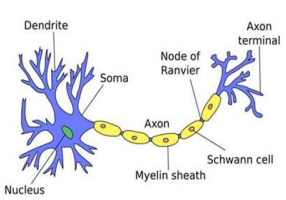
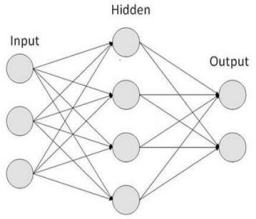
Northeastern University
Electrical & Computer Engineering



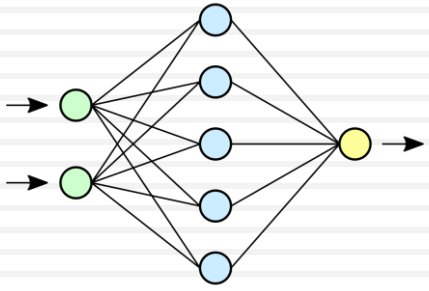


Machine Learning
Introduction to Neural Networks

Northeastern University
Department of Electrical & Computer Engineering – Data Science

Fall 2023

Instructor: Dr. Fatema Nafa



Inclusion and Diversity: A Guiding Statement for Our Classroom

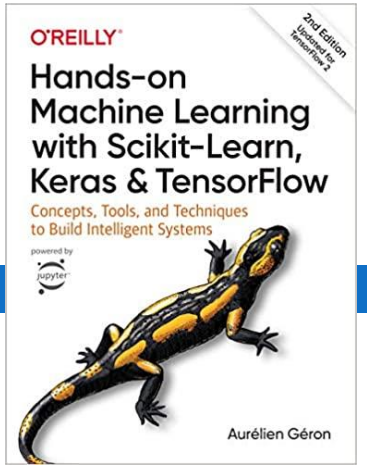
- **Expectations from Dr. Nafa**
  - Our class today will be a place where all students are valued and respected, and where diverse perspectives and experiences are encouraged.
  - I will make every effort to accommodate different teaching styles and clarify any confusion you may have about the topic.
- **Expectations from Students:**
  - I expect all students to contribute to this effort by behaving in a respectful and inclusive manner.
  - It is encouraged that you take notes during lectures as you listen attentively.
  - Ask any questions that you may have because it is your time to learn before you leaf the class.
  - Help me to in maintaining an engaging lecture by making an effort to respond to my questions to the best of your abilities
  - If you have any specific needs or concerns, please don't hesitate to let me know by sending an email to me at fnafa@salemstate.edu.
  - I appreciate students volunteering to provide answers to my questions.

Required

Prerequisite(s)

- A foundation in **Machine Learning**, and
- **Calculus(*Logistic Regression*).**
- **Python** programming is necessary(*familiarity with Jupyter Notebook, and some libraries NumPy, matplotlib, and sklearn*). While you don't need to be highly proficient in multiple programming languages, having some background in
- **Willingness to learn**

Resources



- **Required Resources(s)**
  - Aurèlien Gèron. **Hands on Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to build Intelligent Systems.** 2nd edition. O'Reilly Publications, 2019. By: ***Aurélien Géron***
- **Recommended Resources(s)**
  - https://github.com/thiagordp/machine-learning-books
  - Mark E. Fenner. *Machine Learning with Python for Everyone*. Addison Wesley Publications, 2019.
  - Wei-Meng Lee. *Python Machine Learning. Wiley Publications 2019*.
  - https://colab.google/
  - https://streamlit.io/
  - https://superset.apache.org/docs/creating-charts-dashboards/creating-your-first-dashboard/

Part II. Neural Networks and Deep Learning

Fun Learning

- **The Technique**
  - Each student will have an assigned number. When I ask a question, if the **student called upon doesn't know the answer**, they can pass the question to another student by calling out a random number.
  - This encourages everyone to stay engaged and ready to answer.
- **Assign Numbers**
  - Assign unique numbers to each student in the class. I will make sure every student knows their assigned number.

Standards used in this lecture

This component indicates that background information has been taken.

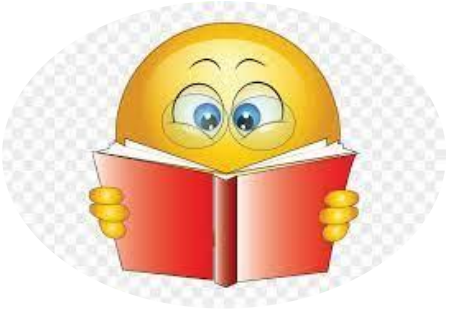
This component indicates whether you have any questions for me

- This component indicates a request for your participation


Any
Questions

Lecture Outcomes

- **Define** what a neural network is and its basic components.
- **Explain** how neural networks are inspired by biological neurons.
- **Understand** the fundamental concept of activation functions.
- **Explain** the architecture of a typical feedforward neural network (input layer, hidden layers, output layer).
- **Describe** the purpose of each layer in a neural network.
- **Discuss** the concept of neural networks and their advantages.

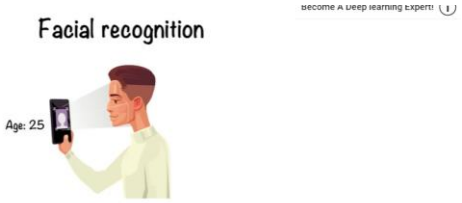


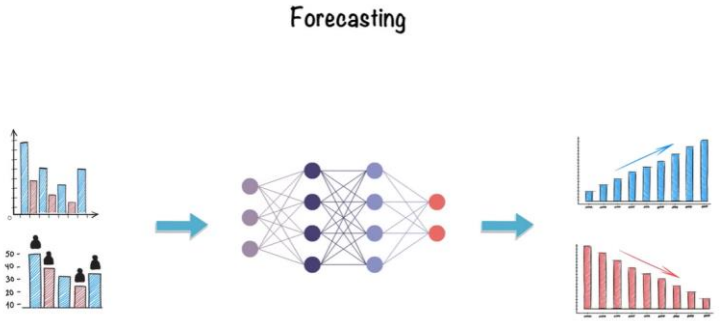
Lecture Outline

1. Introduction to Machine Learning
2. Motivating Example
3. Fundamentals of Neural Networks
   - Definition and Structure: Building Blocks
   - Reviewing Logistic Regression
   - Activation Functions: Sparking Neurons
   - Feedforward Propagation: Information Flow
4. Learning and Training
   - Loss Functions: Measuring Errors
   - Gradient Descent
   - Backpropagation: Fine-Tuning Weights
5. Hands-on Exercises
6. In Class work
7. Take Home Points
8. References

Introduction to Machine Learning

Machine Learning to cut down Email spam





Introduction to Machine Learning

General definition

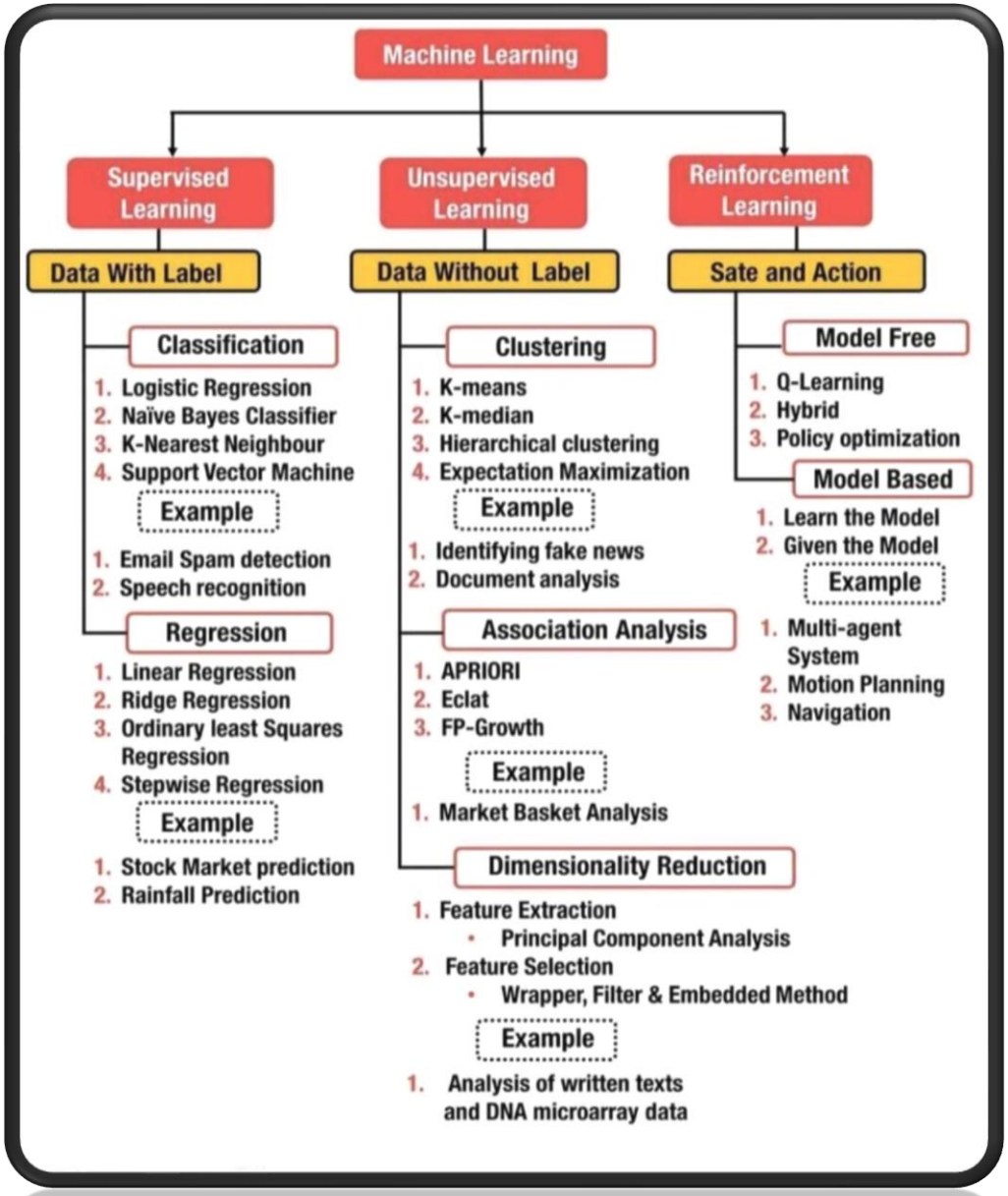
- Machine Learning is the science (and art) of programming computers so they can learn from data.

Here is a slightly more general definition:

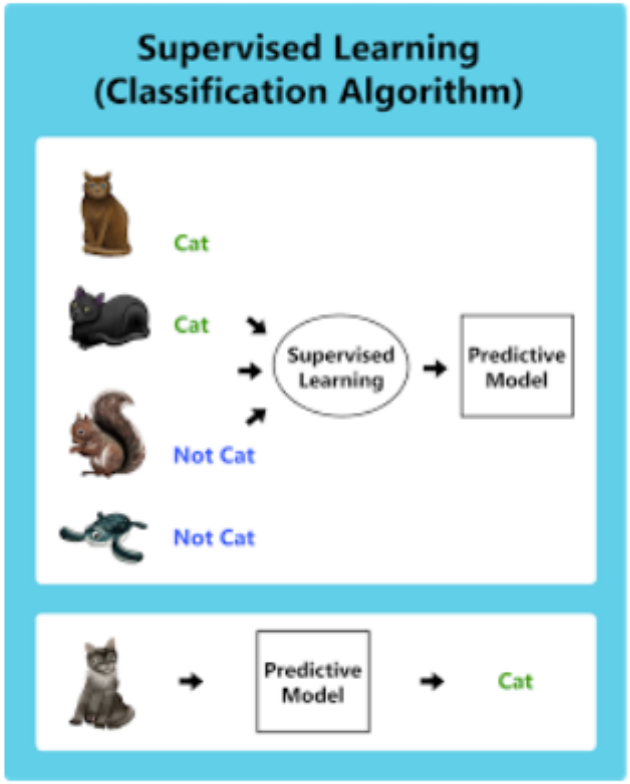
- [Machine Learning is the] **field of study that gives computers the ability to learn without being explicitly programmed**. Arthur Samuel, 1959

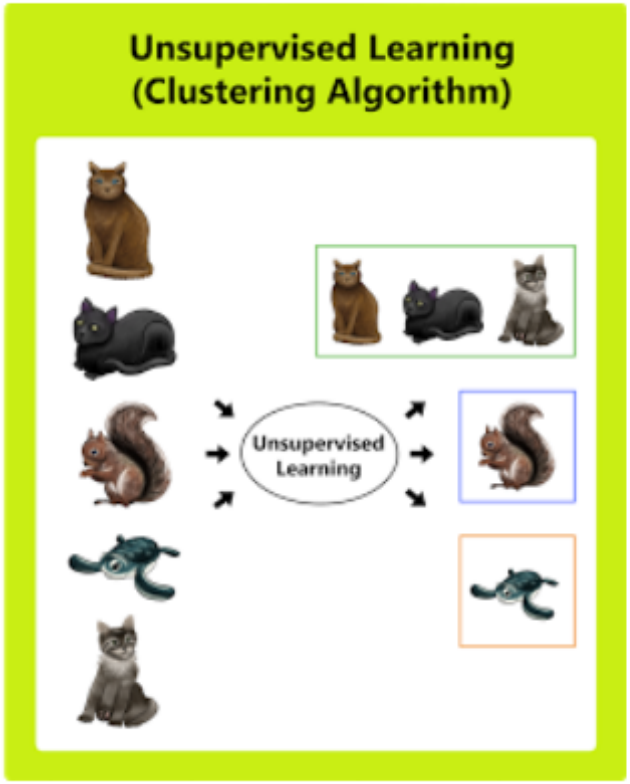
And a more engineering-oriented one:

- A **computer program is said to learn from experience** E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. Tom Mitchell, 1997

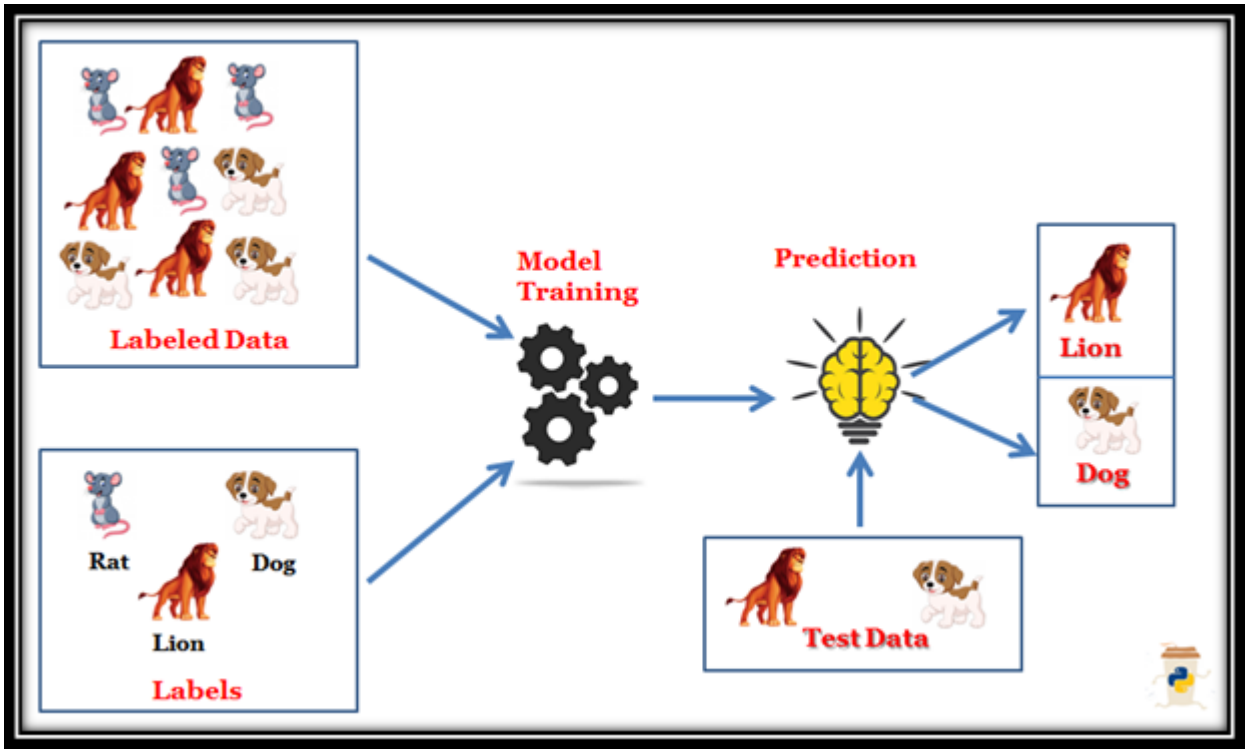

Machine Learning
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Data With Label
Data Without Label
Sate and Action
Classification
1. Logistic Regression
2. Naïve Bayes Classifier
3. K-Nearest Neighbour
4. Support Vector Machine
Example
1. Email Spam detection
2. Speech recognition
Regression
1. Linear Regression
2. Ridge Regression
3. Ordinary least Squares Regression
4. Stepwise Regression
Example
1. Stock Market prediction
2. Rainfall Prediction
Clustering
1. K-means
2. K-median
3. Hierarchical clustering
4. Expectation Maximization
Example
1. Identifying fake news
2. Document analysis
Association Analysis
1. APRIORI
2. Eclat
3. FP-Growth
Example
1. Market Basket Analysis
Dimensionality Reduction
1. Feature Extraction
• Principal Component Analysis
2. Feature Selection
• Wrapper, Filter & Embedded Method
Example
1. Analysis of written texts and DNA microarray data
Model Free
1. Q-Learning
2. Hybrid
3. Policy optimization
Model Based
1. Learn the Model
2. Given the Model
Example
1. Multi-agent System
2. Motion Planning
3. Navigation

Supervised Learning Unsupervised Learning

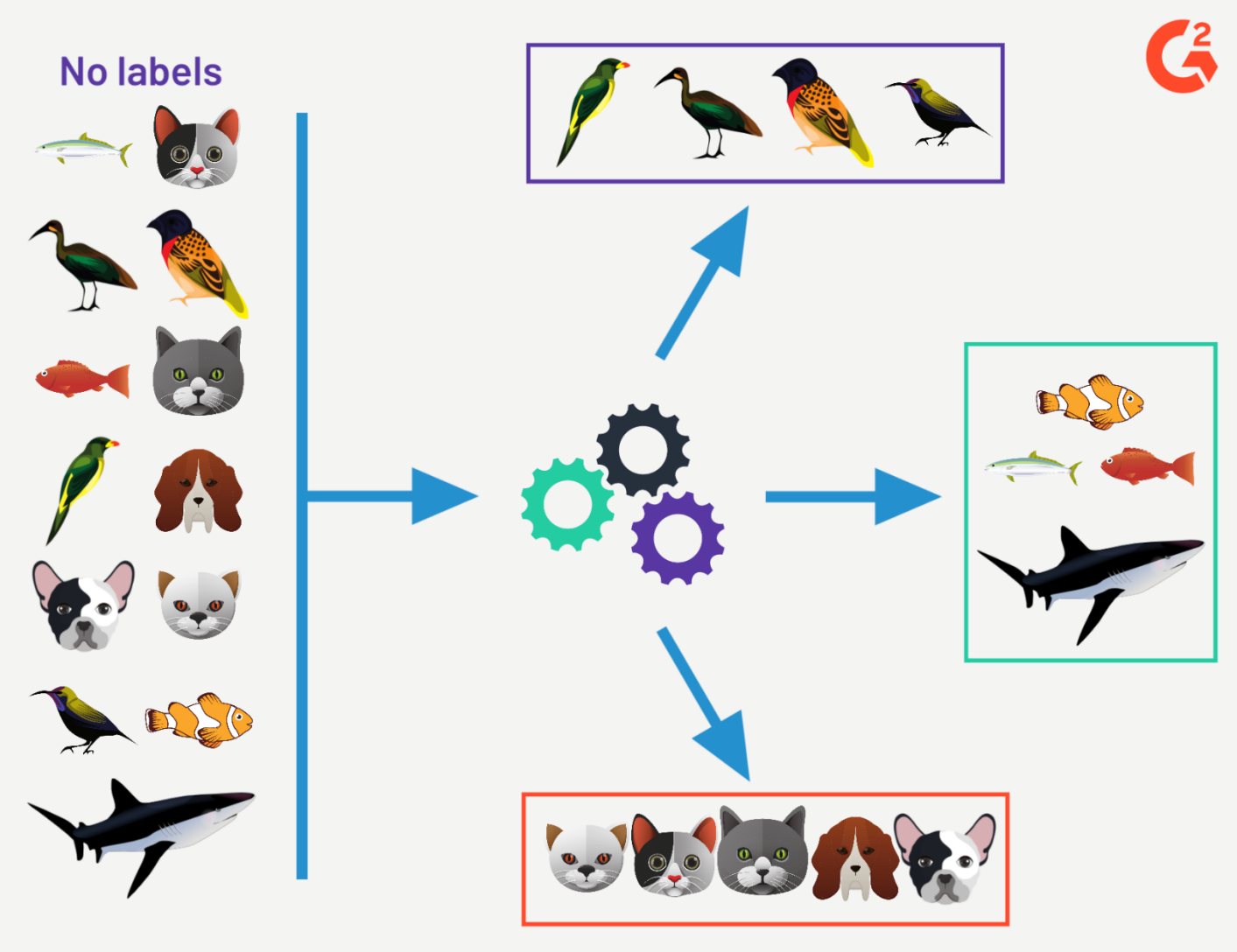


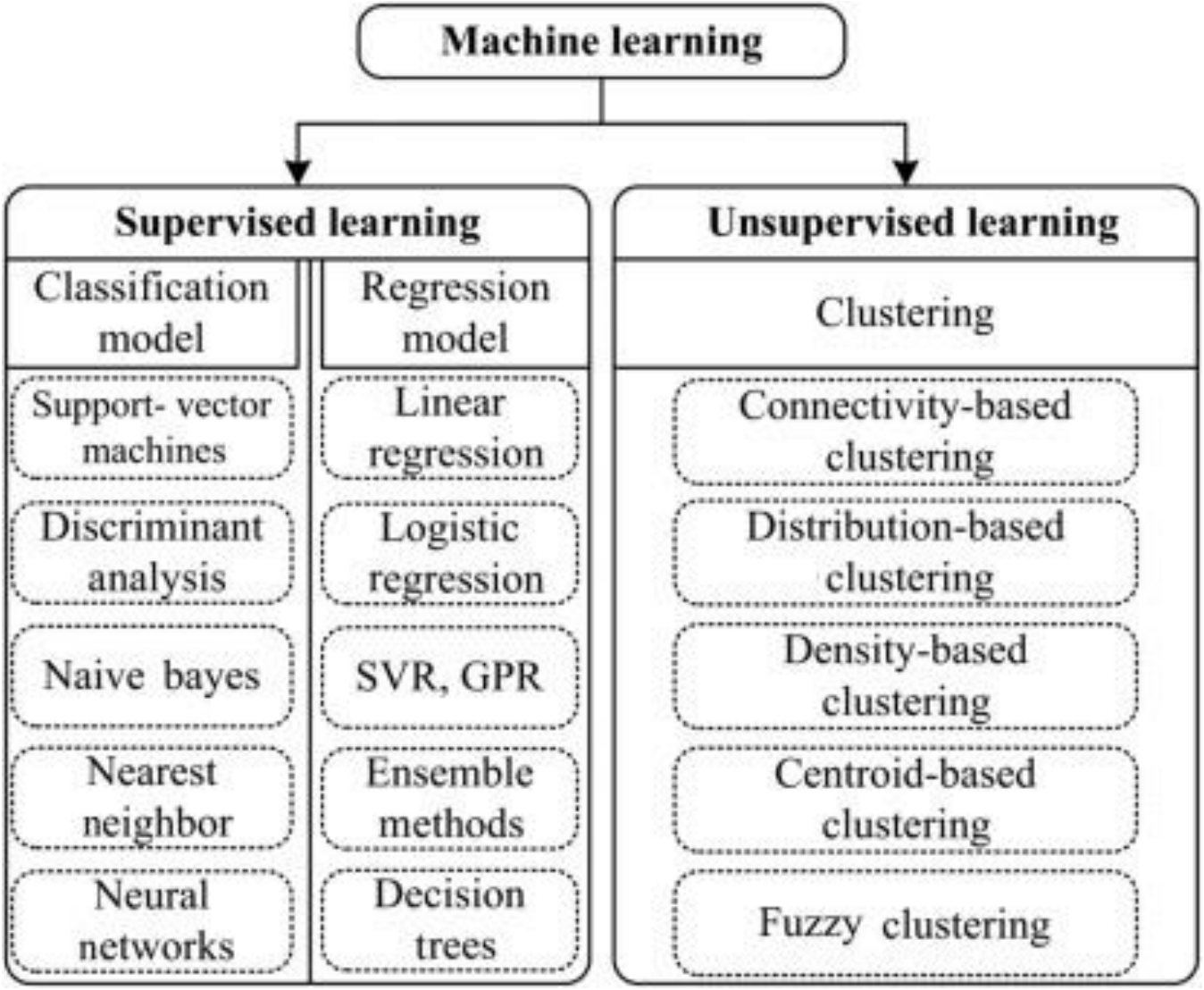


Supervised Learning

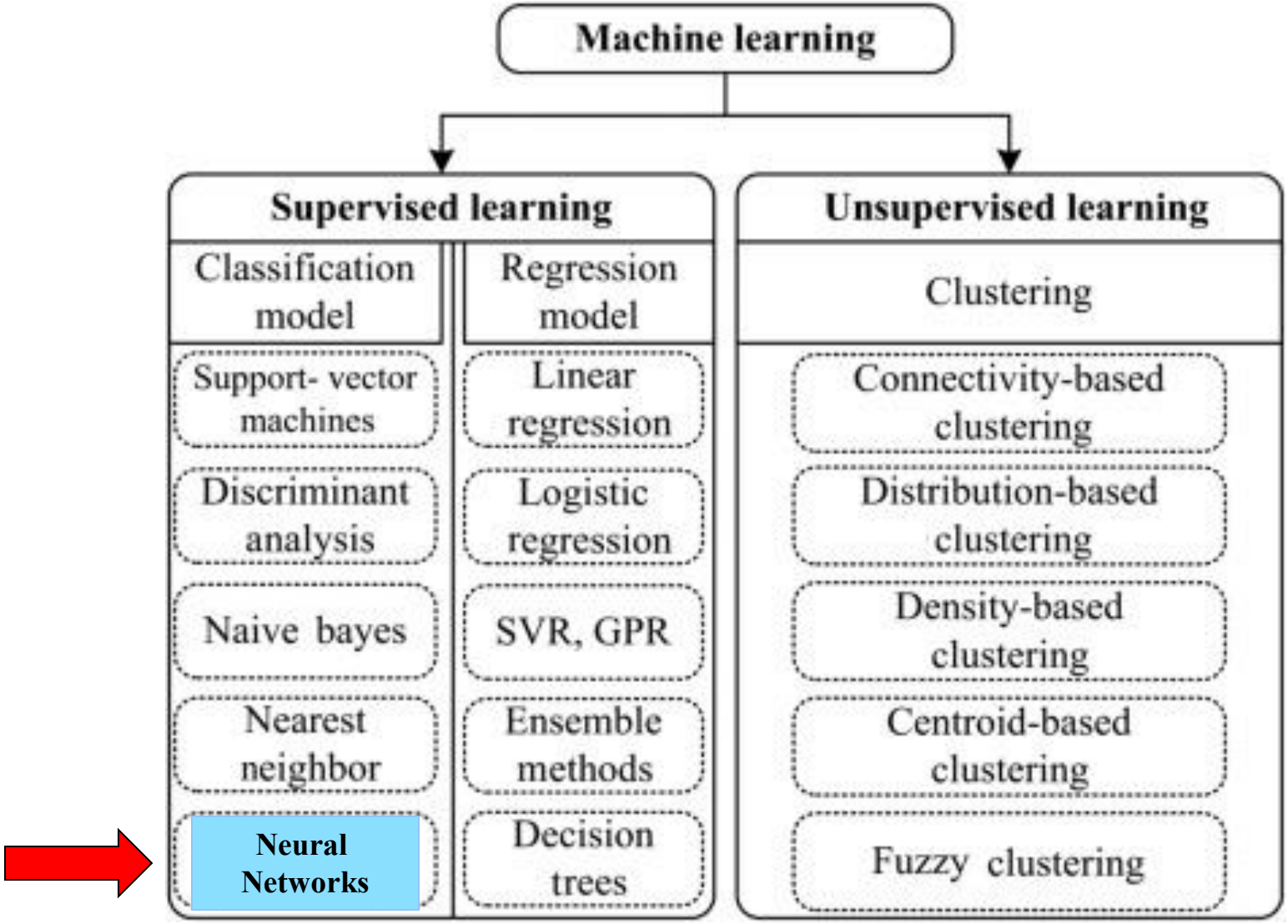


Unsupervised Learning

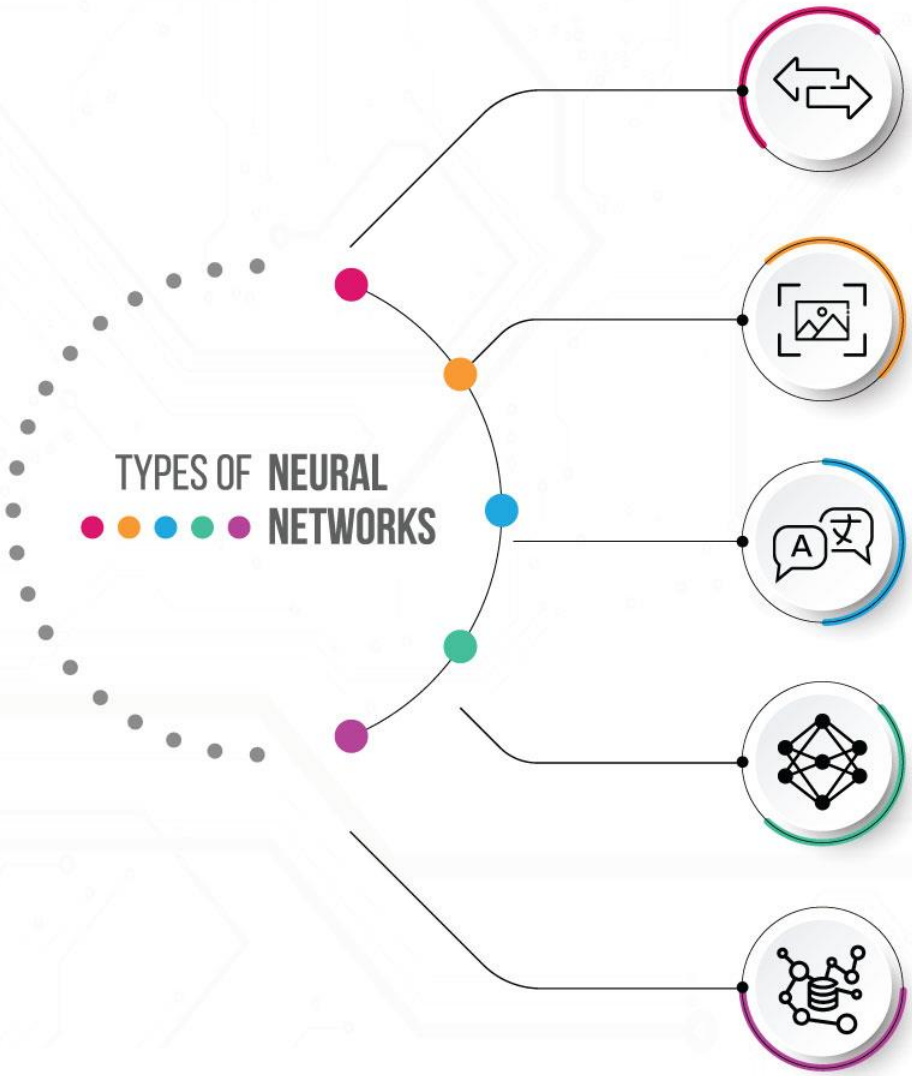



Machine learning
Supervised learning
Classification model
Regression model
Support- vector machines
Linear regression
Discriminant analysis
Logistic regression
Naive bayes
SVR, GPR
Nearest neighbor
Ensemble methods
Neural networks
Decision trees
Unsupervised learning
Clustering
Connectivity-based clustering
Distribution-based clustering
Density-based clustering
Centroid-based clustering
Fuzzy clustering

Northeastern University



Northeastern University



FEEDFORWARD NEURAL NETWORKS

Feedforward neural networks are good at solving problems with a clear relationship between the input and the output, but may not be as effective at figuring out more complex relationships.

CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks are used for tasks that involve data with a grid-like structure, such as image recognition, but may require a large amount of data and be slow.

RECURRENT NEURAL NETWORKS

Recurrent neural networks are used for tasks involving data in a sequence, such as a language translation and speech recognition, but they may need help learning long-term relationships, which can be challenging to train.

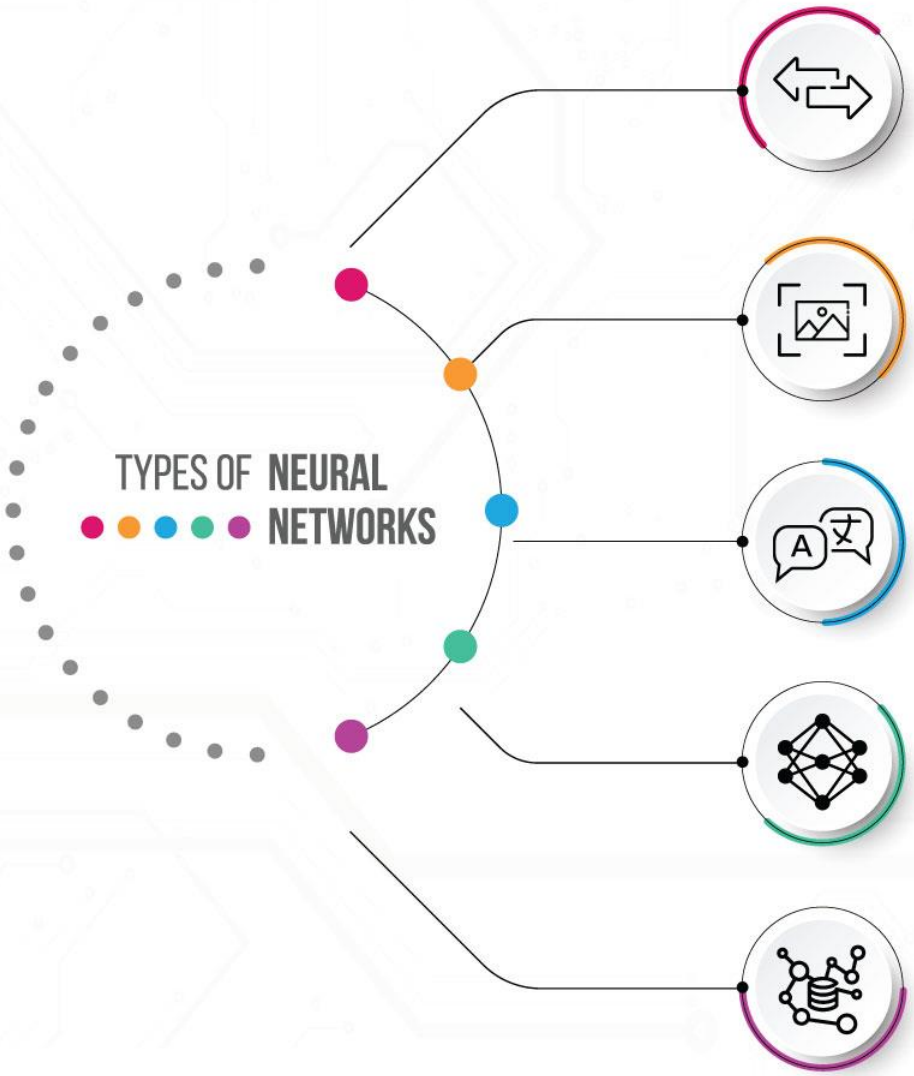
GENERATIVE ADVERSARIAL NETWORKS

Generative adversarial networks are composed of two neural networks that work together to generate synthetic data that appears real but may be challenging to train and require a large amount of data to perform well. They have been used for tasks such as creating realistic images and

AUTOENCODER NEURAL NETWORKS

Autoencoders are used to reduce the complexity of data and learn important features, but they may be sensitive to the settings used and may not always learn meaningful patterns in the data. They have been applied in tasks such as image and speech recognition.

Northeastern University



FEEDFORWARD NEURAL NETWORKS

Feedforward neural networks are good at solving problems with a clear relationship between the input and the output, but may not be as effective at figuring out more complex relationships.

CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks are used for tasks that involve data with a grid-like structure, such as image recognition, but may require a large amount of data and be slow.

RECURRENT NEURAL NETWORKS

Recurrent neural networks are used for tasks involving data in a sequence, such as a language translation and speech recognition, but they may need help learning long-term relationships, which can be challenging to train.

GENERATIVE ADVERSARIAL NETWORKS

Generative adversarial networks are composed of two neural networks that work together to generate synthetic data that appears real but may be challenging to train and require a large amount of data to perform well. They have been used for tasks such as creating realistic images and

AUTOENCODER NEURAL NETWORKS

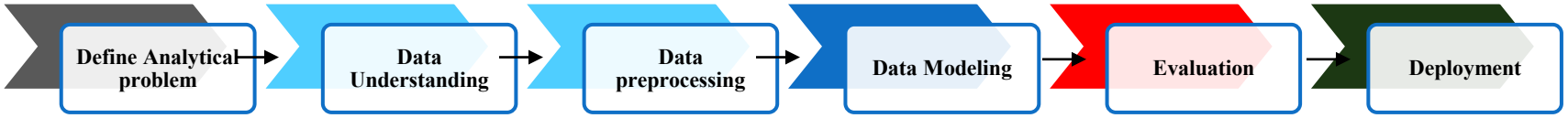
Autoencoders are used to reduce the complexity of data and learn important features, but they may be sensitive to the settings used and may not always learn meaningful patterns in the data. They have been applied in tasks such as image and speech recognition.

1. Imagine you have a dataset with **input features** and corresponding **target outputs(*Label*)**. Which type of learning would you use to **build a predictive model**?

2. If you were given a dataset with only **input features** and **no target outputs(No Label)**, which type of learning could you use to uncover patterns or groupings within the data?

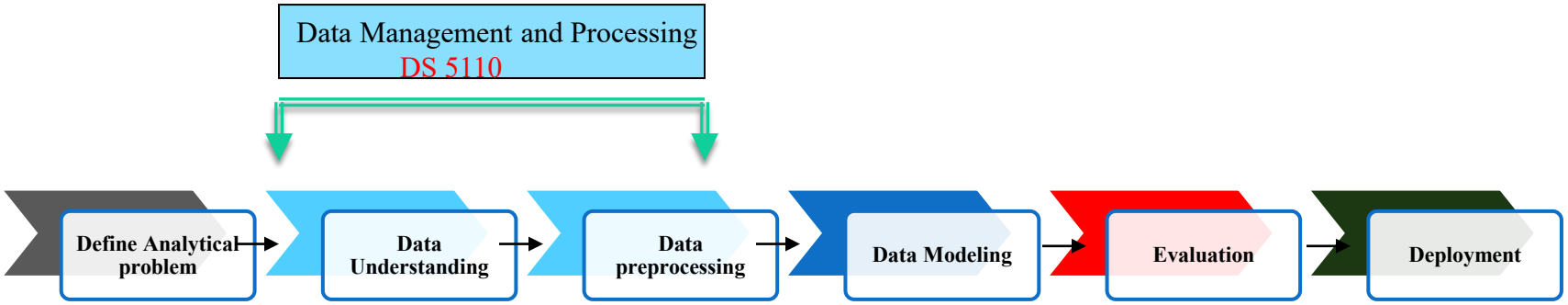
Northeastern University

Overview of Machine Learning Process



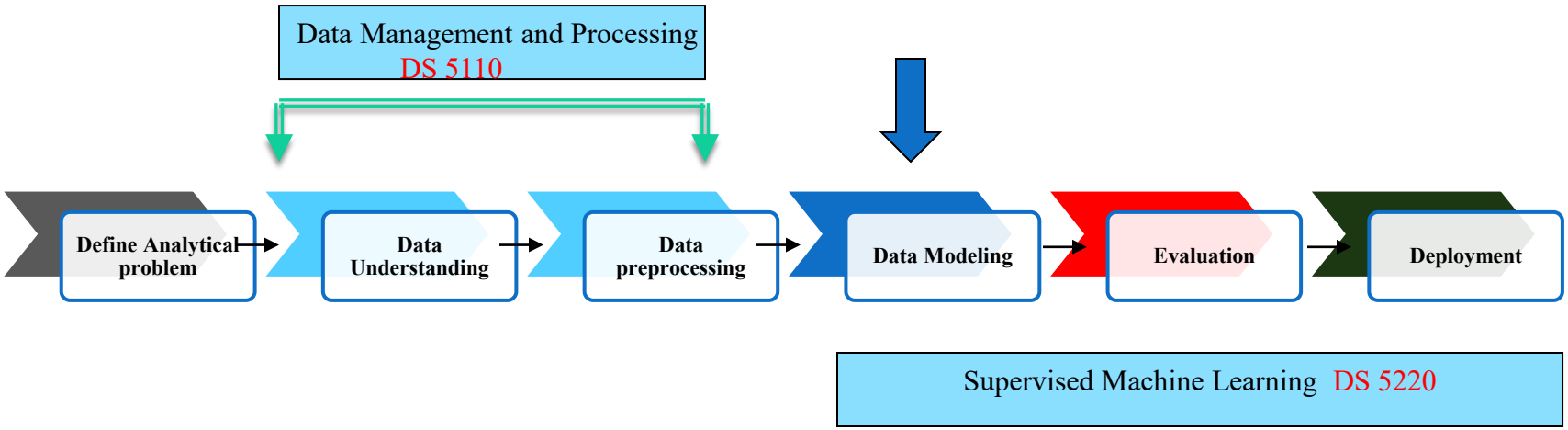
Northeastern University

Overview of Machine Learning Process



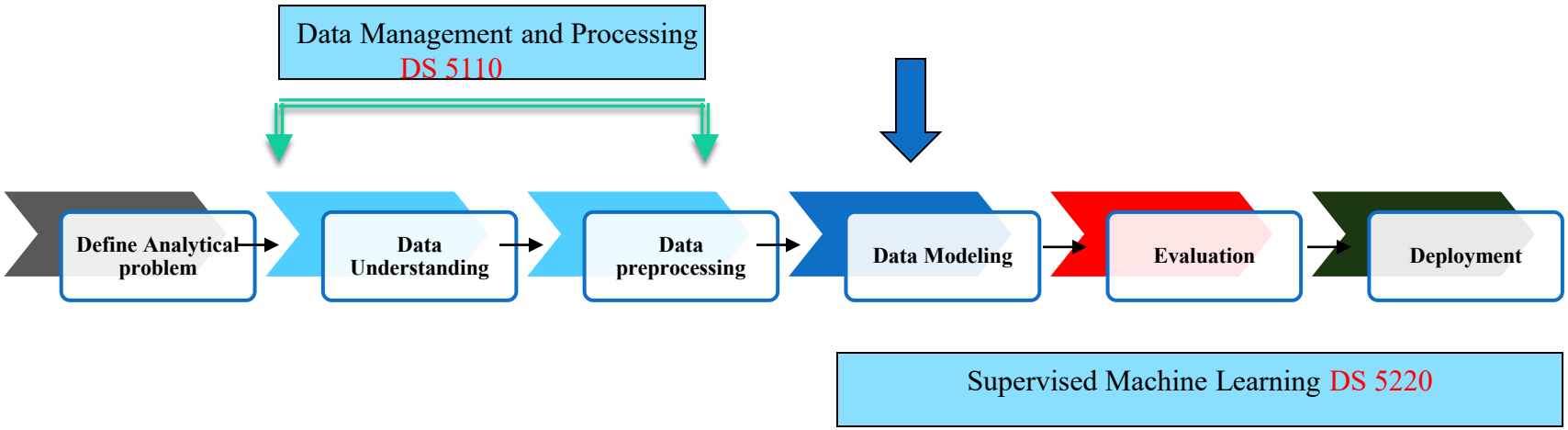
Northeastern University

Overview of Machine Learning Process



Northeastern University

Overview of Machine Learning Process



Northeastern University

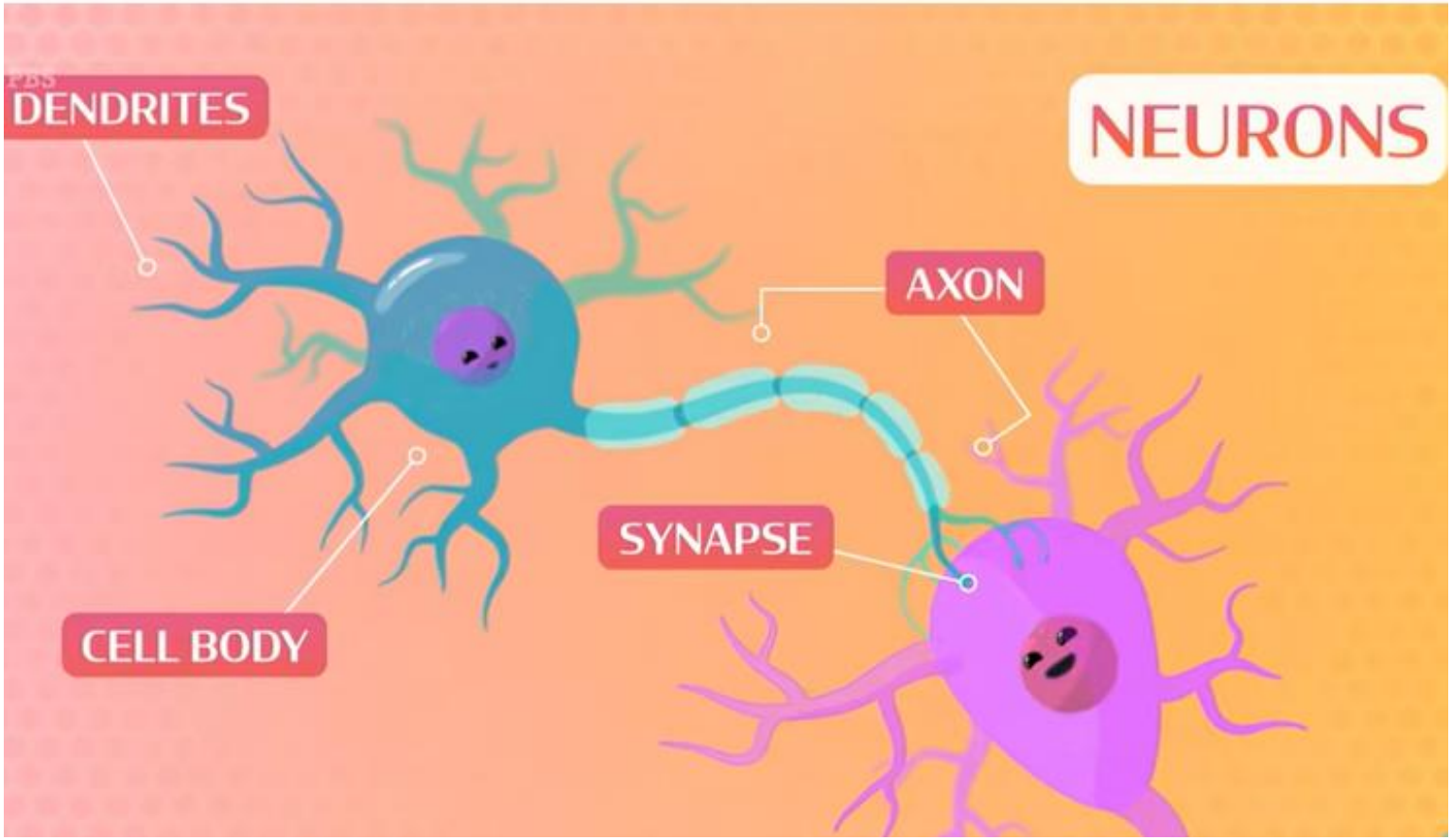
Lecture Outline

1. Introduction to Machine Learning
2. Motivating Example
3. Fundamentals of Neural Networks
   - Definition and Structure: Building Blocks
   - Reviewing Logistic Regression
   - Activation Functions: Sparking Neurons
   - Feedforward Propagation: Information Flow
4. Learning and Training
   - Loss Functions: Measuring Errors
   - Gradient Descent
   - Backpropagation: Fine-Tuning Weights
5. Hands-on Exercises
6. In Class work
7. Take Home Points
8. References


Now, it's your turn to try!


Excellent

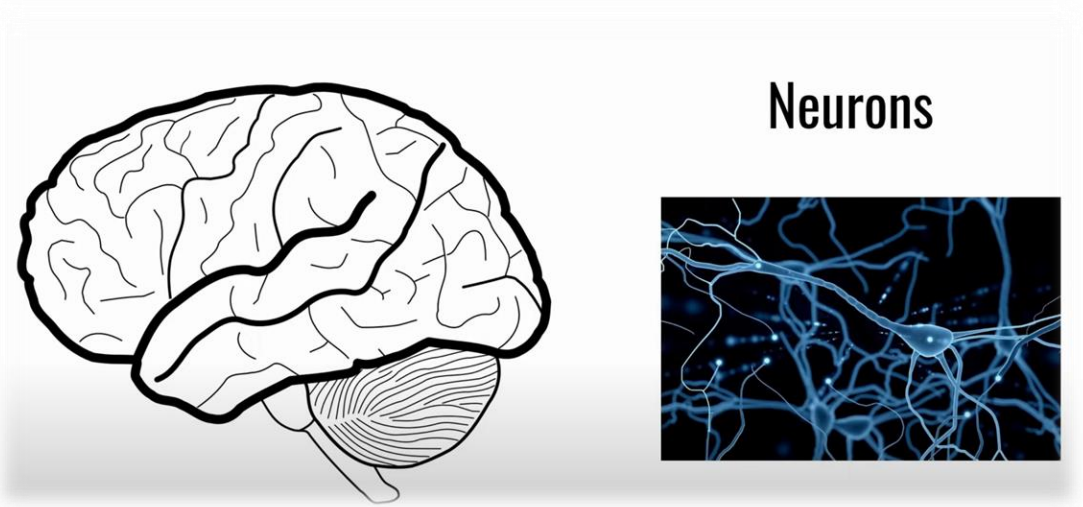
Northeastern University

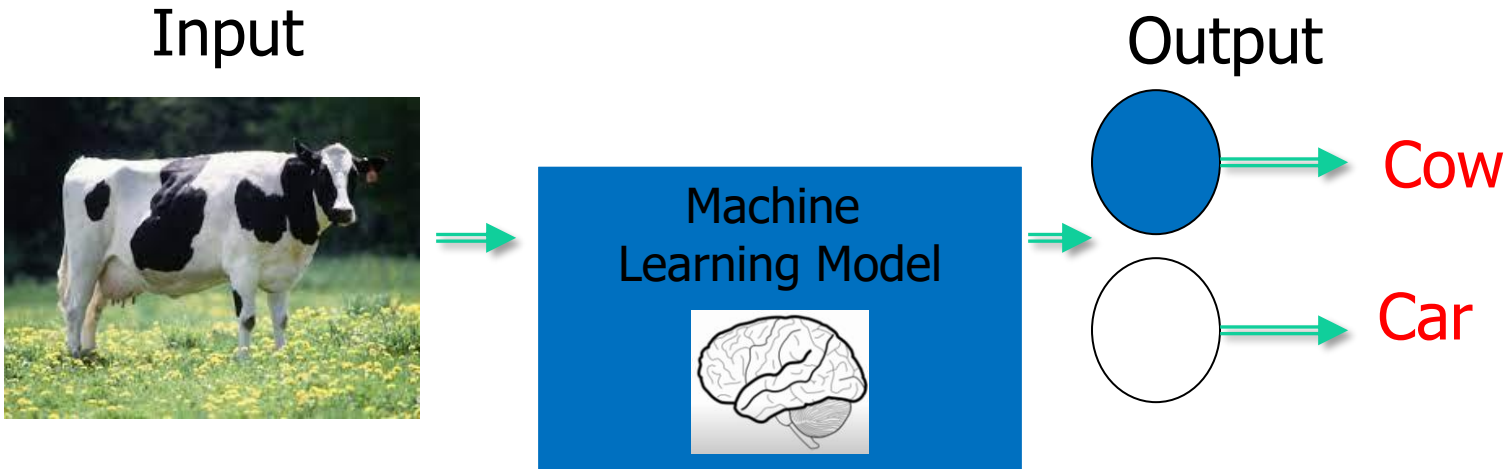

DENDRITES
NEURONS
AXON
SYNAPSE
CELL BODY

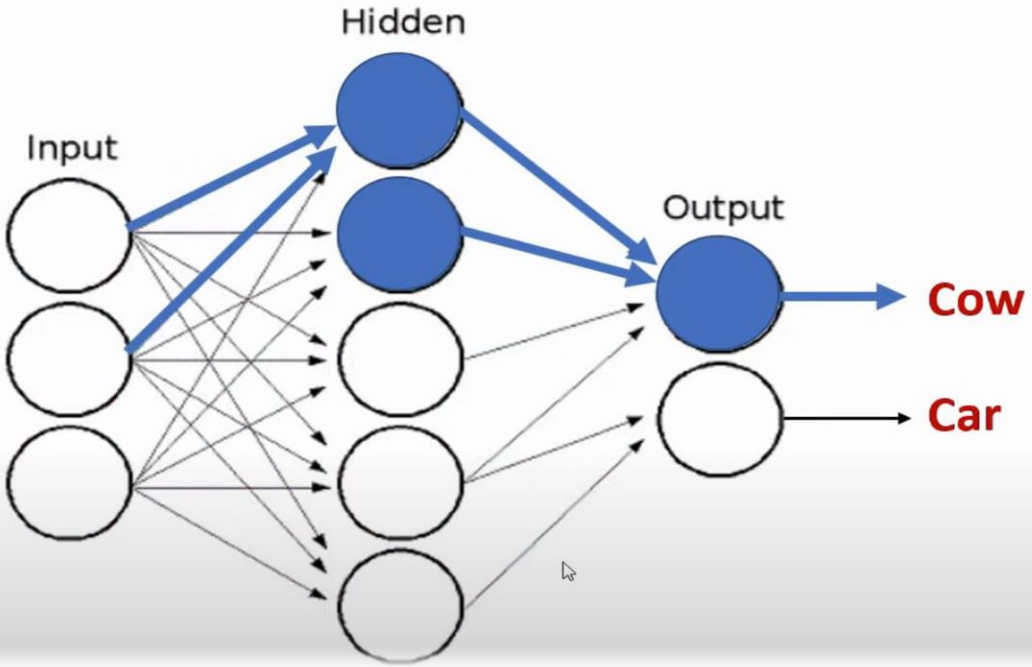
Set of techniques to extract knowledge from available data and use that knowledge to make decisions

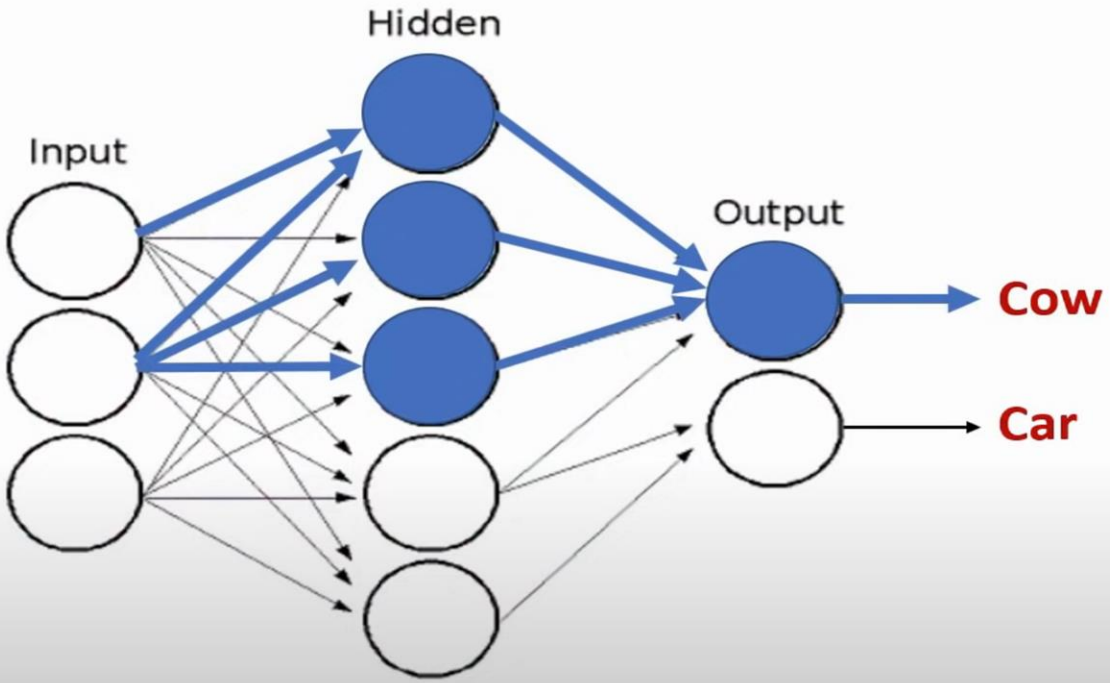
Reference for human biological

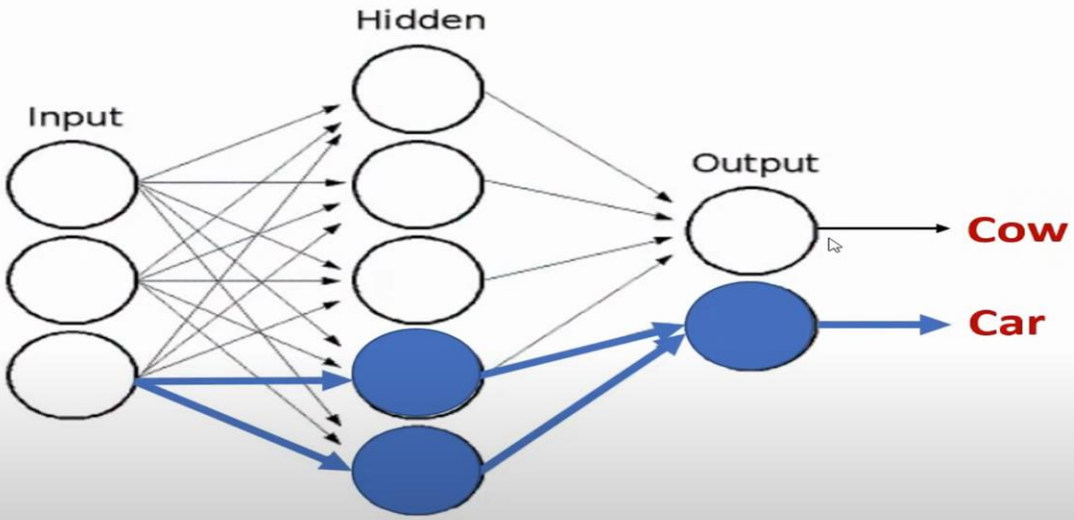
Northeastern University



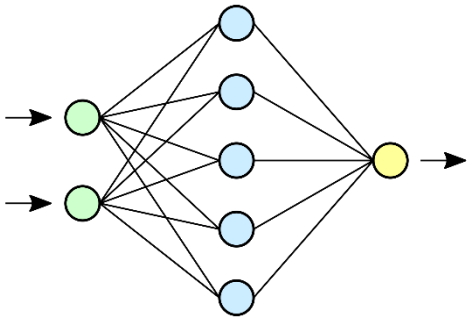


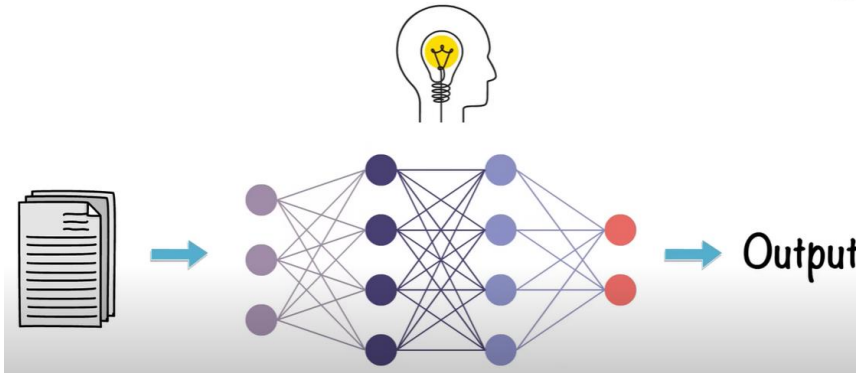

Hidden
Input
Output
Cow
Car


Hidden
Input
Output
Cow
Car


Hidden
Input
Output
Cow
Car

Neural Networks





Plain Vanilla Form

Lecture Outline

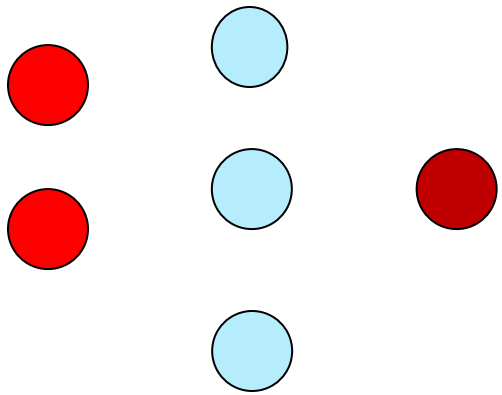
1. Introduction to Machine Learning
2. Fundamentals of Neural Networks
   - Definition and Structure: Building Blocks
   - Motivating Example
   - Activation Functions: Sparking Neurons
   - Feedforward Propagation: Information Flow
3. Learning and Training
   - Loss Functions: Measuring Errors
   - Backpropagation: Fine-Tuning Weights
4. Hands-on Exercises
5. Real-World Applications
6. Summery
7. References

Fundamentals of Neural Networks(NN)

- An artificial neural network (ANN) is a computational model to perform tasks like prediction, classification, decision making, etc.
- It consists of **artificial neurons**. These artificial neurons are a copy of human brain neurons. Neurons in the brain pass the signals to perform the actions. Similarly, artificial neurons connect in a neural network to perform tasks. The connection between the artificial neurons is called **weight**.

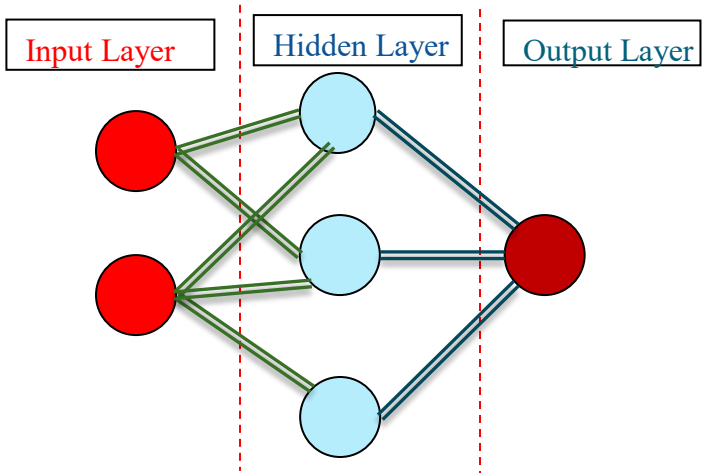
Main component of NN

1. **Neurons (Nodes):** Neurons are the fundamental processing units of a neural network. They receive inputs, perform computations using weights and biases, and produce outputs that are then passed to the next layer. Neurons are inspired by the way biological neurons work in the human brain.



Main component of NN

2. **Layers:** A neural network is organized into layers, which are groups of neurons that process inputs together. There are three main types of layers:
   A. **Input Layer:** The first layer that receives raw input data. The number of neurons in this layer corresponds to the number of input features.
   B. **Hidden Layers:** Intermediate layers between the input and output layers. They process and transform the data through their neurons. Deep neural networks have multiple hidden layers.
   C. **Output Layer:** The final layer that produces the network's output. The number of neurons in this layer depends on the problem type (e.g., regression, classification).



Main component of NN

3. **Neural Network Hyperparameters**
   - There are many hyperparameters to tweak. Not only can you use any imaginable network architecture, but even in a simple NN you can change the number of layers, the number of neurons per layer, the type of activation function to use in each layer, the weight initialization logic, and much more.

- How do you know what combination of hyperparameters is the best for your task?

Main component of NN

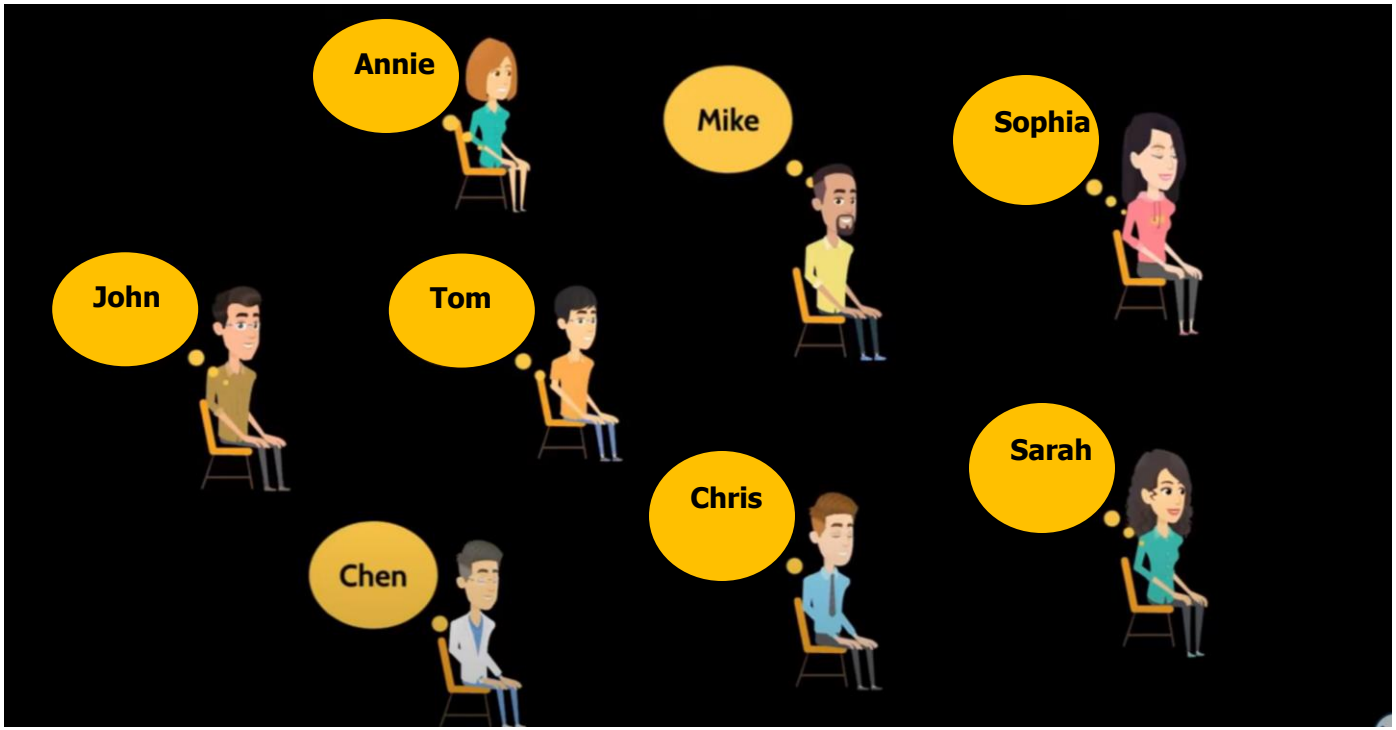
- Number of Neurons per Hidden Layer Obviously the number of neurons in the input and output layers is determined by the type of input and output your task requires.

Main component of NN

- Determining the number of neurons per layer in a neural network, also known as defining the network architecture, is both an art and a science

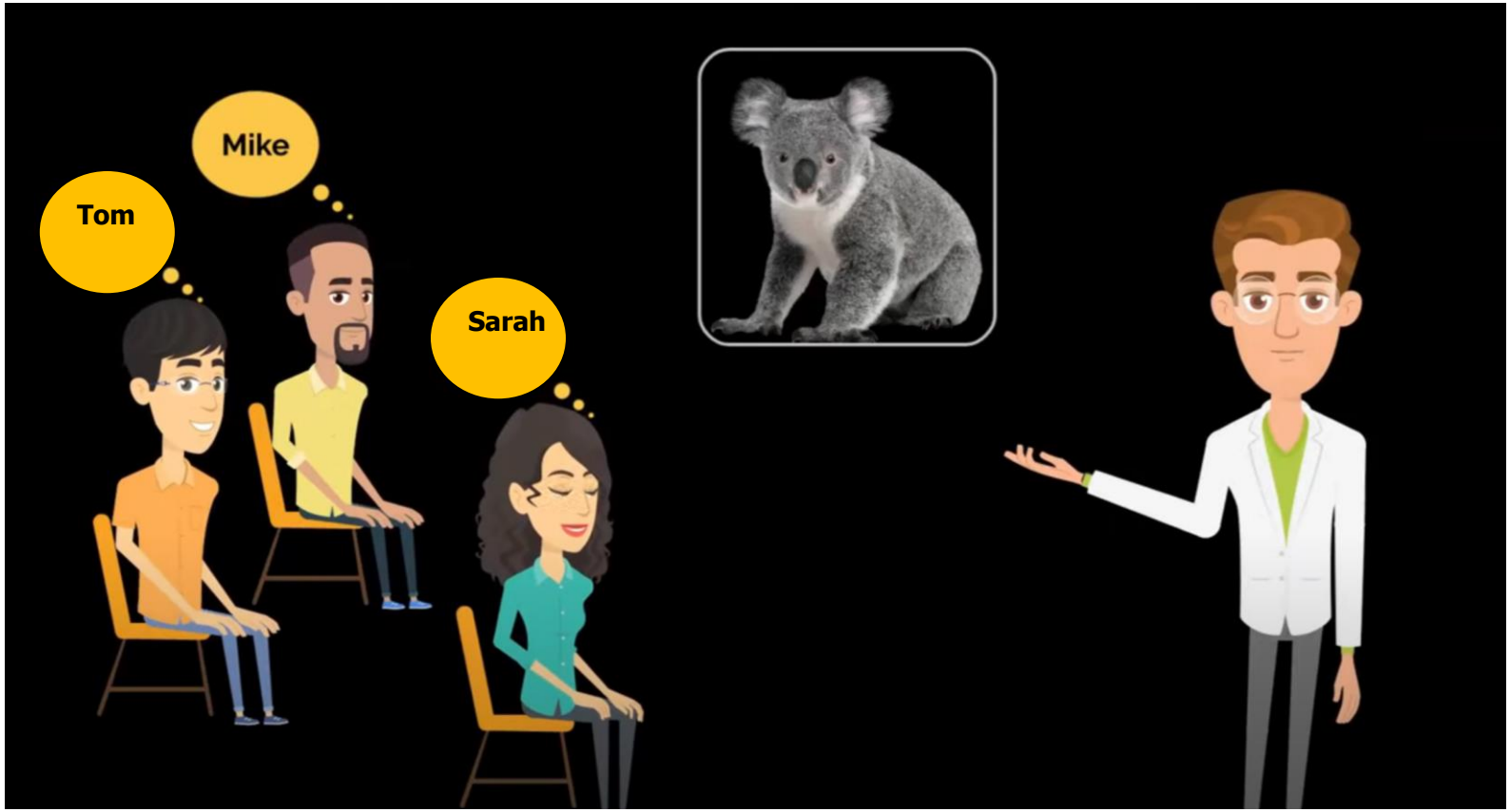

Any
Questions

High Level description for Neural Networks

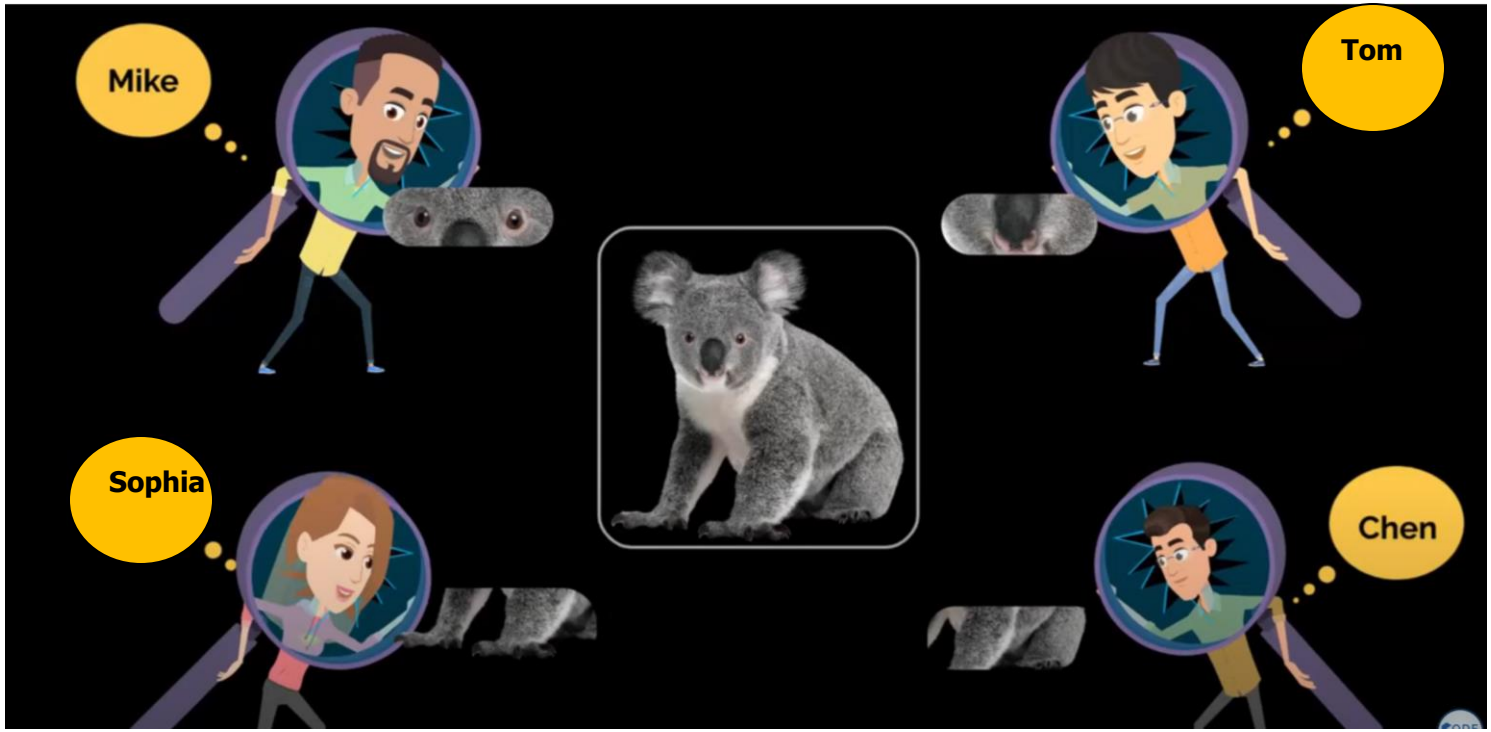


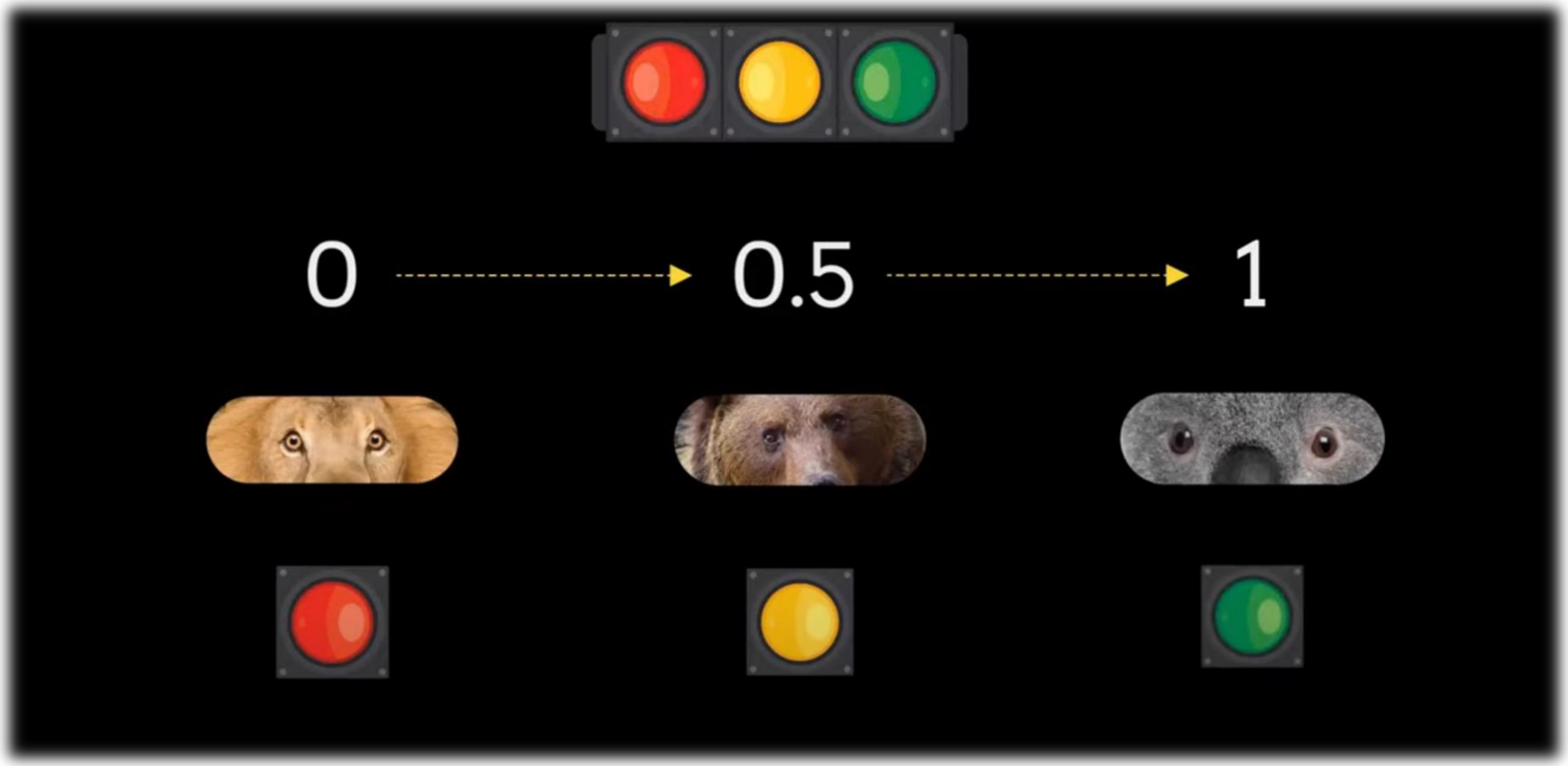

?
Is this Koala?

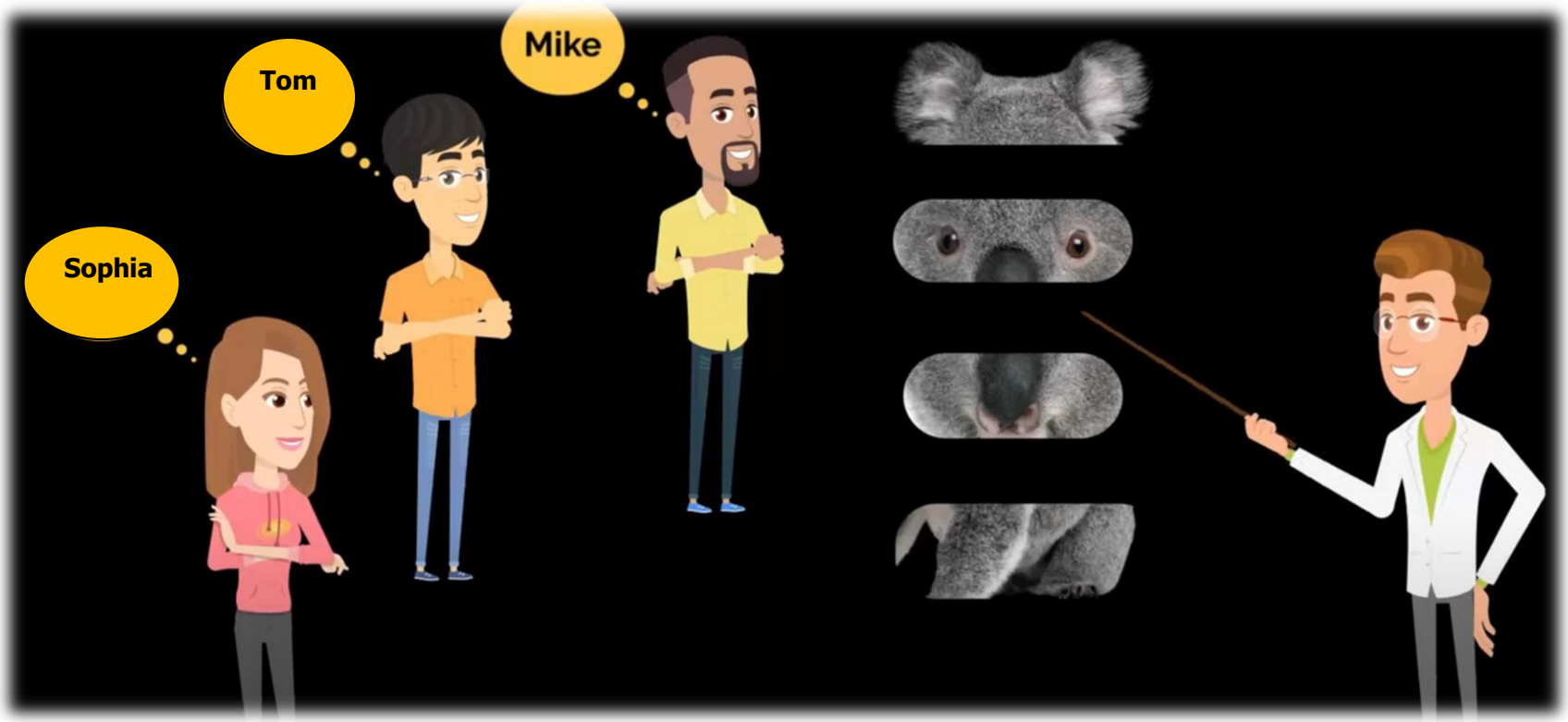

?
?
Is this Koala?
Yes
No

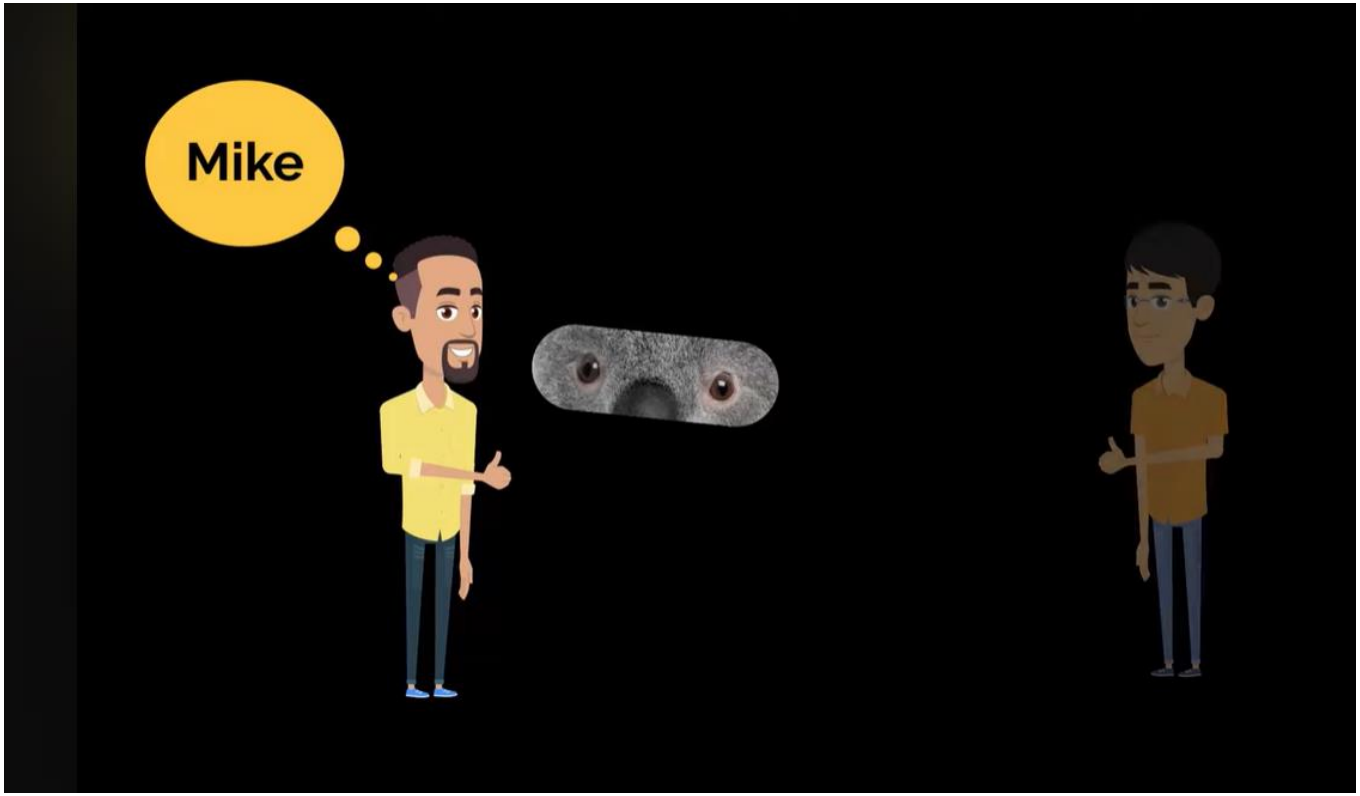

Mike
Tom
Sarah

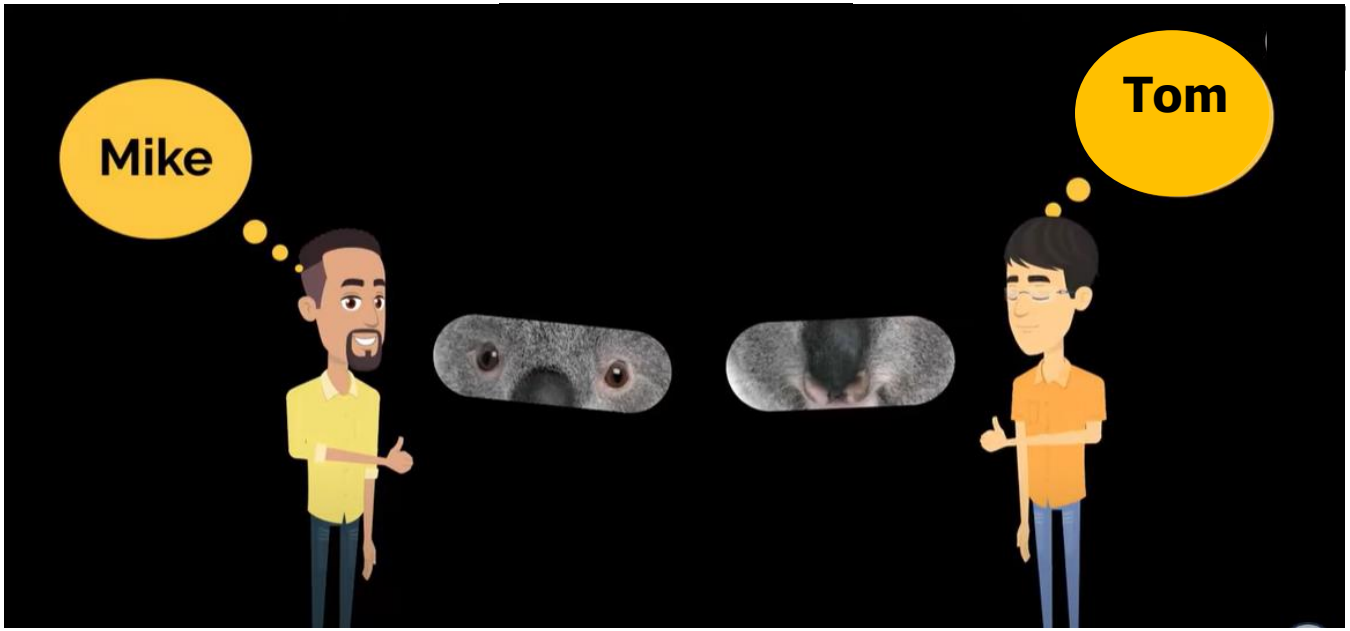

Tom
Mike
Sarah
Is this Koala?

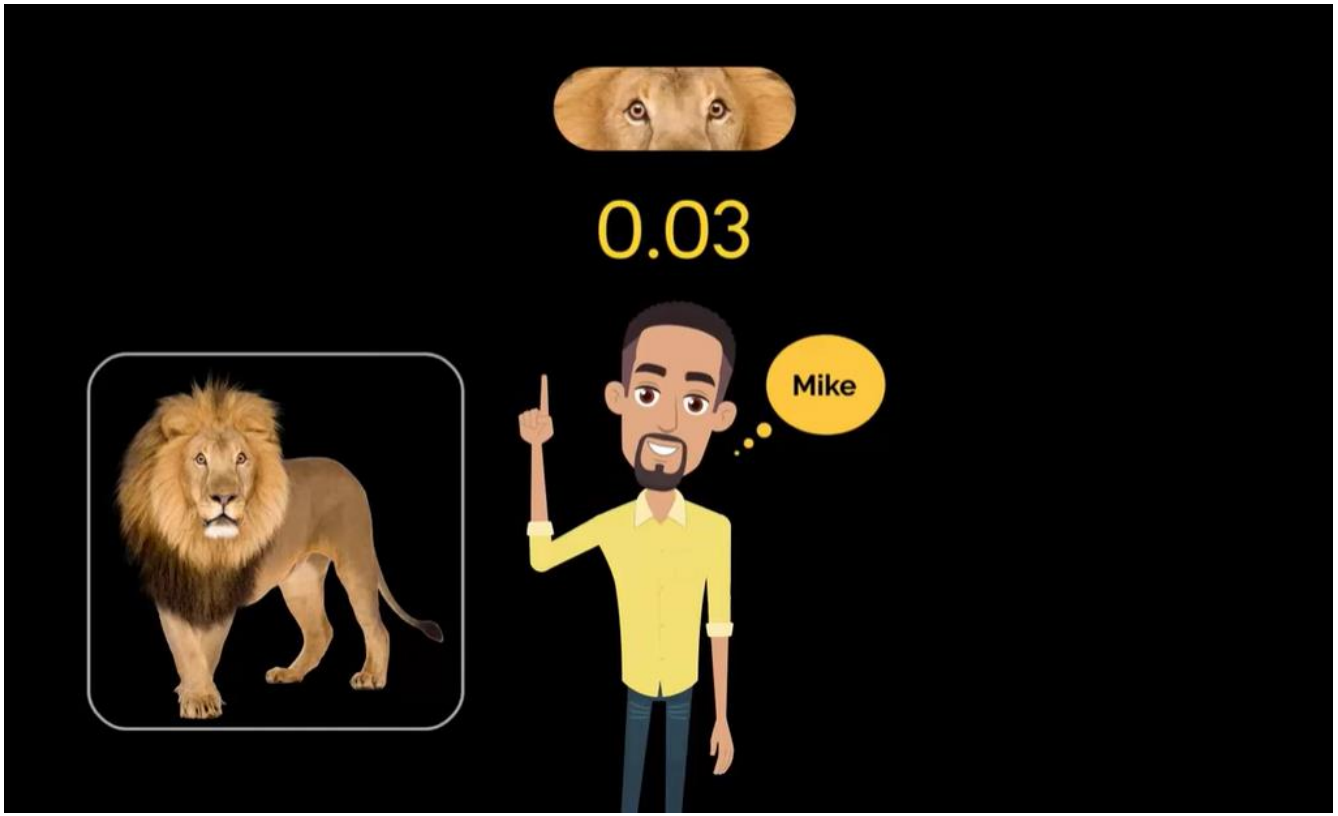

Mike
Tom
Sophia
Chen

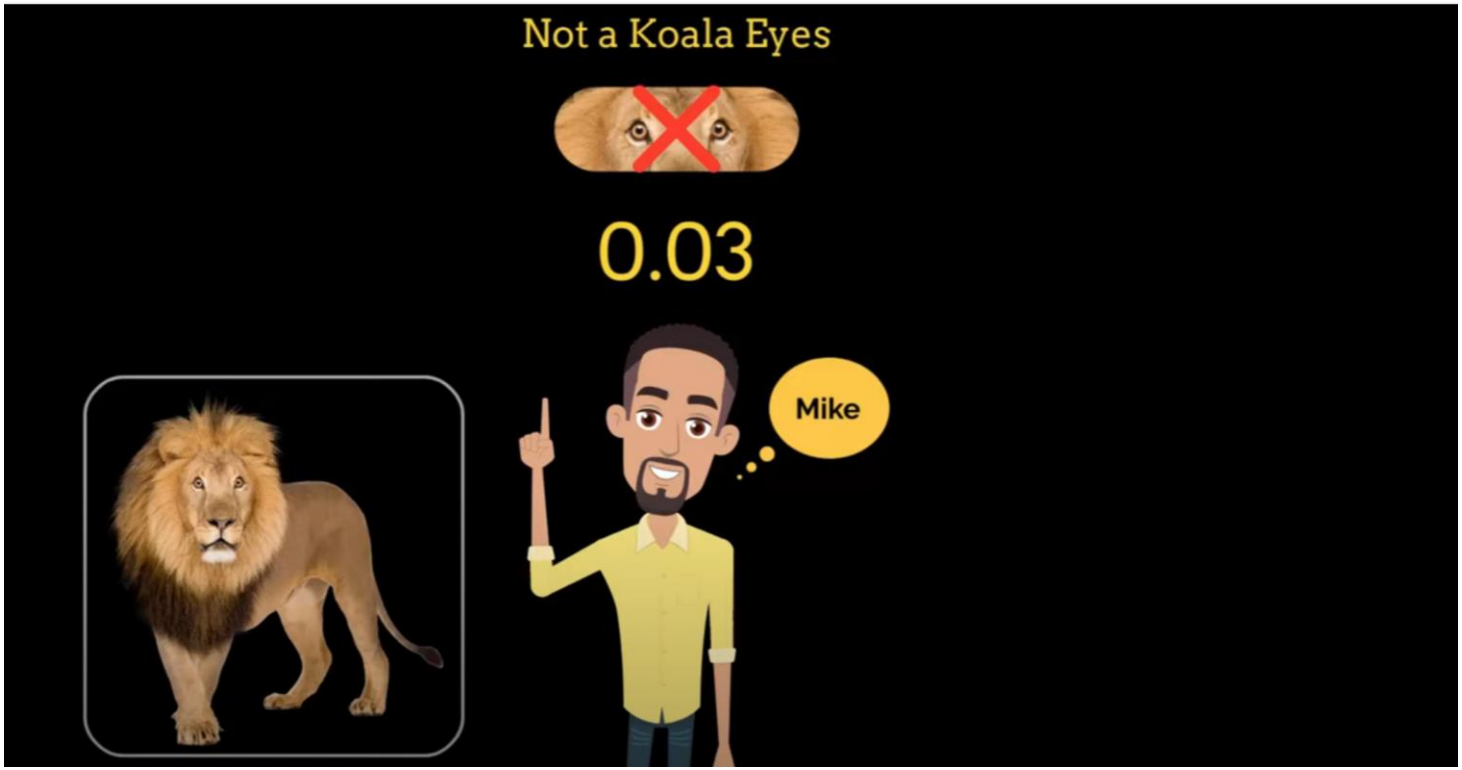

0
0.5
1

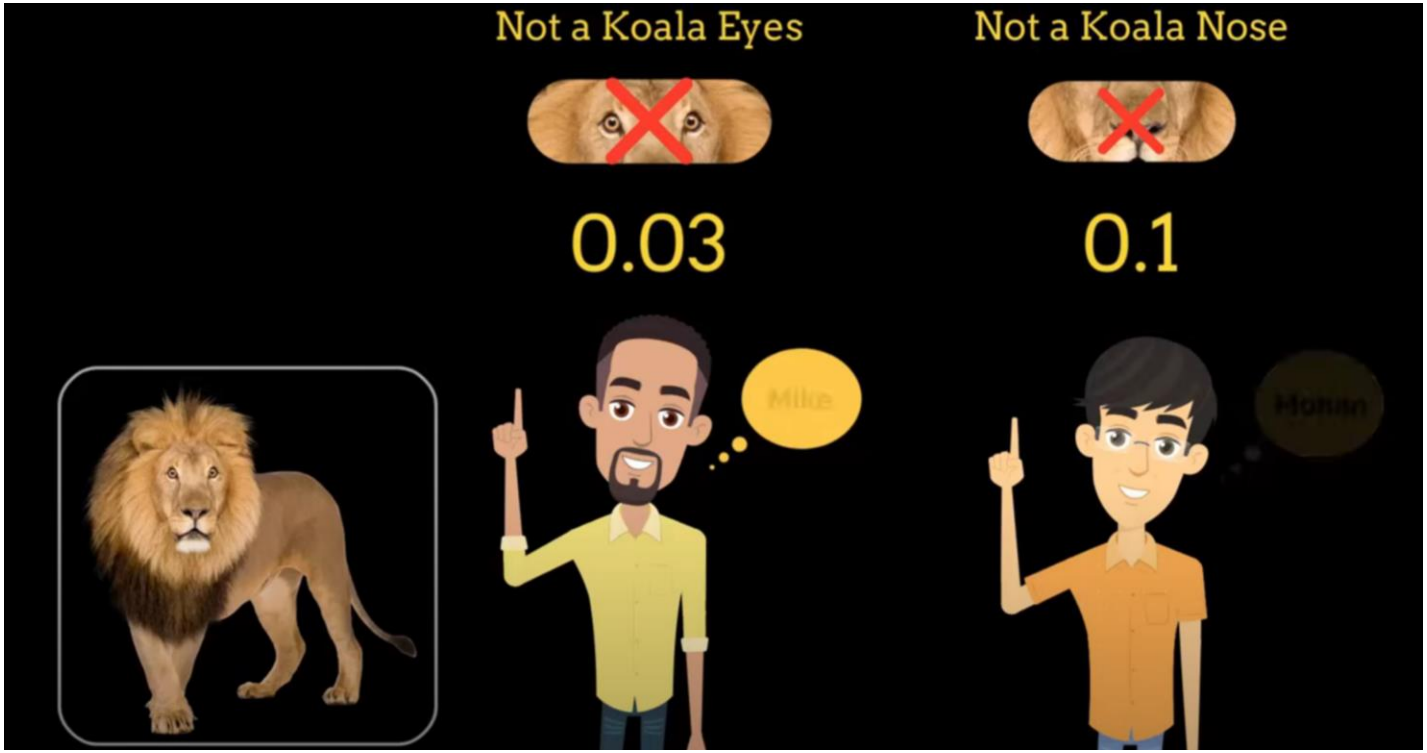

Mike
Tom
Sophia

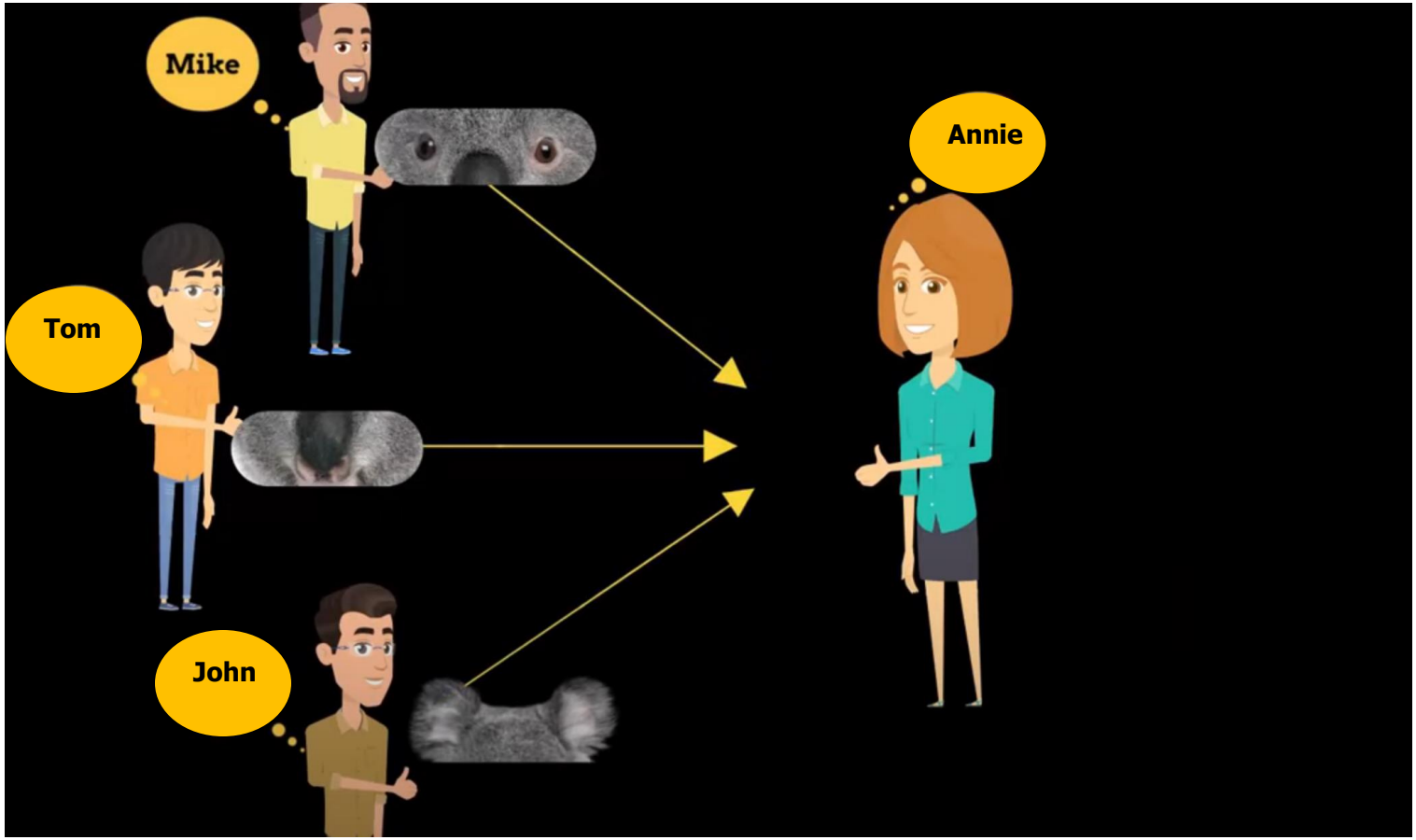

Mike

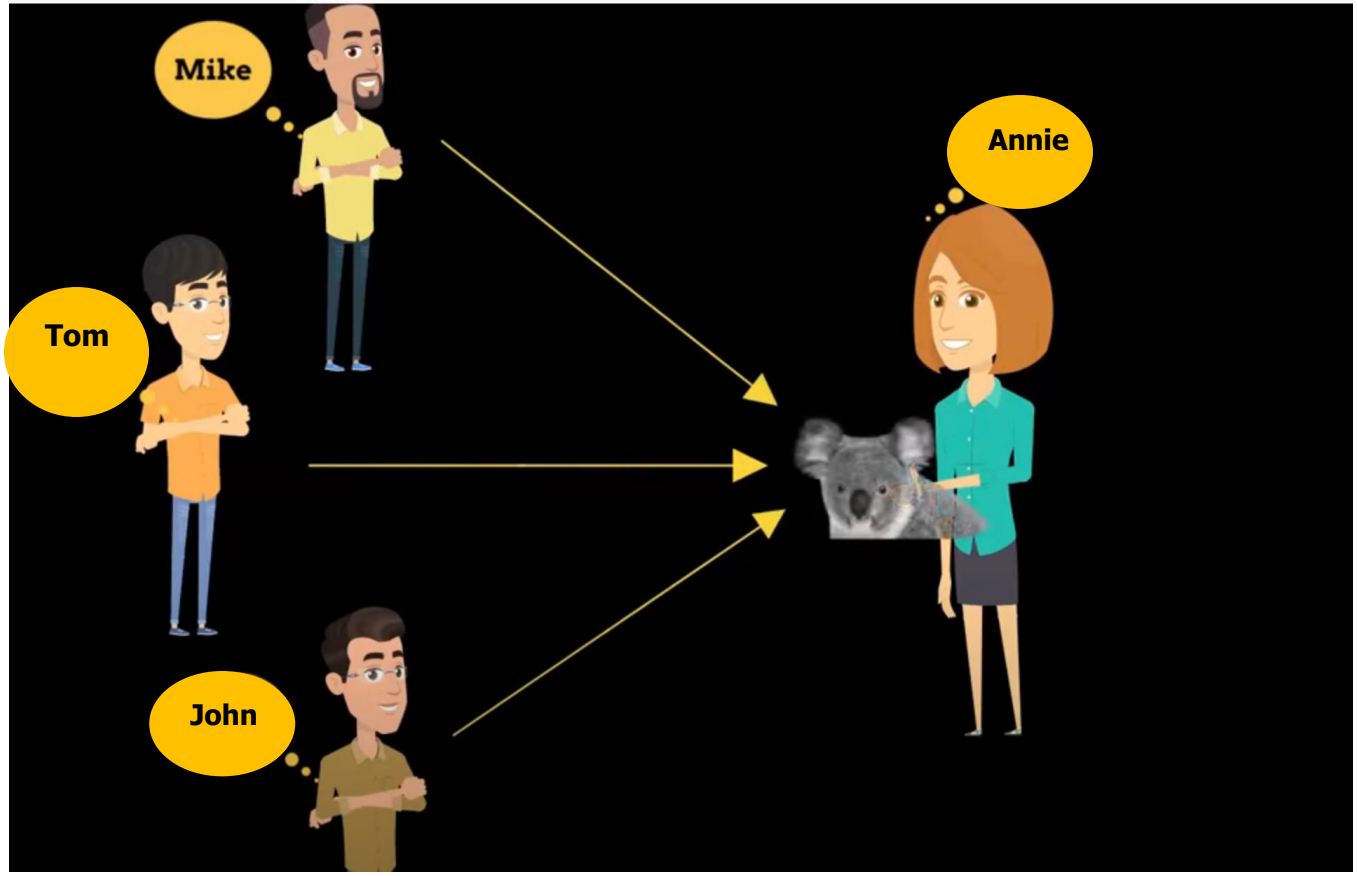

Mike
Tom

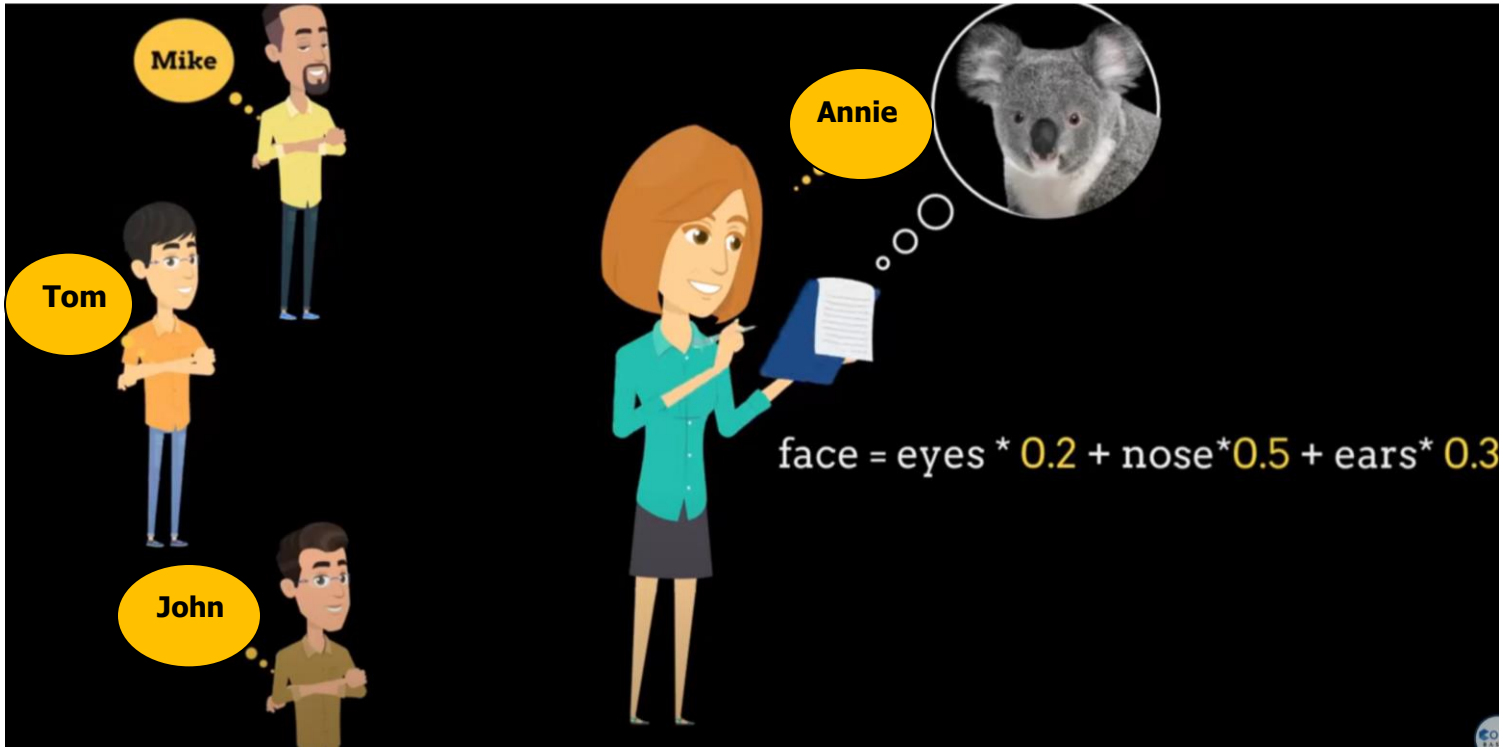

0.03
Mike

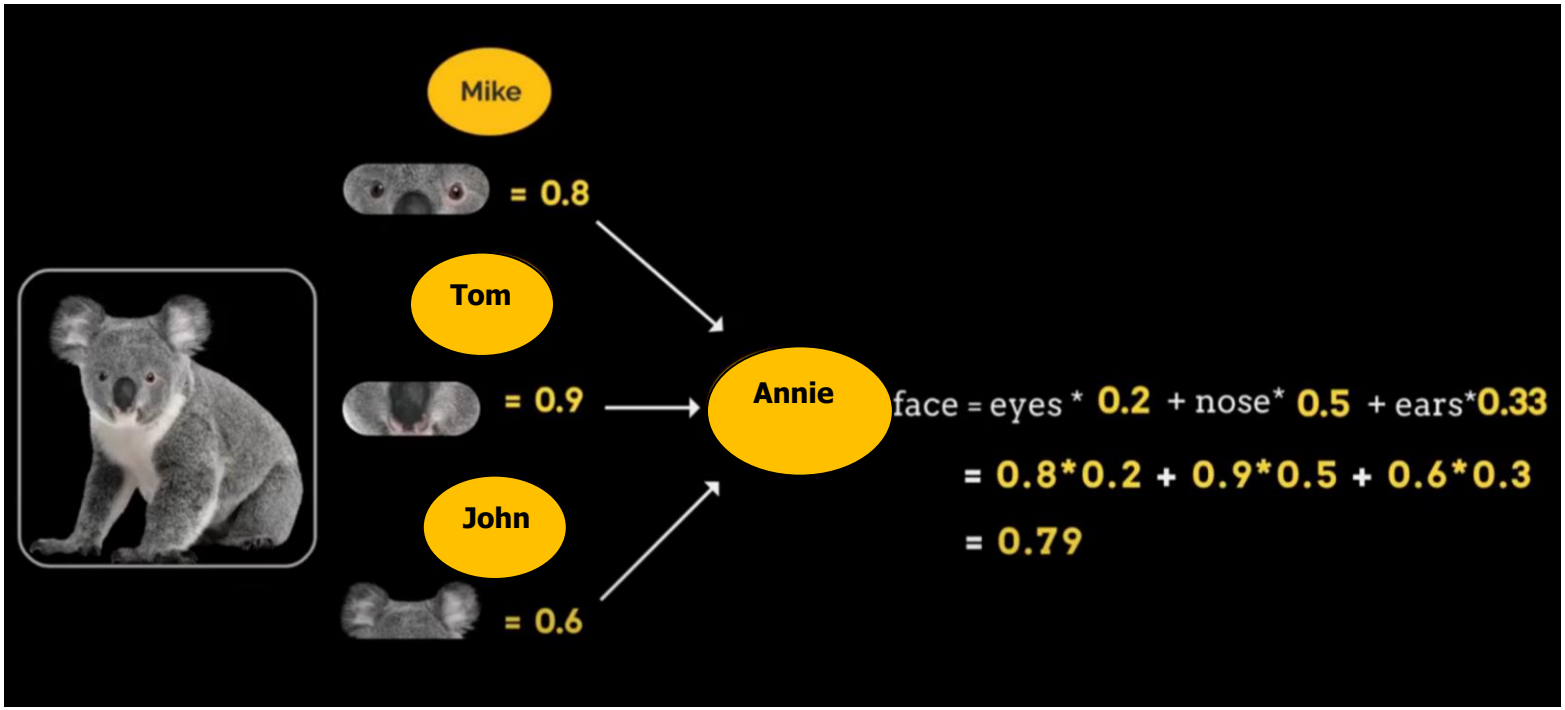

Not a Koala Eyes
0.03
Mike

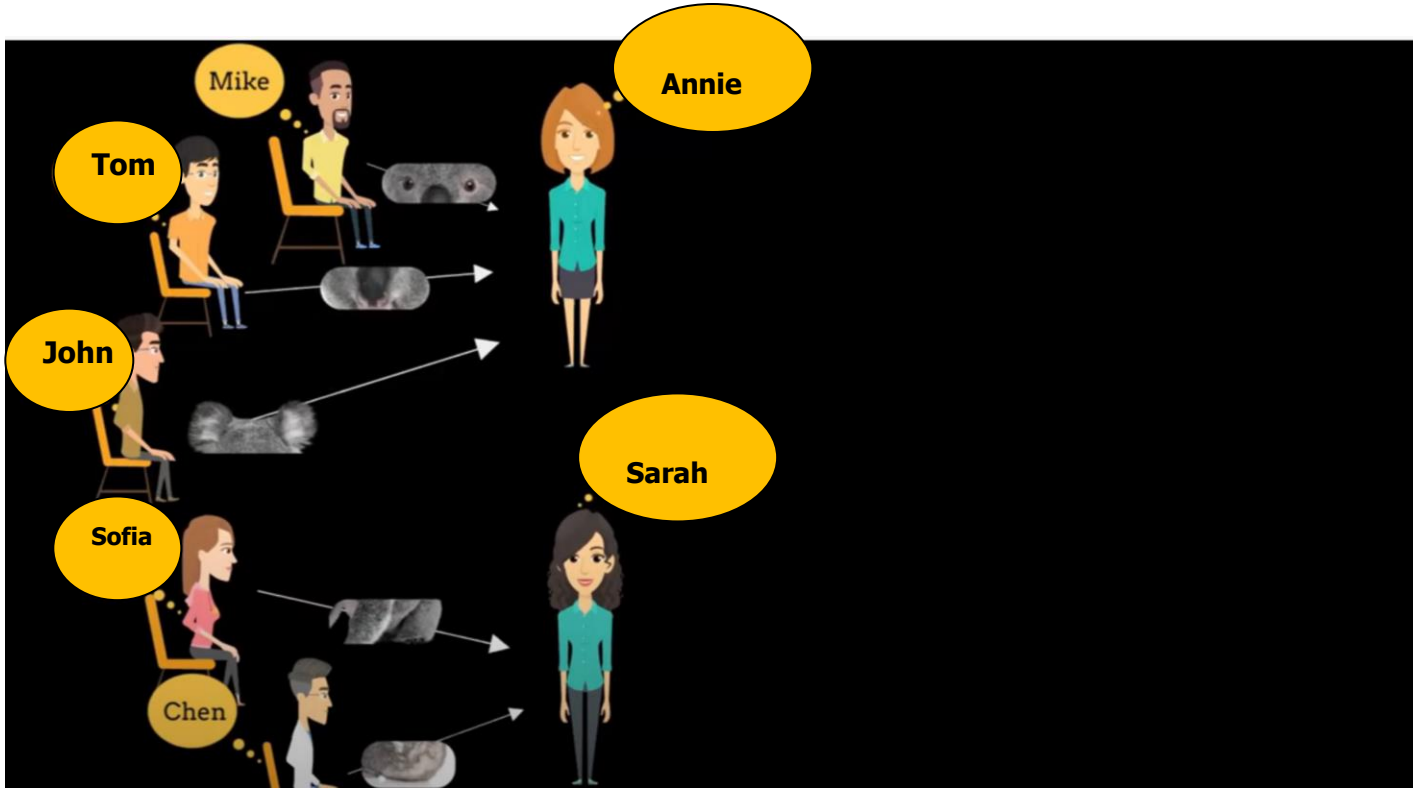

Not a Koala Eyes
Not a Koala Nose
0.03
0.1
Mike
Hanan

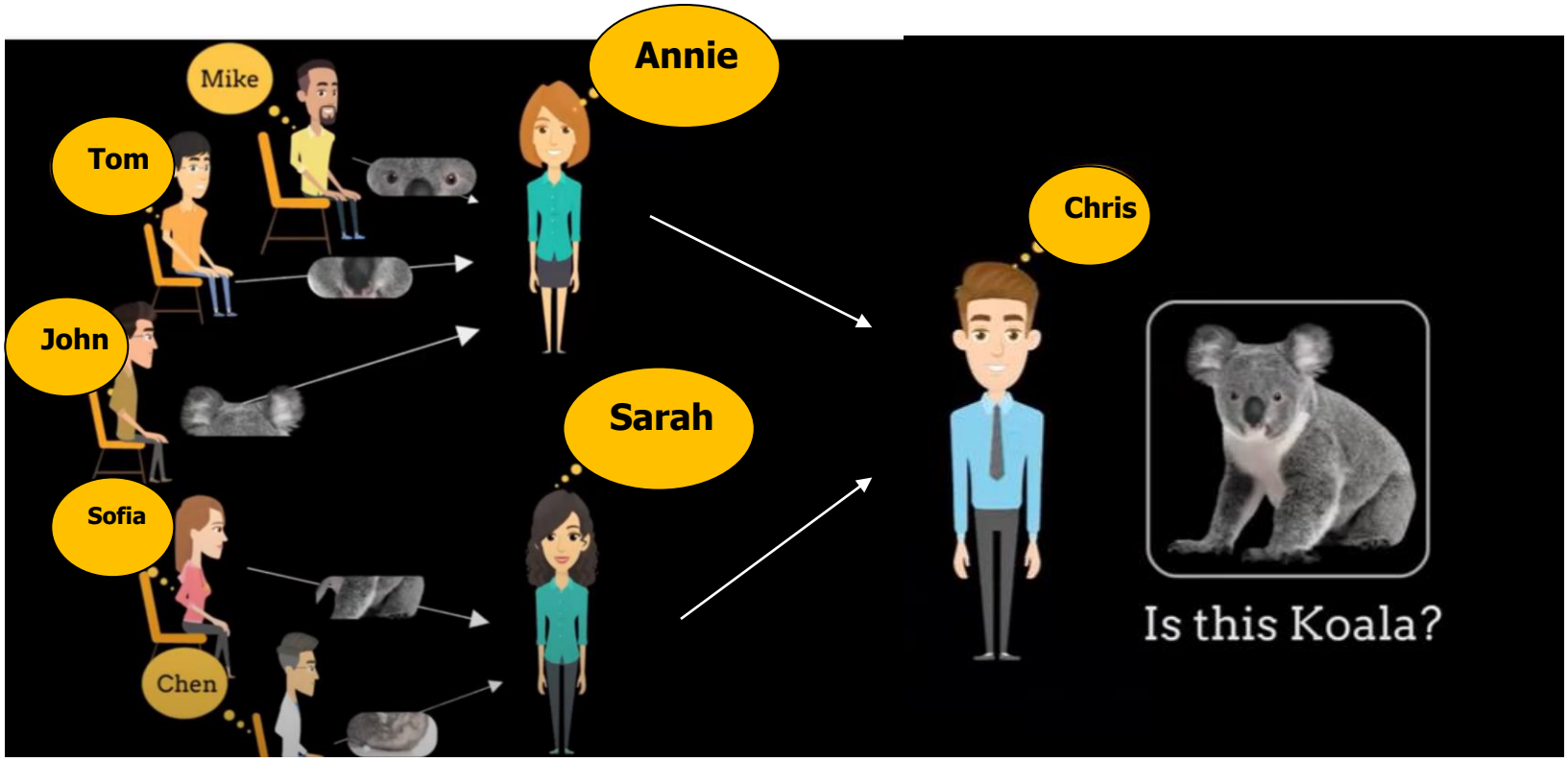

Mike
Tom
John
Annie

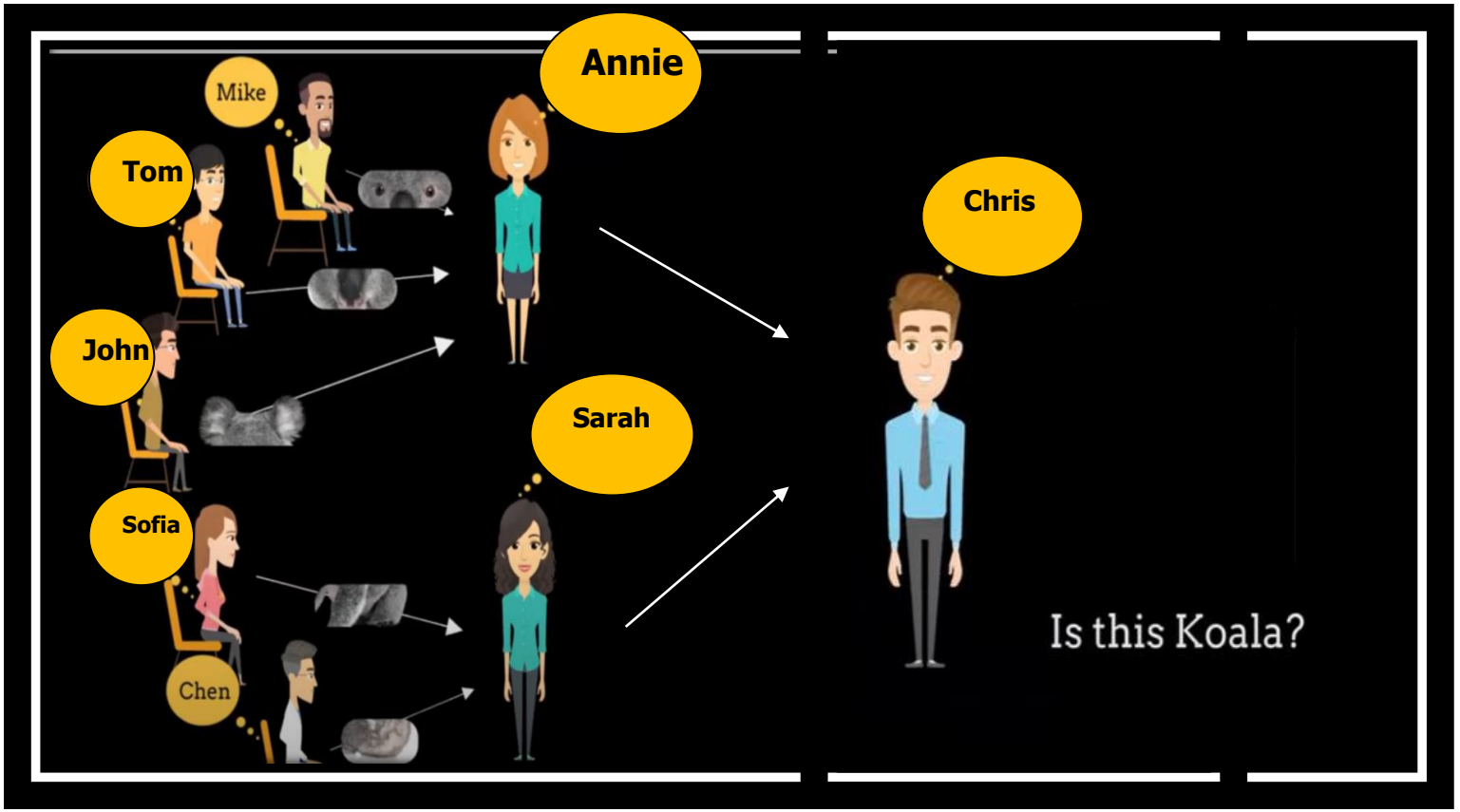

Mike
Tom
John
Annie


Mike
Tom
John
Annie
face = eyes * 0.2 + nose*0.5 + ears* 0.3

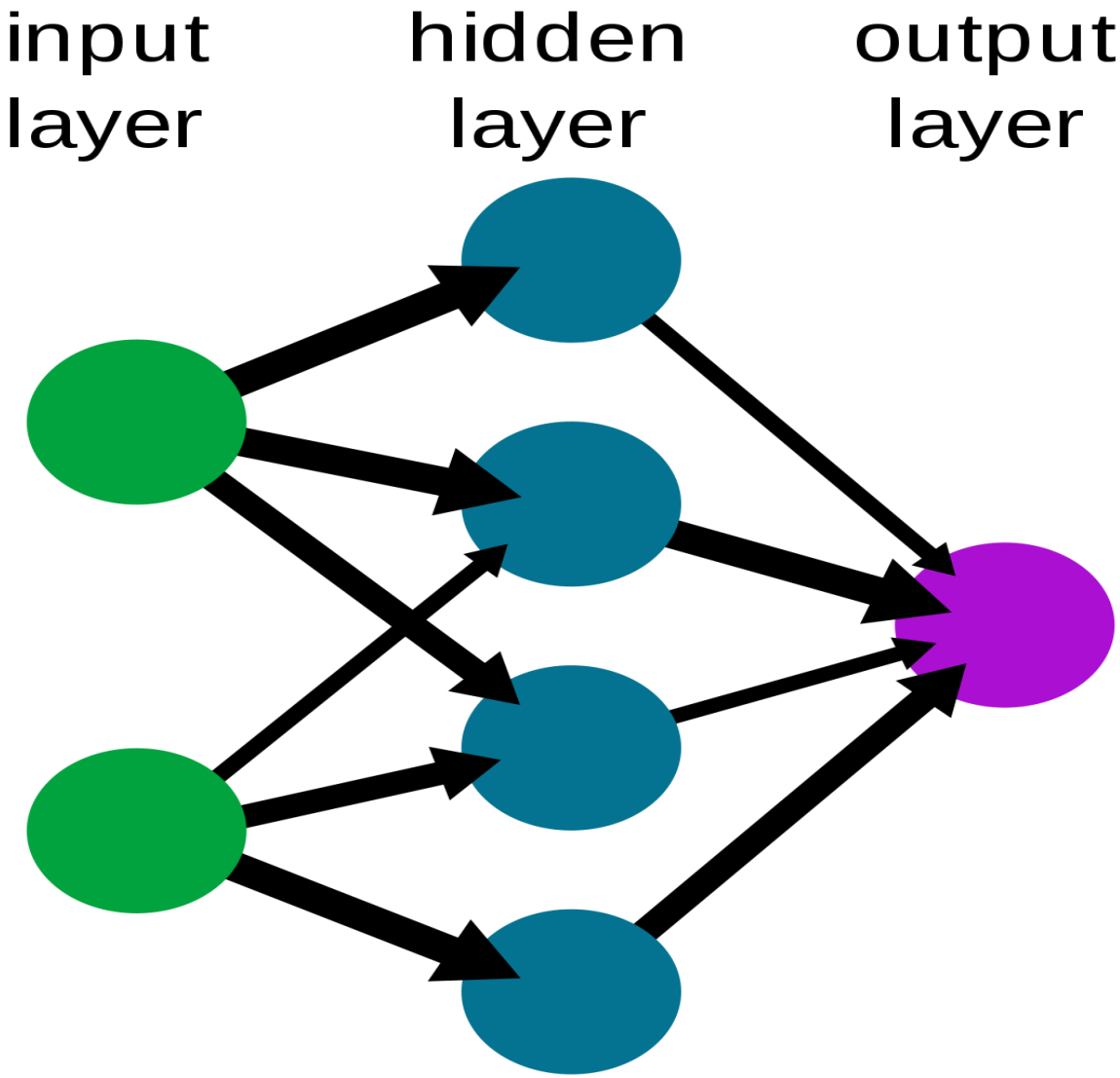

Mike
= 0.8
Tom
= 0.9
John
= 0.6
Annie
face = eyes * 0.2 + nose* 0.5 + ears*0.33
= 0.8*0.2 + 0.9*0.5 + 0.6*0.3
= 0.79

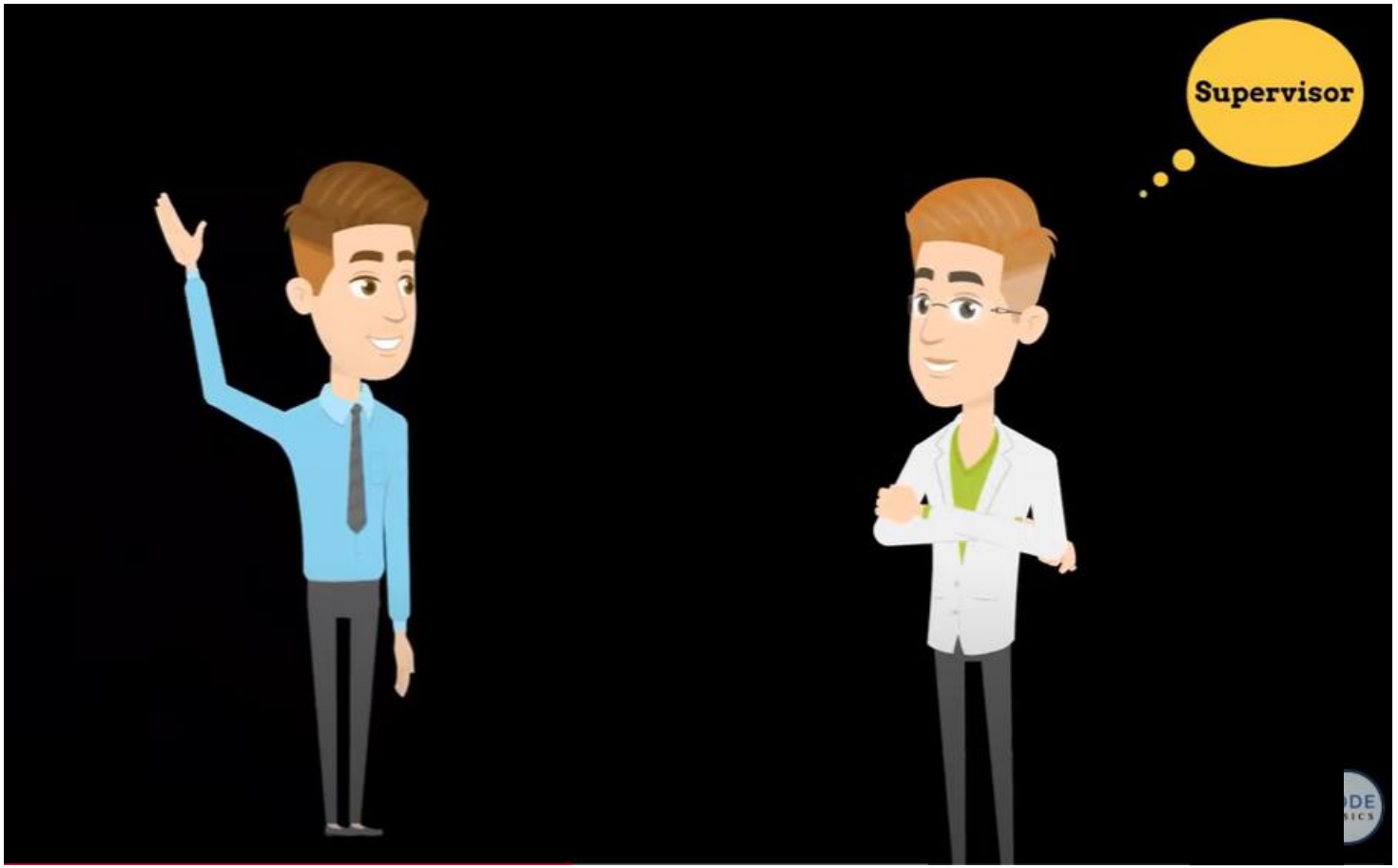

Mike
Annie
Tom
John
Sarah
Sofia
Chen

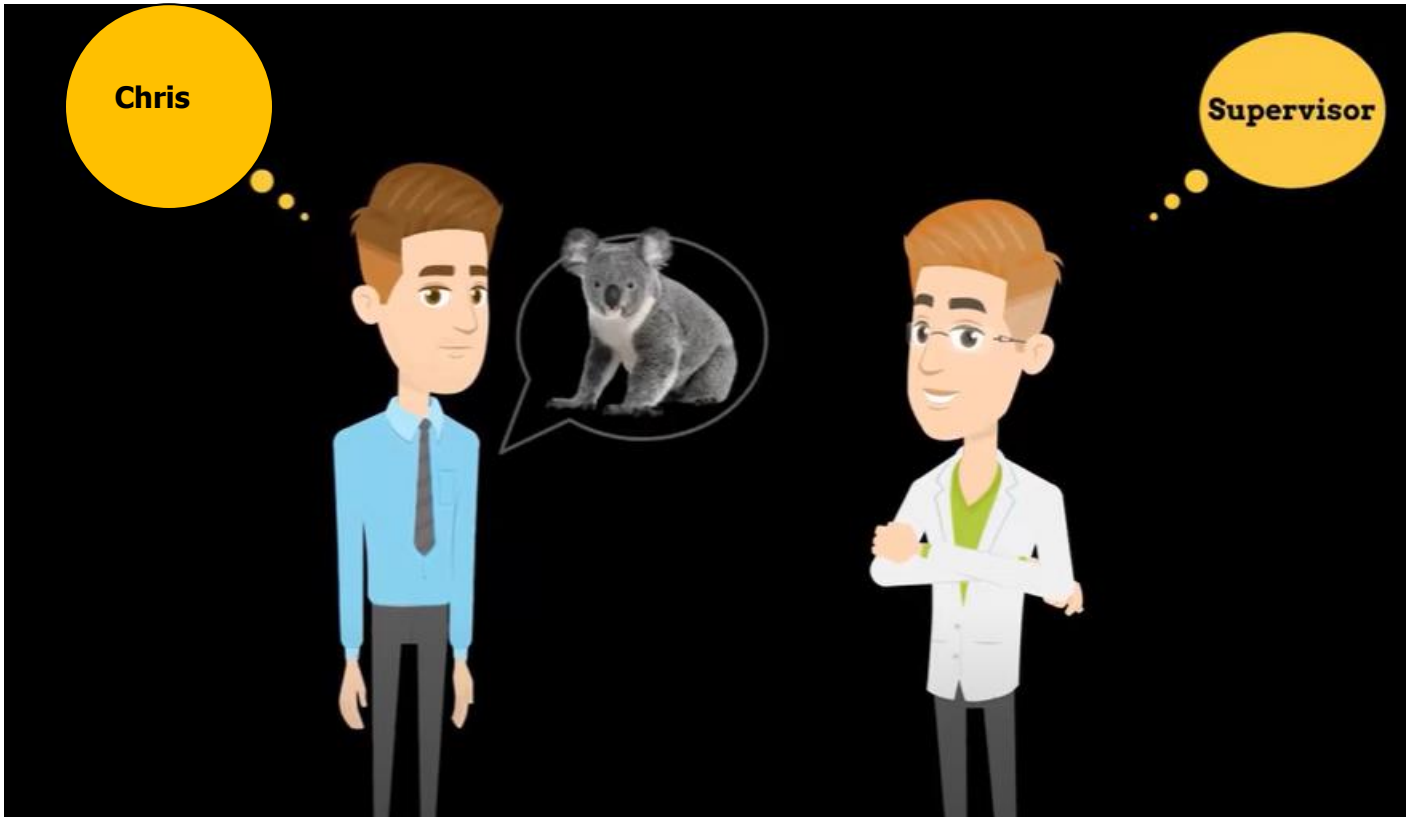

Mike
Tom
John
Sofia
Chen
Annie
Sarah
Chris
Is this Koala?


Annie
Mike
Tom
John
Sarah
Sofia
Chen
Chris
Is this Koala?

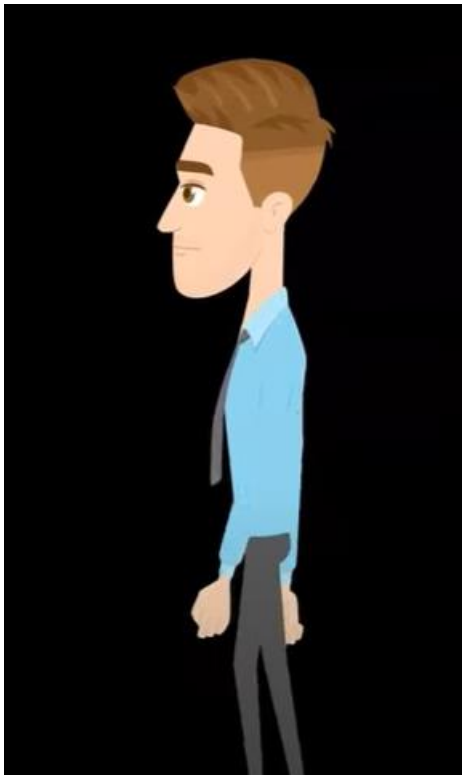
A simple neural network

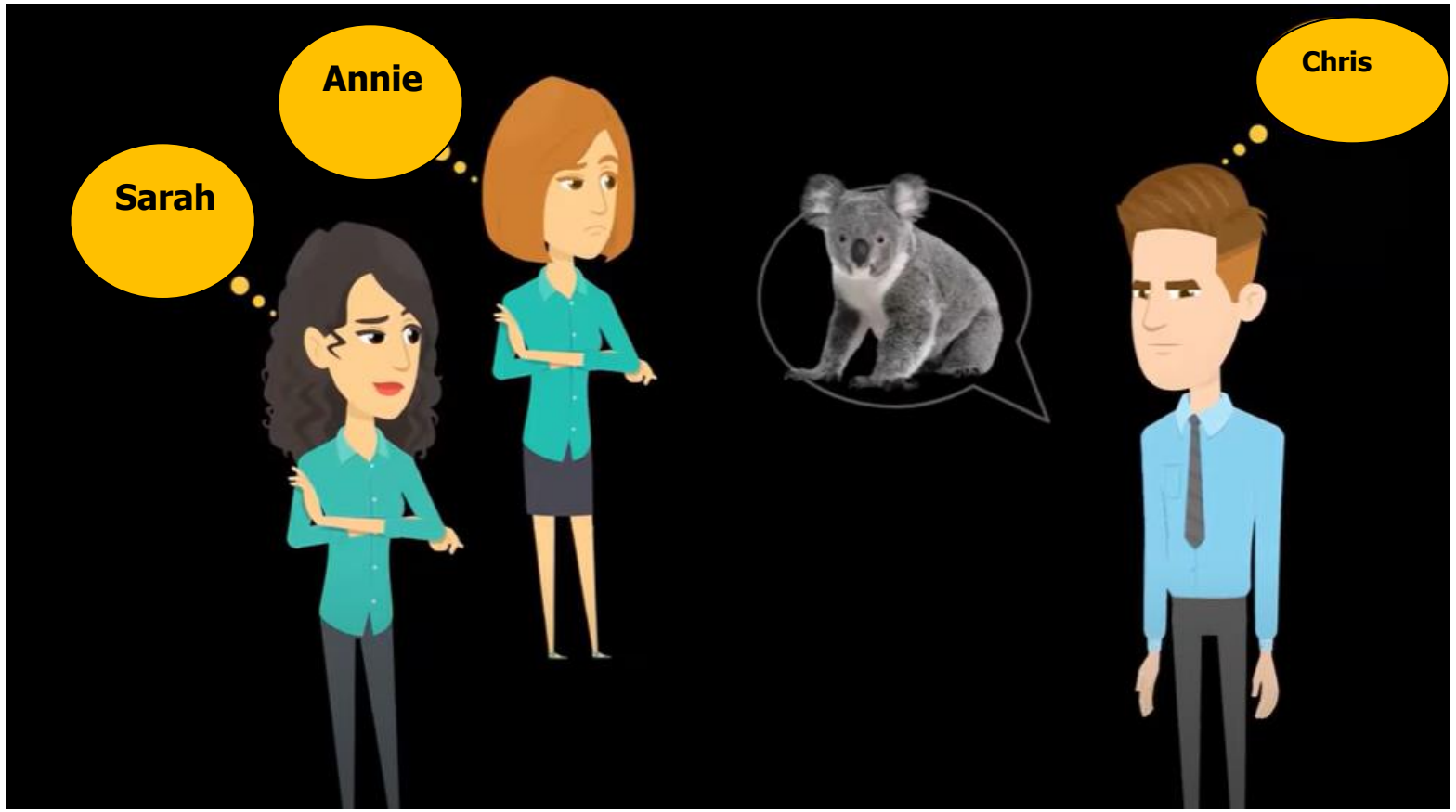


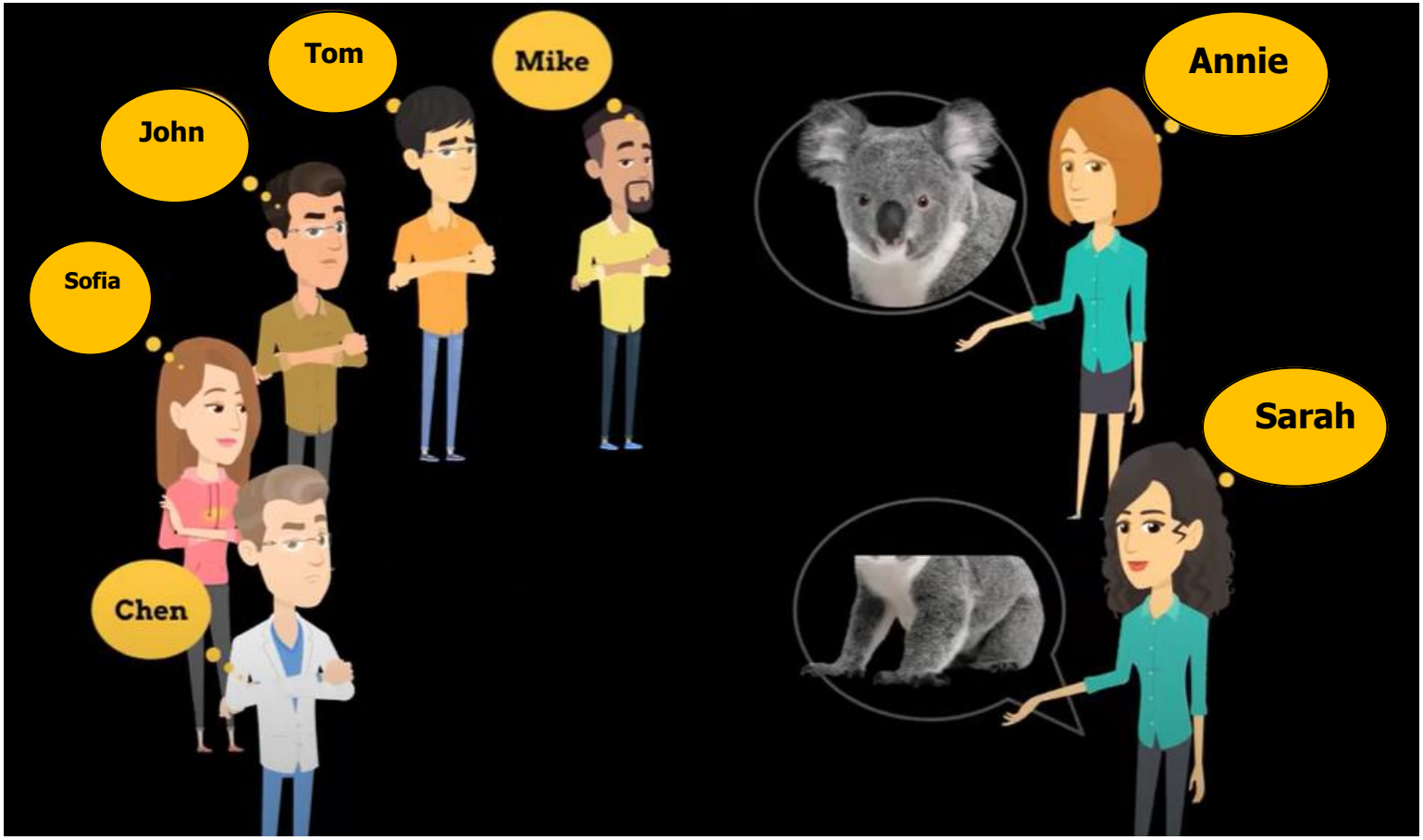

Supervisor

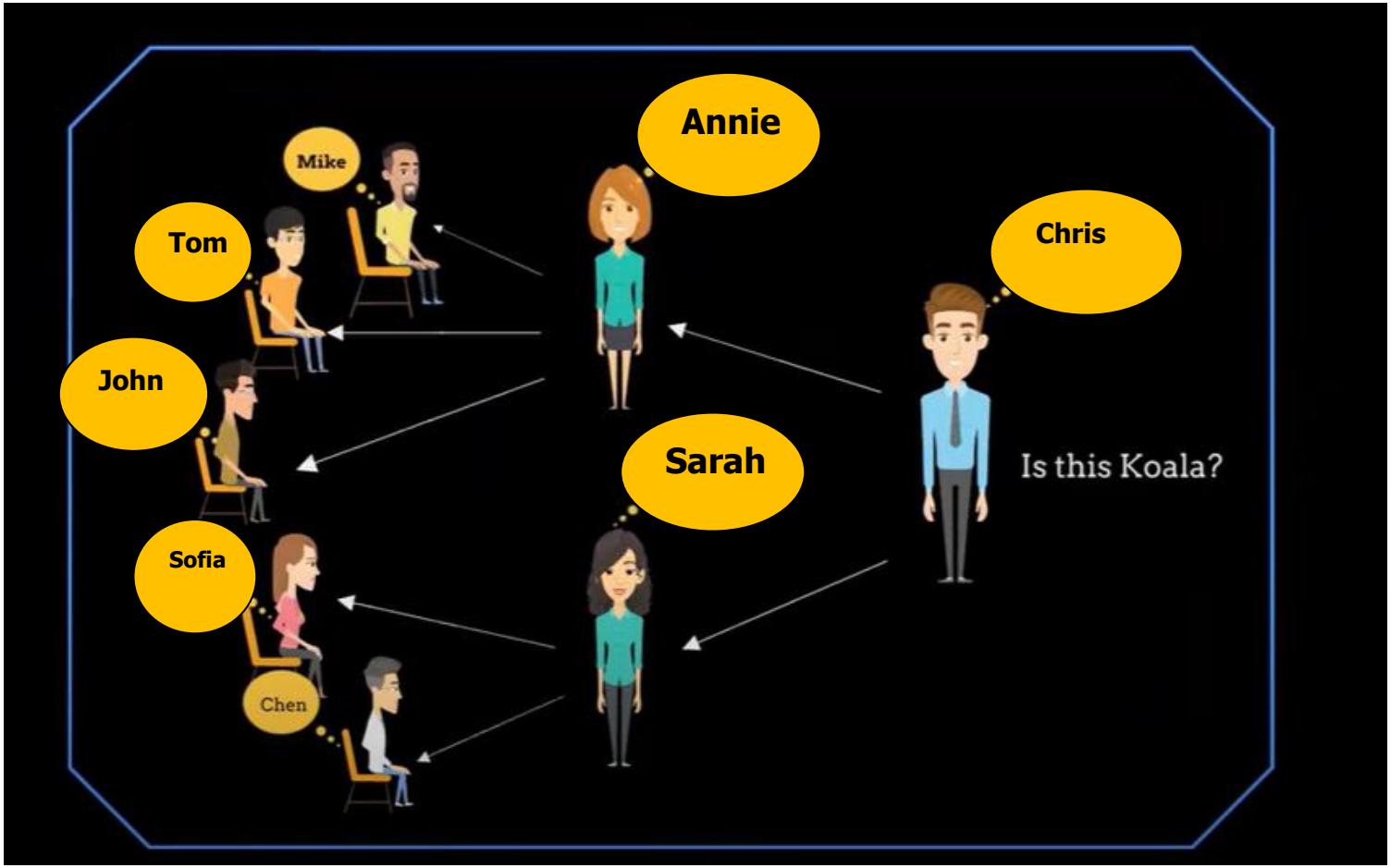

Chris
Supervisor

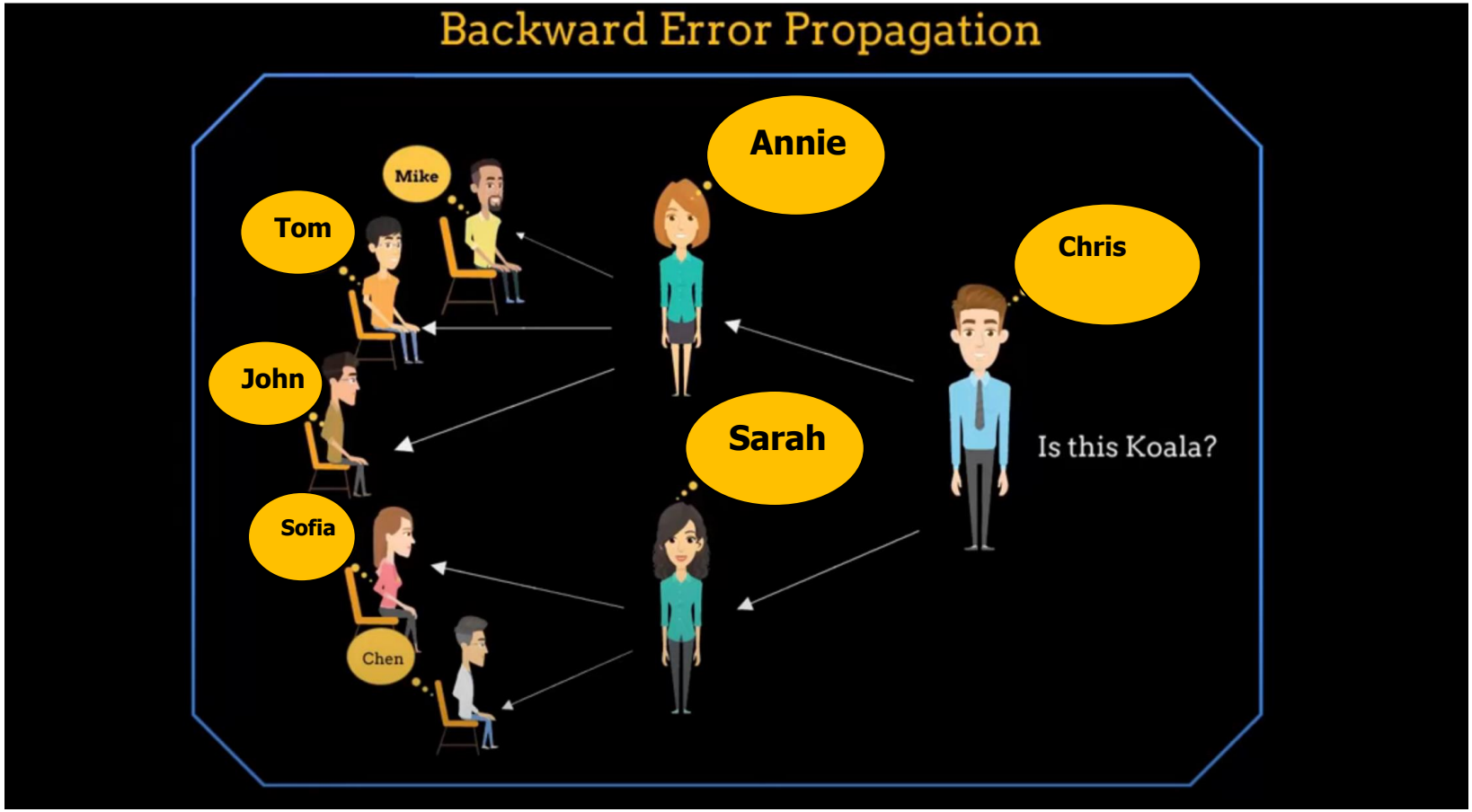
Chris

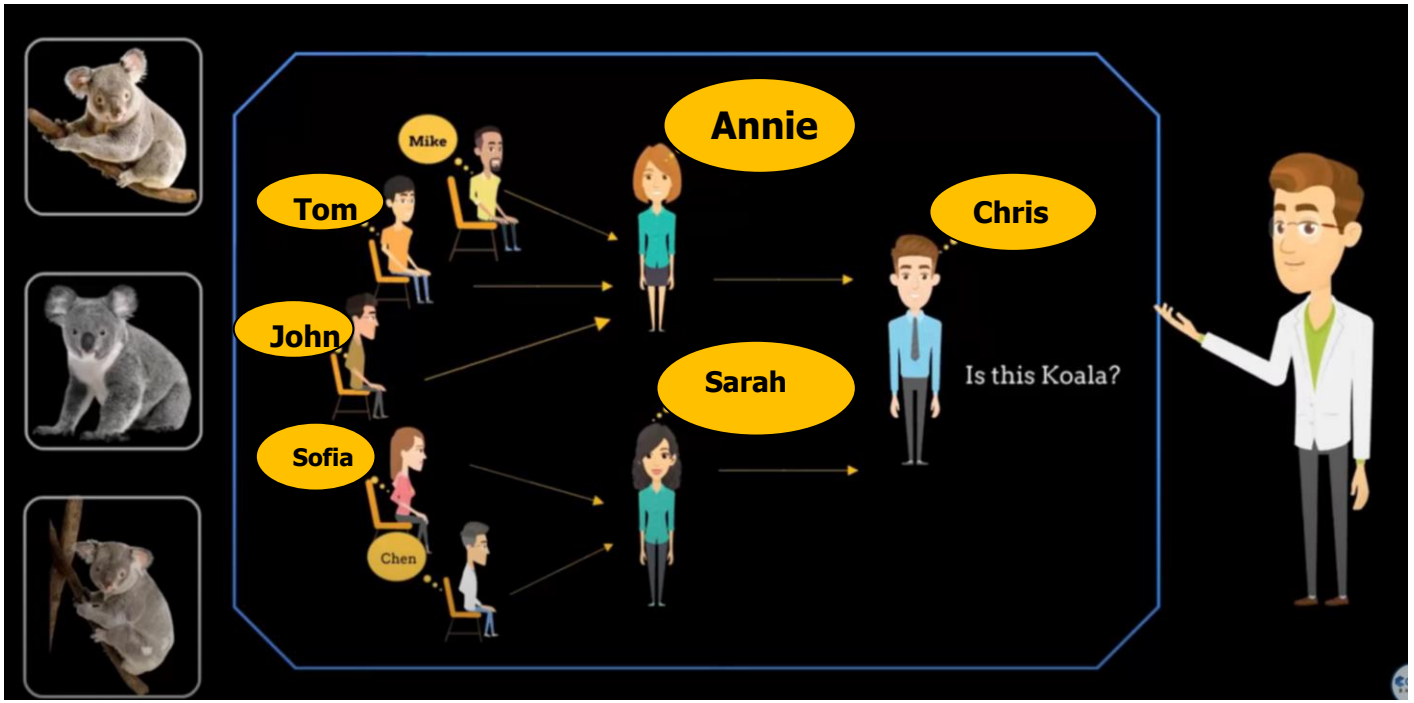


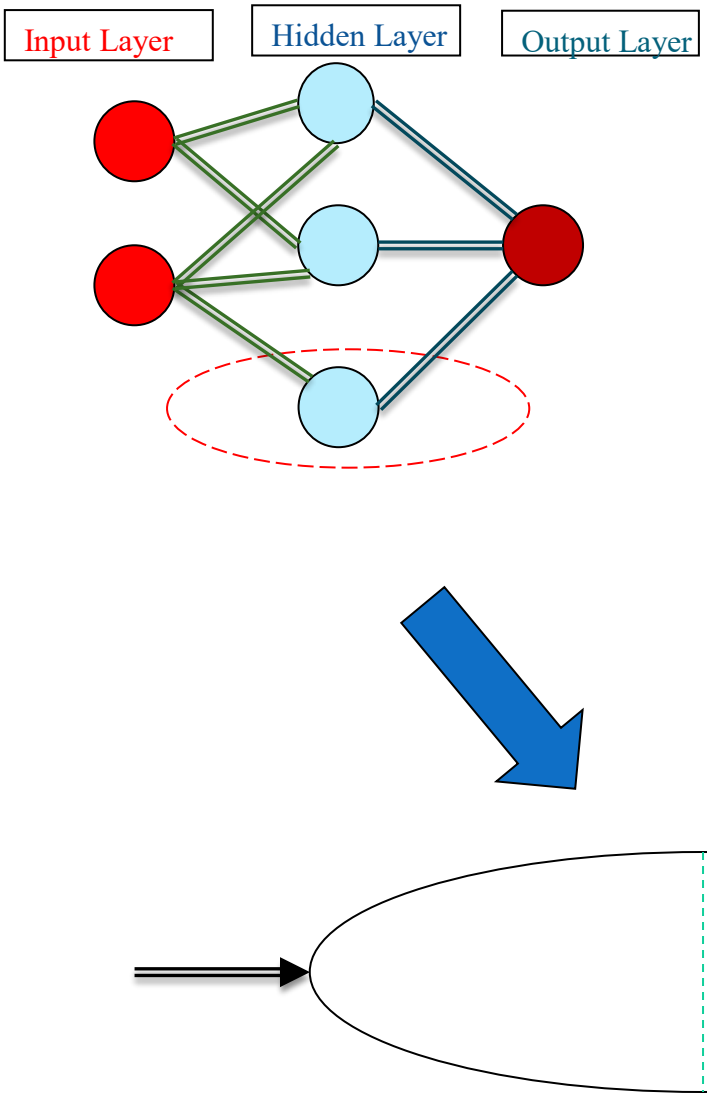

Annie
Sarah
Chris


Tom
Mike
John
Annie
Sofia
Sarah
Chen


Annie
Mike
Tom
Chris
John
Sarah
Is this Koala?
Sofia
Chen


Backward Error Propagation
Annie
Mike
Tom
Chris
John
Sarah
Is this Koala?
Sofia
Chen


Mike
Tom
John
Sofia
Chen
Annie
Sarah
Chris
Is this Koala?


Input Layer
Hidden Layer
Output Layer

How NN Works?

Imagine that you work as Machine Learning scientist at life insurance company and your Boss give you a task to predict how likely your customer will buy the insurance.

age	have_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1

What is the first step you want to do to understand your data?

age	have_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1

What are the distinctions between Linear Regression and Logistic Regression in machine learning, and what specific categories of problems does each method address?

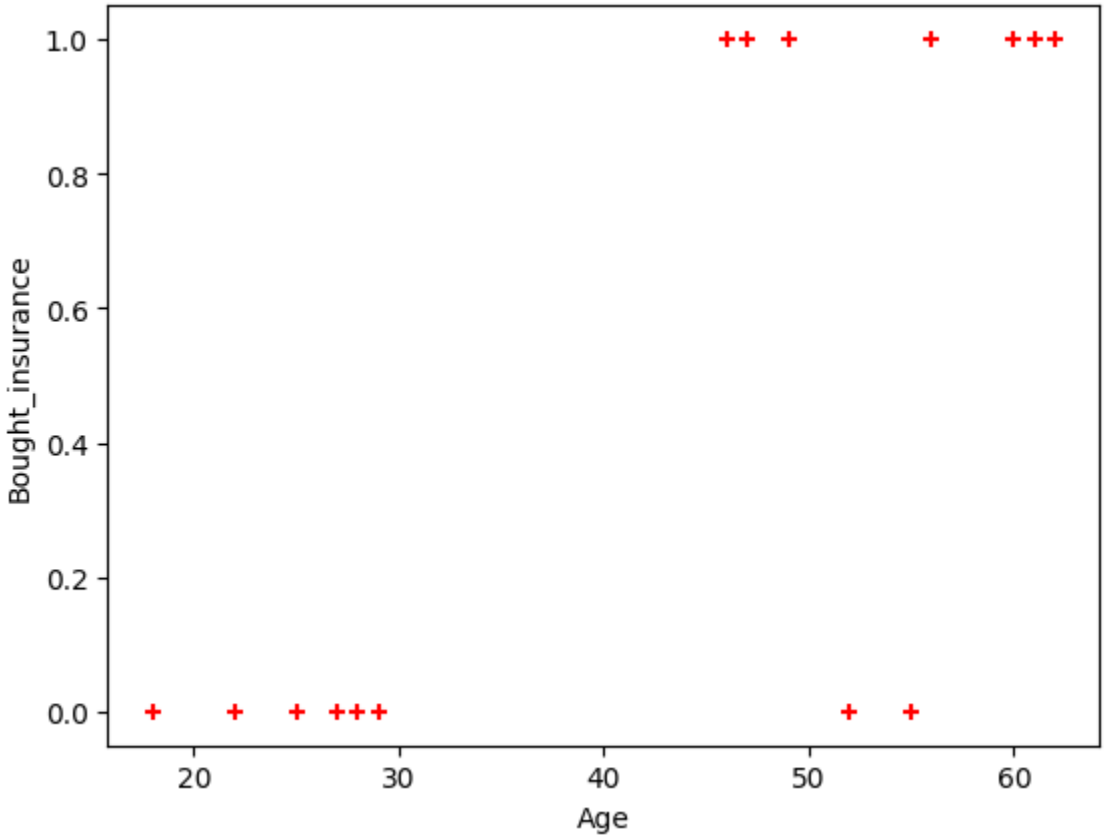
Reference for Regression

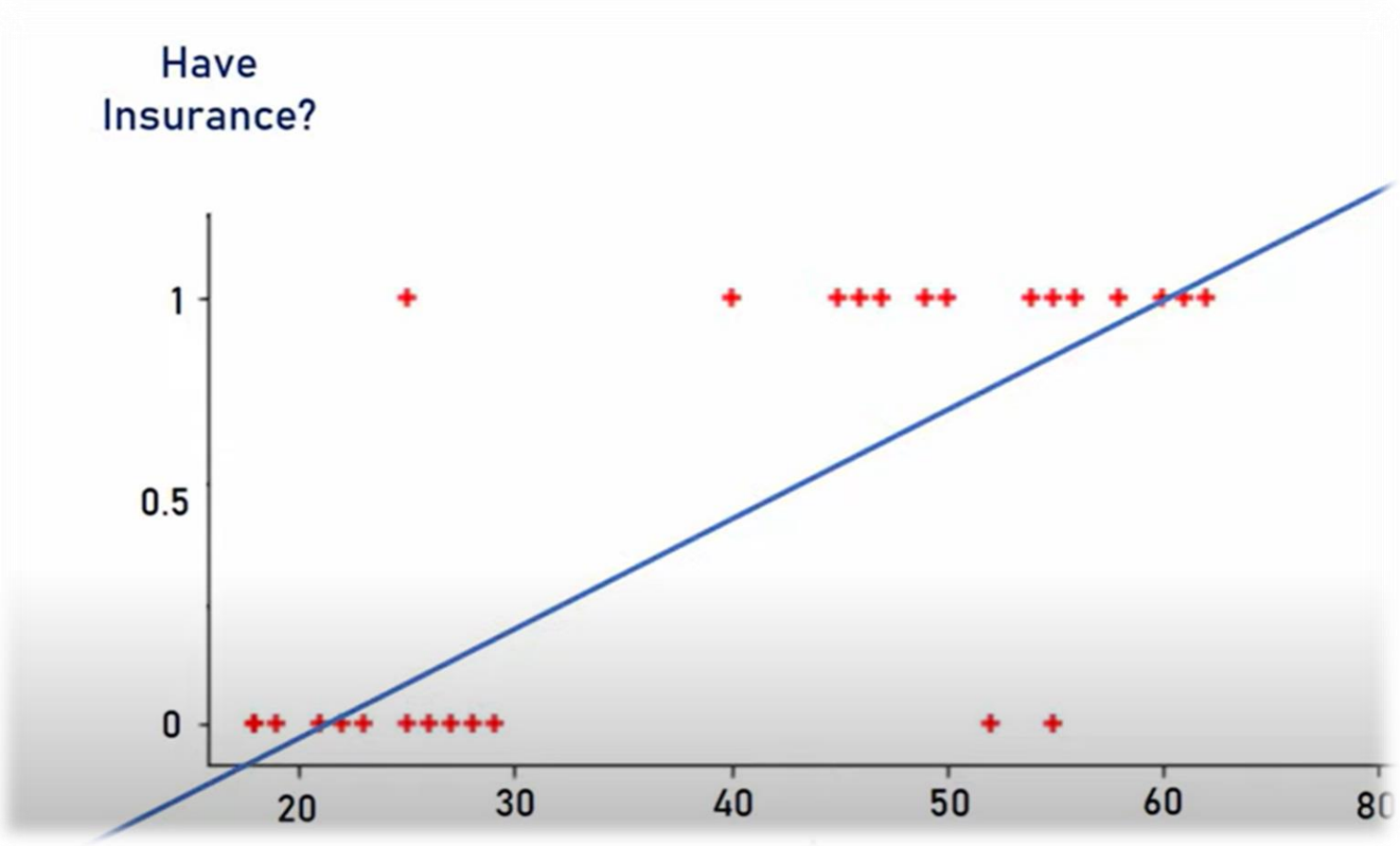
How NN works?

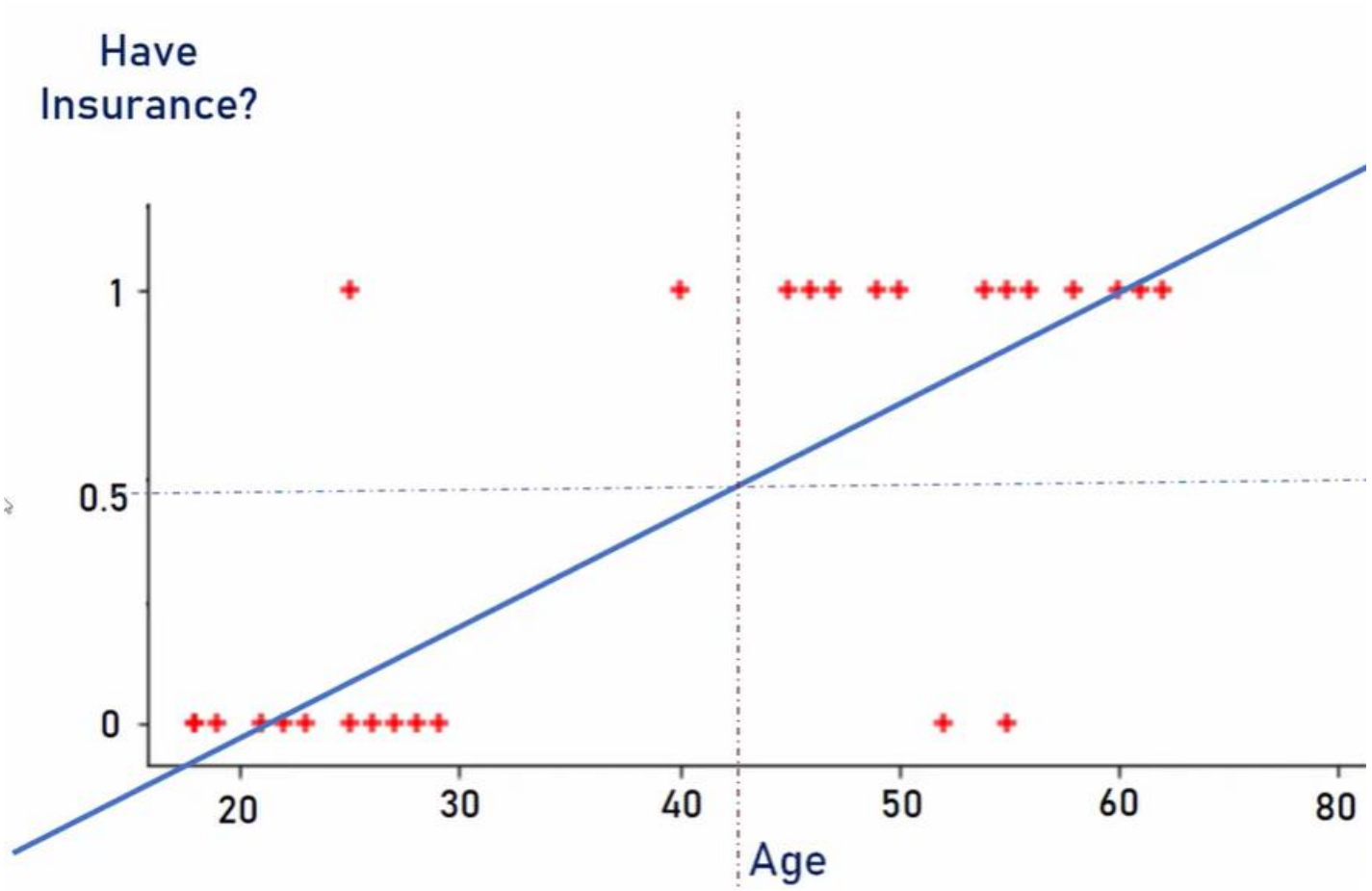
age	bought_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1

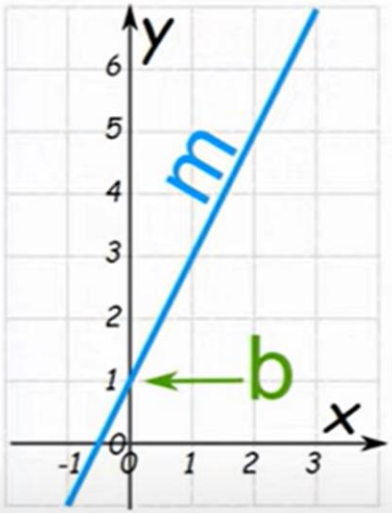
Given an age of a person, come up with a **function** that can predict if person will buy insurance or not

age	bought_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1




Have
Insurance?
1
0.5
0
20
30
40
50
60
80

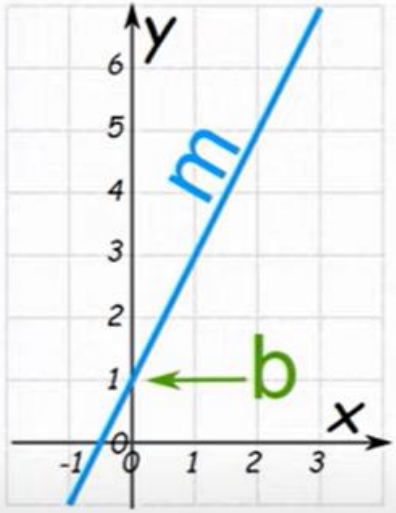

Have
Insurance?
1
0.5
0
20
30
40
50
60
80
Age



$$y = mx + b$$

Slope (or Gradient) Y Intercept

age	bought_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1



Have insurance = m * Age + b

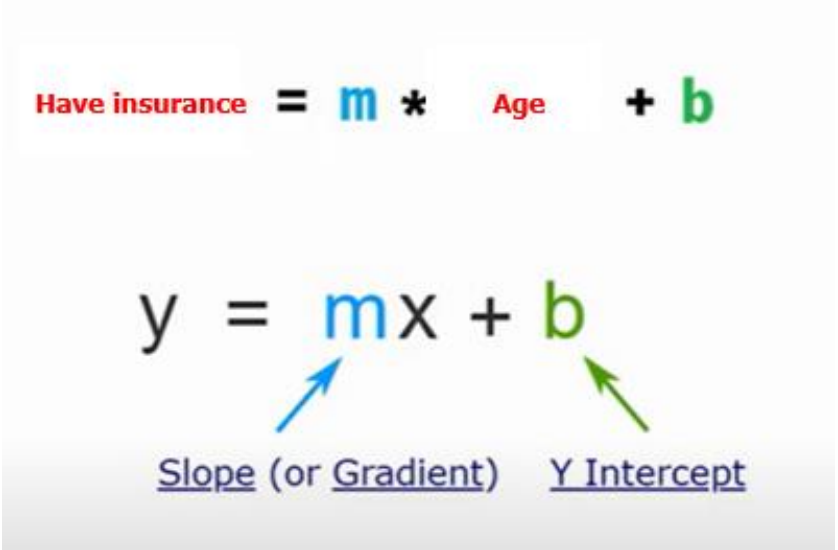
$$y = mx + b$$

Slope (or Gradient) Y Intercept

https://www.mathsisfun.com/algebra/linear-equations.html

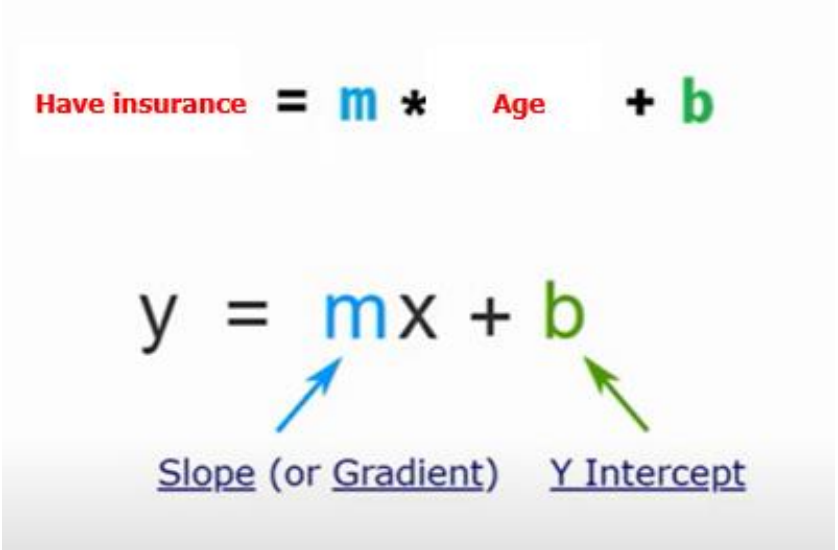
How to Calculate Logistic Regression

age	bought_insurance	
22	0	
25	0	
47	1	
52	0	
46	1	
56	1	
55	0	
60	1	
62	1	
61	1	
18	0	
28	0	
27	0	
29	0	
49	1	



How to Calculate Logistic Regression

age	bought_insurance
22	0
25	0
47	1
52	0
46	1
56	1
55	0
60	1
62	1
61	1
18	0
28	0
27	0
29	0
49	1



Do you know how to calculate m and b?

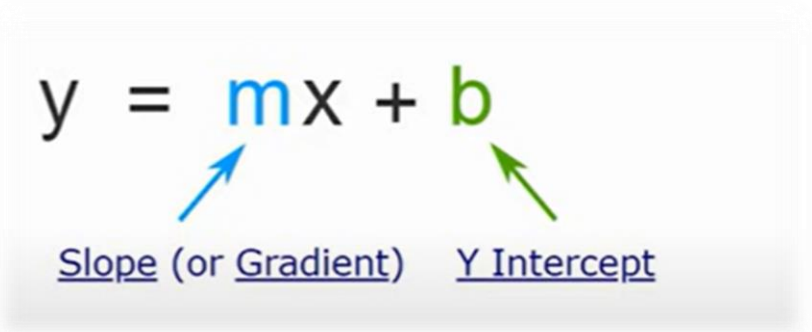
```python
x = [1, 2, 3, 4, 5]
y = [3, 4, 2, 3, 5]

# Mean of x and y
mean_x = sum(x) / len(x)
mean_y = sum(y) / len(y)

# Calculate Summations
numerator = sum((xi - mean_x) * (yi - mean_y) for xi, yi in zip(x, y))
denominator = sum((xi - mean_x)**2 for xi in x)

# Calculate m and c
m = numerator / denominator
c = mean_y - (m * mean_x)

# Prediction function
def predict(x):
    return m * x + c

# Test the function
print(predict(6))  # Output should be the y-value corresponding to x = 6

# Output m and c
print(f"m: {m}, c: {c}")
```

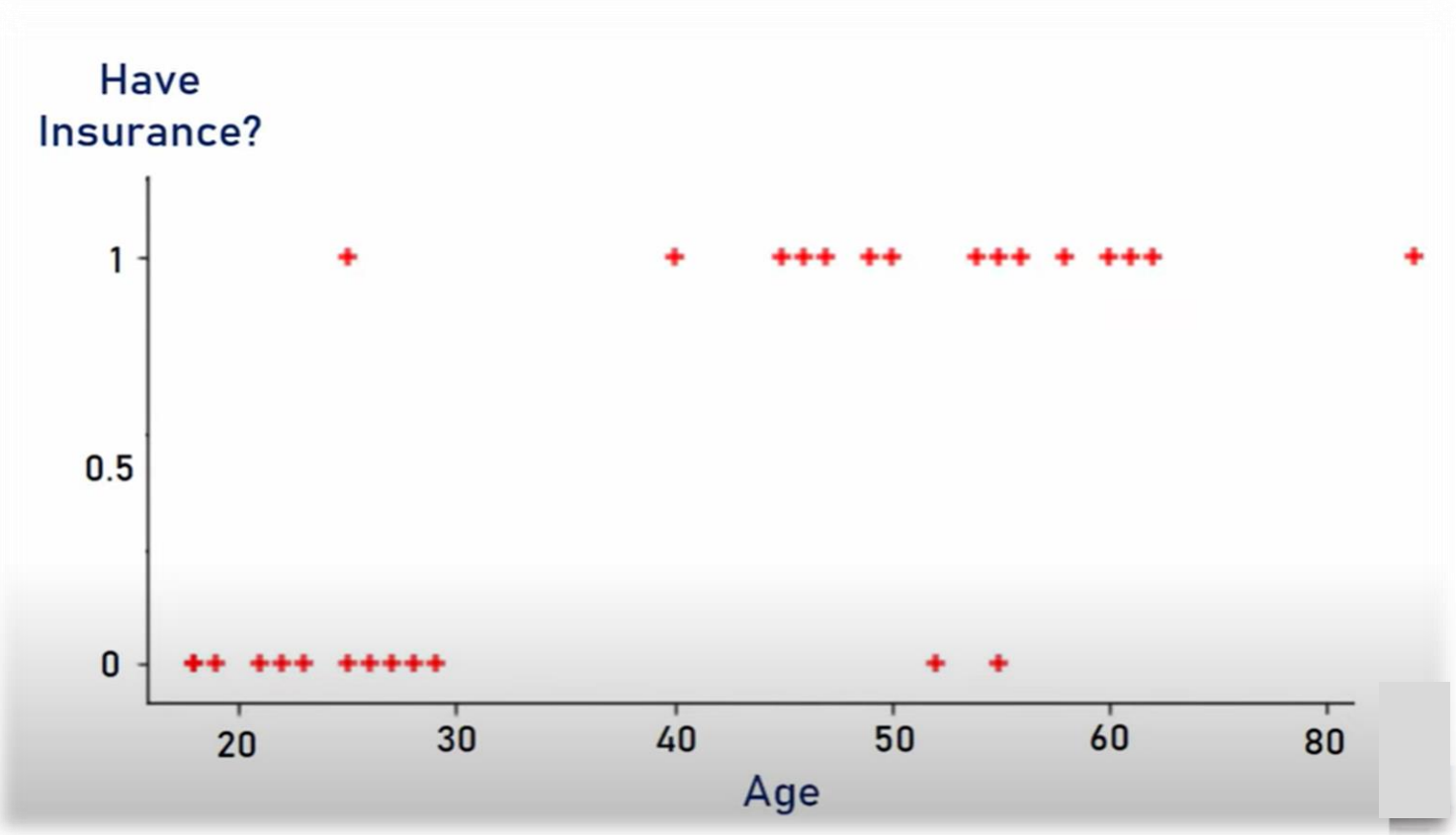
Work outside of class

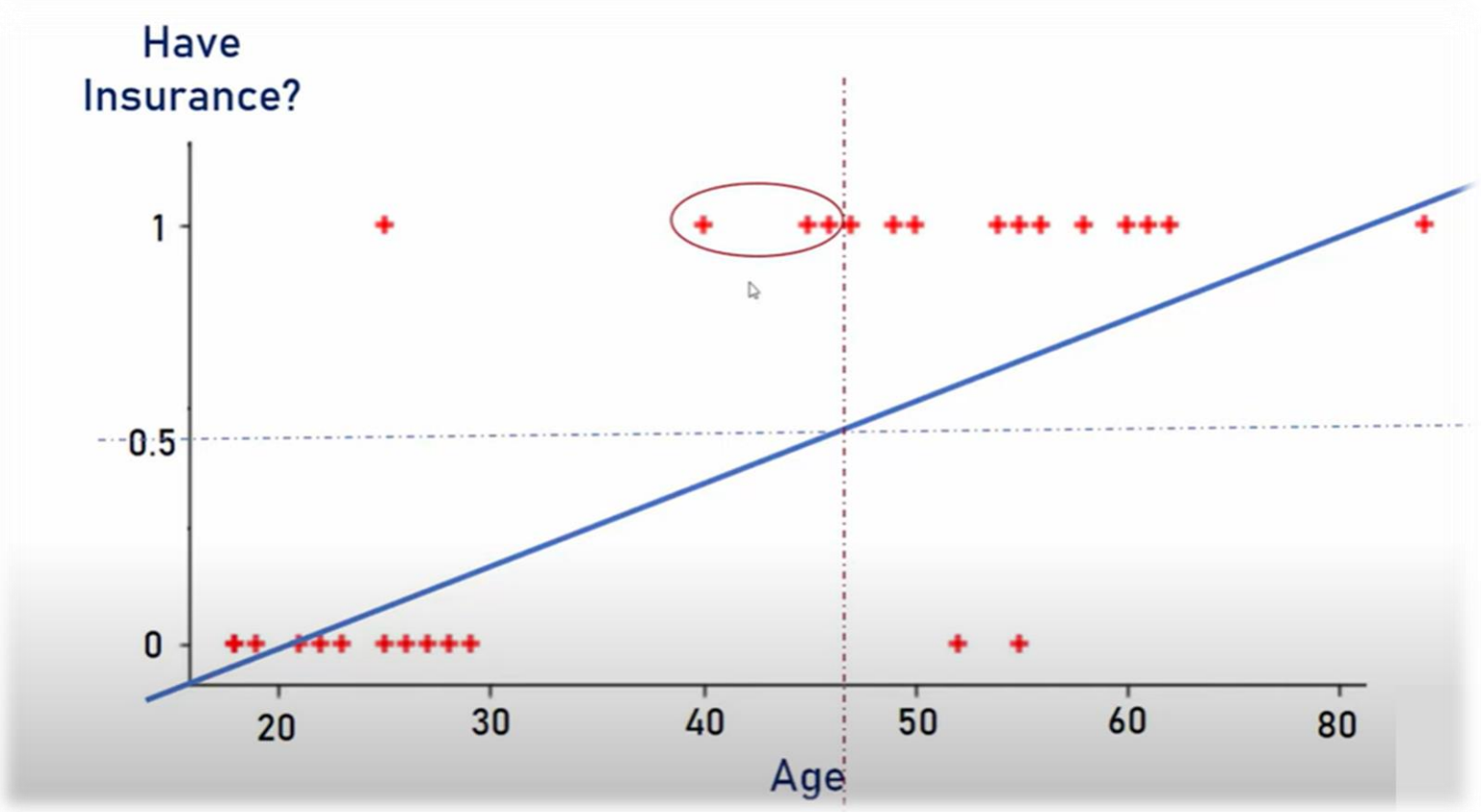
- Compute Logistic Regression on the provided dataset by hand.
- Develop a Python script utilizing the math library to perform Logistic Regression and validate your manual calculations.

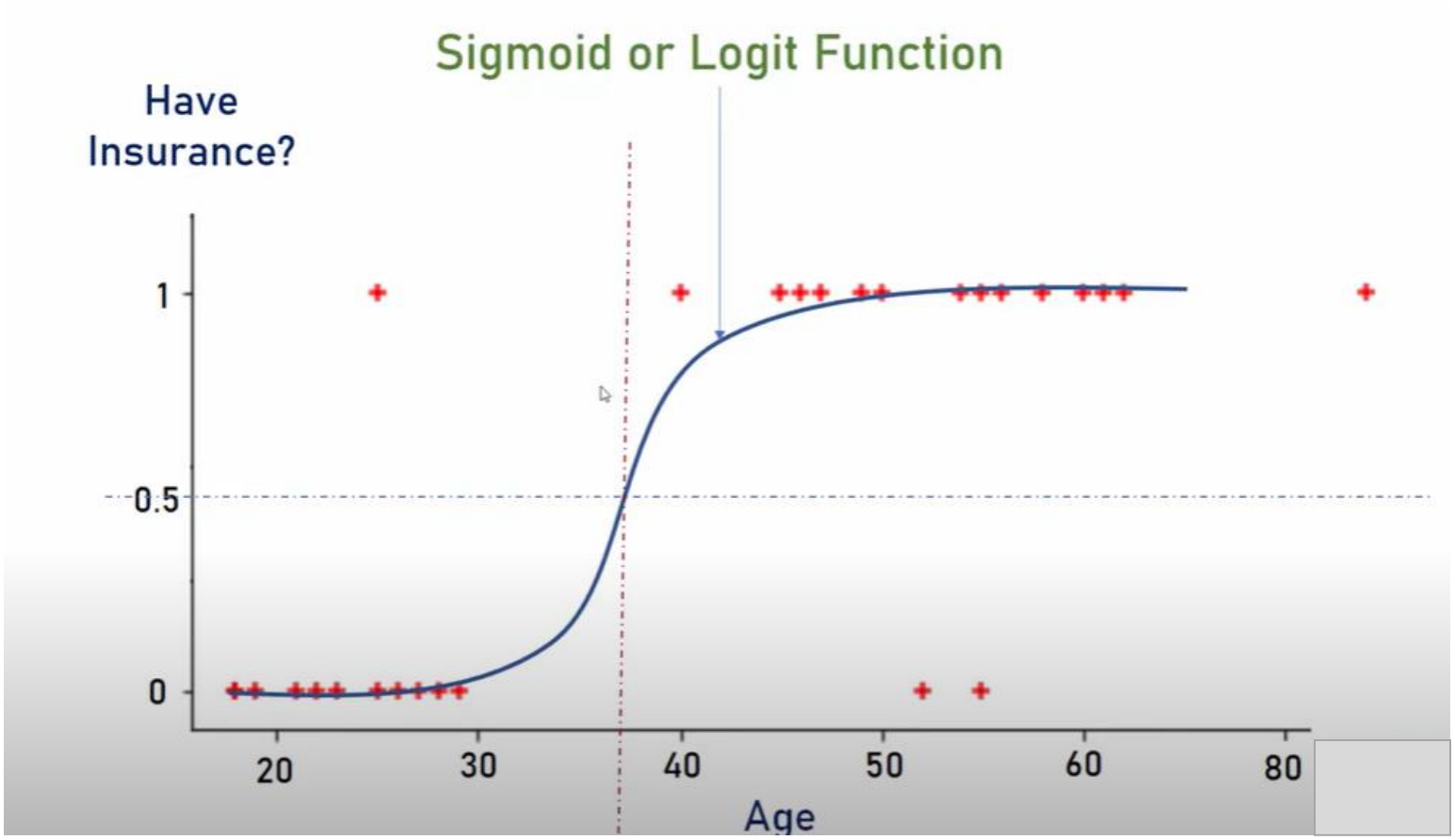
The Result for the Prediction

age	bought_insurance	predicted_insurance
22	0	0.024133191
25	0	0.047808156
47	1	0.575378036
52	0	0.695280281
46	1	0.551397587
56	1	0.791202077
55	0	0.767221628
60	1	0.887123874
62	1	0.935084772
61	1	0.911104323
18	0	0.120054987
28	0	0.119749504
27	0	0.095769055
29	0	0.143729953
49	1	0.623338934




Have Insurance?
1
0.5
0
20
30
40
50
60
80
Age


Have
Insurance?
1
0.5
0
20
30
40
50
60
80
Age


Sigmoid or Logit Function
Have Insurance?
1
0.5
0
20
30
40
50
60
80
Age

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

e = Euler's number ~ 2.71828

$$sigmoid(200) = \frac{1}{1+2.71^{-200}}$$ = almost close to 1

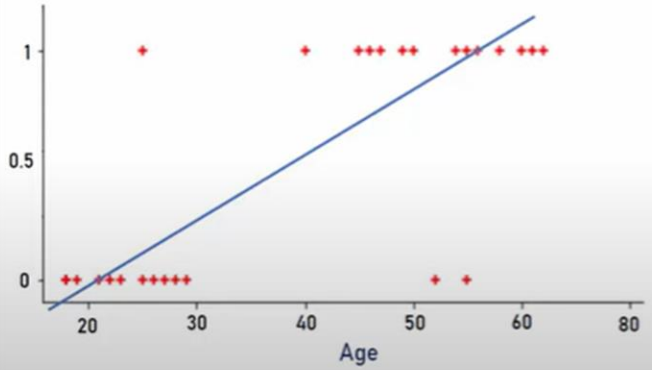
$$sigmoid(-200) = \frac{1}{1+2.71^{200}}$$ = almost close to 0

Sigmoid function converts input into range 0 to 1

Step 1

$$y = m * x + b$$

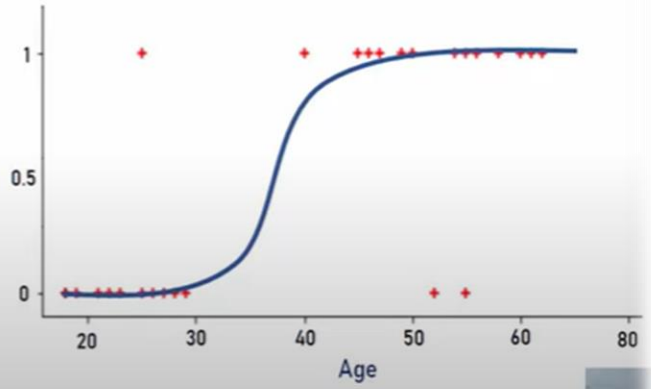
Age



Step 2

$$z = \frac{1}{1 + e^{-y}}$$

If person will buy insurance



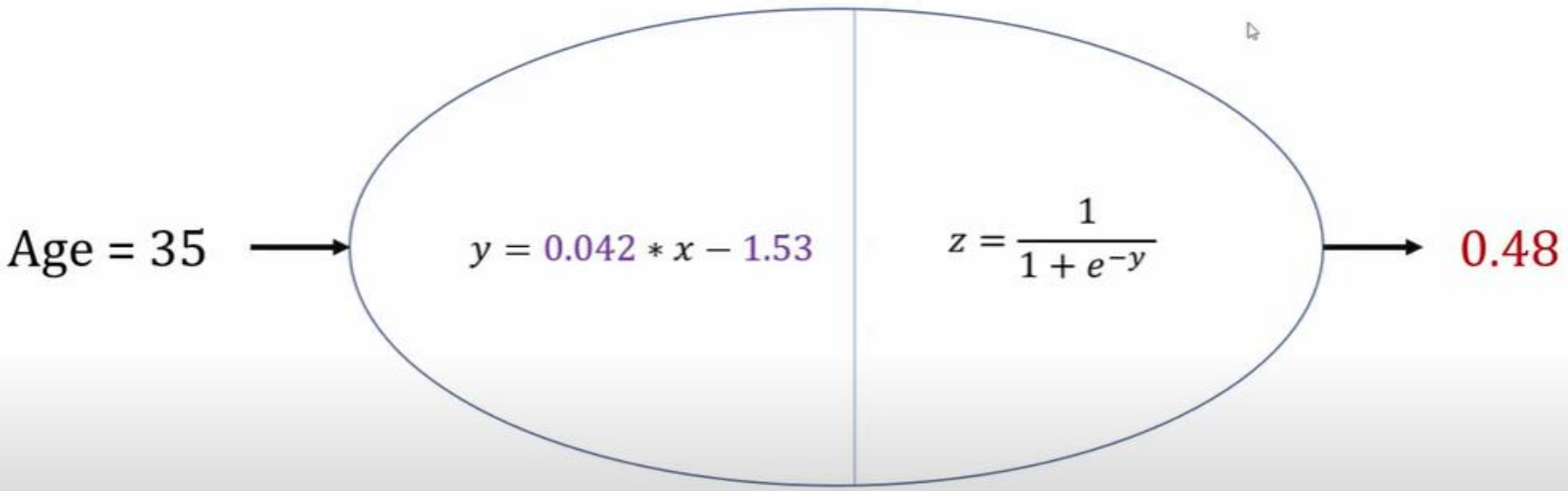
$$y = 0.042 * x - 1.53$$

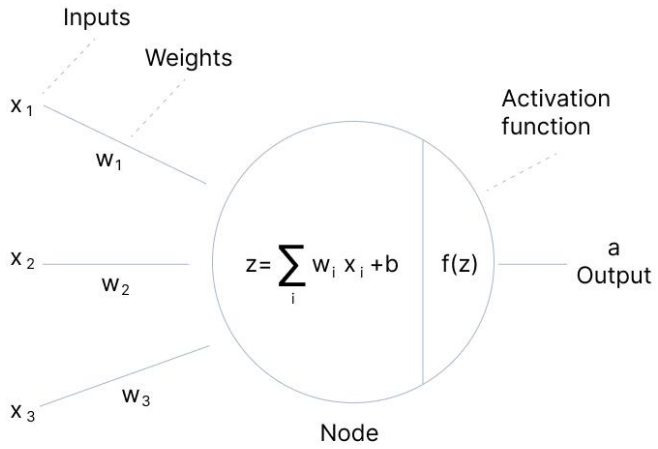
Age

age	bought_insurance	
22	0	
25	0	
47	1	
52	0	
46	1	
56	1	
55	0	
60	1	
62	1	
61	1	
18	0	
28	0	
27	0	
29	0	
49	1	

value < 0.5 = person will not buy insurance

value >= 0.5 = person **will** buy insurance

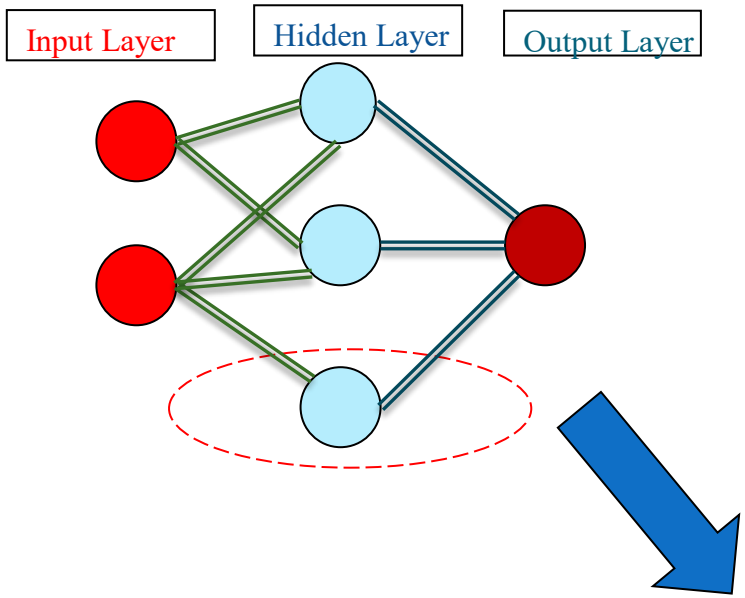


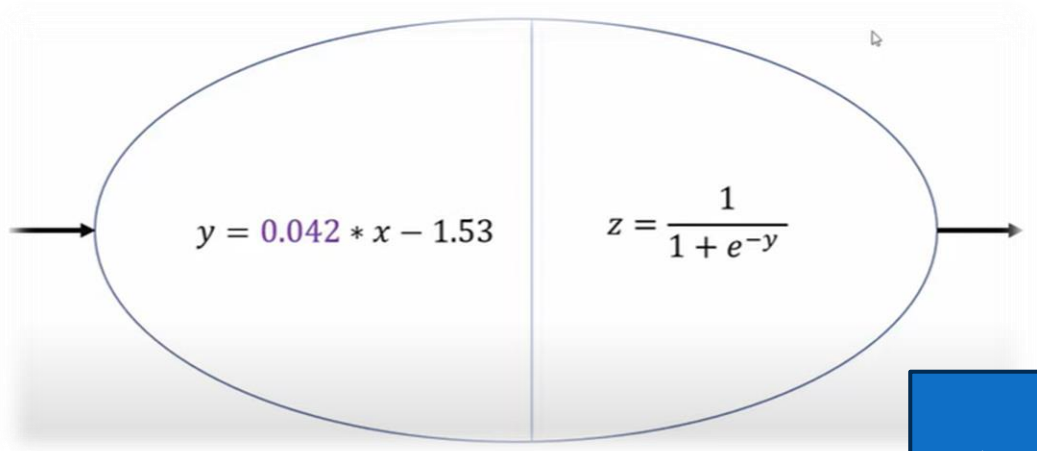


value < 0.5 = person will not buy insurance

value >= 0.5 = person **will** buy insurance

Age = 43 → $y = 0.042 * x - 1.53$ → $z = \frac{1}{1 + e^{-y}}$ → 0.57





Neuron= linear +activation

$$y = mx + b$$

Slope (or Gradient) Y Intercept

$$y = 0.042 * x - 1.53$$

Age

$$y = 0.042 * x1 + 0.008 * x2 + 0.2 * x3 - 1.53$$

Age Income Education

$$y = w1 * x1 + w2 * x2 + w3 * x3 + \boldsymbol{b}$$

$$y = \sum_{i=0}^{n} w^i x^i + b$$

Formulation [edit]

Given a data set $\{y_i,\ x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ of *n* statistical units, a linear regression model assumes that the relationship between the dependent variable *y* and the vector of regressors **x** is linear. This relationship is modeled through a *disturbance term* or *error variable* ε — an unobserved random variable that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

where $^{\mathsf{T}}$ denotes the transpose, so that $\mathbf{x}_i^{\mathsf{T}}\boldsymbol{\beta}$ is the inner product between vectors $\mathbf{x}_i$ and $\boldsymbol{\beta}$.

Often these *n* equations are stacked together and written in matrix notation as

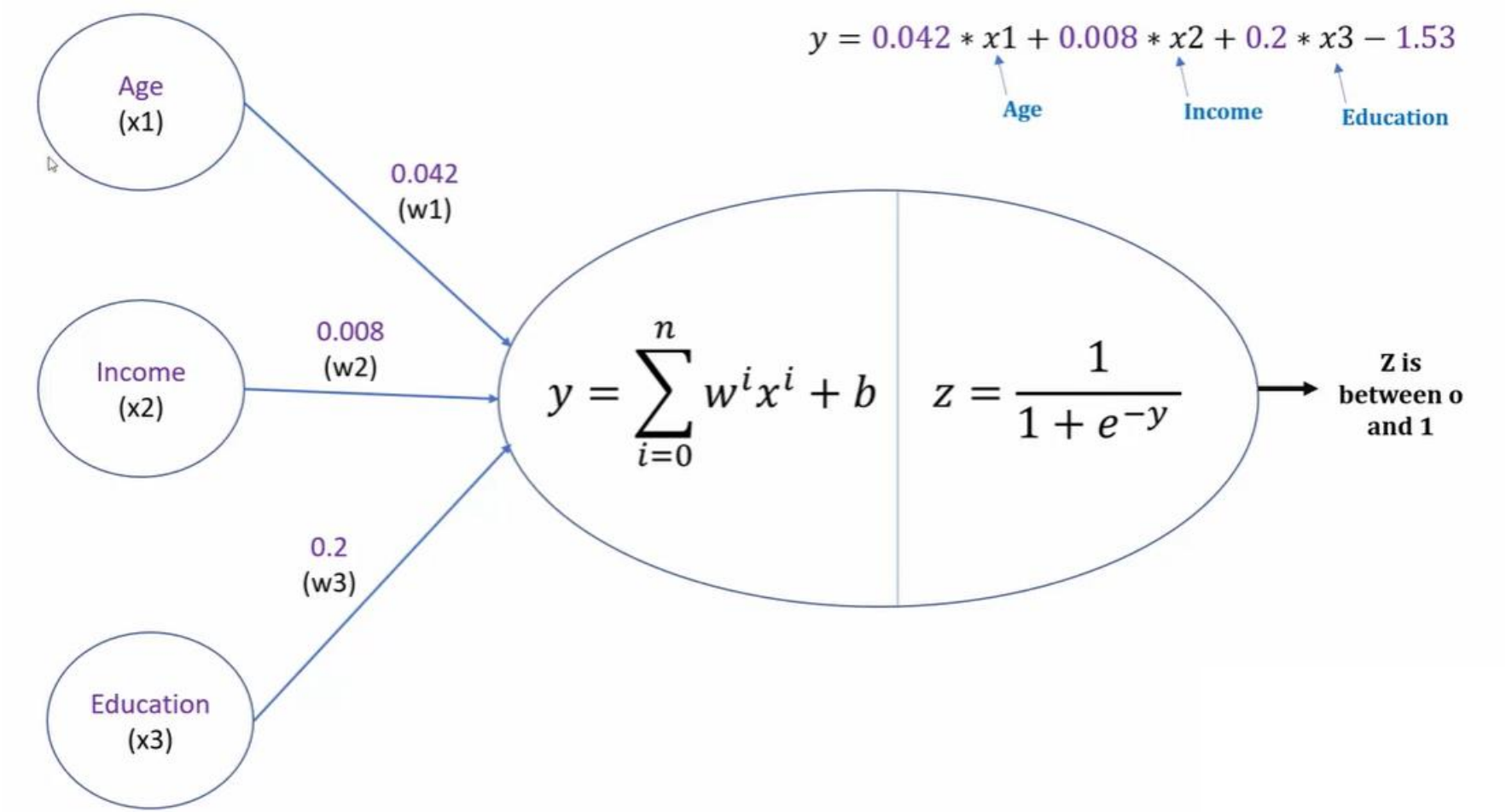
$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

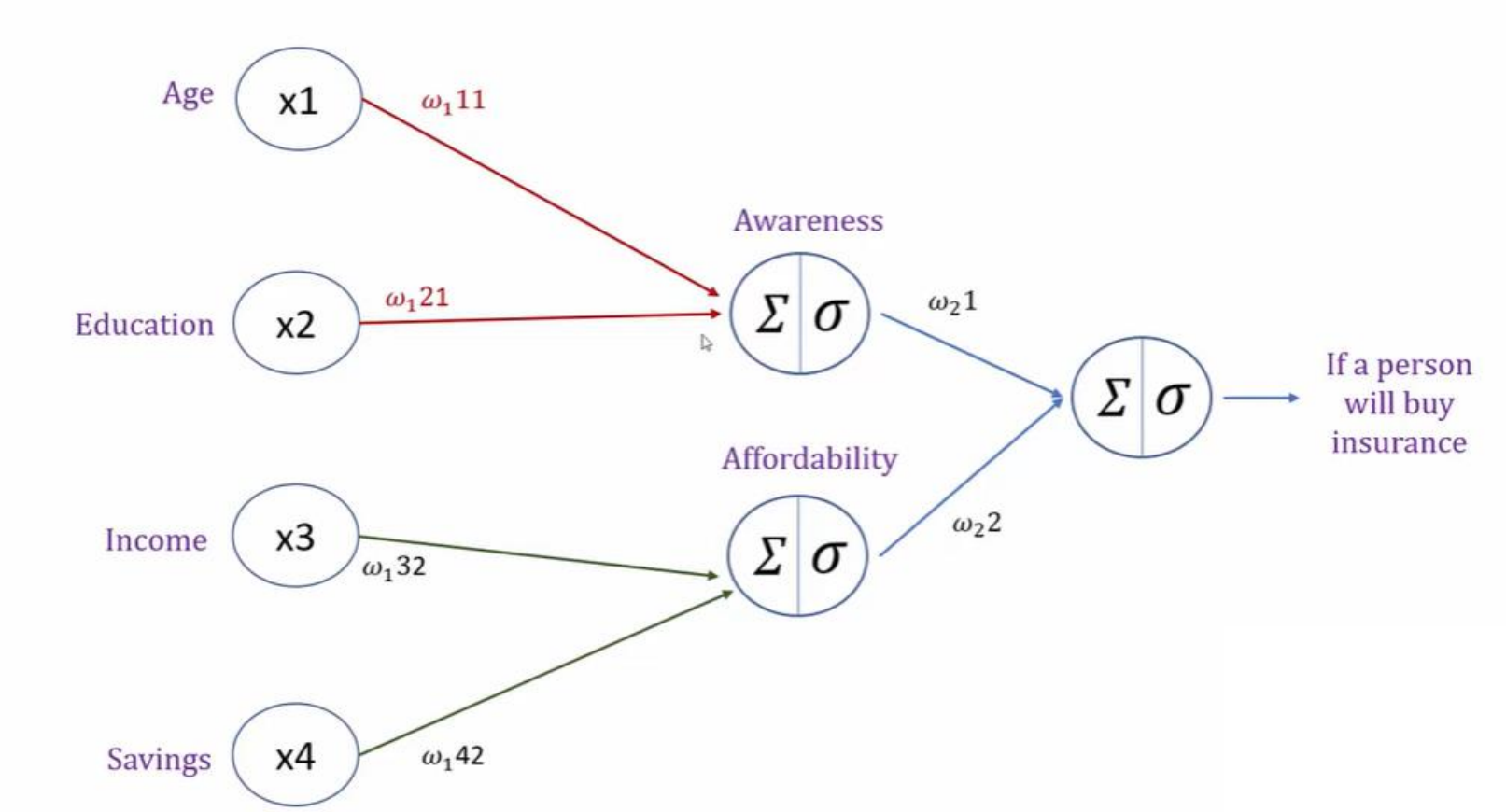
where

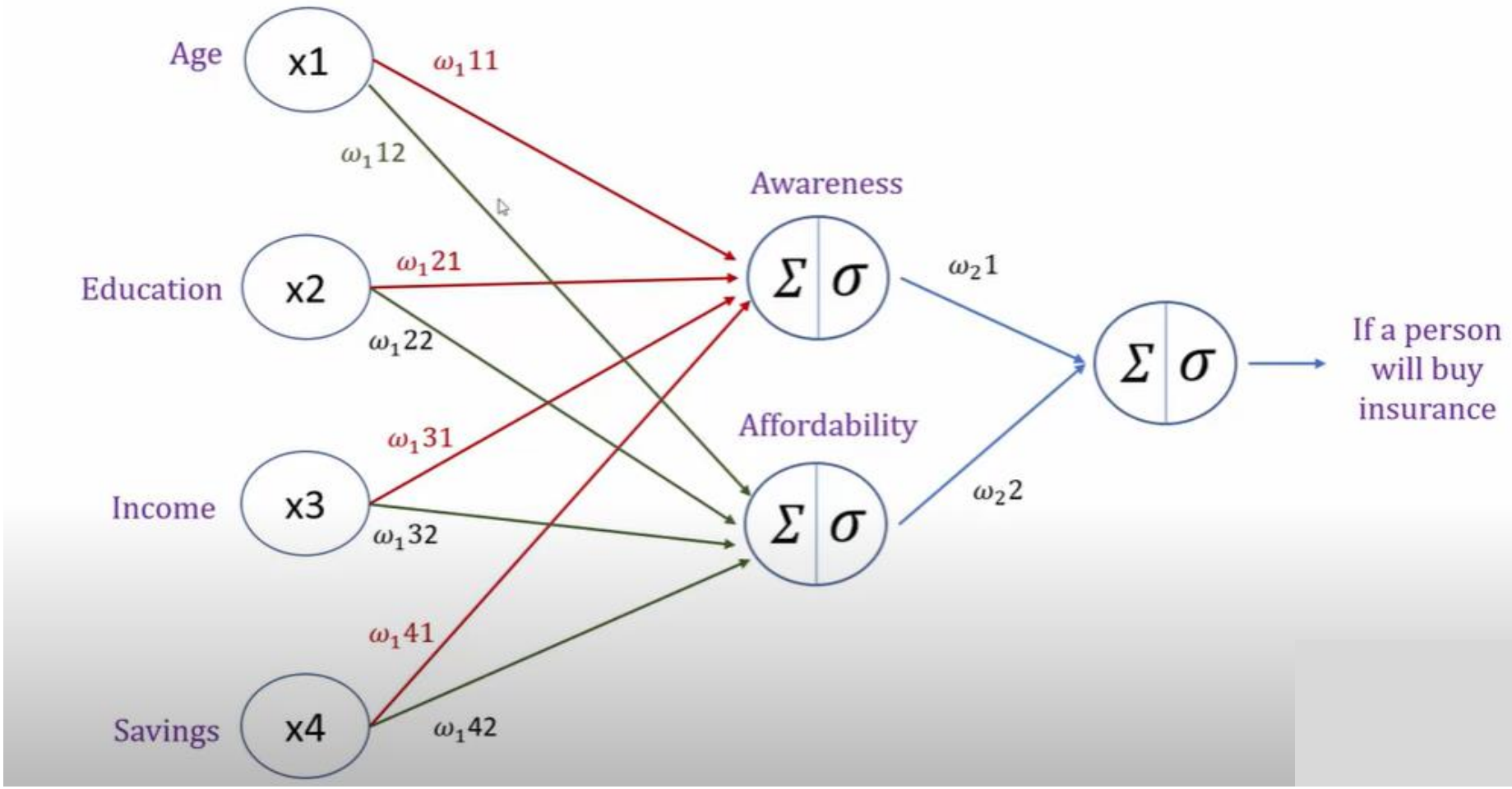
$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \mathbf{x}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_n^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix},$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$


$y = 0.042 * x1 + 0.008 * x2 + 0.2 * x3 - 1.53$
Age
Income
Education
Age
(x1)
Income
(x2)
Education
(x3)
0.042
(w1)
0.008
(w2)
0.2
(w3)
$y = \sum_{i=0}^{n} w^i x^i + b$
$z = \frac{1}{1 + e^{-y}}$
Z is between o and 1


Age
x1
$\omega_1 11$
Awareness
Education
x2
$\omega_1 21$
Σ σ
$\omega_2 1$
Σ σ
If a person will buy insurance
Affordability
Income
x3
$\omega_1 32$
Σ σ
$\omega_2 2$
Savings
x4
$\omega_1 42$


Age
x1
Education
x2
Income
x3
Savings
x4
$\omega_1 11$
$\omega_1 12$
$\omega_1 21$
$\omega_1 22$
$\omega_1 31$
$\omega_1 32$
$\omega_1 41$
$\omega_1 42$
Awareness
Affordability
Σ σ
$\omega_2 1$
$\omega_2 2$
If a person will buy insurance

Activation Functions: Sparking Neurons

Activation functions introduce non-linearity to neural networks.

They determine whether a neuron should be activated or not based on the weighted sum of inputs.

```python
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Input values
inputs = np.array([-1, 0, 1, 2, 3])

# Apply sigmoid activation function
output = sigmoid(inputs)

# Print output
print(output)
```


Any
Questions

Why are activation functions necessary?

1. **The sigmoid activation function maps its input to a range between:**
    1. a) -1 and 1
    2. b) 0 and 1
    3. c) 0 and infinity
    4. d) -infinity and infinity

2. **Sigmoid is commonly used in:**
    1. a) Hidden layers of neural networks
    2. b) Input layers of regression tasks

Take Home Points

1. **Forward Pass:** Feedforward propagation is the process of passing input data through the neural network to compute the predicted output. It involves flowing data from the input layer through the hidden layers to the output layer.
2. **Activation Functions:** During the forward pass, the input to each neuron is transformed using an activation function. This introduces nonlinearity to the network and allows it to capture complex relationships within the data.
3. **Weighted Sum:** At each neuron, the inputs are multiplied by corresponding weights, and the weighted sum is computed. The bias term is added to the sum.
4. **Output Calculation:** After calculating the weighted sum, the output of the neuron is obtained by applying the activation function to the weighted sum.
5. **Propagation:** The output of one layer serves as the input to the next layer. This process continues until the final output is computed.


Any
Questions

Lecture Outline

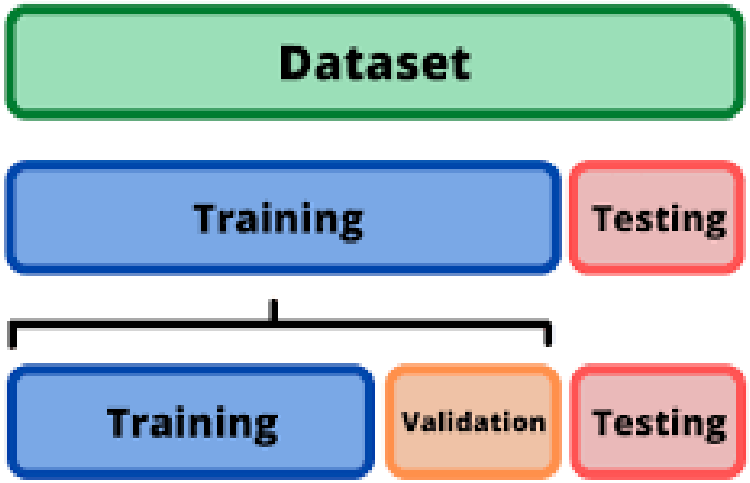
1. **Introduction to Machine Learning**
2. **Fundamentals of Neural Networks**
   - **Motivating Example**
   - **Definition and Structure: Building Blocks**
   - **Activation Functions: Sparking Neurons**
   - **Feedforward Propagation: Information Flow**
3. **Learning and Training**
   - **Loss Functions: Measuring Errors**
   - **Backpropagation: Fine-Tuning Weights**
4. **Hands-on Exercises**
5. **Real-World Applications**
6. **Summery**
7. **References**

Neuron= linear +activation

Model= Architecture
+Parameters

Loos function/Cost Function

- What does it mean training and testing data in machine learning



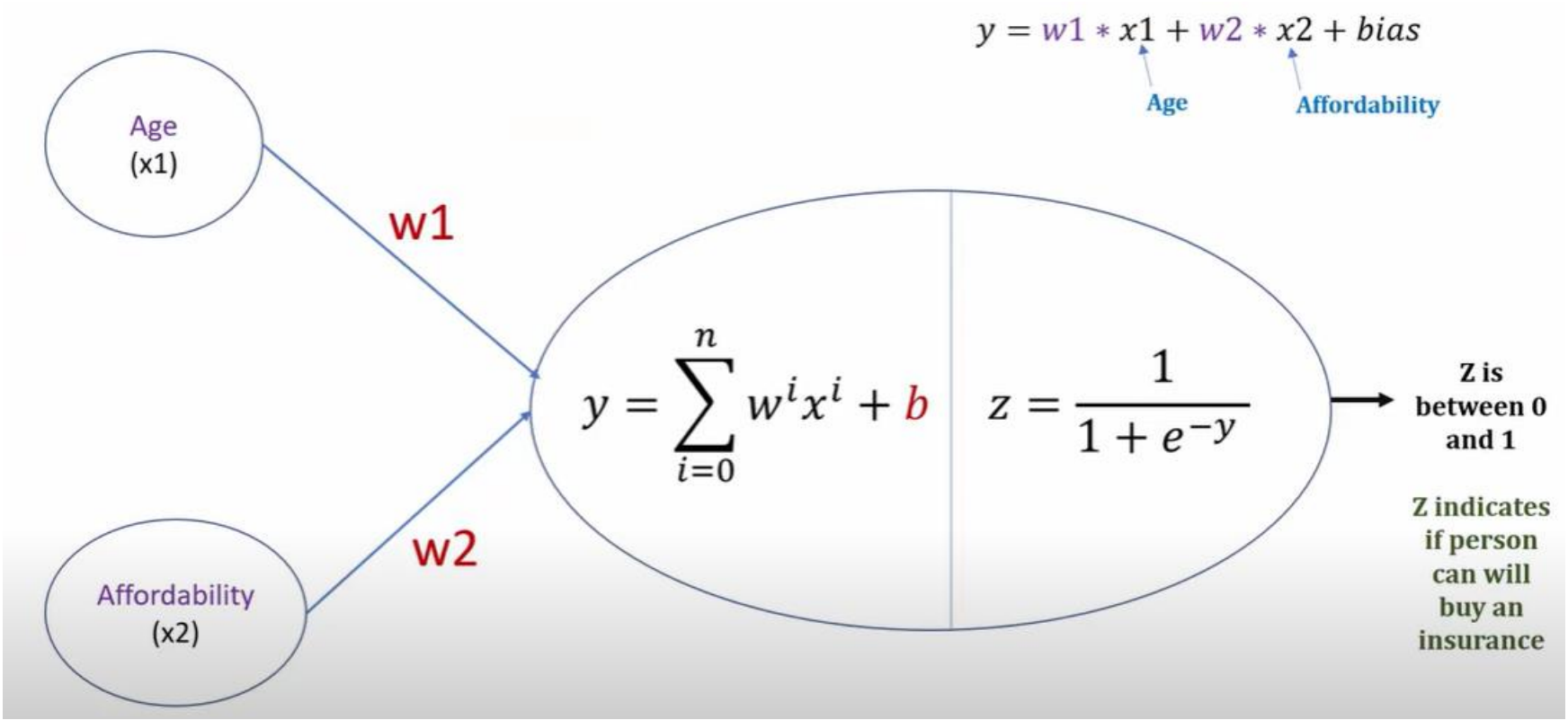
The Result for the Prediction

age	bought_insurance	predicted_insurance
22	0	0.024133191
25	0	0.047808156
47	1	0.575378036
52	0	0.695280281
46	1	0.551397587
56	1	0.791202077
55	0	0.767221628
60	1	0.887123874
62	1	0.935084772
61	1	0.911104323
18	0	0.120054987
28	0	0.119749504
27	0	0.095769055
29	0	0.143729953
49	1	0.623338934

The goal is to reduce the error between predictions and true values.

x		y
age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

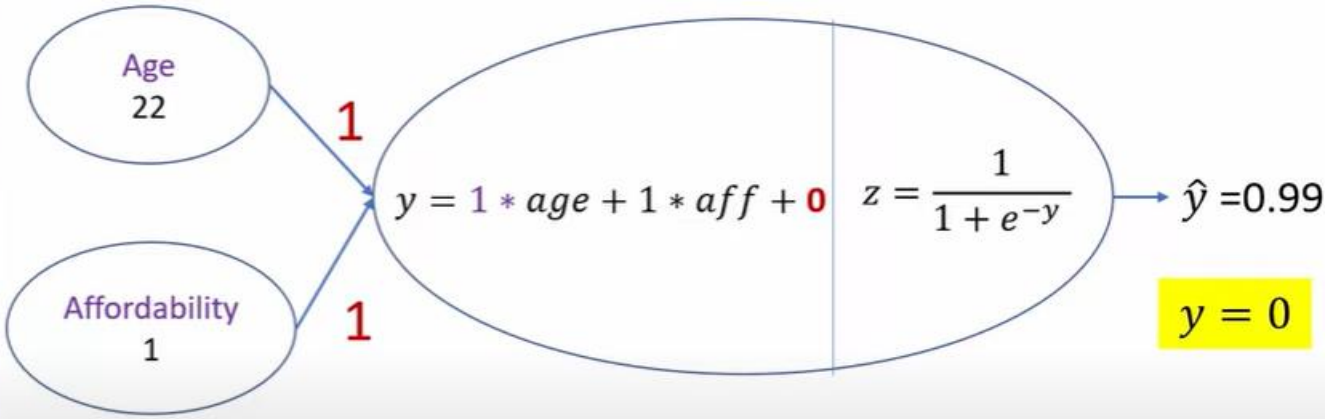
$$y = f(x)$$


$y = w1 * x1 + w2 * x2 + bias$
Age
Affordability
Age
(x1)
w1
w2
Affordability
(x2)
$y = \sum_{i=0}^{n} w^i x^i + b$
$z = \frac{1}{1 + e^{-y}}$
Z is between 0 and 1
Z indicates if person can will buy an insurance

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

$$y = w1 * x1 + w2 * x2 + bias$$

Age Affordability



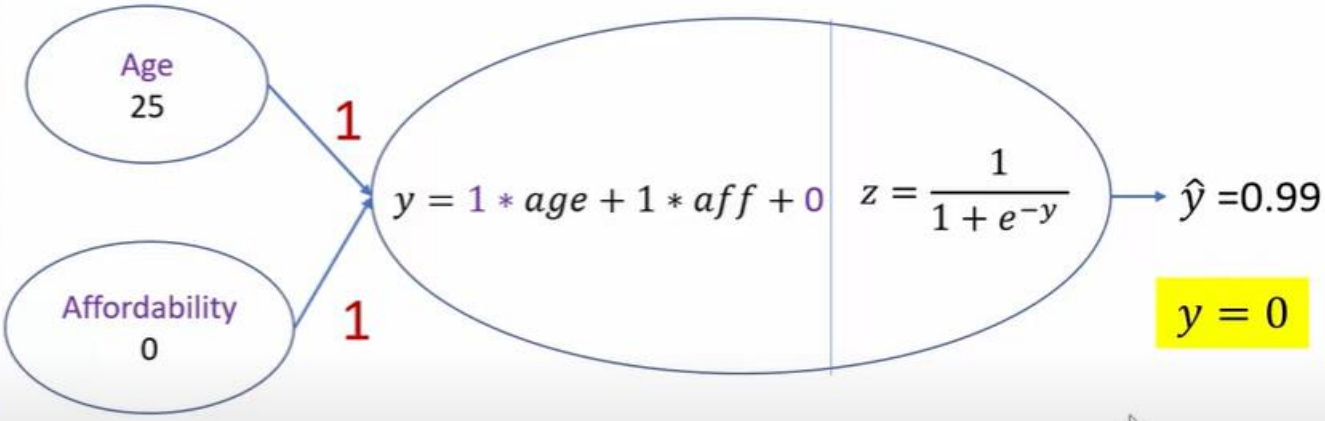
$y = 0$

$error_1 = \text{abs}(y - \hat{y})$
$= 0.99$

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

$$y = w1 * x1 + w2 * x2 + bias$$

Age Affordability



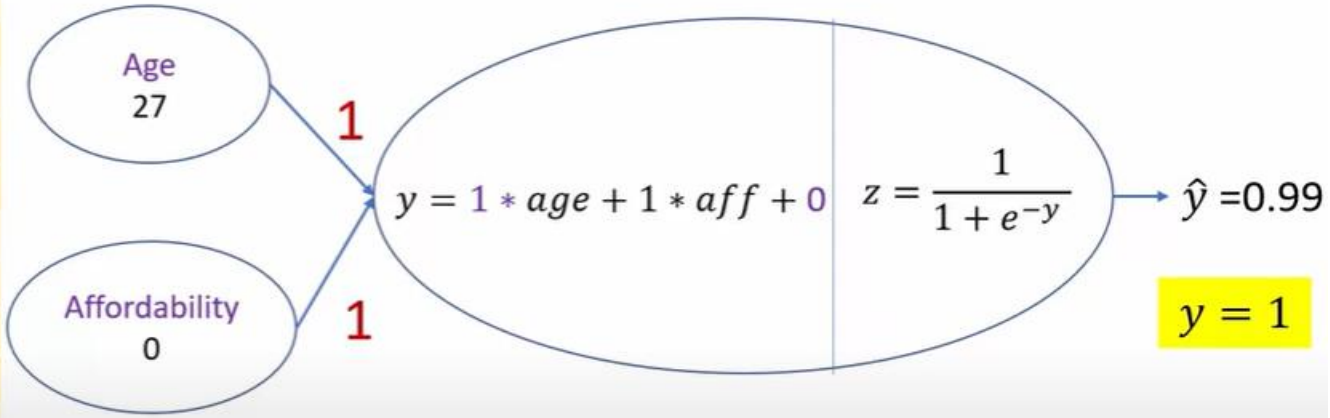
$y = 0$

$$error_2 = abs(y - \hat{y})$$
$$= 0.99$$

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

$$y = w1 * x1 + w2 * x2 + bias$$

Age Affordability



$$error_{13} = abs(y - \hat{y})$$
$$= 0.01$$

Total error = $error_1 + error_2 + \ldots + error_{13}$

$$= \sum_{i=1}^{n} abs(y_i - \hat{y}_i)$$

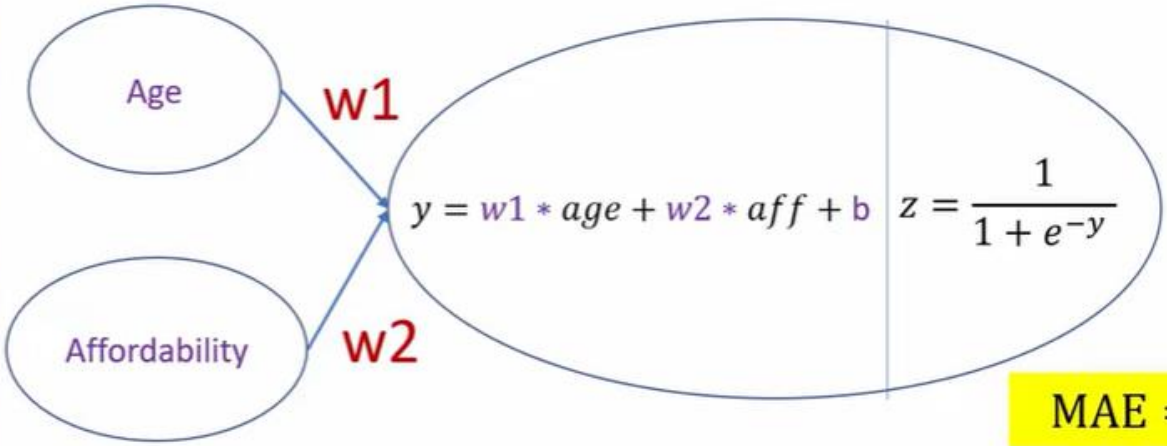
$$Mean\ Absolute\ Error\ (MAE) = \frac{1}{n}\sum_{i=1}^{n} abs(y_i - \hat{y}_i)$$

Mean Squared Error (MSE)

1. **Differentiability:** MSE is a smooth and differentiable function, which is important for optimization algorithms like gradient descent. Neural networks are trained using optimization algorithms that rely on gradients, so having a differentiable cost function is crucial.
2. **Mathematical Simplicity:** The mathematical properties of MSE make it easy to work with. Its derivative with respect to the model's parameters can be calculated analytically, which simplifies the optimization process.
3. **Classification Tasks:** For classification tasks, MSE is generally not used. Instead, cross-entropy loss functions (like binary cross-entropy or categorical cross-entropy) are more common, as they are better suited for probability distribution estimation tasks.

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	0

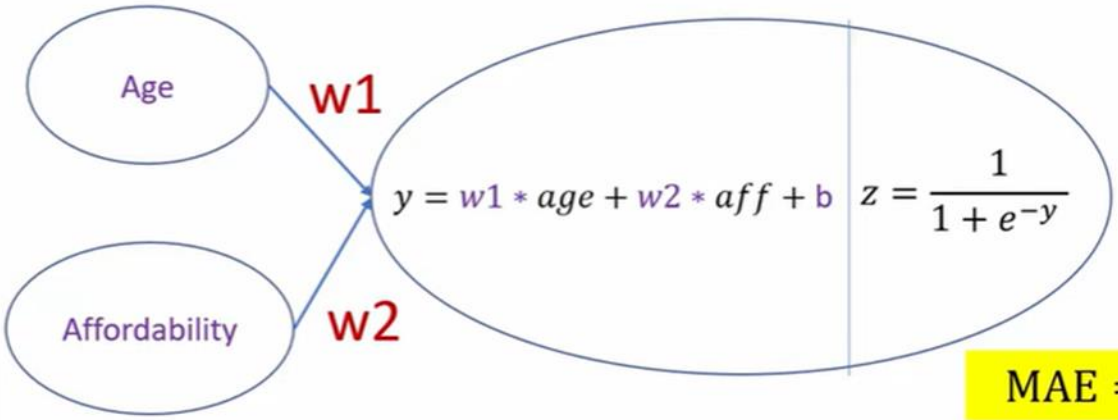
After first **epoch**



MAE = 10.02

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	0

After first **epoch**



$\text{MAE} = 10.02$

```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_flattened, y_train, epochs=5)
```

$$Mean\ Absolute\ Error\ (MAE) = \frac{1}{n}\sum_{i=1}^{n} abs(y_i - \hat{y}_i)$$

```
model.compile(optimizer='adam',
              loss='mean_absolute_error',
              metrics=['accuracy'])
```

$$Mean\ Squared\ Error\ (MSE) = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

```
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

$$Log\ loss\ or\ binary\ cross\ entropy = -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Off class work

- Implement **mean square error** in two way:
  - Without using NumPy(using plain Python)
  - With the use of NumPy.

Why we use log loss for logistic Regression

1. **Probabilistic Interpretation:** Logistic regression aims to model the probability that a given input belongs to a particular class. Log loss is well-suited for probabilistic models as it measures the difference between predicted probabilities and actual class labels.
2. **Well-Established Metric:** Log loss is a well-established metric in the field of machine learning, making it a common choice for logistic regression.
3. **Mathematical Simplicity:** Log loss leads to relatively simple mathematical formulations, making it easier to work with during training and analysis.
4. **Sensitivity to Confidence:** Log loss penalizes confident incorrect predictions more heavily than less confident ones. This aligns with the goal of logistic regression, where confident misclassifications should be more penalized.
5. **Maximum Likelihood Estimation:** Minimizing log loss in logistic regression is equivalent to maximizing the likelihood of the observed data under the assumed probability distribution. This connection to maximum likelihood estimation makes log loss a natural fit.
6. **Regularization:** Log loss can be extended to include regularization terms, which helps prevent overfitting and improves the generalization capability of the model.

Simple Example using python

In [2]:
```
import numpy as np
```

In [3]:
```
y_predicted = np.array([1,1,0,0,1])
y_true = np.array([0.30,0.7,1,0,0.5])
```

Implement Mean Absolute Error

In [4]:
```
def mae(y_predicted, y_true):
    total_error = 0
    for yp, yt in zip(y_predicted, y_true):
        total_error += abs(yp - yt)
    print("Total error is:",total_error)
    mae = total_error/len(y_predicted)
    print("Mean absolute error is:",mae)
    return mae
```

In [5]:
```
mae(y_predicted, y_true)
```

```
Total error is: 2.5
Mean absolute error is: 0.5
```

Weights Initialization

- **Random Initialization:** Setting weights to small random values (e.g., from a Gaussian distribution) helps break symmetry. However, care must be taken to avoid very small or very large initial weights.
- **Xavier/Glorot Initialization:** This technique sets the initial weights based on the number of input and output neurons of a layer. It aims to keep the signal's variance constant across layers, which is important for deeper networks.

Cost Function/Loss Function

- A loss function, also known as a cost function or objective function, measures the discrepancy between the predicted outputs of a neural network and the actual target outputs (ground truth) from the training data.
- The goal is to minimize this discrepancy, which essentially means reducing the error between predictions and true values.
- The choice of the loss function depends on the problem type. For regression tasks, **Mean Squared Error (MSE)** is commonly used, while for classification tasks, **Cross-Entropy loss** is often preferred.


Any
Questions

The Result for the Prediction

age	bought_insurance	predicted_insurance
22	0	0.024133191
25	0	0.047808156
47	1	0.575378036
52	0	0.695280281
46	1	0.551397587
56	1	0.791202077
55	0	0.767221628
60	1	0.887123874
62	1	0.935084772
61	1	0.911104323
18	0	0.120054987
28	0	0.119749504
27	0	0.095769055
29	0	0.143729953
49	1	0.623338934

The goal is to minimize the Cost/loss function.

Gradient Descent

- Gradient Descent is an optimization algorithm used to minimize the loss function.
- It works by iteratively adjusting the parameters (weights and biases) of the neural network in the direction that decreases the loss function.
- The algorithm computes the gradient of the loss function with respect to each parameter. The gradient indicates the direction of the steepest increase in the loss. By taking the negative of the gradient (to move in the opposite direction), the algorithm updates the parameters, aiming to reduce the loss.
- This process involves choosing a learning rate that determines the step size for each parameter update. The learning rate balances the trade-off between making rapid progress and potentially overshooting the optimal point.
- There are variants of Gradient Descent, such as Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent, and more advanced methods like Adam and RMSprop, which introduce adaptive learning rates and momentum to improve convergence speed and stability.

Gradient Descent

- The process of training a neural network involves adjusting the weights based on the data and the desired outcomes.
- The network learns from the data by iteratively updating the weights to minimize the difference between its predictions and the actual target values.
- This process is typically achieved through optimization algorithms like gradient descent.

x	y
2	4
3	6
5	10
7	14
9	18

$y =$

x	y
2	4
3	6
5	10
7	14
9	18

$$y = x * 2$$

x	y
2	6
3	9
5	15
7	21
9	27

x	y
2	6
3	9
5	15
7	21
9	27

$$y = x * 3$$

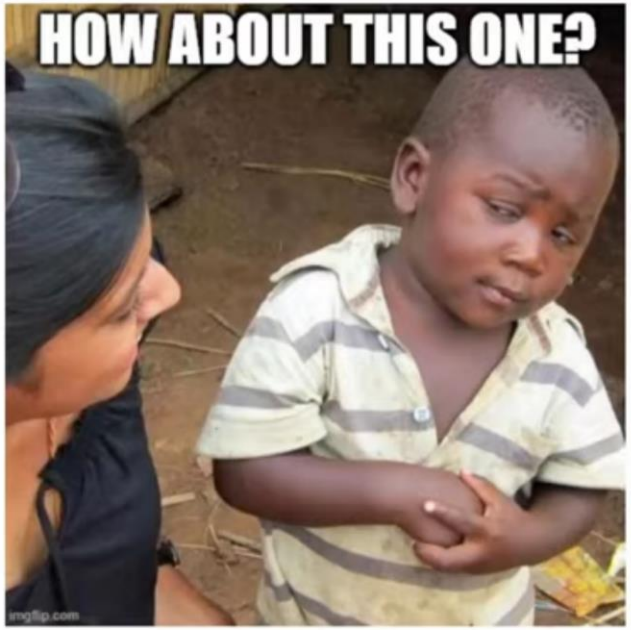
x	y
2	7
3	9
5	13
7	17
9	21

$y =$

x	y
2	7
3	9
5	13
7	17
9	21

$$y = x * 2 + 3$$

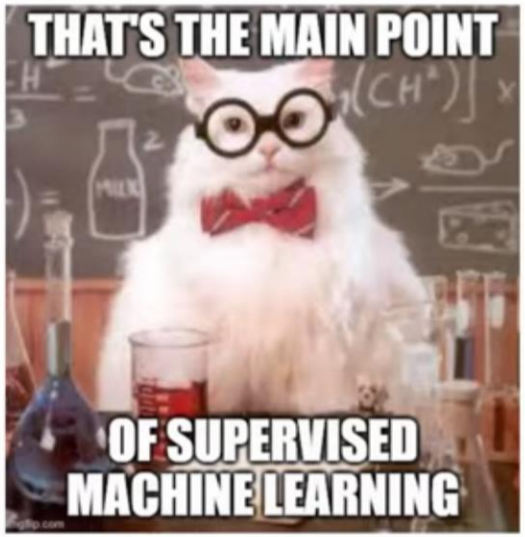
x	y
2	-0.5
3	0
5	1
7	2
9	3



x	y
2	-0.5
3	0
5	1
7	2
9	3

Prediction Function

$$y = x * 0.5 - 1.5$$



x	y
2	-0.5
3	0
5	1
7	2
9	3

$$y = x * 0.5 \ - 1.5$$

Gradient descent is an algorithm that finds best fit line for given training data set

x	y
2	-0.5
3	0
5	1
7	2
9	3

Technique called Gradient Descent
can help you find these parameters

$$y = x * 0.5 - 1.5$$

x	y
2	-0.5
3	0
5	1
7	2
9	3

Technique called Gradient Descent
can help you find these parameters

$$y = x * 0.5 - 1.5$$

Weight Bias

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

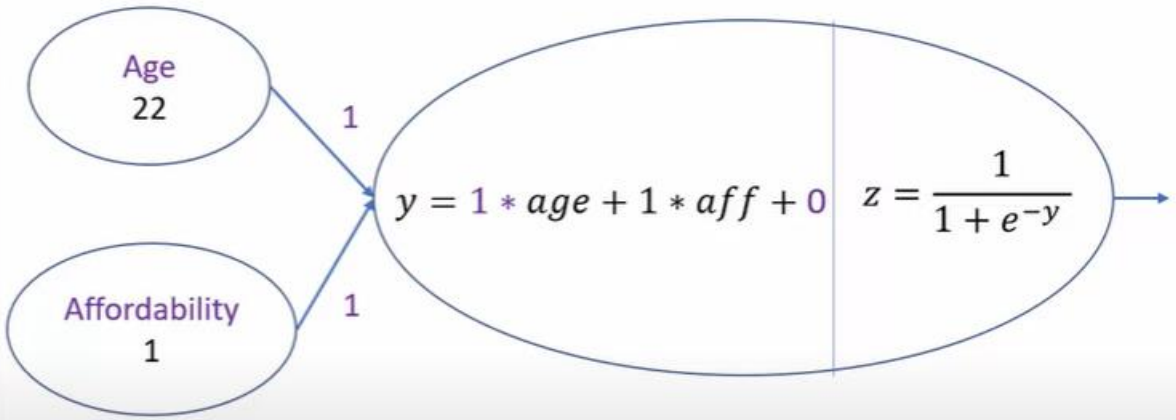
Binary Classification

Based on age and affordability, come up with a **function** that can predict if person will buy insurance or not

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

$$y = w1 * x1 + w2 * x2 + bias$$

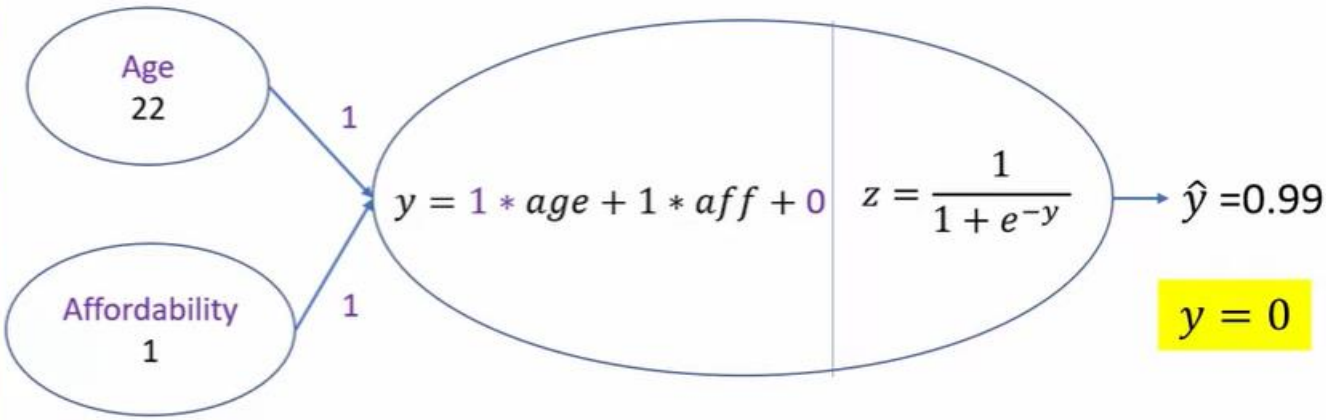
Age Affordability



age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1

$$y = w1 * x1 + w2 * x2 + bias$$

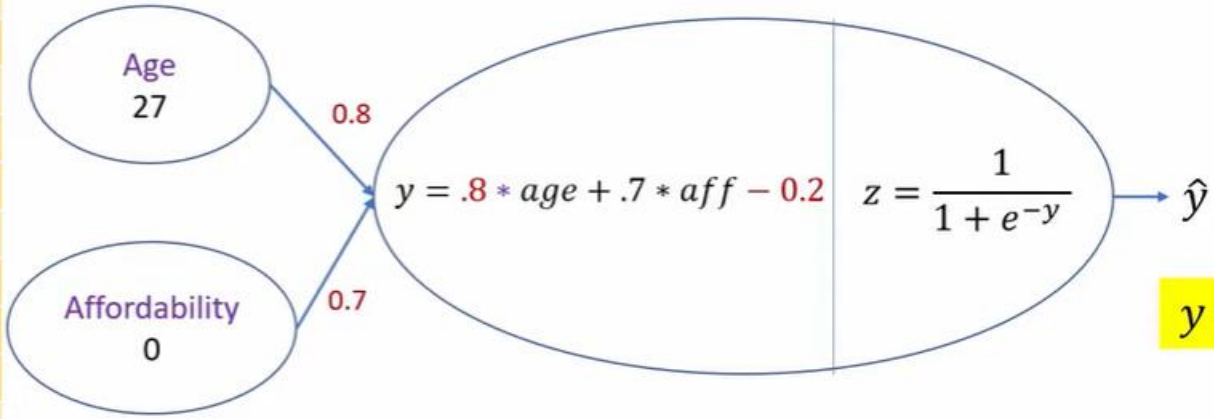
Age Affordability



$$error1 = -(y\log(\hat{y}) + (1 - y)\log(1 - \hat{y}))$$

= 4.6

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1



$y = 0$

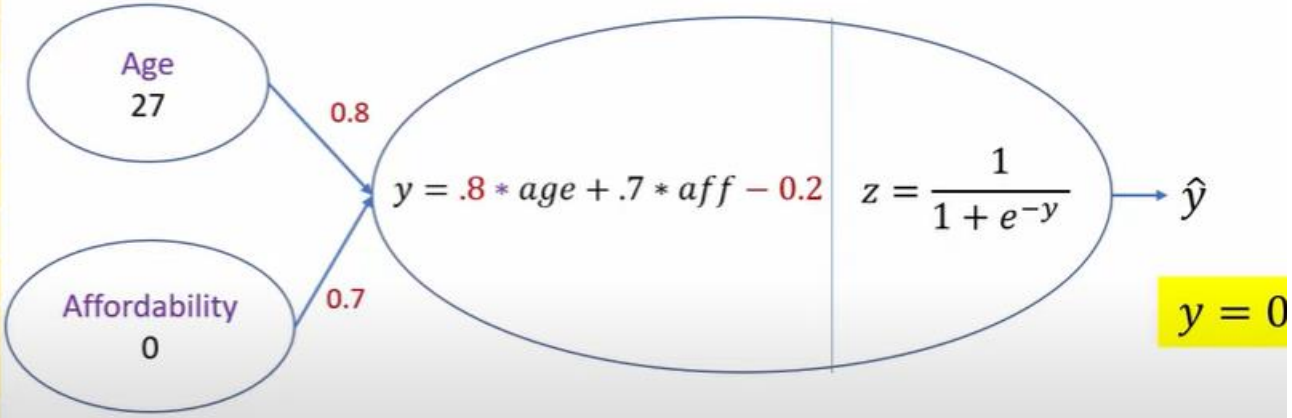
$$error13 = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

Total error = $error_1 + error_2 + \ldots + error_{13}$

End of **second epoch**

```
model = keras.Sequential([
    keras.layers.Dense(10, input_shape=(784,), activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train_flattened, y_train, epochs=5)
```

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1



$$error13 = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

Total error = $error_1 + error_2 + \ldots + error_{13}$

Adam(Adaptive Moment Estimation) optimizer

- the **Adam optimizer** extends the basic gradient descent algorithm by introducing adaptive learning rates, momentum-like behavior, and bias correction. It's a popular choice for training neural networks due to its ability to handle a wide range of problems effectively without requiring extensive manual tuning of learning rates.

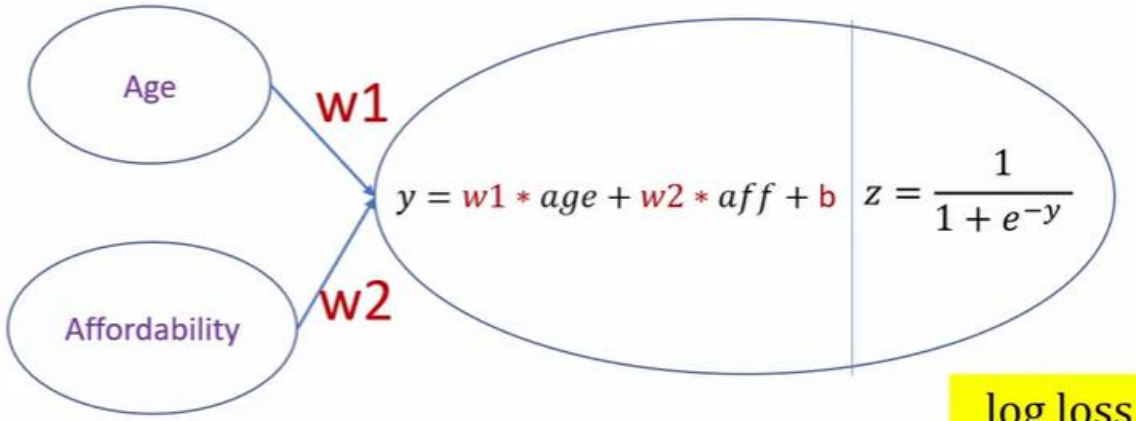
Total error = $error_1 + error_2 + \ldots + error_{13}$

$$Log\ loss\ or\ binary\ cross\ entropy = -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

Loss

After **first epoch**

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	0



$\log \text{loss} = 4.31$

$$Log\ loss\ or\ binary\ cross\ entropy = -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \quad \cdot \log(1 - \hat{y}_i)$$

Loss

$$w1 = w1 - something$$

$$w1 = w1 - something$$

$$w1 = w1 - learning\ rate * {}^{\partial}/_{\partial w1}$$

$$\mathrm{w1} = \mathrm{w1} - something$$

$$\mathrm{w1} = \mathrm{w1} - learning\ rate * {}^{\partial}/_{\partial w1}$$

$$\mathrm{w2} = \mathrm{w2} - learning\ rate * {}^{\partial}/_{\partial w2}$$

$$\mathrm{b} = \mathrm{b} - learning\ rate * {}^{\partial}/_{\partial b}$$

$$Log\ loss\ or\ binary\ cross\ entropy = -\frac{1}{n}\sum_{i=0}^{n} y_i \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

Loss

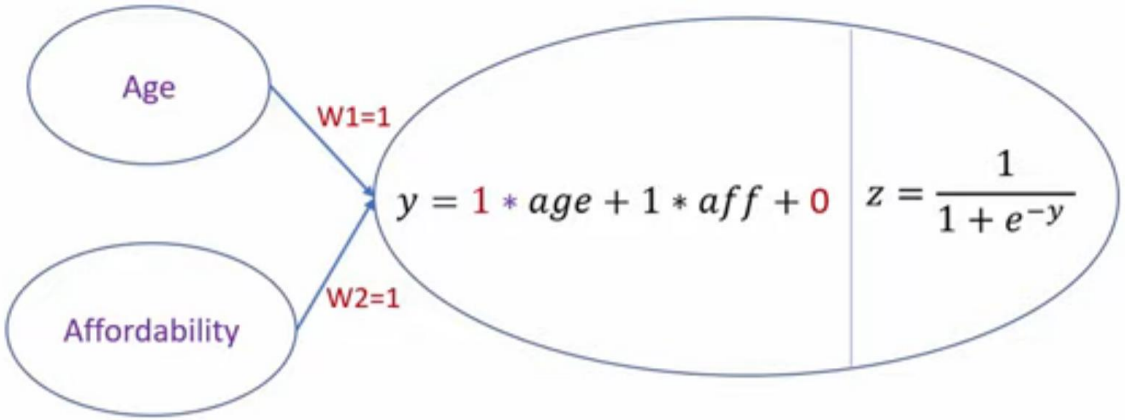
$$w1 = w1 - learning\ rate * {}^{\partial}/_{\partial w1} \qquad {}^{\partial}/_{\partial w1} = \frac{1}{n}\sum_{i=1}^{n} x_i\,(\hat{y}_i - y_i)$$

$$w1 = w1 - learning\ rate * \partial/\partial w1$$

$$\partial/\partial w1 = \frac{1}{n}\sum_{i=1}^{n} x_i\,(\hat{y}_i - y_i)$$

$$b = b - learning\ rate * \partial/\partial b$$

$$\partial/\partial b = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)$$



$$w1 = w1 - learning\ rate * \partial/\partial w1$$

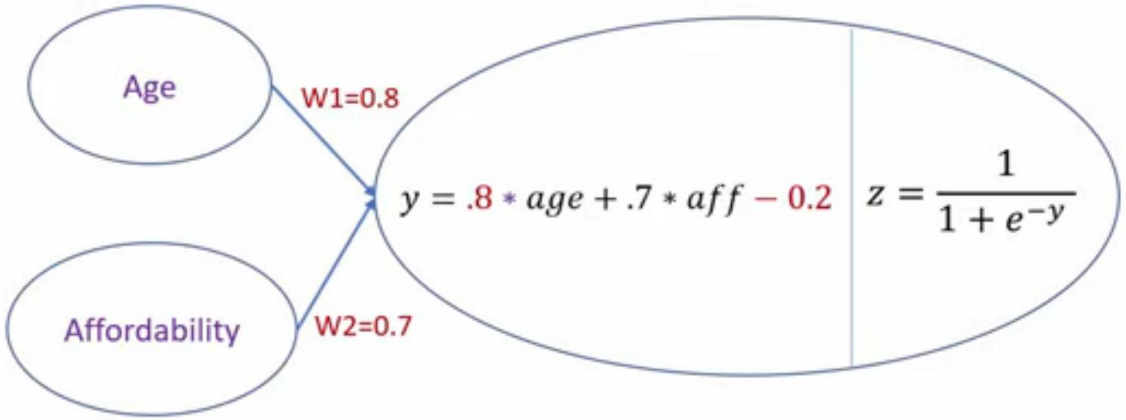
W1 = 1- 0.2 = 0.8

$$w2 = w2 - learning\ rate * \partial/\partial w2$$

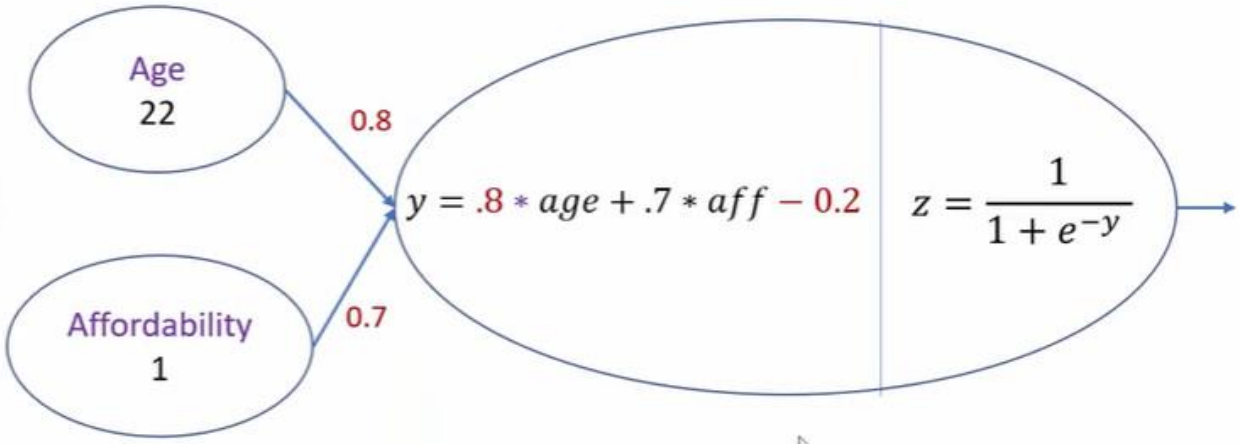
W2 = 1- 0.3 = 0.7

$$b = b - learning\ rate * \partial/\partial b$$

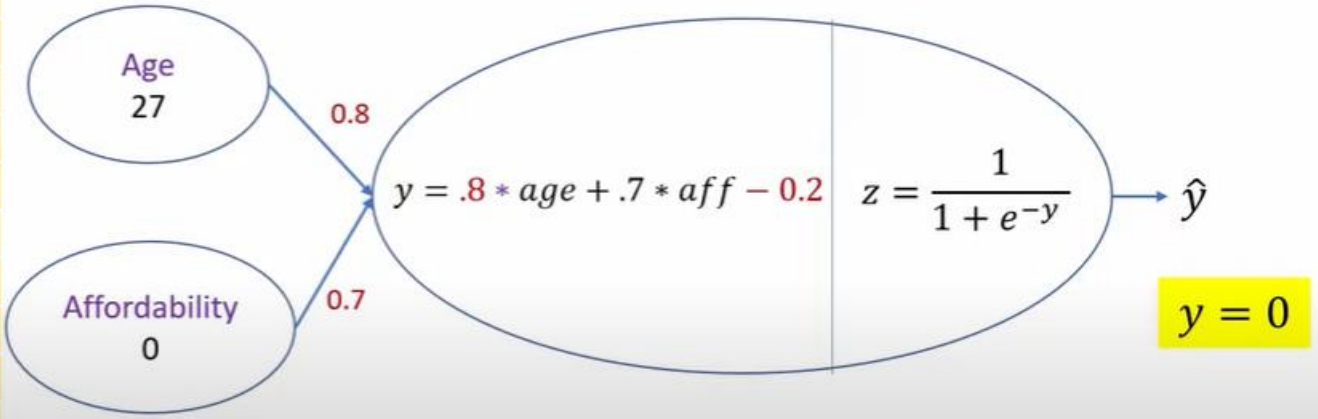
b = 0 – 0.2 = -0.2



age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1



age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	1



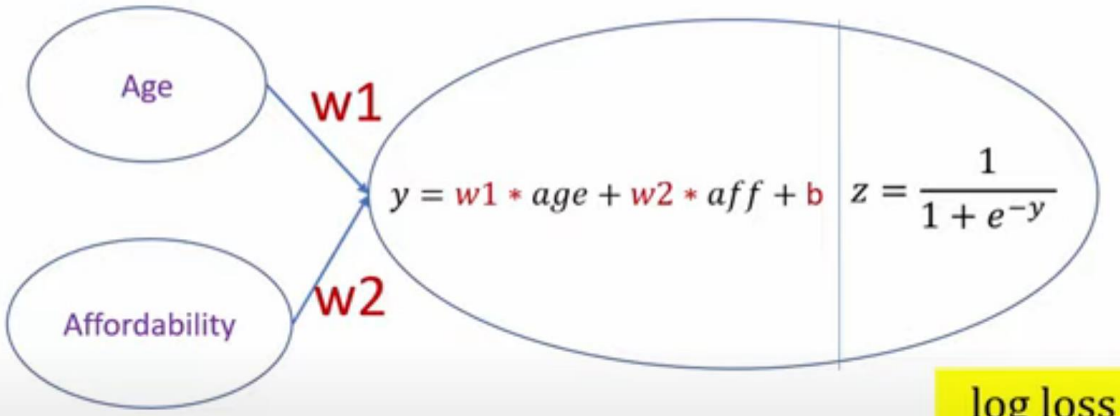
$y = 0$

$error13 = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$

Total error = $error_1 + error_2 + \ldots + error_{13}$

After **first epoch**

age	affordability	have_insurance
22	1	0
25	0	0
47	1	1
52	0	0
46	1	1
56	1	1
55	0	0
60	0	1
62	1	1
61	1	1
18	1	0
28	1	0
27	0	0



$\log loss = 4.31$

Backpropagation

1. **Error Calculation:** After the feedforward pass, the difference between the predicted output and the actual target output is calculated. This difference is the error, often referred to as the loss or cost.
2. **Backward Pass:** Backpropagation is the process of propagating the error backward through the network to compute the gradients of the loss with respect to the network's parameters (weights and biases).
3. **Gradient Calculation:** For each weight and bias in the network, the gradient of the loss with respect to that parameter is calculated. These gradients indicate how much the parameter needs to be adjusted to minimize the loss.
4. **Weight and Bias Updates:** The gradients computed during backpropagation guide the updates of the network's parameters using an optimization algorithm (e.g., gradient descent). This step aims to minimize the error between predicted and actual outputs.
5. **Iterative Process:** The feedforward and backpropagation steps are repeated iteratively over multiple epochs until the network's performance converges to a satisfactory level.


Any
Questions

Northeastern University

Lecture Outline

1. Introduction to Machine Learning
2. Motivating Example
3. Fundamentals of Neural Networks
   - Definition and Structure: Building Blocks
   - Reviewing Linear Regression
   - Activation Functions: Sparking Neurons
   - Feedforward Propagation: Information Flow
   - Keras Recipe(Python)
4. Learning and Training
   - Loss Functions: Measuring Errors
   - Backpropagation: Fine-Tuning Weights
5. Hands-on Exercises
6. In Class work
7. Summery
8. References

Keras Recipe

- Step 1. Describe model architecture
  - Number of hidden units, output units, activations.
- Step 2. Select an optimizer
- Step 3. Select a loss function and compile the model
- Step 4. Fit the model
- Step 5. Test / use the model

Implantation using Python


```
[ ] import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    import pandas as pd
    from matplotlib import pyplot as plt
    %matplotlib inline
```

```
df = pd.read_csv("insurance_data.csv")
df.head()
```

	age	affordibility	bought_insurance
0	22	1	0
1	25	0	0
2	47	1	1
3	52	0	0
4	46	1	1

Split train and test set

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(df[['age','affordibility']],df.bought_insurance,test_size=0.2, random_state=25)
```


Any
Questions

Take Home Points

- Artificial Neural Networks, or **ANNs**, are a class of machine learning models inspired by the structure and function of the human brain. They consist of interconnected units called neurons that process and transmit information. ANNs have gained prominence due to their remarkable ability to solve complex problems across various domains.

Take Home Points

Key Components:

1. **Neurons:** Neurons are the fundamental building blocks of ANNs. They receive input, process it using weights and activation functions, and produce an output.
2. **Layers:** Neurons are organized into layers. The input layer receives data, hidden layers process it, and the output layer generates predictions or classifications.
3. **Weights and Bias:** Weights determine the strength of connections between neurons. Bias adds an offset to the output, enhancing the model's flexibility.
4. **Activation Functions:** Activation functions introduce non-linearity to the model. Common choices include sigmoid, ReLU (Rectified Linear Unit), and tanh.

Take Home Points

Training Process:

1. **Forward Propagation:** Input data passes through the network's layers, and computations using weights and activations generate predictions.
2. **Loss Function:** A loss function measures the difference between predictions and actual values.
3. **Backpropagation:** The network adjusts weights to minimize the loss. Gradients are calculated using the chain rule and propagated backwards through the layers.
4. **Optimization:** Algorithms like gradient descent update weights iteratively to minimize the loss function.

Take Home Points

Applications:

1. **Image Recognition:** ANNs excel at image classification tasks, such as identifying objects or people in images.
2. **Natural Language Processing:** ANNs are used for language translation, sentiment analysis, and text generation.
3. **Healthcare:** They aid in medical diagnosis, drug discovery, and personalized treatment recommendations.
4. **Autonomous Vehicles:** ANNs contribute to self-driving cars by processing sensor data and making real-time decisions.

Take Home Points

Challenges:

1. **Overfitting:** Models can become too specific to training data, impacting generalization.
2. **Hyperparameter Tuning:** Finding optimal settings for parameters requires careful experimentation.
3. **Data Quality:** ANN performance relies on clean, diverse, and representative data.

- Artificial Neural Networks have revolutionized machine learning and continue to evolve with advancements like deep learning, where ANNs have multiple hidden layers. Understanding ANNs opens doors to addressing complex problems and driving innovation across industries.
- Remember, ANNs are a powerful tool in the machine learning toolbox, and continued learning and practice are essential to mastering their design, training, and deployment.

Northeastern University

References

1. http://brokerstir.com/logistic-regression-model-intuition/
2. https://www.geeksforgeeks.org/implement-sigmoid-function-using-numpy/
3. http://ieeexplore.ieee.org/document/6914146/
4. http://www.svms.org/disadvantages.html
5. https://www.mit.edu/~9.520/spring09/Classes/multiclass.pdf
6. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/
7. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/
8. http://www.statistics4u.com/fundstat_eng/cc_scaling.html
9. https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02
10. https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

THANK YOU FOR YOUR ATTENTION