Document

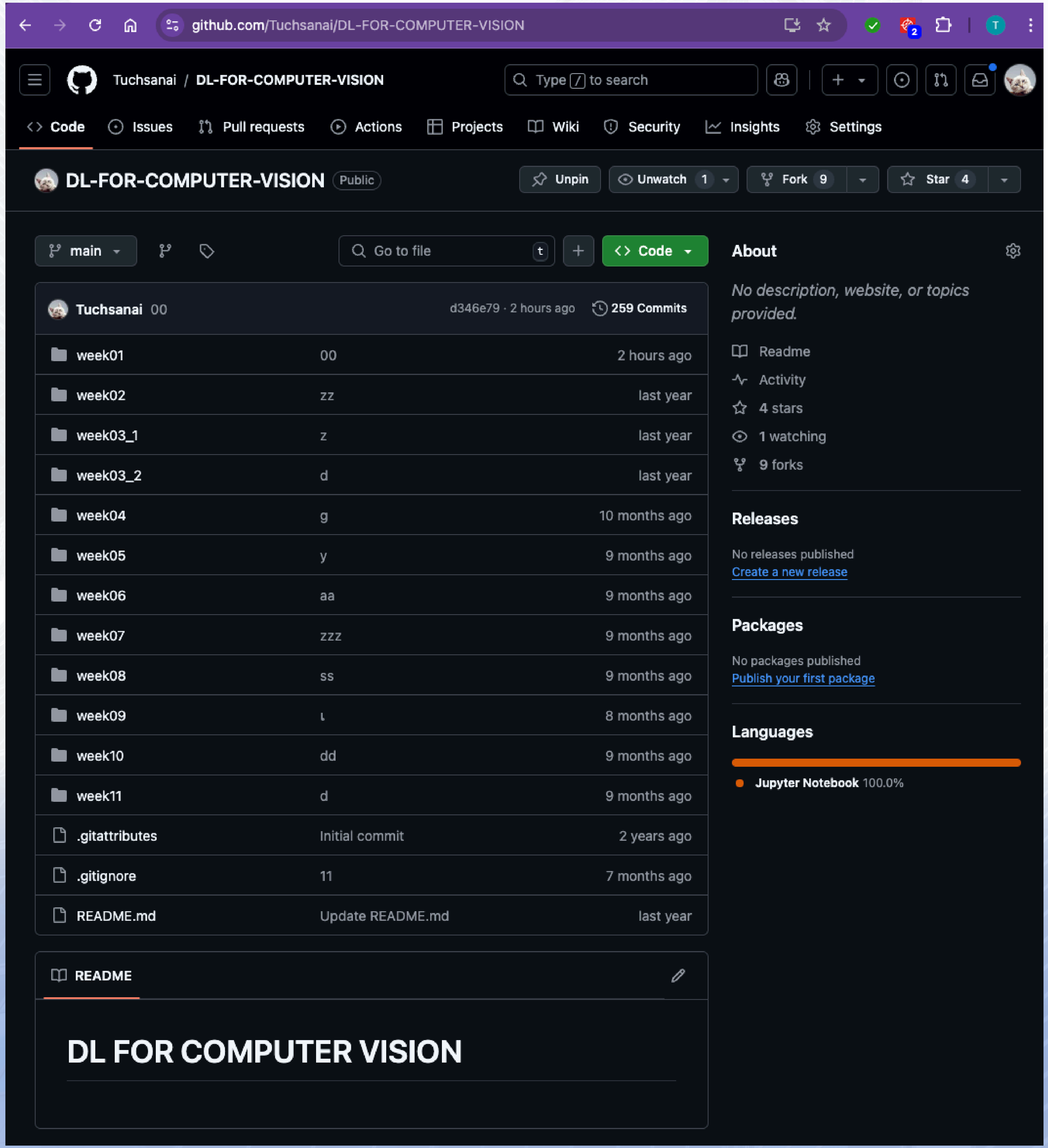
WEEK 1

AI

DEEP LEARNING FOR COMPUTER VISION

Asst. Prof. Dr. Tuchsanai Ploysuwan

Document


github.com/Tuchsanai/DL-FOR-COMPUTER-VISION
Tuchsanai / DL-FOR-COMPUTER-VISION
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
DL-FOR-COMPUTER-VISION
Public
Unpin
Unwatch 1
Fork 9
Star 4
main
Go to file
t
Code
Tuchsanai 00
d346e79 · 2 hours ago
259 Commits
week01 00 2 hours ago
week02 zz last year
week03_1 z last year
week03_2 d last year
week04 g 10 months ago
week05 y 9 months ago
week06 aa 9 months ago
week07 zzz 9 months ago
week08 ss 9 months ago
week09 ι 8 months ago
week10 dd 9 months ago
week11 d 9 months ago
.gitattributes Initial commit 2 years ago
.gitignore 11 7 months ago
README.md Update README.md last year
README
DL FOR COMPUTER VISION
About
No description, website, or topics provided.
Readme
Activity
4 stars
1 watching
9 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Languages
Jupyter Notebook 100.0%

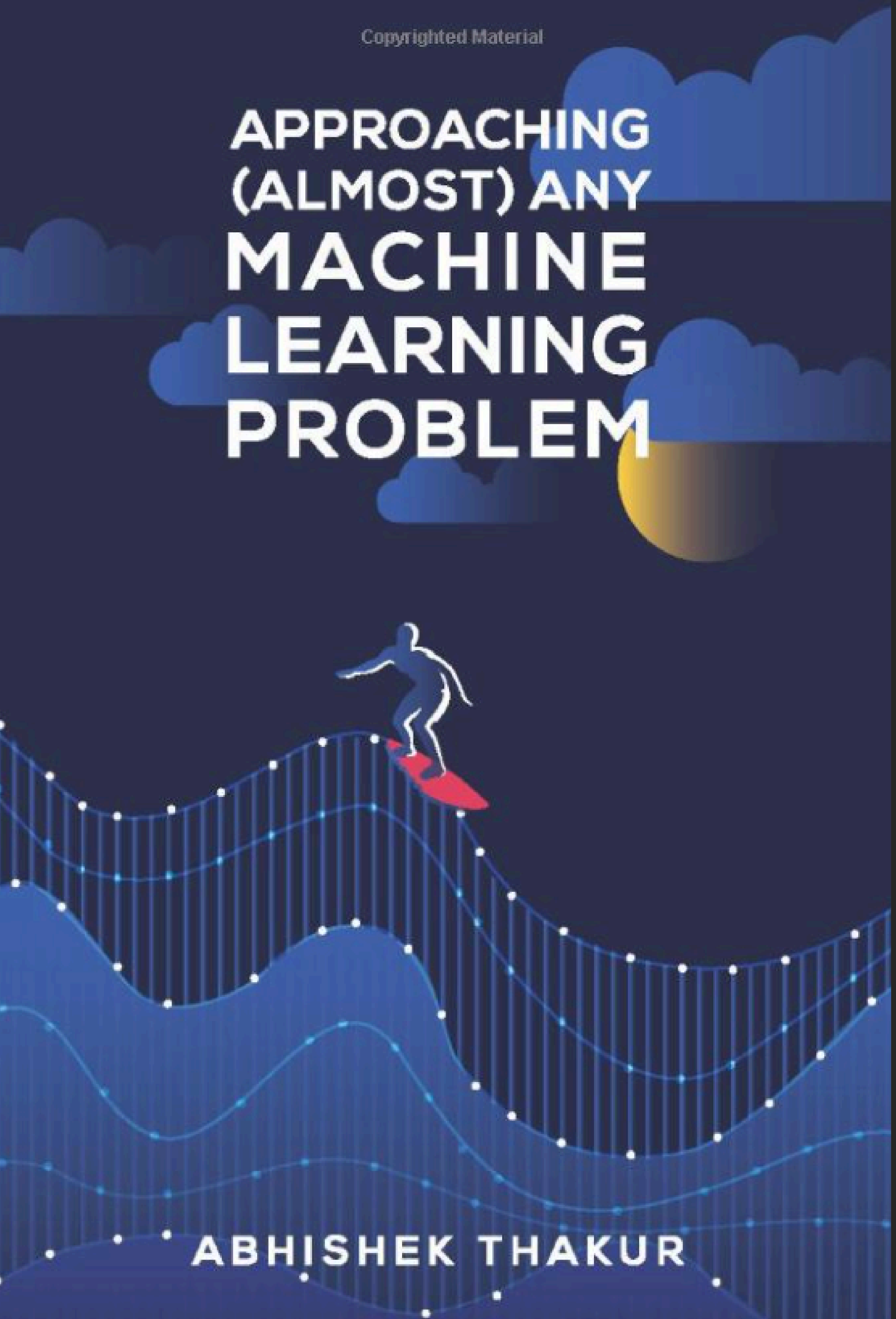
Course Learning Outcomes (CLOs)

Upon completion of this course, students are expected to be able to

CLO ID	CLO Description
CLO-1	ผู้เรียนเข้าใจหลักการพื้นฐานของงาน Deep learning สำหรับงาน Computer Vision
CLO-2	ผู้เรียนเข้าใจหลักการพื้นฐานในการใช้ Convolutional neural network (CNN) สำหรับงาน Image Classifications
CLO-3	ผู้เรียนสามารถอิมพลีเมนต์และเลือกใช้ Modern deep learning architectures ที่เหมาะสมสำหรับงาน Computer Vision ชนิดต่าง เช่น VGG, ResNet, and Efficientnet
CLO-4	ผู้เรียนเข้าใจหลักการการเลือกใช้งาน Loss function, activation functionม stochastic optimization ได้อย่างเหมาะสม
CLO-5	ผู้เรียนเข้าใจหลักการในการใช้ Transfer learning และการ fine tuning ที่เหมาะสม
CLO-6	ผู้เรียนเข้าใจหลักการพื้นฐานในการใช้ Deep learning สำหรับการทำ Image Segmentation ในงาน Computer Vision
CLO-7	ผู้เรียนสามารถเข้าใจหลักการของ Object detection and Object Tracking ในงาน Computer Vision
CLO-8	ผู้เรียนสามารถเข้าใจหลักการของ Generative Adversarial Network

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		

การให้คะแนน	
Midterm Examination	30
Final Examination	30
Mini project, Kaggle	30
Participation	10



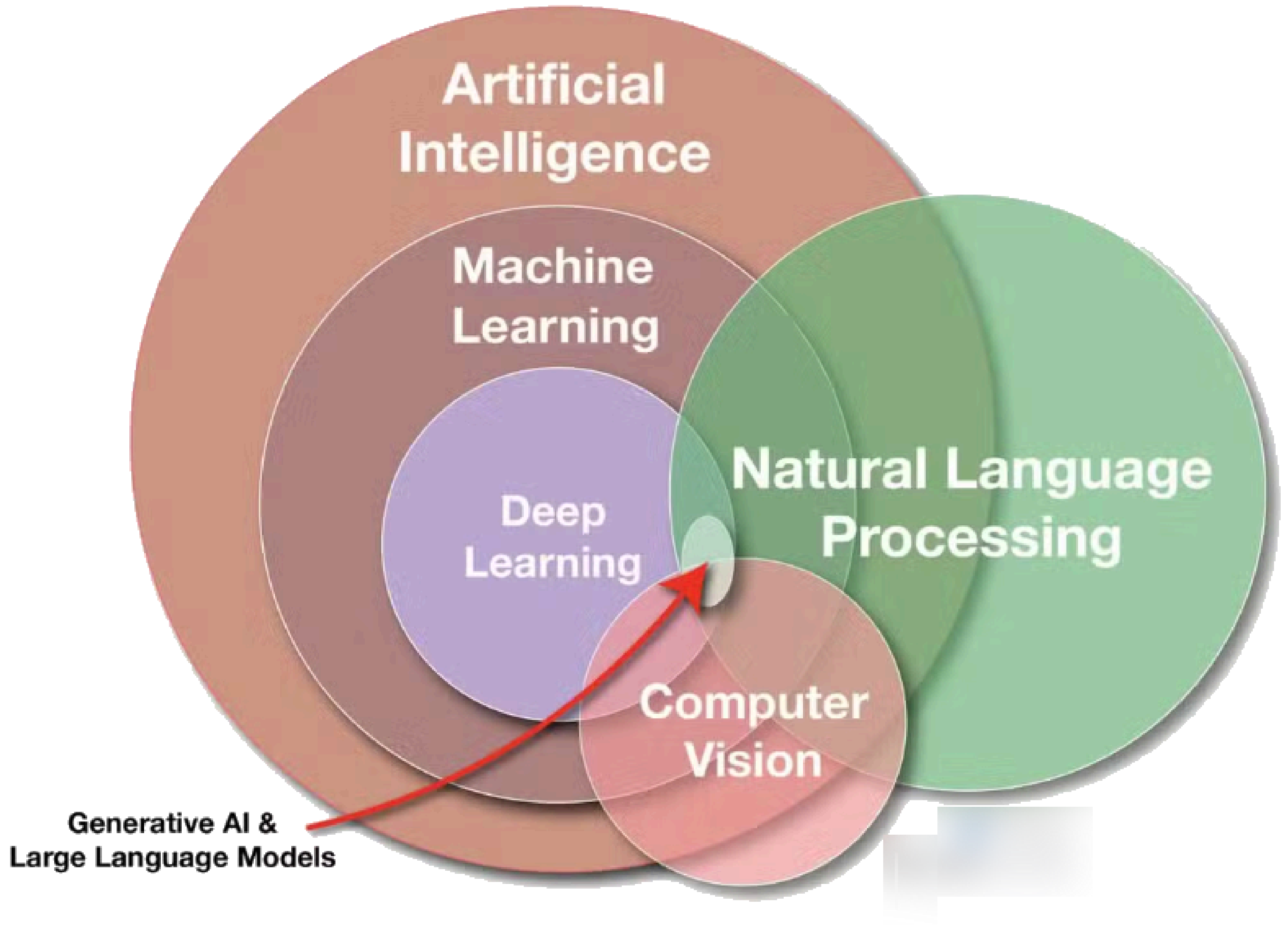
Books

https://github.com/abhishekkrthakur/approachingalmost

Is Computer Vision Artificial Intelligence?

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

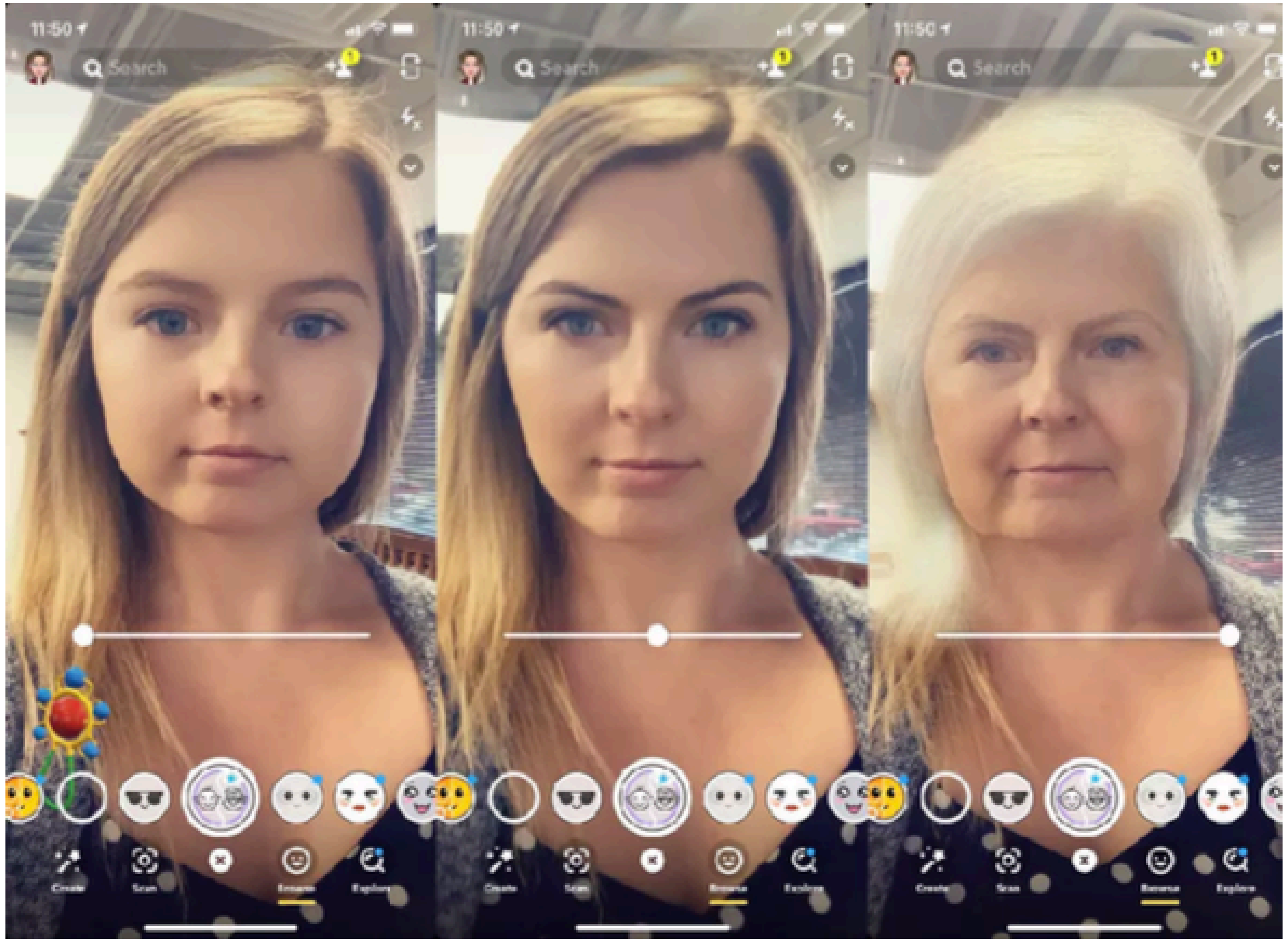
https://www.researchgate.net/figure/Relation-between-Artificial-Intelligence-Machine-Learning-and-Deep-Learning-Computer_fig1_342978934



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

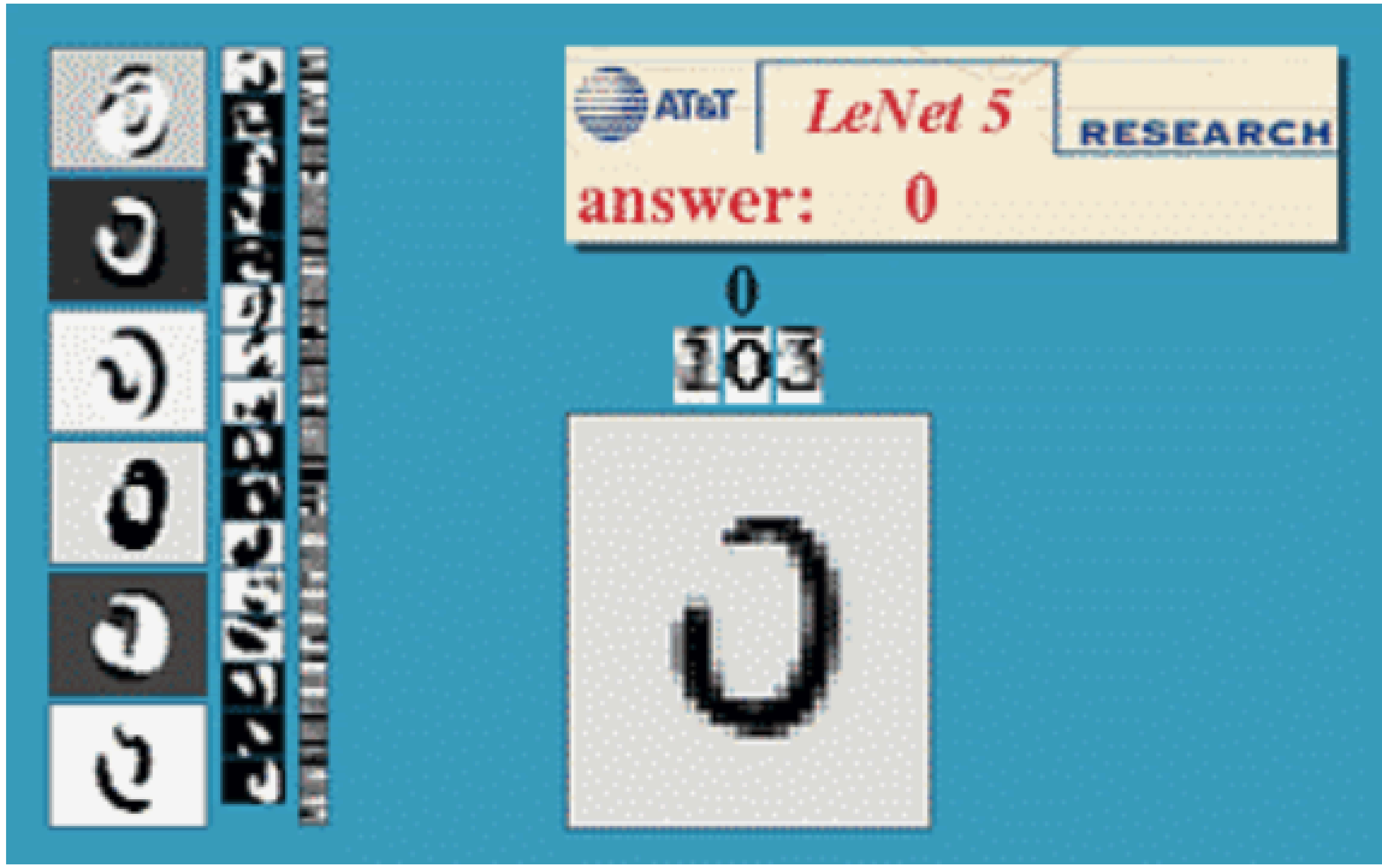


Source - Cnet - Snapchat's Time Machine AR lens creepily shows what you'll look like old

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?

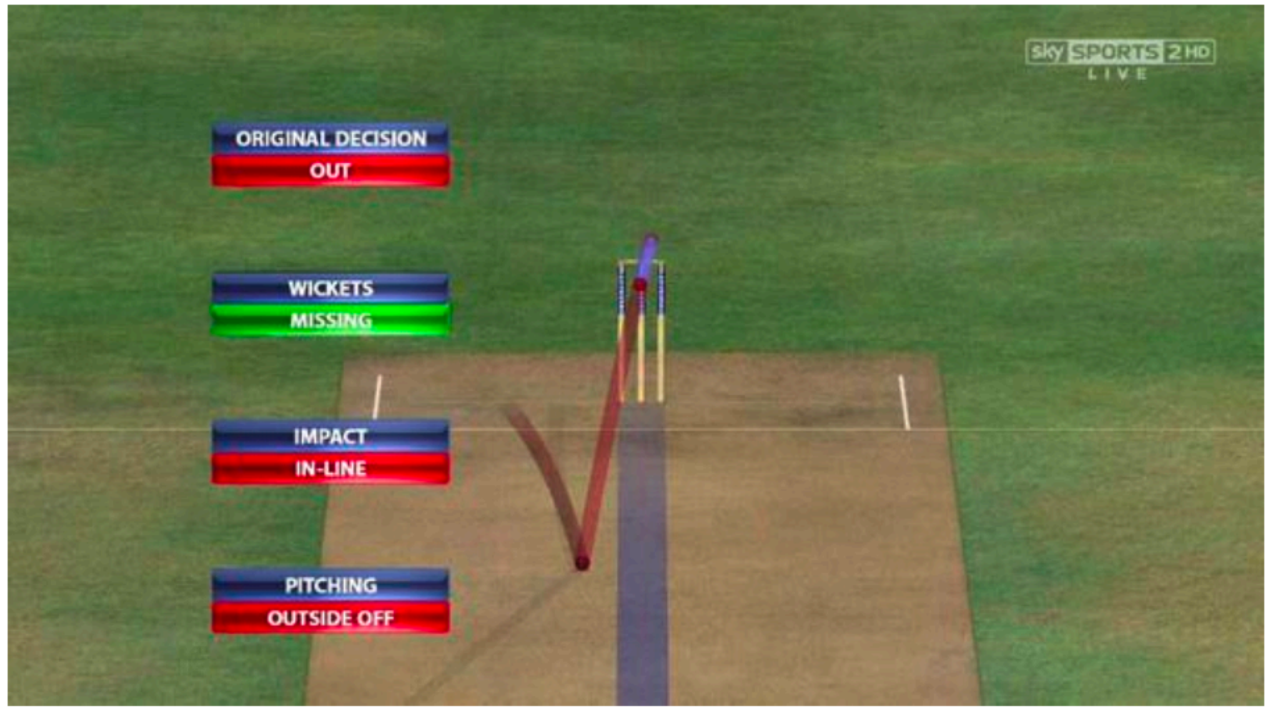
- Snapchat and Instagram filters
- Optical Character Recognition (OCR)
- **Licence Plate Reading**
- Self-driving cars
- Sporting Analysis
- Facial Recognition

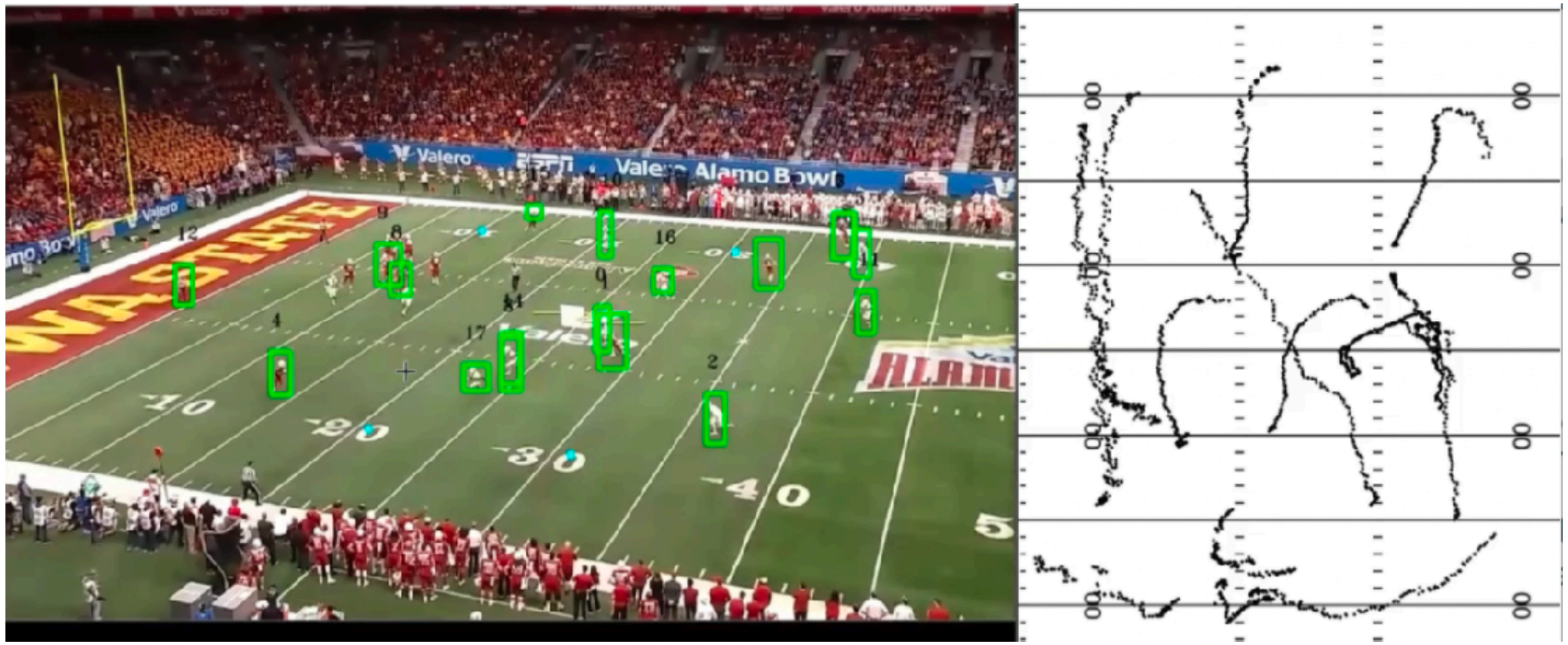
So what can Computer Vision do?

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

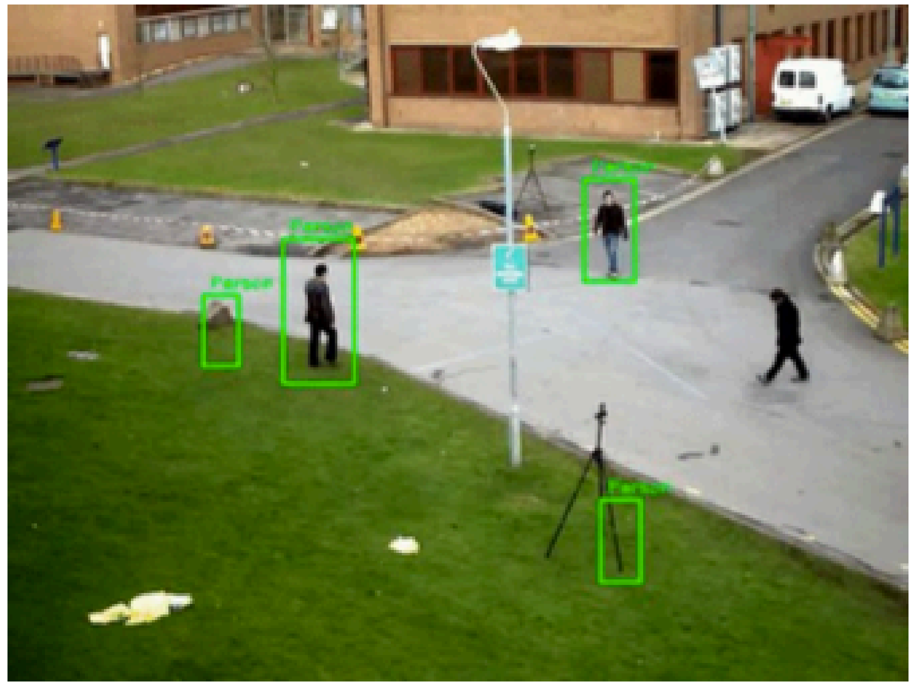




Source -Roboflow - AI Coach

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.

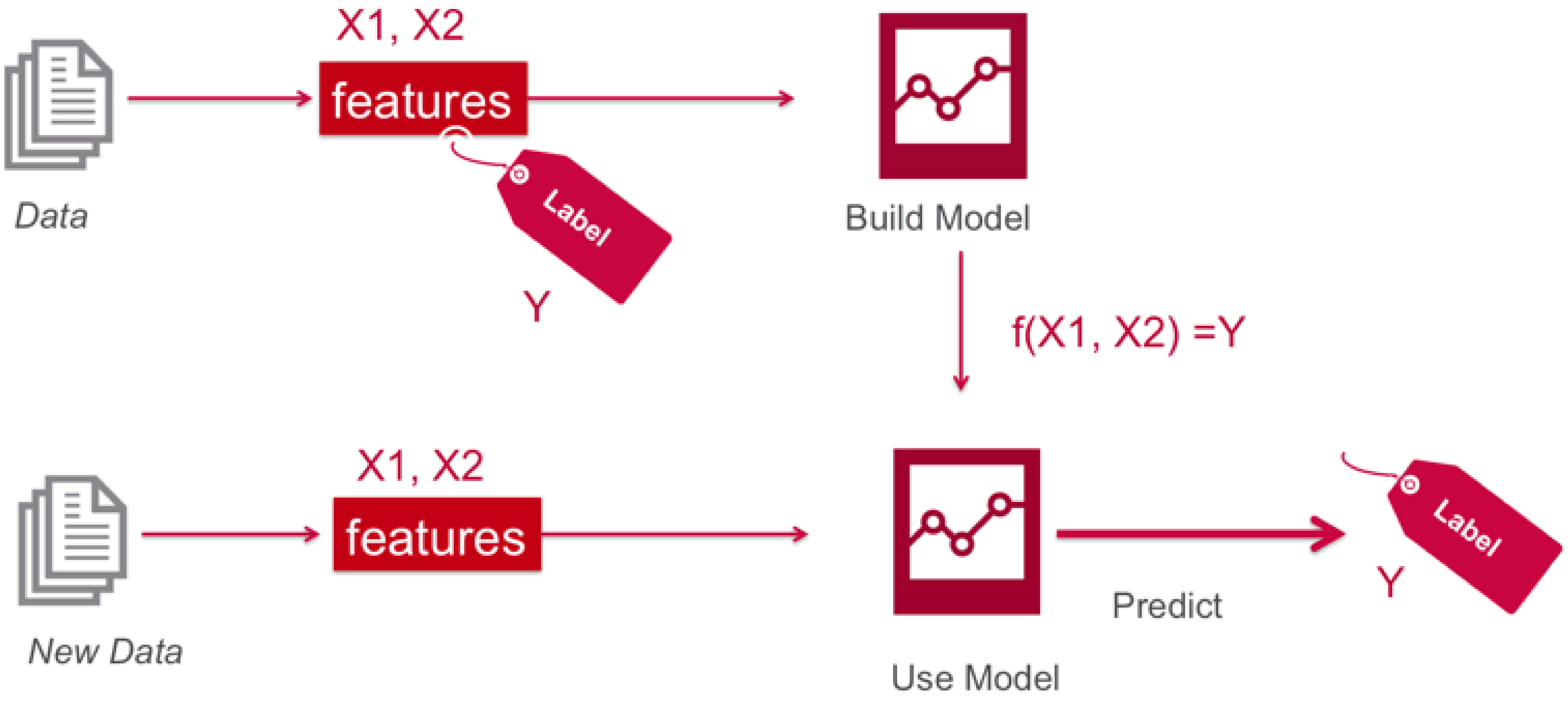
- Until the the mid 2010s…which brought **two important building blocks** together.

  - **Accessible GPU processing** (NVIDIA’s CUDA)

Deep Learning CV vs Classical CV

Deep Learning	**Classical Computer Vision**
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



Source: https://mapr.com/blog/demystifying-ai-ml-dl/

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

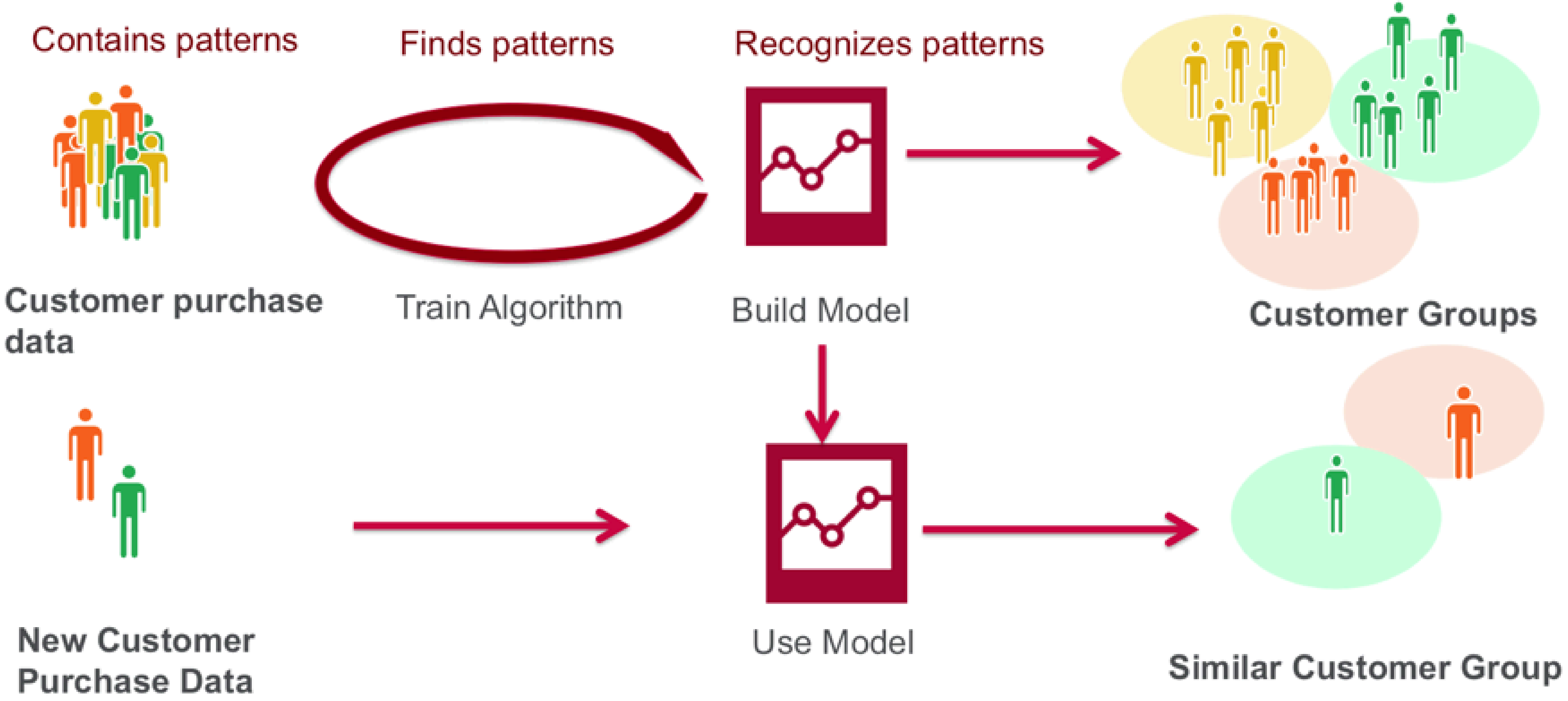
Examples of classification

- 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.

Unsupervised Learning | Self Supervised

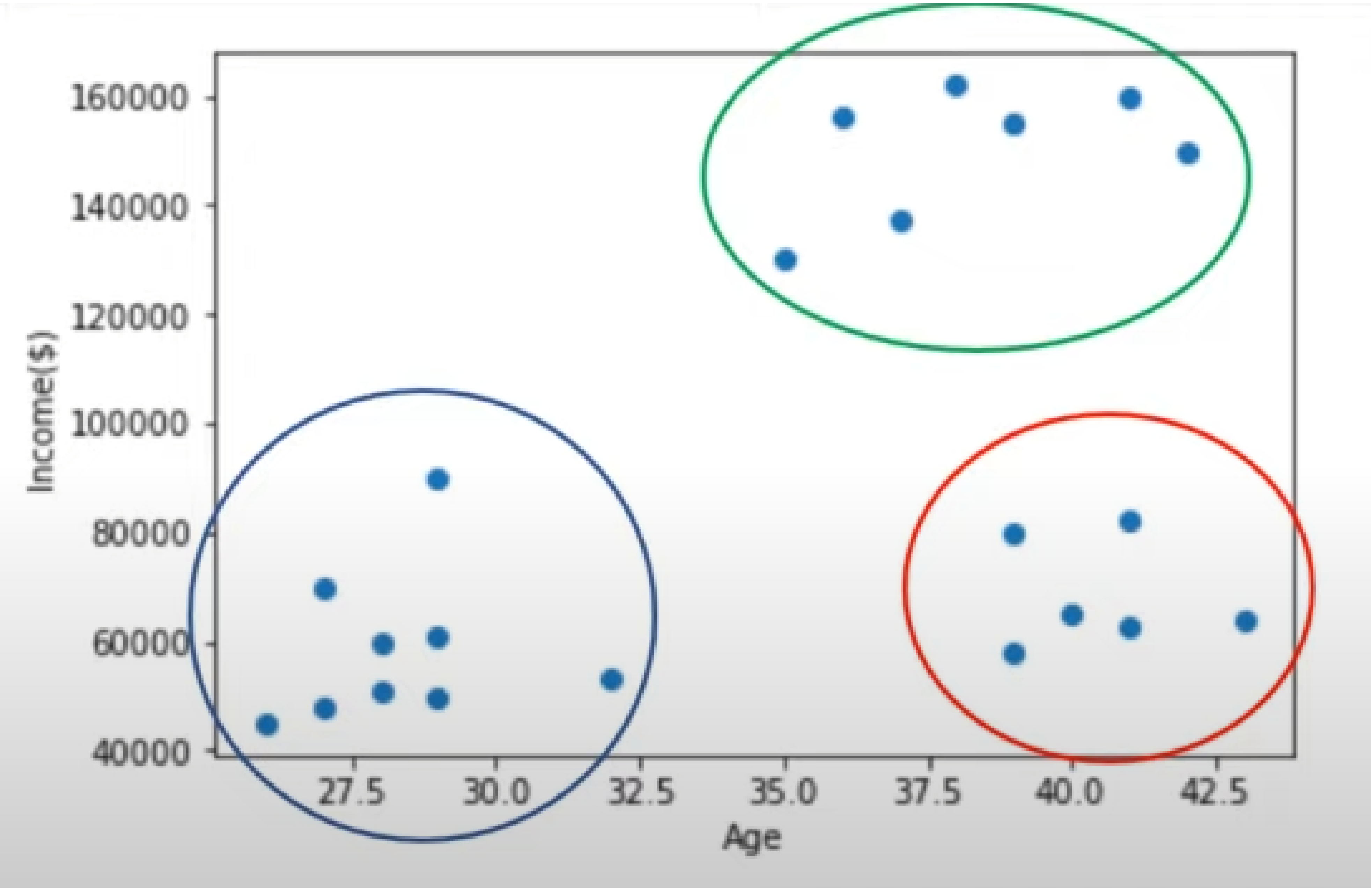


Source: https://mapr.com/blog/demystifying-ai-ml-dl/

Examples of clustering

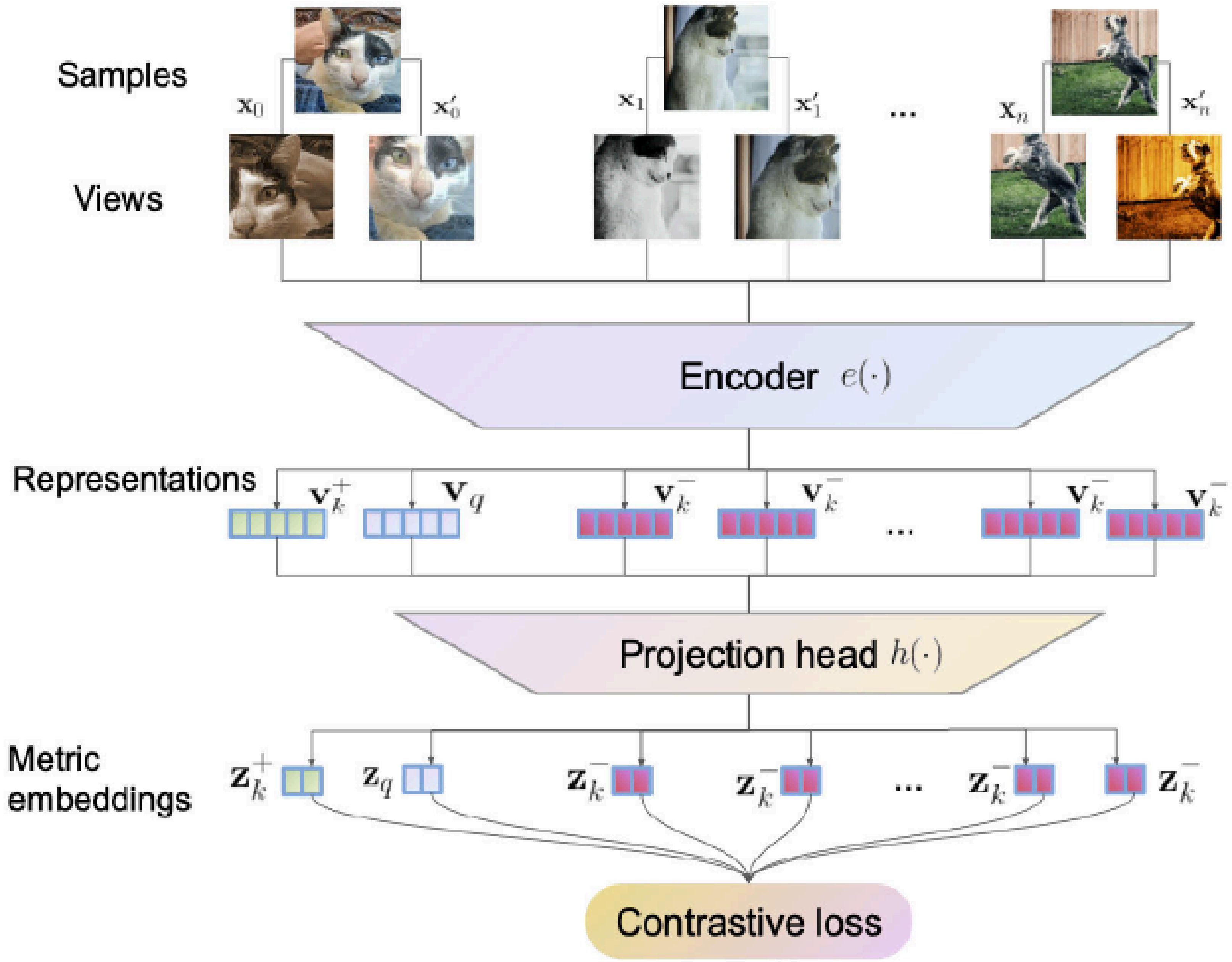
- Search results grouping
- Grouping similar customers
- Grouping similar patients
- Text categorization
- Network Security Anomaly detection

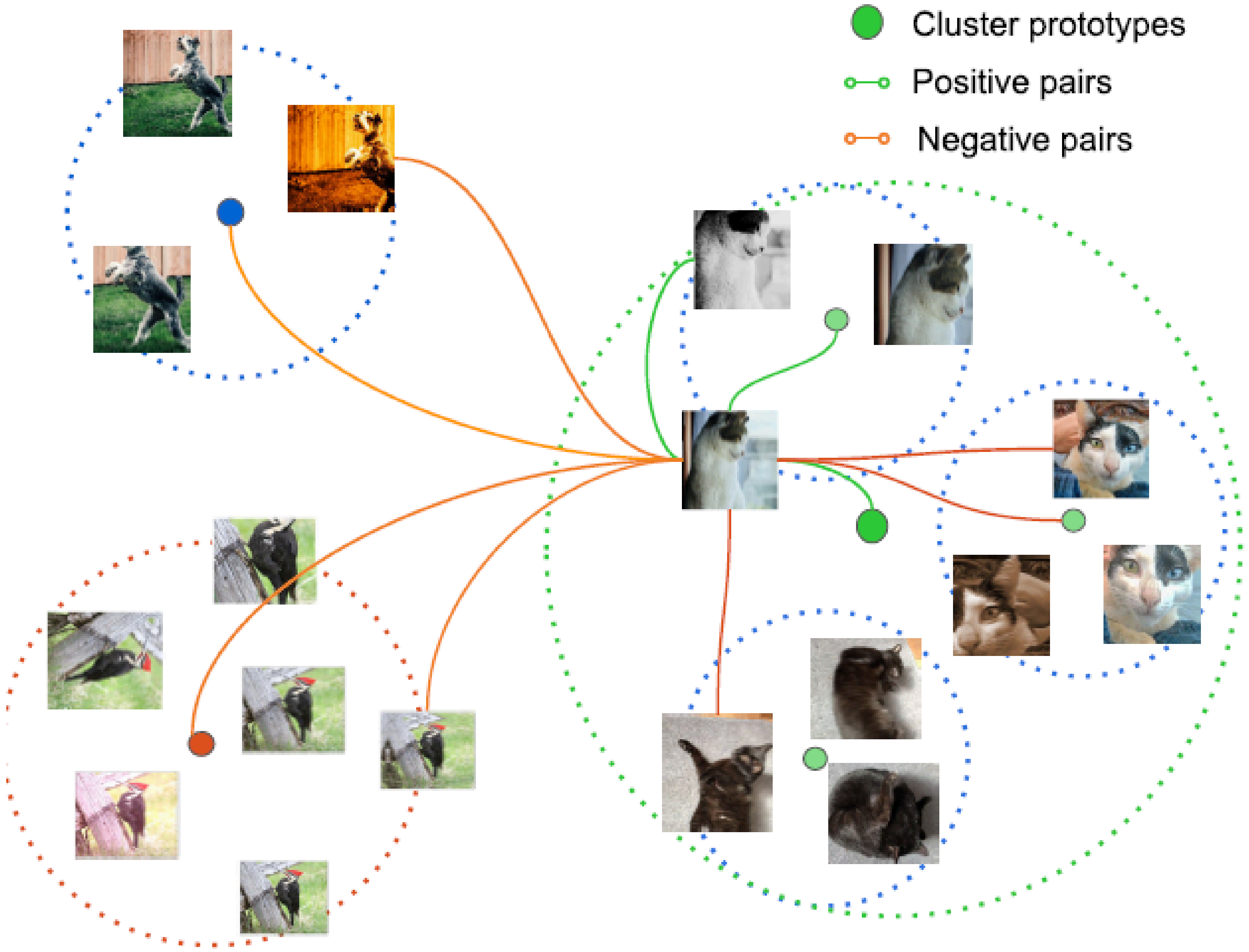
Unsupervised learning



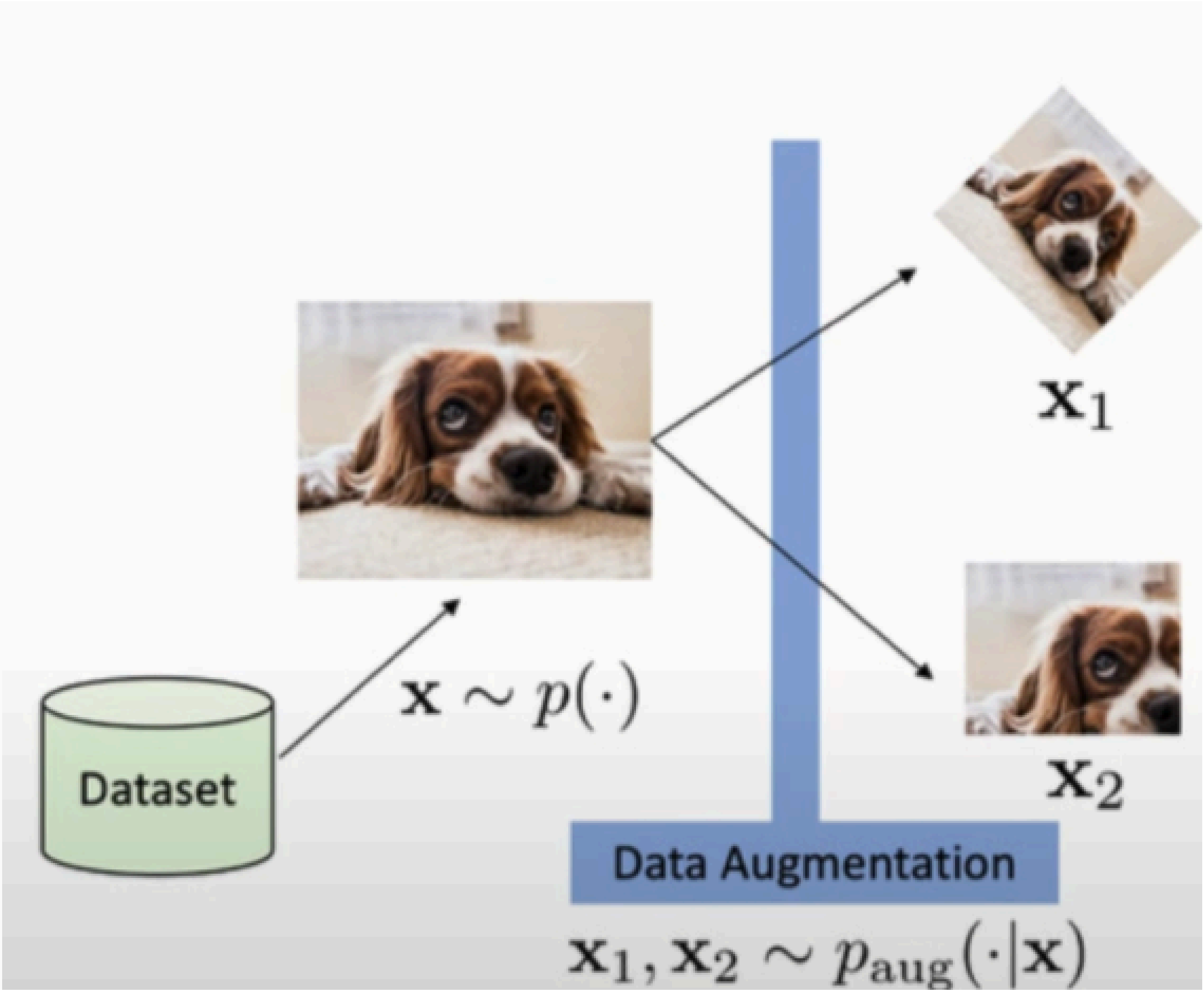
	A	B	C
1	**Name**	**Age**	**Income($)**
2	Rob	27	70000
3	Michael	29	90000
4	Mohan	29	61000
5	Ismail	28	60000
6	Kory	42	150000
7	Gautam	39	155000
8	David	41	160000
9	Andrea	38	162000
10	Brad	36	156000

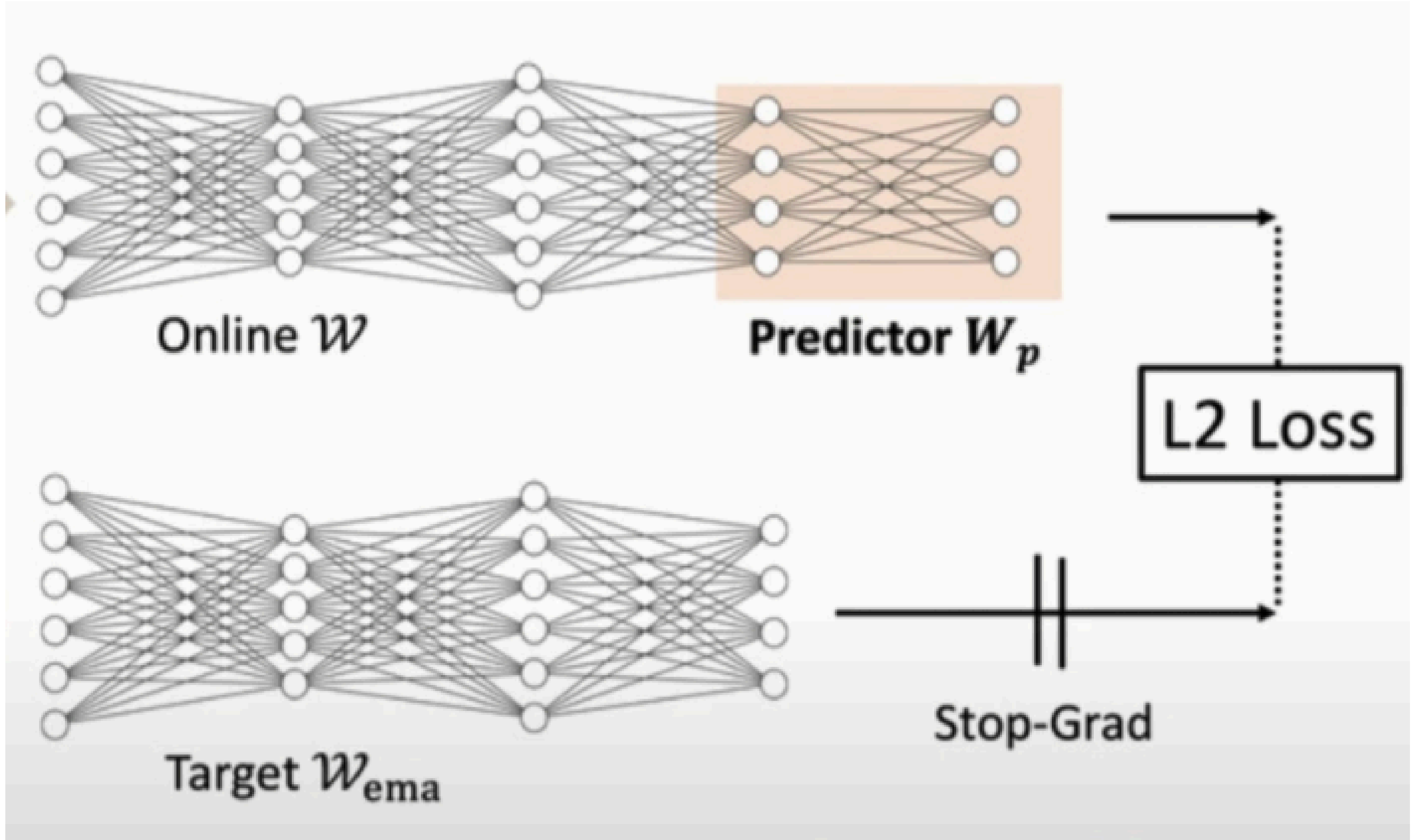
Self-Supervised





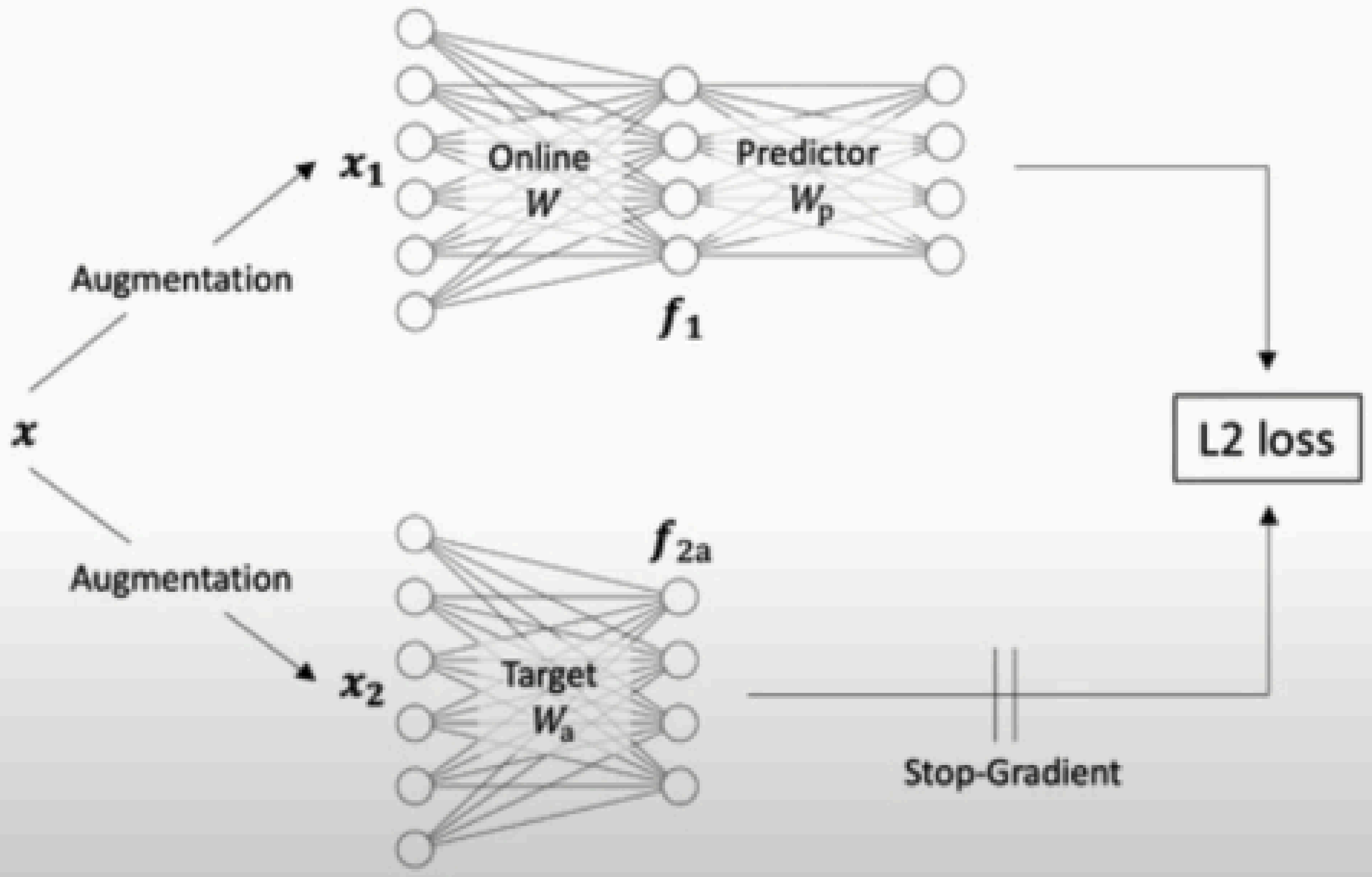
Easy review





Reference : Understanding self-supervised learning dynamics without contrastive | Outstanding Paper | ICML 2021

A simple model



Objective:

$$J(W, W_p) := \frac{1}{2}\mathbb{E}_{\boldsymbol{x}_1, \boldsymbol{x}_2}\left[\|W_p \boldsymbol{f}_1 - \mathrm{StopGrad}(\boldsymbol{f}_{2\mathrm{a}})\|_2^2\right]$$

Linear online network W

Linear target network W_a

Linear predictor W_p

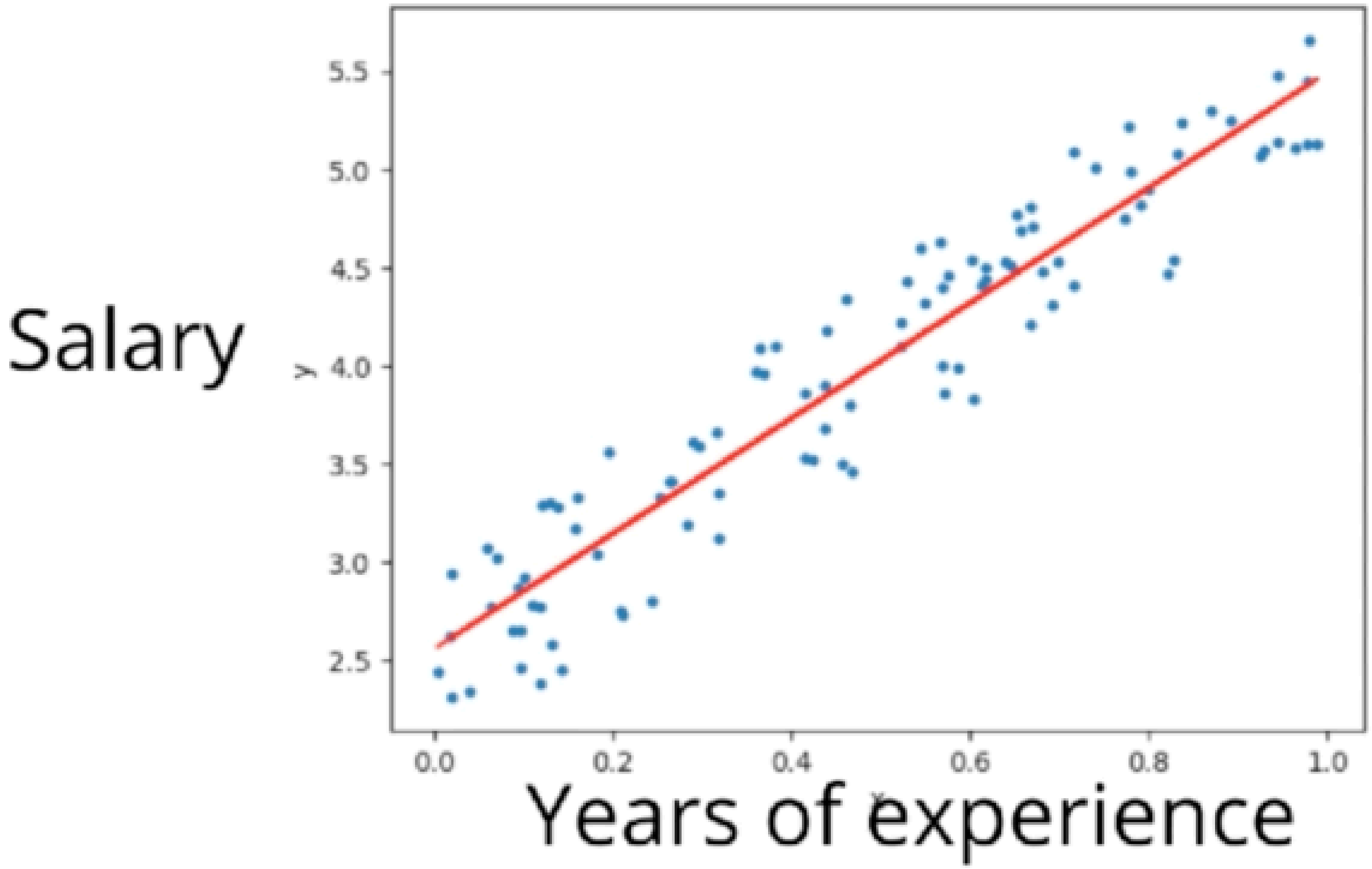
Reference : Understanding self-supervised learning dynamics without contrastive | Outstanding Paper | ICML 2021

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)



```python
import matplotlib.pyplot as plt
import numpy as np
```

✓ 0.4s

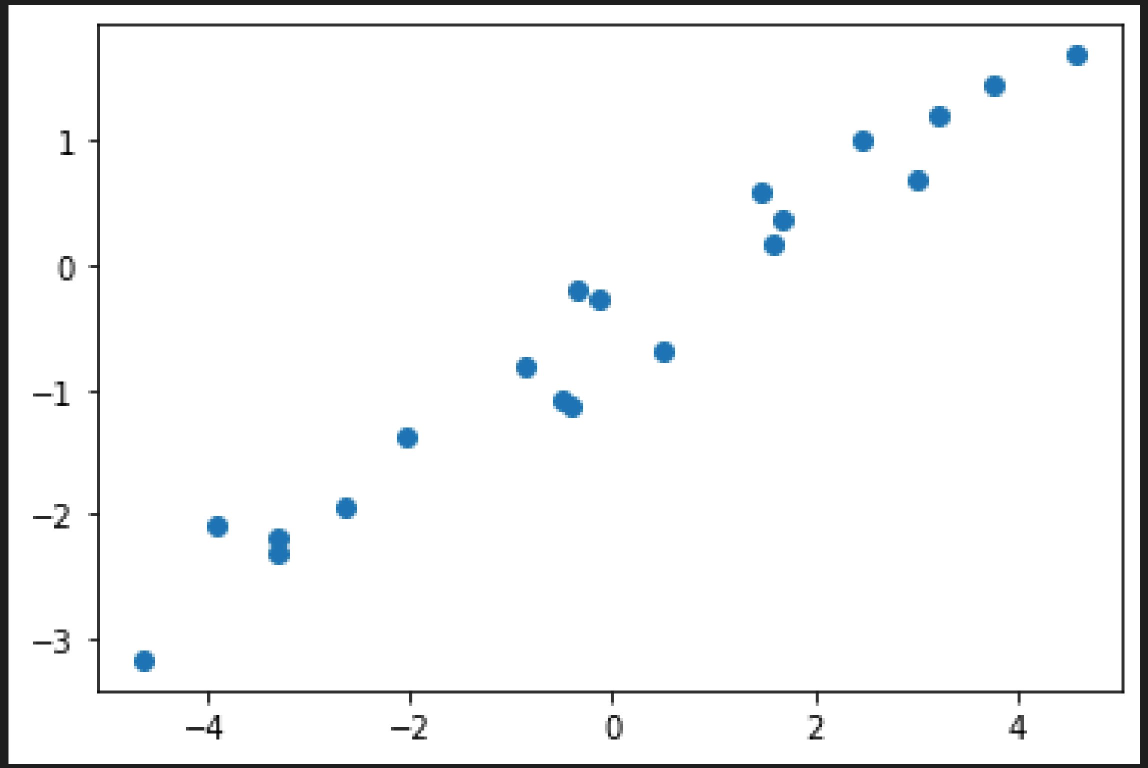
```python
# generate 20 data points
N = 20

#random data on x-axis
x= np.random.rand(N)*10-5

y = 0.5*x -1+ np.random.rand(N)
```

✓ 0.2s

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

✓ 0.1s

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



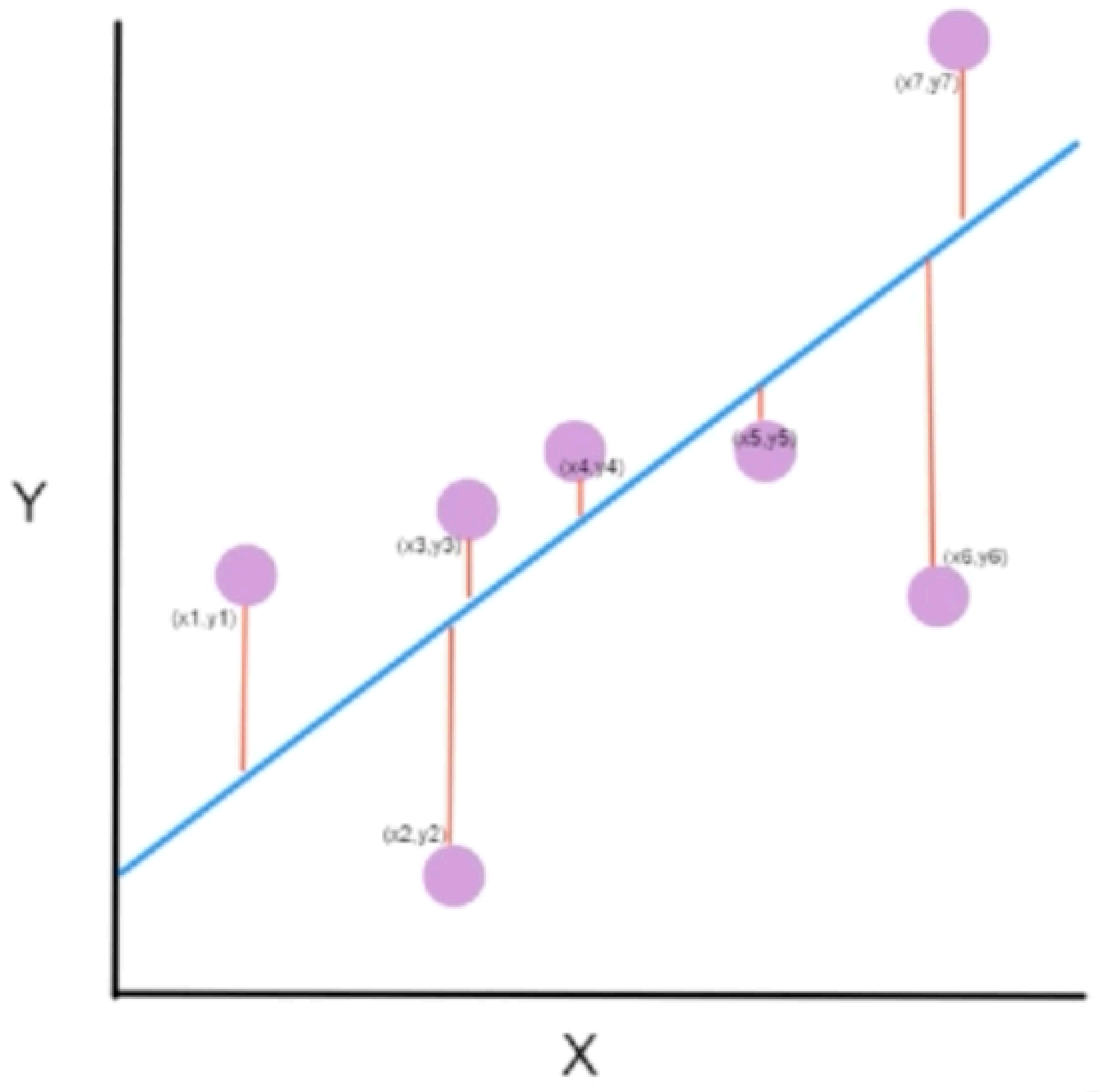
$$\hat{y} = mx + b$$

The Loss

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

MSE = Mean Squared Error

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



Applying the MSE to find slope / intercept

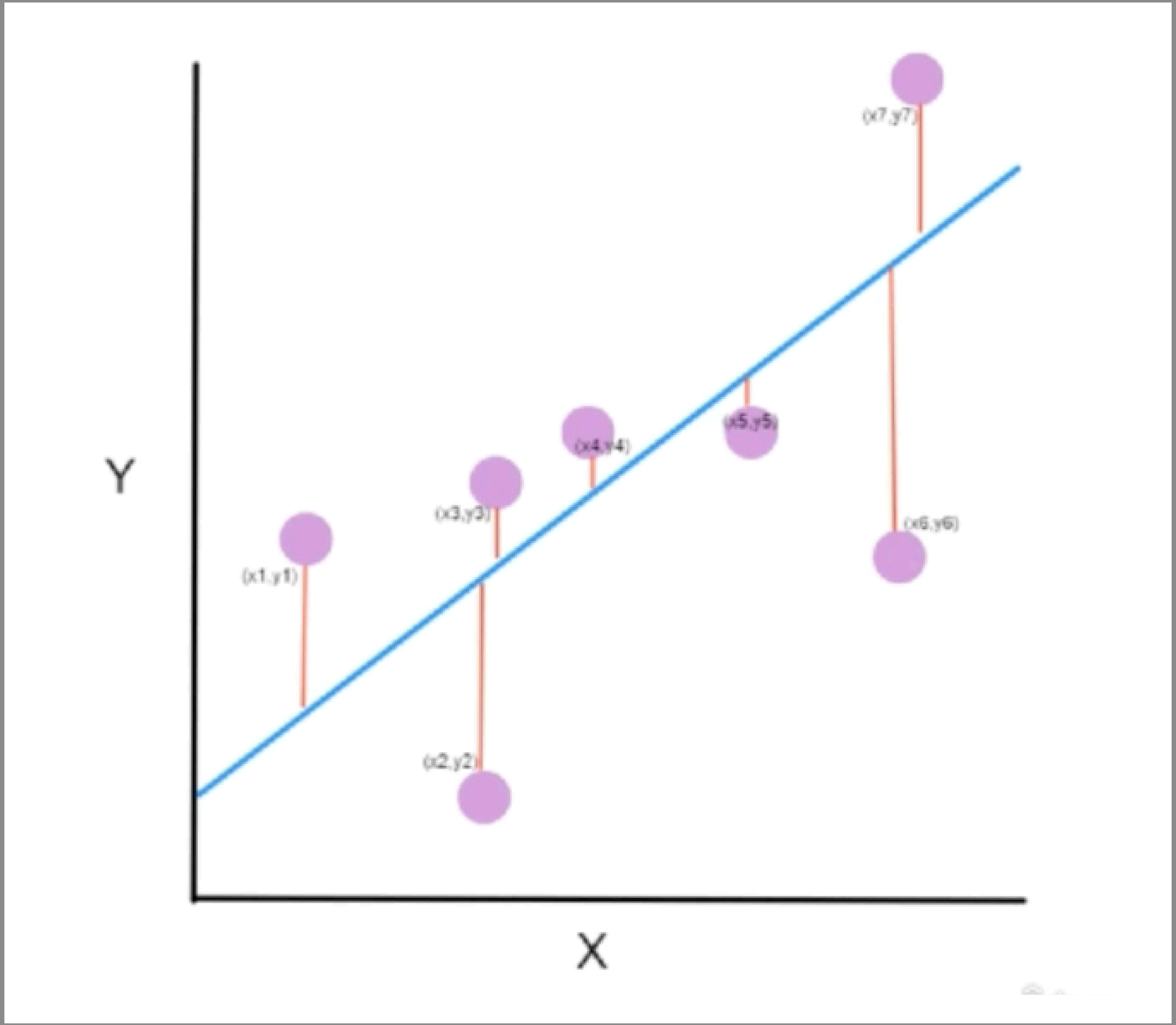
- Plug in the expression for the predictions ($\hat{y}_i$)
- Quiz: What are the **variables** in this expression?

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

Applying the MSE to find slope / intercept

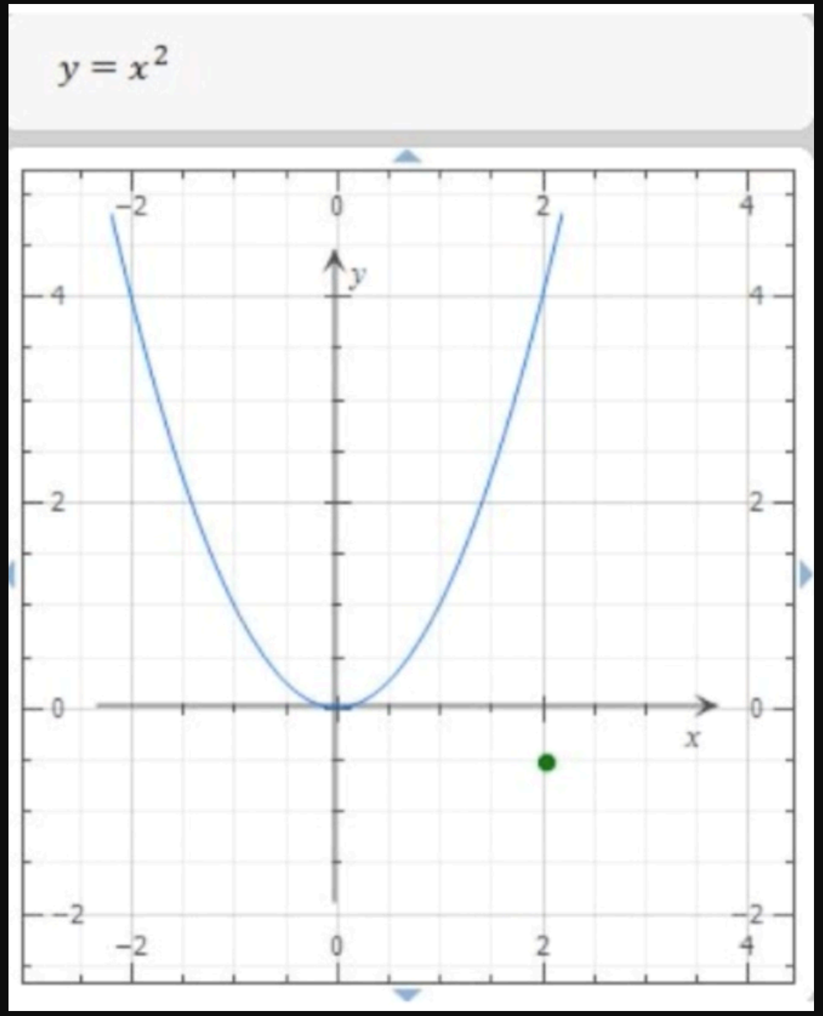
- Note: L (loss) = MSE
- We want to *minimize the loss* with respect to the parameters (*m*, *b*)

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



First Solution

Minimize $f(x) = x^2$



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

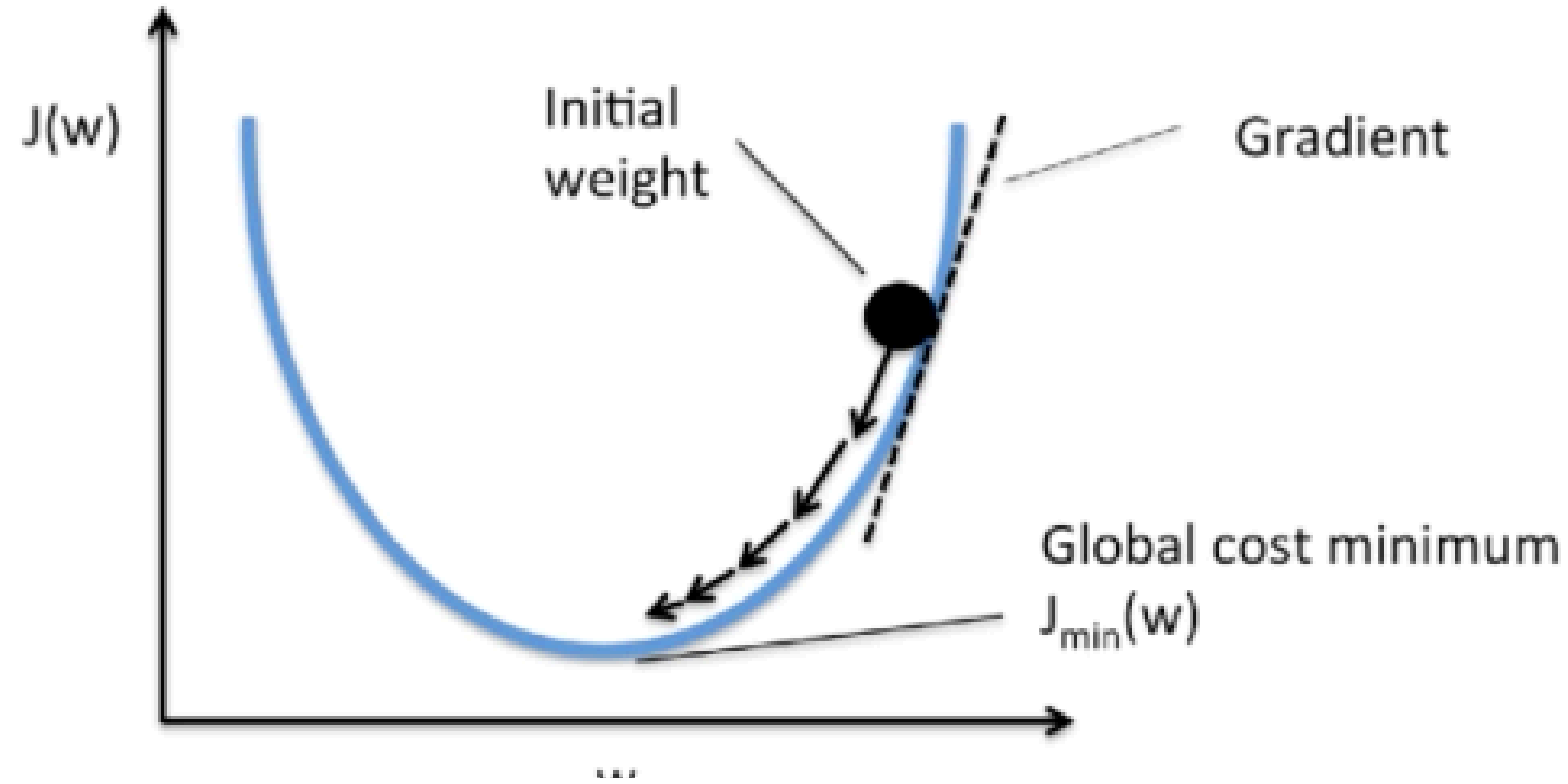
Second Solution : **Gradient Descent**

- 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 $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$

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

Gradient Descent to find hidden parameters

```
# gradient descent
# pseudocode

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

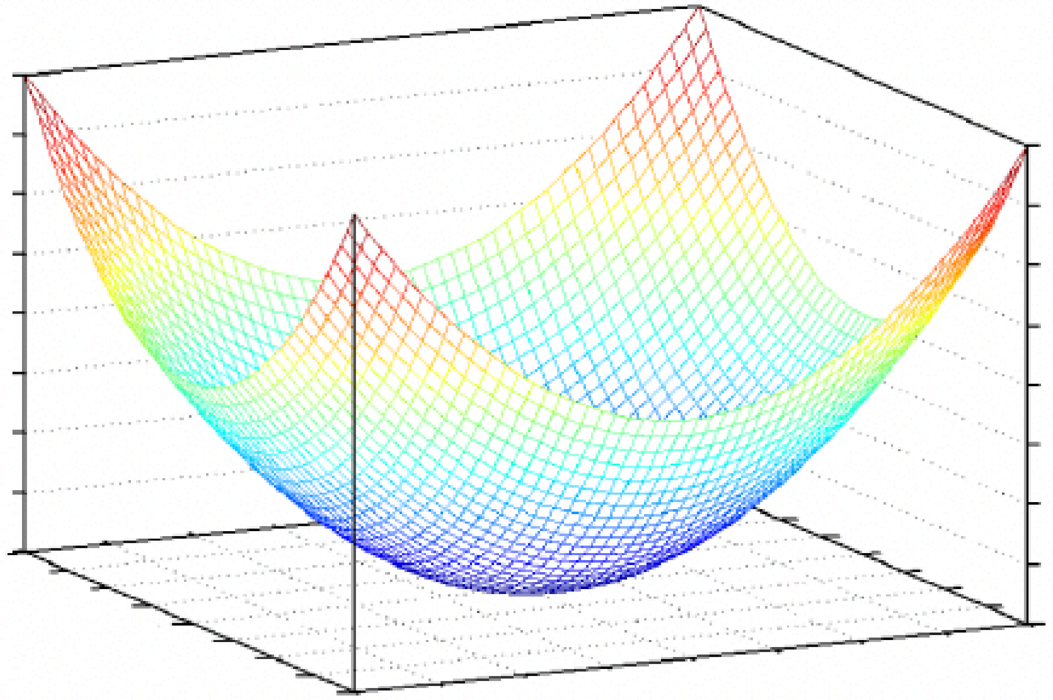


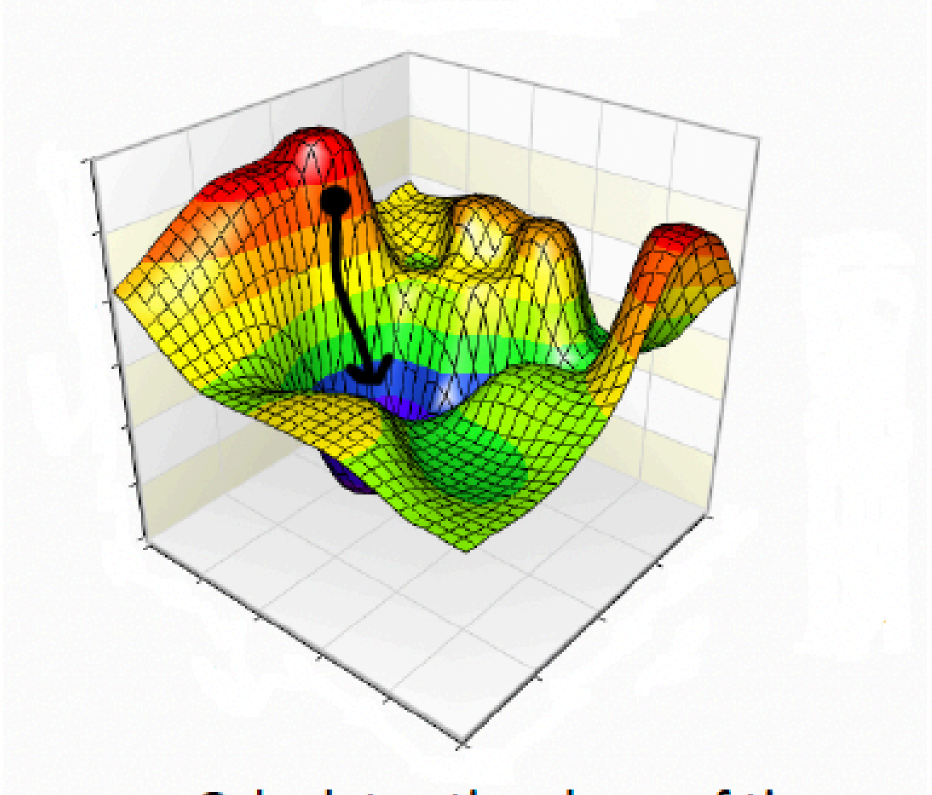
L= $$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

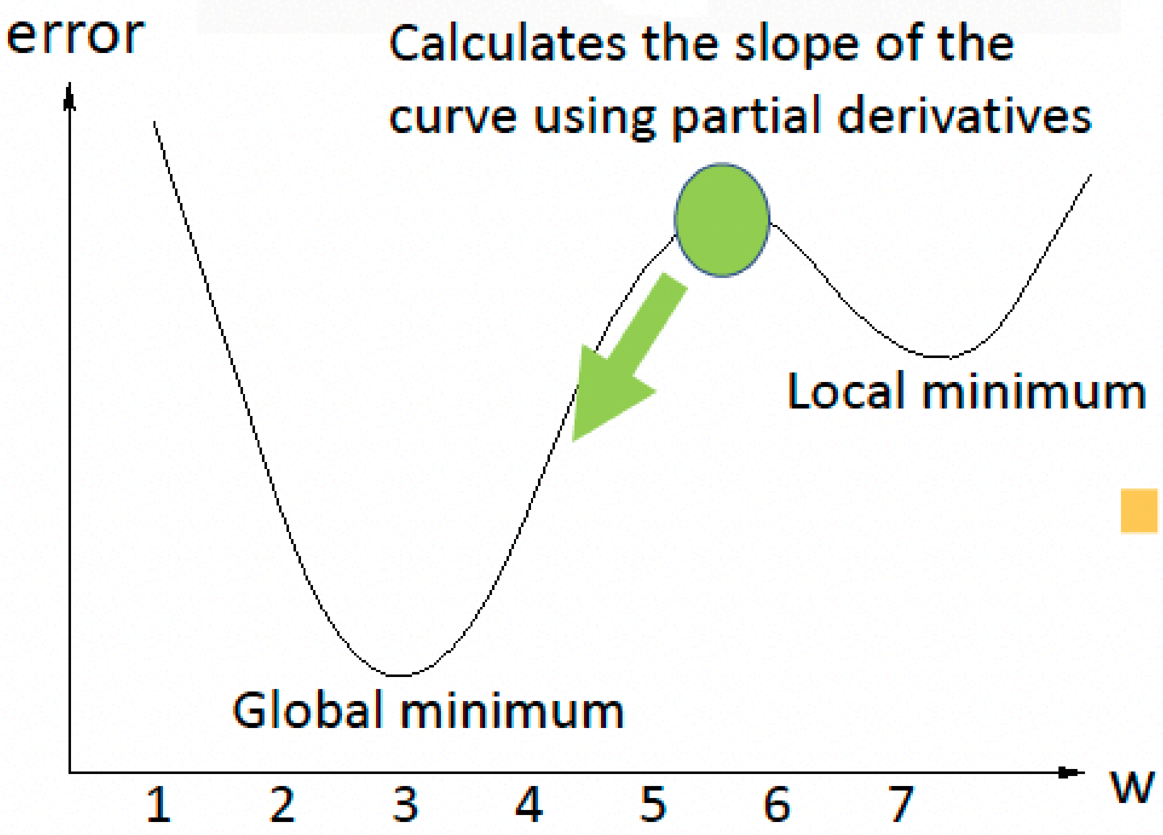
GRADIENT DESCENT

$\min C(w_1, w_2 \ldots w_n)$

Calculate the partial derivative to move to the gradient direction







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

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

```
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]))
```

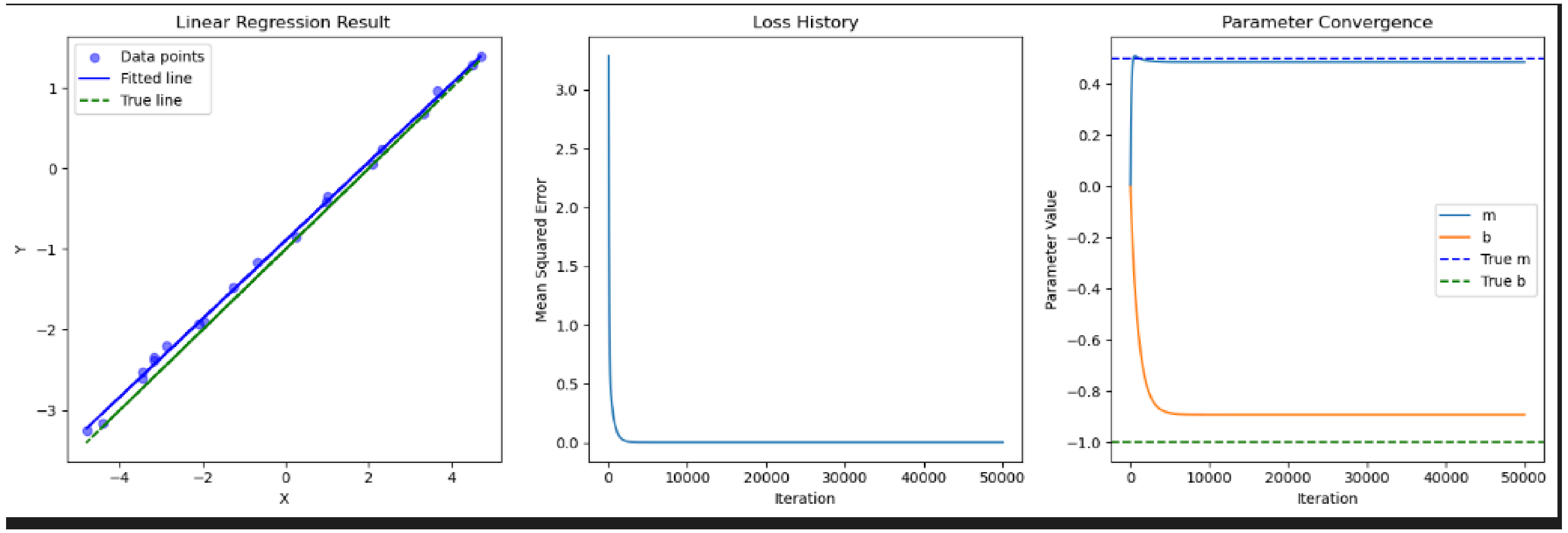
```
# gradient descent
# pseudocode

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

```python
# Define the gradient function
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

# Initialize m and b
m = 0
b = 0

# Set learning rate and number of iterations
learning_rate = 0.001
iterations = 1000

# Perform gradient descent
for _ in range(iterations):
    grad_m, grad_b = Gradient(m, b)
    m -= learning_rate * grad_m
    b -= learning_rate * grad_b

print("Estimated m:", m)
print("Estimated b:", b)
```

✓ 0.0s

```
Estimated m: 0.4860476543029485
Estimated b: -0.8921942796543206
```

```python
# Gradient descent parameters
learning_rate = 0.001
iterations =50000
m = 0  # Initial guess for m
b = 0  # Initial guess for b

# Lists to store parameters history
m_history = []
b_history = []
loss_history = []

# Gradient descent
for i in range(iterations):
    # Calculate gradients
    grad_m, grad_b = Gradient(m, b, X, Y)

    # Update parameters
    m = m - learning_rate * grad_m
    b = b - learning_rate * grad_b

    # Store parameters
    m_history.append(m)
    b_history.append(b)

    # Calculate and store loss
    loss = np.mean((Y - (m*X + b))**2)
    loss_history.append(loss)

print(f"Final parameters: m = {m:.4f}, b = {b:.4f}")
print(f"True parameters: m = 0.5000, b = -1.0000")

# Plotting
plt.figure(figsize=(15, 5))

# Plot 1: Data and regression line
plt.subplot(131)
plt.scatter(X, Y, color='blue', alpha=0.5, label='Data points')
plt.plot(X, m*X + b, color='blue', label='Fitted line')
plt.plot(X, 0.5*X - 1, '--', color='green', label='True line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression Result')
plt.legend()

# Plot 2: Loss history
plt.subplot(132)
plt.plot(loss_history)
plt.xlabel('Iteration')
plt.ylabel('Mean Squared Error')
plt.title('Loss History')

# Plot 3: Parameter convergence
plt.subplot(133)
plt.plot(m_history, label='m')
plt.plot(b_history, label='b')
plt.axhline(y=0.5, color='b', linestyle='--', label='True m')
plt.axhline(y=-1, color='g', linestyle='--', label='True b')
plt.xlabel('Iteration')
plt.ylabel('Parameter Value')
plt.title('Parameter Convergence')
plt.legend()

plt.tight_layout()
plt.show()
```
✓ 1.0s




```
Final parameters: m = 0.4860, b = -0.8922
True parameters: m = 0.5000, b = -1.0000
```

Third Solution :Pytorch

SDG: Stochastic gradient descent

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

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

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

```
# gradient descent
# pseudocode

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

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

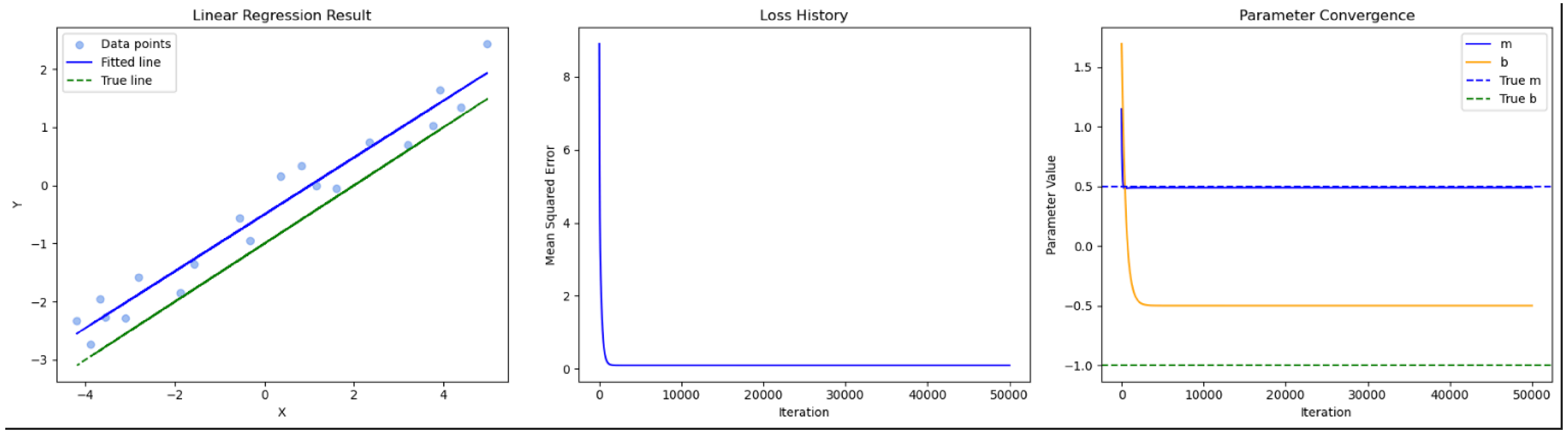
```python
# Train the model
n_epochs = 30
for it in range(n_epochs):
  # zero the parameter gradients
  optimizer.zero_grad()

  # Forward pass
  outputs = model(inputs)
  loss = criterion(outputs, targets)

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

```python
# Convert data to PyTorch tensors
X_train = torch.tensor(X, dtype=torch.float32).view(-1, 1)
Y_train = torch.tensor(Y, dtype=torch.float32).view(-1, 1)

# Initialize m and b as PyTorch parameters
m = torch.randn(1, requires_grad=True, dtype=torch.float32)  # Random initialization for m
b = torch.randn(1, requires_grad=True, dtype=torch.float32)  # Random initialization for b

# Define optimizer
learning_rate = 0.01
optimizer = torch.optim.SGD([m, b], lr=learning_rate)  # Optimizer for m and b

# Training loop
epochs = 1000
for epoch in range(epochs):

    optimizer.zero_grad()  # Clear gradients

    # Forward pass: Calculate y_pred using y = mx + b
    y_pred = m * X_train + b
    # Calculate Mean Squared Error (MSE) Loss
    loss = torch.mean((y_pred - Y_train) ** 2)

    # Backward pass: Compute gradients

    loss.backward()  # Calculate new gradients

    # Update parameters using optimizer1
    optimizer.step()

    # Print the loss every 100 epochs
    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

print("Estimated m:", m.item())
print("Estimated b:", b.item())
```

✓ 0.4s

```
Epoch [100/1000], Loss: 0.0752
Epoch [200/1000], Loss: 0.0240
Epoch [300/1000], Loss: 0.0230
Epoch [400/1000], Loss: 0.0230
Epoch [500/1000], Loss: 0.0230
Epoch [600/1000], Loss: 0.0230
Epoch [700/1000], Loss: 0.0230
Epoch [800/1000], Loss: 0.0230
Epoch [900/1000], Loss: 0.0230
Epoch [1000/1000], Loss: 0.0230
Estimated m: 0.48729196190834045
Estimated b: -0.7519025206565857
```

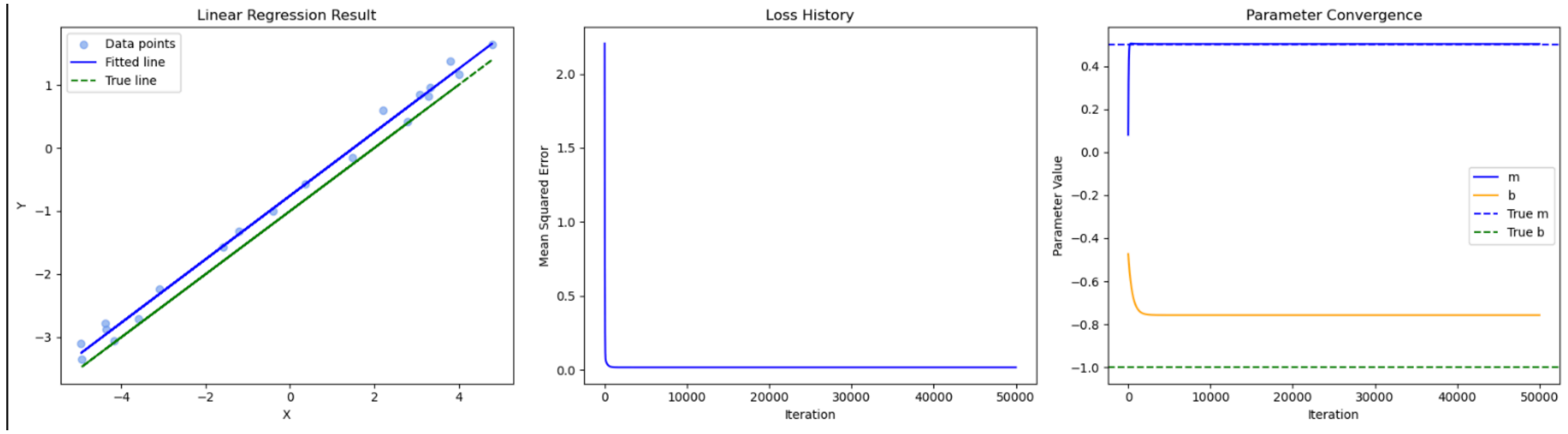
Pytorch Method 1



Pytorch Method 2

```python
# Convert data to PyTorch tensors
X_train = torch.tensor(X, dtype=torch.float32).view(-1, 1)
Y_train = torch.tensor(Y, dtype=torch.float32).view(-1, 1)

# Define the model class
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super(LinearRegressionModel, self).__init__()
        # Initialize m and b as parameters with random values
        self.m = nn.Parameter(torch.randn(1, dtype=torch.float32))
        self.b = nn.Parameter(torch.randn(1, dtype=torch.float32))

    def forward(self, x):
        # Apply the linear transformation y = mx + b
        return self.m * x + self.b
```

```python
# Initialize model and optimizer
model = LinearRegressionModel()
learning_rate = 0.001
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# Training loop with data recording for plots
epochs = 50000
loss_history = []
m_values, b_values = [], []

for epoch in range(epochs):
    # Forward pass
    y_pred = model(X_train)

    # Calculate Mean Squared Error (MSE) Loss
    loss = torch.mean((y_pred - Y_train) ** 2)
    loss_history.append(loss.item())
    m_values.append(model.m.item())
    b_values.append(model.b.item())

    # Backward pass and parameter update
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```



Pytorch Method 3

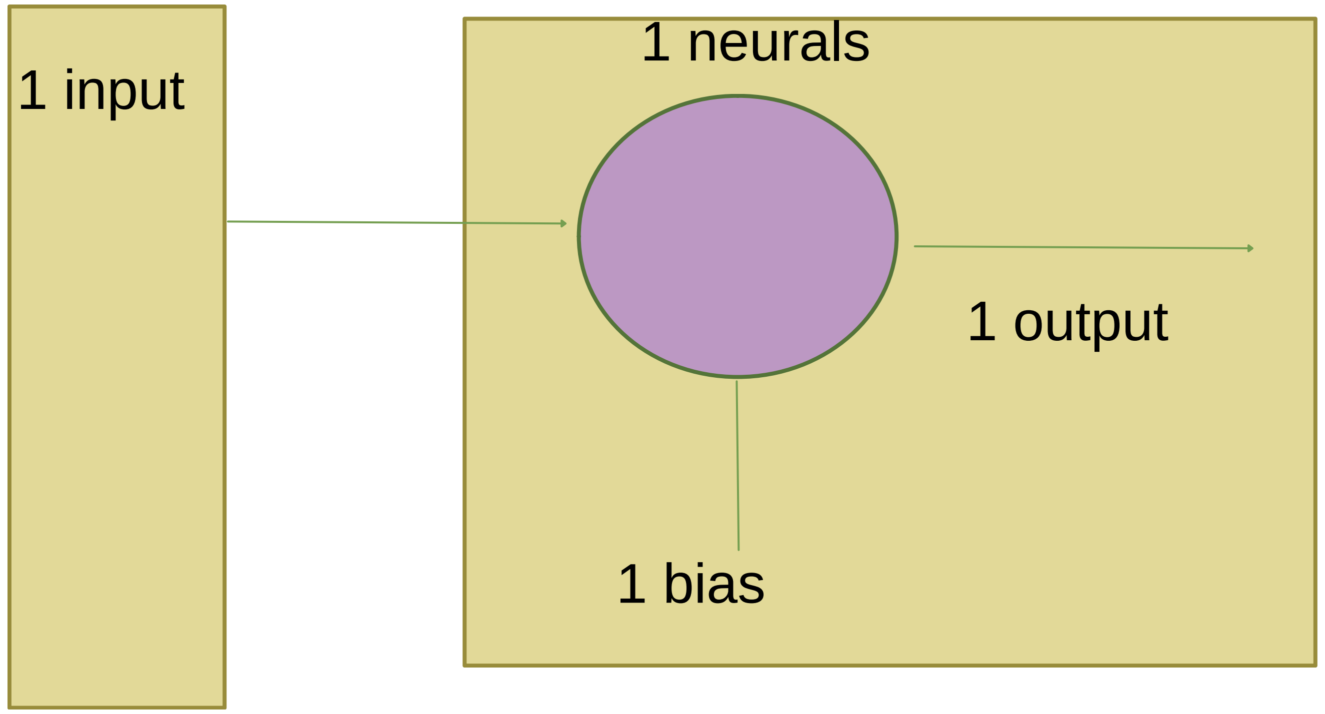
```python
# Train the model
n_epochs = 30
losses = []
for it in range(n_epochs):
  # zero the parameter gradients
  optimizer.zero_grad()

  # Forward pass
  outputs = model(inputs)
  loss = criterion(outputs, targets)

  # keep the loss so we can plot it later
  losses.append(loss.item())

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

  print(f'Epoch {it+1}/{n_epochs}, Loss: {loss.item():.4f}')
```

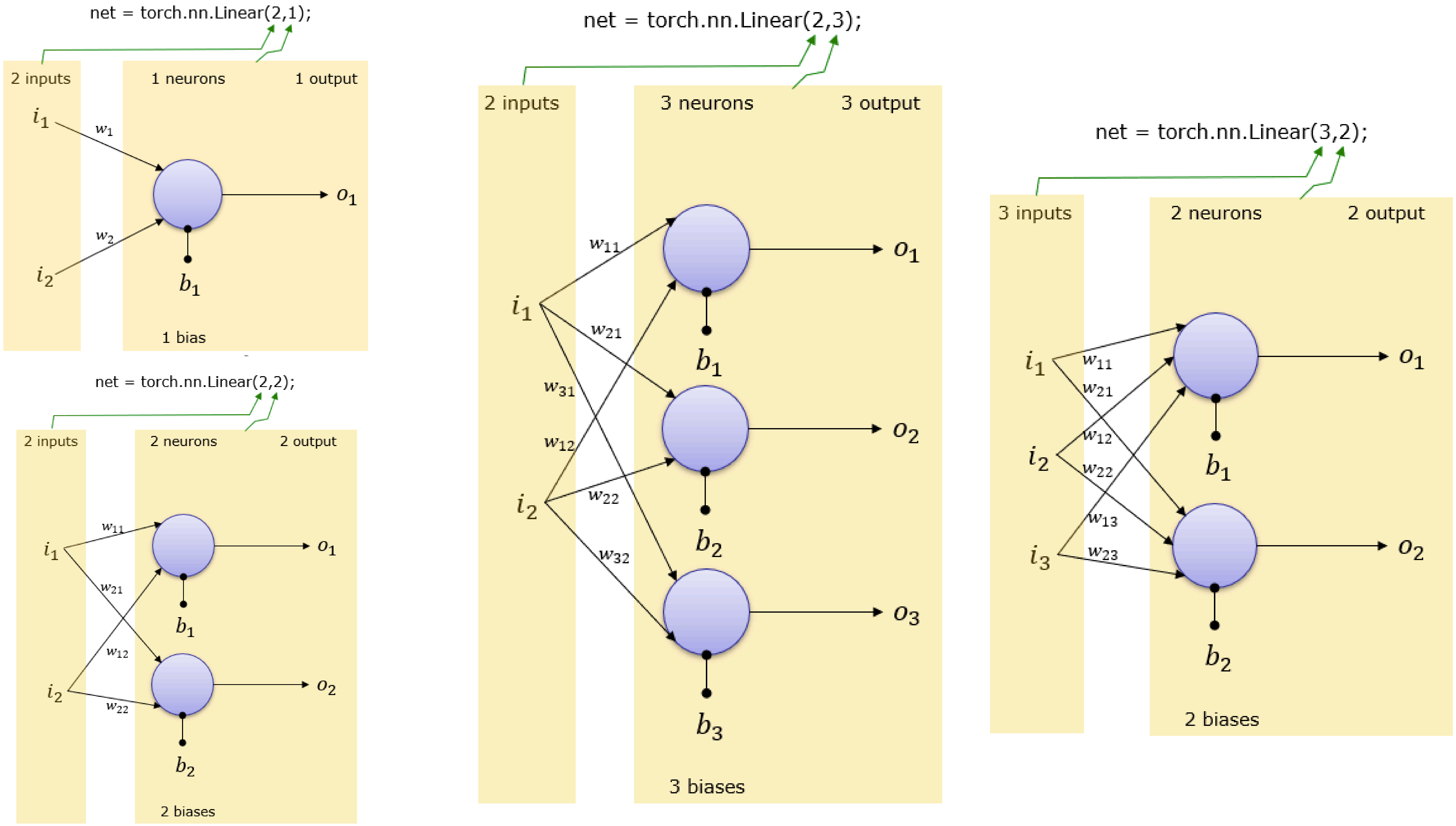
```
Epoch 1/30, Loss: 12.1352
Epoch 2/30, Loss: 2.8066
Epoch 3/30, Loss: 2.4554
Epoch 4/30, Loss: 2.1753
Epoch 5/30, Loss: 1.9483
Epoch 6/30, Loss: 1.7641
Epoch 7/30, Loss: 1.6148
```

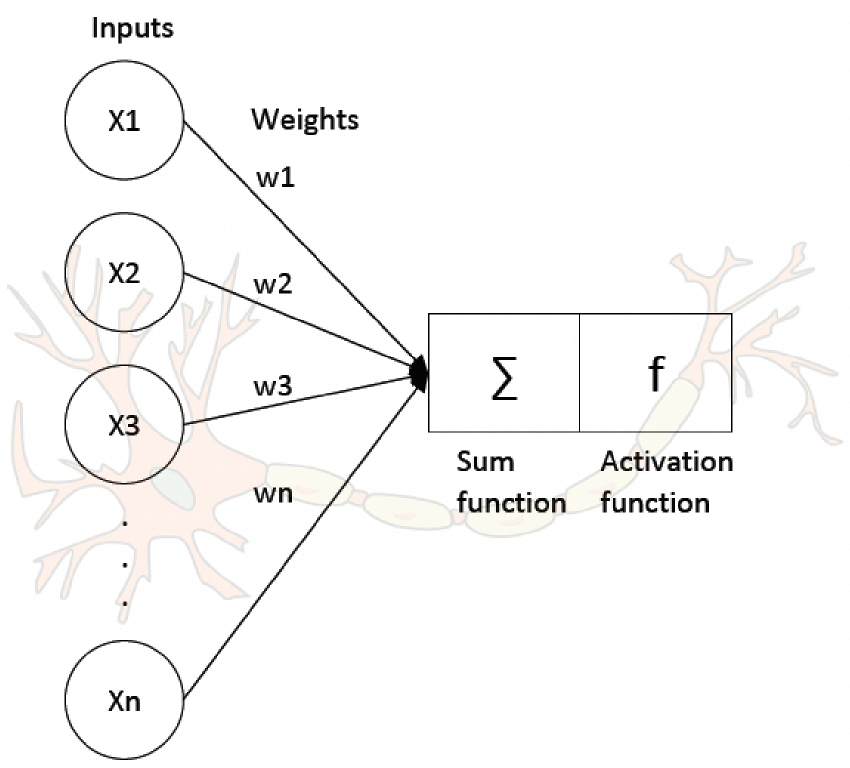
```python
# Create the linear regression model
model = nn.Linear(1, 1)
```

```python
# Loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.05)
```

```python
# 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)

# PyTorch uses float32 by default
# Numpy creates float64 by default
inputs = torch.from_numpy(X.astype(np.float32))
targets = torch.from_numpy(Y.astype(np.float32))
```

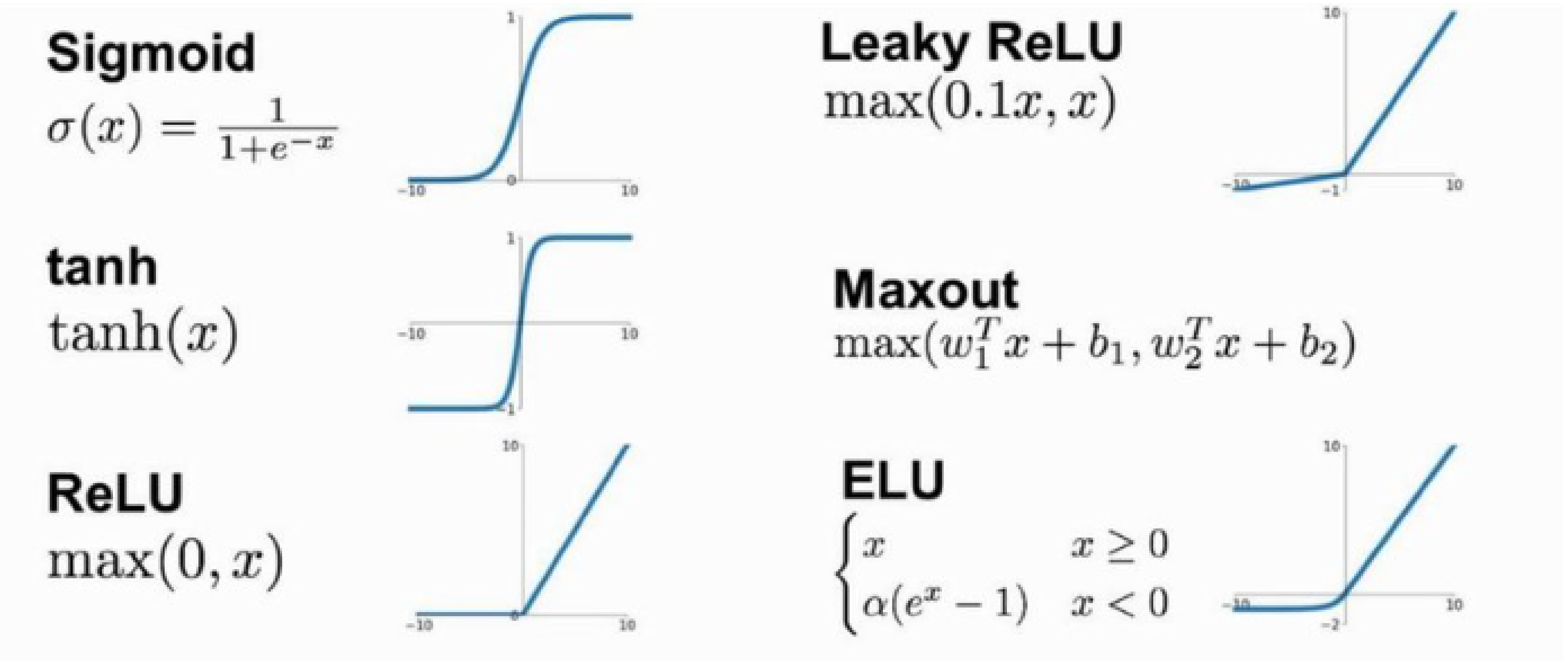


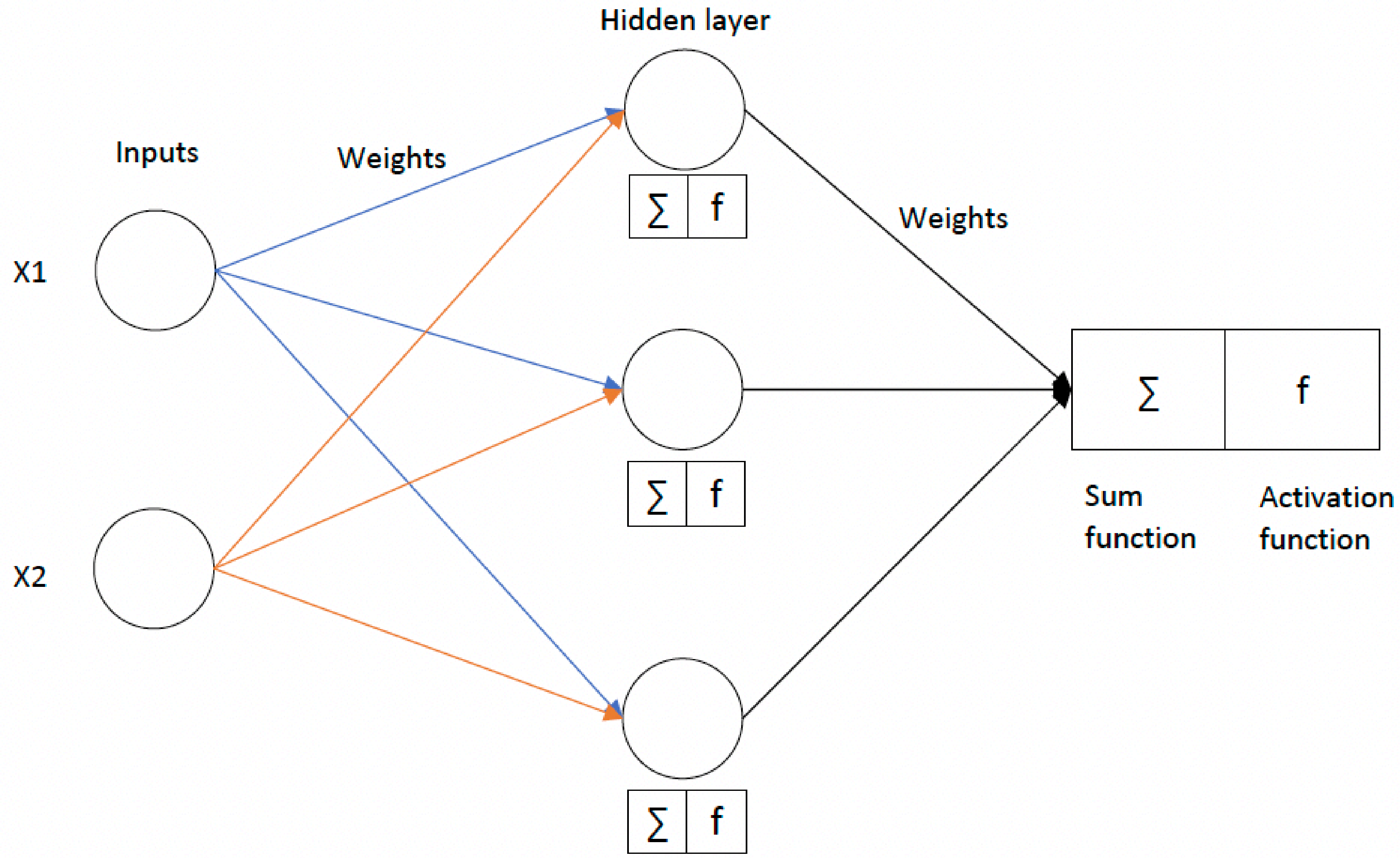

net = torch.nn.Linear(2,1);
2 inputs
1 neurons
1 output
i_1
i_2
w_1
w_2
o_1
b_1
1 bias
net = torch.nn.Linear(2,2);
2 inputs
2 neurons
2 output
i_1
i_2
w_{11}
w_{21}
w_{12}
w_{22}
o_1
o_2
b_1
b_2
2 biases
net = torch.nn.Linear(2,3);
2 inputs
3 neurons
3 output
i_1
i_2
w_{11}
w_{21}
w_{31}
w_{12}
w_{22}
w_{32}
o_1
o_2
o_3
b_1
b_2
b_3
3 biases
net = torch.nn.Linear(3,2);
3 inputs
2 neurons
2 output
i_1
i_2
i_3
w_{11}
w_{21}
w_{12}
w_{22}
w_{13}
w_{23}
o_1
o_2
b_1
b_2
2 biases



$$sum = \sum_{i=1}^{n} xi \ * wi$$

X1 * w1+ X2 * w2 + X3 * w3





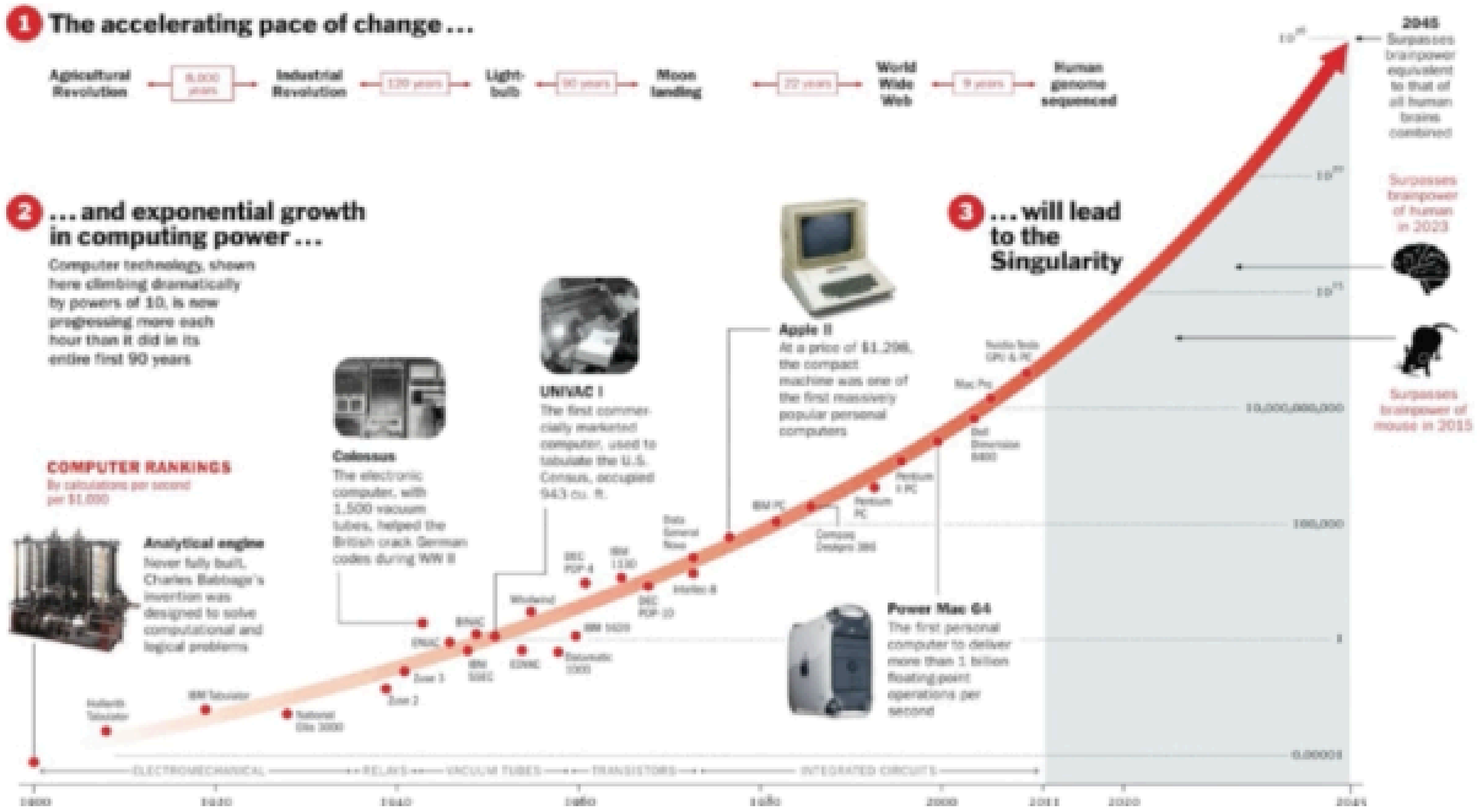
Home Work

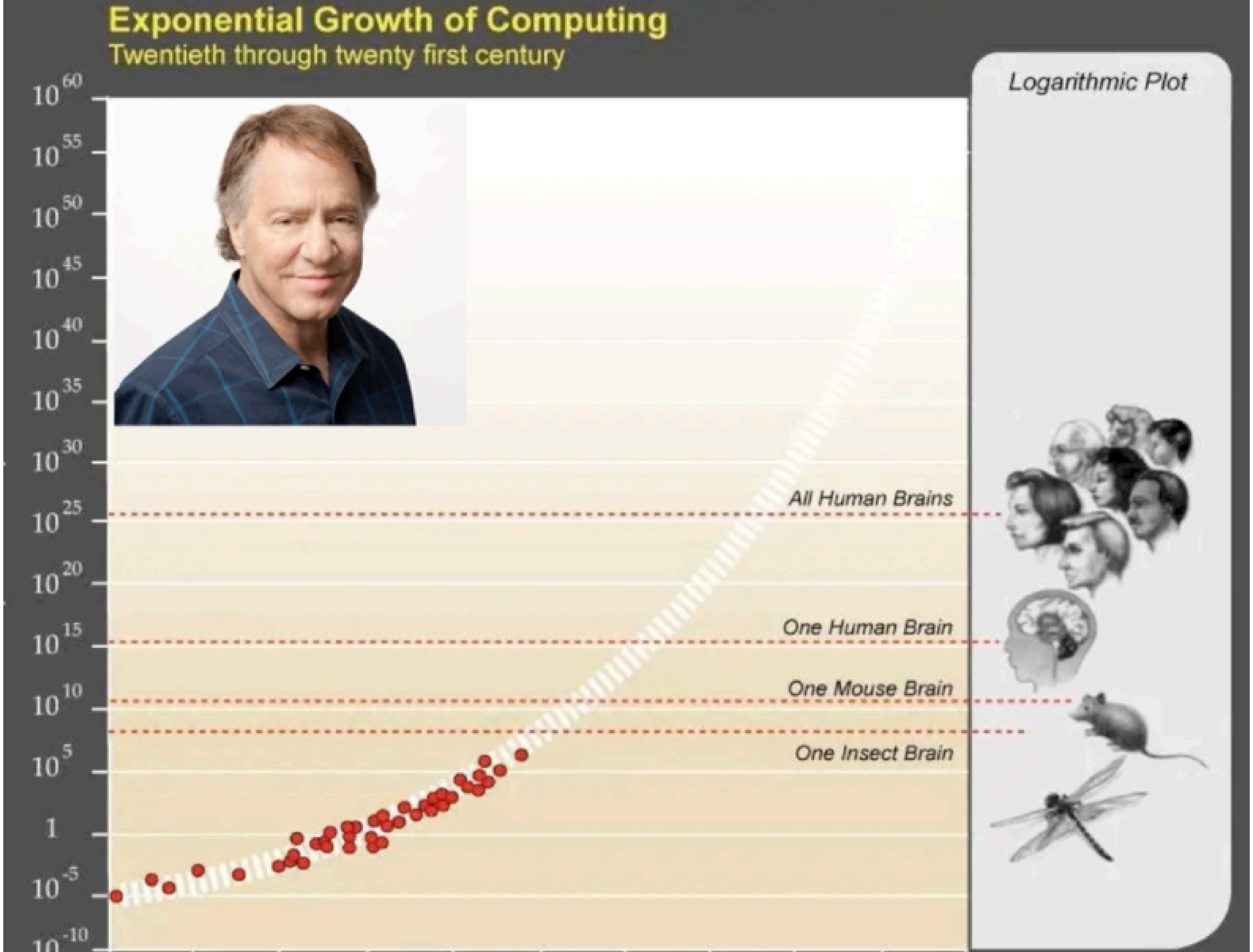
Week 1

HOME WORK 1

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
10^{60}
10^{55}
10^{50}
10^{45}
10^{40}
10^{35}
10^{30}
10^{25}
10^{20}
10^{15}
10^{10}
10^{5}
1
10^{-5}
10^{-10}
All Human Brains
One Human Brain
One Mouse Brain
One Insect Brain

HOME WORK 1

```python
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

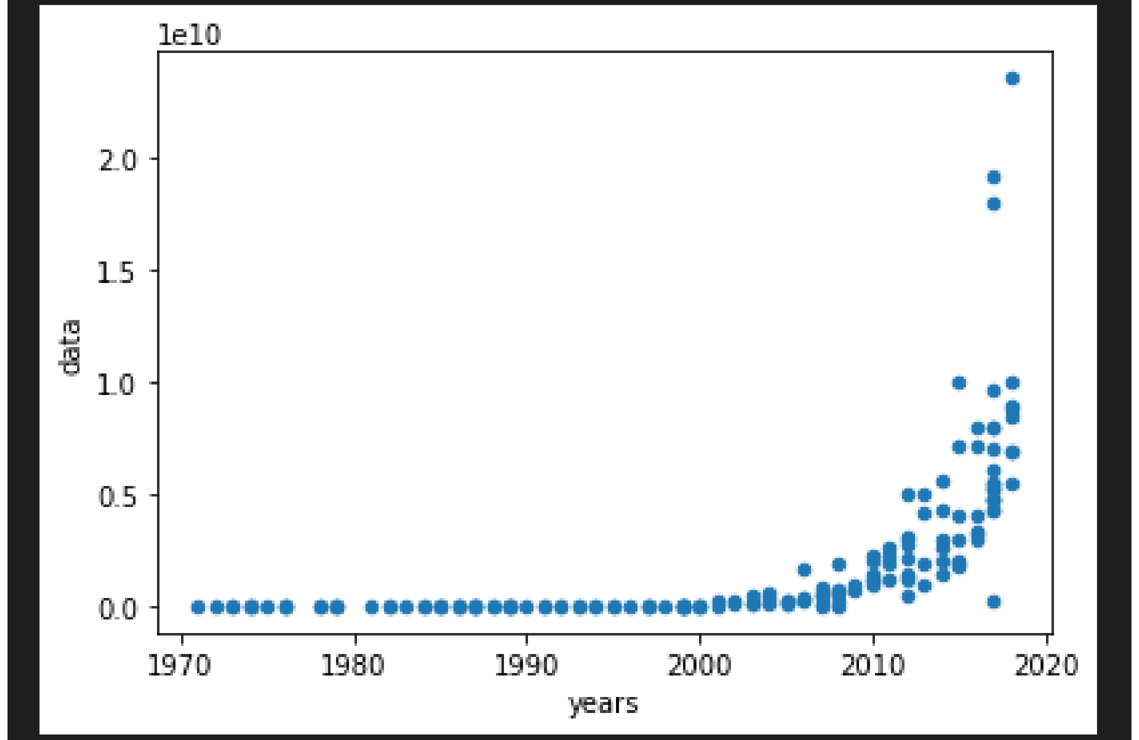
	years	data
0	1971	2300
1	1972	3500
2	1973	2500
3	1973	2500
4	1974	4100
...	...	...
157	2017	18000000000
158	2017	19200000000
159	2018	8876000000
160	2018	23600000000
161	2018	9000000000

162 rows × 2 columns

```python
df.plot(kind='scatter', x='years', y='data')
```

✓ 0.9s

<AxesSubplot:xlabel='years', ylabel='data'>



HOME WORK 2

$$y = A1\sin(f_1 t + \theta_1) + A2cos(f_2 t + \theta_2)$$

$$Parameters = A1, f_1, \theta_1, A2, f_2, \theta_2$$

```python
t = np.arange(0, 10, 0.005)

#random data on x-axis
A1  = 5*np.random.rand()
f1  = 10*np.random.rand()+5
b1  = 20*np.random.rand()

ceta1 = 2*np.random.rand()*np.pi

signal1 = A1 * np.sin(f1 * t + ceta1) + b1

#random data on x-axis
A2  = 5*np.random.rand()
f2  = 10*np.random.rand()+20
b2  = 20*np.random.rand()
ceta2 = 2*np.random.rand()*np.pi

signal2 = A2 * np.cos(f2 * t + ceta2) + b2

y = signal1 + signal2

print(f"A1 = {A1}, f1 = {f1} , ceta1 = {ceta1} b1= {b1}")
print(f"A2 = {A2}, f1 = {f2} , ceta2 = {ceta2} b2= {b2}")
```

✓ 0.0s

```
A1 = 4.757400431132537, f1 = 14.982025297569715 , ceta1 = 4.532944447976331 b1= 4.804226285370636
A2 = 1.448664560946444, f1 = 24.567685969748283 , ceta2 = 2.764982500149891 b2= 19.881588370782723
```