AD for CV, AC, ML, AI, and CL

Jeffrey Mark Siskind

Purdue University
School of Electrical and Computer Engineering

Friday 23 November 2012

Andrei Barbu
Seongwoon Ko
Haonan Yu

Daniel Barrett
Aaron Michaux

Tommy Chang
Zachary Burchill
Pranay Gupta
Siddharth Narayanaswamy

This work was supported, in part, by NSF grant CCF-0438806, by the Naval Research Laboratory under Contract Number N00173-10-1-G023, by the Army Research Laboratory accomplished under Cooperative Agreement Number W911NF-10-2-0060, and by computational resources provided by Information Technology at Purdue through its Rosen Center for Advanced Computing. Any views, opinions, findings, conclusions, or recommendations contained or expressed in this document or material are those of the author(s) and do not necessarily reflect or represent the views or official policies, either expressed or implied, of NSF, the Naval Research Laboratory, the Office of Naval Research, the Army Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.
Outline

1. AD
2. AD for CV
3. AD for AC
4. AD for ML
5. AD for AI
6. AD for CL
Automatic Differentiation (AD)

Wengert (1964), Speelpenning (1980)

\[ d \frac{df}{dx} \] (derivative of \( f \) with respect to \( x \))

\[ \frac{\partial f}{\partial x} \] (derivative of \( \lambda (x) (f x y) \) with respect to \( x \) at \( i \))

\[ \nabla_x f(x) \] (list of derivatives \( \lambda (x) (f x y) \) and \( \lambda (y) (f x y) \))

\[ \arg\min_x f(x) \] (argmin \( f \) at \( x_0 \))

\[ \arg\min_x f(x, p) \] (argmin \( f \) at \( x_0 \))
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx}
\]
Automatic Differentiation (AD)

Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative } f) \]

\[
\nabla x f(x) \quad \text{(list (derivative (lambda (x) (f x y))) (derivative (lambda (y) (f x y))))} \]

\[
\text{argmin}_x f(x) \quad \text{(argmin } f x^0) \]

\[
\text{argmin}_x f(x, p) \quad \text{(argmin (lambda (x) (f x p)) x^0)} \]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[ \frac{df(x)}{dx} \quad \text{(derivative of } f) \]

\[ \frac{\partial f(x)}{\partial x_i} \]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative } f \text{)}
\]

\[
\frac{\partial f(x)}{\partial x_i} \quad \text{(derivative } (\lambda (x) (f x y)) \text{)}
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative of f)}
\]

\[
\frac{\partial f(x)}{\partial x_i} \quad \text{(derivative of (lambda (x) (f x y)))}
\]

\[
\nabla_x f(x)
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative f)}
\]

\[
\frac{\partial f(x)}{\partial x_i} \quad \text{(derivative (lambda (x) (f x y)))}
\]

\[
\nabla_x f(x) \quad \text{(list (derivative (lambda (x) (f x y)))
(derivative (lambda (y) (f x y))))}
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative f)}
\]

\[
\frac{\partial f(x)}{\partial x_i} \quad \text{(derivative (\lambda (x) (f x y)))}
\]

\[
\nabla_x f(x) \quad \text{(list (derivative (\lambda (x) (f x y)))}
\]
\[\text{(derivative (\lambda (y) (f x y))))}
\]

\[
\text{argmin}_x f(x)
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad (\text{derivative } f)
\]

\[
\frac{\partial f(x)}{\partial x_i} \quad (\text{derivative } (\lambda (x) (f x y)))
\]

\[
\nabla_x f(x) \quad (\text{list } (\text{derivative } (\lambda (x) (f x y)))
\]
\[
(\text{derivative } (\lambda (y) (f x y))))
\]

\[
\argmin_x f(x) \quad (\text{argmin } f x0)
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[
\frac{df(x)}{dx} \quad \text{(derivative } f) \\
\frac{\partial f(x)}{\partial x_i} \quad \text{(derivative } (\text{lambda } (x) (f \times y))) \\
\nabla_x f(x) \quad \text{(list } \text{(derivative } (\text{lambda } (x) (f \times y)))) \\
\quad \text{(derivative } (\text{lambda } (y) (f \times y)))) \\
\text{argmin}_x f(x) \quad \text{(argmin } f \times 0) \\
\text{argmin}_x f(x, p)
\]
Automatic Differentiation (AD)
Wengert (1964), Speelpenning (1980)

\[ \frac{df(x)}{dx} \]  \hspace{1cm} \text{(derivative } f) \\
\[ \frac{\partial f(x)}{\partial x_i} \]  \hspace{1cm} \text{(derivative } \lambda(x) (f x y)) \)

\[ \nabla_x f(x) \]  \hspace{1cm} \text{(list } (\text{derivative } \lambda(x) (f x y)) \text{)}\\
\hspace{1cm} (\text{derivative } \lambda(y) (f x y)))

\[ \arg\min_x f(x) \]  \hspace{1cm} \text{(argmin } f x0) \\
\[ \arg\min_x f(x, p) \]  \hspace{1cm} \text{(argmin } \lambda(x) (f x p) \text{)} x0)
Outline

1. AD
2. AD for CV
3. AD for AC
4. AD for ML
5. AD for AI
6. AD for CL
# Demo of Inverting Motor Programs

<table>
<thead>
<tr>
<th>Help</th>
<th>Abort</th>
<th>Park Arm</th>
<th>Reset Arm</th>
<th>Write PPM</th>
<th>Play</th>
<th>Calibrate</th>
<th>Quit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track</td>
<td>Track&amp;View</td>
<td>Viewfinder</td>
<td>Overlay</td>
<td>Beginning</td>
<td>-T</td>
<td>+T</td>
<td>End</td>
</tr>
<tr>
<td>Above Red</td>
<td>Above Green</td>
<td>Above Blue</td>
<td>Above Yellow</td>
<td>Move Left</td>
<td>Move Right</td>
<td>Move Up</td>
<td>Move Down</td>
</tr>
<tr>
<td>Pick Up Green</td>
<td>Put Down</td>
<td>Put On Yellow</td>
<td>Push Green</td>
<td>-Colour</td>
<td>+Colour</td>
<td>-Whiteness</td>
<td>+Whiteness</td>
</tr>
<tr>
<td>-Brightness</td>
<td>+Brightness</td>
<td>-Contrast</td>
<td>+Contrast</td>
<td>-Colour</td>
<td>+Colour</td>
<td>-Whiteness</td>
<td>+Whiteness</td>
</tr>
<tr>
<td>Set Red</td>
<td>Set Green</td>
<td>Set Blue</td>
<td>Set Yellow</td>
<td>Record</td>
<td>Snapshot</td>
<td>Initial</td>
<td>Best</td>
</tr>
<tr>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Yellow</td>
<td>Load RGBY</td>
<td>Save RGBY</td>
<td>Load Pose</td>
<td>Save Pose</td>
</tr>
<tr>
<td>-M 442</td>
<td>-CPM 20</td>
<td>-CP 4</td>
<td>-MPM 200</td>
<td>-MP 8</td>
<td>+Colour</td>
<td>-Whiteness</td>
<td>+Whiteness</td>
</tr>
<tr>
<td>+M 442</td>
<td>+CPM 20</td>
<td>+CP 4</td>
<td>+MPM 200</td>
<td>+MP 8</td>
<td>+Colour</td>
<td>-Whiteness</td>
<td>+Whiteness</td>
</tr>
<tr>
<td>Home</td>
<td>Left</td>
<td>Right</td>
<td>Up</td>
<td>Down</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Robot Images]

- T=10, P=-50
  - B=32768, C=32768, U=32768
- T=10, P=-105
  - B=32768, C=32768, U=32768
- T=10, P=-72.5
  - B=32768, C=32768, U=32768
- T=10, P=-115
  - B=32768, C=32768, U=32768
Demo of Inverting Motor Programs

<table>
<thead>
<tr>
<th>Help</th>
<th>Abort</th>
<th>Park Arm</th>
<th>Reset Arm</th>
<th>Write PPM</th>
<th>Play</th>
<th>Calibrate</th>
<th>Quit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track</td>
<td>Track&amp;View</td>
<td>Viewfinder</td>
<td>Overlay</td>
<td>Beginning</td>
<td>-T</td>
<td>+T</td>
<td>End</td>
</tr>
<tr>
<td>Above Red</td>
<td>Above Green</td>
<td>Above Blue</td>
<td>Above Yellow</td>
<td>Move Left</td>
<td>Move Right</td>
<td>Move Up</td>
<td>Move Down</td>
</tr>
<tr>
<td>Pick Up Green</td>
<td>Put Down</td>
<td>Put On Yellow</td>
<td>Push Green</td>
<td>-Brightness</td>
<td>+Brightness</td>
<td>-Contrast</td>
<td>+Contrast</td>
</tr>
<tr>
<td>Set Red</td>
<td>Set Green</td>
<td>Set Blue</td>
<td>Set Yellow</td>
<td>Record</td>
<td>+Colour</td>
<td>-Whiteness</td>
<td>+Whiteness</td>
</tr>
<tr>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
<td>Yellow</td>
<td>Load RGBY</td>
<td>Save RGBY</td>
<td>Load Pose</td>
<td>Save Pose</td>
</tr>
<tr>
<td>-M 442</td>
<td>-CPM 20</td>
<td>-CP 4</td>
<td>-MPM 200</td>
<td>-MP 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+M 442</td>
<td>+CPM 20</td>
<td>+CP 4</td>
<td>+MPM 200</td>
<td>+MP 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>Left</td>
<td>Right</td>
<td>Up</td>
<td>Down</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

T=10, P=50
B=32768, C=32768, C=32768, W=32768

T=10, P=105
B=32768, C=32768, C=32768, W=32768

T=10, P=72.5
B=32768, C=32768, C=32768, W=32768

T=10, P=115
B=32768, C=32768, C=32768, W=32768
How It Works

- world coordinates of executor's end effector
- arm configuration
- robot pose
- camera poses
- image coordinates of executor's end effector
- forward kinematics
- forward optics
- inverse optics
- binocular vision
- not stereo
- robustness in the face of occlusion
- visual servoing along all axes
- motor program
- inverse kinematics
- closed-loop visual servoing
- imagination capacity
- argmin
- classification
- pose estimation by gradient descent
- calculated using AD

Siskind (Purdue)
z

world coordinates of executor’s end effector
How It Works

- \( z \) world coordinates of executor’s end effector
- \( a \) arm configuration
How It Works

- $z$: world coordinates of executor’s end effector
- $a$: arm configuration
- $b$: robot pose

$$c = (c_{el}, c_{er}), \quad x = (x_{el}, x_{er})$$

Camera poses

Image coordinates of executor’s end effector

$$z = f(a, b)$$

Forward kinematics

Forward optics

$$x = p(z, c)$$

Inverse optics

Binocular vision

Not stereo

Robustness in the face of occlusion

Visual servoing along all axes

$$a = m(x_e)$$

Motor program

Inverse kinematics

Closed-loop visual servoing

$$\hat{x}_o = p(f(m(p(p(z, c), c), b), c))$$

Imagination capacity

$$\arg\min_m \|x_o - \text{lsh}(\hat{x}_o)\|$$

Classification

$$\arg\min_m \min_b, c_o, c_e \|x_o - \text{lsh}(\hat{x}_o)\|$$

Pose estimation by gradient descent

$$\nabla b, c_o, c_e \|x_o - \text{lsh}(\hat{x}_o)\|$$

Calculated using AD
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), \quad c_o = (c_{ol}, c_{or}) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
How It Works

\[ z \]

\[ a \]

\[ b \]

\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]

\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector
How It Works

$z$

$a$

$b$

$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$

$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$

$z = f(a, b)$

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector

forward kinematics
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
How It Works

- $z$: world coordinates of executor’s end effector
- $a$: arm configuration
- $b$: robot pose
- $c_e = (c_{el}, c_{er})$, $c_o = (c_{ol}, c_{or})$: camera poses
- $x_e = (x_{el}, x_{er})$, $x_o = (x_{ol}, x_{or})$: image coordinates of executor’s end effector
- $z = f(a, b)$: forward kinematics
- $x = p(z, c)$: forward optics
- $z = p^{-1}(x_l, x_r, c_l, c_r)$: inverse optics

Siskind (Purdue)
How It Works

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision

z
a
b
c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})
x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})
z = f(a, b)
x = p(z, c)
z = p^{-1}(x_l, x_r, c_l, c_r)
How It Works

\( z \)
\( a \)
\( b \)
\( c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \)
\( x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \)
\( z = f(a, b) \)
\( x = p(z, c) \)
\( z = p^{-1}(x_l, x_r, c_l, c_r) \)

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
How It Works

$z$

$\alpha$

$\beta$

$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$

$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$

$z = f(a, b)$

$x = p(z, c)$

$z = p^{-1}(x_l, x_r, c_l, c_r)$

$\alpha = m(x_e)$

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics

\[ a = m(x_e) \]
How It Works

- $z$:
  - world coordinates of executor’s end effector
- $a$:
  - arm configuration
- $b$:
  - robot pose
- $c_e = (c_{el}, c_{er})$, $c_o = (c_{ol}, c_{or})$:
  - camera poses
- $x_e = (x_{el}, x_{er})$, $x_o = (x_{ol}, x_{or})$:
  - image coordinates of executor’s end effector
- $z = f(a, b)$:
  - forward kinematics
- $x = p(z, c)$:
  - forward optics
- $z = p^{-1}(x_l, x_r, c_l, c_r)$:
  - inverse optics
- $a = m(x_e)$:
  - binocular vision
  - not stereo
  - robustness in the face of occlusion
  - visual servoing along all axes
  - motor program
  - inverse kinematics
  - closed-loop visual servoing
How It Works

- **World coordinates of executor’s end effector**
- **Arm configuration**
- **Robot pose**
- **Camera poses**
- **Image coordinates of executor’s end effector**
- **Forward kinematics**
- **Forward optics**
- **Inverse optics**
- **Binocular vision**
- **Robustness in the face of occlusion**
- **Visual servoing along all axes**
- **Motor program**
- **Inverse kinematics**
- **Closed-loop visual servoing**

Mathematical Formulas:

\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

\[ a = m(x_e) \]
How It Works

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics
closed-loop visual servoing

\[
z = f(a, b)
\]

\[
x = p(z, c)
\]

\[
z = p^{-1}(x_l, x_r, c_l, c_r)
\]

\[
a = m(x_e)
\]

\[
p^{-1}(x_o, c_o)
\]
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

\[ a = m(x_e) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics
closed-loop visual servoing
How It Works

$z$

$\mathbf{a}$

$\mathbf{b}$

$\mathbf{c}_e = (c_{el}, c_{er})$, $\mathbf{c}_o = (c_{ol}, c_{or})$

$x_e = (x_{el}, x_{er})$, $x_o = (x_{ol}, x_{or})$

$z = f(a, b)$

$x = p(z, c)$

$z = p^{-1}(x_l, x_r, c_l, c_r)$

$\mathbf{a} = m(x_e)$

$\mathbf{m}(p(p^{-1}(x_o, c_o), c_e))$

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector

forward kinematics

forward optics

inverse optics

binocular vision

not stereo

robustness in the face of occlusion

visual servoing along all axes

motor program

inverse kinematics

closed-loop visual servoing
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), \quad c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), \quad x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

\[ a = m(x_e) \]

\[ f(m(p(p^{-1}(x_o, c_o), c_e)), b) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics
closed-loop visual servoing
How It Works

\[ z \]
\[ a \]
\[ b \]
\[ c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or}) \]
\[ x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or}) \]
\[ z = f(a, b) \]
\[ x = p(z, c) \]
\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics
closed-loop visual servoing
imagination capacity

\[ a = m(x_e) \]

\[ \hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o) \]
How It Works

\( z \)

\( a \)

\( b \)

\[ c_e = (c_{el}, c_{er}), \quad c_o = (c_{ol}, c_{or}) \]

\[ x_e = (x_{el}, x_{er}), \quad x_o = (x_{ol}, x_{or}) \]

\[ z = f(a, b) \]

\[ x = p(z, c) \]

\[ z = p^{-1}(x_l, x_r, c_l, c_r) \]

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector

forward kinematics

forward optics

inverse optics

binocular vision

not stereo

robustness in the face of occlusion

visual servoing along all axes

motor program

inverse kinematics

closed-loop visual servoing

imagination capacity

classification

\( a = m(x_e) \)

\[ \hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o) \]

\[ \arg\min_m \|x_o - \text{lsh}(\hat{x}_o)\| \]
How It Works

$z$

$\mathbf{a}$

$\mathbf{b}$

$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$

$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$

$z = f(\mathbf{a}, \mathbf{b})$

$x = p(z, \mathbf{c})$

$z = p^{-1}(x_l, x_r, c_l, c_r)$

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector

forward kinematics

forward optics

inverse optics

binocular vision

not stereo

robustness in the face of occlusion

visual servoing along all axes

motor program

inverse kinematics

closed-loop visual servoing

imagination capacity

classification

$\mathbf{a} = m(x_e)$

$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), \mathbf{b}), c_o)$

$\text{argmin}_m \| x_o - \text{lsh}(\hat{x}_o) \|_2$
How It Works

$z$

$\mathbf{a}$

$\mathbf{b}$

$c_e = (c_{el} , c_{er})$, $c_o = (c_{ol} , c_{or})$

$x_e = (x_{el} , x_{er})$, $x_o = (x_{ol} , x_{or})$

$z = f(a, b)$

$x = p(z, \mathbf{c})$

$z = p^{-1}(x_l , x_r , c_l , c_r)$

\[a = m(x_e)\]

$\hat{x}_o = p(f(m(p(p^{-1}(x_o , c_o) , c_e)) , b) , c_o)$

$\arg\min_{m} \|x_o - lsh(\hat{x}_o)\|$

$\arg\min_{m} \min_{b, c_o , c_e} \|x_o - lsh(\hat{x}_o)\|$

world coordinates of executor’s end effector

arm configuration

robot pose

camera poses

image coordinates of executor’s end effector

forward kinematics

forward optics

inverse optics

binocular vision

not stereo

robustness in the face of occlusion

visual servoing along all axes

motor program

inverse kinematics

closed-loop visual servoing

imagination capacity

classification

pose estimation by gradient descent
How It Works

$z$
a
$b$
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$
$z = f(a, b)$
$x = p(z, c)$
$z = p^{-1}(x_l, x_r, c_l, c_r)$

$z = f(a, b)$

$z = p^{-1}(x_l, x_r, c_l, c_r)$

$z = f(a, b)$

$z = p^{-1}(x_l, x_r, c_l, c_r)$

world coordinates of executor’s end effector
arm configuration
robot pose
camera poses
image coordinates of executor’s end effector
forward kinematics
forward optics
inverse optics
binocular vision
not stereo
robustness in the face of occlusion
visual servoing along all axes
motor program
inverse kinematics
closed-loop visual servoing
imagination capacity
classification
pose estimation by gradient descent
calculated using AD
How It Works

Forward kinematics

\[ x = f(\theta, p) \]
How It Works

Forward kinematics

\[ \mathbf{x} = f(\theta, \mathbf{p}) \]

Estimating the parameters of the kinematic chain

\[ \mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_{i} \| \mathbf{x}_i - f(\theta_i, \mathbf{p}) \|^2 \]
How It Works

Forward kinematics

\[ \mathbf{x} = f(\theta, \mathbf{p}) \]

Estimating the parameters of the kinematic chain

\[ \mathbf{p}^* = \arg\min_{\mathbf{p}} \sum_i \| \mathbf{x}_i - f(\theta_i, \mathbf{p}) \|^2 \]

Inverse kinematics

\[ \theta = f^{-1}(\mathbf{x}, \mathbf{p}) \]

\[ = \arg\min_{\theta} \| \mathbf{x} - f(\theta, \mathbf{p}) \|^2 \]
Outline

1. AD
2. AD for CV
3. AD for AC
4. AD for ML
5. AD for AI
6. AD for CL
### Game Theory

<table>
<thead>
<tr>
<th></th>
<th>( b_1 )</th>
<th>( \ldots )</th>
<th>( b_j )</th>
<th>( \ldots )</th>
<th>( b_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( a_i )</td>
<td>( \ldots )</td>
<td>( \text{PAYOFF}(a_i, b_j) )</td>
<td>( \ldots )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( a_m )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td></td>
</tr>
</tbody>
</table>

Game Theory

<table>
<thead>
<tr>
<th></th>
<th>$b_1$</th>
<th>$\ldots$</th>
<th>$b_j$</th>
<th>$\ldots$</th>
<th>$b_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
<td></td>
<td>$\ddots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$A$ $a_i$</td>
<td>$\ldots$</td>
<td>$\text{PAYOFF}(a_i, b_j)$</td>
<td>$\ldots$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td></td>
<td></td>
<td></td>
<td>$\ddots$</td>
<td></td>
</tr>
<tr>
<td>$a_m$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\max_{a \in A} \min_{b \in B} \text{PAYOFF}(a, b)$$

<table>
<thead>
<tr>
<th>$\mathbb{R}^m$</th>
<th>$\mathbb{R}^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$$\max_{a \in \mathbb{R}^m} \min_{b \in \mathbb{R}^n} \text{PAYOFF}(a, b)$$

subroutine deriv2(f, x, y, yprime)
    external f
    adf(tangent(x) = 1.0)
    y = f(x)
    end adf(yprime = tangent(y))
end

function root(f, x0, n)
x = x0
do 1669 i = 1, n
    call deriv2(f, x, y, yprime)
1669 x = x-y/yprime
    root = x
end

function deriv1(f, x)
    external f
    adf(tangent(x) = 1.0)
    y = f(x)
    end adf(deriv1 = tangent(y))
end
Continuous-Strategy Two-Person Nonzero-Sum Game

VLAD/STALINGRAD

(define (deriv2 f x) (j* f (times 0 f) x 1))

(define (root f x n)
  (if (zero? n)
      x
      (let (((cons y yprime) (deriv2 f x))
             (root f (- x (/ y yprime)) (- n 1))))))

(define (deriv1 f x)
  (let (((cons y y-tangent) (j* f (times 0 f) x 1))) y-tangent))
function argmax(f, x0, n)
    function fprime(x)
        fprime = deriv1(f, x)
    end
    argmax = root(fprime, x0, n)
end

subroutine eqlbrm(biga, bigb, astar, bstar, n)
    external biga, bigb
    function f(astar)
        function g(a)
            function h(b)
                h = bigb(astar, b)
            end
        end
    end
    bstar = argmax(h, bstar, n)
    g = biga(a, bstar)
    f = argmax(g, astar, n)-astar
end
    astar = root(f, astar, n)
end
(define (argmax f x0 n)
  (define (fprime x) (deriv1 f x))
  (root fprime x0 n))

(define (eqlbrm biga bigb astar bstar n)
  (define (f astar)
    (define (g a)
      (define (h b) (bigb astar b))
      (biga a (argmax h bstar n))
      (- (argmax g astar n) astar))
    (let ((astar (root f astar n)))
      (define (h b) (bigb astar b))
      (let ((bstar (argmax h bstar n)))
        (cons astar bstar)))))
function gmbiga(a, b)
price = 20-0.1*a-0.1*b
costs = a*(10-0.05*a)
gmbiga = a*price-costs
end

function gmbigb(a, b)
price = 20-0.1*b-0.0999*a
costs = b*(10.005-0.05*b)
gmbigb = b*price-costs
end

program main
read *, astar
read *, bstar
read *, n
call eqlbrm(gmbiga, gmbigb, astar, bstar, n)
print *, astar, bstar
end
(define (gmbiga a b)
  (let ((price (- (- 20 (* 0.1 a)) (* 0.1 b)))
        (costs (* a (- 10 (* 0.05 a)))))
    (- (* a price) costs))

(define (gmbigb a b)
  (let ((price (- (- 20 (* 0.1 b)) (* 0.0999 a)))
        (costs (* b (- 10.005 (* 0.05 b)))))
    (- (* b price) costs))

(let ((cons astar bstar) (eqlbrm gmbiga gmbigb 0 0 (real 500)))
  (cons (write-real astar) (write-real bstar)))
Performance Comparison

CPU Time (ms)

- Tapenade: 7868 ms
- Adifor: 11468 ms
- Stalingrad: 2456 ms
potential: \( p(x; w) = ||x - (10, 10 - w)||^{-1} + ||x - (10, 0)||^{-1} \)

\[ \begin{align*}
\dot{x}(t) &= -\nabla_x p(x)|_{x=x(t)} \\
\dot{x}(t + \Delta t) &= \dot{x}(t) + \Delta t \ddot{x}(t) \\
x(t + \Delta t) &= x(t) + \Delta t \dot{x}(t)
\end{align*} \]

When: \( x_1(t + \Delta t) \leq 0 \)

let: \( \Delta t_f = -x_1(t)/\dot{x}_1(t) \)

\( t_f = t + \Delta t_f \)

\( x(t_f) = x(t) + \Delta t_f \dot{x}(t) \)

Error: \( E(w) = x_0(t_f)^2 \)

Find: \( \arg\min_w E(w) \)


## Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>particle</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>saddle</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FF</td>
<td>FR</td>
<td>RF</td>
<td>RR</td>
<td>FF</td>
<td>FR</td>
<td>RF</td>
<td>RR</td>
<td></td>
</tr>
<tr>
<td>VLAD</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>STALINGRAD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORTRAN</td>
<td>ADIFOR</td>
<td>1.52</td>
<td></td>
<td></td>
<td></td>
<td>2.07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAPENADE</td>
<td>3.40</td>
<td></td>
<td></td>
<td></td>
<td>2.56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>FADBAD++</td>
<td>65.69</td>
<td></td>
<td></td>
<td></td>
<td>22.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>MLTON</td>
<td>53.89</td>
<td>88.88</td>
<td>16.08</td>
<td>28.06</td>
<td>40.39</td>
<td>51.21</td>
<td>1.86</td>
<td>2.67</td>
</tr>
<tr>
<td></td>
<td>OCAML</td>
<td>160.50</td>
<td>340.35</td>
<td>147.91</td>
<td>263.66</td>
<td>107.71</td>
<td>156.33</td>
<td>6.75</td>
<td>13.51</td>
</tr>
<tr>
<td></td>
<td>SML/NJ</td>
<td>106.21</td>
<td>182.45</td>
<td>105.04</td>
<td>185.15</td>
<td>84.38</td>
<td>106.01</td>
<td>3.55</td>
<td>6.31</td>
</tr>
<tr>
<td>HASKELL</td>
<td>GHC</td>
<td>165.22</td>
<td></td>
<td></td>
<td></td>
<td>121.18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheme</td>
<td>BIGLOO</td>
<td>505.90</td>
<td>761.40</td>
<td>104.81</td>
<td>228.56</td>
<td>423.69</td>
<td>440.25</td>
<td>15.77</td>
<td>24.59</td>
</tr>
<tr>
<td></td>
<td>CHICKEN</td>
<td>1120.37</td>
<td>2026.31</td>
<td>425.60</td>
<td>1872.85</td>
<td>889.58</td>
<td>1144.65</td>
<td>35.73</td>
<td>68.94</td>
</tr>
<tr>
<td></td>
<td>GAMBIT</td>
<td>444.13</td>
<td>752.63</td>
<td>138.34</td>
<td>256.30</td>
<td>362.65</td>
<td>420.48</td>
<td>14.08</td>
<td>23.87</td>
</tr>
<tr>
<td></td>
<td>IKARUS</td>
<td>192.07</td>
<td>312.28</td>
<td>61.79</td>
<td>114.87</td>
<td>158.88</td>
<td>205.97</td>
<td>6.75</td>
<td>11.40</td>
</tr>
<tr>
<td></td>
<td>LARCENY</td>
<td>726.59</td>
<td>1108.18</td>
<td>144.55</td>
<td>270.14</td>
<td>571.81</td>
<td>613.65</td>
<td>19.14</td>
<td>29.77</td>
</tr>
<tr>
<td></td>
<td>MIT SCHEME</td>
<td>1472.26</td>
<td>2500.00</td>
<td>309.66</td>
<td>591.36</td>
<td>1243.26</td>
<td>1428.57</td>
<td>51.36</td>
<td>79.10</td>
</tr>
<tr>
<td></td>
<td>MzC</td>
<td>2073.26</td>
<td>3434.64</td>
<td>340.30</td>
<td>655.83</td>
<td>2436.26</td>
<td>1996.40</td>
<td>72.45</td>
<td>150.02</td>
</tr>
<tr>
<td></td>
<td>MzSCHEME</td>
<td>2344.70</td>
<td>4076.16</td>
<td>409.95</td>
<td>843.68</td>
<td>2000.89</td>
<td>2332.43</td>
<td>80.78</td>
<td>134.00</td>
</tr>
<tr>
<td></td>
<td>SCHEME-&gt;C</td>
<td>391.42</td>
<td>605.26</td>
<td>109.77</td>
<td>198.43</td>
<td>324.95</td>
<td>328.84</td>
<td>12.74</td>
<td>18.28</td>
</tr>
<tr>
<td></td>
<td>SCMUTILS</td>
<td>3321.20</td>
<td></td>
<td></td>
<td></td>
<td>2800.71</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STALIN</td>
<td>208.10</td>
<td>366.08</td>
<td>51.84</td>
<td>91.86</td>
<td>166.96</td>
<td>212.93</td>
<td>7.68</td>
<td>11.40</td>
</tr>
</tbody>
</table>

- ■ not implemented but could implement
- □ not implemented in existing tool
- ■■ can’t implement
Outline

1. AD
2. AD for CV
3. AD for AC
4. AD for ML
5. AD for AI
6. AD for CL
(define (e i n)
  (if (zero? n)
      ()
      (cons (if (zero? i) 1.0 0.0)
        (e (- i 1) (- n 1))))

(define ((gradient f) x)
  (let ((n (length x)))
    (map (lambda (i) (tangent ((j* f) (bundle x (e i n)))))
      (iota n))))

(define (gradient-ascent f x0 n eta)
  (if (zero? n)
      (list x0 (f x0) ((gradient f) x0))
      (gradient-ascent f
        (zip (lambda (xi gi) (+ xi (* eta gi)))
          x0
          ((gradient f) x0))
        (- n 1)
        eta)))
(define (e i n)
  (if (zero? n)
      ()
      (cons (if (zero? i) 1.0 0.0)
            (e (- i 1) (- n 1))))
)

(define ((gradient f) x)
  (let ((n (length x)))
    (map (lambda (i) (tangent ((j* f) (bundle x (e i n))))
           (iota n))))
)
(define (e i n)
  (if (zero? n)
    ()
    (cons (if (zero? i) 1.0 0.0)
      (e (- i 1) (- n 1))))

(define ((gradient f) x)
  (let ((n (length x)))
    (map (lambda (i) (tangent ((j* f) (bundle x (e i n))))
      (iota n))))

(define (gradient-ascent f x0 n eta)
  (if (zero? n)
    (list x0 (f x0) ((gradient f) x0))
    (gradient-ascent f
      (zip (lambda (xi gi) (+ xi (* eta gi)))
        x0
        ((gradient f) x0)
        (- n 1)
        eta)))
(define ((gradient f) x) (cdr ((cdr ((*j f) (*j x))) 1.0)))

(define (gradient-ascent f x0 n eta)
  (if (zero? n)
      (list x0 (f x0) ((gradient f) x0))
      (gradient-ascent f
                      (zip (lambda (xi gi) (+ xi (* eta gi)))
                           x0
                           ((gradient f) x0))
                      (- n 1)
                      eta)))
Neural Networks in VLAD

(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))

(define (sigmoid x) (/ 1 (+ (exp (- 0 x)) 1)))

(define ((forward-pass ws-layers) in)
  (if (null? ws-layers)in((forward-pass (cdr ws-layers))(map sigmoid (sum-layer in (car ws-layers)))))

(define ((error-on-dataset dataset) ws-layers)
  ((fold + 0)(map (lambda ((list in target))(*0.5 (magnitude-squared (v- ((forward-pass ws-layers) in) target))))dataset)))

(gradient-descent (error-on-dataset '(((0 0) (0))
  ((0 1) (1))((1 0) (1))((1 1) (0))))'(((0 -0.284227 1.16054) (0 0.617194 1.30467))((0 -0.084395 0.648461)))1000.00.3)
Neural Networks in VLAD

```
(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))
```
Neural Networks in VLAD

(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))

(define (sigmoid x) (/ 1 (+ (exp (- 0 x)) 1)))

(define ((forward-pass ws-layers) in)
  (if (null? ws-layers) in
      ((forward-pass (cdr ws-layers))
       (map sigmoid (sum-layer in (car ws-layers))))))

(define ((error-on-dataset dataset) ws-layers)
  ((fold + 0)
   (map (lambda ((list in target))
         (* 0.5 (magnitude-squared (v- ((forward-pass ws-layers) in) target))))
        dataset)))

(gradient-descent (error-on-dataset '(((0 0) (0))
                                      ((0 1) (1))
                                      ((1 0) (1))
                                      ((1 1) (0))))
                   '(((0 -0.284227 1.16054) (0 0.617194 1.30467))
                     ((0 -0.084395 0.648461))))
                   1000.0 0.3)
Neural Networks in VLAD

(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))

(define (sigmoid x) (/ 1 (+ (exp (- 0 x)) 1)))

(define ((forward-pass ws-layers) in)
  (if (null? ws-layers) in
      ((forward-pass (cdr ws-layers))
       (map sigmoid (sum-layer in (car ws-layers)))))

(define ((error-on-dataset dataset) ws-layers)
  ((fold + 0) (map (lambda ((list in target))(*0.5 (magnitude-squared (v- ((forward-pass ws-layers) in) target))))dataset)))

(gradient-descent (error-on-dataset '(((0 0) (0))
                            ((0 1) (1))
                            ((1 0) (1))
                            ((1 1) (0))))
                    '(((0 -0.284227 1.16054) (0 0.617194 1.30467))
                      ((0 -0.084395 0.648461)))
                    1000.0 0.3)
(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))

(define (sigmoid x) (/ 1 (+ (exp (- 0 x)) 1)))

(define ((forward-pass ws-layers) in)
  (if (null? ws-layers)
      in
      ((forward-pass (cdr ws-layers))
        (map sigmoid (sum-layer in (car ws-layers)))))

(define ((error-on-dataset dataset) ws-layers)
  ((fold + 0)
   (map (lambda ((list in target))
          (* 0.5 (magnitude-squared (v- ((forward-pass ws-layers) in) target))))
        dataset)))
Neural Networks in VLAD

(define ((sum-activities activities) bias ws)
  ((fold + bias) (zip * ws activities)))

(define (sum-layer activities ws-layer)
  (map (sum-activities activities) ws-layer))

(define (sigmoid x) (/ 1 (+ (exp (- 0 x)) 1)))

(define ((forward-pass ws-layers) in)
  (if (null? ws-layers)
      in
      ((forward-pass (cdr ws-layers))
       (map sigmoid (sum-layer in (car ws-layers))))))

(define ((error-on-dataset dataset) ws-layers)
  ((fold + 0)
   (map (lambda ((list in target))
           (* 0.5 (magnitude-squared (v- ((forward-pass ws-layers) in) target))))
        dataset)))

(gradient-descent (error-on-dataset '(((0 0) (0))
                                      ((0 1) (1))
                                      ((1 0) (1))
                                      ((1 1) (0)))))

'(((0 -0.284227 1.16054) (0 0.617194 1.30467))
  ((0 -0.084395 0.648461)))

1000.0
0.3)
## Performance Comparison

<table>
<thead>
<tr>
<th>Language</th>
<th>Tool</th>
<th>Fs</th>
<th>Fv</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vlad</td>
<td>Stalingrad</td>
<td>1.00</td>
<td></td>
<td>1.00</td>
</tr>
<tr>
<td>Fortran</td>
<td>ADIFOR</td>
<td>11.84</td>
<td>2.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TAPENADE</td>
<td>11.35</td>
<td>4.33</td>
<td>6.24</td>
</tr>
<tr>
<td>C</td>
<td>ADIC</td>
<td>16.33</td>
<td>3.93</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>ADOL-C</td>
<td>12.34</td>
<td>3.89</td>
<td>35.53</td>
</tr>
<tr>
<td></td>
<td>CppAD</td>
<td>42.15</td>
<td></td>
<td>23.69</td>
</tr>
<tr>
<td></td>
<td>FADBAD++</td>
<td>98.96</td>
<td>33.15</td>
<td>53.03</td>
</tr>
<tr>
<td>ML</td>
<td>MLTON</td>
<td>73.94</td>
<td></td>
<td>37.94</td>
</tr>
<tr>
<td></td>
<td>OCAML</td>
<td>157.75</td>
<td></td>
<td>149.14</td>
</tr>
<tr>
<td></td>
<td>SML/NJ</td>
<td>142.71</td>
<td></td>
<td>94.97</td>
</tr>
<tr>
<td>Haskell</td>
<td>GHC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheme</td>
<td>Bigloo</td>
<td>577.45</td>
<td></td>
<td>306.60</td>
</tr>
<tr>
<td></td>
<td>Chicken</td>
<td>1391.75</td>
<td></td>
<td>971.91</td>
</tr>
<tr>
<td></td>
<td>Gambit</td>
<td>545.20</td>
<td></td>
<td>341.73</td>
</tr>
<tr>
<td></td>
<td>Ikarus</td>
<td>216.42</td>
<td></td>
<td>147.49</td>
</tr>
<tr>
<td></td>
<td>Larceny</td>
<td>955.98</td>
<td></td>
<td>486.64</td>
</tr>
<tr>
<td></td>
<td>MIT Scheme</td>
<td>1900.04</td>
<td></td>
<td>1141.22</td>
</tr>
<tr>
<td></td>
<td>MzC</td>
<td>2439.93</td>
<td></td>
<td>1571.52</td>
</tr>
<tr>
<td></td>
<td>MzScheme</td>
<td>3477.86</td>
<td></td>
<td>1866.28</td>
</tr>
<tr>
<td></td>
<td>Scheme-&gt;C</td>
<td>484.24</td>
<td></td>
<td>233.75</td>
</tr>
<tr>
<td></td>
<td>SCMUTILS</td>
<td>4544.48</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stalin</td>
<td>832.68</td>
<td></td>
<td>367.84</td>
</tr>
</tbody>
</table>

- ■ not implemented but could implement
- ▭ not implemented in existing tool
- ■ ■ can’t implement
Outline

1. AD
2. AD for CV
3. AD for AC
4. AD for ML
5. AD for AI
6. AD for CL
$P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2$

Probabilistic Lambda Calculus

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{false}) &= 1 - p_1
\end{align*}
\]

Probabilistic Lambda Calculus

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 \\
\Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 \\
\Pr(x_1 \mapsto \text{false}) &= 1 - p_1
\end{align*}
\]

\[
\begin{align*}
\Pr(\mathcal{E}(P) = 0 | p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1 | p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2 | p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

$P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2$

\[
\begin{align*}
Pr(x_0 \leftrightarrow \text{true}) &= p_0 & Pr(x_0 \leftrightarrow \text{false}) &= 1 - p_0 \\
Pr(x_1 \leftrightarrow \text{true}) &= p_1 & Pr(x_1 \leftrightarrow \text{false}) &= 1 - p_1 \\
Pr(\mathcal{E}(P) = 0 | p_0, p_1) &= p_0 \\
Pr(\mathcal{E}(P) = 1 | p_0, p_1) &= (1 - p_0)p_1 \\
Pr(\mathcal{E}(P) = 2 | p_0, p_1) &= (1 - p_0)(1 - p_1) \\
\prod_{v \in \{0, 1, 2, 2\}} Pr(\mathcal{E}(P) = v | p_0, p_1) &= p_0(1 - p_0)^3 p_1(1 - p_1)^2
\end{align*}
\]

\[ P = \text{if } x_0 \text{ then } 0 \text{ else if } x_1 \text{ then } 1 \text{ else } 2 \]

\[
\begin{align*}
\Pr(x_0 \mapsto \text{true}) &= p_0 & \Pr(x_0 \mapsto \text{false}) &= 1 - p_0 \\
\Pr(x_1 \mapsto \text{true}) &= p_1 & \Pr(x_1 \mapsto \text{false}) &= 1 - p_1 \\
\Pr(\mathcal{E}(P) = 0|p_0, p_1) &= p_0 \\
\Pr(\mathcal{E}(P) = 1|p_0, p_1) &= (1 - p_0)p_1 \\
\Pr(\mathcal{E}(P) = 2|p_0, p_1) &= (1 - p_0)(1 - p_1)
\end{align*}
\]

\[
\prod_{v \in \{0, 1, 2, 2\}} \Pr(\mathcal{E}(P) = v|p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2
\]

\[
\text{argmax}_{p_0, p_1} \prod_{v \in \{0, 1, 2, 2\}} \Pr(\mathcal{E}(P) = v|p_0, p_1) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle
\]

\begin{prolog}
\begin{verbatim}
\(p(0).\)
\(p(X) :- q(X).\)
\(q(1).\)
\(q(2).\)
\end{verbatim}
\end{prolog}
$\text{Pr}(p(0).) = p_0$

$\text{Pr}(p(X) : -q(X).) = 1 - p_0$

$\text{Pr}(q(1).) = p_1$

$\text{Pr}(q(2).) = 1 - p_1$
\[
\begin{align*}
\Pr(p(0).) &= p_0 \\
\Pr(p(X) :- q(X).) &= 1 - p_0 \\
\Pr(q(1).) &= p_1 \\
\Pr(q(2).) &= 1 - p_1 \\
\Pr(\neg p(0).) &= p_0 \\
\Pr(\neg p(1).) &= (1 - p_0)p_1 \\
\Pr(\neg p(2).) &= (1 - p_0)(1 - p_1)
\end{align*}
\]
\begin{align*}
\Pr(p(0).) &= p_0 \\
\Pr(p(X) \leftarrow q(X).) &= 1 - p_0 \\
\Pr(q(1).) &= p_1 \\
\Pr(q(2).) &= 1 - p_1 \\
\Pr(\neg p(0).) &= p_0 \\
\Pr(\neg p(1).) &= (1 - p_0)p_1 \\
\Pr(\neg p(2).) &= (1 - p_0)(1 - p_1) \\
\prod_{q \in \{p(0), p(1), p(2), p(2)\}} \Pr(\neg q.) &= p_0(1 - p_0)^3p_1(1 - p_1)^2
\end{align*}
Probabilistic Prolog

\[ \text{Pr}(p(0)) = p_0 \]
\[ \text{Pr}(p(X) :- q(X)) = 1 - p_0 \]
\[ \text{Pr}(q(1)) = p_1 \]
\[ \text{Pr}(q(2)) = 1 - p_1 \]

\[ \text{Pr}(\neg p(0)) = p_0 \]
\[ \text{Pr}(\neg p(1)) = (1 - p_0)p_1 \]
\[ \text{Pr}(\neg p(2)) = (1 - p_0)(1 - p_1) \]

\[ \prod_{q \in \{p(0), p(1), p(2), p(2)\}} \text{Pr}(\neg q) = p_0(1 - p_0)^3p_1(1 - p_1)^2 \]

\[ \text{argmax}_{p_0, p_1} \prod_{q \in \{p(0), p(1), p(2), p(2)\}} \text{Pr}(\neg q) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle \]
(define (evaluate expression environment)
  (cond
    ((constant-expression? expression)
      (singleton-tagged-distribution
        (constant-expression-value expression)))
    ((variable-access-expression? expression)
      (lookup-value
        (variable-access-expression-variable expression) environment))
    ((lambda-expression? expression)
      (singleton-tagged-distribution
        (lambda (tagged-distribution)
          (evaluate
            (lambda-expression-body expression)
            (cons (make-binding (lambda-expression-variable expression)
                                 tagged-distribution)
                 environment))))
    (else (let ((tagged-distribution
                  (evaluate (application-argument expression) environment))
                 (evaluate (application-callee expression) environment)))
              (map-tagged-distribution
               (lambda (value) (value tagged-distribution))
               (evaluate (application-callee expression) environment))))))
(define (evaluate expression environment)
  (cond
   ((constant-expression? expression)
    (singleton-tagged-distribution
     (constant-expression-value expression)))
   ((variable-access-expression? expression)
    (lookup-value
     (variable-access-expression-variable expression) environment))
   ((lambda-expression? expression)
    (singleton-tagged-distribution
     (lambda (tagged-distribution)
      (evaluate
       (lambda-expression-body expression)
       (cons (make-binding (lambda-expression-variable expression)
                         tagged-distribution)
             environment))))
   (else (let ((tagged-distribution
                 (evaluate (application-argument expression) environment))
            (map-tagged-distribution
             (lambda (value) (value tagged-distribution))
             (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
    ((constant-expression? expression)
      (singleton-tagged-distribution
        (constant-expression-value expression)))
    ((variable-access-expression? expression)
      (lookup-value
        (variable-access-expression-variable expression) environment))
    ((lambda-expression? expression)
      (singleton-tagged-distribution
        (lambda (tagged-distribution)
          (evaluate
            (lambda-expression-body expression)
            (cons (make-binding (lambda-expression-variable expression) tagged-distribution)
                  environment))))
    (else (let ((tagged-distribution
                        (evaluate (application-argument expression) environment))
          (map-tagged-distribution
            (lambda (value) (value tagged-distribution))
            (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
    ((constant-expression? expression)
      (singleton-tagged-distribution
        (constant-expression-value expression)))
    ((variable-access-expression? expression)
      (lookup-value
        (variable-access-expression-variable expression) environment))
    ((lambda-expression? expression)
      (singleton-tagged-distribution
        (lambda (tagged-distribution)
          (evaluate
            (lambda-expression-body expression)
            (cons (make-binding (lambda-expression-variable expression)
                              tagged-distribution)
                  environment))))))
  (else (let ((tagged-distribution
                (evaluate (application-argument expression) environment))
            (map-tagged-distribution
              (lambda (value) (value tagged-distribution))
              (evaluate (application-callee expression) environment))))))}
(define (evaluate expression environment)
 (cond
   ((constant-expression? expression)
    (singleton-tagged-distribution
     (constant-expression-value expression)))
   ((variable-access-expression? expression)
    (lookup-value
     (variable-access-expression-variable expression) environment))
   ((lambda-expression? expression)
    (singleton-tagged-distribution
     (lambda (tagged-distribution)
      (evaluate
       (lambda-expression-body expression)
       (cons (make-binding (lambda-expression-variable expression)
                         tagged-distribution)
             environment))))
   (else (let ((tagged-distribution
                (evaluate (application-argument expression) environment))
             (map-tagged-distribution
              (lambda (value) (value tagged-distribution))
              (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
   ((constant-expression? expression)
    (singleton-tagged-distribution
     (constant-expression-value expression)))
   ((variable-access-expression? expression)
    (lookup-value
     (variable-access-expression-variable expression) environment))
   ((lambda-expression? expression)
    (singleton-tagged-distribution
     (lambda (tagged-distribution)
      (evaluate
       (lambda-expression-body expression)
       (cons (make-binding (lambda-expression-variable expression) tagged-distribution)
          environment))))
   (else (let ((tagged-distribution
                  (evaluate (application-argument expression) environment)))
     (map-tagged-distribution
      (lambda (value) (value tagged-distribution))
      (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
   ((constant-expression? expression)
     (singleton-tagged-distribution
      (constant-expression-value expression)))
   ((variable-access-expression? expression)
     (lookup-value
      (variable-access-expression-variable expression) environment))
   ((lambda-expression? expression)
     (singleton-tagged-distribution
      (lambda (tagged-distribution)
        (evaluate
         (lambda-expression-body expression)
         (cons (make-binding (lambda-expression-variable expression)
                         tagged-distribution)
               environment)))))
   (else (let ((tagged-distribution
                 (evaluate (application-argument expression) environment))
             (map-tagged-distribution
              (lambda (value) (value tagged-distribution))
              (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
   ((constant-expression? expression)
    (singleton-tagged-distribution
     (constant-expression-value expression)))
   ((variable-access-expression? expression)
    (lookup-value
     (variable-access-expression-variable expression) environment))
   ((lambda-expression? expression)
    (singleton-tagged-distribution
     (lambda (tagged-distribution)
      (evaluate
       (lambda-expression-body expression)
       (cons (make-binding (lambda-expression-variable expression)
                  tagged-distribution)
             environment))))
   (else (let ((tagged-distribution
                (evaluate (application-argument expression) environment))
            (map-tagged-distribution
             (lambda (value) (value tagged-distribution))
             (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
    ((constant-expression? expression)
     (singleton-tagged-distribution
      (constant-expression-value expression)))
    ((variable-access-expression? expression)
     (lookup-value
      (variable-access-expression-variable expression) environment))
    ((lambda-expression? expression)
     (singleton-tagged-distribution
      (lambda (tagged-distribution)
        (evaluate
         (lambda-expression-body expression)
         (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
               environment))))
    (else (let ((tagged-distribution
                   (evaluate (application-argument expression)
                              environment)))
            (map-tagged-distribution
             (lambda (value) (value tagged-distribution))
             (evaluate (application-callee expression) environment)))))))
(define (evaluate expression environment)
  (cond
    ((constant-expression? expression)
     (singleton-tagged-distribution
      (constant-expression-value expression)))
    ((variable-access-expression? expression)
     (lookup-value
      (variable-access-expression-variable expression) environment))
    ((lambda-expression? expression)
     (singleton-tagged-distribution
      (lambda (tagged-distribution)
        (evaluate
         (lambda-expression-body expression)
         (cons (make-binding (lambda-expression-variable expression)
                        tagged-distribution)
               environment))))
    (else (let ((tagged-distribution
                   (evaluate (application-argument expression) environment))
              (map-tagged-distribution
               (lambda (value) (value tagged-distribution))
               (evaluate (application-callee expression) environment)))))))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
           (evaluate (if x0 then 0 else if x1 then 1 else 2
                       (list Pr (x0 \(\mapsto\) true) = p0  Pr (x0 \(\mapsto\) false) = 1 - p0
                               Pr (x1 \(\mapsto\) true) = p1  Pr (x1 \(\mapsto\) false) = 1 - p1
                               ...))))))

   (map-reduce *
    1.0
    (lambda (value)
      (likelihood value tagged-distribution))
    '((0 1 2 2)))
   '((0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
       (evaluate if x0 then 0 else if x1 then 1 else 2
         (list Pr(x0 \rightarrow true) = p_0 \quad Pr(x0 \rightarrow false) = 1 - p_0
            Pr(x1 \rightarrow true) = p_1 \quad Pr(x1 \rightarrow false) = 1 - p_1
            )))

   (map-reduce
    * 1.0
    (lambda (value)
      (likelihood value tagged-distribution))
    `(0 1 2 2)))

  `(0.5 0.5)
  1000.0 0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
            (evaluate
              if x0 then 0 else if x1 then 1 else 2
              (list
                Pr(x₀ ← true) = p₀  Pr(x₀ ← false) = 1 − p₀
                Pr(x₁ ← true) = p₁  Pr(x₁ ← false) = 1 − p₁
                ...
              )))
      (map-reduce
       * 1.0
       (lambda (value)
            (likelihood value tagged-distribution))
       '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
           (evaluate
                (if x0 then 0 else if x1 then 1 else 2
                 (list
                   Pr(x0 \mapsto true) = p0  Pr(x0 \mapsto false) = 1 - p0
                   Pr(x1 \mapsto true) = p1  Pr(x1 \mapsto false) = 1 - p1
                   ...))))
      (map-reduce
       * 1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0 1 2 2) )))
    '(0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
           (evaluate
            (if x0 then 0 else if x1 then 1 else 2
            (list
              Pr(x0 \mapsto true) = p0  Pr(x0 \mapsto false) = 1 - p0
              Pr(x1 \mapsto true) = p1  Pr(x1 \mapsto false) = 1 - p1
              ...))))))

   (map-reduce
    *         
    1.0       
    (lambda (value)
      (likelihood value tagged-distribution))
    '(0 1 2 2))}))

  '(0.5 0.5)
1000.0
0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
          (evaluate if x0 then 0 else if x1 then 1 else 2
            (list Pr(x0 ↦ true) = p0  Pr(x0 ↦ false) = 1 − p0
                  Pr(x1 ↦ true) = p1  Pr(x1 ↦ false) = 1 − p1
                  ...))))
      (map-reduce
       * 1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
      (evaluate if x0 then 0 else if x1 then 1 else 2
        (list Pr(x0 \rightarrow true) = p, Pr(x0 \rightarrow false) = 1 - p,
            Pr(x1 \rightarrow true) = p1, Pr(x1 \rightarrow false) = 1 - p1
            ...) ))))

    (map-reduce *
      1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2)))

  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
 (lambda (p)
   (let ((tagged-distribution
            (evaluate if x_0 then 0 else if x_1 then 1 else 2
                       (list Pr(x_0 \mapsto true) = p_0  Pr(x_0 \mapsto false) = 1 - p_0
                           Pr(x_1 \mapsto true) = p_1  Pr(x_1 \mapsto false) = 1 - p_1
                           ...))))
    (map-reduce
      *
      1.0
      (lambda (value)
        (likelihood value tagged-distribution))
      '(0 1 2 2))))
  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
            (evaluate
                (if x0 then 0 else if x1 then 1 else 2
                (list Pr(x0 \rightarrow true) = p0
                      Pr(x0 \rightarrow false) = 1 - p0
                      Pr(x1 \rightarrow true) = p1
                      Pr(x1 \rightarrow false) = 1 - p1
                      ...
                    )))
      (map-reduce
       * 1.0
       (lambda (value)
         (likelihood value tagged-distribution))
       '(0 1 2 2)))
    '(0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
  (lambda (p)
    (let ((tagged-distribution
            (evaluate
              (if x0 then 0 else if x1 then 1 else 2
               (list Pr(x0 \mapsto true) = p_0 Pr(x0 \mapsto false) = 1 - p_0
                     Pr(x1 \mapsto true) = p_1 Pr(x1 \mapsto false) = 1 - p_1
                      ...)))))

      (map-reduce
       *
       1.0
       (lambda (value)
         (likelihood value tagged-distribution))
      '(0 1 2 2)))))
'(0.5 0.5)
1000.0
0.1)
(define (proof-distribution term clauses)
  (let ((offset ...
    (map-reduce
      append
      ()
      (lambda (clause)
        (let ((clause (alpha-rename clause offset)))
          (let loop ((p (clause-p clause))
              (substitution (unify term (clause-term clause)))
              (terms (clause-terms clause)))
            (if (boolean? substitution)
              ()
              (if (null? terms)
                (list (make-double p substitution))
                (map-reduce
                  append
                  ()
                  (lambda (double)
                    (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
                    (proof-distribution
                      (apply-substitution substitution (first terms)) clauses))))))))
  clauses)))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     '()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
       (let loop ((p (clause-p clause))
                  (substitution (unify term (clause-term clause)))
                  (terms (clause-terms clause)))
        (if (boolean? substitution)
          '()
          (if (null? terms)
            (list (make-double p substitution))
            (map-reduce
             append
             '()
             (lambda (double)
              (loop (* p (double-p double))
               (append substitution (double-substitution double))
               (rest terms)))
              (proof-distribution
               (apply-substitution substitution (first terms) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     '()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
        (let loop ((p (clause-p clause))
             (substitution (unify term (clause-term clause)))
             (terms (clause-terms clause)))
          (if (boolean? substitution)
            '()
            (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                  (append substitution (double-substitution double))
                  (rest terms)))
                (proof-distribution
                 (apply-substitution substitution (first terms)) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     '()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
       (let loop ((p (clause-p clause))
                  (substitution (unify term (clause-term clause)))
                  (terms (clause-terms clause)))
        (if (boolean? substitution)
          '()
          (if (null? terms)
            (list (make-double p substitution))
            (map-reduce
             append
             '()
             (lambda (double)
              (loop (* p (double-p double))
                (append substitution (double-substitution double))
                (rest terms)))
            (proof-distribution
             (apply-substitution substitution (first terms)) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
      append
      '()
      (lambda (clause)
        (let ((clause (alpha-rename clause offset)))
          (let loop ((p (clause-p clause))
                      (substitution (unify term (clause-term clause)))
                      (terms (clause-terms clause)))
            (if (boolean? substitution)
                '()
              (if (null? terms)
                  (list (make-double p substitution))
                  (map-reduce
                    append
                    '()
                    (lambda (double)
                      (loop (* p (double-p double))
                        (append substitution (double-substitution double))
                        (rest terms)))
                      (proof-distribution
                        (apply-substitution substitution (first terms)) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     '()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
       (let loop ((p (clause-p clause))
                  (substitution (unify term (clause-term clause)))
                  (terms (clause-terms clause)))
        (if (boolean? substitution)
            '()
            (if (null? terms)
                (list (make-double p substitution))
                (map-reduce
                 append
                 '()
                 (lambda (double)
                  (loop (* p (double-p double))
                   (append substitution (double-substitution double))
                   (rest terms)))
                 (proof-distribution
                  (apply-substitution substitution (first terms)) clauses))))))
  clauses)))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     '()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
       (let loop ((p (clause-p clause))
                  (substitution (unify term (clause-term clause)))
                  (terms (clause-terms clause)))
        (if (boolean? substitution)
          ()
          (if (null? terms)
            (list (make-double p substitution))
            (map-reduce
             append
             '()
             (lambda (double)
              (loop (* p (double-p double))
                   (append substitution (double-substitution double))
                   (rest terms)))
              (proof-distribution
               (apply-substitution substitution (first terms)) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
     append
     ()
     (lambda (clause)
      (let ((clause (alpha-rename clause offset)))
       (let loop ((p (clause-p clause))
                  (substitution (unify term (clause-term clause)))
                  (terms (clause-terms clause)))
        (if (boolean? substitution)
            ()
            (if (null? terms)
                (list (make-double p substitution))
                (map-reduce
                 append
                 ()
                 (lambda (double)
                  (loop (* p (double-p double))
                    (append substitution (double-substitution double))
                    (rest terms)))
                 (proof-distribution
                  (apply-substitution substitution (first terms)) clauses))))))))
  clauses))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
      append
      '()
      (lambda (clause)
        (let ((clause (alpha-rename clause offset)))
          (let loop ((p (clause-p clause))
                        (substitution (unify term (clause-term clause)))
                        (terms (clause-terms clause)))
            (if (boolean? substitution)
                '()
                (if (null? terms)
                    (list (make-double p substitution))
                    (map-reduce
                      append
                      '()
                      (lambda (double)
                        (loop (* p (double-p double))
                          (append substitution (double-substitution double))
                          (rest terms)))
                      (proof-distribution
                        (apply-substitution substitution (first terms) clauses))))))))
  clauses)))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
      append
      ()
      (lambda (clause)
        (let ((clause (alpha-rename clause offset)))
          (let loop ((p (clause-p clause))
                        (substitution (unify term (clause-term clause)))
                        (terms (clause-terms clause)))
            (if (boolean? substitution)
                ()
                (if (null? terms)
                  (list (make-double p substitution))
                  (map-reduce
                    append
                    ()
                    (lambda (double)
                      (loop (* p (double-p double))
                        (append substitution (double-substitution double))
                        (rest terms)))
                      (proof-distribution
                        (apply-substitution substitution (first terms)) clauses))))))))
  clauses)))
(define (proof-distribution term clauses)
  (let ((offset ...))
    (map-reduce
      append
      ()
      (lambda (clause)
        (let ((clause (alpha-rename clause offset)))
          (let loop ((p (clause-p clause))
                        (substitution (unify term (clause-term clause)))
                        (terms (clause-terms clause)))
            (if (boolean? substitution)
                ()
                (if (null? terms)
                    (list (make-double p substitution))
                    (map-reduce
                      append
                      ()
                      (lambda (double)
                        (loop (* p (double-p double))
                          (append substitution (double-substitution double))
                          (rest terms)))
                        (proof-distribution
                          (apply-substitution substitution (first terms)) clauses)))))))))
  clauses)))
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      Pr(p(0).) = \( p_0 \)
      Pr(p(X):-q(X).) = 1 - p_0
      Pr(q(1).) = p_1
      Pr(q(2).) = 1 - p_1))

    (map-reduce
      *
      1.0
      (lambda (query)
        (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2))))
  '(0.5 0.5)
  1000.0
  0.1)
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      \(Pr(p(0).) = p_0\)
      \(Pr(p(X) ; q(X).) = 1 - p_0\)
      \(Pr(q(1).) = p_1\)
      \(Pr(q(2).) = 1 - p_1\))))

    (map-reduce
      * 
      1.0
      (lambda (query)
        (likelihood (proof-distribution query clauses)))
      '(p(0) p(1) p(2) p(2)))

    )
  )

'(0.5 0.5)
1000.0
0.1)
(gradient-ascent
 (lambda (p)
   (let ((clauses (list
                  Pr(p(0).) = \(p_0\)
                  Pr(p(X):-q(X).) = 1 - \(p_0\)
                  Pr(q(1).) = \(p_1\)
                  Pr(q(2).) = 1 - \(p_1\))))
     (map-reduce
      * 1.0
      (lambda (query)
        (likelihood (proof-distribution query clauses)))
      '(p(0) p(1) p(2) p(2))))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      Pr(p(0).) = p₀
      Pr(p(X):-q(X).) = 1 - p₀
      Pr(q(1).) = p₁
      Pr(q(2).) = 1 - p₁)))
      (map-reduce
        * 1.0
        (lambda (query)
          (likelihood (proof-distribution query clauses)))
        '(p(0) p(1) p(2) p(2))))
    )
    (0.5 0.5)
    1000.0
    0.1)
(gradient-ascent
  (lambda (p)
    (let ((clauses (list
      Pr(p(0).) = p₀
      Pr(p(X):-q(X).) = 1 - p₀
      Pr(q(1).) = p₁
      Pr(q(2).) = 1 - p₁)))
      (map-reduce
        *  
        1.0
        (lambda (query)
          (likelihood (proof-distribution query clauses)))
        '(p(0) p(1) p(2) p(2))))
    '(0.5 0.5)
    1000.0
    0.1))
(gradient-ascent
 (lambda (p)
  (let ((clauses (list
                 Pr(p(0).) = p₀
                 Pr(p(X):-q(X).) = 1 − p₀
                 Pr(q(1).) = p₁
                 Pr(q(2).) = 1 − p₁))))
   (map-reduce
    * 1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))
   '(0.5 0.5)
   1000.0
   0.1))
(gradient-ascent
 (lambda (p)
   (let ((clauses (list
                 Pr(p(0).) = p_0
                 Pr(p(X):-q(X).) = 1 - p_0
                 Pr(q(1).) = p_1
                 Pr(q(2).) = 1 - p_1)))
    (map-reduce
     * 1.0
     (lambda (query)
       (likelihood (proof-distribution query clauses)))
     '(p(0) p(1) p(2) p(2))))
  '(0.5 0.5)
  1000.0
  0.1)
static void f2679(double a_f2679_0,double a_f2679_1,double a_f2679_2,double a_f2679_3){
    int t272381=((a_f2679_2==0.)?0:1);
    double t272406;
    double t272405;
    double t272404;
    double t272403;
    double t272402;
    if((t272381==0)){
        double t272480=(1.-a_f2679_0);
        double t272572=(1.-a_f2679_1);
        double t273043=(a_f2679_0+0.);
        double t274185=(t272480*a_f2679_1);
        double t274426=(t274185+0.);
        double t275653=(t272480*t272572);
        double t275894=(t275653+0.);
        double t277121=(t272480*t272572);
        double t277362=(t277121+0.);
        double t277431=(t277362*1.);
        double t277436=(t275894*t277431);
        double t277441=(t274426*t277436);
        double t277446=(t273043*t277441);
        ...
        double t1777107=(t1774696+t1715394);
        double t1777194=(0.-t1745420);
        double t1778533=(t1777194+t1419700);
        t272406=a_f2679_0;
        t272405=a_f2679_1;
        t272404=t277446;
        t272403=t1778533;
        t272402=t1777107;
    }else {...}
    r_f2679_0=t272406;
    r_f2679_1=t272405;
    r_f2679_2=t272404;
    r_f2679_3=t272403;
    r_f2679_4=t272402;}

Siskind (Purdue)
## Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>probabilistic-</th>
<th>probabilistic-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lambda-calculus</td>
<td>prolog</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>R</td>
</tr>
<tr>
<td>VLAD</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>STALINGRAD</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ML</td>
<td>106.45</td>
<td>124.95</td>
</tr>
<tr>
<td>OCAML</td>
<td>215.73</td>
<td>538.68</td>
</tr>
<tr>
<td>SML/NJ</td>
<td>197.75</td>
<td>272.45</td>
</tr>
<tr>
<td>HASKELL</td>
<td>832.92</td>
<td>1048.11</td>
</tr>
<tr>
<td>GHC</td>
<td>2305.98</td>
<td>3283.00</td>
</tr>
<tr>
<td>SCHEME</td>
<td>879.88</td>
<td>1153.86</td>
</tr>
<tr>
<td>CHICKEN</td>
<td>437.46</td>
<td>531.10</td>
</tr>
<tr>
<td>GAMBIT</td>
<td>1651.01</td>
<td>1673.22</td>
</tr>
<tr>
<td>IKARUS</td>
<td>3491.10</td>
<td>4130.19</td>
</tr>
<tr>
<td>LARCENY</td>
<td>5289.17</td>
<td>5929.14</td>
</tr>
<tr>
<td>MZSCHEME</td>
<td>6235.78</td>
<td>7134.71</td>
</tr>
<tr>
<td>SCHEME-&gt;C</td>
<td>682.15</td>
<td>794.31</td>
</tr>
<tr>
<td>SCMUTILS</td>
<td>6456.99</td>
<td>80100.23</td>
</tr>
<tr>
<td>STALIN</td>
<td>1240.73</td>
<td>1137.41</td>
</tr>
</tbody>
</table>

- ▶ not implemented but could implement, including FORTRAN, C, and C++
- ▶ not implemented in existing tool
- ▶ can’t implement
(\text{flip } p)
(flip \( p \))

(probability \( e \))
(flip $p$)

(probability $e$)

(argmax $f$ $x_0$)
Reduced Gradient
Wolfe (1962, 1967)

\[
\arg\max_x f(x)
\]

\[
\sum x = 1
\]
\[ \arg\max_x f(x) \]

\[ \arg\max_{x \geq 0} f(x) \]

\[ Ax = b \]
Reduced Gradient
Wolfe (1962, 1967)

\[
\arg\max_x f(x)
\]

\[
\arg\max_x f(x) \quad x \geq 0 \quad Ax = b
\]

\[
\arg\max_x f(x) \quad x \geq 0 \quad \sum x = 1
\]
Reduced Gradient
Wolfe (1962, 1967)

\[
\arg\max_x f(x) \\
\arg\max_x f(x) \\
\arg\max_x f(x) \\
\arg\max_x f(x)
\]

\[
x \geq 0 \\
A x = b
\]

\[
x \geq 0 \\
\sum x = 1
\]

\[
(\arg\max f \ x_0)
\]
cost functions over observable and hidden variables:
cost functions over observable and hidden variables:
images, sensor data, robot control parameters, representations of game and assembly states, game rules, strategies, policies, assembly plans, utterances, parse trees, word, phrase, and utterance meanings
cost functions over observable and hidden variables: images, sensor data, robot control parameters, representations of game and assembly states, game rules, strategies, policies, assembly plans, utterances, parse trees, word, phrase, and utterance meanings

terms encode the relationships between different variables:
cost functions over observable and hidden variables:
images, sensor data, robot control parameters, representations of game and assembly states, game rules, strategies, policies, assembly plans, utterances, parse trees, word, phrase, and utterance meanings

terms encode the relationships between different variables:
compositional semantics, visual perception, motor control
cost functions over observable and hidden variables: images, sensor data, robot control parameters, representations of game and assembly states, game rules, strategies, policies, assembly plans, utterances, parse trees, word, phrase, and utterance meanings

- terms encode the relationships between different variables: compositional semantics, visual perception, motor control

- taking different variables to be dependent vs. independent and deciding which to marginalize and which to optimize (i.e., perform maximum-likelihood estimation), allows the same cost function and stochastic model to be used for a variety of different purposes
By taking images as input, marginalizing over game states, and optimizing over rule representations, we can learn game rules from observed game play.

By taking images, game rules, and word meanings as input, marginalizing over game states and parse trees, and optimizing over utterance likelihood, we can produce natural-language descriptions of game play and learned game rules.

By taking utterances and word meanings as input, marginalizing over game rules, and optimizing over robotic control sequences we can teach a robot game rules or command robotic game play with linguistic input.
By taking images as input, marginalizing over game states, and optimizing over rule representations, we can learn game rules from observed game play.
By taking images as input, marginalizing over game states, and optimizing over rule representations, we can learn game rules from observed game play.

By taking images, game rules, and word meanings as input, marginalizing over game states and parse trees, and optimizing over utterance likelihood, we can produce natural-language descriptions of game play and learned game rules.
By taking images as input, marginalizing over game states, and optimizing over rule representations, we can learn game rules from observed game play.

By taking images, game rules, and word meanings as input, marginalizing over game states and parse trees, and optimizing over utterance likelihood, we can produce natural-language descriptions of game play and learned game rules.

By taking utterances and word meanings as input, marginalizing over game rules, and optimizing over robotic control sequences we can teach a robot game rules or command robotic game play with linguistic input.
(define (lexicon game-state)
  (let ((things
        (append
          (map-n (lambda (position) (list 'position position)) 9)
          (map-n (lambda (position)
                   (list 'position-state
                         position
                         (list-ref game-state position)))
                  9)))
      (list
       (cons 'the (meaning for the))
       (cons 'x (meaning for x))
       (cons 'is-on (meaning for is on))
       (cons 'center (meaning for center))))))
(define (draw distribution)
  (let loop ((p 1)
             (pairs
               (remove-if
                (lambda (pair) (zero? (cdr pair)))
                distribution)))
    (if (or (null? (rest pairs))
            (flip (/ (cdr (first pairs)) p)))
      (car (first pairs))
      (loop (- p (cdr (first pairs)))
            (rest pairs))))
(define (position-state-draw distribution)
  (draw (map cons
         '(empty x o)
        distribution)))
(define (word-draw distribution)
  (draw (map cons
        '(the x is-on center)
        distribution)))
(define (interpret words game-state)
  (if (= (length words) 1)
      (cdr (assq (first words) (lexicon game-state)))
    (let* ((i (+ (draw
               (map
               cons
               (enumerate (- (length words) 1))
               (uniform (- (length words) 1)))))
            1))
      (left (interpret
             (sublist words 0 i)
             game-state))
      (right (interpret
              (sublist words i (length words))
              game-state)))
    (if (flip 0.5) (left right) (right left)))))
A Unified Computational Framework
Language Generation

\[(\text{argmax} \ (\lambda (\text{distributions}) \ (\text{probability} \ (\text{interpret} \ (\text{map word-draw distributions}) \\
\text{')(empty empty empty empty empty x empty empty empty empty empty)))) \ (\text{map-n} \ (\lambda (i) \ (\text{uniform 4})) \ 5))\]
\[(\text{argmax} \ (\lambda (\text{distributions}) \ (\text{probability} \ (\text{interpret} \ (\text{map word-draw distributions)} \ ' (\text{empty empty empty empty empty empty empty empty empty}))) \ (\text{map-n} (\lambda (i) (\text{uniform 4})) 5))) \]

\[
\#
\begin{align*}
\# & (0 \ 41 \ 0 \ 59) \quad ; x/center \\
\# & (100 \ 0 \ 0 \ 0) \quad ; \text{the} \\
\# & (0 \ 0 \ 100 \ 0) \quad ; \text{is-on} \\
\# & (100 \ 0 \ 0 \ 0) \quad ; \text{the} \\
\# & (0 \ 41 \ 0 \ 59) \quad ; x/center
\end{align*}
\]
(argmax
  (lambda (distributions)
    (probability
      (interpret
        '(the x is-on the center)
        (map position-state-draw distributions)))
    (map-n (lambda (i) (uniform 3)) 9))
A Unified Computational Framework

Language Understanding

\( \text{argmax} \)
\( \lambda (\text{distributions}) \)
\( \text{probability} \)
\( \lambda \)
\( \text{interpret} \)
\( '(\text{the x is-on the center}) \)
\( \text{(map position-state-draw distributions)}) \)
\( \text{(map-n (lambda (i) (uniform 3)) 9))} \)

#(#(67 0 33) #(67 0 33) #(67 0 33) ;empty/o empty/o empty/o
 #((67 0 33) #(0 100 0) #(67 0 33) ;empty/o x empty/o
 #((67 0 33) #(67 0 33) #(67 0 33)) ;empty/o empty/o empty/o