Project Advice

- **Start simple; get fancier** ("anytime algorithm")
- **Pick a sensible evaluation metric** (and test set?)
- **Baselines**
  - Majority vote or mean
  - Logistic or linear regression
- **Sensible methods**
  - CNN, Random Forest …
- **Something clever**
  - Semi-supervised

Project Advice

- **Build on other's work**
  - Pretrained CNNs
  - Hugging face embeddings for NLP
  - …

RL Recitation

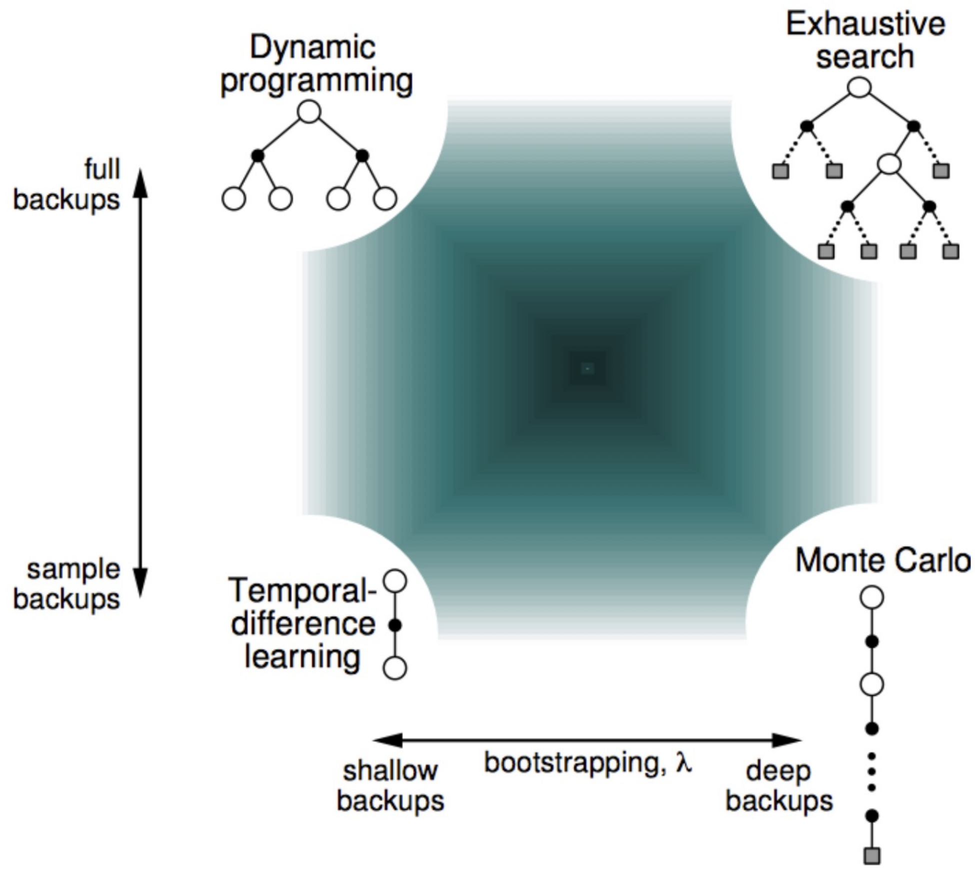
Lyle Ungar

RL Types

- **Model based**
  - Explicitly learn $p(s_{t+1}|s_t, a_t)$, $r(s_t, a_t)$
  - Markov Decision Process (MDP) or POMDP
- **Model free**
  - Learn expected value of each state, $V(s_t)$, *given a policy*
  - Learn expected value of each state and action, $Q(s_t, a_t)$
  - Learn an optimal policy, while learning V or Q
    - Can learn on- and off-policy

State can be discrete or real, *V* and *Q* can be neural nets

Which is model-based?



From David Silver UCL Course on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

RL – what to use when?

- **Model-based / model-free**
  - Model-based is nice in theory, but model-free is more widely used - why?
- **Episodic / infinite (discount factor)**
  - Immediate updates vs. end of episode (est. by Monte Carlo)
- **V-learning vs. Q-learning**
- **On-policy vs. Off-Policy**
  - Exploration vs. exploitation
- Imitation learning

Notation summary

- s_t **state**
- $\pi(a_t \mid s_t)$ **policy** π **and action** a_t
- $V_\pi(s_t)$ **estimated value of** s_t
- $Q_\pi(s_t,a_t)$ **estimated value taking action** a_t **in** s_t
- $r(s_t,a_t)$ **reward** (usually simply $r(s_t)=R_t$)
- $G_t\ (s_t)$ **expected discounted reward ('return')**
- γ **discount factor**
- $p(s_{t+1}|s_t,a_t)$ **model**

- **What is the relationship between** $V(s_t)$ **and** $G_t\ (s_t)$**?**
- **V is an approximation to G**

TD(0)

- **One could learn** V(s) **by updating it at the end of each game based on who won.**
- **Why update** *V(s)* **as soon as one makes a move and sees the opponent's response?**
- **What alternative is often used?**

TD(0)

- **One could learn** V(s) **by updating it at the end of each game based on who won.**
- **Why update** *V(s)* **as soon as one makes a move and sees the opponent's response?**
  - you learn immediately, and avoid the "noise" of all the moves between now and the end of the game
- **What alternative is often used?**
  - Monte Carlo "roll-out"

Model based vs. Model free

- **One can learn a model of the world** $p(s_{t+1}|s_t,a_t)$ **and use that to find an optimal policy**
  - How would one learn such a model?
- **Or one can learn the value** $V(s_t)$ **of each state - or of the action in each state** $Q(s_t,a_t)$ **- without a world model**
- **When is each better?**

A problem?

- $V(s)$ depends on π
- But π is a function of $V(s)$

A 0.812	B 0.868	C 0.918	Food 1.00
D 0.762		E 0.660	Shock -1.00
J 0.705	G 0.655	H 0.611	I 0.388

So how can we learn $V(s)$ and π^*?

Policy Iteration

Policy iteration (using iterative policy evaluation)

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\quad \Delta \leftarrow 0$
   $\quad$ For each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Assuming deterministic policy $\pi(s)$

From Sutton *Reinforcement Learning: An Introduction* (2016 draft)

Model based vs. Model free

- **One can learn a model of the world** $p(s_{t+1}|s_t,a_t)$ **and use that to find an optimal policy**
  - How would one learn such a model?
    - Just count how often you end up in each state, given the preceding state and action
- **When to use** $V(s_t)$ **vs** $Q(s_t,a_t)$**?**
  - People mostly use Q-learning, since it lets you easily find and optimal policy for a given model
  - And it lets you cleanly control exploration and exploitation

Bellman Equation

Bellman's Equation: Holds for all policies $\pi(a|s)$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}$$

The expected value of s

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

The expected value of (s,a)

Bellman Equation (Optimality)

Bellman's Optimality Equation: Holds for optimal policies $\pi^*(s)$

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a) \left[r + \gamma v_*(s')\right], \forall s \in \mathcal{S}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[r + \gamma \max_{a'} q_*(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

Can be used model-based or model-free

Bellman Equation (Q version)

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

$$Q(s,a) := r(s,a) + \gamma \max_{a} Q(s',a)$$

We still need to figure out how to adjust the policy - on policy or off policy

Look at each method for gridworld

- **Q values in each state**
  - State = A, B, C, ..
  - Action = l, r, u, d   (left, right, up, down)

e.g.
Q(A,r) = .8

A **Q(A,r)=.8** **Q(A,d)=.7**	**B** **Q(B,l)= .75** **Q(B,r)= .9**	**C** **Q(C,l)= .8** **r=.95; d=0.7**	**Food 1**
D Q(D,u)= .8 Q(D,d)= .7	XXXXXXXX XXXXXXXX XXXXXXXX	E Q(E,u)= .8 Q(E,d)= .65	Shock -1
J Q(J,u)= .7 Q(J,r)= .65	G Q(G,l)= .7 Q(G,r)= .6	H Q(H,l)= .65 r=.6; u=0.75	I Q(I,l) = .65 Q(I,u) = -.5

Dynamic Programing, γ=1

- **Start in B, with π^*, assume**
  - p(C|B,r)=0.9 P(A|B,r)=0.1 p(C|B,l)=0.1 P(A|B,l)=0.9
- **What is the new estimate of *Q(B,r)*?**

A **Q(A,r)=.8** **Q(A,d)=.7**	**B** **Q(B,l)= .75** **Q(B,r)= .9**	**C** **Q(C,l)= .8** **r=.95; d=0.7**	**Food 1**
D Q(D,u)= .8 Q(D,d)= .7	XXXXXXXX XXXXXXXX XXXXXXXX	E Q(E,u)= .8 Q(E,d)= .65	Shock -1
J Q(J,u)= .7 Q(J,r)= .65	G Q(G,l)= .7 Q(G,r)= .6	H Q(H,l)= .65 r=.6; u=0.75	I Q(I,l) = .65 Q(I,u) = -.5

Bellman's Equation

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} q_*(s')\right], \forall s \in \mathcal{S}, \forall a \in \mathcal{A}(s)$$

Q(B,r) = p(C|B,r)[0+1*Q(C,r)] + p(A|B,r)[0+1*Q(A,r)]
= 0.9 [0.95] + 0.1 [0.8]
= 0.935

TD(0) - Q-learning, γ=1, α=0.6

- **Start in *s=B*, pick best action *a=r***
- **Observe new state *s=C***
- **What is the new estimate of *Q(B,r)*?**

A **Q(A,r)=.8** **Q(A,d)=.7**	**B** **Q(B,l)= .75** **Q(B,r)= .9**	**C** **Q(C,l)= .8** **r=.95; d=0.7**	**Food 1**
D Q(D,u)= .8 Q(D,d)= .7	XXXXXXXXX XXXXXXX XXXXXXXX	E Q(E,u)= .8 Q(E,d)= .65	Shock -1
J Q(J,u)= .7 Q(J,r)= .65	G Q(G,l)= .7 Q(G,r)= .6	H Q(H,l)= .65 r=.6; u=0.75	I Q(I,l) = .65 Q(I,u) = -.5

TD(0)

- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma Q\big(s_{t+1}, \pi(s_{t+1})\big) - Q(s_t, a_t)\right)$
- $Q(B, r) \leftarrow Q(B, r) + \alpha\big(0 + 1Q(C, r) - Q(B, r)\big)$
- $Q(B, r) \leftarrow 0.75 + 0.6 * (0 + 0.95 - 0.75)$
- $Q(B, r) = 0.87$

MC Q-learning, γ=1, α=0.6

- Start in *s=B,* pick best action *a=r*
- Observe new state *s=C*
- What is the new estimate of *Q(B,r)?*

A **Q(A,r)=.8** **Q(A,d)=.7**	**B** **Q(B,l)= .75** **Q(B,r)= .9**	**C** **Q(C,l)= .8** **r=.95; d=0.7**	**Food 1**
D Q(D,u)= .8 Q(D,d)= .7	XXXXXXXXX XXXXXXX XXXXXXXX	E Q(E,u)= .8 Q(E,d)= .65	Shock -1
J Q(J,u)= .7 Q(J,r)= .65	G Q(G,l)= .7 Q(G,r)= .6	H Q(H,l)= .65 r=.6; u=0.75	I Q(I,l) = .65 Q(I,u) = -.5

MC Q-learning

- $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_t + \gamma V^{\pi}(s_{t+1}) - Q(s_t, a_t)\right)$
- Use MC roll-out to estimate $\mathrm{V}(\mathrm{s}_{\mathrm{t+1}})$

Monte Carlo Q-learning, γ=1, α=0.6

- **Do 3 rollouts from *C*, taking action *r* every time**

1) C →Food

3) C → E → Shock

2) C → Food

- **3 rollouts give** V(C) = (1+1-1)/3 = 2/3

A Q(A,r)=.8 Q(A,d)=.7	**B Q(B,l)= .75 Q(B,r)= .9**	**C Q(C,l)= .8 r=.95; d=0.7**	**Food 1 Q(F,-)=1**
D Q(D,u)= .8 Q(D,d)= .7	XXXXXXXXX XXXXXXX XXXXXXXX	E Q(E,u)= .8 Q(E,d)= .65	Shock Q(S,-)=-1
J Q(J,u)= .7 Q(J,r)= .65	G Q(G,l)= .7 Q(G,r)= .6	H Q(H,l)= .65 r=.6; u=0.75	I Q(I,l) = .65 Q(I,u) = -.5

MC Q-learning

- $Q(s_t, a_t) \leftarrow \; Q(s_t, a_t) + \alpha\big(r_t + \gamma V^{\pi}(s_{t+1}) - Q(s_t, a_t)\big)$
- Use roll-out to estimate $V(s_{t+1}{=}C) = 2/3$
- $Q(B, r) \leftarrow \; Q(B, r) + \alpha\left(0 + 1\,(\frac{2}{3}) - Q(B, r)\right)$
- $Q(B, r) \leftarrow 0.75 + 0.6 * (0 + 0.66 - 0.75)$
- $Q(B, r) = 0.7$

Following not yet covered

DQN (TD(0)), γ=1 , α=0.6

- **Now move to a real space.**
- Represent every state *i* by its location $\mathbf{x}_i$
  - A = (1,1), B=(1,2), D= (2,1),…
- Assume a linear $Q(s_i,a_j) = \mathbf{w}_j^T\mathbf{x}_i$ so every action *j* has a $\mathbf{w}_j$
- Initialize all $\mathbf{w}_j$ = (1,1);
- Start in state A, take action r, end up on B.
- What is the updated value of Q(A,r)?

A Q(A,r)=.8 Q(A,d)=.7	**B Q(B,l)= .75 Q(B,r)= .9**	**C Q(C,l)= r=.95; c**
D Q(D,u)= .8	XXXXXXXXX XXXXXXX	E Q(E,u)=

Deep Q-Learning

$$Argmin_{\theta} \left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_{a} Q(s',a;\theta) \right) \right]^2$$

Which Q() are we updating?
How do we update it?

MC-DQN, γ=1 , α=0.6

- **Still in the real space.**
  - Again, represent every state *i* by its location x_i
- **Again, assume linear model**
  - $q(s_{i,}\ a_j) = \ \boldsymbol{w}_j^T\boldsymbol{x}_i$
  - Initialize all weights to (1,1);
- Start in state A, follow policy π 3 times,
  - pick r every time
  - End up twice with Food, once with Shock
- What is the updated value of $q(A_, r)$?

A Q(A,r)=.8 Q(A,d)=.7	B Q(B,l)= .75 Q(B,r)= .9	C Q(C r=.9
D Q(D,u)= .8	XXXXXXXXX XXXXXXX	E Q(E,

AlphaZero-style, γ=1 , α=0.6

- **Assume linear models**
  - $\pi(s_i)$ = *softmax*$(\mathbf{w}_a^T \mathbf{x}_i)$, value $V(s_i) = \mathbf{w}^T \mathbf{x}_i$ $\quad \mathbf{w}_a = \mathbf{w} = (1,1)$;
- Start in state A, follow policy π 3 times,
  - Pick r every time
  - End up twice with Food, once with Shock
- What is the updated value of V(A)?
  - What is V(A)?
  - What is the formula for updating it?

AlphaZero loss function

NNet: $(\mathbf{p}, v) = f_\theta(s)$

- Minimizes the error between the predicted outcome (value function) *v(s)* and the actual game outcome *z*
- Maximizes the similarity of the policy vector **p**(s) to the MCTS probabilities π*(s)*.
- L2 regularize the weights θ

$$l = (z - v)^2 - \pi^{\mathrm{T}} \log \mathbf{p} + c\|\theta\|^2,$$

Q-Learning

$$Q(s,a) := r(s,a) + \gamma \max_a Q(s',a)$$

How might we pick the policy?

Pure greedy: argmax$_a$ Q(s,a)
ε-greedy
Using an older network for Q
Using a policy network (maybe a fast one)
Preferring actions that have been taken less

Deep Q-Learning (DQL)

$$Argmin_{\theta} \left[Q(s,a;\theta) - \left(r(s,a) + \gamma \max_{a} Q(s',a;\theta) \right) \right]^2$$

Represent *Q* with a neural net