Reinforcement Learning

CS 59300: RL1

August 28, 2025

Joseph Campbell

Department of Computer Science

PURDUE UNIVERSITY

Today's lecture

1. Deep learning recap

2. Behavior cloning

Deep learning recap

Resources

This will be an extremely high-level refresher.

If you need to brush up on DL, here are some useful resources.

- Stanford CS231n: https://cs231n.stanford.edu/
- Deep Learning (Goodfellow, Bengio, Courville): https://www.deeplearningbook.org/
- Patterns, Predictions, and Actions (Hardt, Recht): https://mlstory.org/
- PyTorch tutorials: https://docs.pytorch.org/tutorials/

Types of machine learning

Supervised learning

- Given both inputs x and labels y, learn a function $y = f(x)$
- Regression, classification, …

Unsupervised learning

- Given only inputs x, discover patterns within the data
- Clustering, dimensionality reduction, …

Types of machine learning

Self-supervised learning

- Given inputs x, the model itself generates labels y to learn a function $y = f(x)$
- Masked language modeling, auto-encoding, …

Reinforcement learning

- An agent interacts with an environment to learn a function $y = f(x)$ which maximizes a reward signal
- If stochastic, $p(y|x) = \pi(y|x) = \pi(a|s)$ *(remember last time?)*

Types of machine learning

Starting here right!?

Self-supervised learning

- Given inputs x, the model itself generates labels y to learn a function $y = f(x)$
- Masked language modeling, auto-encoding, …

Reinforcement learning

- An agent interacts with an environment to learn a function $y = f(x)$ which maximizes a reward signal
- If stochastic, $p(y|x) = \pi(y|x) = \pi(a|s)$ *(remember last time?)*

Not so fast: supervised learning

Classification

- Task: given an input x predict a classification label y
- How: learn a function f which maps x to y

Example: the iris dataset: *https://archive.ics.uci.edu/dataset/53/iris*

- Inputs consist of 4 physical features
- Classification label is the type of iris plant

Iris classification

Input:

- Sepal length
- Sepal width
- Petal length
- Petal width

Iris Setosa

Iris Versicolor

Iris Virginica

Output:

- {Iris Setosa, Iris Versicolor, Iris Virginica}

How to train your model

Train | Test

$\underbrace{\qquad\qquad\qquad}_{(x, y) \in D_{\text{train}}}$

Train model

Input x → f (Model) → Label y

How to train your model

Train | Test

$$\underbrace{\text{Train}}_{(x,y)\,\in\, D_{\text{train}}}$$

Train model

Input x → f (Model) → Label y

What is the difference between **optimization** and **machine learning**?

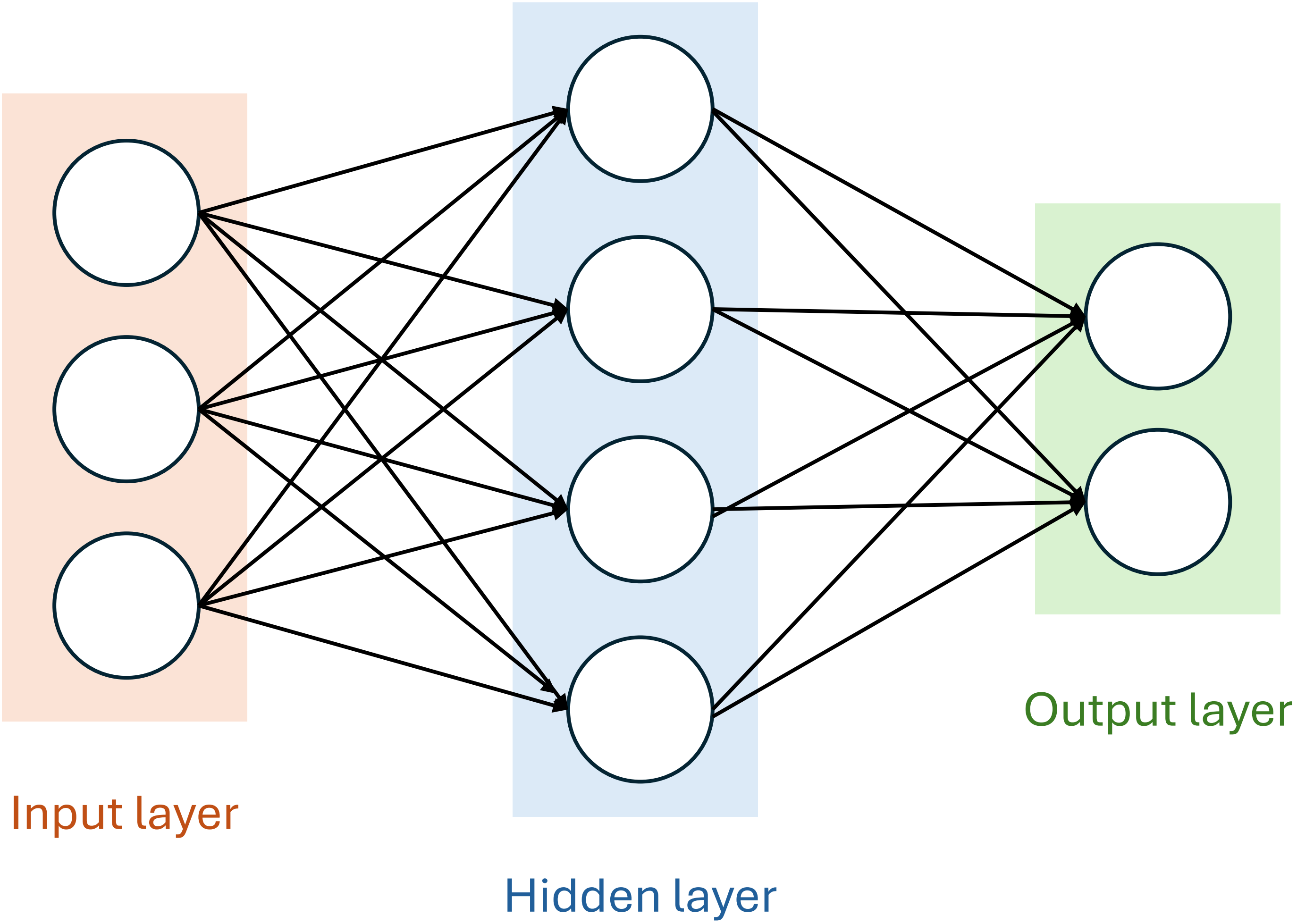
Many types of models!

- K-nearest neighbor
- Linear classifier
- Multinomial logistic regression
- Support vector machine
- …
- **Neural network**

Focus of this course

Neural networks

Fully-connected layers



Neural Network: math

Mathematical model for the network just shown:

$$h = \sigma(W_1 x + b_1)$$
$$y = g(W_2 h + b_2)$$

Non-linear activation functions

Learnable weights

Learnable biases

Neural Network: math

Mathematical model for the network just shown:

$$h = \sigma(W_1 x + b_1)$$
$$y = g(W_2 h + b_2)$$

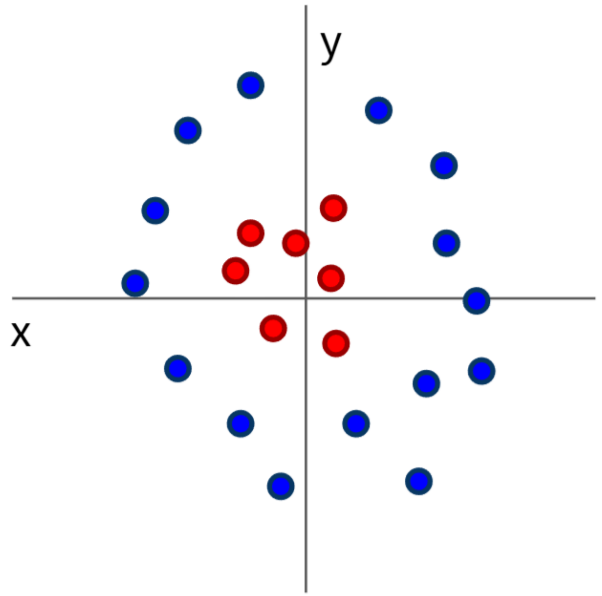
Non-linear activation functions

Learnable weights

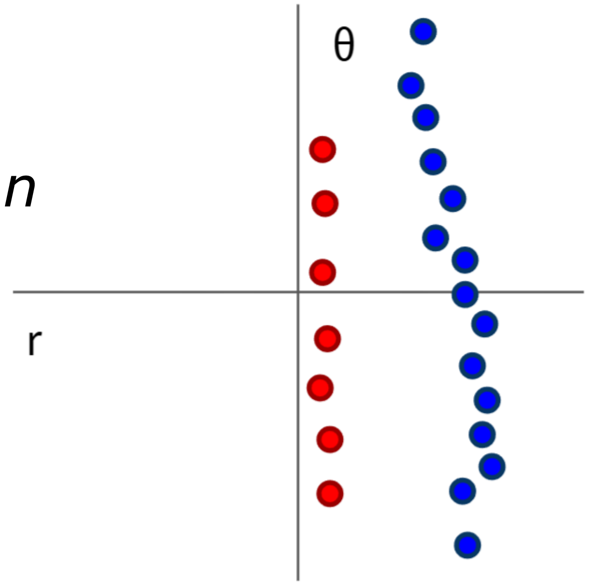
Learnable biases

What is the purpose of the **activation functions**?

Why do we want non-linearity?



Non-linear transformation

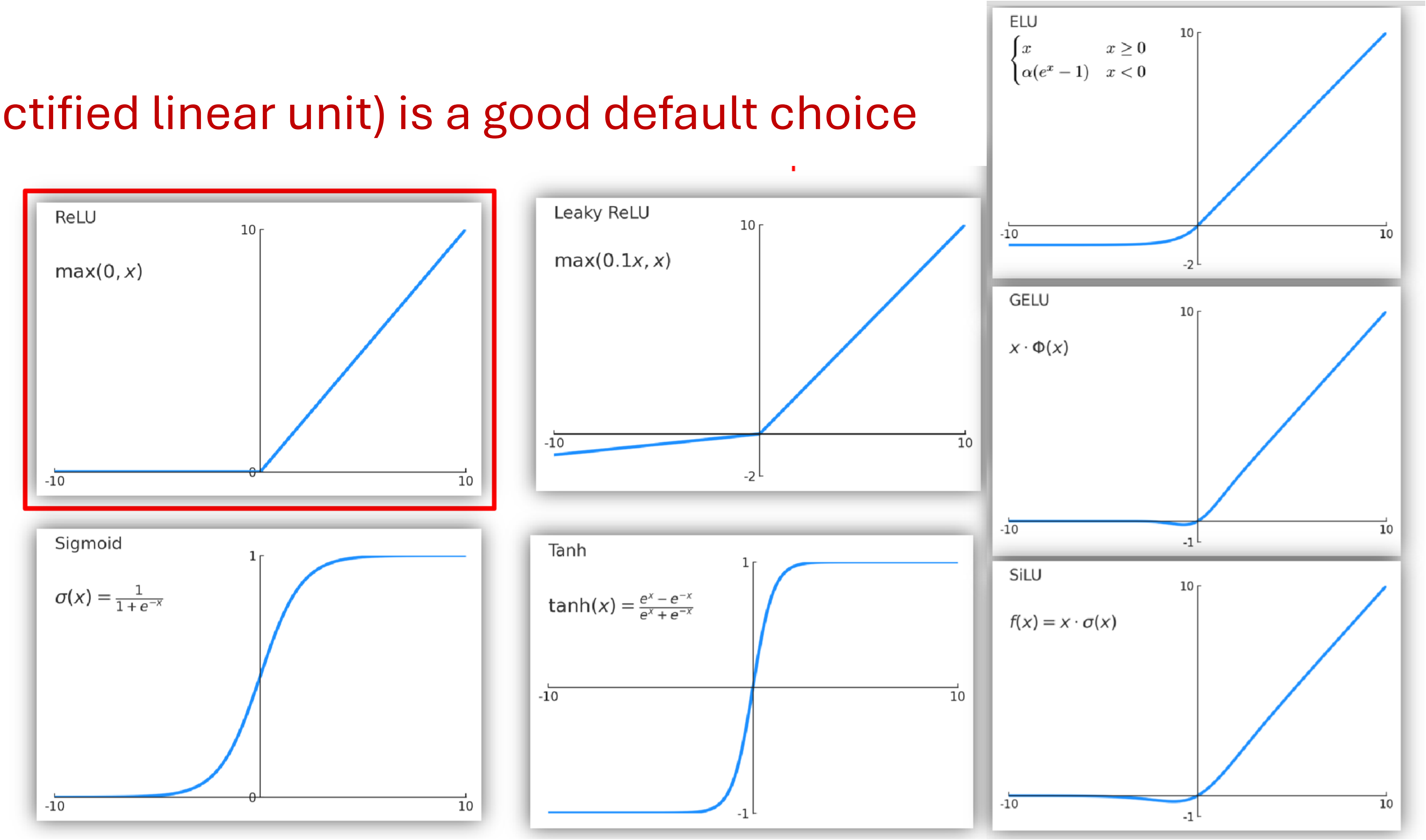


Cannot linearly separate red and blue points

But we can after applying a non-linear transformation!

What kind of non-linearity?

ReLU (rectified linear unit) is a good default choice



Credit: Stanford CS231n

Sample implementation

```python
import numpy as np # type: ignore
from numpy.random import randn # type: ignore

N, D_in, H, D_out = 64, 4, 100, 3
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    hidden = np.max(0, x.dot(w1))
    logits = hidden.dot(w2)
    y_pred = np.exp(logits) / np.exp(logits).sum(axis=1, keepdims=True)

    loss = -np.log(y_pred[np.arange(N),y]).mean()
    grad_y_pred = y_pred.copy()
    grad_y_pred[np.arange(N), y] -= 1.0
    grad_y_pred /= N
    grad_w2 = hidden.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T); grad_h[hidden <= 0] = 0
    grad_w1 = x.T.dot(grad_h)

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```

Sample implementation

```python
import numpy as np # type: ignore
from numpy.random import randn # type: ignore

N, D_in, H, D_out = 64, 4, 100, 3
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    hidden = np.max(0, x.dot(w1))
    logits = hidden.dot(w2)
    y_pred = np.exp(logits) / np.exp(logits).sum(axis=1, keepdims=True)

    loss = -np.log(y_pred[np.arange(N),y]).mean()
    grad_y_pred = y_pred.copy()
    grad_y_pred[np.arange(N), y] -= 1.0
    grad_y_pred /= N
    grad_w2 = hidden.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T); grad_h[hidden <= 0] = 0
    grad_w1 = x.T.dot(grad_h)

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```

Forward pass

Hidden activation = relu

Output activation = softmax

Sample implementation

```python
import numpy as np # type: ignore
from numpy.random import randn # type: ignore

N, D_in, H, D_out = 64, 4, 100, 3
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    hidden = np.max(0, x.dot(w1))
    logits = hidden.dot(w2)
    y_pred = np.exp(logits) / np.exp(logits).sum(axis=1, keepdims=True)

    loss = -np.log(y_pred[np.arange(N),y]).mean()
    grad_y_pred = y_pred.copy()
    grad_y_pred[np.arange(N), y] -= 1.0
    grad_y_pred /= N
    grad_w2 = hidden.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T); grad_h[hidden <= 0] = 0
    grad_w1 = x.T.dot(grad_h)

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```

Calculate gradient

Sample implementation

```
import numpy as np # type: ignore
from numpy.random import randn # type: ignore

N, D_in, H, D_out = 64, 4, 100, 3
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    hidden = np.max(0, x.dot(w1))
    logits = hidden.dot(w2)
    y_pred = np.exp(logits) / np.exp(logits).sum(axis=1, keepdims=True)

    loss = -np.log(y_pred[np.arange(N),y]).mean()
    grad_y_pred = y_pred.copy()
    grad_y_pred[np.arange(N), y] -= 1.0
    grad_y_pred /= N
    grad_w2 = hidden.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T); grad_h[hidden <= 0] = 0
    grad_w1 = x.T.dot(grad_h)

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```

Gradient descent

Learning rate = 1e-4

Sample implementation

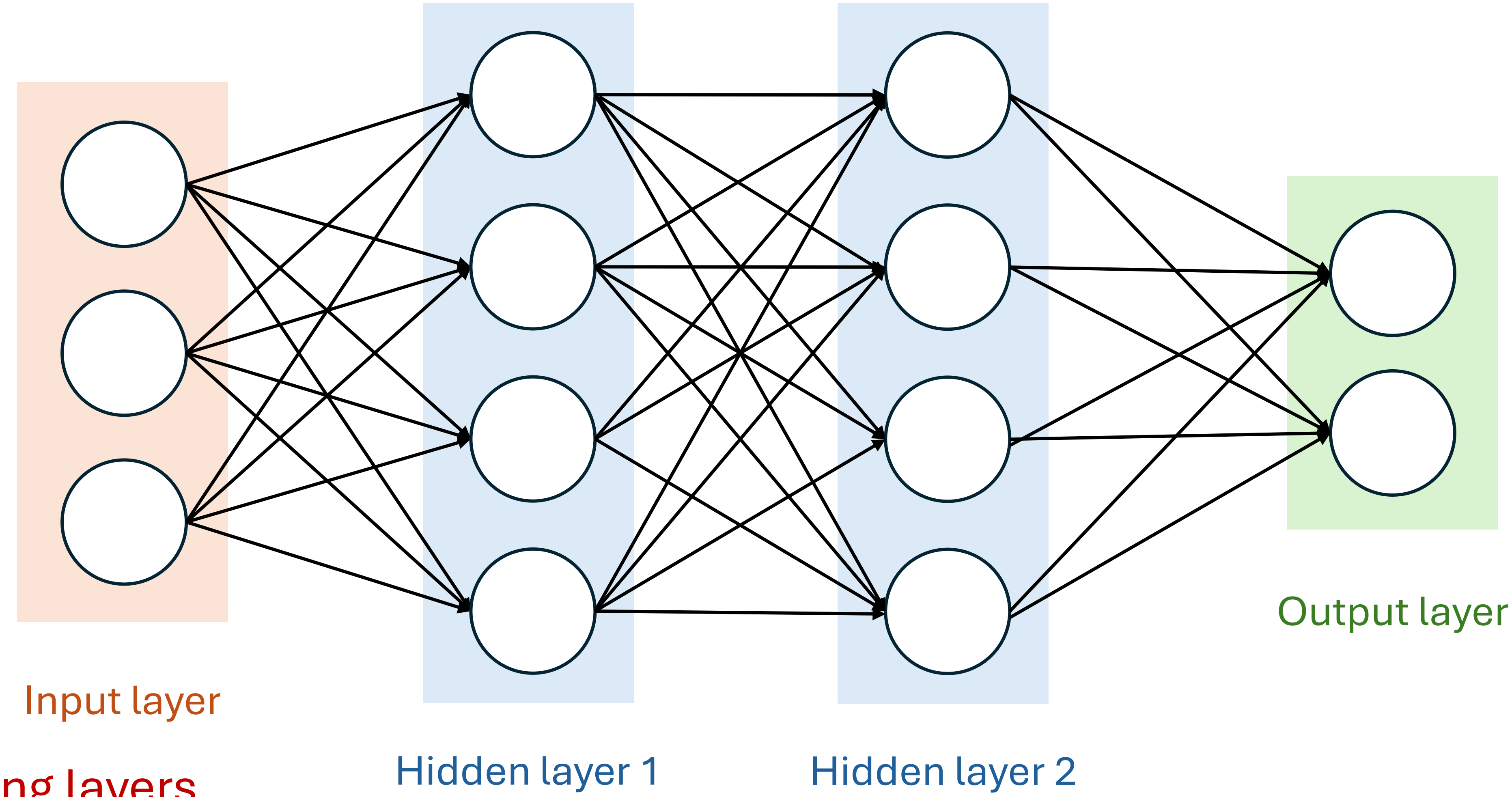
```python
import numpy as np # type: ignore
from numpy.random import randn # type: ignore

N, D_in, H, D_out = 64, 4, 100, 3
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
    hidden = np.max(0, x.dot(w1))
    logits = hidden.dot(w2)
    y_pred = np.exp(logits) / np.exp(logits).sum(axis=1, keepdims=True)

    loss = -np.log(y_pred[np.arange(N),y]).mean()
    grad_y_pred = y_pred.copy()
    grad_y_pred[np.arange(N), y] -= 1.0
    grad_y_pred /= N
    grad_w2 = hidden.T.dot(grad_y_pred)
    grad_h = grad_y_pred.dot(w2.T); grad_h[hidden <= 0] = 0
    grad_w1 = x.T.dot(grad_h)

    w1 -= 1e-4 * grad_w1
    w2 -= 1e-4 * grad_w2
```

Note: Ignored bias term here.

Gradient descent

Learning rate = 1e-4

What is a “deep” neural network?



Just keep adding layers...

(Typically “deep” networks have far more than 2 layers)

How do we train deep neural networks?

Analytical gradients not practical to calculate with many layers...

- Tedious
- Breaks if we change loss functions
- Not possible if model is very deep/complex

Solution: use backpropagation

- Backpropagation + PyTorch auto-differentiation = easy

```
loss.backward()
```

How do we train deep neural networks?

Analytical gradients not practical to calculate with many layers...

- Tedious
- Breaks if we change loss functions
- Not possible if model is very deep/c

Will not cover backprop theory here.
Please consult resources.

Solution: use backpropagation

- Backpropagation + PyTorch auto-differentiation = easy

```
loss.backward()
```

Classification with images

Does our feed-forward neural network work if x is an image?

Input x → f (Model) → Label y

Classification with images

Does our feed-forward neural network work if x is an image?

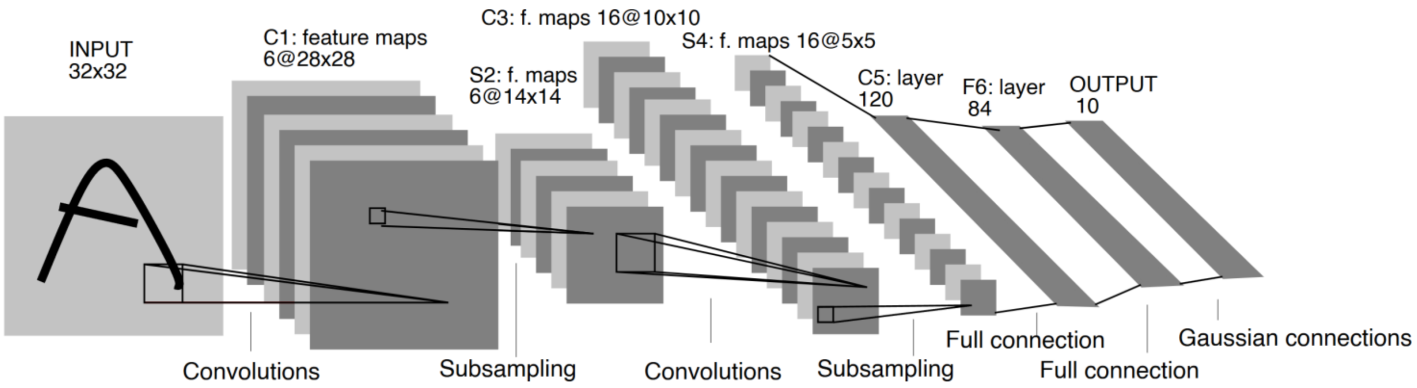
Not very well!

- Each pixel is treated as an independent feature
- No translation invariance
- The number of parameters explodes (224x224x3 image with 1,000 hidden units = 150M parameters)

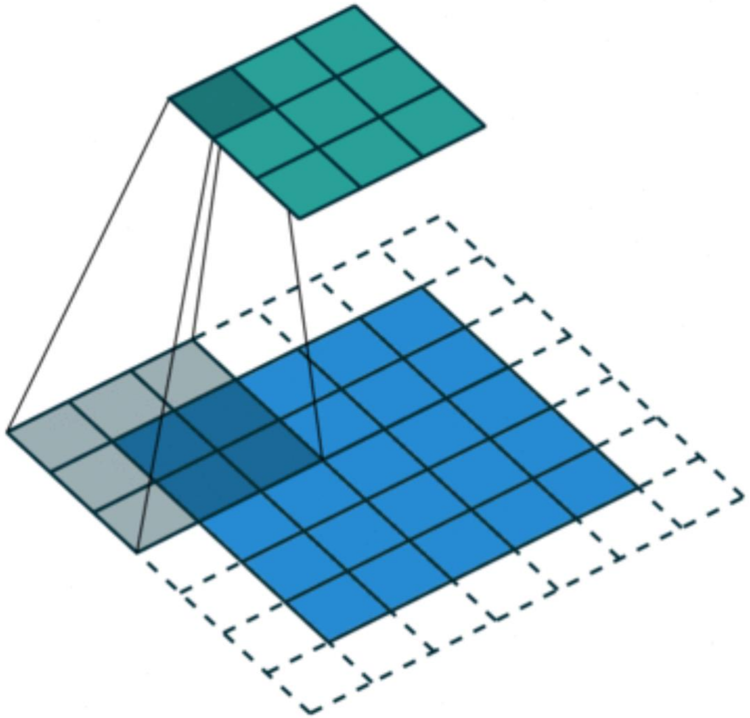
Convolutional neural networks

Captures spatial dependencies in images through use of filters

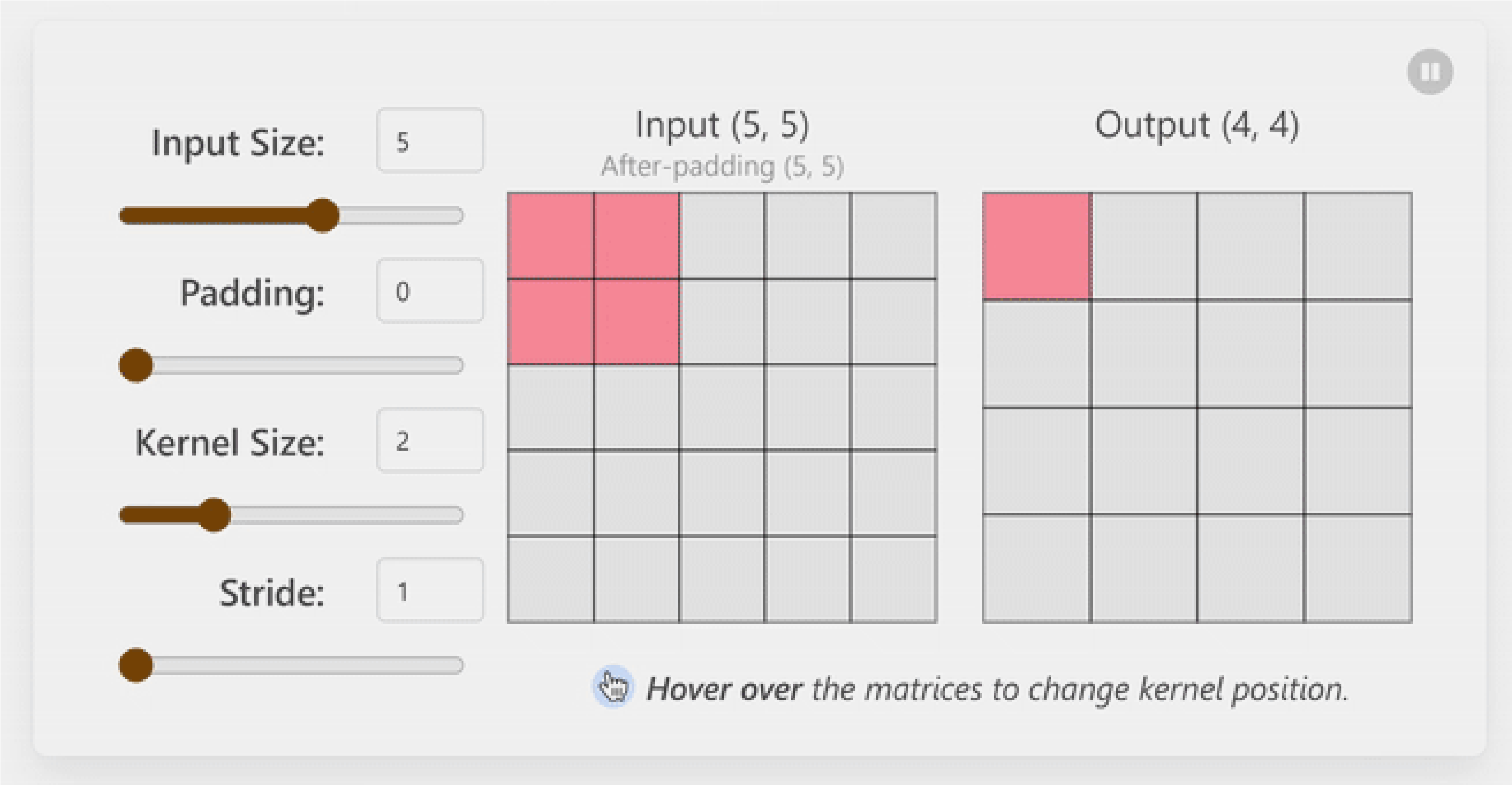
- Spatial locality
- Translation invariance
- Parameter sharing



Yann LeCun's LeNet (1998)



How do filters work?

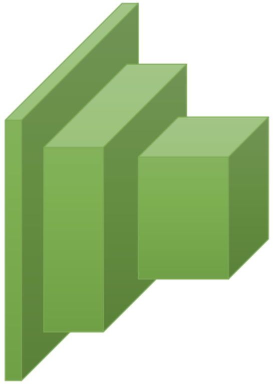


https://poloclub.github.io/cnn-explainer/

Classification with CNNs

Replace our feed-forward DNN with a CNN

Input x $\longrightarrow$



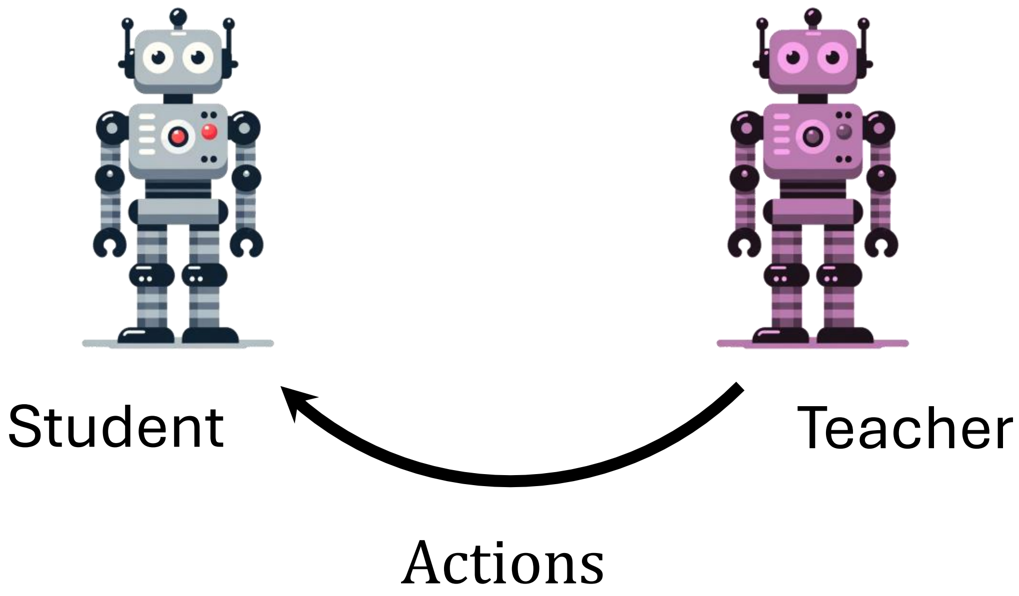
Model f

$\longrightarrow$ Label y

Behavior cloning

Imitation learning

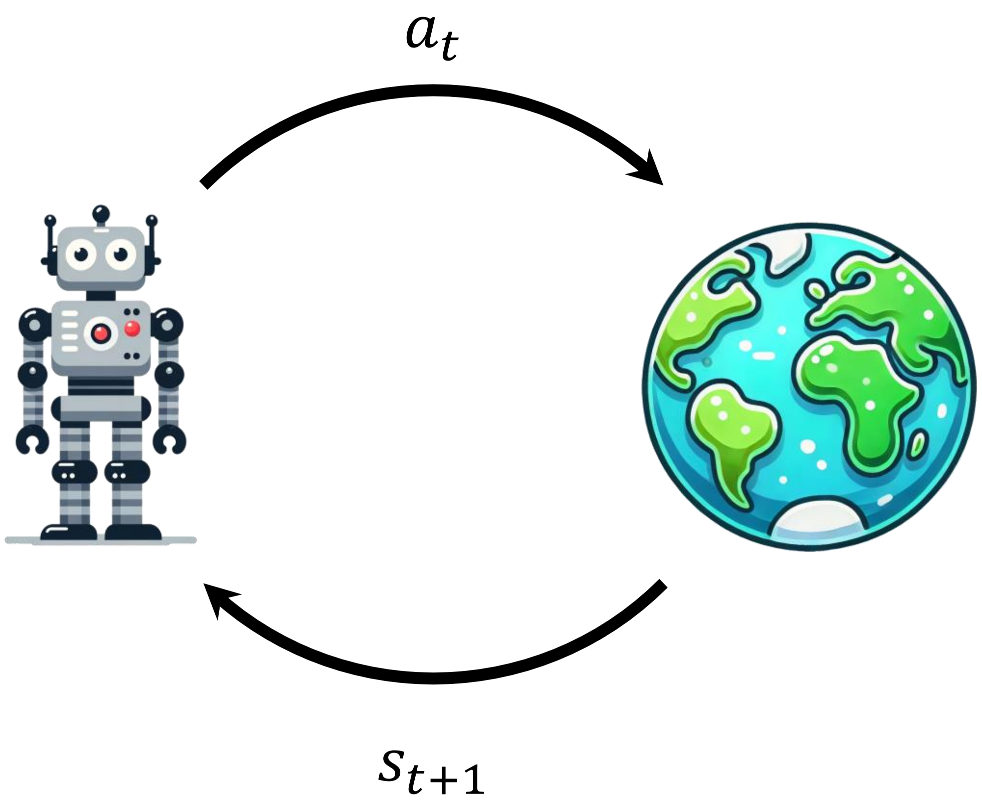
The student tries to learn and imitate the teacher's actions



The basics

The agent and environment operate at discrete timesteps $t = 0,1,2, \ldots$

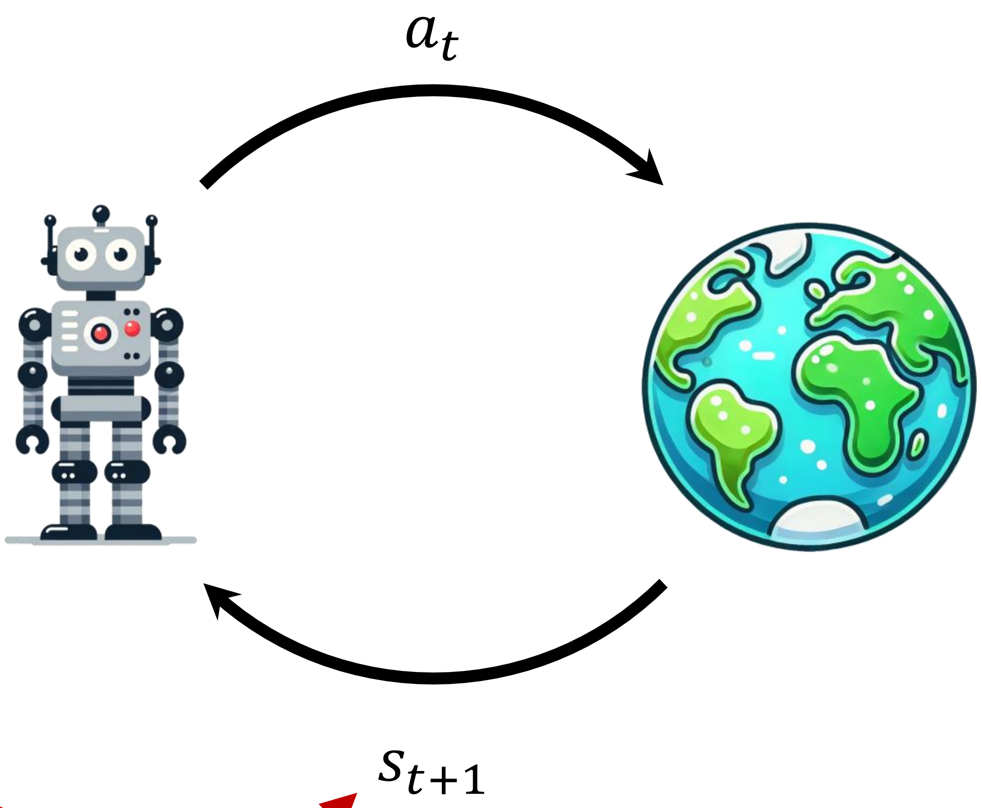
- The agent observes state s_t at time t
- The agent takes action a_t
- The agent gets the subsequent state s_{t+1}



The basics

The agent and environment operate at discrete timesteps $t = 0,1,2,\ldots$

- The agent observes state s_t at time t
- The agent takes action a_t
- The agent gets the subsequent state s_{t+1}



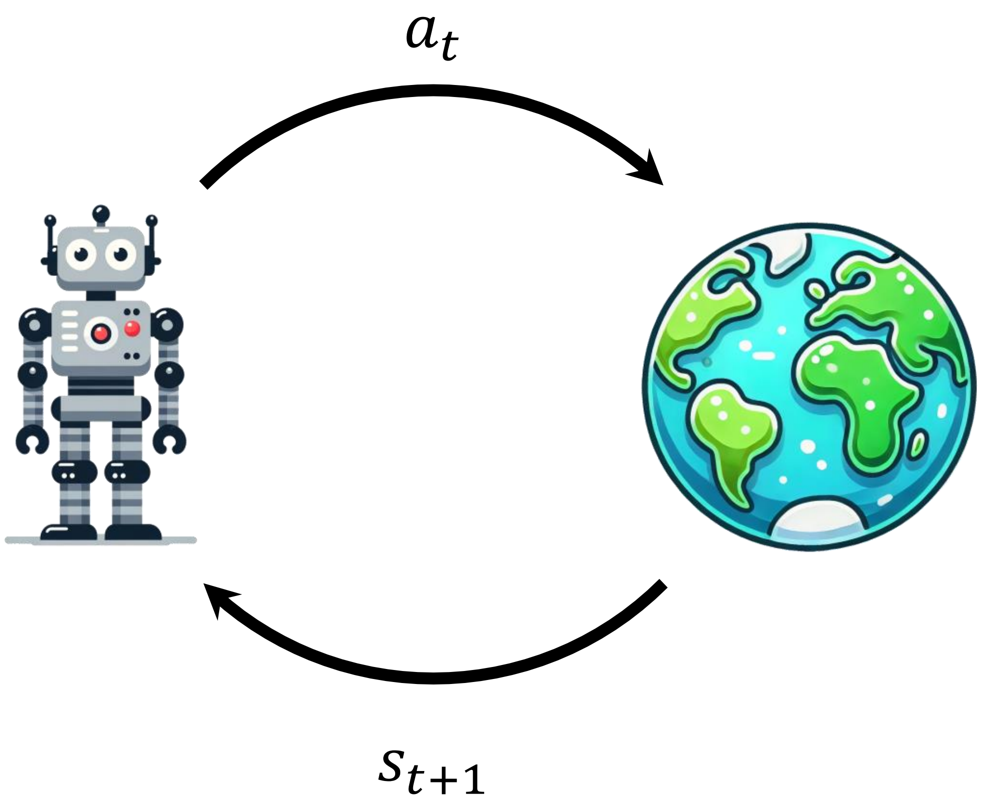
Unlike last time, we no longer have a reward!

The basics

Action a_t is chosen by sampling actions from a probability distribution

$$a_t \sim \pi(a|s)$$

The probability distribution π is referred to as a policy.



Types of imitation learning

Assumption: teacher provides a set of demonstrations consisting of sequences of state-action pairs (episode rollouts)

Behavior cloning

- The student performs supervised learning and tries to approximate the teacher's underlying policy π

Inverse reinforcement learning + reinforcement learning

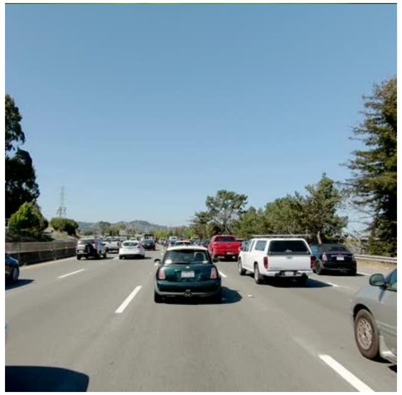
- The student first infers the teacher's reward function (IRL)
- The student then performs RL using this reward function

Supervised learning

"Iris Setosa"

Input x Model f Output $y = f(x)$

Behavior cloning



"Slow down"

Input x Model f Output $y = f(x)$

Behavior cloning: simple imitation learning

Most basic algorithm is simply supervised learning with two steps.

1) Collect training data of demonstrations.

$$d_j = \{(s_j^1, a_j^1), (s_j^2, a_j^2), \ldots, (s_j^k, a_j^k)\}$$

2) Train policy

$$D = \{d_1, d_2, \ldots, d_N\} \rightarrow \boxed{\text{Algorithm}} \rightarrow \pi = \mathcal{S} \rightarrow \mathcal{A}$$

Koenemann, J., et al. Real-time imitation of human whole-body motions by humanoids. ICRA 2014.

How can we obtain demonstrations?

Assumption: have access to a better-than-random policy

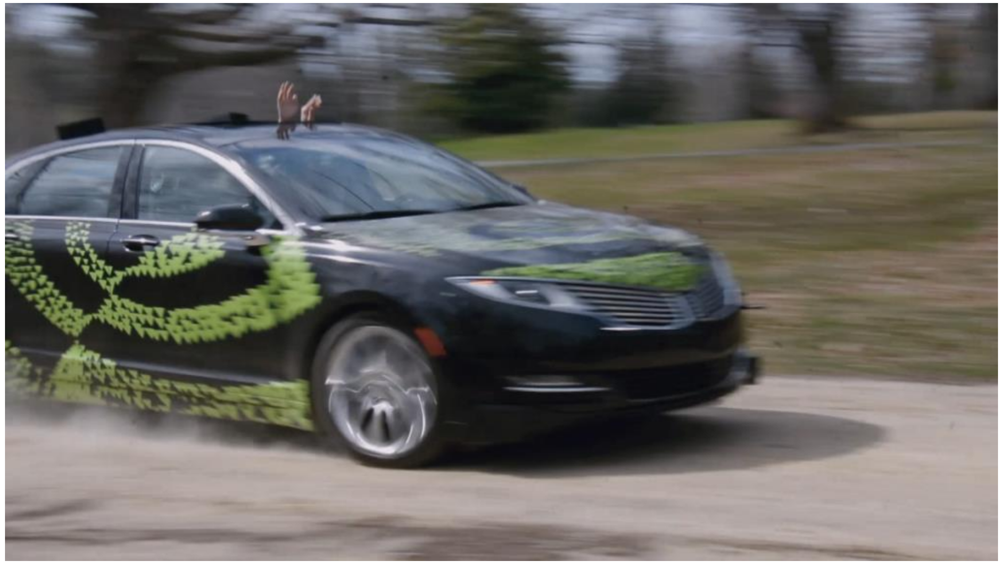
- Humans (domain experts)
- Previously trained policies
- Heuristics
- ...

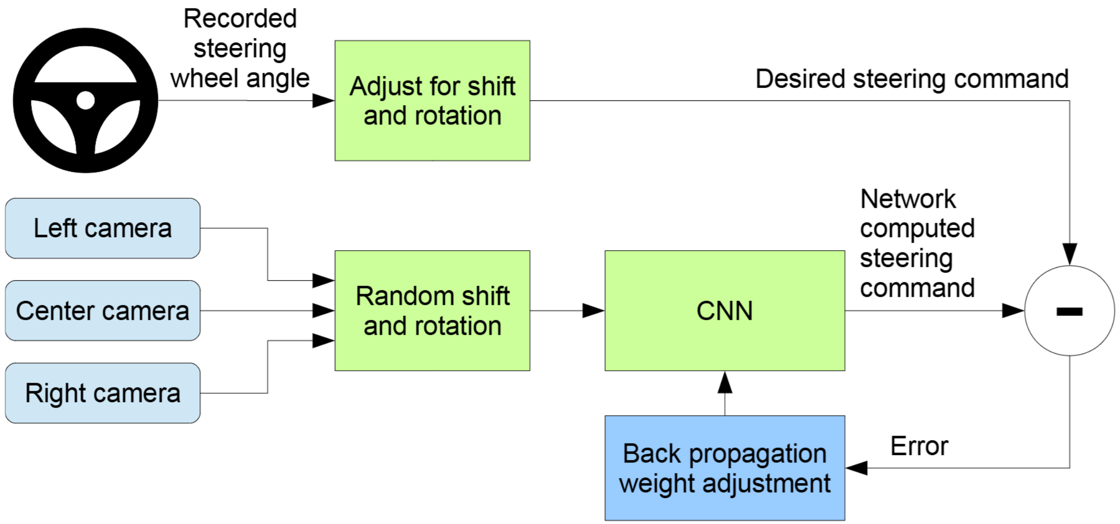
Fu et al. *Mobile Aloha*. https://mobile-aloha.github.io/

Fu et al. *Mobile Aloha*. https://mobile-aloha.github.io/

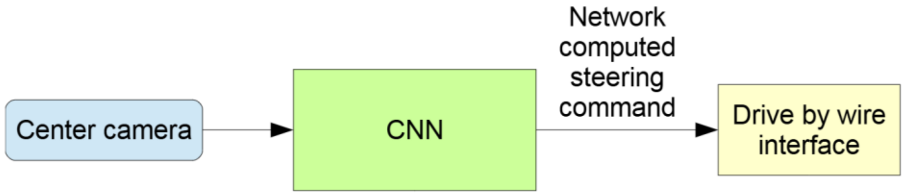
Real world example: autonomous driving

NVIDIA trained a model for self-driving cars in 2016 using BC.





Train

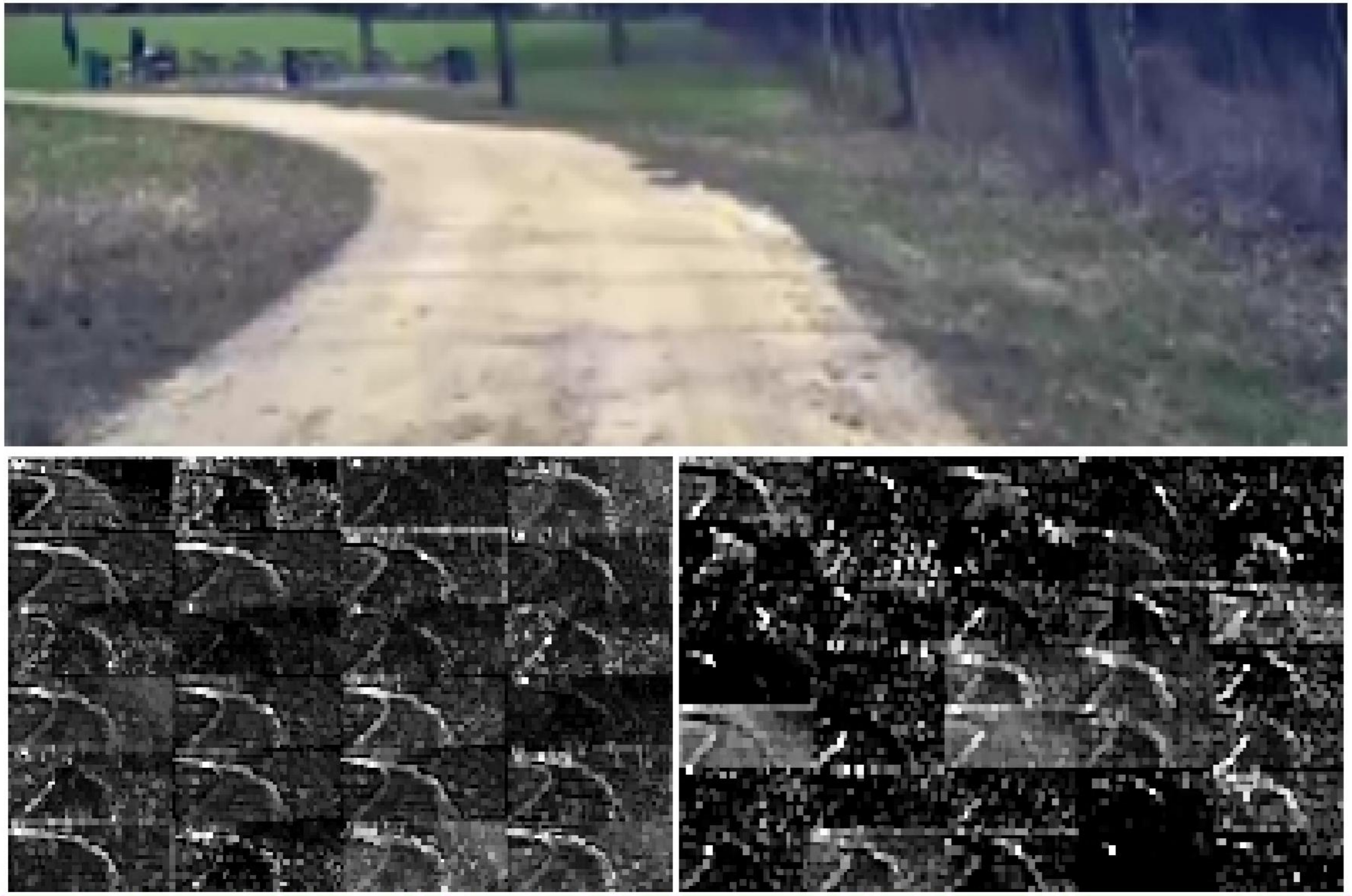


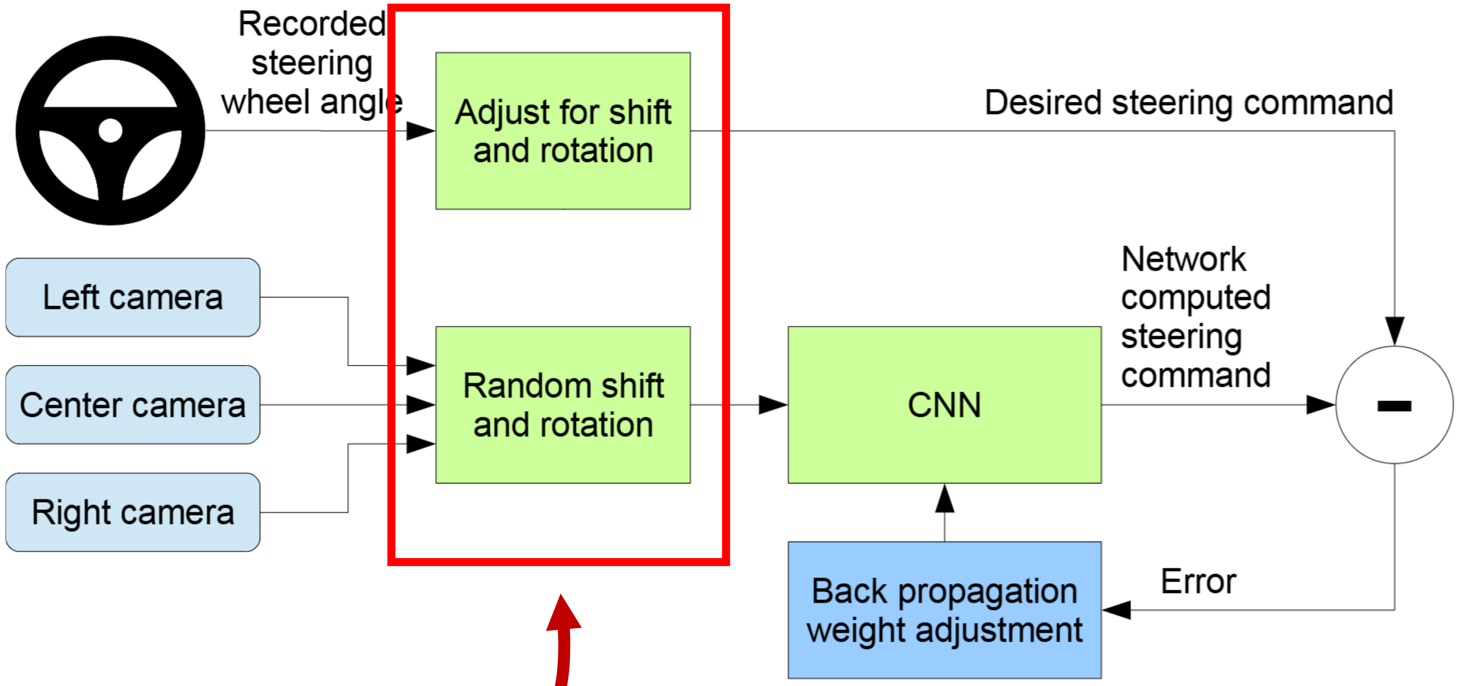
Test

https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

https://youtu.be/NJU9ULQUwng?si=sAPKpPIis2xlVFfL&t=68

CNNs learn to detect road features





But wait, what is this for!?

Example: expert teacher

How does the student know how to recover if it makes a mistake that the teacher never did?

Teacher (Expert)

Student

Supervised learning and the i.i.d. assumption

Identically distributed

- Every data point comes from the same probability distribution
- *E.g., flipping a coin is always a 50/50 chance of heads or tails*

Independent

- Data points are independent events, they do not influence each other
- *E.g., a tails on one flip does not affect the outcome of the next flip*

Example: binary classifier

Suppose we have a simple binary classifier (e.g. logistic regression)

If $y_i = 1$ then $p(y_i = 1|x_i; \theta) = f_\theta(x_i)$

If $y_i = 0$ then $p(y_i = 0|x_i; \theta) = 1 - f_\theta(x_i)$

Example: binary classifier

The likelihood function represents the probability of observing the dataset given the model parameters θ

- For n datapoints $\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$

$$\mathcal{L}(\theta) = p(y_0, y_1, \dots, y_{n-1} | x_0, x_1, \dots, x_{n-1}; \theta)$$

If samples are independent then this simplifies to…

$$\mathcal{L}(\theta) = \prod_{i=0}^{n-1} p(y_i | x_i; \theta)$$

Example: binary classifier

The likelihood function represents the probability of observing the dataset given the model parameters θ

- For n datapoints $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$

$$\mathcal{L}(\quad \theta)$$

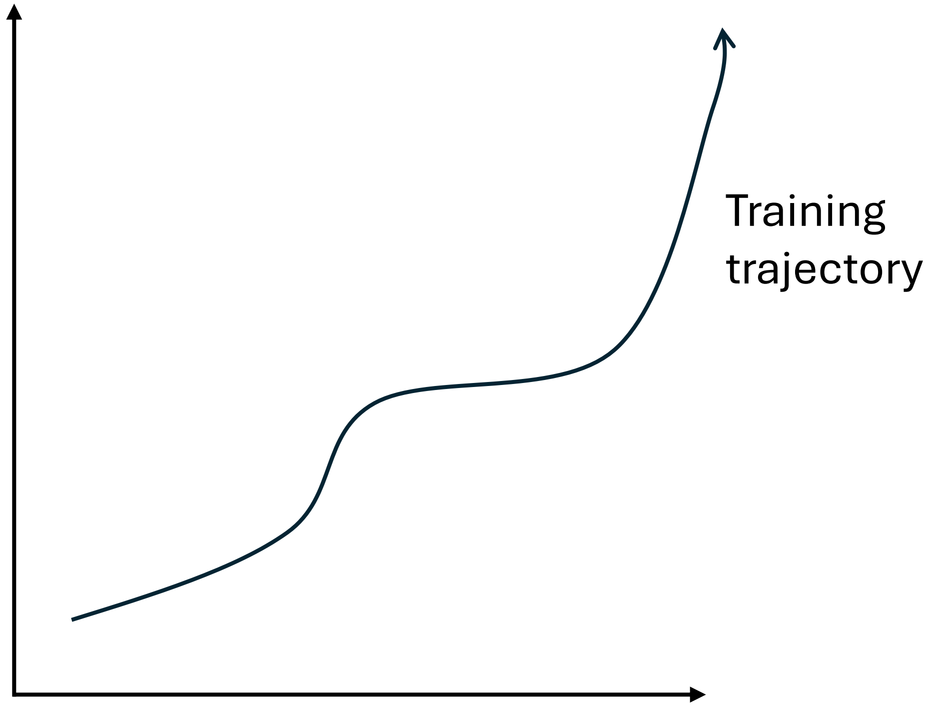
Many ML algorithms rely on the i.i.d. assumption for theoretical guarantees

If samples are independent then this simplifies to…

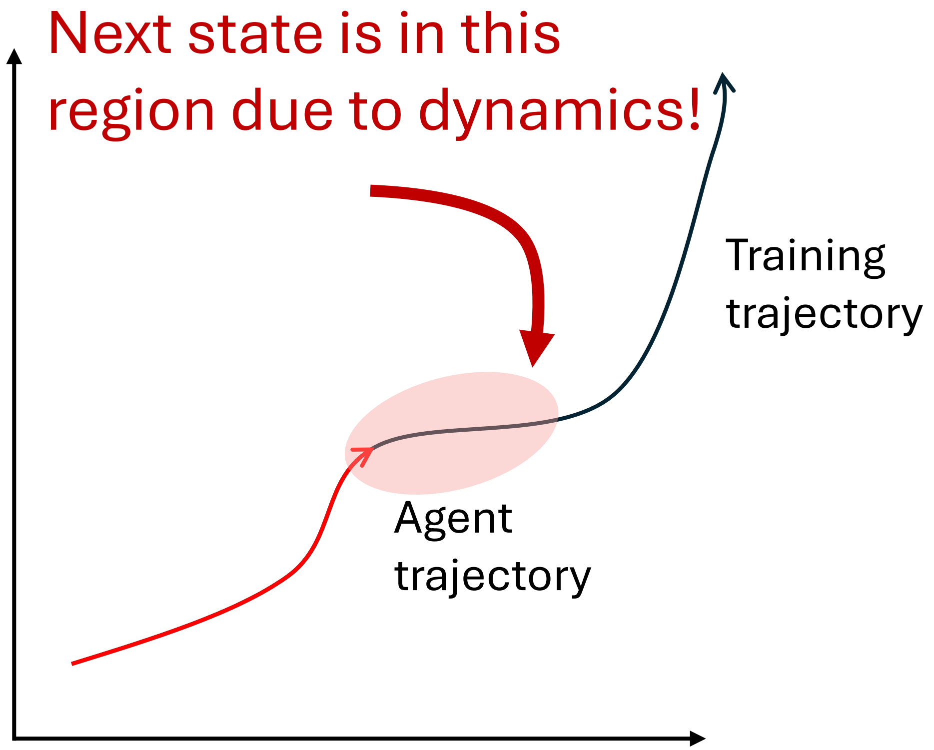
$$\mathcal{L}(\theta) = \prod_{i=0}^{n-1} p(y_i | x_i; \theta)$$

Does i.i.d. hold in behavior cloning?

Does i.i.d. hold in behavior cloning?



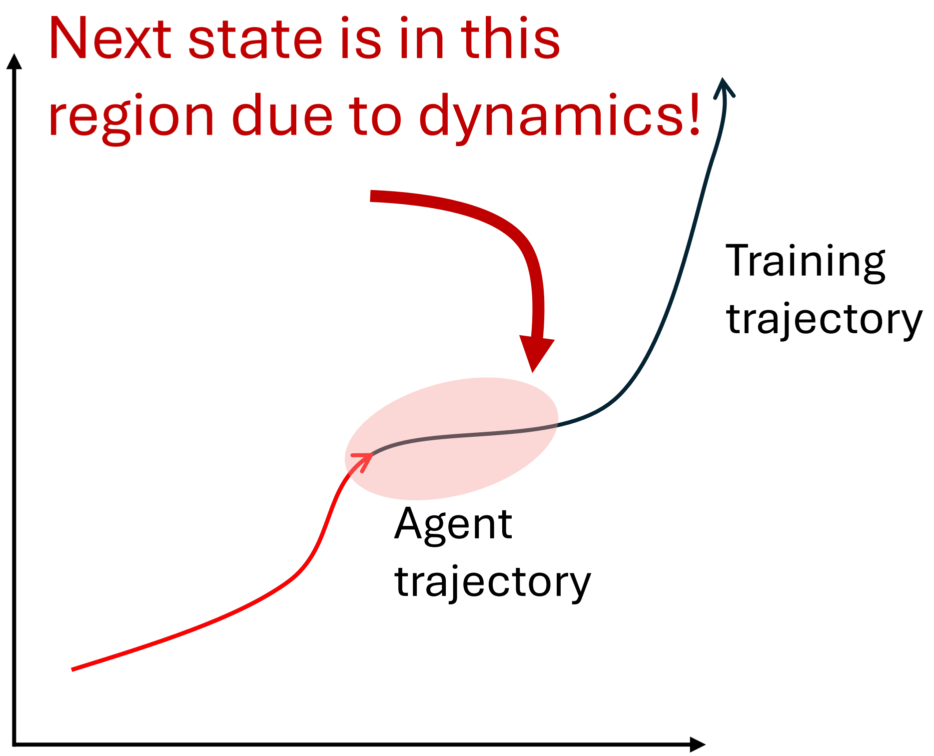
Does i.i.d. hold in behavior cloning?



Does i.i.d. hold in behavior cloning?

Samples are **not** independent!

- The next state is influenced by the current state

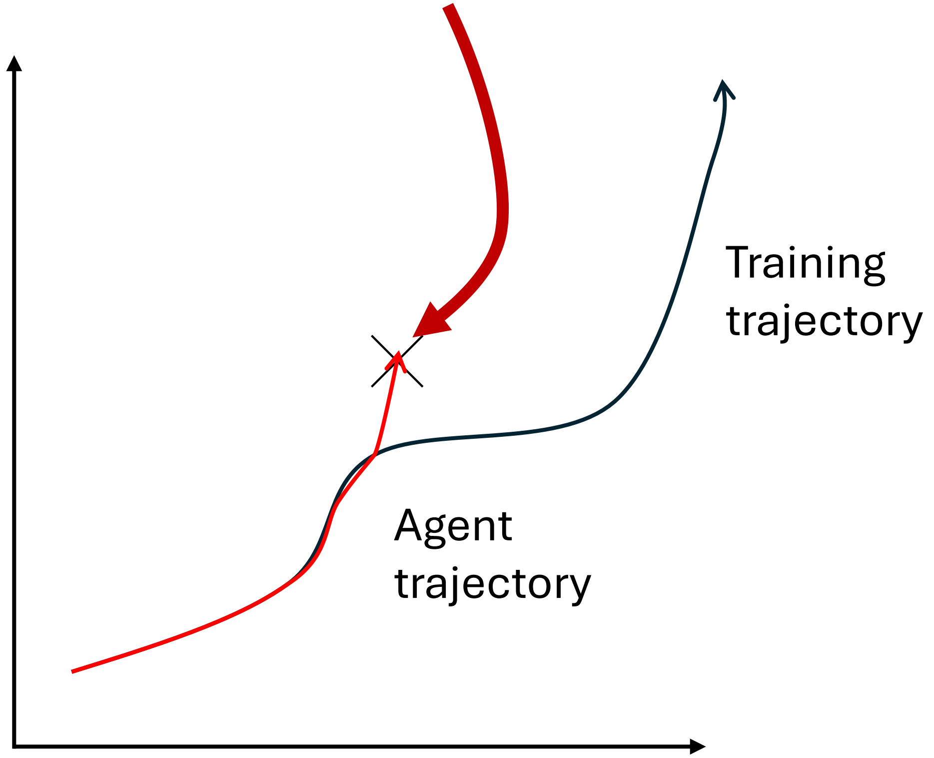


Does i.i.d. hold in behavior cloning?

Samples are **not** independent!

- The next state is influenced by the current state

Suppose the agent makes a mistake and is now in an "unseen" location

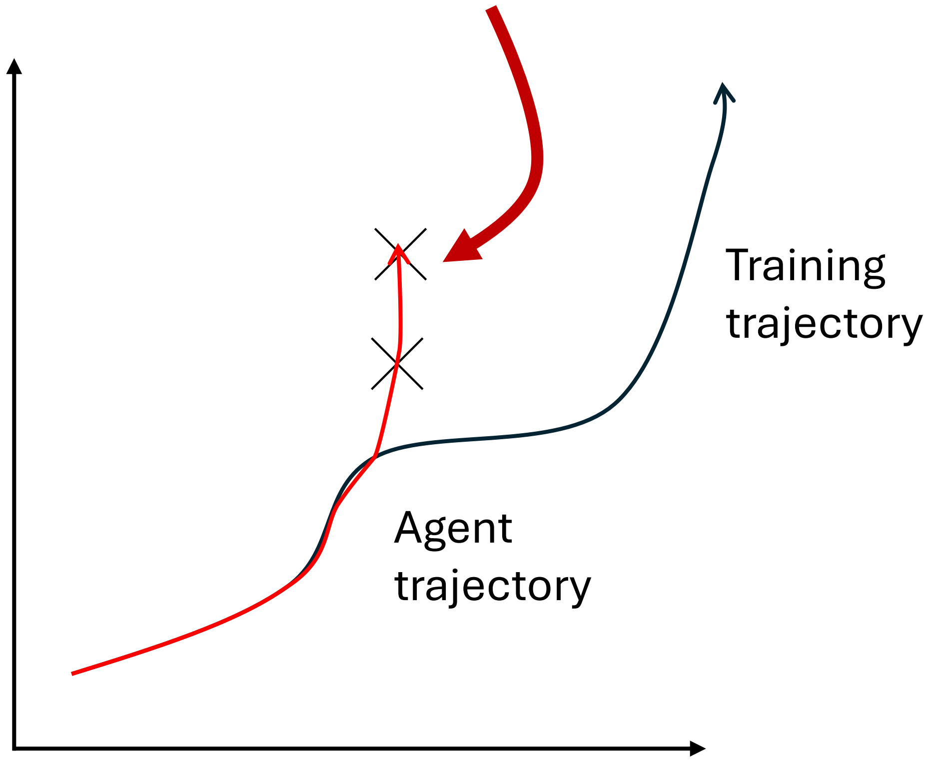


Does i.i.d. hold in behavior cloning?

Samples are **not** independent!

- The next state is influenced by the current state

It is now *more* likely to make a mistake in the future! Errors accumulate.



Does i.i.d. hold in behavior cloning?

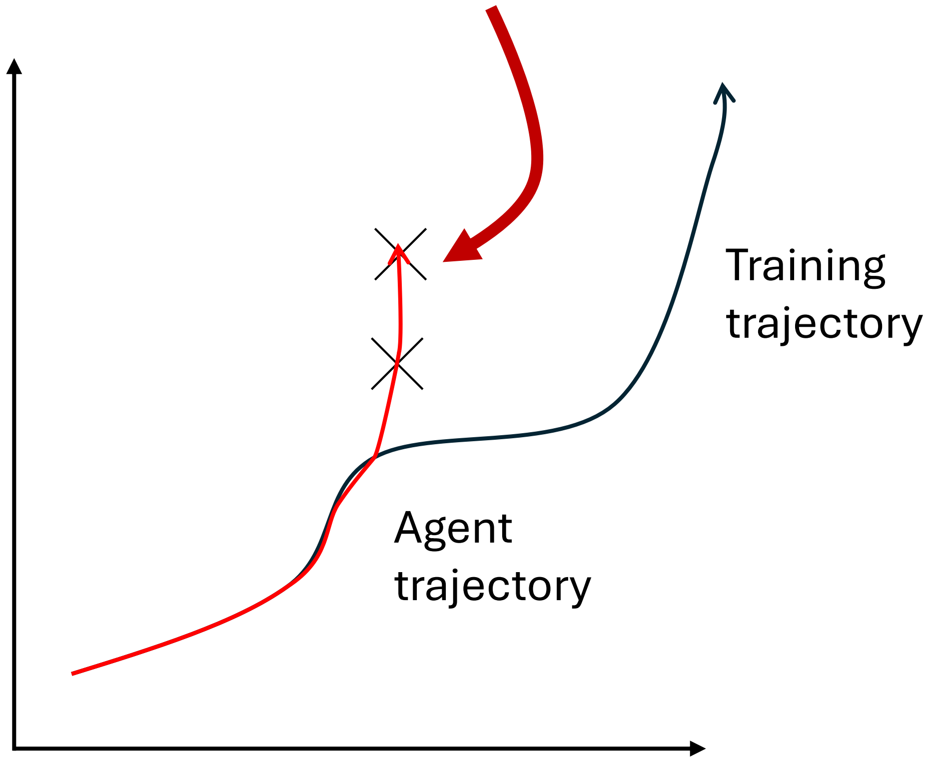
Samples are **not** independent!

- The next state is influenced by the current state

Samples are **not** identically distributed!

- Error accumulation means $p_{\text{data}}(s_t) \neq p_{\pi}(s_t)$

It is now *more* likely to make a mistake in the future! Errors accumulate.



Does i.i.d. hold in behavior cloning?

If a classifier makes a mistake with probability ϵ, it can make as many as $T^2\epsilon$ mistakes over T steps under the distribution of states the classifier itself induces.

itself induces (Ross and Bagnell, 2010). Intuitively this is because as soon as the learner makes a mistake, it may encounter completely different observations than those under expert demonstration, leading to a compounding of errors.

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

Stéphane Ross
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
stephaneross@cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
ggordon@cs.cmu.edu

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dbagnell@ri.cmu.edu

Abstract

Sequential prediction problems such as imitation learning, where future observations depend on previous predictions (actions), violate the common i.i.d. assumptions made in statistical learning. This leads to poor performance in theory and often in practice. Some recent approaches (Daumé III et al., 2009; Ross and Bagnell, 2010) provide stronger guarantees in this setting, but remain somewhat unsatisfactory as they train either non-stationary or stochastic policies and require a large number of iterations. In this paper, we propose a new iterative algorithm, which trains a stationary deterministic policy, that can be seen as a no regret algorithm in an online learning setting. We show that any such no regret algorithm, combined with additional reduction assumptions, must find a policy with good performance under the distribution of observations it induces in such sequential settings. We demonstrate that this new approach outperforms previous approaches on two challenging imitation learning problems and a benchmark sequence labeling problem.

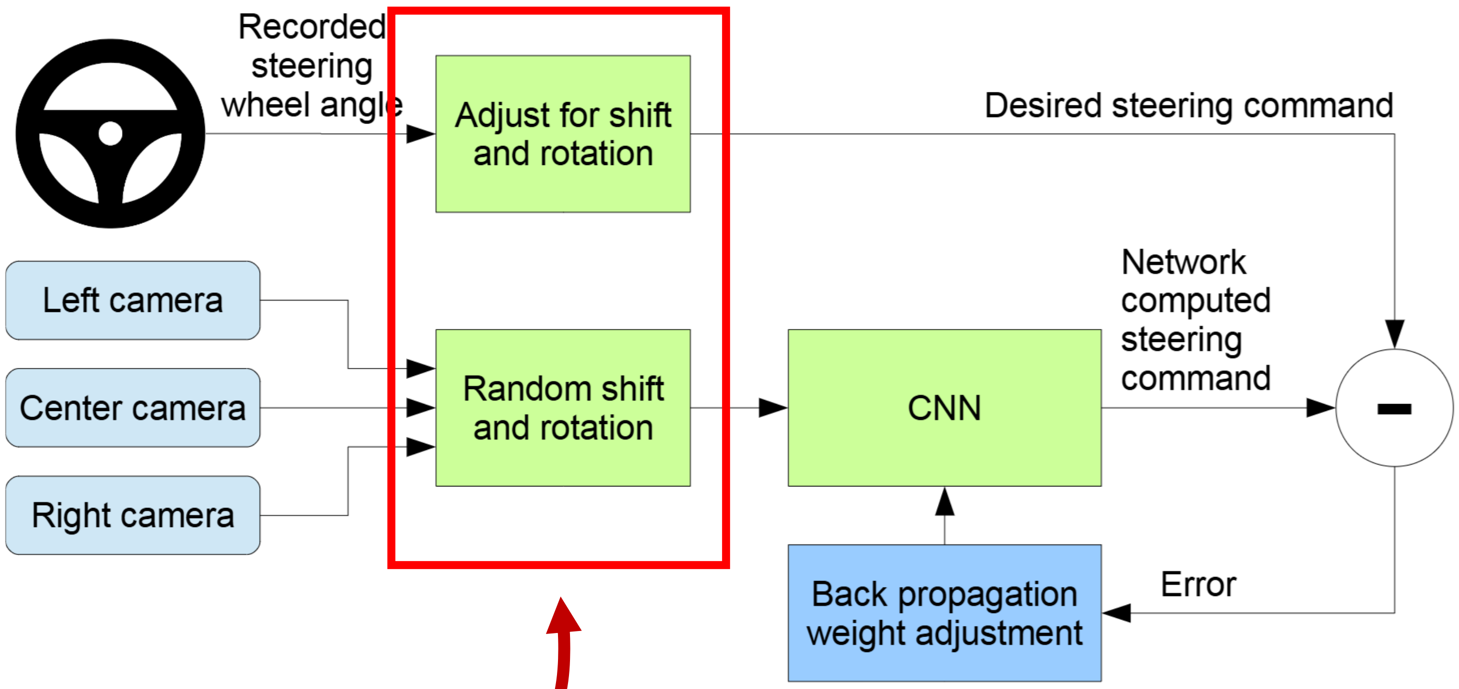
1 INTRODUCTION

Sequence Prediction problems arise commonly in practice. For instance, most robotic systems must be able to predict/make a sequence of actions given a sequence of observations revealed to them over time. In complex robotic systems where standard control methods fail, we must often resort to learning a controller that can make such predictions. Imitation learning techniques, where expert demonstrations of good behavior are used to learn a controller, have proven very useful in practice and have led to state-of-the art performance in a variety of applications (Schaal, 1999; Abbeel and Ng, 2004; Ratliff et al., 2006; Silver et al., 2008; Argall et al., 2009; Chernova and Veloso, 2009; Ross and Bagnell, 2010). A typical approach to imitation learning is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert. However since the learner's prediction affects future input observations/states during execution of the learned policy, this violate the crucial i.i.d. assumption made by most statistical learning approaches.

Appearing in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15. Copyright 2011 by the authors.

Ignoring this issue leads to poor performance both in theory and practice (Ross and Bagnell, 2010). In particular, a classifier that makes a mistake with probability ϵ under the distribution of states/observations encountered by the expert can make as many as $T^2\epsilon$ mistakes in expectation over T-steps under the distribution of states the classifier itself induces (Ross and Bagnell, 2010). Intuitively this is because as soon as the learner makes a mistake, it may encounter completely different observations than those under expert demonstration, leading to a compounding of errors.

Recent approaches (Ross and Bagnell, 2010) can guarantee an expected number of mistakes linear (or nearly so) in the task horizon T and error ϵ by training over several iterations and allowing the learner to influence the input states where expert demonstration is provided (through execution of its own controls in the system). One approach (Ross and Bagnell, 2010) learns a non-stationary policy by training a different policy for each time step in sequence, starting from the first step. Unfortunately this is impractical when T is large or ill-defined. Another approach called SMILe (Ross and Bagnell, 2010), similar to SEARN (Daumé III et al., 2009) and CPI (Kakade and Langford, 2002), trains a stationary stochastic policy (a finite mixture of policies) by adding a new policy to the mixture at each iteration of training. However this may be unsatisfactory for practical applications as some policies in the mixture are worse than



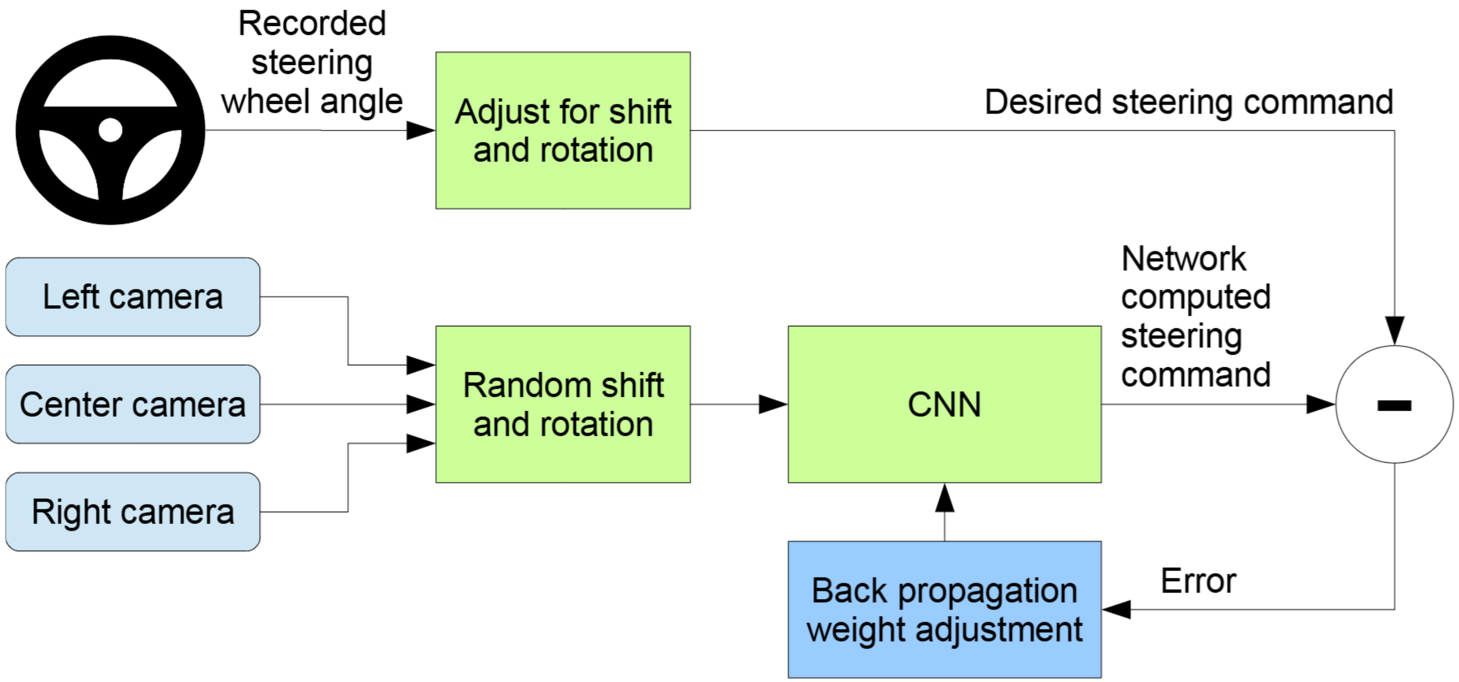
This is data augmentation!

Recovering from mistakes

Original "good" image

Transform it to represent a shift towards center-line

Change the corresponding action to account for recovery,
e.g. steer more to the right.



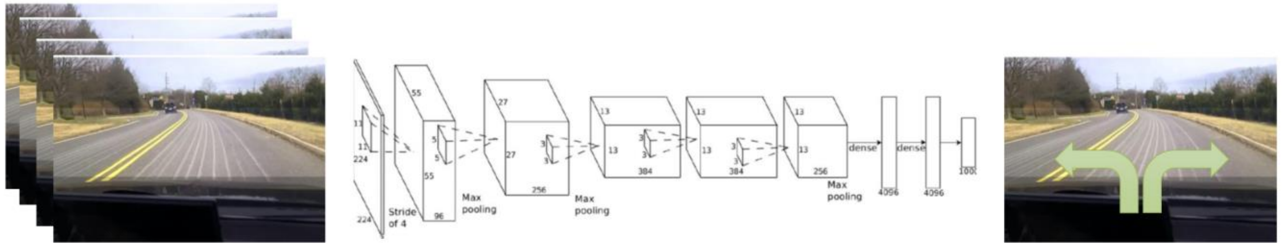
Training dataset has been augmented to include artificial shifts and rotations to teach the network how to recover from a poor position or orientation.

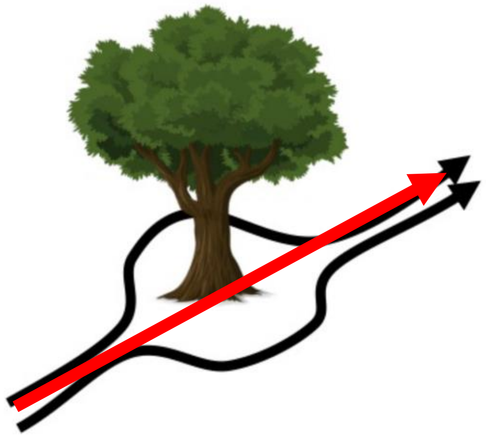
Bojarski et al, “End to End Learning for Self-Driving Cars”, 2016.

Why do errors occur?

Models are *not* perfect

- Insufficient training data
- Insufficient model capacity
- Non-Markovian dynamics
- Multimodal behavior
- …

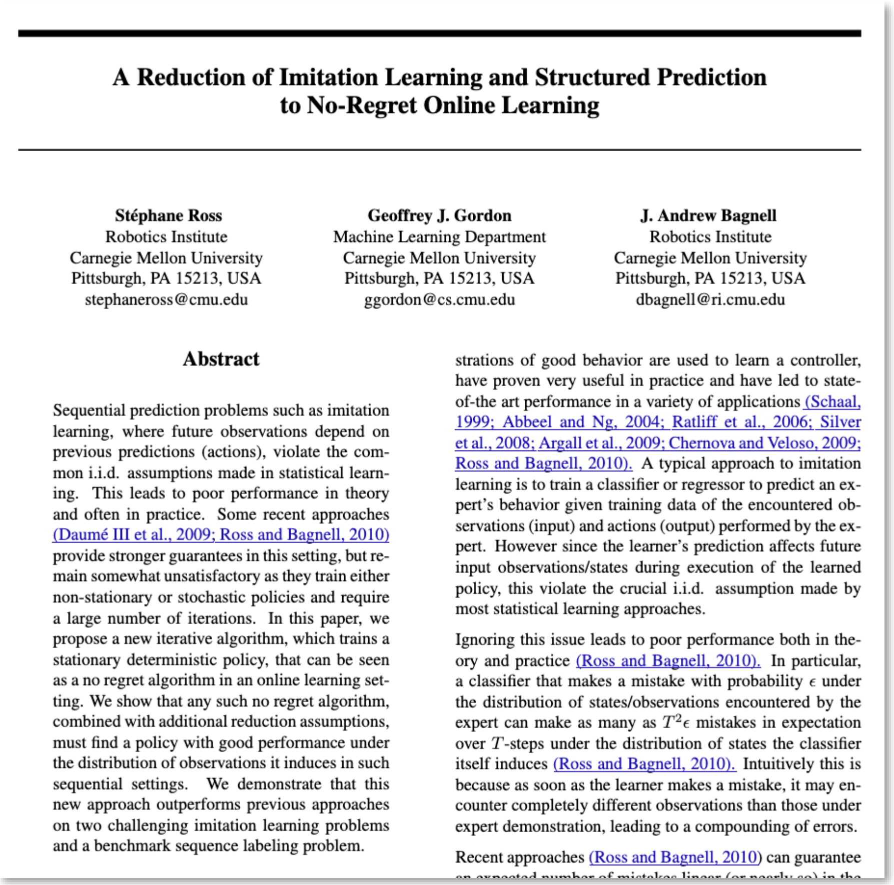




Another solution: dataset aggregation

- Step 1: Sample state-action pairs from environment and add to dataset
  - Teacher starts by taking all actions
  - Slowly allow student to start taking actions
  - After collection, re-label all actions with what teacher would have taken
- Step 2: Train policy over dataset using BC

Idea: student makes mistakes, teacher shows how to correct them



A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

Stéphane Ross
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
stephaneross@cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
ggordon@cs.cmu.edu

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dbagnell@ri.cmu.edu

Abstract

Sequential prediction problems such as imitation learning, where future observations depend on previous predictions (actions), violate the common i.i.d. assumptions made in statistical learning. This leads to poor performance in theory and often in practice. Some recent approaches (Daumé III et al., 2009; Ross and Bagnell, 2010) provide stronger guarantees in this setting, but remain somewhat unsatisfactory as they train either non-stationary or stochastic policies and require a large number of iterations. In this paper, we propose a new iterative algorithm, which trains a stationary deterministic policy, that can be seen as a no regret algorithm in an online learning setting. We show that any such no regret algorithm, combined with additional reduction assumptions, must find a policy with good performance under the distribution of observations it induces in such sequential settings. We demonstrate that this new approach outperforms previous approaches on two challenging imitation learning problems and a benchmark sequence labeling problem.

strations of good behavior are used to learn a controller, have proven very useful in practice and have led to state-of-the art performance in a variety of applications (Schaal, 1999; Abbeel and Ng, 2004; Ratliff et al., 2006; Silver et al., 2008; Argall et al., 2009; Chernova and Veloso, 2009; Ross and Bagnell, 2010). A typical approach to imitation learning is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert. However since the learner's prediction affects future input observations/states during execution of the learned policy, this violate the crucial i.i.d. assumption made by most statistical learning approaches.

Ignoring this issue leads to poor performance both in theory and practice (Ross and Bagnell, 2010). In particular, a classifier that makes a mistake with probability ϵ under the distribution of states/observations encountered by the expert can make as many as $T^2\epsilon$ mistakes in expectation over T-steps under the distribution of states the classifier itself induces (Ross and Bagnell, 2010). Intuitively this is because as soon as the learner makes a mistake, it may encounter completely different observations than those under expert demonstration, leading to a compounding of errors.

Recent approaches (Ross and Bagnell, 2010) can guarantee

Dataset Aggregation: incremental learning

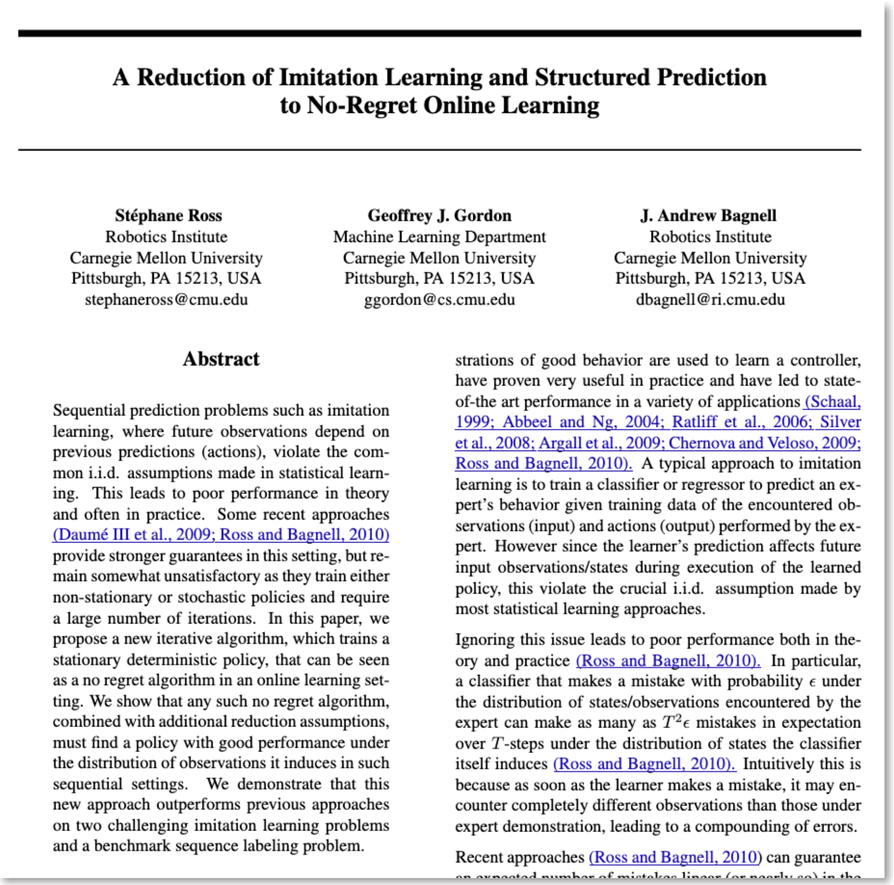
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in Π.
for $i = 1$ **to** N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
  Sample T-step trajectories using π_i.
  Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by π_i and actions given by expert.
  Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
  Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
end for
Return best $\hat{\pi}_i$ on validation.

Algorithm 3.1: DAGGER Algorithm.

Another solution: dataset aggregation

- Step 1: Sample state-action pairs from environment and add to dataset
  - Teacher starts by taking all actions
  - Slowly allow student to start taking actions
  - After collection, re-label all actions with what teacher would have taken
- Step 2: Train policy over dataset using BC

Assumption: teacher is available and capable of recovering from mistakes



A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning

Stéphane Ross
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
stephaneross@cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
ggordon@cs.cmu.edu

J. Andrew Bagnell
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dbagnell@ri.cmu.edu

Abstract

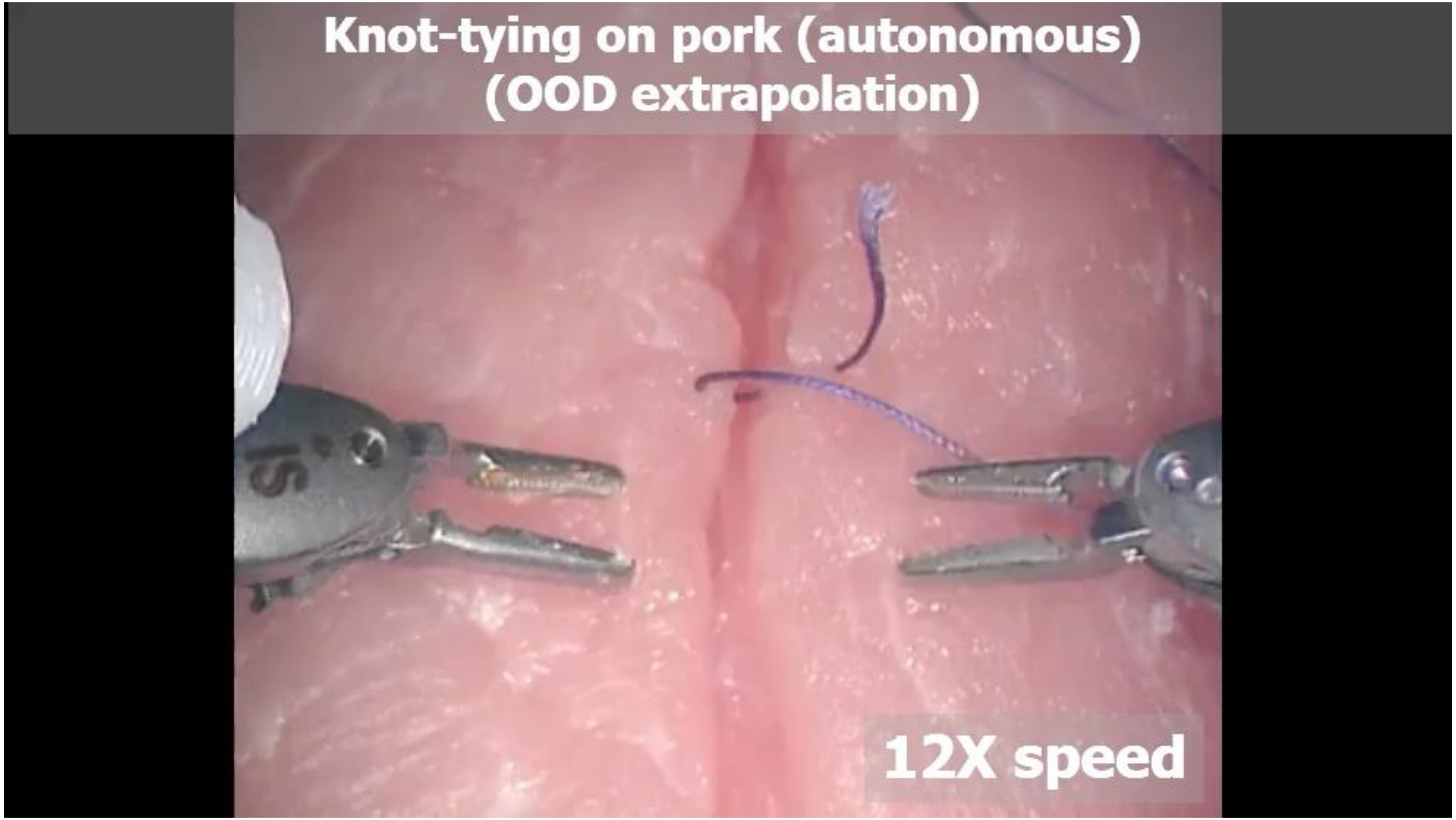
Sequential prediction problems such as imitation learning, where future observations depend on previous predictions (actions), violate the common i.i.d. assumptions made in statistical learning. This leads to poor performance in theory and often in practice. Some recent approaches (Daumé III et al., 2009; Ross and Bagnell, 2010) provide stronger guarantees in this setting, but remain somewhat unsatisfactory as they train either non-stationary or stochastic policies and require a large number of iterations. In this paper, we propose a new iterative algorithm, which trains a stationary deterministic policy, that can be seen as a no regret algorithm in an online learning setting. We show that any such no regret algorithm, combined with additional reduction assumptions, must find a policy with good performance under the distribution of observations it induces in such sequential settings. We demonstrate that this new approach outperforms previous approaches on two challenging imitation learning problems and a benchmark sequence labeling problem.

strations of good behavior are used to learn a controller, have proven very useful in practice and have led to state-of-the art performance in a variety of applications (Schaal, 1999; Abbeel and Ng, 2004; Ratliff et al., 2006; Silver et al., 2008; Argall et al., 2009; Chernova and Veloso, 2009; Ross and Bagnell, 2010). A typical approach to imitation learning is to train a classifier or regressor to predict an expert's behavior given training data of the encountered observations (input) and actions (output) performed by the expert. However since the learner's prediction affects future input observations/states during execution of the learned policy, this violate the crucial i.i.d. assumption made by most statistical learning approaches.

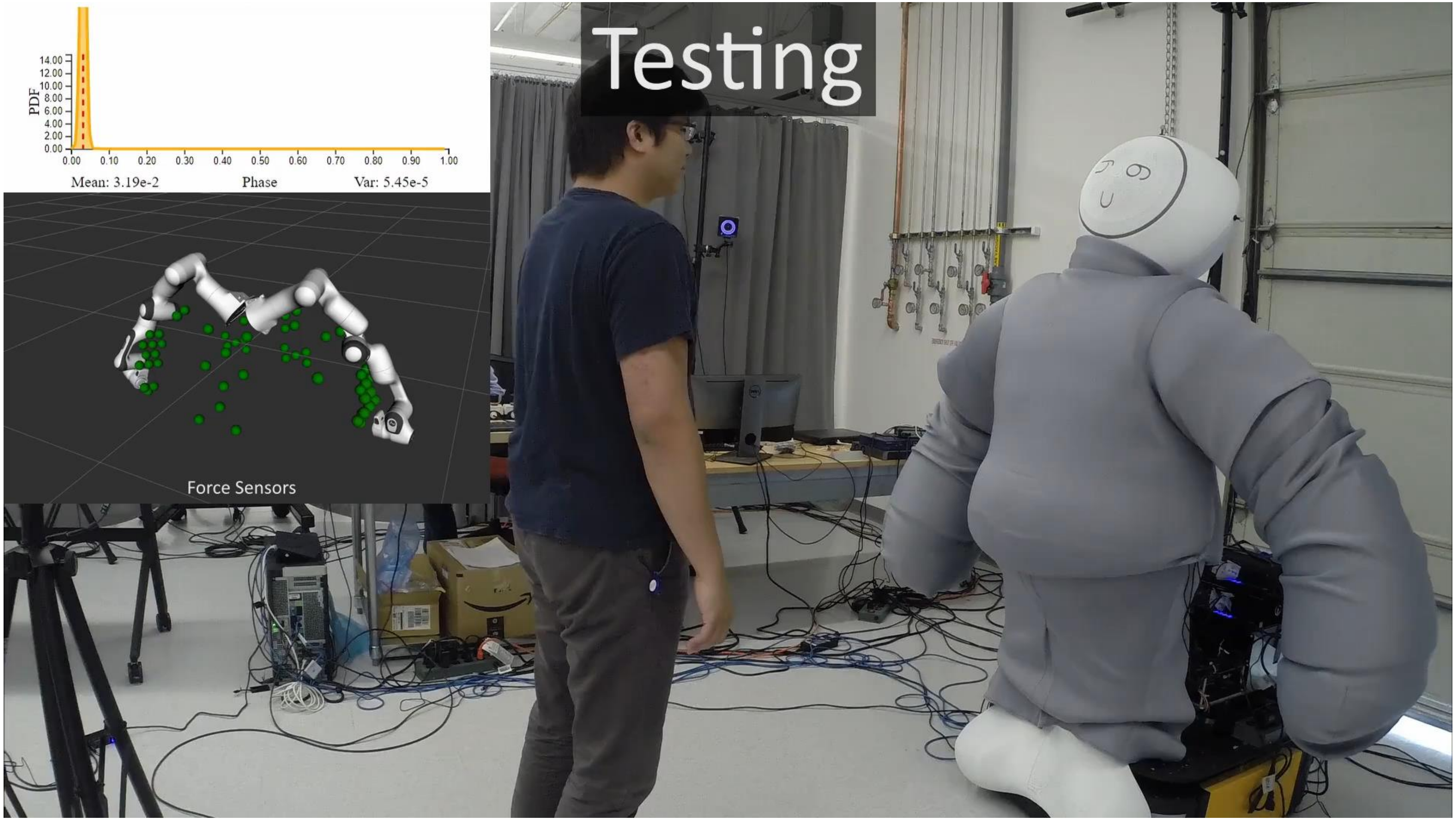
Ignoring this issue leads to poor performance both in theory and practice (Ross and Bagnell, 2010). In particular, a classifier that makes a mistake with probability ϵ under the distribution of states/observations encountered by the expert can make as many as $T^2\epsilon$ mistakes in expectation over T-steps under the distribution of states the classifier itself induces (Ross and Bagnell, 2010). Intuitively this is because as soon as the learner makes a mistake, it may encounter completely different observations than those under expert demonstration, leading to a compounding of errors.

Recent approaches (Ross and Bagnell, 2010) can guarantee

Other examples of imitation learning



Other examples of imitation learning



Take-aways

- Imitation learning is the simplest way to learn policies
- Behavior cloning is supervised learning with state-action pairs
- Data augmentation and DAGGER can be used to mitigate the i.i.d. assumption violation