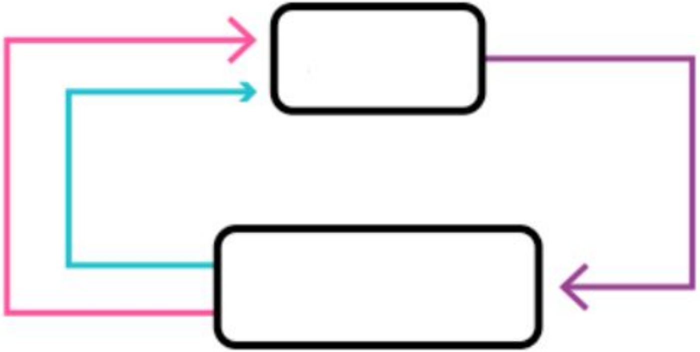


A Short Introduction To Reinforcement Learning

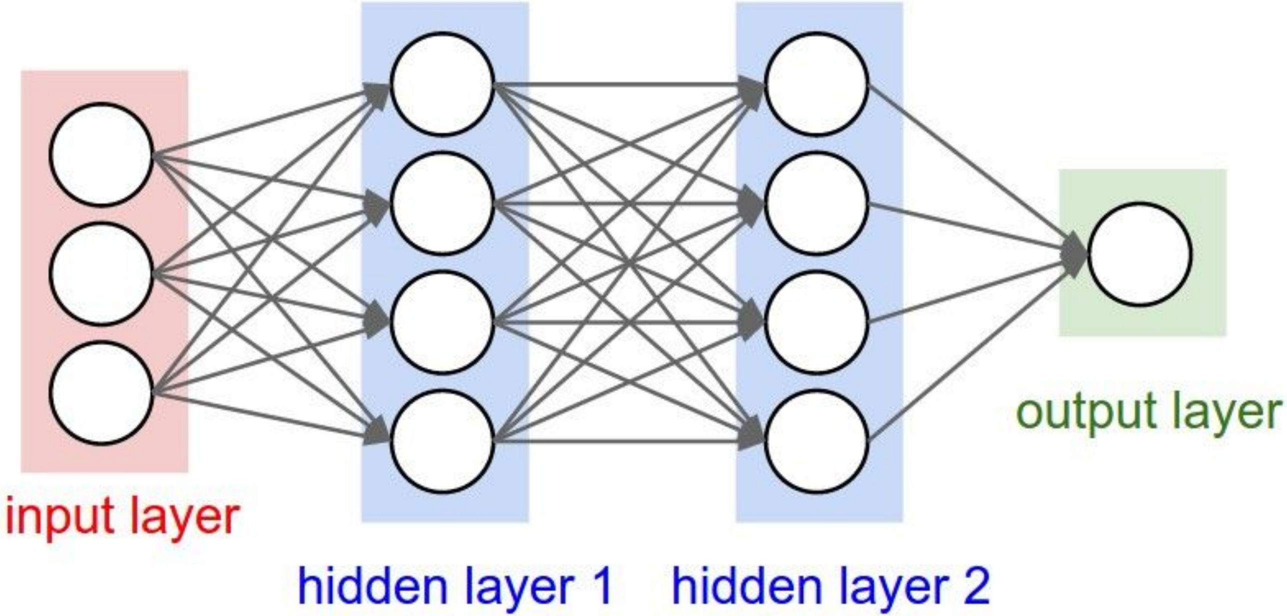
Chris Foster

Machine Learning

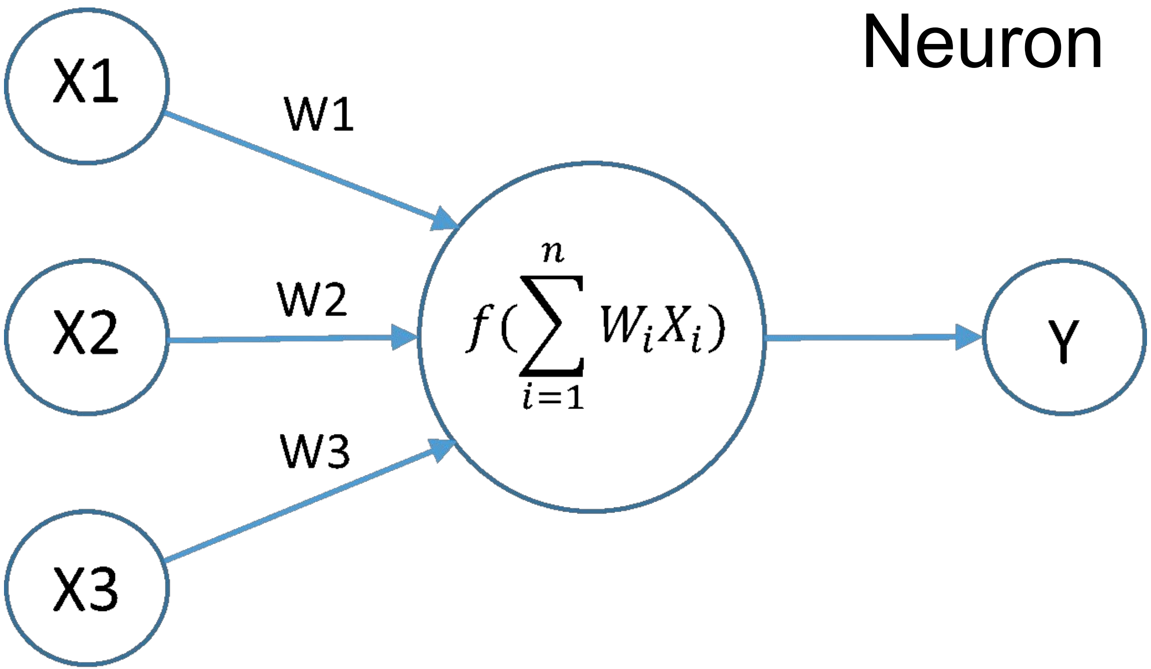
Web Development

Computer Security

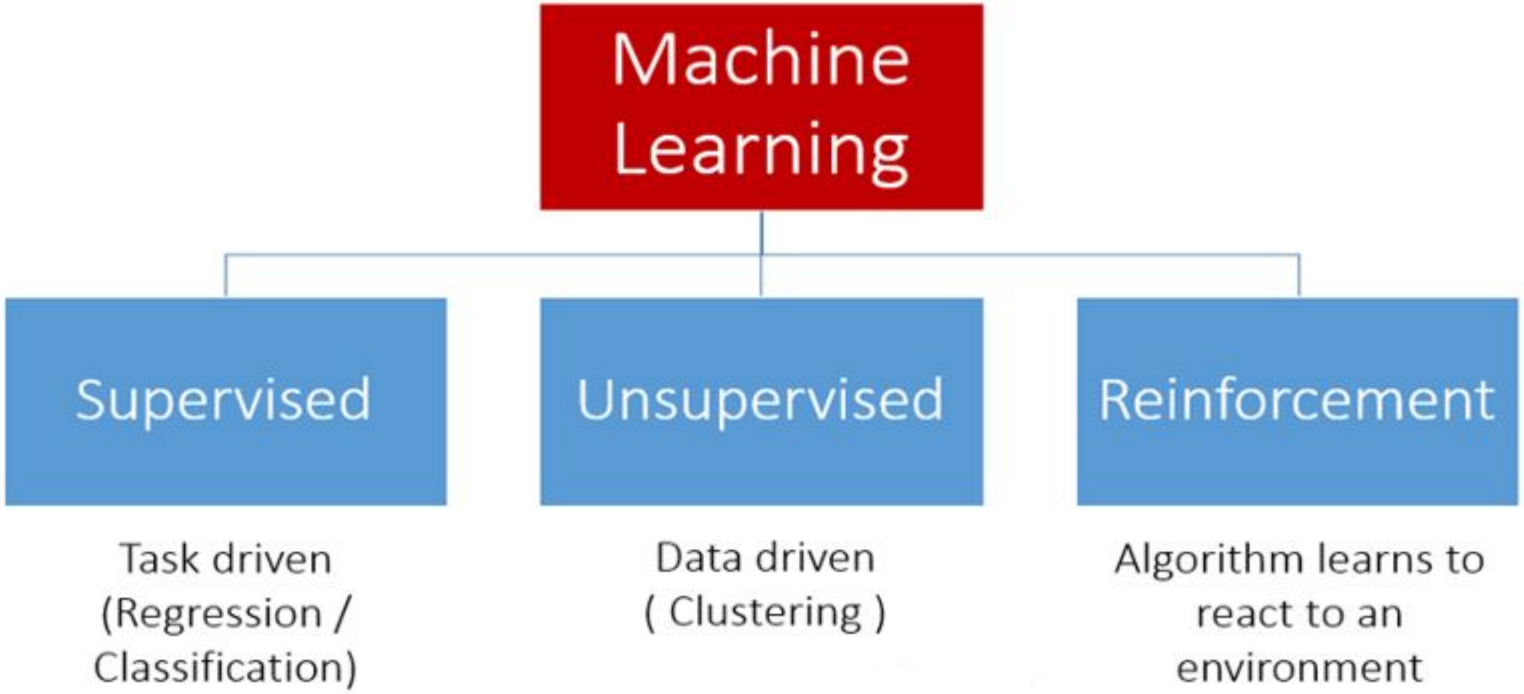
Neural Network



"A computer system modeled on the human brain and nervous system"

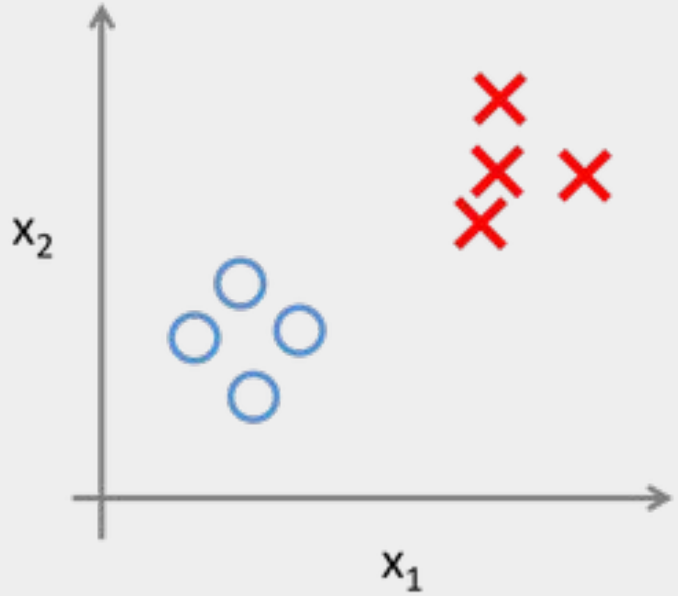

Neuron
X1
X2
X3
W1
W2
W3
$f(\sum_{i=1}^{n} W_i X_i)$
Y

Types of Machine Learning



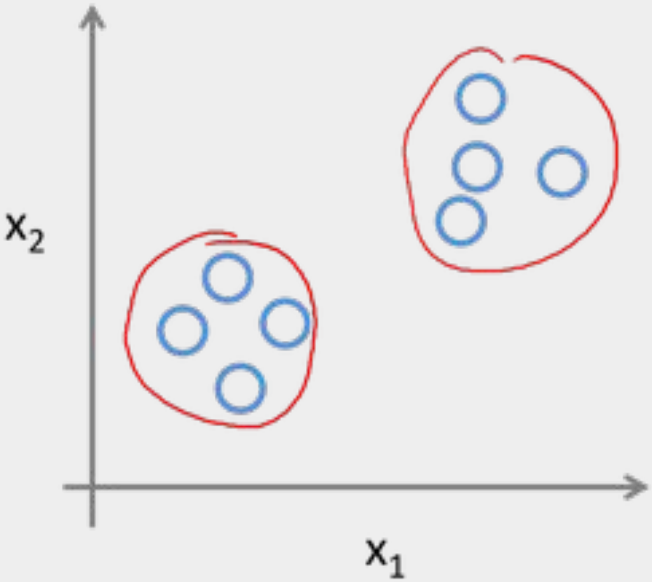
Supervised Learning

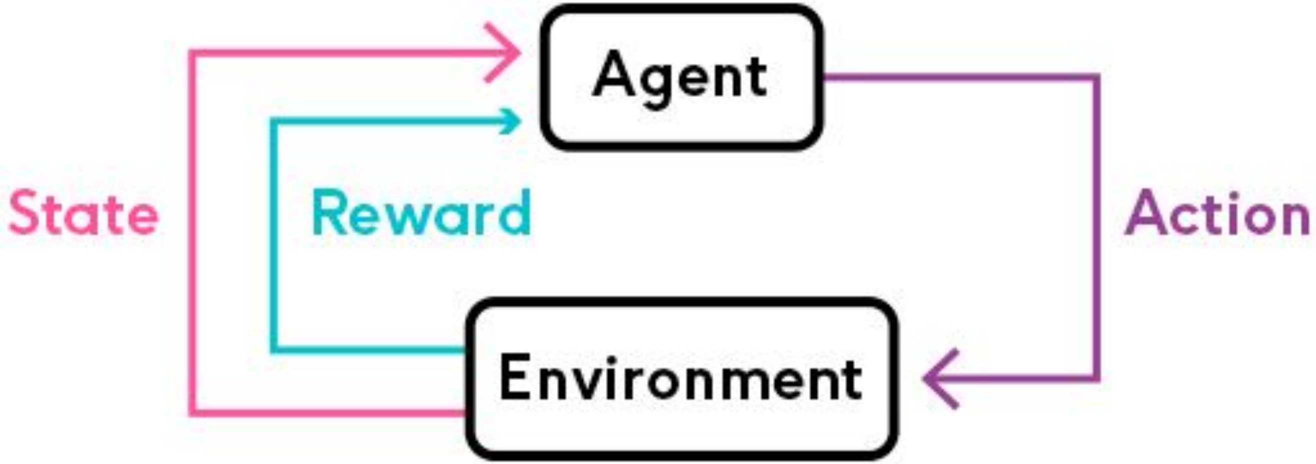
- Classification and regression
- Require training labels
- Example: image categorizing



Unsupervised Learning

- Clustering and reduction
- Does not require labels
- Example: fraud detection





Reinforcement Learning

An example reward

An example reward

An example reward

Applications

- Resource Management
- Traffic Light Control
- Robotics
- Chemistry
- Recommendations
- Advertising
- Game Playing
- AGI Research
- Audio Transcription
- ...and much more!

Tasks with easy sampling

Considerations

- Tend to be more unstable
- Very active research area
- Often difficult to reproduce results
- Requires large number of samples
- Can be outperformed

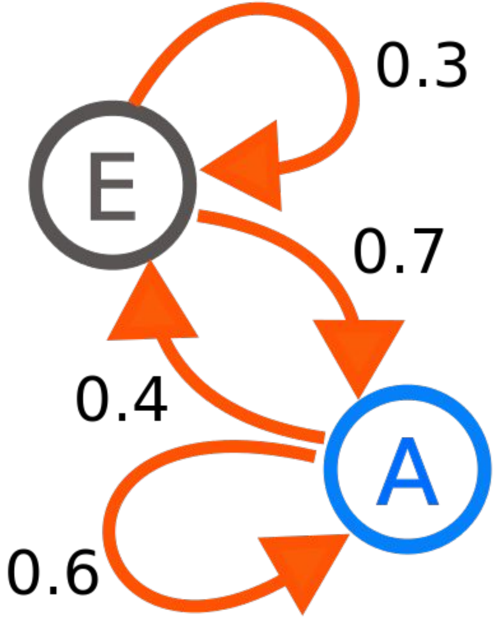
All Things Markov

Markov Property

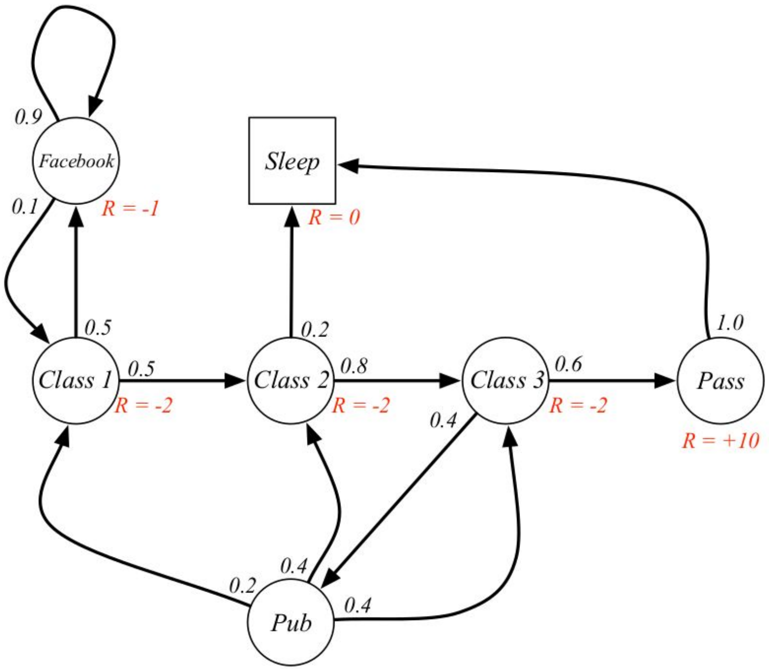
P[St+1 | St] = P[St+1 | S1, ….. , St]

Markov Process: (S, P)

Markov Chain: S is discrete

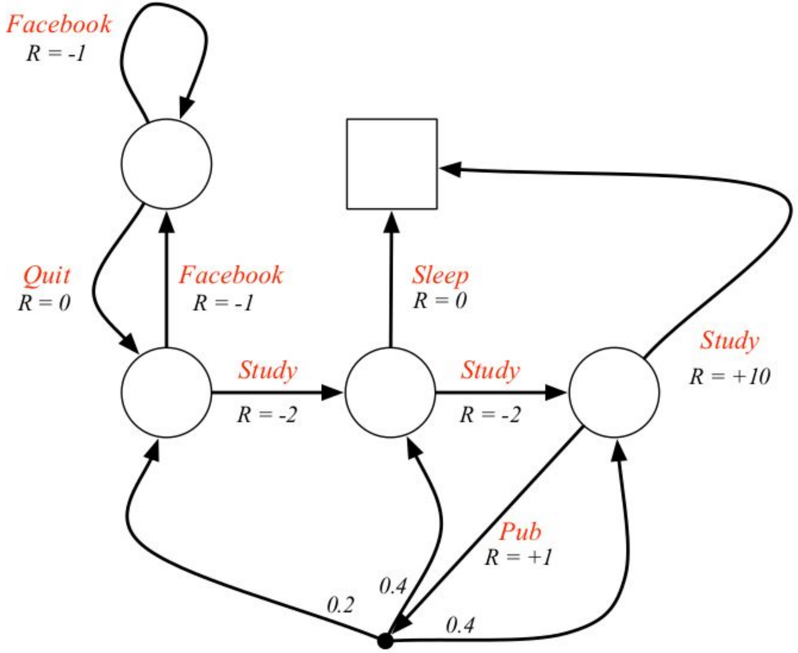


Markov Reward Process



$(\boldsymbol{S}, \boldsymbol{P}, \boldsymbol{R}, \gamma)$

Markov Decision Process



$(\boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \gamma)$

Describing our agent

Our goal is to find a policy that maximizes this total reward: $\sum_{t \geq 0} \gamma^t r_t$

A policy is defined as follows: $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$

And the optimal policy is then: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$

How do we evaluate the decisions the agent makes?

We introduce a **value function**: $V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$

We can also evaluate a state-action pair: $Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$

We can find the best strategy easily:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q^*(s,a)$$

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

π^* vs. Q^*

Approximated Cross-Entropy Method

Given M (f.i, 20), N (f.i. 200) and function approximation (f.i. NN) depending on θ

Initialize θ randomly

repeat

  Sample N roll-outs of the policy and collect for each R_t

  elite $= M$ best samples

  $\theta = \theta + \alpha \nabla \left[\sum_{s,a \in elite} \log \pi_\theta(a|s) \right]$

until convergence

return π_θ

Taking on a RL problem

State: Cart position / velocity, Pole angle / velocity

Action: Push cart left, or push cart right

Reward: 1 for every step

Termination: Pole is more than 12 degrees. Cart is more than 2.4 units. Episode goes longer than 200.

Solve target: An average score of 195

Demo

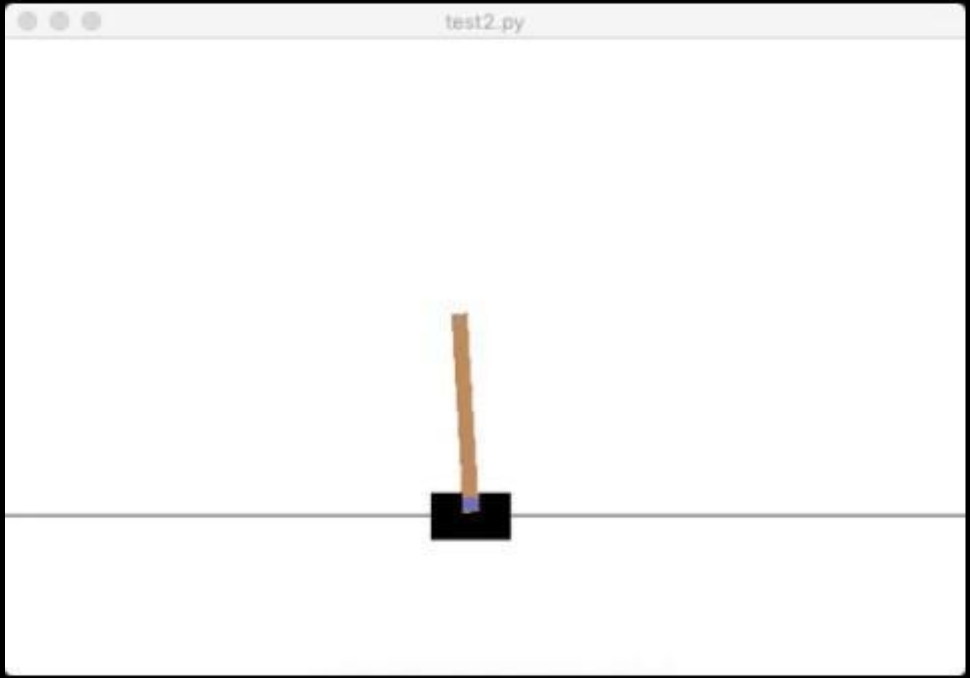
Training a CEM agent!

```python
class CEM(nn.Module):

    def __init__(self, obs_size, n_actions):
        super(CEM, self).__init__()
        self.fc1 = nn.Linear(obs_size, 200)
        self.fc2 = nn.Linear(200, n_actions)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

```python
def filter_batch(states, actions, rewards, percentile=70):

    reward_threshold = np.percentile(rewards, percentile)

    elite_states = []
    elite_actions = []

    for i in range(len(rewards)):
        if rewards[i] > reward_threshold:
            for j in range(len(states[i])):
                elite_states.append(states[i][j])
                elite_actions.append(actions[i][j])

    return elite_states, elite_actions
```

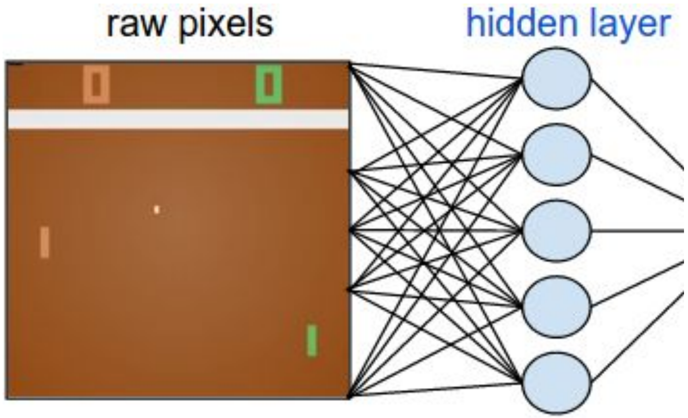
Best parameters?

Is this gradient free?


test2.py

Exploration / Exploitation Tradeoff

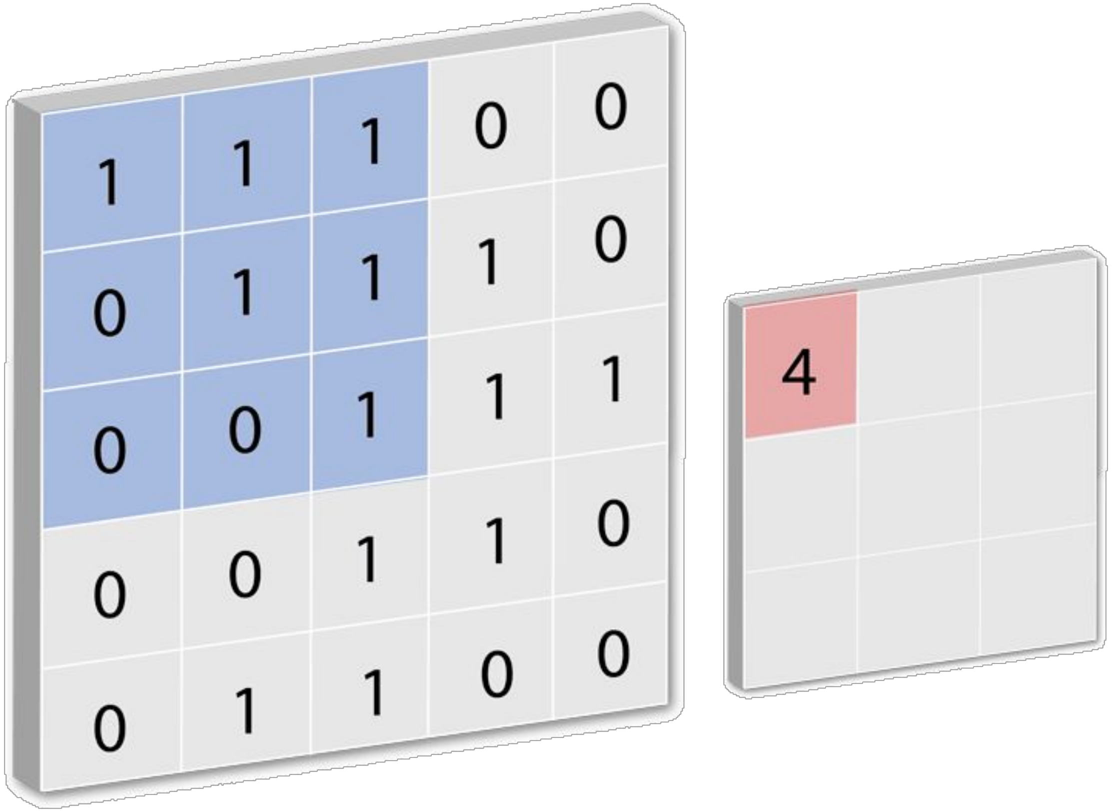
Pixels

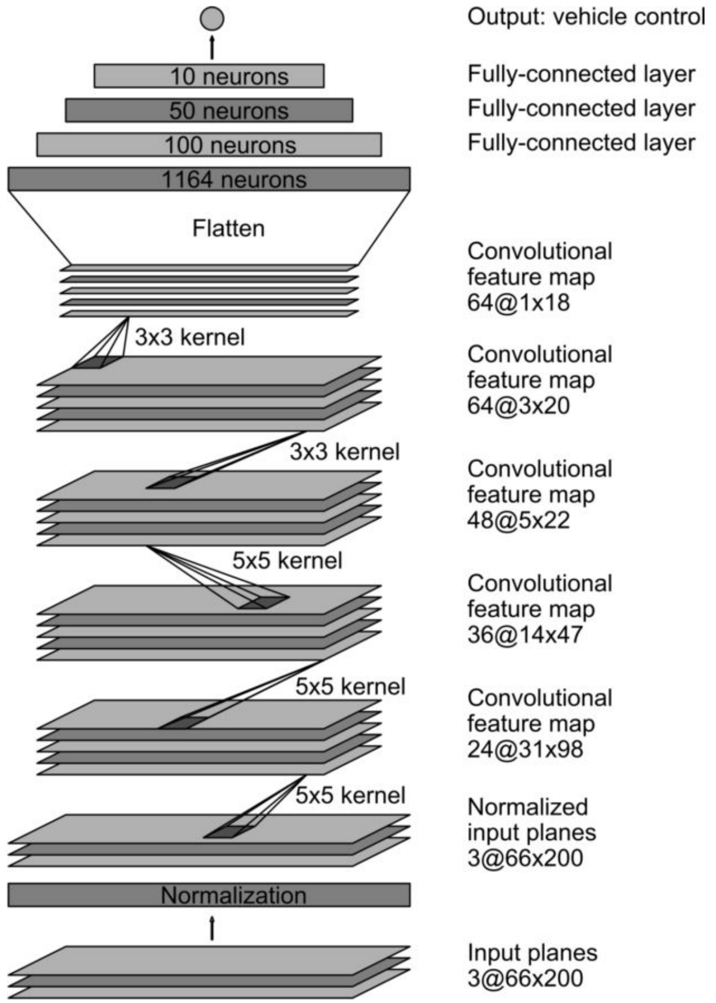


State

```
[
  cart_pos,
  cart_vel,
  pole_deg,
  pole_vel
]
```

Convolutional Neural Networks




Output: vehicle control
10 neurons
Fully-connected layer
50 neurons
Fully-connected layer
100 neurons
Fully-connected layer
1164 neurons
Flatten
Convolutional feature map 64@1x18
3x3 kernel
Convolutional feature map 64@3x20
3x3 kernel
Convolutional feature map 48@5x22
5x5 kernel
Convolutional feature map 36@14x47
5x5 kernel
Convolutional feature map 24@31x98
5x5 kernel
Normalized input planes 3@66x200
Normalization
Input planes 3@66x200

Combining Value and Policy Approaches

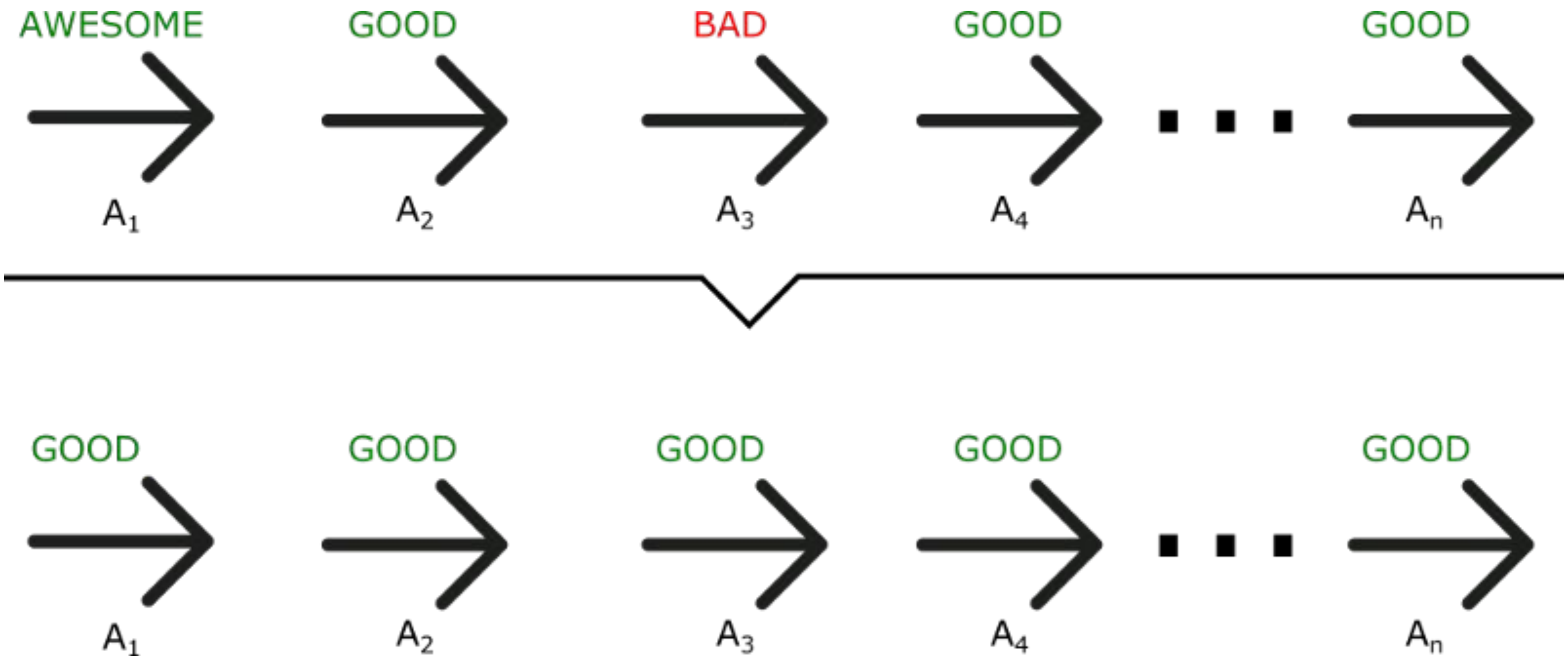
Caveats with value-based RL

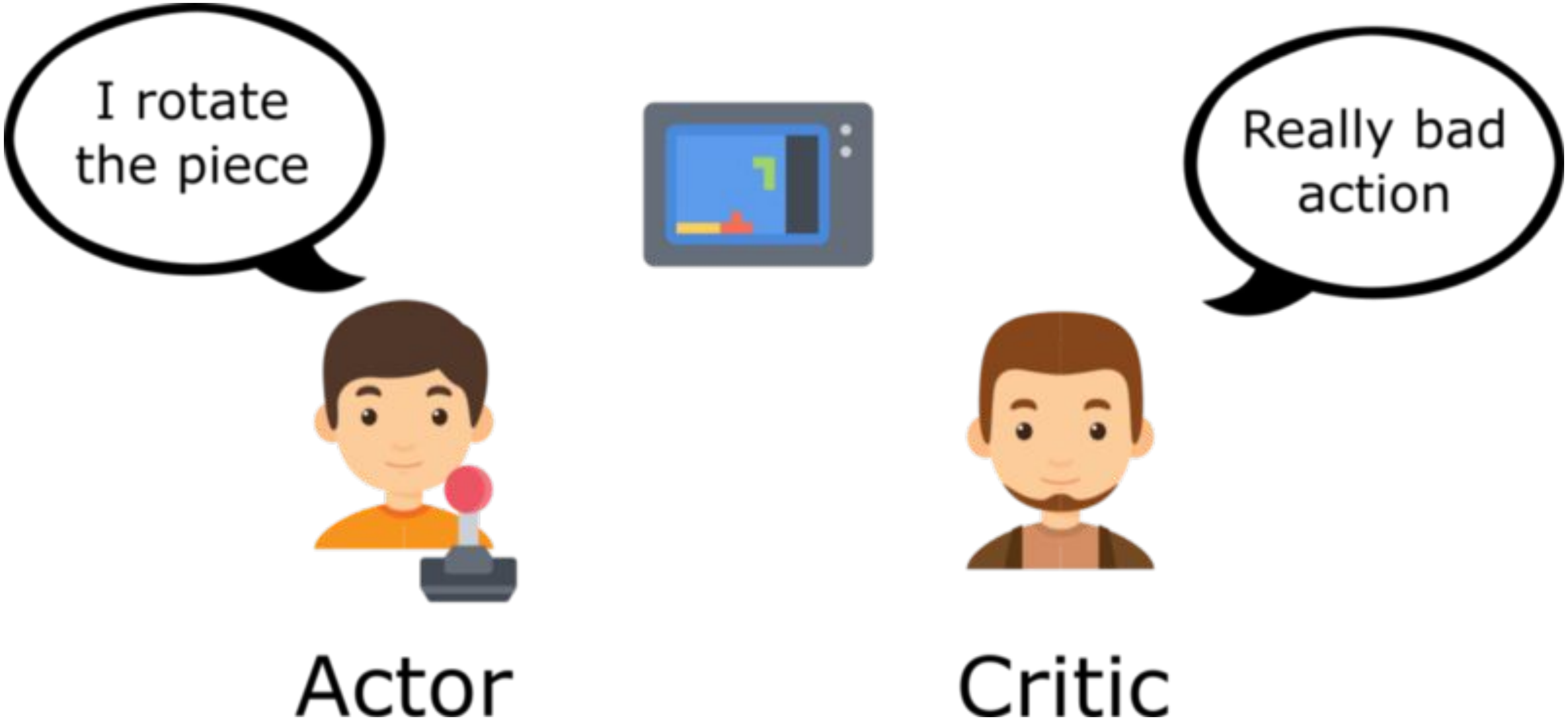
1. Convergence

2. Fails to work in a continuous action space

3. Requires manual exploration-exploitation adjustment

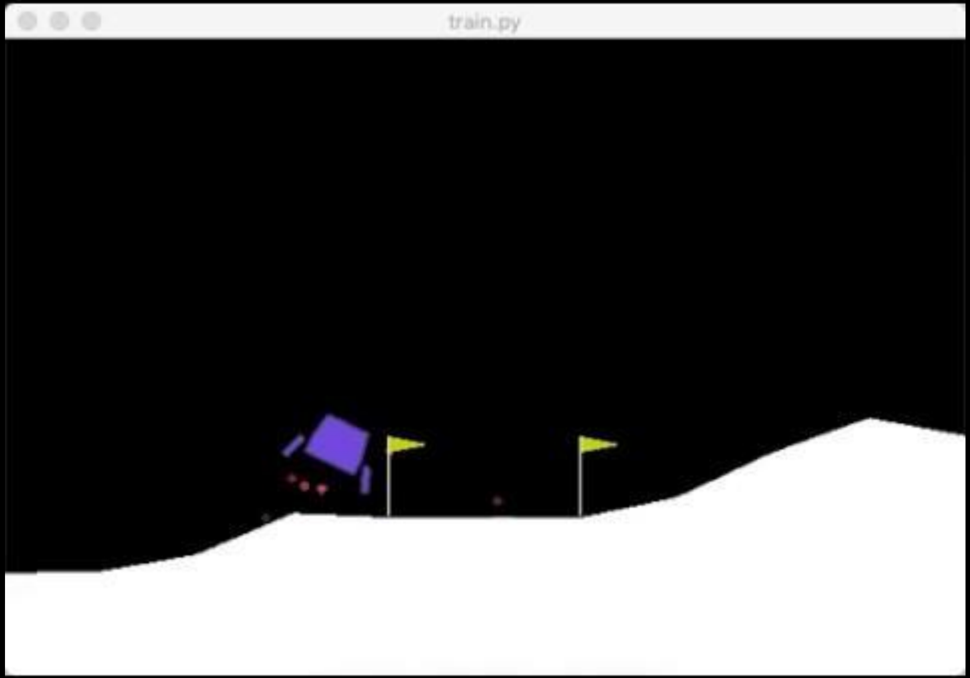
Caveats with policy-based RL




I rotate the piece
Really bad action
Actor
Critic

Demo

Actor-Critic Agent


train.py


train.py

Next Steps

- Battlesnake - Local Victoria event!
- OpenAI Gym - Challenge yourself with Atari games
- AlphaZero - Learn about MCTS and AlphaGo

Thanks!

Sources

https://www.analyticsvidhya.com/
https://towardsdatascience.com/
https://www.cs.upc.edu/~mmartin/
http://karpathy.github.io/
https://github.com/nikhilbarhate99/
https://pytorch.org/
https://github.com/nikhilbarhate99/
http://rail.eecs.berkeley.edu/
https://medium.freecodecamp.org/
https://github.com/openai/gym/
https://www.coursera.org/
http://learning.mpi-sws.org/mlss2016/
https://medium.com/coinmonks/
http://cs231n.stanford.edu/
https://medium.com/@m.alzantot/
https://medium.com/coinmonks/