Reinforcement Learning Cheat Sheet

- Part 1: Watch the Alpha Go Movie. Deep Mind. **Reinforcement Learning** - learning to make a good sequence of decisions. Atari, Robotics, Education, Healthcare, NLP, Computer Vision all use RL. 4 components of RL: **optimization** - about yielding best outcomes, **delayed consequences** - planning, sacrificing reward now for reward later, **exploration** - you don't know the consequences of decisions not taken so you need to learn from experience, **generalization** - programming policies to act in certain ways given certain situations. **AI planning** - involves optimization, delayed consequences, generalization (you know the rules). **supervised and unsupervised learning** - only optimization and generalization, just that unsupervised learning doesn't have labels. **imitation learning** - learns from experience of others, involves optimization, delayed consequences, and generalization, assumes an input of good policies, reduces RL to supervised learning. **Issues with RL** - having the right rewards, robustness vs risk sensitivity (exploration vs reward trade-off), multi-agent RL. **sequential decision making** - cycle of: agent $\rightarrow$ (action a_t) $\rightarrow$ world $\rightarrow$ (observation o_t, reward r_t) $\rightarrow$ agent, goal is to select actions to maximize total future reward, may need to balance immediate and long term rewards, strategic behavior for high rewards, not the **discrete time step** is t. **history** - history of A, O, R that agent uses to make decisions. **world state** - full state of the world. **agent state** - state of world that agent needs to make decisions. **markov state** - state s_t is Markov iff $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|h_t, a_t)$ i.e. that the future is independent of the past given the present. Setting $s_t = h_t$ allows any world to be markov. In practice, most recent observation is sufficient statistic of history i.e. setting $s_t = o_t$. **full observability / MDP markov decision process** - agent state same as world state **partial observability / POMDP partially observable MDP** - agent constructs its own state i.e. $s_t = h_t$, beliefs, RNN, etc (poker players only sees own cards, healthcare doesn't see all physiological processes). **bandits seq decision process** - actions have no influence on next observations, no delayed rewards. **deterministic** - given history and action, single observation and reward. **stochastic** - given history and action, many possible (probability distribution of) observations and reward. **model** - agent's understanding (model) of how the world changes in response the agent taking an action. **policy** - map from agent state to action to take. **value function** - future rewards of being in a state and/or action when following a particular policy. **transition / dynamics model** - predicts agent next state $P(s_{t+1} = s'|s_t = s, a_t = a)$. **reward model** - predicts immediate reward $R(s_t = s, a_t = a) = E[r_t|s_t = s, a_t = a]$ (reward will depend on which state one probabilistically ends up at, that's why it's an expectation?). **deterministic policy** - $\pi(s) = a$. **stochastic policy** - $\pi(a|s) = P(a_t = a|s_t = s)$. **value function** - $V^\pi(s_t = s) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s]$, **discount factor** γ weights immediate vs. future rewards, expectation is taken over all the different paths that can be taken from the state s by the policy π. Types of RL agents (what the agent/algo learns) - **value based**: explicitly learns value function, implicitly learns policy; **policy based**: explicitly learns policy, there is no value function; **actor critic**: explicitly learns policy, explicitly learns value function. Types of RL agents - **model based**: has an explicit model, may or may not have a policy or value function; **model free**: explicit value function and/or policy function, no model. **planning** - algo computes how to act given model of world. **RL** - agent doesn't know how world works, interacts with world to explicitly/implicitly learn how it works, improves its policy. **exploration vs exploitation** - as the trade-off sounds. **evaluation** - estimate/predict expected rewards from following a given policy. **control** - optimization to find the best policy.

- Part 2: MDPs can model a huge number of interesting problems and settings. **bandits** - use single state MDPs. **optimal control** - mostly about continuous state MDPs. **POMDP** - state is history. **markov process / markov chain** - memoryless random process, sequence of random states with markov property, S - finite set of states, P - transition model, no rewards, no actions. **markov reward process** - S (finite), P, R - reward function, γ - discount factor, no actions. **horizon** - number of time-steps in each episode in a process (can be a **finite** or infinite MRP). **return G_t of MRP** - discounted sum of rewards from time-step t to horizon $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...$ Note that $V(s) = E[G_t|s_t = s]$ (expectation over all paths that start at state s). **computing value of MRP empirically** - can simulate large number of episodes and average the returns. **bellman equation** - markov property of MRPs yields the following: $V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$, i.e. the value at a state is the sum of *the immediate reward* and *the discounted sum of future rewards.* **matrix form bellman equation for finite state MRP** - $V = R + \gamma PV$. **analytic solution for value of MRP** - $V = (I - \gamma P)^{-1}R$, can be computed in $O(n^3)$. **dynamic programming algo for computing value of an MRP** - (1) initialize $V_0(s) = 0$ for all s (2) for $k = 1$ until convergence, for all $s \in S$, $V_k(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V_{k-1}(s')$, algo is $O(s^2)$ for each t. In finite horizon case, V_k^π is the exact value of k-horizon value of state s under policy π. In the infinite horizon case, $V_k^\pi(s)$ is an estimate of $E_\pi[r_t + \gamma V_{i-1}|s_t = s]$ **markov decision process MDP** - is a tuple: (S [finite], A - actions [finite], P, R, γ). **MDP + $\pi(a|s)$ is a MRP** - (S, R^π, P^π, γ) where $R^\pi(s) = \sum_{a \in A} \pi(a|s)R(s, a)$ and $P^\pi(s'|s) = \sum_{a \in A} \pi(a|s)P(s'|s, a)$. **modification of previous iterative algo for computing value of MDP + policy** - (1) initialize $V_0(s) = 0$ for all s (2) for $k = 1$ until convergence, for all $s \in S$, $V_k^\pi(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s)V_{k-1}^\pi(s')$ – here a **bellman backup** for the policy π is applied. **MDP Control** - compute optimal policy: $\pi^*(s) = \underset{\pi}{\text{argmax}} V^\pi(s)$, the optimal policy for an MDP in an infinite horizon problem is deterministic, stationary (does not depend on time step), not necessarily unique. **policy search** - through enumeration searches $|A|^{|S|}$ deterministic policies. **state-action value of a policy** - take action a, then follow policy: $Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P^\pi(s'|s, a)V^\pi(s')$. **policy iteration PI** - (1) i=0; init $\pi_0(s)$ randomly for all states s (2) while i==0 or $|\pi_i - \pi_{i-1}| > 0$, do policy evaluation, policy improvement, i=i+1. **policy evaluation** - compute the value of π_i using iterative algo (equivalent to applying Bellman repeatedly till convergence). **policy improvement** - $\pi_{i+1}(s) = \underset{a}{\text{argmax}} Q^{\pi_i}(s, a) \forall s \in S$. We did this improvement step because we know $\max_a Q^{\pi_i}(s, a) >= V^{\pi_i}(s)$, but still that only suggests that following π_{i+1} for one action and then following π_i is better than just following π_i. However, it can be proved (by recursively plugging in the definition of Q) that π_{i+1} provides **monotonic**

improvement ($V^{\pi_{i+1}} >= V^{\pi_i}$ aka $V^{\pi_{i+1}}(s) >= V^{\pi_i}(s) \forall s \in S$) over π_i. PI can take at most $|A|^{|S|}$ iterations. **bellman backup** - applied to a value function, improves it if possible: $BV(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$ **value iteration VI** - considering longer and longer episodes to improve value function: (1) init $V_0(s) = 0 \forall s$ (2) set k = 1 (3) loop until finite horizon, convergence: (3a) for each state s, $V_{k+1}(s) = \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_k(s')$ (essentially doing a bellman backup $V_{k+1} = BV_k$) (3b) $\pi_{k+1}(s) = \text{argmax}_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V_k(s')$. **bellman for a particular policy** - $B^\pi V(s) = R^\pi(s) + \gamma \sum_{s' \in S} P^\pi(s'|s)V(s')$, for policy evaluation repeatedly apply B^π, i.e. $V^\pi = B^\pi B^\pi ... B^\pi V$. **contraction operator** - $|OV - OV'| <= |V - V'|$. Bellman backup is a contraction on V, $|V - V'|$ is the **infinity norm**, the max difference over states $max(s)|V(s) - V'(s)|$. In VI for finite horizon k, optimal policy in general is not stationary (depends on time step). **VI vs PI** - VI: compute optimal policy for horizon=k and increment k, PI: compute infinite horizon value of policy (policy eval), use it to select a better policy (policy improvement).

- Part 3: G_t, $V^\pi(s)$, and $Q^\pi(s,a)$ can be defined with respect to taking a particular policy π. **bootstrapping** - in the dynamic programming algo for policy eval, when you use a value estimate (in cache) for the future value V_{i-1}. **Monte Carlo policy evaluation** - generate a number of trajectories (state action paths till episode terminates) following policy π, average their returns to create a value estimate for the state – doesn't need MDP dynamics/rewards?, no bootstrapping, state doesn't have to be Markov, only for episodic MDPs. **first visit MC on policy eval** - (1) after each episode i (1.1) define $G_{i,t}$ as the return from timestep t onwards in the i^{th} episode (1.2) for each state s visited in episode i, for the first time t that state s is visited in episode i, increment counter of total first visits $N(s) = N(s) + 1$, increment total return $S(s) = S(s) + G_{i,t}$, update estimate $V^\pi(s) = S(s)/N(s)$ (2) By law of large numbers, as $N(s) \Rightarrow \inf, V^\pi(s) \Rightarrow E_\pi[G_t|s_t = s]$. **every visit MC on policy eval** - replace each instance of "first" in "first visit MC" description with "every". **incremental MC on policy eval** - same as "every visit MC" except update estimate should be done in the following way: $V^\pi(s) = V^\pi(s) * \frac{N(s)-1}{N(s)} + \frac{G_{i,t}}{N(s)} = V^\pi(s) + \frac{1}{N(s)} * (G_{i,t} - V^\pi(s))$. **incremental MC on policy eval running mean** - same as "incremental MC on policy eval" except update is: $V^\pi(s) = V^\pi(s) + \alpha * (G_{i,t} - V^\pi(s))$ where alpha can be manipulated – if $\alpha > \frac{1}{N(s)}$, then you are forgetting older data. **MC off policy eval** - even though a behavior (old) policy will have different distribution of rewards across episodes, we use the behavior policy to estimate the value of the new policy – useful when we don't have history for new policy like in medical field (not like *we have an entirely new treatment for cancer* type of situation, but a situation like *how would the outcome change if we had*). **bias** - $Bias_\theta(\hat{\theta}) = E_{x|\theta}[\hat{\theta}] - \theta$ **variance** - $Var(\hat{\theta}) = E_{x|\theta}[(\hat{\theta} - E[\hat{\theta}])^2]$ **MSE** - $MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias_\theta(\hat{\theta})^2$ **Importance Sampling** - you have data $x_1 ... x_n$ sampled from distribution $p(x)$ and you have $E_{x \sim p}[f(x)]$ but you want $E_{x \sim q}[f(x)]$, we can show $E_{x \sim q}[f(x)] = \int_x q(x) f(x) dx$ which turns out to be $\approx \frac{1}{N} \sum_{i=1}^N \frac{q(x_i)}{p(x_i)} f(x_i)$ Note we could always evaluate q and p at particular data points, but we originally could not find the expectation of f with respect to the q distribution which is what we wanted. **Importance Sampling for Policy Evaluation (also off policy)** - $V^{\pi_1}(s) \approx \frac{1}{N} \sum_{j=1}^N \prod_t \frac{\pi_1(a_t|s_t)}{\pi_2(a_t|s_t)} G(h_j)$ so in MC policy eval we can use the empirical average or we can use a reweighted empirical average (importance sampling) as necessary. **temporal difference learning for estimating V** - $V^\pi(s_t) = V^\pi(s_t) + \alpha * ([r_t + \gamma * V^\pi(s_{t+1})] - V^\pi(s_t))$ (note *td error* is the expression in parens that α scales). **usability of methods** - no model: MC, TD; non episodic: DP, TD; non markov: MC, converges (when tabular, alpha ¡ 1): DP, MC, TD; unbiased: MC; **tradeoffs** - TD lower variance because only 1 random decision, MC high variance no bias, TD only converges with tabular representation but MC converges even with functional representation. **MC/TD convergence** - MC converges to minimize MSE whereas TD converges to DP policy with maximum likelihood estimates for P(s'—s,a) and r(s,a). **certainty equivalence MLE MDP model estimates** - after each (s,a,r,s') tuple, recompute MLE MDP model for s,a; compute V^π using MLE MDP model; data efficient but computationally expensive to update model.

- Part 4: **model free policy iteration** - we already know how to do policy eval model free; need to modify policy eval for deterministic policies because you can't compute $Q(s,a)$ when $\pi(s) \neq a \Rightarrow$ so we need to try all (s,a) pairs; have to interleave policy eval and policy improvement $\Rightarrow$ so we need to ensure that Q estimate is good enough so policy improvement is a monotonic operator. ϵ **greedy policy** - with respect to state-action value $Q^\pi(s,a)$ is $\pi(a|s) = \begin{cases} a, & \text{with prob. } \frac{\epsilon}{|A|} \\ \text{argmax}_a Q^\pi(s,a), & \text{with prob. } \frac{1-\epsilon}{|A|} \end{cases}$. **Monotonic Improvement Thm for ϵ-greedy policies** - For any ϵ-greedy policy π_{i+1} (the ϵ-greedy policy w.r.t. Q_{π_i}), π_{i+1} is a monotonic improvement: $V_{\pi_{i+1}} \geq V_{\pi_i}$.**Greedy in the Limit of Infinite Exploration (GLIE)** - all state-action pairs are visited an infinite number of times, a simple example would be ϵ-greedy where ϵ is reduced to 0 at the following rate: $\epsilon_i = 1/i$. **Monte Carlo Online Control / On Policy Improvement** - Initialize $Q(s,a) = 0$, $Returns(s,a) = 0 \; \forall (s,a)$, Set $\epsilon = 1, k = 1$, $\pi_k = \epsilon$-greedy(Q) // Create initial ϵ-greedy policy. loop. Sample k-th episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, ..., s_T)$ given π_k for t = 1, ... , T do if First visit to (s, a) in episode k then Append $\sum_{j=t}^T r_{kj}$ to $Returns(s_t, a_t)$. $Q(s_t, a_t) = average(Returns(s_t, a_t))$ end if. end for. $k = k + 1$, $\epsilon = 1/k$. $\pi_k = \epsilon$-greedy(Q_π) // Policy improvement. end loop. **GLIE MC converges** to the optimal state-action value function. **Our convention for this cheat sheet is to take an action from a state and then receive a reward.** Applying TD gives us **SARSA** - Set initial ϵ-greedy policy π, t = 0, initial state $s_t = s_0$. Take $a_t \sim \pi(s_t)$ // Sample action from policy. Observe (r_t, s_{t+1}). loop. Take action $a_{t+1} \sim \pi(s_{t+1})$. Observe (r_{t+1}, s_{t+2}). Update Q given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$: $Q^\pi(s_t, a_t) \Leftarrow Q^\pi(s_t, a_t) + \alpha * (r_t + \gamma * (Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t)))$ Perform policy improvement: $\pi = \epsilon$-greedy(Q^π). $t = t + 1$. end loop. **conditions for SARSA convergence** (1) policy sequence $\pi_t(a|s)$ satisfies the condition of GLIE (2) $\sum_t^\infty \alpha_t = \infty$ but $\sum_t^\infty \alpha_t^2 < \infty$. textbfImportance Sampling for Off-Policy TD - the new update is $V^{\pi_e}(s_t) = V^{\pi_e}(s_t) + \alpha * (\frac{\pi_e(a_t|s_t)}{\pi_b(a_t|s_t)}(r_t + \gamma * V^{\pi_e}(s_{t+1}) - V^{\pi_e}(s_t)))$ – we have lower variance than MC because there is just 1 ratio of π's not a string of them we are multiplying, π_b does not need to be the same at every time step, conditions: π_b has

the same support in that $\pi_b(a|s) > 0$ whenever $\pi_e(a|s) * V(s) > 0$. **Q Learning** - maintain state-action Q estimates and use to bootstrap-use the value of the best future action. Initialize $Q(s,a) \forall s \in S, a \in A$. $t = 0$, initial state $s_t = s_0$. Set π_b to be ϵ-greedy w.r.t. Q. loop. Take $a_t \sim \pi_b(s_t)$ // Sample action from policy. Observe (r_t, s_{t+1}). Update Q given (s_t, a_t, r_t, s_{t+1}): $Q(s,a) \Leftarrow Q(s,a) + \alpha * (r_t + \gamma * \max_{a'}(Q(s_{t+1}, a') - Q^{\pi}(s,a)))$ Perform policy improvement: set π_b to be ϵ-greedy w.r.t. Q. $t = t+1$. end loop. Q-Learning must visit all s,a pairs infinitely often to ensure convergence to the optimal q and must decay ϵ to ensure convergence to the optimal π^*. **Maximization Bias** - even though each estimate of the state-action values is unbiased in Q learning, the estimate of $\hat{\pi}$'s value $\hat{V}^{\hat{\pi}}$ can be biased because Jensen's inequality can be used to show that it is greater than or equal to V^*. **double Q learning** - split samples to create 2 unbiased estimates – Initialize $Q_1(s,a)$ and $Q_2(s,a)$, $\forall s \in S$, $a \in A$, $t = 0$, initial state $s_t = s_0$. loop. Select a_t using ϵ-greedy $\pi(s) = \underset{a}{argmax} Q_1(s_t, a) + Q_2(s_t, a)$. Observe (r_t, s_{t+1}). if (with 0.5 probability) then $Q_1(s_t, a_t) \Leftarrow Q_1(s_t, a_t) + \alpha$ else $Q_2(s_t, a_t) \Leftarrow Q_2(s_t, a_t) + \alpha * (r_t + \gamma * Q_2 * (s_{t+1}, \underset{a'}{max} Q_1(s_{t+1}, a')) - Q_2(s_t, a_t))$ end if. $t = t + 1$ end loop.

- Part 5: **value function approximation VFA** - represent a (state-action/state) value function with a parameterized function instead of a table – reduce memory, computation, experience needed – will use differentiable functions. **stochastic gradient descent SGD** - to find the parameter vector w that minimizes the loss between a true value function $v_\pi(s)$ and its approximation $\hat{v}$ as represented with a particular function class parameterized by w. We use mean squared error and define **the loss** as $J(w) = E_\pi[(v_\pi(s) - \hat{v}(S,w))^2]$. **gradient descent** - we iteratively step in the direction of the negative gradient of the loss (loss and its gradient are calculated using all of the data we have). **stochastic gradient descent** - do gradient descent but compute the loss with a small fraction of the data points randomly picked (in our case we "sample" the gradient from particular randomly picked states, rather than taking the full expectation). Note $\nabla_w J(w) = E_\pi[2(v_\pi(s) - \hat{v}(S,w)) * \nabla_w \hat{v}(S,w)]$ and $\nabla_w J(w) = \alpha(v_\pi(s) - \hat{v}(S,w)) * \nabla_w \hat{v}(S,w)$ for a particular state S. **passive reinforcement learning** - have to use an estimate for $v_\pi(s)$ since we don't have it, this is policy evaluation without a model, "passive" because not trying to learn the optimal decision policy. **Linear Value Function Approximation for Prediction With An Oracle** - $\hat{v}(S,w) = \sum_{j=1}^{n} x_j(S) * wj = (x(S))^T w$. Update is $\triangle w = \alpha * ((v^\pi(s) - \hat{v}^\pi(S,w)) * x(S)$. **monte carlo value function approximation** - $\triangle w = (G_t - \hat{v}(S_t, w))x(S_t)$. Algo: Initialize $w = 0, Returns(s) = 0, \forall s, k = 1$. loop. Sample k-th episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, ..., s_{k,L_k})$ given π for $t = 1, ..., L_k$ do: if First visit to (s) in episode k then Append $\sum_{j=t}^{L_k} r_{kj}$ to $Returns(s_t)$. Update weights. $\triangle w = \alpha * (Returns(S_t) - \hat{v}(S_t, w))x(S_t)$ end if. end for. $k = k + 1$. end loop. **Temporal Difference Learning with Value Function Approximation** - TD(0) - $V^\pi(s) = V^\pi(s) + \alpha(r + \gamma * V^\pi(s') - V^\pi(s))$ - 3 approximations here – sampling an expectation, bootstrapping, function approximation. Here we have $\triangle w = \alpha * (r + \gamma * \hat{v}^\pi(s', w) - \hat{v}^\pi(s, w))x(s)$. **mean squared error of linear value function** - $MSVE(w) = \sum_{s \in S} d(s) * (v^\pi(s) - \hat{v}^\pi(s,w))^2$ where d(s) is the stationary distribution of π in the true decision process and $\hat{v}^\pi(s) = x(s)^T w$, a linear value function approximation. **MC with VFA for Policy Eval converges** - to $MSVE(w_{MC}) = \min_w \sum_{s \in S} d(s) * (v^\pi(s) - \hat{v}^\pi(s,w))^2$, the minimum MSE possible. **TD with VFA for Policy Eval converges** - to $MSVE(w_{TD}) = \frac{1}{1-\gamma} \min_w \sum_{s \in S} d(s) * (v^\pi(s) - \hat{v}^\pi(s,w))^2$. (note lin case deriv just x(s)) **Control using VFA** - $\hat{q}^\pi(s,a,w) \approx q^\pi$ for policy eval and ϵ-greedy policy improvement. Still minimizing J(w) using SGD analogously to Policy Eval with VFA. Linear relationship between q and w, analogous to linear relation between v and w. Updates follow. **MC with VFA for Control Update** is $\triangle w = \alpha(G_t - \hat{q}(s_t, a_t, w)) \nabla_w \hat{q}(s_t, a_t, w)$. **SARSA with VFA for Control Update** is $\triangle w = \alpha(r + \gamma * \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s, a, w)$. **Q-learning with VFA for Control Update** is $\triangle w = \alpha(r + \gamma * \max_{a'} \hat{q}(s', a', w) - \hat{q}(s, a, w)) \nabla_w \hat{q}(s, a, w)$. Note VFAs entail approximate Bellman operations and hence do not guarantee contraction. Convergence for Tabular VFA: MC, SARSA, Q-learning. Convergence for Linear VFA: MC, SARSA. Convergence for Non-Linear VFA (Deep Learning): None. Use deep learning instead of designing features. **deep neural network** - (see Neural Network or Convolutional Neural Network notes for details) uses artificial neurons (out = g(w x + b)), hidden layers can exist, can use sigmoid or ReLU, much better than clustering algos because doesn't need exponentially more parameters given more groups, NNs are **universal approximators** in that "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" (Hornik), deeper networks allow for fewer params used however, train using SGD, calculate gradients using backprop algo which breaks down each differentiable subfunction into a graph and forward propagates computation and backward propagates gradients using the chain rule (the gradients at each node are the derivative of the Loss with respect to that node's output), usually we have train, validate, and test sets but in RL there's no test, minibatch rather than single example in SGD is a useful technique for more accurate gradients, momentum lets us keep average of previous gradients – it's another technique, each layer learns a distributed representation of input, unsupervised pretraining to predict your own input is a technique to prevent overfitting (autoencoders), dropout to prevent overfitting, Batch Normalization – normalize pre-activation hidden layers to speed up training.

- Part 6: q function can take in a state and output a different value for each action, or it can take in an action and output a scalar. **convolutional neural networks** - (see Neural Network or Convolutional Neural Network notes for details) - filter (weights) that has same depth as image but smaller height/width goes over (one **stride** at a time) a zero padded image to produce an output block (which is a **feature map** / a hidden layer), can have multiple filters to check for multiple/different features, underlying operation is a dot product (see convolution demo under http://cs231n.github.io/convolutional-networks/ for details on how the operation works), **pooling layers** help downsample, for instance maxpooling takes the max of each quadrant. **DQN Deep Q Networks** - represent value function by Q-network with weights w, $\hat{q}(s,a,w) \sim q(s,a) - v(S,w)$ or $q(s,a,w)$ for each action possible. Still use SGD to minimize mean-squared error between the true action-value function $q^\pi(s,a)$ and the approximate action-value function $\hat{q}^\pi(s,a,w)$. Uses the same VFA approach for Q-Learning. **DQN in Atari** - end-to-end learning (no transfer learning, train only on the game that we are testing on), input state s is a stack of raw pixels from last 4 frames, reward is score

change for each step. **2 issues with VFA for Q-Learning in Atari** - (1) correlations between samples, samples are not IID (samples near by each other are dependent on each other, agent movement-wise) (2) non-stationary targets, in that the target function of VFA keeps changing. **Experience Replay** - solves correlation, stores a dataset (called **replay buffer**), of prior experiences (in the form sars' for each experience), sample one of the sars' from the replay buffer, use it to compute the Q-learning weights update $\triangle w$ for the state s from the sampled sars', use SGD to update network weights, we are effectively replaying the experience of some state. **fixed Q targets** - fix **target network** weights w^- and update weights w, use w^- inside max for computation of target, "when to update $w-$" is a hyperparameter. **DQN advantages** - performs better than best linear performers, replay is the secret to the success (also fixed targets to a lesser extent). **Double DQN** - extend double Q-learning to DQN context, current Q-network w is used to select actions, older Q-network w^- is used to evaluate actions. Double DQN performs even better than DQN. **value and advantage function** - features to heed to determine value may be different from those to heed to determine actions, $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, advantage function is less than zero for suboptimal actions and zero for the optimal action (calculates how much worse the action you are looking at is compared to the value of the state), given Q there is no identifiable A and V (2 unknown, 1 equation), because for policy just need ordinal relationships between actions. **Dueling DQN** - decomposes Q function into a value V and advantage A. Learn better features by focusing on advantage instead of Q function for the policy update. **options to resolve unidentifiability of V/A functions given Q** - (1) force advantage to be 0 if its the action that is taken $A(s,a) = 0$ if a is the action that is taken (2) use the mean as the baseline (more stable)

- Part 7: **RL algos that learn from only exploring state space** - not data / computation efficient or feasible in healthcare, driving, many real world applications. **reward shaping** - can demonstrate through demonstration. **imitation learning** - used when **demonstration trajectories** can help us determine rewards and policies; uses input state space, action space, transition model, no reward function, set teacher demos. **behavior cloning** - formulate as a supervised learning problem where we are given (s,a) pairs from teacher, compounding errors possible because if we make a wrong action we may end up in a state where we have no previous data. **DAGGER** - idea: get more labels of the right action along the path taken by the policy computed by behavior cloning; obtains a stationary deterministic policy with good performance under its induced state distribution; algo: (1) init $D = \emptyset$ (2) Initial $\hat{\pi}_1$ to any policy in $\prod$ (3) for $i = 1$ to N do: $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$ Get dataset $D_i = (s, \pi^*(s))$ of visited states by π_i and actions given by expert. Aggregate datasets: $D \Leftarrow D \cup D_i$. Train classifier $\hat{\pi}_{i+1}$ on D. end for. Return best $\hat{\pi}_i$ on validation.