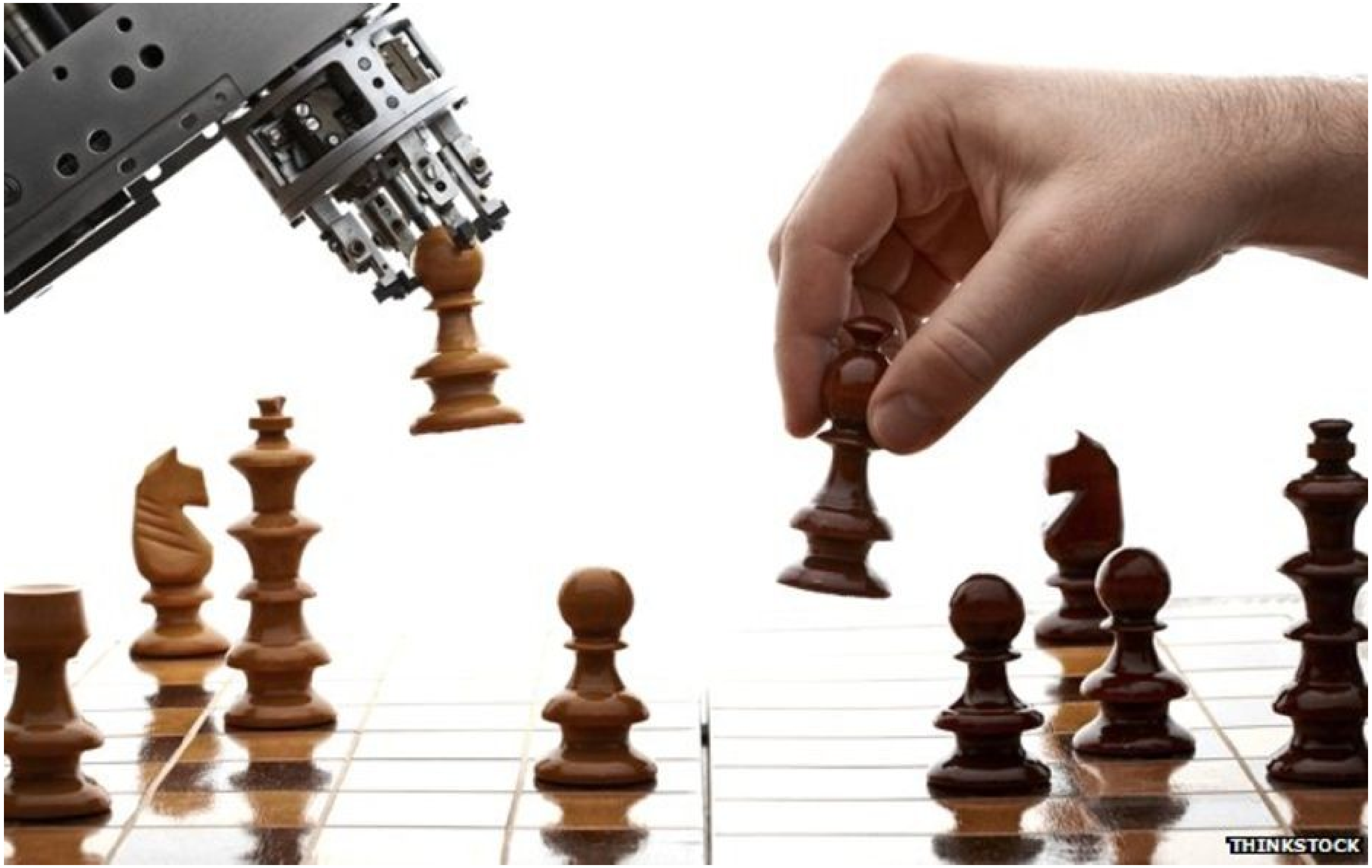


Introduction to Artificial Intelligence

COSC 4550 / COSC 5550

Professor Cheney

11/10/17

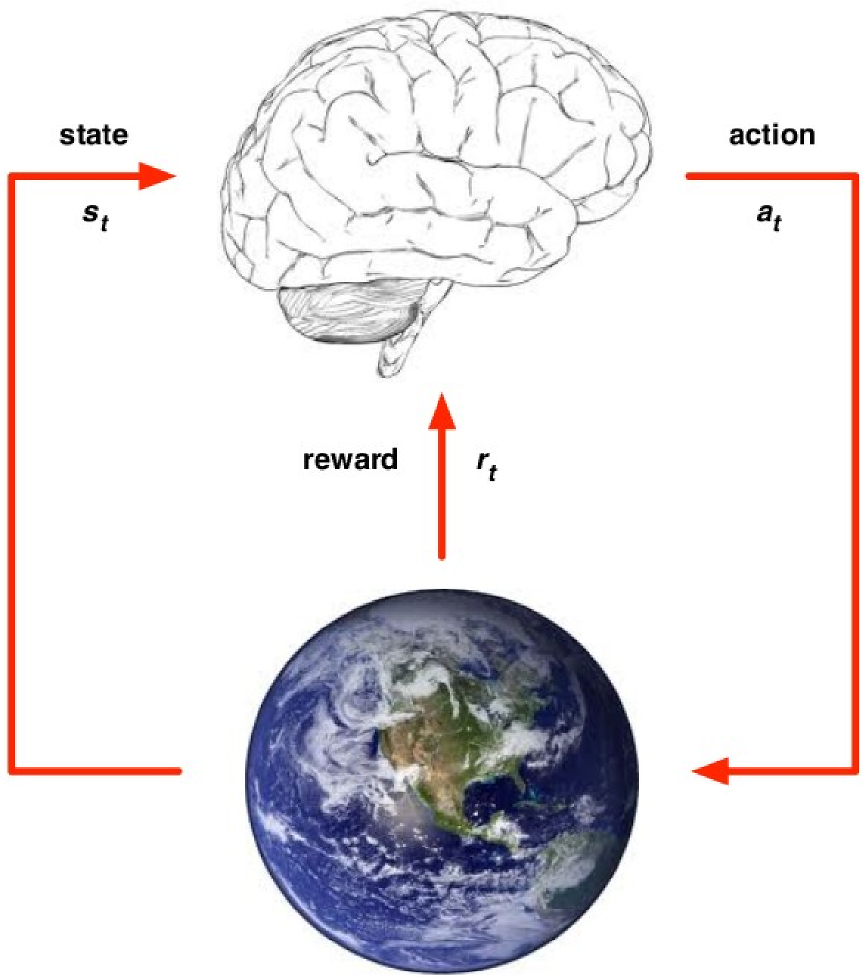
deep reinforcement learning!

Deep Reinforcement Learning

from

David Silver, Google DeepMind

Agent and Environment



- At each step t the agent:
  - Receives state s_t
  - Receives scalar reward r_t
  - Executes action a_t
- The environment:
  - Receives action a_t
  - Emits state s_t
  - Emits scalar reward r_t

Approaches To Reinforcement Learning

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Model-based RL

- Build a transition model of the environment
- Plan (e.g. by lookahead) using model

Deep Reinforcement Learning

- Can we apply deep learning to RL?
- Use deep network to represent value function / policy / model
- Optimise value function / policy /model end-to-end
- Using stochastic gradient descent

learning value networks

Bellman Equation

- Value function can be unrolled recursively

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right] \\ &= \mathbb{E}_{s'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right] \end{aligned}$$

- Optimal value function $Q^*(s, a)$ can be unrolled recursively

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right]$$

- Value iteration algorithms solve the Bellman equation

$$Q_{i+1}(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s', a') \mid s, a\right]$$

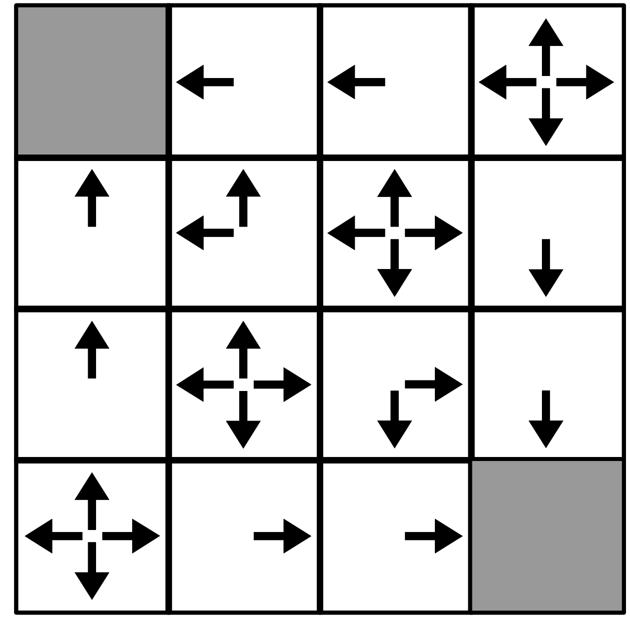
before (in our reinforcement learning lecture)
we represented the value of a state (V(s))
or the value of state-action pair (Q(s,a))
as a table with a different entry for
every possible state in our game

current value (V_k) for a random policy

greedy policy (π_k) for a this value function

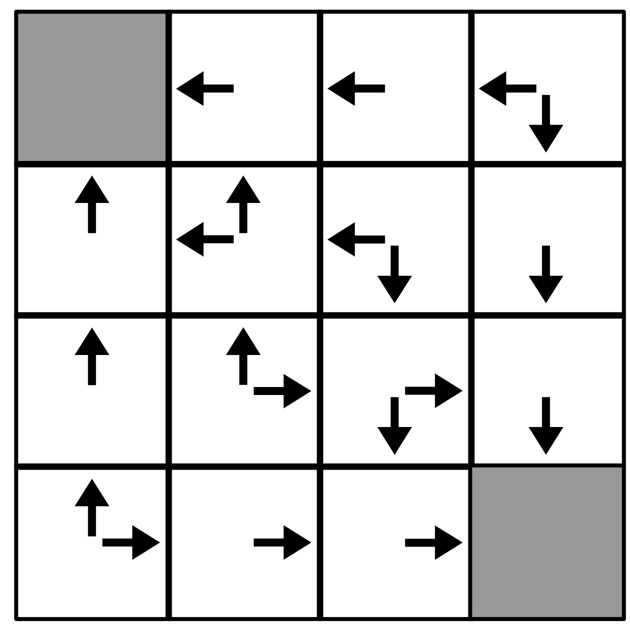
k=2

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0



k=3

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



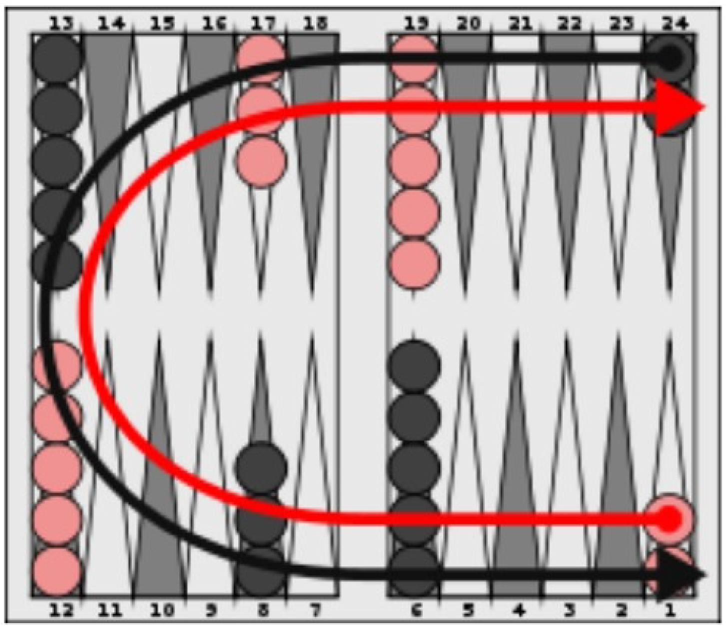
but this falls apart if we have many states
(e.g. complex games, or continuous actions)

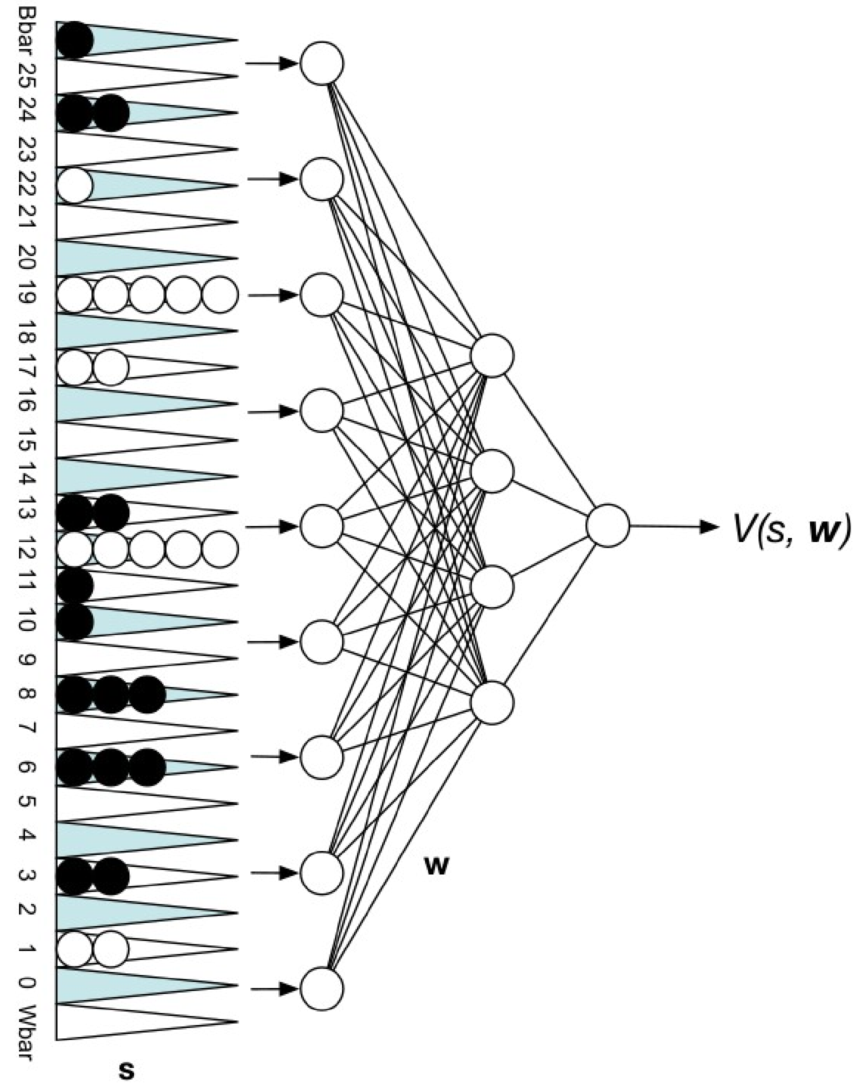
unfortunately this happens to be most scenarios

so we'll apply the same ideas, but now build
a continuous function from features to values

this function will be a neural network!

Example: TD Gammon





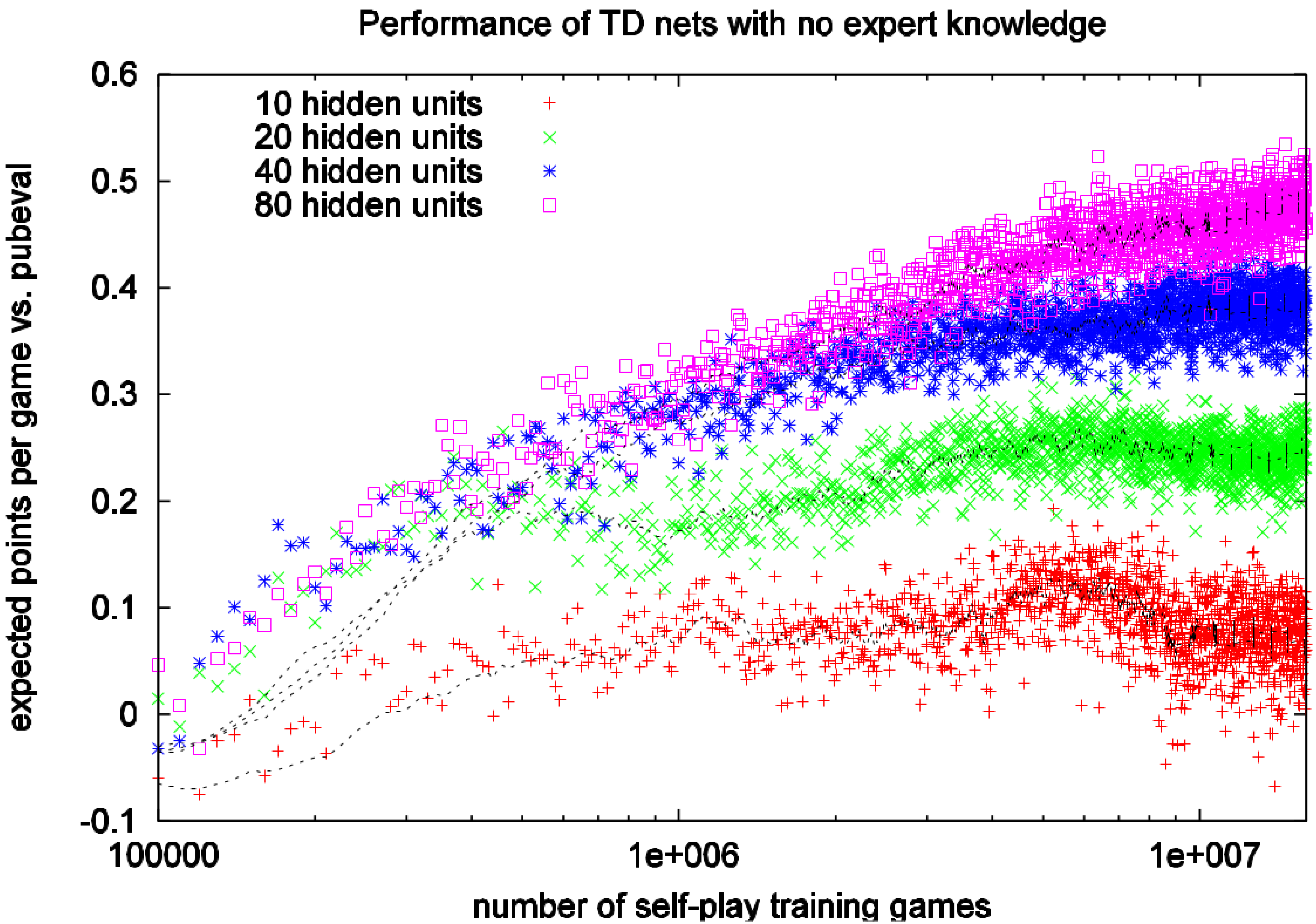
Self-Play Non-Linear Sarsa

- Initialised with random weights
- Trained by games of self-play
- Using non-linear Sarsa with afterstate value function

$$Q(s, a, w) = \mathbb{E}\left[V(s', w)\right]$$

- Greedy policy improvement (no exploration)
- Algorithm converged in practice (not true for other games)
- TD Gammon defeated world champion Luigi Villa 7-1 (Tesauro, 1992)

New TD-Gammon Results



Deep Q-Learning

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
2. Policy changes rapidly with slight changes to Q-values
   - Policy may oscillate
   - Distribution of data can swing from one extreme to another
3. Scale of rewards and Q-values is unknown
   - Naive Q-learning gradients can be large
     unstable when backpropagated

Deep Q-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
2. Freeze target Q-network
   - Avoid oscillations
   - Break correlations between Q-network and target
3. Clip rewards or normalize network adaptively to sensible range
   - Robust gradients

Stable Deep RL (1): Experience Replay

To remove correlations, build data-set from agent's own experience

- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]$$

Stable Deep RL (2): Fixed Target Q-Network

To avoid oscillations, fix parameters used in Q-learning target

- Compute Q-learning targets w.r.t. old, fixed parameters w^-

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

- Optimise MSE between Q-network and Q-learning targets

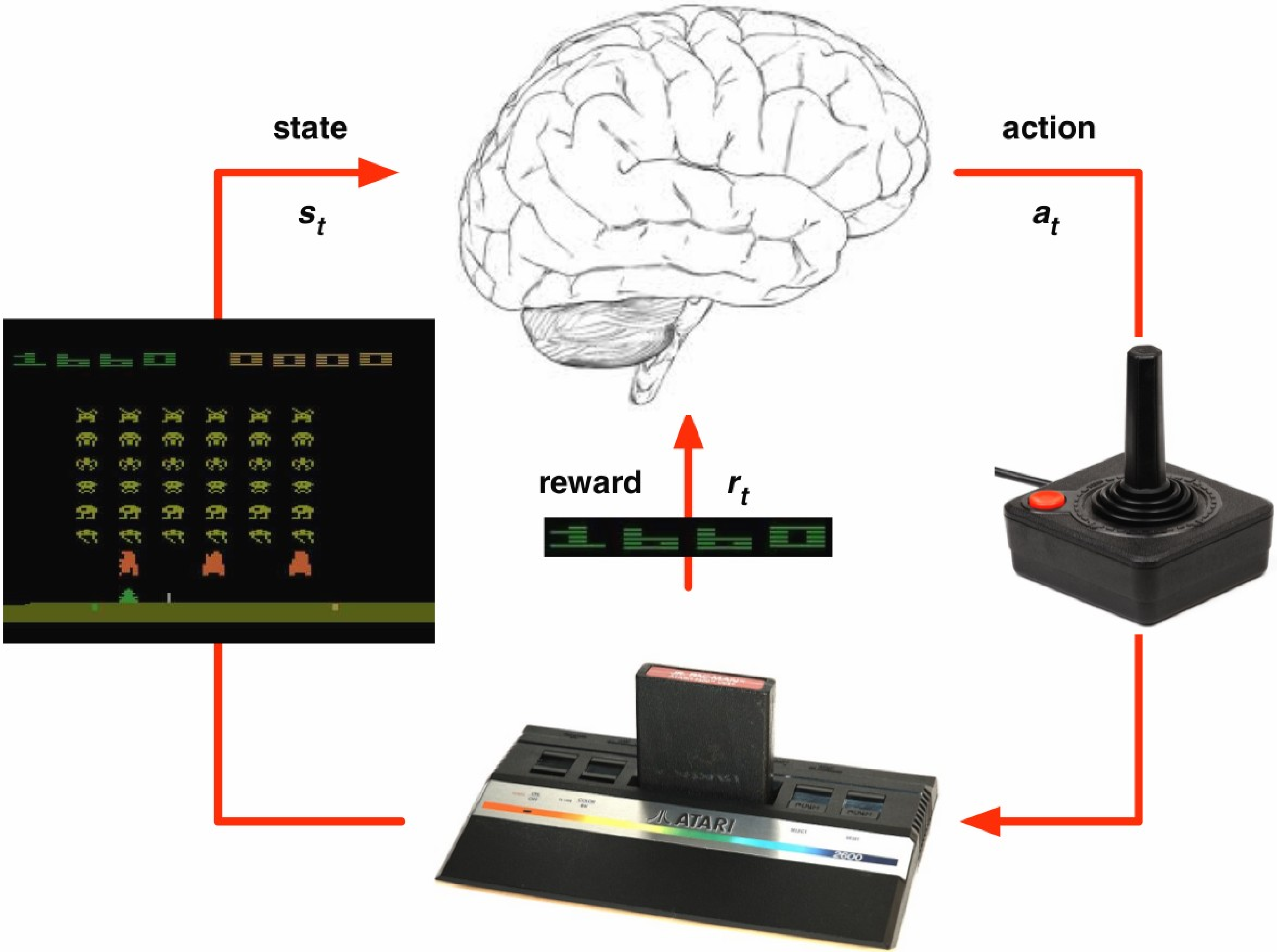
$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

- Periodically update fixed parameters $w^- \leftarrow w$

Stable Deep RL (3): Reward/Value Range

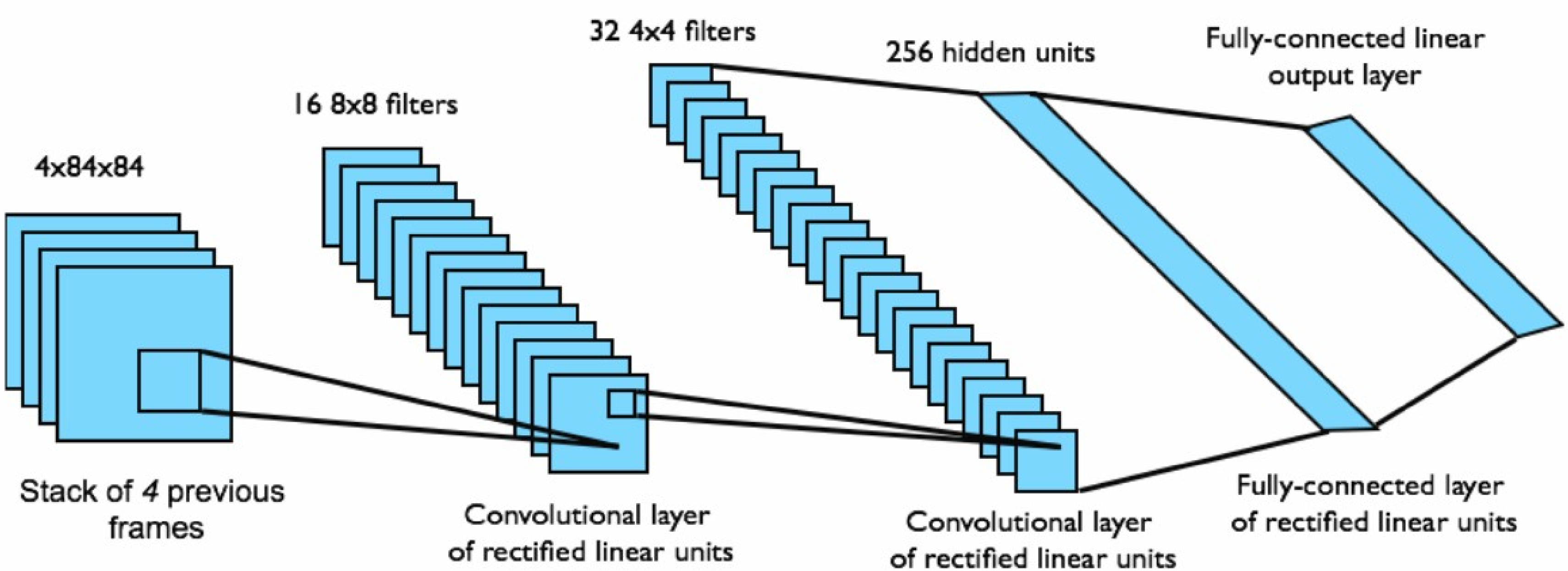
- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
- Can't tell difference between small and large rewards

Reinforcement Learning in Atari



DQN in Atari

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

Space Invaders

DQN controls the green laser cannon to clear columns of space invaders descending from the sky and also destroys two pink motherships at the top of the screen

DQN Results in Atari