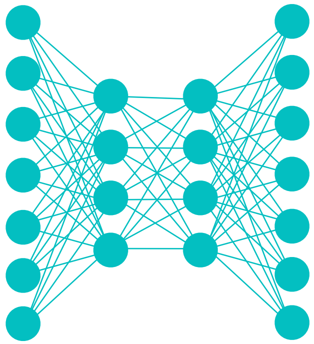
Lecture Notes for

Neural Networks and Machine Learning

Introduction to Reinforcement Learning



Logistics and Agenda

- Logistics
  - Grading Update
  - Finish Student Presentation Next Time
- Agenda
  - Final Lab Town Hall
  - Basics of PyTorch (4 slides)
  - Basics of Reinforcement Learning (4 slides)
  - Markov Processes and Markov Rewards
  - Reinforcement Learning Categorization
  - OpenAI Gym
  - The Cross Entropy Method

Basics of Pytorch

Wait, why are we switching to Pytorch?

- Well, its good to know more than just Tensorflow
- Pytorch has some distinct advantages:
  - No need to setup a static computation graph—graph can be dynamic (like eager execution)
    - Lazy computations still happen on dynamic graph
  - Integration with numpy code on the fly is much easier and faster (compared to TF)
    - Can tradeoff computations with numpy easily, though not neccessarily with autograd
- Also, all the book examples are in Pytorch… so this is nice for following along with the examples

	Keras	TensorFlow	PyTorch
Level of API	high-level API[1]	Both high & low level APIs	Lower-level API[2]
Speed	Slow	High	High
Architecture	Simple, more readable and concise	Not very easy to use	Complex[3]
Debugging	No need to debug	Difficult to debugging	Good debugging capabilities
Dataset Compatibility	Slow & Small	Fast speed & large	Fast speed & large datasets
Popularity Rank	1	2	3
Uniqueness	Multiple back-end support	Object Detection Functionality	Flexibility & Short Training Duration
Created By	Not a library on its own	Created by Google	Created by Facebook[4]
Ease of use	User-friendly	Incomprehensive API	Integrated with Python language
Computational graphs used	Static graphs	Static graphs	Dynamic computation graphs[5]

Pytorch General Flow Training Flow

- Inherit from `torch.nn.Module`
- Define `__init__` and `forward`
- Run epochs in a loop with explicit calls to:
  - `loss` creation (for batch) as variable
  - `loss.backward()` calculation, gradient for batch
  - `optimizer.step()` of optimizer for batch
  - Gives a great deal of flexibility to design and optimization process
- Lots of different pythonic ways to carry this out
  - Your book likes to setup steps of model through iterators (**yield** the batch, loss, etc.)

A Simple Definition (much like Keras!)

```
import torch
import torch.nn as nn

class OurModule(nn.Module):
    def __init__(self, num_inputs, num_classes, dropout_prob=0.3):
        super(OurModule, self).__init__()
        self.pipe = nn.Sequential(
            nn.Linear(num_inputs, 5),
            nn.ReLU(),
            nn.Linear(5, 20),
            nn.ReLU(),
            nn.Linear(20, num_classes),
            nn.Dropout(p=dropout_prob),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        return self.pipe(x)

if __name__ == "__main__":
    net = OurModule(num_inputs=2, num_classes=3)
    print(net)
    v = torch.FloatTensor([[2, 3]])
    out = net(v)
    print(out)
    print("Cuda's availability is %s" % torch.cuda.is_available())
    if torch.cuda.is_available():
        print("Data from cuda: %s" % out.to('cuda'))
```

Sequential Definitions

Common Functions

The MNIST Example (like gradient tape)

Functional Definitions

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Definitions

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)
```

`view(*shape) → Tensor` reshape without copy

https://github.com/pytorch/examples/blob/master/mnist/main.py

Training One Epoch

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Training Multiple Epochs

```
model = Net().to( “cpu” )
optimizer = optim.SGD(model.parameters()

for epoch in range(1, args.epochs + 1):
    train(args, model, “cpu” , train_loader, optimizer, epoch)
```

Utils

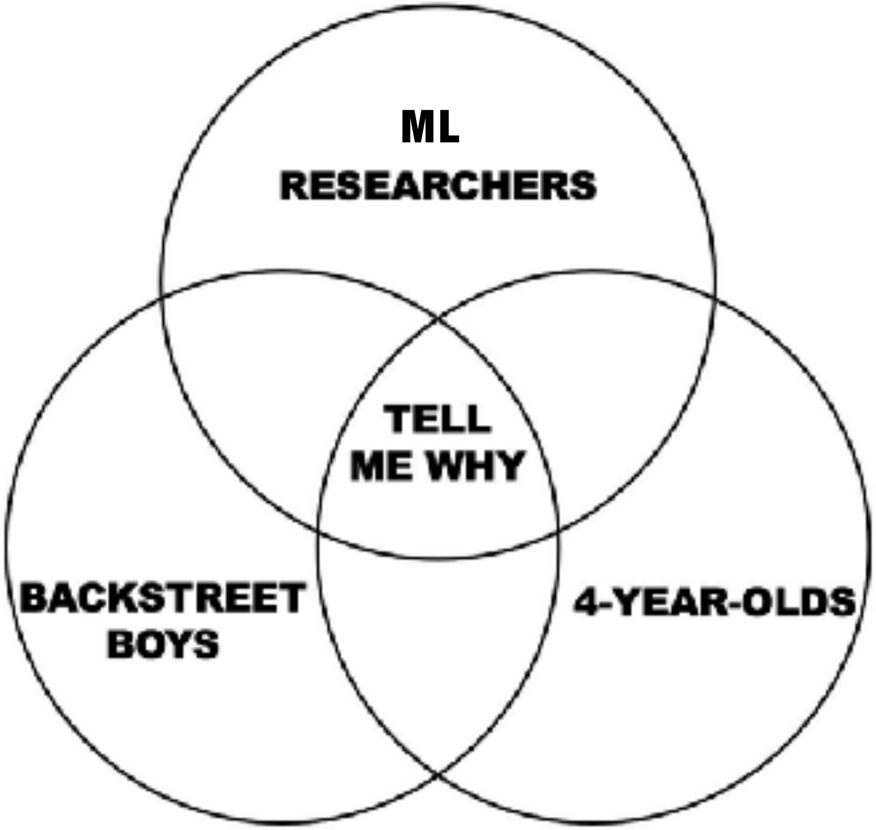
```
from torchvision import datasets, transforms
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=args.batch_size, shuffle=True, **kwargs)
```

mean, std

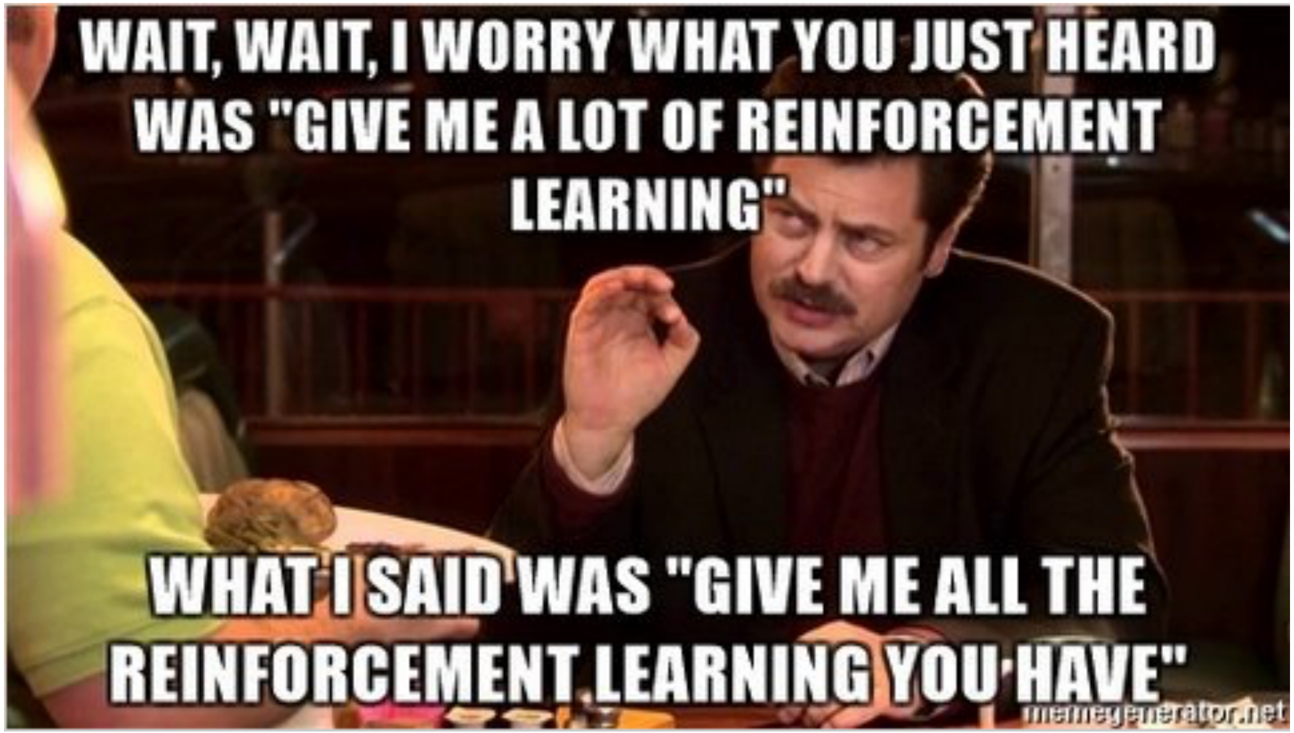
PyTorch

- In the past: we would have used this for GANs and this would not be the introduction (but still high level)
- We will go more into this as needed for demonstrations using RL (skipping code for brevity)
- Its only a tool for optimizing and defining a neural network
- And its nice to know Tensorflow and PyTorch so that you can work with either

Final Project Draft Town Hall



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

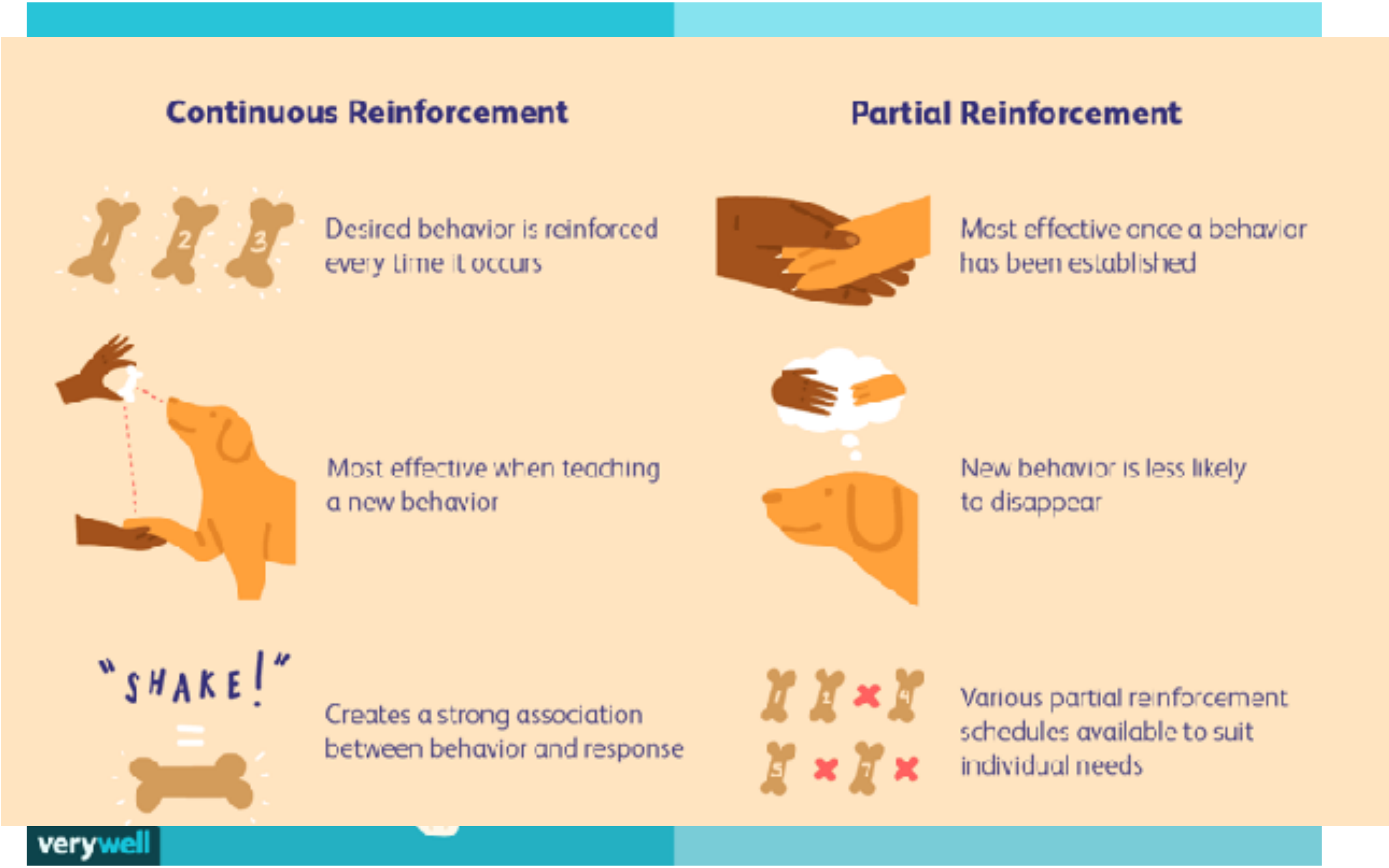

Bernard Widrow


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?

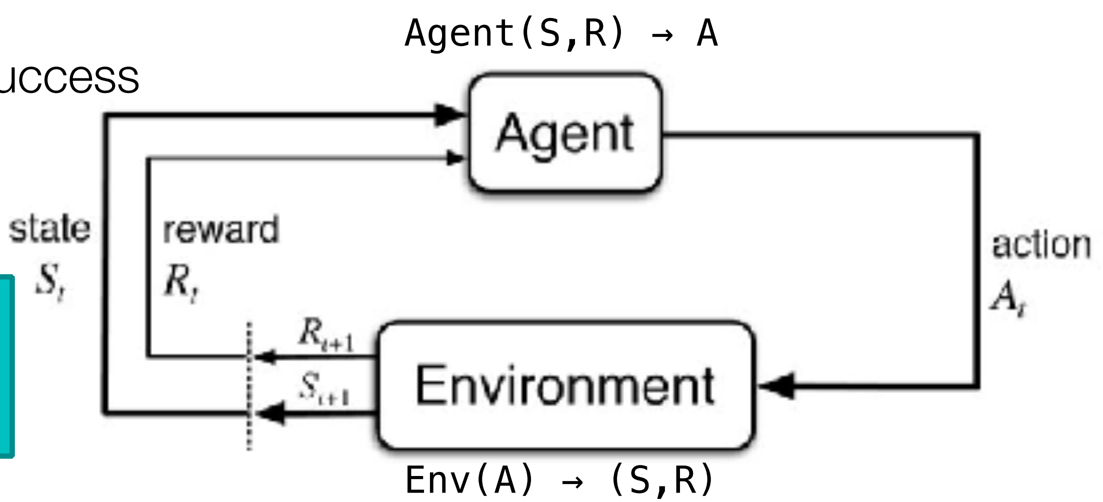
- Hybrid of **Supervised** and **Unsupervised** Learning
- **Reinforcement** Learning
  - Possibly "specific" labels given, but not necessarily with supervision for how labels are achieved
    - labels can also be probabilistic
  - Uses many techniques from supervised learning, but applied towards a 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 😳



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

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?

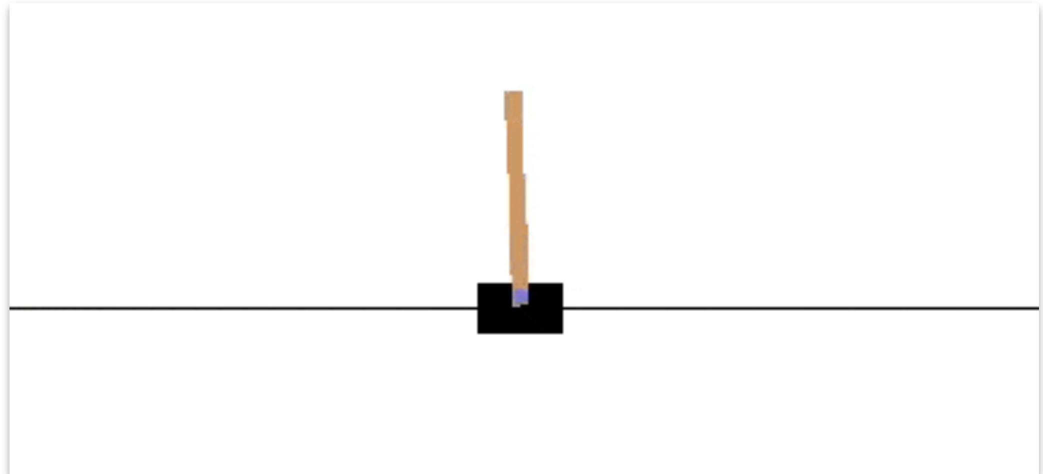
Basics of Cartpole

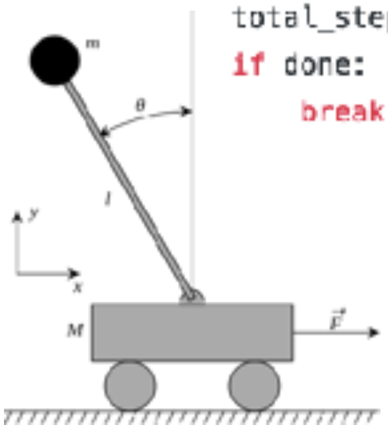
```python
import gym


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`

```
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,

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! 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
  - Iterate over possible action values to choose action
  - Policy becomes choosing the best action based on value function

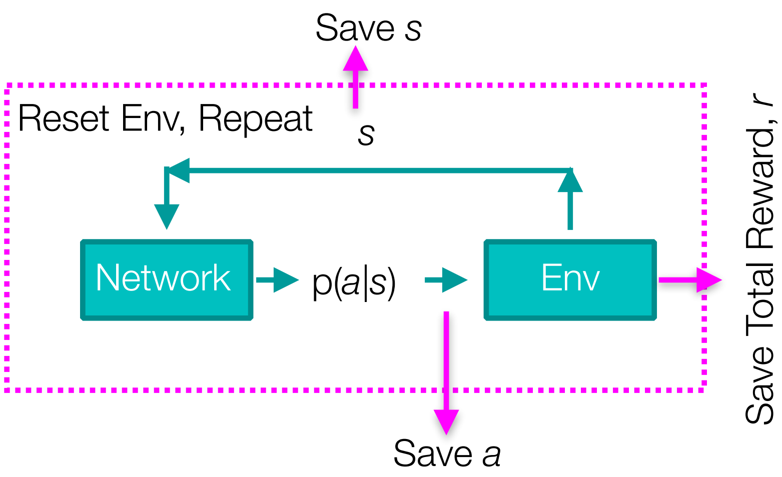
Cross Entropy Method

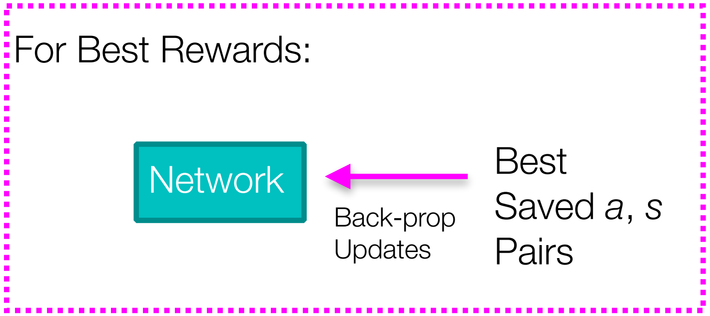
Direct Policy Exploration and Optimization

- 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 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

Mathematical Motivation

- 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 can only optimize upper bound … minimizing (neg) cross entropy of samples

Performance Measure

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

min CrossEntropy(*neural_net_actions*, *best_actions*)

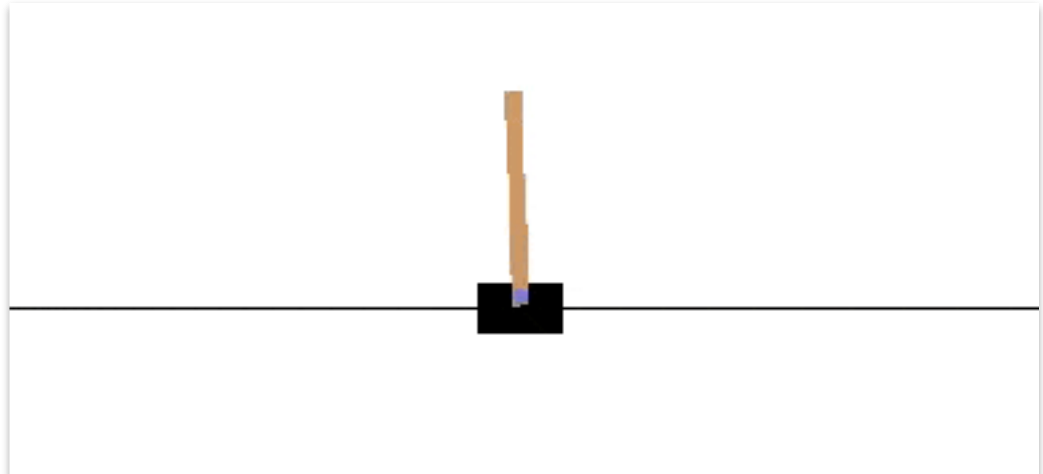
Review: Basics of Cartpole

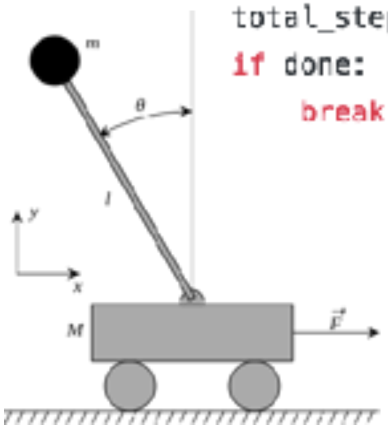
```python
import gym


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

Cross Entropy Reinforcement Learning

M. Lapan Implementation for CartPole and Frozen Lake

Follow Along:
`08a_Basics_Of_Reinforcement_Learning.ipynb`