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, 1:30pm - 2:50pm (Zoom for the first two weeks, in person thereafter)

• Information about office hours available in the Syllabus: http://web.stanford.edu/class/cs237b/pdfs/syllabus.pdf

• Course websites:
  • https://cs237b.stanford.edu (course content and announcements)
  • https://canvas.stanford.edu/courses/147336 (course-related discussions)
  • https://www.gradescope.com/courses/339520 (HW submissions)
  • https://canvas.stanford.edu/courses/147336 (lecture videos)

• To contact the teaching staff, use the email: cs237b-w2122-staff@lists.stanford.edu
Grading and units

• Course grade calculation
  • (60%) homework
  • (40%) midterm exams (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**

Marion Lepert  
Head CA

Robert Dyro

Yilun Hao
# Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/03</td>
<td>Course overview, intro to ML for robotics</td>
<td></td>
</tr>
<tr>
<td>01/05</td>
<td>Markov decision processes</td>
<td>HW1 out</td>
</tr>
<tr>
<td>01/07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01/10</td>
<td>Model-based and model-free RL for robot control</td>
<td></td>
</tr>
<tr>
<td>01/12</td>
<td>Learning-based perception</td>
<td></td>
</tr>
<tr>
<td>01/17</td>
<td>Martin Luther King, Jr., Day (no classes)</td>
<td></td>
</tr>
<tr>
<td>01/19</td>
<td>Fundamentals of grasping</td>
<td></td>
</tr>
<tr>
<td>01/21</td>
<td></td>
<td>HW1 due, HW2 out</td>
</tr>
<tr>
<td>01/24</td>
<td>Grasp force optimization and planar pushing</td>
<td></td>
</tr>
<tr>
<td>01/26</td>
<td>Learning-based grasping and manipulation</td>
<td></td>
</tr>
<tr>
<td>01/28</td>
<td></td>
<td>Exam 1</td>
</tr>
<tr>
<td>01/31</td>
<td>Interactive perception</td>
<td></td>
</tr>
<tr>
<td>02/02</td>
<td>Guest Lecture by Karol Hausman</td>
<td></td>
</tr>
<tr>
<td>02/07</td>
<td>Imitation learning I</td>
<td></td>
</tr>
<tr>
<td>02/09</td>
<td>Imitation learning II</td>
<td></td>
</tr>
<tr>
<td>02/11</td>
<td></td>
<td>HW2 due, HW3 out</td>
</tr>
<tr>
<td>02/14</td>
<td>Learning from pairwise comparisons and physical feedback</td>
<td></td>
</tr>
<tr>
<td>02/16</td>
<td>Interaction-aware control, intent inference, and shared autonomy</td>
<td></td>
</tr>
<tr>
<td>02/18</td>
<td></td>
<td>Exam 2</td>
</tr>
</tbody>
</table>

02/21 Presidents’ Day (no classes)
02/23 System architectures
02/25 HW4 out
02/28 Specifications and model checking
03/02 System-level verification via stress testing
03/04 Exam 3, HW3 due
03/07 Guest Lecture by Anirudha Majumdar
03/09 Paper presentations
03/11 HW4 due
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

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

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

\[
\hat{A} = (X^T X)^{-1} X^T Y
\]

Numerical optimization

(example: linear least squares)

(\text{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:

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

\[ \text{KNN: } K=1 \]

\[ \text{KNN: } K=10 \]

\[ \text{KNN: } K=100 \]
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] → Feature Extraction vectors describing various image statistics → 10 numbers, indicating class scores

Gradient Descent

[32x32x3] → Feature Extraction f training 10 numbers, indicating class scores
Perceptron – analogy to a neuron

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

\[ y^i_1 = f(x^i w_1 + b_1) \]
\[ y^i_2 = f(x^i w_2 + b_2) \]
\[ y^i_3 = f(x^i w_3 + b_3) \]
\[ y^i_4 = f(x^i w_4 + b_4) \]
\[ y = f(xW + b) \]
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

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

Leaky ReLU

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

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

\[
\begin{align*}
\text{tanh} & \quad \text{tanh}(x) \\
\text{ReLU} & \quad \text{max}(0, x)
\end{align*}
\]
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 network to compute 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

\[ \Rightarrow \text{Gradient is a bunch of matrix multiplications} \]
Training neural networks

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

Diagram:
- Forward pass: "dog" 
- Backward pass: error
- Labels: "human face"
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