Principles of Robot Autonomy II

Course overview and intro to
machine learning for robot autonomy
From Principles of Robot Autonomy I: the see-think-act cycle
Outstanding questions and new trends

• How do we build models for complex tasks? Can we use data / prior experience?

• How should the robot reason in terms of actively interacting with the environment?

• And how should the robot reason when interacting with other decision-making agents?

• Is the see-think-act cycle the only way to architect the autonomy stack? And how do I know if my autonomy stack is a good one?
Course goals

• Obtain a fundamental understanding of advanced principles of robot autonomy, including:
  1. robot learning
  2. physical interaction with the environment
  3. interaction with humans
  4. system architectures and V&V
Course structure

• Four modules, roughly of equal length
  1. learning-based control and perception
  2. interaction with the physical environment
  3. interaction with humans
  4. system architectures, verification & validation

• Requirements
  • AA 174A / AA 274A / CS 237A / EE 260A
  • CS 106A or equivalent
  • CME 100 or equivalent (for linear algebra)
  • CME 106 or equivalent (for probability theory)
Logistics

• Lectures:
  • Monday and Wednesday, 1:00pm - 2:20pm (Zoom)

• Sections
  • Weeks 2, 5, and 8
  • Goal: developing practical skills

• Office hours:
  • Dr. Bohg: Fridays, 1:00–2:00pm (Zoom), by appointment
  • Dr. Pavone: Fridays, 4:00–5:00pm (Zoom), by appointment
  • Dr. Sadigh: Fridays, 9:00–10:00am (Zoom), by appointment
  • CAs: Tuesdays, 4:00–6:00pm, and Fridays, 10:00am–12:00pm (Zoom)
Logistics

• Course websites:
  • https://cs237b.stanford.edu (course content and announcements)
  • https://piazza.com/stanford/winter2021/cs237b (course-related discussions)
  • https://www.gradescope.com/courses/217291 (HW submissions)
  • https://canvas.stanford.edu/courses/128328 (lecture videos)

• To contact the teaching staff, use the email: cs237b-win2021-staff@lists.stanford.edu
Grading and units

• Course grade calculation
  • (60%) homework
  • (40%) final exam (for each student, the lowest exam grade will be dropped)
  • (extra 5%) participation on Piazza

• Units: 3 or 4. Taking this class for 4 units entails additionally presenting a paper at the end of the quarter
Team

Instructors

Jeannette Bohg
Assistant Professor CS

Marco Pavone
Associate Professor AA, and CS/EE (by courtesy)

Dorsa Sadigh
Assistant Professor CS and EE

CAs

Erdem Bıyık

Abhishek Cauligi

1/11/21
<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/11</td>
<td>Course overview, intro to ML for robotics</td>
<td>HW1 out</td>
</tr>
<tr>
<td>01/13</td>
<td>Markov decision processes, intro to RL</td>
<td></td>
</tr>
<tr>
<td>01/15</td>
<td>Model-based and model-free RL for robot control</td>
<td></td>
</tr>
<tr>
<td>01/18</td>
<td>Martin Luther King, Jr., Day (no classes)</td>
<td></td>
</tr>
<tr>
<td>01/25</td>
<td>Learning-based perception</td>
<td></td>
</tr>
<tr>
<td>01/27</td>
<td>Fundamentals of grasping</td>
<td>HW2 due, HW3 out</td>
</tr>
<tr>
<td>01/29</td>
<td>HW1 due, HW2 out</td>
<td></td>
</tr>
<tr>
<td>02/01</td>
<td>Guest lecture</td>
<td>Exam 1</td>
</tr>
<tr>
<td>02/03</td>
<td>Grasp force optimization and planar pushing</td>
<td></td>
</tr>
<tr>
<td>02/05</td>
<td>Interactive perception</td>
<td></td>
</tr>
<tr>
<td>02/08</td>
<td>Learning-based grasping and manipulation</td>
<td></td>
</tr>
<tr>
<td>02/10</td>
<td>HW2 due, HW3 out</td>
<td></td>
</tr>
<tr>
<td>02/15</td>
<td>Presidents’ Day (no classes)</td>
<td>HW3 due, HW4 out</td>
</tr>
<tr>
<td>02/17</td>
<td>Imitation learning I</td>
<td></td>
</tr>
<tr>
<td>02/19</td>
<td>Imitation learning II</td>
<td>HW3 due, HW4 out</td>
</tr>
<tr>
<td>02/22</td>
<td>Imitation learning II</td>
<td></td>
</tr>
<tr>
<td>02/24</td>
<td>Learning from pairwise comparisons and physical feedback</td>
<td>Exam 2</td>
</tr>
<tr>
<td>02/26</td>
<td>HW4 due</td>
<td></td>
</tr>
<tr>
<td>03/01</td>
<td>Interaction-aware control, intent inference, and shared autonomy</td>
<td></td>
</tr>
<tr>
<td>03/03</td>
<td>System architectures</td>
<td>HW3 due, HW4 out</td>
</tr>
<tr>
<td>03/05</td>
<td>Specifications and model checking</td>
<td>Exam 3</td>
</tr>
<tr>
<td>03/10</td>
<td>Formal verification of neural networks</td>
<td></td>
</tr>
<tr>
<td>03/12</td>
<td>System-level verification via stress testing</td>
<td></td>
</tr>
<tr>
<td>03/15</td>
<td>Paper presentations</td>
<td>HW4 due</td>
</tr>
<tr>
<td>03/17</td>
<td>HW4 due</td>
<td></td>
</tr>
</tbody>
</table>
Intro to Machine Learning (ML)

• Aim
  • Present and motivate modern ML techniques

• Courses at Stanford
  • EE 104: Introduction to Machine Learning
  • CS 229: Machine Learning

• Reference
Machine learning

• Supervised learning (classification, regression)

  • Given \((x^1, y^1), \ldots, (x^n, y^n)\), choose a function \(f(x) = y\)

    \[x_i = \text{data point}\]
    \[y_i = \text{class/value}\]

• Unsupervised learning (clustering, dimensionality reduction)

  • Given \((x^1, x^2, \ldots, x^n)\) find patterns in the data
Supervised learning

- Regression

- Classification
Learning models

Parametric models

Non-parametric models

Linear regression

Linear classifier

Spline fitting

k-Nearest Neighbors
Loss functions

In selecting $f(x) \approx y$ we need a quality metric, i.e., a loss function to minimize

- **Regression**
  - $\ell^2$ loss: $\sum_i |f(x^i) - y^i|^2$
  - $\ell^1$ loss: $\sum_i |f(x^i) - y^i|$

- **Classification**
  - $0 - 1$ loss: $\sum_i 1\{f(x^i) \neq y^i\}$
  - Cross entropy loss: $-\sum_i (y^i)^T \log(f(x^i))$
Machine learning as optimization

How can we choose the best (loss minimizing) parameters to fit our training data?*

Analytical solution

\[
\begin{bmatrix}
y_1^1 & y_2^1 \\
y_1^2 & y_2^2 \\
\vdots \\
y_1^n & y_2^n \\
\end{bmatrix} \approx \begin{bmatrix}
x_1^1 & x_2^1 & \cdots & x_k^1 \\
x_1^2 & x_2^2 & \cdots & x_k^2 \\
\vdots & \vdots & \ddots & \vdots \\
x_1^n & x_2^n & \cdots & x_k^n \\
\end{bmatrix} \begin{bmatrix}
a_{11} & a_{12} \\
a_{11} & a_{12} \\
\vdots & \vdots \\
a_{k1} & a_{k2} \\
\end{bmatrix}
\]

\[
f_A(x) = xA, \quad \ell^2 \text{ loss}
\]

\[
\hat{A} = (X^TX)^{-1}X^TY
\]

Numerical optimization

(Example: linear least squares)

(Example: gradient descent)

* we’ll come back to worrying about test data
Stochastic optimization

Our loss function is defined over the entire training dataset:

\[ L = \frac{1}{n} \sum_{i=1}^{n} \left| f(x^i) - y^i \right|^2 = \frac{1}{n} \sum_{i=1}^{n} L_i \]

Computing \( \nabla L \) could be very computationally intensive. We approximate:

\[ \nabla L \approx \frac{1}{|S|} \sum_{i \in S \subset \{1, \ldots, n\}} \nabla L_i \]
Regularization

To avoid overfitting on the training data, we may add additional terms to the loss function to penalize “model complexity”

\[ \ell^2 \text{ regularization: } \|A\|_2 \]

often corresponds to a Gaussian prior on parameters \( A \)

\[ \ell^1 \text{ regularization: } \|A\|_1 \]

often encourages sparsity in \( A \) (easier to interpret/explain)

Hyperparameter regularization:
Linear classifiers

\[ f(x, W) = Wx + b \]

- \( f(x, W) \): 10x1
- \( W \): 3072x1
- \( x \): 3072x1
- \( b \): 10x1

\[ [32x32x3] \]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or “weights”
Linear classifiers

Example with an image with 4 pixels, and 3 classes (\textit{cat/dog/ship})

\begin{equation}
\begin{array}{c}
\text{input image} \\
\begin{array}{c|c}
0 & 0.25 \\
1.5 & 0.2 \\
0 & -0.3 \\
\hline
2 & 2 \\
\end{array}
\end{array}
\end{equation}

\begin{equation}
\begin{array}{c|c|c}
W & b & f(x_i; W, b) \\
\hline
0.2 & -0.5 & 56 \\
1.5 & 1.3 & 231 \\
0 & 0.1 & 24 \\
2.0 & 2.1 & 1.1 \\
0.0 & 0.0 & 3.2 \\
\hline
56 & 1.1 & -96.8 \\
231 & 3.2 & 437.9 \\
24 & -1.2 & 61.95 \\
\end{array}
\end{equation}

\begin{equation}
\text{stretch pixels into single column}
\end{equation}
Linear classifiers – Interpretation

Each row of $W$ can be thought of as a “template” for nearest neighbor classification.

$$f(x_i, W, b) = Wx_i + b$$

Example trained weights of a linear classifier trained on CIFAR-10:
Softmax regression

Our class scores can be turned into a probability vector over classes using the softmax function:

\[
\sigma(z) = \begin{bmatrix}
e^{z_1} \\
\sum_k e^{z_k}
\end{bmatrix} \\
\vdots \\
e^{z_m} \\
\sum_k e^{z_k}
\]

\[
p(y^i = j | x^i) = \frac{e^{x^i W_j + b_j}}{\sum_k e^{x^i W_k + b_k}}
\]
Generalizing linear models

Linear regression/classification can be very powerful when empowered by the right features.

Nonlinearity via basis functions

Eigenfaces
Feature extraction

Human Ingenuity

[32x32x3]

Gradient Descent

[32x32x3]

Feature Extraction

vector describing various image statistics

f

10 numbers, indicating class scores

training

f

10 numbers, indicating class scores

training
Bio people are apparently somewhat skeptical.

Just the math: \( y = f(xw + b) \) (with input as a row vector)
Single layer neural network

Original perceptron: binary inputs, binary output

\[ y_1^i = f(x^i w_1 + b_1) \]
\[ y_2^i = f(x^i w_2 + b_2) \]
\[ y_3^i = f(x^i w_3 + b_3) \]
\[ y_4^i = f(x^i w_4 + b_4) \]

\[ y = f(xW + b) \]
Multi-layer neural network

Also known as the Multilayer Perceptron (MLP)

Also known as the foundations of DEEP LEARNING

Like the brain, we’re connecting neurons to each other sequentially.

\[
h_1 = f_1(xW_1 + b_1) \\
h_2 = f_2(h_1W_2 + b_2) \\
y = f_3(h_2W_3 + b_3)
\]
Activation functions

Can’t go only linear:

\[ y = ((xW_1 + b_1)W_2 + b_2)W_3 + b_3? \]

\[ \implies y = xW_1W_2W_3 + (b_1W_2W_3 + b_2W_3 + b_3) \]

Sigmoid

\[ \sigma(x) = 1/(1 + e^{-x}) \]

Leaky ReLU

\[ \text{max}(0.1x, x) \]

Secret theme:
All of these functions are super easy to differentiate

tanh \quad \tanh(x)

ReLU \quad \text{max}(0,x)
Training neural networks

We want to use some variant of gradient descent.

How to compute gradients?

1. Sample a batch of data
2. Forward propagate it through the graph to compute the loss
3. Backpropagate to calculate the gradient of the loss with respect to the weights/biases
4. Update these parameters using SGD

The Chain Rule

$$\nabla (f \circ g)(x) = ((Dg)(x))^T (\nabla f)(g(x))$$

Leveraging the intermediate results of forward propagation with “easy” to differentiate activation functions

✓ Gradient is a bunch of matrix multiplication
Training neural networks

- **Training**: Large N
- **Inference**: Smaller, varied N

Diagram showing the process of training and inference in neural networks.
Training neural networks

Lots of regularization tricks:

Dropout: (randomly zero out some neurons each pass)

Transform input data to artificially expand training set:
Neural networks example

http://playground.tensorflow.org/
Next time