CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 10: Deep Reinforcement Learning

CSCI 566 Roadmap

Module 1:
Neural Network Basics

Module 2:
Deep Learning Applications

Module 3:
Advanced Topics in Deep Learning

January 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

February 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28				

March 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

April 2023

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

[April 7] Endgame Deliverables

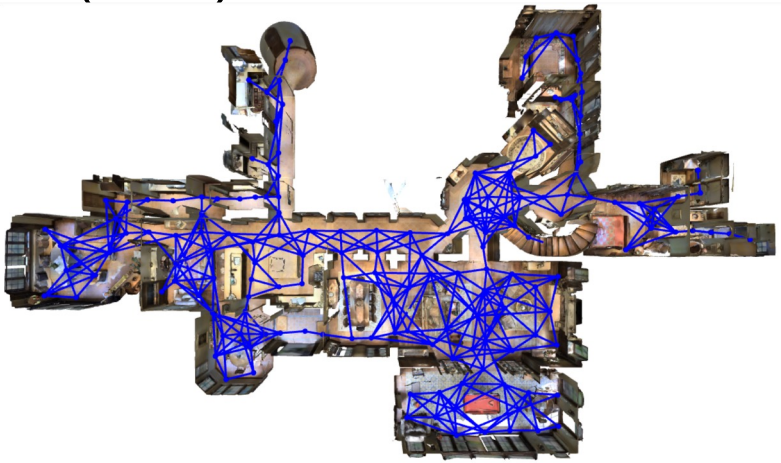
Paper Roleplaying Breakout Sessions	April 7, April 14
Final Project Presentations	April 21, April 28
Project Final Report	May 3

Overview of Today's Plan

- ~~Course organization and deliverables~~
  - Any questions before we move on?
- Lecture 9 Recap
- Deep Reinforcement Learning
- April 7 Project Roleplaying Breakouts

Vision-and-Language Navigation (VLN)

Leave the bedroom, and enter the kitchen. Walk forward, and take a left at the couch. Stop in front of the window.



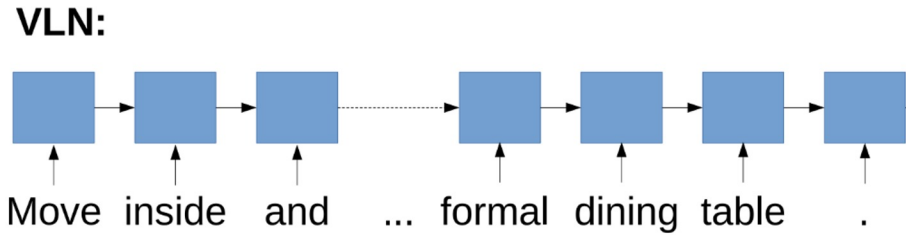
Visual Observations	$\mathcal{I}$
Language Instructions	$\mathcal{L}$
Action Space	$\mathcal{A}$
History	$\mathcal{H} = \{(v_i, l_i), i = 1 \dots T; v_i \in \mathcal{I}, l_i \in \mathcal{L}\}$
Policy	$\pi : \mathcal{I} \times \mathcal{L} \times \mathcal{H} \rightarrow \mathcal{A}$

https://bringmeaspoon.org/

World Actions and Consequences

- Agents perform sequence prediction
- We are no longer taking in input and producing a single output
  - E.g., retrieval of an image from language
- Want to learn a *policy* $\pi(a|s)$ from *states* to *actions*
- We are taking in observations (e.g., language and vision) and producing *one action at a time* that causes *new observations we can consider;* our state is estimated by observations
  - Open-loop: Produce whole sequence at once
  - Closed-loop: Consider observations after each action

Vision-and-Language Navigation with LSTM



Visual Observations	$\mathcal{I}$
Language Instructions	$\mathcal{L}$
Action Space	$\mathcal{A}$
History	$\mathcal{H} = \{(v_i, l_i), i = 1 \dots T; v_i \in \mathcal{I}, l_i \in \mathcal{L}\}$
Policy	$\pi : \mathcal{I} \times \mathcal{L} \times \mathcal{H} \rightarrow \mathcal{A}$

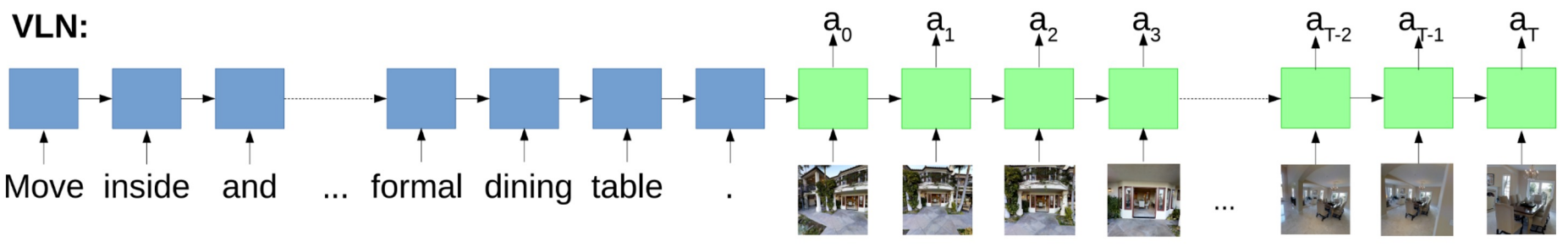
Vision-and-Language Navigation with LSTM

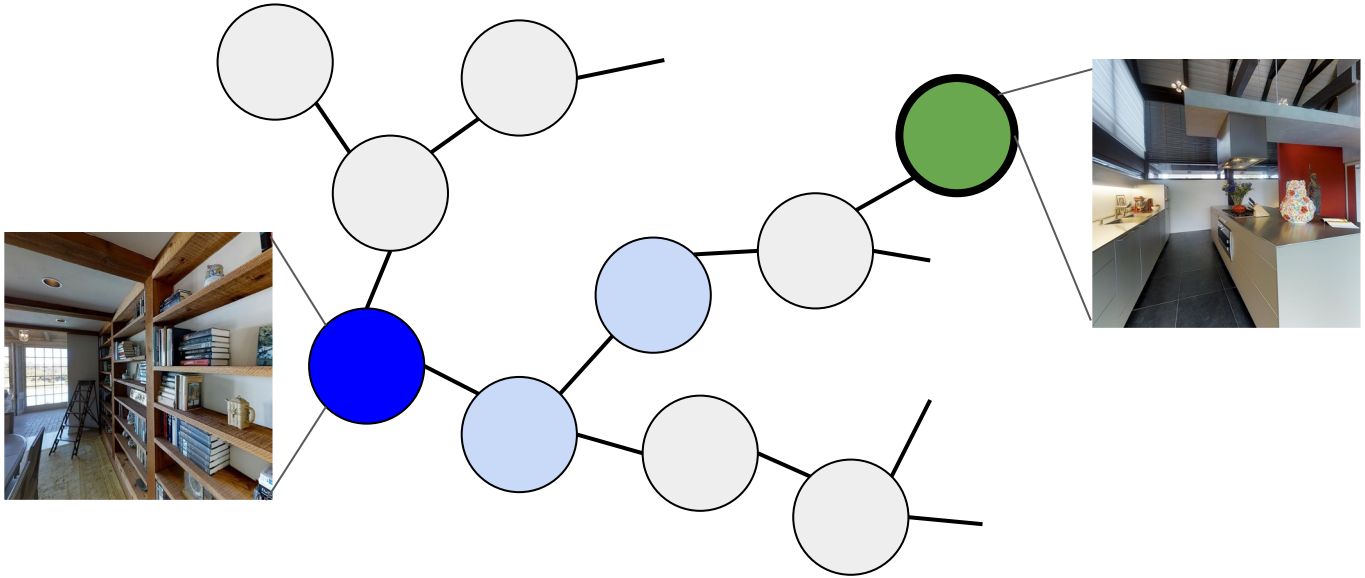
- The resulting LSTM is a *policy* from *states* represented as history so far, image observations, and language instructions to *actions* in the environment
- With training data, this is just *supervised learning* still; in this community you'll also see this called *imitation learning*

Visual Observations	$\mathcal{I}$
Language Instructions	$\mathcal{L}$
Action Space	$\mathcal{A}$

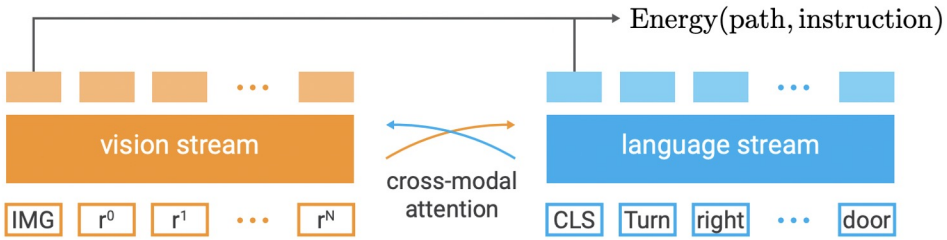
History	$\mathcal{H} = \{(v_i, l_i), i = 1 \dots T; v_i \in \mathcal{I}, l_i \in \mathcal{L}\}$
Policy	$\pi : \mathcal{I} \times \mathcal{L} \times \mathcal{H} \rightarrow \mathcal{A}$

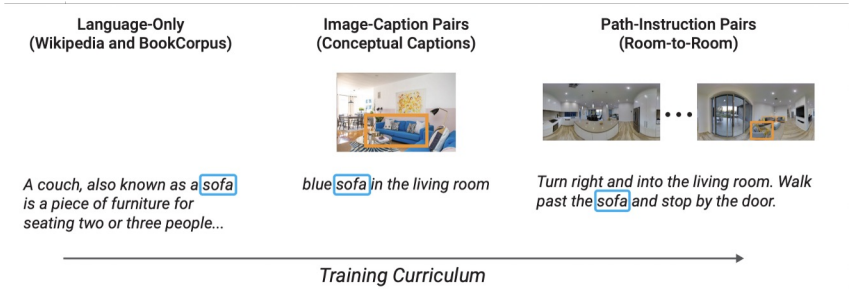
Vision-and-Language Navigation with LSTM

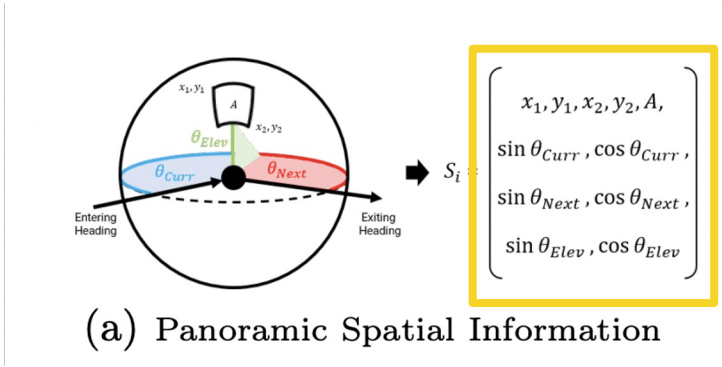


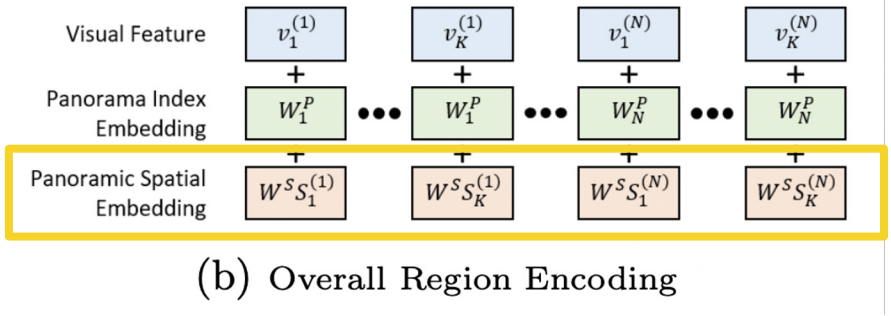


Language and Vision and “Time”



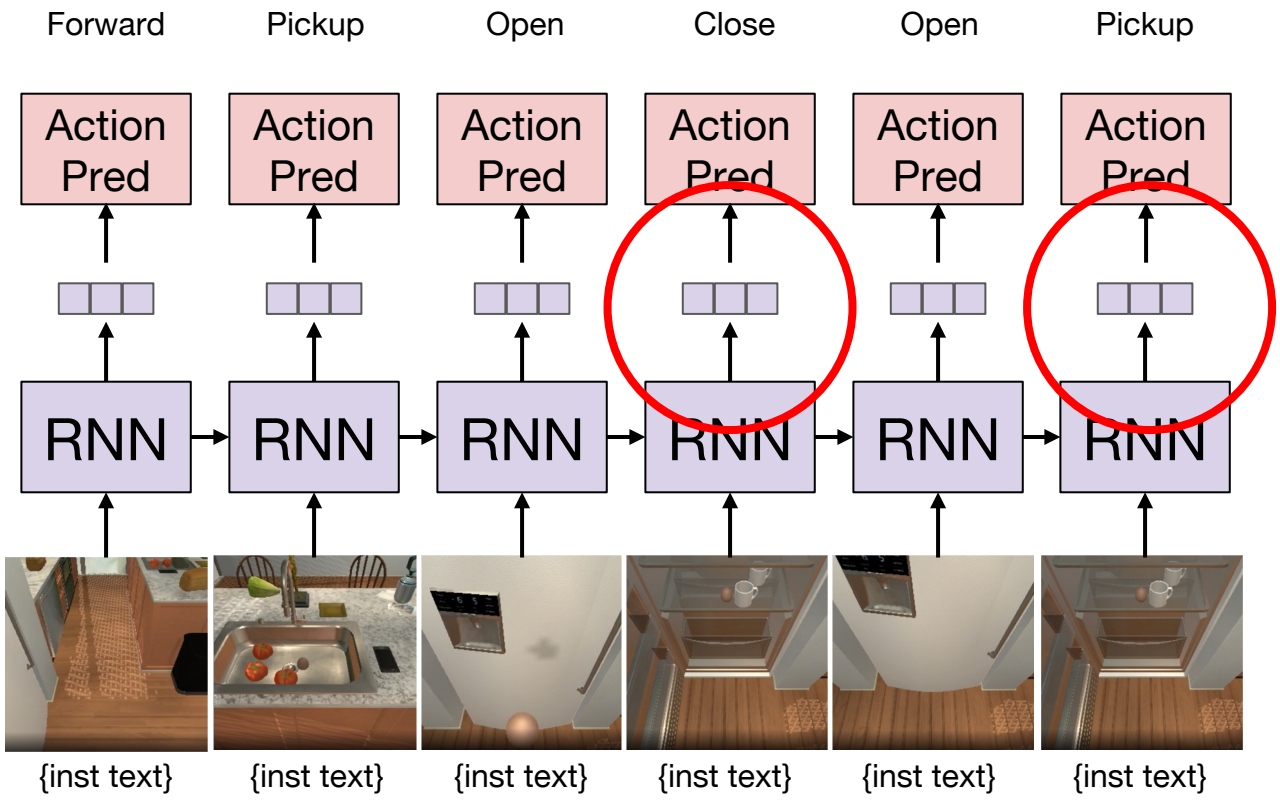




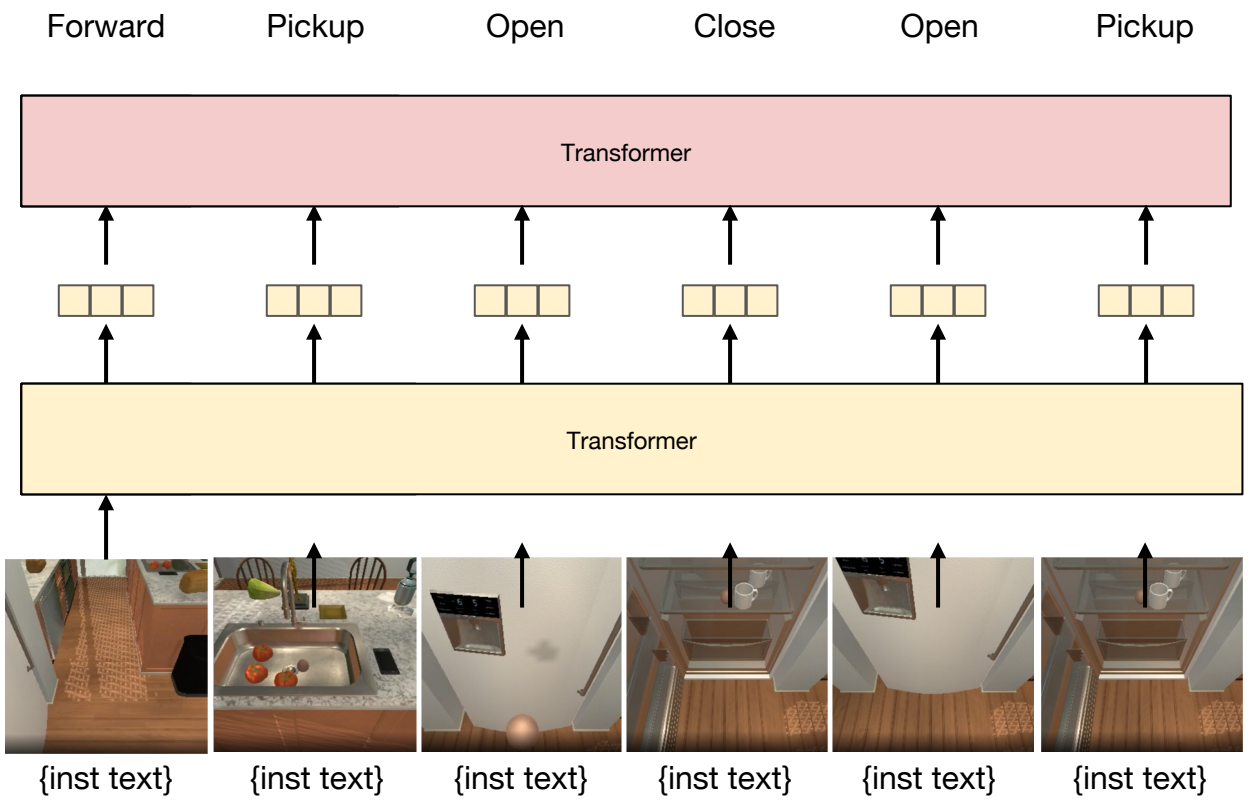


[Improving Vision-and-Language Navigation with Image-Text Pairs from the Web]

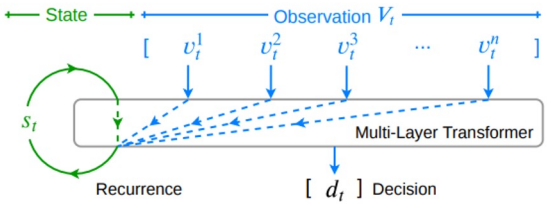
The *quality* of your learned state can be crucial

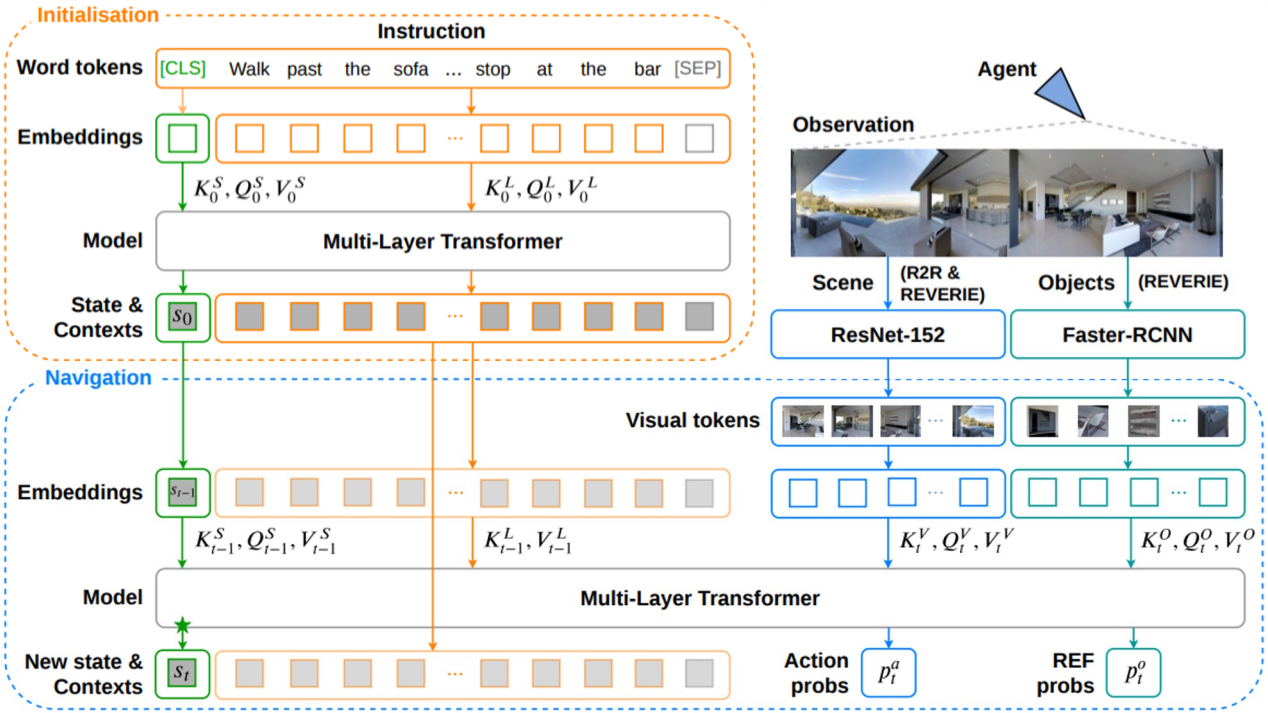


Without States, Our Transformer has to Learn So Much!



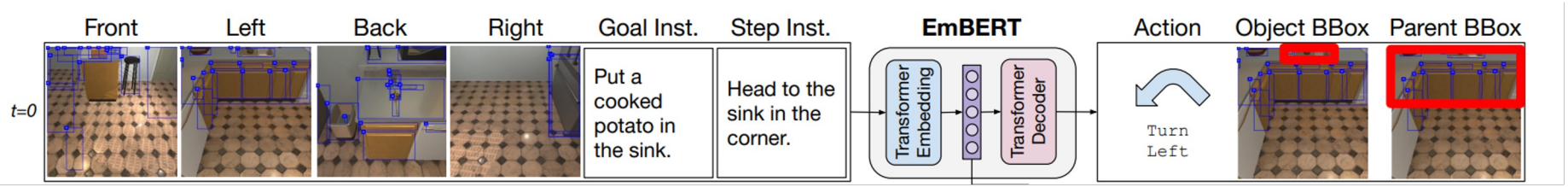
Stateful Transformers: VLN

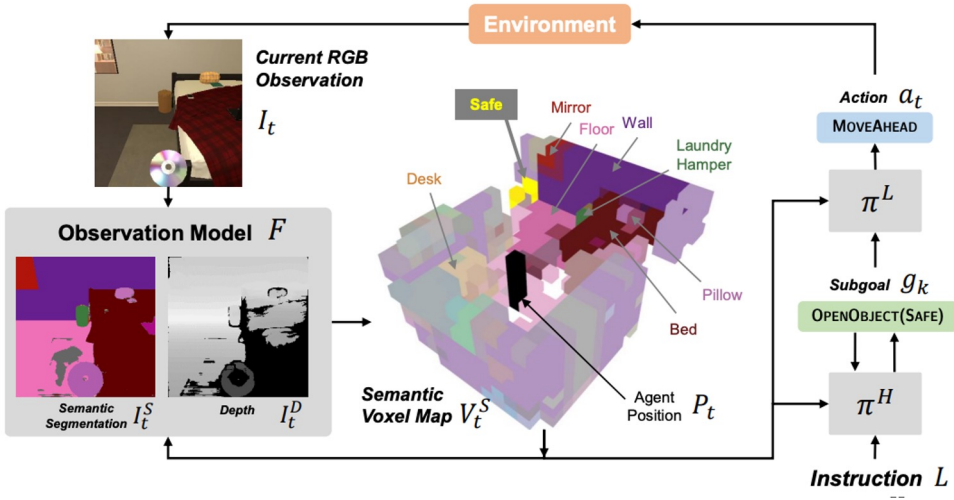


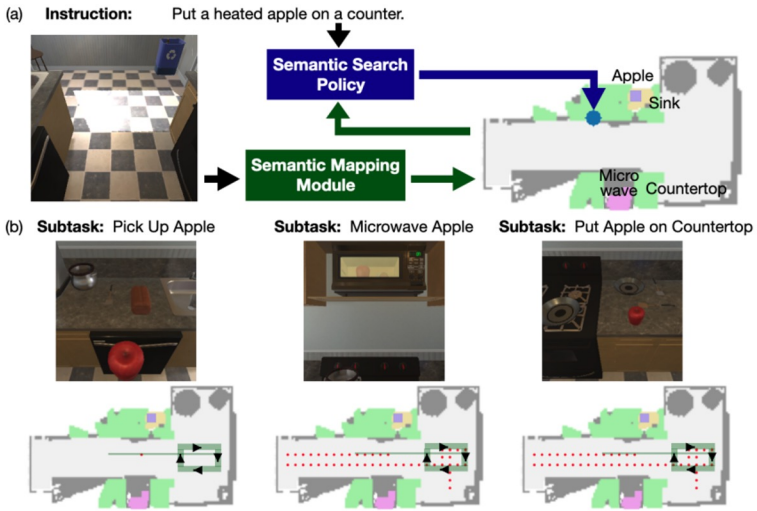


https://arxiv.org/abs/2011.13922

Implicit and Explicit States







https://arxiv.org/abs/2108.04927 ; https://arxiv.org/abs/2107.05612; https://arxiv.org/abs/2110.07342

Deep Learning for Agents

- So far, we've talked about situations where we have *demonstrations* available as training data
  - Enables us to perform learning from demonstrations / imitation learning / supervised learning
- What if we have no demonstrations, just a binary signal that tells us something like "yep, you completed the goal" or "nope, it's not done yet"?
- Then we move into the realm of *reinforcement learning*

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Lecture 9 Recap~~
  - Any questions before we move on?
- Deep Reinforcement Learning
- April 7 Project Roleplaying Breakouts

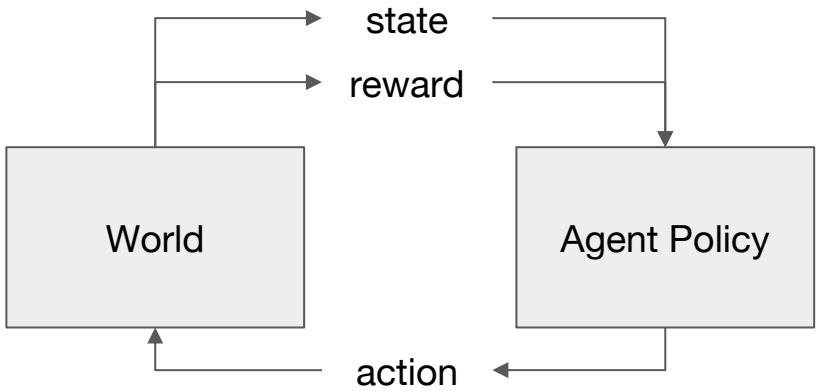
Markov Decision Process with Rewards

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

- MPD assumption: the *state* captures everything we need to know about what the optimal *action* should be
- Note *rewards* could be defined for every *(state, action)*
  - In practice, we'll need to estimate the quality of states, state-action pairs, etc., especially when reward is *sparse*
- Note that we allow for a probabilistic *transition function*

Markov Decision Process with Rewards

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

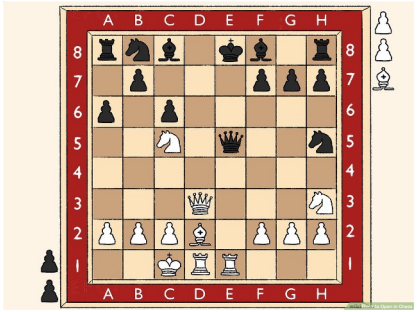


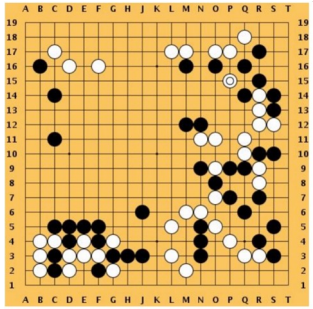
Goal: learn a policy that maximizes cumulative reward

Example - Atari Games

- Objective: Win the game (with scores as high as possible)
- State?
  - Pixels on the screen completely determine current state
- Actions?
  - Game controls (e.g. U, D, L, R, Shoot)
- Reward?
  - Score delta between t-1 and t for action taken at time t-1

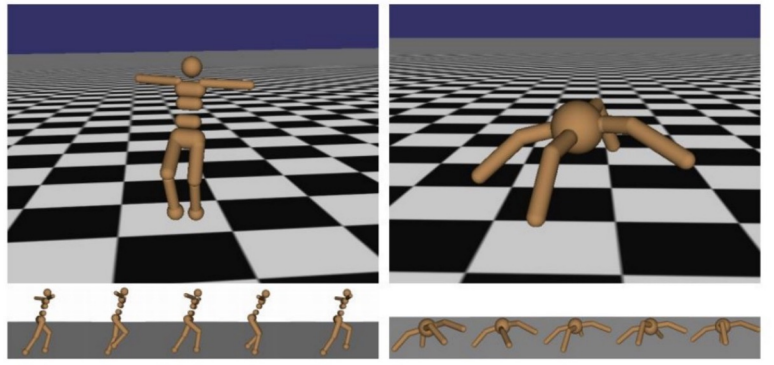
Example - Chess / Go





- Objective: Win the game
- State?
  - Positions of game pieces both on and off the board
- Actions?
  - Admissible game moves
- Reward?
  - Sparse: 1 if win, -1 if lose. Dense: heuristics (e.g., capture)

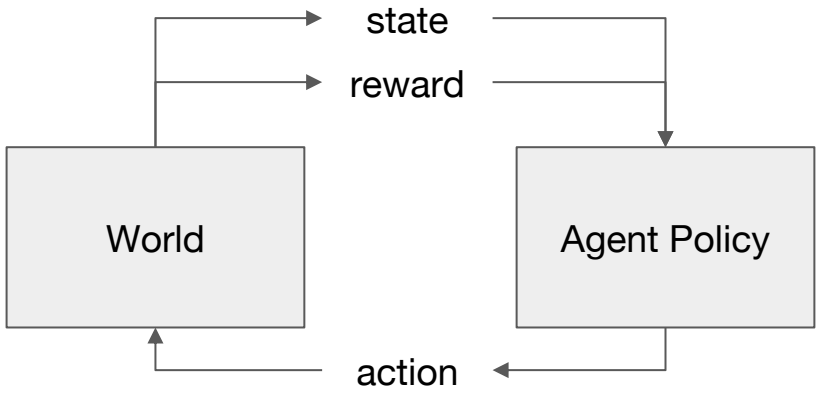
Example - Control



- Objective: Cover ground by walking
- State?
  - Position and angles of the joints
- Actions?
  - Torques applied on joints
- Reward?
  - Increases in distance from origin at each timestep

Markov Decision Process with Rewards

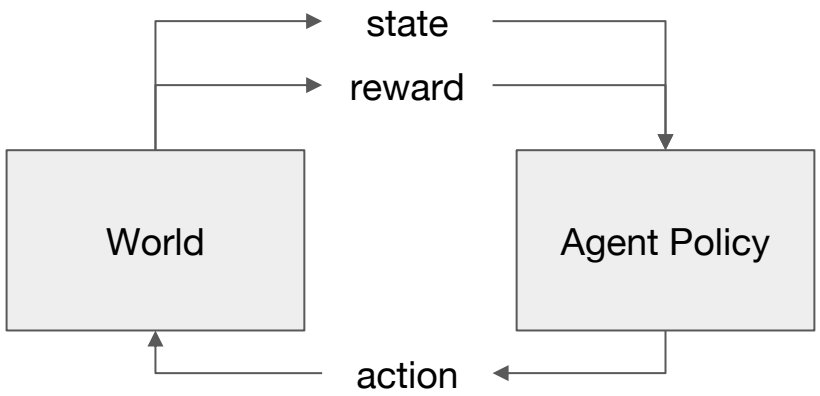
$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor



- At $t=0$, get initial state s_0
- Get action a_t from policy $\pi(s_t;\theta)=a_t$
- World samples next state s_{t+1} from transition function $P(s_t,a_t)=s_{t+1}$
- Agent receives reward $r_t=R(s_t,a_t)$
  - Can also be defined as $r_t=R(s_{t+1})$

Markov Decision Process with Rewards

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor



- Optimization Goal: learn policy parameters θ to maximize *cumulative discounted reward*
  - For a given *trajectory* with sequence of rewards *r*: $\Sigma_t \gamma^t r_t$

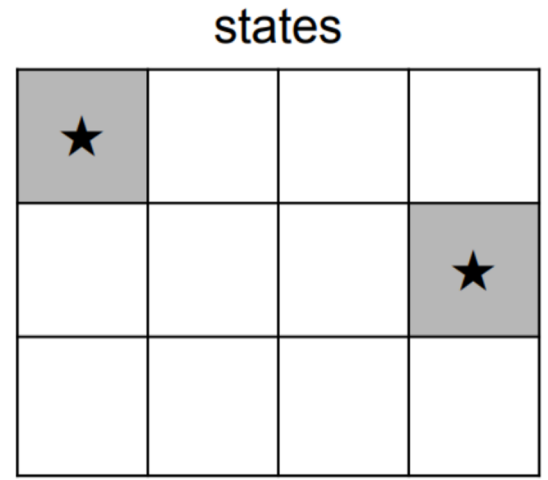
Simple MPD Example And Solution

- Transition?
  - Deterministic
- Rewards?
  - +10 for reaching star
  - -1 for each action

actions = {

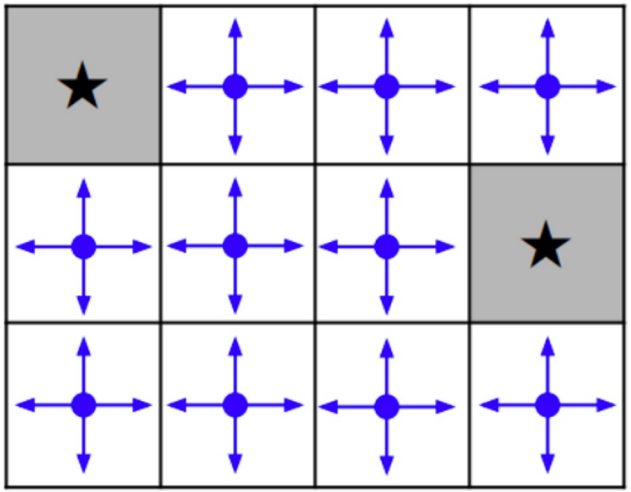
1. right
2. left
3. up
4. down

}

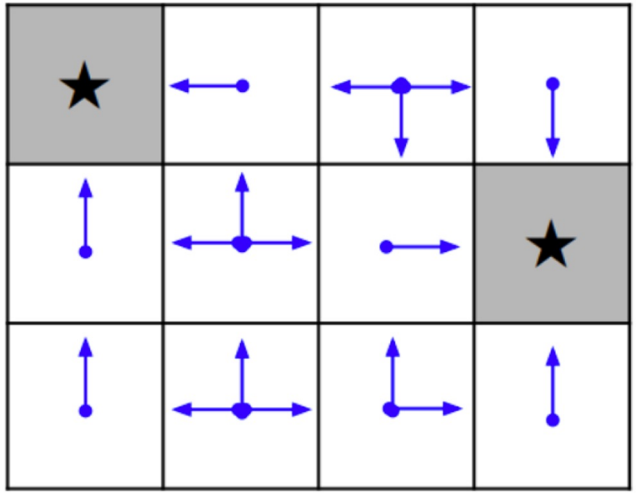


Objective: Reach stars (either one) with least number of actions

Simple MPD Example And Solution



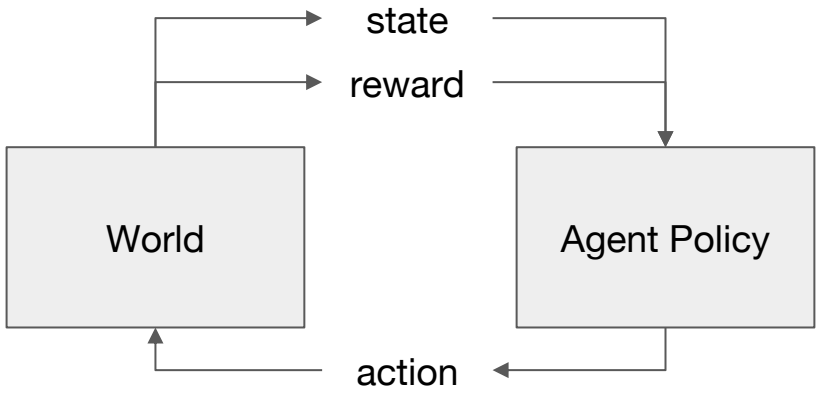
Random Policy



Optimal Policy

Markov Decision Process with Rewards

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor



$$s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$$

Seeking An Optimal Policy

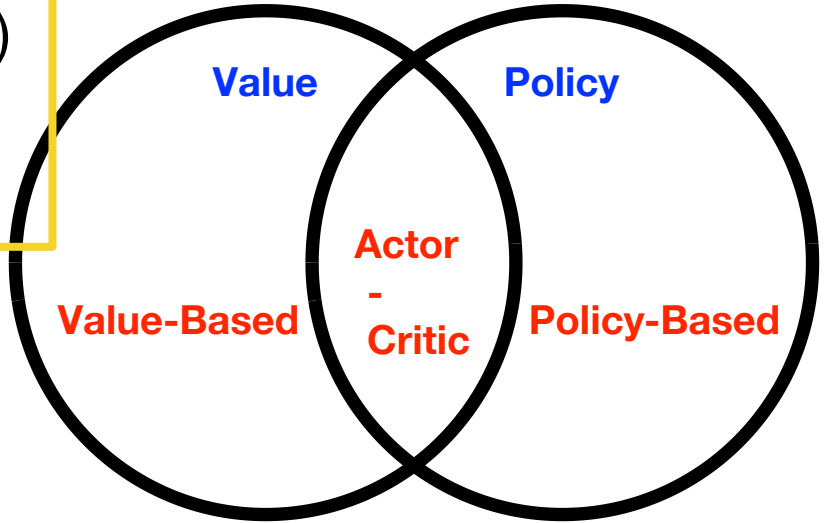
- Suppose we have a set of sampled trajectories
- A trajectory is defined by the tuple *(s, a, r)* through time
  - E.g., $<(s_0, a_0, r_0), (s_1, a_1, r_1), (s_2, a_2, r_2), \ldots>$
- One approach we can take is to estimate the *quality* of (*state, action)* pairs in terms of the eventual expected reward
- *Q-value* function:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- The expected cumulative reward if following the policy at state s, taking the action a

Seeking An Optimal Policy

- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy

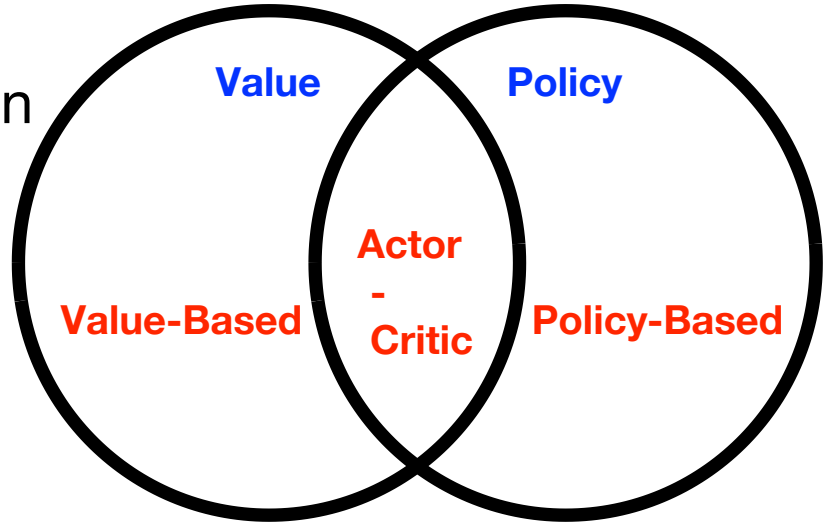


Seeking An Optimal Policy

- Valued-based: a function that measures how valuable action *a* is at state *s*
  - Indirectly yields a policy

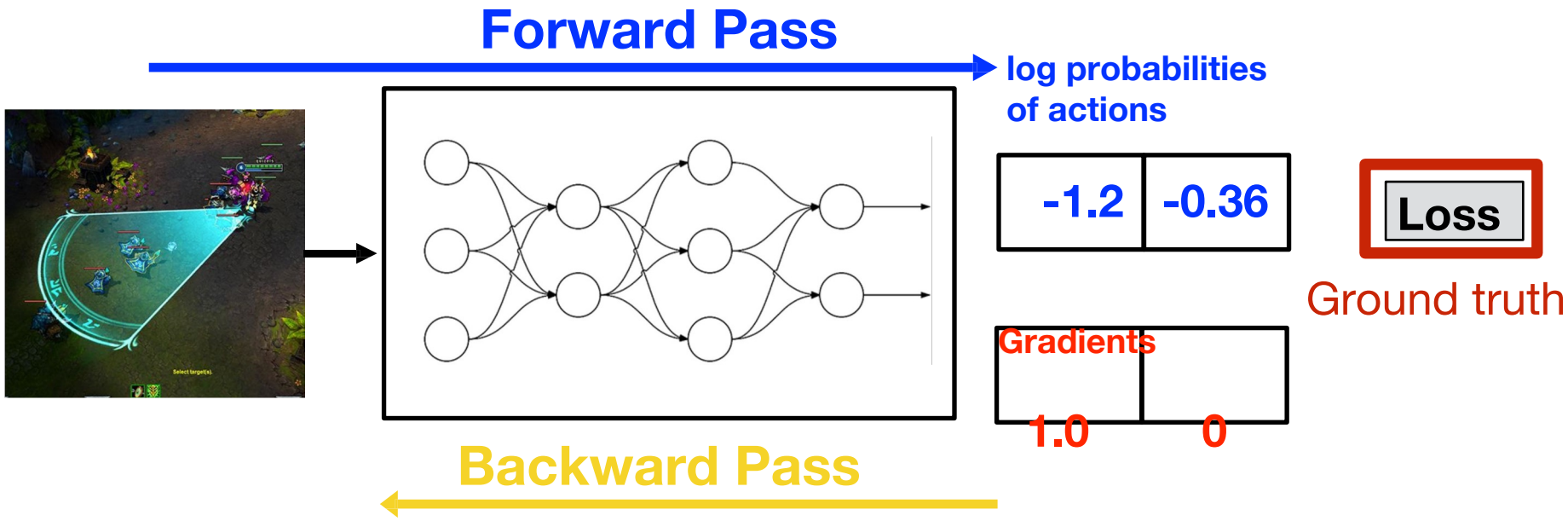
$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^{t} r_t | s_0 = s, a_0 = a, \pi\right]$$

- Policy-based: a function that measures the probability of action *a* leading to the best final outcome from a given state *s*
  - Directly yields a policy



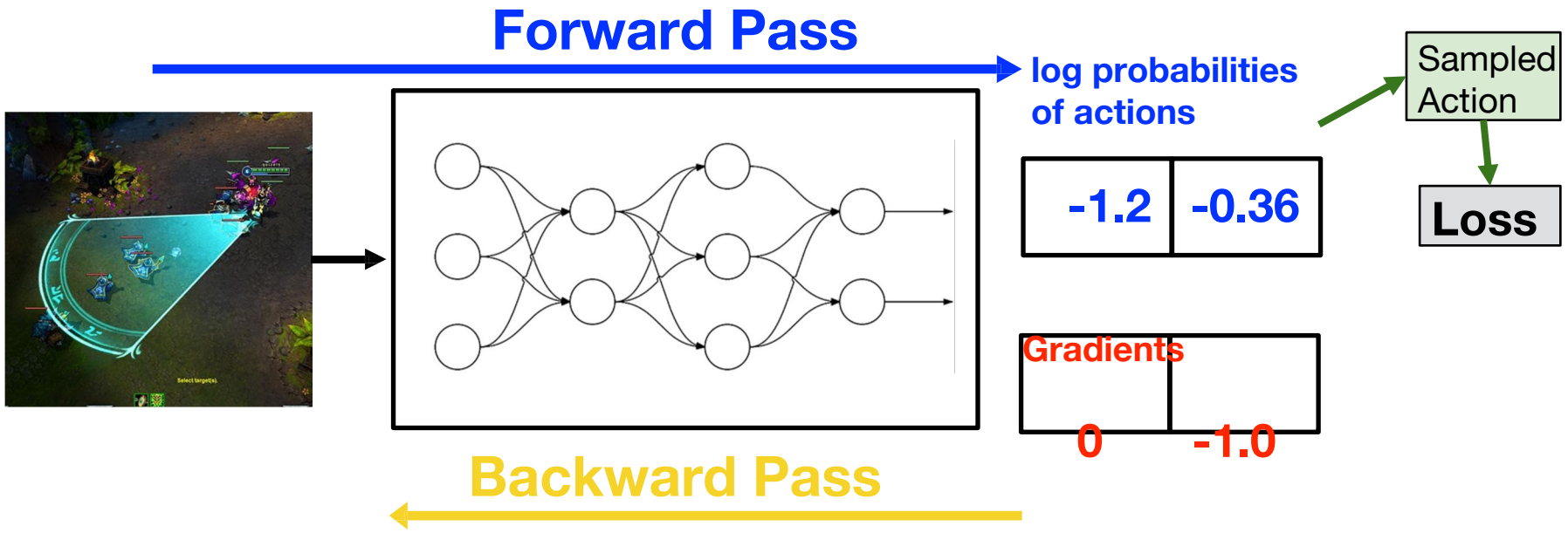
$$\theta^{*} = \underset{\theta}{\operatorname{argmax}}\, J(\theta) \qquad J(\theta) = \mathbb{E}_{\pi(\theta)}\left[\sum \gamma^{t} r_t\right]$$

Supervised Learning versus Policy Gradient



Learn: π: $S \rightarrow A$ for π parameterized by θ

Supervised Learning versus **Policy Gradient**

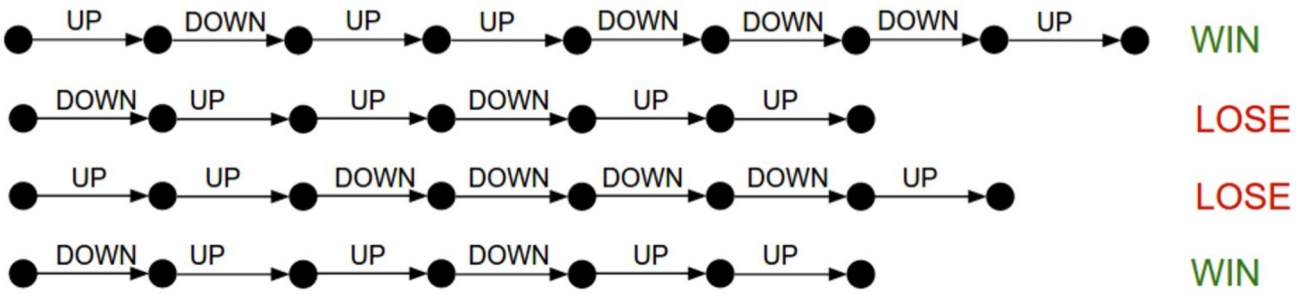


Learn: π: $S \rightarrow A$ for π parameterized by θ

Eventual Reward= -1.0

Policy Gradient

- Given a (possibly random) policy, run it for several iterations
- See what sticks!
- Upweight *(state, action)* transitions that lead to high eventual reward, downweight others
  - Policy Gradients



Policy Gradient

- Objective to optimize: $J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$
- Trajectory: $\tau = (s_0, a_0, r_0, s_1, ...)$
- Policy Gradient Theorem:
  - The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy π_θ .

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_\tau p(\tau;\theta) r(\tau)$$
$$= \mathbb{E}_{\tau \sim p(\tau;\theta)}[r(\tau) \nabla_\theta \log p(\tau;\theta)]$$

Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)} [r(\tau) \nabla_\theta \log p(\tau;\theta)]$$

Learning step

Gradient direction
(Scoring function)

Training a Network with Policy Gradient

- Monte-Carlo Policy Gradient (REINFORCE algorithm)
- Upshot:
  - 1) Initialize a (possibly random) policy
  - 2) Generate sample trajectories <(s, a, r)>
  - 3) Update reward estimates at each timestep based on discounted reward from eventual result
  - 4) Improve the policy by updating the parameters based on discounted rewards

Training a Network with Policy Gradient (REINFORCE)

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

$$\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$$

Write out the probability of the observed trajectory under the current policy

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

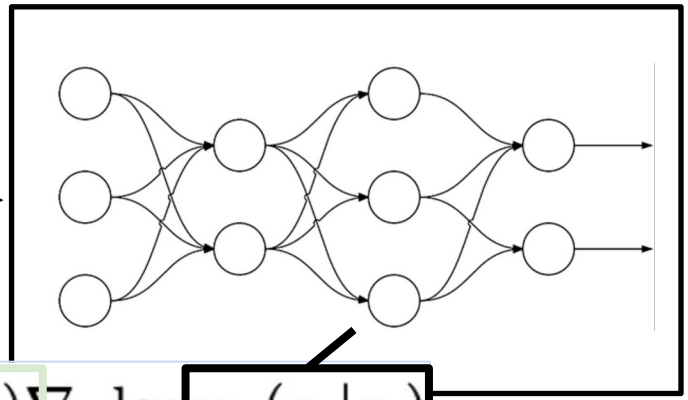
Differentiate with respect to policy parameters to get gradient

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Note that our estimate is now independent of the true state transition function that we can't observe directly!

Training a Network with Policy Gradient (REINFORCE)

Forward Pass



log probabilities of actions

-1.2	-0.36

Sampled Action

Loss

Gradients

0	-1.0

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Pass

Learn: π: $S \rightarrow A$ for π parameterized by θ

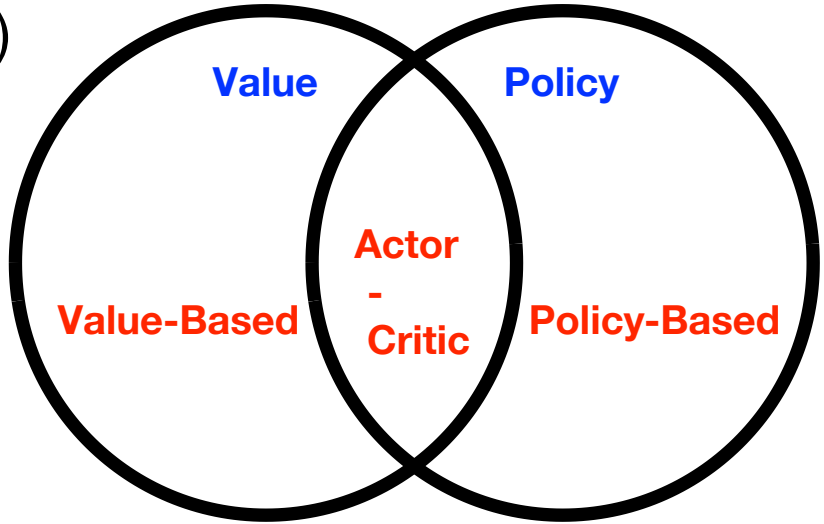
Eventual Reward= -1.0

Training a Network with Policy Gradient

- Very general method, but policy gradients suffer from high variance and so can require a lot of samples (trajectories per policy) and iterations (policy refinements)
- Policy gradient methods converge to a local minima that is often “good enough” for a task, but that convergence can be costly in terms of time and compute

Seeking An Optimal Policy

- Value Based (e.g., Deep Q-Network)
  - Learn Value Function
  - Implicit Policy
- Policy Based (e.g., guided policy)
  - No Value Function
  - Directly Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy

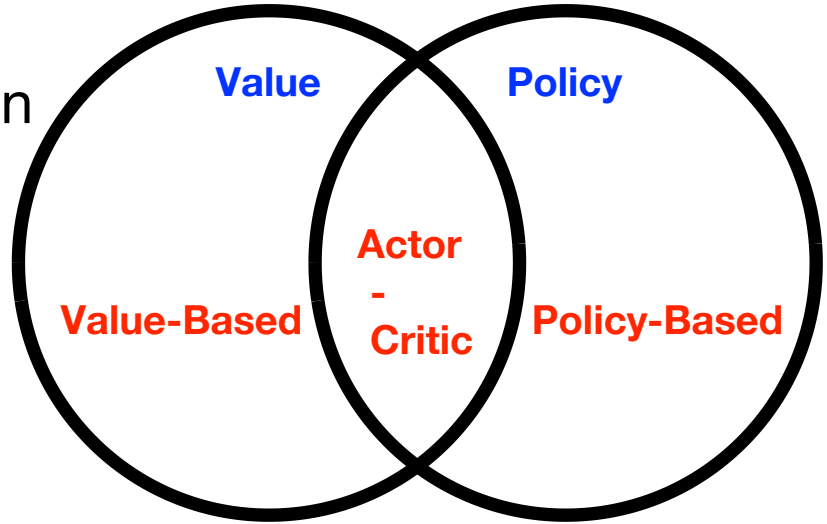


Seeking An Optimal Policy

- Valued-based: a function that measures how valuable action *a* is at state *s*
  - Indirectly yields a policy

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^{t} r_t | s_0 = s, a_0 = a, \pi\right]$$

- Policy-based: a function that measures the probability of action *a* leading to the best final outcome from a given state *s*
  - Directly yields a policy



$$\theta^{*} = \underset{\theta}{\operatorname{argmax}}\, J(\theta) \qquad J(\theta) = \mathbb{E}_{\pi(\theta)}\left[\sum \gamma^{t} r_t\right]$$

Q-value Function

- Want a function that measures how valuable action *a* is at given state *s*

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0}\gamma^{t}r_{t}|s_{0}=s,a_{0}=a,\pi\right]$$

- Optimal Q-value function:

$$Q^{*}(s,a) = \max_{\pi}\mathbb{E}\left[\sum_{t\geq 0}\gamma^{t}r_{t}|s_{0}=s,a_{0}=a,\pi\right]$$

  - The best expected cumulative reward at state *s* when taking the action *a* next

Bellman Equation

- Optimal Q-value function gives maximum expected return

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- Such an optimal function satisfies the optimality described by the *Bellman Equation*:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

- which in a nutshell says: the highest value of a (state, action) pair is the immediate reward plus the discounted best future reward from here

Value Iteration

- **Intuition:** Update the Q-value function iteratively

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

- Note similarity to *Bellman Equation*:

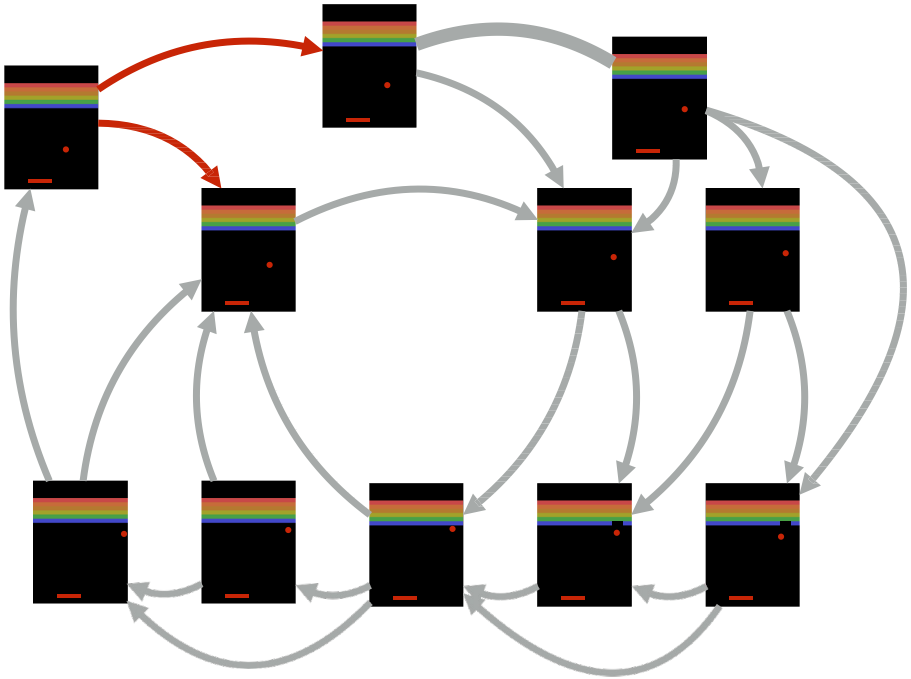
$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

- In expectation, such an iterative update *will converge* to the optimal estimate Q* if trajectories show us all *(s, a)* pairs enough times

Value Iteration

- Update: Q_0

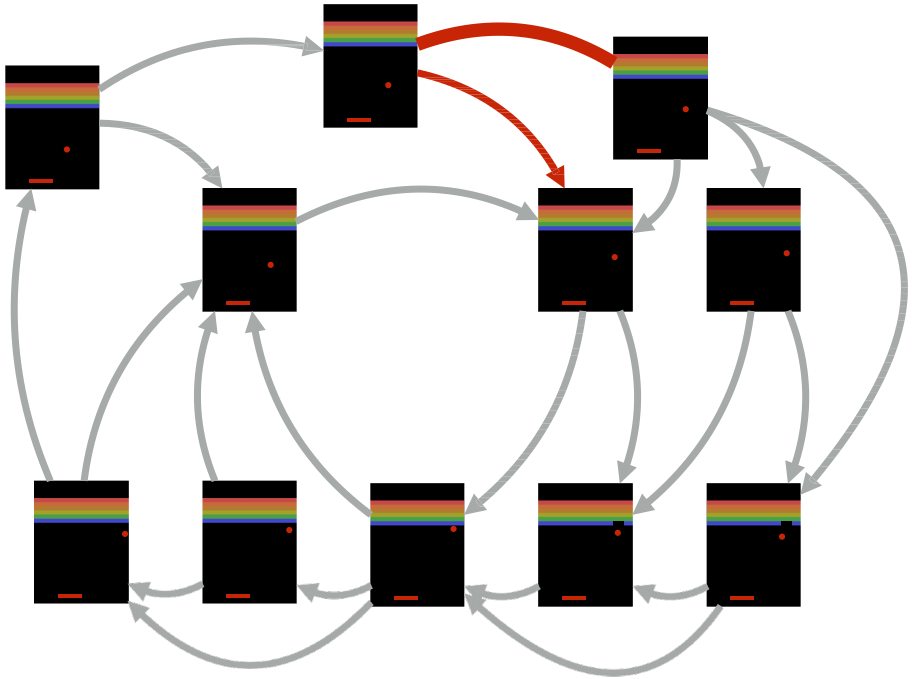
$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$



Value Iteration

- Update: Q_1

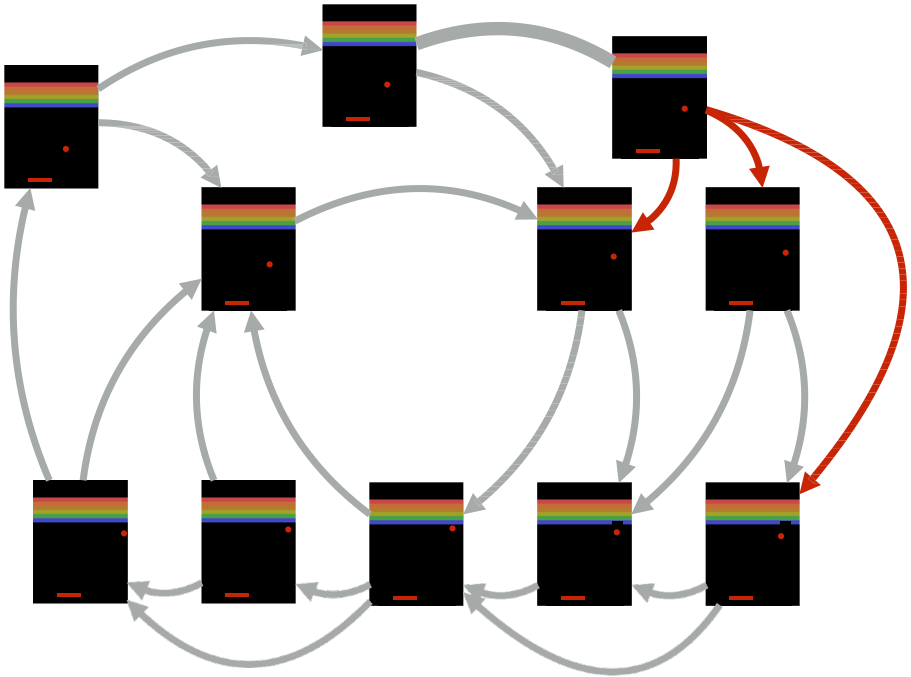
$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$



Value Iteration

- Update: Q_2

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$



Value Iteration

- **Intuition:** Update the Q-value function iteratively

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

- Getting an accurate estimate *Q*(s,a)* for every state-action pair is intractable for continuous state/action spaces and large discrete spaces too (e.g., all possible space invaders screens)
- What if we could *approximate* the Q function?

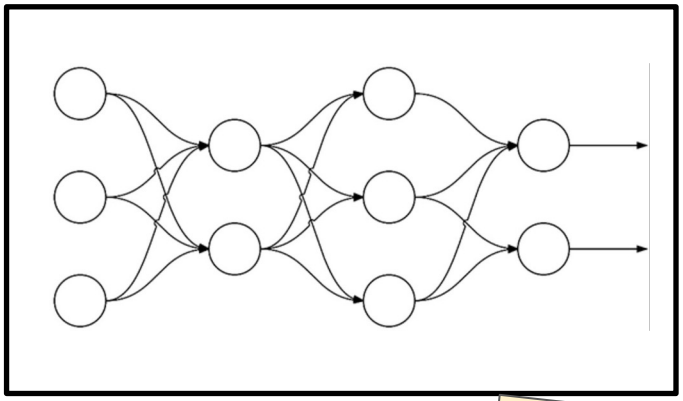
$$Q(s,a;\theta) \approx Q^*(s,a)$$

- What's a good way to do function approximation?

Training a Network with Value Iteration

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Forward Pass



Action Q-value estimates

.94	.87

Note: our network here is from states to actions still; why can it model Q(s,a;$\boldsymbol{\theta}$)?

Backward Pa

Sharing params among action Q value estimate networks; *what we're estimating depends on the loss function*

Training a Network with Value Iteration

$$Q(s, a; \theta) \approx Q^*(s, a)$$

Forward Pass

The Loss: $L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s, a; \theta_i))^2\right]$

"Ground truth" Q*

Approximation

$\mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})|s, a\right]$

Estimate via value iteration intuition

Backward Pass

Update $\boldsymbol{\theta}$: $\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))\nabla_{\theta_i} Q(s, a; \theta_i)\right]$

Overview of Today's Plan

- ~~Course organization and deliverables~~
- ~~Lecture 9 Recap~~
- ~~Deep Reinforcement Learning~~
    - Any questions before we move on?
- April 7 Project Roleplaying Breakouts

Action Items for You

- For those doing a Paper Roleplaying Breakout session today, turn in your 2-page paper role report by 11:59pm tonight through the gDrive submission form
- If your *final project presentation* is on April 21st (two weeks from today), turn in your project question + multiple choice answers this week or on Friday next (April 14th)
    - We'll post on Piazza to remind you about that

Paper Roleplaying Breakouts: April 7 [Schedule]

- High-Resolution Image Synthesis with Latent Diffusion Models **(Find Jesse outside THH 201)**
- Deep Residual Learning for Image Recognition **(Virtual with Bingjie)**
- U-Net: Convolutional Networks for Biomedical Image Segmentation **(Find Shihan in THH 201 front)**
- Language Model Cascades **(Find Tejas in THH 201 back)**
- The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks **(Virtual with Gautam)**

CSCI 566: Deep Learning and Its Applications

Jesse Thomason

Lecture 10: Deep Reinforcement Learning