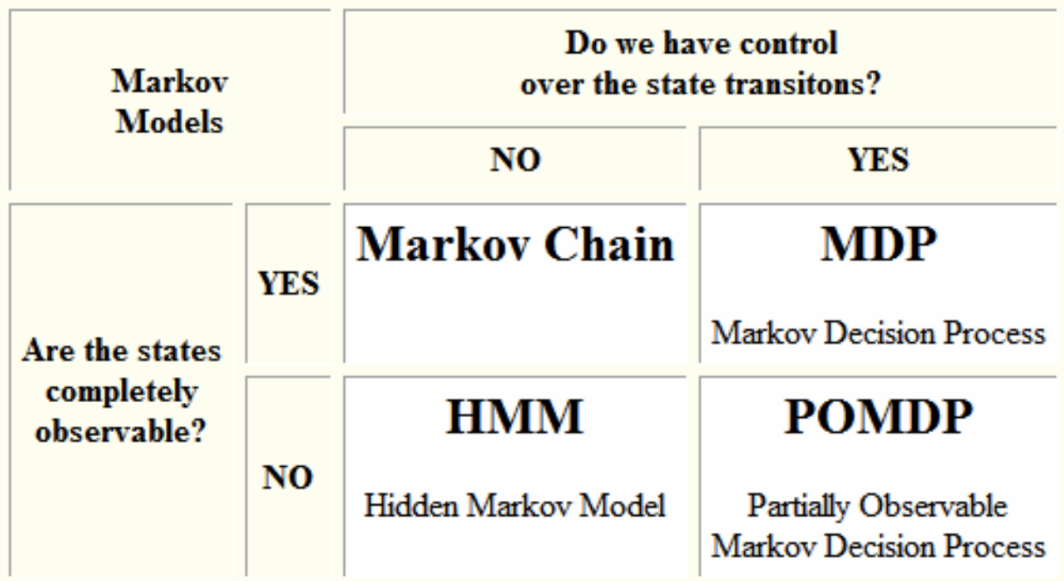
exploration

- exploration vs exploitation refers to the general tradeoff of when to explore more vs use the current known good strategies. in particular, if an agent is playing a game live without a ton of training before a tournament, it has to play well (it is competing) while figuring out good moves.
- here, we will try to come up with some formal notions of optimality for exploration. we will restrict ourselves to small state and action spaces. it turns out to be intractable for larger ones:
  - multi-arm bandits: stateless, 1-step MDP
  - contextual bandits: stateful, 1-step MDP
  - small, finite MDPs: doable.
  - larger continuous spaces (ones we developed algorithms for in this course): intractable
- A POMDP is one where the system dynamics are determined by the MDP but the agent cannot fully observe the underlying state.
- a helpful chart:



- A multi-armed bandit, like deciding which casino game to play where each game just involves one action with different probabilities of success → we just have to decide which of multiple arms to pull, each arm has a different probability of success, each of which is unknown. so this is a POMDP. we could explore like crazy and estimate the probability of success in each. but that can be overkill.
- there are other exploration algorithms. how do we measure each. by something called regret: Regret is the difference between the best possible expected reward, the one that is achieved by the optimal policy vs the one we use. **exploration optimality can be defined as regret minimization.**
- By the above definition, the expected regret of the optimal policy is 0.
- now, how can we minimize regret and what theoretical guarantees on expected regret can we provide? we will discuss the multi-arm bandit for now.
- optimistic exploration:
  - keep track of average reward for each action μ_a

    - pick one that maximizes the reward with some weight given to actions we have a bad estimate of: $a = argmax\ \mu_a + C\sigma_a$, where σ_a is the estimated variance in the estimate. the more the variance, the more chance of picking it.

  the intuition above is to try each arm until you're reasonably sure its not great. there are different ways to come up with σ_a. the specific algorithm above is called the upper confidence bound algorithm (UCB).
- probability matching/posterior sampling/Thompson sampling:
    - maintain a belief state of the probabilities of success in each bandit, $p(\theta_1, \theta_2, ...\theta_n)$. sample from this distribution and take an action based on this. note how this **belief/distribution of goodness of each state is basically just a distribution over q-functions**!
    - why is this good? this is better than random because if your estimate of at least a few bad bandits is good, you are less likely to pick them.
    - of course, after each action we update the model
    - in summary, sample from distribution over models, act on the model and update the distribution.
- empirically both the above algorithms perform quite well in different situations.
- information gain/bayesian experimental design:
    - the entropy of a distribution is $\mathcal{H} = \sum p(x) log\ p(x)$ or $\int_{-\infty}^{\infty} p(x) log\ p(x) dx$.
    - Now we have some probability distribution over what actions are good/what the values are or some such.
    - here, we choose which action to take based on which action provides the most information gain, on expectation. Lets say z is our latent variable (the underlying thing we have an estimate of).
    - information gain for action a is $IG(z, a) = E_a[\mathcal{H}(p(z)) - \mathcal{H}(p(z|a))]$. yes this is kinda circular reasoning → estimated information gain of our model is based on our current model.
- the above 3 strategies: optimistic exploration, information gain and posterior/thompson sampling have theoretical results in simple 1-step MDPs (multi-armed bandits). we can prove stuff here and use these basic methods as foundations for methods of exploration on more complex spaces/tasks. these more complicated spaces get quite hard to analyze quite fast.
- we didn't talk about contextual bandits here → contextual bandits are also 1-step MDPs except that you have some context/prior information about state that you can use before making a decision → its still a 1-step decision though.
- now how can we use this to solve more complex spaces? instead of investing more theoretical coverage in these small cases, we will go onto more approximate methods for complex cases and see how each of the above 3 algorithms leads to a deep RL version of it.
- note that none of the above algorithms explicitly optimize regret. regret is just a metric to compare them.
- the deep RL version of optimistic exploration/UCB:
    - in UCB, we can estimate the variance term as $\sqrt{\frac{2ln\ T}{N(a)}}$, where T is the total number of steps/actions taken and $N(a)$ is the total number of times we've taken action a. This intuitively says the more number of times I've taken this action, the less the variance of my estimate, without any formal notion of estimate. this is just more of a heuristic but tends to work well.
    - This term is called the exploration bonus because if we haven't chosen an action too many times, it encourages us to choose it by assigning a higher exploration bonus to it.
    - Sometimes we instead of $N(a)$ we can use $N(s, a)$ or $N(s)$ depending on the problem.
    - Now how do we extend this discrete notion of $N(s)$ to continuous spaces/high dimensional spaces where $N(s)$ is probably 0 for most of states?
    - We will learn a **density model** over all the visited states to estimate $p(s)$. With some simple math, we can use $p(s)$ to get a pseudo-count of the number of times we've visited a state $N(s)$ which we

can plug into the bonus term to get its contribution.
    - The above still assumes discrete number of actions and we will argmax over expected reward + exploration bonus as before.
    - what kind of density model should we use? GANs and other popular density models are good to sample from but hard to estimate densities from. so sergey shows some other kind of density model here.
    - an alternative method is to map the state to a k-bit bottleneck using an autoencoder. Assuming we have learnt the autoencoder in a suitable way (for example to reduce reconstruction error) we can map a bunch of states to a k-bit bottleneck. assuming k is small enough, we will hopefully see a lot of similar states mapping to the same code. then we can count these k-bit codes. this is called **counting with hashes,** since we can think of this bottleneck representation as a hash.
    - these exploration bonuses we've seen here can be added to the q-function for example, before picking an action, if we are in a domain where we think lack of exploration is hurting us.
    - an alternative way to see if a state is "novel" to encourage exploration is to try to build a classifier that tries to distinguish a new state from previous visited states. if it easily classifies it (weighted by probability of classification) then it is not so novel and we can use the resulting probability as the exploration bonus. if it is easy to build that classifier and the classifier assigns a high probability to "it is novel" then it gets a higher exploration bonus.
- deep RL version of posterior/thompson sampling:
    - we said that posterior sampling distribution is basically a distribution over Q functions. just like we did in the multi armed bandit case, we can sample a set of q functions and take an action based on those samples and then update the q-function estimate model based on the thus taken step to make it more accurate for sampling on the next iteration.
    - in the deep RL version, we will use an ensemble of neural net q function approximators, and we will sample from this ensemble.
    - the reason this ensemble of neural nets approach is different that simple deep Q learning is that we have some randomness and that encourages more exploration, which is the whole point of these methods!
- how do we approximate the information gain method in complex environments?
    - sergey doesn't go into the math too much here but he says that we can approximate information gain exploration methods by making each neural network weight a gaussian distribution to model the uncertainty. we can then take a new sample and run backprop k times if we have k choices of actions. we can see how each of these affect our gaussian weights by computing a KL divergence between the weights of the two neural nets, which is apparently mathematically simple because the weights are all gaussian. then we can pick the action that will give us the most information gain.
    - one question that's unanswered is: how do we know how much information gain we will get without actually taking each of the k actions → ¯_(ツ)_/¯.
    - there are also variants that approximate this. instead of taking a strict definition of information gain, we could do things like exploration bonus for model error, exploration bonus for model gradient from the new point etc.