DEEP REINFORCEMENT LEARNING

Oliver Schulte

Simon Fraser University

CMPT 728

Introduction to Deep Learning

OUTLINE

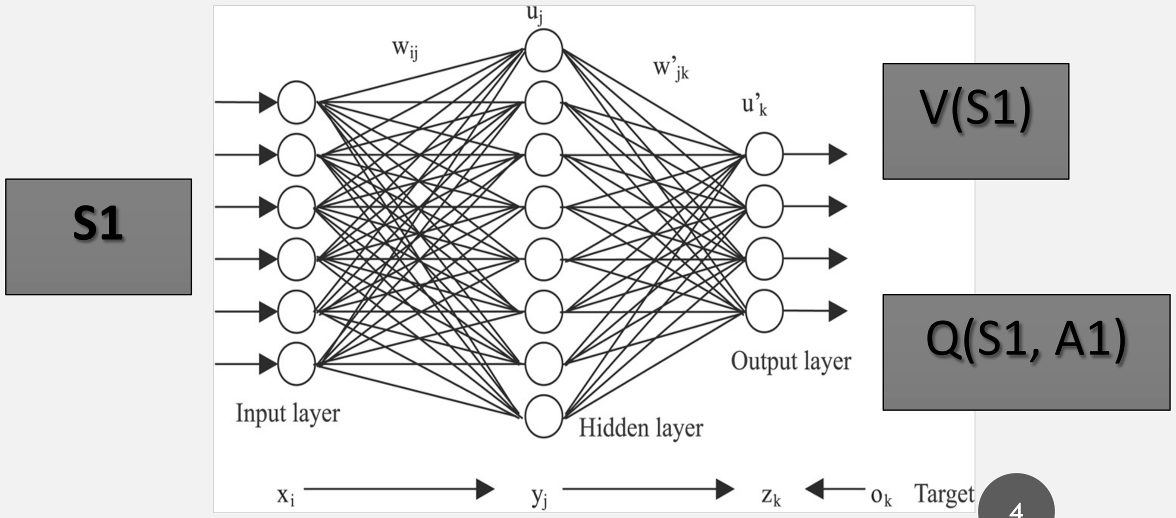
- Reinforcement Learning as Function Learning
- Deep RL using Episodes
  - Monte Carlo Learning
  - Temporal Difference Learning
  - Policy Gradient
  - Actor-Critic

REINFORCEMENT LEARNING AS FUNCTION LEARNING

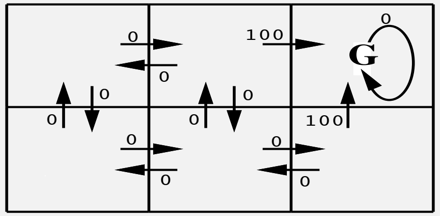
STATE-BASED FUNCTIONS

- RL goal is to learn functions that map states to outputs
- Tabular/grid representation: 1 row per state
- <u>Factored representation</u>: state = list of features
- Neural net: #of features = #input nodes

State	Action	V(s)	Q(s, a)
S1	A1	0.345	0.012

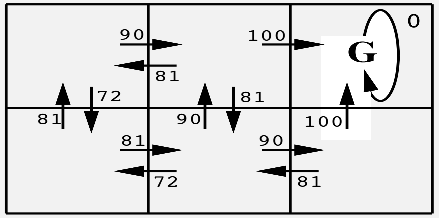


REVIEW: RL CONCEPTS

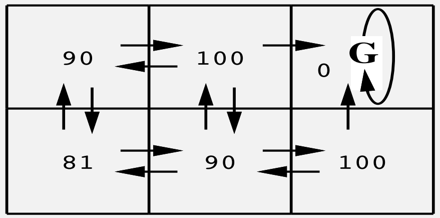


$r(s, a)$ (immediate reward) values

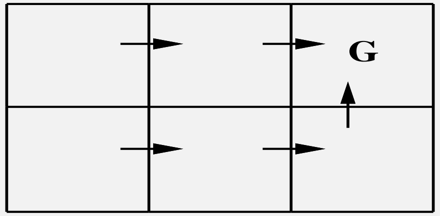
These 3 functions can be computed by neural networks



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

WHEN TO USE FUNCTION APPROXIMATORS?

- State space is too large to handle by tabular representation
  - E.g. #possible chess positions > #atoms in the universe
- Curse of dimensionality: linearly more features→ exponentially more states
- States/actions are continuous
  - E.g. location, time in sports
  - Choose location, speed in video game
  - See puckworld demo

EXAMPLES

TOY EXAMPLE: CART POLE

- Open AI virtual environment
- State = 4 numbers
- (pos cart t-1, angle t-1, pos cart t, angle t)
- Move Left or Right
- Reward = 1 for move that doesn't topple the cart

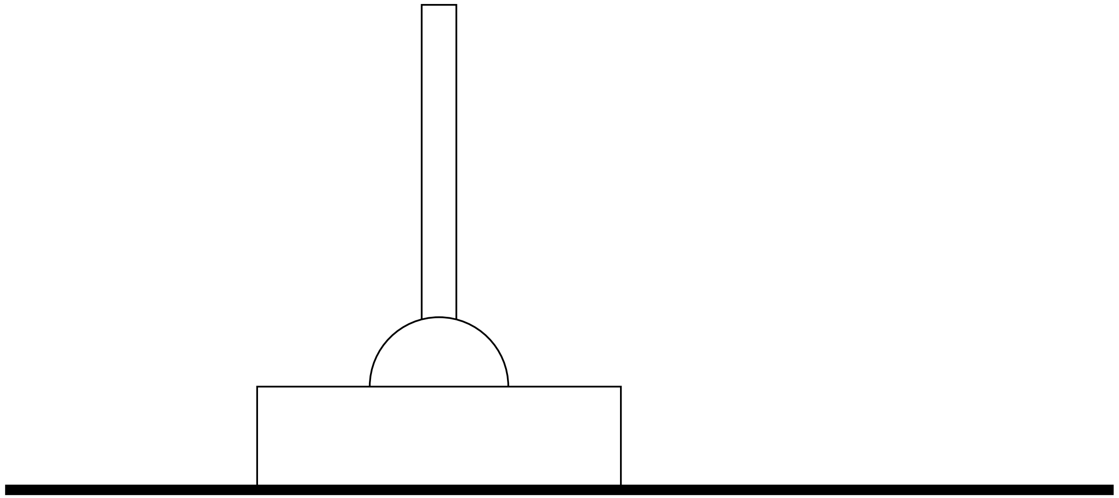


Figure 6.8: A cart pole

GAMES

- Board Games: input position of pieces
  - TD-Gammon for backgammon
  - Also chess, checkers
- Video Games: input (sequence of) video frames
  - Use CNN, maybe combined with LSTM
  - Starcraft

EXAMPLE: ALPHA* GAMES

- data generated by self-play
- Neural net outputs 2 quantities
  1. V(s), the win rate from a position
  2. P(a|s): vector of move probabilities
     - more promising moves should have higher probability
     - Like node ordering in tree search
- To play, performs a (Monte Carlo) tree search using the neural net output
- Watch the alphago movie

Table 1: Dataset Example

GID	PID	GT	TID	X	Y	MP	GD	Action	OC	P
1365	126	14.3	6	-11.0	25.5	Even	0	Lpr	S	A
1365	126	17.5	6	-23.5	-36.5	Even	0	Carry	S	A
1365	270	17.8	23	14.5	35.5	Even	0	Block	S	A
1365	126	17.8	6	-18.5	-37.0	Even	0	Pass	F	A
1365	609	19.3	23	-28.0	25.5	Even	0	Lpr	S	H
1365	609	19.3	23	-28.0	25.5	Even	0	Pass	S	H

Table 2: Derived Features

Velocity	TR	D	Angle	H/A	PN
(-23.4, 1.5)	3585.7	3.4	0.250	A	4
(-4.0, -3.5)	3582.5	3.1	0.314	A	4
(-27.0, -3.0)	3582.2	0.3	0.445	H	4
(0, 0)	3582.2	0.0	0.331	A	4
(-30.3, -7.5)	3580.6	1.5	0.214	H	5
(0,0)	3580.6	0.0	0.214	H	5

GID=GameId, PID=playerId, GT=GameTime, TID=TeamId, MP=Manpower, GD=Goal Difference, OC = Outcome, S=Succeed, F=Fail, P = Team Possess puck, H=Home, A=Away, H/A=Team who performs action, TR = Time Remain, PN = Play Number, D = Duration

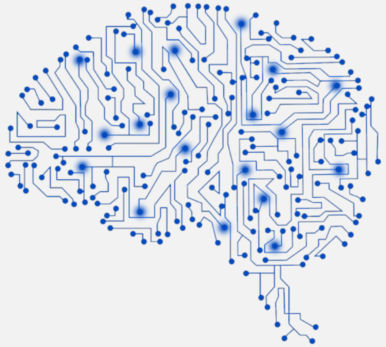
ICE HOCKEY EXAMPLES

- I’ve done a lot of work applying RL to ice hockey. Youtube talk.
- Using millions of events from NHL games

PIPELINE

- Computer Vision Techniques: Video tracking

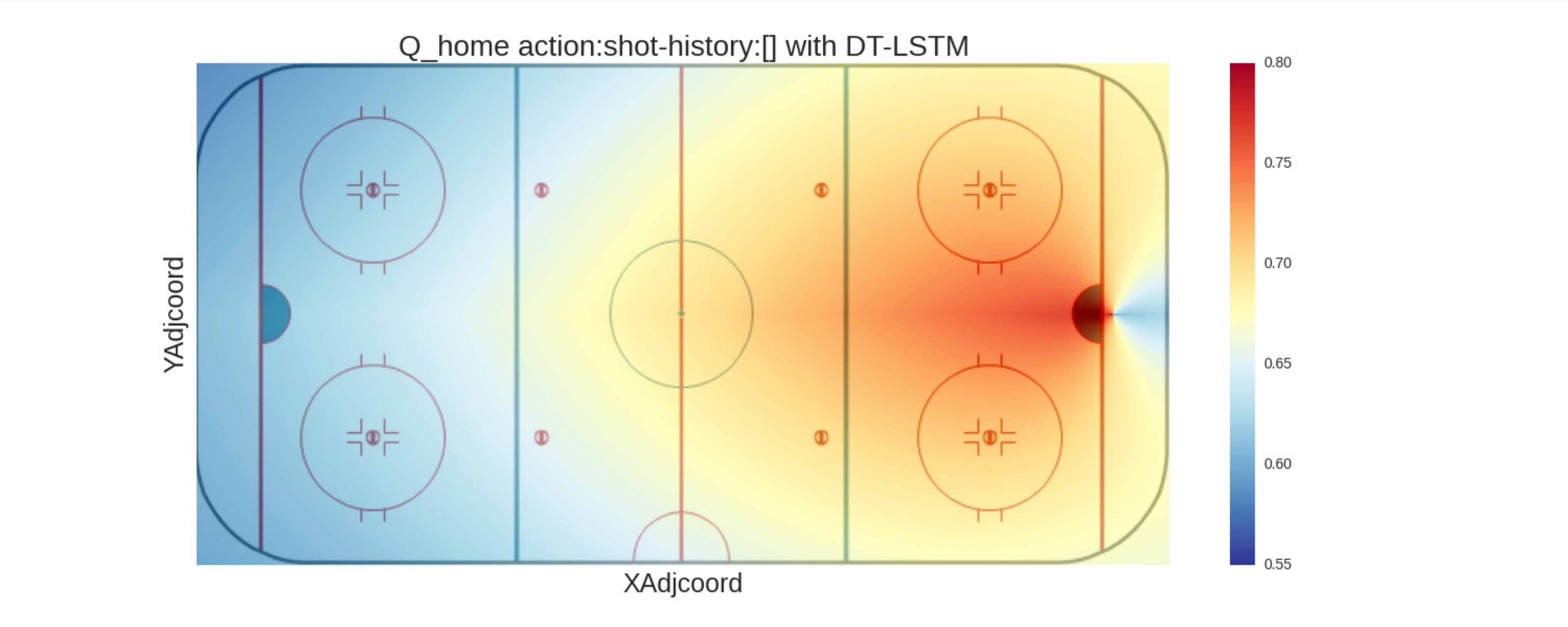
- Play-by-play Dataset



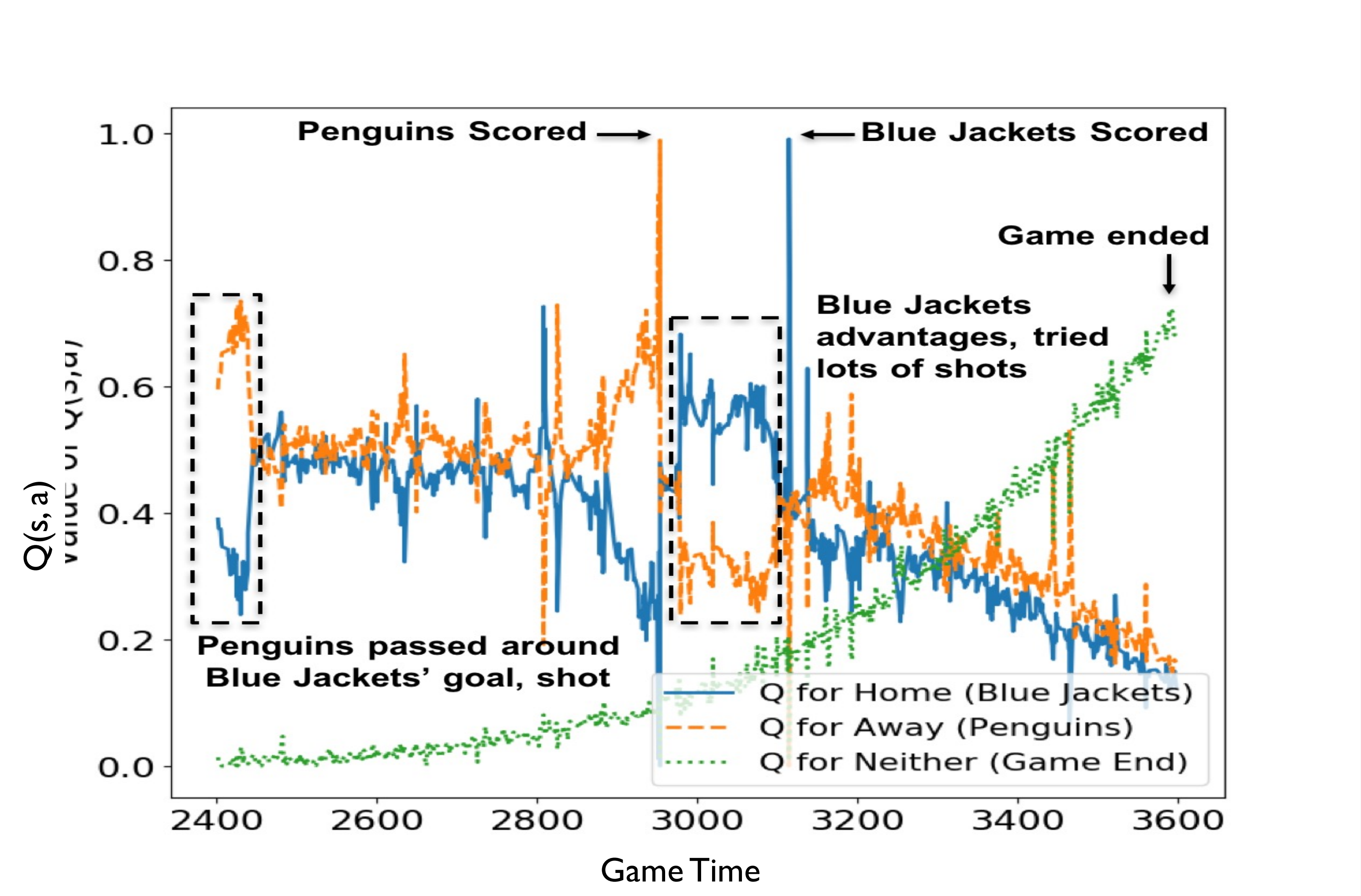
- Large-scale Machine Learning

Spatial Projection

Q-value for the action “shot” action over the rink.



Value Ticker: Temporal Projection



MODEL-FREE LEARNING

THE PROBLEM WITH TRANSITION PROBABILITIES

- Value iteration computes value functions, policy *given transition probabilities*
- In continuous/massive state spaces, working with transition probabilities is hard. Why?
  - Each transition occurs only once in data → cannot estimate probability
  - Cannot represent in matrix
  - Cannot sum over all states, actions (recall Bellman equation)

LEARNING WITHOUT TRANSITION PROBABILITIES

- ”model-free” learning = no transition probabilities
  - Unfortunate term, because we use a NN model to do “model-free” RL
- Learn target function directly <u>from episodes</u>
  - (Many people who know Markov decision processes but not reinforcement learning don’t know about model-free learning)
- The main problem: what is the loss function?
  - No supervision signal

EPISODES

- A **transition** is a 5-item sequence s,a,r,s',a'
- An **episode** is a sequence of transitions $s_0,a_0,r_0,s_1,a_1,r_1,\ldots,s_n,a_n,r_n$
- At every time *t*, we denote the episode return or (discounted) sum of future rewards as v_t.
- Examples:
  - Sports: each game is an episode. At each time *t* in the game, the return v_t specifies whether the team won or lost.
  - Tennis: each match is broken into set episodes which are broken into point episodes
  - Degree: each course is broken into components.
    - The return v_t specifies learning outcomes/grade at the end of course.

DRIVING HOME EXAMPLE

State	Elapsed Minutes	v_t: future time	Elapsed Minutes	v_t: future time
leaving office	0	43	0	44
reach car, raining	5	38	5	39
exit highway	20	23	22	22
behind truck	30	13	31	13
home street	40	3	41	3
arrive home	43	0	44	0

- Could have different states/actions in different episodes
  - E.g. some days no rain

MONTE CARLO LEARNING

Regression approach

REGRESSION APPROACH

- Start with a dataset of episodes
- For every observed state s_t, make the return v_t the target.
- Train a model to predict (the expected value of) v_t given s_t as input.

REGRESSION APPROACH: EXAMPLE

State = X	v_t = Y
leaving office	43
reach car, raining	38
exit highway	23
behind truck	13
home street	3
arrive home	0
leaving office	44
reach car, raining	39
exit highway	22
behind truck	13
home street	3
arrive home	0

- Regression: Predict Y given X
- People who know machine learning but not reinforcement learning use this approach
- Sports examples:
  - How to serve in volleyball
  - Valuing Actions by Estimating Probabilities in soccer
- Main Problem: regression methods assume i.i.d data
- But subsequent states have highly correlated return values

➢ Slow learning

TEMPORAL DIFFERENCE LEARNING

Learning a Value Function

TD LEARNING

- A key idea in CS RL, used in almost all RL AI systems
- Key intuition: compare 2 estimates

1. Current return estimate $V(s_t)$
2. Return estimate from look ahead 1 time step: $r_t + \gamma V(s_{t+1})$
3. TD-error for NN gradient = $[r_t + \gamma V(s_{t+1}) - V(s_t)]^2$

- Can be generalized to look-ahead multiple time steps

EXAMPLE

State	Elapsed Minutes	Predicted Time to Go	Predicted Total Time	TD-target	TD-error
leaving office	0	30	30	40 = 5 + 35	$(40-30)^2=100$
reach car, raining	5	35	40	30 = 15 + 15	$(35-30)^2 = 25$
exit highway	20	15	35	20 = 10 + 10	$(20-15)^2=25$
behind truck	30	10	40	13 = 10 + 3	$(13-10)^2=9$
home street	40	3	43	3 = 3 + 0	$(3-3)^2=0$
arrive home	43	0	43		

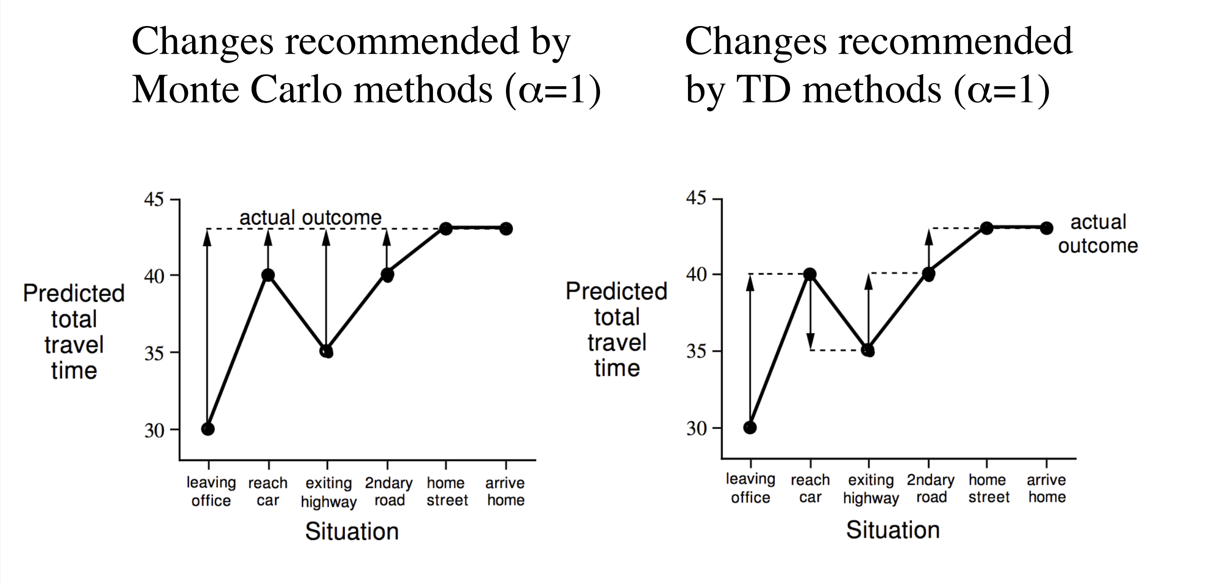
$$\text{TD-error} = [r_t + \gamma V(s_{t+1}) - V(s_t)]^2 \qquad \gamma=1$$

EXAMPLE

State	Elapsed Minutes	Predicted Time to Go	Predicted Total Time	TD-target	TD-error
leaving office	0	30	30	40 = 5 + 35	$(40\text{-}30)^2=100$
reach car, raining	5	35	40	30 = 15+15	$(35\text{-}40)^2 = 25$
exit highway	20	15	35	40	$(40\text{-}35)^2=25$
behind truck	30	10	40	43	$(43\text{-}40)^2=9$
home street	40	3	43	43	$(43\text{-}43)^2=0$
arrive home	43	0	43		

$$\text{TD-error} = [r_t + \gamma V(s_{t+1}) - V(s_t)]^2$$

TD VS. MC (REGRESSION)



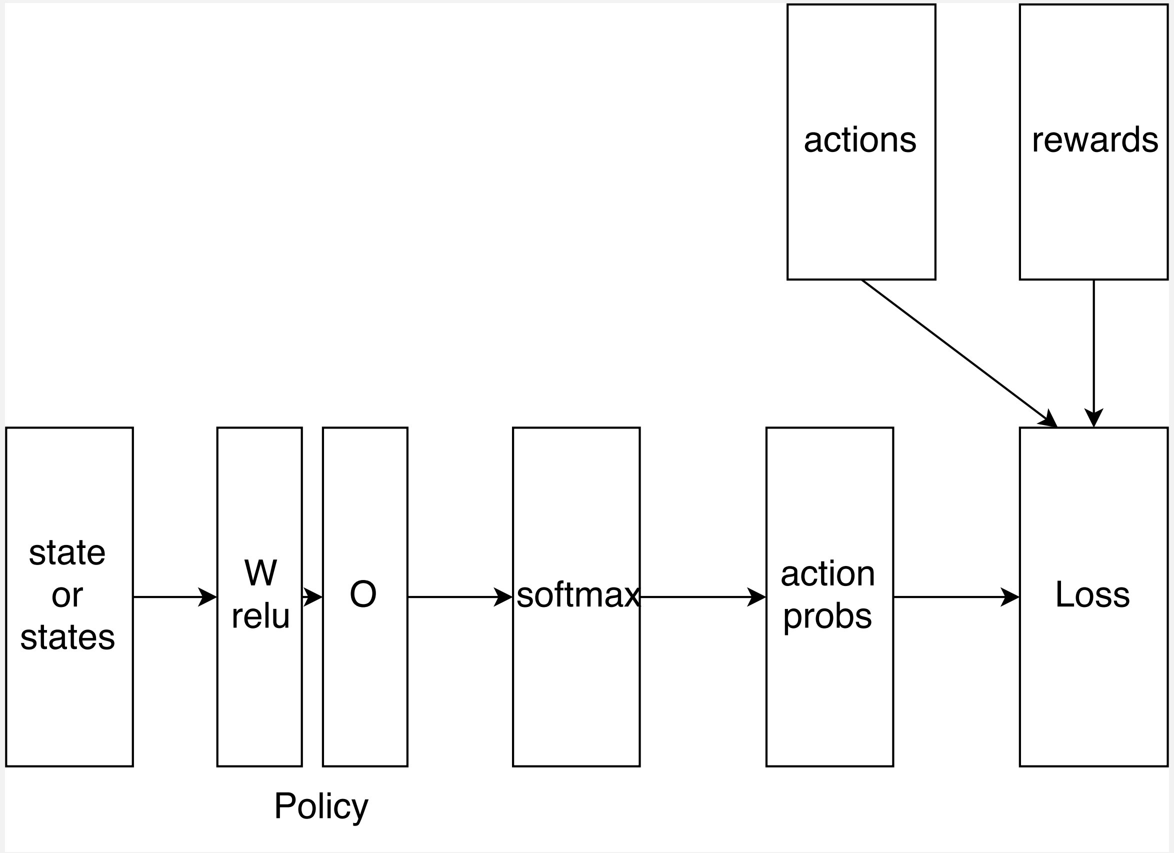
TD Pros

- Exploits temporal dependencies rather than ignoring them
- Can start learning before end of episode
- Deep versions converge faster than MC
- Technically: TD has lower variance, higher bias

POLICY LEARNING

POLICY LEARNING

- A policy maps a state to a distribution over a finite set of actions
  - Like multi-class learning
- Can be represented in a NN
- What is the objective function?



EXAMPLE: TRAINING ROBOT TO WALK

- Goal: learn a fast AIBO walk
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by policy gradient
- Performance metric = field traversal time

FREQUENCY DECOMPOSITION OF VALUE FUNCTION

- Consider the following frequency expression for the value of a policy
  $V^{\pi}(s_0) = \sum_s P^{\pi}(s|s_0) \times \sum_a Q^{\pi}(s,a) \times \pi(a|s)$
- $P^{\pi}(s|s_0)$ is the stationary (limiting) frequency of reaching s starting from s_0
  - We have seen an efficient dynamic programming algorithm for computing this

ANALYZING POLICIES

- The frequency decomposition is often a nice way to explain the performance of an agent
$V^{\pi}(s_0) = \sum_s P^{\pi}(s|s_0) \times \sum_a Q^{\pi}(s,a) \times \pi(a|s)$

What will the agent do in state s?

How often does the agent reach state s?

How well does the agent do in state s?

Hockey Example

- How often does a team achieve a powerplay state (s)?
- How successful are they with powerplay ($V^{\pi}(s)$)?

GENERAL APPROACH TO OPTIMIZING POLICIES

$$V^{\pi}(s_0) = \sum_s P^{\pi}(s|s_0) \times \sum_a Q^{\pi}(s,a) \times \pi(a|s)$$

Estimate observed #visits to initial state s_0

Estimate from data

optimize

- Amazingly, for gradient descent we can treat P and Q as fixed!

THE POLICY GRADIENT THEOREM

- Assume the policy is parametrized as $\pi_{\boldsymbol{\theta}}$
  - E.g., $\boldsymbol{\theta}$ = weights in neural network
- Constant independent of policy

Then $$\frac{dV^{\pi_\theta}(s_0)}{d\theta} = cE_\pi[Q^\pi(s,a)\frac{d\ln(\pi(a|s))}{d\theta}]$$

- Expected number of times (s,a) occurs under policy
- Can simply sum over all data points

Log probability of choosing action *a* in state *s*

POLICY OPTIMIZATION OPTIONS

- Average over observed episodes one of the following.
- Monte Carlo: $\text{argmax}_{\pi} \sum_t \pi(a_t|s_t) \times v_t(s_t,a_t)$
  where $v_t(s_t,a_t)$ is the observed episode return as in Monte Carlo learning
- Actor-Critic: $\text{argmax}_{\pi} \sum_t \pi(a_t|s_t) \times Q'(s_t,a_t)$
  where Q' is a value function estimated by the critic
  - Typically using TD-learning
- Advantage Actor-Critic
  - replace $Q'(s_t,a_t)$ by $Q'(s_t,a_t) - V'(s_t)$ = "advantage"/impact
  - Advantage values have lower variance than Q values →easier to learn
  - Current state of the art
  - Demo

EXAMPLE: REINFORCE ALGORITHM

1. Initialize $\boldsymbol{\theta}$
2. For each episode $<s_1,a_1,r_1,\ldots,s_T,a_T,r_T>$ do

   for t=1 to T do

   $\boldsymbol{\theta} := \boldsymbol{\theta} + \alpha\, \boldsymbol{\Delta}_{\theta} \log(\pi_{\theta}(a_t|s_t))\, v_t(s_t,a_t)$

   end for

   end for
3. Return $\boldsymbol{\theta}$

Learning rate

Estimated Return

DATA GATHERING ISSUES

ACTIONS INFLUENCE DATA

- A fundamental difference between RL and passive learning is that the agent's <u>actions influence their observations.</u>

➢ No fixed dataset to analyze

- Actions influence not only the rewards but also the quality of the information the agent receives
- Hockey example: maybe we need to place risky bets to get information about the payoffs?

EXPLORATION VS. EXPLOITATION

- An agent needs to do both
  - Select actions that seem optimal to keep high rewards
    - “exploit” its current knowledge
  - Select new actions to gather enough data to estimate a value function
    - “explore” the state space

BASIC EXPLORATION APPROACHES

- A simple but often effective approach is ε-greedy
  - With probability ε, select a random action (e.g. ε =10% of the time)
  - With probability 1-ε, select an action that is optimal according to the current value function
  - Grid world demo with TD learning
- ε-decreasing: decrease ε with every time step
  - Like a learning rate
- Policy-driven: sample actions from *probabilistic* policy
  - Works best with policy gradient methods
- State-of-the-Art: Upper Confidence Bound (UCB)
  - Estimate uncertainty in value functions at a state
  - Visit states with more uncertainty more often

EXPERIENCE REPLAY

- Recall that temporally successive states have highly correlated values

➢ Successive updates change value functions only little

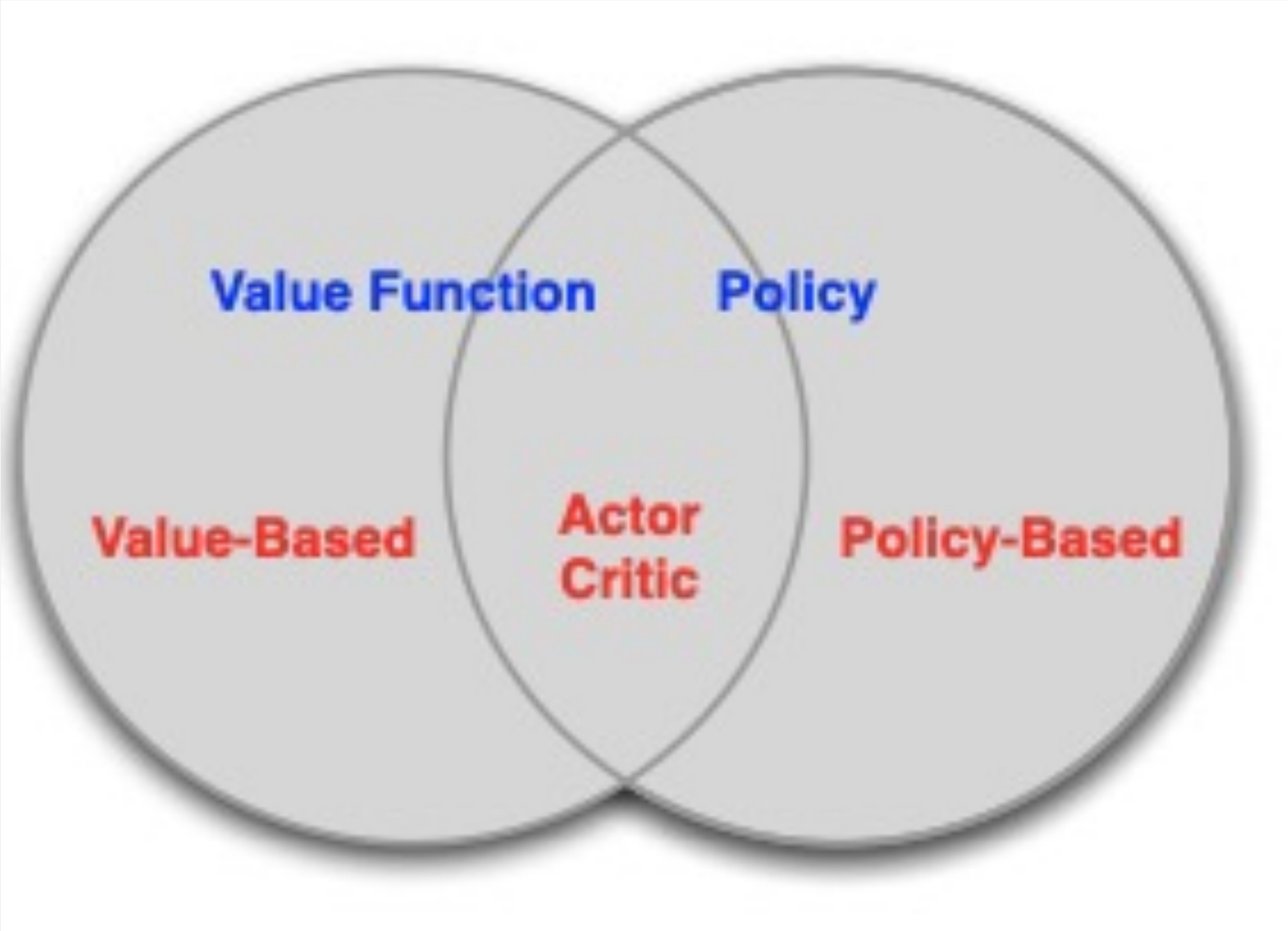
- Possible solution:

1. Store transitions $s_{t-1}, a_{t-1}, r_{t-1}, s_t, a_t$
2. Sample randomly from past transitions to update

- But is throwing away temporal information really a good thing?

SUMMARY OF APPROACHES

- Value-based
  - Learn Value Function
  - Implicit policy (e.g. ε–greedy)
- Policy-based
  - No Value Function
  - Learnt policy
- Actor-Critic
  - Learn value function
  - Learnt policy



SUMMARY RL

- Reinforcement learning aims to learn various key functions
- Value function, policy
- Can be naturally implemented as neural networks
- In large/infinite state spaces, need model-free learning
- Learning techniques for value functions
  - Monte-Carlo: reduce to regression
  - Temporal difference: make different value estimates at different times consistent with each other