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?

• 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?

• 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?
Course goals

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

• Implement these concepts on real robot platforms
Course structure

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

• Extensive use of the Robot Operating System (ROS)

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

• Lectures:
  • Monday and Wednesday, 1:30pm -2:50pm (Gates B1)

• Sections
  • Schedule TBD
  • First half of the quarter: perfecting autonomy stack from Robot Autonomy I
  • Second half of the quarter: preparation for final project

• Office hours:
  • Dr. Bohg: Fridays, 1:00–2:00pm (Gates 140), after class, by appointment
  • Dr. Pavone: Tuesdays, 1:00–2:00pm (Durand 261), after class, by appointment
  • Dr. Sadigh: Fridays, 9:00–10:00am (Gates 142), after class, and by appointment
  • CAs: Tuesdays, 10:00am–12:00pm, and Fridays, 3:00–5:00pm, in Durand 023
Logistics

• Course websites:
  • http://cs237b.stanford.edu
  • http://piazza.com/stanford/winter2020/cs237b
  • http://www.gradescope.com/courses/77478
  • http://canvas.stanford.edu/courses/112347

• To contact the teaching staff, use the email: cs237b-win1920-staff@lists.stanford.edu
Grading

• Course grade calculation
  • (60%) homework
  • (20%) final exam
  • (20%) final project
  • (extra 5%) participation on Piazza
Team

Instructors

Jeannette Bogh
Assistant Professor CS

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

Dorsa Sadigh
Assistant Professor CS and EE

CAs

Erdem Bıyık

Jenna Lee

Toki Migimatsu

1/6/20
## Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/06</td>
<td>Course overview, intro to ML for robotics</td>
<td></td>
</tr>
<tr>
<td>01/08</td>
<td>Markov decision processes</td>
<td>HW1 out (on Friday)</td>
</tr>
<tr>
<td>01/13</td>
<td>Reinforcement learning for robot control</td>
<td></td>
</tr>
<tr>
<td>01/15</td>
<td>Learning-based perception</td>
<td></td>
</tr>
<tr>
<td>01/20</td>
<td>Martin Luther King, Jr., Day (no classes)</td>
<td></td>
</tr>
<tr>
<td>01/22</td>
<td>System architectures</td>
<td>HW1 due, HW2 out</td>
</tr>
<tr>
<td>01/27</td>
<td>Specifications and model checking</td>
<td></td>
</tr>
<tr>
<td>01/29</td>
<td>Formal verification of neural networks</td>
<td></td>
</tr>
<tr>
<td>02/03</td>
<td>System-level verification via stress testing</td>
<td></td>
</tr>
<tr>
<td>02/05</td>
<td>Fundamentals of grasping</td>
<td>HW2 due, HW3 out</td>
</tr>
<tr>
<td>02/10</td>
<td>Feedback/ feedforward control for contacts</td>
<td></td>
</tr>
<tr>
<td>02/12</td>
<td>Planning through contact</td>
<td></td>
</tr>
<tr>
<td>02/17</td>
<td>Presidents’ Day (no classes)</td>
<td></td>
</tr>
<tr>
<td>02/19</td>
<td>Interactive perception</td>
<td>Final project released</td>
</tr>
<tr>
<td>02/24</td>
<td>Foundations of imitation learning</td>
<td>HW3 due, HW4 out</td>
</tr>
<tr>
<td>02/26</td>
<td>Intent inference</td>
<td></td>
</tr>
<tr>
<td>03/02</td>
<td>Planning in the worst case</td>
<td></td>
</tr>
<tr>
<td>03/04</td>
<td>Planning in the stochastic case</td>
<td>Final project check-in</td>
</tr>
<tr>
<td>03/09</td>
<td>Final exam (in class)</td>
<td>HW4 due</td>
</tr>
<tr>
<td>03/11</td>
<td>Conclusions</td>
<td></td>
</tr>
<tr>
<td>TBD</td>
<td>Final Project Demo</td>
<td></td>
</tr>
</tbody>
</table>

1/6/20
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

Linear regression

Non-parametric models

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**

  \[ \ell^2 \text{ loss : } \sum |f(x^i) - y^i|^2 \]

  \[ \ell^1 \text{ loss : } \sum |f(x^i) - y^i| \]

• **Classification**

  \[ 0 - 1 \text{ loss : } \sum_1 \{ f(x^i) \neq y^i \} \]

  Cross entropy loss : \[ -\sum (y^i)^T \log(f(x^i)) \]
Machine learning as optimization

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

Analytical solution

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

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

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

(example: linear least squares)

Numerical optimization

(\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:

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

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

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

10 numbers, indicating class scores

parameters, or “weights”

[32x32x3] array of numbers 0...1
Linear classifiers

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)
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:

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

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

Image: [32x32x3] → Feature Extraction → [32x32x3]

Gradient Descent

Image: [32x32x3] → Feature Extraction → [32x32x3]

Vector describing various image statistics

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

\[
\begin{align*}
    y_1^i &= f(x^i w_1 + b_1) \\
    y_2^i &= f(x^i w_2 + b_2) \\
    y_3^i &= f(x^i w_3 + b_3) \\
    y_4^i &= f(x^i w_4 + b_4)
\end{align*}
\]

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

Also known as the Multilayer Perceptron (MLP)
Also known as the foundations of **DEEP LEARNING**

$$h_1 = f_1(xW_1 + b_1)$$
$$h_2 = f_2(h_1W_2 + b_2)$$
$$y = f_3(h_2W_3 + b_3)$$

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

tanh \( \tanh(x) \)

ReLU \( \text{max}(0, x) \)
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

$\Rightarrow$ Gradient is a bunch of matrix multiplication
Training neural networks

Training

Inference

Large N

Smaller, varied N

forward

backward

“dog”

=?

labels

“human face”

error
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