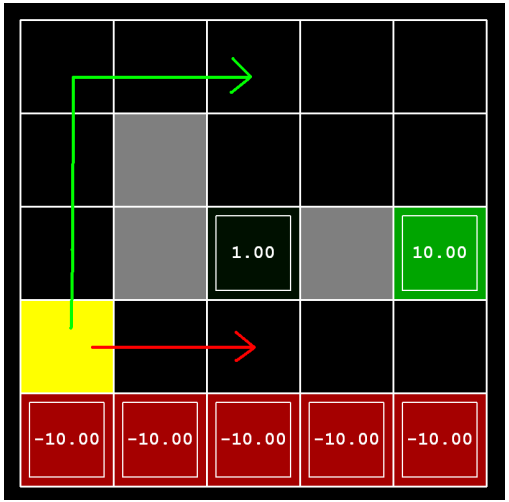
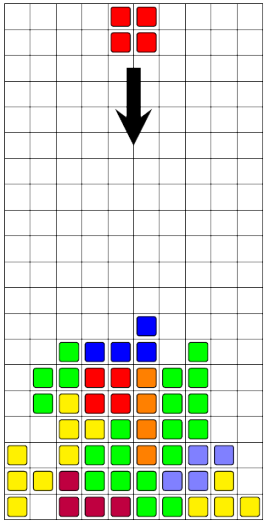
Can tabular methods scale?

- Discrete environments



Gridworld
10^1

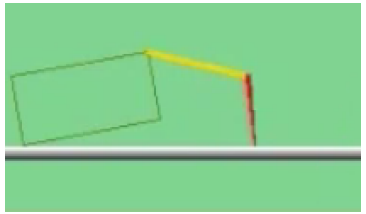


Tetris
10^60

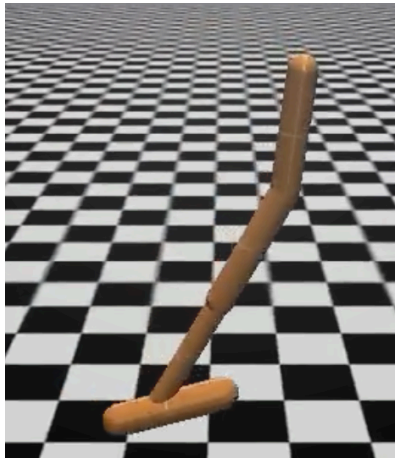
Atari
10^308 (ram) 10^16992 (pixels)

Can tabular methods scale?

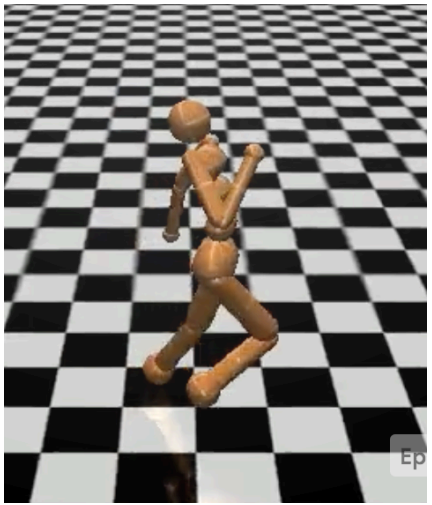
- Continuous environments (by crude discretization)



Crawler
10^2



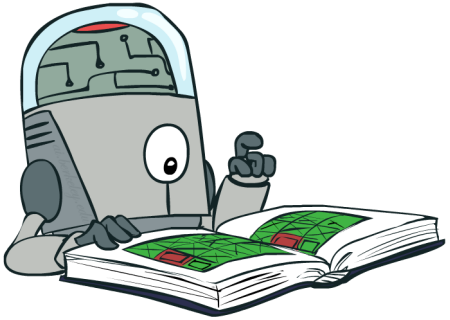
Hopper
10^10

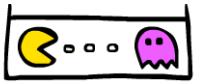


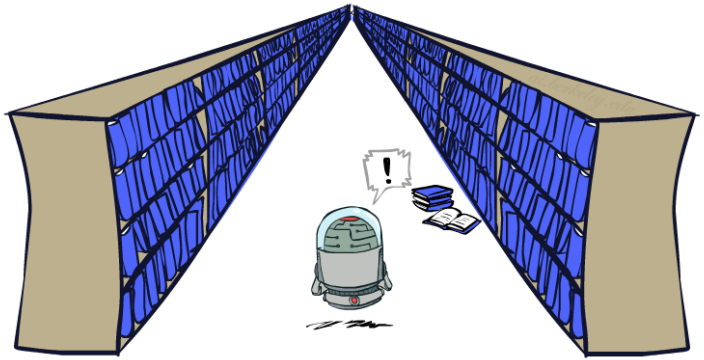
Humanoid
10^100

Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again







Approximate Q-Learning

- Instead of a table, we have a parametrized Q function: $Q_\theta(s, a)$
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a complicated neural net
- Learning rule:
  - Remember: $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$
  - Update:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

Connection to Tabular Q-Learning

- Suppose $\theta \in \mathbb{R}^{|S|\times|A|}, \quad Q_\theta(s,a) \equiv \theta_{sa}$

$$\nabla_{\theta_{sa}} \left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2\right]$$
$$= \nabla_{\theta_{sa}} \left[\frac{1}{2}(\theta_{sa} - \text{target}(s'))^2\right]$$
$$= \theta_{sa} - \text{target}(s')$$

- Plug into update: $\theta_{sa} \leftarrow \theta_{sa} - \alpha(\theta_{sa} - \text{target}(s'))$
$$= (1-\alpha)\theta_{sa} + \alpha[\text{target}(s')]$$

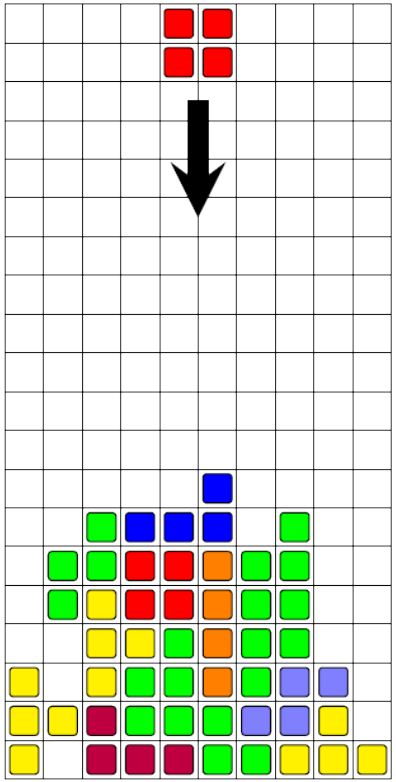
- Compare with Tabular Q-Learning update:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\,[\text{target}(s')]$$

Engineered Approximation Example: Tetris

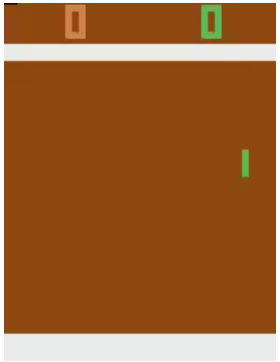
- state: naïve board configuration + shape of the falling piece ~10^{60} states!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, . . . , 9, *mapping the state to the height h[k] of each* column.
  - Nine basis functions, 10, . . . , 18, *each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 9$.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$

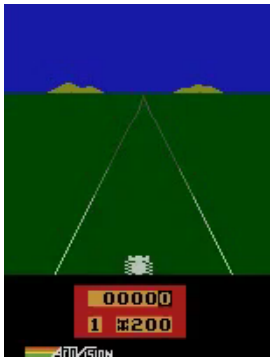


[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

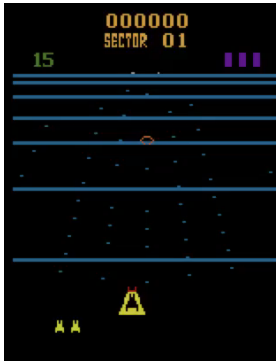
Deep Reinforcement Learning



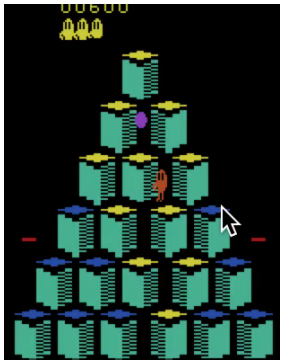
Pong



Enduro



Beamrider



Q*bert

- From pixels to actions
- Same algorithm (with effective tricks)
- CNN function approximator, w/ 3M free parameters

Recap: Approximate Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[(Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

$$s \leftarrow s'$$

Chasing a nonstationary target!

Updates are correlated within a trajectory!

DQN

- High-level idea - **make Q-learning look like supervised learning**.
- Two main ideas for stabilizing Q-learning.
- Apply Q-updates on batches of past experience instead of online:
  - **Experience replay** (Lin, 1993).
  - Previously used for better data efficiency.
  - Makes the data distribution more stationary.

- Use an older set of weights to compute the targets (**target network**):
  - Keeps the target function from changing too quickly.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

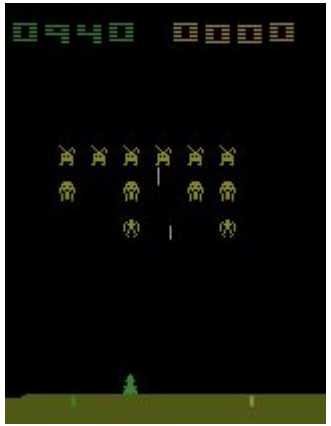
"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Target Network Intuition

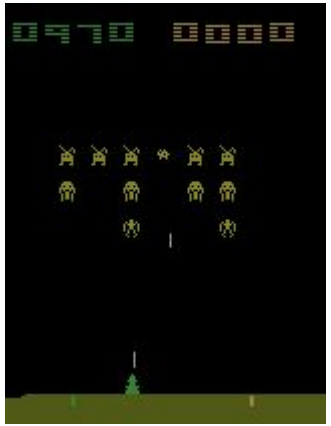
- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

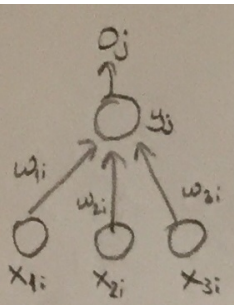
$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

s



s'





$y' = \sum w_i x_i$ $\quad$ $\hat{y}$: Desired output

$E = \frac{1}{2}(\hat{y} - y')^2$ $\quad$ y' : Calculated output

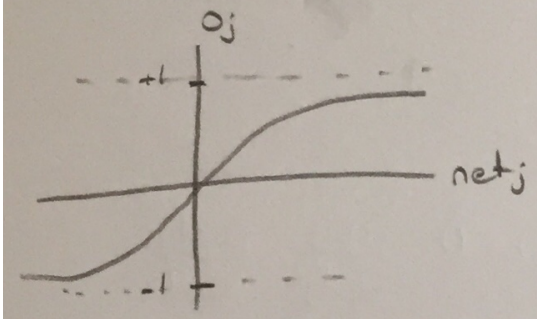
$$\Delta w_i = -\alpha \cdot \frac{\partial E}{\partial w_i} = -\alpha \cdot (\hat{y} - y') \cdot (-1) \cdot \frac{dy'}{dw_i}$$

$$= \alpha \cdot (\hat{y} - y') \cdot x_i$$

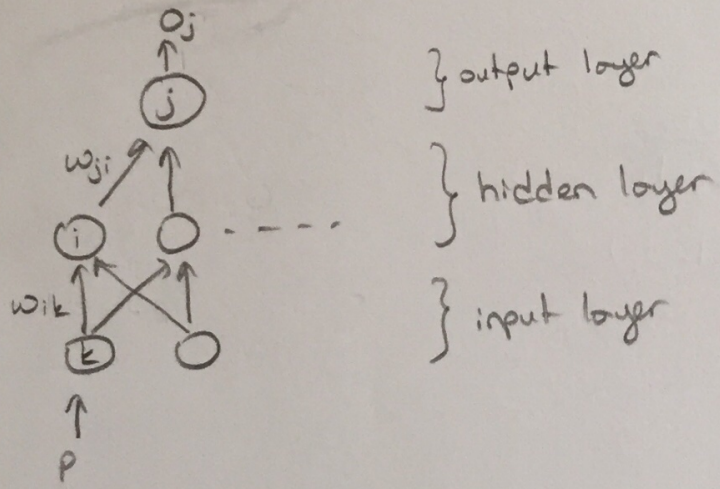
$net_j = \sum_i w_j \cdot x_i$ $\quad$ $o_j = \begin{cases} 1 & \text{if } net_j \geq 0 \\ 0 & \text{otherwise} \end{cases}$

For normalized or precise output, we are going to use squashing function

$f(x) = \tanh(x) \Rightarrow o_j = \tanh(net_j)$



$\Rightarrow \frac{\partial f(x)}{\partial x} = 1 - f^2(x)$



t_{pj} = desired output for node j

$$Error_{on_j} = \sum_j \frac{1}{2}(t_{pj} - o_j)^2$$

$net_j = \sum_i w_{ji} \cdot x_i$ $\quad$ $net_i = \sum_k w_{ik} \cdot x_k$

$o_j = \tanh(net_j)$ $\quad$ $o_i = \tanh(net_i)$

$x_i = o_i$

~~A~~ We want to calculate $\frac{\partial E}{\partial w}$, the rate of change of error with respect to the given connective weight, so we can minimize it.

$$\frac{\partial E}{\partial \omega_{ji}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial \omega_j}$$

$$= (t_{pj} - o_j) \cdot (-1) \cdot (1 - o_j^2) \cdot x_i$$

$$\Delta \omega_{ji} = -\alpha \cdot \frac{\partial E}{\partial \omega_{ji}}$$

$$= +\alpha \cdot \underbrace{(t_{pj} - o_j) \cdot (+1) \cdot (1 - o_j^2)}_{\delta_{pj}} \cdot x_i$$

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_j}$$

$$\Delta \omega_{ji} = \alpha \cdot \delta_{pj} \cdot x_i$$

$$\Delta \omega_{ik} = -\alpha \cdot \frac{\partial E}{\partial \omega_{ik}} = -\alpha \cdot \underbrace{\frac{\partial E}{\partial o_i} \cdot \frac{\partial o_i}{\partial net_i}}_{-\delta_{pi}} \cdot \frac{\partial net_i}{\partial \omega_{ik}}$$

$$\Delta \omega_{ik} = \alpha \cdot \delta_{pi} \cdot x_k$$

$$\delta_{pi} = \sum_j (t_{pj} - o_j) \cdot (-1) \cdot \frac{\partial o_j}{\partial net_i} = \sum_j (t_{pj} - o_j) \cdot (-1) \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial net_i}$$

$$\sum_j (t_{pj} - o_j) \cdot (-1) \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial o_i} \cdot \frac{\partial o_i}{\partial net_i}$$

$$= \left(\sum_j \underbrace{(t_{pj} - o_j) \cdot (-1) \cdot (1 - o_j^2)}_{-\delta_{pj}} \cdot \omega_{ji} \right) (1 - o_i^2)$$

$$\delta_{pi} = (1 - o_i^2) \cdot \sum_j -\delta_{pj} \cdot \omega_{ji}$$