Deep Learning

Bài 1: Làm quen với AI và Tensorflow & Keras

Van-Khoa LE, Ph.D
CyberLab, 10/2022

(Ảnh: Internet)

CYBERLEARN
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

CYBERSOFT
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

CYBERLAB
ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

Giảng viên và Trợ giảng

Giảng viên: TS. Lê Văn Khoa
email: vankhoa21991@gmail.com

Facebook: https://www.facebook.com/tqkhai2705/
Email: tqkhai0527@gmail.com

Đề cương khoá học (phần cơ bản)

	Lí thuyết	Thực Hành
Tuần 1	**Làm quen với AI và TensorFlow & Keras** • Giới thiệu về deep learning, AI • Linear Regression • Gradient descent • Tensorflow cơ bản	• Các hàm cơ bản TensorFlow và Keras • Dự đoán giá nhà với linear regression trên TF-KR
Tuần 2	**Artificial Neural Network (ANN) và Deep Neural Network (DNN)** • Neuron, weight, activation function, softmax, argmax • Thuật toán back propagation • Hàm mất mát loss function • Evaluation Metrics • Các thuật toán optimization • Logistic Regression • Các vấn đề khi huấn luyện mô hình deep learning	Dataloader, optimizer, loss, metrics - Xây dựng ANN và DNN - Nhận dạng sản phẩm thời trang Fashion MNIST

Đề cương khoá học (phần cơ bản)

	Lí thuyết	Thực Hành
Tuần 3	**Mạng neuron tích chập (Convolutional Neural Network - CNN)** • Convolution • Padding và stride • Pooling • LeNet **Một số kỹ thuật cải tiến CNN:** • BatchNorm/LayerNorm • Dropout • Augmentation **Các kiến trúc CNN hiện đại:** • Alex-Net • GoogleNet • VGG • Dense-Net • Res-Net	Thực hiện lại các bài tập trước với CNN: - DEMO: Nhận dạng chữ số MNIST - Nhận dạng Fashion MNIST - Nhận dạng vật thể trong CIFAR100
Tuần 4	**Mạng neuron hồi quy (Recurrent Neural Network)** • Mô hình RNN • Thuật toán back probagation through time • Mô hình LSTM, GRU • GRU	- Thực hành trên time series data - Thực hành với dữ liệu chuỗi thời gian, dự đoán dữ liệu chứng khoán
Tuần 5	**Cơ chế Attention** • Mô hình seq2seq kết hợp attention • Transformer	Thực hành với NLP: - Machine translation - Sentiment classification

Đề cương khoá học (phần nâng cao)

	Li thuyet	Thực Hành
Tuan 6	**Dự án giữa kỳ: bài toán nhận dạng chữ viết tay - OCR** (Optical Character Recognition) Dùng CNN cho OCR Kết hợp CNN và LSTM cho OCR	Thực hành OCR với chữ viết tay mẫu tự Latin
Tuần 7	**Bài toán phân khúc ảnh (segmentation)** • U-Net • Các metric liên quan • Huấn luyện mô hình với Pytorch	- Bài toán segmentation cho ảnh thời trang (fashion images) hoặc ảnh y tế - Pedestrian segmentation
Tuần 8	**Bài toán Object Detection** • Các khái niệm căn bản (anchors, NMS) • Mô hình YOLO v1,2,3 • Mô hình SSD • Mô hình RetinaNet • Mô hình Faster-RCNN	YOLOv3, YOLOv4, YOLOv5 - Ứng dụng mạng Darknet cho YOLO - Nhận dạng các loài vật trong ảnh

Đề cương khoá học (phần nâng cao)

	Li thuyet	Thực Hành
Tuần 9	**Các mạng tạo sinh (Generative networks)** • AutoEncode (AE) • Variational AE (VAE) • Các mạng GANs • Diffusion Network	- Thực hành AE/VAE trên MNIST - Thực hành AE/VAE trên dữ liệu số - Thực hành GANs cho ảnh CIFAR
Tuần 10	**Xử lý ngôn ngữ tự nhiên (Natural Language Processing)** • Các khái niệm trong NLP • Word embeddings • Bert • ChatGPT	Thực hành Sentiment analysis: phân tích phản hồi khách hàng
Tuần 11	- Cách tổ chức và trình bày dự án. - MLOps - Project cuối khóa	

Thông tin khoá học

- Một tuần 2 buổi gồm một buổi lí thuyết online và một buổi bài tập offline

- Coding plaform: google colab
- Slack: deeplearningc-n931024
- Email: vankhoa21991@gmail.com

Yêu cầu trước khi tham gia khoá học

- Lập trình python
- Kiến thức căn bản về Machine Learning
- Đại số tuyến tính
- Xác suất thống kê

Yêu cầu khi tham gia khoá học

- Bật camera, chỉ tắt trong thời gian ngắn khi có việc gấp
- Đặt câu hỏi ngay khi có thắc mắc
- Tương tác

Kết quả sau khoá học

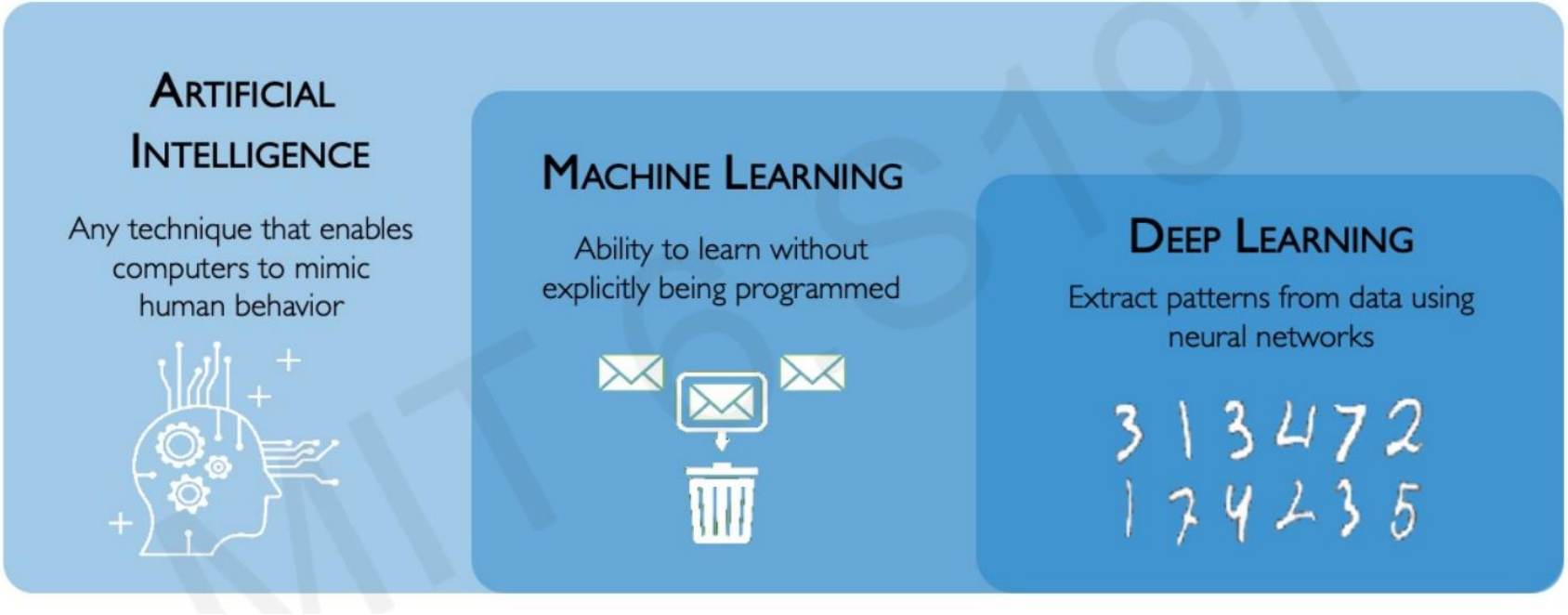
- Nắm được các kiến thức nền tảng của deep learning
- Các chuyên đề nâng cao ứng dụng deep learning
- Hiểu bản chất của các ứng dụng AI phổ biến hiện tại
- Lập trình huấn luyện mô hình bằng pytorch và tensorflow keras
- Cách trình bày dự án, vòng đời của một sản phẩm

Nội dung

1. Giới thiệu
2. Mô hình tuyến tính
3. Làm quen với tensorflow & keras
4. Bài tập & Thảo Luận

Phần 1: Giới thiệu

Deep learning là gì?



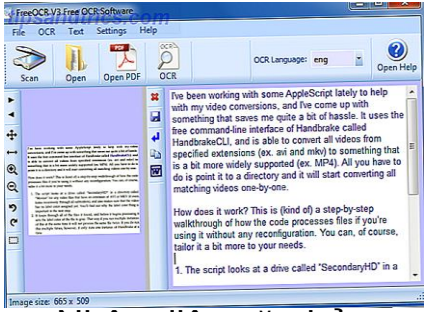
- Một nhánh trong học máy (machine learning)
- Mô hình xây dựng dựa trên dữ liệu, để thực hiện một nhiệm vụ cụ thể
- Đạt kết quả cao hơn các thuật toán học máy truyền thống
- Thường được sử dụng trong xử lý ảnh (CV), ngôn ngữ tự nhiên (NLP)

Ứng dụng của deep learning

Nhận diện giọng nói

Nhận diện khuôn mặt

Dịch ngôn ngữ

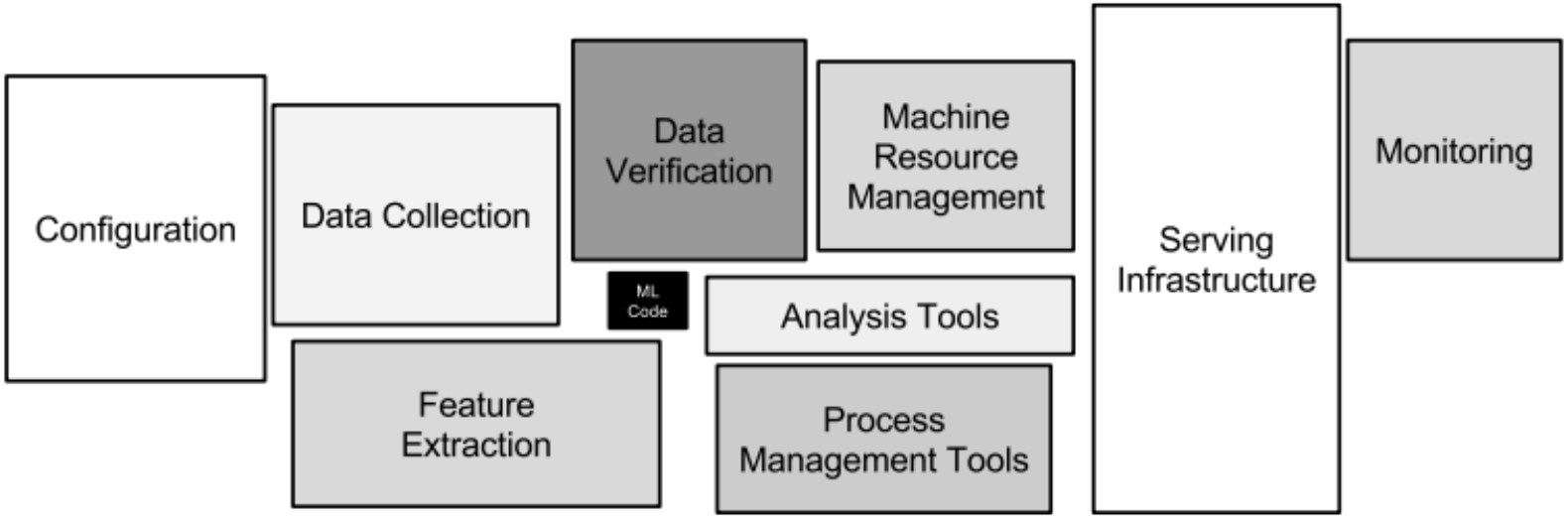


Nhận diện văn bản

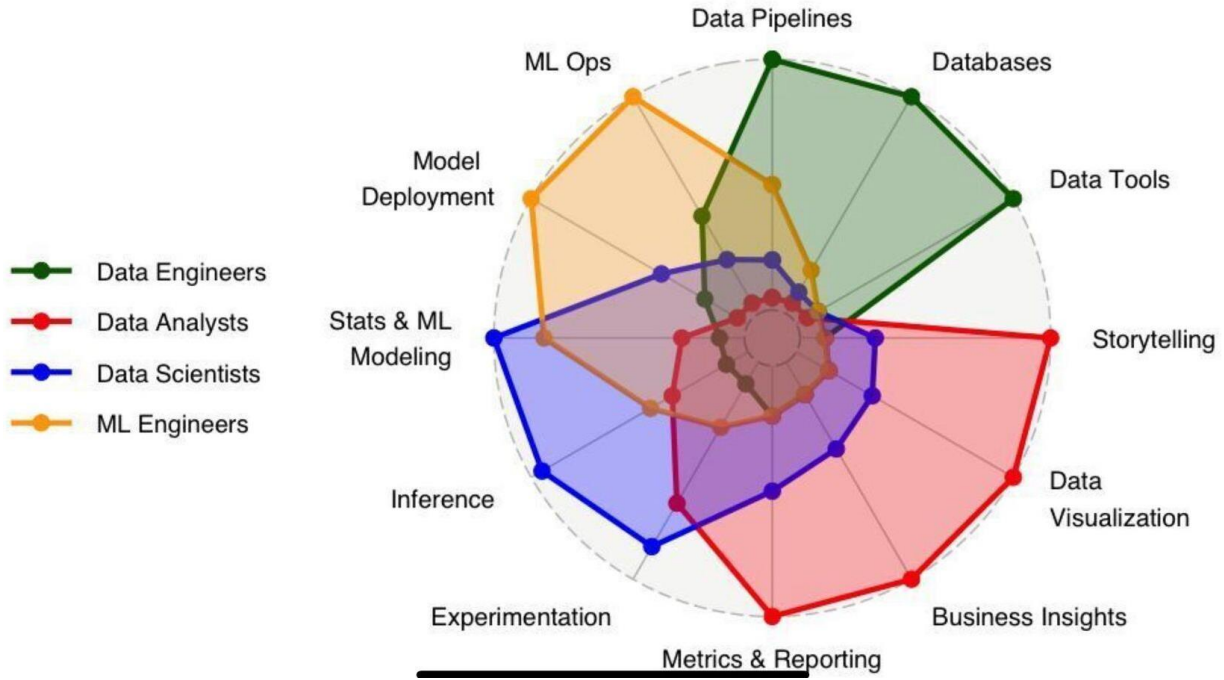
Xe tự lái

Y tế

Vai trò của deep learning trong thực tế



Skill set của các vị trí trong data team



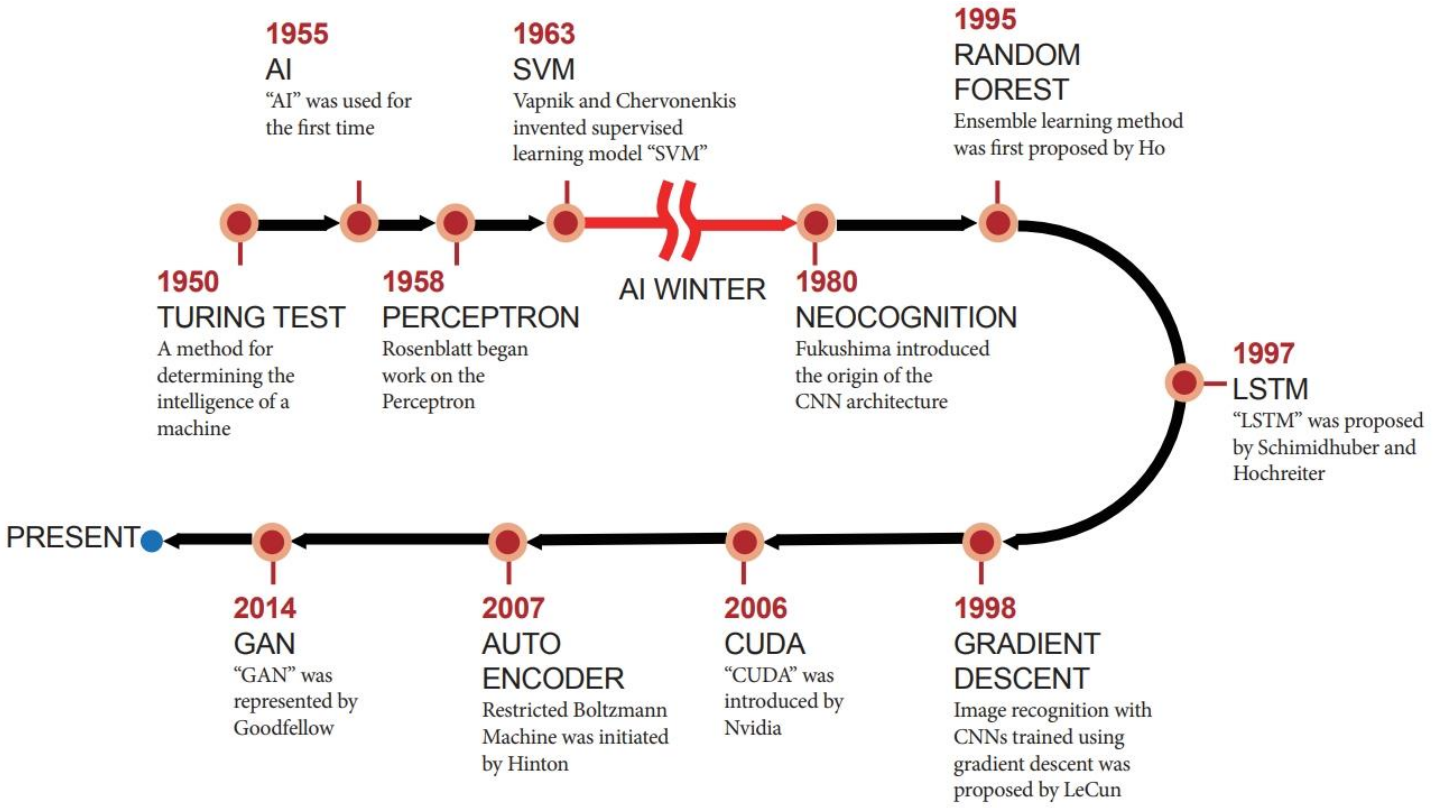
Khác biệt giữa DataScience và Software Engineering

Black box

People with no idea about AI saying it will take over the world:

My Neural Network:

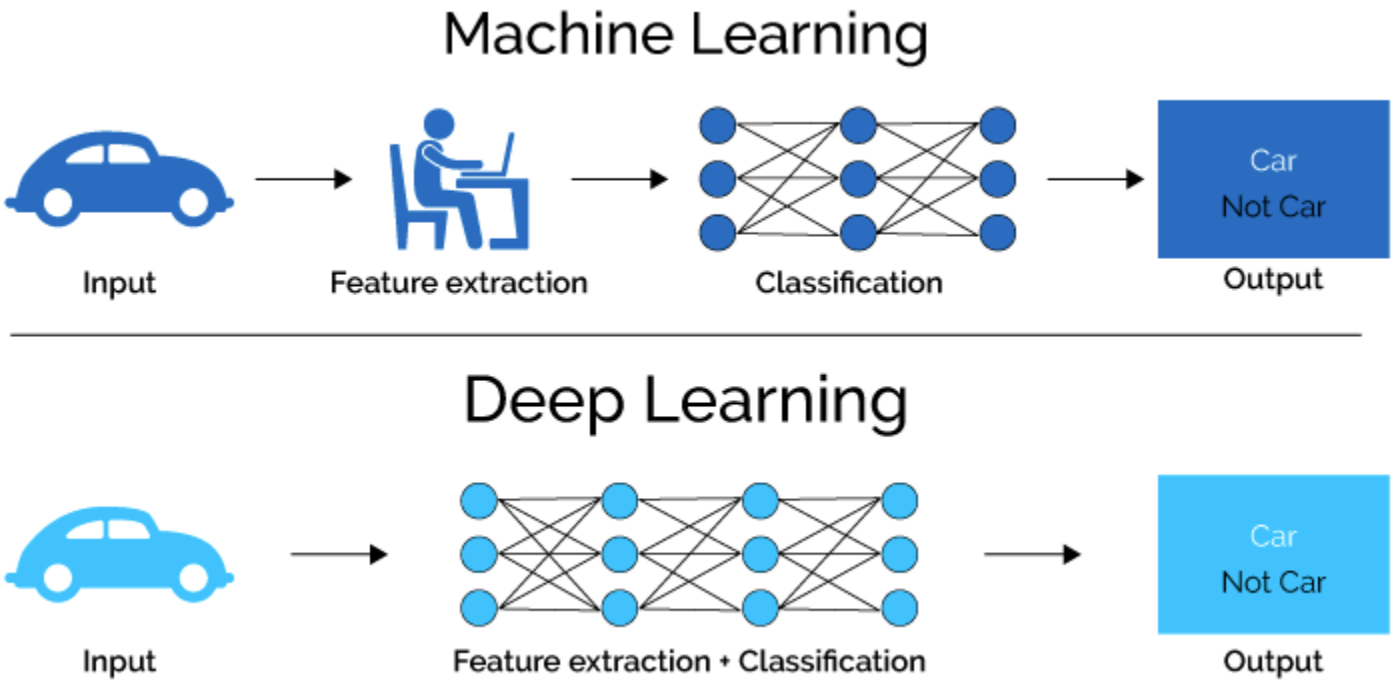
Lịch sử phát triển



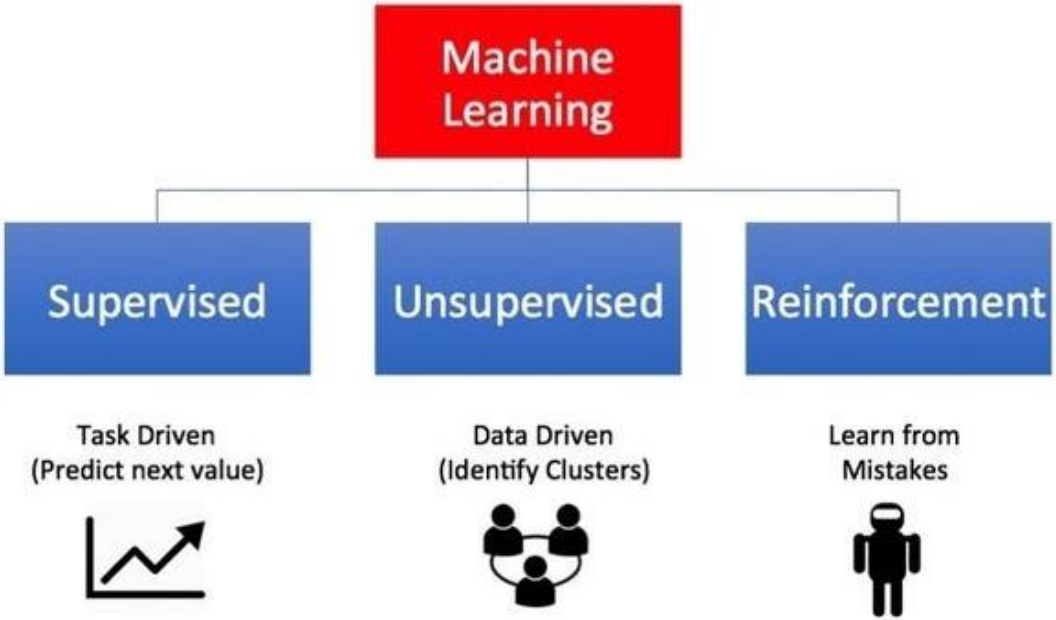
Lịch sử phát triển

Thập niên	Dữ liệu	Dung lượng	FLOPS	Thuật toán
1940				Multilayer perceptrons
1970	100 (Iris)	1 KB	100 KF (Intel 8080)	
1980	1 K (House prices in Boston)	100 KB	1 MF (Intel 80186)	
1990	10 K (optical character recognition)	10 MB	10 MF (Intel 80486)	CNN LSTM
2000	10 M (web pages)	100 MB	1 GF (Intel Core)	
2010	10 G (advertising)	1 GB	1 TF (Nvidia C2050)	Dropout Attention GAN Transformer
2020	1 T (social network)	100 GB	1 PF (Nvidia DGX-2)	

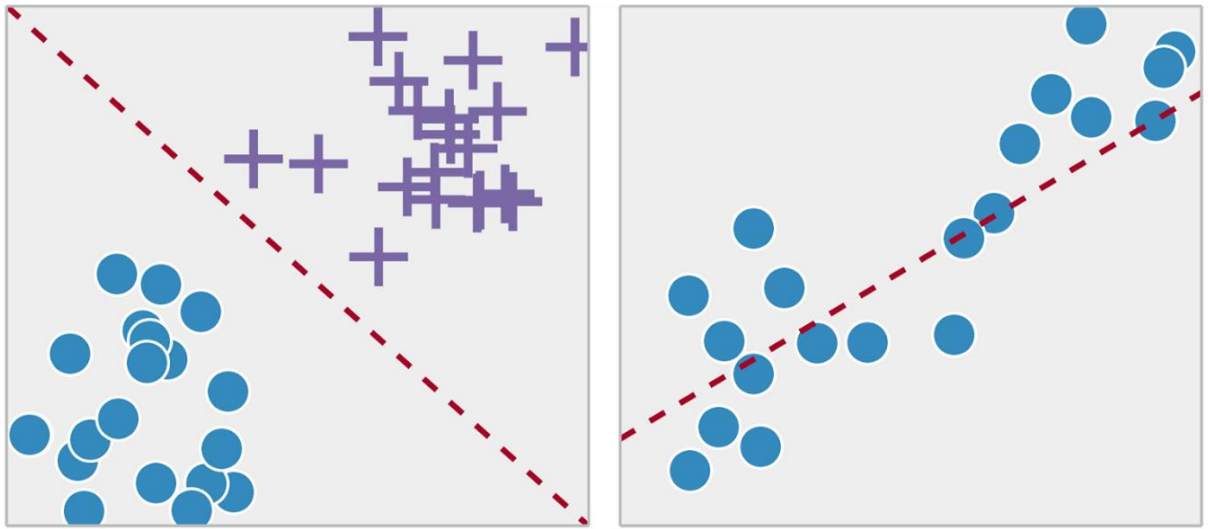
So sánh học máy và học sâu (machine learning vs deep learning)



Các loại vấn đề giải quyết bởi ML/DL

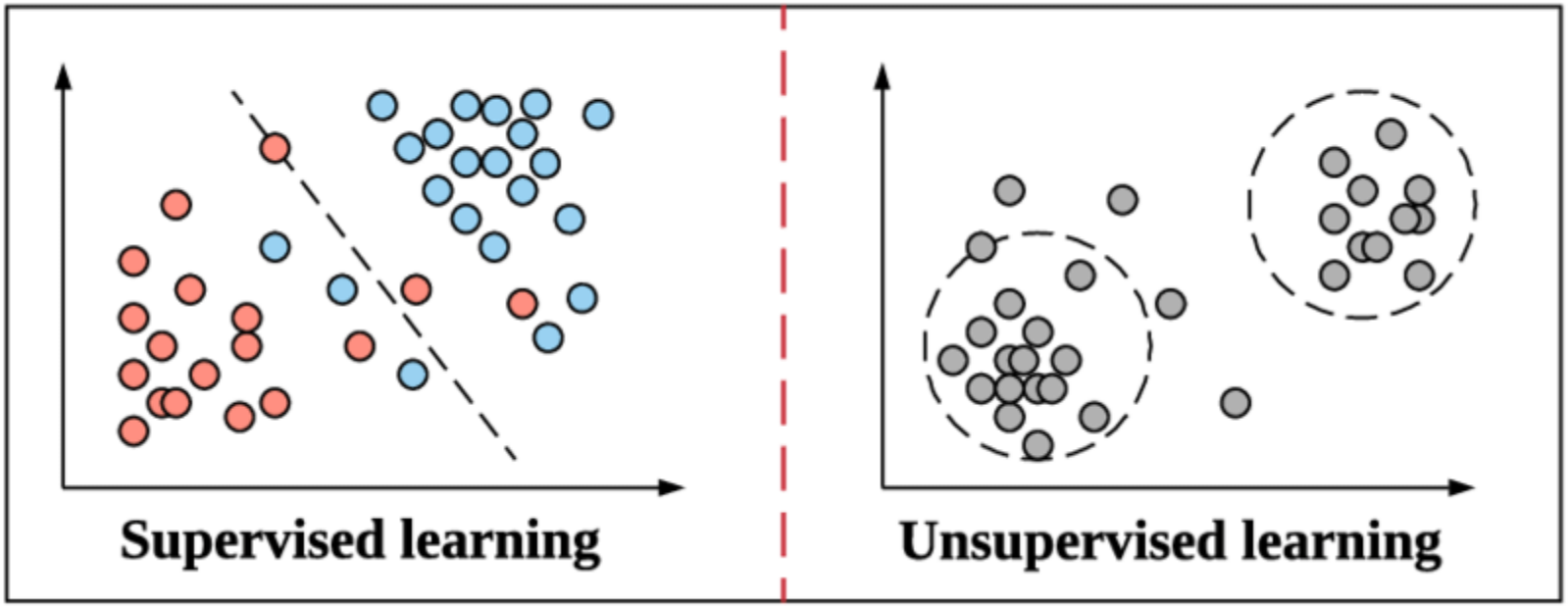


Supervised learning (học có giám sát)



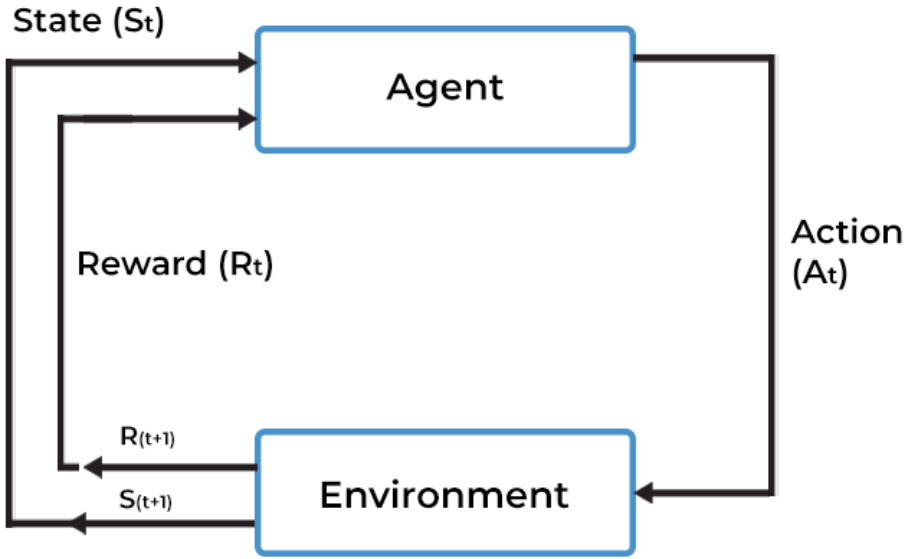
Phân loại và dự đoán

Unsupervised learning (học không giám sát)



Xây dựng mô hình dựa vào thông tin sẵn có trong dữ liệu

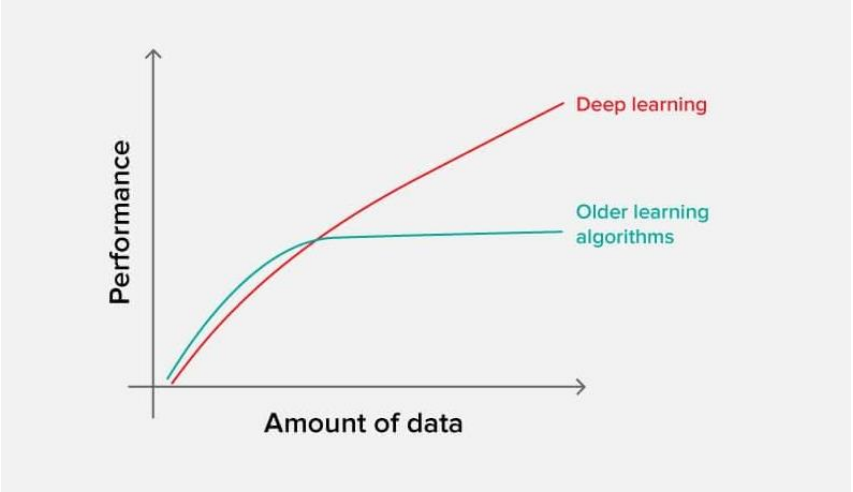
Reinforcement learning (học tăng cường)



Xây dựng mô hình dựa trên việc thử sai của một agent
thực hiện hành động vào môi trường xung quanh và nhận lại phản hồi từ môi trường

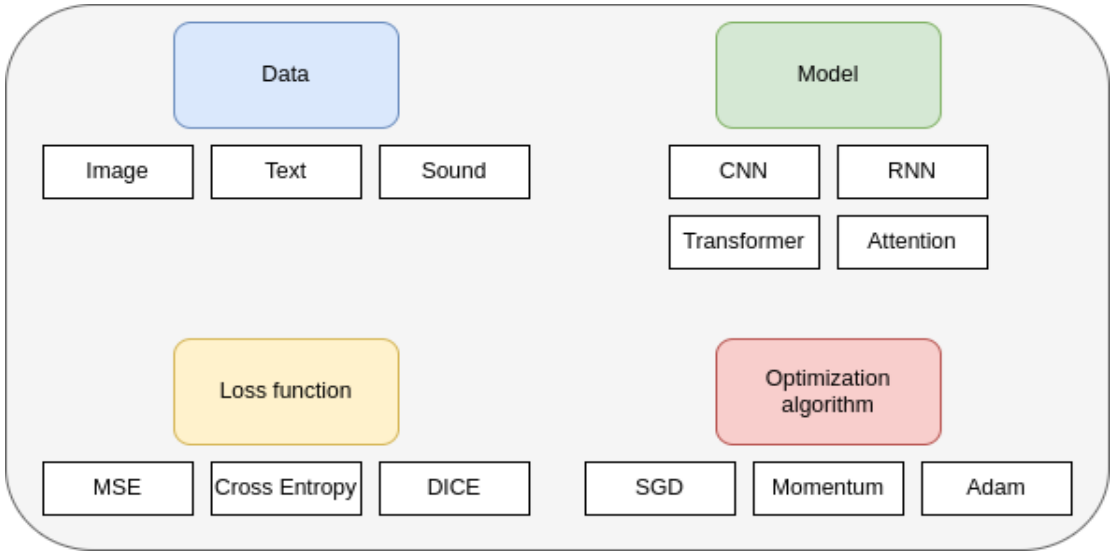
Khi nào sử dụng deep learning

- Khi có nhiều dữ liệu
- Dữ liệu không cấu trúc (unstructured) như văn bản, hình ảnh, âm thanh, …
- Có phần cứng thích hợp (GPU, TPU)

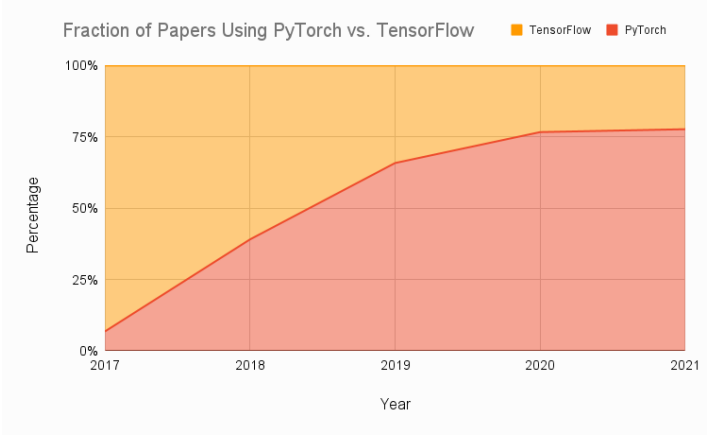


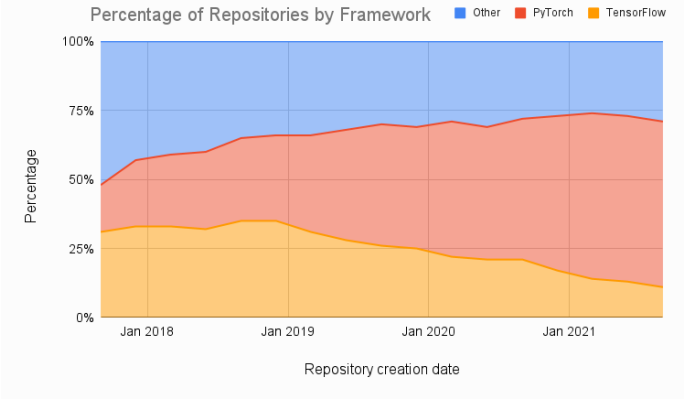
Thành phần chính của deep learning

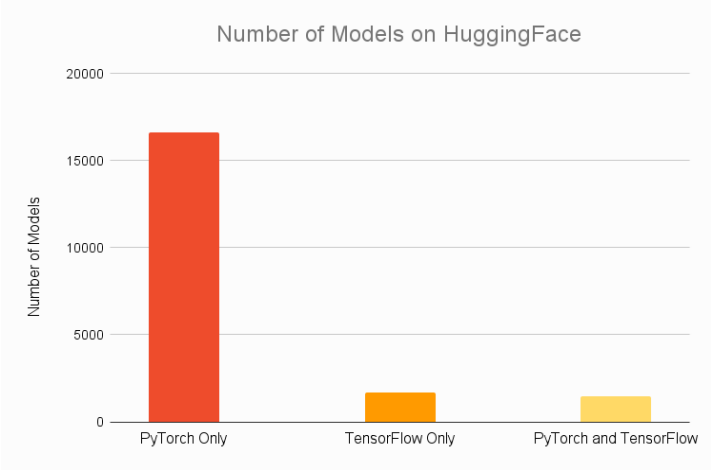
- Bốn thành phần chính của deep learning:
  - Dữ liệu
  - Mô hình
  - Hàm mất mát
  - Thuật toán tối ưu



Các framework



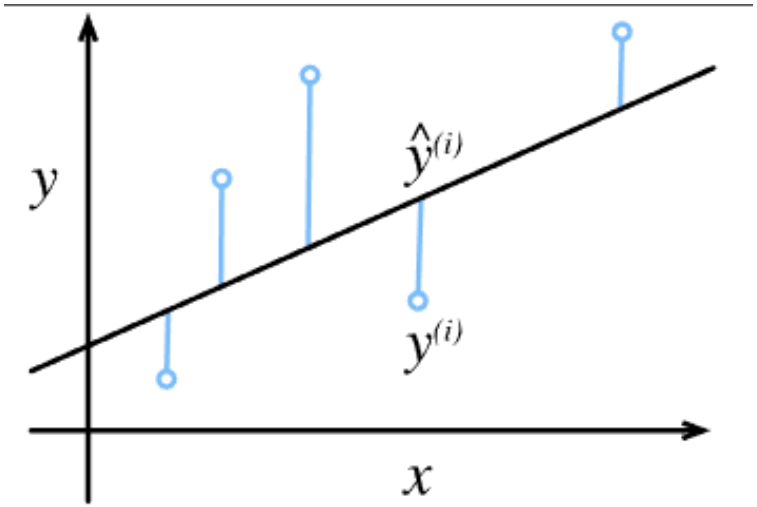




Phần 2: Mô hình tuyến tính

Mô hình tuyến tính (linear regression)

- Mô hình cơ bản dùng để so sánh quan hệ giữa 2 biến
  - Đầu vào: điểm dữ liệu đa chiều
  - Đầu ra: giá trị cần tính toán



- Mô hình (liên kết giữa đầu vào và đầu ra):

$$\hat{y} = w_1 x_1 + w_2 x_2 + \ ... \ + w_2 x_2 + w_0 * 1$$

- Viết dạng vector:

$$\hat{Y} = W^T X$$

Với $W = [w_0, w_1, w_2, w_3]^T$ và $X = [1, x_1, x_2, x_3]^T$

Xây dựng hàm mất mát

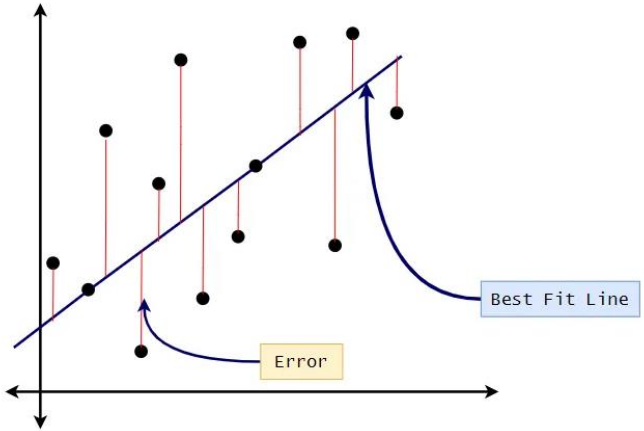
- Tiêu chí của đường thẳng thích hợp nhất là gì?

Khoảng cách giữa các điểm đến đường thẳng là nhỏ nhất

Xây dựng hàm mất mát:

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{N} (y_i - wx_i\ )^2$$

Với y là giá trị đầu ra thật còn $\hat{y}$ là giá trị đầu ra tính toán bởi mô hình



- Đường thẳng thích hợp nhất tương đương với L đạt giá trị min:

$$w^* = arg\min_{w} L(w)$$

Tính toán hệ số w

- Tìm cực tiểu hàm mất mát bằng cách giải phương trình đạo hàm hàm mất mát bằng 0

$$L = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2}\|Y - XW\|_2^2$$

- Đạo hàm hàm mất mát bằng 0:

$$\frac{\partial L(w)}{\partial w} = X^T(XW - Y) = 0$$

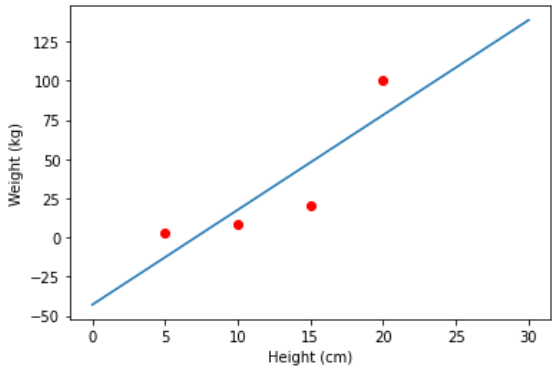
- Nghiệm W của phương trình:

$$W = (X^T X)^{-1} XY$$

Mô hình tuyến tính một chiều

Dữ liệu

Đầu vào	Đầu ra
Diện tích (x)	Gía trị (y)
5	3
10	8
15	20
20	100




```
# diện tích (m2)
X = np.array([[5, 10, 15, 20]]).T
# giá (tỷ VND)
y = np.array([[ 3, 8 , 20, 100]]).T

# Xây dựng Xbar
one = np.ones((X.shape[0], 1))
Xbar = np.concatenate((one, X), axis = 1) # thêm w0 vào X

# tính toán w dựa trên công thức
A = np.dot(Xbar.T, Xbar)
b = np.dot(Xbar.T, y)
w = np.dot(np.linalg.pinv(A), b)
print('A = ', A)
print('b = ', b)
print('w = ', w)

# tính toán đường thẳng dựa trên w
w_0 = w[0][0]
w_1 = w[1][0]
x0 = np.linspace(0, 30, 2)
y0 = w_0 + w_1*x0

# Vẽ dữ liệu và đường thẳng tìm được
plt.plot(X.T, y.T, 'ro')       # data
plt.plot(x0, y0)               # the fitting line
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

[9] ✓ 0.1s

```
A =  [[  4.  50.]
 [ 50. 750.]]
b =  [[ 131.]
 [2395.]]
w =  [[-43.  ]
 [  6.06]]
```

Đa số trường hợp là không tìm được do:

- Phương trình đạo hàm phức tạp
- Điểm dữ liệu nhiều chiều
- Nhiều điểm dữ liệu

Gradient descent

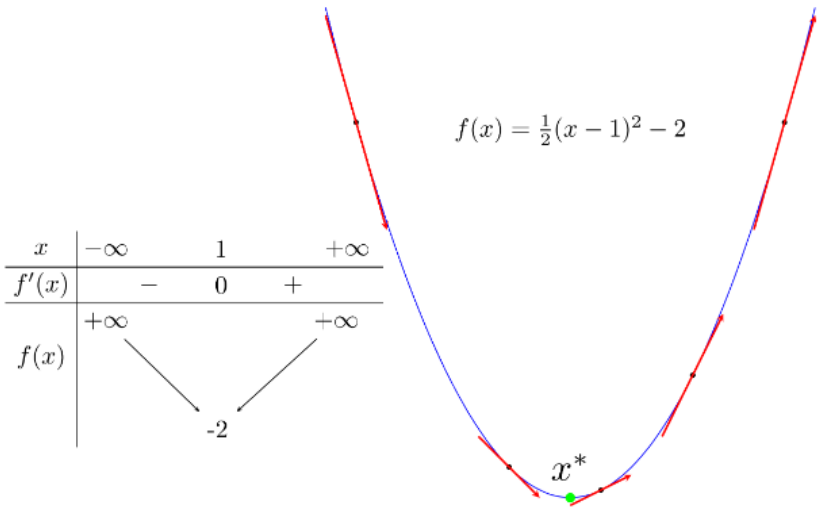
- Tìm cực tiểu bằng cách khởi động tại một điểm và di chuyển về hướng cực tiểu
- Gỉa sử ta có 1 hàm số

$$y = f(x) = \frac{1}{2}(x-1)^2 - 2$$

có điểm cực tiểu global minimum tại $f'(x) = 0$, tương ứng với $x^* = 1$

- Với ví dụ trên thì :
  - $f'(x) < 0$ nếu $x < x^*$(x ở bên trái cực tiểu)
  - $f'(x) > 0$ nếu $x > x^*$(x ở bên phải cực tiểu)
- Trong hàm mất mát, biến là w

→ Phải di chuyển ngược dấu với đạo hàm để tìm w cho hàm L đạt cực tiểu



Gradient descent

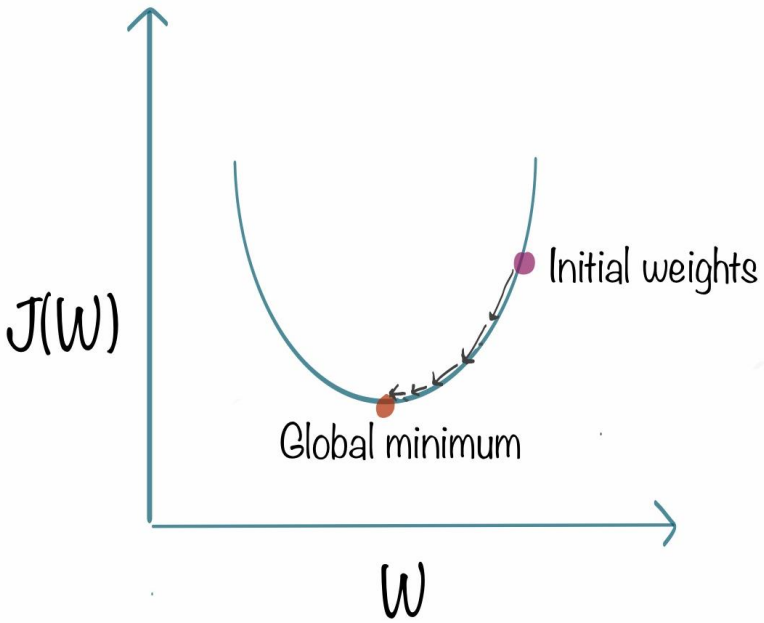
- Gradient tại một điểm là đạo hàm của hàm số tại điểm đó

- Cách tìm cực tiểu theo gradient descent:
  - Khởi tạo w_0 tại điểm bất kỳ
  - Tính gradient $f'(w_0)$=J(w_0)
  - Dịch chuyển w_0 một lượng ngược dấu với gradient:

$$w_1 = w_0 - \eta f'(w_0)$$

Với η là tốc độ học (learning rate)

  - Lập lại quá trình cho đến khi w hội tụ



Cách huấn luyện mô hình với gradient descent

Bước 1: Xác định hàm mất mát (ràng buộc giữa đầu ra tính toán và đầu ra thật (groundtruth))

Bước 2: Khởi động mô hình bằng cách chọn ngẫu nhiên w_0, b_0

Bước 3: Tính toán đầu ra bằng w, b hiện tại

Bước 4: Tính toán hàm mất mát giữa đầu ra hiện tại và đầu ra thật

Bước 5: Sử dụng hàm mất mát để cập nhật w,b theo công thức

$$(w, b) \leftarrow (w, b) - \frac{\eta}{n} \sum_{i \epsilon N} \partial_{(w,b)} l^i(w, b)$$

Quay lại bước 3

Các hàm mất mát phổ biến cho bài toán dự đoán

- Hàm bậc hai: (Mean square error MSE, L2 loss)
  - Tại điểm dữ liệu i:

$$l^{(i)}(w,b) = \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

  - Trung bình giá trị mất mát:

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

- Hàm bậc một – Mean Absolute Error, L1 loss :
  - Tại điểm dữ liệu i:

$$l^{(i)}(w,b) = \left|\hat{y}^{(i)} - y^{(i)}\right|$$

  - Trung bình giá trị mất mát:

$$MSE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$$

Ví dụ mô hình tuyến tính nhiều chiều

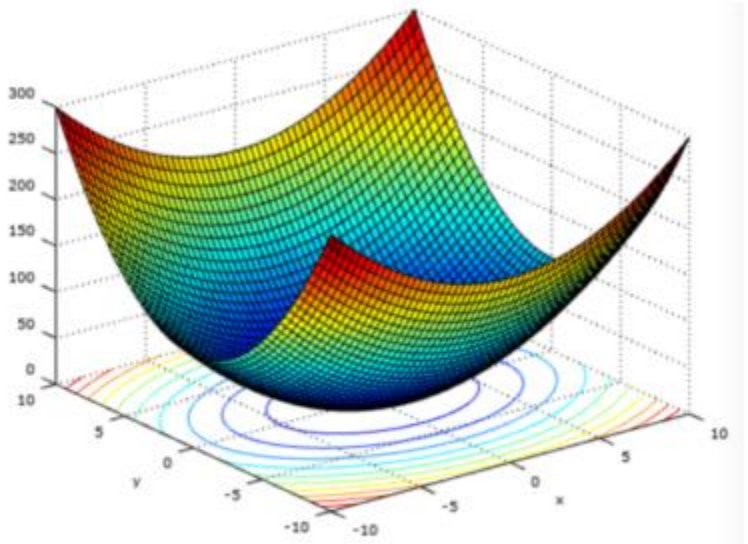
- Mô hình tính giá bất động sản:

Đầu vào		Đầu ra
Diện tích (area)	Số năm xây dựng (yr)	Gía trị (y)
5	7	3
10	9	8
15	8	20
20	3	100

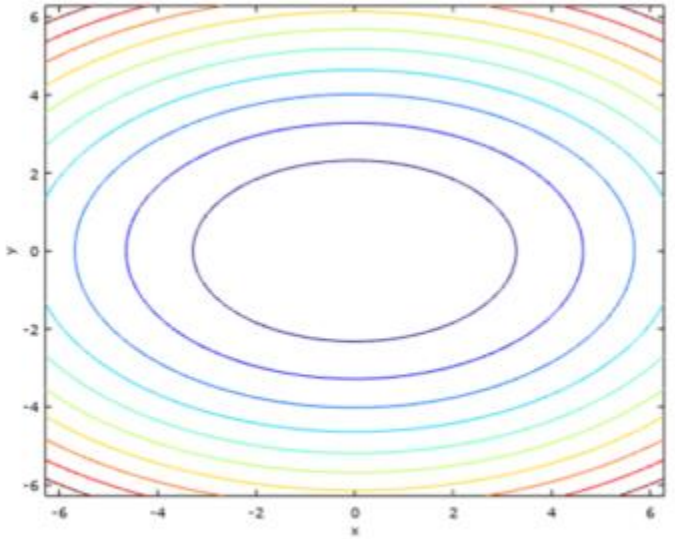
- Mô hình tuyến tính:

$$\hat{y} = w_{area}x_{area} + w_{yr}x_{yr} + w_0$$

Gradient descent hàm nhiều biến



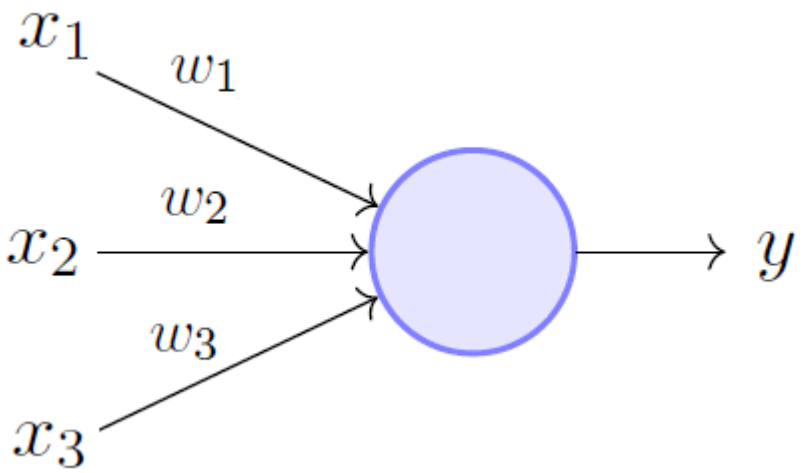
f(x, y) = x*x + 2y*y



Contours of f(x, y)

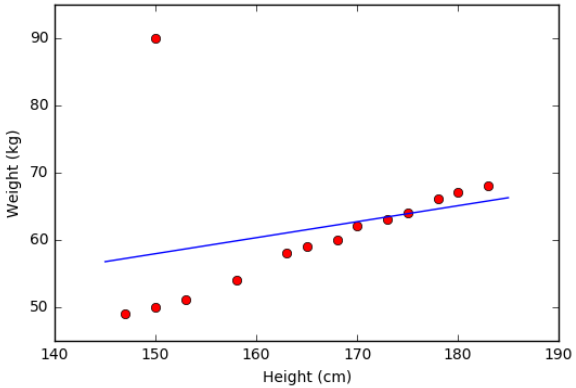
Từ mô hình tuyến tính đến mạng neuron

- Mô hình tuyến tính là một mạng neuron đơn giản 1 lớp (perceptron)

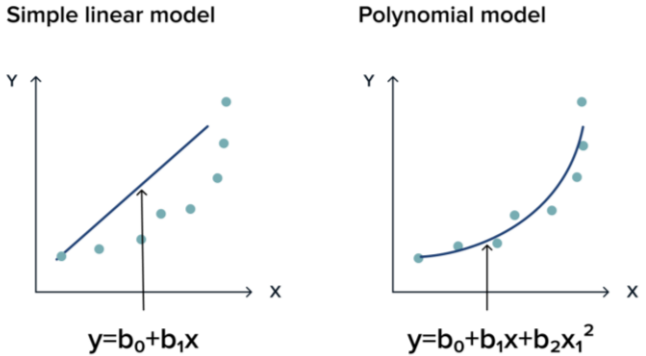


Hạn chế của linear regression

- Nhạy cảm với nhiễu



- Không biểu diễn được mô hình phi tuyến tính



Phần 3: Làm quen với tensorflow và keras

Tensorflow và keras

- Tensorflow: 

  - Thư viện mã nguồn mở phục vụ cho Machine Learning
  - Phát triển bởi google từ năm 2011, nhưng phiên bản chính thức được phát triển từ năm 2017
  - Cho phép xây dựng mô hình trên nhiều phần cứng (CPU, GPU, TPU)
  - Một trong hai thư viện deep learning được sử dụng nhiều nhất

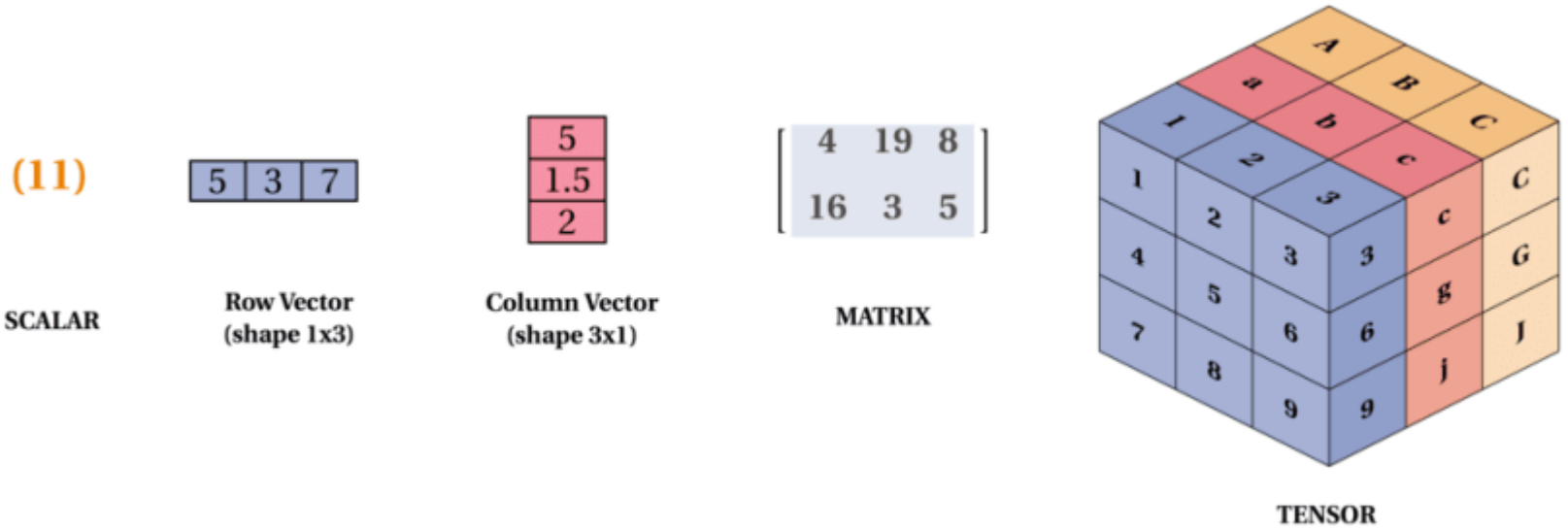
- Keras 

  - Thư viện mã nguồn mở, một phần của tensorflow
  - Xây dựng mô hình deep neural network nhanh chóng

Tensorflow

Tensor là gì

- Cấu trúc dữ liệu trong tensorflow, khi sử dụng tensorflow thì dữ liệu phải được chuyển về dạng tensor
- Tensor có thể chứa float, int và những dạng khác như số phức, chuỗi

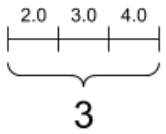


Code tạo tensor

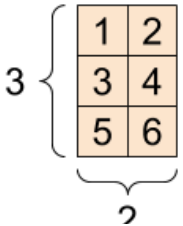
A scalar, shape: [ ]

4

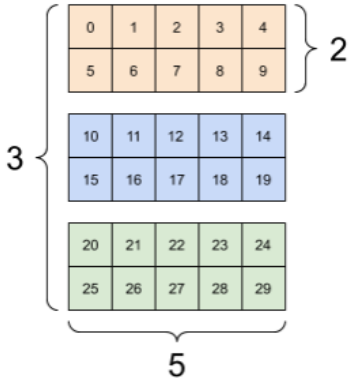
A vector, shape: [3]

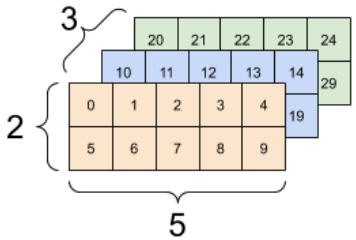


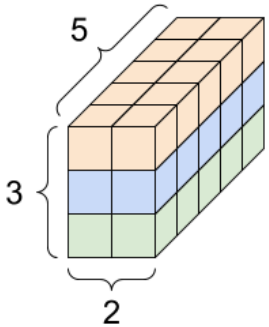
A matrix, shape: [3, 2]



A 3-axis tensor, shape: [3, 2, 5]





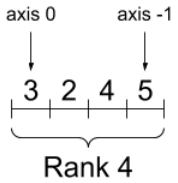


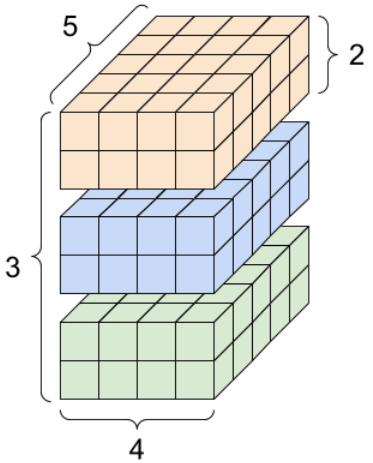
Thuộc tính của tensor

Rank:

- Scalar: Khi Tensor có rank bằng 0
- Vector: Vector là một Tensor rank 1. .
- Matrix: Đây là một Tensor rank 2 hay mảng hai chiều theo khái niệm của Python
- N-Tensor: Khi rank của Tensor tăng lên lớn hơn 2, chúng được gọi chung là N-Tensor.

```
rank_4_tensor = tf.zeros([3, 2, 4, 5])
```





Shape (kích thước của tensor), ví dụ:

- Tensor = [[[1,1,1],[178,62,74]]] sẽ có Shape = (1,2,3)
- Tensor = [[1,1,1],[178,62,74]] sẽ có Shape = (2,3)

Type (kiểu dữ liệu của tensor):

- Ví dụ int, float

Size: số phần tử của tensor

Khởi tạo tensor

```
[2]
# Tạo một tensor có rank = 0
rank_0_tensor = tf.constant(4)
print(rank_0_tensor)
```
Python

```
tf.Tensor(4, shape=(), dtype=int32)
```

```
[3]
# Tạo một tensor có rank = 1
rank_1_tensor = tf.constant([2.0, 3.0, 4.0])
print(rank_1_tensor)
```
Python

```
tf.Tensor([2. 3. 4.], shape=(3,), dtype=float32)
```

```
[4]
# Tạo một tensor có rank = 2
rank_2_tensor = tf.constant([[1, 2],
                             [3, 4],
                             [5, 6]], dtype=tf.float16)
print(rank_2_tensor)
```
Python

```
tf.Tensor(
[[1. 2.]
 [3. 4.]
 [5. 6.]], shape=(3, 2), dtype=float16)
```

```
[5]
rank_4_tensor = tf.zeros([3, 2, 4, 5])

print("Type of every element:", rank_4_tensor.dtype)
print("Number of axes:", rank_4_tensor.ndim)
print("Shape of tensor:", rank_4_tensor.shape)
print("Elements along axis 0 of tensor:", rank_4_tensor.shape[0])
print("Elements along the last axis of tensor:", rank_4_tensor.shape[-1])
print("Total number of elements (3*2*4*5): ", tf.size(rank_4_tensor).numpy())
```
Python

```
Type of every element: <dtype: 'float32'>
Number of axes: 4
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
Total number of elements (3*2*4*5):  120
```

Index tensor

```
## Index and slicing
rank_1_tensor = tf.constant([0, 1, 1, 2, 3, 5, 8, 13, 21, 34])
print(rank_1_tensor.numpy())
```
[6] Python

```
[ 0  1  1  2  3  5  8 13 21 34]
```

```
print("First:", rank_1_tensor[0].numpy())
print("Second:", rank_1_tensor[1].numpy())
print("Last:", rank_1_tensor[-1].numpy())
```
[7] Python

```
First: 0
Second: 1
Last: 34
```

```
print("Everything:", rank_1_tensor[:].numpy())
print("Before 4:", rank_1_tensor[:4].numpy())
print("From 4 to the end:", rank_1_tensor[4:].numpy())
print("From 2, before 7:", rank_1_tensor[2:7].numpy())
print("Every other item:", rank_1_tensor[::2].numpy())
print("Reversed:", rank_1_tensor[::-1].numpy())
```
[8] Python

```
Everything: [ 0  1  1  2  3  5  8 13 21 34]
Before 4: [0 1 1 2]
From 4 to the end: [ 3  5  8 13 21 34]
From 2, before 7: [1 2 3 5 8]
Every other item: [ 0  1  3  8 21]
Reversed: [34 21 13  8  5  3  2  1  1  0]
```

```
# Multi-axis indexing
# Pull out a single value from a 2-rank tensor
print(rank_2_tensor[1, 1].numpy())
```
[9] Python

```
4.0
```

```
# Get row and column tensors
print("Second row:", rank_2_tensor[1, :].numpy())
print("Second column:", rank_2_tensor[:, 1].numpy())
print("Last row:", rank_2_tensor[-1, :].numpy())
print("First item in last column:", rank_2_tensor[0, -1].numpy())
print("Skip the first row:")
print(rank_2_tensor[1:, :].numpy(), "\n")
```
[10] Python

```
Second row: [3. 4.]
Second column: [2. 4. 6.]
Last row: [5. 6.]
First item in last column: 2.0
Skip the first row:
[[3. 4.]
 [5. 6.]]
```

```
rank_3_tensor = tf.constant([
  [[0, 1, 2, 3, 4],
   [5, 6, 7, 8, 9]],
  [[10, 11, 12, 13, 14],
   [15, 16, 17, 18, 19]],
  [[20, 21, 22, 23, 24],
   [25, 26, 27, 28, 29]],])

print(rank_3_tensor)
```
[11] Python

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
```

Thay đổi kích thước tensor

```
[12]
x = tf.constant([[1], [2], [3]])
print(x.shape)
```
Python

```
(3, 1)
```

```
[13]
print(rank_3_tensor)
print(tf.reshape(rank_3_tensor, [-1]))
print(tf.reshape(rank_3_tensor, [3*2, 5]), "\n")
print(tf.reshape(rank_3_tensor, [3, -1]))
```
Python

```
tf.Tensor(
[[[ 0  1  2  3  4]
  [ 5  6  7  8  9]]

 [[10 11 12 13 14]
  [15 16 17 18 19]]

 [[20 21 22 23 24]
  [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32)
tf.Tensor(
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29], shape=(30,), dtype=int32)
tf.Tensor(
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]], shape=(6, 5), dtype=int32)

tf.Tensor(
[[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]], shape=(3, 10), dtype=int32)
```

Tính toán với tensor

```
[14]
# Cộng trừ nhân chia
a = tf.constant(2, dtype=tf.float32)
b = tf.constant(3, dtype=tf.float32)
c = tf.constant(5, dtype=tf.float32)

add = tf.add(a, b)
sub = tf.subtract(a, b)
mul = tf.multiply(a, b)
div = tf.divide(a, b)

print("add =", add)
print("sub =", sub)
print("mul =", mul)
print("div =", div)
                                                    Python
```

```
add = tf.Tensor(5.0, shape=(), dtype=float32)
sub = tf.Tensor(-1.0, shape=(), dtype=float32)
mul = tf.Tensor(6.0, shape=(), dtype=float32)
div = tf.Tensor(0.6666667, shape=(), dtype=float32)
```

```
[15]
# Tính tổng và trung bình
mean = tf.reduce_mean([a, b, c])
sum = tf.reduce_sum([a, b, c])
max = tf.reduce_max([a, b, c])
# Access tensors value.
print("mean =", mean)
print("sum =", sum)
print("max =", max)
                                                    Python
```

```
mean = tf.Tensor(3.3333333, shape=(), dtype=float32)
sum = tf.Tensor(10.0, shape=(), dtype=float32)
max = tf.Tensor(5.0, shape=(), dtype=float32)
```

```
d = tf.constant([[4.0, 5.0], [10.0, 1.0]])

# Tìm index của phần tử lớn nhất
print(tf.math.argmax(d))
# Tính softmax
print(tf.nn.softmax(d))
[16]                                                Python
```

```
tf.Tensor([1 0], shape=(2,), dtype=int64)
tf.Tensor(
[[2.6894143e-01 7.3105860e-01]
 [9.9987662e-01 1.2339458e-04]], shape=(2, 2), dtype=float32)
```

```
# Chuyển list thành tensor
print(tf.convert_to_tensor([1,2,3]))
print(tf.convert_to_tensor(np.array([[1,2,3],[4,5,6]])))
[17]                                                Python
```

```
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
```

```
# Nhân 2 ma trận
matrix1 = tf.constant([[1., 2.], [3., 4.]])
matrix2 = tf.constant([[5., 6.], [7., 8.]])

product = tf.matmul(matrix1, matrix2)
print("product =", product)
[18]                                                Python
```

```
product = tf.Tensor(
[[19. 22.]
 [43. 50.]], shape=(2, 2), dtype=float32)
```

```
x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
z = tf.constant(2)
f = x*x*y + y + z
print(x)
print(y)
print(z)
print(f)
[19]                                                Python
```

```
<tf.Variable 'x:0' shape=() dtype=int32, numpy=3>
<tf.Variable 'y:0' shape=() dtype=int32, numpy=4>
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(42, shape=(), dtype=int32)
```

Biến (variable)

```
my_tensor = tf.constant([[1.0, 2.0], [3.0, 4.0]])
my_variable = tf.Variable(my_tensor)

# Variables can be all kinds of types, just like tensors
bool_variable = tf.Variable([False, False, False, True])
complex_variable = tf.Variable([5 + 4j, 6 + 1j])

print("Shape: ", my_variable.shape)
print("DType: ", my_variable.dtype)
print("As NumPy: ", my_variable.numpy())
```

[56] Python

```
Shape:  (2, 2)
DType:  <dtype: 'float32'>
As NumPy:  [[1. 2.]
 [3. 4.]]
```

```
with tf.device('CPU:0'):
  a = tf.Variable([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
  b = tf.Variable([[1.0, 2.0, 3.0]])

with tf.device('GPU:0'):
  # Element-wise multiply
  k = a * b

print(k)
```

[57] Python

```
tf.Tensor(
[[ 1.  4.  9.]
 [ 4. 10. 18.]], shape=(2, 3), dtype=float32)
```

Gradient

```
[24]
x = tf.Variable(3.0)

with tf.GradientTape() as tape:
    y = x**2
```
Python

```
[25]
# dy = 2x * dx
dy_dx = tape.gradient(y, x)
dy_dx.numpy()
```
Python

```
...
6.0
```

```
[51]
w = tf.Variable(tf.ones([3,2]), name='w')
b = tf.Variable(tf.zeros(2, dtype=tf.float32), name='b')
x = [[1., 2., 3.]]
print(x, w, b)

with tf.GradientTape(persistent=True) as tape:
    y = x @ w + b
    loss = tf.reduce_mean(y**2)
    print(loss)
```
Python

```
...
[[1.0, 2.0, 3.0]] <tf.Variable 'w:0' shape=(3, 2) dtype=float32, numpy=
array([[1., 1.],
       [1., 1.],
       [1., 1.]], dtype=float32)> <tf.Variable 'b:0' shape=(2,) dtype=float32, numpy=array([0.,
tf.Tensor(36.0, shape=(), dtype=float32)
```

Gradient

```
fixed_kernel = tf.ones([3,2])

layer = tf.keras.layers.Dense(2, use_bias=False,
                              kernel_initializer=tf.keras.initializers.Constant(fixed_kernel))
x = tf.constant([[1., 2., 3.]])

with tf.GradientTape() as tape:
  # Forward pass
  y = layer(x)
  loss = tf.reduce_mean(y**2)

print(loss)
# Calculate gradients with respect to every trainable variable
grad = tape.gradient(loss, layer.trainable_variables)
print(grad)
for var, g in zip(layer.trainable_variables, grad):
  print(f'{var.name}, shape: {g.shape}')
```

[54] Python

```
tf.Tensor(36.0, shape=(), dtype=float32)
[<tf.Tensor: shape=(3, 2), dtype=float32, numpy=
array([[ 6.,  6.],
       [12., 12.],
       [18., 18.]], dtype=float32)>]
dense_7/kernel:0, shape: (3, 2)
```

Xây dựng mô hình

```
class LinearRegression(tf.Module):
    def __init__(self, in_features, out_features, name=None):
        super().__init__(name=name)
        self.w = tf.Variable(
        tf.random.normal([in_features, out_features]), name='w')
        self.b = tf.Variable(tf.zeros([out_features]), name='b')
    def __call__(self, x):
        y = tf.matmul(x, self.w) + self.b
        return y
```

Huấn luyện mô hình với tensorflow

- Ví dụ linear regression với tensorflow grad

Keras

Xử lý dữ liệu

Mở tệp dữ liệu nhà
Đầu vào là 13 cột dữ liệu đầu tiên
Đầu ra là cột cuối cùng

```
[5]  # load dataset
     dataframe = pd.read_csv("housing.csv", delim_whitespace=True, header=None)
     dataset = dataframe.values
     # split into input (X) and output (Y) variables
     X = dataset[:,0:13]
     Y = dataset[:,13]

     dataframe.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Xây dựng mô hình

- Khởi tạo mô hình với class Sequential của tensorflow
- Thêm lớp đầu tiên với 13 input ứng với 13 đặc trưng
- Thêm lớp đầu ra với 1 neuron

- Xây dựng mô hình, sử dụng hàm mất mát MSE, và thuật toán tối ưu adam

```
[6]  from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense

     model = Sequential()
     model.add(Dense(13, input_shape=(13,), kernel_initializer='normal', activation='relu'))
     model.add(Dense(1, kernel_initializer='normal'))
     # Compile model
     model.compile(loss='mean_squared_error', optimizer='adam')
```

Huấn luyện mô hình

Sử dụng method fit của class model. Đầu ra là log của tập huấn luyện và tập validation (nếu có).
Các trọng số của mô hình được cập nhật sau mỗi vòng (epoch)
Hàm mất mát của tập huấn luyện giảm xuống sau mỗi epoch

```
[11] model.fit(X,Y, epochs=10)

Epoch 1/10
16/16 [==============================] - 0s 2ms/step - loss: 545.6085
Epoch 2/10
16/16 [==============================] - 0s 2ms/step - loss: 376.4868
Epoch 3/10
16/16 [==============================] - 0s 2ms/step - loss: 247.1058
Epoch 4/10
16/16 [==============================] - 0s 2ms/step - loss: 181.5879
Epoch 5/10
16/16 [==============================] - 0s 2ms/step - loss: 163.4968
Epoch 6/10
16/16 [==============================] - 0s 3ms/step - loss: 148.9221
Epoch 7/10
16/16 [==============================] - 0s 2ms/step - loss: 134.7903
Epoch 8/10
16/16 [==============================] - 0s 3ms/step - loss: 121.6886
Epoch 9/10
16/16 [==============================] - 0s 2ms/step - loss: 109.9621
Epoch 10/10
16/16 [==============================] - 0s 2ms/step - loss: 98.8927
<keras.callbacks.History at 0x7fd25652aa10>
```

Đánh giá mô hình

Sử dụng method evaluate của class model. Đầu ra là giá trị mất mát:

```
[31] test_results = model.evaluate(X,Y, verbose=0)
     test_results

     30.73972511291504
```

Predict

- Sử dụng method predict, đầu ra là các dự đoán tương ứng của đầu vào:

```
[33] y = model.predict(X)
     print(y)
```

```
[18.139883 ]
[19.196932 ]
[25.372005 ]
[20.627333 ]
[19.404673 ]
[20.808952 ]
[23.354195 ]
[23.153221 ]
[23.142855 ]
[22.556904 ]
[19.731813 ]
[19.817871 ]
[22.163544 ]
```

Linear regression với tensorflow keras

- Code theory

Phần 4: Bài tập và thảo luận

Tóm tắt

- Deep learning là một phần trong Machine Learning
- Deep learning là thuật toán được huấn luyện dự trên dữ liệu
- Pytorch and tensorflow là hai thư viện phổ biến nhất cho deep learning
- Mô hình tuyến tính là dạng đơn giản nhất của mạng neuron
- Xây dựng hàm mất mát dựa trên đầu ra tính toán và đầu ra thật
- Huấn luyện mô hình bằng gradient descent

Bài tập

- Dự đoán giá nhà bằng linear regression

THANK YOU!