot

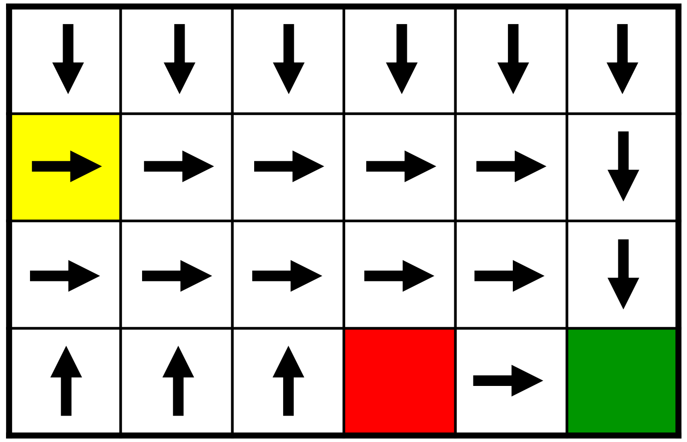


Is this a valid solution?
No, actions are stochastic.

Example cleaning robot



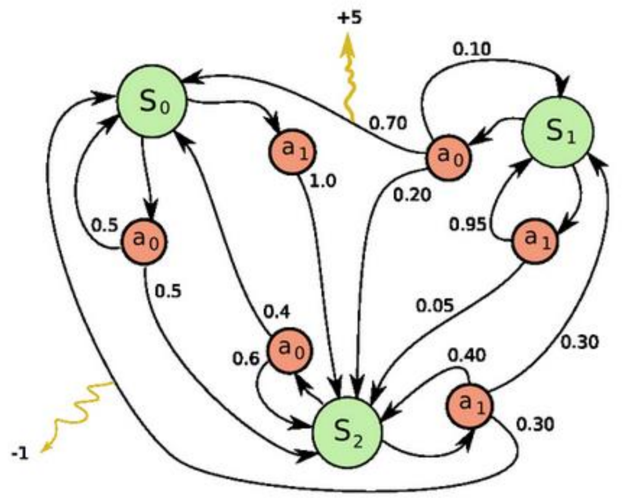
Is this a valid solution?
No, actions are stochastic.

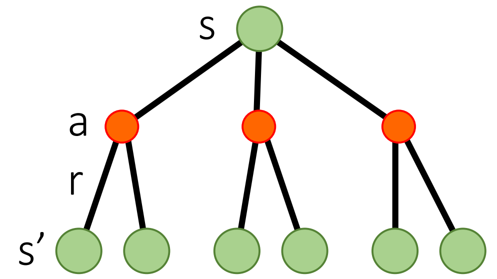


Optimal solution.

Markov Decision Process (MDP)

- The interaction between agent and environment is formalized as a Markov Decision Process (MDP).
- An MDP is a 4-tuple (S, A, T(s‘,a,s), R(s‘,a,s)). S, A are *finite* sets of states, actions.
- Transitions, and thus policies are independent of the history (Markov assumption).
- Learning is based on experience: (s, a, r, s‘).
- If T and R are known, we have a classic optimal control problem. If they are unknown, we have a classic RL problem.





Markov Decision Process (MDP)

- An MDP is a 4-tuple (S, A, T(s‘,a,s), R(s‘,a)).
- But in order to solve the RL problem, something is missing...

RL objective function

- An MDP is a 4-tuple (S, A, T(s‘,a,s), R(s‘,a)).
- But in order to solve the RL problem, something is missing...

  ...the objective function! Want to find the optimal policy π w.r.t. expected return:

$$\pi^* = \text{argmax}_\pi \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \pi(s_t))\right] = \text{argmax}_\pi V^\pi(s_0), \quad 0 < \gamma < 1$$

- We are interested in the expected discounted long-term return.
  - Effectively puts pressure on acting as soon as possible.
  - Discounting ensures that the return is bounded.

What is RL used for?

- **Planning problem:** Solve a known MDP.
- **Prediction problem:** Estimate a value function that represents how much future reward we can expect at any state, given a fixed policy.
- **Control problem:** Optimise the value function. This yields an optimal policy which allows for action selection and optimal control.

Bellman equations

- **Planning problem:** Solve a known MDP.
- **Prediction problem:** Estimate a value function that represents how much future reward we can expect at any state, given a fixed policy.
- **Control problem:** Optimise the value function. This yields an optimal policy which allows for action selection and optimal control.

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \pi(s_t))\right] \\
&= \sum_{a\in\mathcal{A}} \pi(s,a) \left(\sum_{s'\in\mathcal{S}} T(s,a,s')\,(R(s,a)+\gamma V(s'))\right) \\
&= \sum_{a\in\mathcal{A}} p(a|s) \left(\sum_{s'\in\mathcal{S}} p(s'|a,s)\,(R(s,a)+\gamma V(s'))\right)
\end{aligned}
$$

Bellman equations

How to pick the best action given a known MDP?

Planning: dynamic programming (DP)

- Find the best policy π based on knowledge of transitions T and rewards R.
- Idea
  - Initialize π arbitrarily
  - Repeat until convergence
    - Compute V^{π} from π (policy evaluation)

$$V^{\pi}(s) \leftarrow \sum_{a \in \mathcal{A}} p(a|s) \left(\sum_{s' \in \mathcal{S}} p(s'|a,s) \left(R(s,a) + \gamma V^{\pi}(s') \right) \right)$$

Prediction

    - Improve π based on V^{π} (policy improvement)

$$\pi(s) = \mathrm{argmax}_a \sum_{s'} p(s'|a,s) \left(R(s,a) + \gamma V^{\pi}(s') \right)$$

Control

Prediction using Monte Carlo (MC) methods

- Replace knowledge of T, R with experience, to approximate the policy evaluation step.
- Idea: Estimate state value / expected return by averaging over sample returns iteratively (this implements a running average).
- Repeat until convergence
  - Generate episode N given π
  - For each state s occuring in episode:
    Set R_N to the return following the first occurance of s and update moving average

$$\begin{aligned}
\bar{R}_N &= \frac{1}{N}\sum_{n=1}^{N} R_n \\
&= \frac{1}{N}\left(R_N + \sum_{n=1}^{N-1} R_n\right) \\
&= \frac{1}{N}\left(R_N + (N-1)\bar{R}_{N-1}\right) \\
&= \bar{R}_{N-1} + \frac{1}{N}\left(R_N + \bar{R}_{N-1}\right)
\end{aligned}$$

- Using a fixed alpha corresponds to an exponential moving average. This makes recent samples more important and forgets the distant past.

$$\hat{R}_N = \hat{R}_{N-1} + \alpha\left(R_N + \hat{R}_{N-1}\right)$$

Prediction: Temporal difference (TD) Learning

- Combination of ideas from Monte Carlo and Dynamic Programming
  - Model-free: Sample the environment according to policy (MC)
  - Bootstrap: approximate current estimate based on previously learned estimates (DP). Thus, we can update at every step.

$$\hat{V}(s_t) \approx R_t = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i} = r_{t+1} + \gamma R_{t+1}$$

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha(R_t - \hat{V}(s_t))$$

Need to approximate R_t ...

Prediction: Temporal difference (TD) Learning

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha(R_t - \hat{V}(s_t))$$

$$\begin{aligned} R_t &= r_{t+1} + \gamma R_{t+1} \\ &\approx r_{t+1} + \gamma \hat{V}(s_{t+1}) \end{aligned}$$

$$\begin{aligned} \hat{V}(s_t) &\leftarrow \hat{V}(s_t) + \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \\ &= \hat{V}(s_t) + \alpha\delta \end{aligned}$$

- We essentially assume that the value of the next state is an accurate estimate of the expected return downstream.
- No need to know T, R. We use the frequencies of observations as estimates to directly update V.

Prediction: Temporal difference (TD) Learning

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha(R_t - \hat{V}(s_t))$$

$$\begin{aligned} R_t &= r_{t+1} + \gamma R_{t+1} \\ &\approx r_{t+1} + \gamma \hat{V}(s_{t+1}) \end{aligned}$$

$$\begin{aligned} \hat{V}(s_t) &\leftarrow \hat{V}(s_t) + \alpha(r_{t+1} + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)) \\ &= \hat{V}(s_t) + \alpha\delta \end{aligned}$$

- We essentially assume that the value of the next state is an accurate estimate of the expected return downstream.
- No need to know T, R. We use the frequencies of observations as estimates to directly update V.
- Great idea! But we really want to solve the control problem.

$$\pi(s) = \text{argmax}_a \sum_{s'} p(s'|a,s)\left(R(s,a) + \gamma V^{\pi}(s')\right)$$

Bellman equations

- **Planning problem:** Solve a known MDP.
- **Prediction problem:** Estimate a value function that represents how much future reward we can expect at any state, given a fixed policy.
- **Control problem:** Optimise the value function. This yields an optimal policy which allows for action selection and optimal control.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \pi(s_t))\right] \\ &= \sum_{a\in\mathcal{A}} \pi(s,a) \left(\sum_{s'\in\mathcal{S}} T(s,a,s') \left(R(s,a) + \gamma V(s')\right)\right) \\ &= \sum_{a\in\mathcal{A}} p(a|s) \left(\sum_{s'\in\mathcal{S}} p(s'|a,s) \left(R(s,a) + \gamma V(s')\right)\right) \end{aligned}$$

Bellman equations

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}\left[R(s_0,a) + \sum_{t=1}^{\infty} \gamma^{t-1} R(s_t, \pi(s_t))\right] \\ &= \sum_{s'\in\mathcal{S}} T(s,a,s') \left(R(s,a) + \gamma \left(\sum_{a'\in\mathcal{A}} \pi(s',a') Q^\pi(s',a')\right)\right) \\ &= \sum_{s'\in\mathcal{S}} p(s'|a,s) \left(R(s,a) + \gamma \left(\sum_{a'\in\mathcal{A}} p(a'|s') Q^\pi(s',a')\right)\right) \end{aligned}$$

Control with Q-Learning

- Q-values make control-life really simple.

$$\begin{aligned}\pi(s) &= \text{argmax}_a \sum_{s'} p(s'|a,s)\left(R(s,a) + \gamma V^\pi(s')\right) \\ &= \text{argmax}_a Q^\pi(s,a)\end{aligned}$$

- When updating Q-values, we assume the agent acts greedily.

$$\hat{Q}(s_t,a_t) \quad \leftarrow \quad \hat{Q}(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1},a) - \hat{Q}(s_t,a_t)\right]$$

Control with Q-Learning

- Repeat until convergence:
  - Choose action according to some policy.
  - Update Q-values assuming that the agent is acting greedily.

$$\hat{Q}(s_t, a_t) \quad \leftarrow \quad \hat{Q}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1}, a) - \hat{Q}(s_t, a_t) \right]$$

- This is amazing: It is guaranteed to converge *independent* of the policy (as long as every state is reachable).
- It does not solve the exploration vs. exploitation issue (which policy should we use to *efficiently* learn the Q-values?).
- Want to choose a policy that appears to maximize Q most of the time, but choose some other action the rest of the time.
- Nice demo: http://blog.allanbishop.com/q-learning/

Which policy to use?

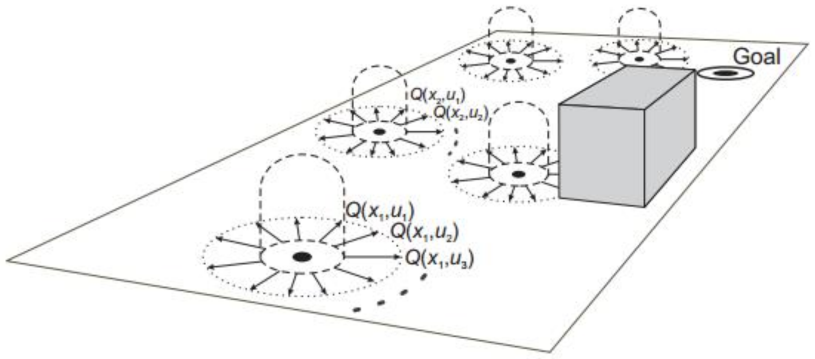
- Choosing the right policy can greatly influence convergence speed.
- Greedy: Always exploit. Not very useful.
- ε-greedy: With probability ε choose uniformly from non-optimal actions.
- Softmax: Choose actions according to their values. If the temperature T is large, all actions are approximately equally probable.

$$p(a|s) = \frac{\exp(T^{-1}\hat{Q}(a,s))}{\sum a' \exp(T^{-1}\hat{Q}(a',s))}$$

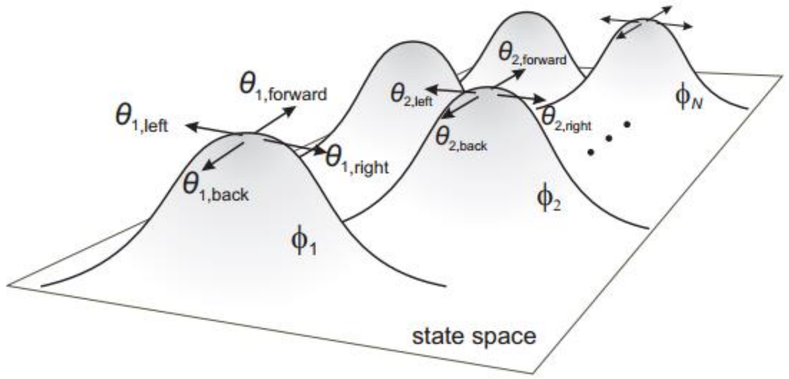
- Optimistic exploration: Preferably choose actions that have been undersampled.

Dealing with the real world

- In the real world, we cannot usually learn about every single state, or even action.
- We can usually discretize action space, but state space is too high-dimensional.
- We can use function approximation. This allows generalization.



Action discretization

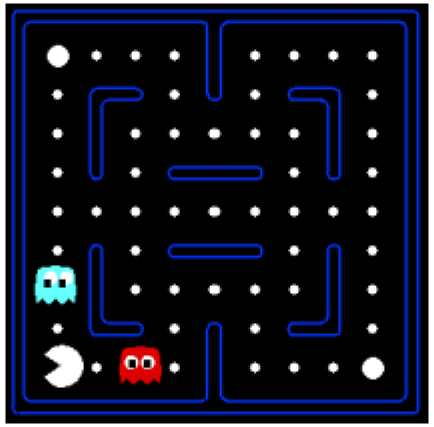


State approximation

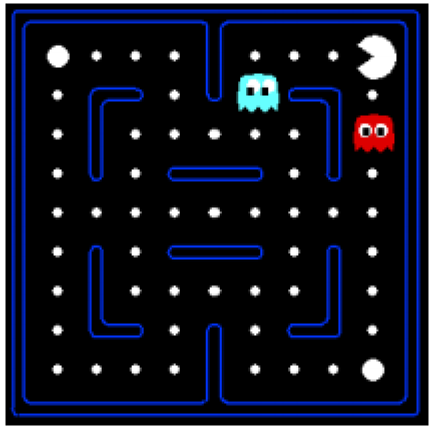
Dealing with the real world

- Pacman has learned that (A) is bad. What about (B)?
- We can describe a state using a vector of features **f**(s,a)
  - Distance to closest ghost, to closest dot
  - Number of ghosts
  - No. of action moves bringing Pacman closer to dots
  - ...etc.

(A)



(B)



Function approximation

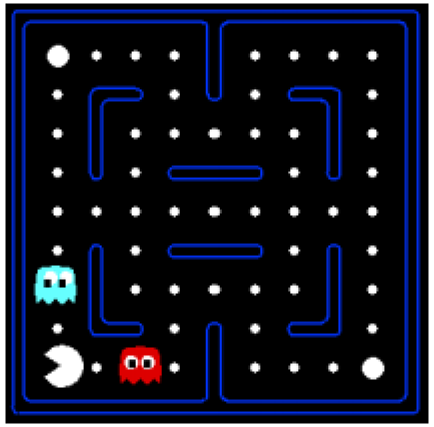
- Use linear combination of features

$$\begin{aligned} Q(s,a) &= \mathbf{w}^T \mathbf{f}(s,a) \\ \hat{Q}(s_t,a_t) &\leftarrow \hat{Q}(s_t,a_t) + \alpha\delta \\ \mathbf{w} &\leftarrow \mathbf{w} + \alpha\delta\mathbf{f}(s,a) \end{aligned}$$

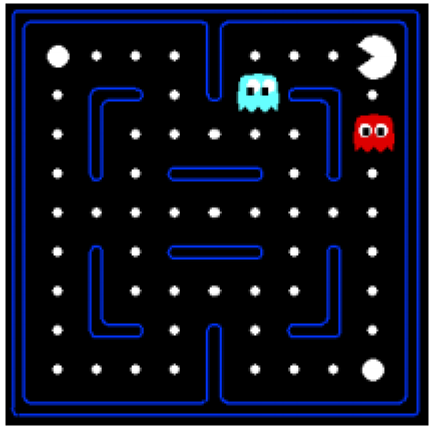
- This is equivalent to the update rule in online least-squares regression.
- General supervised learning principle:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\left[y - \hat{y}(\mathbf{x})\right]\nabla_{\mathbf{w}}\hat{y}(\mathbf{x})$$

(A)



(B)



Thanks for your attention!

Jürgen Schmidhuber

- First universal reinforcement learner for essentially arbitrary computable environments - the first optimal general rational agent.
- This motivated the recent work on the Goedel machine for universal reinforcement learning with limited computational resources.
- http://people.idsia.ch/~juergen/rl.html
- Realistic environments are not fully observable. General learning agents need an internal state to memorize important events in case of POMDPs. The essential question is: how can they learn to identify and store those events relevant for further optimal action selection? Schmidhuber has studied reinforcement learners with recurrent neural network value function approximators.

Human-level control through deep RL

- RL provides a normative account of optimal control.
- It is deeply rooted in psychological & neuroscientific perspectives
- How to derive efficient representations of the environment from high-dimensional sensory inputs?
- How to generalize past experience to new situations?
- Humans combine RL and hierarchical sensory processing
- We want to use RL in combination with deep learning to build something like a deep Q-network (DQN, Mnih et al, Nature 2015). This principle draws on neurobiological evidence that reward signals during perceptual learning may influence the representations within primate visual cortex

Human-level control through deep RL

- Deep convolutional network: Hierarchical layers of tiled convolutional filters mimic the effects of receptive fields. This exploits the local spatial correlations present in images, and builds robustness to natural transformations such as changes of viewpoint or scale.
- Approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t=t_0}^{T} \gamma^t R(s_t) | s_0 = s, a_0 = a, \pi\right]$$

- But RL using nonlinear function approximations the Q-function is unstable. This is due to (i) correlations in the observations, (ii) the fact that small changes to Q may significantly change the policy and (iii) correlations between Q(s,a)-values and target values

$$r + \gamma \max_{a'} Q(s', a')$$

- A novel iterative update rule adjusts Q-values such that correlations with target values are reduced.
- Use approximate value function $Q(s,a;\theta_i)$, where θ_i are the parameters of a deep convolutional neural network at iteration i

- Experience replay randomizes over data. This removes correlations in the observation sequence and smoothes over changes in the data distribution. During learning, we sample uniformly from the set of the agent‘s experiences
  and apply Q-Learning to minibatches of experience e using the following loss function

$$\{e_t\},\ \text{where } e_t = (s_t, a_t, r_t, s_{t+1})$$

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

- Convergent evidence suggests that the hippocampus may support the physical realization of such a process in the brain, with the time-compressed reactivation of recently experienced trajectories during offline periods (for example, waking rest) providing a putative mechanism by which value functions may be efficiently updated through interactions with the basal ganglia.

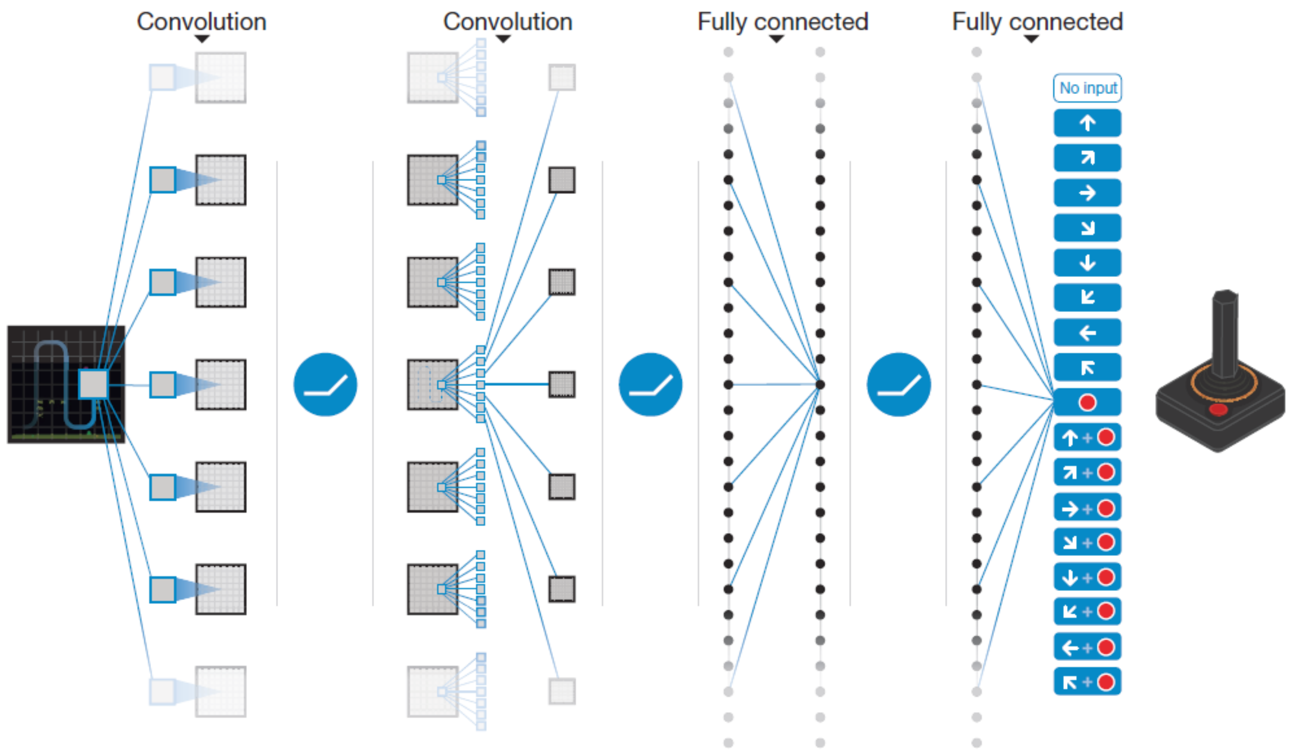


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an 84 × 84 × 4 image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Mnih et al, Nature 2015

TODO

- How to bias content of experience replay towards salient events? (in relation to prioritized sweeping in RL)
- Jürgen Schmidhuber‘s opinion: http://people.idsia.ch/~juergen/naturedeepmind.html
- DEMO: http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html