Nhận dạng mẫu (Pattern Recognition)

Mạng nơ-ron trong PyThon

By Hoàng Hữu Việt
Email: viethh@vinhuni.edu.vn
Viện Kỹ thuật và Công nghệ, Đại học Vinh

Vinh, 5/2019

Tài liệu tham khảo

- Tài liệu chính

  [1] Martin T. Hagan, Howard B. Demuth, Mark Hudson Beale. Neural Network Design, 2nd.

  link: hagan.okstate.edu/nnd.html.

  [2] Deep Learning with Python, FRANCOIS CHOLLET, Manning, 2018.

- Tài liệu khác

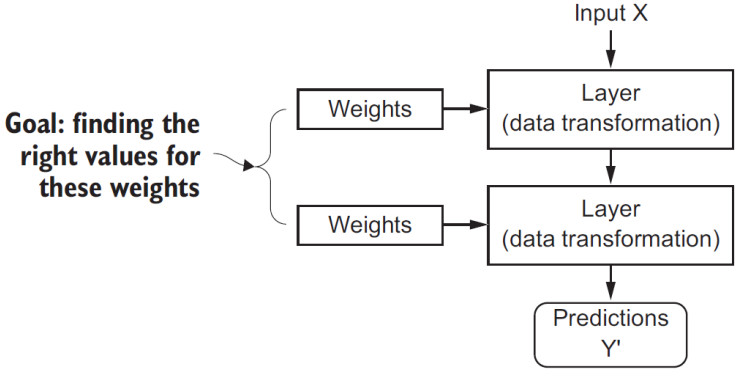
  [2] Các nguồn từ internet

Nội dung

- Giới thiệu
- Lập trình mạng neural với Keras
- Mạng Perceptron đa tầng với PyThon
- Đánh giá mô hình học máy
- Overfitting và Underfitting
- Bài tập

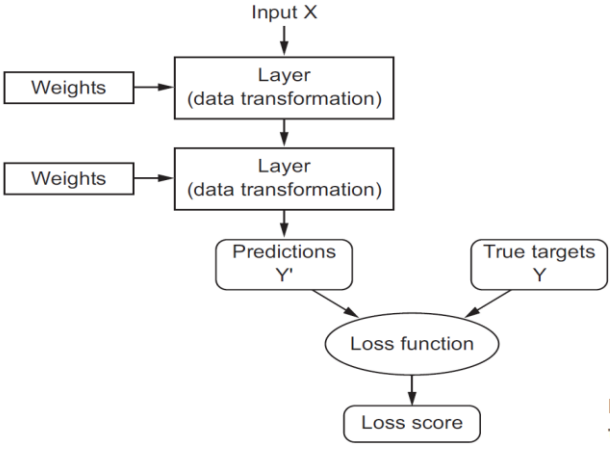
Giới thiệu

- Mạng neuron được tham số hóa bởi các trọng số.
- Học (learning): tìm tập giá trị cho các trọng số của tất cả các tầng mạng.
- Mạng học sâu có thể chứa hàng triệu tham số.



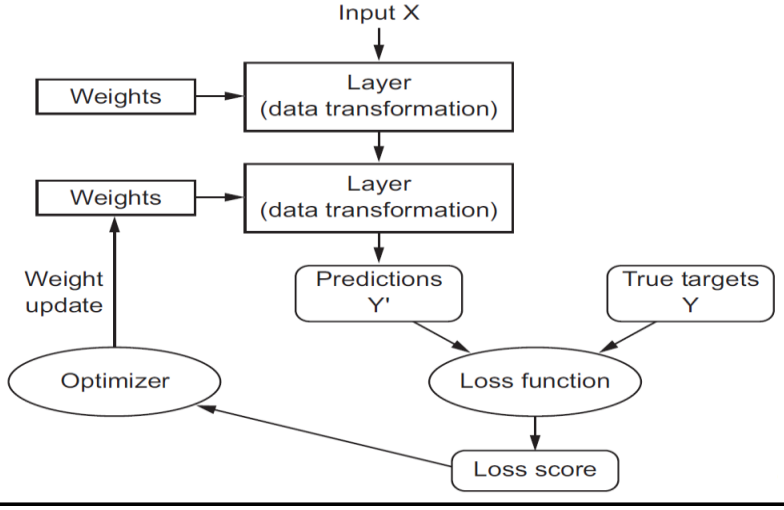
Giới thiệu

- Hàm mất mát (loss function) hay hàm mục tiêu (objective function): đo sự sai khác giữa đầu ra thực tế và đầu ra của mạng.



Giới thiệu

- Hàm mất mát (loss function) dùng để điều chỉnh tập tham số của mạng để đạt sự mất mát là bé nhất.

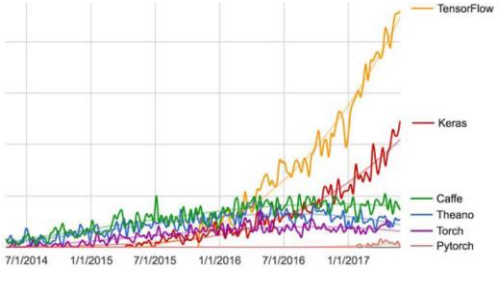


Keras

- Keras là một thư viện cho Python nhằm cung cấp một cách thuận tiện để định nghĩa và huấn luyện hầu hết các mô hình học sâu.
- Ban đầu Keras được phát triển cho những người nghiên cứu với mục đích thiết lập môi trường thử nghiệm nhanh nhất.
- Đặc điểm chính:
  - Cho phép chạy cùng mã lệnh trên bộ vi xử lý CPU và GPU.
  - Cung cấp các giao diện API thân thiện để phát triển nhanh mạng nơ-ron.

Keras

- Keras được cung cấp miễn phí bởi MIT.
- Tương thích với các phiên bản của Python từ 2.7 đến 3.6 (giữa năm 2017)
- Hiện nay Keras có hơn 200,000 người dùng.
- Keras được sử dụng ở Google, Netflix, Uber, CERN, Yelp, Square, và ở hàng trăm công ty khởi nghiệp.



Biểu diễn dữ liệu cho mạng nơ-ron

- Hiện nay tất cả các hệ thống học máy sử dụng tensors như là những cấu trúc dữ liệu cơ bản.
- Bản chất của tensors là các mảng dữ liệu số.
- Số vô hướng:

```
import numpy as np
x = np.array(12)
```

- Vector là một mảng các số và được gọi là 1D tensor:

```
import numpy as np
x = np.array([12,3,6,14])
```

Biểu diễn dữ liệu cho mạng neural

- Ma trận hai chiều (2D tensors) là một mảng các vector:

```
import numpy as np
x = np.array([[5,78,2,34,0],
              [6,79,3,35,1],
              [7,80,4,36,2]])
```

- Ma trận 3 chiều (3D tensors) là 3 ma trận hai chiều được ghép với nhau:

```
import numpy as np
x = np.array([[[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]],
              [[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]],
              [[5, 78, 2, 34, 0], [6, 79, 3, 35, 1], [7, 80, 4, 36, 2]]])
```

Biểu diễn dữ liệu cho mạng neural

- Dữ liệu vector:
  - Một lô (`batch`) dữ liệu vector được biểu diễn bởi một 2D tensors - `(samples, features)`
  - Mỗi dòng biểu diễn đặc các trưng (`features`) của các mẫu (`samples`).
  - Ví dụ:

$$\left\{p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, t_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\},$$
$$\left\{p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, t_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}, \left\{p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, t_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\},$$
$$\left\{p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_5 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\}, \left\{p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, t_6 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\},$$
$$\left\{p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, t_7 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}, \left\{p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, t_8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}.$$

Biểu diễn dữ liệu cho mạng neural

- Mã lệnh

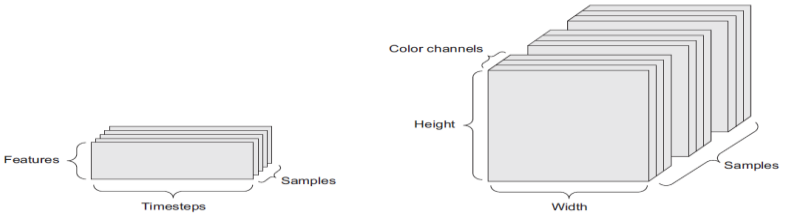
```
import numpy as np
#Định nghĩa dữ liệu
train_data = np.array([[ 1,1],[ 1,2],[ 2,-1],[ 2, 0],
                       [-1,2],[-2,1],[-1,-1],[-2,-2]])
#Định nghĩa lớp đầu ra
train_label = np.array([[0,0],[0,0],[0,1],[0,1],
                        [1,0],[1,0],[1,1],[1,1]])
```

Biểu diễn dữ liệu cho mạng neural

- Dữ liệu chuỗi thời gian: một lô (batch) dữ liệu chuỗi thời gian được biểu diễn theo dạng:

```
(samples, timesteps, features)
```

- Dữ liệu hình ảnh:
  - Một ảnh được biểu diễn bởi ma trận ba chiều của `(height, width, color_depth)`.
  - Một lô (batch) của các ảnh được biểu diễn theo dạng `(samples, height, width, color_depth)`.



Biểu diễn dữ liệu cho mạng neural

- Dữ liệu video:
  - Một video được biểu diễn bởi theo dạng `(frames, height, width, color_depth)`.
  - Một lô (batch) của các video được biểu diễn theo dạng `(samples, frames, height, width, color_depth)`.
- Ví dụ có 4 videos với mỗi video có 240 khung (`frame`) và mỗi khung có kích thước 144×256 được lưu trữ theo dạng (4,240,144,256,3).

Lập trình mạng neural với Keras

- Các bước cơ bản để lập trình mạng nơ-ron với Keras:
  1. Khai báo các thư viện.
  2. Định nghĩa dữ liệu huấn luyện, bao gồm các mẫu dữ liệu vào và đầu ra tương ứng.
  3. Tạo mạng nơ-ron.
  4. Cấu hình mạng nơ-ron cho quá trình huấn luyện.
  5. Huấn luyện mạng.
  6. Nhận dạng các mẫu dữ liệu.

Mạng Perceptron đa tầng với PyThon

- Các bước cơ bản:
  1. Khai báo các thư viện
  2. Định nghĩa tập mẫu huấn luyện
  3. Định nghĩa lớp đầu ra cho các mẫu huấn luyện
  4. Tạo mô hình mạng: `model = models.Sequential()`
  5. Tạo tầng đầu tiên: `model.add(layers.Dense(...))`
  6. Tạo tạo tầng tiếp theo: `model.add(layers.Dense(...))`
  7. Cấu hình mạng cho huấn luyện: `model.compile(...)`
  8. Huấn luyện mạng: `model.fit(...))`
  9. Nhận dạng mẫu dữ liệu vào: `model.predict(...))`
  10. Hiện thị trọng số: `model.layers[k].get_weights()`

Mạng Perceptron đa tầng với PyThon

- Ví dụ 1: phân lớp dữ liệu với các mẫu dữ liệu huấn luyện như sau:

$$\left\{p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; t_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\}, \left\{p_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \; t_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right\},$$
$$\left\{p_3 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \; t_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\}, \left\{p_4 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \; t_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right\},$$
$$\left\{p_5 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \; t_5 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\}, \left\{p_6 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \; t_6 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right\},$$
$$\left\{p_7 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \; t_7 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}, \left\{p_8 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \; t_8 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\}.$$

Mạng Perceptron đa tầng với PyThon

- Khai báo các thư viện Numpy và Keras:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from keras import models
from keras import layers
```

- Định nghĩa dữ liệu huấn luyện:

```
train_data = np.array([[1,1],[1,2],[2,-1],[2,0],
                       [-1,2],[-2,1],[-1,-1],[-2,-2]],
                       "float32")
train_label= np.array([[0,0],[0,0],[0,1],[0,1],
                       [1,0],[1,0],[1,1],[1,1]],
                       "float32")
```

Mạng Perceptron đa tầng với PyThon

- Tạo mạng nơ-ron: số tầng? số nơ-ron cho mỗi tầng?

```
model = models.Sequential()
model.add(layers.Dense(2,activation ='linear',input_dim = 2))
```

- Cấu hình mạng: thiết lập phương pháp tối ưu và hàm mất mát cho mạng.

```
model.compile(optimizer = 'sgd',loss = 'mse'))
```

- Huấn luyện mạng: tìm tham số của mạng để tối thiểu hàm mất mát.

```
model.fit(train_data,train_label,epochs = 500,batch_size = 2)
```

- Nhận dạng mẫu dữ liệu: đưa vào mạng một tập mẫu và tính đầu ra của tập mẫu.

```
print(model.predict(train_data).round())
```

- Hiển thị trọng số:

```
print(model.get_weights())
```

Mạng Perceptron đa tầng với PyThon

```
import numpy as np
from keras import models
from keras import layers
train_data = np.array([[1,1],[1,2],[2,-1],[2,0],
                       [-1,2],[-2,1],[-1,-1],[-2,-2]],
                       "float32")
train_label = np.array([[0,0],[0,0],[0,1],[0,1],
                        [1,0],[1,0],[1,1],[1,1]],
                        "float32")
model = models.Sequential()
model.add(layers.Dense(2,activation='linear',input_dim=2))
model.compile(optimizer = 'sgd',loss = 'mse')
model.fit(train_data,train_label,epochs=500,batch_size=2)
print(model.predict(train_data).round())
print(model.get_weights())
```

Mạng Perceptron đa tầng với PyThon

- Ví dụ 2: phân lớp dữ liệu với các mẫu dữ liệu huấn luyện như sau (bài toán XOR):

$$\left\{p_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ t_1 = 0\right\}, \left\{p_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ t_2 = 1\right\},$$

$$\left\{p_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ t_3 = 1\right\}, \left\{p_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ t_4 = 1\right\}.$$

- Cần mạng tối thiểu có bao nhiêu tầng? số lượng nơ-ron cho mỗi tầng?

Mạng Perceptron đa tầng với PyThon

- Khai báo các thư viện Numpy và Keras:

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from keras import models
from keras import layers
```

- Định nghĩa dữ liệu huấn luyện:

```
train_data = np.array([[0,0],[0,1],[1,0],[1,1]],
                      "float32")
train_label = np.array([[0],[1],[1],[0]],"float32")
```

- Tạo mạng:

```
model = models.Sequential()
model.add(layers.Dense(4,activation='tanh',input_dim=2))
model.add(layers.Dense(1,activation='sigmoid'))
```

Mạng Perceptron đa tầng với PyThon

- Cấu hình mạng: thiết lập phương pháp tối ưu và hàm mất mát cho mạng.

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

- Huấn luyện mạng: tìm tham số của mạng.

```
model.fit(train_data,train_label,epochs=1000,batch_size=1)
```

- Nhận dạng mẫu dữ liệu: đưa vào mạng một tập mẫu và tính đầu ra của tập mẫu.

```
print(model.predict(train_data).round())
```

- Hiển thị trọng số:

```
print(model.layers[0].get_weights())
print(model.layers[1].get_weights())
```

Mạng Perceptron đa tầng với PyThon

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np
from keras import models
from keras import layers
train_data = np.array([[0,0],[0,1],[1,0],[1,1]],"float32")
train_targets = np.array([[0],[1],[1],[0]],"float32")

model = models.Sequential()
model.add(layers.Dense(16,activation ='tanh',input_dim = 2))
model.add(layers.Dense(2,activation = 'sigmoid'))
model.compile(optimizer = 'sgd',loss = 'binary_crossentropy',
                                metrics = ['accuracy'])
model.fit(train_data,train_targets,epochs = 1000,batch_size = 1)
print(model.predict(train_data).round())
print(model.layers[0].get_weights())
print(model.layers[1].get_weights())
```

Mạng Perceptron đa tầng với PyThon

- Một số vấn đề cần xem xét:
  - Định nghĩa đầu ra

```
train_label = np.array([[0],[1],[1],[0]],"float32")
```

    do đó mạng chỉ có 1 neuron đầu ra để phân 2 lớp.
  - Nếu định nghĩa đầu ra

```
train_label = np.array([[0,0],[1,1],[1,1],[0,0]],"float32")
```

    thì mạng sẽ cần 2 nơ-ron đầu ra để phân 2 lớp.
  - Tăng số nơ-ron đầu vào lên 8, 16 và chạy lại mạng, cho nhận xét.
  - Tăng số `batch_size`, chạy mạng và cho nhận xét.
  - Giảm số epochs, chạy mạng và cho nhận xét.
  - Tăng thêm 1 tầng mạng có 8 nơ-ron, chạy mạng và cho nhận xét.

Mạng Perceptron đa tầng với PyThon

- Ví dụ 3: nhận dạng 10 chữ số viết tay trong cơ sở dữ liệu MNIST [http://yann.lecun.com/exdb/mnist/].
- Tập dữ liệu huấn luyện bao gồm 60,000 ảnh chữ số viết tay và tập nhận dạng bao gồm 10,000 ảnh chữ số viết tay, mỗi chữ số là một ảnh đa mức xám kích thước 28×28.

Mạng Perceptron đa tầng với PyThon

- Đọc cơ sở dữ liệu ảnh vào các mảng dữ liệu

```
from keras.datasets import mnist
(train_images,train_labels),(test_images,test_labels) =
 mnist.load_data()
```

- Hiển thị 100 chữ số trong tập huấn luyện

```
import matplotlib.pyplot as plt
fig = plt.figure()
cols = 10
rows = 10
for i in range(1,rows*cols+1):
    digit = train_images[i-1]
    fig.add_subplot(rows,cols,i)
    plt.axis('off')
    plt.imshow(digit,cmap = plt.cm.binary)
plt.show()
```

Mạng Perceptron đa tầng với PyThon

- Khai báo các thư viện Numpy và Keras:

```
import warnings
warnings.filterwarnings('ignore')
from keras import models
from keras import layers
```

- Đọc cơ sở dữ liệu ảnh vào các mảng dữ liệu:

```
from keras.datasets import mnist
(train_images,train_labels),(test_images,test_labels) =
 mnist.load_data()
```

- Chuyển kiểu ma trận dữ liệu và chuyển kiểu dữ liệu:

```
train_images = train_images.reshape((60000,28*28))
train_images = train_images.astype('float32')/255
test_images = test_images.reshape((10000,28*28))
test_images = test_images.astype('float32')/255
```

Mạng Perceptron đa tầng với PyThon

- Chuyển định dạng phân lớp dữ liệu:

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

- Tạo mô hình mạng:

```
model = models.Sequential()
model.add(layers.Dense(256,activation ='relu',
                       input_shape = (28*28,)))
model.add(layers.Dense(128,activation ='relu'))
model.add(layers.Dense(10,activation ='softmax'))
```

- Cấu hình mạng:

```
model.compile(optimizer ='rmsprop',
  loss ='categorical_crossentropy',metrics=['accuracy'])
```

Mạng Perceptron đa tầng với PyThon

- Huấn luyện mạng:

```
history = model.fit(train_images,train_labels,
                    epochs = 5,batch_size=128)
```

- Nhận dạng mẫu dữ liệu:

```
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels)
print('test_acc:', test_acc)
```

- Chạy chương trình và kiểm tra tỷ lệ nhận dạng chính xác?

Mạng Perceptron đa tầng với PyThon

- Một số vấn đề xem xét:
  - Bài toán nhận dạng có sử dụng đặc trưng của các chữ số không?
  - Đặc trưng của các chữ số là gì?
  - Tìm xem những chữ số bị nhận dạng sai trong 100 chữ số đầu tiên?. Hiển thị các chữ số bị nhận dạng sai.
  - Chuyển thành mạng 2 tầng, tỷ lệ nhận dạng chính xác là bao nhiêu?
  - Chuyển thành mạng 4 tầng, tỷ lệ nhận dạng chính xác là bao nhiêu?
  - Thay đổi số lượng nơ-ron trên các tầng và chạy chương trình, quan sát tỷ lệ nhận dạng chính xác.

Mạng Perceptron đa tầng với PyThon

- Tầng mạng (layers): số tầng/nơ-ron cần chọn?
- Hàm mất mát (loss): sử dụng hàm gì cho bài toán?
- Phương pháp tối ưu (optimizer): chọn phương pháp nào?
- Hàm truyền và hàm mất mát cho tầng cuối cùng:

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Đánh giá mô hình học máy

- Trong ví dụ 3, dữ liệu được chia thành hai bộ là dữ liệu huấn luyện (training) và dữ liệu thử nghiệm (test).
- Đánh giá mô hình trên bộ dữ liệu huấn luyện sẽ không chính xác vì mạng sẽ hội tụ nhanh trong trên các bộ dữ liệu đã được huấn luyện
- Trong học máy, mục đích là cần phải đạt được một mô hình tổng quát, tức là mạng thực hiện tốt cho cả những dữ liệu chưa biết trước.

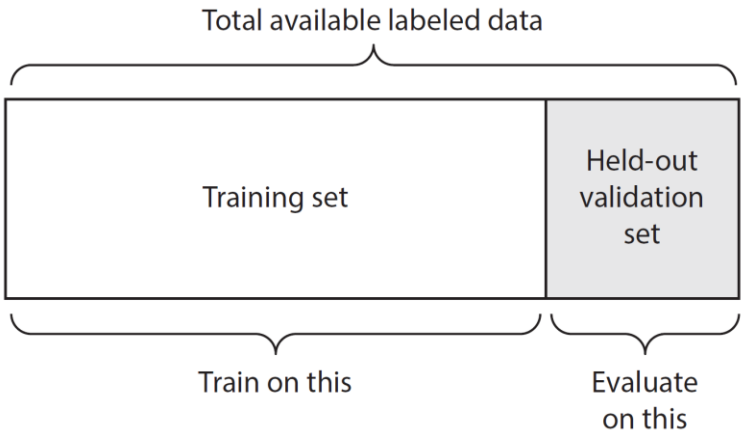
Hướng giải quyết?

Đánh giá mô hình học máy

- Đánh giá một mô hình luôn phải chia tách dữ liệu thành ba bộ: huấn luyện (`training`), đánh giá (`validation`) và kiểm tra (`test`).
- Mạng được huấn luyện trên bộ dữ liệu huấn luyện.
- Mạng được đánh giá trên bộ dữ liệu đánh giá.
- Khi có được mô hình đúng đắn, mạng sẽ được sử dụng trên bộ dữ liệu kiểm tra.
- Tại sao chúng ta không dùng 2 tập dữ liệu: tập dữ liệu huấn luyện và tập dữ liệu kiểm tra?
  - Huấn luyện mạng trên tập dữ liệu huấn luyện và đánh giá trên tập dữ liệu thử nghiệm sẽ đơn giản hơn!

Đánh giá mô hình học máy

- Tạo tập dữ liệu đánh giá đơn giản:



- Xem thêm code trang 98 [2]

Đánh giá mô hình học máy

- Khai báo các thư viện Numpy và Keras:

```
import warnings
warnings.filterwarnings('ignore')
from keras import models
from keras import layers
```

- Đọc cơ sở dữ liệu ảnh vào các mảng dữ liệu:

```
from keras.datasets import mnist
(train_images,train_labels),(test_images,test_labels) =
 mnist.load_data()
```

- Chuyển kiểu ma trận dữ liệu và chuyển kiểu dữ liệu:

```
train_images = train_images.reshape((60000,28*28))
train_images = train_images.astype('float32')/255
test_images = test_images.reshape((10000,28*28))
test_images = test_images.astype('float32')/255
```

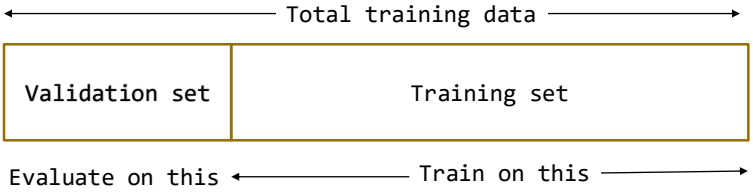
Đánh giá mô hình học máy

- Chuyển định dạng phân lớp dữ liệu:

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

- Tạo tập dữ liệu đánh giá (validation):

```
val_train_images = train_images[:10000]
val_train_labels = train_labels[:10000]
partial_train_images = train_images[10000:]
partial_train_labels = train_labels[10000:]
```



Đánh giá mô hình học máy

- Tạo mô hình mạng:

```
model = models.Sequential()
model.add(layers.Dense(256,activation = 'relu',
                       input_shape =(28*28,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(10,activation = 'softmax'))
```

- Cấu hình mạng:

```
model.compile(optimizer = 'rmsprop',
  loss ='categorical_crossentropy',metrics=['accuracy'])
```

- Huấn luyện mạng:

```
history = model.fit(partial_train_images,
  partial_train_labels,epochs=40,batch_size=128,verbose=2,
  validation_data = (val_train_images,val_train_labels))
```

Đánh giá mô hình học máy

- Nhận dạng mẫu dữ liệu:

```
test_loss, test_acc = model.evaluate(test_images,
                                     test_labels)
print('test_acc:', test_acc)
```

- Hiển thị hàm mất mát:

```
import matplotlib.pyplot as plt
plt.figure(1)
plt.plot(history.history['loss'],'b',
                         label = 'Training loss')
plt.plot(history.history['val_loss'],'r',
                         label = 'Validation loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(loc = 'upper left')
```

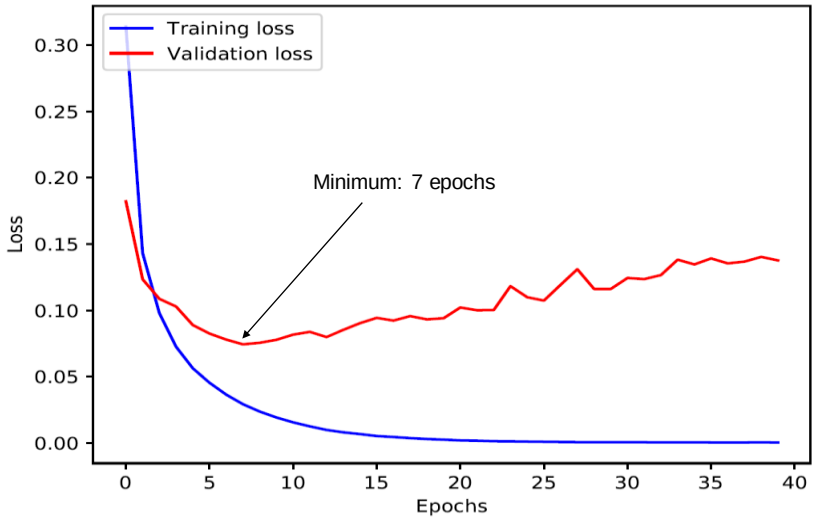
Đánh giá mô hình học máy

- Hiển thị hàm tỷ lệ nhận dạng chính xác:

```
plt.figure(2)
plt.plot(history.history['acc'],'b',
         label = 'Training accurancy')
plt.plot(history.history['val_acc'],'r',
         label = 'Validation accurancy')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(loc = 'upper left')
plt.show()
```

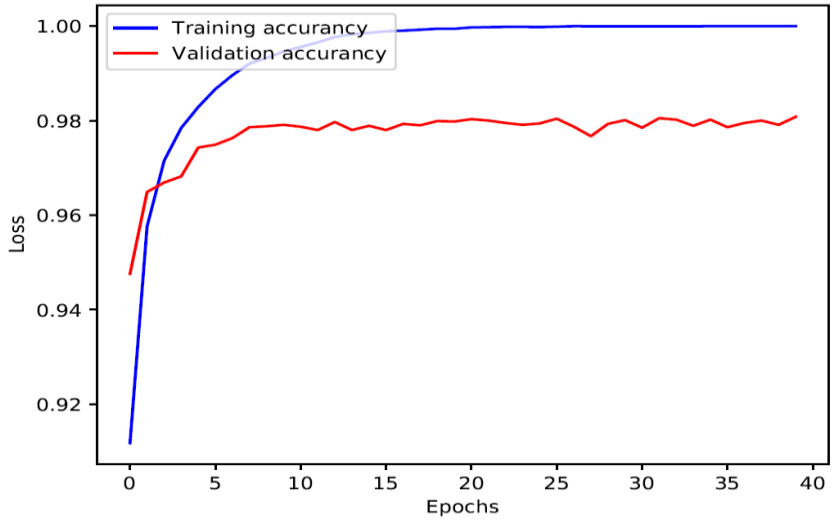
Đánh giá mô hình học máy

- Đồ thị hàm mất mát



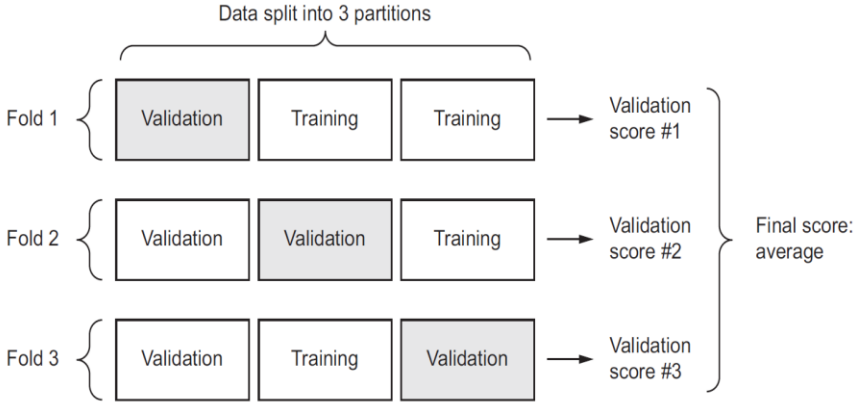
Đánh giá mô hình học máy

- Đồ thị tỷ lệ nhận dạng chính xác



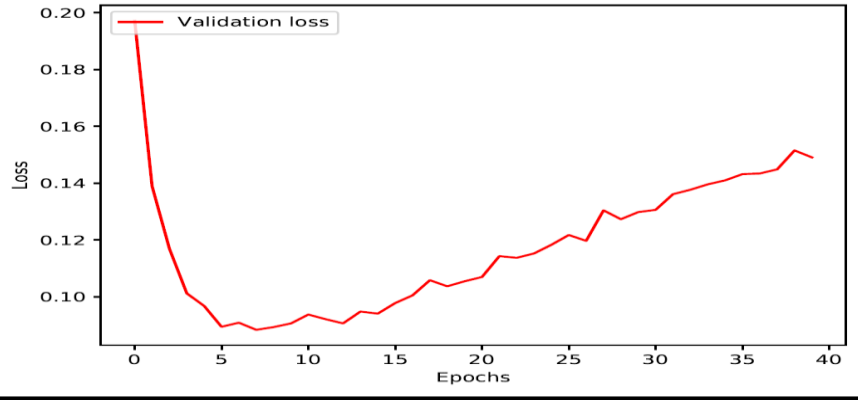
Đánh giá mô hình học máy

- Phương pháp kiểm tra chéo K-Fold:
  - Đối với những bài toán với dữ liệu nhỏ, phương pháp đánh giá mô hình bằng kiểm tra chéo (cross validation) rất hiệu quả (xem code trang 99 [2]).



Overfitting và Underfitting

- Sau một số tập (epoch) mô hình đạt được tối thiểu hàm mất mát (loss function) và sau đó tăng dần.
  - Underfitting: quá trình giảm; Overfitting: quá trình tăng.
- Overfitting luôn xẩy ra trong mọi bài toán học máy

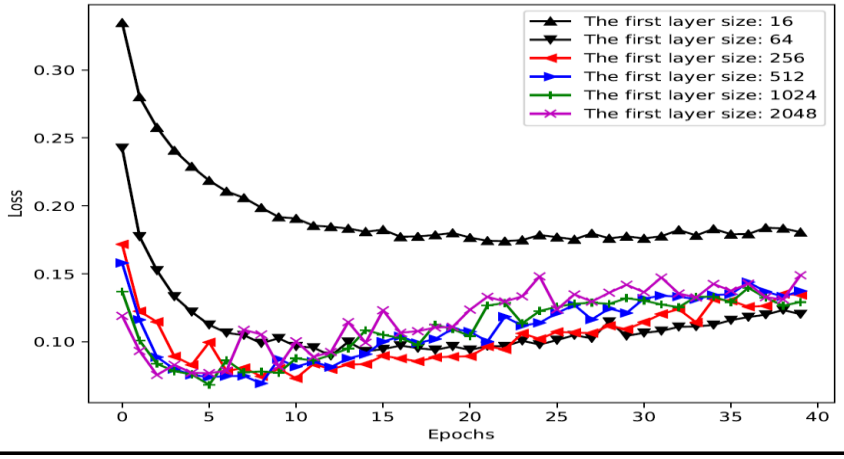


Overfitting và Underfitting

- Tối ưu hóa mô hình đề cập tới quá trình điều chỉnh một mô hình để nhận được sự khả năng thực hiện tốt nhất có thể trên dữ liệu huấn luyện.
- Tổng quát hóa mô hình đề cập tới vấn đề làm thế nào để một mô hình đã được huấn luyện thực hiện tốt nhất trên dữ liệu mà trước đó chưa được huấn luyện.
- Mục đích của học máy là tìm được một mô hình mà tổng quát hóa tốt dựa trên quá trình học dữ liệu huấn luyện.

Một số kỹ thuật giải quyết Overfitting

- Giảm kích thước mạng: giảm số tầng và số nơ-ron trên mỗi tầng.
- Ví dụ bài toán nhận dạng chữ số viết tay:

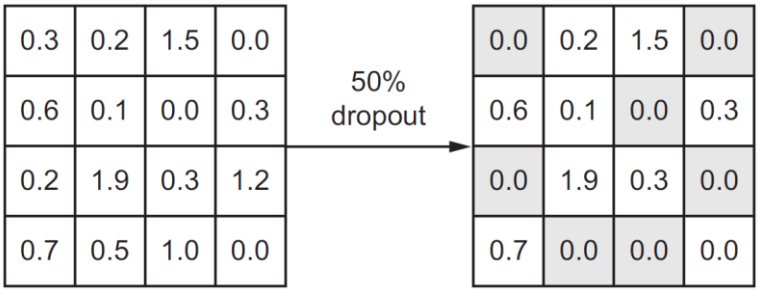


Một số kỹ thuật giải quyết Overfitting

- Thêm trọng số ”regularization”: thiết lập các ràng buộc trọng của mạng để các trọng số chỉ lấy các giá trị nhỏ.
- Được thực hiện bằng cách cộng thêm hàm mất mát của mạng một giá trị được kết hợp với các trọng số có giá trị lớn.
- “`L1 regularization`”: Giá trị được cộng tỷ lệ với giá trị tuyệt đối của các hệ số trọng số.
- “`L2 regularization`”: Giá trị được cộng tỷ lệ với bình phương của giá trị của các hệ số trọng số.
- Xem ví dụ trong tài liệu.

Một số kỹ thuật giải quyết Overfitting

- Kỹ thuật dropout:
  - Là một trong những kỹ thuật hiệu quả nhất và thường xuyên được sử dụng cho mạng neuron để giảm overfitting.
  - Ví dụ dropout cho đầu ra của một tầng mạng với tỷ lệ 50%:



- Thêm tầng dropout

```
model.add(layers.Dropout(0.5))
```

Bài tập 5.1

- Cho 2 thư mục ifd-train và ifd-test chứa các ảnh mặt người của 61 người, trong đó thư mục ifd-train chứa ảnh dùng để huấn luyện và thư mục ifd-test dùng chứa các ảnh nhận dạng.
- Tên tệp trong 2 thư mục chứa một dấu '-' và các ký tự trước dấu '-' biểu diễn lớp dữ liệu của ảnh, các ký tự sau dấu '-' là số thứ tự hình ảnh mặt của mỗi người.
- Hãy thiết kế và lập trình một mạng neuron để nhận dạng mặt người dựa trên các ảnh đã cho.

Hướng dẫn

- Đọc dữ liệu huấn luyện:

```
import numpy as np
import glob
from PIL import
Image train_dir = 'ifd-train/'
class_num = 61
train_images = []
train_labels = []
for i in range(1,class_num+1):
     filenames = train_dir + str(i) + '-*.jpg'
     for filename in glob.glob(filenames):
         train_labels.append(i-1)
         train_image = Image.open(filename)
         train_image = np.array(train_image)
         train_images.append(train_image)
train_images = np.array(train_images)
```

Hướng dẫn

- Đọc dữ liệu kiểm tra:

```
test_dir = 'ifd-test/'
test_images = []
test_labels = []
for i in range(1,class_num+1):
    filenames = test_dir + str(i) + '-*.jpg'
    for filename in glob.glob(filenames):
        test_labels.append(i-1)
        test_image = Image.open(filename)
        test_image = np.array(test_image)
        test_images.append(test_image)
test_images = np.array(test_images)
```

- Tạo mạng, huấn luyện và nhận dạng -> tương tự các ví dụ trên.