RL Part 1

Monday, January 13, 2025

5:38 PM

Intro



Reinforcement Learning Basics

* RL involves communication between an agent and an environment
* Agent can modify the environment slightly by performing actions. Then, rewards are given to the agent based on observing the environment.
* A plan is a fixed sequence of actions
* Conditional plan includes if statements
* Stationary policy/universal plan - mapping from state to action
  + Lots of if statements/policy for entire environment
  + Very large
  + Theres always an optimal stationary policy
  + Might be too much to learn and makes the learning problem difficult
* Summarize of sequences - average: Expectation
* Evaluating a learner
  + Quality of policy
  + Computational complexity - Given same policy output of two learners, wall clock time could be a tiebreaker
  + Experience complexity (how much data it needs)
  + Space complexity can be a factor, but usually is not a limitation

Markov Decision Processes

* Main problem in RL is designed the environment well. An environment that is well suited to RL also.
* Once the environment is designed well, can use "off the shelf" RL algorithms to solve the environment
* RL agents have three internal steps. Interact, evaluate, and improve
* Value iteration and policy iteration is planning not RL
* Given an action, the environment outputs the next state and the reward, and may transition state. Reward is typically considered separate from the environment
* Agent is the decision maker, the environment is everything else.
* State space is all possible actions
* Observation and observation space can be the same as the state and state space (accordingly) in the grid world type scenarios
* MDPs that do not have observations are called MDPs and MDPs with observations are called POMDps
* There is also an action space, which is the set of all actions the agent can use to interact with the environment
* The function that is responsible for transitions is the transition function, which provides a probability of ending up in some state
* For a transition, the environment emits a new observation and provides a reward signal as a response from the reward function
* Sometime the transition and reward functions are referred to as the model of the environment
* There are terminal states where the "problem" is done
* The state space is the set of all variables necessary for optimal decision making if designed properly
* A timestep is a global clock syncing all parties and discretizing time
  + There is episodic timesteps (finite number of timesteps) and continuing tasks which go on forever
  + Can set a max timestep for continuing tasks to have them finish, aka truncated
* Discount factor will exponentially decay the value of later rewards, value goes down as time progresses

TD and Friends

* Three main types of reinforcement learning algorithms
  + Model based - model learner, MDP solver, take argmax of Q for best policy
  + Value-Function-Based/model-free = no longer feeding back transitions and rewards, just Q\*
  + Policy Search - Directly update policy, harder learning problem as the feedback from policy isn't directly useful for updating policy
  + TD Lambda combines the benefits of what happens at the end of the episode (TD(1)) with what happens between every step of the episode (TD(0))
  + TD Learning is value based
  + Learning Rate Properties - sum of all learning rate values is infinite and sum of learning rate squared is not infinite (is finite)

Planning Methods

Convergence

* Contraction Mapping - After running two Q functions through the contraction mapping and the distance is closer together than the original Q functions, the operator is a contraction mapping
* The episode proved that the bellman operator is a contraction operator, so value iteration converges
* Q-learning will converge given infinite time, and the learning rate is set properly
* As long as we pick operators that are non-expansive, algorithms similar to q-learning will converge
* One example of handling infinite actions. Sample from the set of actions and choose the best action in that sample. Some probability of missing the max
* Many convex combinations are non-expansive. One case where they are not. If the weights of the convex combination depend on magnitude of the values, then it is expansive.
  + Order-statistics
  + Fixed convex combinations
* Some not non-expansion operators have been shown to converge, so this property does not mean non-convergence in general.
* Converges isn't always to Q\*. Convergence is to some fixed point
* Simplicity is often called elegance - charles isbell

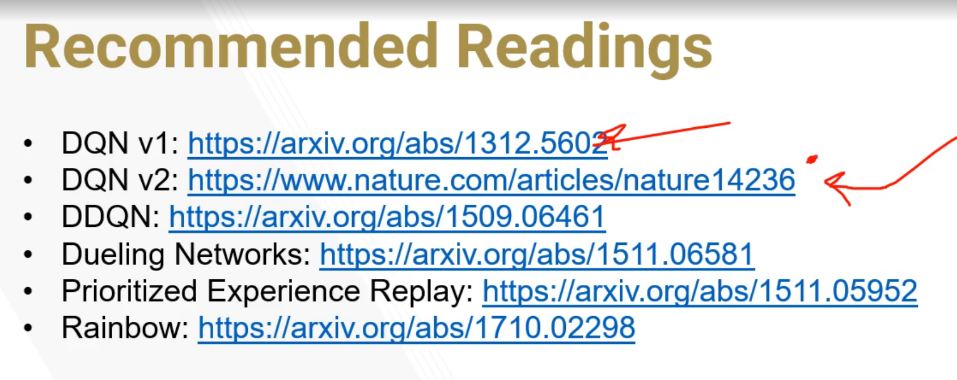
Prediction and Control

* Prediction - get values for any given policy
* Control - given accurate estimate and an exploration/exploitation trade-off, get better policy
* Regret - add up difference from chosen actions to optimal action
* Bandit system is a very common problem for recommender systems
* Explore vs exploit (in infinity)
  + Always explore - Get perfect knowledge of system but very inefficient, high regret
  + Always exploit - Likely to get stuck with imperfect knowledge and therefore poor actions
* Prediction
  + Having perfect value estimates makes policy iteration trivial
  + Monte-Carlo Prediction
    - Sample lots of states and average the results to estimate values
  + Temporal Difference
    - Use estimates of value to help predict next estimates, called bootstrapping
  + TD Lambda
    - The in-between of MC and TD, based on amount to consider previous steps. MC considers no previous steps while TD considers all previous steps. TD Lambda considers N previous steps. MC with Lamda 1, TD with lamda 0
* Control
  + Monte-Carlo Control
    - Monte carlo prediction with epsilon greedy improvement
  + Sarsa
    - Uses TD prediction, epsilon greedy policy improvement, improves specific action to possibly random action when using epsilon greedy
  + Q-learning
    - Can choose random action to improve, but improves it with respect to actual action chosen, not possibly random with epsilon greedy
  + Convergence
    - GLIE Greedy in the Limit with Infinite Exploration and Stochastic Approximation
      * All state-action pairs must be explored infinitely often
      * The policy must converge on a greedy policy
    - Must slowly decay epsilon towards zeros. Too fast, 1st condition might not hold, too slow, convergence might take too long.
    - Since samples have noise and we are learning from samples, also must push the learning rate towards zero (not get to zero)

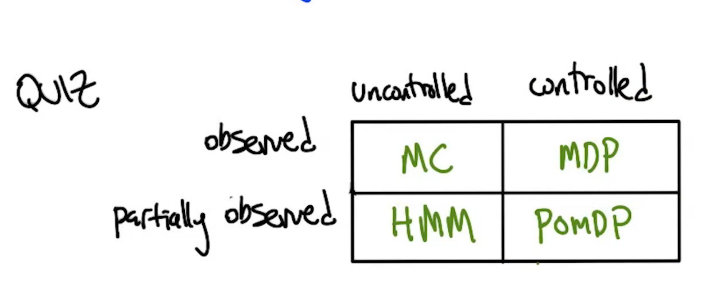
Generalization

* Goal is to leverage learning in states we have been to make predictions about states where we haven't been
* Lots of states could benefit from similar actions
* Could be function approximation for the Q function and model (i.e., learning the state action next state transitions
* Function approximation for the value function
* TD error tells us if our current prediction is too high or low, or right. TD algorithms move in the direction of TD error
* Averagers - kind of an MDP itself. So, there is a unique value function that it converges to, uses anchor points. More anchor points gets closer to right answer
* There is a need and way to do generalization, as many problems have huge state spaces
  + Need - large state spaces
  + Method - Apply supervised learning methods
    - Linear function approximation - value functions, policies, and models, gradient form
    - Q fitted iteration
    - LSPI - least squares policy improvement

Value-Based Deep Reinforcement Methods

* This section is reinforcement learning with function approximation. The currently used function approximation is deep learning
* Tabular reinforcement learning is sequential and evaluation, while deep reinforcement learning is sequential, evaluative, and sampled
* Planning requires seeing the entire "map", including transition probabilities and rewards. While agents typically do not have all that information at once
* Image based state spaces can have a ridiculously large number of states. This is true in many real world scenarios. In the image, we should be able to make some generalizations, though. That is where function approximation steps in.
* Also, continuous state spaces can have infinite states so these cases benefit from function approximation.
* Function approximation can help us learn complex problems faster. A lot of the theoretical convergence guarantees no longer apply with nonlinear function approximation currently
* Current types of RL approaches with function approximation
  + Derivative free
    - Use black box optimization methods like genetic algos
  + Policy based
    - Train a policy network
  + Actor-Critic
    - Train a policy and value network
  + Value-based
    - Train a value network
  + Model-based
    - Train a transition and/or reward function network
* Do not want to pass gradients through both parts of the q-learning algo
* Neuro-fitted q iteration - has some problems, similar to GAN issues. Can fix issues by fixing the target network for some significant number of timesteps
* DQN with a Replay buffer
  + Stores state/reward in a replay buffer, samples from that buffer as a minibatch. Starts with a warmup set of samples to start filling replay buffer
  + Online network is constantly updated, target network is updated after some steps. This helps stabilize gradient flow
  + DQN issues - overestimates targets. Always assumes that there is a higher value. (positive bias)
* DDQN: Double Learning
  + Helps fix some DQN issues by using the online weights to select the action but still use the frozen target networks weights to get the estimate
  + 

POMDPs

* Partially observable markov decision processes
* POMDPs do not necessarily have the markov property. That is, you can see the same observation twice in a row but be in different states
* We have indicative observations of the state, but the observations do not directly/perfectly indicate state
* Anything you can represent as an MDP, you can also represent as a POMDP. So POMDPs generalize MDPs
* MDP has <State, Action, Transitions, and Reward>, while POMDP has <State, Action, Observables, Transitions, Reward, and Observation function>. The overlapping values map directly, and Observables can also be considered the State, while the observation function can be considered the transition function.
* Observation function represents the probability of being in a state given some observation(s)
* Instead of knowing state, we have a vector representing probability of being in each state
* Reward is not observed in the belief MDP. Rewards are moreso based on the observation
* The Belief MDP has an infinite state space, so algorithms like value iteration need to be adapted to work. Carefully represent the infinite case in a finite way
* There is a method called purge to remove vectors that do not influence the piecewise linear function. If all elements of the vector are larger than another vector, it is not needed to represent the piecewise linear function
* Since POMDPs have infinite states, value iteration does not necessarily converge like with regular MDPs. Though, it is shown that we get an arbitrarily good approximation of the value function after some finite number of iterations
* Life is undecidable per Isbell
* Model based RL - learns a model and uses it
* Model free RL - does not learn a model
* 
  + MC - Markov Chain
  + HMM - Hidden Markov Model
* Expectation Maximization - iterate between two things we do not know. The model/hypothesis and the expected value of hidden variables.
* Bayesian RL is a thing, currently too expensive to be practically useful
* Predictive State Representation (PSR)
  + POMDPs - belief state is a distribution over states, states never observed. Do states exist?
  + PSR - probabilities of future prediction outcome's. Are there tests we could do to predict what state we are in? If you do certain things, what happens? Can represent POMDPs as PSRs
* Bayesian RL blurs lines between planning and learning. Recognizes that "learning" is estimating probabilities of things and is therefore not really learning. There is no separation between exploration and exploitation, acting given knowledge of truth. Exploration is exploitation
* States and predictors are kind of the same thing

Policy-Based and Actor-Critic Deep RL methods

Some optimal polices can be stochastic, meaning having randomness in the policy

A3C

* Multiple agents collecting trajectories
* A3C is a good actor critic method when you do not have a GPU
* A3C has asynchronous and lock-free network updates
* Uses an update style called Hogwild!, which has been shown to achieve a near-optimal rate of convergence and also outperform alternative schemes that use locking by an order of magnitude

Generalized Advantage Estimation

* Robust estimate of the advantage function

A2C

* Synchronous model updates
* Single agent driving interaction with environment. Though, the environment is multiprocess that gathers samples from multiple environments at once
* Enables GPUs to be the most important resource over CPUs

There are attempts to share weights between policy and value functions, but have been difficult to train so far

DDPG

* No longer outputs a distribution over actions and values for the network, directly outputs best action, while the "value" network outputs the value given a state and the best action
* Slightly update target network at every step, instead of a larger update every N steps as DQN does



Exploring Exploration

* Exploration is the main separator from supervised learning and reinforcement learning
* Confidence of estimated probabilities is determined from the number of samples
* RMAX (deterministic, markov case, sequential) - algorithm optimistically assumes that unexplored states are the max value, so helps push exploration towards those values
* Bandits (stochastic, non-markov case, nonsequential) - We can use hoeffding and union bounds to estimate what we know about an environment. We used this to convince ourselves that we had a sufficiently accurate estimate to exploit and get approximately optimal reward.
* RMAX + bandits (stochastic, markov case, sequential) combined allows us to explore well while being able to estimate how much we know about the world
* KWIK - know what it knows learning

AAA (Advanced Algorithmic Analysis)

* Compare the difference between two policies, then we can estimate how far we are from an optimal policy, assuming you use a small gamma
* Horizon is 1/(1 - gamma)
* Linear programming is the only way we know of to solve MDPs in polynomial time
* Policy Iterations moves toward optimal solution in no less time than value iteration
  + Faster convergence at the cost of more compute. Pretty much does all the work of value iteration with extra steps
  + Don't know if convergence time is linear or exponential, its somewhere in between. If its closer to linear then its much better than value iteration, but much worse if closer to exponential
* Domination - Policy dominates another policy if all states in the first policy have greater than or equal to value than the second policy
* At every step of policy iteration, the policy will not be made worse. The new policy will be better or the same. If there's a way to improve, policy iteration will improve

Game Theory

* Game Theory - mathematics of conflict
  + multiple agents
  + Came out of economics, on the scale of millions of agents can represent economy
* Pure strategy - strategy stays the same throughout the game
* Mixed strategy - chooses a strategy with some probability distribution
* Prisoners dilemma - highest value for an agent is not always the best outcome in the multi-agent environment
* Nash Equilibrium - occurs when, if randomly choosing an agent and allowing them to switch their strategy after the other actions have been set, the agent would not choose to. This is true if all agents are this way
* Mechanism design - design system in such a way that encourages certain behavior

Introduction to Deep Multi-Agent reinforcement learning

* [Introduction to Deep Multi-Agent Reinforcement Learning](https://www.youtube.com/playlist?list=PLFihX_3MLxS9iS3Hz8VDWAyU4QCdjQUKr)
* Distributions of the world are shifting as other agents improve
* Controlling multiple entities with the same controller is not multi-agent
* TD Estimate - use previous value of a state to predict future value
  + Trajectory is cut off at the current state to update previous states without considering future trajectory
  + Using the estimates to update estimates is bootstrapping which can introduce bias to the learning. Usually bias is bad but introducing bias here reduces variance which can improve learning speed
* Monte Carlo - estimate state values and average them without using previous estimates. Gather full trajectory first.
* Monte Carlo fully focuses on output while making individual actions less important. Problem is there are sometimes good things you did when you lost
* A3C - one worker per CPU collecting samples, asynchronously update network as data buffers are full. No synchronization point
* A2C - Advantage actor critic method but does not update asynchronously, that’s the main difference between a3c and A2C. A2C can be better for GPUs
* Execution and training are separate things. Can plan during training to do better during execution, but it is separate
* Independent actor critic - Naïve approach is to train independent agents separately
* Centralized Critic - Separate policies per agent but the value function is shared. This allows adding privileged information to the training. Since the separate policies are all that is needed in the real world, the agents can be separated while having some overlapping knowledge. This works for both adversarial and team scenarios, any other relevant info as well
* Sharing Weights - actor and critic weights are shared. Not necessarily better than centralized critic, but a different approach that may be applicable in some scenarios. If the actions are very similar, then this may be a good way to go. If the behaviors are very different than it may not be. In this case, either want the policies to be separate or add an additional input for agent type/number
* Centralized Training with Decentralized Execution - fundamental to MARL. Shared information during training phase, while agents do not communicate directly during execution phase

Game Theory Reloaded

* Repeated games and the folk theorem - in repeated games, the possibility of retaliation opens the door for cooperation
* Folk theorem in general - known to experts in the field and considered established status, but not published in a proven, complete way
* Folk theorem in game theory - describes the set of payoffs that can result from nash strategies in repeated games
* Grim Trigger - cooperation - mutual benefit. Cross the line - deal out vengeance
* A plausible threat - a threat of lower reward can happen with a strategy
  + Subgame perfect - always best response independent of history
* Computational Folk Theorem - Can build pavlov-like machines for any game given a 2-player bimatrix game -> average reward repeated
* Stochastic games provide a formal model for multiagent RL
* If there are multiple players, could make it a regular mdp by making the other players irrelevant
* Discounting in RL is related to repeated games in game theory. Repeated games can encourage collaboration

Game Theory Revolutions

* Correlated Equilibrium - Have some other signal for coordination
* Cooperative-competitive values (COCO) - share, split extra payoff, binding side payments
* Mechanism Design - Given desirable behavior, make a game that produces that behavior
* Peer Teaching - should put the difficulty bar where people get things right that they should get wrong and get things wrong that they should get right equally
* Solution Concepts are the name for the above
* All of the above are also forms of mechanism design - used to get better outcomes using rewards. Driving Behavior via rewards.

CCC (Coordinating Communicating Coaching)

* Decentralized Partially Observable Markov Decision Process (DEC-POMDP)
  + I - Finite set of agents
  + S - states
  + A - agent I's actions
  + T - Joint transition function - T(s, a, s') - a is a set of actions, one per agent
  + R - Shared reward function - R\_i (splitting reward per agent) if POSG - Partially observable stochastic game
  + Z\_i - agent I's observations
  + O - observation function
* DEC-POMDPs are NEXP-complete (for finite horizon). Best solved between exponential and double exponential time (Coordinating and Communication)
* Inverse Reinforcement Learning - provide example trajectories and learn a reward function
* MLIRL (Maximum Likelihood Inverse RL) - guess reward function, compute optimal policy with that reward function, measure probability of trajectories given policy, compute gradient on R. Can be slow and get stuck in local minima. Really cool way to learn reward functions from expert/human trajectories. (Coaching)
* In research, it is often the case that we develop a performant ML algorithm and say these are the data we need for it. Though, it is more practical to work with what we have and develop an ML algo from that.
* Policy Shaping -
* With human feedback, give 1 positive reward for approved actions and -1 reward for actions labeled bad. It works to treat this as reward shaping and convert this into rewards. In practice, people are not thinking in terms of rewards, they are thinking in terms of policies. In this case the people are giving direct policy advice so it may make more sense to convert this into a policy and not an rewards.
* Person might not always be right - so use feedback to try and fit a policy not directly use.
* Persons yes vs no's gives evidence of the policy.
* Should also consider multiple sources. Still consider feedback from the environment + the noisy human policy coaching
* Can communicate with agents in multiple ways
  + Demonstrations
  + Rewards
  + Policies
* Trajectories as stories. Plot points are vertexes. Then, stories can be turned into MDPs.
* Trajectories as MDPs
  + States - partial sequences
  + Actions - Story Actions
  + Model - Player model (given sequence of states, an action is being taken, what plot state are we likely in
  + Rewards - author evaluation
* The states above as partial sequences - the number of possible setups of these is very large
* The player will enjoy themselves but only according to how the author defines enjoyment. Having to find an answer forces you down a particular, specific path based on reward.
* TTD-MDPS - Targeted Trajectory Distribution
  + Trajectories: Sequence so far
  + Actions: actions (story actions from director or similar)
  + Model: P(t' | a, t) - probability of what the next trajectory is given previous traj, and action
  + ~~Rewards~~ target distribution: P(T) - T is final trajectories or end of stories. High probabilities are good stories and low is bad.
* Relaxing a hard constraint to a soft constraint tends to allow you to solve harder problems. E.g., search
* DEC POMDP - Agents don't share brain but share reward
* IRL - Go from behavior to reward instead of reward to behavior. A type of reward shaping in the form of behavior demonstrations.
* Demonstrations is one way to do coaching
* Policy shaping - Given human feedback, directly create policy instead of reward.
* TTD-MDPs
* People think a particular way, machines think a different way. We should design the machines to think similarly to humans rather than move humans toward machines to close this gap.

Messing With Rewards

* Reward shape to make the MDP easier to solve and learn something similar to what it would have learned otherwise
* Change it because we don’t have one yet……. The true reward function exists but probably cannot be found in practice
* Rewards are kind of like a programming language. Specify rewards and out would come the corresponding behavior
* AI is a lot about turning a problem into an easier problem, which the easier problem may not be interesting anymore
* Can change reward function to be different in that it provides different behavior, or you can change it to keep the same behavior but learn more efficiently in terms of number of examples, space, and ability to solve it in a reasonable amount of time.
* Changing reward function without changing optimal policy:
  + Multiplying by a positive constant doesn't make a difference in absolute impact of reward
  + Shifting reward does not change behavior.
* A dolphin will not just jump through a flaming hoop and expect some reward. It must be coaxed gradually to get there. Reward parts of the steps
* Potential-Based shaping in RL (Changed in state based bonuses)
  + Achieve state - get bonus
  + Unachieve state - lose that bonus
* Rewards are hints to get to ultimate goal
* Modify rewards to learn more quickly
  + Potential Functions (Learn faster and the same thing)
  + Shift (Don't change anything)
  + Scale (positively) (Don't change anything)
* Reward shaping has dangers, get into feedback loops where the agent "cheats the system" to get high reward without achieving the goal.

Options

* Issues in RL
  + Estimates based on estimates can be a bad assumption, have to do extra things like reward shaping to get good learning
  + Delayed Reward
  + Having too many states or actions or both can make learning difficult. Reducing this helps learning in general
* Can add new actions that abstract other actions to make learning easier. Learning over time is hard, reducing the number of states needed to complete the goal makes the learning easier (Super actions)
* So, action shaping, reward shaping, and policy shaping are ways to make learning easier
* Temporal Abstraction Options
  + Options <I, Pi, Beta>
  + I = Inititation set of states (where an action makes sense)
  + Pi = 3.141592653589793 policy (S, a)
  + Beta = Termination set of states
* Allows us to reason over policies
* SMDP - Semi markov decision process - makes jumps in time. Reduces to an MDP and can treat it as such
* Temporal Abstraction - grouping small actions into large actions allows us to separate concerns. Can focus on getting really good at the specific actions
* Can ignore large portions of the state space when solving smaller problems like specifically eating dots in pacman and using that focused policy in a temporal abstraction
* Temporal Abstraction allows us to focus on major decisions and ignore unimportant actions and states, therefore improving exploration of important states
* Modular Reinforcement learning - divide the world into a bunch of modules or options. Need to decide what program to let allow run next.
* Greatest Mass Q-learning - add all action sequences together all the actions and follow that one
* Top Q learning - always pick the max q value
* Negotiated W-learning
* Previous 3 methods are impossible
* Monte carlo tree search
  + Select, expand, simulate, backup
  + Operate with constraints - explore randomly but avoid dying. Allows you to be smarter about how you do your rollout policy
  + Compatible with options
  + Policy search with an inner loop of policy evaluation
  + Useful for large state spaces
  + Need lots of samples to get a good estimate
  + Planning time independent of state size
  + Running time exponential in the horizon (how far in the future you need to look)
* Generalizations beyond function approximation
  + Temporal abstraction - options and semi-mdps
  + Goal abstraction - modular RL and arbitration
  + State abstraction - sort of what function approximation does
* Quiescence
* Hierarchical RL
* MSTS - becomes more important as we want to apply RL to the real world

Introduction

* Missing concepts
  + Burlap
* Key part of RL vs SL is RL has interaction with an environment
* Elevators are one application of RL
* RL to fly a model helicopter
* Fly a drone
* RL has had an impact on other sciences like neuroscience
* RL is a way of solving problems