

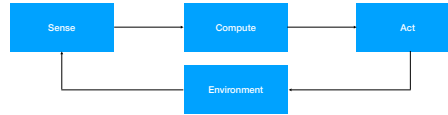
Perception

CS4501 - Robotics for Software Engineers

By Carl Hildebrandt

1

Robot Conceptual Architecture



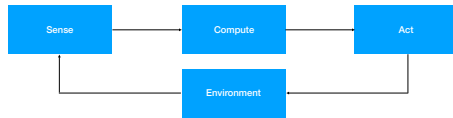
2

Self-driving Case Study



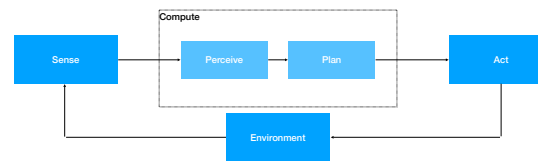
3

Robot Conceptual Architecture



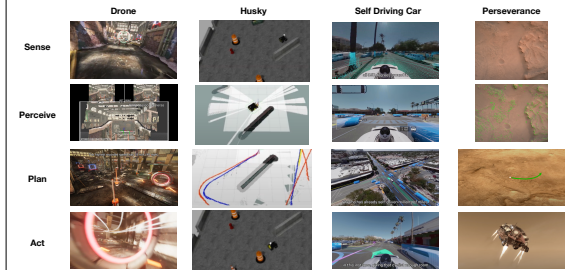
4

Robot Conceptual Architecture

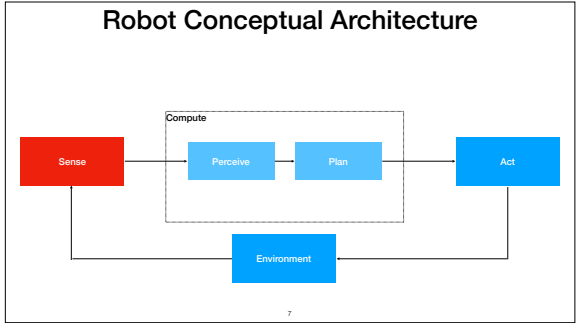


5

Robot Conceptual Architecture



6



7

Image Data

ROS: sensor_msgs/Image

[sensor_msgs/Image Message](#)

File: sensor_msgs/Image.msg

Compact Message Definition

```

std_msgs/Header header
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
  
```

Annotations:

- Header header
- uint32 height
- uint32 width
- string encoding
- uint8 is_bigendian
- uint32 step
- uint8[] data
- RGB / BGR / HSV
- How data is stored

8

Image Data

9

Perception

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

—Perdew, Scott Drew, et al. “Perception, planning, control, and coordination for autonomous vehicles.” *Machined 5.1* (2017)

10

Perception Examples

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

Camera Sensor

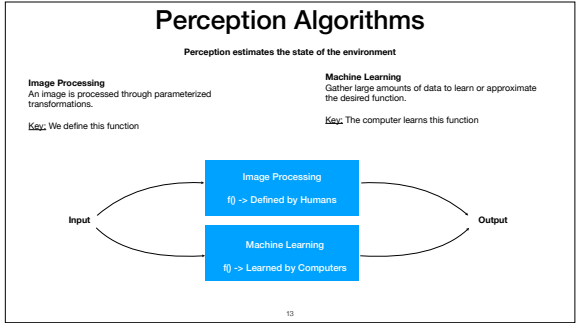
Perception

11

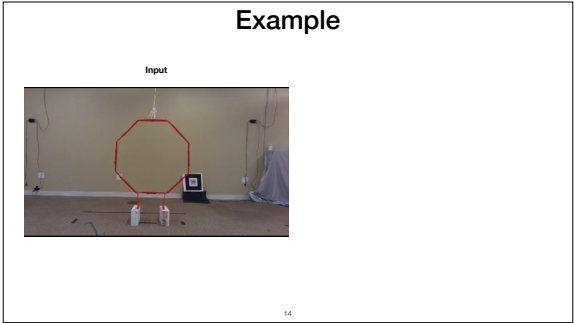
Main Types of Perception

Classification Object Detection Interpretation

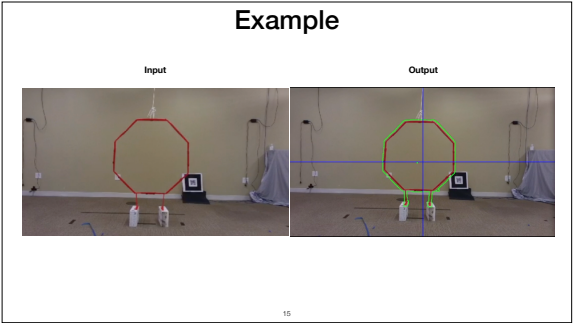
12



13



14



15

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

16

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

17

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 Hue, Saturation, Value, 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 to the to brightest

Code

```

1: # Import cv2
2: # Read the image
3: # Convert the image to HSV
4: # Define the color range
5: # Apply the color filter
6: # Show the result
7: # Wait for a key press
8: # Destroy the window
9: # Exit the program

```

18

Example: Color Filtering

Raw Data

Mask

Output

```

1 # Create a background subtraction object
2 obj = cv2.BackgroundSubtractorMOG2()
3
4 # Load the frame
5 cap = cv2.VideoCapture(0)
6 ret, frame = cap.read()
7
8 # Create the mask
9 mask = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
10
11 # Apply the mask
12 result = cv2.bitwise_and(frame, frame, mask=mask)

```

19

Image Processing Techniques

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

20

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.

```

1 obj = cv2.BackgroundSubtractorMOG2()
2
3 while(cap.isOpened()):
4     ret, frame = cap.read()
5
6     # Create the mask
7     mask = obj.apply(frame)
8
9     # Multiply mask to original with image
10    result = cv2.bitwise_and(frame, frame, mask=mask)

```

21

Example: Background Subtraction

Raw Data

Mask

Output

```

1 obj = cv2.BackgroundSubtractorMOG2()
2
3 while(cap.isOpened()):
4     ret, frame = cap.read()
5
6     # Create the mask
7     mask = obj.apply(frame)
8
9     # Multiply mask to original with image
10    result = cv2.bitwise_and(frame, frame, mask=mask)

```

22

Image Processing Techniques

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

23

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

24

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

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

0	0	0
0	1	0
0	0	0

=

Input

Kernel 3X3

Output

25

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

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

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

Input

Kernel 3X3

Output

26

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

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

Code

```

22 # Blur image using normalized filter kernel
23 blur1 = cv2.blur(frame, (3,3))
24 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 x 3 (approximate)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 x 5 (approximate)	$\frac{1}{224} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

27

Example: Blurring

Raw Data
3x3 Kernel
25x25 Kernel

```

22 # Blur image using normalized filter kernel
23 blur1 = cv2.blur(frame, (3,3))
24 blur2 = cv2.blur(frame, (25,25))
  
```

28

Image Processing Techniques

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

29

(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)

$$L_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} L_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} L_z$$

Code

```

12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
  
```

Original Image

Canny Edge Detection: <https://www.kdnuggets.com/2017/01/canny-edge-detection.html>

30

(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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Gaussian Filter



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Gradient Magnitude



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Non Max Suppression



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Double Thresholding



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Edge Tracking



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

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

```
12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame, (5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)
```

Original Image **Gaussian Filter** **Gradient Magnitude**



Non Max Suppression **Double Thresholding** **Edge Tracking**



Canny Edge Detector: Edge Filter: Gaussian Filter: Sobel Operator: Non Max Suppression: Double Thresholding: Edge Tracking

36

Example: Edge Detection

Raw Data

Edge Detection

5x5 Blur -> Edge Detection

```

12 # Find the edges
13 edges = cv2.Canny(frame, 100, 200)
14
15 # Find the edges
16 blur = cv2.blur(frame,(5,5))
17 edges_blur = cv2.Canny(blur, 100, 200)

```

37

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

38

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

39

Perception Algorithms

Input

output = f(input)

Output

Input

Input

Input

40

Machine Learning

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

Input

output = f(input)

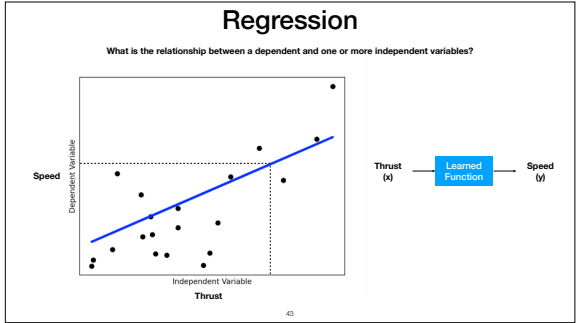
Output

41

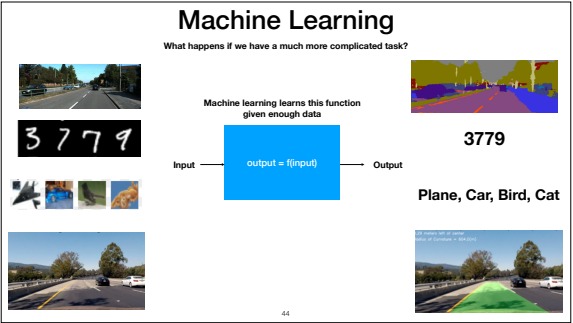
Regression

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

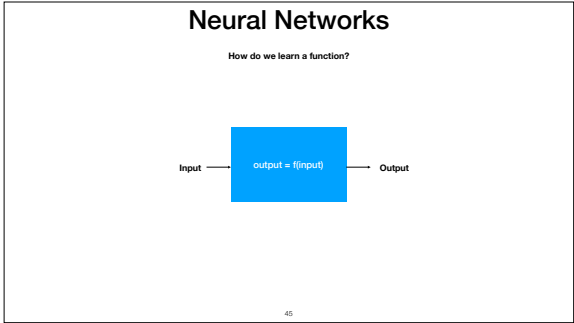
42



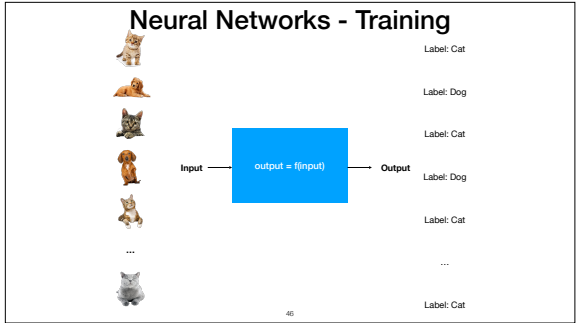
43



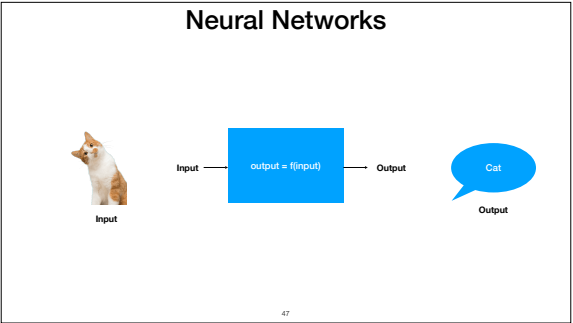
44



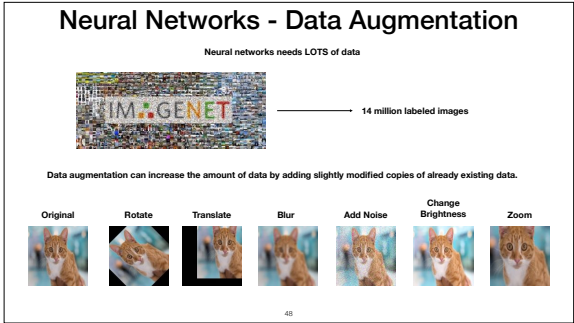
45



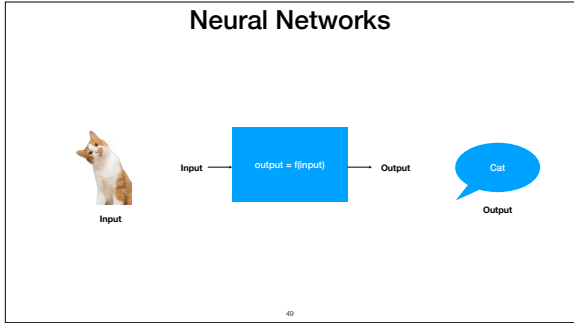
46



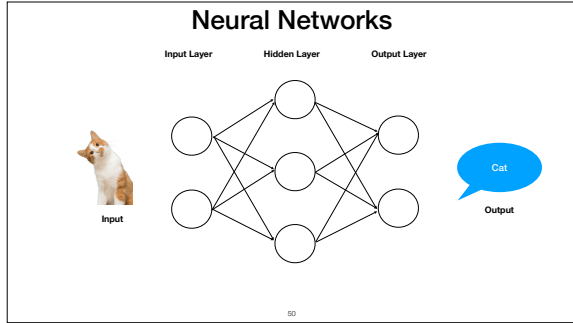
47



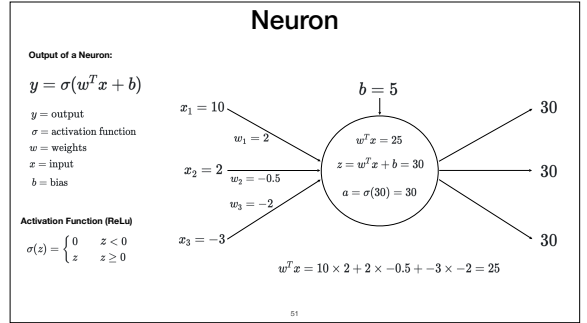
48



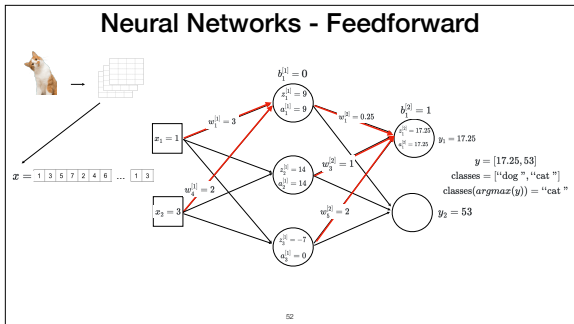
49



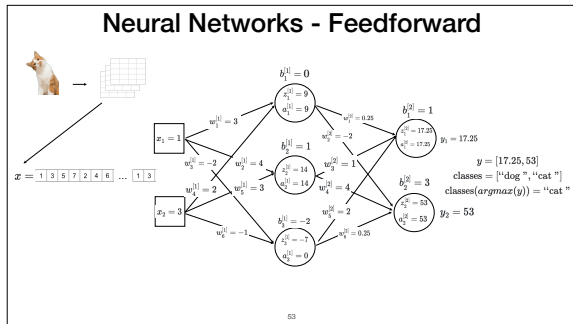
50



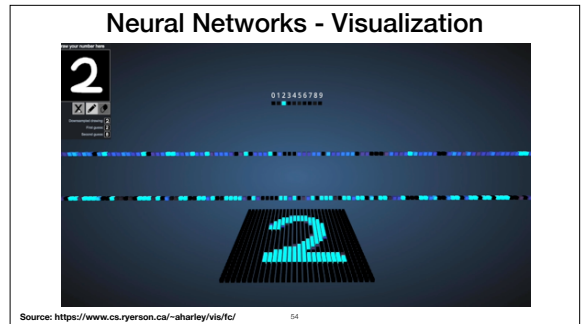
51



52



53



54

Neural Networks - Structure

Activation functions

Sigmoid
 $\sigma(x) = \frac{1}{1 + e^{-x}}$

Leaky ReLU
 $\max(0, x)$

tanh
 $\tanh(x)$

ReLU
 $\max(0, x)$

ELU
 $\max(0, x)$
 $\min(0, e^{-x} - 1)$

Linear
 $y = x$

Learning Parameters:

- Learning rate
- Optimizer
- Batch Size
- Early stopping
- Number training epochs

55

55

Neural Networks - Weights

How do we compute these weights?

56

56

Neural Networks - Updating Weights

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error
- 4) Run gradient descent to update weights

Input Data: $y^i = [0, 1]$

Output Label: $y_i = 17.25$

57

57

Neural Networks - Prediction Error

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error ✓
- 4) Run gradient descent to update weights

Prediction Error:

Mean squared error:

$$error = \sum \frac{1}{2} (y^i - y)^2$$

Also known as the cost function

58

58

Neural Networks - Prediction Error

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error ✓
- 4) Run gradient descent to update weights

Prediction Error:

$$error = \sum \frac{1}{2} (y^i - y)^2$$

$$error = \frac{1}{2} (0 - 17.25)^2 + \frac{1}{2} (1 - 53)^2$$

$$error = 1500.7813$$

59

59

Neural Networks - Gradient Descent

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error ✓
- 4) Run gradient descent to update weights ?

Gradient descent:

$$error = \sum \frac{1}{2} (y^i - y)^2$$

Function of the weights

To minimize the error, we can change the weights

$$w_k = w_k - \eta \left(\frac{\partial error}{\partial w_k} \right)$$

60

60

Neural Networks - Gradient Descent

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error ✓
- 4) Run gradient descent to update weights ?

error = 1500.7813

Gradient descent:
Goal: Update the weights

$$error = \sum \frac{1}{2} (y' - y)^2$$

$$w_k = w_k - \eta \left(\frac{\partial error}{\partial w_k} \right)$$

Chain Rule

$$\frac{\partial error}{\partial w_k} = \frac{\partial error}{\partial y_1} \times \frac{\partial y_1}{\partial z_1^{[1]}} \times \frac{\partial z_1^{[1]}}{\partial w_k^{[1]}}$$

61

Neural Networks - Gradient Descent

$$\frac{\partial error}{\partial w_k} = \frac{\partial error}{\partial y_1} \times \frac{\partial y_1}{\partial z_1^{[1]}} \times \frac{\partial z_1^{[1]}}{\partial w_k^{[1]}}$$

$$error = \frac{1}{2} (y_1' - y_1)^2 + \frac{1}{2} (y_2' - y_2)^2$$

$$\frac{\partial error}{\partial y_1} = 2 \times \frac{1}{2} (y_1' - y_1) \times -1 = 0$$

$$\frac{\partial error}{\partial y_2} = -1 \times (0 - 17.25)$$

$$\frac{\partial error}{\partial w_1} = 17.25$$

$$y_1 = a_1 = \sigma(z_1^{[1]}) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

$$\frac{\partial z_1^{[1]}}{\partial w_1^{[1]}} = |z| = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

$$\frac{\partial y_1}{\partial z_1^{[1]}} = 1$$

$$z_1^{[2]} = a_1^{[1]} \times w_1^{[2]} + a_2^{[1]} \times w_2^{[2]} + a_3^{[1]} \times w_3^{[2]}$$

$$\frac{\partial z_1^{[2]}}{\partial w_1^{[2]}} = a_1^{[1]} = 9$$

62

Neural Networks - Gradient Descent

Computing networks weights:

- 1) For each observation in training set:
- 2) Feedforward the observation ✓
- 3) Compute error ✓
- 4) Run gradient descent to update weights ✓

error = 1500.7813

Gradient descent:
Goal: Update the weights

$$w_k = w_k - \eta \left(\frac{\partial error}{\partial w_k} \right)$$

$$\frac{\partial error}{\partial w_k} = 17.25 \times 1 \times 9 = 155.25$$

$$\eta = \text{learning rate} = 0.001$$

$$w_k = 0.25 - 0.001 (155.25) = 0.0948$$

63

Neural Networks

Classification

Source: https://www.shutterstock.com/stock-photo/1204878

Object Detection

Source: https://www.shutterstock.com/stock-photo/1204878

Image Segmentation

Source: https://www.shutterstock.com/stock-photo/1204878

Source: https://www.shutterstock.com/stock-photo/1204878

Source: https://www.shutterstock.com/stock-photo/1204878

64

Perception Algorithms

Perception estimates the state of the environment

Image Processing

An image is processed through transformations, filters, or algorithms. We can then use this information to infer something about that image.
Key Difference: We define this function

Pros:

- Does not require huge labeled datasets
- Are easier to interpret by humans
- Does not require heavy computation resources

Cons:

- Encode relatively simple functions

Machine Learning

Gather large amounts of data and use this data to learn or approximate the desired function. We can then use this information to infer something about that image.
Key Difference: We learn this function

Pros:

- Improves with more data
- Can learn complicated functions
- Can be used as an end-to-end solution

Cons:

- Requires huge labeled datasets
- Requires heavy computation resources to train
- Difficult to interpret what they have learned
- Not robust to scenarios outside its training data

65

Research

Cost of Failure

Simulation

Reality

Mixed Reality

66