Deep Reinforcement learning

11-364: Intro to deep learning | Rameez Remsudeen

RL Example

Difference b/w RL and un/supervised learning

01

Actions affect future events

02

There is not supervisor, only reward signals

03

Rewards are far in the future – cannot be myopic

04

Number of examples seen are limited before a critical action has to be made

05

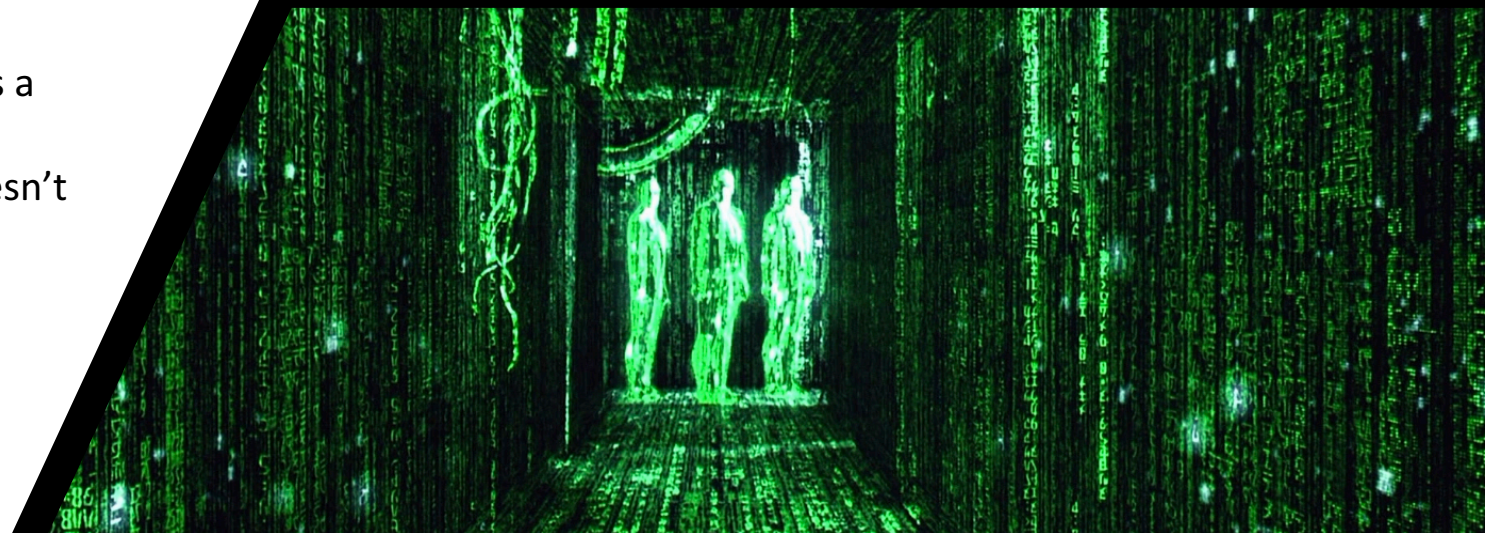
Composing patterns learned is easy, but composing behaviors is harder

RL.setup()

Given timestep $t = 0,1 \ldots n$

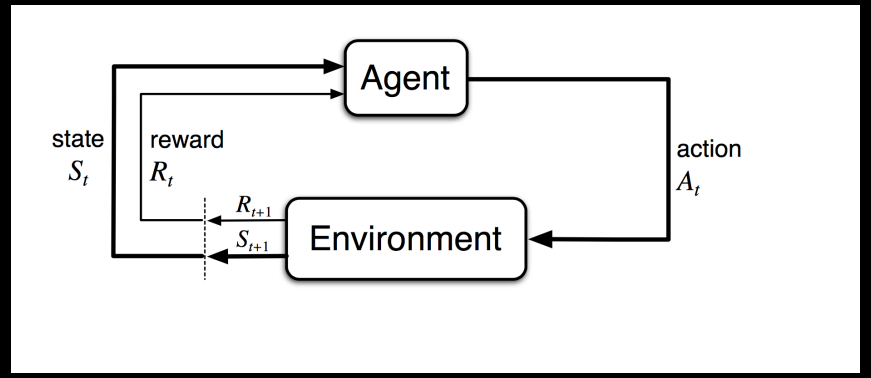
- Agent
  - Performs action $a_t \in A(s_t)$where $s_t \in S$
    - $A(s)$ - set of possible actions available for state s
    - s_t – state of agent at timestep t

- Environment
  - Receives an action a_t from agent and gives a reward $r_{t+1}$and new state $s_{t+1}$to agent
  - Only environment gives reward. Agent doesn't make its own reward



Inside the mind of the agent

- Reward is a scalar feedback signal – how well agent is doing at time step t
- Based on Reward hypothesis
  - All goals can be described by maximization of expected cumulative reward
- Only what should be achieved is specified, not how to
- Agent implements mapping from state to *probability of selecting possible action* through policy π_t
  - $\pi_t(a\,|s)$ =probability that $a_t = a\ is\ selected\ given\ s_t = s$
  - Policy fully determines behavior of the agent
- Agent tries to maximize cumulative rewards



Rewards

- The return $G_t = R_{t+1} + R_{t+2} \ldots . R_T$ where G is cumulative future rewards
- To ensure convergence we have discounting factor γ
- Like Epochs, RL uses Episodes – end at time step T or at terminal state
- Discounting – sum of discounted rewards over future is maximized
  - a_t is chosen s.t. expected discounted return is maximized
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots . \cong \sum_{k=0}^{\infty} \gamma^k R_{t+K+1}$
- γ can be either 0 or 1
  - 0 = myopic
  - 1 = far sighted

Key insight: *Future Independent of past, given the present*

Markov Property

Models of RL

- Markov Process (chain) given by tuple $< S, P >$
- Markov Reward Process given by $< S, P, R, \gamma >$
- Markov Decision Process given by $< S, A, P, R, \gamma >$
  - S – set of states
  - A – set of actions
  - P – transition probability matrix
    - P says how likely the next state is given the current state and action
  - R – reward function, $R(a, s) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
  - γ - discount factor between 0 and 1

Value function

- Value function determines how good it is for an agent to be in a state or take an action
  - State-value function – expected return from starting at state s and following policy π
    - $v_\pi(s) = \mathbb{E}[G_t|S_t = s]$
  - Action-value function (also called q value function) – expected return from starting at state s, taking action a and following policy π
    - $q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]$

Value function redux: optimized

- Optimal value function suggests best possible performance in MDP
  - Optimal state value function is maximum value function over all policies
    - $v_*(s) = \max_\pi v_\pi(s)$
  - Optimal action value function is maximum action value function over all policies
    - $q_*(s,a) = \max_\pi q_\pi(s,a)$
- MDP is solved when optimal value function is known

Optimal Policy

- For any MDP, $\exists\ optimal\ policy\ \pi_* \geq \pi_a \forall\ a \in \mathbb{N}$
- Optimal policies achieve optimal value functions and optimal action value functions
  - $v_{\pi_*}(s) = v_*(s)\ \forall s \in S$
  - $q_{\pi_*}(s,a) = q_*(s,a)\ \forall s \in S$

RL Problems

1

Prediction

Given MDP and a policy, find the state and action value functions

2

Planning

Given MDP find the optimal policy

Approaches to RL

- Value-based => Estimate optimal value function
- Policy based => Search for optimal policy
- Model based => Builds model of the environment, often through planning
  - Model is often learned from experience

Q-learning

Q-value function

- Recall discounted future returns - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots.$
- Thus action value function is
  - $q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} \ldots | s,a]$
  - $q_\pi(s,a) = \mathbb{E}_{s',a'}[R_{t+1} + \gamma q_\pi(s',a') \mid s,a]$, Bellman Equation
- Similarly, optimal action value function can be a Bellman equation too
  - $q_{*_\pi}(s,a) = \mathbb{E}_{s'}\left[R_{t+1} + \gamma \max'_a q_*(s',a') \mid s,a\right]$

Deep RL

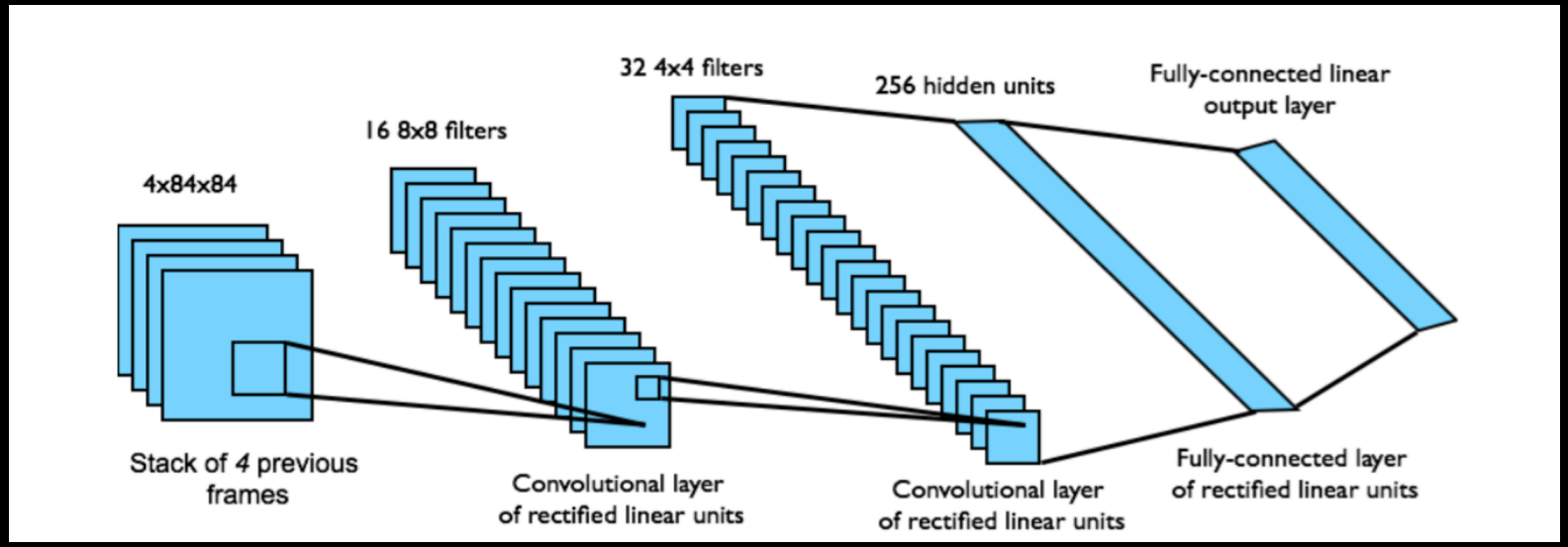
- Key Idea:
  - Represent using neural nets
    - Value Function
    - Policy
    - Model
  - Use SGD to minimize cost function value

Deep Q Networks

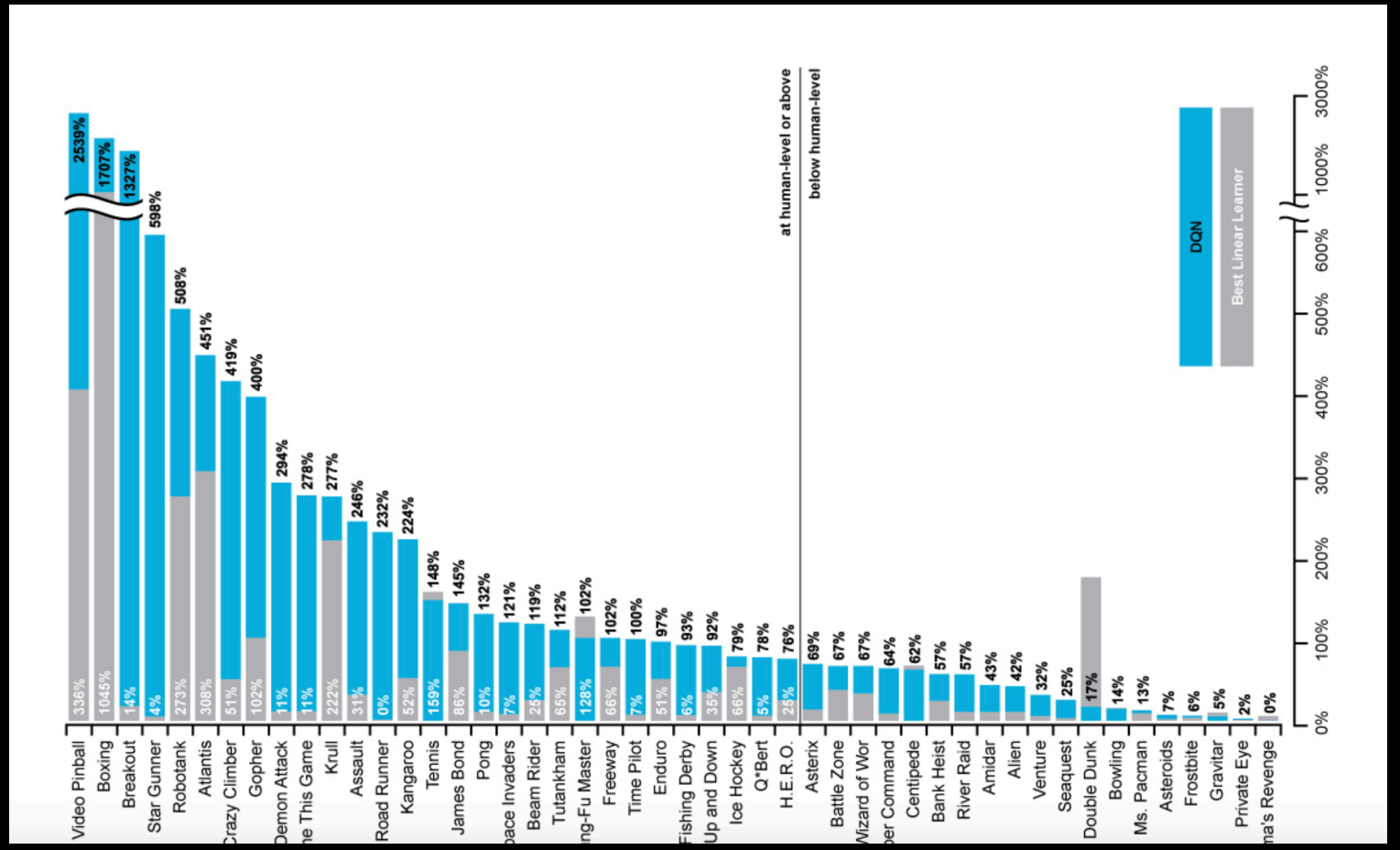
- Represent Q values with weights $q_*(s,a) \approx q(s,a,w)$
- Use discounted returns given by Bellman equation as target for cost function
  - $\mathrm{r}+\gamma \max'_a q(s',a',w)$
- Minimize loss when comparing target to output as current q value
  - With quadratic cost, $C = \left(\mathrm{r} + \gamma \max'_a q(s',a',w) - q(s,a,w)\right)^2$
- Stores history of transitions in “experience replay memory”
- Minibatch is taken from experience replay memory
- $< s, v_{pi} > \sim D \; where \; D \; is \; experience \; replace \; memory$
- $\Delta w = \eta\left(v_\pi - v'(s,w)\right)\nabla_{\mathrm{w}}\mathrm{v}'(\mathrm{s},\mathrm{w})$

Atari Deep Q Networks

- End to end learning of action values from pixel inputs using CNNs
- Input state is stack of raw pixels from last 4 frames
- Output state is q(s,a) for one of the 18 possible joystick positions
- Reward is change in score in time step t



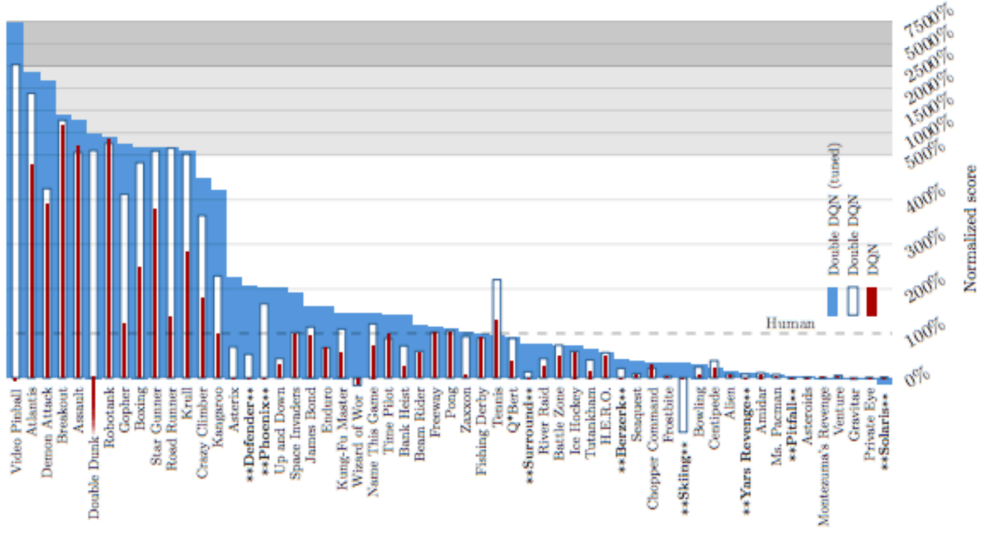
DQN vs non-DQN learners

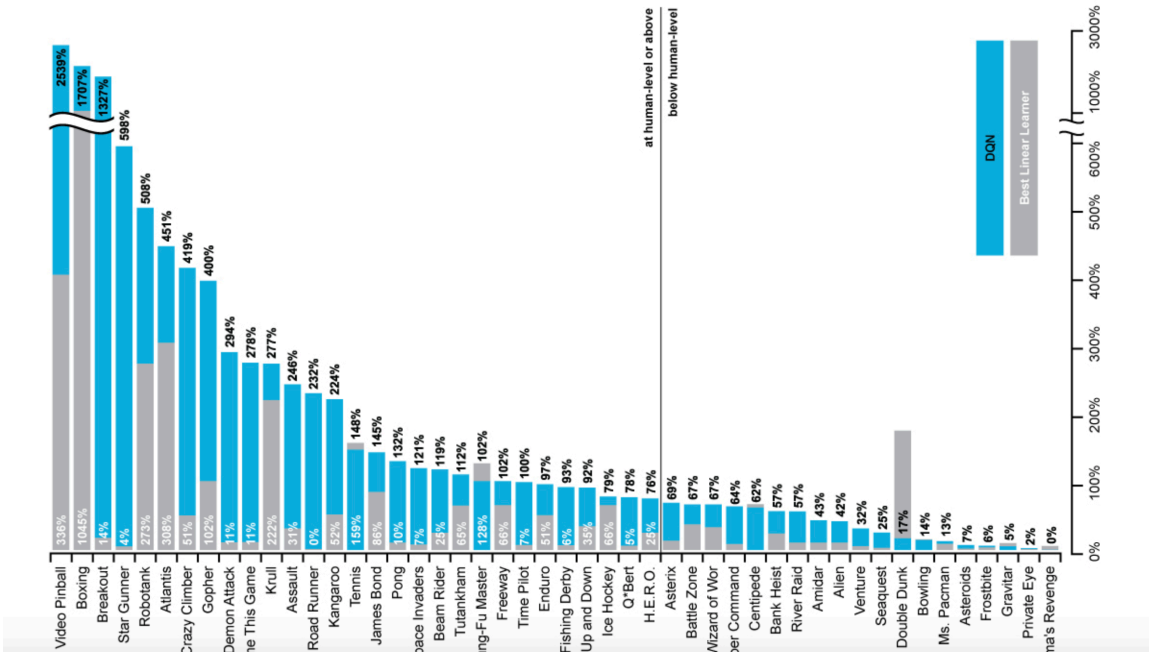


Double DQNs (Hasset et. Al, deep mind)

- Has two action value functions $q_1 and\ q_2$
- Alternate between q-learning on either one, every time step t
  - $q_1 or\ q_2$ is picked with 50% probability
- $q_1(s,a) \leftarrow q_1(s,a) + \eta\left(r + q_2\left(s' + argmax_a q_1(s',a)\right) - q_1(s,a)\right)$
- In each episode chose action a(s) using policy derived from $q_1 and\ q_2$
- Reduces over estimation of values by decomposing max operation to action selection and action evaluation

Double DQN





DDQN performance

Transfer Learning/Neural Episodic Control

- CNN for pixel input (same as DQN)
  - Backprop works through updating weights and biases of CNN
  - Key encoded by CNN to represent change in env
- Memory module per action
  - Neural differentiable dictionary (lookup/write)
  - Lookup maps key h to output value $o = \sum_i w_i v_i$
  - NDD produces an estimate of q(s,a) for a single a
- Network converting action from memory module to q(s,a) values
  - Agent takes the action with the highest q-value estimate at each time step