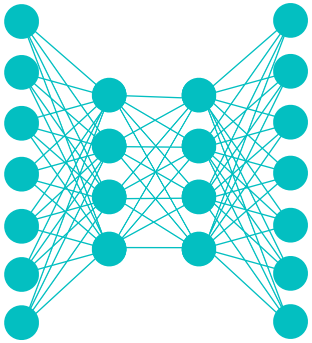
Lecture Notes for

Neural Networks and Machine Learning

Introduction to Reinforcement Learning

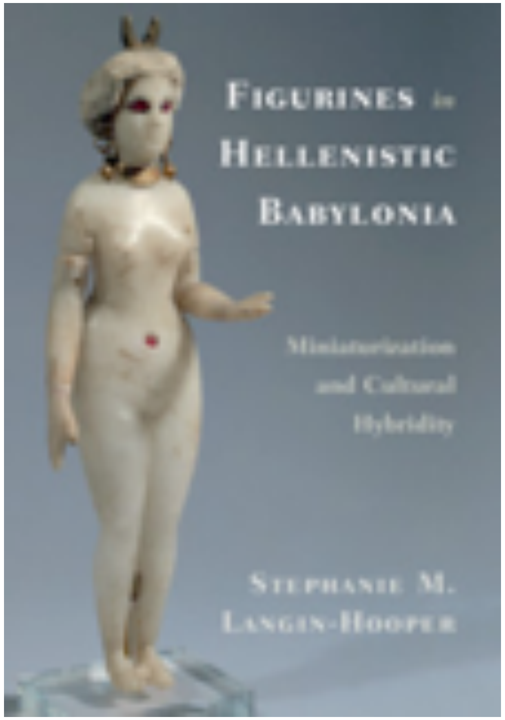


Logistics and Agenda

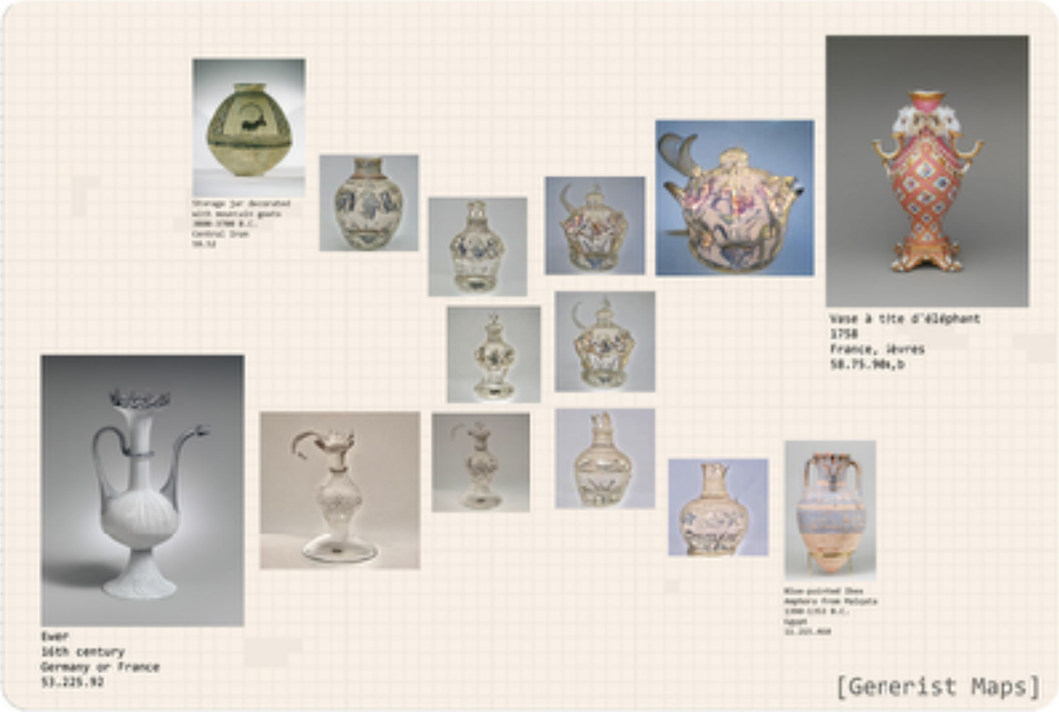
- Logistics
  - Student Presentation Next Time
- Agenda
  - Final Projects (if needed)
  - Basics of Reinforcement Learning
  - Markov Processes
  - Reinforcement Learning Categorization
  - OpenAI Gym
  - The Cross Entropy Method

Final Project

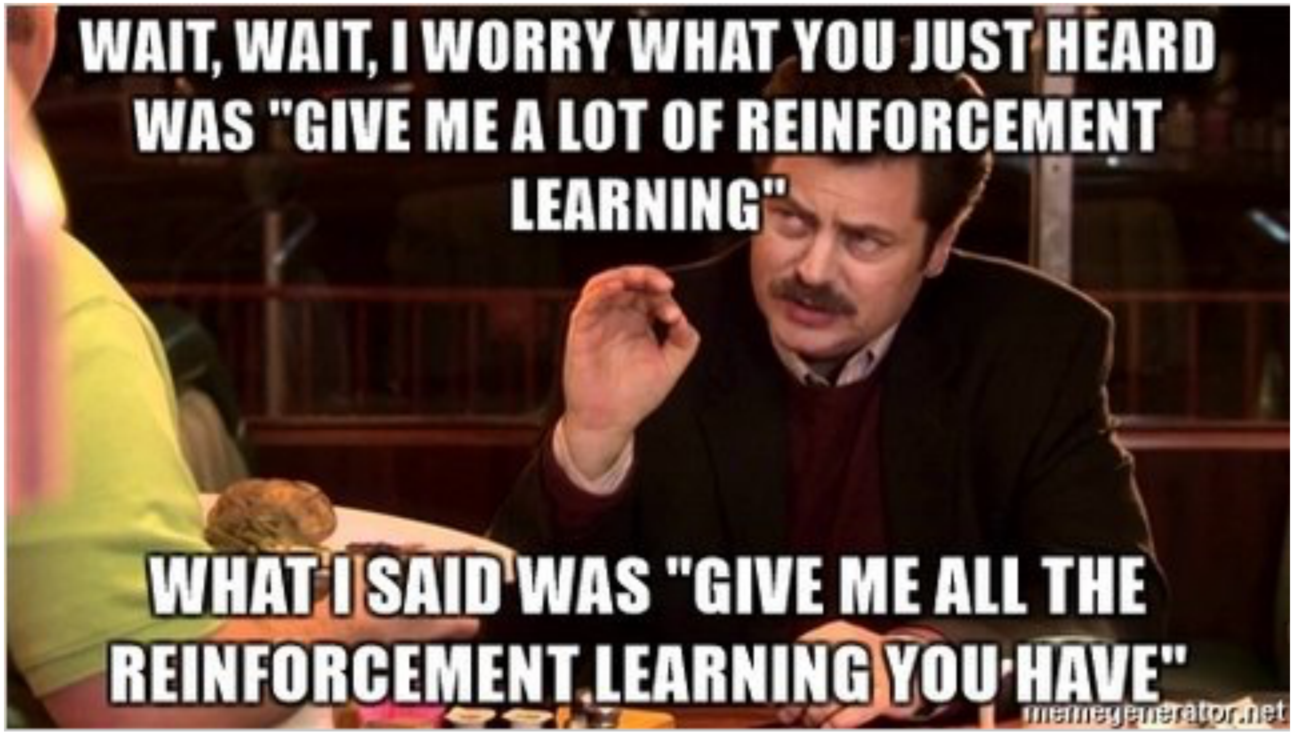
One Idea from Professor Stephanie Langin-Hooper
SMU Meadows



Figurines in Hellenistic Babylonia
Miniaturization and Cultural Hybridity
Stephanie M. Langin-Hooper



Reinforcement Learning Basics


WAIT, WAIT, I WORRY WHAT YOU JUST HEARD WAS "GIVE ME A LOT OF REINFORCEMENT LEARNING"
WHAT I SAID WAS "GIVE ME ALL THE REINFORCEMENT LEARNING YOU HAVE"
memegenerator.net

History of RL from Two Paths

- **Optimal Control**
  - Model processes via Markov property
  - Optimal paths through states calculated through dynamic programming
- **Animal Behavioral Learning (psychology)**
  - Animals learn by trial and error
  - Formalized by Thorndike, 1911. Strengthen through pleasure and weaken through pain
  - Pavlov and B.F. Skinner would conduct experiments proving that behavior could be influenced with RL
  - **Motivation for many pioneering Researchers:** Claude Shannon, J. Deutsch, Marvin Minsky, F. Rosenblatt, Widrow, Hoff

Edward Thorndike

B.F. Skinner

Ivan Pavlov

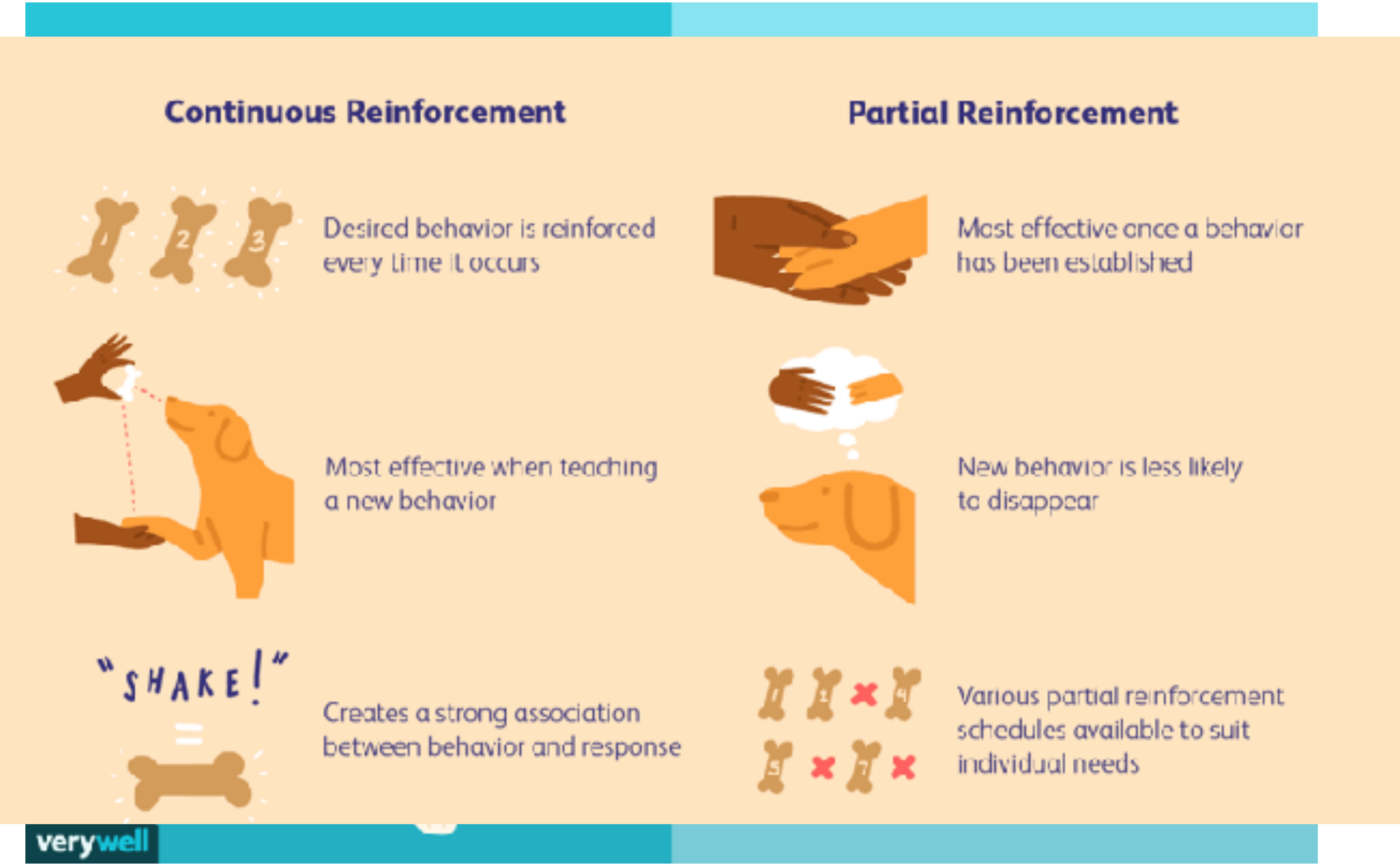
Bernard Widrow

Marvin Minsky

Ted Hoff

Claude Shannon

Conditioning, Skinner and Pavlov



Continuous Reinforcement

Desired behavior is reinforced every time it occurs

Most effective when teaching a new behavior

Creates a strong association between behavior and response

Partial Reinforcement

Most effective once a behavior has been established

New behavior is less likely to disappear

Various partial reinforcement schedules available to suit individual needs

https://www.verywellmind.com/classical-vs-operant-conditioning-2794861

How to condition a machine learning model?

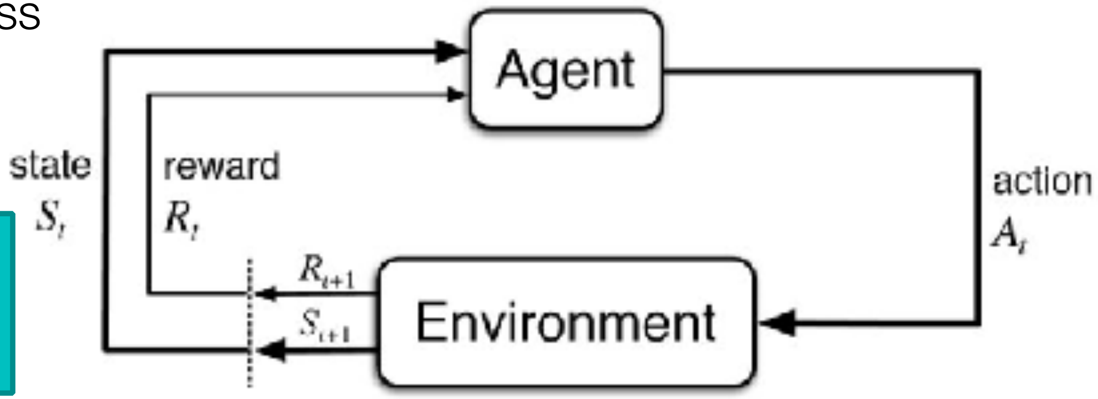
- Hybrid of **Supervised** and **Unsupervised** Learning
- **Reinforcement** Learning
  - Possibly specific labels given, but not necessarily with supervision for how labels are achieved
    - labels are typically stochastic
  - Uses many techniques from supervised learning, but applied towards a slightly different objective function
  - Rewards (positive and negative) are possible to assess behavior in an environment (just like with animals RL)
  - Not specific to Machine Learning community

Generic RL Landscape

- **Agent**
  - Interacts with the environment. Your model guides the Agent’s decisions
- **Environment**
  - Anything that is not the agent
- **Observations**
  - What the agent knows about the environment (usually state)
- **Actions**
  - What an agent can perform with the given environment (possibly stochastic)
- **Rewards**
  - Local measure of success
  - Can compound local rewards over time

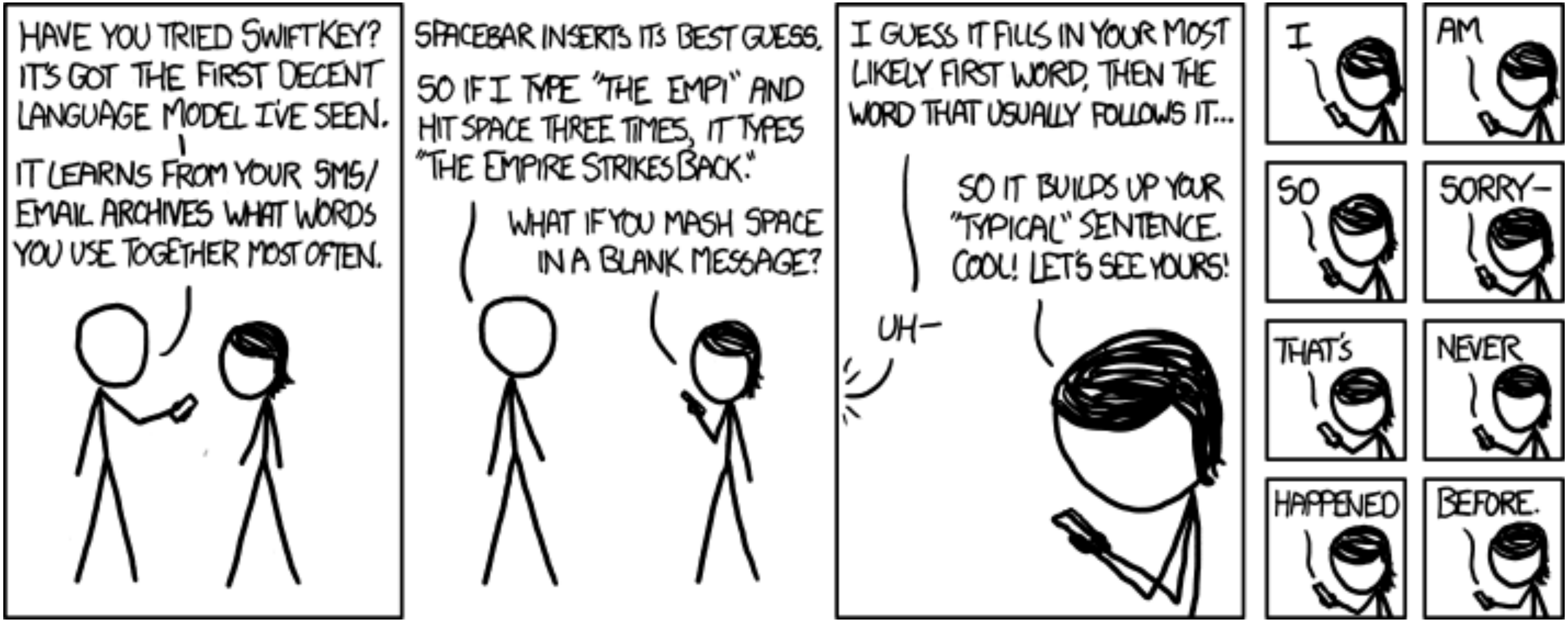
State, Action, Reward, Next State

SARS 😳



https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

Markov Building Blocks


HAVE YOU TRIED SWIFTKEY? IT'S GOT THE FIRST DECENT LANGUAGE MODEL I'VE SEEN.
IT LEARNS FROM YOUR SMS/EMAIL ARCHIVES WHAT WORDS YOU USE TOGETHER MOST OFTEN.
SPACEBAR INSERTS ITS BEST GUESS.
SO IF I TYPE "THE EMPI" AND HIT SPACE THREE TIMES, IT TYPES "THE EMPIRE STRIKES BACK."
WHAT IF YOU MASH SPACE IN A BLANK MESSAGE?
I GUESS IT FILLS IN YOUR MOST LIKELY FIRST WORD, THEN THE WORD THAT USUALLY FOLLOWS IT...
SO IT BUILDS UP YOUR "TYPICAL" SENTENCE. COOL! LET'S SEE YOURS!
UH—
I
AM
SO
SORRY—
THAT'S
NEVER
HAPPENED
BEFORE.

Markov Processes

- **Definition**: Any process that can be explained (or simplified) through a sequential set of states that depend only on the previous state
- **Practical Meaning**: For N states, there will be the probability of transition to any other state, encoded through an NxN transition matrix of discrete probabilities
- State sequences are not deterministic, they are sampled from these distributions
- Despite **simplicity**, they can model a number of real processes with **good enough** precision

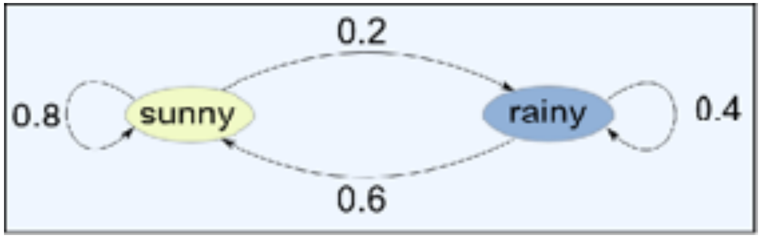
Next State, s_{t+1}

Current State, s_t

0.1	0.2	0.1	0.6	0.0
0.9	0.0	0.1	0.0	0.0
0.0	0.4	0.0	0.4	0.2
0.0	0.4	0.2	0.0	0.4
0.0	0.0	0.6	0.0	0.4

MP Example from Maxim Lapan

	Sunny’	Rainy’
Sunny	0.8	0.2
Rainy	0.6	0.4

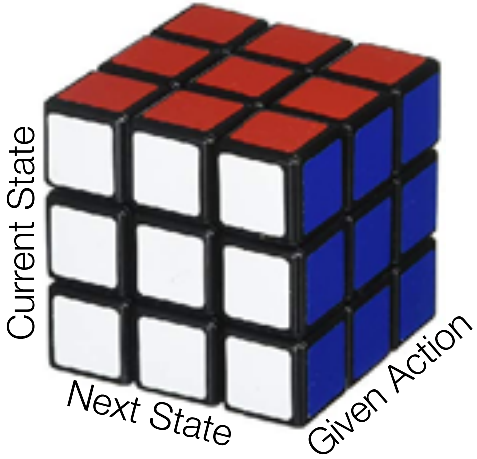


				…	
Sun+Summer					
Rainy+Summer					
Sun+Fall					
Rainy+Fall					
Sun+Else					
Rainy+Else					

Adding One Variable Can Have Drastic Effect on State Space Size

Markov Decision Processes (MDP)

- **New Definition**: any state to state transition can be altered by an action that is given by a Markov Process
- **Definition**: An MDP consists of:
  - Env. States, s_t
  - Actions set for each time $\boldsymbol{a}_t$
  - Reward function for each state, $\boldsymbol{r}(s_t)$
  - A transition model, $P(s_{t+1}, s_t \mid a)$ a matrix of probabilities
    - Not ***guaranteed*** next state by given action



Markov Reward Process (MRP)

- **Total reward** is given by sum of all rewards in sequence

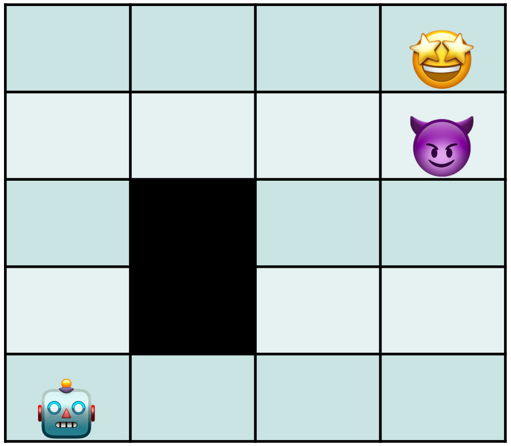
$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_k \gamma^k R_{t+k+1}$$

- Gamma defines future reward far- and short-sightedness
  - Common values are **0** (short), **0.9**, **0.99**, and **1** (far)
- This reward calculation can be used to estimate the "**Value**" of each state based upon the average total reward a state *should* give, $V(s) = \mathbf{E}[G \mid s_t{=}s]$
- Typically, this value must be estimated from the model over fixed sequences, otherwise some reward values can become arbitrarily large by looping actions

MDPs and MRPs

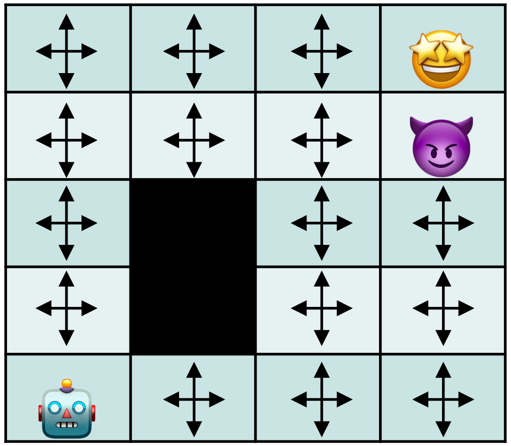
- The million dollar question:
  How do we select a good action given a current state?
- If γ is not 0, this can get really complicated as we need to look at all possible future actions to measure value
- Instead of defining what is optimal, let's instead setup a comparison of different actions we might take (**policy**)
- A **policy** is defined as $\pi(a, s) = P(a_t = a \mid s_t = s)$
  - Given the current state, we have a certain probability of selecting each action
  - Action selection is **probabilistic**, but easy to discover **deterministic** actions (*set one action to 1.0, all others to 0.0*)
- Try different policies, select one with best average reward

An Illustrative Example: Grid World

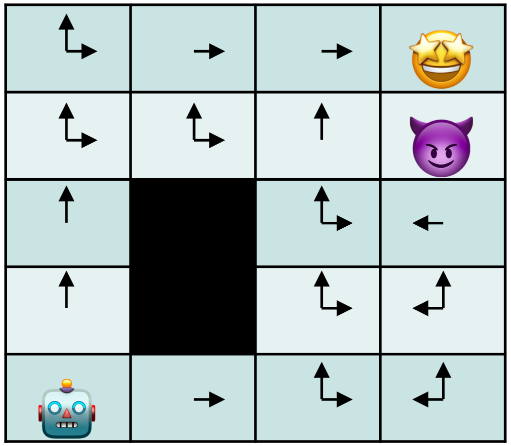


- **State**: Every square in grid
- **Action**: Move to make (l,r,u,d), *with probability*
- **Reward**: Goal, Death
- **Policy**: Given state, where should we move?
- **Optimal Policy**:

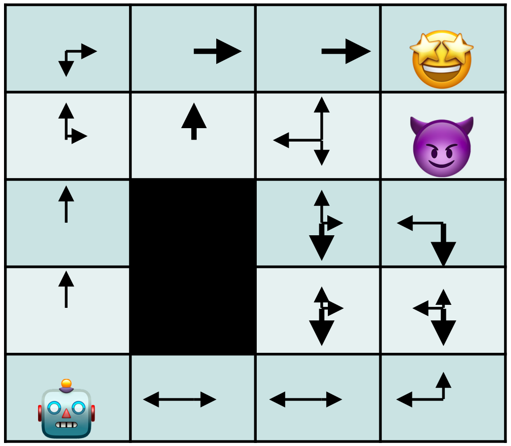
$$\pi^* = \arg\max_{\pi} \mathbf{E}\left[\sum_k \gamma^k R_{t+k+1} \,|\, \pi\right]$$



Random Policy

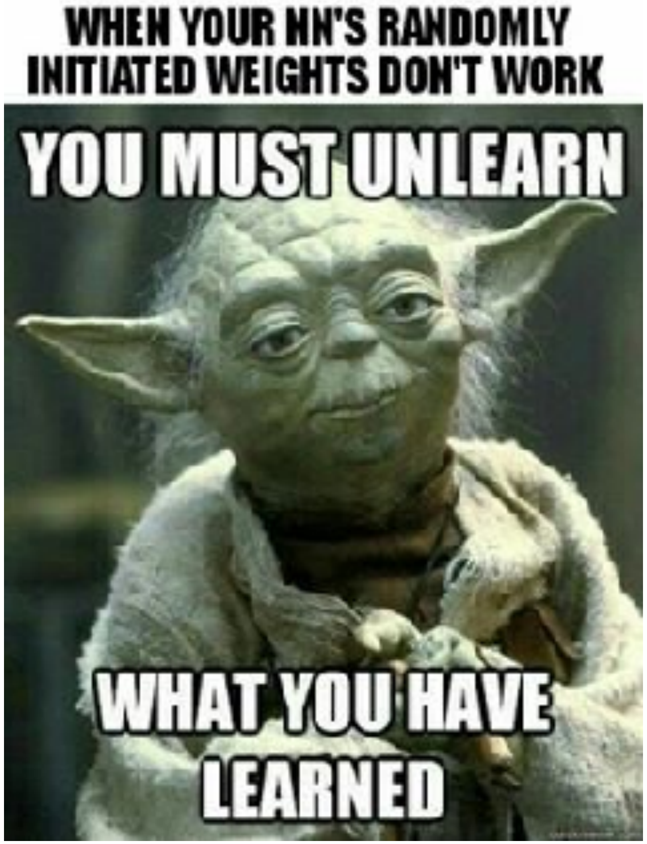


Another Policy



Another Policy

RL Categorization



RL Categorizations

- Model-based versus Model-free
- Policy-based versus Value-based
- On-Policy, Off-Policy

- On-policy
  - We must interact with environment to learn a policy
- Off-policy
  - Can learn also from historical data or humans

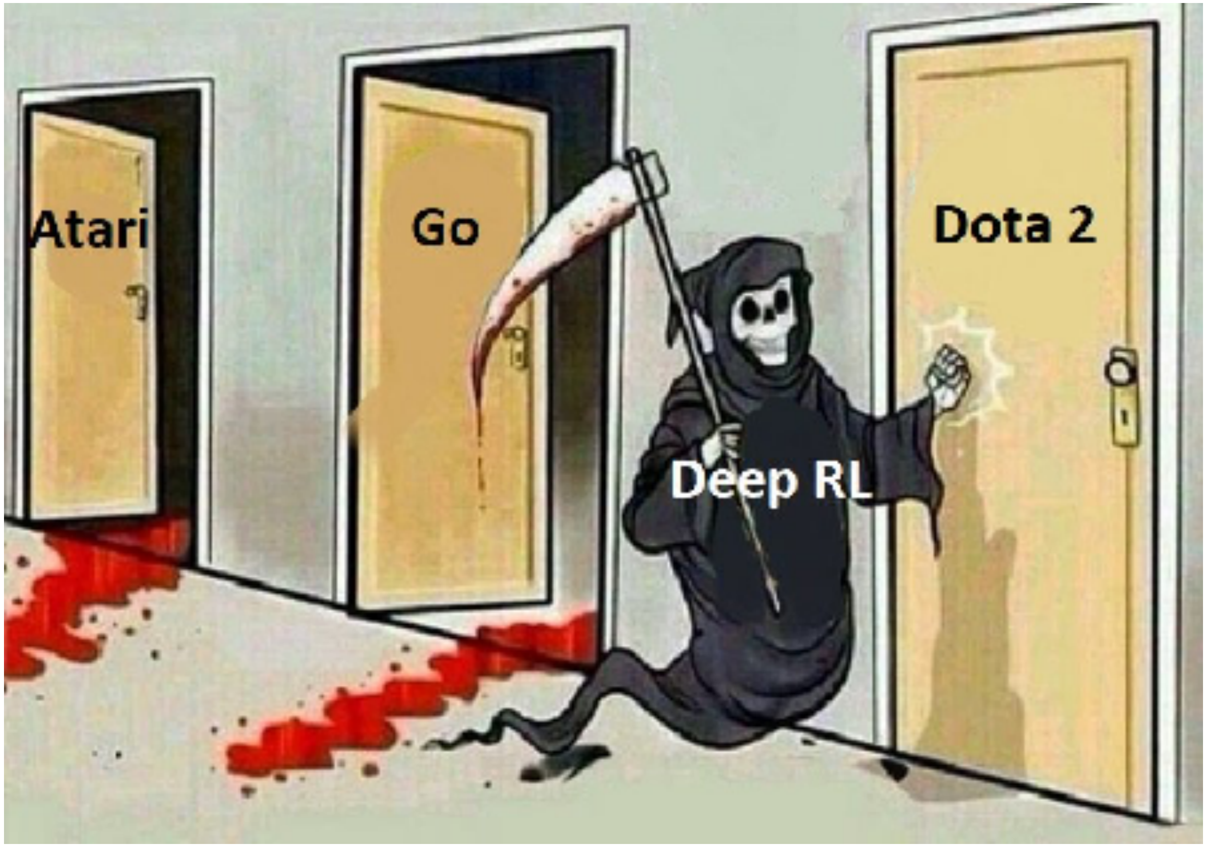
Model-based versus Model-free

- Model Based
    - Predict the next observation and reward based on an understanding (model) of the rules in environment
    - Often look a number of moves ahead (like in chess or similar game)
    - Hard to construct in complex environments
    - NOT what we will be studying… needs domain expertise
- Model Free
    - Don’t care what the environment is
    - Directly try to connect observations to actions (or values from which an action can be inferred)
    - Just use a neural network! That is our style!
- Mixed: Sure, like Alpha-Go

Policy Based versus Value Based

- Policy Based Learning
  - Directly approximate the policy of the agent
  - Policy is typically a probability distribution of actions that we sample from for next action
  - Could also be a “see this, do that” configuration
- Value Based
  - Calculate an intermediate value function for all possible actions
  - Policy becomes choosing the best action based on value function

OpenAI Gym

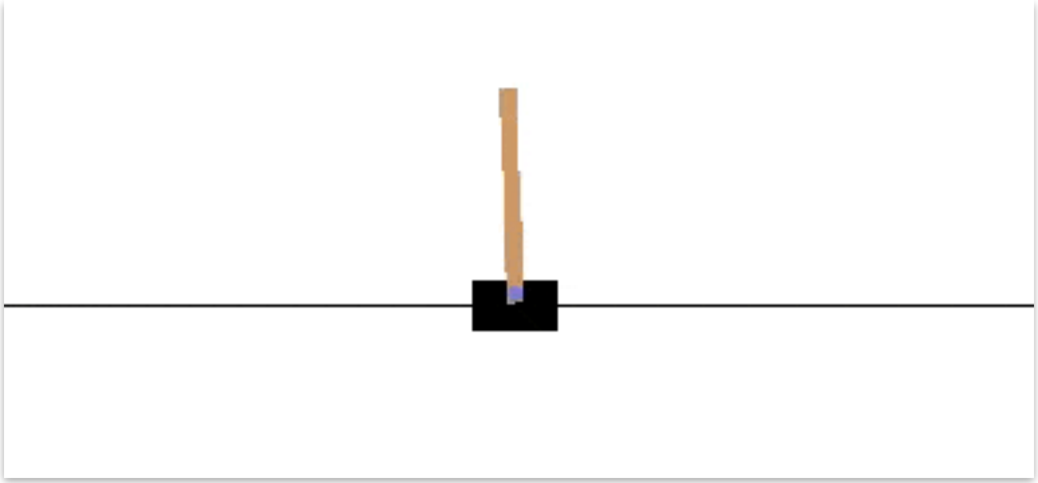


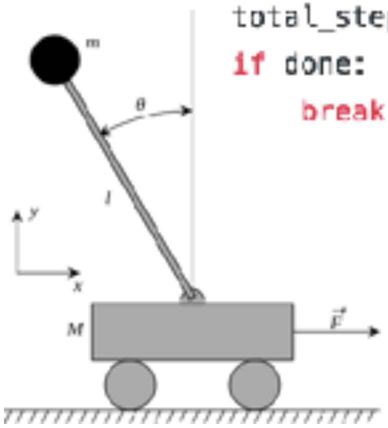
Object Oriented RL

- Basics:
  - Define object instance for `Agent()` and the `Env()`
  - Define what observations will return
  - Run `env.step(action)`
  - Get new observations and reward from env
- `action_space` and `observation_space`
  - Possible actions to execute, Observations to get
  - Discrete or continuous?
  - Can actions be given simultaneously?

Basics of Cartpole

```python
import gym


if __name__ == "__main__":
    env = gym.make("CartPole-v0")

    total_reward = 0.0
    total_steps = 0
    obs = env.reset()

    while True:
        action = env.action_space.sample()
        obs, reward, done, _ = env.step(action)
        total_reward += reward
        total_steps += 1
        if done:
            break
```





Action Space: One input, [0, 1] pull left or pull right

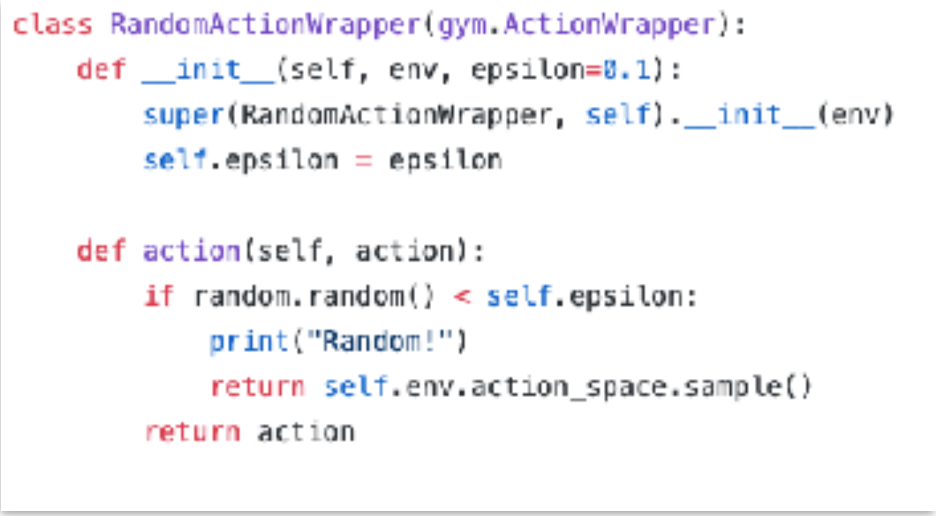
Obs Space: Dynamic state variables (continuous and four dimensional)

End: When more than 15 degrees off or too far from center

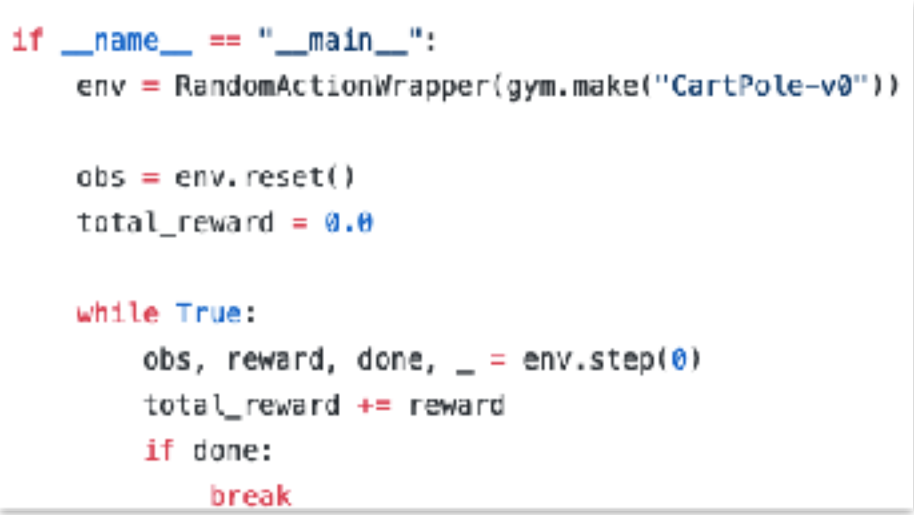
Reward: +1 for each time step

Wrapping the Environment

- When you want some extra action, observation, reward processing
- Expose function with `ActionWrapper`, `RewardWrapper`, `ObservationWrapper`



```
class RandomActionWrapper(gym.ActionWrapper):
    def __init__(self, env, epsilon=0.1):
        super(RandomActionWrapper, self).__init__(env)
        self.epsilon = epsilon

    def action(self, action):
        if random.random() < self.epsilon:
            print("Random!")
            return self.env.action_space.sample()
        return action
```

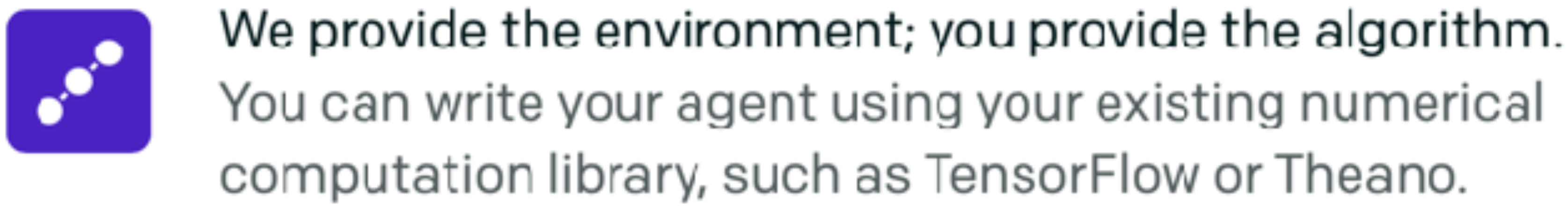


```
if __name__ == "__main__":
    env = RandomActionWrapper(gym.make("CartPole-v0"))

    obs = env.reset()
    total_reward = 0.0

    while True:
        obs, reward, done, _ = env.step(0)
        total_reward += reward
        if done:
            break
```

Might return different action than user supplied with small probability

OpenAI Gym

https://gym.openai.com

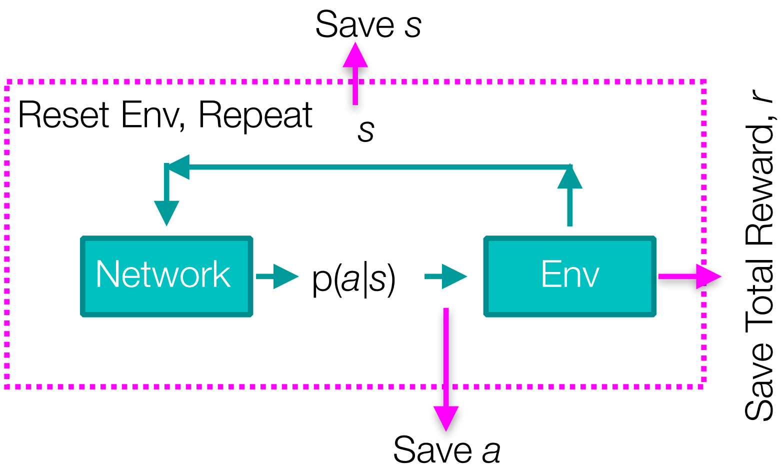


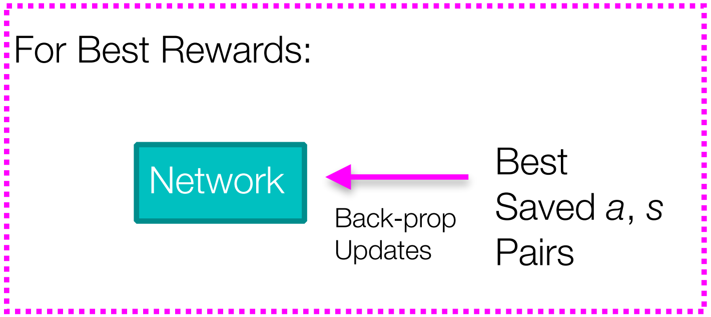
We provide the environment; you provide the algorithm. You can write your agent using your existing numerical computation library, such as TensorFlow or Theano.

Cross Entropy Method

Optimize Best Random Models

- Create a random neural network, with output $p(a|s)$
- Let it interact with the environment (randomly)
    - For some set of episodes (*e.g.*, 20)
        - Use network output to sample from possible actions
        - Run episode to completion
        - Repeat
- Calculate reward for each episode
- Keep best episodes (some percentile, e.g., best five)
- For the given best episodes, develop loss function incentivizing the actions taken based upon the input observations





Repeat until desired performance!

Cross Entropy Method

- Model based or Model Free?
  - Model Free (no assumptions of problem)
- Value or Policy Based?
  - Policy Based (randomly sample actions based on policy)
- On-policy or Off-Policy?
  - On-Policy (need to interact with environment to get better)
- Has some similarity to **Simulated Annealing** Optimization

How to Make this More Mathy?

- If we have the optimal policy p(x) and a reward function H(x), then maximize

$$\mathbf{E}_{x\leftarrow p(x)}[H(x)] = \mathbf{E}_{x\leftarrow q(x)}[\frac{p(x)}{q(x)}H(x)]$$

- We can approximate the distribution by: $\frac{1}{N}\sum_{i}\frac{p(x_i)}{q(x_i)}H(x_i)$
- Proven that this is optimized when $\mathbf{KL}(\ q(x)\ ||\ p(x)H(x)\ )$ is minimized. But its intractable, so we drop terms … and end up just minimizing (neg) cross entropy of samples

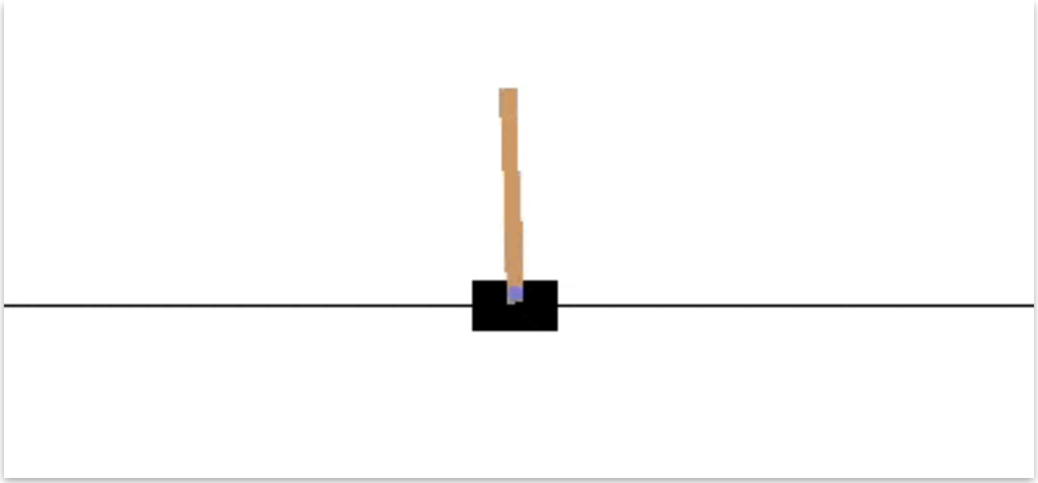
Performance Measure

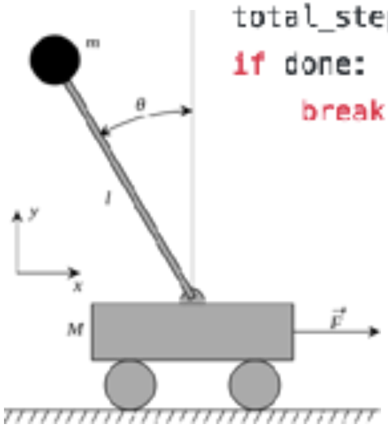
$$\pi_{k+1}(a\,|\,s) = \arg\max_{\pi_k} \mathbf{E}_{z\leftarrow\pi_k}[\mathbf{1}_{R(z)>\psi} \log \pi_k(a\,|\,s)]$$

min CrossEntropy(*net_actions*, *best_actions*)

Review: Basics of Cartpole

```
import gym


if __name__ == "__main__":
    env = gym.make("CartPole-v0")

    total_reward = 0.0
    total_steps = 0
    obs = env.reset()

    while True:
        action = env.action_space.sample()
        obs, reward, done, _ = env.step(action)
        total_reward += reward
        total_steps += 1
        if done:
            break
```





Action Space: One input, [0, 1] pull left or pull right

Obs Space: Dynamic state variables (continuous and four dimensional)

End: When more than 15 degrees off or too far from center

Reward: +1 for each time step

Cross Entropy Reinforcement Learning

M. Lapan Implementation for CartPole and Frozen Lake

```
Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb
```

Lecture Notes for

Neural Networks and Machine Learning

Intro to Reinforcement Learning

Next Time:
Q-Learning

Reading: Lapan CH4-CH6