Reinforcement learning

Master CPS, Year 2 Semester 1

Lucian Buşoniu, Florin Gogianu

Part VIII

Deep Reinforcement Learning

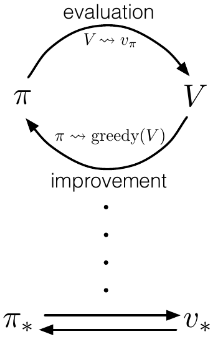
Recap

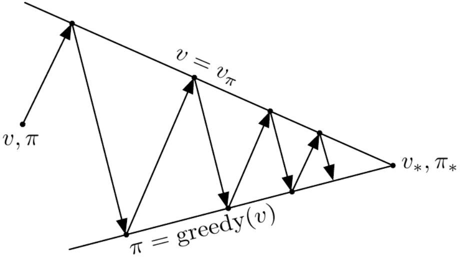
- The two fundamental problems of RL-based control:
  - Policy **evaluation**
  - Policy **improvement**

Generalized Policy Iteration

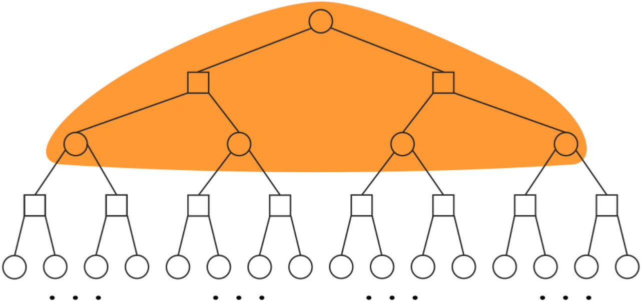
Generalized Policy Iteration demo

Generalized Policy Iteration





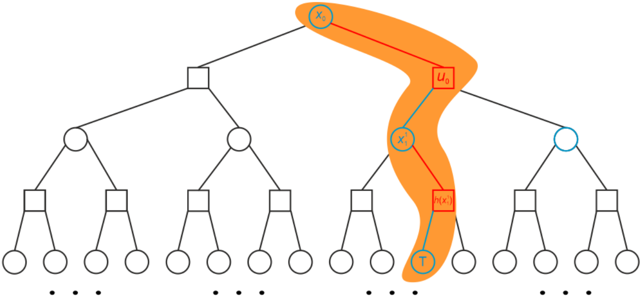
Dynamic Programming



$$V(x) \leftarrow \sum_{u} h(x,u) \sum_{x'} f(x,u,x') \left[r + \gamma V(x') \right]$$

Problem: we need access to $f(x,u,x')$.

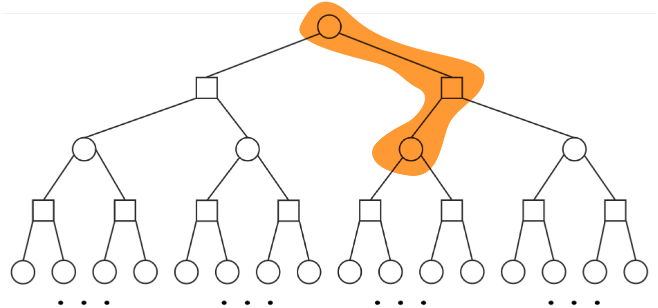
Monte Carlo Estimation



$$V(x) \leftarrow V(x) + \alpha\left[R - V(x)\right]$$

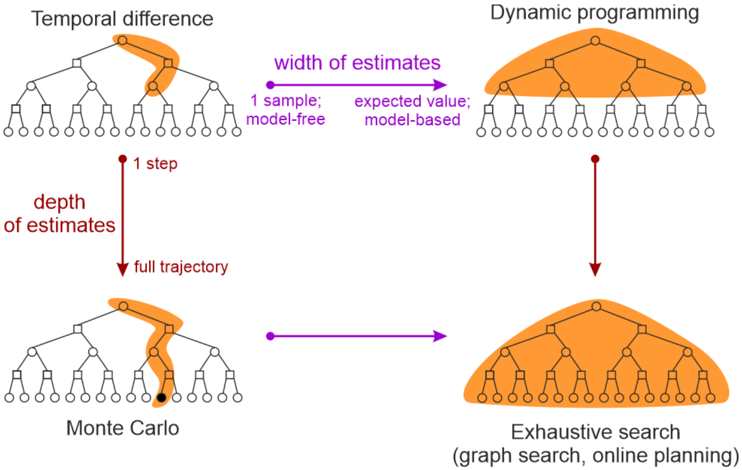
Problem: we need to wait until the end of the episode.

Temporal Differences



$$V(x) \leftarrow V(x) + \alpha \left[r + \gamma V(x') - V(x) \right]$$

Unified Perspective



Recap

- The two fundamental problems in RL-based control:
  - Policy evaluation
  - Policy improvement
- Evaluation (and control) with **function approximation**.

Learning with Approximation

We approximate $V^h(x) \approx \widehat{V}^h(x;\theta)$ and minimize the objective:

$$\mathcal{L}(\theta) = \sum_{x \in \mathcal{X}} \mu(x) \left[V^h(x) - \widehat{V}^h(x;\theta) \right]^2,$$

where $\mu(x)$ is the state distribution.

- converges to a local minimum in the general case,
- in the linear case, there is a single minimum point.

Learning with Approximation

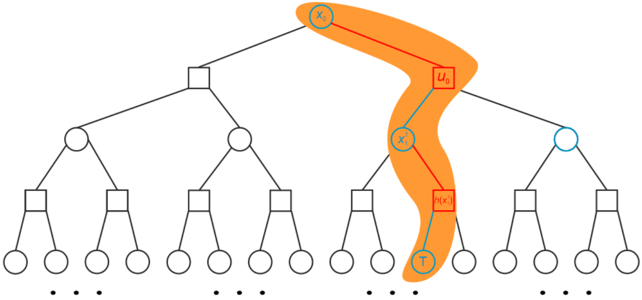
Iterative parameter update rule (gradient descent) for $\mathcal{L}(\theta)$:

$$\begin{aligned}\theta_{t+1} &= \theta_t - \frac{1}{2}\alpha\nabla\left[V^h(x) - \widehat{V}^h(x;\theta)\right]^2 \\ &= \theta_t + \alpha\left[V^h(x) - \widehat{V}^h(x;\theta)\right]\nabla\widehat{V}^h(x,\theta)\end{aligned}$$

Converges with $\alpha \to 0$ under relatively strict conditions (e.g., linearity).

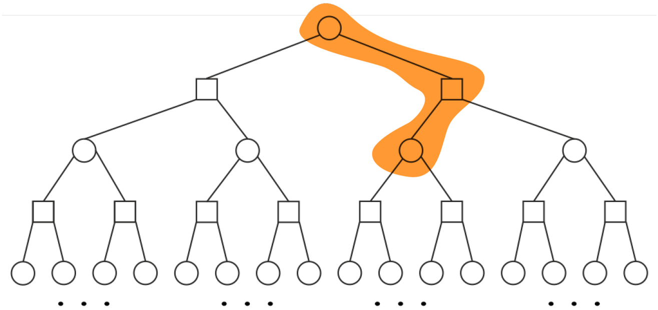
Problem: we do not know $V^h(x)$!

Approximation with MC Regression Targets



$$\theta_{t+1} = \theta_t + \alpha \left[R - \widehat{V}^h(x;\theta) \right] \nabla \widehat{V}^h(x,\theta)$$

Approximation with the Temporal Difference Objective

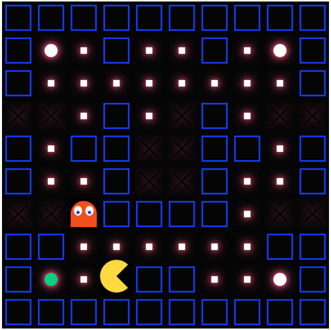


$$\theta_{t+1} = \theta_t + \alpha \left[r + \gamma \widehat{V}^h(x'; \theta) - \widehat{V}^h(x; \theta) \right] \nabla \widehat{V}^h(x, \theta)$$

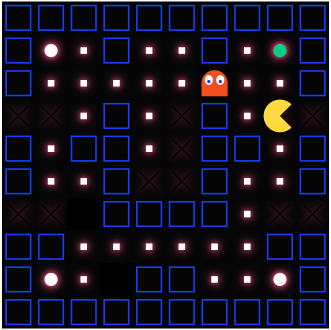
Problem: this is no longer the true gradient!

Why neural networks in reinforcement learning?

Motivation



We want $\widehat{V}(x;\theta)$ trained on x ...

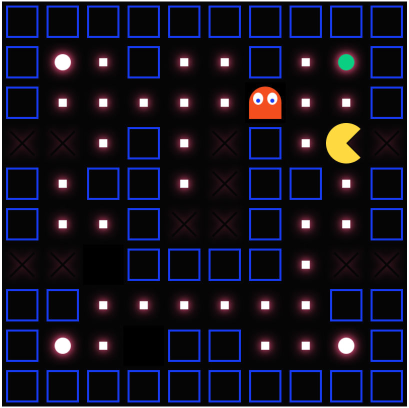


... to be similar for a new x.

Motivation

Even with access to the internal state of the game, we would need to extract various attributes:

- distances to the ghosts
- distance to the nearest power-up
- distance to the walls
- whether Pac-Man is in a bottleneck
- distance to the nearest dots
- dots already consumed



Attributes that are relevant only for a single *MDP*.

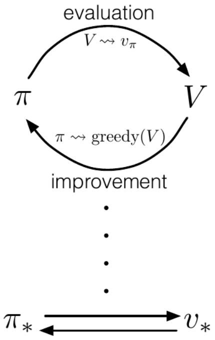
(Neural) Fitted Q-Learning

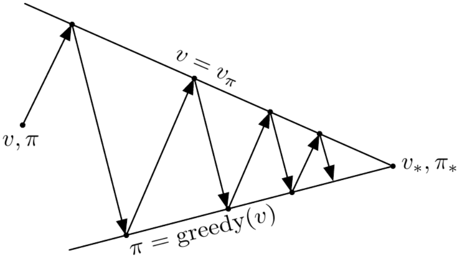
Strategy: approximate $Q(x,u)$ with a neural network $Q(x,u;\theta)$ and attempt to implement an algorithm that solves the policy evaluation problem.

1. initialize $Q(x,u;\theta)$ such that $Q(x,u;\theta) \approx 0, \forall \{x,u\}$.
2. collect a dataset $\mathcal{D} = \{(x,u,r,x'),...\}$
3. construct regression targets $Y_k^Q = r + \gamma \max_u Q(x',u';\theta)$
4. minimize $\mathcal{L}(\theta) = \left[Q(x,u;\theta) - Y_k^Q\right]^2$ using minibatch gradient descent
5. repeat from (3)
6. repeat from (2)

Problem: updating θ changes the targets in a correlated way; for expressive estimators like neural networks, this leads to error accumulation.

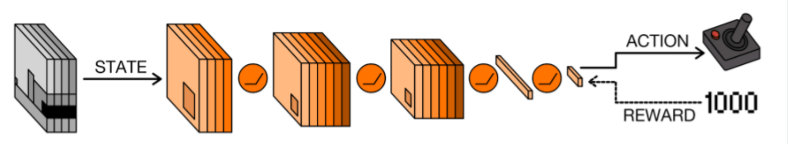
(Neural) Fitted Q-Learning





NFQI alternates infrequently between the two problems, and as a result, converges slowly.

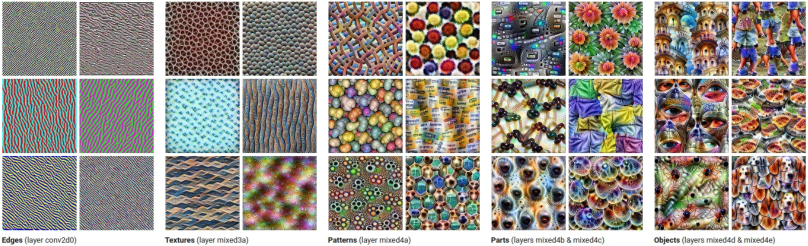
Neural Network Architecture



Deep Q-Networks architecture

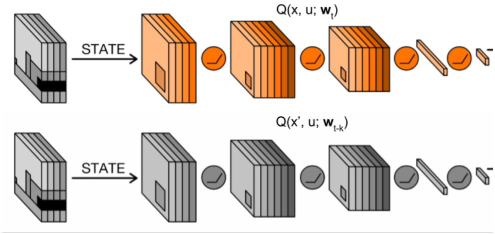
- receives the last 4 greyscaled game screens
- three hidden convolutional layers
- one hidden linear layer
- one linear output layer (sometimes with a shared bias for all actions)
- one output for every action
- RELU activations

Convolutional Networks



Hierarchical representations learned by a convolutional network from a natural image dataset.

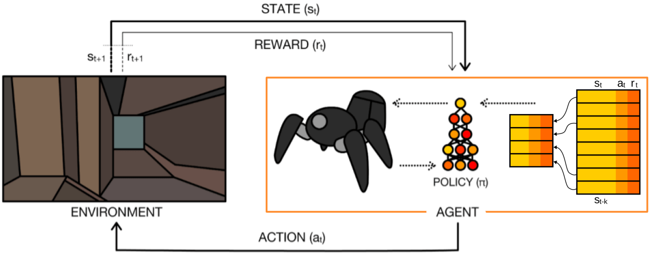
Deep Q-Networks. Fix #1: Target Network



DQN introduces an additional network for computing the regression target, which ‘tracks’ the online estimator.

The target network is not trained — its weights are copied from the online network every C steps.

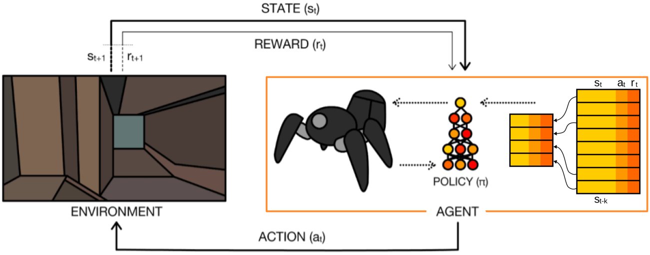
Deep Q-Networks. Fix #2: Experience Replay



DQN uses a cyclic buffer, which allows simultaneously:

- training estimators with mini-batch stochastic gradient descent (as in NFQI),
- and enabling rapid iteration of policy improvement.

Other Important Adjustments

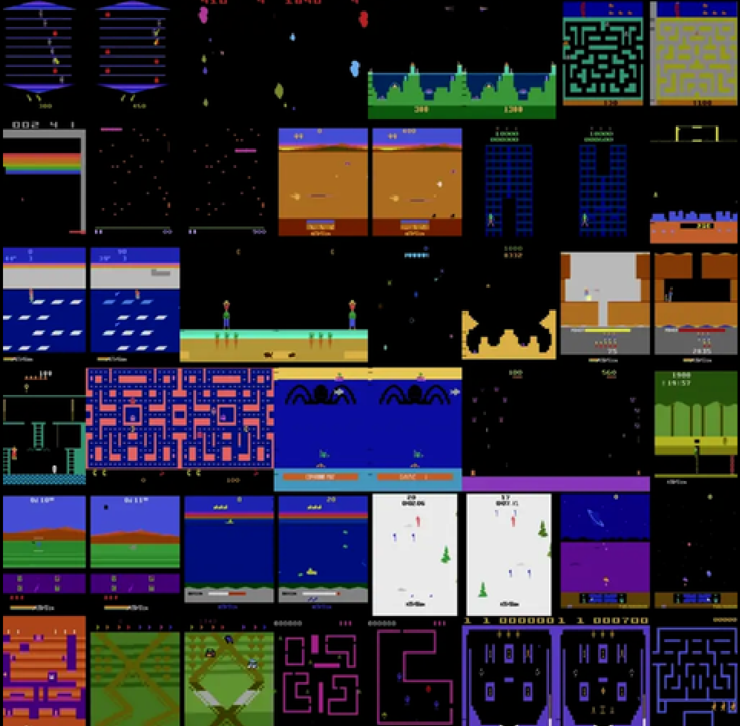


- rewards are clipped to $[-1, 1]$,
- the optimizer used is RMSprop (more recently, Adam),
- the loss function is Huber/Smooth L1 (L2 near 0, L1 for large values),
- in some implementations, gradient norm clipping is used.

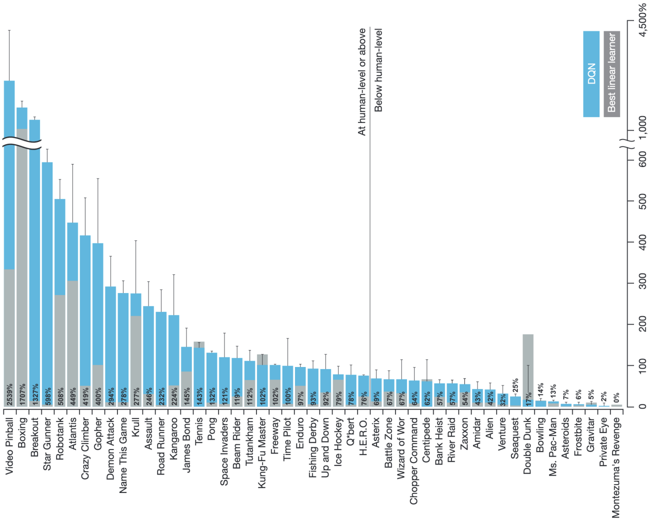
Algorithm

Deep Q-Networks

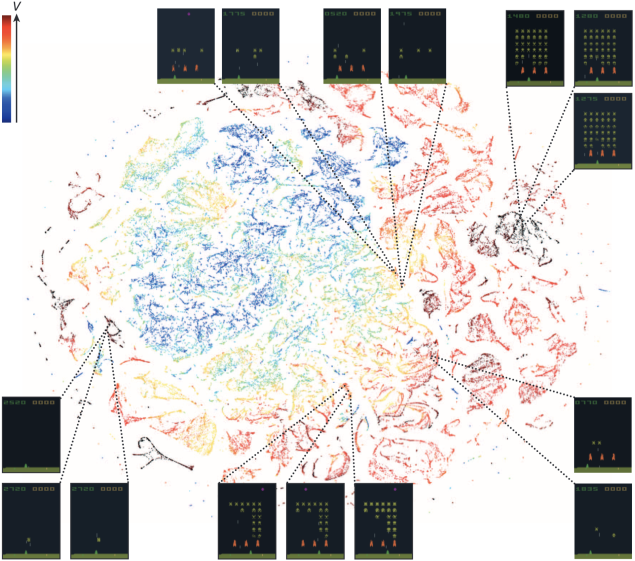
init replay buffer D with capacity N
init action-value function Q with random weights θ
init target action-value function $\bar{Q}$ with weights $\bar{\theta} = p$
for each episode **do**
  receive initial state $x_0 \leftarrow$ env()
  for $t = 1, T$ **do**
    $u_t \leftarrow$ epsilon-greedy $(Q(x_t, u; \theta), \varepsilon)$
    $r_t, x_{t+1} \leftarrow$ env (u_t)
    $D \leftarrow (x_t, u_t, r_t, x_{t+1})$
    sample batch of K transitions (x_j, u_j, r_j, x_{j+1}) from D
    $$y_j = \begin{cases} r_j & \text{if episode is terminal} \\ r_j + \gamma \max_u \bar{Q}\left(x_{t+1}, u; \bar{\theta}\right) & \text{otherwise} \end{cases}$$
    Do gradient descent on $\left(y_j - Q\left(x_t, u_t; \theta\right)\right)^2$ w.r.t weights θ
    Every C steps $\bar{\theta} = \theta$
  end for
end for



Deep Q-Networks. Perf. Relative to a Human Player



Deep Q-Networks. Representation Learning



Beyond Deep Q-Networks

Several directions for improving DQN:

- improved objectives: Double-DQN, Dueling DQN, Munchausen-DQN, n-step TD
- distributions instead of point estimates: Categorical DQN, IQM
- improved sampling: Prioritized Experience Replay
- improved exploration: Random Network Distillation, Go-Explore, Bootstrapped DQN

Open Problems in Deep RL

- Exploration is an open problem (not only with function approximation)
- Scalling deep neural networks
- Transfer and continual learning