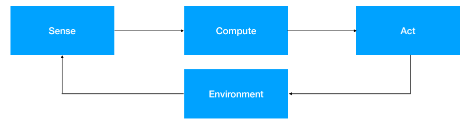
Perception

CS4501 - Robotics for Software Engineers

By Carl Hildebrandt

1

Robot Conceptual Architecture

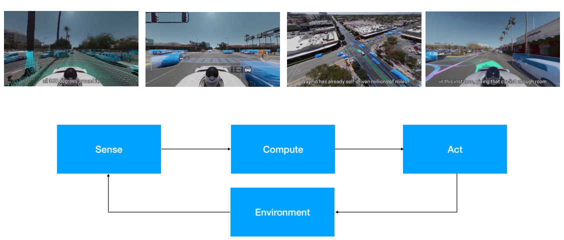


2

Self-driving Case Study

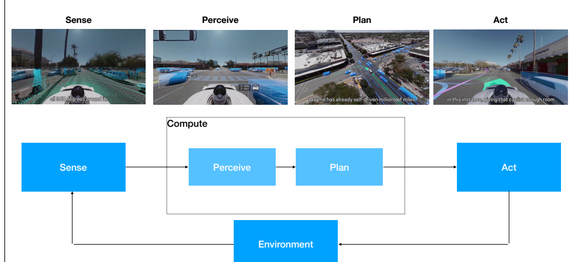
3

Robot Conceptual Architecture



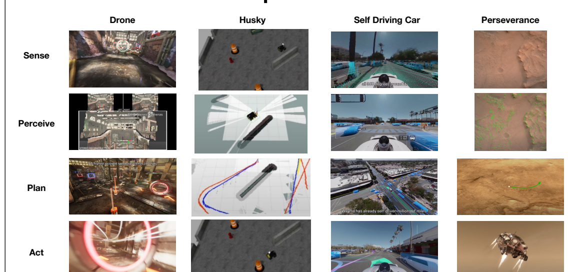
4

Robot Conceptual Architecture



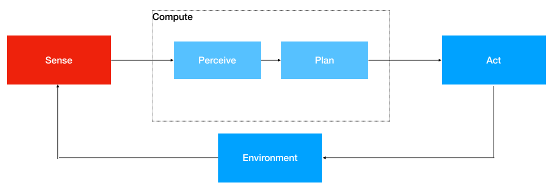
5

Robot Conceptual Architecture



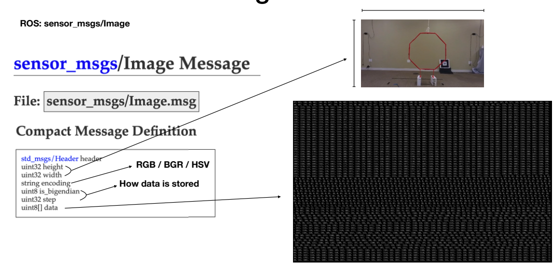
6

Robot Conceptual Architecture



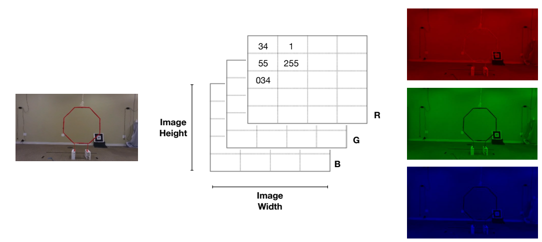
7

Image Data



8

Image Data



9

Perception

"Perception refers to the ability of an autonomous system to collect information and extract relevant knowledge from the environment."

–Pendleton, Scott Drew, et al. "Perception, planning, control, and coordination for autonomous vehicles." Machines 5.1 (2017)

10

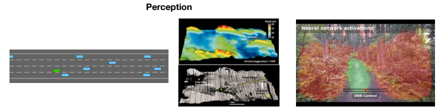
Perception Examples

How: Processing sensor data to create a higher-level abstraction of the data

Camera Sensor

Perception

Traffic Light: Stop



11

Main Types of Perception

Classification

Object Detection

Interpretation

12

Perception Algorithms

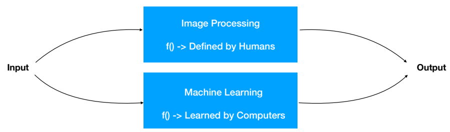
Perception estimates the state of the environment

Image Processing
An image is processed through parameterized transformations.

Key: We define this function

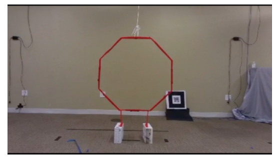
Machine Learning
Gather large amounts of data to learn or approximate the desired function.

Key: The computer learns this function



Example

Input



Example

Input

Output

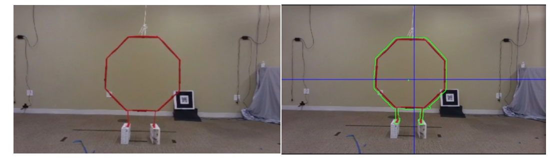


Image Processing Techniques

- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- …

OpenCV

NumPy

scikit-image
image processing in python

SciPy

Image Processing Techniques

- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- …

Color Filtering

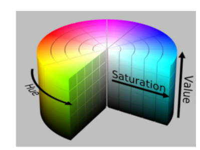
Idea
Remove all colors from an image except for a small range

Technical Implementation
Convert image into a format that makes selecting colors easy
Look at each pixel, if it is not in your selected range remove it

HSV Image Format
HSV stands for **H**ue, **S**aturation, **V**alue, and is a cylindrical color space.
Hue: Are colors rotating around a central vertical axis
Saturation: Defines the shade of the color from least saturated to most
Value: Defines brightness from darkest at the to brightest

Code

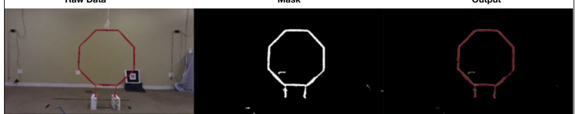
```
# Convert from BGR to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Look for orange
lower_color = np.array([0, 80, 80])
upper_color = np.array([255, 255, 255])

# Mask out all other colors
mask = cv2.inRange(hsv, lower_color, upper_color)

# Multiply mask (0 values) with image
result = cv2.bitwise_and(frame, frame, mask = mask)
```



By SharkDderivative work: SharkD [CC BY-SA 3.0 or GFDL], via Wikimedia Commons

Example: Color Filtering

Raw Data | **Mask** | **Output**



```
# Convert from BGR to HSV color space
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

# Look for orange
lower_color = np.array([0, 80, 80])
upper_color = np.array([255, 255, 255])

# Mask out all other colors
mask = cv2.inRange(hsv, lower_color, upper_color)

# Multiply mask (0 values) with image
result = cv2.bitwise_and(frame, frame, mask = mask)
```


Image Processing Techniques

Basic Image Operations

- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- ...

Background Subtraction

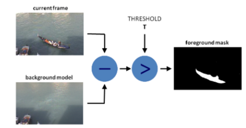
Idea
Remove background from current image

Technical Implementation
1) Estimate background for time t
2) Subtract estimated background from current frame
3) Apply threshold to absolute difference

Background Model
This technique requires a background model that contains the static part of the scene. Best suited for a static camera.

Code

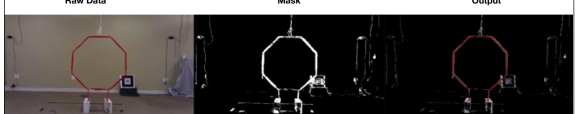
```
fgbg = cv2.createBackgroundSubtractorMOG2()

while(cap.isOpened()):
    ret, frame = cap.read()

    # Get the mask
    fgmask = fgbg.apply(frame)

    # Multiply mask (0 values) with image
    result = cv2.bitwise_and(frame, frame, mask = fgmask)
```



OpenCV Docs: https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

Example: Background Subtraction

Raw Data | **Mask** | **Output**



```
fgbg = cv2.createBackgroundSubtractorMOG2()

while(cap.isOpened()):
    ret, frame = cap.read()

    # Get the mask
    fgmask = fgbg.apply(frame)

    # Multiply mask (0 values) with image
    result = cv2.bitwise_and(frame, frame, mask = fgmask)
```


Image Processing Techniques

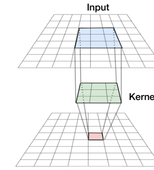
- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- ...

Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filtered Image | **Filter Kernel** | **Original Image**



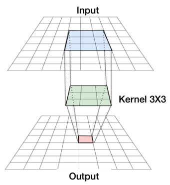
Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filtered Image — **Filter Kernel** — **Original Image**

$$\begin{bmatrix} 5 & 7 & 4 & 25 & 67 & 81 \\ 1 & 10 & 9 & 7 & 157 & 94 \\ 7 & 2 & 3 & 9 & 183 & 100 \\ 21 & 10 & 15 & 45 & 123 & 156 \\ 34 & 23 & 58 & 89 & 224 & 238 \\ 78 & 85 & 100 & 123 & 227 & 240 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} & & & & & \\ & & & & & \\ & & 3 & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$



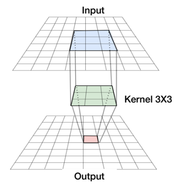
Convolution

Definition: Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy),$$

Filtered Image — **Filter Kernel** — **Original Image**

$$\begin{bmatrix} 5 & 7 & 4 & 25 & 67 & 81 \\ 1 & 10 & 9 & 7 & 157 & 94 \\ 7 & 2 & 3 & 9 & 183 & 100 \\ 21 & 10 & 15 & 45 & 123 & 156 \\ 34 & 23 & 58 & 89 & 224 & 238 \\ 78 & 85 & 100 & 123 & 227 & 240 \end{bmatrix} \times \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} & & & & & \\ & & & & & \\ & & 12.2 & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix}$$



Blurring

Idea
Remove high frequency content (e.g. noise, edges, etc)

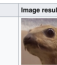
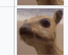
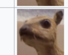
Technical Implementation
Convolve image with a normalized box filter
i.e. take an average of all pixel under the kernel area and replace the central element with this average.

Kernel

$$k = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Code

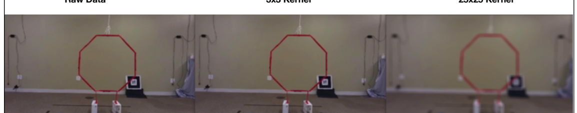
```
# Blur image using averaging filter kernel
blur1 = cv2.blur(frame,(3,3))
blur2 = cv2.blur(frame,(25,25))
```

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Kernel: https://en.wikipedia.org/wiki/Kernel_(image_processing)

Example: Blurring

Raw Data — **3x3 Kernel** — **25x25 Kernel**



```
# Blur image using averaging filter kernel
blur1 = cv2.blur(frame,(3,3))
blur2 = cv2.blur(frame,(25,25))
```


Image Processing Techniques

- Thresholding
- Color Filtering
- Blurring
- Smoothing
- Background subtraction
- Edge Detection
- Corner Detection
- Feature Matching
- Haar Cascade Object Detection
- …

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Original Image

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Gaussian Filter

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

31

31

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Gradient Magnitude

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

32

32

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Non Max Suppression

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

33

33

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Double Thresholding

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

34

34

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```

Edge Tracking

Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

35

35

(Canny) Edge Detection

Idea
Determine the horizontal and vertical gradient, large gradient == edge

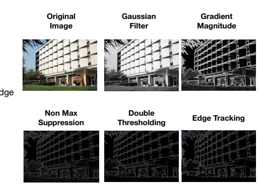
Technical Key
1) Apply gaussian filter to smooth the image and remove noise
2) Find the gradients of the image using Sobel operator
3) Apply non max suppression to thin edges
4) Apply double threshold to determine strong and weak edges
5) Track edges to remove edges that are not connected to a strong edge

Finding Gradients (Sobel Operator)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L \quad \text{and} \quad L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L.$$

Code

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```



Canny Edge Detector: https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html

36

36

Example: Edge Detection

Raw Data | **Edge Detection** | **5x5 Blur -> Edge Detection**

```
# Find the edges
edges = cv2.Canny(frame, 100, 200)

# Find the edges
blur = cv2.blur(frame,(5,5))
edges_blur = cv2.Canny(blur, 100, 200)
```


Perception Algorithms

Perception estimates the state of the environment

Image Processing
An image is processed through parameterized transformations.

Key: We define this function

Machine Learning
Gather large amounts of data a to learn or approximate the desired function.

Key: We learn this function

Perception Algorithms

Perception estimates the state of the environment

Image Processing
An image is processed through parameterized transformations.

Key: We define this function

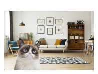
Pros:
Does not require datasets at all
Are easier to interpret by humans
Most do not require heavy computation resources
Libraries available to perform most standard functions

Cons:
Encode relatively simple functions

Machine Learning
Gather large amounts of data a to learn or approximate the desired function.

Key: We learn this function

Perception Algorithms

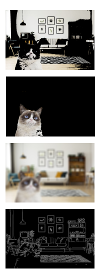


Input → output = f(input) Color Filtering → **Output**

Input → output = f(input) Background Subtraction → **Output**

Input → output = f(input) Blurring → **Output**

Input → output = f(input) Edge Detection → **Output**



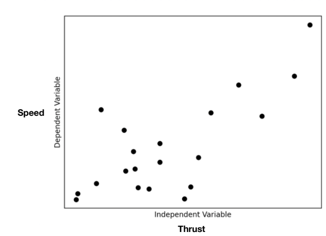
Machine Learning

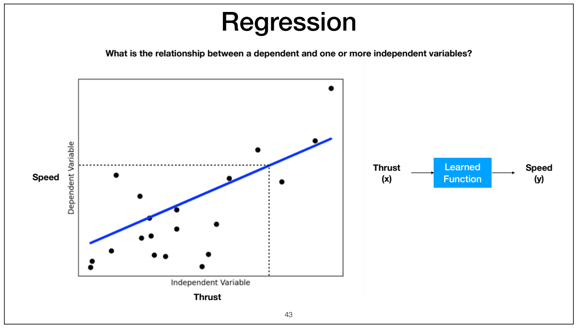
What happens if we don't know exactly how to define the function?

Input → output = f(input) → **Output**

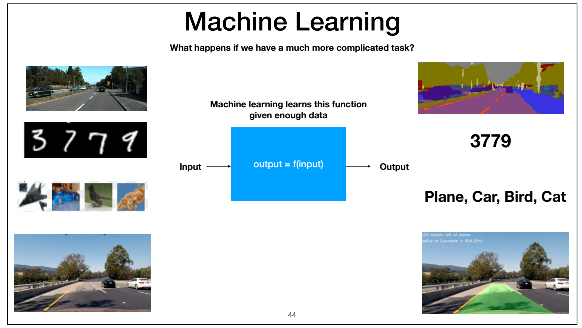
Regression

What is the relationship between a dependent and one or more independent variables?

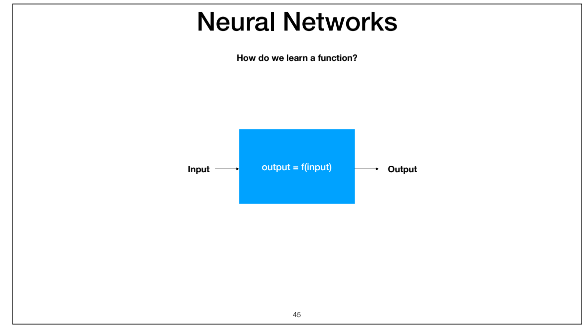




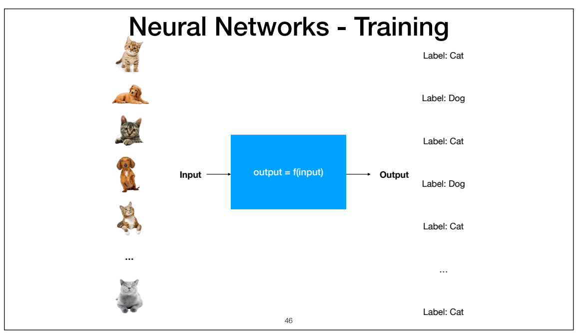
43



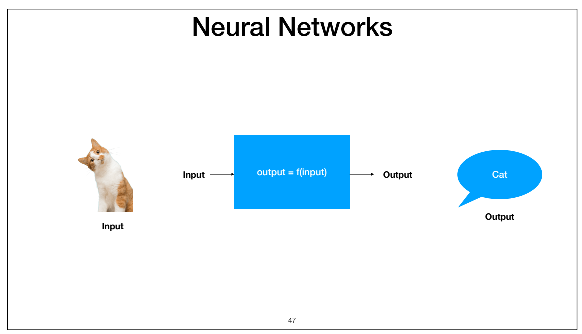
44



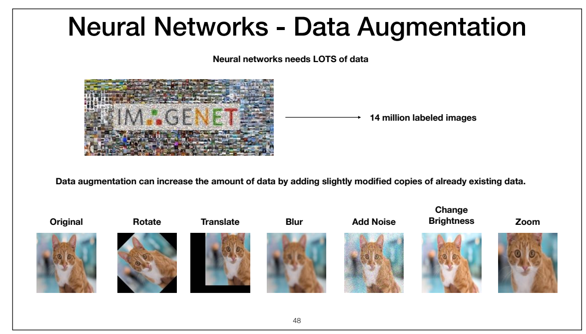
45



46

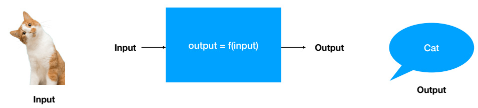


47



48

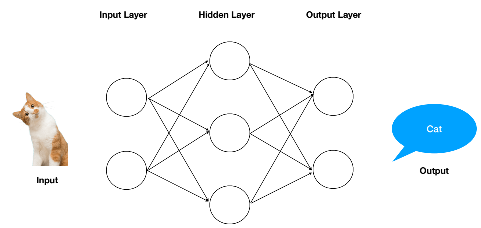
Neural Networks



49

49

Neural Networks



50

50

Neuron

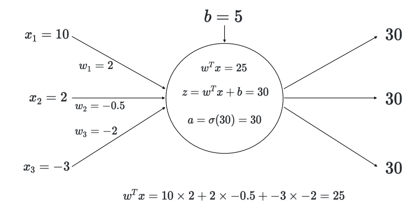
Output of a Neuron:

$y = \sigma(w^T x + b)$

y = output
σ = activation function
w = weights
x = input
b = bias

Activation Function (ReLu)

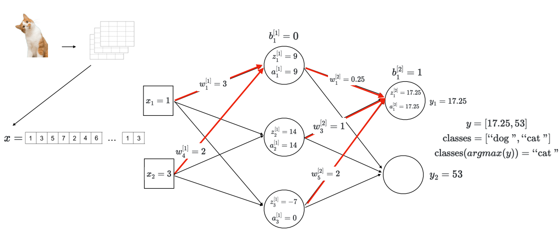
$\sigma(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$



51

51

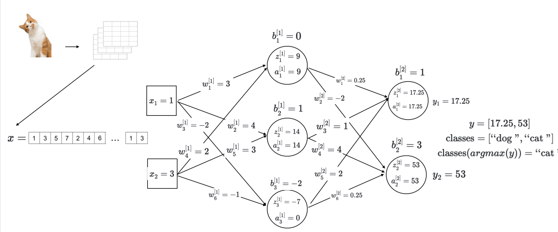
Neural Networks - Feedforward



52

52

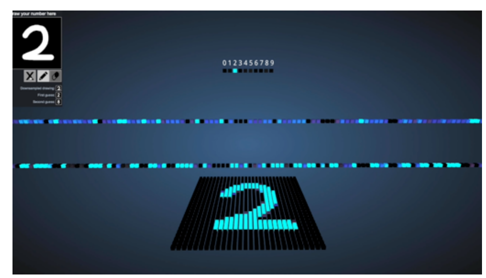
Neural Networks - Feedforward



53

53

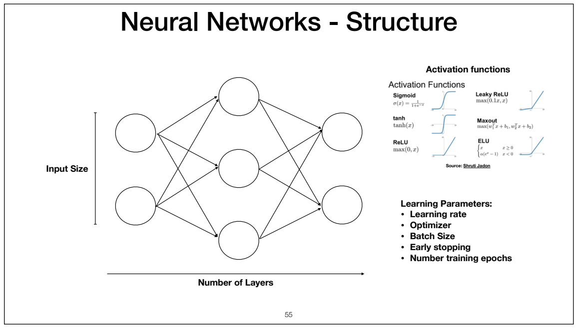
Neural Networks - Visualization



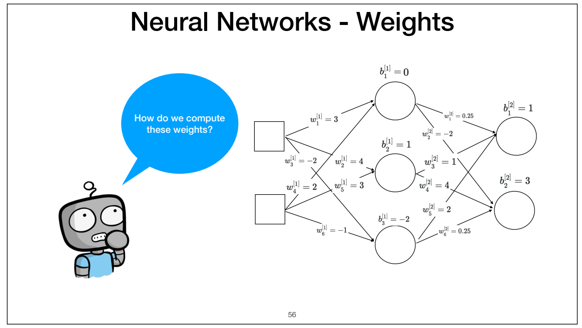
Source: https://www.cs.ryerson.ca/~aharley/vis/fc/

54

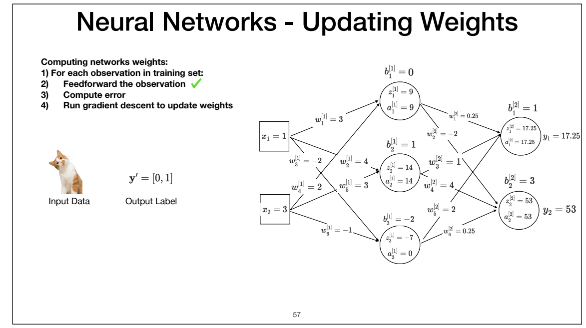
54



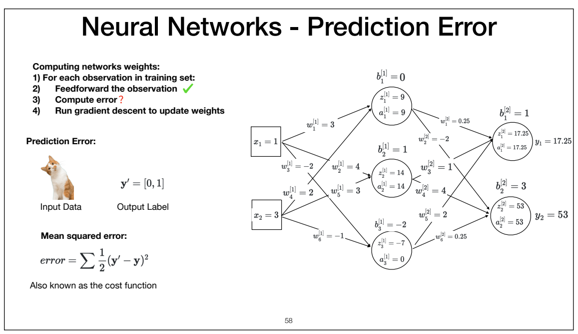
55



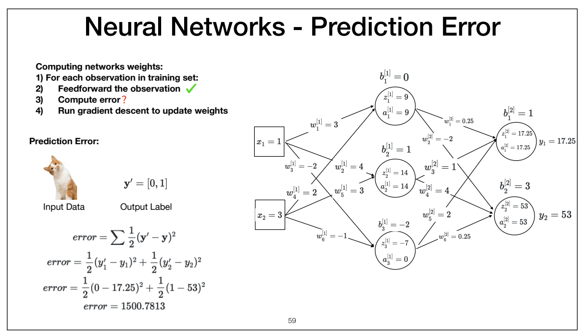
56



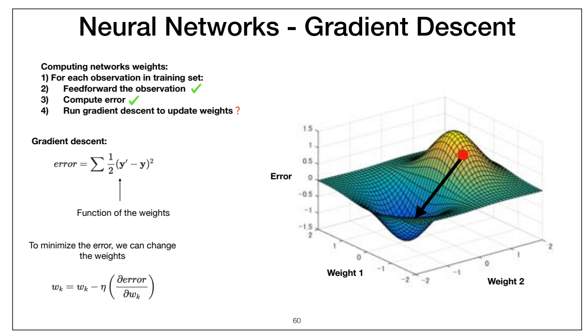
57



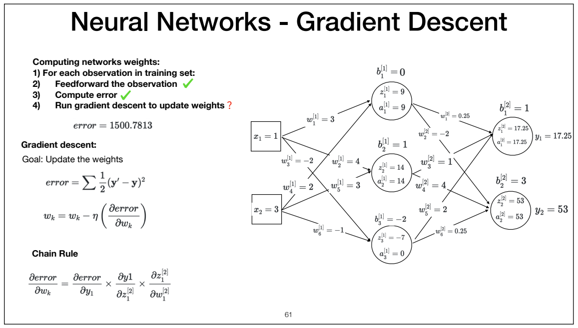
58



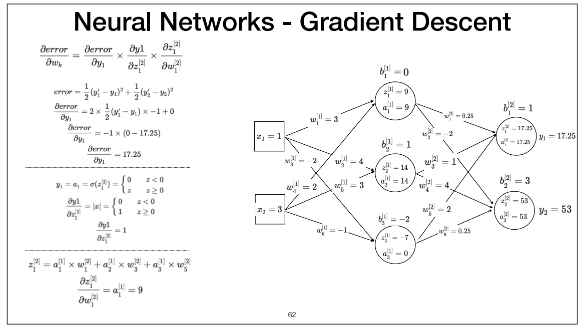
59



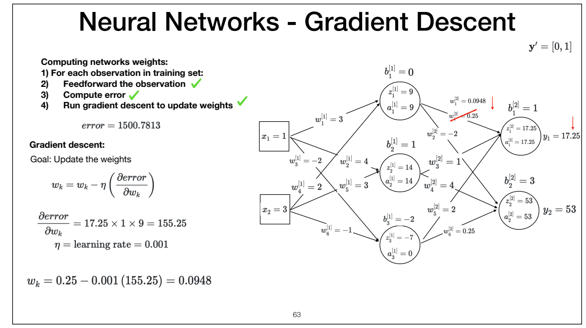
60



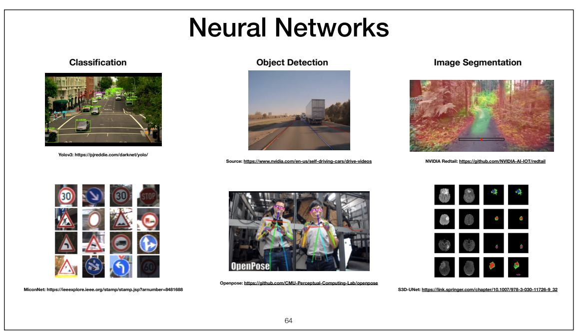
61



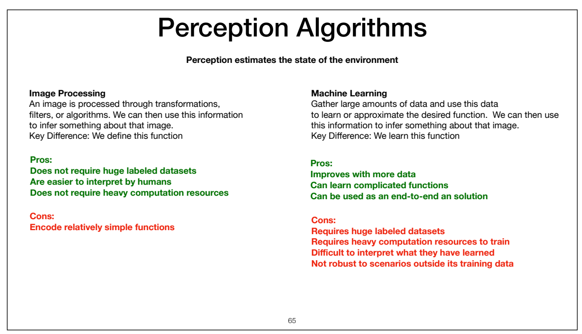
62



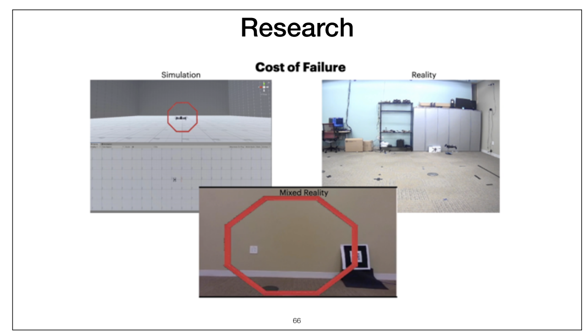
63



64



65



66