Reinforcement Learning

Lecture 8:
Function Approximation

Yishay Mansour, Tel-Aviv University

Lecture 8: outline

- Reduction to ML
  - Stochastic Gradient Descent
    - MC
    - TD(0)
    - $TD(\lambda)$
- Linear function approximation
  - On-policy
  - TD(0)
- Off-policy
  - Bad examples
  - Least squares
- Applications
  - DQN
  - TD-Gammon

What is a large MDP

- Large state space:
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Robot: continuous state space
- Large action space:
  - Continuous actions
- Complex dynamics
- Need to scale up !!!

Large scale MDP

- Concentrate of large state space
- Today:
  - Approximate value function
- Next week:
  - Policy optimization

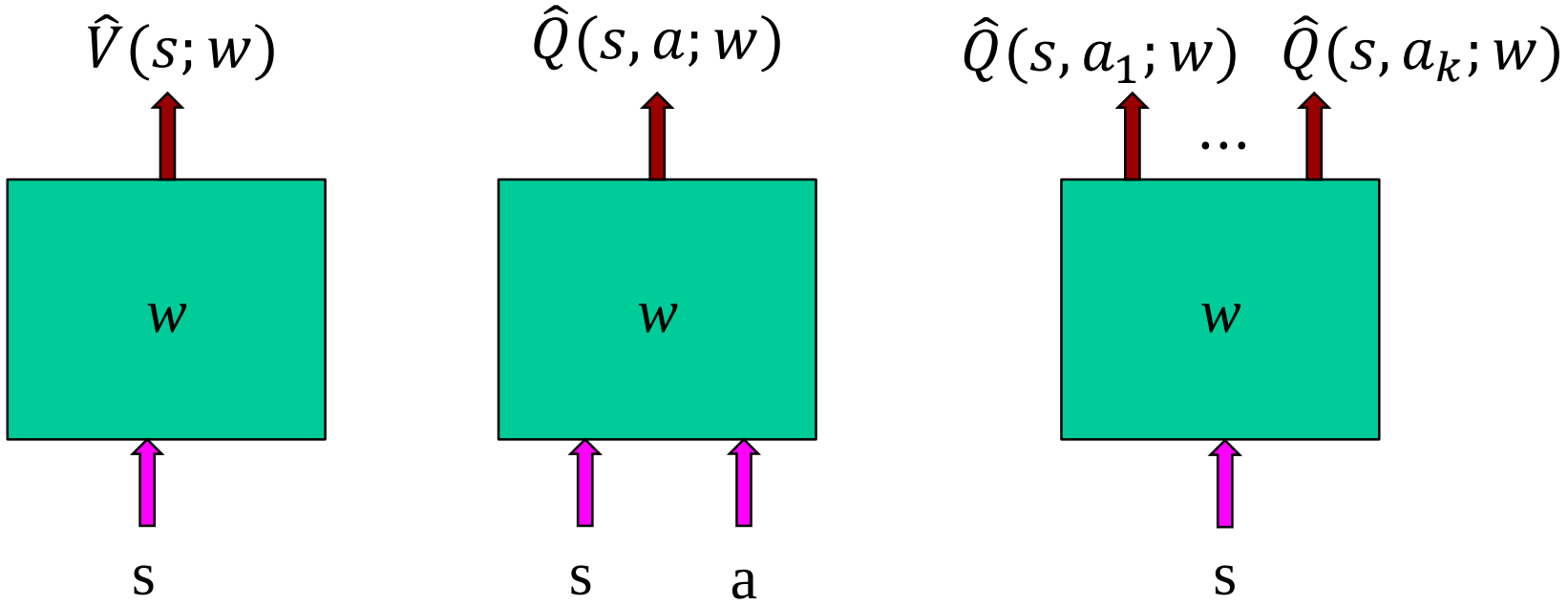
Value function approximation

- Previous lectures:
  - Look-up table: $V(s)$ and $Q(s,a)$
- Problems with large MDP
  - Too many states
    - Almost states never re-appears
    - too slow to learn individually
  - Need (implicit) assumptions

Solutions for large state MDP

- Parametrize value function
  - $\hat{V}(s;w) \approx V^{\pi}(s)$
  - $\hat{Q}(s,a;w) \approx Q^{\pi}(s,a)$
- Generalization
  - From observed states to unseen states
  - Update parameters w
    - Using MC or TD learning

Types of value functions



Which function Approximation

- Similar to Machine Learning:
  - Linear functions
  - Neural Networks
  - Decision Trees
  - Nearest neighbors
  - Fourier/wavelet basis
  - …

Which function Approximation

- Similar to Machine Learning:
  - **Linear functions** ⟸
  - **Neural Networks** ⟸
  - Decision Trees
  - Nearest neighbors
  - Fourier/wavelet basis
  - …

Basic Approximation

- Let V be near optimal
  - $\|V - V^*\|_\infty \leq \epsilon$
- Let π be a greedy policy w.r.t. V
  - $\pi(s) = argmax_a\{r(s,a) + \gamma E[V(s')]$
- Theorem:
  - $\|V^\pi - V^*\|_\infty \leq \frac{2\gamma\epsilon}{1-\gamma}$

Basic Approximation

- Proof:
  - $(H^{\pi}v)(s) = r(s,\pi(s)) + \gamma E^{\pi}[v(s')]$
  - $(H^{*}v)(s) = \max_{a} r(s,a) + \gamma E[v(s')]$
  - Since π greedy w.r.t. V:
    - $H^{\pi}V = H^{*}V$
      - But not for any V !

- Then,
  - $\|V^{\pi} - V^{*}\| = \|H^{\pi}V^{\pi} - V^{*}\|$
  - $\leq \|H^{\pi}V^{\pi} - H^{\pi}V\| + \|H^{\pi}V - V^{*}\|$
  - $\leq \gamma\|V^{\pi} - V\| + \|H^{*}V - H^{*}V^{*}\|$
    - $H^{\pi}V = H^{*}V$, since π greedy w.r.t. V
  - $\leq \gamma\|V^{\pi} - V\| + \gamma\|V - V^{*}\|$
  - $\leq \gamma(\|V^{\pi} - V^{*}\| + \|V^{*} - V\|) + \gamma\|V - V^{*}\|$
- $(1 - \gamma)\|V^{\pi} - V^{*}\| \leq 2\epsilon\gamma$
- QED

Reduction from RL to ML

- Holy grail:
  - Generate a sample
    - Using states, actions rewards
  - Use ML algorithm
    - Stochastic Gradient Descent
  - Apply the ML algo in RL

Idealized reduction: RL to ML

- Fix a policy π
- Generate examples:
  - $\{(s_1, V^\pi(s_1)), \cdots, (s_m, V^\pi(s_m))\}$
- How to sample states?
  - Need them to be i.i.d.
  - Solution:
    - Sample the steady state of π
    - Sample from a trajectory spaced by mixing time

Idealized reduction: RL to ML

- States: Using π
- Actions (if needed): immediate from π
- Generating the labels?
  - What do we do in ML
    - Get a labeled data set
      - Supervised learning
  - What are the labels? $V^{\pi}(s)$
  - BUT: this is what we want to learn!

Idealized reduction: RL to ML

- Three parts:

- States and actions
  - Inputs
  - Using policy π

- Loss function:
  - Tradeoff between inaccuracies
- Labels
  - Need to “invent” them
    - Not given explicitly!

Gradient Descent in ML

- Differential loss function
  - $J(w)$
- Compute a Gradient:
  - $\nabla_w J(w) = (\frac{\partial}{\partial w_1} J(w), \cdots, \frac{\partial}{\partial w_d} J(w))$
- Adjust w: $\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$
  - α step size
- Converges to local minima

Loss: Mean Square Error

- Loss function:
  - $J(w) = \sum_s \mu(s) \left(V^{\pi}(s) - \hat{V}^{\pi}(s; w) \right)^2$
    - $\mu(s)$ steady state distribution of π
- Gradient descent (for local minima):
  - $\Delta w = -\frac{1}{2} \alpha \nabla_w J(w)$
  - $= \alpha \sum_s \mu(s) \left(V^{\pi}(s) - \hat{V}^{\pi}(s; w) \right) \nabla_{\mathrm{w}} \hat{V}^{\pi}(s; w)$
- SGD: $\Delta w = \alpha \left(V^{\pi}(s) - \hat{V}^{\pi}(s; w) \right) \nabla_{\mathrm{w}} \hat{V}^{\pi}(s; w)$

Building an RL sample

- Assume we sample $s_t \sim \mu(\cdot)$
  - Samples $(s_t, V^\pi(s_t))$
    - How do we get $V^\pi(s_t)$
- Unbiased estimator
  - Define U_t s.t. $E[U_t|s_t] = V^\pi(s_t)$
  - Update $\left[U_t - \hat{V}^\pi(s_t; w_t)\right]\nabla\hat{V}^\pi(s_t; w_t)$
- How to get unbiased estimators?

Monte-Carlo: Gradient Descent

- Monte-Carlo estimate (discounted)
  - $R_T(s_j) = \sum_{t=j}^{T} \gamma^{t-j} r_t$
- Unbiased estimator (first visit):
  - $U_T(s_j) = R_T(s_j)$
  - $E\left[U_T(s_j)\right] = V^{\pi}(s_j)$
- Update:
  - $\Delta w = \alpha\left[R_T(s_j) - \hat{V}(s_j; w)\right]\nabla\hat{V}(s_j; w)$

TD(0): Gradient Descent

- TD(0) estimate
  - $R_t(s_t) = r_t + \gamma \hat{V}(s_{t+1}; w)$
- Biased estimator:
  - $E[r_t + \gamma V^{\pi}(s_{t+1})\ ] = V^{\pi}(s_t)$
  - $E\left[r_t + \gamma \hat{V}\ (s_{t+1}; w)\ \right] \neq V^{\pi}(s_t)$
- Update:
  - $\Delta w = \alpha\left[r_t + \gamma \hat{V}\ (s_{t+1}; w) - \hat{V}(s_t; w)\right]\nabla \hat{V}(s_t; w)$

TD(0): Gradient Descent

- Gradient:
  - $\nabla_{\mathrm{w}} \mathrm{E}\left(r_t + \gamma \hat{V}\left(s_{t+1}; w\right) - \hat{V}\left(s_t; w\right)\right)^2 \neq$
  - $2\left(r_t + \gamma \hat{V}\left(s_{t+1}; w\right) - \hat{V}\left(s_t; w\right)\right) \nabla \hat{V}\left(s_t; w\right)$
- We will call it semi-gradient
  - Note that w also influences $R_t(s_t) = r_t + \gamma \hat{V}(s_{t+1}; w)$

$TD(\lambda)$: Gradient Descent

- Update (Forward):
  - $\Delta w = \alpha\left[R_t^{\lambda} - \hat{V}(s_t; w)\right]\nabla_w \hat{V}(s_t; w)$
- Update (Backward):
  - $e_t = \gamma\lambda e_{t-1} + \nabla_w \hat{V}(s_t; w)$
  - $\Delta_t = r_t + \gamma\hat{V}(s_{t+1}; w) - \hat{V}(s_t; w)$
  - $\Delta w = \alpha\Delta_t e_t$

Encoding states: Features

- How do we encode states?
  - Feature vector
  - $x(s) = (x_1(s), \cdots, x_d(s))$
    - $x(s) \in \mathbb{R}^d$
- Features are problem dependent
  - Location in 3D
  - Pixels in a picture
  - Meaningful attributes

Linear Value Function

- Simplest model
  - Maintain weight vector $w = (w_1, \ldots, w_d)$
  - $\hat{V}(s; w) = w^\top x(s) = \sum_{i=1}^{d} w_i x_i(s)$
- Gradient
  - $\nabla_w \hat{V}(s; w) = x(s)$
- SGD updates:
  - $w_{t+1} = w_t + \alpha\left[U_t - \hat{V}(s_t; w_t)\right] x(s_t)$

Table lookup: a linear function

- We can encode look-up table in linear function
  - $d = |S|$
  - $x_i(s) = I(s = s_i)$
  - $\hat{V}(s; w) = \sum_{i=1}^{d} w_i I(s = s_i)$
  - Note that $\hat{V}(s_i; w) = w_i$

Linear model : MC and TD(0)

- MC:
  - $\Delta w = \alpha\left[R_t - \hat{V}(s_t; w_t)\right]x(s_t)$

- TD(0)
  - $\Delta w = \alpha\left[r_t + \gamma\hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t)\right]x(s_t)$

Linear models: $TD(\lambda)$

- $TD(\lambda)$ Forward:
  - $\Delta w = \alpha\left[R_t^\lambda - \hat{V}(s_t; w_t)\right]x(s_t)$
- $TD(\lambda)$ Backward:
  - $\Delta_t = r_t + \gamma\hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t)$
  - $e_t = \gamma\lambda e_{t-1} + x(s_t)$
    - $e_t \in \mathbb{R}^d$
  - $\Delta w = \alpha\Delta_t e_t$

Linear Models: MC convergence

- Loss function:
  - $J(w) = \sum_s \mu(s) \left(V^\pi(s) - \hat{V}^\pi(s; w) \right)^2$
  - Convex: unique minimum
- MC = Stochastic Gradient Descent
  - $\Delta w = \alpha \left[R_t - \hat{V}(s_t; w_t) \right] x(s_t)$
- Convergence:
  - From basic results in optimization

Linear Models: TD convergence

- Convergence
  - Uses semi-gradient
  - Can be proved
    - Requires a proof
    - Depends being on-policy

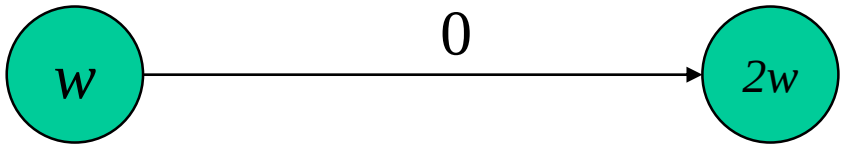
- TD Loss
  - w_{TD}
  - w_{min}
  - $J(w_{TD}) \leq \frac{1}{1-\gamma} J(w_{min})$
- MC Loss
  - Optimal
    - w_{min}

Off-policy Function Approximation

- What happens if we do not select actions?
  - Exists in many settings
- Look-up tables
  - No problems
- Can we extend to off-policy?
  - Samples are generated from some policy
  - Only observe (s_t, a_t, r_t, s_{t+1})

Basic Divergence “example”

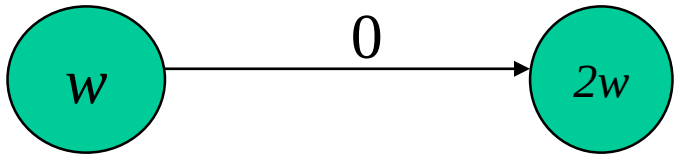
- We want to understand what can go wrong?
  - Assume the following part of an MDP:



  - Assume this is the only place *w* appear
    - Toy example

Basic Divergence "example"

- Assume some $w_0 > 0$ and $\gamma > {}^1/_2$
- TD update:
  - $\Delta_t = 0 + \gamma(2w_t) - w_t = (2\gamma - 1)w_t$
  - $w_{t+1} = w_t + \alpha(2\gamma - 1)w_t = \big(1 + \alpha(2\gamma - 1)\big)w_t$
  - $w_T = \big(1 + \alpha(2\gamma - 1)\big)^{\mathrm{T}} \mathrm{w}_0 \to \infty$



Basic Divergence “example”

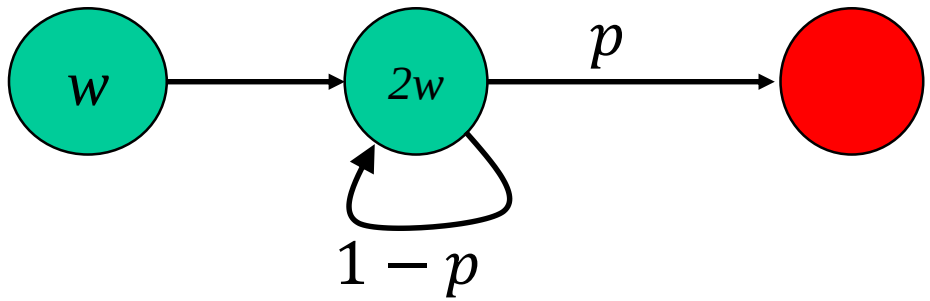
- Can this happen also with “full trajectory”?
  - TD errors should cancel out
    - For on-policy
  - From the second state:
    - Have outgoing edge,
    - Either decrease *w*
      - *Great*
    - Or, yet increase
      - Building up for a larger decrease!

Basic Divergence “example”

- So far: This is more a conceptual problem
  - We did not present:
    - Complete MDP
    - A sampling policy

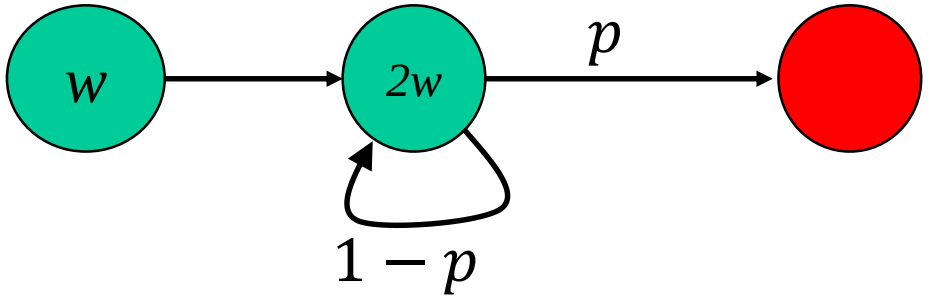
Bad Example

- Three states
- All rewards are zero
- Initially $w_0 > 0$
- Three types of updates:
  - $\alpha[0 + \gamma(2w_t) - w_t] = \alpha(2\gamma - 1)w_t$
  - $\alpha[0 + \gamma(2w_t) - (2w_t)] = -2\alpha(1 - \gamma)w_t$
  - $\alpha[0 + \gamma 0 - (2w_t)] = -2\alpha w_t$



Bad Example

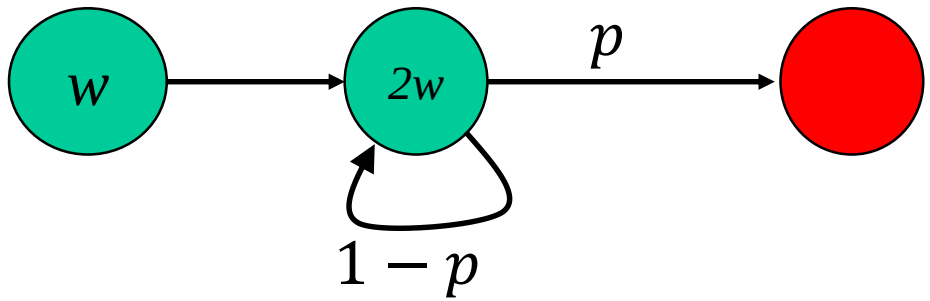
- Full trajectory
  - Observe all three states
    - n time the second state



- Net update:
  - $\frac{w_{t+1}}{w_t} =$
  - $\left(1+\alpha(2\gamma-1)\right)\left(1-2\alpha(1-\gamma)\right)^{n}(1-2\alpha)$
  - $\leq \left(1+\alpha(2\gamma-1)\right)(1-2\alpha) < 1-\alpha$

Bad Example

- Partial trajectory:
  - Observe prefix n



- Most updates:
  - $\frac{w_{t+1}}{w_t} = \left(1 + \alpha(2\gamma - 1)\right)\left(1 - 2\alpha(1-\gamma)\right)^n > 1$
    - For $n \ll \frac{1}{p}, \frac{1}{1-\gamma}$
    - $w_t \Rightarrow \infty$

DP with least squares

- Is the problem online updates?
  - Consider minimizing squared error

- Algorithm:
  - Given w_t compute:
  - $w_{t+1} =$
  - $argmin_w \sum_s [V(s;w) - E^{\pi}[r_t + \gamma V(s_{t+1};w_t)]]^2$

Back to the bad example

- $w_{t+1} =$
- $argmin_w \sum_s [V(s;w) - E^{\pi}[r_t + \gamma V(s_{t+1}; w_t)]]^2$
- $argmin_w \left(w - \gamma(2w_t)\right)^2 + \left(2w - (1-p)\gamma(2w_t)\right)^2$
  - Take derivative
- $2(w - 2\gamma w_t) + 4(2w - 2\gamma(1-p)w_t) = 0$
- $10w_{t+1} = 4\gamma w_t + 8\gamma(1-p)w_t$
- $w_{t+1} = \frac{6\gamma - 4p}{5} w_t$
  - $w_t \Rightarrow \infty$ for $\gamma > \frac{5}{6} + 4p$

The Deadly Triad

- Function approximation
  - Update influences multiple states
- Bootstrapping
  - Estimate depends on “our” value function
- Off-policy training
  - Full versus partial trajectory

Convergence: Value Function Approximation

	Algorithm	Table look-up	Linear model	Non-linear
On-policy	MC	✚	✚	✚
	$TD(0)$	✚	✚	✖
	$TD(\lambda)$	✚	✚	✖
Off-policy	MC	✚	✚	✚
	$TD(0)$	✚	✖	✖
	$TD(\lambda)$	✚	✖	✖

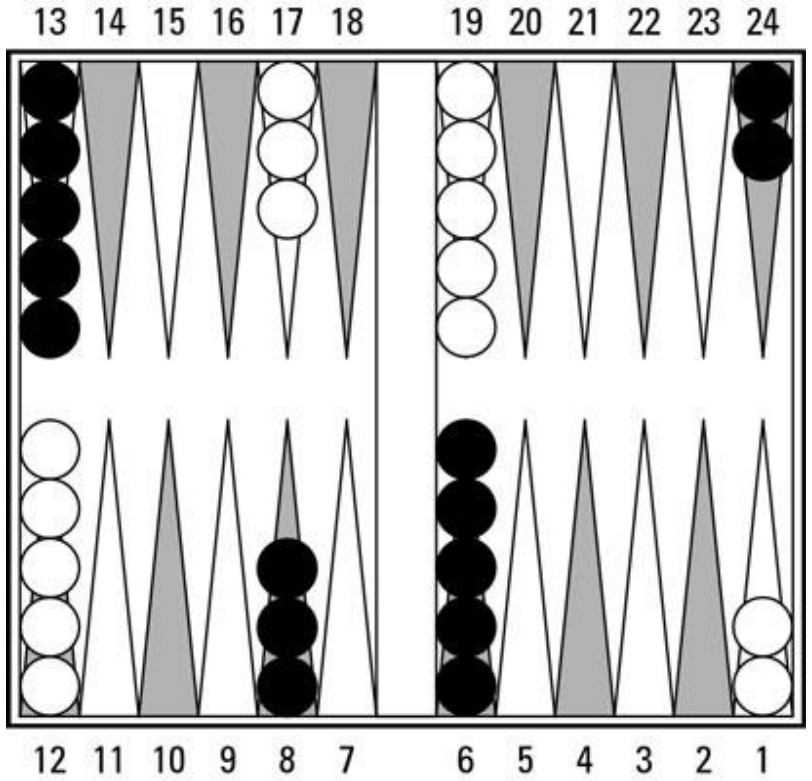
✚ Guaranteed convergence

✖ Possible divergence

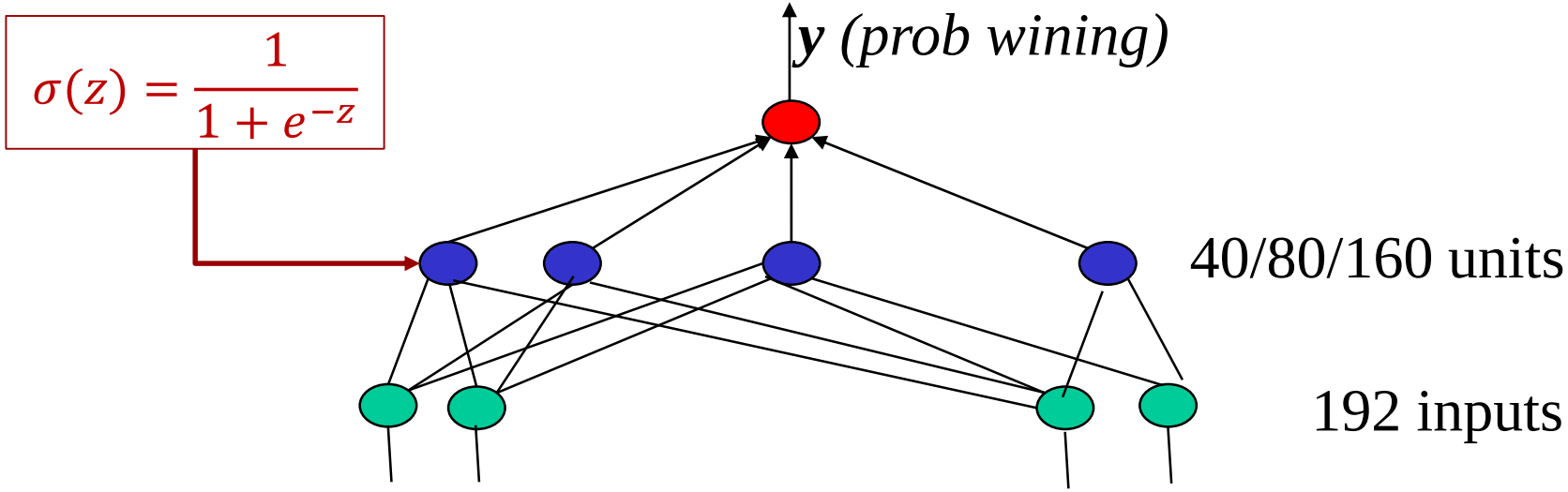
Applications: Games

Tesauro: TD-gammon

- Time 1989-1997
- Function Approx
  - Neural Network
    - One hidden layer
  - Updates using TD
- Training: self-play
- Performance:
  - Best program
  - Eventually, best human



TD-gammon: Neural Network



TD-gammon: Inputs

- Encode board:
- Per slot (24) color (2):
  - 4 Boolean inputs
  - Encode (unary) number of chips
    - (1,2+,3,4+)
- Move:
  - White/Black (2 inputs)

- Per color:
  - (Num pieces)/2
  - (Num pieces out)/15

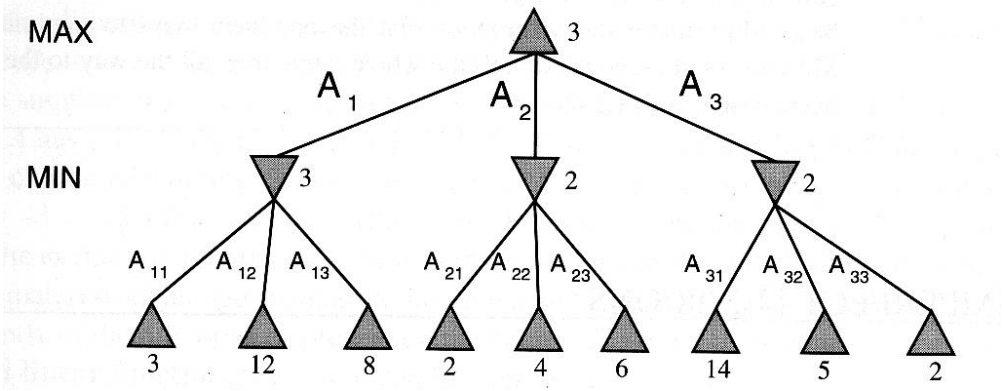
- Total: 198 inputs

TD-gammon: updates

- Using $TD(\lambda)$
- During the game:
  - $\Delta w_t = \alpha(y_t - y_{t-1}) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w y_k$
- At the end of the game
  - Replace y_t by z the outcome.
- Game using random policies ends!
  - After few hundreds or thousand moves

Evaluation

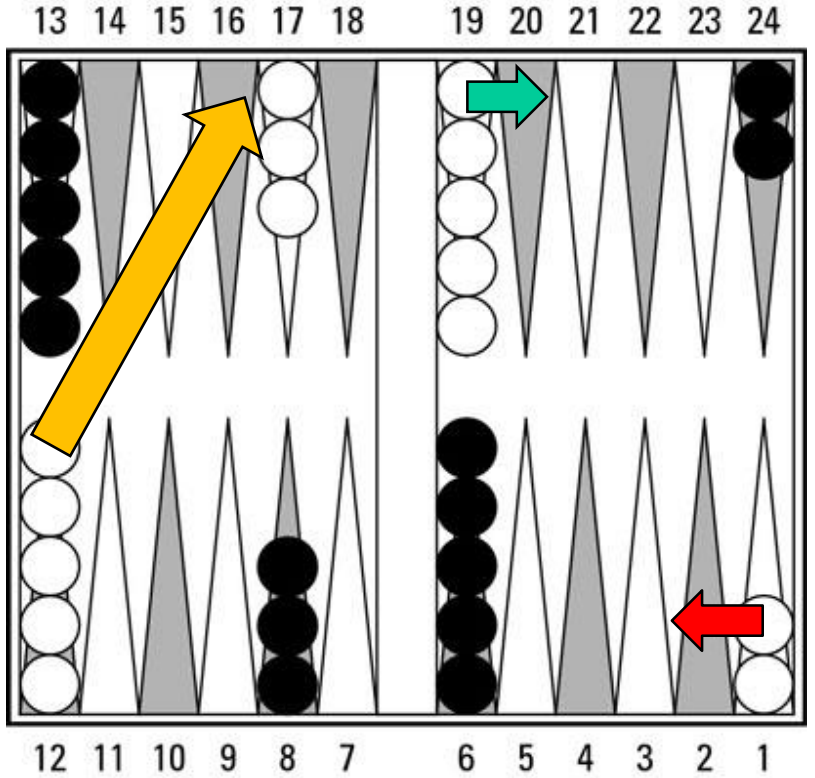
- Given a board position
- Generate few move
  - 1/2/3
- Evaluate end positions
  - 20/400/8000
- Estimate value
  - Use min-max tree



Performance

version	Hidden layer	Games	Num. ply
0.0	40	300K	1
1.0	80	300K	1
2.0	40	800K	2
2.1	80	1.5M	2
3.0	80	1.5M	3

Performance: best human
Changed human play!



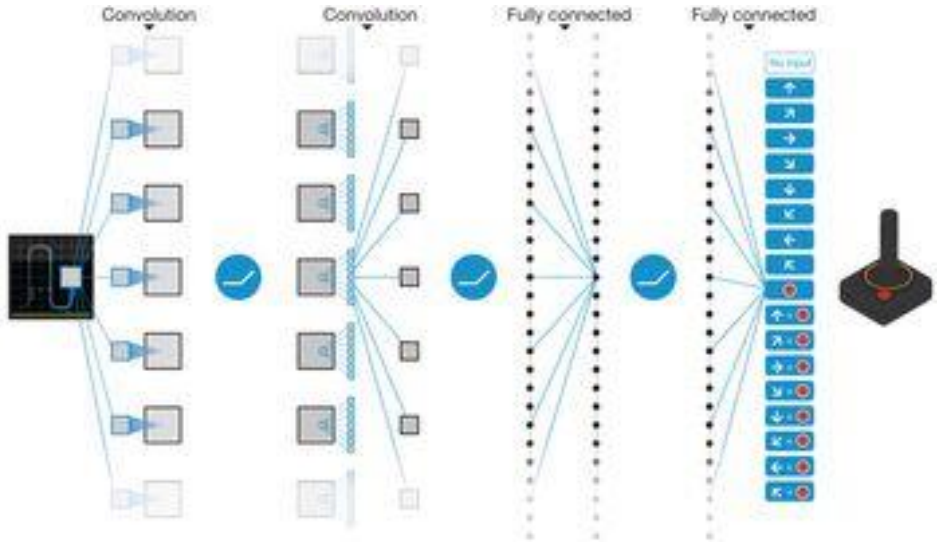
DeepMind: Attari Games

- Time: 2013-2015
- Learn to play Attari games
  - 49 games
  - Reach human level
  - Uses Q-learning
    - DQN

DeepMind: Neural Network

- Input preprocessing
  - 210x160 128-color
    - Map to 84x84 4-color
    - Identical to all games
  - Group 4 frames

- Network:
  - 3 convolutional
  - 1 complete connected
  - Multiple outputs
  - RelU gates



DQN: flow

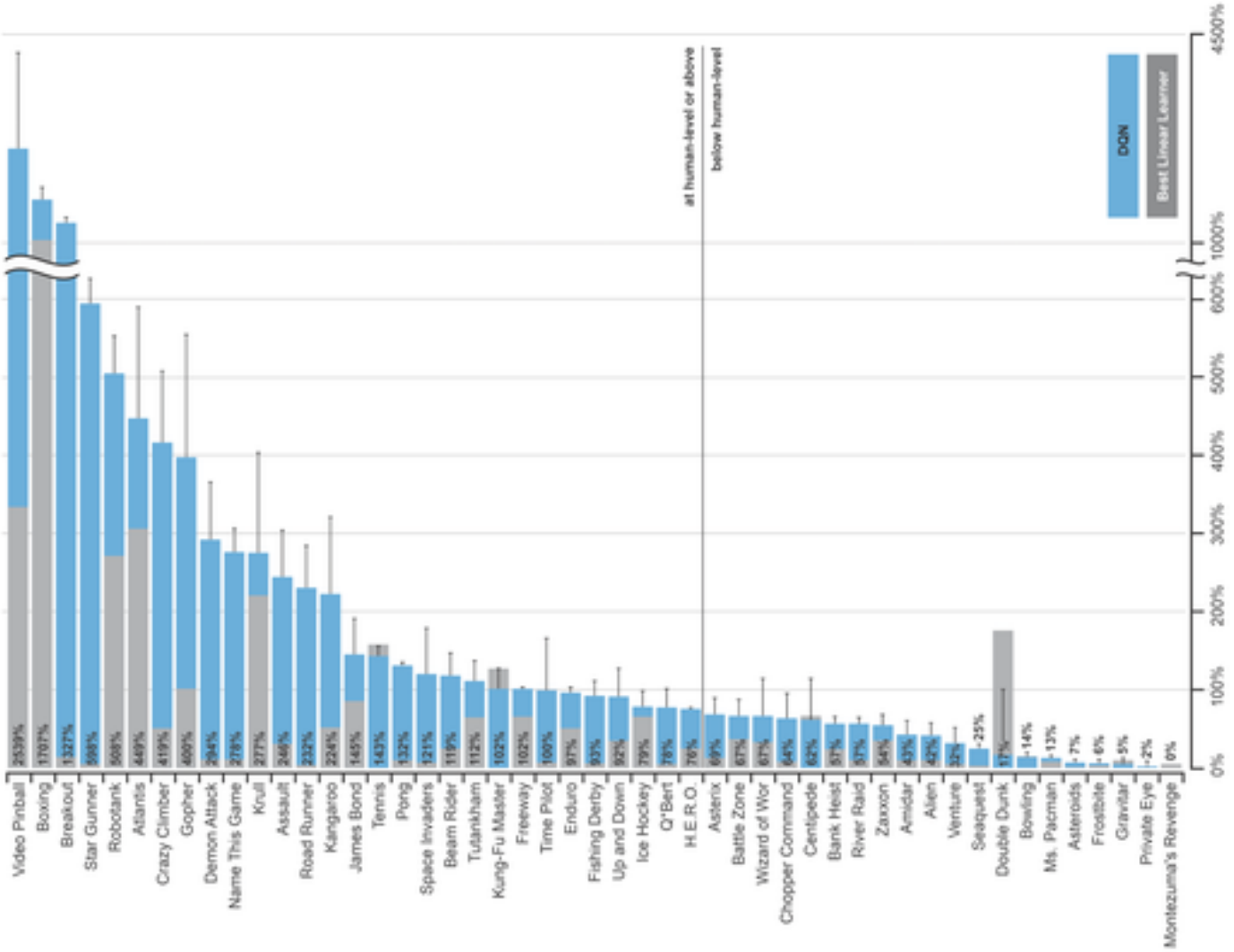
- DQN innovations:
  - Experience replay
    - Replay $e_t = (s_t, a_t, r_t, s_{t+1})$
  - Fixed Q-target
    - When doing the update
  - Clipping Errors

- Flow:
  - Do a_t (ϵ-greedy)
  - Store (s_t, a_t, r_t, s_{t+1})
    - In replay memory
  - Sample mini-batch
    - From replay memory
  - Compute Q-target
    - w.r.t. “old weights”
  - Minimize MSE
    - SGD

DQN: fixed-Q target

- Two sets of weights w_t and w_t^-
- Update (each time)
  - $\Gamma_t = r_t + \gamma \max_a Q\left(s_{t+1}, a; w_t^-\right) - Q(s_t, a_t; w_t)$
  - $\Delta w_t = \alpha \Gamma_t \nabla_{w_t} Q(s_t, a_t; w_t)$
- Update (every long time)
  - $w_t^- = w_t$
- Clipping: $\Gamma_t \in [-1, +1]$

DQN: performance



Lecture 8: outline

- Reduction to ML
  - Stochastic Gradient Descent
    - MC
    - TD(0)
    - $TD(\lambda)$
- Linear function approx
  - On-policy
  - TD(0)
- Off-policy
  - Bad examples
  - Least squares
- Applications
  - DQN
  - TD-Gammon