

DEEP LEARNING

Giới thiệu

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

Nội dung

- Trí tuệ nhân tạo (AI) là gì?
- Máy học là gì?
- Các hạn chế của máy học
- Deep Learning to the Rescue
- Học sâu là gì?
- Các ứng dụng học sâu

1

AI là gì?

- Xét ví dụ sau



2

AI là gì?

- Xét ví dụ sau



3

AI là gì?

- AI là khả năng của máy tính bắt chước **hành vi thông minh của con người**
- AI có được bằng cách nghiên cứu
 - cách suy nghĩ của bộ não người
 - cách con người học, ra quyết định khi giải quyết 1 vấn đề nào đó
- Kết quả nghiên cứu này được sử dụng để tạo nên các phần mềm và hệ thống thông minh

4

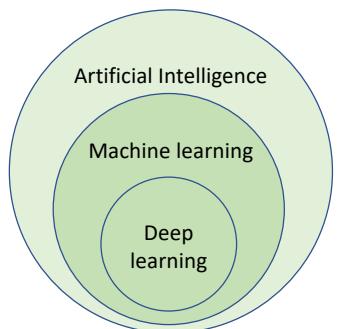
Các ứng dụng AI



Image Recognition

5

Các lĩnh vực con của AI



- Machine learning là một lĩnh vực con của AI
- Deep learning là lĩnh vực con của machine learning
 - Sử dụng mạng nơ ron để mô phỏng quá trình ra quyết định giống như người

6

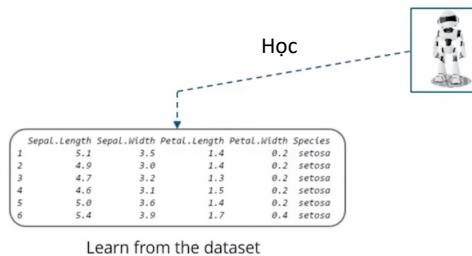
Machine learning

- Machine learning là một loại AI cung cấp cho máy tính khả năng học mà không cần phải lập trình một cách tưởng minh
- Ví dụ: học cách phân loại hoa IRIS dựa vào tập dữ liệu huấn luyện

7

Machine learning

- Ví dụ: học cách phân loại hoa IRIS dựa vào tập dữ liệu huấn luyện

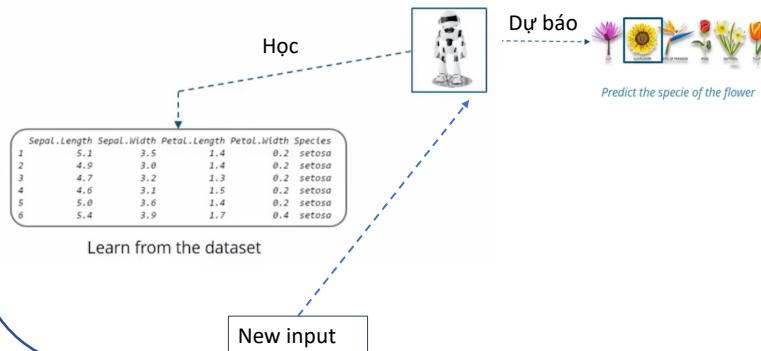


Learn from the dataset

8

Machine learning

- Ví dụ: học cách phân loại hoa IRIS dựa vào tập dữ liệu huấn luyện



Dự báo

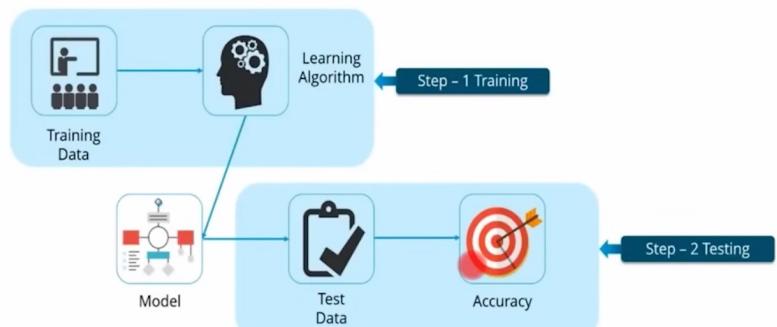


Predict the species of the flower

9

Học có giám sát (supervised learning)

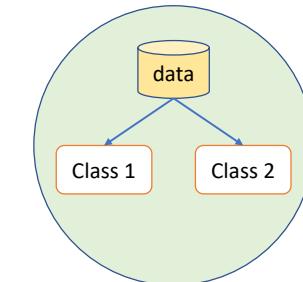
- Từ tập dữ liệu huấn luyện (X, y) tìm/học một mô hình f sao cho $f(x) = y$



10

Học không giám sát (unsupervised learning)

- Học/Huấn luyện một mô hình từ X mà không dùng nhãn y của dữ liệu
- Mô hình kết quả có thể dùng để gom cụm (cluster) dữ liệu dựa trên các thuộc tính thống kê của dữ liệu

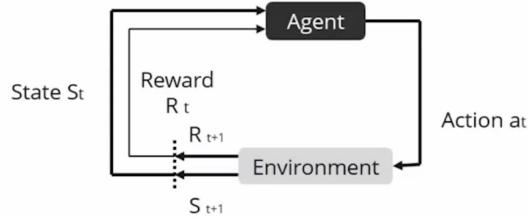


- Các phần tử trong cùng lớp có độ tương đồng cao
- Các phần tử ở hai lớp khác nhau có độ tương đồng thấp

11

Học tăng cường (reinforcement learning – RL)

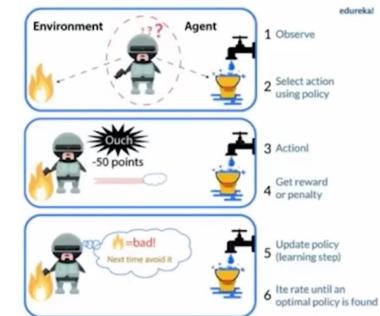
- RL là học bằng cách tương tác với môi trường thông qua cơ chế thưởng phạt



12

Học tăng cường (reinforcement learning – RL)

- Một tác tử RL (agent) học từ kết quả của các hành động chứ không học một tường minh từng ví dụ như học có giám sát.
- Tác tử lựa chọn hành động dựa trên các kinh nghiệm của quá khứ (exploitation) và các lựa chọn mới (exploration)



13

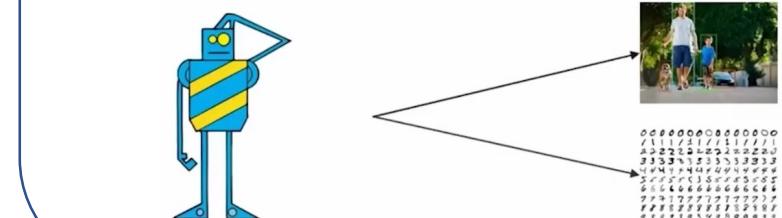
Các hạn chế của ML

- Không hiệu quả khi dữ liệu có số chiều lớn
 - Số phần tử lớn, đầu ra lớn
- Khó khăn khi giải quyết các bài toán khó như xử lý ngôn ngữ tự nhiên, nhận dạng ảnh, ...

14

Các hạn chế của ML

- Một thách thức lớn của các mô hình ML truyền thống là trích đặc trưng của dữ liệu (feature extraction)
 - Đối với các bài toán phức tạp nhận dạng đối tượng hay nhận dạng chữ viết tay, đây thật sự là một thách thức rất lớn.



15

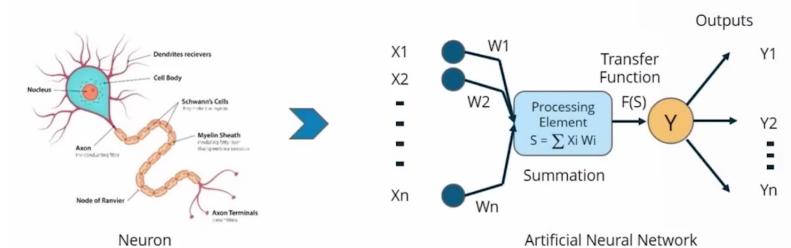
Deep learning to the rescue

- Mô hình học sâu tự nó có khả năng tập trung vào các đặc trưng tốt, mà cần rất ít từ lập trình viên
- Các mô hình này phần nào giải quyết được bài toán số chiều lớn
- Ý tưởng chính của học sâu là xây dựng các thuật toán học bắt chước bộ não con người.

16

Deep learning to the rescue

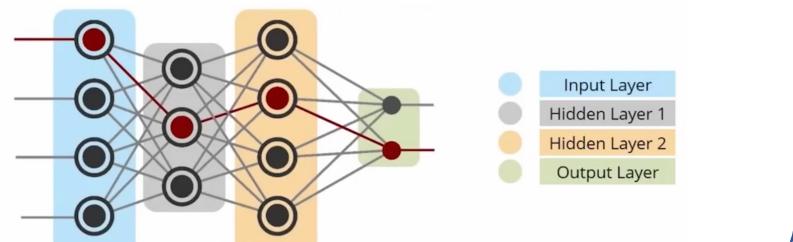
- Học sâu được cài đặt bằng mạng nơ ron
- Ý tưởng phía sau của các mạng nơ ron là các nơ ron sinh học



17

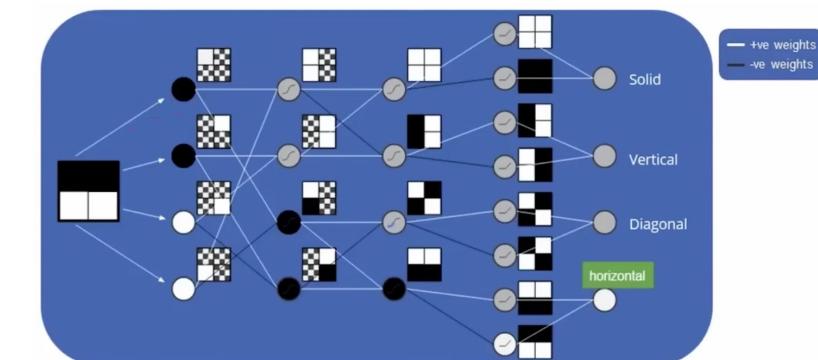
Học sâu là gì?

- Mô tập hợp các kỹ thuật học dựa trên thống kê dùng để học sự phân cấp của đặc trưng (feature hierarchies), thường là dựa trên các mạng nơ ron



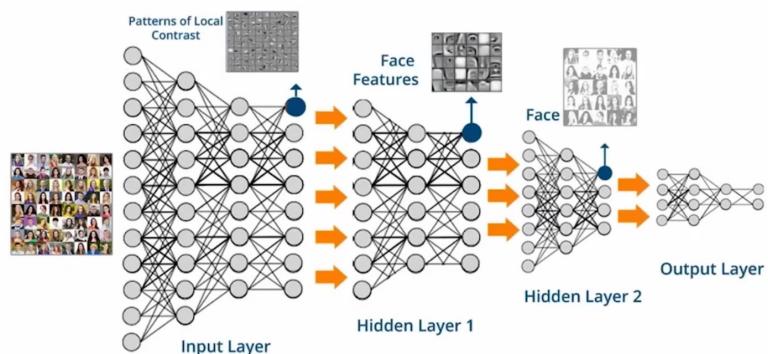
18

Học sâu là gì?



19

Học sâu là gì?



20

Ứng dụng của học sâu



Self Driving Cars



Voice Controlled Assistance



Automatic Image Caption Generation



Automatic Machine Translation

21

THANK YOU



22

DEEP LEARNING

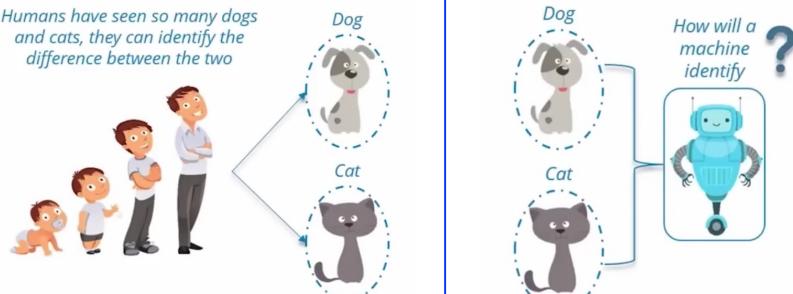
Perceptron đa tầng (MLP)

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

Học sâu là gì?

Humans have seen so many dogs and cats, they can identify the difference between the two



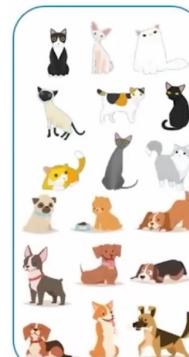
2

Nội dung

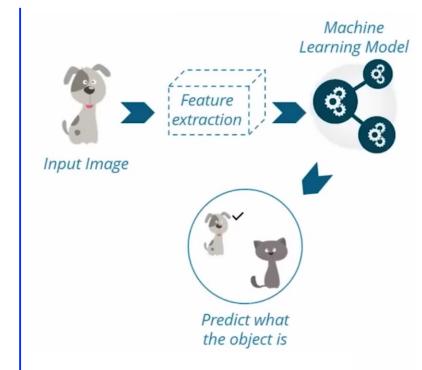
- Học sâu là gì?
- Học sâu hoạt động như thế nào?
- Perceptron đơn tầng
- Hạn chế của Perceptron đơn tầng
- Perceptron đa tầng (MLP)

1

Học sâu là gì?



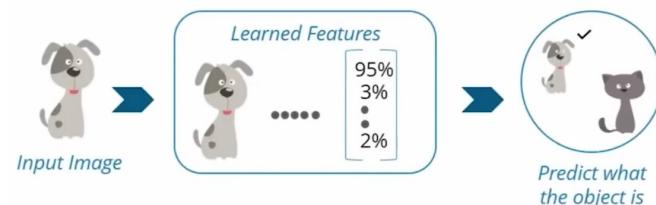
Training



3

Học sâu là gì?

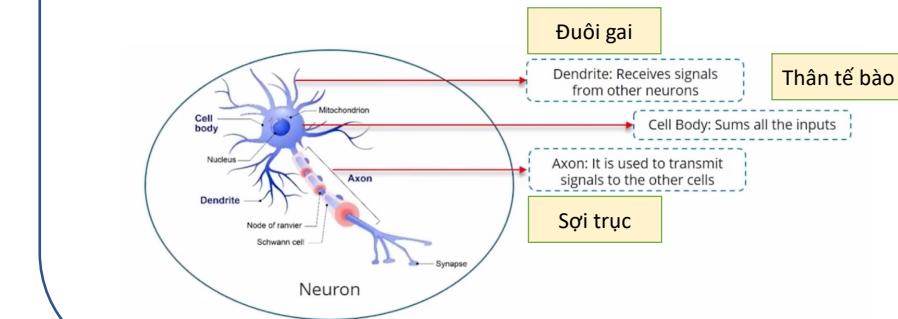
- Học sâu bỏ qua bước trích đặc trưng thủ công (feature extraction), ảnh đầu vào được đưa trực tiếp vào thuật toán/mô hình học sâu, để dự báo đối tượng là gì.



4

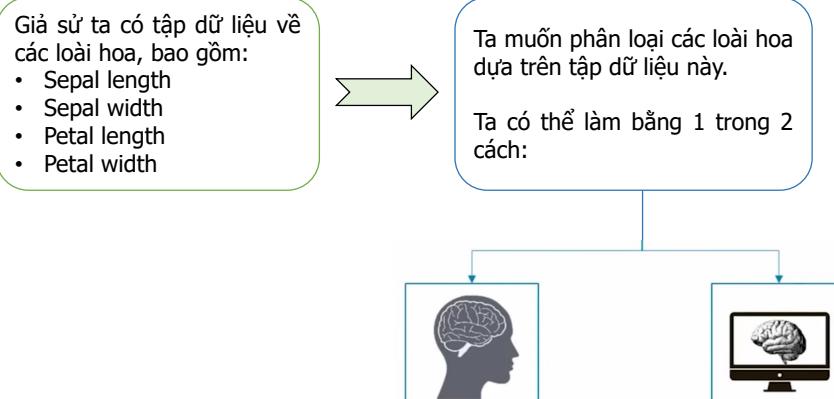
Học sâu hoạt động như thế nào?

- Học sâu là một dạng của học máy (machine learning), sử dụng một mô hình tính toán dựa trên bộ não của con người



5

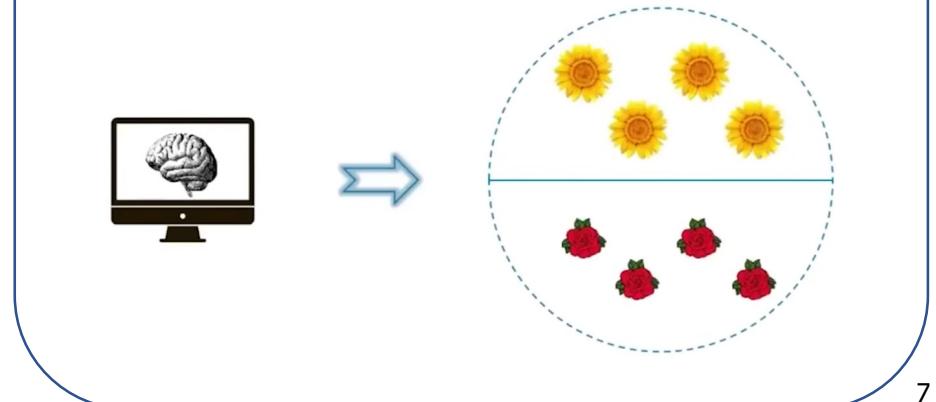
Tại sao ta cần nơ rơnh nhân tạo?



6

Tại sao ta cần nơ rơnh nhân tạo?

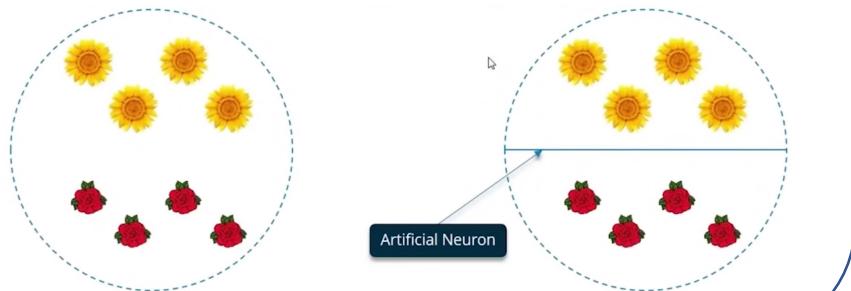
- Phân loại hoa bằng AI



7

Tại sao ta cần nơ ron nhân tạo?

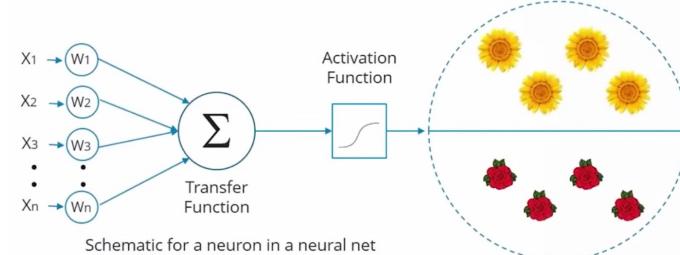
- Hệ thống AI phân loại 2 loài hoa
 - Nhờ vào 1 nơ ron nhận tạo, ta có thể phân loại 2 loài hoa



8

Perceptron

- Perceptron là một loại nơ ron nhân tạo có mô hình tính toán gồm:
 - Hàm truyền (transfer function), thường là tổng
 - Hàm kích hoạt (activation function): ngưỡng, sigmoid

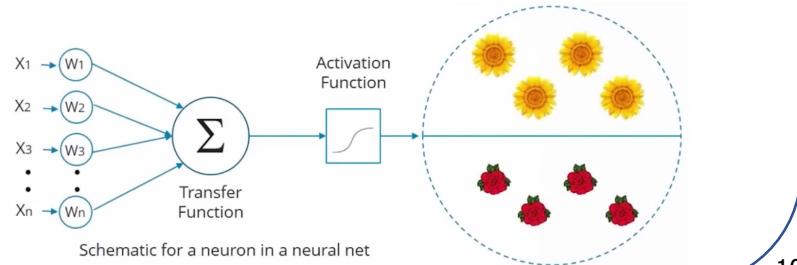


9

Perceptron

- x_j : đầu vào
- w_j : các trọng số
- f : hàm kích hoạt

$$\hat{y} = f \left(\sum_{j=1}^n x_j w_j \right)$$

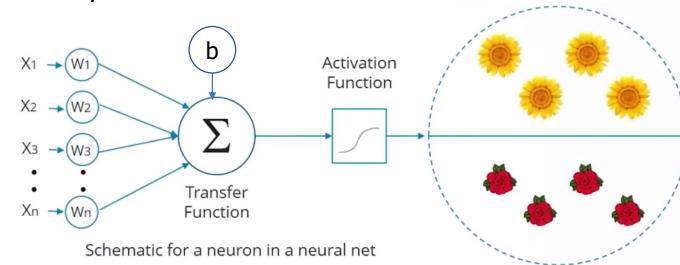


10

Perceptron

- x_j : đầu vào
- w_j : các trọng số
- f : hàm kích hoạt
- b : bias/threshold

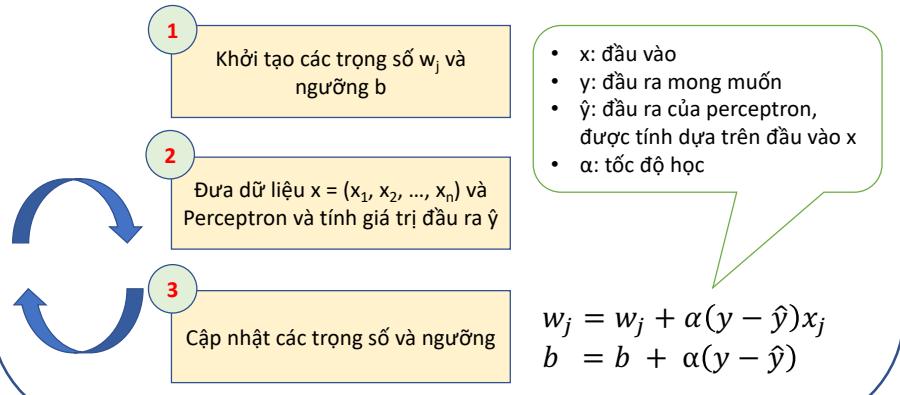
$$\hat{y} = f \left(\sum_{j=1}^n x_j w_j + b \right)$$



11

Perceptron

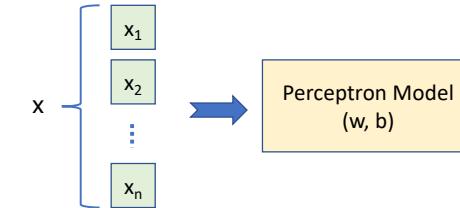
- Thuật toán huấn luyện Perceptron



12

Perceptron

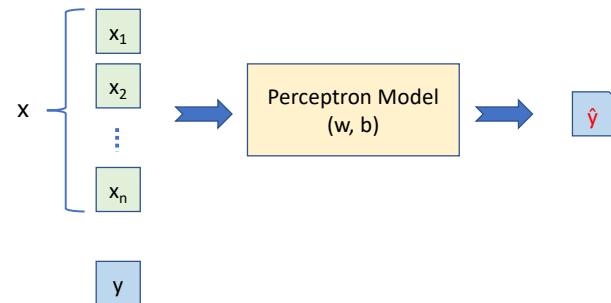
- Tổng quát hóa



13

Perceptron

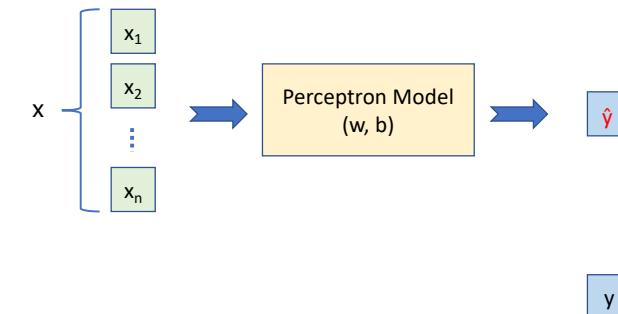
- Tổng quát hóa



14

Perceptron

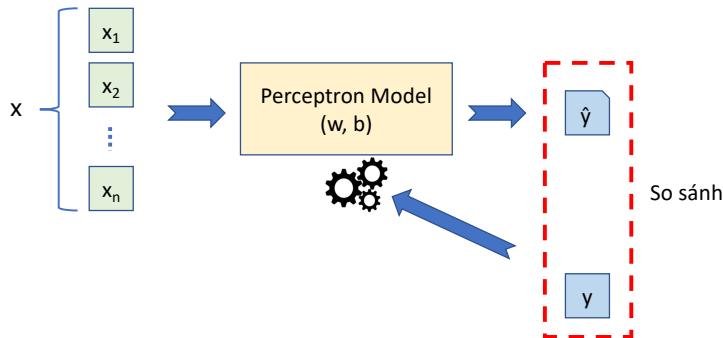
- Tổng quát hóa



15

Perceptron

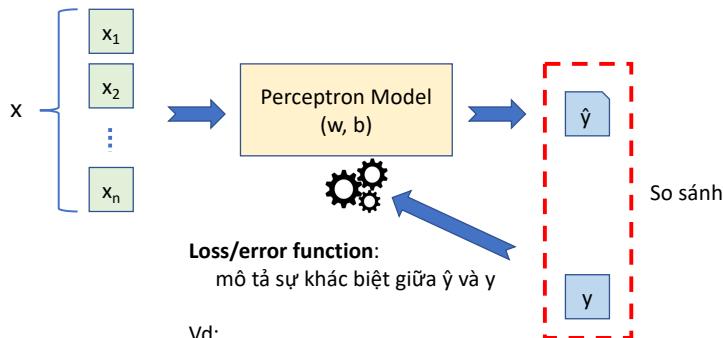
- Tổng quát hóa



16

Perceptron

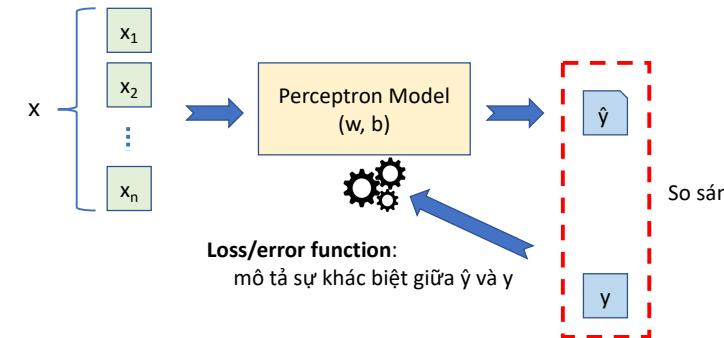
- Tổng quát hóa



18

Perceptron

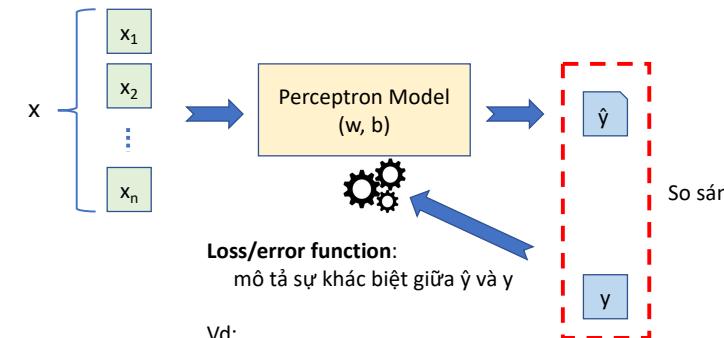
- Tổng quát hóa



17

Perceptron

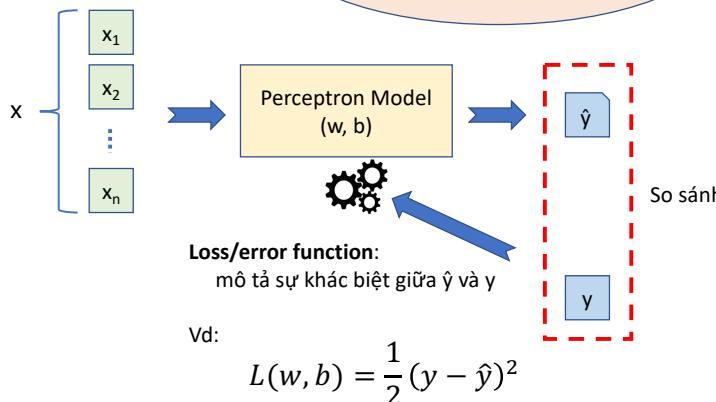
- Tổng quát hóa



19

Perceptron

- Tổng quát hóa

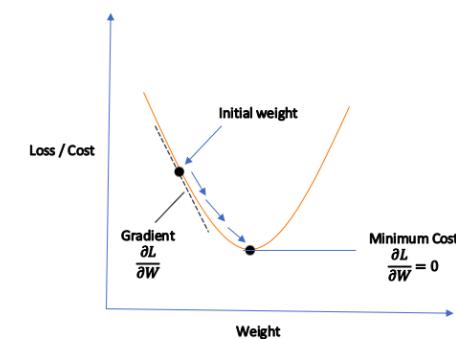


20

Perceptron

- Tìm w, b sao cho L nhỏ nhất
 - Thuật toán giảm gradient

$$L(w, b) = \frac{1}{2} (y - \hat{y})^2$$



21

Perceptron

- Tìm w, b sao cho L nhỏ nhất

- Thuật toán giảm gradient

$$L(w, b) = \frac{1}{2} (y - \hat{y})^2$$

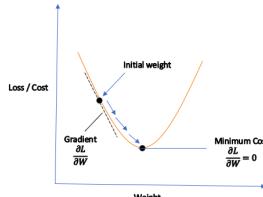
- Khởi tạo ngẫu nhiên w, b

- Lặp

- Cập nhật w, b theo hướng ngược hướng của gradient L

$$w_j = w_j - \alpha \frac{\partial L}{\partial w_j}$$

$$b = b - \alpha \frac{\partial L}{\partial b}$$



22

Perceptron

- Tính đạo hàm riêng

$$L(w, b) = \frac{1}{2} (y - \hat{y})^2$$

$$\frac{\partial L}{\partial w_j} = ?$$

$$\hat{y} = f\left(\sum_{j=1}^n x_j w_j + b\right)$$

$$\frac{\partial L}{\partial b} = ?$$

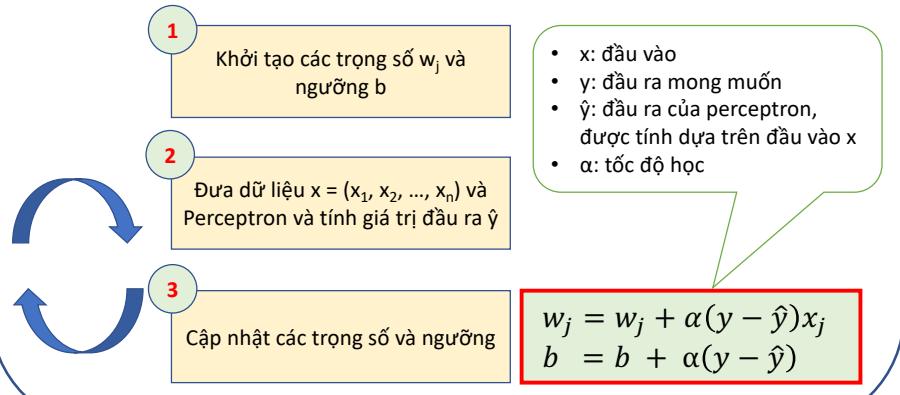
$$f(u) = u$$

x_j, y : hằng số

23

Perceptron

- Thuật toán huấn luyện Perceptron (xem lại)



24

Perceptron

- Quy trình

- Xây dựng mô hình, xác định các tham số
- Định nghĩa hàm loss/error
- Tìm tham số sao cho hàm loss nhỏ nhất

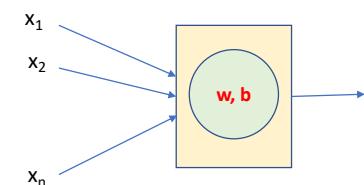
25

Perceptron

- Quy trình
- Xây dựng mô hình, xác định các tham số
 - Định nghĩa hàm loss/error
 - Tìm tham số sao cho hàm loss nhỏ nhất

- Cài đặt bằng Keras
- Tạo mô hình chỉ gồm 1 tầng, 1 nơ ron
 - Liên kết với mô hình với hàm loss và phương pháp tối ưu
 - Thực hiện tối ưu hàm loss

26



27

Perceptron

- Cài đặt Perceptron bằng Keras

- Tạo mô hình chỉ gồm 1 tầng, 1 nơ ron
- Liên kết với mô hình với hàm loss, phương pháp tối ưu và tiêu chí đánh giá
- Thực hiện tối ưu hàm loss

Perceptron

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# 1. create model
model = Sequential()
model.add(Dense(1, use_bias=True, input_shape=(n,), activation=None))

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

# 3. fit the model
model.fit(X_train, y_train, epochs=150,
          batch_size=32, verbose=0)
```

28

Perceptron

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# 1. create model
model = Sequential()
model.add(Dense(1, use_bias=True, input_shape=(n,), activation='sigmoid'))

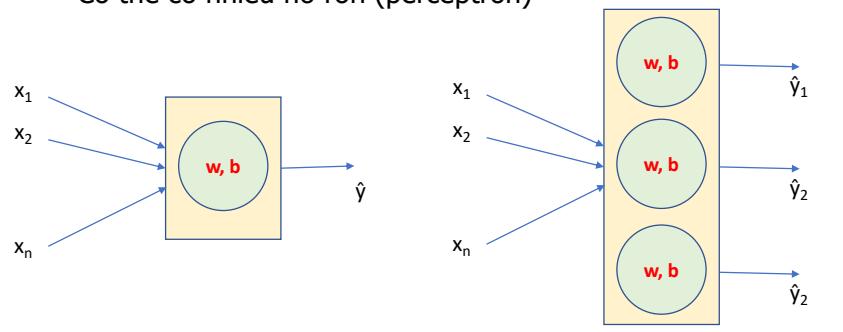
# 2. compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
              metrics=['accuracy'])

# 3. fit the model
model.fit(X_train, y_train, epochs=150,
          batch_size=32, verbose=0)
```

29

Perceptron đơn tầng (SLP)

- Mạng nơ ron chỉ gồm 1 tầng
 - Có thể có nhiều nơ ron (perceptron)



30

Thực hành

- Phân loại hoa iris (2 lớp)
 - Download tập dữ liệu iris (<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/>)
 - Giữ lại 100 dòng đầu tiên => lưu vào biến Data
 - Biến đổi nhãn (cột cuối)
 - Iris-setosa => 0
 - Iris-versicolor => 1
 - Xáo trộn dữ liệu Data, tách ra thành train (80%) và test (20%)

31

Thực hành

- Xây dựng mô hình
 - 1 tầng (Dense) gồm 1 nơ ron duy nhất (2 lớp)
 - input_shape: (4,)
 - use_bias: true (mặc định)
 - Activation: sigmoid
- Biên dịch mô hình
 - Optimizer: 'adam'
 - Loss: 'binary_crossentropy'
 - Metrics: ['accuracy']

32

Thực hành

- Xây dựng mô hình
- Biên dịch mô hình
- Huấn luyện mô hình

```
model.fit(train[:, 0:4], train[:, -1],  
          epochs=150, batch_size=32, verbose=0)
```
- Đánh giá mô hình

```
score = model.evaluate(test[:, 0:4],  
                      test[:, -1], verbose=0)  
print("Test loss:", score[0])  
print("Test accuracy:", score[1])
```

33

Thực hành

- Phân loại hoa iris (3 lớp)
 - Download tập dữ liệu iris (<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/>)
 - Xử lý toàn bộ tập dữ liệu
 - Biến đổi nhãn
 - Iris-setosa => 0
 - Iris-versicolor => 1
 - Iris-virginica => 2

34

Thực hành

- Xây dựng mô hình
 - 1 tầng (Dense) gồm 3 nơ ron (3 lớp)
 - input_shape: (4,)
 - use_bias: true (mặc định)
 - Activation: softmax
- Biên dịch mô hình
 - Optimizer: 'adam'
 - Loss: 'sparse_categorical_crossentropy'
 - Metrics: ['accuracy']

35

Tuần tới

- Nhược điểm của Perceptron
- Mạng perceptron đa tầng (MLP)
- Cài đặt mạng perceptron đa tầng bằng Keras
- Behind the scene of Keras
 - Tensorflow
 - Lan truyền ngược (Backpropagation)
 - Tính đạo hàm tự động (Gradients and automatic differentiation)

36

THANK YOU



37

DEEP LEARNING

Perceptron đa tầng (MLP)

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

Tensorflow

- Tensors là phương pháp biểu diễn dữ liệu chuẩn trong học sâu
- Tensors là các mảng nhiều chiều

2

Nội dung

- Tensor là gì?
- Đồ thị tính toán
- Cài đặt Perceptron bằng Tensorflow

1

Tensorflow

- Tensors là các mảng nhiều chiều
 - Số (0 chiều)
 - Vector (1 chiều)
 - Ma trận (2 chiều)
 - Mảng nhiều chiều (n-d array)

't'
'e'
'n'
'g'
'o'
'r'

Tensor of dimensions[6]

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

Tensor of dimensions[6,4]

2	1	2	1	8
2	4	5	0	5
2	3	6	2	8
7	4	1	5	2
7	7	3	2	6

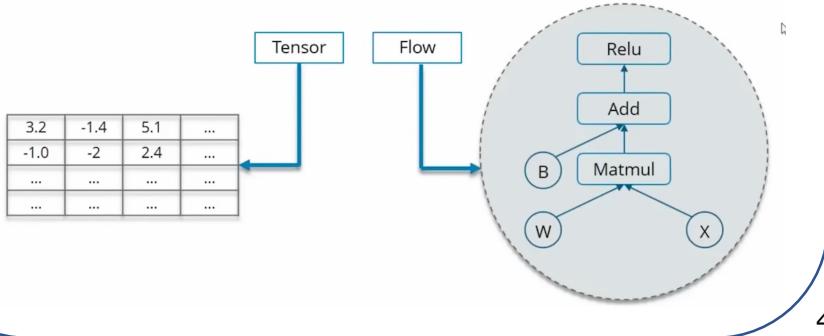
Tensor of dimensions[6,4,2]

3

Tensorflow

- Tensorflow (luồng tensors)

- Trong Tensorflow các tính toán được biểu diễn dưới dạng đồ thị luồng dữ liệu (dataflow graph) hay đồ thị tính toán (computation graph)



4

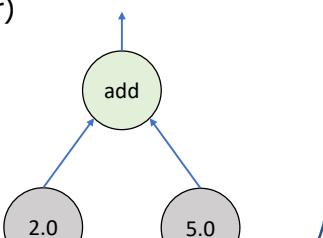
Tensorflow

- Đồ thị tính toán (computational graph)

- Các biểu thức tính toán được tổ chức dưới dạng đồ thị
 - Nút/định nhận 0/nhiều đầu vào (inputs), tính toán và cho ra 1 đầu ra (output)

- Nút trong đồ thị tính toán (tf.Tensor)

- Hằng (tf.constant)
 - Biến (tf.Variable)
 - Phép toán (tf.Operation)
 - Hàm (tf.function)**



6

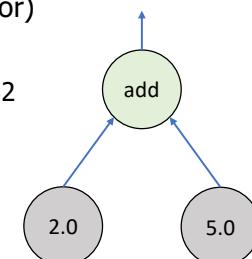
Tensorflow

- Đồ thị tính toán (computational graph)

- Các biểu thức tính toán được tổ chức dưới dạng đồ thị
 - Nút/định nhận 0/nhiều đầu vào (inputs), tính toán và cho ra 1 đầu ra (output)

- Nút trong đồ thị tính toán (tf.Tensor)

- Hình dạng (shape)
- Kiểu dữ liệu (dtype): float32, int32



5

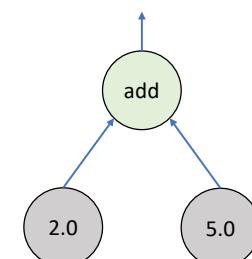
Tensorflow

- Đồ thị tính toán (computational graph)

- Các biểu thức tính toán được tổ chức dưới dạng đồ thị
 - Nút/định nhận 0/nhiều đầu vào (inputs), tính toán và cho ra 1 đầu ra (output)

```
import tensorflow as tf  
  
a = tf.constant(2.0, tf.float32)  
b = tf.constant(5.0, tf.float32)  
c = tf.add(a, b)  
print(a, b, c)
```

```
tf.Tensor(2.0, shape=(), dtype=float32)  
tf.Tensor(5.0, shape=(), dtype=float32)  
tf.Tensor(7.0, shape=(), dtype=float32)
```

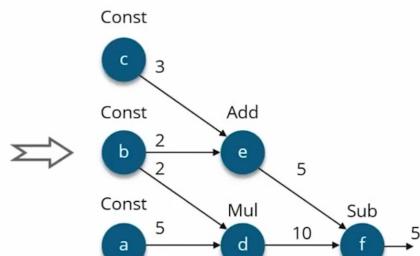


7

Tensorflow

- Ví dụ

```
import tensorflow as tf  
  
a = tf.constant(5)  
b = tf.constant(2)  
c = tf.constant(3)  
  
d = tf.multiply(a, b)  
e = tf.add(c, b)  
f = tf.subtract(d, e)
```

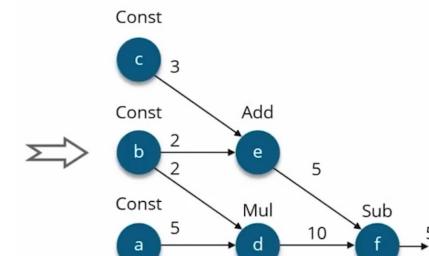


8

Tensorflow

- Ví dụ

```
import tensorflow as tf  
  
a = tf.constant(5)  
b = tf.constant(2)  
c = tf.constant(3)  
  
d = a * b  
e = c + b  
f = d - e
```



9

Tensorflow

- Sử dụng hàm trong Tensorflow
 - Hàm tương đương với một đồ thị tính toán
 - Tham số: input
 - Giá trị trả về: output
 - **Hàm có thể được dùng như một nút trong đồ thị tính toán**

10

Tensorflow

- Sử dụng hàm trong Tensorflow

```
import tensorflow as tf  
  
def my_add(x, y):  
    return x + y  
  
a = tf.constant(5)  
b = tf.constant(2)  
  
my_func = tf.function(my_add)  
print(my_func)  
c = my_func(a, b)  
print(c)
```

<tensorflow.python.eager.def_function.Function object at 0x115b7b5e0>
tf.Tensor(7, shape=(), dtype=int32)

11

Tensorflow

- Sử dụng hàm trong Tensorflow

```
import tensorflow as tf

@tf.function
def my_func(x, y):
    return x + y

a = tf.constant(5)
b = tf.constant(2)

print(my_func)
c = my_func(a, b)
print(c)

<tensorflow.python.eager.def_function.Function object at 0x115b7b5e0>
tf.Tensor(7, shape=(), dtype=int32)
```

12

Tensorflow

- Sử dụng hàm trong Tensorflow

```
import tensorflow as tf

@tf.function
def h(x, w1, w0):
    return x * w1 + w0

x = tf.constant(5.0)
y = tf.constant(10.0)

w0 = tf.Variable(1.0)
w1 = tf.Variable(2.5)
diff = h(x, w1, w0) - y
```

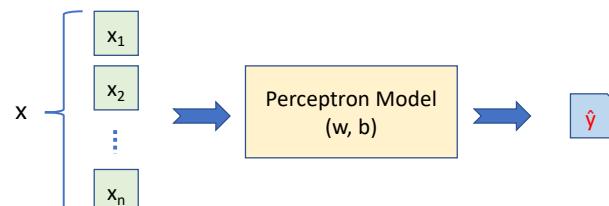


tf.Tensor(3.5, shape=(), dtype=float32)

13

Cài đặt Perceptron bằng TF

- Perceptron nhận đầu vào x và sinh ra \hat{y}

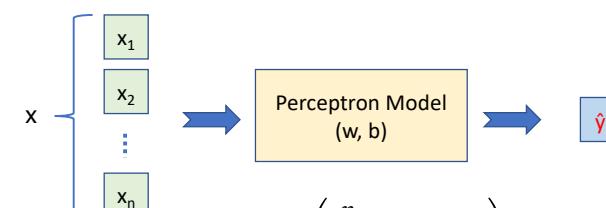


14

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

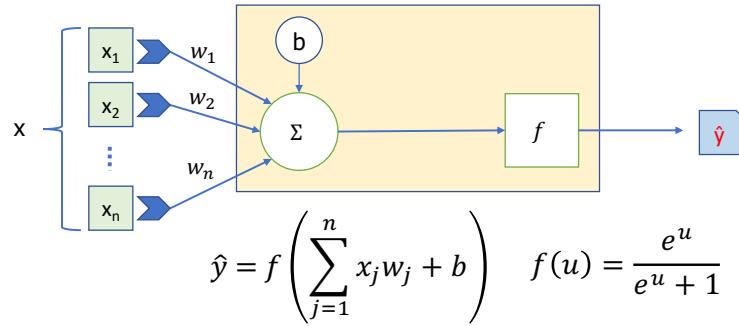


$$\hat{y} = f \left(\sum_{j=1}^n x_j w_j + b \right) \quad f(u) = \frac{e^u}{e^u + 1}$$

15

Cài đặt Perceptron bằng TF

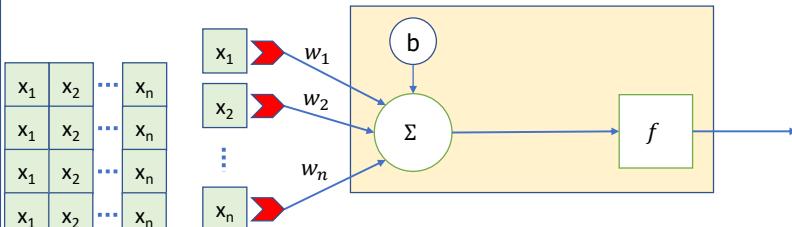
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



16

Cài đặt Perceptron bằng TF

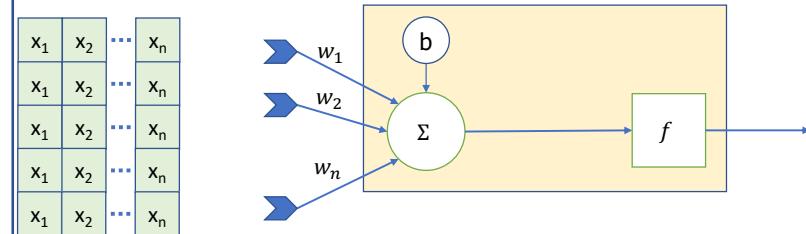
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



18

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



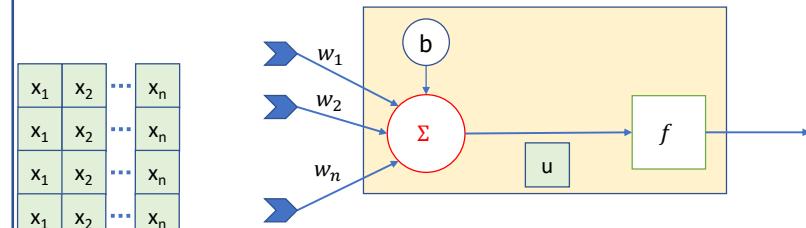
Thực tế, mỗi mô hình deep learning nhận đầu vào là **một lô (batch)** dữ liệu đầu vào chứ không phải 1 phần tử dữ liệu.

Vì thế cần định nghĩa hàm tính toán của mô hình trên toàn bộ lô dữ liệu này.

17

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

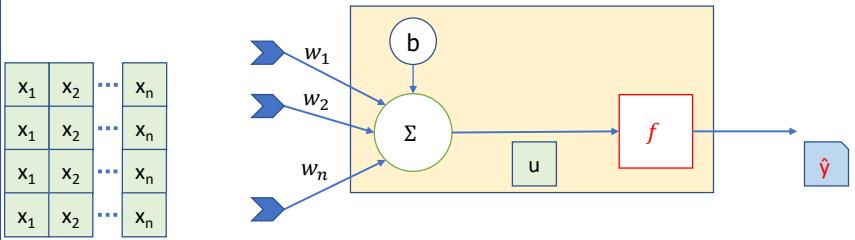


19

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

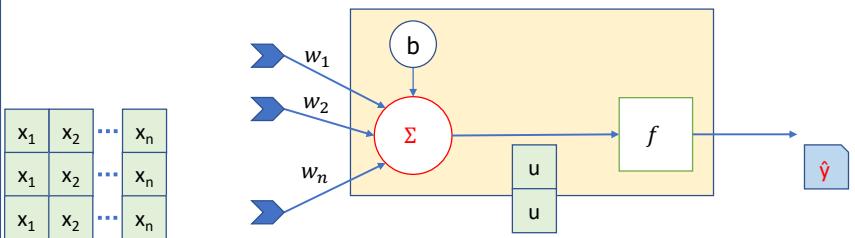


20

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

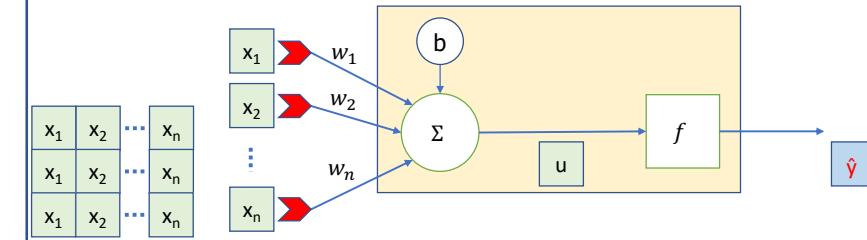


22

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

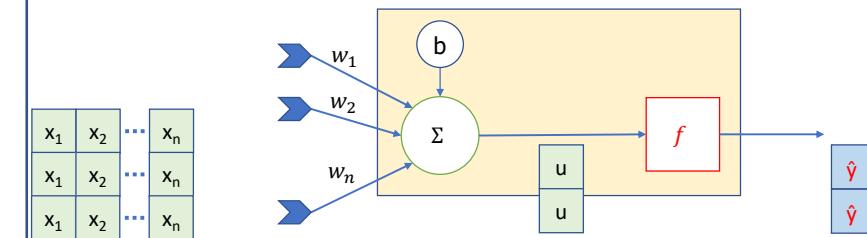


21

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

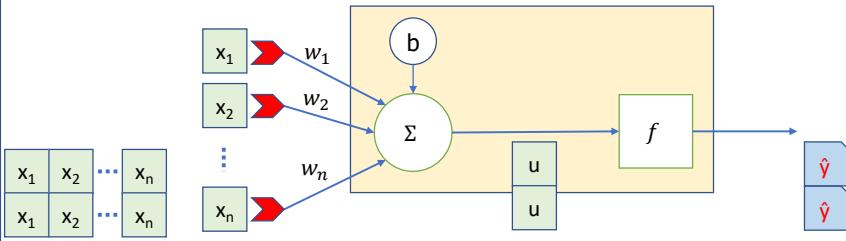


23

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

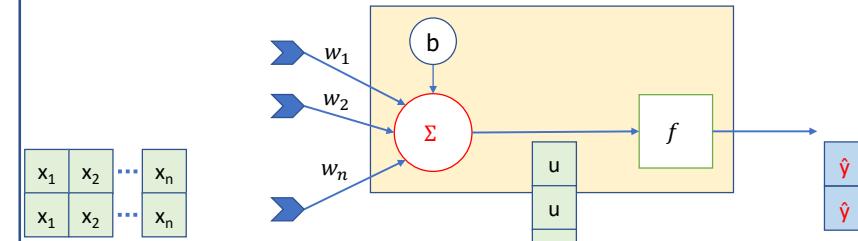


24

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

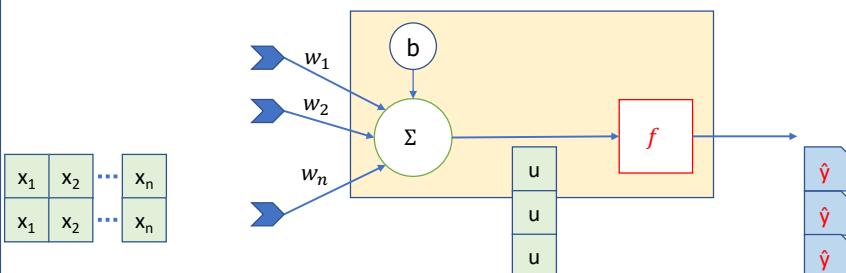


25

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .

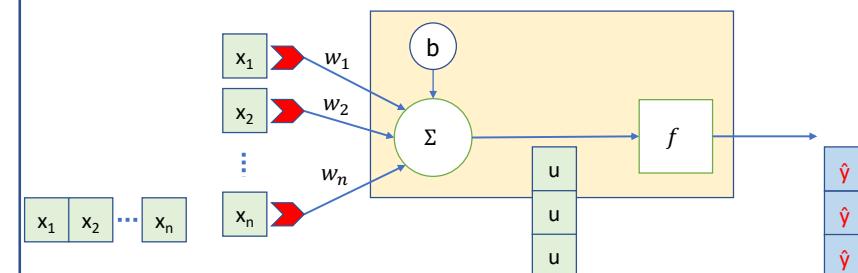


26

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như

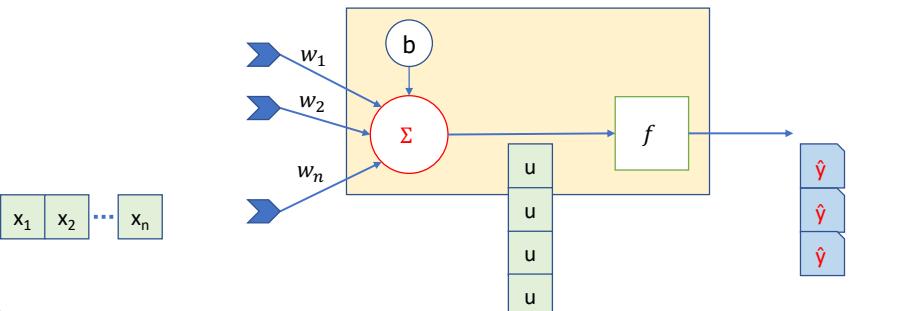
- Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



27

Cài đặt Perceptron bằng TF

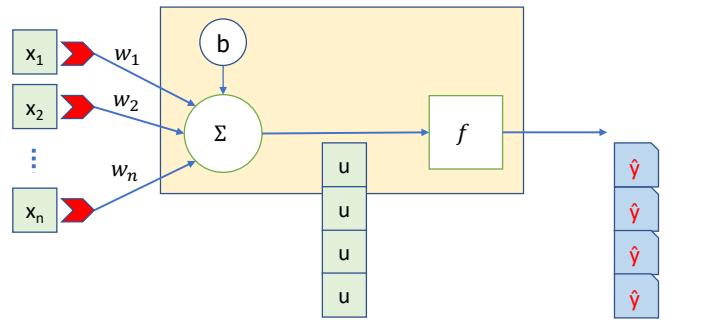
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



28

Cài đặt Perceptron bằng TF

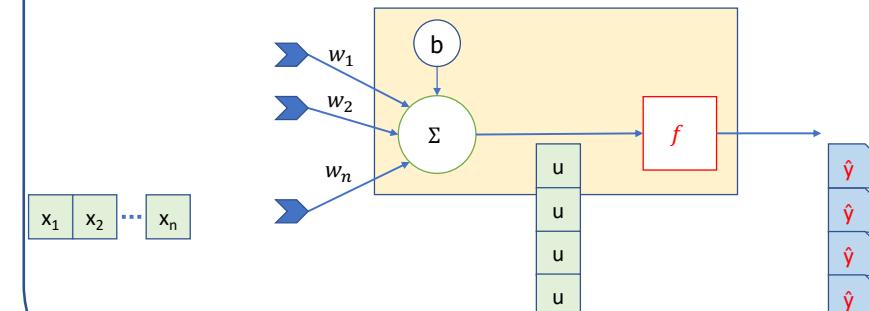
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



30

Cài đặt Perceptron bằng TF

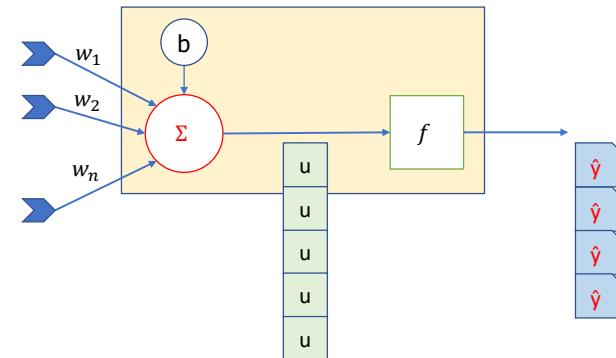
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



29

Cài đặt Perceptron bằng TF

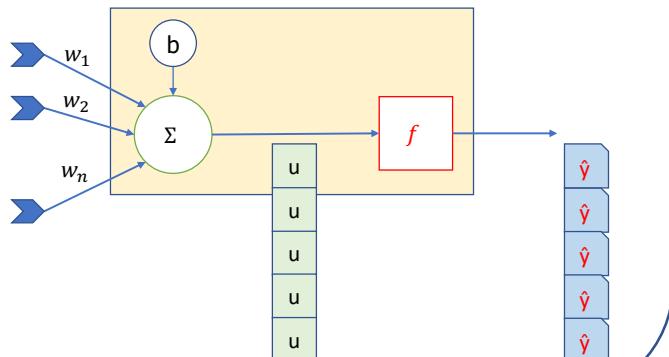
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



31

Cài đặt Perceptron bằng TF

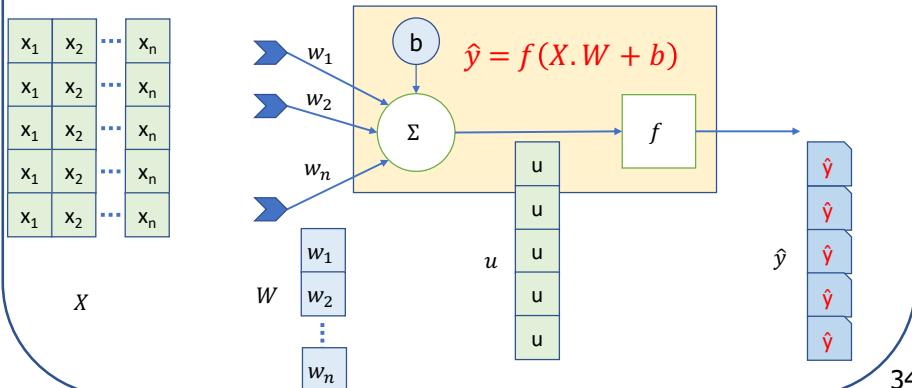
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



32

Cài đặt Perceptron bằng TF

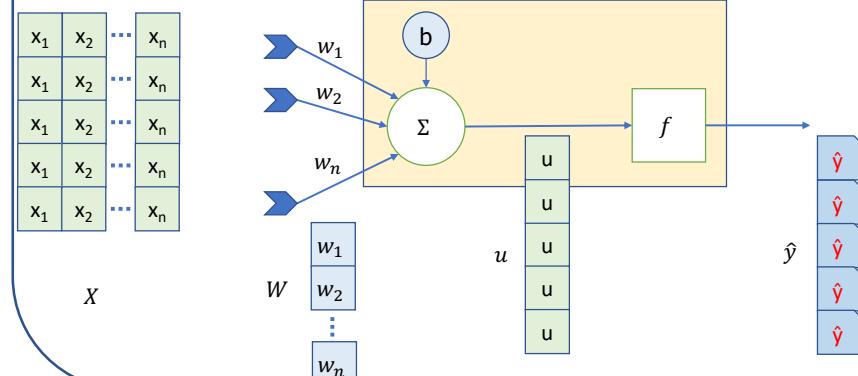
- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



34

Cài đặt Perceptron bằng TF

- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



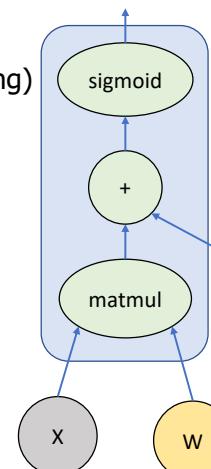
33

Cài đặt Perceptron bằng TF

- Đồ thị tính toán của Perceptron
 - X : ma trận dữ liệu huấn luyện (hàng)
 - W : ma trận trọng số (biến)
 - b : độ lệch (biến)

$$\hat{y} = f(X \cdot W + b)$$

$$f(u) = \frac{e^u}{e^u + 1}$$



35

Cài đặt Perceptron bằng TF

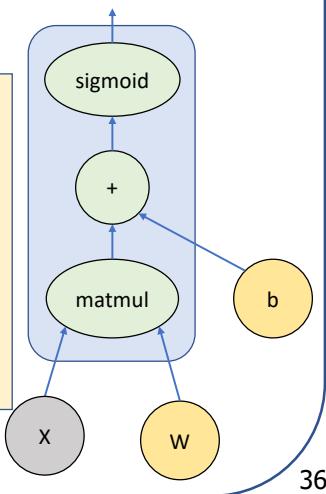
- Đồ thị tính toán của Perceptron

```
@tf.function
def predict(X, W, b):
    return tf.nn.sigmoid(tf.matmul(X, W) + b)

X = tf.constant([[0.0, 0], [0, 1],
                 [1, 0], [1, 1]])

W = tf.Variable([[1.0], [2]])
b = tf.Variable(0.0)

y_hat = predict(X, W, b)
print(y)
```

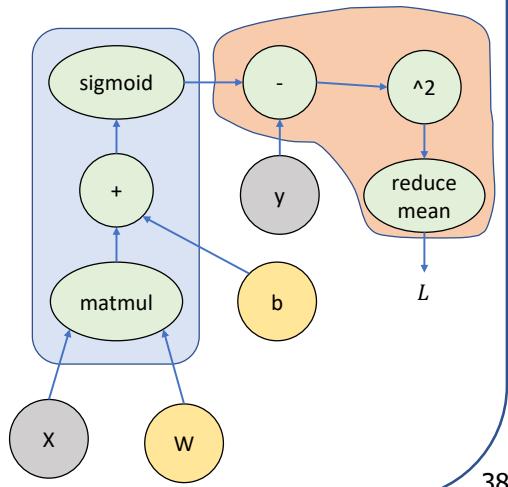


36

Cài đặt Perceptron bằng TF

- Kết hợp với hàm lỗi

$$L = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$



38

Cài đặt Perceptron bằng TF

- Đồ thị tính toán của Perceptron

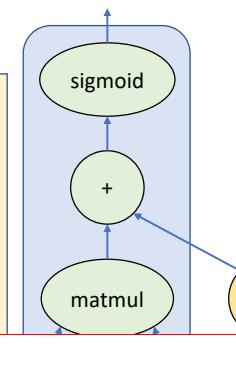
```
@tf.function
def predict(X, W, b):
    return tf.nn.sigmoid(tf.matmul(X, W) + b)

X = tf.constant([[0.0, 0], [0, 1],
                 [1, 0], [1, 1]])

W = tf.Variable([[1.0], [2]])
b = tf.Variable(0.0)

y_hat = predict(X, W, b)
print(y)
```

```
tf.Tensor([0.5
          [0.88079715
          [0.7310586
          [0.95257413]], shape=(4, 1), dtype=float32)
```



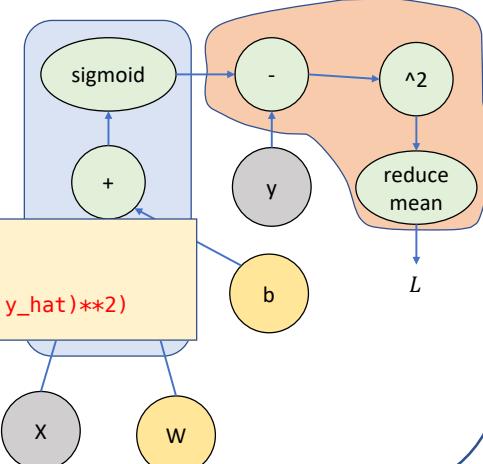
37

Cài đặt Perceptron bằng TF

- Kết hợp với hàm lỗi

$$L = \frac{1}{m} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$

```
@tf.function
def L(y, y_hat):
    return tf.reduce_mean((y - y_hat)**2)
```



39

Cài đặt Perceptron bằng TF

- Huấn luyện mô hình \Leftrightarrow Tối ưu hàm lỗi bằng gradient descent
 - Khởi tạo W, b
 - Lặp
 - Tính y_{hat}
 - Tính giá trị hàm lỗi
 - Tính đạo hàm riêng
 - Cập nhật W và b

$$W = W - \alpha \frac{\partial L}{\partial W}$$

$$b = b - \alpha \frac{\partial L}{\partial b}$$

40

Cài đặt Perceptron bằng TF

- Tính đạo hàm riêng tự động với GradientTape

```
X = tf.constant([[0.0, 0], [0, 1], [1, 0], [1, 1]])
y = tf.constant([[0.0], [0], [0], [1]])
W = tf.Variable([[1.0], [2]])
b = tf.Variable(0.0)

alpha = 0.1
for it in range(500):
    with tf.GradientTape() as t:
        current_loss = L(y, predict(X, W, b))

        print("it", it, current_loss)
        dW, db = t.gradient(current_loss, [W, b])
        W.assign_sub(alpha * dW)
        b.assign_sub(alpha * db)
```

42

Cài đặt Perceptron bằng TF

- Tính đạo hàm riêng tự động với GradientTape

```
X = tf.constant([[0.0, 0], [0, 1], [1, 0], [1, 1]])
y = tf.constant([[0.0], [0], [0], [1]])
W = tf.Variable([[1.0], [2]])
b = tf.Variable(0.0)

alpha = 0.1
for it in range(500):
    with tf.GradientTape() as t:
        current_loss = L(y, predict(X, W, b))

        print("it", it, current_loss)
        dW, db = t.gradient(current_loss, [W, b])
        W.assign_sub(alpha * dW)
        b.assign_sub(alpha * db)
```

41

Cài đặt Perceptron bằng TF

- Tính đạo hàm riêng tự động với GradientTape

```
alpha = 0.1
for it in range(500):
    with tf.GradientTape() as t:
        current_loss = L(y, predict(X, W, b))

        print("it", it, current_loss)
        dW, db = t.gradient(current_loss, [W, b])
        W.assign_sub(alpha * dW)
        b.assign_sub(alpha * db)

y_hat = predict(X, W, b)
print(y_hat)
```

43

Thực hành 1

- Tổng hợp các mã lệnh được trình bày trong phần trên để xây dựng một mạng nơ ron perceptron đơn tăng mô phỏng phép toán AND
- Khởi tạo ngẫu nhiên W và b thay vì gán sẵn
 - Xem module tf.random
- Sau khi huấn luyện xong, mô hình cho đầu ra là 1 số thực từ 0 đến 1. Cần phải viết thêm phần xử lý để xác định nhãn dự báo (so sánh với 0.5)

```
X = tf.constant([[0.0, 0],  
                 [0, 1],  
                 [1, 0],  
                 [1, 1]])  
  
y = tf.constant([[0.0],  
                 [0],  
                 [0],  
                 [1]])
```

44

Thực hành 2

- Làm lại bài thực hành 1 với hàm lỗi binary crossentropy được định nghĩa như sau:

$$L = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(\hat{y}^{(i)}))$$

45

Thực hành 3

- Làm lại bài phân loại hoa iris (2 lớp) với Tensorflow thay vì dùng Keras
 - Sử dụng hàm lỗi Binary crossentropy
 - Cần phân ngưỡng kết quả đầu ra để có được nhãn chính xác
 - Tính độ chính xác phân lớp bằng cách so sánh nhãn dự báo và nhãn mong muốn
- Có thể dùng hàm Tensor.numpy() để lấy giá trị của Tensor về dạng numpy để hậu xử lý.

46

THANK YOU



47

DEEP LEARNING

Perceptron đa tầng (MLP)

(tiếp theo)

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

Percetron

- 1 perceptron = mạng perceptron đơn tầng chỉ gồm 1 nơ ron (perceptron) duy nhất
 - Có khả năng phân lớp tuyến tính (đường thẳng, mặt phẳng, siêu phẳng) nhị phân (hai lớp)

2

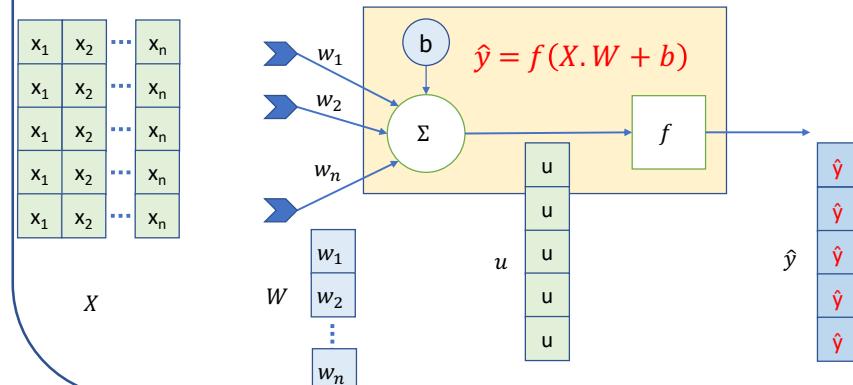
Nội dung

- Cài đặt Perceptron bằng Tensorflow (nhắc lại)
- Mở rộng
 - Mạng nơ ron đơn tầng đa lớp (nhiều perceptron)
- Mạng nơ ron đa tầng
 - hai lớp
 - đa lớp

1

Cài đặt Perceptron bằng TF (nhắc lại)

- Cài đặt Perceptron như
 - Một hàm nhận đầu vào x , tính toán và cho ra đầu ra \hat{y} .



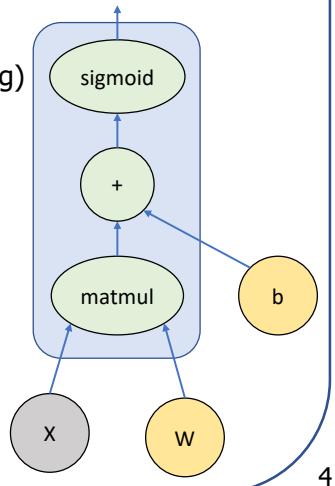
3

Cài đặt Perceptron bằng TF (nhắc lại)

- Đồ thị tính toán của Perceptron
 - X: ma trận dữ liệu huấn luyện (hằng)
 - W: ma trận trọng số (biển)
 - b: độ lệch (biển)

$$\hat{y} = f(X \cdot W + b)$$

$$f(u) = \frac{e^u}{e^u + 1}$$



Mạng Percetron đơn tầng (SLP)

- Mạng Single Layer Perceptron (SLP)
 - Mạng đơn tầng gồm nhiều (≥ 3) nơ ron (perceptron)
 - Có khả năng phân lớp tuyến tính (đường thẳng, mặt phẳng, siêu phẳng) đa phân (từ 3 lớp trở lên)

6

Cài đặt Perceptron bằng TF (nhắc lại)

- Đồ thị tính toán của Perceptron

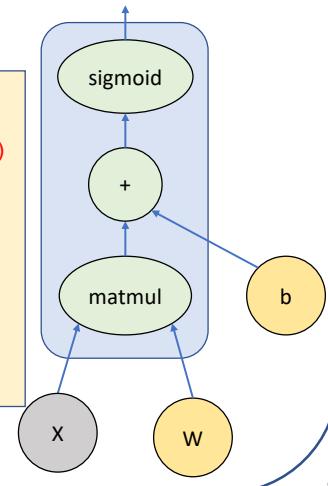
```

@tf.function
def predict(X, W, b):
    return tf.nn.sigmoid(tf.matmul(X, W) + b)

X = tf.constant([[0.0, 0], [0, 1],
                 [1, 0], [1, 1]])

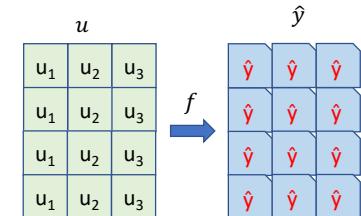
W = tf.Variable([[1.0], [2]])
b = tf.Variable(0.0)

y_hat = predict(X, W, b)
print(y_hat)
  
```



Mạng Perceptron đơn tầng, đa lớp

- Mạng SLP (đa lớp)
 - 1 tầng duy nhất gồm nhiều (≥ 3) nơ ron



$$\hat{y} = f(X \cdot W + b)$$

$$u = X \cdot W + b$$

X

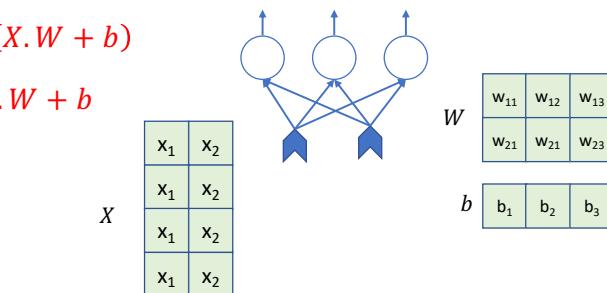
x_1	x_2

W

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}

b

b_1	b_2	b_3
-------	-------	-------



7

Cài đặt mạng Perceptron đơn tầng, đa lớp

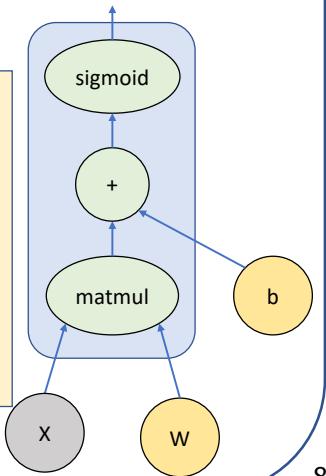
- Đồ thị tính toán SLP (không lỗi)

```
@tf.function
def predict(X, W, b):
    return tf.nn.softmax(tf.matmul(X, W) + b)

X = tf.constant([[0.0, 0], [0, 1],
                 [1, 0], [1, 1]])

W = tf.Variable(tf.random.normal((2, 3))
b = tf.Variable([0.0, 0.0, 0.0])

y_hat = predict(X, W, b)
print(y)
```



8

Cài đặt mạng Perceptron đơn tầng, đa lớp

- Đầu ra dự báo của phần tử i: $\hat{y}^{(i)}$
 - 1 vector, có số phần tử = số nơ ron
- Mã hoá đầu ra mong muốn
 - One hot encoding
 - $y^{(i)}$: vector, có số phần tử = số lớp, chỉ có duy nhất 1 phần tử = 1, các phần tử còn lại = 0
 - Ví dụ:
 - Setosa [1, 0, 0]
 - Versicolor [0, 1, 0]
 - Virginica [0, 0, 1]

10

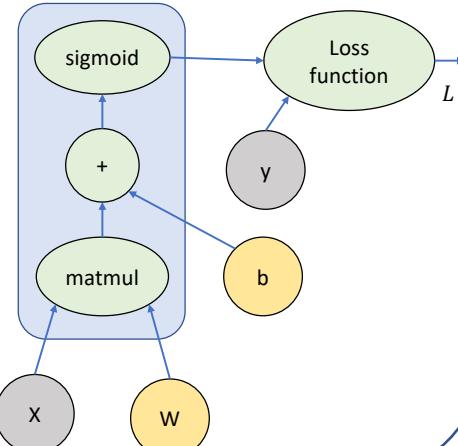
Cài đặt mạng Perceptron đơn tầng, đa lớp

- Kết hợp với hàm lỗi

$$L = \frac{1}{m} \sum_{i=1}^m d(y^{(i)}, \hat{y}^{(i)})$$

$d(y^{(i)}, \hat{y}^{(i)})$: đo sự khác biệt giữa đầu ra mong muốn ($y^{(i)}$) và đầu ra dự báo ($\hat{y}^{(i)}$) trên mẫu dữ liệu huấn luyện thứ i

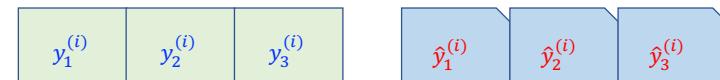
Hàm lỗi/mất mát = trung bình sự khác biệt trên toàn bộ tập huấn luyện



9

Cài đặt mạng Perceptron đơn tầng, đa lớp

- Đo sự khác biệt giữa đầu ra mong muốn và đầu ra dự báo của phần tử i
 - $d(y^{(i)}, \hat{y}^{(i)})$ = sự khác biệt của 2 vectors $y^{(i)}$ và $\hat{y}^{(i)}$



$$d(y^{(i)}, \hat{y}^{(i)}) = - \sum_{j=1}^k y_j^{(i)} \cdot \log(\hat{y}_j^{(i)})$$

Categorical Cross-Entropy

11

Cài đặt mạng Perceptron đơn tầng, đa lớp

- Kết hợp với hàm lỗi

$$L = \frac{1}{m} \sum_{i=1}^m \left(- \sum_{j=1}^k y_j^{(i)} \cdot \log(\hat{y}_j^{(i)}) \right) = -\frac{1}{m} \sum_{i=1}^m \left(\sum_{j=1}^k y_j^{(i)} \cdot \log(\hat{y}_j^{(i)}) \right)$$

```
@tf.function
def L(y, y_hat):
    return -tf.reduce_mean(tf.reduce_sum(y * log(y_hat), axis=1))
```

12

Thực hành 1

- Tổng hợp các mã lệnh được trình bày trong phần trên để xây dựng một mạng nơ ron perceptron đơn tầng, 3 lớp với dữ liệu huấn luyện như bên dưới

```
X = tf.constant([[0.0, 0],
                 [0, 1],
                 [1, 0],
                 [1, 1]])

y = tf.constant([[1.0, 0, 0],
                 [0, 1, 0],
                 [0, 0, 1],
                 [0, 0, 1]])
```

14

Cài đặt mạng Perceptron đơn tầng, đa lớp

Tổng kết cài đặt SLP đa lớp

- Mạng chỉ gồm 1 tầng, nhiều ($>=3$) nơ ron
- Xác định số nơ ron của mạng = số lớp của bài toán
- Định nghĩa kích thước của các tham số (Variables)
 - W: (n, k) n: số chiều dữ liệu, k: số lớp
 - b: (k,)
- Hàm activation: dùng softmax thay cho sigmoid
- Định nghĩa hàm lỗi: sử dụng Categorical Cross-Entropy (đa lớp) thay cho Binary Cross-Entropy (hai lớp)
- Mã hoá đầu ra mong muốn
 - One hot encoding

13

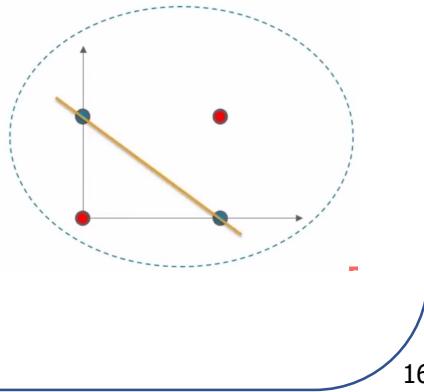
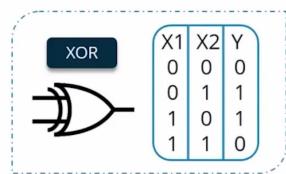
Thực hành 2

- Làm lại bài phân loại hoa iris (3 lớp) với Tensorflow thay vì dùng Keras
 - Cần mã hoá đầu ra mong muốn: có thể xử lý thủ công hoặc dùng công cụ
 - Sử dụng hàm lỗi Categorical Cross-Entropy
 - Cần xử lý đầu ra để tìm nhãn chính xác (sử dụng hàm argmax để tìm cột có giá trị lớn nhất)
 - Tính độ chính xác phân lớp bằng cách so sánh nhãn dự báo và nhãn mong muốn
- Có thể dùng hàm Tensor.numpy() để lấy giá trị của Tensor về dạng numpy để hậu xử lý.

15

Nhược điểm của mạng SLP

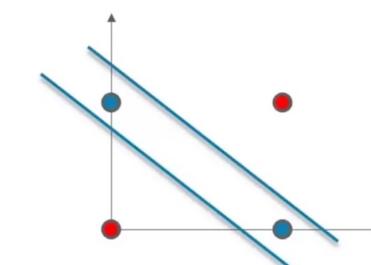
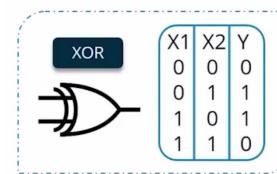
- Mô phỏng cổng XOR
 - 1 nơ ron



16

Nhược điểm của mạng SLP

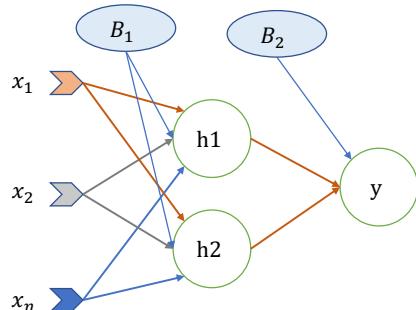
- Mô phỏng cổng XOR
 - 2 nơ ron và (1 nơ ron output)



17

Mạng nơ ron đa tầng MLP

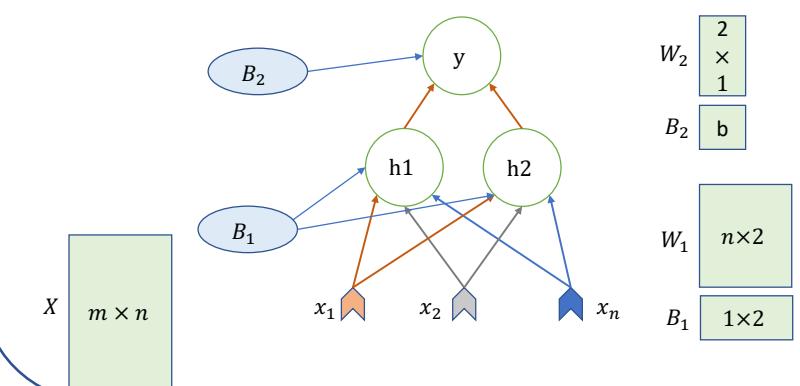
- Mô phỏng cổng XOR
 - 2 nơ ron và (1 nơ ron output)



18

Mạng nơ ron đa tầng MLP

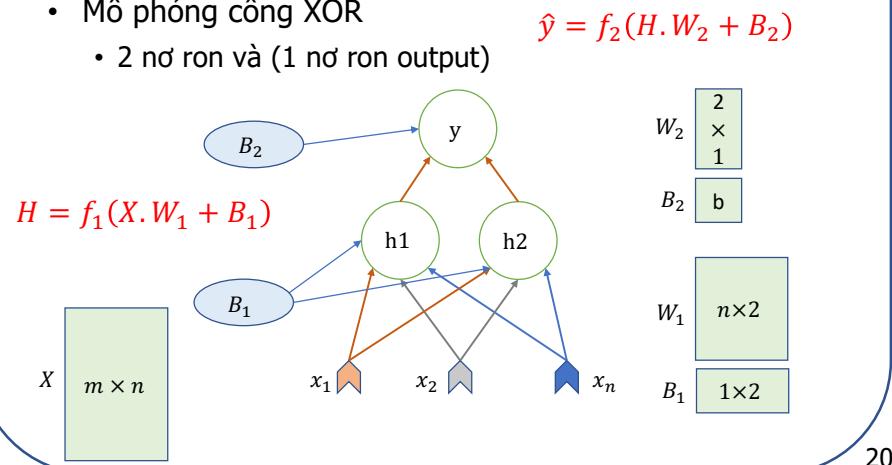
- Mô phỏng cổng XOR
 - 2 nơ ron và (1 nơ ron output)



19

Mạng nơ ron đa tầng MLP

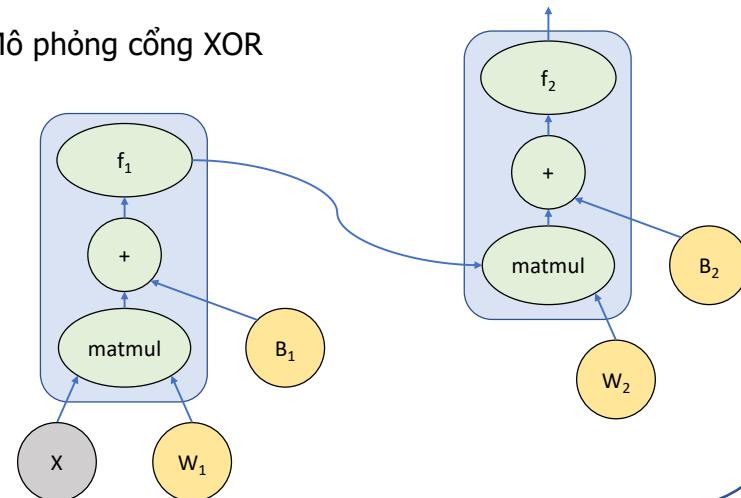
- Mô phỏng cổng XOR
 - 2 nơ ron và (1 nơ ron output)



20

Mạng nơ ron đa tầng MLP

- Mô phỏng cổng XOR



21

Mạng nơ ron đa tầng MLP

- Mô phỏng cổng XOR

```
@tf.function
def layer1(X, W, b):
    return tf.nn.relu(tf.matmul(X, W) + B)

@tf.function
def layer2(X, W, B):
    return tf.nn.sigmoid(tf.matmul(X, W) + B)

@tf.function
def predict(X, W1, B1, W2, B2):
    return layer2(layer1(X, W1, B1), W2, B2)
```

22

Mạng nơ ron đa tầng MLP

- Mô phỏng cổng XOR

```
n = 2
W1 = tf.Variable(tf.random.normal((n, 2))
B1 = tf.Variable([0.0, 0.0])

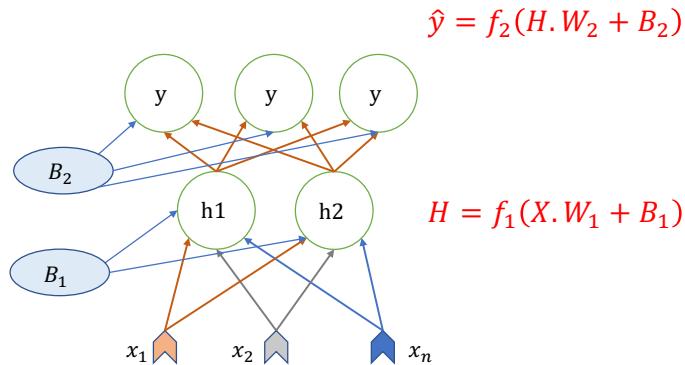
W2 = tf.Variable(tf.random.normal((2, 1)))
B2 = tf.Variable([0.0])

y_hat = predict(X, W1, B1, W2, B2)
print(y_hat)
```

23

Mạng nơ ron đa tầng MLP

- MLP đa lớp (tương tự MLP 2 lớp)



24

Thực hành 4

- Xây dựng một mạng nơ ron perceptron 2 tầng, 2 lớp. Đọc tập dữ liệu trong file data.csv, chia dữ liệu thành 2 phần 80% - 20%, dùng 80% huấn và 20% còn lại để đánh giá

26

Thực hành 3

- Tổng hợp các mã lệnh được trình bày trong phần trên để xây dựng một mạng nơ ron perceptron đa tầng (2 tầng), 2 lớp cho bài toán XOR với dữ liệu huấn luyện như bên dưới

```
x = tf.constant([[0.0, 0],  
                 [0, 1],  
                 [1, 0],  
                 [1, 1]])  
  
y = tf.constant([[0.0, 0, 0],  
                 [0, 1, 0],  
                 [0, 1, 0],  
                 [0, 0, 0]])
```

25

Thực hành 5

- Xây dựng một mạng nơ ron MLP xử lý tập dữ liệu: <https://archive.ics.uci.edu/ml/machine-learning-databases/character-trajectories/>
 - Chia dữ liệu thành 2 phần 80% - 20%, dùng 80% huấn và 20% còn lại để đánh giá

27

THANK YOU



DEEP LEARNING

Mạng nơ ron tích chập (CNN)

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

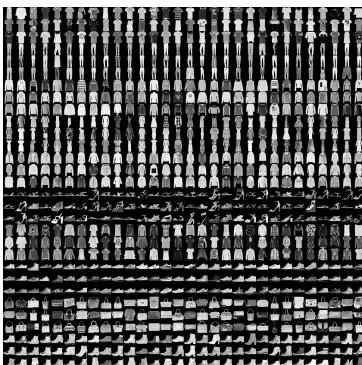
Nội dung

- Phân lớp ảnh với mạng nơ ron
- Nhược điểm của mạng nơ ron truyền thống (FNN) trong xử lý ảnh
- Giới thiệu CNN
- Kiến trúc mạng CNN
- Phân lớp ảnh MNIST với CNN

1

Phân lớp ảnh với mạng nơ ron

- Phân lớp ảnh MNIST Fashion (<https://www.tensorflow.org/tutorials/keras/classification>)
 - 70.000 ảnh xám
 - 10 lớp



2

Phân lớp ảnh với mạng nơ ron

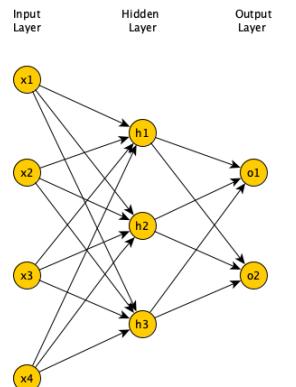
- Phân lớp ảnh MNIST Fashion (<https://www.tensorflow.org/tutorials/keras/classification>)

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

3

Nhược điểm của mạng nơ ron truyền thẳng trong xử lý ảnh

- Trong mạng FNN đầy đủ (Dense), mỗi nút ở tầng t sẽ kết nối với tất cả các nút ở tầng t+1.
 - Vd:
 - Input: ảnh xám 64x64x1 = 4096
 - Tầng ẩn 1: 500 nút => $4.096 \times 500 = 2.048.000$ trọng số!
- Trải dài ảnh 2D (flatten) => mất thông tin không gian



4

Giới thiệu mạng CNN

- Năm 1980, dựa trên đề xuất về cấu trúc phân cấp của các tế bào, Fukushima đề xuất Neocognitron dùng để nhận dạng ký tự Nhật viết tay. Có thể nói Neocognitron là mạng CNN đầu tiên.
- Năm 1989, Yann Lecun đề xuất CNN có thể được huấn luyện bằng thuật toán lan truyền ngược. CNN vượt xa các mô hình khác tại kỳ thi ILSVRC (ImageNet Large Scale Visual Recognition Challenge)
- Các mô hình CNN thắng tại ILSVRC
 - AlexNet (2012), ZFNet (2013), GoogLeNet và VGG (2014), ResNet (2015).

6

Giới thiệu mạng CNN

- Năm 1959, Hubel và Wiesel thực hiện các thí nghiệm để hiểu các thần kinh thị giác của bộ não xử lý các thông tin hình ảnh như thế nào.
 - Chiếu ánh sáng phía trước 1 con mèo và quan sát phản ứng của các tế bào thanh kinh thị giác
 - Một số tế bào phản ứng mạnh khi ánh sáng chiếu theo một góc nào đó => tế bào đơn (simple cells)
 - Một số tế bào khác phản ứng mạnh khi ánh sáng chiếu nhưng không quan tâm tới góc chiếu và dường như khi có sự chuyển động của ánh sáng => tế bào phức (complex cells)
 - Dường các dựa vào phản ứng tín hiệu từ các tế bào đơn và nên một cấu trúc phân cấp
- Nhận giải Nobel vào năm 1981 vì phát hiện này

5

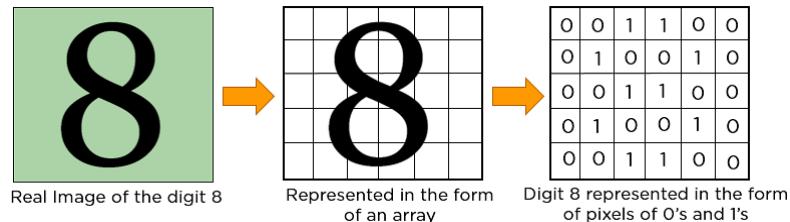
Kiến trúc mạng CNN

- Mạng CNN tiêu biểu gồm có 4 tầng
 - Đầu vào (Input)
 - Tích chập (Convolution)
 - Giảm kích thước (Pooling)
 - Kết nối đầy đủ (Fully connected)

7

Tầng đầu vào

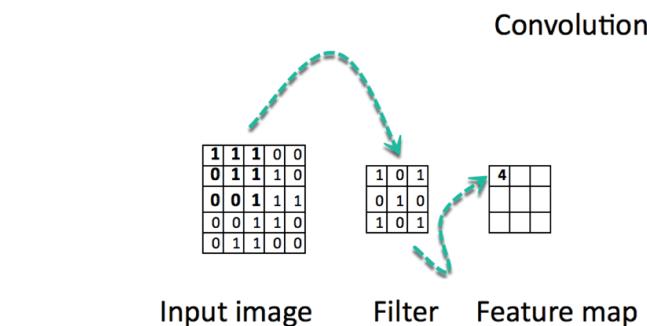
- Ảnh 2D xám hoặc màu
 - Không làm phẳng thành 1D để giữ lại thông tin không gian



8

Tầng tích chập

- Phép tích chập (xem lại trong học phần Xử lý ảnh)



9

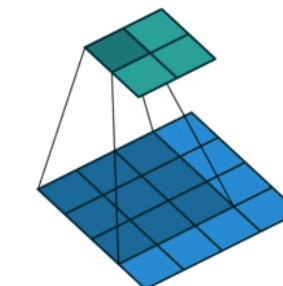
Tầng tích chập

- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - Kích thước mặt nạ (Filter size)
 - Mở rộng đầu vào (Padding)
 - Bước nhảy (Stride)
 - Co (Dilation)
 - Hàm kích hoạt (Activation function)

10

Tầng tích chập

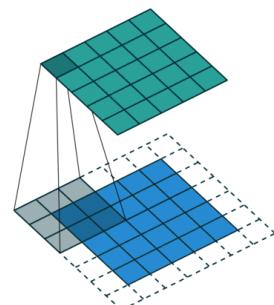
- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - Kích thước mặt nạ (Filter size)**
 - Ảnh 4x4 chập bằng mặt nạ 3x3 => ảnh kết quả 2x2



11

Tâng tích chập

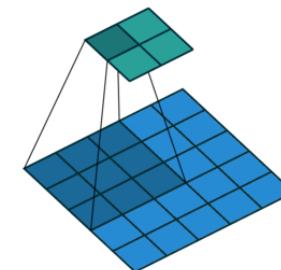
- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - **Mở rộng đầu vào (Padding)**
 - Ảnh 5x5 chập bằng mặt nạ 3x3, mở rộng 1 pixel => kết quả 5x5



12

Tâng tích chập

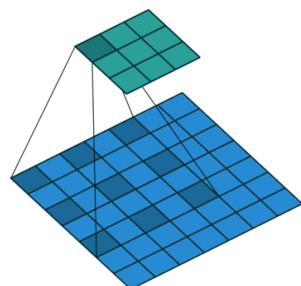
- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - **Bước nhảy (Stride)**
 - 5x5 chập bằng 3x3, bước nhảy 2 => kết quả 2x2



13

Tâng tích chập

- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - **Co (Dilation)**
 - Ảnh 7x7 chập bằng 3x3, co 2 => kết quả 3x3



14

Tâng tích chập

- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - **Hàm kích hoạt (Activation function)**
 - Thông thường là ReLU hoặc biến thể của nó

0	1	-4
0	2	0
-1	4	3

Filter output

0	1	0
0	2	0
0	4	3

Filter output after ReLU

15

Tâng tích chập

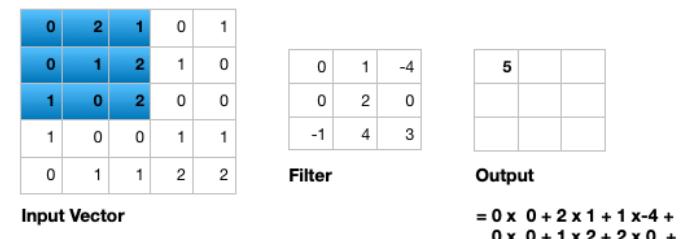
- Phép tích chập (xem lại trong học phần Xử lý ảnh)
 - Kích thước đầu ra

$$\frac{\text{input size} - (\text{filter size} + (\text{filter size} - 1) * (\text{dilation} - 1)) + 2 * \text{padding}}{\text{stride}} + 1$$

16

Tâng tích chập

- Phép tích chập 2D trên đầu vào chỉ có 1 kênh
 - Có thể cộng thêm bias vào tổng



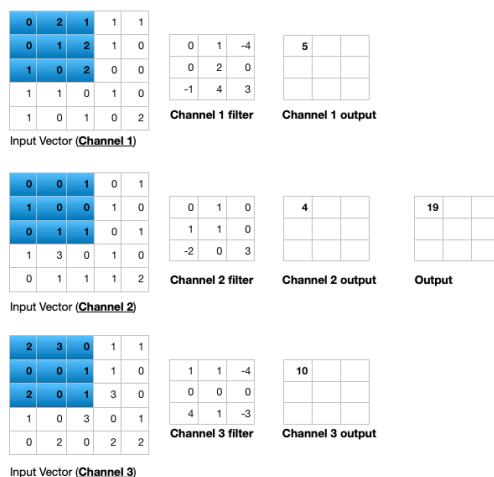
Filter

Output

17

Tâng tích chập

- Phép tích chập 2D trên đầu vào có nhiều kênh
 - Mỗi kênh có 1 mặt nạ khác nhau
 - Có thể cộng thêm bias vào tổng sau cùng



18

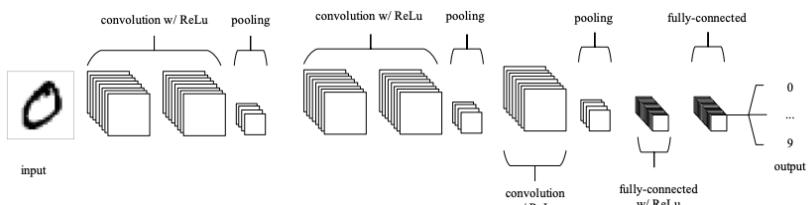
Tâng giảm kích thước (pooling)

- Thực hiện lấy mẫu giảm (giảm kích thước đầu vào), làm giảm số lượng tham số của mô hình, giảm overfitting
 - Lớn nhất: Max Pooling
 - Trung bình: Average Pooling

19

Tầng kết nối đầy đủ

- Liên kết đầu ra của tất cả các nơ ron từ tầng trước đó đến các nơ ron của tầng này, thực hiện phân lớp
 - Thông thường trước, khi đến tầng này, ta thêm tầng làm phẳng (Flatten).



20

Xây dựng mô hình CNN với Keras

- Lớp Conv2D

```
tf.keras.layers.Conv2D(filters,  
                      kernel_size,  
                      strides=(1, 1),  
                      padding="valid",  
                      data_format=None,  
                      dilation_rate=(1, 1),  
                      groups=1,  
                      activation=None,  
                      use_bias=True,  
                      ...  
)
```

21

Xây dựng mô hình CNN với Keras

- Lớp Conv2D
 - Dữ liệu đầu vào
 - 4+D tensor: batch_shape + (channels, rows, cols) if data_format='channels_first'
 - 4+D tensor: batch_shape + (rows, cols, channels) if data_format='channels_last'.
 - Dữ liệu đầu ra
 - 4+D tensor: batch_shape + (filters, new_rows, new_cols) if data_format='channels_first'
 - 4+D tensor: batch_shape + (new_rows, new_cols, filters) if data_format='channels_last'.
 - Số lượng tham số = (số channels*filter size + bias)*số filters

22

Xây dựng mô hình CNN với Keras

- Lớp Conv2D

```
# Đầu vào là ảnh màu RGB kích thước 28x28 với định dạng dữ liệu 'channels_last' và batch size 4.
```

```
input_shape = (4, 28, 28, 3)  
x = tf.random.normal(input_shape)  
y = tf.keras.layers.Conv2D(2, 3, activation='relu',  
                        input_shape=input_shape[1:])(x)  
print(y.shape)
```

Kết quả: (4, 26, 26, 2)

23

Xây dựng mô hình CNN với Keras

- Lớp MaxPooling2D

```
tf.keras.layers.MaxPooling2D(  
    pool_size=(2, 2),  
    strides=None,  
    padding="valid",  
    data_format=None,  
    **kwargs  
)
```

- Số lượng tham số = 0

24

Xây dựng mô hình CNN với Keras

- Lớp MaxPooling2D

Padding="valid"
output_shape = math.floor((input_shape - pool_size) /
strides) + 1 (khi input_shape >= pool_size)

Padding="same"
output_shape = math.floor((input_shape - 1) / strides) + 1

25

Xây dựng mô hình CNN với Keras

- Lớp Flatten

```
tf.keras.layers.Flatten(  
    data_format=None,  
    **kwargs  
)
```

- Số lượng tham số = 0

26

Xây dựng mô hình CNN với Keras

- Lớp Dense

```
tf.keras.layers.Dense(  
    units,  
    activation=None,  
    use_bias=True,  
    kernel_initializer="glorot_uniform",  
    ...  
)
```

- Số lượng tham số = số đầu vào * số units + bias

27

Thực hành 1

- Xây dựng mô hình phân lớp tập dữ liệu ảnh chữ số viết tay MNIST:
https://keras.io/examples/vision/mnist_convnet/

28

Thực hành 2

- Thu thập một tập dữ liệu ảnh màu (tối thiểu 3 lớp, tối thiểu 500 ảnh/lớp)
- Xây dựng một mô hình CNN để phân lớp tập dữ liệu này
 - Thực nghiệm với nhiều kiến trúc mạng khác nhau
 - So sánh độ chính xác trên tập test giữa các kiến trúc mạng khác nhau
 - Soạn slides để báo cáo

29

THANK YOU



30

DEEP LEARNING

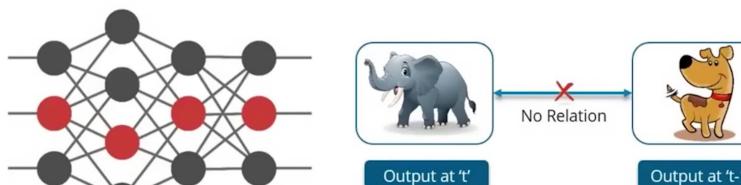
Mạng nơ ron hồi quy (RNN)

Phạm Nguyên Khang
pnkhang@cit.ctu.edu.vn

CAN THO, 22/12/2022

Hạn chế của mạng truyền thẳng

- Mạng truyền thẳng (feedforward network) được dùng để dự đoán nhãn của một tập ảnh ngẫu nhiên
- Kết quả của việc dự đoán lần thứ nhất, không ảnh hưởng đến kết quả dự đoán lần thứ hai



2

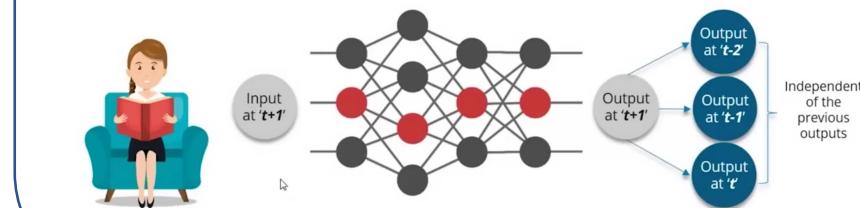
Nội dung

- Hạn chế của mạng truyền thẳng
- Mạng RNN là gì?
- Các vấn đề của mạng RNN
- Suy giảm và bùng nổ gradient
- Mạng Bộ nhớ ngắn hạn dài (LSTM)

1

Hạn chế của mạng truyền thẳng

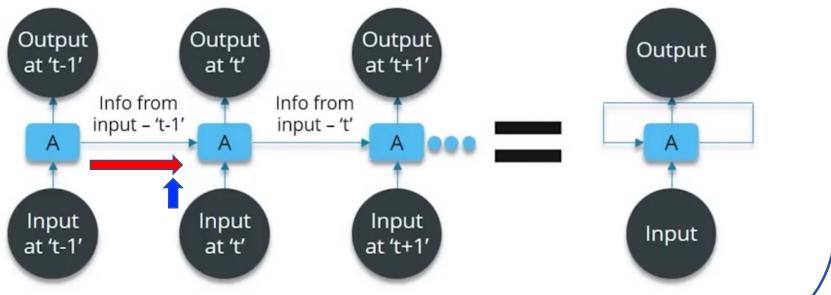
- Khi đọc sách, bạn hiểu được nhờ vào sự hiểu biết của bạn về các từ trước đó
- Không thể dự đoán được từ tiếp theo trong một câu nếu dùng mạng truyền thẳng



3

Giải pháp

- Kết quả dự báo của **đầu ra tại thời điểm $t-1$** kết hợp với **đầu vào tại thời điểm t** được dùng để dự báo kết quả **đầu ra tại thời điểm t** .



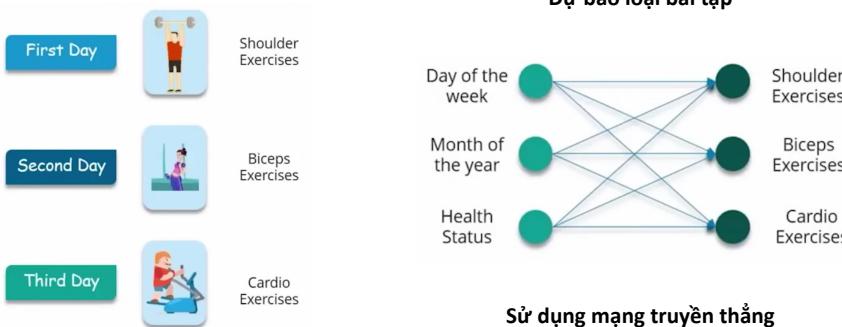
4

Mạng RNN

- Mạng RNN là mạng nơ ron được thiết kế để nhận các mẫu (pattern) trong mỗi chuỗi dữ liệu (sequences of data), như: văn bản, gien, chữ viết tay, giọng nói, time series, giá cổ phiếu, ...
- Ví dụ:
 - Gym PT lập thời khóa biểu tập luyện cho bạn. Các bài tập được lặp lại sau mỗi 3 ngày.

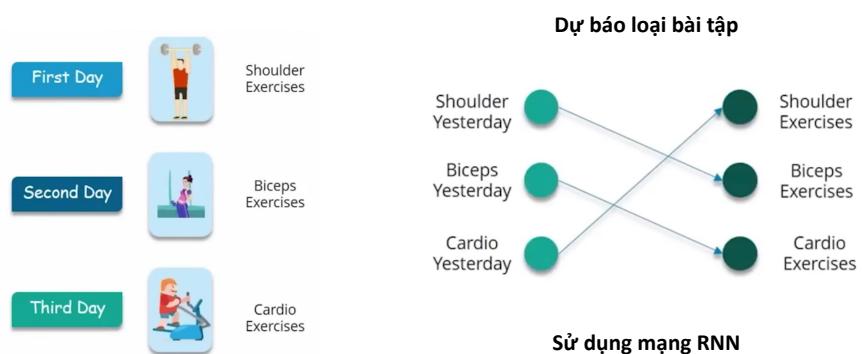
5

Mạng RNN



6

Mạng RNN



7

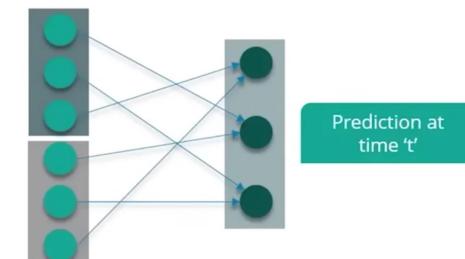
Mạng RNN



8

Mạng RNN

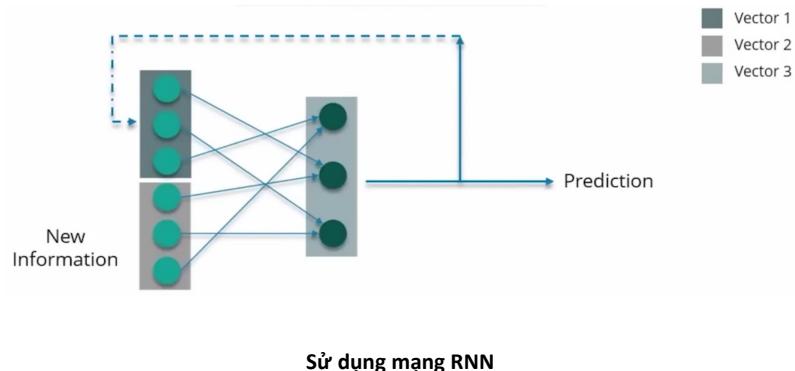
Dự báo loại bài tập



9

Mạng RNN

Dự báo loại bài tập



10

Mạng RNN

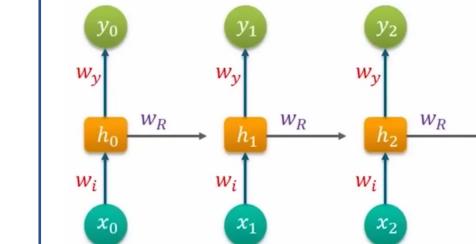
Mỗi nơ-ron/module có

- Trọng số: w_i, w_R, w_y
- Bias: b_h, b_y
- 2 đầu ra: h (ẩn), y (hiện)
- 2 hàm kích hoạt: g_h, g_y

Công thức toán

$$h^{(t)} = g_h (w_i x^{(t)} + w_R h^{(t-1)} + b_h)$$

$$y^{(t)} = g_y (w_y h^{(t)} + b_y)$$



Mô hình RNN

11

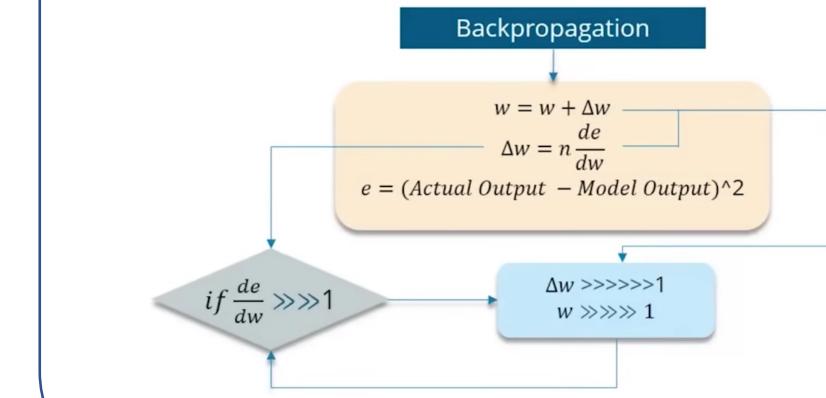
Huấn luyện mạng RNN

- Sử dụng thuật toán lan truyền ngược theo thời gian (Backpropagation Through Time - BTT)
 - Áp dụng lan truyền ngược tại mỗi thời điểm t
- Vấn đề



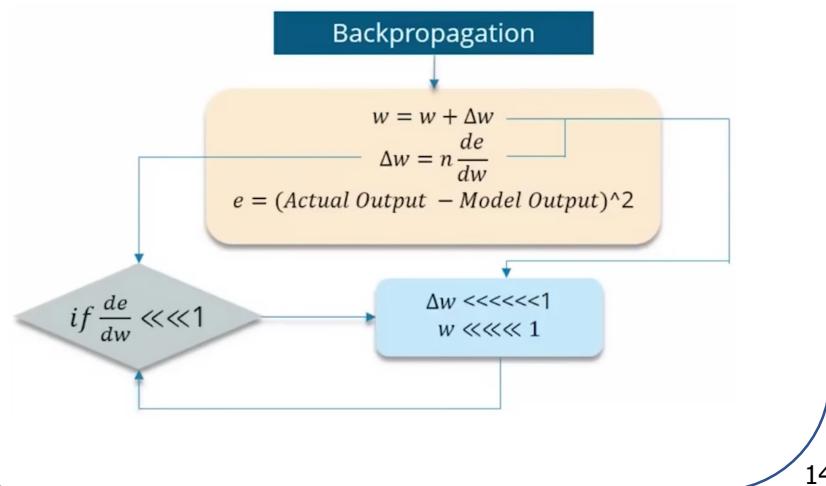
12

Suy giảm gradient



13

Bùng nổ Gradient



14

Giải pháp

Bùng nổ gradient

- Truncated BTT
 - Thay vì sử dụng lan truyền ngược từ thời điểm cuối, ta chọn thời điểm sớm hơn chút, ví dụ sau: 10 thời điểm
- Cắt gradient khi vượt quá ngưỡng
- Dùng RMSprop điều chỉnh tốc độ học

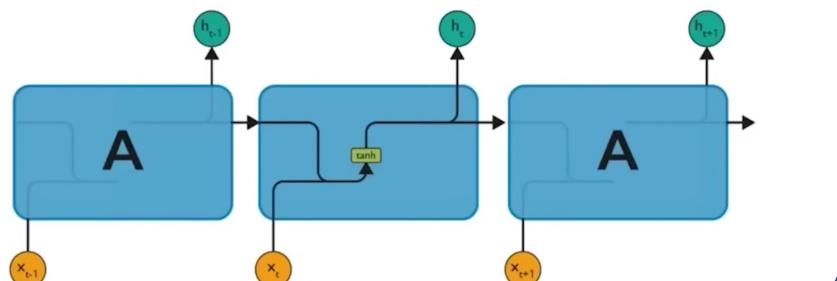
Suy giảm gradient

- Dùng hàm kích hoạt ReLU
 - Gradient = 1
- Cắt gradient khi quá thấp
- Dùng RMSprop
- Sử dụng mạng LTMS hay GRU

15

Mạng LSTM

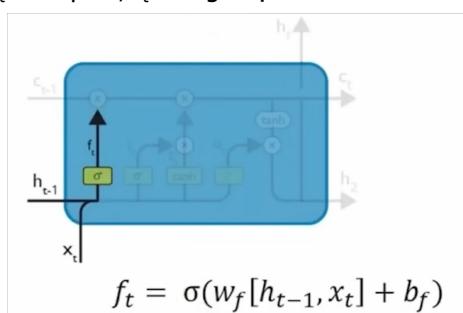
- Long Short-Term Memory (Bộ nhớ ngắn hạn lớn/dài)
 - Có khả năng học được các sự phụ thuộc dài hạn (long-term)



16

Mạng LSTM

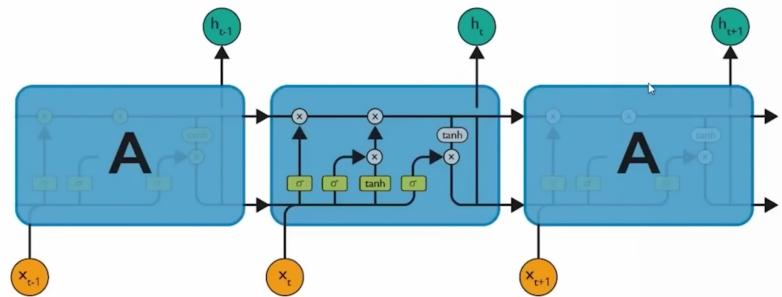
- Bước 1:
 - Xác định thông tin không cần thiết (cần quên đi) bằng cách sử dụng tầng sigmoid (cổng quên)
 - $f_t \sim 0$ quên, $f_t \sim 1$ giữ lại



18

Mạng LSTM

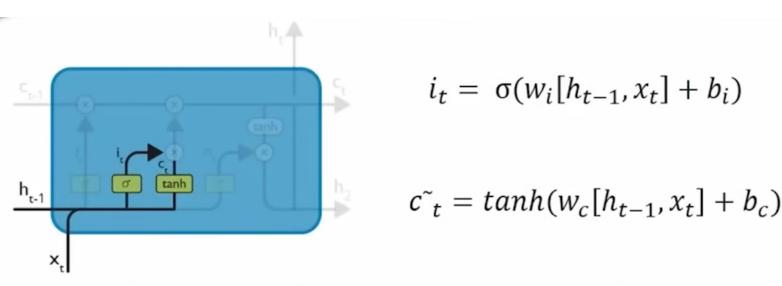
- Long Short-Term Memory (Bộ nhớ ngắn hạn lớn/dài)
 - Có khả năng học được các sự phụ thuộc dài hạn (long-term)



17

Mạng LSTM

- Bước 2:
 - Xác định thông tin cần thiết để lưu lại
 - Dùng 2 tầng: tầng sigmoid (cổng vào) và tầng tanh tạo ra vector ứng viên mới.

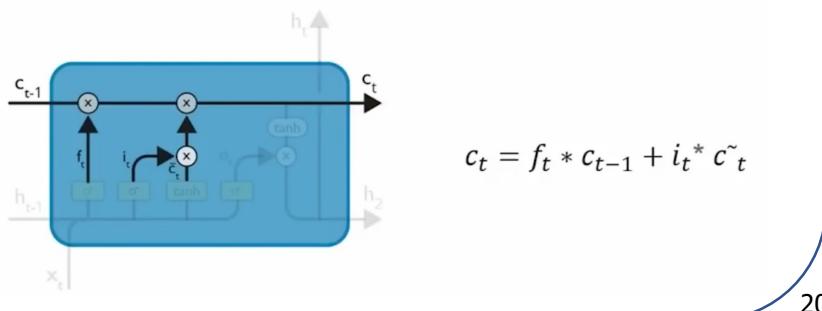


19

Mạng LSTM

- Bước 3:

- Cập nhật trạng thái mới từ trạng thái cũ (c_{t-1}) và ứng viên mới (\tilde{c}_t)

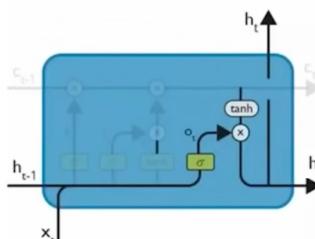


20

Mạng LSTM

- Bước 4:

- Xác định đầu ra h_t
 - Đưa đầu ra trước đó h_{t-1} và đầu vào hiện tại x_t qua cổng sigmoid
 - Nhân kết quả với tanh (trạng thái hiện tại - c_t)



$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(c_t)$$

21

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv

Index	Features	Format
1	Date Time	01.01.2009 00:10:00
2	p (mbar)	996.52
3	T (degC)	-8.02
4	Tpot (K)	265.4
5	Tdew (degC)	-8.9
6	rh (%)	93.3

7	VPmax (mbar)	3.33
8	VPact (mbar)	3.11
9	VPdef (mbar)	0.22
10	sh (g/kg)	1.94
11	H2OC (mmol/mol)	3.12
12	rho (g/m ** 3)	1307.75
13	wv (m/s)	1.03
14	max. wv (m/s)	1.75
15	wd (deg)	152.3

22

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Import các thư viện cần thiết

```
import tensorflow as tf
from keras.layers import LSTM, Dense
import pandas as pd
```

23

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Đọc dữ liệu**

```
csv_file = "/Users/pnkhang/DL/climate.csv"

#Đọc dữ liệu, phân cách bằng dấu phẩy
df = pd.read_csv(csv_file, sep=',')

# Lấy cột nhiệt độ (cột thứ 2)
data = df.iloc[:, [2]].values
print(data)
```

24

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo dữ liệu huấn luyện**

```
#Xác định phương thức dự báo
past = 720
future = 72
step = 6
batch_size = 256
```

```
start = past + future
end = start + train_split

x_train = train_data
y_train = data[start:end]
sequence_length = int(past / step)
```

26

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Chuẩn hóa dữ liệu**

```
#Chuẩn hóa dữ liệu mean = 0, std = 1
def normalize(data, train_split):
    data_mean = data[:train_split].mean(axis=0)
    data_std = data[:train_split].std(axis=0)
    return (data - data_mean) / data_std
```

```
#Chuẩn hóa và tách dữ liệu huấn luyện
train_split = int(0.715 * int(df.shape[0]))

data = normalize(data, train_split)

train_data = data[:train_split]
val_data = data[train_split:]
```

25

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo dữ liệu huấn luyện**

```
#Xác định phương thức dự báo
past = 720
future = 72
step = 6
batch_size = 256
```

```
# Tạo dữ liệu huấn luyện
dataset_train =
tf.keras.preprocessing.timeseries_dataset_from_array(
    x_train,
    y_train,
    sequence_length=sequence_length,
    sampling_rate=step,
    batch_size=batch_size,
)
```

27

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo dữ liệu kiểm chứng**

```
dataset_val =  
tf.keras.preprocessing.timeseries_dataset_from_array(  
    x_val,  
    y_val,  
    sequence_length=sequence_length,  
    sampling_rate=step,  
    batch_size=batch_size,  
)
```

```
x_end = len(val_data) - past - future  
label_start = train_split + past + future  
  
x_val = val_data[:x_end]  
y_val = data[label_start:]
```

28

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo mô hình**

```
model = tf.keras.Sequential()  
model.add(Input(shape=(sequence_length, 1)))  
model.add(LSTM(32))  
model.add(Dense(1))  
  
model.compile(optimizer=tf.keras.optimizers.Adam(  
    learning_rate=learning_rate), loss="mse")  
model.summary()
```

30

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo mô hình**

```
model = tf.keras.Sequential()  
model.add(Input(shape=(sequence_length, 1)))  
model.add(LSTM(32))  
model.add(Dense(1))  
  
model.compile(optimizer=tf.keras.optimizers.Adam(  
    learning_rate=learning_rate), loss="mse")  
model.summary()
```

29

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Tạo mô hình**

```
model = tf.keras.Sequential()  
model.add(Input(shape=(sequence_length, 1)))  
model.add(LSTM(32))  
model.add(Dense(1))  
  
model.compile(optimizer=tf.keras.optimizers.Adam(  
    learning_rate=learning_rate), loss="mse")  
model.summary()
```

Layer (type)	Output Shape	Param #
LSTM (LSTM)	(None, 32)	4352
dense (Dense)	(None, 1)	33
Total params:	4,385	
Trainable params:	4,385	
Non-trainable params:	0	

31

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Huấn luyện**

```
# Huấn luyện
epochs = 10
history = model.fit(
    dataset_train,
    epochs=epochs,
    validation_data=dataset_val,
    callbacks=[es_callback, modelckpt_callback],
)
```

32

Ví dụ 1 – Dự báo nhiệt độ

```
1172/1172 [=====] - ETA: 0s - loss: 0.2153 2023-03-13 12:30:47.934179:
2023-03-13 12:30:47.985474: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:24] Replaced NodeDef with OpDef from external library: Placeholder
1172/1172 [=====] - 61s 51ms/step - loss: 0.2153 - val_loss: 0.1596
Epoch 2/10
1172/1172 [=====] - 58s 50ms/step - loss: 0.1426 - val_loss: 0.1623
Epoch 3/10
1172/1172 [=====] - 59s 50ms/step - loss: 0.1363 - val_loss: 0.1828
Epoch 4/10
1172/1172 [=====] - 59s 51ms/step - loss: 0.1357 - val_loss: 0.1852
Epoch 5/10
1172/1172 [=====] - 59s 50ms/step - loss: 0.1360 - val_loss: 0.1810
Epoch 6/10
1172/1172 [=====] - 60s 51ms/step - loss: 0.1338 - val_loss: 0.1791
Epoch 7/10
1172/1172 [=====] - 60s 52ms/step - loss: 0.1324 - val_loss: 0.1793
Epoch 8/10
1172/1172 [=====] - 61s 52ms/step - loss: 0.1294 - val_loss: 0.1805
Epoch 9/10
1172/1172 [=====] - 60s 51ms/step - loss: 0.1299 - val_loss: 0.1769
Epoch 10/10
1172/1172 [=====] - 61s 52ms/step - loss: 0.1278 - val_loss: 0.1733
```

33

Ví dụ 1 – Dự báo nhiệt độ

- Bài toán: Dự báo nhiệt độ của ngày thứ 6, dựa vào dữ liệu của 5 ngày trước đó
 - Tập dữ liệu: climate.csv
 - Vẽ đồ thị**

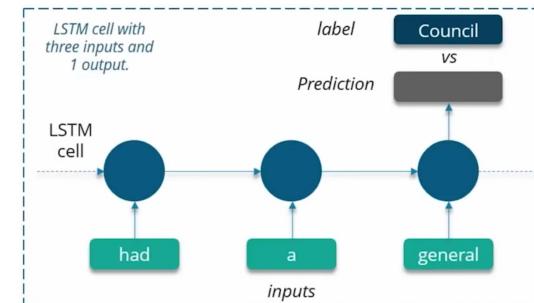
```
def visualize_loss(history, title):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    epochs = range(len(loss))
    plt.figure()
    plt.plot(epochs, loss, "b", label="Training loss")
    plt.plot(epochs, val_loss, "r", label="Validation loss")
    plt.title(title)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()
```

```
visualize_loss(history, "Training and Validation Loss")
```

34

Ví dụ 2 – Sinh từ

- Bài toán
 - Dự báo từ kế tiếp dựa vào 3 từ trước đó



LSTM cell có:
• 3 input (timestep = 3)
• 1 output

35

Ví dụ 2 – Sinh từ

- Huấn luyện
 - Chuẩn bị dữ liệu

Trích từ truyện ngụ ngôn
của Aesop

Có tất cả 112 từ khác nhau.

long ago , the mice had a general council to consider what measures they could take to outwit their common enemy , the cat . some said this , and some said that but at last a young mouse got up and said he had a proposal to make , which he thought would meet the case . you will all agree , said he , that our chief danger consists in the sly and treacherous manner in which the enemy approaches us . now , if we could receive some signal of her approach , we could easily escape from her . i venture , therefore , to propose that a small bell be procured , and attached by a ribbon round the neck of the cat . by this means we should always know when she was about , and could easily retire while she was in the neighborhood . this proposal met with general applause , until an old mouse got up and said that is all very well , but who is to bell the cat ? the mice looked at one another and nobody spoke . then the old mouse said it is easy to propose impossible remedies .

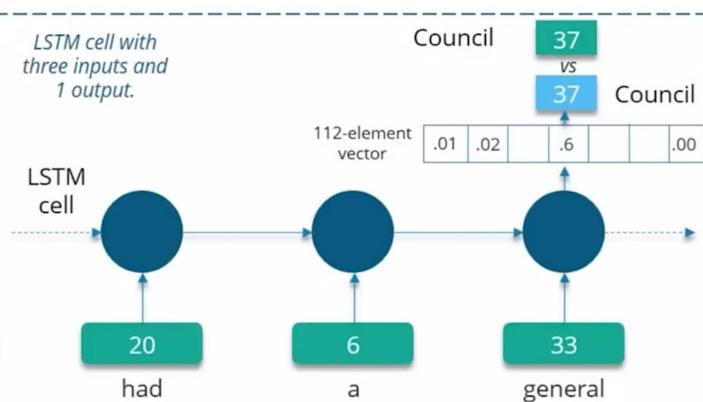
36

Ví dụ 2 – Sinh từ

- Huấn luyện
 - Chuẩn bị dữ liệu
 - Thu thập dữ liệu: 1 đoạn văn bản
 - Gán cho mỗi từ 1 số nguyên (id)

37

Ví dụ 2 – Sinh từ



38

Ví dụ 2 – Sinh từ

- Import

```
import tensorflow as tf  
import numpy as np  
import collections  
from keras.layers import LSTM, Dense
```

39

Ví dụ 2 – Sinh từ

- Hàm đọc file

```
def read_data(fname):
    with open(fname) as f:
        content = f.readlines()
    content = [x.strip() for x in content]
    words = []
    for line in content:
        words.extend(line.split())

    return np.array(words)
```

40

Ví dụ 2 – Sinh từ

- Hàm tạo từ điển

```
def build_dataset(words):
    count = collections.Counter(words).most_common()
    word2id = {}
    for word, freq in count:
        word2id[word] = len(word2id)

    id2word = dict(zip(word2id.values(), word2id.keys()))
    return word2id, id2word
```

41

Ví dụ 2 – Sinh từ

- Đọc dữ liệu và tạo từ điển

```
data = read_data('/Users/pnkhang/DL/toto.txt')
print(data)

w2i, i2w = build_dataset(data)

vocab_size = len(w2i)
timestep = 3
```

42

Ví dụ 2 – Sinh từ

- Tạo dữ liệu huấn luyện

```
X, Y = [], []
for i in range(timestep, len(data)):
    X.append([w2i[data[k]] for k in range(i-timestep, i)])
    Y.append(w2i[data[i]])
```

```
encoded_data = [w2i[x] for x in data]
X = encoded_data[:-1]
Y = encoded_data[timestep:]
```

```
train_data = tf.keras.preprocessing.timeseries_dataset_from_array(
    X, Y, sequence_length=timestep, sampling_rate=1
)
```

43

Ví dụ 2 – Sinh từ

- Tạo mô hình và huấn luyện

```
model = tf.keras.Sequential()  
model.add(LSTM(512, return_sequences=True,  
    input_shape=(timestep, 1)))  
model.add(LSTM(512, return_sequences=False))  
model.add(Dense(vocab_size))  
  
model.summary()
```

```
model.compile(optimizer='adam',  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=['accuracy'])  
  
model.fit(train_data, epochs=500)
```

44

Ví dụ 2 – Sinh từ

- Tạo mô hình và huấn luyện

```
model = tf.keras.Sequential()  
model.add(LSTM(512, return_sequences=True,  
    input_shape=(timestep, 1)))  
model.add(LSTM(512, return_sequences=False))  
model.add(Dense(vocab_size))  
  
model.summary()
```

```
model.compile(optimizer='adam',  
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=['accuracy'])  
  
model.fit(train_data, epochs=500)
```

45

Ví dụ 2 – Sinh từ

- Tạo mô hình và huấn luyện

```
2/2 [=====] - 0s 19ms/step - loss: 0.1252 - accuracy: 0.9261  
Epoch 496/500  
2/2 [=====] - 0s 21ms/step - loss: 0.1260 - accuracy: 0.9489  
Epoch 497/500  
2/2 [=====] - 0s 20ms/step - loss: 0.1105 - accuracy: 0.9375  
Epoch 498/500  
2/2 [=====] - 0s 19ms/step - loss: 0.1145 - accuracy: 0.9659  
Epoch 499/500  
2/2 [=====] - 0s 20ms/step - loss: 0.1070 - accuracy: 0.9489  
Epoch 500/500  
2/2 [=====] - 0s 24ms/step - loss: 0.0999 - accuracy: 0.9602
```

46

Ví dụ 2 – Sinh từ

- Dự báo

```
def encode(sent):  
    return [[w2i[w] for w in sent.split()]]  
  
pred = model.predict(encode("had a general"))  
pred_word = i2w[np.argmax(pred)]  
print(pred_word)  
  
pred = model.predict(encode("a general  
council"))  
pred_word = i2w[np.argmax(pred)]  
print(pred_word)
```

2023-05-15 15:15:11
council
to

47

Thực hành 1

- Làm lại bài tập dự báo thời tiết và bài tập sinh từ

48

Thực hành 3

- Thu tập một tập dữ liệu văn bản lớn hơn. Huấn luyện mô hình.
- Tạo giao diện cho phép nhập vào 3 từ và sinh ra từ 10 từ kế tiếp từ 3 từ nhập vào:
 - Lấy 3 từ nhập vào sinh ra từ thứ 4.
 - Lấy các từ 2, 3, 4 sinh ra từ thứ 5
 - Cứ như thế cho đến từ thứ 10.

50

Thực hành 2

- Mở rộng bài toán dự báo thời tiết với nhiều cột dữ liệu hơn

- p (mbar)
- T (degC)
- rh (%)
- VPact (mbar)
- VPdef (mbar)
- H2OC (mmol/mol)
- rho (g/m ** 3)

```
csv_file = "/Users/pnkhang/DL/climate.csv"
```

```
#Đọc dữ liệu, phân cách bằng dấu phẩy  
df = pd.read_csv(csv_file, sep=',')
```

```
# Lấy cột nhiệt độ (cột thứ 2)  
data = df.iloc[:, [2]].values  
print(data)
```

- Gợi ý: Thay chỗ dữ liệu [2] bằng mảng [2, 3, ...]

49

THANK YOU



51

VIP Cheatsheet: Deep Learning

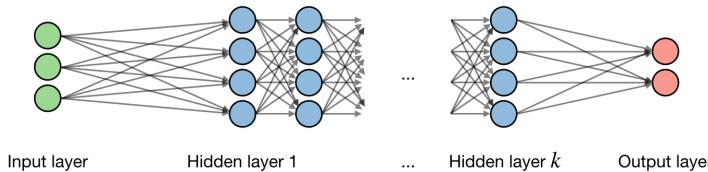
Afshine AMIDI and Shervine AMIDI

September 15, 2018

Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

Architecture – The vocabulary around neural networks architectures is described in the figure below:

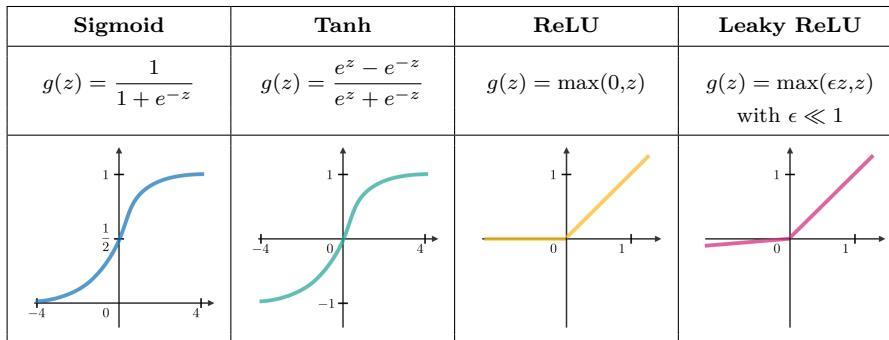


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$z_j^{[i]} = w_j^{[i]T} x + b_j^{[i]}$$

where we note w , b , z the weight, bias and output respectively.

Activation function – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:



Cross-entropy loss – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$L(z,y) = -[y \log(z) + (1-y) \log(1-z)]$$

Learning rate – The learning rate, often noted η , indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

Backpropagation – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}$$

As a result, the weight is updated as follows:

$$w \leftarrow w - \eta \frac{\partial L(z,y)}{\partial w}$$

Updating weights – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

Dropout – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1-p$.

Convolutional Neural Networks

Convolutional layer requirement – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$N = \frac{W - F + 2P}{S} + 1$$

Batch normalization – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Recurrent Neural Networks

□ Types of gates – Here are the different types of gates that we encounter in a typical recurrent neural network:

Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

□ LSTM – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

Reinforcement Learning and Control

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

□ Markov decision processes – A Markov decision process (MDP) is a 5-tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where:

- S is the set of states
- A is the set of actions
- $\{P_{sa}\}$ are the state transition probabilities for $s \in S$ and $a \in A$
- $\gamma \in [0, 1]$ is the discount factor
- $R : S \times A \rightarrow \mathbb{R}$ or $R : S \rightarrow \mathbb{R}$ is the reward function that the algorithm wants to maximize

□ Policy – A policy π is a function $\pi : S \rightarrow A$ that maps states to actions.

Remark: we say that we execute a given policy π if given a state s we take the action $a = \pi(s)$.

□ Value function – For a given policy π and a given state s , we define the value function V^π as follows:

$$V^\pi(s) = E \left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi \right]$$

□ Bellman equation – The optimal Bellman equations characterizes the value function V^{π^*} of the optimal policy π^* :

$$V^{\pi^*}(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi^*}(s')$$

Remark: we note that the optimal policy π^ for a given state s is such that:*

$$\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

□ Value iteration algorithm – The value iteration algorithm is in two steps:

- We initialize the value:

$$V_0(s) = 0$$

- We iterate the value based on the values before:

$$V_{i+1}(s) = R(s) + \max_{a \in A} \left[\sum_{s' \in S} \gamma P_{sa}(s') V_i(s') \right]$$

□ Maximum likelihood estimate – The maximum likelihood estimates for the state transition probabilities are as follows:

$$P_{sa}(s') = \frac{\# \text{times took action } a \text{ in state } s \text{ and got to } s'}{\# \text{times took action } a \text{ in state } s}$$

□ Q-learning – Q-learning is a model-free estimation of Q , which is done as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R(s, a, s') + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

VIP Cheatsheet: Machine Learning Tips

Afshine AMIDI and Shervine AMIDI

September 9, 2018

Metrics

Given a set of data points $\{x^{(1)}, \dots, x^{(m)}\}$, where each $x^{(i)}$ has n features, associated to a set of outcomes $\{y^{(1)}, \dots, y^{(m)}\}$, we want to assess a given classifier that learns how to predict y from x .

Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

Confusion matrix – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:

		Predicted class	
		+	-
Actual class	+	TP True Positives	FN False Negatives Type II error
	-	FP False Positives Type I error	TN True Negatives

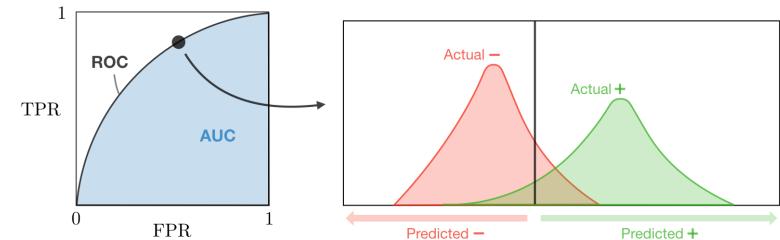
Main metrics – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

ROC – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{TP}{TP + FN}$	Recall, sensitivity
False Positive Rate FPR	$\frac{FP}{TN + FP}$	1-specificity

AUC – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



Regression

Basic metrics – Given a regression model f , the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$SS_{tot} = \sum_{i=1}^m (y_i - \bar{y})^2$	$SS_{reg} = \sum_{i=1}^m (f(x_i) - \bar{y})^2$	$SS_{res} = \sum_{i=1}^m (y_i - f(x_i))^2$

Coefficient of determination – The coefficient of determination, often noted R^2 or r^2 , provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

Main metrics – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{SS_{res} + 2(n+1)\hat{\sigma}^2}{m}$	$2[(n+2) - \log(L)]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

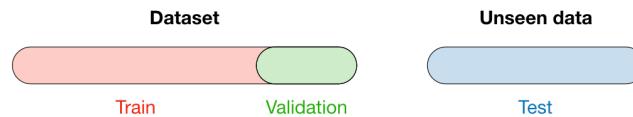
where L is the likelihood and $\hat{\sigma}^2$ is an estimate of the variance associated with each response.

Model selection

Vocabulary – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

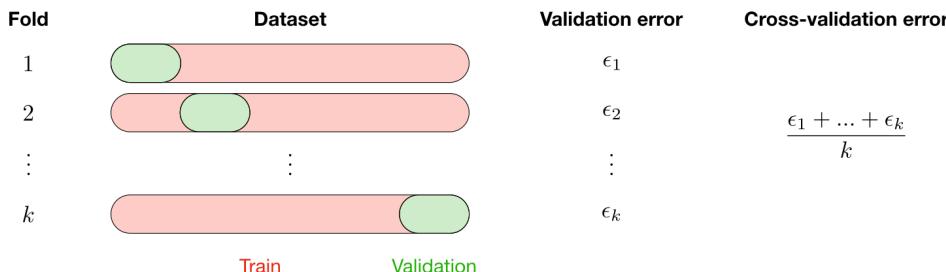
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



Cross-validation – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

<i>k</i> -fold	Leave- <i>p</i> -out
- Training on $k - 1$ folds and assessment on the remaining one - Generally $k = 5$ or 10	- Training on $n - p$ observations and assessment on the p remaining ones - Case $p = 1$ is called leave-one-out

The most commonly used method is called *k*-fold cross-validation and splits the training data into *k* folds to validate the model on one fold while training the model on the $k - 1$ other folds, all of this *k* times. The error is then averaged over the *k* folds and is named cross-validation error.



Regularization – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
$\dots + \lambda \theta _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \theta _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda [(1-\alpha) \theta _1 + \alpha \theta _2^2]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

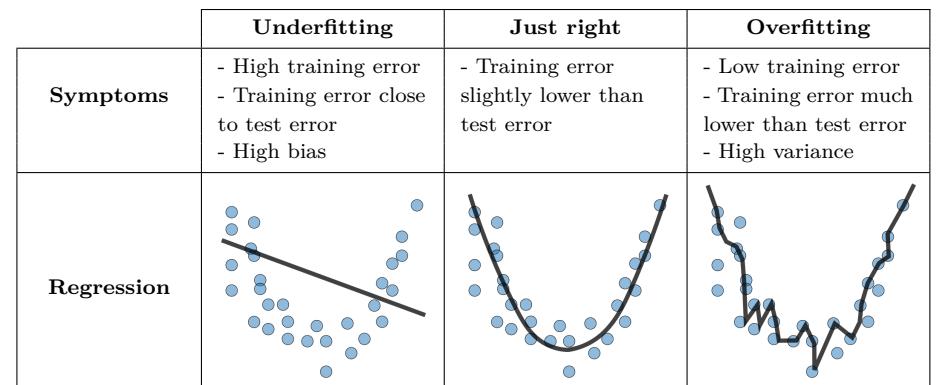
Model selection – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

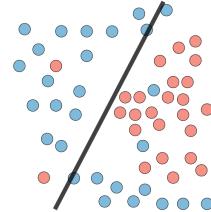
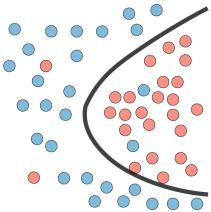
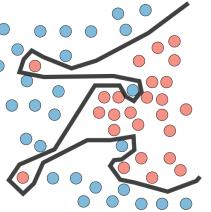
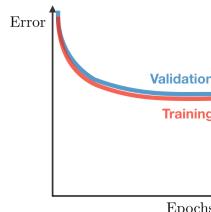
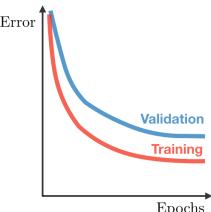
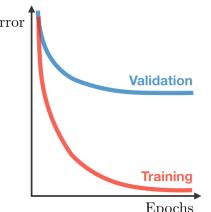
Diagnostics

Bias – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

Variance – The variance of a model is the variability of the model prediction for given data points.

Bias/variance tradeoff – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.



Classification			
Deep learning			
Remedies	<ul style="list-style-type: none"> - Complexify model - Add more features - Train longer 		<ul style="list-style-type: none"> - Regularize - Get more data

❑ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❑ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

VIP Cheatsheet: Supervised Learning

Afshine AMIDI and Shervine AMIDI

September 9, 2018

Introduction to Supervised Learning

Given a set of data points $\{x^{(1)}, \dots, x^{(m)}\}$ associated to a set of outcomes $\{y^{(1)}, \dots, y^{(m)}\}$, we want to build a classifier that learns how to predict y from x .

Type of prediction – The different types of predictive models are summed up in the table below:

	Regression	Classifier
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

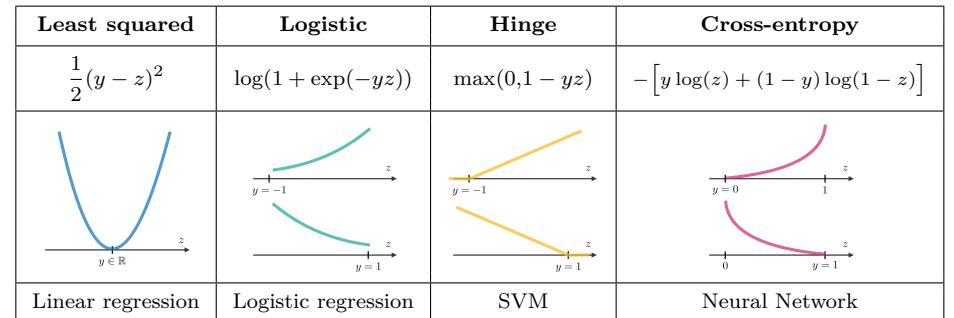
Type of model – The different models are summed up in the table below:

	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Notations and general concepts

Hypothesis – The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$, the model prediction output is $h_\theta(x^{(i)})$.

Loss function – A loss function is a function $L : (z,y) \in \mathbb{R} \times Y \mapsto L(z,y) \in \mathbb{R}$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are. The common loss functions are summed up in the table below:

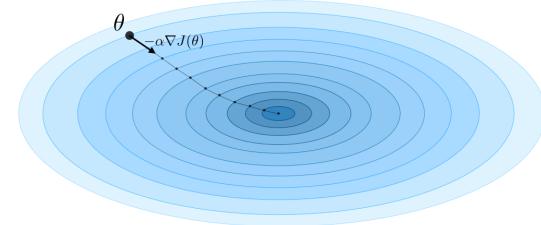


Cost function – The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$J(\theta) = \sum_{i=1}^m L(h_\theta(x^{(i)}), y^{(i)})$$

Gradient descent – By noting $\alpha \in \mathbb{R}$ the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function J as follows:

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$



Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

Likelihood – The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through maximizing the likelihood. In practice, we use the log-likelihood $\ell(\theta) = \log(L(\theta))$ which is easier to optimize. We have:

$$\theta^{\text{opt}} = \arg \max_{\theta} L(\theta)$$

Newton's algorithm – The Newton's algorithm is a numerical method that finds θ such that $\ell'(\theta) = 0$. Its update rule is as follows:

$$\theta \leftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta \leftarrow \theta - (\nabla_{\theta}^2 \ell(\theta))^{-1} \nabla_{\theta} \ell(\theta)$$

Linear regression

We assume here that $y|x; \theta \sim \mathcal{N}(\mu, \sigma^2)$

Normal equations – By noting X the matrix design, the value of θ that minimizes the cost function is a closed-form solution such that:

$$\theta = (X^T X)^{-1} X^T y$$

LMS algorithm – By noting α the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of m data points, which is also known as the Widrow-Hoff learning rule, is as follows:

$$\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^m [y^{(i)} - h_\theta(x^{(i)})] x_j^{(i)}$$

Remark: the update rule is a particular case of the gradient ascent.

LWR – Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by $w^{(i)}(x)$, which is defined with parameter $\tau \in \mathbb{R}$ as:

$$w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

Classification and logistic regression

Sigmoid function – The sigmoid function g , also known as the logistic function, is defined as follows:

$$\forall z \in \mathbb{R}, \quad g(z) = \frac{1}{1 + e^{-z}} \in]0, 1[$$

Logistic regression – We assume here that $y|x; \theta \sim \text{Bernoulli}(\phi)$. We have the following form:

$$\phi = p(y=1|x; \theta) = \frac{1}{1 + \exp(-\theta^T x)} = g(\theta^T x)$$

Remark: there is no closed form solution for the case of logistic regressions.

Softmax regression – A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, we set $\theta_K = 0$, which makes the Bernoulli parameter ϕ_i of each class i equal to:

$$\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^K \exp(\theta_j^T x)}$$

Generalized Linear Models

Exponential family – A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter, also called the canonical parameter or link function, η , a sufficient statistic $T(y)$ and a log-partition function $a(\eta)$ as follows:

$$p(y; \eta) = b(y) \exp(\eta T(y) - a(\eta))$$

Remark: we will often have $T(y) = y$. Also, $\exp(-a(\eta))$ can be seen as a normalization parameter that will make sure that the probabilities sum to one.

Here are the most common exponential distributions summed up in the following table:

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1 + \exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^η	$\frac{1}{y!}$
Geometric	$\log(1 - \phi)$	y	$\log\left(\frac{e^\eta}{1-e^\eta}\right)$	1

Assumptions of GLMs – Generalized Linear Models (GLM) aim at predicting a random variable y as a function of $x \in \mathbb{R}^{n+1}$ and rely on the following 3 assumptions:

$$(1) \quad y|x; \theta \sim \text{ExpFamily}(\eta) \quad (2) \quad h_\theta(x) = E[y|x; \theta] \quad (3) \quad \eta = \theta^T x$$

Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

Support Vector Machines

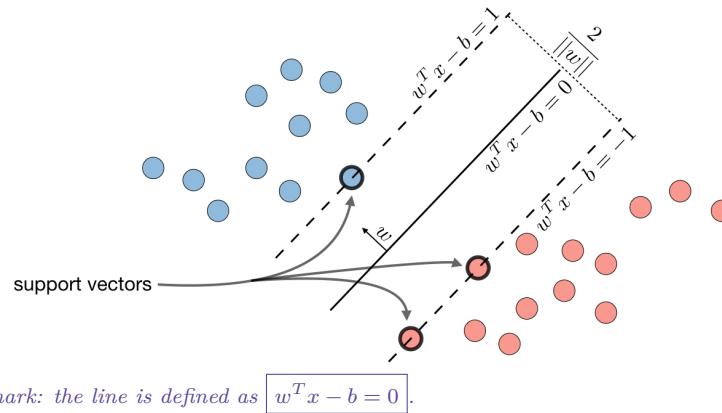
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

Optimal margin classifier – The optimal margin classifier h is such that:

$$h(x) = \text{sign}(w^T x - b)$$

where $(w, b) \in \mathbb{R}^n \times \mathbb{R}$ is the solution of the following optimization problem:

$$\min \frac{1}{2} \|w\|^2 \quad \text{such that} \quad y^{(i)}(w^T x^{(i)} - b) \geq 1$$



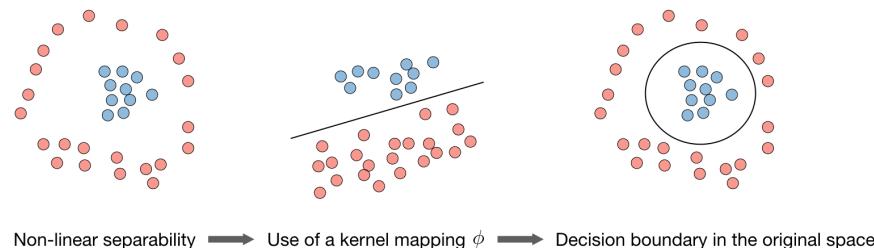
Hinge loss – The hinge loss is used in the setting of SVMs and is defined as follows:

$$L(z,y) = [1 - yz]_+ = \max(0, 1 - yz)$$

Kernel – Given a feature mapping ϕ , we define the kernel K to be defined as:

$$K(x,z) = \phi(x)^T \phi(z)$$

In practice, the kernel K defined by $K(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used.



Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ , which is often very complicated. Instead, only the values $K(x,z)$ are needed.

Lagrangian – We define the Lagrangian $\mathcal{L}(w,b)$ as follows:

$$\mathcal{L}(w,b) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

Remark: the coefficients β_i are called the Lagrange multipliers.

Generative Learning

A generative model first tries to learn how the data is generated by estimating $P(x|y)$, which we can then use to estimate $P(y|x)$ by using Bayes' rule.

Gaussian Discriminant Analysis

Setting – The Gaussian Discriminant Analysis assumes that y and $x|y = 0$ and $x|y = 1$ are such that:

$$y \sim \text{Bernoulli}(\phi)$$

$$x|y = 0 \sim \mathcal{N}(\mu_0, \Sigma) \quad \text{and} \quad x|y = 1 \sim \mathcal{N}(\mu_1, \Sigma)$$

Estimation – The following table sums up the estimates that we find when maximizing the likelihood:

$\hat{\phi}$	$\hat{\mu}_j \quad (j = 0, 1)$	$\hat{\Sigma}$
$\frac{1}{m} \sum_{i=1}^m 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^m 1_{\{y^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$

Naive Bayes

Assumption – The Naive Bayes model supposes that the features of each data point are all independent:

$$P(x|y) = P(x_1, x_2, \dots | y) = P(x_1|y)P(x_2|y)\dots = \prod_{i=1}^n P(x_i|y)$$

Solutions – Maximizing the log-likelihood gives the following solutions, with $k \in \{0, 1\}$, $l \in \llbracket 1, L \rrbracket$

$$P(y = k) = \frac{1}{m} \times \#\{j | y^{(j)} = k\}$$

$$\text{and} \quad P(x_i = l | y = k) = \frac{\#\{j | y^{(j)} = k \text{ and } x_i^{(j)} = l\}}{\#\{j | y^{(j)} = k\}}$$

Remark: Naive Bayes is widely used for text classification and spam detection.

Tree-based and ensemble methods

These methods can be used for both regression and classification problems.

CART – Classification and Regression Trees (CART), commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

Random forest – It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

Remark: random forests are a type of ensemble methods.

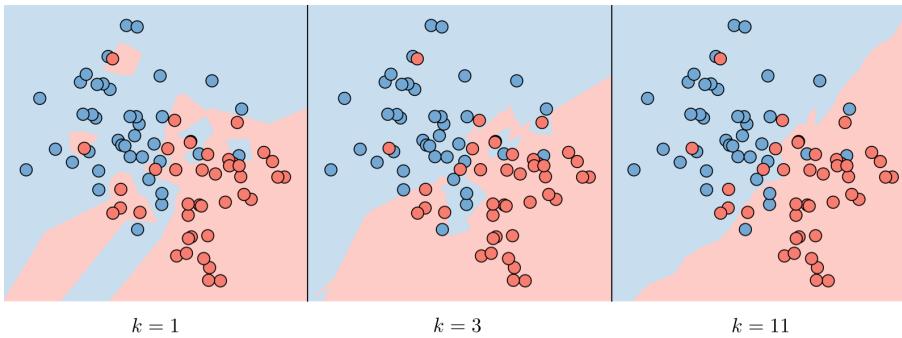
Boosting – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

Adaptive boosting	Gradient boosting
<ul style="list-style-type: none"> - High weights are put on errors to improve at the next boosting step - Known as Adaboost 	<ul style="list-style-type: none"> - Weak learners trained on remaining errors

Other non-parametric approaches

□ **k-nearest neighbors** – The k -nearest neighbors algorithm, commonly known as k -NN, is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.

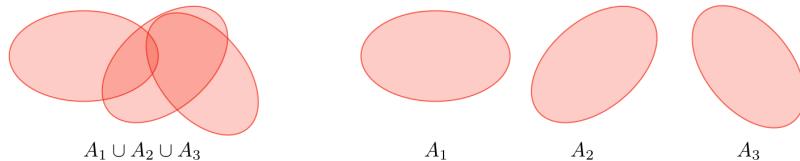
Remark: The higher the parameter k , the higher the bias, and the lower the parameter k , the higher the variance.



Learning Theory

□ **Union bound** – Let A_1, \dots, A_k be k events. We have:

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k)$$



□ **Hoeffding inequality** – Let Z_1, \dots, Z_m be m iid variables drawn from a Bernoulli distribution of parameter ϕ . Let $\hat{\phi}$ be their sample mean and $\gamma > 0$ fixed. We have:

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 m)$$

Remark: this inequality is also known as the Chernoff bound.

□ **Training error** – For a given classifier h , we define the training error $\hat{\epsilon}(h)$, also known as the empirical risk or empirical error, to be as follows:

$$\hat{\epsilon}(h) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{\{h(x^{(i)}) \neq y^{(i)}\}}$$

□ **Probably Approximately Correct (PAC)** – PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:

- the training and testing sets follow the same distribution
- the training examples are drawn independently

□ **Shattering** – Given a set $S = \{x^{(1)}, \dots, x^{(d)}\}$, and a set of classifiers \mathcal{H} , we say that \mathcal{H} shatters S if for any set of labels $\{y^{(1)}, \dots, y^{(d)}\}$, we have:

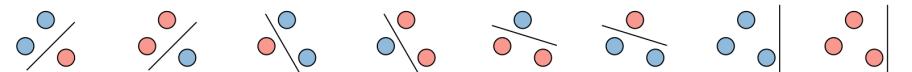
$$\exists h \in \mathcal{H}, \quad \forall i \in [1, d], \quad h(x^{(i)}) = y^{(i)}$$

□ **Upper bound theorem** – Let \mathcal{H} be a finite hypothesis class such that $|\mathcal{H}| = k$ and let δ and the sample size m be fixed. Then, with probability of at least $1 - \delta$, we have:

$$\hat{\epsilon}(h) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h) \right) + 2 \sqrt{\frac{1}{2m} \log \left(\frac{2k}{\delta} \right)}$$

□ **VC dimension** – The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class \mathcal{H} , noted $\text{VC}(\mathcal{H})$ is the size of the largest set that is shattered by \mathcal{H} .

Remark: the VC dimension of $\mathcal{H} = \{\text{set of linear classifiers in 2 dimensions}\}$ is 3.



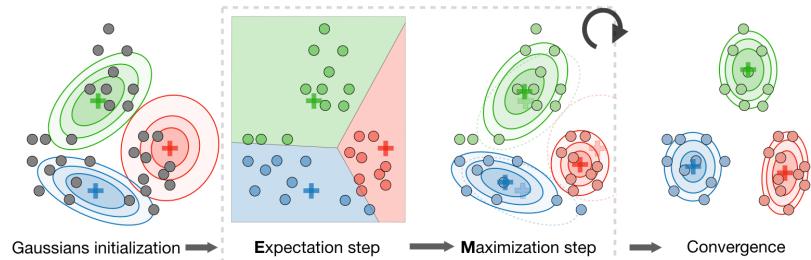
□ **Theorem (Vapnik)** – Let \mathcal{H} be given, with $\text{VC}(\mathcal{H}) = d$ and m the number of training examples. With probability at least $1 - \delta$, we have:

$$\hat{\epsilon}(h) \leq \left(\min_{h \in \mathcal{H}} \epsilon(h) \right) + O \left(\sqrt{\frac{d}{m} \log \left(\frac{m}{d} \right)} + \frac{1}{m} \log \left(\frac{1}{\delta} \right) \right)$$

VIP Cheatsheet: Unsupervised Learning

Afshine AMIDI and Shervine AMIDI

September 9, 2018



Introduction to Unsupervised Learning

Motivation – The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x^{(1)}, \dots, x^{(m)}\}$.

Jensen's inequality – Let f be a convex function and X a random variable. We have the following inequality:

$$E[f(X)] \geq f(E[X])$$

Expectation-Maximization

Latent variables – Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted z . Here are the most common settings where there are latent variables:

Setting	Latent variable z	$x z$	Comments
Mixture of k Gaussians	Multinomial(ϕ)	$\mathcal{N}(\mu_j, \Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0, I)$	$\mathcal{N}(\mu + \Lambda z, \psi)$	$\mu_j \in \mathbb{R}^n$

Algorithm – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $Q_i(z^{(i)})$ that each data point $x^{(i)}$ came from a particular cluster $z^{(i)}$ as follows:

$$Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)}; \theta)$$

- M-step: Use the posterior probabilities $Q_i(z^{(i)})$ as cluster specific weights on data points $x^{(i)}$ to separately re-estimate each cluster model as follows:

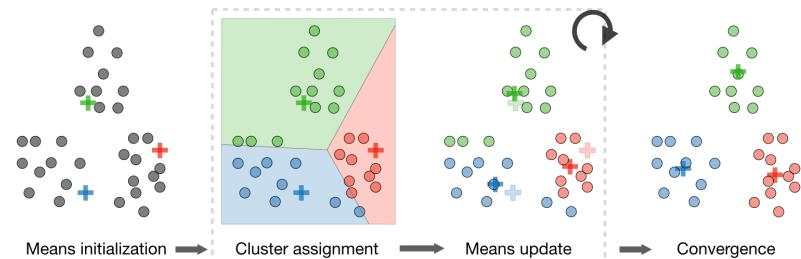
$$\theta_i = \operatorname{argmax}_{\theta} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log \left(\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right) dz^{(i)}$$

k-means clustering

We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j .

Algorithm – After randomly initializing the cluster centroids $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$, the k -means algorithm repeats the following step until convergence:

$$c^{(i)} = \arg \min_j \|x^{(i)} - \mu_j\|^2 \quad \text{and} \quad \mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}$$



Distortion function – In order to see if the algorithm converges, we look at the distortion function defined as follows:

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Hierarchical clustering

Algorithm – It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

Types – There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

□ **Silhouette coefficient** – By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$s = \frac{b - a}{\max(a, b)}$$

□ **Calinski-Harabaz index** – By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matrices respectively defined as

$$B_k = \sum_{j=1}^k n_{c(i)} (\mu_{c(i)} - \mu) (\mu_{c(i)} - \mu)^T, \quad W_k = \sum_{i=1}^m (x^{(i)} - \mu_{c(i)}) (x^{(i)} - \mu_{c(i)})^T$$

the Calinski-Harabaz index $s(k)$ indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1}$$

Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

□ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n \times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \setminus \{0\}$, called eigenvector, such that we have:

$$Az = \lambda z$$

□ **Spectral theorem** – Let $A \in \mathbb{R}^{n \times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n \times n}$. By noting $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

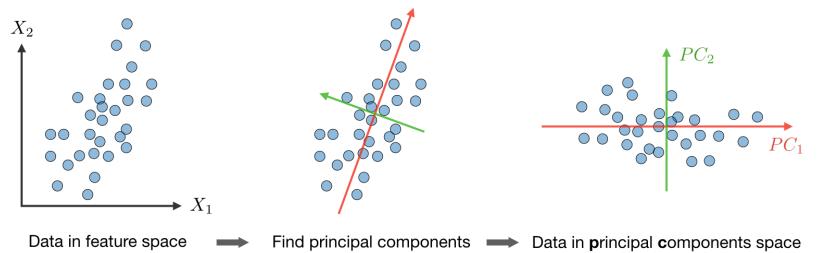
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A .

□ **Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:

- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad \text{where} \quad \mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \text{and} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

- Step 2: Compute $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \in \mathbb{R}^{n \times n}$, which is symmetric with real eigenvalues.
- Step 3: Compute $u_1, \dots, u_k \in \mathbb{R}^n$ the k orthogonal principal eigenvectors of Σ , i.e. the orthogonal eigenvectors of the k largest eigenvalues.
- Step 4: Project the data on $\text{span}_{\mathbb{R}}(u_1, \dots, u_k)$. This procedure maximizes the variance among all k -dimensional spaces.



Independent component analysis

It is a technique meant to find the underlying generating sources.

□ **Assumptions** – We assume that our data x has been generated by the n -dimensional source vector $s = (s_1, \dots, s_n)$, where s_i are independent random variables, via a mixing and non-singular matrix A as follows:

$$x = As$$

The goal is to find the unmixing matrix $W = A^{-1}$ by an update rule.

□ **Bell and Sejnowski ICA algorithm** – This algorithm finds the unmixing matrix W by following the steps below:

- Write the probability of $x = As = W^{-1}s$ as:

$$p(x) = \prod_{i=1}^n p_s(w_i^T x) \cdot |W|$$

- Write the log likelihood given our training data $\{x^{(i)}, i \in [1, m]\}$ and by noting g the sigmoid function as:

$$l(W) = \sum_{i=1}^m \left(\sum_{j=1}^n \log \left(g'(w_j^T x^{(i)}) \right) + \log |W| \right)$$

Therefore, the stochastic gradient ascent learning rule is such that for each training example $x^{(i)}$, we update W as follows:

$$W \leftarrow W + \alpha \begin{pmatrix} \begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)T} + (W^T)^{-1} \end{pmatrix}$$

VIP Refresher: Linear Algebra and Calculus

Afshin AMIDI and Shervine AMIDI

October 6, 2018

General notations

Vector – We note $x \in \mathbb{R}^n$ a vector with n entries, where $x_i \in \mathbb{R}$ is the i^{th} entry:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

Matrix – We note $A \in \mathbb{R}^{m \times n}$ a matrix with m rows and n columns, where $A_{i,j} \in \mathbb{R}$ is the entry located in the i^{th} row and j^{th} column:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Remark: the vector x defined above can be viewed as a $n \times 1$ matrix and is more particularly called a column-vector.

Identity matrix – The identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones in its diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remark: for all matrices $A \in \mathbb{R}^{n \times n}$, we have $A \times I = I \times A = A$.

Diagonal matrix – A diagonal matrix $D \in \mathbb{R}^{n \times n}$ is a square matrix with nonzero values in its diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

Remark: we also note D as $\text{diag}(d_1, \dots, d_n)$.

Matrix operations

Vector-vector multiplication – There are two types of vector-vector products:

- inner product: for $x, y \in \mathbb{R}^n$, we have:

$$x^T y = \sum_{i=1}^n x_i y_i \in \mathbb{R}$$

- outer product: for $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, we have:

$$xy^T = \begin{pmatrix} x_1 y_1 & \cdots & x_1 y_n \\ \vdots & & \vdots \\ x_m y_1 & \cdots & x_m y_n \end{pmatrix} \in \mathbb{R}^{m \times n}$$

Matrix-vector multiplication – The product of matrix $A \in \mathbb{R}^{m \times n}$ and vector $x \in \mathbb{R}^n$ is a vector of size \mathbb{R}^m , such that:

$$Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^n a_{c,i} x_i \in \mathbb{R}^m$$

where $a_{r,i}^T$ are the vector rows and $a_{c,j}$ are the vector columns of A , and x_i are the entries of x .

Matrix-matrix multiplication – The product of matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is a matrix of size $\mathbb{R}^{m \times p}$, such that:

$$AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^n a_{c,i} b_{r,i}^T \in \mathbb{R}^{m \times p}$$

where $a_{r,i}^T, b_{r,i}^T$ are the vector rows and $a_{c,j}, b_{c,j}$ are the vector columns of A and B respectively.

Transpose – The transpose of a matrix $A \in \mathbb{R}^{m \times n}$, noted A^T , is such that its entries are flipped:

$$\forall i, j, \quad A_{i,j}^T = A_{j,i}$$

Remark: for matrices A, B , we have $(AB)^T = B^T A^T$.

Inverse – The inverse of an invertible square matrix A is noted A^{-1} and is the only matrix such that:

$$AA^{-1} = A^{-1}A = I$$

Remark: not all square matrices are invertible. Also, for matrices A, B , we have $(AB)^{-1} = B^{-1}A^{-1}$

Trace – The trace of a square matrix A , noted $\text{tr}(A)$, is the sum of its diagonal entries:

$$\text{tr}(A) = \sum_{i=1}^n A_{i,i}$$

Remark: for matrices A, B , we have $\text{tr}(A^T) = \text{tr}(A)$ and $\text{tr}(AB) = \text{tr}(BA)$

Determinant – The determinant of a square matrix $A \in \mathbb{R}^{n \times n}$, noted $|A|$ or $\det(A)$ is expressed recursively in terms of $A_{\setminus i, \setminus j}$, which is the matrix A without its i^{th} row and j^{th} column, as follows:

$$\det(A) = |A| = \sum_{j=1}^n (-1)^{i+j} A_{i,j} |A_{\setminus i, \setminus j}|$$

Remark: A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

Matrix properties

□ **Symmetric decomposition** – A given matrix A can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}$$

□ **Norm** – A norm is a function $N : V \rightarrow [0, +\infty[$ where V is a vector space, and such that for all $x, y \in V$, we have:

- $N(x + y) \leq N(x) + N(y)$
- $N(ax) = |a|N(x)$ for a scalar
- if $N(x) = 0$, then $x = 0$

For $x \in V$, the most commonly used norms are summed up in the table below:

Norm	Notation	Definition	Use case
Manhattan, L^1	$\ x\ _1$	$\sum_{i=1}^n x_i $	LASSO regularization
Euclidean, L^2	$\ x\ _2$	$\sqrt{\sum_{i=1}^n x_i^2}$	Ridge regularization
p -norm, L^p	$\ x\ _p$	$\left(\sum_{i=1}^n x_i^p\right)^{\frac{1}{p}}$	Hölder inequality
Infinity, L^∞	$\ x\ _\infty$	$\max_i x_i $	Uniform convergence

□ **Linearly dependence** – A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others.

Remark: if no vector can be written this way, then the vectors are said to be linearly independent.

□ **Matrix rank** – The rank of a given matrix A is noted $\text{rank}(A)$ and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of A .

□ **Positive semi-definite matrix** – A matrix $A \in \mathbb{R}^{n \times n}$ is positive semi-definite (PSD) and is noted $A \succeq 0$ if we have:

$$A = A^T \quad \text{and} \quad \forall x \in \mathbb{R}^n, \quad x^T A x \geq 0$$

Remark: similarly, a matrix A is said to be positive definite, and is noted $A \succ 0$, if it is a PSD matrix which satisfies for all non-zero vector x , $x^T A x > 0$.

□ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n \times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \setminus \{0\}$, called eigenvector, such that we have:

$$Az = \lambda z$$

□ **Spectral theorem** – Let $A \in \mathbb{R}^{n \times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n \times n}$. By noting $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$, we have:

$$\exists \Lambda \text{ diagonal}, \quad A = U \Lambda U^T$$

□ **Singular-value decomposition** – For a given matrix A of dimensions $m \times n$, the singular-value decomposition (SVD) is a factorization technique that guarantees the existence of U $m \times m$ unitary, Σ $m \times n$ diagonal and V $n \times n$ unitary matrices, such that:

$$A = U \Sigma V^T$$

Matrix calculus

□ **Gradient** – Let $f : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$ be a function and $A \in \mathbb{R}^{m \times n}$ be a matrix. The gradient of f with respect to A is a $m \times n$ matrix, noted $\nabla_A f(A)$, such that:

$$\left(\nabla_A f(A) \right)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}$$

Remark: the gradient of f is only defined when f is a function that returns a scalar.

□ **Hessian** – Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be a function and $x \in \mathbb{R}^n$ be a vector. The hessian of f with respect to x is a $n \times n$ symmetric matrix, noted $\nabla_x^2 f(x)$, such that:

$$\left(\nabla_x^2 f(x) \right)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

Remark: the hessian of f is only defined when f is a function that returns a scalar.

□ **Gradient operations** – For matrices A, B, C , the following gradient properties are worth having in mind:

$$\nabla_A \text{tr}(AB) = B^T$$

$$\nabla_A f(A) = (\nabla_A f(A))^T$$

$$\nabla_A \text{tr}(ABA^T C) = CAB + C^T AB^T$$

$$\nabla_A |A| = |A|(A^{-1})^T$$

VIP Refresher: Probabilities and Statistics

Afshine AMIDI and Shervine AMIDI

August 6, 2018

Introduction to Probability and Combinatorics

Sample space – The set of all possible outcomes of an experiment is known as the sample space of the experiment and is denoted by S .

Event – Any subset E of the sample space is known as an event. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in E , then we say that E has occurred.

Axioms of probability – For each event E , we denote $P(E)$ as the probability of event E occurring. By noting E_1, \dots, E_n mutually exclusive events, we have the 3 following axioms:

$$(1) \quad 0 \leq P(E) \leq 1 \quad (2) \quad P(S) = 1 \quad (3) \quad P\left(\bigcup_{i=1}^n E_i\right) = \sum_{i=1}^n P(E_i)$$

Permutation – A permutation is an arrangement of r objects from a pool of n objects, in a given order. The number of such arrangements is given by $P(n, r)$, defined as:

$$P(n, r) = \frac{n!}{(n - r)!}$$

Combination – A combination is an arrangement of r objects from a pool of n objects, where the order does not matter. The number of such arrangements is given by $C(n, r)$, defined as:

$$C(n, r) = \frac{P(n, r)}{r!} = \frac{n!}{r!(n - r)!}$$

Remark: we note that for $0 \leq r \leq n$, we have $P(n, r) \geq C(n, r)$.

Conditional Probability

Bayes' rule – For events A and B such that $P(B) > 0$, we have:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Remark: we have $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$.

Partition – Let $\{A_i, i \in [1, n]\}$ be such that for all i , $A_i \neq \emptyset$. We say that $\{A_i\}$ is a partition if we have:

$$\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^n A_i = S$$

Remark: for any event B in the sample space, we have $P(B) = \sum_{i=1}^n P(B|A_i)P(A_i)$.

Extended form of Bayes' rule – Let $\{A_i, i \in [1, n]\}$ be a partition of the sample space. We have:

$$P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^n P(B|A_i)P(A_i)}$$

Independence – Two events A and B are independent if and only if we have:

$$P(A \cap B) = P(A)P(B)$$

Random Variables

Random variable – A random variable, often noted X , is a function that maps every element in a sample space to a real line.

Cumulative distribution function (CDF) – The cumulative distribution function F , which is monotonically non-decreasing and is such that $\lim_{x \rightarrow -\infty} F(x) = 0$ and $\lim_{x \rightarrow +\infty} F(x) = 1$, is defined as:

$$F(x) = P(X \leq x)$$

Remark: we have $P(a < X \leq b) = F(b) - F(a)$.

Probability density function (PDF) – The probability density function f is the probability that X takes on values between two adjacent realizations of the random variable.

Relationships involving the PDF and CDF – Here are the important properties to know in the discrete (D) and the continuous (C) cases.

Case	CDF F	PDF f	Properties of PDF
(D)	$F(x) = \sum_{x_i \leq x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leq f(x_j) \leq 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^x f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geq 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

Variance – The variance of a random variable, often noted $\text{Var}(X)$ or σ^2 , is a measure of the spread of its distribution function. It is determined as follows:

$$\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$$

Standard deviation – The standard deviation of a random variable, often noted σ , is a measure of the spread of its distribution function which is compatible with the units of the actual random variable. It is determined as follows:

$$\sigma = \sqrt{\text{Var}(X)}$$

Expectation and Moments of the Distribution – Here are the expressions of the expected value $E[X]$, generalized expected value $E[g(X)]$, k^{th} moment $E[X^k]$ and characteristic function $\psi(\omega)$ for the discrete and continuous cases:

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	$\psi(\omega)$
(D)	$\sum_{i=1}^n x_i f(x_i)$	$\sum_{i=1}^n g(x_i) f(x_i)$	$\sum_{i=1}^n x_i^k f(x_i)$	$\sum_{i=1}^n f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} x f(x) dx$	$\int_{-\infty}^{+\infty} g(x) f(x) dx$	$\int_{-\infty}^{+\infty} x^k f(x) dx$	$\int_{-\infty}^{+\infty} f(x) e^{i\omega x} dx$

Remark: we have $e^{i\omega x} = \cos(\omega x) + i \sin(\omega x)$.

Revisiting the k^{th} moment – The k^{th} moment can also be computed with the characteristic function as follows:

$$E[X^k] = \frac{1}{i^k} \left[\frac{\partial^k \psi}{\partial \omega^k} \right]_{\omega=0}$$

Transformation of random variables – Let the variables X and Y be linked by some function. By noting f_X and f_Y the distribution function of X and Y respectively, we have:

$$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|$$

Leibniz integral rule – Let g be a function of x and potentially c , and a, b boundaries that may depend on c . We have:

$$\frac{\partial}{\partial c} \left(\int_a^b g(x) dx \right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x) dx$$

Chebyshev's inequality – Let X be a random variable with expected value μ and standard deviation σ . For $k, \sigma > 0$, we have the following inequality:

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2}$$

Jointly Distributed Random Variables

Conditional density – The conditional density of X with respect to Y , often noted $f_{X|Y}$, is defined as follows:

$$f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}$$

Independence – Two random variables X and Y are said to be independent if we have:

$$f_{XY}(x,y) = f_X(x)f_Y(y)$$

Marginal density and cumulative distribution – From the joint density probability function f_{XY} , we have:

Case	Marginal density	Cumulative function
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i, y_j)$	$F_{XY}(x,y) = \sum_{x_i \leq x} \sum_{y_j \leq y} f_{XY}(x_i, y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y) dy$	$F_{XY}(x,y) = \int_{-\infty}^x \int_{-\infty}^y f_{XY}(x',y') dx' dy'$

Distribution of a sum of independent random variables – Let $Y = X_1 + \dots + X_n$ with X_1, \dots, X_n independent. We have:

$$\psi_Y(\omega) = \prod_{k=1}^n \psi_{X_k}(\omega)$$

Covariance – We define the covariance of two random variables X and Y , that we note σ_{XY}^2 or more commonly $\text{Cov}(X,Y)$, as follows:

$$\text{Cov}(X,Y) \triangleq \sigma_{XY}^2 = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X \mu_Y$$

Correlation – By noting σ_X, σ_Y the standard deviations of X and Y , we define the correlation between the random variables X and Y , noted ρ_{XY} , as follows:

$$\rho_{XY} = \frac{\sigma_{XY}^2}{\sigma_X \sigma_Y}$$

Remarks: For any X, Y , we have $\rho_{XY} \in [-1,1]$. If X and Y are independent, then $\rho_{XY} = 0$.

Main distributions – Here are the main distributions to have in mind:

Type	Distribution	PDF	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n, p)$ Binomial	$P(X = x) = \binom{n}{x} p^x q^{n-x}$ $x \in \llbracket 0, n \rrbracket$	$(pe^{i\omega} + q)^n$	np	npq
	$X \sim \text{Po}(\mu)$ Poisson	$P(X = x) = \frac{\mu^x}{x!} e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	μ	μ
(C)	$X \sim \mathcal{U}(a, b)$ Uniform	$f(x) = \frac{1}{b-a}$ $x \in [a, b]$	$\frac{e^{i\omega b} - e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu, \sigma)$ Gaussian	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2} \left(\frac{x-\mu}{\sigma} \right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	μ	σ^2
	$X \sim \text{Exp}(\lambda)$ Exponential	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1 - \frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

Parameter estimation

Random sample – A random sample is a collection of n random variables X_1, \dots, X_n that are independent and identically distributed with X .

Estimator – An estimator $\hat{\theta}$ is a function of the data that is used to infer the value of an unknown parameter θ in a statistical model.

Bias – The bias of an estimator $\hat{\theta}$ is defined as being the difference between the expected value of the distribution of $\hat{\theta}$ and the true value, i.e.:

$$\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta$$

Remark: an estimator is said to be unbiased when we have $E[\hat{\theta}] = \theta$.

Sample mean and variance – The sample mean and the sample variance of a random sample are used to estimate the true mean μ and the true variance σ^2 of a distribution, are noted \bar{X} and s^2 respectively, and are such that:

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i \quad \text{and} \quad s^2 = \hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$$

Central Limit Theorem – Let us have a random sample X_1, \dots, X_n following a given distribution with mean μ and variance σ^2 , then we have:

$$\bar{X} \underset{n \rightarrow +\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$