R.O.B.O.T. Comics

Having EATR programmed by strict vegans to appease the public has unintended consequences.

CS 4649/7649 Robot Intelligence: Planning

Machine Learning IV: Deep Reinforcement Learning

CS 4649/7649 – Asst. Prof. Matthew Gombolay

Slides adapted from:
MIT 16.410 Emilio Frazolli
Russell and Norvig AIMA

Assignments

- Due 2/24
  - Reading: Russel & Norvig Ch. 20 (getting ready for POMDP)
- Due 3/1
  - PSet6
- Due 3/3
  - Reading TBD

Midterm 1

Thursday, February 10th at 9:30 AM – 10:45 AM

Midterm 1 – CS 4649/7649

CS 4649/7649 Robot Intelligence: Planning
Instructor: Dr. Matthew Gombolay

Notes:

- **CS 4649 Students – Please Select 6 of 8 Problems to complete (including the subparts for each problem).**
- **CS 7649 Students – Please Select 7 of 8 Problems to complete (including the subparts of each problem). Each of the 8 problems are equally weighted.**

Midterm 1

Problem 1.B

In each of the following, circle T if the statement is TRUE and F otherwise. Assume a finite graph with either directed or undirected edges.

i.	T	F		Assuming an edge cost of 1, BFS with a visited list is guaranteed to find a shortest path to the goal if one exists.
ii.	T	F		Assuming an edge cost of 1, DFS with a visited list is guaranteed to find a shortest path to the goal if one exists.
iii.	T	F		IDS with a visited list will always find the optimal path.
Iv.	T	F		DFS without a visited list is guaranteed to find a path to the goal if one exists, assuming no loop exists in the graph.
v.	T	F		BFS will typically perform with better speed than IDS.

Midterm 1

Problem 2

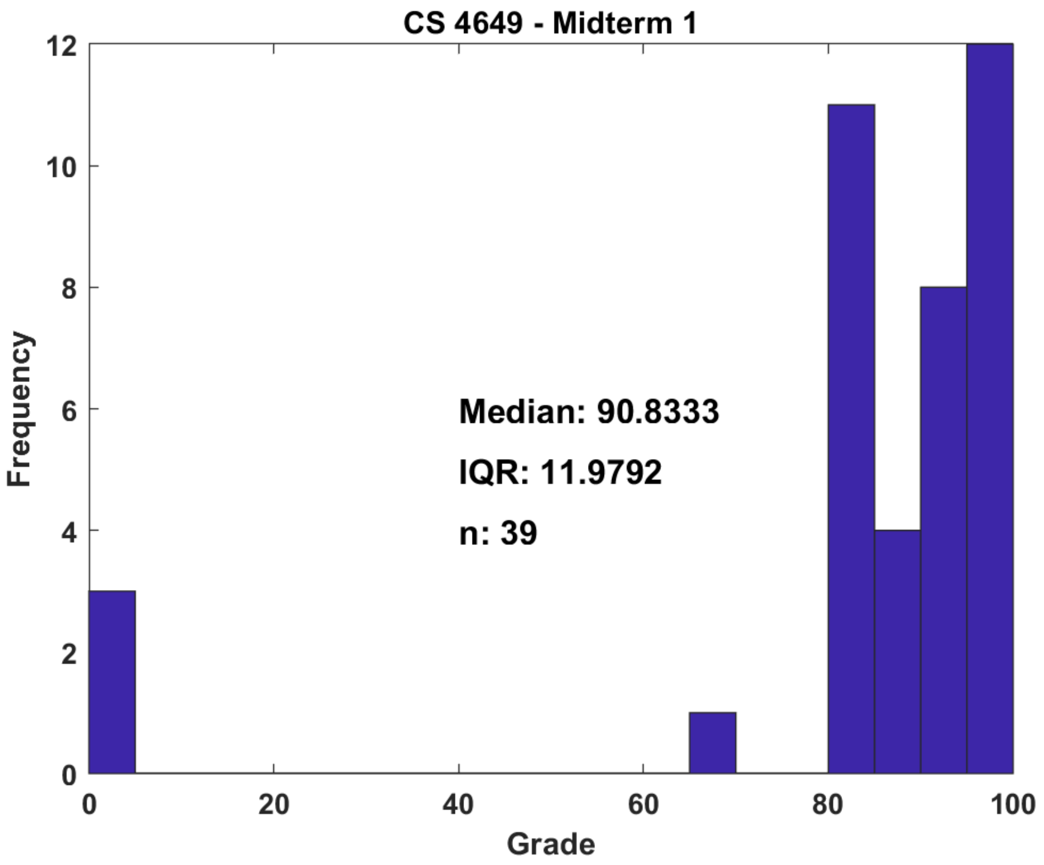
Zookeeper George was in charge of feeding all of the animals in the morning. He had a regular schedule that he followed every day. Each animal is fed once a day at one of the times in the table below ($t_1 = 6{:}30\ AM, t_2 = 6{:}45\ AM, t_3 = 7{:}00\ AM, t_4 = 7{:}15\ AM$, and $t_5 = 7{:}30\ AM$). In addition:

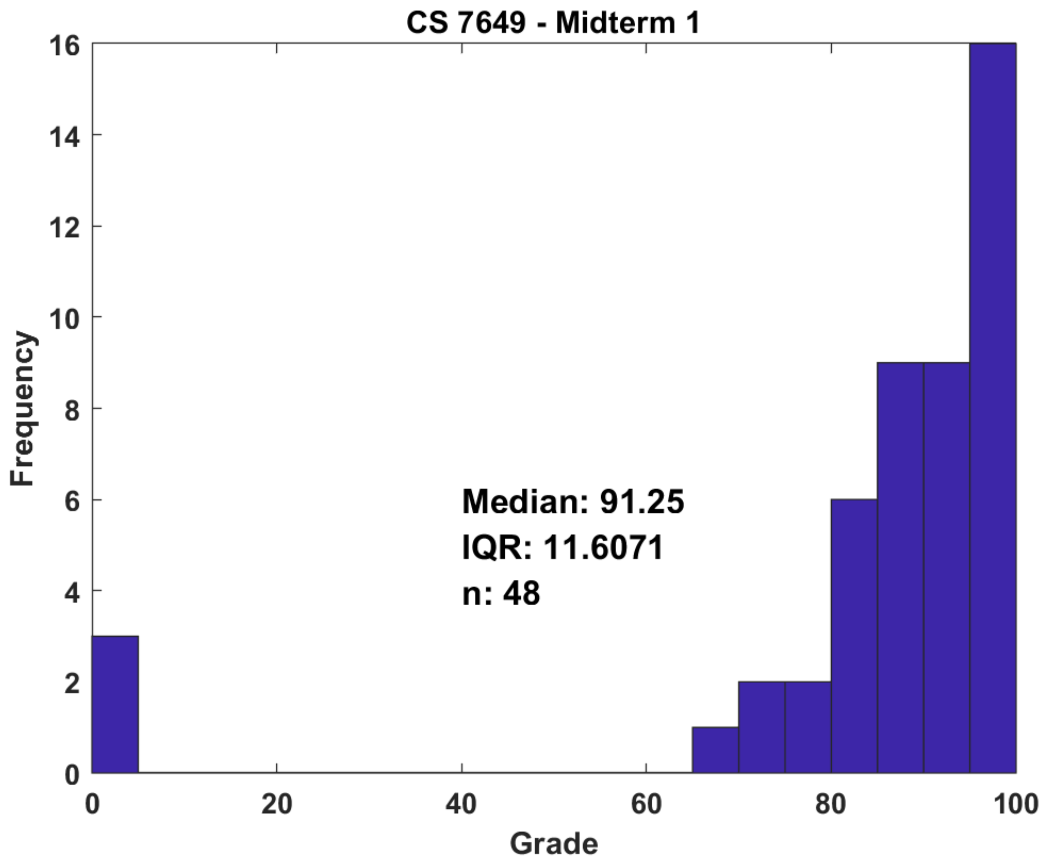
i. The monkeys were fed after the giraffes but before the zebras.
ii. The lions were fed 15 minutes after the bears.
iii. The zebras were fed after the lions.

	6:30 AM	**6:45 AM**	**7:00 AM**	**7:15 AM**	**7:30 AM**
Bears					
Giraffes					
Lions					
Monkeys					
Zebras					

To figure out George's schedule, please formulate this problem as a constraint satisfaction problem (CSP).

Midterm 1





~~Estimate: Overall Grade > 91% → A~~

Estimate: Overall Grade > 90% → A

Estimate: Overall Grade > 88% → A

Outline

- Review:
  - MDPs
  - Value Iteration
  - Policy Iteration

- Deep Q-Learning

- Policy Gradients

From deterministic to stochastic planning problems

A basic planning model for deterministic systems (e.g., graph/tree search algorithms, etc.) is:

Planning Model (Transition system + goal)

A discrete, deterministic, feasible planning model is defined by

- A countable set of states, S.
- A countable set of actions, A.
- A transition relation $\rightarrow \subseteq S \times A \times S$
- An initial state, $s_1 \in S$
- A set of goal states $\boldsymbol{s_G} \subset S$

- We considered the case in which the transition relation is purely deterministic: if (s, a, s') are in relation, i.e., $(s, a, s') \in \rightarrow$ or, more concisely, $s \xrightarrow{a} s'$, then taking action *a* from state *s* will always take the state to s'.
- Can we extend this model to include (probabilistic) uncertainty in the transitions?

From deterministic to stochastic planning problems

Instead of a (deterministic) transition relation, let us define transition probabilities; also, let us introduce a reward (or cost) structure:

Markov Decision Process (MDP) (Stochastic transition system + reward)

A discrete, deterministic, feasible planning model is defined by

- A countable set of states, S.
- A countable set of actions, A.
- A transition relation $T: S \times \mathrm{A} \times S \rightarrow [0,1]$
- An initial state, $s_1 \in S$
- A reward function, $R: S \times A \times S \rightarrow \mathcal{R}$

- In other words: if action a is applied from state s, a transition to state s! will occur with probability, $T(s, a, s')$
- Furthermore, every time a transition is made from s to s' using action a, a reward, $R(s, a, s')$, is collected.

Some Remarks

- In a Markov Decision Process, both transition probabilities and rewards only depend on the **present** state, not on the history of the state. In other words, the future states and rewards are independent of the past, given the present.

- A Markov Decision Process has many common features with Markov Chains and Transition Systems.

- In an MDP:
  - Transitions and rewards are stationary.
  - The state is known exactly. (Only transitions are stochastic.)

- MDPs in which the state is not known exactly (HMM + Transition Systems) are called Partially Observable Markov Decision Processes (POMDP's): these are very hard problems.

Total reward in an MDP

- Let us assume that it is desired to maximize the total reward collected over infinite time.
- In other words, let us assume that the trajectory, τ, is $\tau = \langle s_0, a_0, s_1, a_1, \dots, s_t, a_t, \dots \rangle$ along with rewards, $\langle r_0, r_1, \dots, r_t, \dots \rangle$, then the total collected reward is as follows, where $\gamma \in [0,1]$ is the discount factor.

$$V = \sum_{t=0}^{\infty} \gamma^t r_t$$

- Philosophically: it models the fact that an immediate reward is better than an uncertain reward in the future.
- Mathematically: $\gamma \in [0,1)$ it ensures that the sum is always finite, if the rewards are bounded (e.g., finitely many states/actions).

Decision-making in MDPS

- Notice that the actual sequence of states, and hence the actual total reward, is unknown a priori.
- We could choose a plan, i.e., a sequence of actions, $\langle \boldsymbol{a_0}, \boldsymbol{a_1}, \dots, \boldsymbol{a_t}, \dots \rangle$.
- In this case, transition probabilities are fixed and one can compute the probability of being at any given state at each time step—in a similar way as the forward algorithm in HMMs—and hence compute the expected reward:

$$\mathbb{E}[R(s_t, a_t, s_{t+1}) | s_t, a_t] = \sum_{s' \in S} T(s_t, a_t, s) R(s_t, a_t, s)$$

- Such approach is essentially open loop, i.e., it does not take advantage of the fact that at each time step the actual state reached is known, and a new feedback strategy can be computed based on this knowledge.

Introduction to Value Iteration

- Let us assume we have a function $V_i : S \rightarrow \mathcal{R}$ that associates to each state s a lower bound on the optimal (discounted) total reward $V^*(s)$ that can be collected starting from that state. Note the connection with admissible heuristics in informed search algorithms.

- For example, we can start with $V_o(s) = 0, \forall s \in S$

- As a feedback strategy, we can do the following: at each state, choose the action that maximizes the expected reward of the present action + estimate total reward from the next step onwards.

- Using this strategy, we can get an update V_{i+1} on the function V_i. Iterate until convergence...

Value Iteration Pseudocode

1. Set $V_0(s) \leftarrow 0, \forall s \in S; i = 0$
2. WHILE TRUE
3. $\quad V_{i+1}(s) \leftarrow \max_a \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_i(s')], \forall s \in S$
4. $\quad$ IF $\max_{s \in S} |V_{i+1}(s) - V_i(s)| < \epsilon$
5. $\quad\quad$ RETURN $\mathrm{V}^*(s) \approx V_{i+1}(s), \forall s \in S$
6. $\quad i++$
7. END WHILE

Under some technical assumptions the optimal value function, $\mathrm{V}^*(s)$, satisfies Bellman's Equation:

$$V^*(s) = \max_a \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

Q-Learning

Model-free MDPs

- In many cases, exact details of the MDP (i.e., transition probabilities) not known.
- Reinforcement learning: learn good control policies via the analysis of state/action/rewards sequences collected in simulation and/or experiments.
- Several options, e.g., :
  - Certainty equivalence: estimate transition probabilities through data, then apply standard methods. (Expensive, not on-line)
  - Temporal Difference learning: Only maintain an estimate of the value function, V. For each transition, e.g., $s \xrightarrow{a} s'$, update the estimate:

$$V(s) \leftarrow (1 - \alpha_t)V(s) + \alpha_t[R(s,a,s') + \gamma V(s')]$$

where $\alpha_t \in (0,1)$ is a learning parameter. Note: α_t should decay (e.g., $a_t = \frac{1}{t}$), as the number of updates goes to infinity. Learning depends on the particular policies applied.

Q-learning

- Estimate total collected reward for state-action pairs.
- Q-factor Q(s, a): estimate of the total collected reward collected

  (i) starting at state s,

  (ii) applying action a at the first step,

  (iii) acting optimally for all future times.
- Q-factor update law, based on an observed transition, $\mathrm{s} \overset{a}{\rightarrow} s'$

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t \left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$

Note: α_{t} should decay over time for convergence, e.g., $\alpha_{\mathrm{t}} = \frac{1}{\mathrm{t}}$

- Q-learning does not depend on a particular policy, π.
- Issue: exploitation (choose “best” action, a) versus exploration (choose a poorly characterized, action a.

Q-learning Pseudocode

Set $Q_0(s,a) \leftarrow 0, \forall s \in S, a \in A; i \leftarrow 0$

WHILE TRUE

$s_0 \sim \rho(s); \tau \leftarrow \langle \quad \rangle$

  FOR $t =$ 0 to T // "Policy" rollouts

    $a_t \leftarrow \begin{cases} \sim U(A), \text{with } p = \epsilon \\ \underset{a \in A}{\operatorname{argmax}} Q_i(s_t, a), \text{otherwise} \end{cases}$

    $s_{t+1} \sim T(s_t, a_t, \cdot)$

    $r_t \leftarrow R(s_t, a_t, s_{t+1})$

    $\tau \leftarrow \tau \cup \langle s_t, a_t, s_{t+1}, r_t \rangle$

  ENDFOR

  FOR all $\langle s, a, s', r \rangle \in \tau$ // Update Q-function

    $Q_{i+1}(s,a) \leftarrow (1 - \alpha_t) Q_i(s,a) + \alpha_t \left[r + \gamma \max_{a'} Q_i(s', a') \right]$

  ENDFOR

  IF $\max_{s \in \tau} |Q_{i+1}(s,a) - Q_i(s,a)| < \epsilon$

    RETURN $Q^*(s,a) \approx Q_{i+1}(s,a), \forall s \in S, a \in A$

  ENDIF

  $i{+}{+}$

ENDWHILE

Example

Approximation techniques

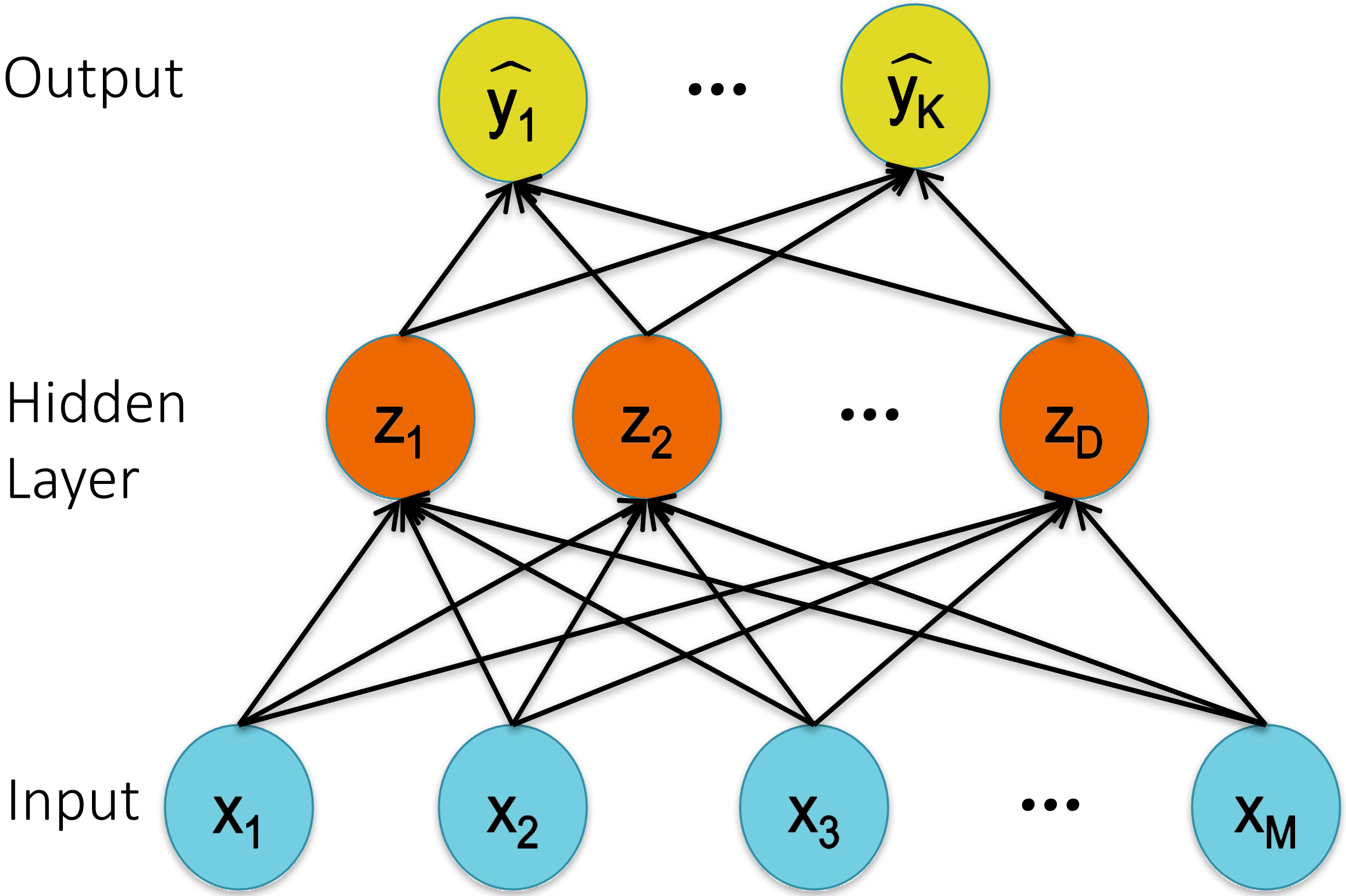
- Very often (e.g., when the state space, S, is a discretization of a continuous state space, e.g., $\mathcal{R}^n$), the dimensions of the state space make value iteration/policy iteration/Q-learning/etc. unfeasible in practice.
- Choose an approximation architecture, ψ, with m parameters, θ , e.g.,
  - ψ: basis functions, θ : coefficients
  - ψ : "feature vector", θ : coefficients
  - ψ : neural network, θ parameters (weights/biases, etc.)

$$\hat{Q}_\theta(s,a) = \sum_{k=1}^{m} \theta_k \psi_k(s,a)$$

- Updates to $Q(s,a)$ correspond to updates to the lower-dimensional vector, θ
- Goal: find best θ s.t. that $Q(s,a) \approx \hat{Q}_\theta(s,a), \forall s \in S, a \in A$

Deep Q-learning

Multi-class Output



(F) **Loss** $L = \frac{1}{2}(y - \hat{y})^2$

(E) **Output (softmax)** $\hat{y} = \frac{\exp\left(z_j^{(2)}\right)}{\sum_k \exp\left(z_k^{(2)}\right)}$

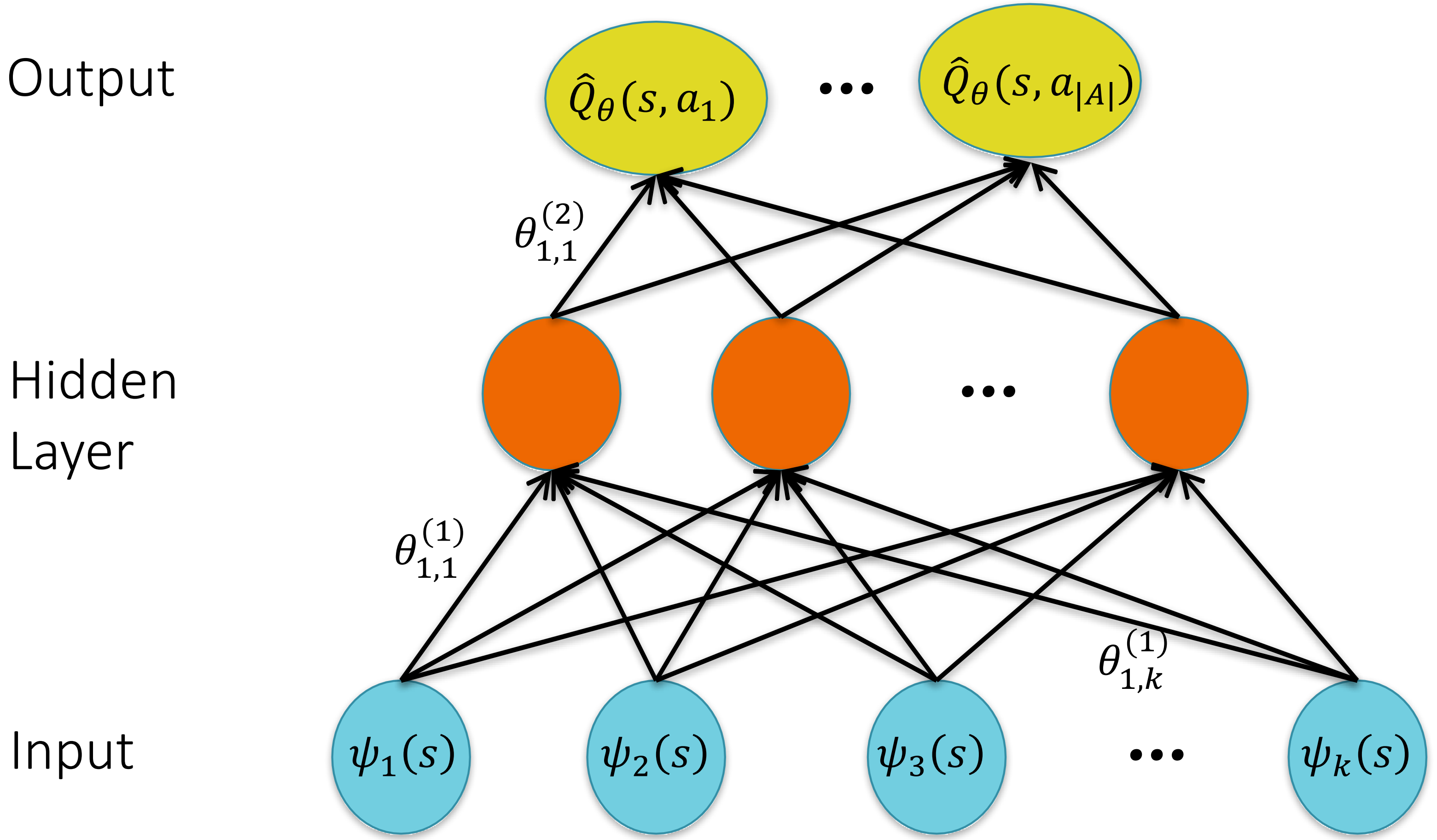
(D) **Output (linear)** $z_j^{(2)} = \sum_i \theta_{j,i}^{(2)} o_i^{(1)}$

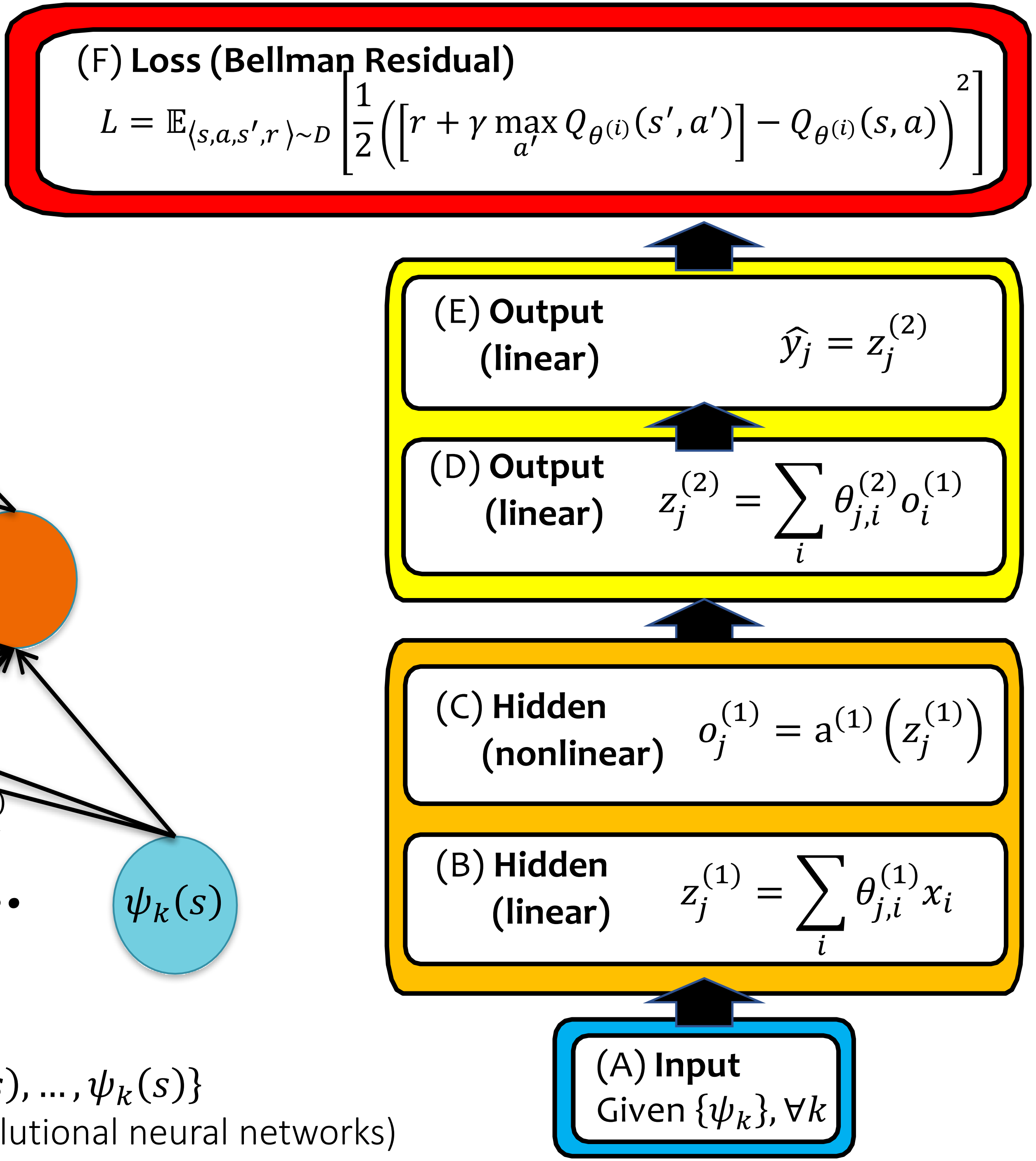
(C) **Hidden (nonlinear)** $o_j^{(1)} = \mathrm{a}^{(1)}\left(z_j^{(1)}\right)$

(B) **Hidden (linear)** $z_j^{(1)} = \sum_i \theta_{j,i}^{(1)} x_i$

(A) **Input**
Given $x_i, \forall i$

Deep Q-learning





Notes:
- Often use "feature" embedding of your state, $\{\psi_1(s), \psi_2(s), \ldots, \psi_k(s)\}$
- Could also use pixels of a game image (typically with convolutional neural networks)

Q-learning Pseudocode

Set $Q_0(s, a) \leftarrow 0, \forall s \in S, a \in A; i = 0$
WHILE TRUE
$s_0 \sim \rho(s); \tau \leftarrow \langle \quad \rangle$
FOR t = 0 to T // "Policy" rollouts
$a_t \leftarrow \begin{cases} \sim U(A), \text{with } p = \epsilon \\ \underset{a \in A}{\operatorname{argmax}} Q_i(s_t, a), \text{otherwise} \end{cases}$
$s_{t+1} \sim T(s_t, a_t, \cdot)$
$r_t \leftarrow R(s_t, a_t, s_{t+1})$
$\tau \leftarrow \tau \cup \langle s_t, a_t, s_{t+1}, r_t \rangle$
ENDFOR
FOR all $\langle s, a, s', r \rangle \in \tau$ // Update Q-function
$Q_{i+1}(s, a) \leftarrow (1 - \alpha_t) Q_i(s, a) + \alpha_t \left[r + \gamma \max_{a'} Q_i(s', a') \right]$
ENDFOR
IF $\max_{s \in \tau} |Q_{i+1}(s, a) - Q_i(s, a)| < \epsilon$
RETURN $\mathrm{Q}^*(\mathrm{s}, \mathrm{a}) \approx Q_{i+1}(s, a), \forall s \in S, a \in A$
ENDIF
$i++$
ENDWHILE

Deep Q-learning Pseudocode

```
Initialize Q_{θ^{(0)}}(s, a), ∀s ∈ S, a ∈ A as a neural network; i ← 0; Initialize replay buffer, D = {  }
WHILE TRUE
    s_0 ~ ρ(s);
    FOR t = 0 to T    // "Policy" rollouts
        a_t ← { ~U(A), with prob, ϵ
              { argmax_{a∈A} Q_{θ^{(i)}}(s_t, a), otherwise
        s_{t+1} ~ T(s_t, a_t, ·)
        r_t ← R(s_t, a_t, s_{t+1})
        D ← D ∪ ⟨s_t, a_t, s_{t+1}, r_t⟩
    ENDFOR
    θ^{(i+1)} ← θ^{(i)} + α_t Σ_{⟨s,a,s′,r⟩~D′⊂D} (1/|D′|) ([r + γ max_{a′} Q_{θ^{(i)}}(s′, a′)] − Q_{θ^{(i)}}(s, a)) ∇_θ Q_{θ^{(i)}}(s, a)
    IF (|θ^{(i+1)} − θ^{(i)}|) < ϵ
        RETURN Q*(s, a) ≈ Q_θ(s, a), ∀s ∈ S, a ∈ A
    ENDIF
    i++
ENDWHILE
```

$$\theta^{(i+1)} \leftarrow \theta^{(i)} + \alpha_t \sum_{\langle s,a,s',r\rangle \sim D' \subset D} \frac{1}{|D'|} \underbrace{\Big(\underbrace{\Big[r + \gamma \max_{a'} Q_{\theta^{(i)}}(s',a')\Big]}_{\text{Target } (y)} - \underbrace{Q_{\theta^{(i)}}(s,a)}_{\text{Guess } (\hat{y})} \Big)}_{\text{Bellman Residual}} \nabla_\theta Q_{\theta^{(i)}}(s,a)$$

Bellman Residual

Target (y)

Guess ($\hat{y}$)

Target and Guess use same model (i.e., bootstrapping)

Only differentiate w.r.t. the second $Q_{\theta^{(i)}}$ term

Example

Mid-lecture Break

Simple Lego building task

Confederate
the

icipant builds Legos

Particip y round

Schrum, M.L., Neville, G., Johnson, M., Moorman, N., Paleja, R., Feigh, K.M. and Gombolay, M.C., 2021, March. Effects of Social Factors and Team Dynamics on Adoption of Collaborative Robot Autonomy. In Proceedings of the 2021 ACM/IEEE International Conference on Human-Robot Interaction (pp. 149-157).

Deadly Triad

1. Function approximation
   - e.g., Deep Q-function

2. Off-policy learning
   - The fact that our “policy” rollouts do not perfectly reflect what the function approximator says the best action to take in each state would be

3. Boostrapping
   - The Q-function itself is both what we are regressing and what we are regressing towards (“chasing your tail”)

Same function!

$$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha_t \left(\underbrace{\left[r + \gamma \max_{a'} Q_\theta(s',a') \right]}_{\text{Regressing towards}} - \underbrace{Q_\theta(s,a)}_{\text{Regressing}} \right) \nabla_\theta \mathrm{Q}(\mathrm{s},\mathrm{a})$$

Double Deep Q-learning Pseudocode

1. Initialize $Q_{\theta^{(0)}}(s,a)$ and $Q_{\phi^{(0)}}(s,a), \forall s \in S, a \in A$ as a neural network; Initialize replay buffer, D
2. WHILE TRUE
3. $s_0 \sim \rho(s)$;
4. FOR t = 0 to T // "Policy" rollouts
5. $a_t \leftarrow \begin{cases} \sim U(A), \text{with prob}, \epsilon \\ \underset{a \in A}{\text{argmax}}\, Q_{\theta^{(i)}}(s_t, a), \text{otherwise} \end{cases}$
6. $s_{t+1} \sim T(s_t, a_t, \cdot)$
7. $r_t \leftarrow R(s_t, a_t, s_{t+1})$
8. $D \leftarrow D \cup \langle s_t, a_t, s_{t+1}, r_t \rangle$
9. ENDFOR
10. $\theta^{(i+1)} \leftarrow \theta^{(i+1)} + \alpha \sum_{\sim D' \subset D} \frac{1}{|D'|} \left(\left[r + \gamma Q_{\phi^{(i)}}(s', \text{argmax}_{a'}\, Q_{\theta^{(i)}}(s', a')) \right] - Q_{\theta^{(i)}}(s,a) \right) \nabla_{\theta^{(i)}} Q(s,a)$
11. $\phi^{(i+1)} \leftarrow \phi^{(i)}(1 - \alpha') + \alpha \theta^{(i)}$
12. IF $\left(\left| \theta^{(i+1)} - \theta^{(i)} \right| \right) < \epsilon \wedge \left(\left| \phi^{(i+1)} - \phi^{(i)} \right| < \epsilon \right)$
13. RETURN $\mathrm{Q}^*(\mathrm{s}, \mathrm{a}) \approx Q_\theta(s,a), \forall s \in S, a \in A$
14. ENDIF
15. $i++$
16. ENDWHILE

Gets rid of bootstrapping!

Example

Policy Gradient-based Methods

Instead of learning a Q-function that estimates the value of taking an action in each state, let's learn a policy that directly tells us what action to take. This policy will output a probability distribution over actions.

Policy Iteration Pseudocode

Set $V_0(s) \leftarrow 0, \forall s \in S, \forall s \in S, i = 0$

Initialize $\pi(s)$ to pick one action in each state, $s \in S$

WHILE TRUE

    // Solve linear program:

    $V_{i+1}(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_i(s')], \forall s \in S$

    IF $\max_{s \in S} |V_{i+1}(s) - V_i(s)| < \epsilon$

        RETURN $V^*(s) \approx V_{i+1}(s), \forall s \in S$

    // Update policy

    $\pi(s) \leftarrow \mathrm{argmax}_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{i+1}(s')], \forall s \in S$

    $i++$

END WHILE

REINFORCE

Goal:

$$\pi_\theta^*(s) = \mathrm{argmax}_\theta V^{\pi_\theta}(s)$$

$$V^{\pi_\theta}(s) = \mathbb{E}_{a \sim \pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right]$$

How do we get there in a model-free way?

REINFORCE

$$
\begin{aligned}
V^{\pi_\theta}(s) &= \mathbb{E}_{a\sim\pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right] \\
\nabla_\theta V^{\pi_\theta}(s) &= \nabla_\theta \mathbb{E}_{a\sim\pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right] \\
&= \nabla_\theta \sum_a \pi(a|s) Q_\phi^{\pi_\theta}(s,a) \\
&= \sum_a \nabla_\theta \left(\pi(a|s) Q_\phi^{\pi_\theta}(s,a)\right) \\
&= \sum_a \left[Q_\phi^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a|s) + \pi_\theta(a|s)\nabla_\theta Q_\phi^{\pi_\theta}(s,a)\right]
\end{aligned}
$$

0

$$
\begin{aligned}
&= \sum_a \left[Q_\phi^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a|s) + \pi_\theta(a|s)\nabla_\theta \sum_{s'} T(s'|s,a)\left(R(s,a,s') + \gamma V^{\pi_\theta}(s')\right)\right] \\
&= \sum_a \left[Q_\phi^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a|s) + \pi_\theta(a|s)\sum_{s'} T(s'|s,a)\gamma \nabla_\theta V^{\pi_\theta}(s')\right]
\end{aligned}
$$

$$
\nabla_\theta V^{\pi_\theta}(s') = \sum_{a'} \left[Q_\phi^{\pi_\theta}(s',a')\nabla_\theta \pi_\theta(a'|s') + \pi_\theta(a'|s')\sum_{s''} T(s''|s',a')\gamma \nabla_\theta V^{\pi_\theta}(s'')\right]
$$

REINFORCE

$$V^{\pi_\theta}(s) = \mathbb{E}_{a\sim\pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right]$$

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_a \left[Q_\phi^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a|s) + \pi_\theta(a|s)\sum_{s'} T(s'|s,a)\gamma\left[\sum_{a'}\left[Q_\phi^{\pi_\theta}(s',a')\nabla_\theta\pi_\theta(a'|s') + \pi_\theta(a'|s')\sum_{s''} T(s''|s',a')\gamma\nabla_\theta V^{\pi_\theta}(s'')\right]\right]\right]$$

$$= \sum_a \left[Q_\phi^{\pi_\theta}(s,a)\nabla_\theta\pi_\theta(a|s)\right]$$

Probability of $\boldsymbol{s} \rightarrow \boldsymbol{s'}$, i.e. $\boldsymbol{Pr}[\boldsymbol{s'}|\boldsymbol{k} = \boldsymbol{1}, \boldsymbol{s}, \boldsymbol{\pi}]$

$$+\gamma\sum_a \left[\pi_\theta(a|s)\sum_{s'}\left[T(s'|s,a)\sum_{a'}\left[Q_\phi^{\pi_\theta}(s',a')\nabla_\theta\pi_\theta(a'|s')\right]\right]\right]$$

$$+\gamma^2\sum_a\left[\pi_\theta(a|s)\sum_{s'}\left[T(s'|s,a)\sum_{a'}\left[\pi_\theta(a'|s')\sum_{s''}\left[T(s''|s',a')\nabla_\theta V^{\pi_\theta}(s'')\right]\right]\right]\right]$$

Probability of $\boldsymbol{s} \rightarrow \boldsymbol{s''}$, i.e. $\boldsymbol{Pr}[\boldsymbol{s''}|\boldsymbol{k} = \boldsymbol{2}, \boldsymbol{s}, \boldsymbol{\pi}]$

REINFORCE

$$V^{\pi_\theta}(s) = \mathbb{E}_{a\sim\pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right]$$

$$\nabla_\theta V^{\pi_\theta}(s) = \sum_{s'\in S}\sum_{k=0}^{\infty}\gamma^k \Pr[s\to s'|k,\pi]\sum_a Q(s',a)\nabla\pi_\theta(a|s')$$

Policy Gradient Update:

$$\nabla_\theta V^{\pi_\theta}(\{s_t\}_{t=0}^{\infty}) = \Delta\theta$$

$$\Delta\theta = \sum_{t=0}^{\infty} Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta\pi_\theta(a_t|s_t) = \sum_{t=0}^{\infty}\pi_\theta(a_t|s_t)Q_\phi^{\pi_\theta}(s_t,a_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}$$

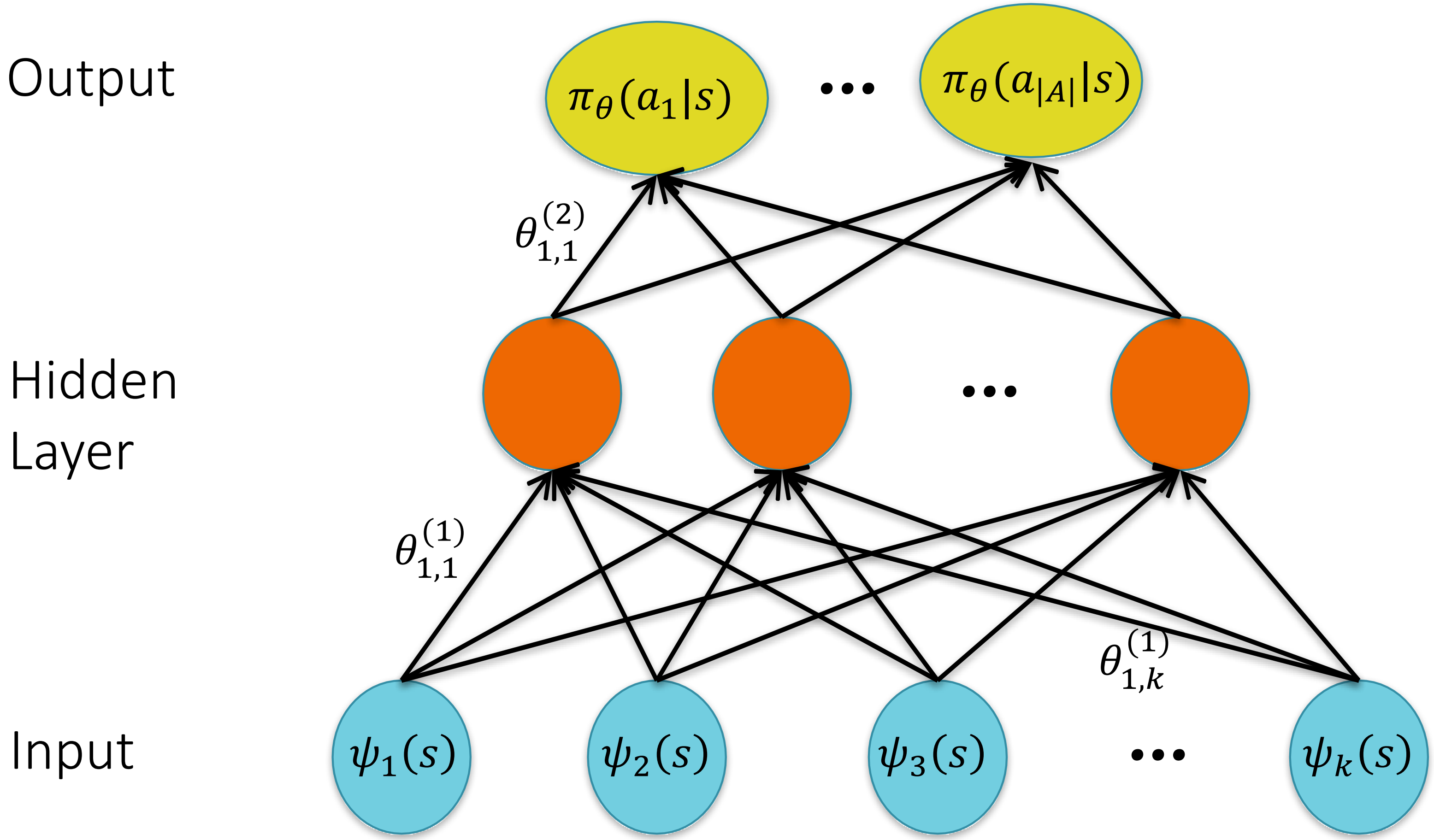
$$= \sum_{t=0}^{\infty}\pi_\theta(a_t|s_t)Q_\phi^{\pi_\theta}(s_t,a_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)} = \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}\left[Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta\log\pi_\theta(a_t|s_t)\right]$$

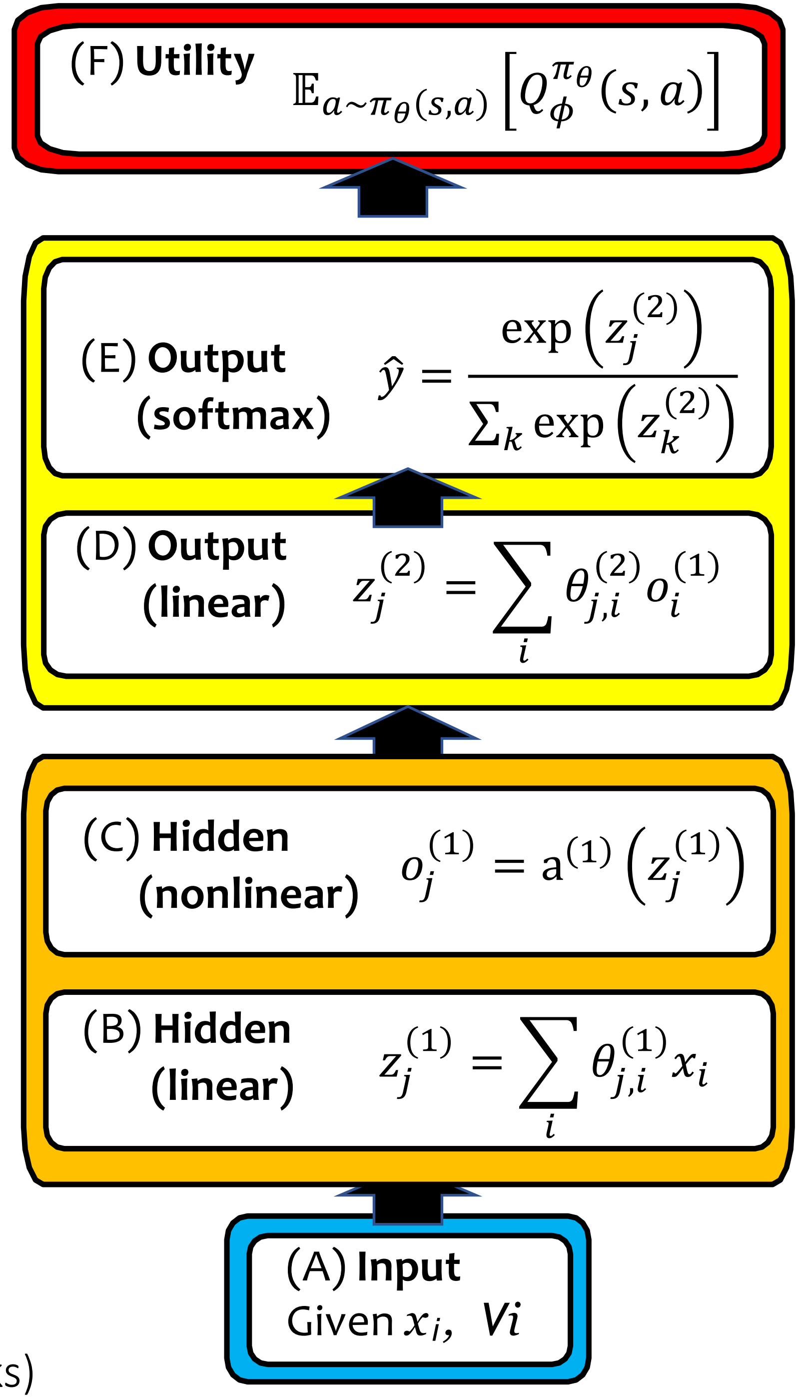
Actual REINFORCE Update

$$\sim Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta\log\pi_\theta(a_t|s_t)$$

Simple approximation: $Q_\phi^{\pi_\theta}(s_t,a_t) \approx \sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}$

Deep Policy Gradients





Notes:
- Often use “feature” embedding of your state, $\{\psi_1(s), \psi_2(s), \dots, \psi_k(s)\}$
- Could also use pixels of a game image (typically with convolutional neural networks)

REINFORCE Pseudocode

Initialize $\pi_{\theta^{(0)}}(s,a), \forall s \in S, a \in A$ as a neural network; $i \leftarrow 0$

WHILE TRUE

  $s_0 \sim \rho(s)$; $\tau = \langle \quad \rangle$

  FOR t = 0 to T // Policy rollouts

    $a_t \sim \pi_{\theta^{(i)}}(\cdot, s_t)$

    $s_{t+1} \sim T(\cdot\,|s_t, a_t)$

    $r_t \leftarrow R(s_t, a_t, s_{t+1})$

    $\tau \leftarrow \tau \cup \langle s_t, a_t, s_{t+1}, r_t \rangle$

  ENDFOR

  $\theta^{(i+1)} \leftarrow \theta^{(i)} + \frac{\alpha}{|\tau|} \sum_{\langle s_t, a_t, s_{t+1}, r_t \rangle \in \tau} \left(\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_{\theta^{(i)}}(a_t|s_t)$

  IF $\left|\theta^{(i+1)} - \theta^{(i)}\right| < \epsilon$

    RETURN $\pi^*(\mathrm{s}, \mathrm{a}) \approx \pi_{\theta^{(i+1)}}(s,a), \forall s \in S, a \in A$

  ENDIF

  $i++$

ENDWHILE

Example

REINFORCE

$$V^{\pi_\theta}(s) \quad = \mathbb{E}_{a\sim\pi_\theta(s,a)}\left[Q_\phi^{\pi_\theta}(s,a)\right]$$

$$\nabla_\theta V^{\pi_\theta}(s) \quad = \sum_{s'\in S}\sum_{k=0}^{\infty}\gamma^k \Pr[s \to s'|k,\pi]\sum_a Q(s',a)\nabla\pi_\theta(a|s')$$

Policy Gradient Update:

$$\nabla_\theta V^{\pi_\theta}(\{s_t\}_{t=0}^{\infty}) = \quad \Delta\theta$$

$$\Delta\theta = \quad \sum_{t=0}^{\infty} Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta\pi_\theta(a_t|s_t) = \sum_{t=0}^{\infty}\pi_\theta(a_t|s_t)Q_\phi^{\pi_\theta}(s_t,a_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)}$$

$$= \quad \sum_{t=0}^{\infty}\pi_\theta(a_t|s_t)Q_\phi^{\pi_\theta}(s_t,a_t)\frac{\nabla_\theta\pi_\theta(a_t|s_t)}{\pi(a_t|s_t)} = \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}\left[Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)\right]$$

$$\sim \quad Q_\phi^{\pi_\theta}(s_t,a_t)\nabla_\theta \log\pi_\theta(a_t|s_t)$$

Simple approximation: $Q_\phi^{\pi_\theta}(s_t,a_t) \approx \sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}$

Can we actually learn the Q-function?

Actor-Critic (AC) Method Pseudocode

Initialize $\pi_{\theta^{(0)}}(s,a)$ and $Q_{\phi^{(0)}}(s,a)$ $\forall s \in S, a \in A$ as neural networks; $i \leftarrow 0$

Initialize $D \leftarrow \{\quad\}$

WHILE TRUE

$s_0 \sim \rho(s)$; $\tau = \langle\quad\rangle$

FOR t = 0 to T // Policy rollouts

$a_t \sim \pi_{\theta^{(i)}}(\cdot, s_t)$

$s_{t+1} \sim T(\cdot\,|s_t, a_t)$

$r_t \leftarrow R(s_t, a_t, s_{t+1})$

$\tau \leftarrow \tau \cup \langle s_t, a_t, s_{t+1}, r_t\rangle$; $D \leftarrow D \cup \langle s_t, a_t, s_{t+1}, r_t\rangle$

ENDFOR

$\theta^{(i+1)} \leftarrow \theta^{(i)} + \alpha \sum_{\langle s_t, a_t\rangle \in \tau} \frac{1}{|\tau|} Q_{\phi^{(i)}}(s_t, a_t) \nabla_\theta \log \pi_{\theta^{(i)}}(a_t|s_t)$

$\phi^{(i+1)} \leftarrow \phi^{(i)} + \alpha' \sum_{\langle s,a,s',r\rangle \sim D' \subset D} \frac{1}{|D'|} \left(\left[r + \gamma \max_{a'} Q_{\phi^{(i)}}(s', a')\right] - Q_{\phi^{(i)}}(s,a)\right) \nabla_\phi^{(i)} Q_{\phi^{(i)}}(s,a)$

IF $\left(\left|\theta^{(i+1)} - \theta^{(i)}\right|\right) < \epsilon \wedge \left(\left|\phi^{(i+1)} - \phi^{(i)}\right| < \epsilon\right)$

RETURN $\pi^*(\mathrm{s}, \mathrm{a}) \approx \pi_{\theta^{(i+1)}}(s,a), \forall s \in S, a \in A$

ENDIF

$i++$

ENDWHILE

Advantage Function Actor-Critic (A2C)

Initialize $\pi_{\theta^{(0)}}(s,a)$ and $Q_{\phi^{(0)}}(s,a)$ and $V^{\pi}_{\psi^{(0)}}(s,a)$, $\forall s \in S, a \in A$ as neural networks; $i \leftarrow 0$

Initialize $D \leftarrow \{\quad\}$

WHILE TRUE

    $s_0 \sim \rho(s);\ \tau = \langle\quad\rangle$

    FOR t = 0 to T // Policy rollouts

        $a_t \sim \pi_{\theta^{(i)}}(\cdot, s_t)$

        $s_{t+1} \sim T(\cdot\,|s_t, a_t)$

        $r_t \leftarrow R(s_t, a_t, s_{t+1})$

        $\tau \leftarrow \tau \cup \langle s_t, a_t, s_{t+1}, r_t\rangle;\quad D \leftarrow D \cup \langle s_t, a_t, s_{t+1}, r_t\rangle$

    ENDFOR

    $\theta^{(i+1)} \leftarrow \theta^{(i)} + \alpha \sum_{\langle s_t, a_t\rangle \in \tau} \frac{1}{|\tau|}\left(Q_{\phi^{(i)}}(s_t, a_t) - V^{\pi}_{\psi^{(i)}}(s_t)\right) \nabla_\theta \log \pi_{\theta^{(i)}}(a_t|s_t)$

    $\phi^{(i+1)} \leftarrow \phi^{(i)} + \alpha' \sum_{\langle s,a,s',r\rangle \sim D' \subset D} \frac{1}{|D'|}\left(\left[r + \gamma \max_{a'} Q_{\phi^{(i)}}(s', a')\right] - Q_{\phi^{(i)}}(s,a)\right) \nabla^{(i)}_{\phi} Q_{\phi^{(i)}}(s,a)$

    $\psi^{(i+1)} \leftarrow \psi^{(i)} + \alpha'' \sum_{\langle s,a,s',r\rangle \sim \tau} \frac{1}{|\tau|}\left(\left[r + \gamma V^{\pi}_{\psi^{(i)}}(s')\right] - V^{\pi}_{\psi^{(i)}}(s)\right) \nabla_\psi V^{\pi}_{\psi^{(i)}}(\mathrm{s_t})$

    IF $\left(\left|\theta^{(i+1)} - \theta^{i}\right|\right) < \epsilon \wedge \left(\left|\phi^{(i+1)} - \phi^{i}\right| < \epsilon\right) \wedge \left(\left|\psi^{(i+1)} - \psi^{i}\right| < \epsilon\right)$

        RETURN $\pi^*(\mathrm{s,a}) \approx \pi_{\theta^{(i+1)}}(s,a), \forall s \in S, a \in A$

    ENDIF

    $i++$

ENDWHILE

Example

Takeaways

1. Value Iteration : Q-Learning :: Policy Iteration : A2C

2. The deadly Triad

3. How do design NN’s for approximating a Q-function, policy, or value function

4. Deep Q-learning, Double-Deep Q-Learning, REINFORCE, AC, A2C

5. Derivation of REINFORCE