Once again $S_0, A_0, R_1, S_1, A_1, R_2, \ldots$

Recall:

TD(0):

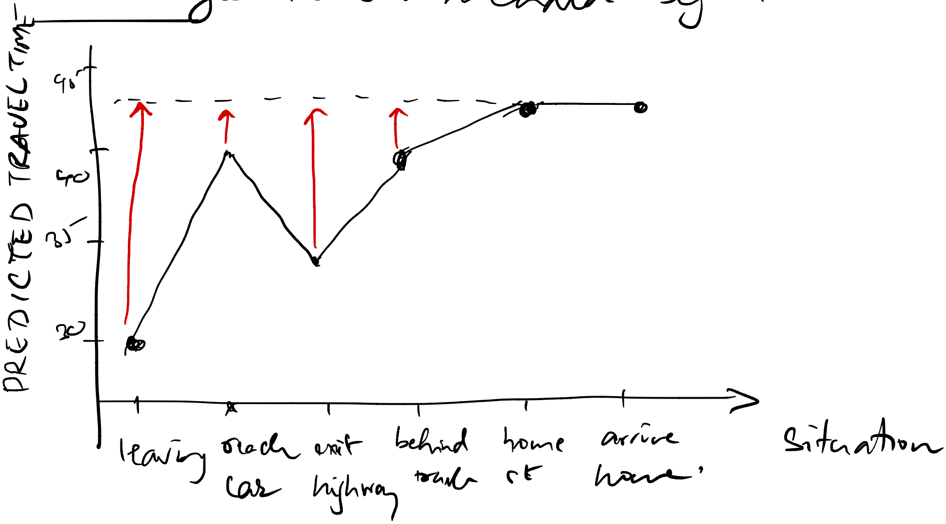
$$V(S_t) \leftarrow V(S_t) + \alpha\big(\underbrace{R_{t+1} + \gamma V(S_{t+1}) - V(S_t)}_{\text{TD error}}\big)$$

Example:

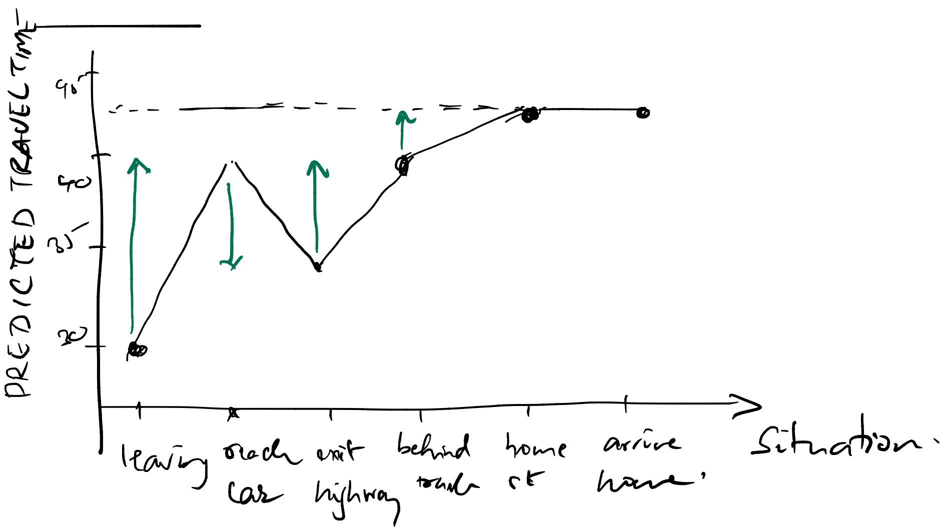
Driving Home example: (Barto & Sutton, David Silver)

State	Elapsed time	Predicted time to go	Predicted Total time
leaving office	0	30	30
reach car, rain	5	35	40
exit highway	20	15	35
behind truck	30	10	40
Home street	40	3	43
arrive home	43	0	43

Changes recommended by MC.



Changes Recommended by TD:



Batch MC & TD:

MC & TD converge; $V(s) \to v_\pi(s)$ as experience $\to \infty$;

- Batch solution for finite experience:

$$\left.\begin{array}{l} s_1^1, a_1^1, r_2^1, \ldots s_{T_1}^1 \\ \vdots \\ s_1^K\ a_1^K\ r_2^K \ldots s_{T_K}^K \end{array}\right\}$$

- repeatedly sample episode $k \in [1, K]$
- Apply MC or TD(0) to episode k.

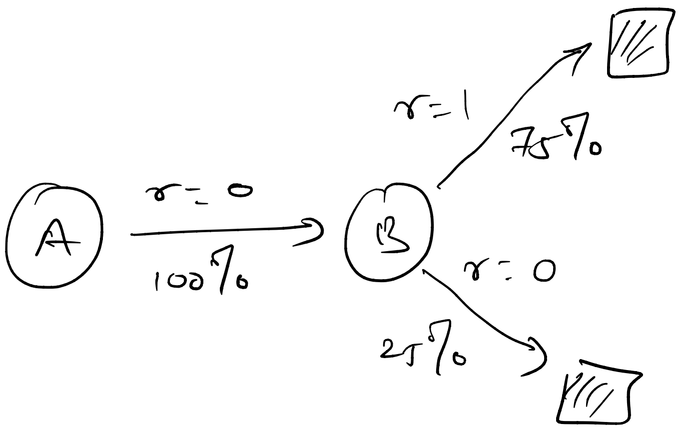
What do MC & TD converge to?

Ex: Two states; no discounting; 8 episodes of experience;

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

MC: $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$

TD: $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$



MC converges to solution with min mean-sq error.

Best fit to observed returns,

$$\sum_{k=1}^{K} \sum_{t=1}^{T_k} \left(G_t^k - V(s_t^k)\right)^2$$

for the example $V(A) = 0$;

TD(0) converges to a solution of max likelihood Markov model;

$\langle S, A, \hat{P}, \hat{R}, \gamma \rangle$ which best fits the data.

$$\hat{P}^{a}_{s,s'} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbb{1}\left(s_t^k, a_t^k, s_{t+1}^k\right)$$

with $s_t^k = s$, $a_t^k = a$, $s_{t+1}^k = s'$.

$$\hat{R}^{a}_{s} = \frac{1}{N(s,a)} \sum_{k=1}^{K} \sum_{t=1}^{T_k} \mathbb{1}\left(s_t^k, a_t^k\right) r_t^k.$$

with $s_t^k = s$, $a_t^k = a$.

for the example, $V(A) = 0.75$;

ADV of TD — exploits Markov property.
more effective in Markov env.

ADV of MC: Does not exploit Markov property.
effective in non Markov env.

Last time:

TD(n):

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step TD learning:

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^n - V(S_t)\right).$$

Now TD(λ):

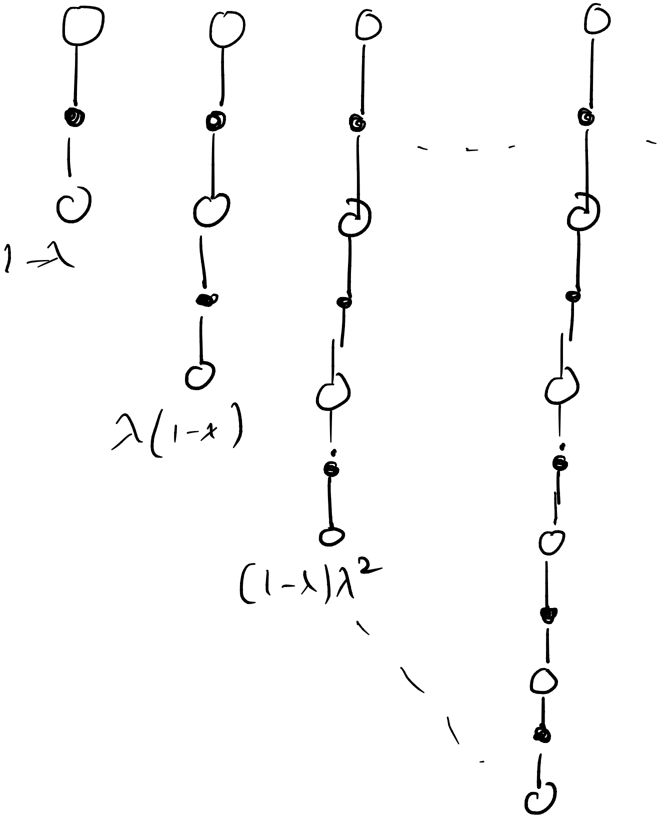
- Average n-step returns over different n;

Ex: $\frac{1}{2} G^{(2)} + \frac{1}{2} G^{(4)}$

- Can we combine such information from all time steps?

-

TD(λ): (From Silver)



$$G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{n}$$

λ return used for update.

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{\lambda} - V(S_t)\right)$$

- Needs entire episode.

Eligibility traces:

- Which states should get credit for current return?

Frequency Heuristic

Assign credit to frequent states

Recency.

Assign credit to recent states;

Eligibility trace: Combines these heuristics;

$$E_0(s) = 0$$
$$E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbb{1}(S_t = s)$$

- Idea: Keep an eligibility trace for all states;

Update $V(s)$ $\forall s$, in proportion to δ_t, and $E_t(s)$

Backward view TD(λ).

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s).$$

- Last time - we saw function approximation.

unavoidable when we want to solve large problems;

Backgammon - 10^{20} states;
Go 10^{170} states.

- idea: $\hat{v}(s,w) \simeq v_\pi(s)$; value function Approximation

$\hat{q}(s,a,w) \simeq q_\pi(s,a)$ .. Q value approximation

Example: Represent a state by a feature vector

$$\begin{pmatrix} x_1(s) \\ \vdots \\ x_n(s) \end{pmatrix}$$

A linear approximation of value function is

$$\hat{v}(s,w) = \sum_{i=1}^{n} w_i \, x_i(s)$$

• Natural objective function to minimize

$$\mathcal{L}(\pi, w) \triangleq \mathbb{E}_{\pi}\left(\left(V_{\pi}(s) - x(s)^T w\right)^2\right)$$

$=$

we don't know this unlike in supervised learning!

<u>Idea:</u> Substitute a target for this;

$$\nabla_w \mathcal{L} = \mathbb{E}_{\pi}\, 2\left(\left(V_{\pi}(s) - x(s)^T w\right) \cdot x(s)\right)$$

$$\therefore\ w \leftarrow w - \alpha\left(v_{\pi}(s) - x(s)^T w\right) x(s).$$

Stochastic gradient using one sample;

<u>In MC:</u> $V_{\pi}(s)$ replaced by G_t;

<u>In TD0:</u> target replaced by $R_{t+1} + \gamma \hat{V}(S_{t+1}, w)$

<u>update:</u>

$$w \leftarrow w - \alpha\left(R_{t+1} + \gamma \hat{V}(S_{t+1}, w) - \hat{V}(S_t, w)\right) x(s)$$

Monte-Carlo + function approximation:

- G_t - is an unbiased estimator of $v_\pi(S_t)$;

- $\langle S_1, G_1 \rangle \langle S_2, G_2 \rangle \cdots \langle S_T, G_T \rangle$ - thought of as supervised training data;

$$\Delta w \leftarrow \alpha \left(G_t - \hat{v}(S_t, w)\right) \nabla_w \hat{v}(S_t, w)$$

works even if this is not a linear fn of w.

- MC - converges to a local opt
- MC " even if the function approximator is non linear;

- Can do TD learning with function value approximation

TD target $R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$ current w

Biased sample of $v_\pi(S_t)$.

- Can still do supervised learning as before

$\langle S_1, R_1 + \gamma \hat{v}(S_2, w) \rangle, \langle S_2, R_3 + \gamma \hat{v}(S_3, w) \rangle, \ldots$

Apply supervised learning to above;

$TD(0)$: $\Delta w = \alpha \left(R + \gamma \hat{v}(S', w) - \hat{v}(S, w) \right) \nabla_w \hat{v}(S, w)$.

$w \leftarrow w - \alpha \Delta w = w - \alpha \delta x(S)$, in the linear case;

<u>THM</u>: Linear $TD(0)$ converges to global optimum;

Can do the same with TD^λ

$$\Delta w = \alpha \left(G_t^\lambda - \hat{v}(S_t, w) \right) \nabla_w \hat{v}(S_t, w)$$

- In backward linear $TD(\lambda)$:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)$$
$$E_t = \gamma \lambda E_{t-1} + x(S_t)$$

vector.

a vector;

$\Delta w = \alpha \, \delta_t E_t$;

Last time: Compatible function approximator for state-value function.

- In general: $\hat{q}(S, A, w) \approx q_\pi(S, A)$.

- Minimize least square error;

- Examples: $x(S, A) = \begin{pmatrix} x_1(S, A) \\ \vdots \\ x_n(S, A) \end{pmatrix}$

action-value a linear function of these features,

- Use stochastic gradient as before;
- Targets for gradient updates: Like before;

G_t, $R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w)$, δ_t ....

- The previous algorithms - a single sample used as an unbiased estimator.

BATCH REINFORCEMENT LEARNING:

- Given value function approximation $\hat{v}(s,w) \approx v_\pi(s)$ and experience

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

GOAL: Find parameters w which give the best fitting value function $\hat{v}(s,w)$.

LEAST SQUARE:

$$LS(w) = \sum_{t=1}^{T} \left(v_t^\pi - \hat{v}(s_t, w)\right)^2$$

Algorithm:

- Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

- Apply SGD update

$$\Delta w = \alpha \left(v^\pi - \hat{v}(s,w)\right) \nabla_w \hat{v}(s,w)$$

- Converges to $w^{\pi} = \underset{w}{\arg\min}\, LS(w)$.

<u>Experience replay in DQN:</u>

- Take action a_t according to ϵ-greedy.
- Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$ - replay memory
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t old $\bar{w}$
- OPTIMIZE MSE b/w Q-network & Q-learning target.

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

Use stoch gradient descent.

DQN: Nature - 2015; Human level control through deep RL

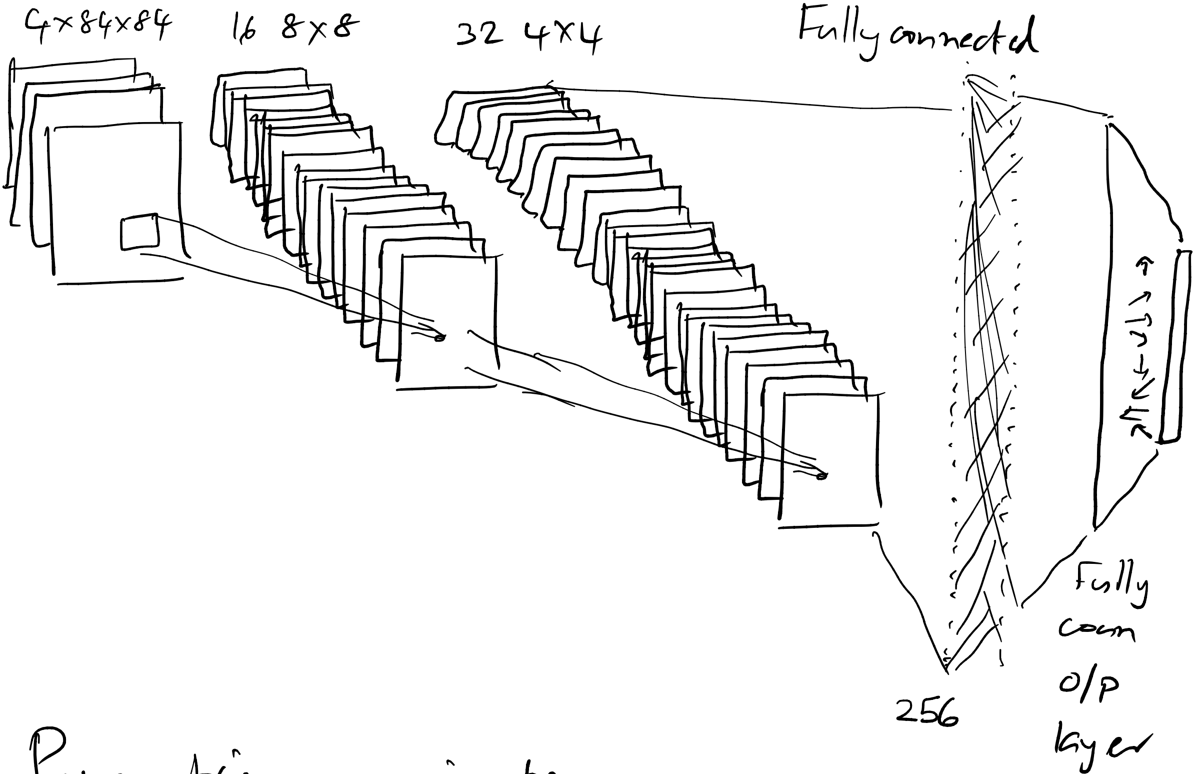
Sets up a DQN to approximate the optimal action-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[r_t + \gamma r_{t+1} + \cdots \,\middle|\, s_t = s, a_t = a, \pi\right]$$

* Experience replay - biologically inspired. Since it randomizes over the data it removes correlations in the observation sequences.

* Uses an iterative update, adjusting action-value towards target values which are only periodically updated and so reducing correlations with the target.

• End to end learning of values $Q(s,a)$ from pixels



Parametrize approximate action value function $Q(s, a, \theta_i)$

θ_i – are weights of the DQNN.

- Perform experience replay by storing agents experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t in $D_t = \{e_1, \ldots, e_t\}$.

Q-learning update at iteration i uses loss fn

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

Network parameters used to compute the target at iteration i.

Parameters of the network in iteration i

- Target network parameters θ_i^- are updated with the Q-network parameters θ_i every C steps and are held fixed b/w individual updates.

Algorithm:

- At each time step t, agent selects a_t, a legal game action and passes it to the emulator.

- Emulator modifies its internal state & game score.

- The agent observes an image from the emulator, the vector of pixel values representing current screen. Agent receives reward r_t, the change in game score.