

# DEEP LEARNING FOR COMPUTER VISION

## Week1



Dr. Tuchsanai. PloySuwan

สัปดาห์ที่	หัวข้อ	วิธีการเรียน	ผลการเรียนรู้	หมายเหตุ
1	The basic concept of deep learning and their applications in computer vision	บรรยาย	CLO-1	Chapter 1
2	Introduction to Images and Videos Basics with Python and OpenCV	บรรยาย	CLO-1	Chapter 1
3	Advance in OpenCV	บรรยาย	CLO-1	Chapter 1
4	Introduction to Pytorch	บรรยาย	CLO-2, CLO-3, CLO-4	Chapter 2
5	Fundamental Neural network	บรรยาย	CLO-2	Chapter 2
6	Stochastic optimization methods	บรรยาย	CLO-4	Chapter 2
7	Convolutional neural network for image classification	บรรยาย	CLO-2, CLO-3	Chapter 3
9	Modern architectures such as VGG, ResNet, and Efficientnet	บรรยาย	CLO-3,	Chapter 3
10	Transfer learning and fine tuning	บรรยาย	CLO-5	Chapter 3
11	Deep learning for Segmentation for images	บรรยาย	CLO-6	Chapter 4
12	Deep learning Object detection and Object Tracking	บรรยาย	CLO-7	Chapter 4
13	GAN architecture (Generative Adversarial Network) for photo-realistic images	บรรยาย	CLO-8	Chapter 5
14	Generative image from text CLIP model	บรรยาย	CLO-8	Chapter 5
15	Discussions	บรรยาย		

# APPROACHING (ALMOST) ANY MACHINE LEARNING PROBLEM



ABHISHEK THAKUR

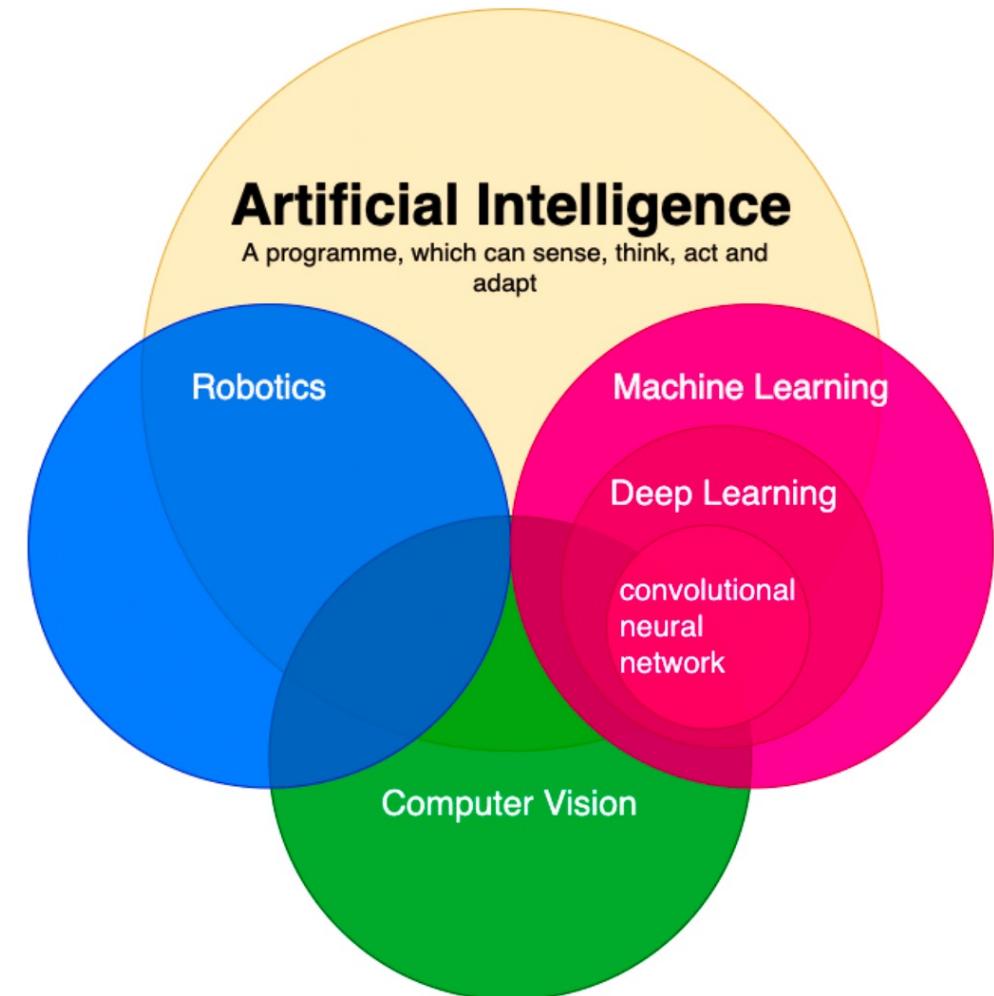
# BOOKS

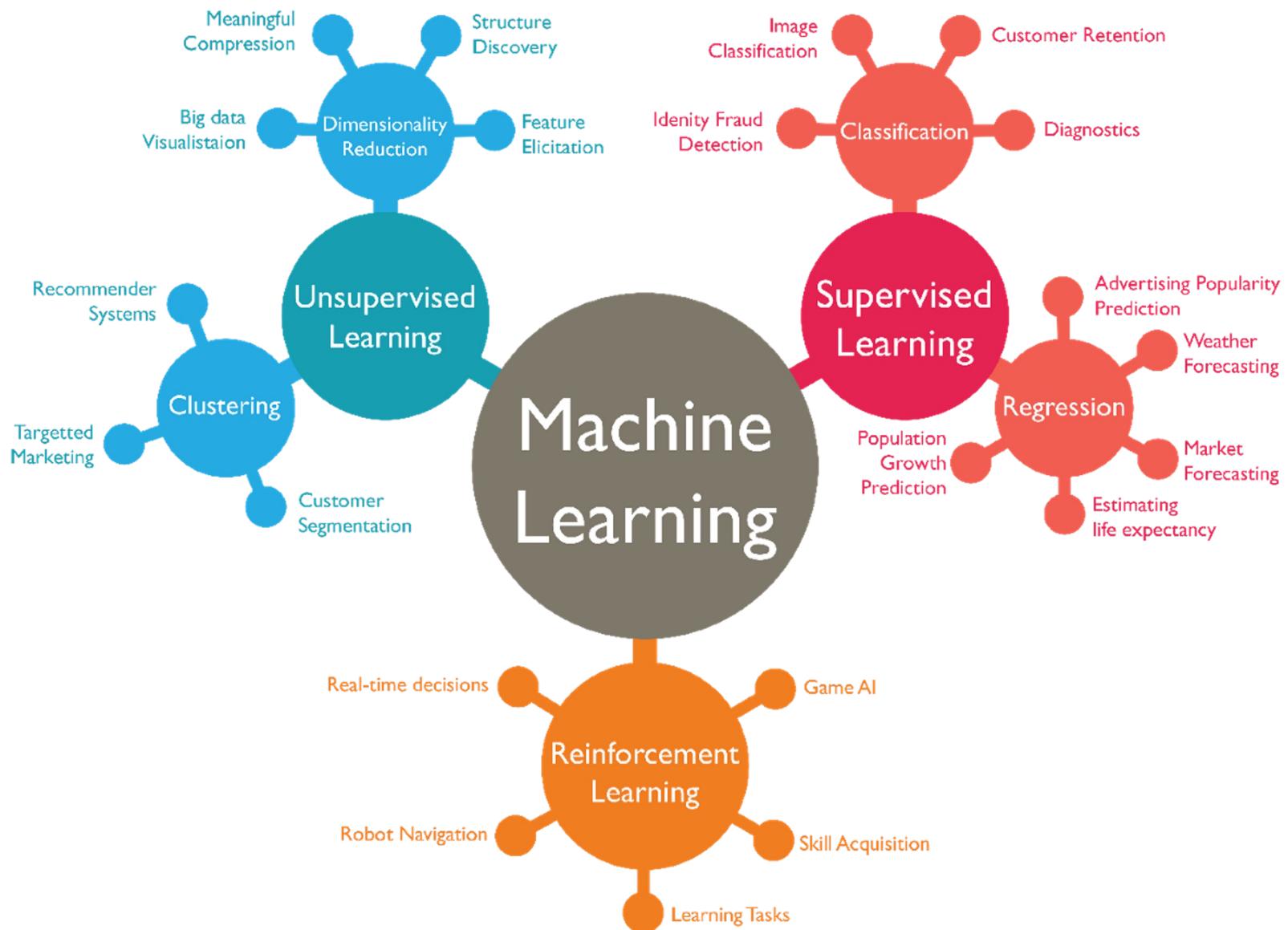
<https://github.com/abhishekrthakur/approachingalmost>

# Is Computer Vision Artificial Intelligence?

- Relation between Artificial Intelligence, Machine Learning and Deep Learning, Computer Vision.

[https://www.researchgate.net/figure/Relation-between-Artificial-Intelligence-Machine-Learning-and-Deep-Learning-Computer\\_fig1\\_342978934](https://www.researchgate.net/figure/Relation-between-Artificial-Intelligence-Machine-Learning-and-Deep-Learning-Computer_fig1_342978934)

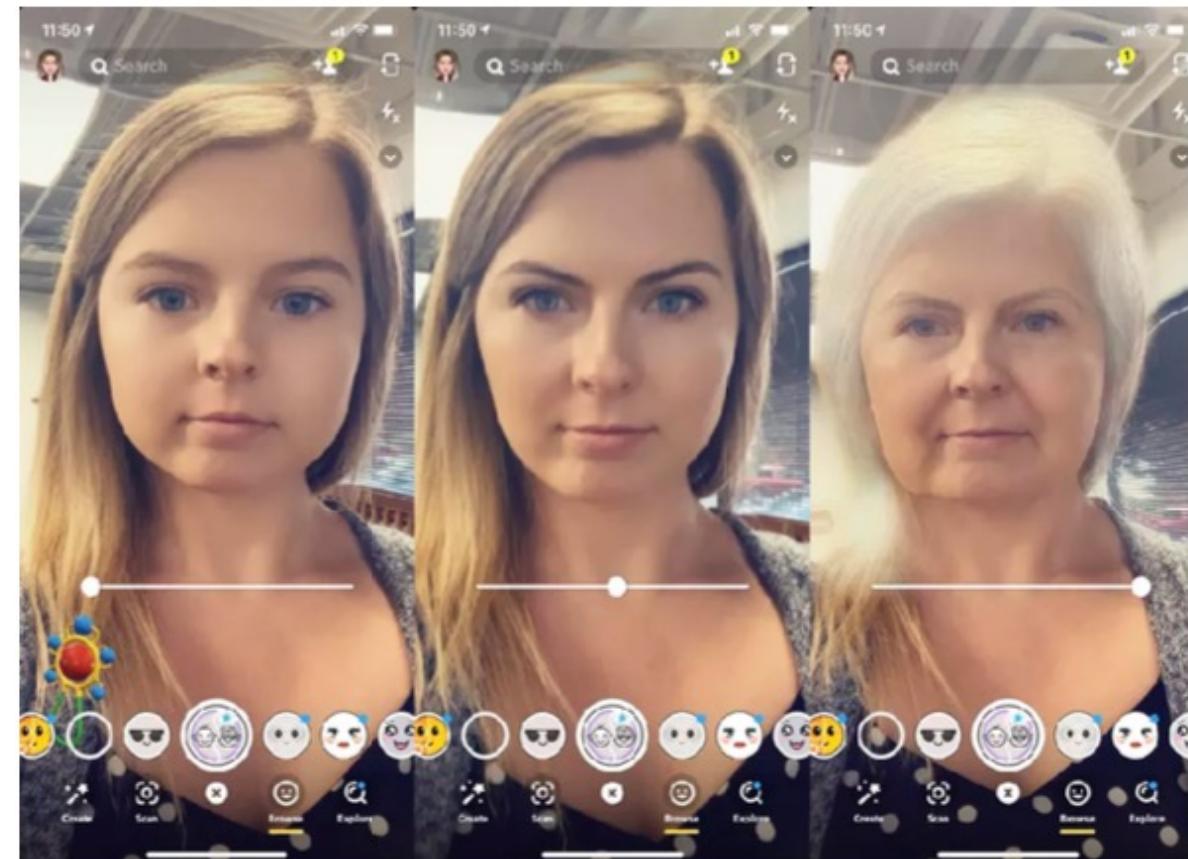




# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- Sporting Analysis
- Facial Recognition

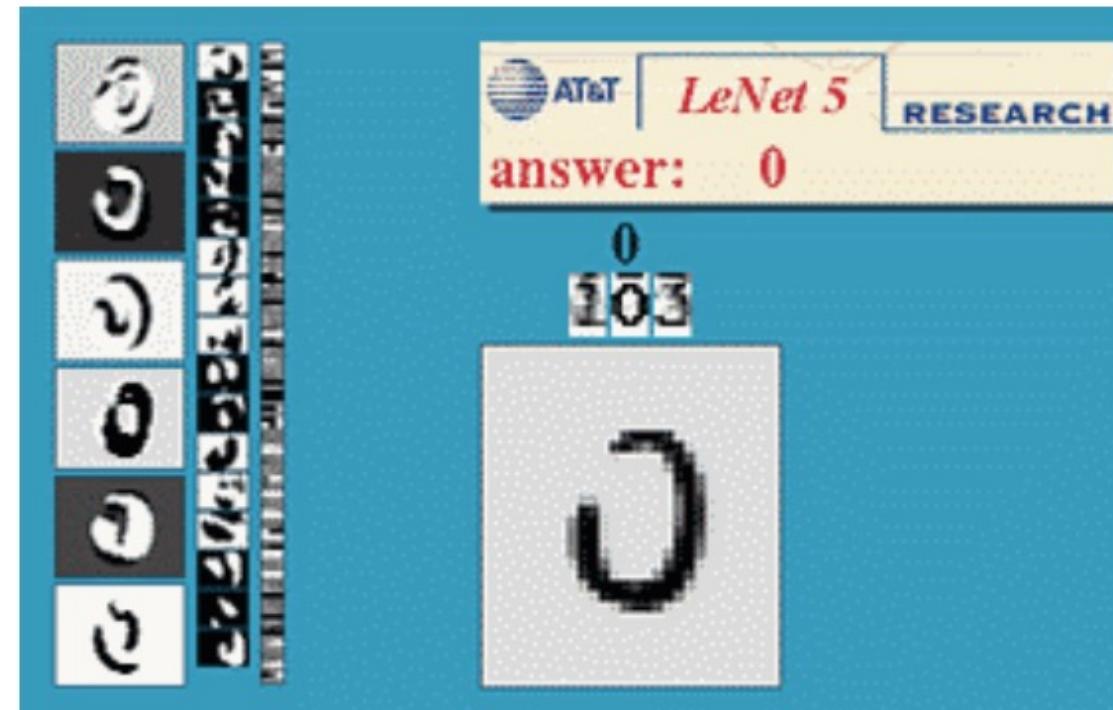


Source - Cnet - Snapchat's Time Machine AR lens creepily shows what you'll look like old

# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- Sporting Analysis
- Facial Recognition



Source -AT&T's LeNet OCR for Handwritten Digits

# SO WHAT CAN COMPUTER VISION DO?

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- **Licence Plate Reading**
- Self-driving cars
- Sporting Analysis
- Facial Recognition

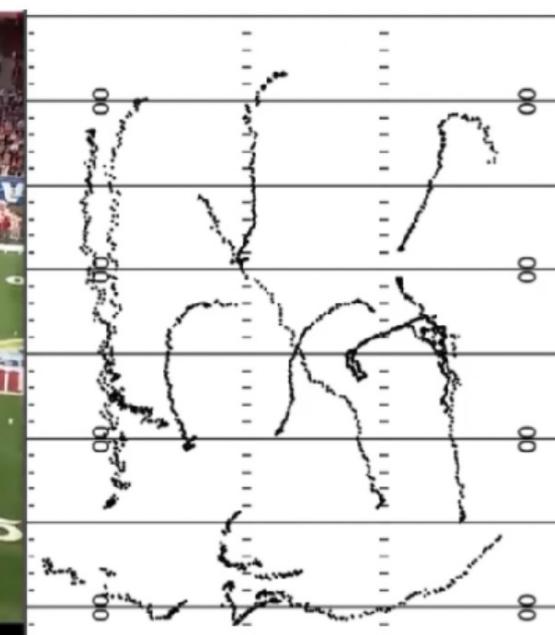
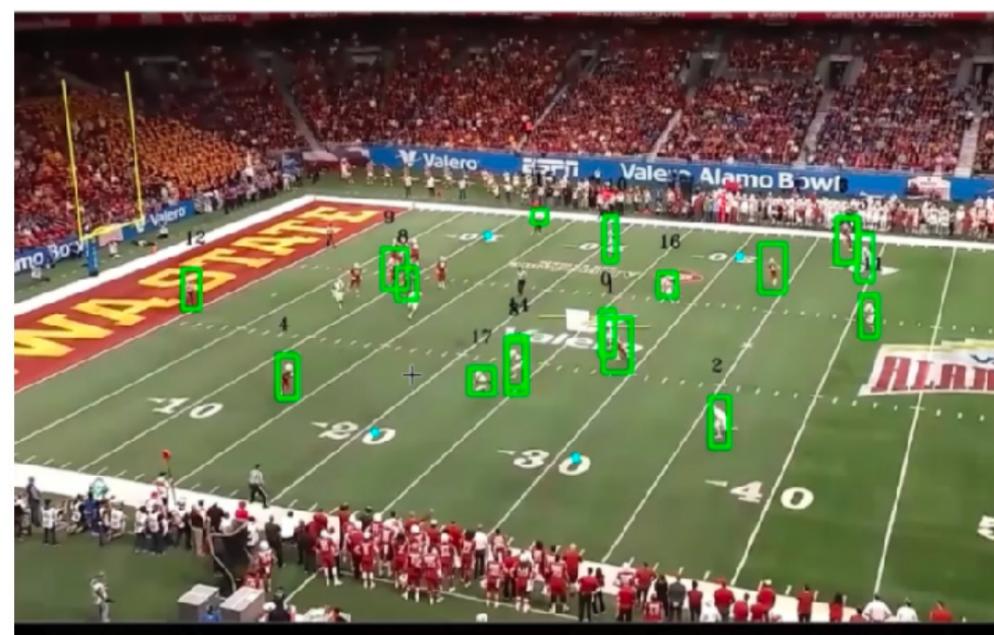
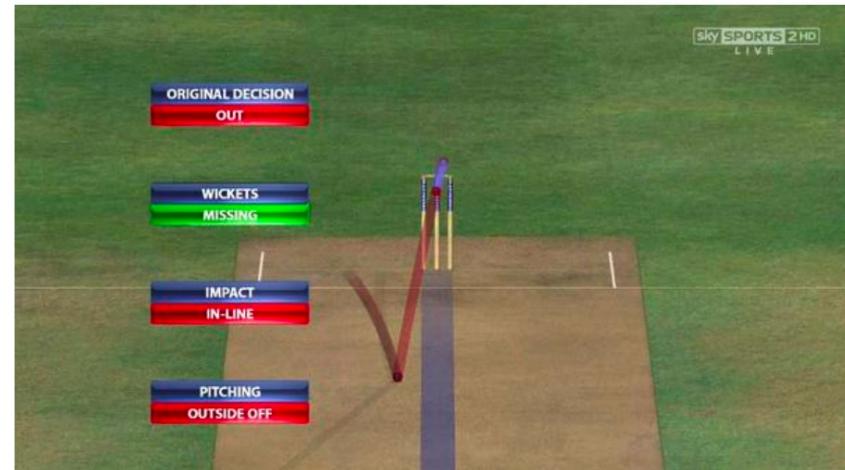


# So what can Computer Vision do?

You might be familiar with these...

- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- Licence Plate Reading
- Self-driving cars
- **Sporting Analysis**
- Facial Recognition

HawkEye in Cricket



Source -Roboflow - AI Coach

# Classical Computer Vision?

- What is meant by **Classical Computer Vision?**
- It encompasses Computer Vision algorithms that **do not** involve Machine Learning
- Before the advent of Machine Learning and Deep Learning, Computer Vision was a deeply explored field and many useful algorithms were developed for things like **feature extraction, OCR, Segmentation and simple transformations.**
- **OpenCV** is the Classical Computer Vision library of choice!



# Deep Learning Computer Vision

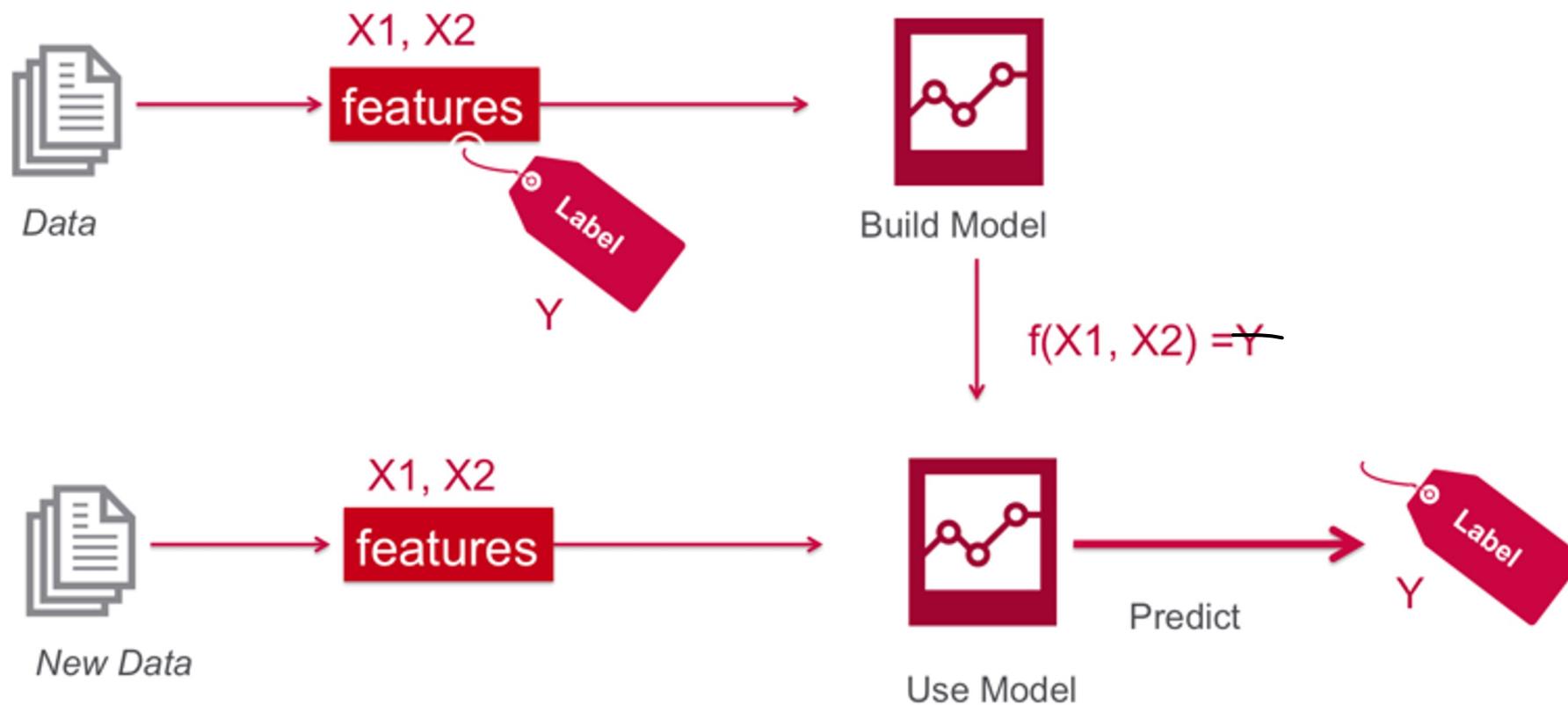
- Deep Learning was used in Computer Vision since the 1990s, however due to the computational requirements and intricate design, it remained on the sidelines for decades.
- Until the mid 2010s...which brought **two important building blocks** together.
  - **Accessible GPU processing** (NVIDIA's CUDA)



# Deep Learning CV vs Classical CV

<b>Deep Learning</b>	<b>Classical Computer Vision</b>
Adapts to new images well (assuming it's similar to the data it was trained on)	Small changes can have big negative impacts
Requires Models to be trained	Doesn't require training and can be used once coded
Model weights learn to adapt to varying image conditions	Relies on hardcode features and parameters
Requires GPU hardware (most times)	Can be run on CPU

# Supervised Learning



# Classification v.s. Regression

## Classification

- Predict which category an item belongs to based on labeled examples of known items
- Use in classify a new data

## Regression

- Predict a continuous value target.
- Linear regression predicts a numeric value
- Logistic regression predicts a probability

# Examples of classification

- Credit card fraud detection (fraud, not fraud)
- Credit card application (good credit, bad credit)
- Email spam detection (spam, not spam)
- Text sentiment analysis (happy, not happy)
- Predicting patient risk (high risk patient, low risk patient)
- Classifying a tumor as malignant or not

# Examples of regression

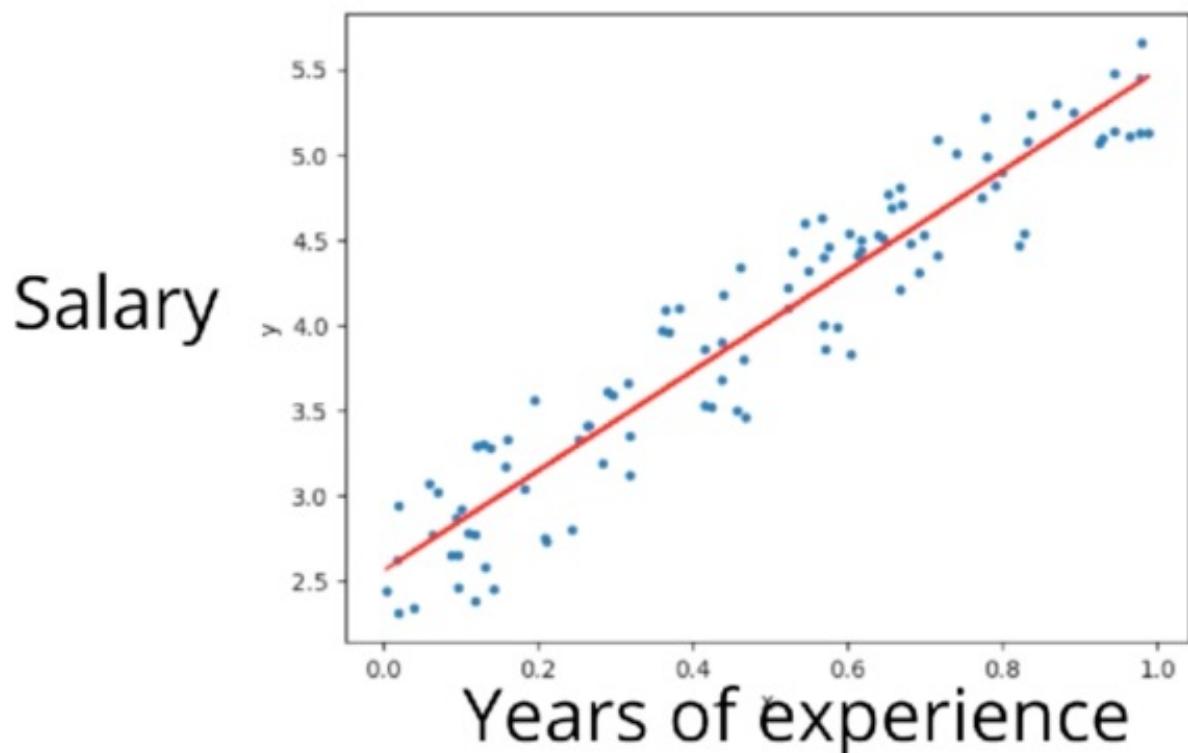
- Predicting the price of the house.
- Predicting age of a person
- Predicting the stock price for tomorrow.
- Predicting next year GDP.

# Example: Regression

- Fit a line or curve
- This is why grumpy old statisticians like to say: “machine learning is just glorified curve-fitting”

$$\hat{y} = mx + b$$

(sometimes we also use “a” for the slope)



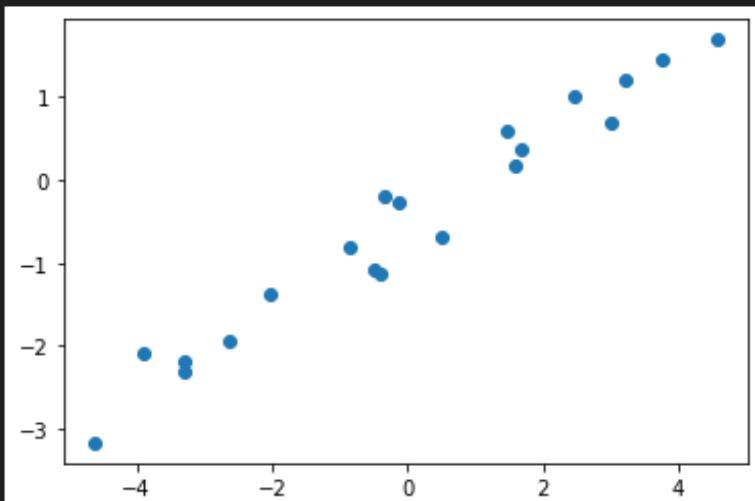
```
import matplotlib.pyplot as plt  
import numpy as np  
✓ 0.4s
```

```
# generate 20 data points  
N = 20  
  
#random data on x-axis  
x= np.random.rand(N)*10-5  
  
y = 0.5*x -1+ np.random.rand(N)  
✓ 0.2s
```

```
plt.scatter(x,y)
```

```
✓ 0.1s
```

```
<matplotlib.collections.PathCollection at 0x7fc1b147a410>
```



```
# In ML we want our data to be of shape:  
# (num_samples x num_dimensions)  
X = x.reshape(N, 1)  
Y = y.reshape(N, 1)
```

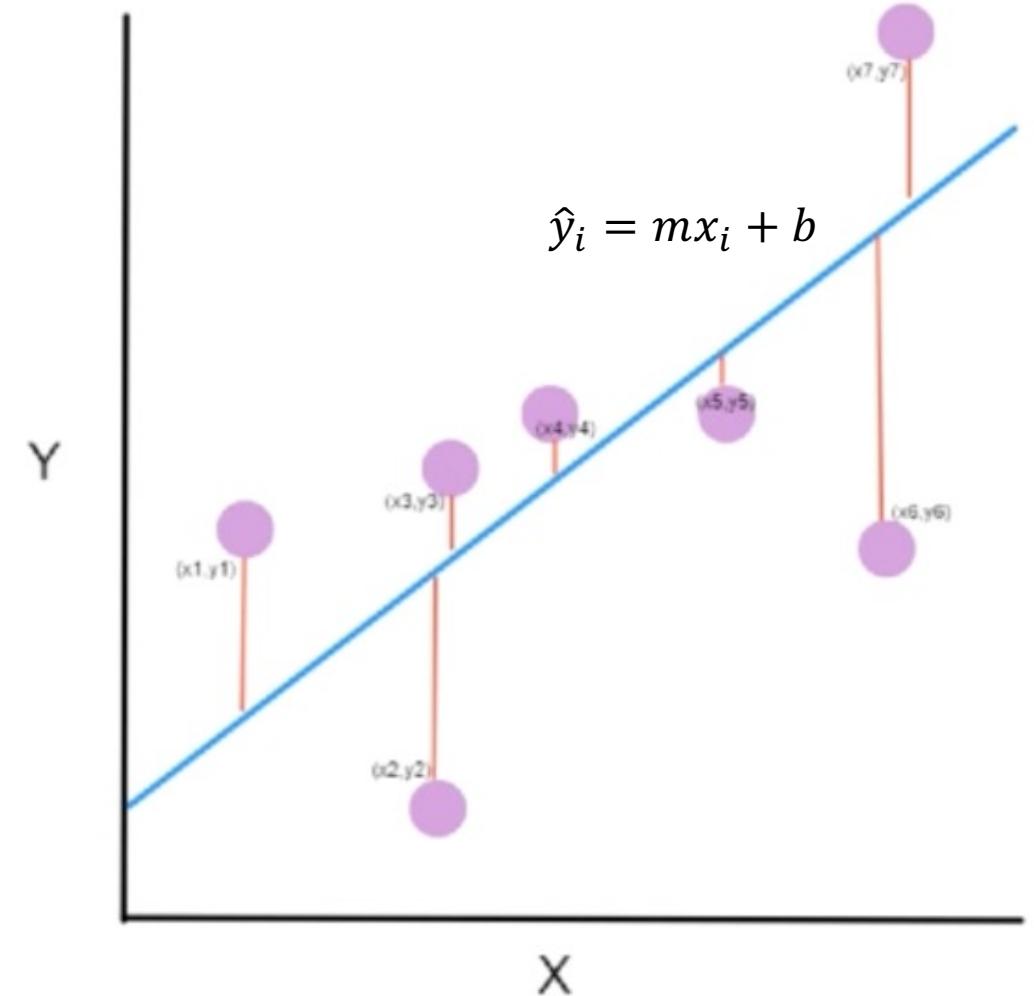
```
✓ 0.2s
```

# The Loss

- Data =  $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- N = number of samples in the dataset
- The line **cannot** perfectly pass through all the data points

MSE = Mean Squared Error

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$



## Applying the MSE to find slope / intercept

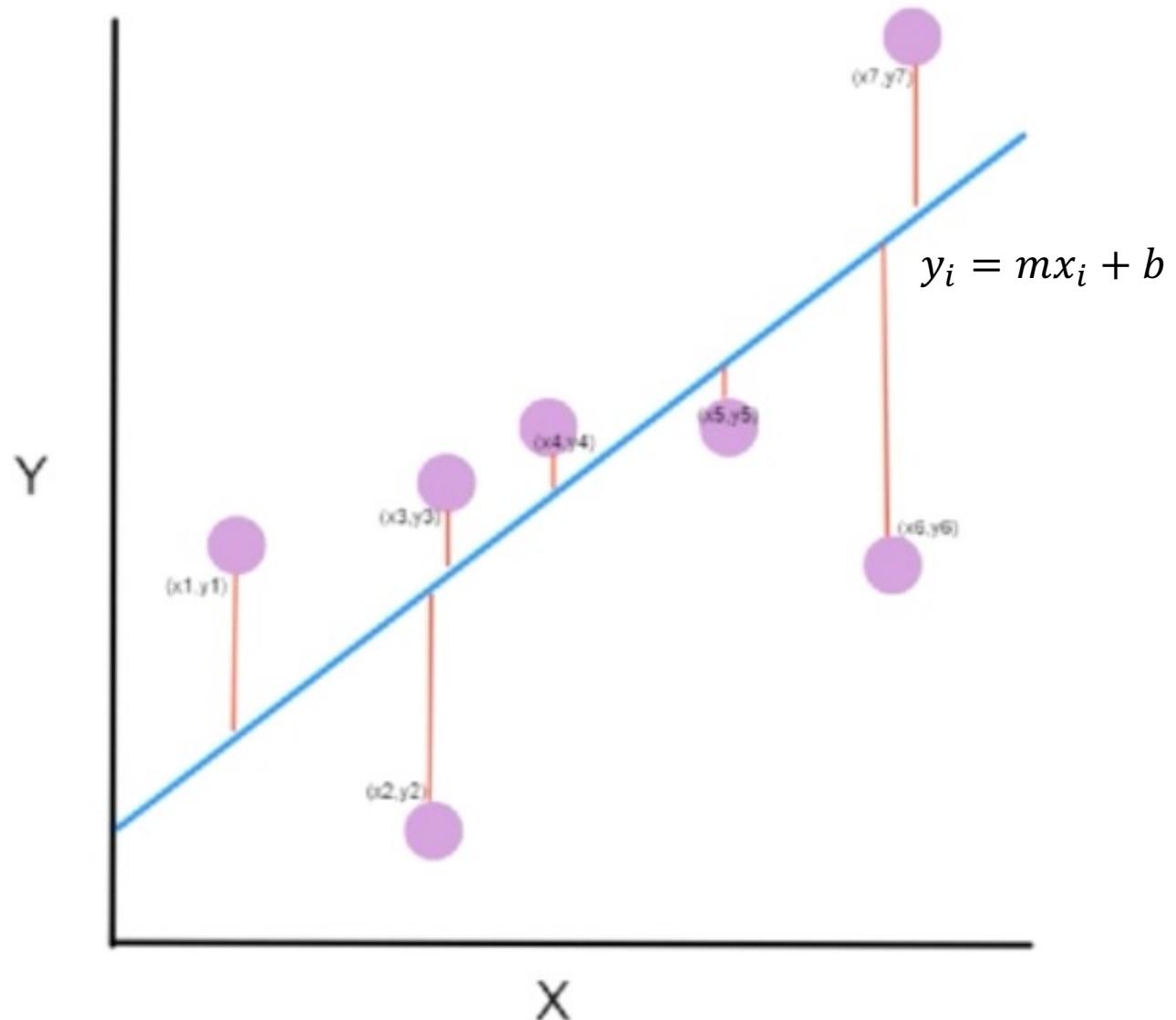
- Plug in the expression for the predictions ( $\hat{y}_i$ )
- Quiz: What are the **variables** in this expression?

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - (mx_i + b))^2$$

## Applying the MSE to find slope / intercept

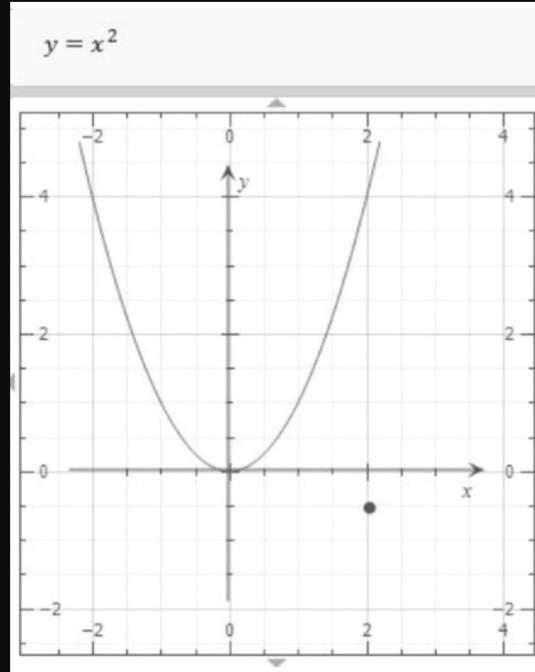
- Note:  $L(\text{loss}) = \text{MSE}$
- We want to *minimize the loss* with respect to the parameters  $(m, b)$

$$m^*, b^* = \arg \min_{m, b} L$$



# First Solution

Minimize  $f(x) = x^2$



- Use calculus!
- $df / dx = 2x = 0$
- Solve for  $x$ :  $x = 0$

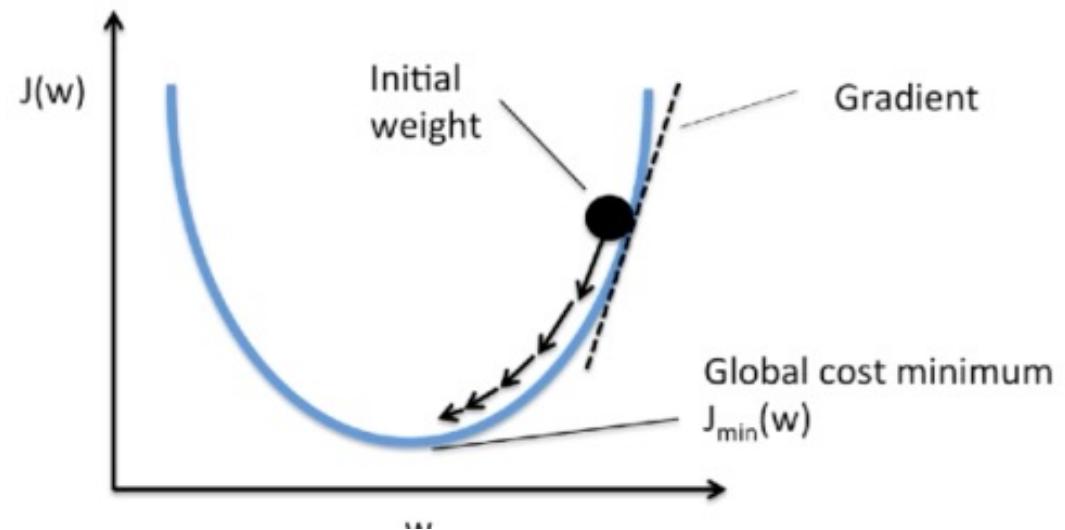
# Second Solution : Gradient Descent

- Find the derivatives and set them to 0, solve for the parameters
- Yields 2 equations and 2 unknowns: can solve for  $(m, b)$
- Try it as an exercise!: Should be in terms of  $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$

$$\partial L/\partial m = 0, \partial L/\partial b = 0$$

# Gradient Descent to find hidden parameters

```
# gradient descent  
# pseudocode  
  
 $\theta = (m, b) = \text{random}()$   
for i in range(n_epochs):  
     $\theta = \theta - \eta \nabla_{\theta} L$ 
```



$$L = MSE = \frac{1}{N} \sum_{i=1}^N (y_i - (mx_i + b))^2$$

$$\frac{\partial}{\partial m} = \frac{2}{n} \sum_{i=1}^n -x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial b} = \frac{2}{n} \sum_{i=1}^n -(y_i - (mx_i + b))$$

```
def Gradient(m,b) :  
    grad_m =0; grad_b = 0;  
  
    for i in range(N) :  
        grad_m += -X[i]*(Y[i]- (m*X[i]+b))  
        grad_b += - (Y[i]- (m*X[i]+b) )  
  
    return grad_m, grad_b  
  
Gradient(-1,-1)  
✓ 0.3s  
(array([-209.28382035]), array([-10.12894605]))
```

```
# gradient descent
# pseudocode

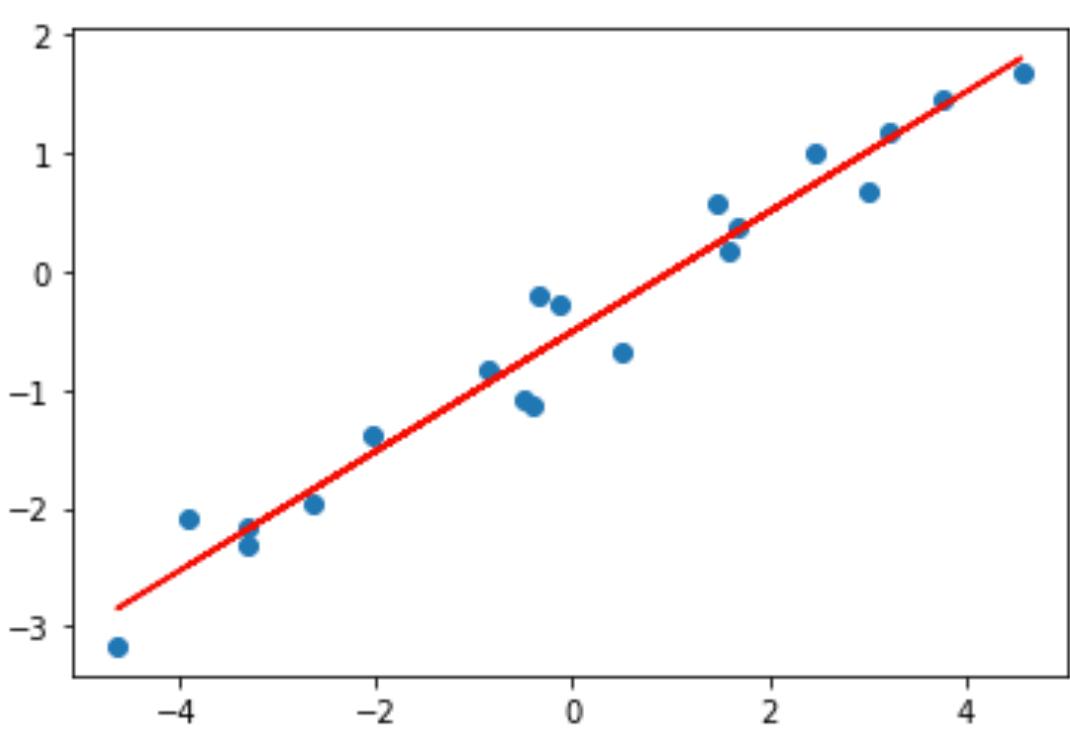
θ = (m, b) = random()
for i in range(n_epochs):
    θ = θ - η∇θL
```

```
n_epochs      = 1000
learning_rate = 0.01
theta         = np.random.rand(2,1)

for i in range(n_epochs):
    grad_m, grad_b = Gradient(theta[0], theta[1])
    theta[0] -= learning_rate*grad_m
    theta[1] -= learning_rate*grad_b

theta
✓ 0.1s
array([[ 0.50714187],
       [-0.51174366]])
```

```
predict = theta[0]*X + theta[1]
plt.plot(X, predict, 'r')
plt.scatter(x,y)
✓ 0.8s
```



SDG: Stochastic gradient descent

$$\theta = \theta - \eta \nabla_{\theta} \mathbb{L}$$

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

```
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
```

# Third Solution :Pytorch

```
# gradient descent
# pseudocode

θ = (m, b) = random()
for i in range(n_epochs):
    θ = θ - η∇_θ L
```

```
# Train the model
n_epochs = 30

for it in range(n_epochs):
    # zero the parameter gradients
    optimizer.zero_grad()

    # Forward pass  $\hat{y}_i = mx_i + b$ 
    outputs = model(inputs)
    loss = criterion(outputs, targets)

    # Backward and optimize
    loss.backward()
    optimizer.step()
```

```

1 # Train the model
2 n_epochs = 30
3 losses = []
4 for it in range(n_epochs):
5     # zero the parameter gradients
6     optimizer.zero_grad()
7
8     # Forward pass
9     outputs = model(inputs)
10    loss = criterion(outputs, targets)
11
12    # keep the loss so we can plot it later
13    losses.append(loss.item())
14
15    # Backward and optimize
16    loss.backward()
17    optimizer.step()
18
19    print(f'Epoch {it+1}/{n_epochs}, Loss: {loss.item():.4f}')

```

Epoch 1/30, Loss: 12.1352  
Epoch 2/30, Loss: 2.8066  
Epoch 3/30, Loss: 2.4554  
Epoch 4/30, Loss: 2.1753  
Epoch 5/30, Loss: 1.9483  
Epoch 6/30, Loss: 1.7641  
Epoch 7/30, Loss: 1.6148

```

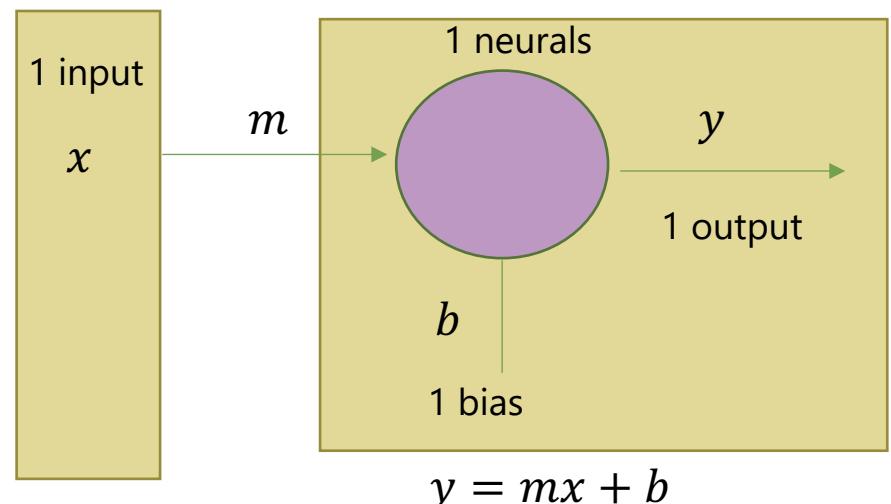
[ ] 1 # Create the linear regression model
2 model = nn.Linear(1, 1)

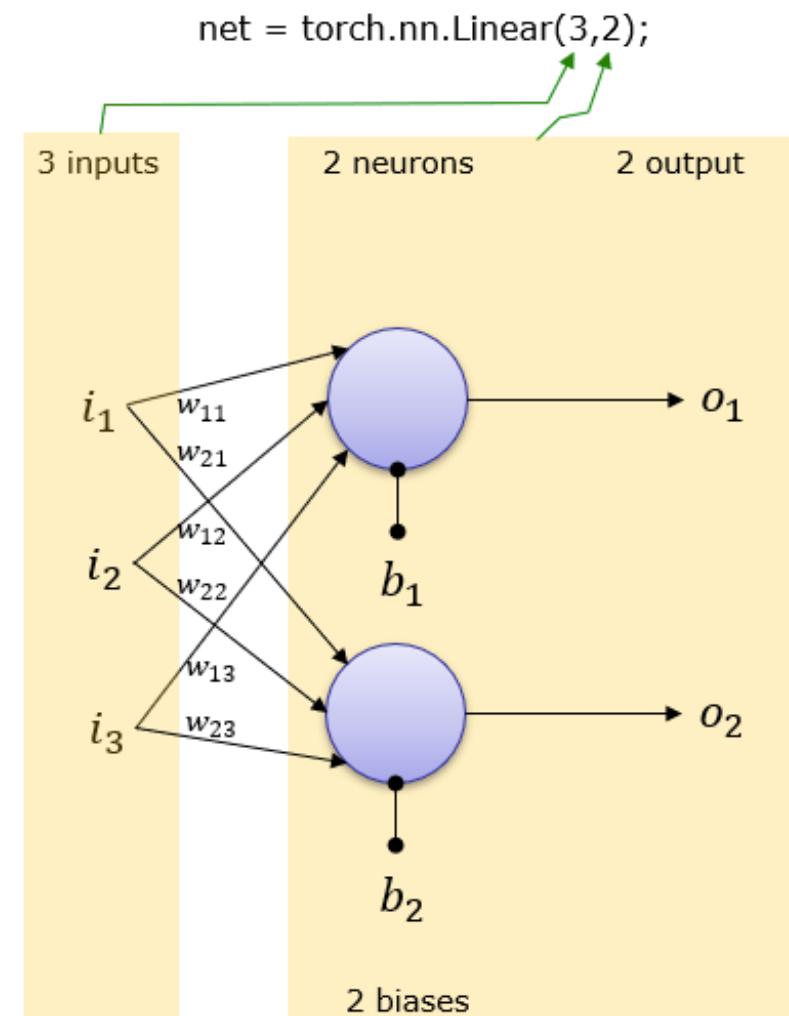
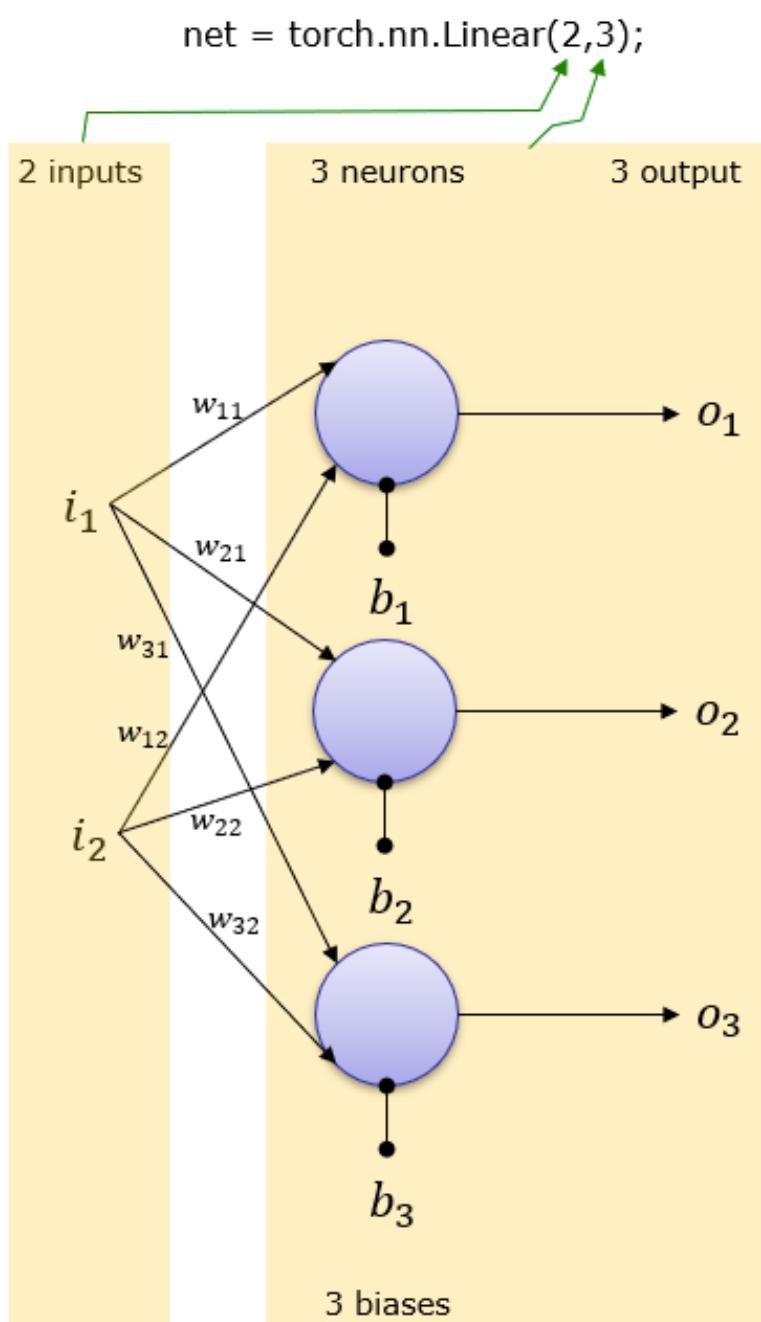
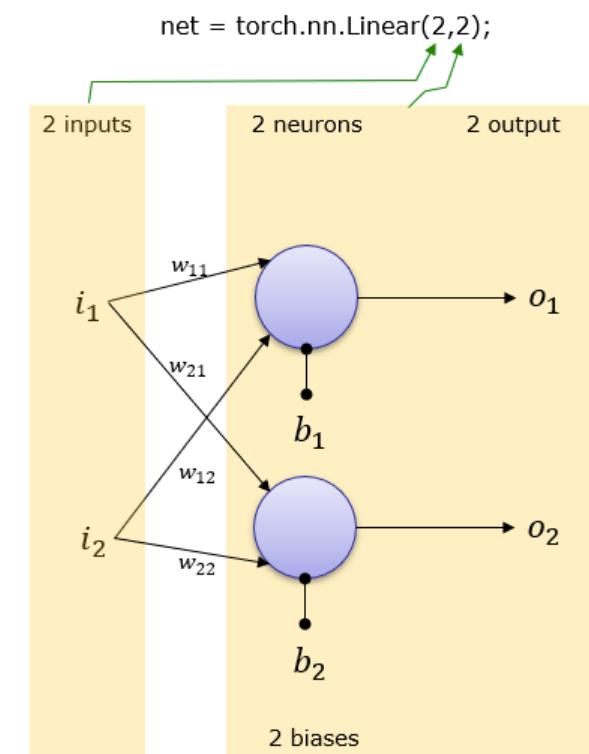
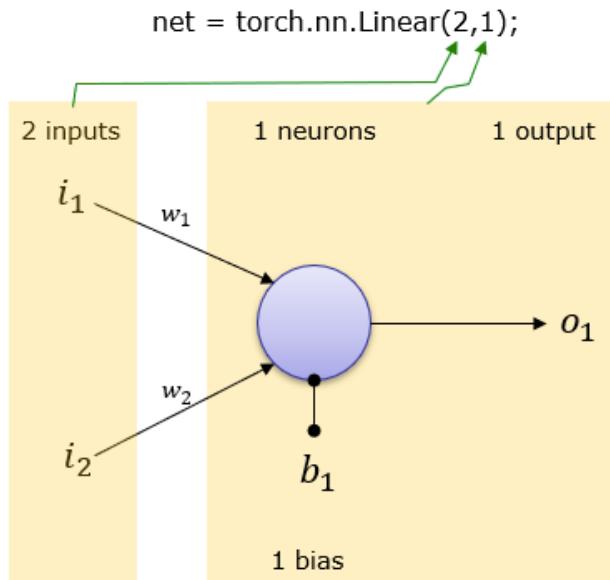
[ ] 1 # Loss and optimizer
2 criterion = nn.MSELoss()
3 optimizer = torch.optim.SGD(model.parameters(), lr=0.05)

[ ] 1 # In ML we want our data to be of shape:
2 # (num_samples x num_dimensions)
3 X = X.reshape(N, 1)
4 Y = Y.reshape(N, 1)
5
6 # PyTorch uses float32 by default
7 # Numpy creates float64 by default
8 inputs = torch.from_numpy(X.astype(np.float32))
9 targets = torch.from_numpy(Y.astype(np.float32))

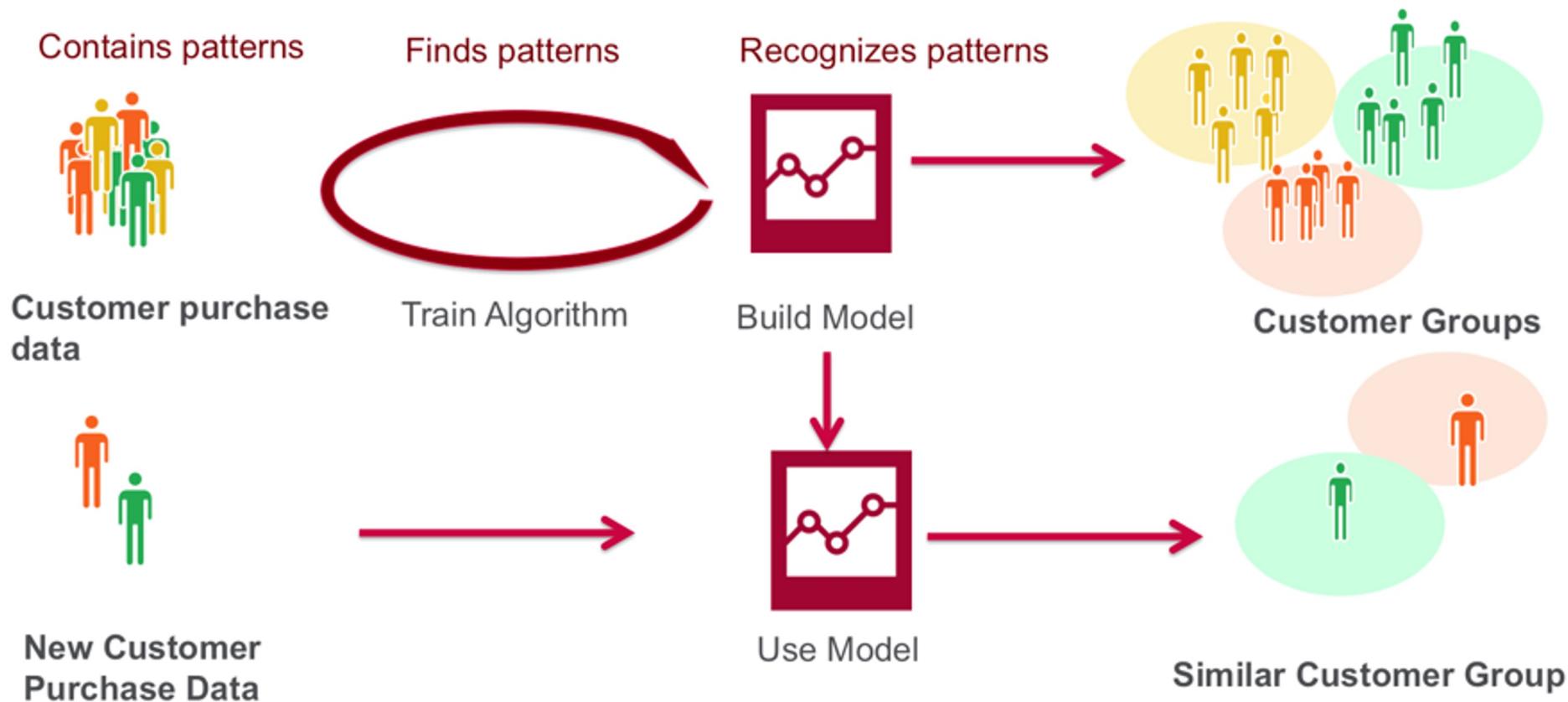
```

$$net = \text{torch}.Linear(1,1)$$





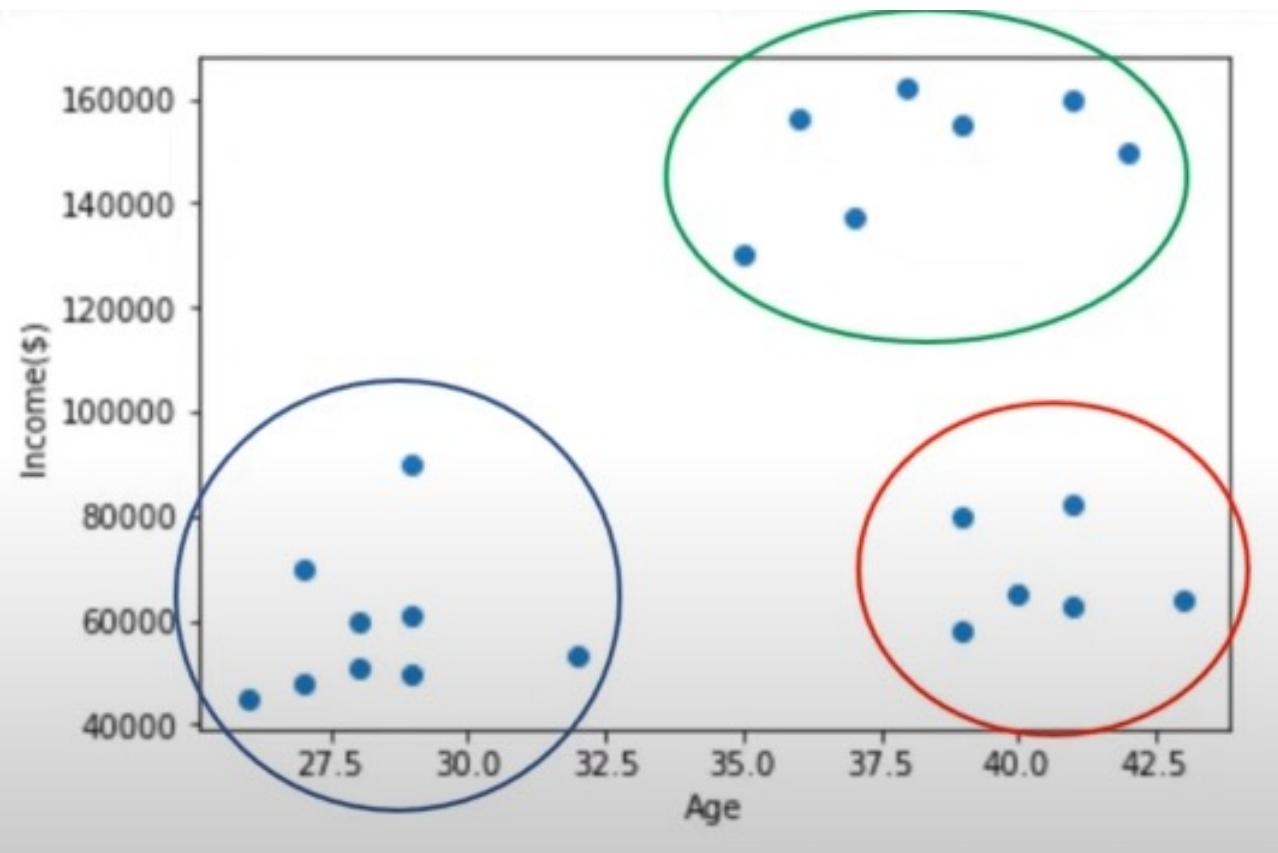
# Unsupervised Learning | Self Supervised



# Examples of clustering

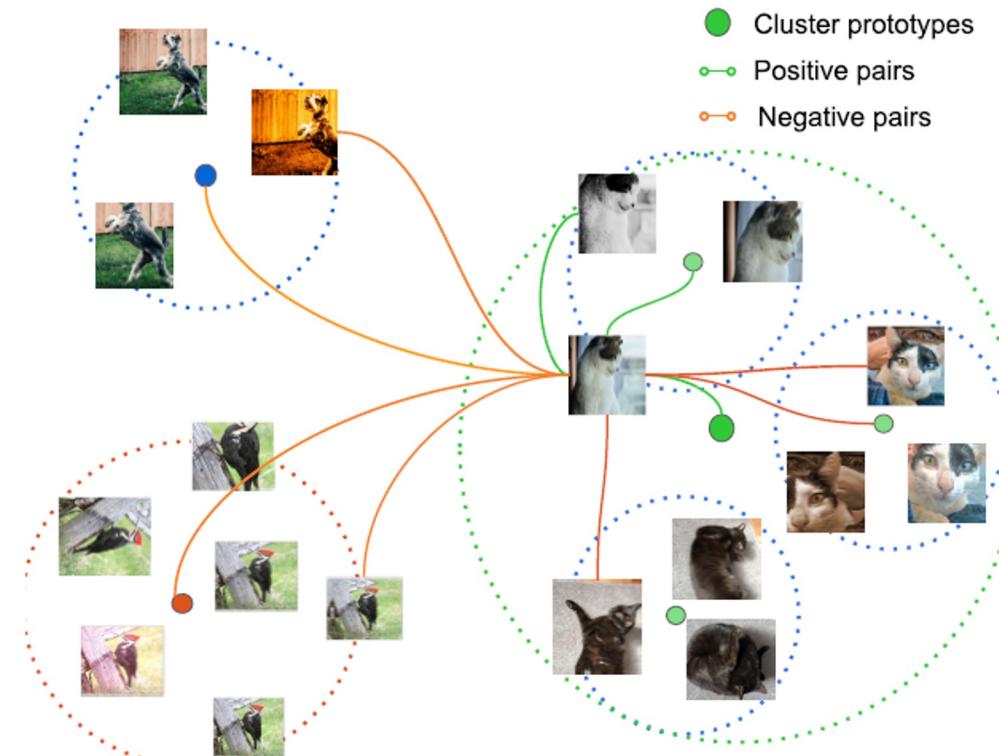
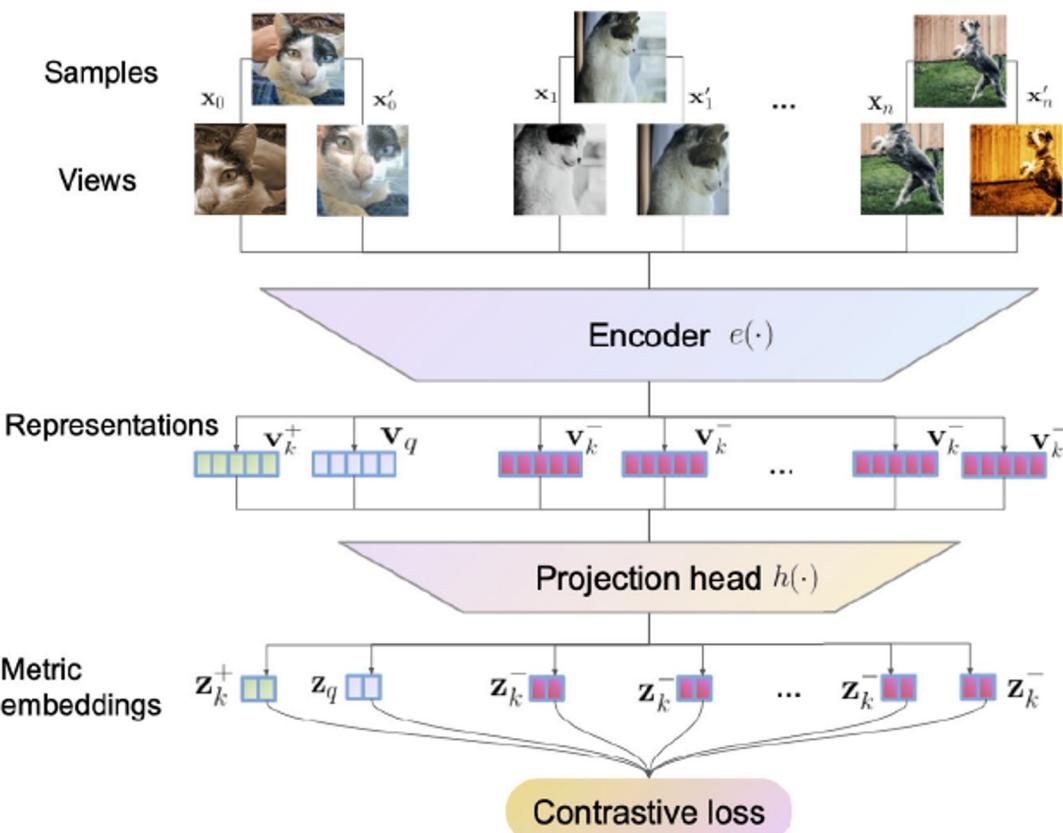
- Search results grouping
- Grouping similar customers
- Grouping similar patients
- Text categorization
- Network Security Anomaly detection

# Unsupervised learning

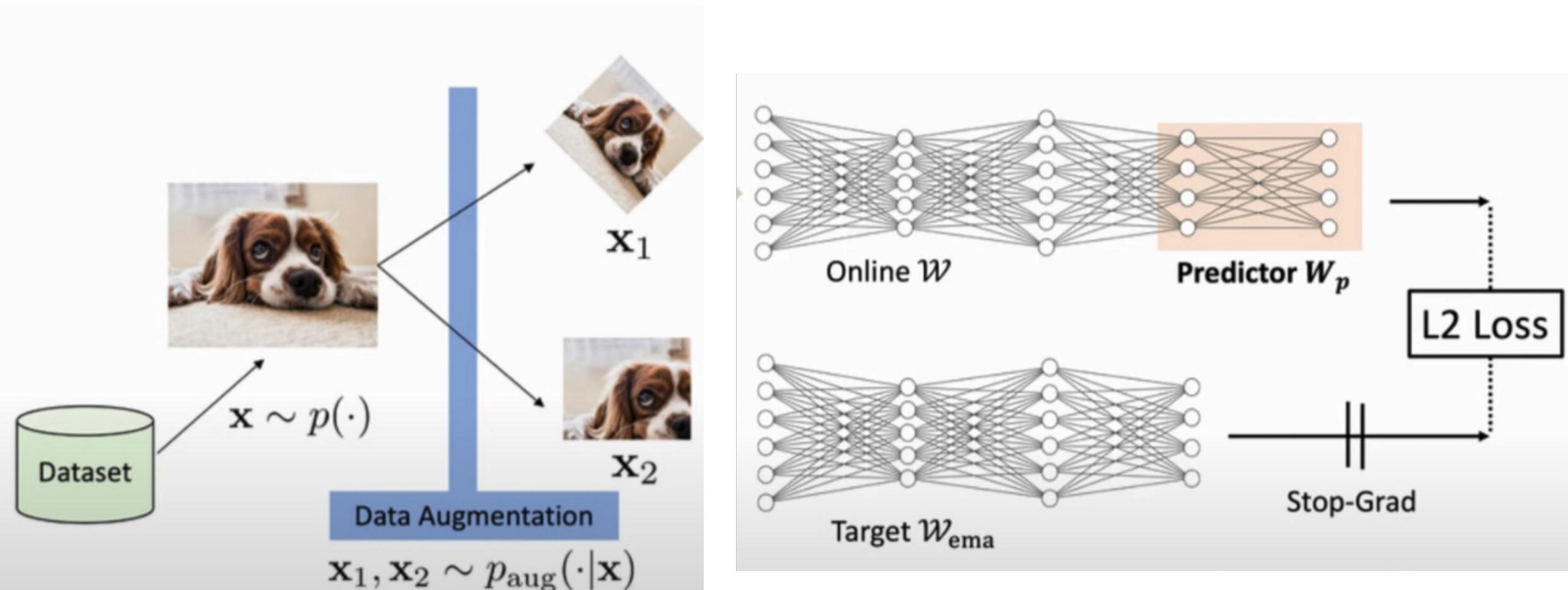


	A	B	C
1	Name	Age	Income(\$)
2	Rob	27	70000
3	Michael	29	90000
4	Mohan	29	61000
5	Ismail	28	60000
6	Kory	42	150000
7	Gautam	39	155000
8	David	41	160000
9	Andrea	38	162000
10	Brad	36	156000

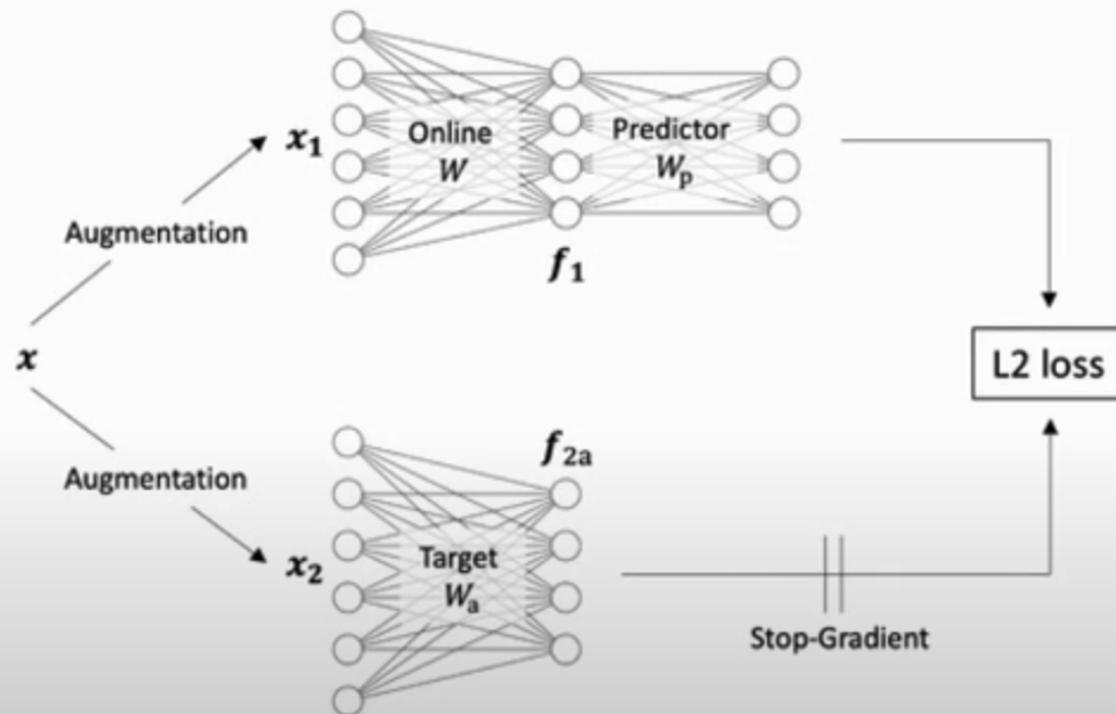
# Self-Supervised



# Easy review



# A simple model



Objective:

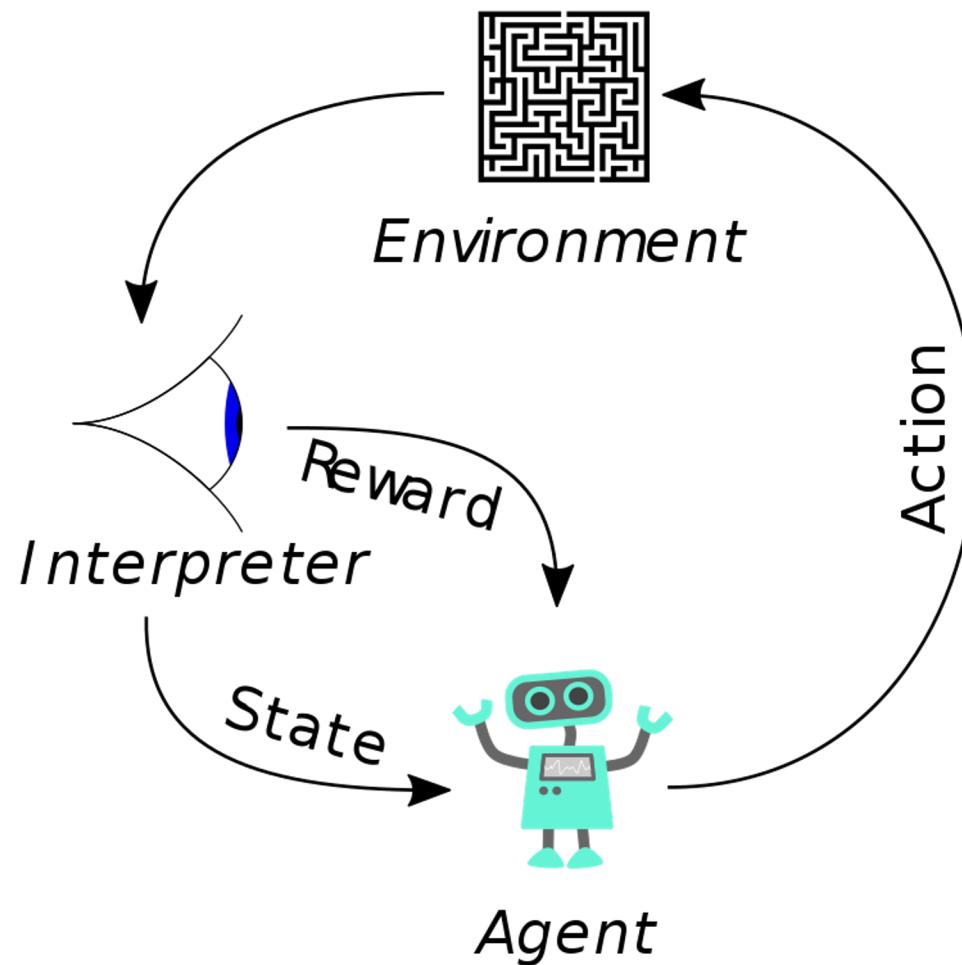
$$J(W, W_p) := \frac{1}{2} \mathbb{E}_{x_1, x_2} [\|W_p f_1 - \text{StopGrad}(f_{2a})\|_2^2]$$

Linear online network  $W$

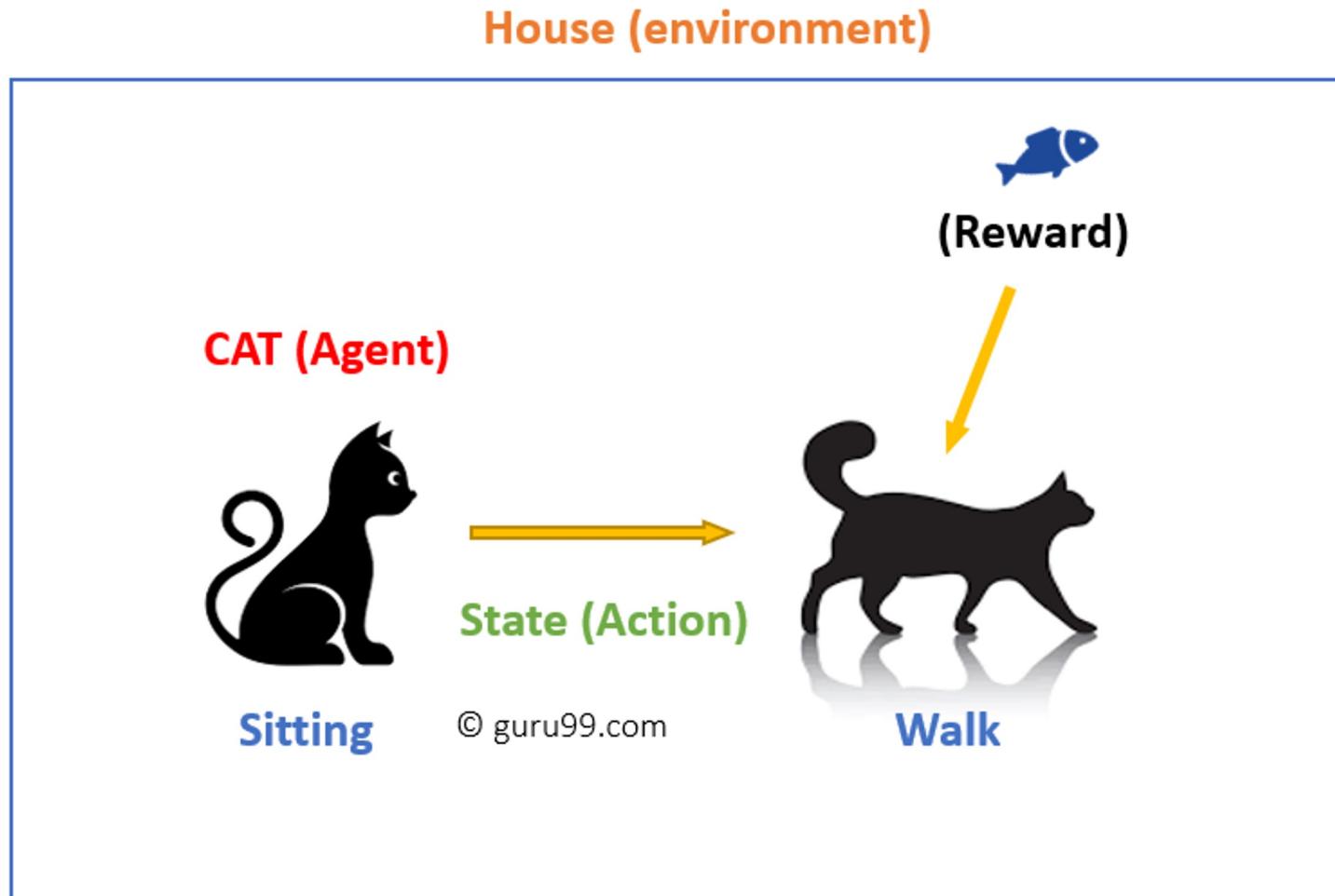
Linear target network  $W_a$

Linear predictor  $W_p$

# Reinforcement learning



# How Reinforcement Learning works?



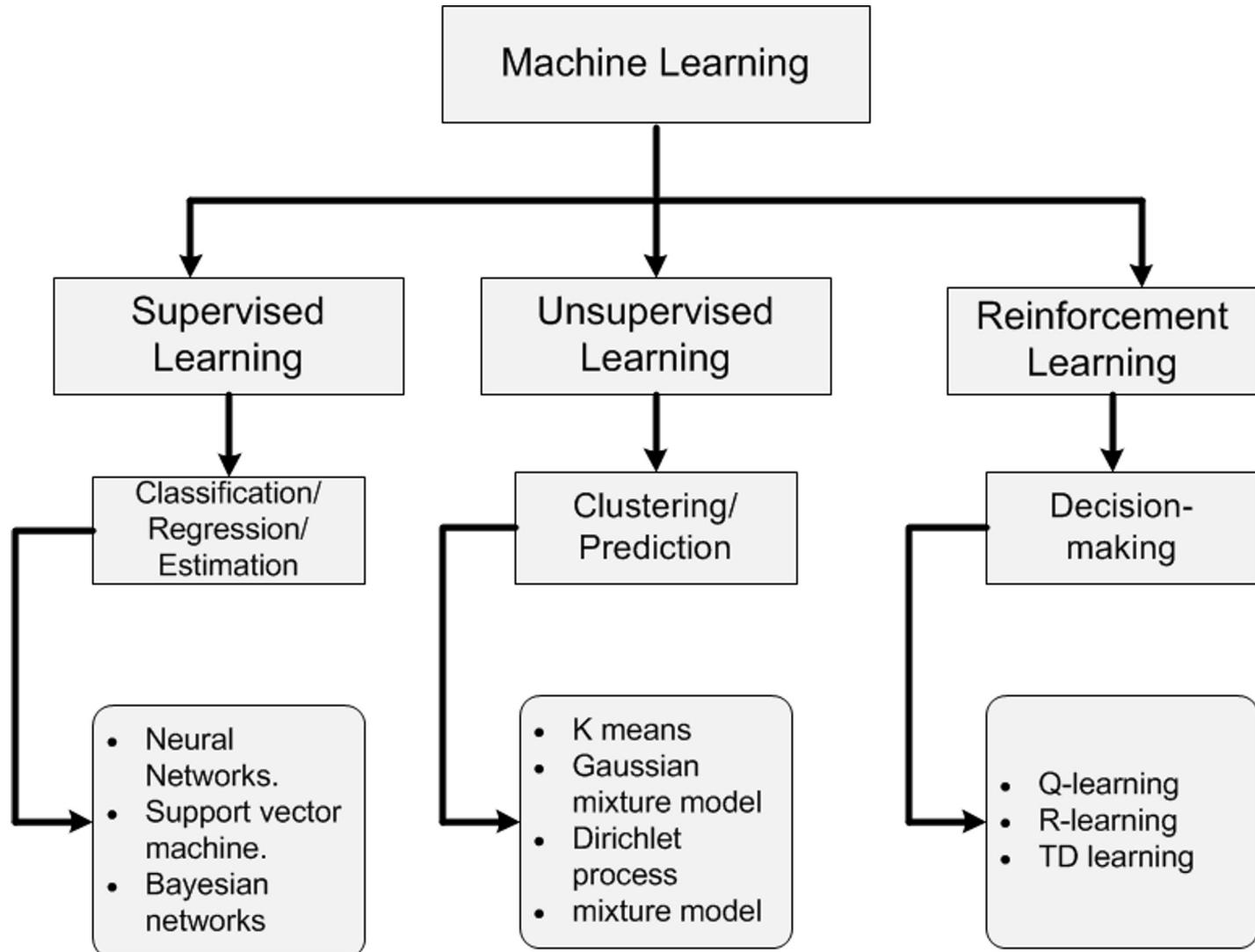
Source: Reinforcement Learning: What is, Algorithms, Applications,  
Episode 2, pg. 22

# How Reinforcement Learning works?

- Your cat is an agent that is exposed to the environment. In this case, it is your house. An example of a state could be your cat sitting, and you use a specific word in for cat to walk.
- Our agent reacts by performing an action transition from one "state" to another "state."
- For example, your cat goes from sitting to walking.
- The reaction of an agent is an action, and the policy is a method of selecting an action given a state in expectation of better outcomes.
- After the transition, they may get a reward or penalty in return.

# Examples of Reinforcement Learning

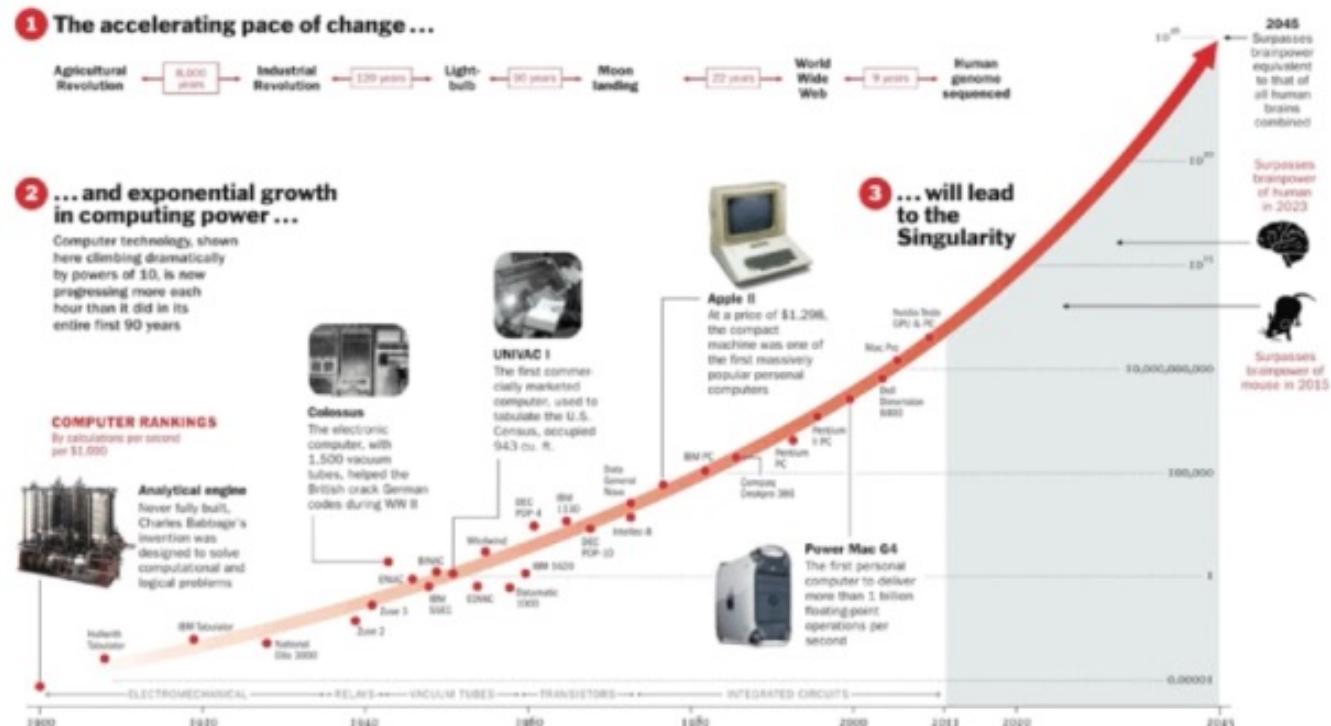
- Robotics for industrial automation.
- Business strategy planning
- Self-driving car
- It helps you to create training systems that provide custom instruction and materials according to the requirement of students.
- Aircraft control and robot motion control



Source: [https://www.researchgate.net/figure/Comparison-of-different-types-of-machine-learning\\_fig6\\_325928183](https://www.researchgate.net/figure/Comparison-of-different-types-of-machine-learning_fig6_325928183)

# Real-World Dataset: Moore's Law

- The number of transistors per square inch on integrated circuits doubles approximately every 2 years



# Exponential Growth of Computing

Twentieth through twenty first century



Logarithmic Plot

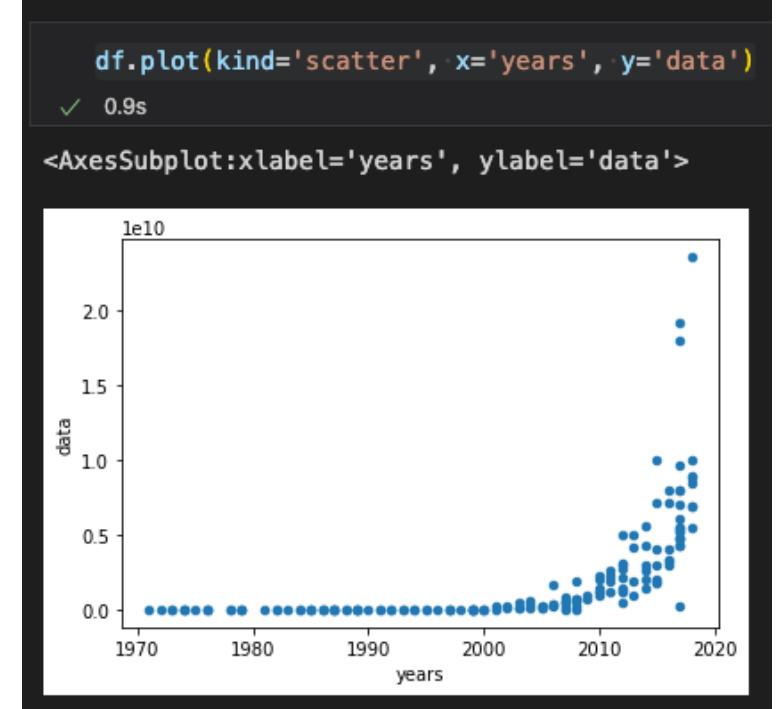


## Home Work

```
df = pd.read_csv('https://raw.githubusercontent.com/lazyprogrammer/machine_learning_examples/master/tf2.0/moore.csv', header=None)
df= df.rename(columns={0: "years", 1: "data"})
df
0.2s
```

	years	data
0	1971	2300
1	1972	3500
2	1973	2500
3	1973	2500
4	1974	4100
...	...	...
157	2017	18000000000
158	2017	19200000000
159	2018	8876000000
160	2018	23600000000
161	2018	9000000000

162 rows x 2 columns



[https://github.com/Tuchsanai/DL-FOR-COMPUTER-VISION-2565\\_1/blob/main/week1/code/Moore\\_rules.ipynb](https://github.com/Tuchsanai/DL-FOR-COMPUTER-VISION-2565_1/blob/main/week1/code/Moore_rules.ipynb)