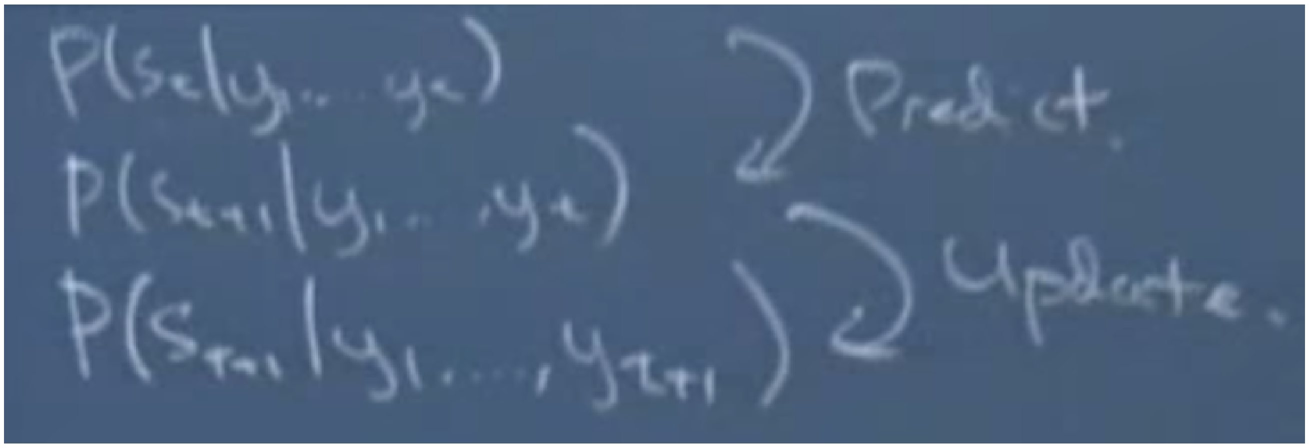
RL continued: more MDPs, kalman filters

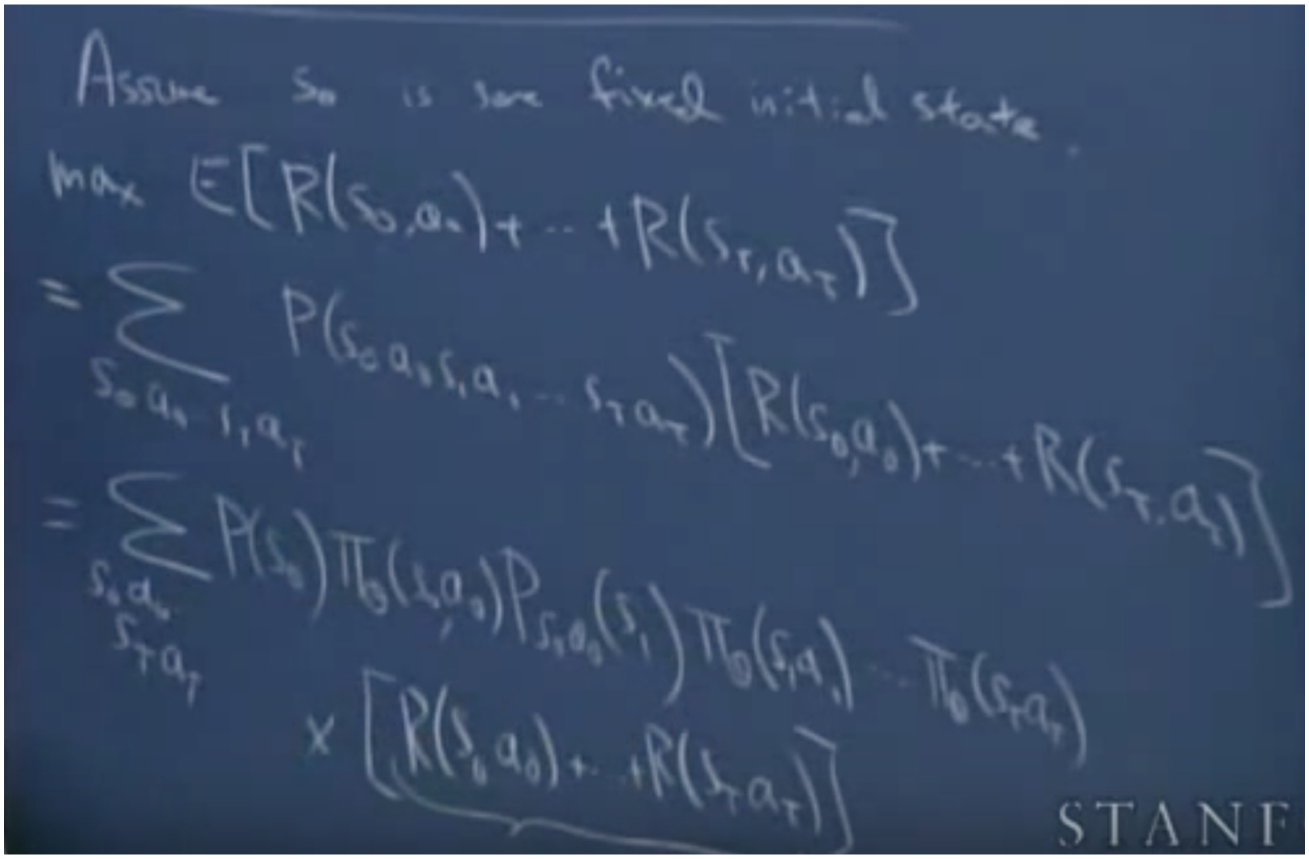
- State action rewards:
  - an alternative formulation of the reward function is instead of $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_3) + \gamma^3 R(s_4)$, we have $R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \gamma^3 R(s_3, a_3)$. this formulation allows more flexibility when dealing with problems and the old formulation is a special case of this.
  - The Bellman equation is very similar except instead of $R(s) + max_a \, \gamma \sum ...$, we now have $max_a[R(s,a) + \gamma \sum ...]$. The max operator comes out because R depends on s, a and not just s.
  - Value iteration and other stuff is pretty similar and follows just like it did previously
- Finite horizon MDPs:
  - Instead of discount factor γ, in this formulation, we have T which is the maximum time horizon. That means we can take at max T actions and the total reward function is just sum of rewards at each step, without any discount factor.
  - some terminology: if something doesn't depend on time, it is called stationary. Non-stationary means it does depend on time.
  - the optimal value from a certain state in the previous formulations were stationary. Here, the optimal value is non-stationary. (think about why)
  - P_{sa} can also become $P_{s_t a_t}$ (a function of time, non-stationary).
  - to make it fully general, the reward function could also be $R^t(s_t, a_t)$ instead of $R(s_t, a_t)$.
  - The optimal value function is then, obviously, also a function of time.
  - we can derive a nice dynamic programming algorithm where we first compute $V^{*T}(s)$ and then for $T-1, T-2$ etc.
- Linear Quadratic Regulation (LQR):
  - this deals with continuous space state and continuous action space with finite time horizon, so $S \in \mathbb{R}^n, A \in \mathbb{R}^d, P_{sa}, T, R$.
  - P_{sa}^t: $s_{t+1} = A_t s_t + B_t a_t + w_t$ where w_t is, say gaussian noise with mean 0 and some covariance matrix. (just some linear function of state and action + some noise)
  - the reward function is written as $R^t(s_t, a_t) = -(s_t^T U_t s_t + a_t^T V_t a_t)$ where U_t, V_t are positive semidefinite which means that the reward is always negative.
  - now our goal is find the optimal policy that maximizes $R^t(s_t, a_t) + R^{t+1}(s_{t+1}, a_{t+1}) + ... R^T(s_T, a_T)$.
  - this follows a dynamic programming problem very similar to the one we saw in finite state horizon MDPs. andrew doesn't do the derivation of the math for this, but we will see it later in the advanced robotics course.
  - Again, andrew states some LQR equations and talks about something called differential dynamic programming which is a way to do LQR iteratively. he doesn't go into the details of the math here. We will do the detailed math and derivations in the course notes on cs 287 advanced robotics course from Berkeley.
- advice for debugging RL algorithms. lets say you have a model that doesn't do well in the real world:
  - if the model does well in the simulator but not in the real world improve simulator
  - if human doesn't do well on reward function but flies well, work on the reward function

  - if the human does better on the reward function than the model does, then its a problem with the RL algorithm. its not finding optimal values.
- Kalman filters:
  - Lets say we have a robot that's moving with state $(x, y, \dot{x}, \dot{y})$ → (position and velocity). lets say we have a simple dynamics model like $x_{t+1} = x_t + \dot{x}_t + noise$ and $\dot{x}_{t+1} = 0.9\dot{x}_t + noise$. But lets say true state (position and velocity) are not observable. Instead we have a camera (or radar) measurement of just position. Lets say the radar measurement is modeled by $y_t = Cs_t + noise$ with the noise having mean 0 and some covariance matrix (its gaussian noise).
  - We get to observe y_t and we'd like to estimate $p(x_t|y_1, y_2...y_t)$, that is true position given the radar measurements.
  - The joint distribution of $x_1..x_T, y_1...y_T$ is a gaussian, which we can find and then find a formula for $p(x_t|y_1, y_2...y_t)$. This is conceptually good, but computationally hard since the covariance matrix grows as n^2 where n is the number of time steps. and note that we'll have to invert the big covariance matrix when we compute gaussian pdf. inverting huge matrices is bad.
  - Instead there is an algorithm called kalman filter that helps us do this more efficiently.
  - Summary of kalman filter:


$P(s_t|y_1 \dots y_t)$
$P(s_{t+1}|y_1, \dots, y_t)$
$P(s_{t+1}|y_1, \dots, y_{t+1})$
Predict.
Update.

  - Again for kalman filters, andrew does not derive any of the stuff 😞 we'll do this too in cs 287 advanced robotics.
- Andrew mentions something called linear quadratic gaussian system (LQG) that puts together Kalman and LQR. Again, cs 287 😃
- Partially Observable MDPs (POMDPs):
  - this is same as MDPs except the states aren't always fully observable
  - the formulation is same as for MDP with two additional things in the tuple:
    - Y → set of possible observations.
    - O_s → observation distributions. If we're at state s_t, we observe y_t ~ O_s.
  - We won't say much more here → cs 287.
  - "solving POMDPs in general is NP-hard"
- policy search:
  - we'll develop this for MDPs but then briefly mention how they can applied to POMDPs.
  - we'll come up with a set of policies $\prod$. Then we will search for a good policy $\pi \, \epsilon \, \prod$.
  - this is analogous to how we found a good hypothesis among a set of hypotheses in supervised learning

- in the RL algorithms we've seen so far, we've relied on computing V^* (or estimating it) and then choosing our policy based on whatever maximizes that. this policy search approach is more direct → we won't rely on computing the value function
- A stochastic policy is a function π : S x A → $\mathbb{R}$ where $\pi(s, a)$ is the probability of taking action a in state s. so, $\sum_a \pi(s, a) = 1$
- in policy search, we kind of phrase the problem of learning like we do for supervised learning. so we will have a function with some parameters and we'll find a good setting of those parameters. That function will give the probability of a in state s.
- In other words we will find θ that maximizes the expected value of the payoff.
- reinforce algorithm:
  - basically, we write the expected reward of following a policy as follows:


Assume s_0 is some fixed initial state.

$$\max E[R(s_0,a_0)+\dots+R(s_T,a_T)]$$

$$= \sum_{s_0 a_0 s_1 a_1 \dots s_T a_T} P(s_0 a_0 s_1 a_1 \dots s_T a_T)[R(s_0,a_0)+\dots+R(s_T,a_T)]$$

$$= \sum_{\substack{s_0 a_0 \\ s_T a_T}} P(s_0)\pi_\theta(s_0,a_0)P_{s_0 a_0}(s_1)\pi_\theta(s_1,a_1)\cdots\pi_\theta(s_T,a_T) \times \underbrace{[R(s_0,a_0)+\dots+R(s_T,a_T)]}$$

  - once we have this, we take the derivative of this w.r.t the parameters θ and do SGD (stochastic gradient descent) to find the optimal parameters. the math is just chain rule, we've done this enough times, won't write it here again → more discussion in the notes on deep RL class.
- These kinds of policy search algorithms often work well for problems where the action (or probability of action really) is expressible as a function of state. this often turns out to be true for control problems like driving, autonomous flying etc, and turns out to be less true for problems where long term, multi step thinking is required like chess.
- pegasus policy search:
  - the idea here is that we have a simulator and sample a few values of θ, follow the policy a bunch of times, take the avg (since the policy is stochastic) and plot the total value.
  - we do this for different θ's. once done, we fit a curve of different expected values vs policy and do gradient descent (ascent?) to find the the best value of θ

  - yes this is an approximate solution but we the advantage is this works pretty well on really complex problems, like andrew's helicopter control project.
- RL is really dealing with sequential decision making. when you have to make a list of decisions in sequence with the goal of optimizing some overall objective.
- Done, congrats!