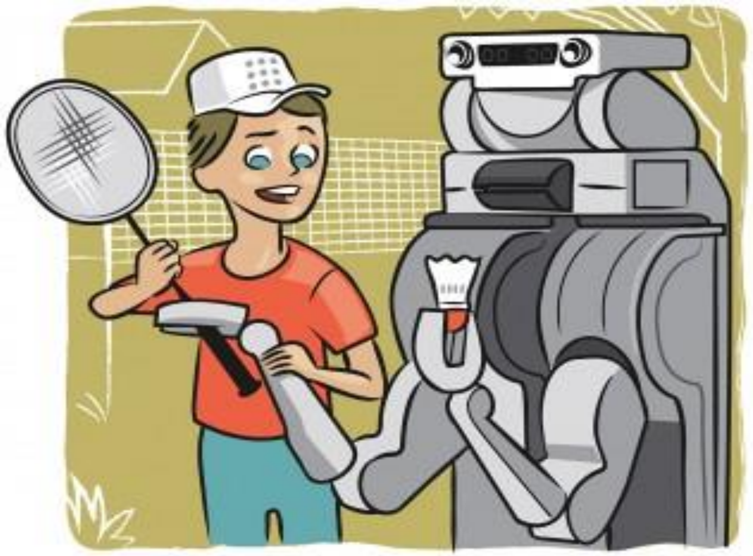
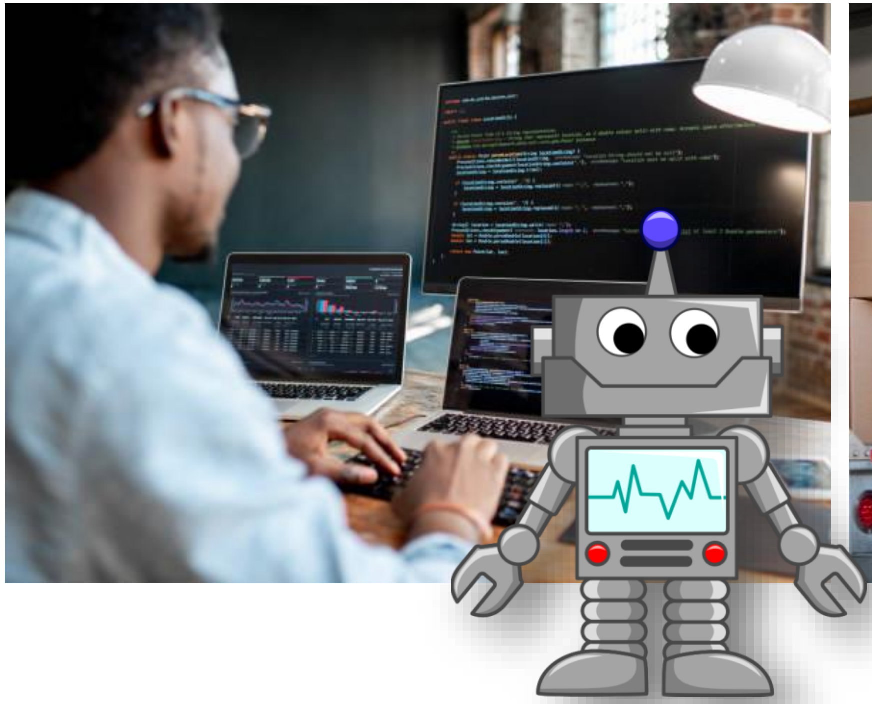
Behavioral Cloning and Interactive Imitation Learning



Instructor: Daniel Brown

[Some slides adapted from Sergey Levine (CS 285) and Alina Vereshchaka (CSE4/510)]



Brief Machine Learning Refresher

There are roughly 3 main branches of machine learning

- Supervised Learning – labeled data (x, y)
- Unsupervised Learning – structure, representation learning
- Reinforcement Learning
  - trial & error learning
  - reward signal

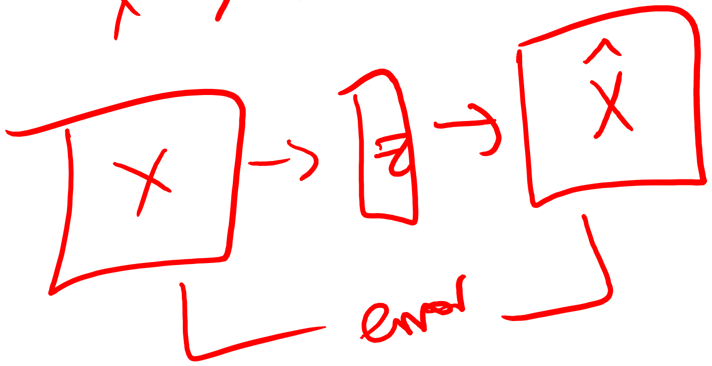
$f_\theta(x) \rightarrow y$

$\mathcal{L}$

input ↑ output

self-supervised

$x \rightarrow z \rightarrow x$



Supervised Learning

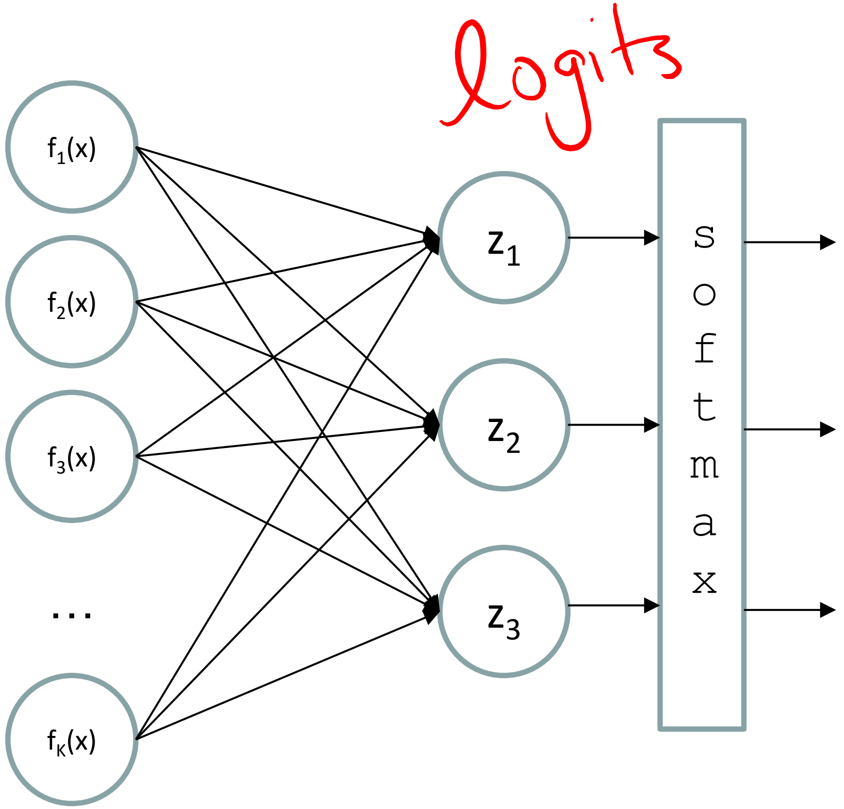
- **Setting/Assumptions**: In supervised learning, the model is trained on labeled data, where the input data is paired with the correct output (i.e., the "ground truth").
- **Goal**: To learn a mapping from inputs to outputs so that the model can predict the output for new, unseen inputs.
- **Common Use Cases**:
  - Classification (e.g., spam email detection, image recognition).
  - Regression (e.g., predicting house prices, stock market trends).
- **Example models**:
  - Linear regression, decision trees, support vector machines, and neural networks.

Multi-class Logistic Regression

classes = 3

- = special case of neural network

softmax

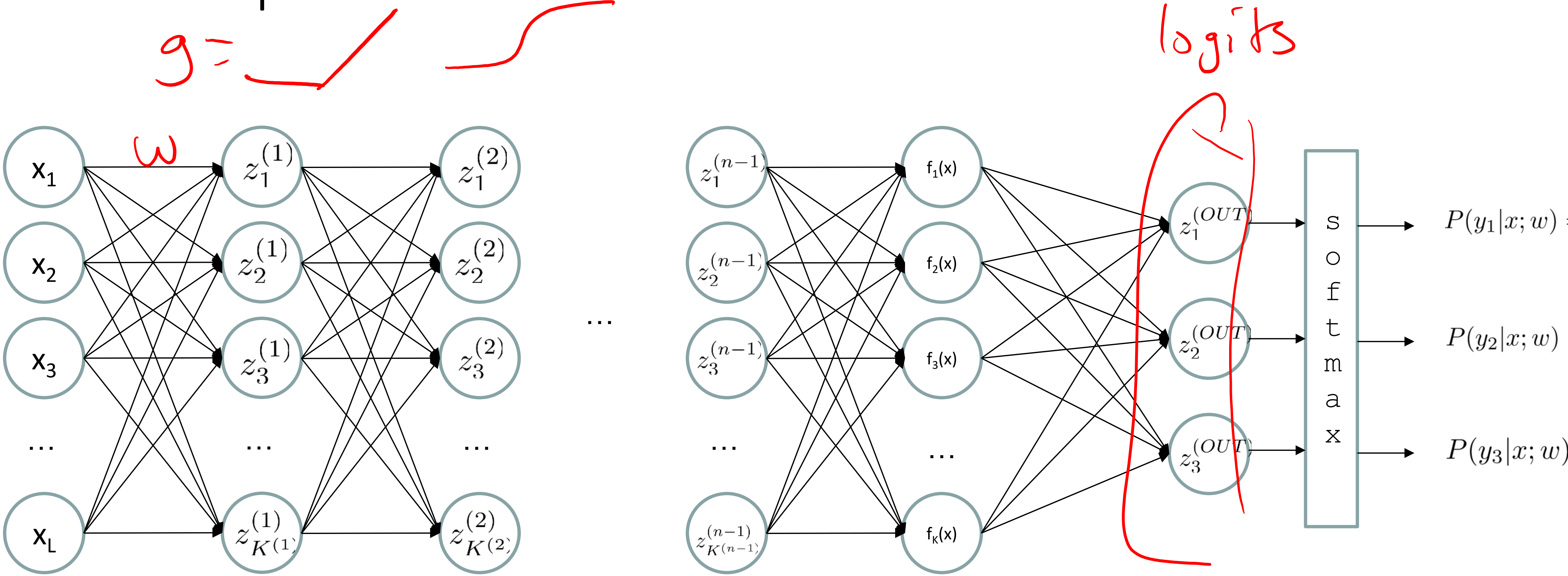


$$P(y_1|x;w) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_2|x;w) = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$P(y_3|x;w) = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Deep Neural Network = Also learn the features!

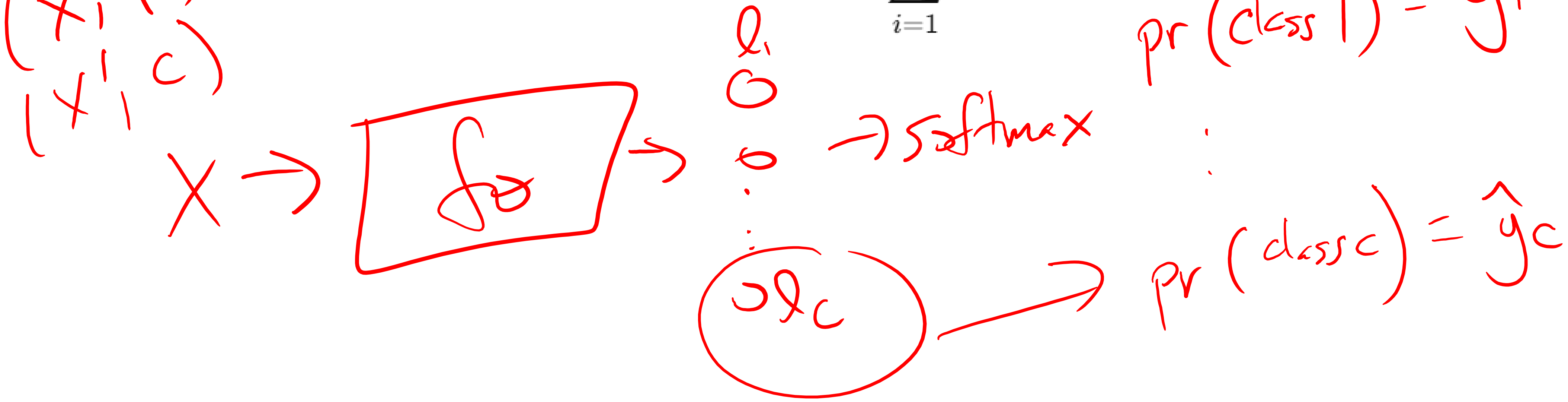


$$z_i^{(k)} = g(\sum_j W_{i,j}^{(k-1,k)} z_j^{(k-1)})$$

g = nonlinear activation function

Classification

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$



PyTorch Example

```python
import torch.nn as nn
import torch.optim as optim

class ClassificationNetwork(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(ClassificationNetwork, self).__init__()
        self.fc = nn.Linear(input_dim, num_classes)


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

model = ClassificationNetwork(input_dim, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    for inputs, labels in dataloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

fc1

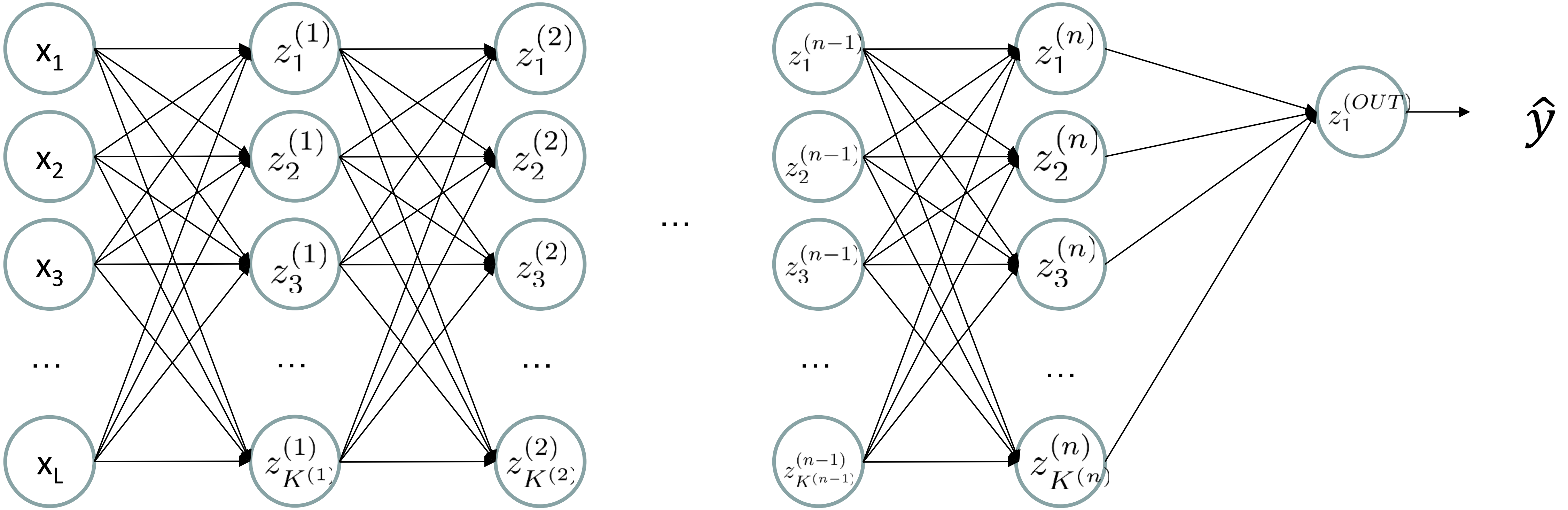
fc2

PyTorch Example (MLP)

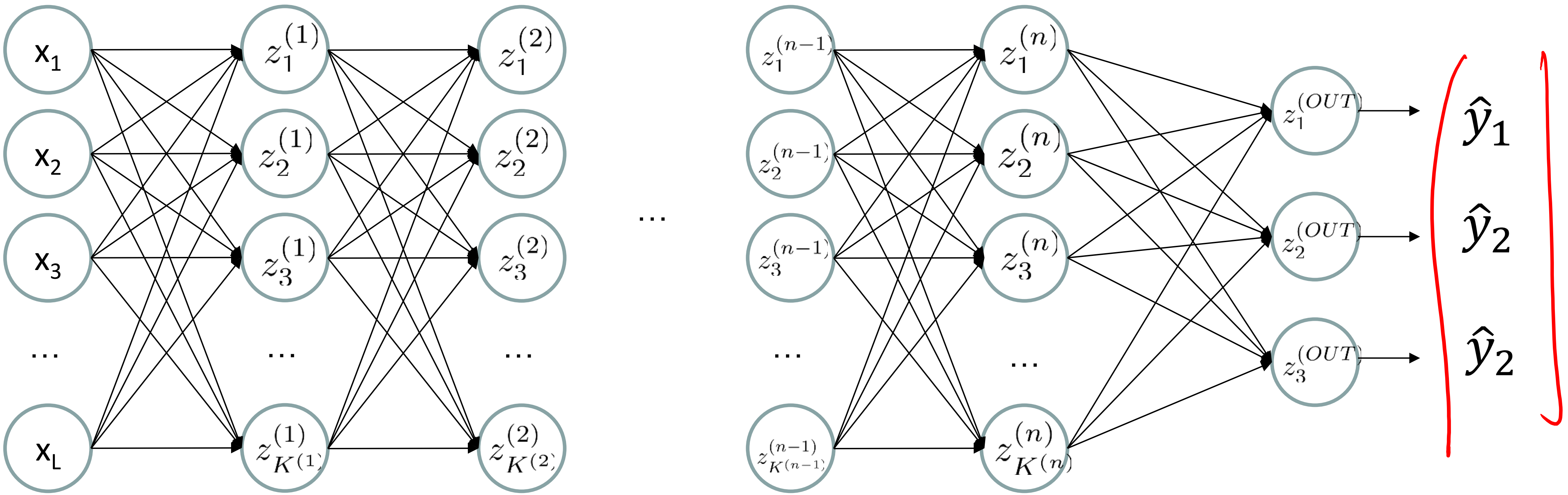
```
import torch.nn as nn
import torch.optim as optim

class ClassificationNetwork(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(ClassificationNetwork, self).__init__()
        self.fc1 = nn.Linear(input_dim, num_hidden)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(num_hidden, num_classes)

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

model = ClassificationNetwork(input_dim, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    for inputs, labels in dataloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

layer 2

Deep Neural Networks for Regression



Deep Neural Networks for Regression



Regression

pred true label

$$\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

PyTorch Example

```
import torch.nn as nn
import torch.optim as optim

class ClassificationNetwork(nn.Module):
    def __init__(self, input_dim, num_classes):
        super(ClassificationNetwork, self).__init__()
        self.fc = nn.Linear(input_dim, num_classes)

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

model = ClassificationNetwork(input_dim, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

for epoch in range(num_epochs):
    for inputs, labels in dataloader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Regression

=1

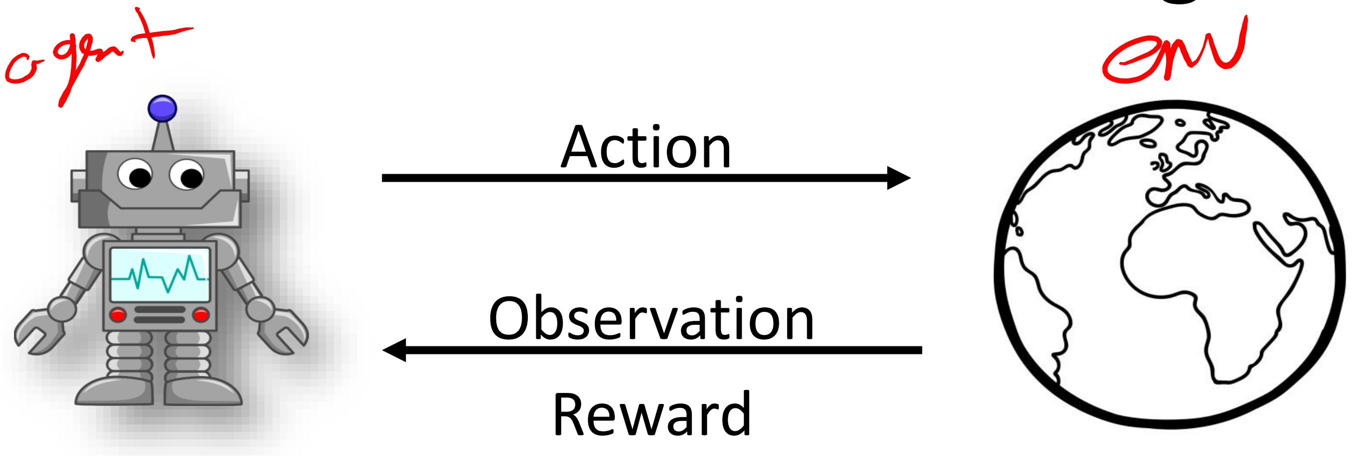
=1

MSE

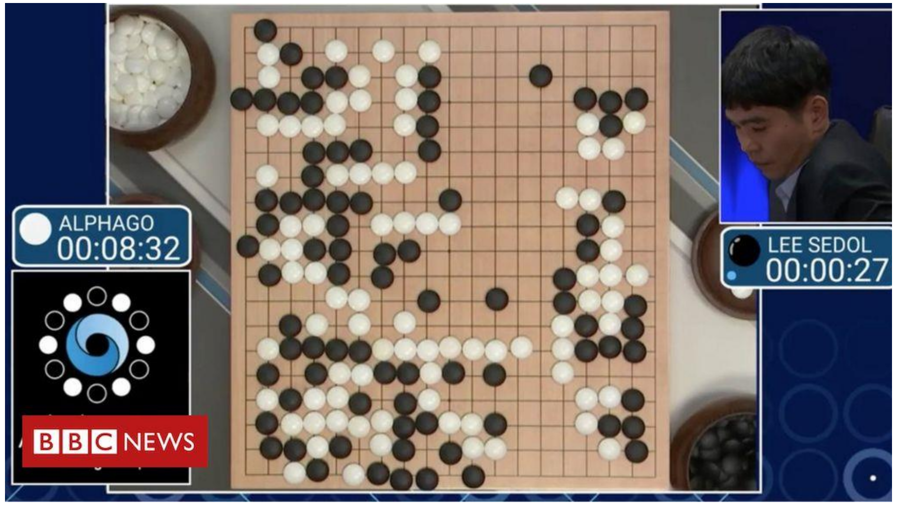
Reinforcement Learning

- **Setting/Assumptions**: Reinforcement learning (RL) involves training an agent to make decisions by interacting with an environment. The agent learns through trial and error (receiving rewards and penalties), optimizing its behavior to maximize cumulative rewards.
- **Goal**: To learn a policy that maps states of the environment to actions that achieve the highest reward.
- **Common Use Cases**:
  - Game-playing AI (e.g., AlphaGo, chess-playing bots).
  - Robotics (e.g., autonomous navigation).
  - Dynamic resource allocation (e.g., in networking or traffic management).
- **Examples**:
  - Q-learning, Deep Q-Networks (DQN), and Proximal Policy Optimization (PPO).

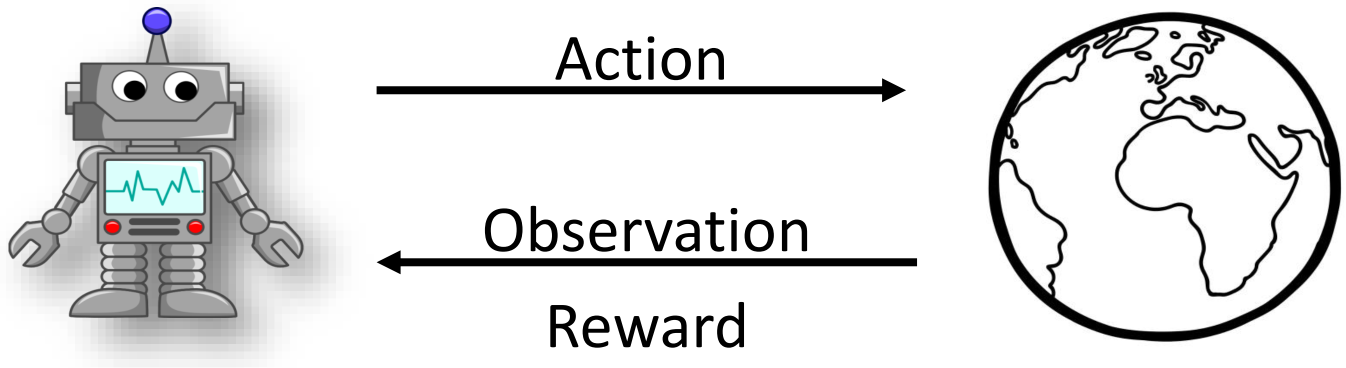
Reinforcement Learning



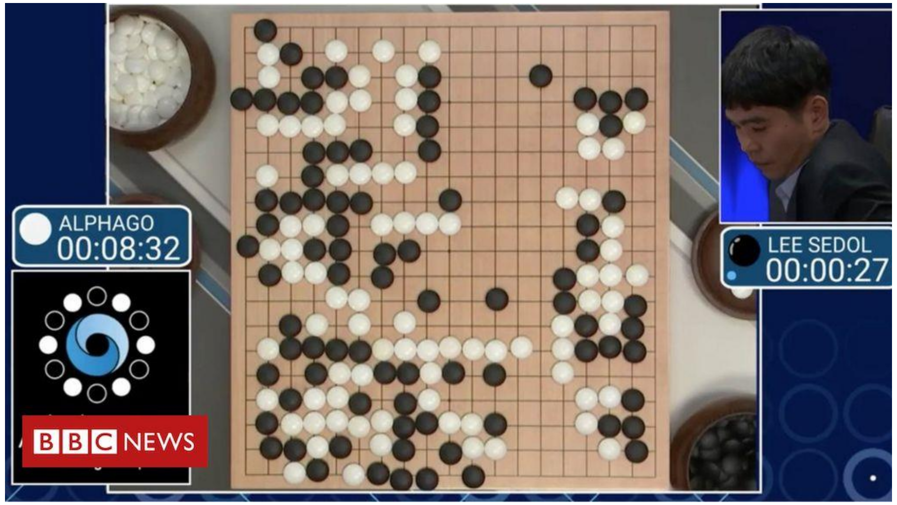
r=+1 win -1 lose

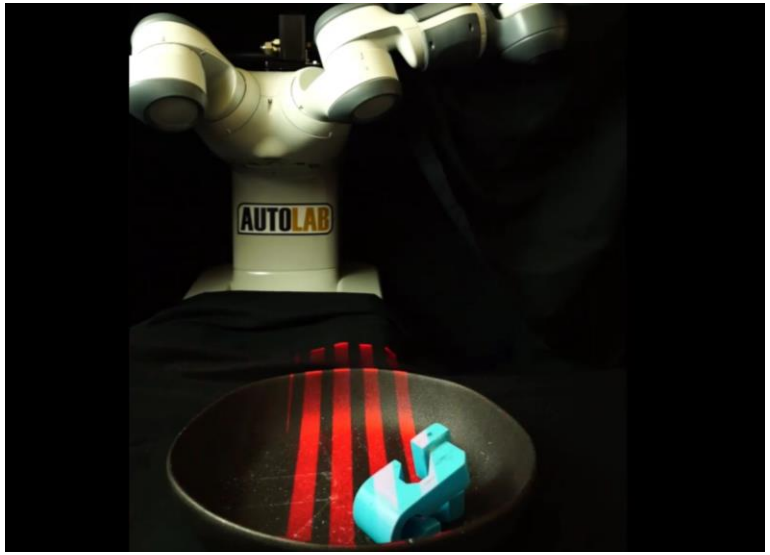


Reinforcement Learning

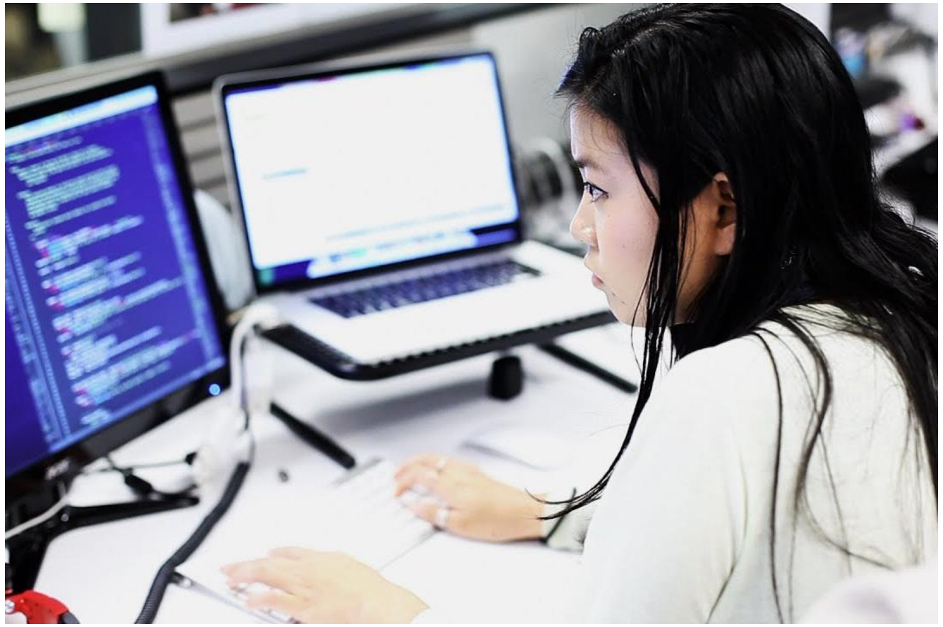


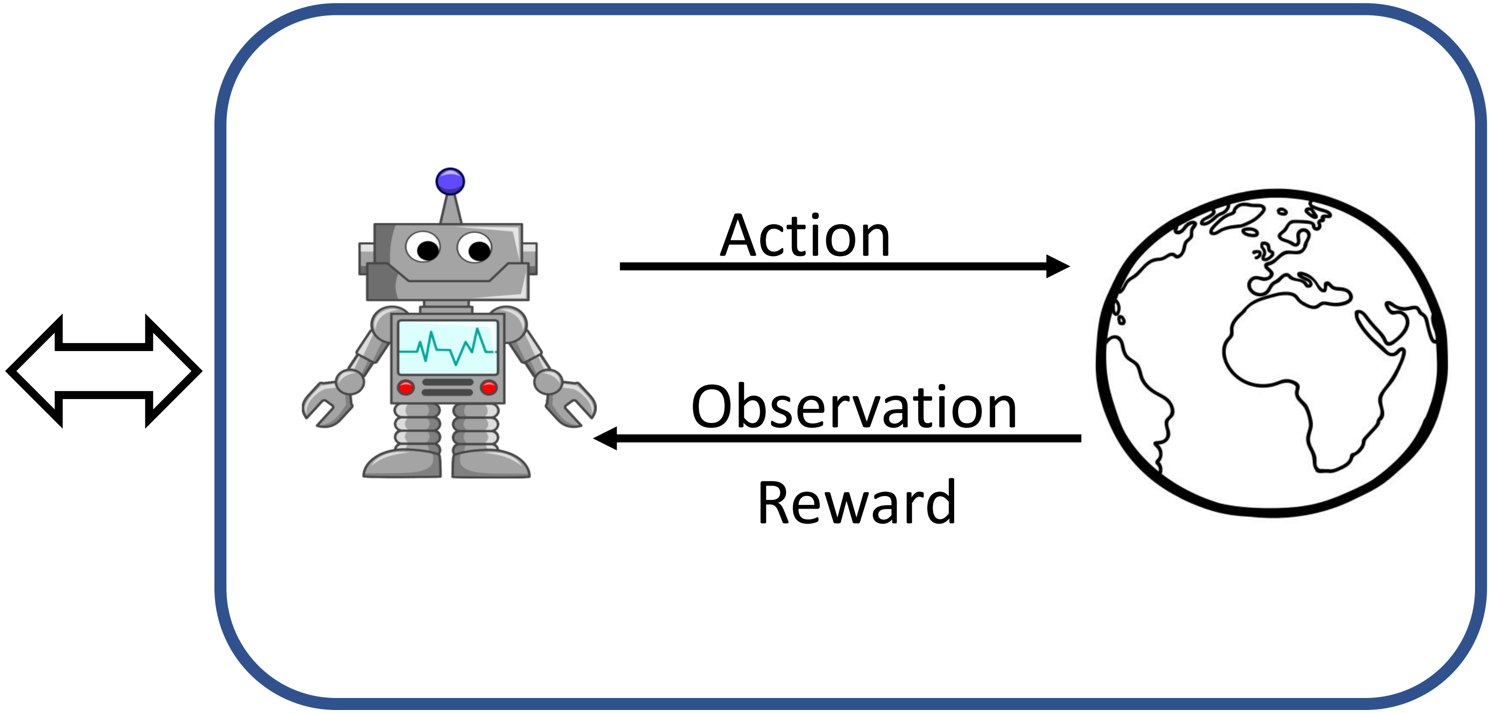
+1 0



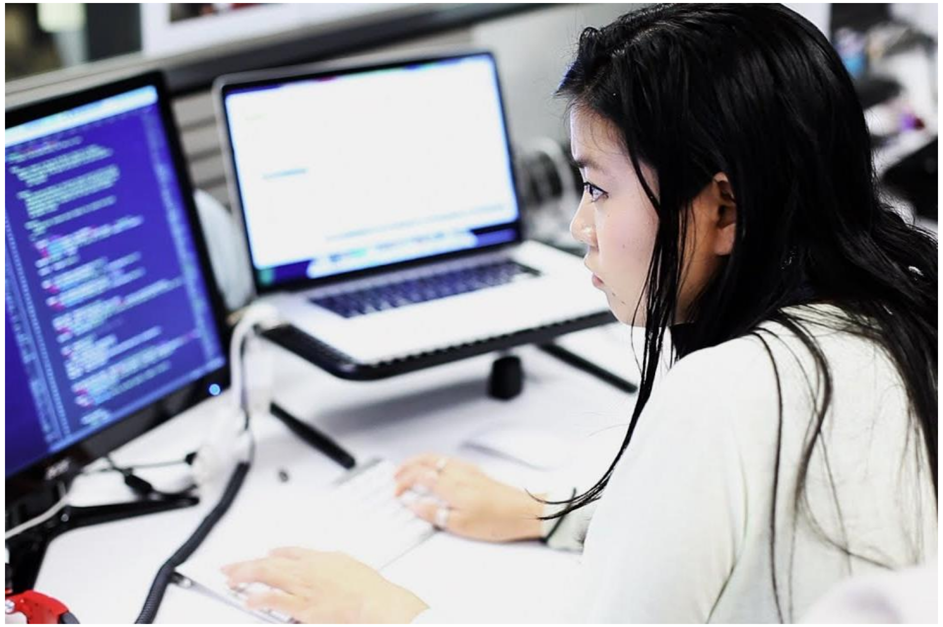


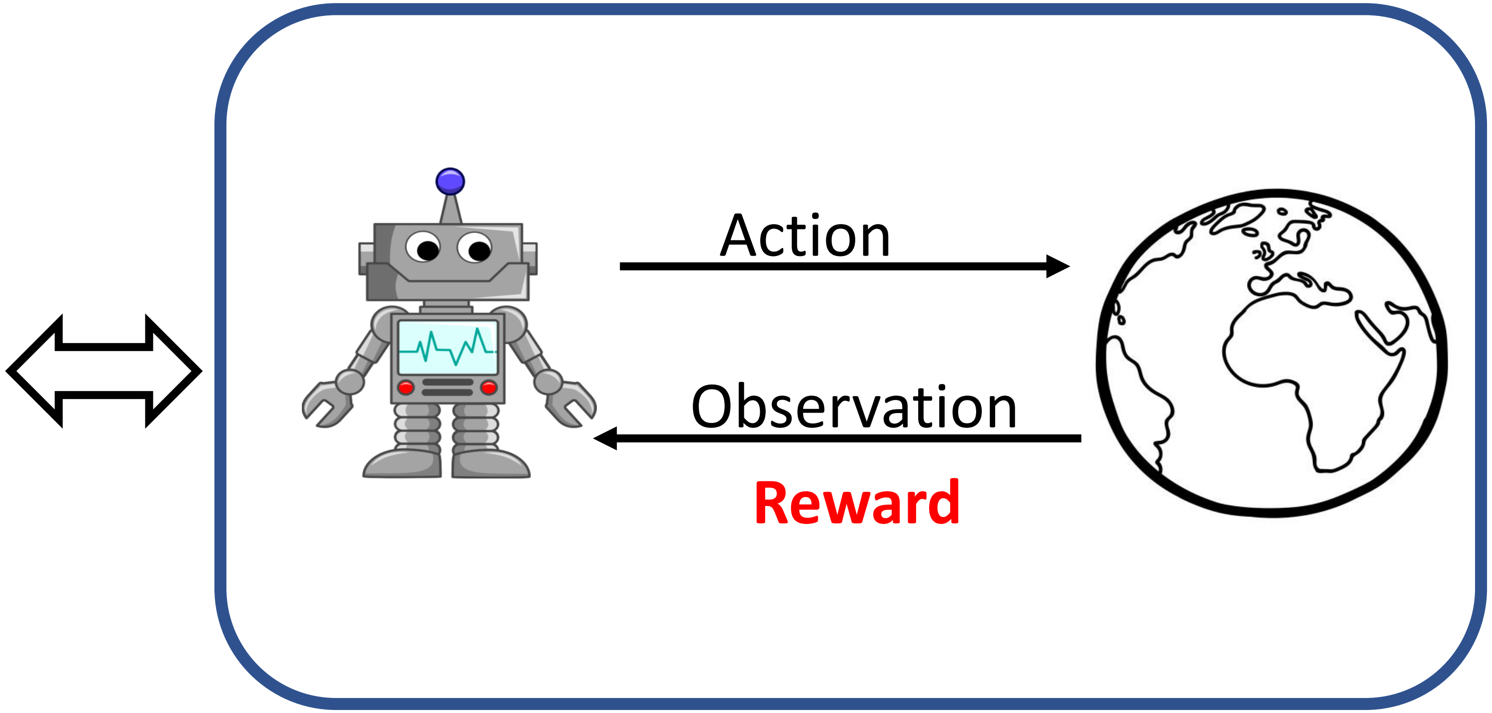
Reward engineering is hard!



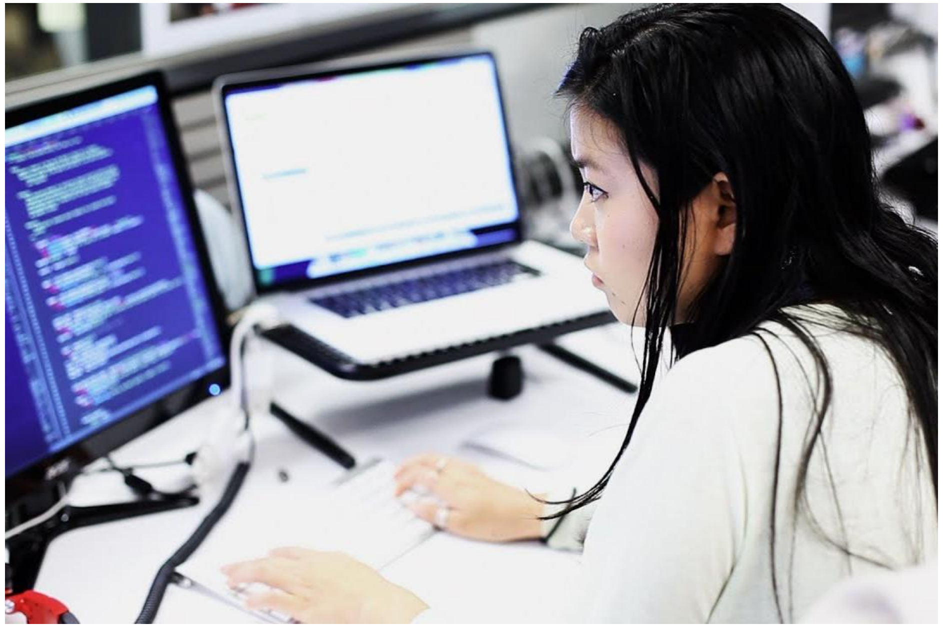


Reward engineering is hard!

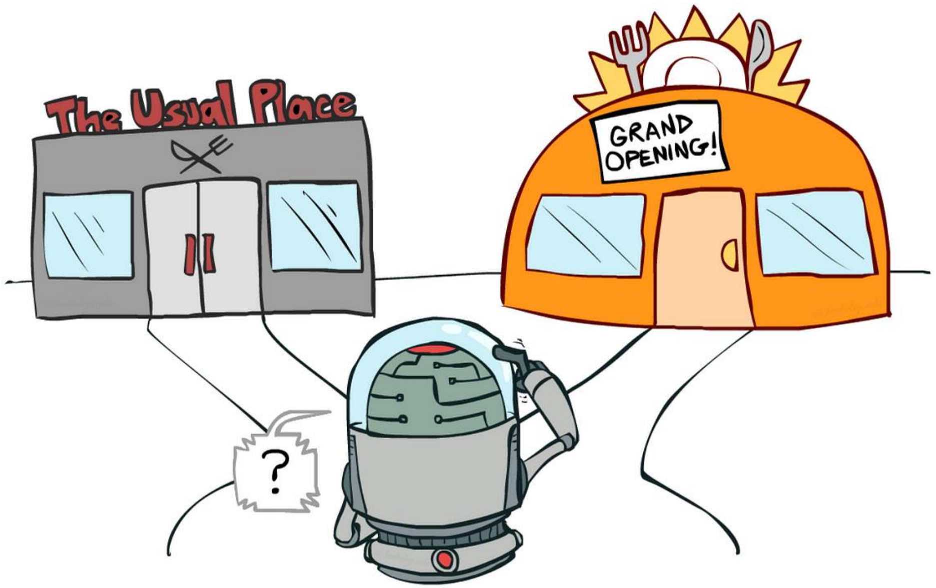




Reward engineering is hard!



Reinforcement learning is hard…even with a reward function!



Imitation Learning (Learning from Demonstrations):

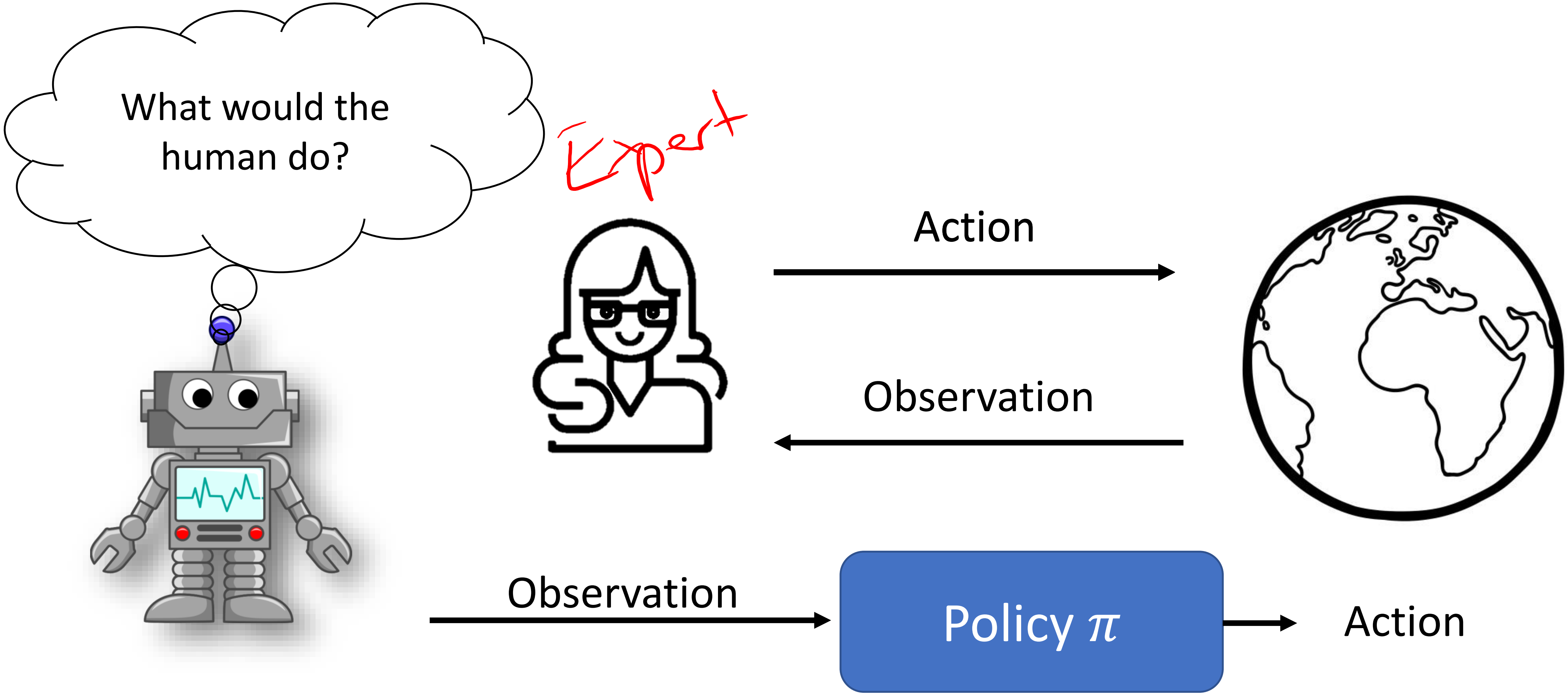
Learn a policy from examples of good behavior.

$\pi(s) \to a$

- Often showing is easier than telling.
- Alleviates problem of exploration.

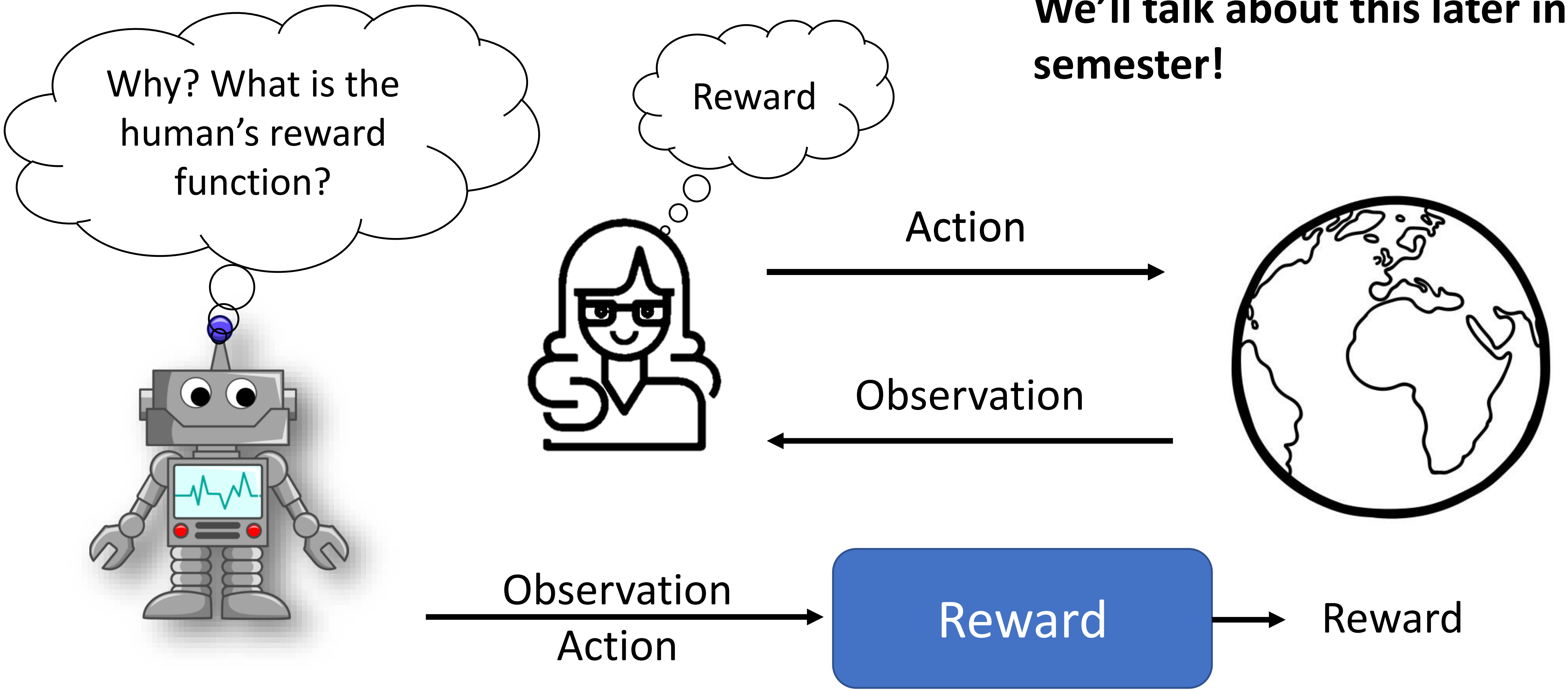
x $p(x)$

Behavioral Cloning

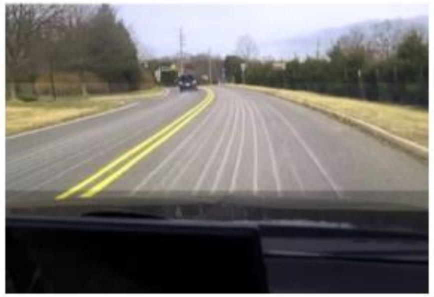


Inverse Reinforcement Learning

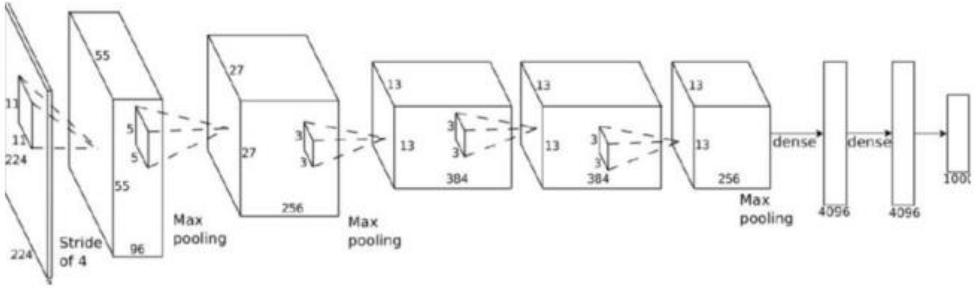
We'll talk about this later in the semester!



Imitation Learning via Behavioral Cloning



$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

$\mathbf{a}_t$

$\mathbf{o}_t$

$\mathbf{a}_t$

→ training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

output input

Live demo

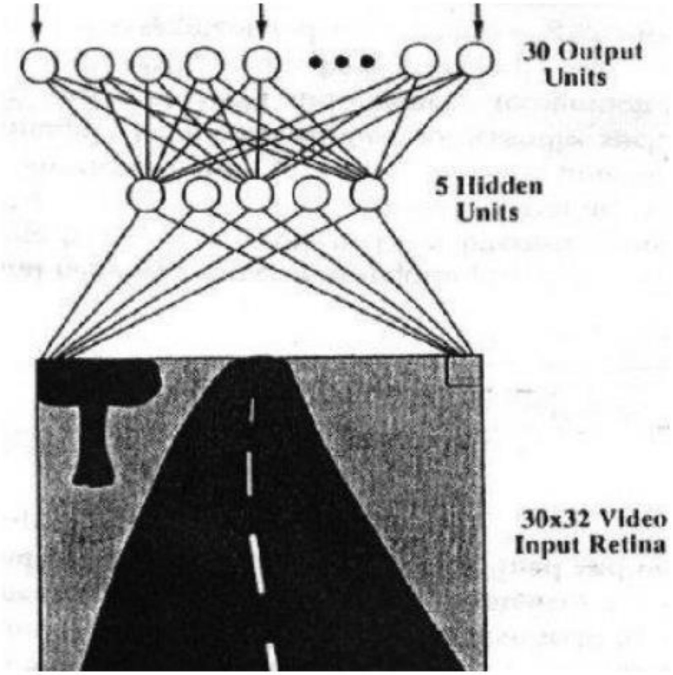
https://github.com/dsbrown1331/imitation_learning/blob/main/README.md

python test_gym.py

python mountain_car_bc.py --num_demos 1

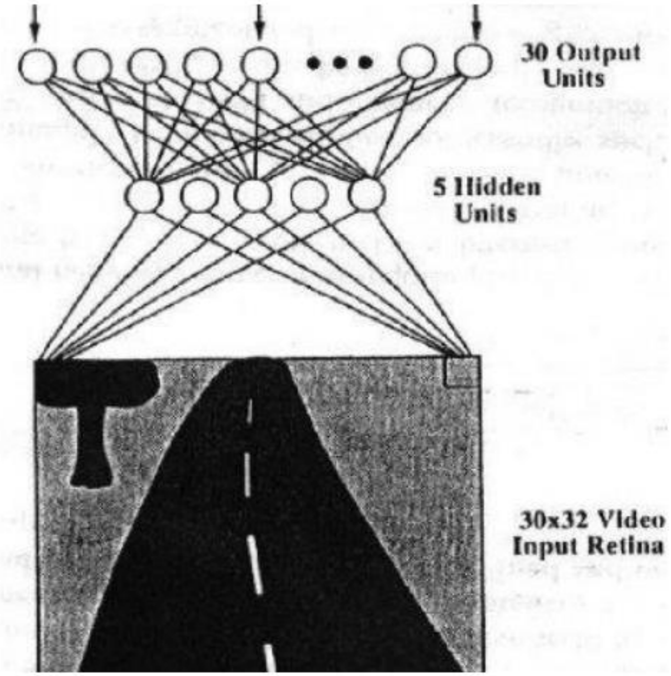
ALVINN: One of the first imitation learning systems

ALVINN: **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork
1989

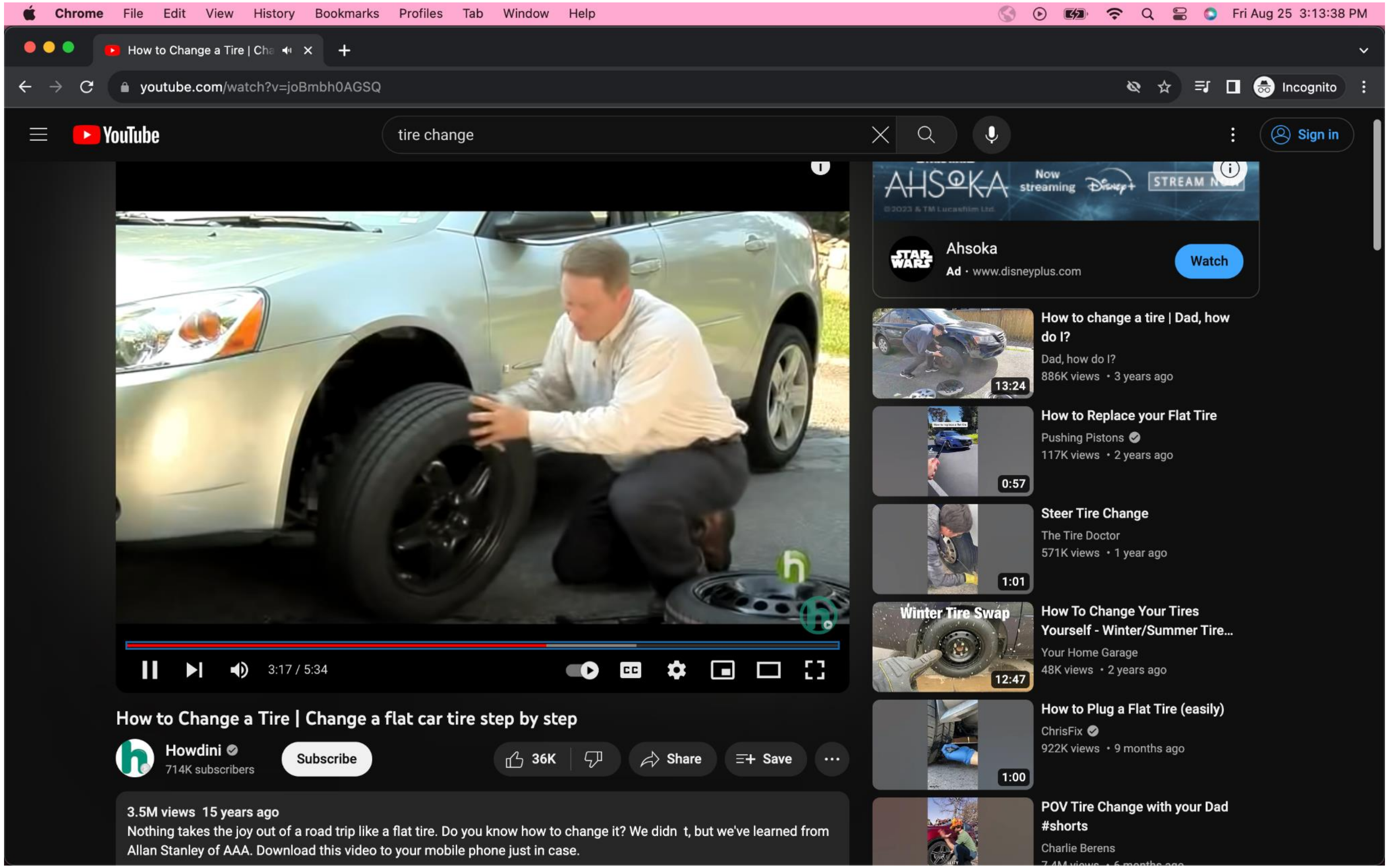


ALVINN: One of the first imitation learning systems

ALVINN: **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork
1989



What if you don't have actions?



Behavioral Cloning from Observation (Torabi et al. 2018)

$D = \{(s, a) \ldots\}$

$D = \{(s_0, s_1, s_2 \ldots s_T), (s'_0, s'_1, \ldots, s'_T)\}$