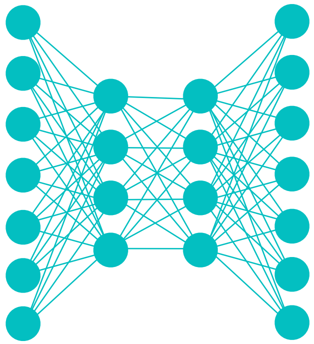
Lecture Notes for

Neural Networks and Machine Learning

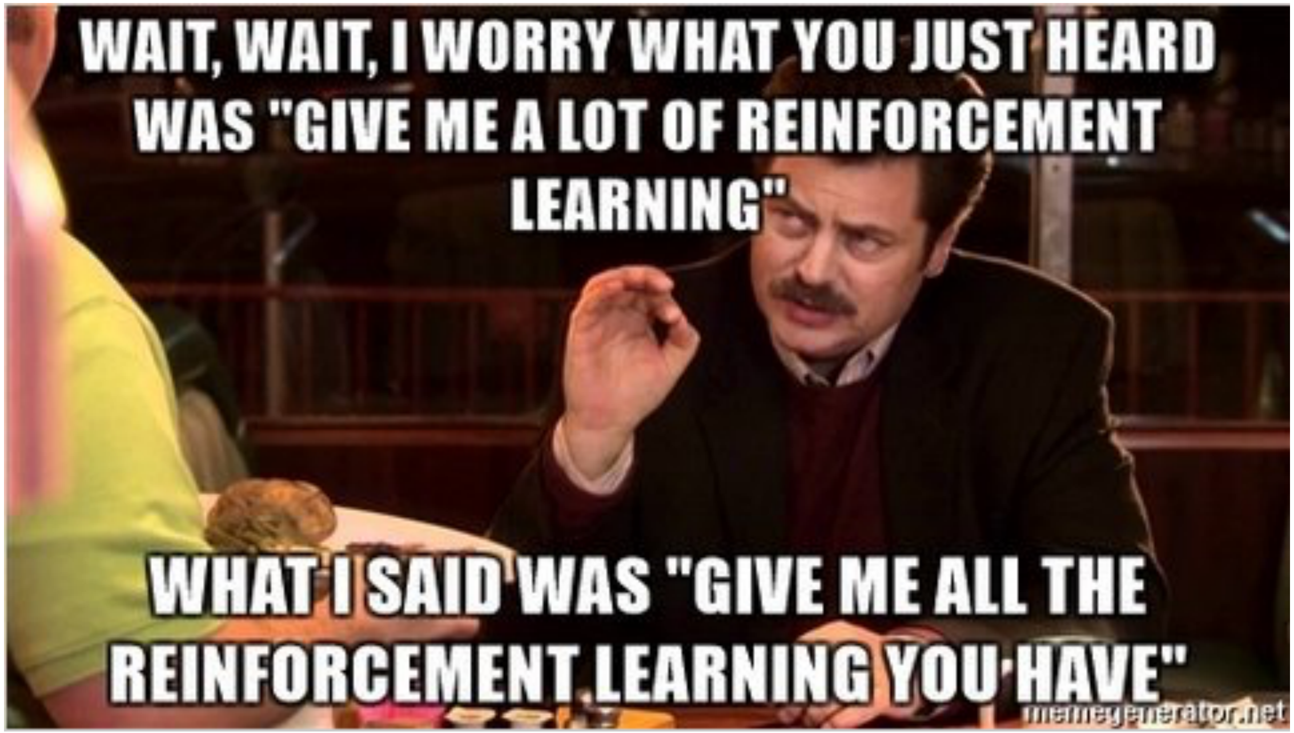
Introduction to Reinforcement Learning



Logistics and Agenda

- Logistics
  - Grading Update
- Agenda
  - Basics of Reinforcement Learning (4 slides)
  - Markov Processes and Markov Rewards
  - Reinforcement Learning Categorization
  - OpenAI Gym
  - The Cross Entropy Method

Reinforcement Learning Basics



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

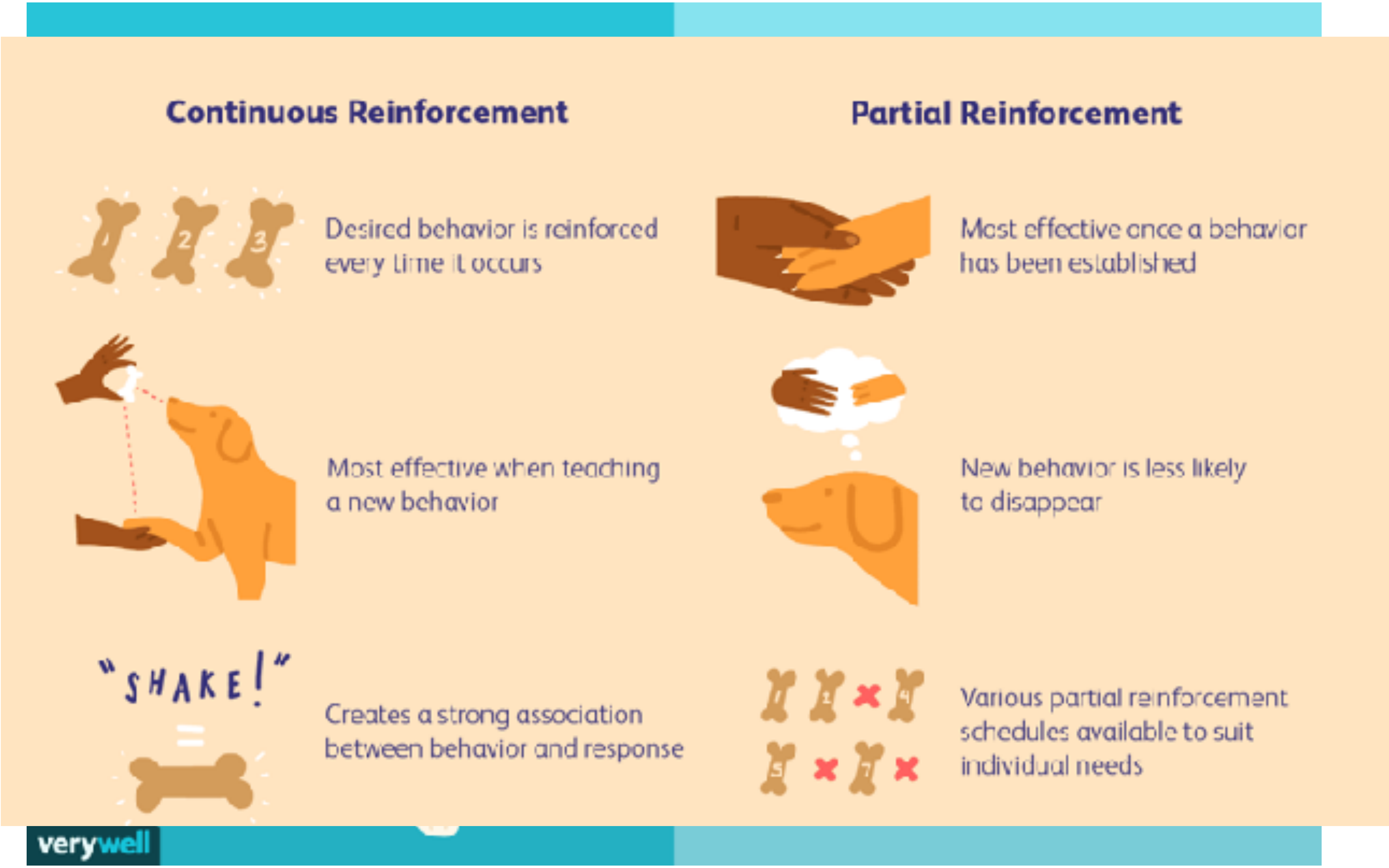

Bernard Widrow


Marvin Minsky


Ted Hoff


Claude Shannon

Conditioning, Skinner and Pavlov



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 can also be probabilistic
  - Uses many techniques from supervised learning, but applied towards a **different objective function**
  - **Rewards** (positive and negative) are possible to assess behavior in an environment
  - **Not specific** to Machine Learning community, is a major part of optimization, control, and **psychology**

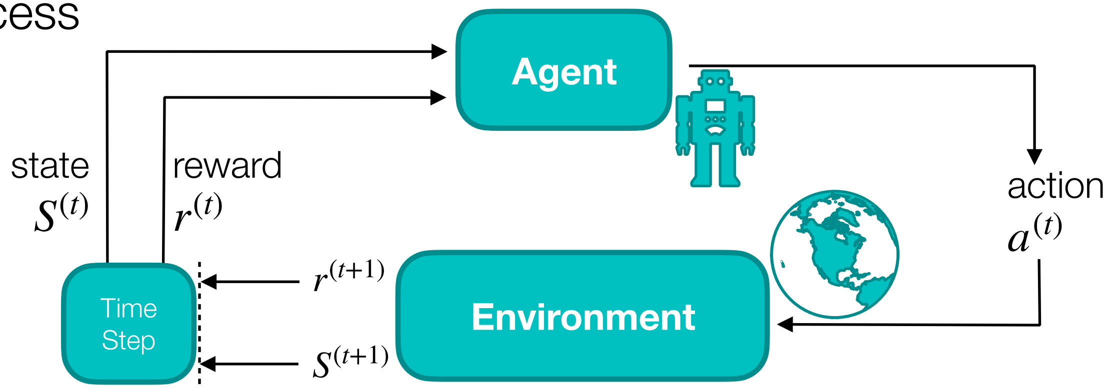
Generic RL Landscape

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

State, Action, Reward, Next State

SARS 😳

`Agent`$(S^{(t)}, r^{(t)}) \rightarrow a^{(t)}$



`Env`$(a^{(t)}) \rightarrow (S^{(t+1)}, r^{(t+1)})$

OpenAI Gym

Object Oriented Agent and Environment

- 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 multiple actions be given simultaneously?

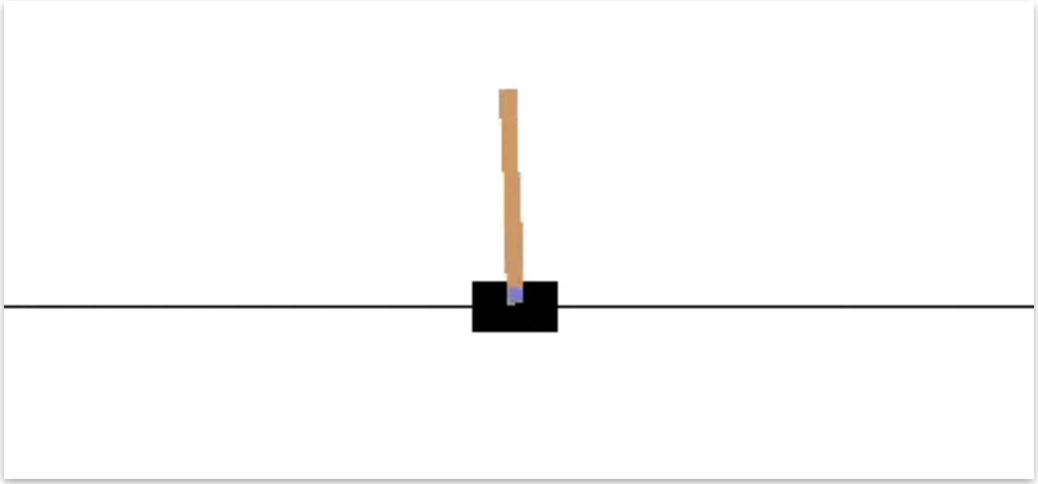
Basics of Cartpole

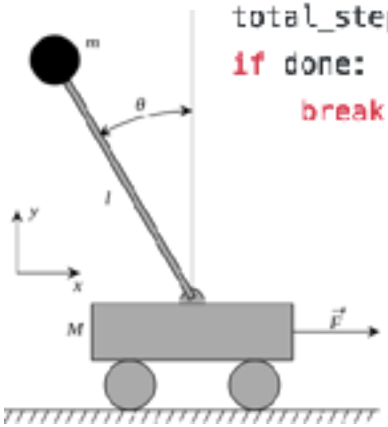
```python
import gym


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`

```python
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
```

```python
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,

RL Categorization

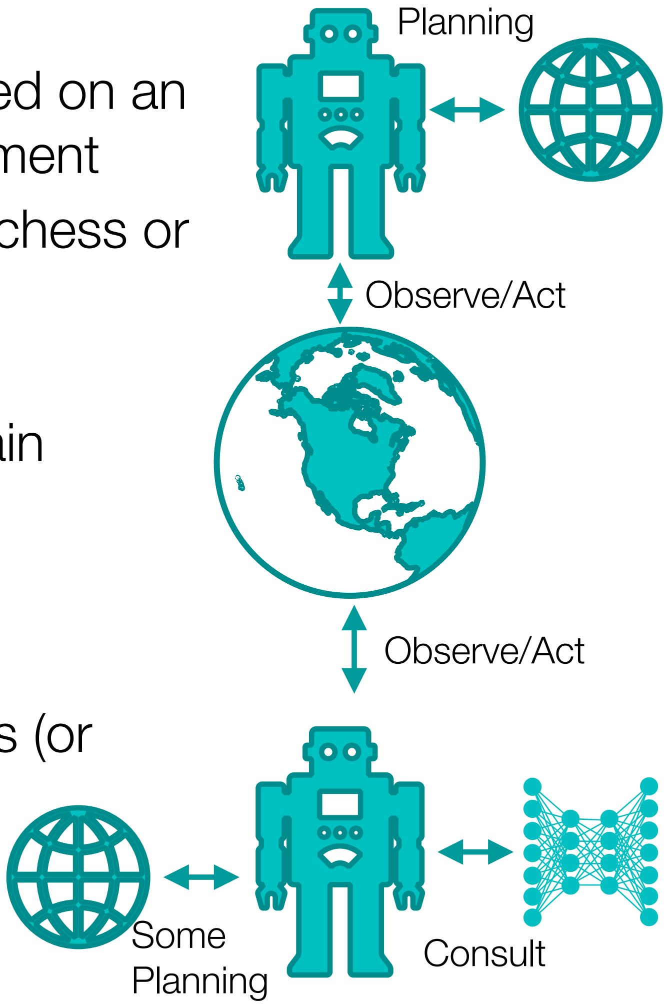
RL Categorizations

- 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
- Policy-based versus Value-based

Model-based versus Model-free

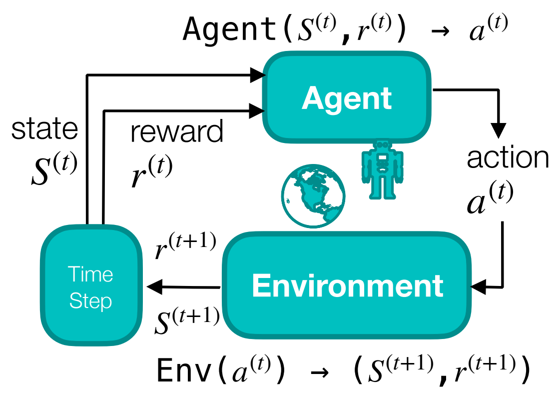
- **Model Based**
    - Predict the next observation and reward based on an understanding (model) of the rules in environment
    - Often look a number of moves ahead (like in chess or similar game)
    - Hard to construct in complex environments
    - NOT what we will be studying… needs domain expertise
- **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! Perhaps better generalization?
- **Mixed**: Many examples, yes (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
  - Iterate over possible action values to choose action
  - Policy becomes choosing the best action based on value function

$$a^{(t)} \leftarrow f(S^{(t)}_{env})$$



$$v^{(t)} \leftarrow f(S^{(t)}_{env})$$

$$a^{(t)} \leftarrow \arg\max_{v} v^{(t)}$$

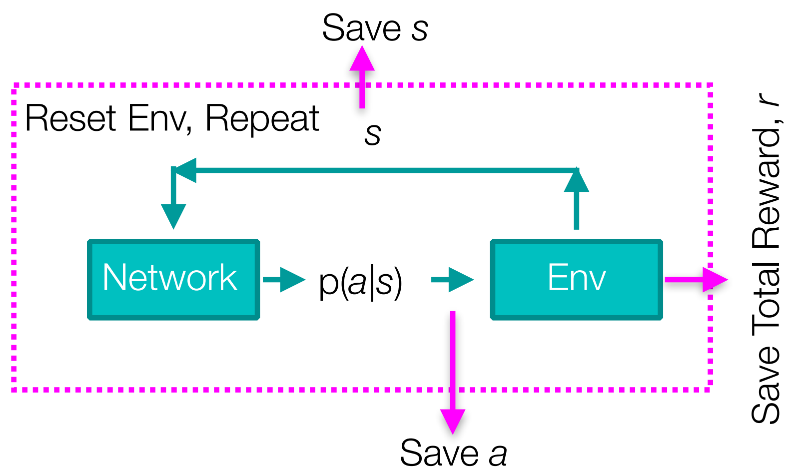
Cross Entropy Method

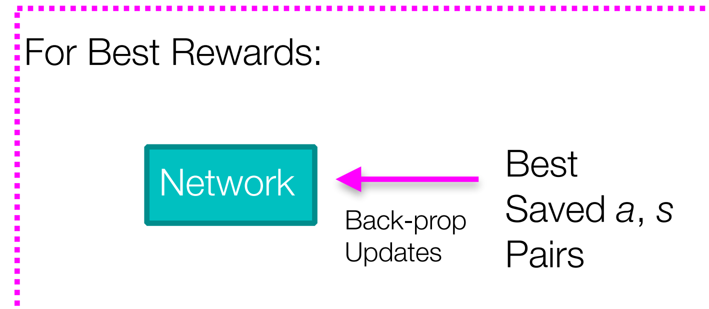
Direct Policy Exploration and Optimization

- Instead of defining what is optimal, just 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
- First try: Cross Entropy Method

Cross Entropy Method

- Create a **random neural network**, with output $p(a|s)$
- Let it **interact** with the **environment** (randomly) for 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)

Loss Function

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

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

- 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 $D_{KL}\left(q(x) \| p(x)\right)$ is minimized. But its intractable, so we can only optimize upper bound … minimizing (neg) cross entropy of samples

Performance Measure

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

min CrossEntropy(*neural_net_actions*, *best_actions*)

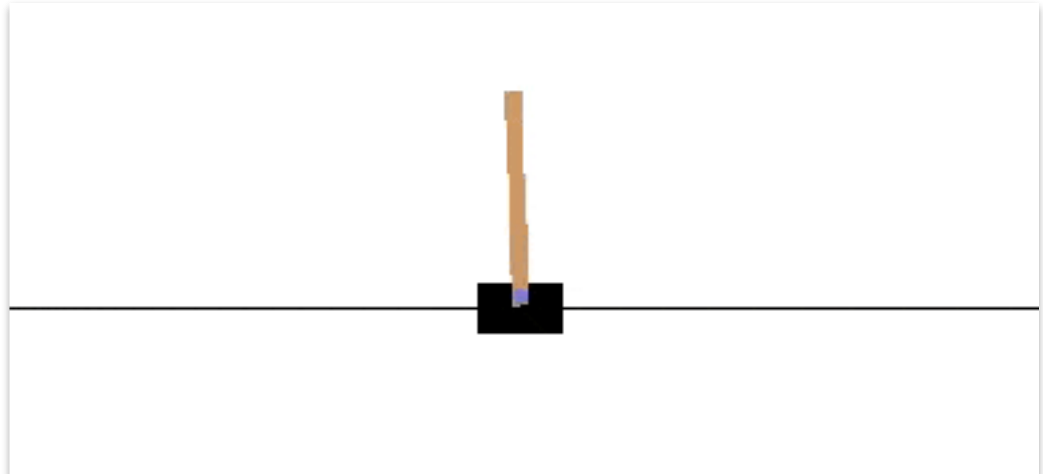
Review: Basics of Cartpole

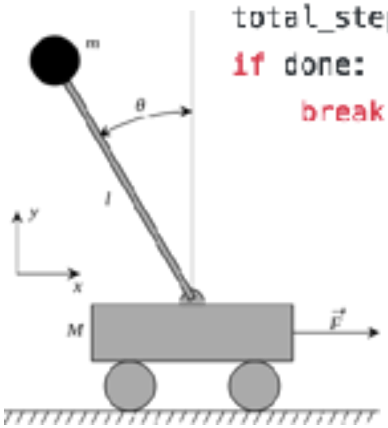
```python
import gym


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
```





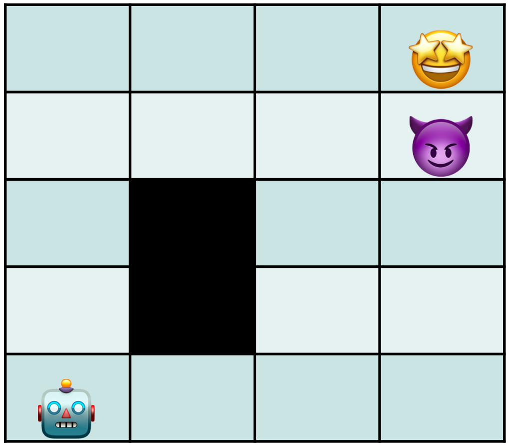
Action Space: One input, [0, 1] pull left or pull right

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

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

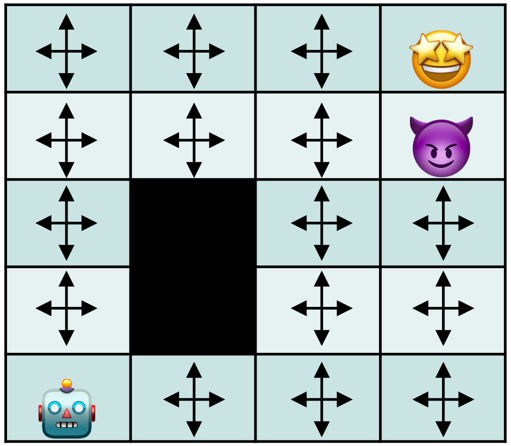
Reward: +1 for each time step

Another Example: Frozen Lake

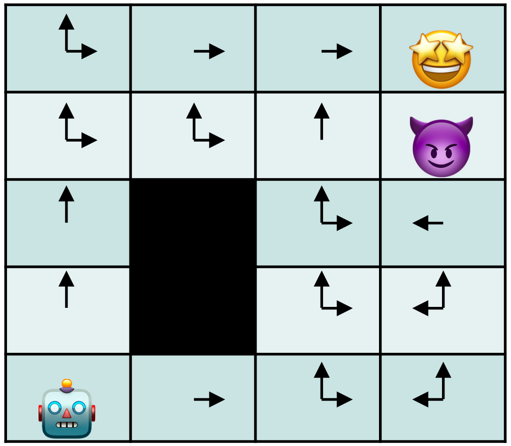


- **State**: Every square in grid
- **Action**: Move l,r,u,d *with probability in 3 axes*
- **Reward**: Goal, Death
- **Policy**: Given state, where should we move?
- **Optimal Policy**:

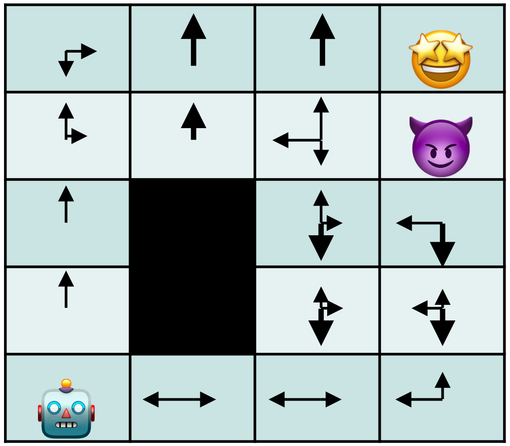
$$\pi^* = \arg\max_{\pi} \mathbf{E}\left[\sum_k \gamma^k R_{t+k+1} \mid \pi\right]$$



Random Policy



Another Policy



Another Policy

Cross Entropy Reinforcement Learning

M. Lapan Implementation for CartPole and Frozen Lake

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