

Documents

- + [The latest in Machine Learning | Papers With Code](#)
- + [Paperswithcode](#) , <https://machinelearningcoban.com/> (Trang của Vũ Hữu Tiệp)
- + Machine Learning cơ bản 2018 : Vũ Hữu Tiệp
 - + Các thuật toán học máy – giải thích chi tiết, kiến thức toán,... và nhiều thứ khác nữa .
- + Deep learning cơ bản : Nguyễn Thanh Tuấn . [Blog | Deep Learning cơ bản \(nttuan8.com\)](#) .
https://github.com/nttuan8/DL_Tutorial (Source code + dataset)
- + Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf
- + Deep Learning for Natural Language Processing.pdf
- + [Introduction - Hugging Face NLP Course](#)
- + Tính Accuracy, Recall, F1-score : <https://www.miai.vn/2020/06/16/oanh-gia-model-ai-theo-cach-mi-an-lien-chuong-2-precision-recall-va-f-score/>
- + <https://github.com/AmoliR/nlp-for-book-recommendation>

- + Tranh thu thập dataset : <https://app.activeloop.ai/public/> , <https://app.activeloop.ai/activeloop/lfw>
- + [WHAT IS TRANSFER LEARNING?](#)
- + [Tổng hợp data - THI GIÁC MÁY TÍNH \(thigiacmaytinh.com\)](#)
- + Full source and Dataset : [phamvanhanh6720/LicensePlateDetection \(github.com\)](#) (Phát hiện biển số xe)(Darknet & YOLO4)
- + PCA – LSTM : <http://mocap.cs.cmu.edu/search.php> (dùng dữ liệu : c3d - amc)

Keywords

- + state-of-the-art (SOTA) : Hiện đại nhất
- + Artificial Neural Network (ANN) : Mạng thần kinh nhân tạo
- + Convolutional neural network (CNN) : Mạng nơ ron tích chập
- + [Getting started with Keras](#)
- + Data visualization : Trực quan hóa dữ liệu
- + 7 Phương pháp đánh giá độ chính xác cho mô hình hồi quy
 1. Mean Squared Error (MSE)
 2. Mean Absolute Error (MAE)
 3. R squared (R^2)
 4. Adjusted R squared
 5. Root Mean Squared Error (RMSE)
 6. Mean Absolute Percentage Error (MAPE)
 7. Mean Percentage Error (MPE)
- + [Home - Keras-vis Documentation \(raghakot.github.io\)](#) - keras visualizer
- + Vẽ được **heatmap** với vùng càng nóng thì xác suất mô hình dự đoán càng cao

Task and Read & Research [Quan trọng]

Read Now

- [lecture_8-Recurrent Neural Networks.pdf](#)
- [Pythonfusion - Hay.pdf](#)
- [Selective Search for Object Recognition.pdf](#)
- [Flickr 8k Dataset \(kaggle.com\)](#)

Task 1 : Face recognition and variable label numbers.

- Đặt vấn đề : Với bài toán nhận diện (nhận diện hoa , lá , khuông mặt , động vật ,...)
- 1. Nếu mặc định là số nhãn cố định : Theo thông thường thì cho qua softmax thì nó đều ra cho tỷ lệ % là bao nhiêu đó rồi sau đó sẽ có nhãn dựa trên nhãn có % cao nhất .
Vấn đề : Nếu ví dụ có 10 loài hoa (hoặc 10 khuông mặt) nhưng có khuông mặt mới vào nếu làm cách này thì vẫn cho ra được nhãn , tuy nhiên trên thực tế thì ta thấy rõ ràng nhãn này không nằm trong tập .
⇒ Cách giải quyết : Không dùng softmax để nhận diện nữa (lúc train thì vẫn dùng) nhưng lúc nhận diện sẽ là trích xuất vector đặc trưng sau đó **bằng cách nào đó cho ra A% động thời tìm được một ngưỡng P%** , nếu A lớn hơn hoặc bằng P thì chọn , (nếu có nhiều $A \geq P$ thì chọn cái có A% cao nhất)
 - ❖ **Những cách có thể triển khai để tìm A%**
 - a. Trích xuất vector đặc trưng và so khớp bằng SVM hoặc nhiều pp so khớp khác để tìm ra A%
 - b. Other... Tìm hiểu thêm cách khác
- 2. Nếu có số nhãn thay đổi : Ví dụ như bài toán nhận diện khuông mặt để chấm công . Nếu có số nhân viên thay đổi , nghĩa là số khuông mặt thay đổi .
 - a. Thêm khuông mặt vào
 - o Thêm một ảnh : Trích xuất vector đặc trưng của ảnh này và lưu vào database nhưng những ảnh đã lưu trong database
Ví dụ trước đó có 10 người thì lưu 5 vector đặc trưng của 5 người , giờ lưu thêm 1 vector đặc trưng nữa là 6 .
 - o Thêm nhiều ảnh hoặc video :
 - Tao ra nhiều vector đặc trưng cho 1 nhãn : Chia video thành các ảnh rời , ví dụ một video chia thành 10 ảnh , trích xuất vector đặc trưng cho cả 10 ảnh này và lưu vào database , như vậy trước có 5 người có $5*10$ vector đặc trưng trong database , giờ thêm một người nữa thì có thêm 10 vector đặc trưng là 60 vector cho 6 người . (Ví dụ như video thì ta phải cho nó quay các góc của khuông mặt , dẫn đến có 10 ảnh cho 10 góc) .
 - Cách thức so khớp : Ví dụ khi người áp mặt vào camera thì ta cắt ảnh khuông mặt đó (1 ảnh) rồi trích xuất ra vector đặc trưng sau đó **đem đi so khớp với toàn bộ 60 vector đặc trưng** vì ảnh đó là 1 ảnh đó là 1 góc nào đó của khuông mặt người đó . Đem đi so khớp toàn bộ , nhãn ứng với độ khớp cao nhất thì lấy ra người đó .
 - Tạo ra 1 vector đặc trưng cho 1 nhãn : Cũng trích xuất ra 10 đặc trưng cho 10 ảnh , tuy nhiên sau đó sử dụng **LSTM** hoặc **các phương pháp khác** để gộp 10 đặc

trung lại thành 1 đặc trưng bằng cách cho nó học 10 đặc trưng đó rồi gom lại thành 1 đặc trưng hoặc nhiều cách ,...

- Cách thức so khớp : Khi có vector đặc trưng của ảnh cần nhận diện thì chỉ cần so khớp với vector đó là được .

b. Training lại toàn bộ model

- Các hướng triển khai

		Xây dựng model	Transfer learning model
Số nhãn cố định			
Số nhãn thay đổi	Tranning lại toàn bộ nhãn cho model Trích xuất vector đặc trưng nhãn		
	1 ảnh Nhiều ảnh hoặc video	Lặp qua và so sánh Gộp đặc trưng bằng LSTM,...	

Task 2 : Object detection với Faster R-CNN

1. Tìm code implement Faster R-CNN trên github, download pre-trained model về và chạy thử với ảnh mới.
2. Customize dataset tùy ý và dùng pre-trained model ở trên để train Faster RCNN với dữ liệu mới.

Task 3 : Code một bài toán RNN (mạng nơ ron tái phát)

- Xem các bài tập ở : Phần 7.1 RNN để code một bài cơ bản sử dụng RNN với Dataset và dạng bài là **Many to one** (nhiều dữ liệu đầu vào và một nhãn đầu ra)

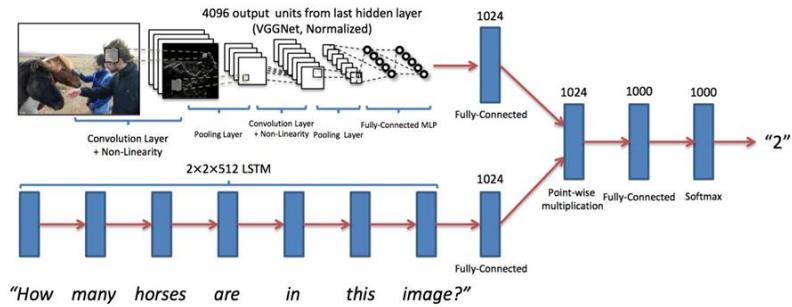
Task 4 : Photo Description (Bài toán mô tả ảnh)

- Chapter7 – RNN (Code bài này)

Task 5 : Visual Question Answering (VQA) (Slide 82 – Lecture 8)

- Tài liệu tham khảo : [lecture_8-Recurrent Neural Networks.pdf](#)
 - Tham khảo bài “Mô tả ảnh” ở Chapter7 RNN .
- ⇒ Từ bài toán mô tả ảnh (**Photo Description**) kết hợp với bài toán Seq2Seq cũng trong Chapter7 để làm bài toán VQA .

Visual Question Answering (VQA)



Code Task

Task 1 : Face Recognition and label number changes

Improve ConvNeXt Model with dataset Alzheimer

Các bước tiến hành

1. ConvNeXt Implement and save Model at best epoch
2. Load Model ConvNeXt Implement , after remove Flatten and Dense as base Model
3. Transfer learning - Feature Extractor using base Model

4. Build LogisticRegression using GridSearchCV

5. Save model LogisticRegression

Implement ConvNeXt Model with dataset Alzheimer

Source code :

[C:\Users\ADMIN\Downloads\DeepLearning\Research\Alzheimer_s\ConvNeXt_Alzheimer-Improve.ipynb](#)

Transfer Learning Feature Extractor Implement ConvNeXt Model with dataset Alzheimer

[C:\Users\ADMIN\Downloads\DeepLearning\Research\Alzheimer_s\ConvNeXt_Alzheimer-Improve-Transfer learning-FeatureExtractor.ipynb](#)

Transfer Learning Fine tuning Implement ConvNeXt Model with dataset Alzheimer

Updating...

Lưu ý : Ta hoàn toàn có thể tiếp tục sử dụng model tự triển khai này để transfer learning : fine tuning cho nó .

- Load model ConvNeXt_model-022.keras (model tốt nhất) ra
- Chỉ lấy base model bỏ đi flatten và dense
- Tự build head model bằng keras
- Nối lại thành model mới
- Đóng băng base model và chỉ train head model
- Mở đóng băng một phần hoặc toàn bộ rồi training lại một lần nữa
- Sau khi train xong thu được model mới , lưu model mới này .

Nói thêm : Với model ResNet ta khó có thể Implement và traning nó . VỚI dữ liệu 102 loài hoa , lấy ra 15 loài hoa sau gần 2500 mẫu dữ liệu , với batch size = 2 và epoch = 4 thôi cũng không chạy được , RAM lên đến gần 23G , chỉ chạy kaggle được những cũng mất khoảng vài tiếng , do parameter của nó lên đến 54 triệu tham số .

Face Recognition and label number changes

Source code : [+ C:\Users\ADMIN\Downloads\DeepLearning\Research\Face Recognition](#)

[C:\Users\ADMIN\Downloads\DeepLearning\Research\Face_Recognition\README.md](#)

Tập dữ liệu

- + Dataset.zip : tập dữ liệu gốc train , test cho file VGGFace_SENet50_Transfer_Fine_Tuning.ipynb
- + Dataset_fine_tuning_train_valid_test.zip : Từ tập Dataset tổ chức thành dữ liệu train , test , valid để Transfer learning : Fine tuning (để nguyên ảnh)
- + Dataset_face_fine_tuning_train_valid_test.zip : Từ tập Dataset_fine_tuning_train_valid_test dùng face_recognition để trích xuất khuôn mặt , resize lại thành 224*224 và lưu lại thành ảnh

Ipynb

1. VGGFace_SENet50_Transfer_Feature_Extractor_Facerognition.ipynb (MAIN)

- + Với mục đích dùng cho bài toán thay đổi số lượng nhãn
- + Sử dụng ngay tập Datatset , resize và detect khuôn mặt trên ảnh đó sau đó chuyển thành tensor
- + Sử dụng Conv của VGG Face và FCs tự build của mình để trích xuất đặc trưng khuôn mặt từ tensor ảnh

- + Tính khoảng cách vector rồi dự đoán nhãn
- + Tìm ngưỡng cho độ chính xác cao nhất (threshold = 15 . cho độ chính xác gần 90%) để phân biệt nhãn UNKNOW (không có trong tập vector cần tính khoảng cách)

2. VGGFace_SENet50_Transfer_Fine_Tuning.ipynb (ĐỌC THÊM)

- + Mục đích là cũng sử dụng Conv của VGG Face + FCs của mình
- + Sau đó transfer learning : fine tuning lại bằng dữ liệu của mình để tăng độ chính xác cho nó , tuy nhiên vì dữ liệu của mình quá ít mà model lại quá phức tạp , dù đã cố gắng giảm độ phức tạp bằng cách thêm các lớp Dropout nhưng vẫn bị overfitting
 - + Thủ lưu lại model và load ra (bỏ đi lớp softmax ở cuối để dùng cho việc trích xuất vector đặc trưng) (VGGFace.TestModel_Fine_tuning.Facerognition.ipynb)
 - nhưng kết quả là độ chính xác chỉ khoảng 30% .

Kết luận

- + Sử dụng file VGGFace_SENet50_Transfer_Feature_Extractor_Facerognition.ipynb
- + Sử dụng tập Dataset.zip
- + Model : save_models\VGGFace_SENet_Transfer_Model.h5

Tham khảo & Research

- + <https://github.com/facebookresearch/faiss>
- + sử dụng FAISS của facebook research để tối ưu cho việc tính khoảng cách và tự động tìm threshold
- + Tự động cập nhật threshold cho mình khi dữ liệu thay đổi (thêm/bớt vector đặc trưng - cũng như classes)

Build App

- + Build App (Client : Vuejs, API Model : Django , API DB : Laravel)
- + Source code : + <C:\Users\ADMIN\Downloads\DeepLearning\App>

Plus

Keras

Using Keras to build the CNN model

1. Các thư viện cần thiết

```
# 1. Thêm các thư viện cần thiết
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.utils import np_utils # thay cho from keras.utils import np_utils (error)
from keras.datasets import mnist
```

2. Chuẩn bị dữ liệu

- **(X_train, y_train), (X_test, y_test) = mnist.load_data()** : Load data từ keras về (tên các biến ta có thể đặt tên lại miễn đúng theo thứ tự để nhận dữ liệu là được)
- Đầu vào của keras **1 tensor 4 chiều (N, W, H, D)** . **D=1** nếu là ảnh xám và **D=3** nếu là ảnh màu
- Dữ liệu có thể được upload lên keras ta chỉ cần load về , dữ liệu này là dữ liệu đã qua xử lý chuyển từ ảnh sang dạng ma trận hoặc tensor
- Ta cũng có thể tự xử lý từ các ảnh của ta để có được tập dữ liệu đầu vào như này
- Dữ liệu load về cần phải kiểm tra số mẫu của nó và kích thước H, W của nó nhằm mục đích reshape lại để có được kích thước đúng như yêu cầu đầu vào của keras
- **Chuẩn hóa giá trị Y_train, Y_val, Y_test về one hot**
- Lưu ý : **Kích thước của ảnh cũng chính là kích thước của dữ liệu sau khi chuyển thành ma trận**, ví dụ ảnh có kích thước là 28*28 thì khi chuyển thành ma trận sẽ là 28*28*3 nếu là ảnh màu và 28*28*1 nếu là ảnh xám . **Nếu transfer learning model thì phải điều chỉnh lại kích thước sao cho phù hợp với đầu vào của model đó .**

```
# Load ảnh và resize về đúng kích thước mà VGG 16 cần là (224,224)
list_image = []
for (j, imagePath) in enumerate(image_path):
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)

    image = np.expand_dims(image, 0)
    image = imagenet_utils.preprocess_input(image)

    list_image.append(image)

list_image = np.vstack(list_image)
```

2. Load dữ liệu MNIST

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_val, y_val = X_train[50000:60000,:], y_train[50000:60000]
X_train, y_train = X_train[:50000,:], y_train[:50000]
print(X_train.shape) # (50000, 28, 28)
# 50.000 ảnh => cho ra 50.000 ma trận , mỗi ma trận có kích thước là W*H = 28*28 , đây là ảnh xám nên chỉ có W và H (D=1)
# Nếu là ảnh màu thì W*H*D với D = 3
# 3. Reshape lại dữ liệu cho đúng kích thước mà keras yêu cầu
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1) # (N, W, H, D) = (50.000, 28, 28, 1)
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1) # (N, W, H, D) = (10.000, 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1) # (N, W, H, D) = (10.000, 28, 28, 1)
# 4. One hot encoding label (Y)
Y_train = np_utils.to_categorical(y_train, 10)
Y_val = np_utils.to_categorical(y_val, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print('Dữ liệu y ban đầu ', y_train[0])
print('Dữ liệu y sau one-hot encoding ', Y_train[0])
```

3. Build Model

Thiết kế model

- **model = Sequential()** : Khai báo
- **model.add(Conv2D(F, (k, k), activation='sigmoid', input_shape=(X_train.shape[1:])))** : Lớp đầu tiên
 - hoặc **model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:]), strides=(a, b), padding='valid'))**
 - **input_shape** : (W,H,D)
 - W là weight của mỗi ma trận
 - H là height của mỗi ma trận
 - D là depth là độ sâu , D=1 nếu là ảnh xám và D=3 nếu là ảnh màu
 - **F, (k, k)** : **Can Change to Optimize**
 - F : Số lượng kernel
 - k : kích thước của kernel w : k*k
 - **activation** : hàm kích hoạt **Can Change to Optimize** **Tùy vào bài toán**
 - **'relu'**: Hàm kích hoạt Rectified Linear Unit (ReLU) là một trong những hàm kích hoạt phổ biến nhất trong deep learning, được sử dụng để giảm hiện tượng biến mất gradient và tăng tốc độ hội tụ của mạng nơ-ron.
 - **'sigmoid'**: Hàm kích hoạt Sigmoid chuyển đổi giá trị đầu vào thành một khoảng giữa 0 và 1, thường được sử dụng trong các lớp kết thúc của mạng nơ-ron để đưa ra dự đoán xác suất.
 - **'tanh'**: Hàm kích hoạt Tanh chuyển đổi giá trị đầu vào thành một khoảng giữa -1 và 1, giống như sigmoid nhưng có dải giá trị rộng hơn.
 - **'softmax'**: Hàm kích hoạt Softmax thường được sử dụng trong lớp cuối cùng của mạng nơ-ron phân loại nhiều lớp. Nó chuyển đổi giá trị đầu vào thành một phân phối xác suất trên các lớp đầu ra.
 - **'linear'**: Hàm kích hoạt Linear (tức là không có hàm kích hoạt) giữ nguyên đầu vào mà không thực hiện bất kỳ biến đổi nào.
 - Others
 - **strides và padding** : (nếu có, không chỉ ở lớp đầu tiên có thể ở các lớp khác nữa) **Can Change to Optimize**
 - **strides(a,b)**
 - Mặc định là (1,1) nếu không được khai báo
 - Di chuyển theo chiều ngang a bước
 - Di chuyển theo chiều dọc b bước
 - **Padding**
 - Mặc định là valid (không có)
 - **'valid'** : Không có
 - **'same'** : (1,1)
 - (a,a) : kích thước là a*a
- **model.add(Conv2D(F, (k, k), activation='sigmoid'))** : Lớp thứ 2 trở đi . **Can Change to Optimize**
 - Tương tự , chỉ là không có **input_shape**
- **model.add(MaxPooling2D(pool_size=(p,p)))** : Phương pháp pooling **Can Change to Optimize**
 - **pool_size** : Kích thước của ma trận pooling (p*p) . **Can Change to Optimize**
- 1. **MaxPooling2D** : Phép phân tầng thường được sử dụng trong mạng nơ-ron tích chập (CNN) để giảm kích thước của đầu vào và trích xuất các đặc trưng quan trọng.

2. **AveragePooling2D**: Thực hiện pooling bằng cách tính trung bình các giá trị trong mỗi vùng.
 3. **GlobalAveragePooling2D**: Tương tự như AveragePooling2D, nhưng thực hiện pooling trên toàn bộ đầu vào thay vì trên các cửa sổ cụ thể.
 4. **GlobalMaxPooling2D**: Tương tự như MaxPooling2D, nhưng thực hiện pooling trên toàn bộ đầu vào thay vì trên các cửa sổ cụ thể.
 5. **MaxPooling1D** và **AveragePooling1D**: Tương tự như MaxPooling2D và AveragePooling2D, nhưng dùng cho dữ liệu 1D.
 ⇒ Các hàm pooling này được sử dụng để giảm kích thước của đầu vào và trích xuất các đặc trưng quan trọng, giúp giảm chi phí tính toán và nguy cơ overfitting.
- **model.add(Flatten())** : Lớp Flatten trong mạng nơ-ron là một lớp chuyên đổi, nó được sử dụng để chuyển đổi dữ liệu từ định dạng tensor nhiều chiều thành định dạng vector một chiều (Vector đặc trưng) .
 - **model.add(Dense(n, activation='sigmoid'))** : Lớp kết cuối . Khai báo lớp fully connected . Tùy vào bài toán
 - o n : Số nodes của layer , có thể tùy chỉnh Can Change to Optimize
 - o activation : sigmoid cho giá trị output là 0 hoặc 1
 - **model.add(Dense(n, activation='softmax'))** : Lớp cuối . Tùy vào bài toán
 - o n : Số nodes của layer output , tùy chỉnh theo yêu cầu bài toán bao nhiêu nhãn .
 - o activation : softmax cũng phụ thuộc vào yêu cầu bài toán bao nhiêu nhãn thì nên dùng activation function nào
 - **model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])** : Xác định hàm loss , thuật toán tối ưu và phương pháp đánh giá Can Change to Optimize
 - o loss : softmax cũng phụ thuộc vào yêu cầu bài toán bao nhiêu nhãn thì nên dùng activation function nào Can Change to Optimize
 1. **(categorical_crossentropy)** Categorical Crossentropy: Thích hợp cho bài toán phân loại nhiều lớp, trong đó mỗi mẫu dữ liệu có thể thuộc vào một trong các lớp.
 2. **(binary_crossentropy)** Binary Crossentropy: Thích hợp cho bài toán phân loại nhị phân, trong đó mỗi mẫu dữ liệu chỉ thuộc vào một trong hai lớp.
 3. **(sparse_categorical_crossentropy)** Sparse Categorical Crossentropy : Tương tự như categorical crossentropy nhưng sử dụng với dữ liệu mục tiêu ở dạng số nguyên (integer).
 4. **(mse)** Mean Squared Error: Thích hợp cho bài toán regression, trong đó mục tiêu là một giá trị liên tục.
 5. **(mae)** Mean Absolute Error : Tương tự như mean squared error nhưng tính toán độ lỗi bằng giá trị tuyệt đối thay vì bình phương.
 6. **(hinge)** Hinge Loss: Thích hợp cho bài toán phân loại nhị phân trong SVM (Support Vector Machine).
 7. **(huber)** Huber Loss: Sự kết hợp giữa mean squared error và mean absolute error, thích hợp cho bài toán regression với dữ liệu nhiễu.
 - o optimizer Can Change to Optimize
 1. **(sgd)** Stochastic Gradient Descent (SGD): Là thuật toán tối ưu cơ bản nhất trong deep learning. Nó cập nhật trọng số của mạng bằng cách tính toán gradient của hàm mất mát trên từng mẫu dữ liệu đầu vào.
 2. **(adam)** Adam: Kết hợp cả momentum và RMSprop, Adam là một trong những thuật toán tối ưu phổ biến và hiệu quả. Nó tính toán tỉ lệ học tập tương ứng cho từng tham số của mạng.

3. **(rmsprop)** RMSprop: Thuật toán này điều chỉnh tỉ lệ học tập của từng tham số dựa trên giá trị trung bình bình phương của gradient.
 4. **(adagrad)** Adagrad: Tối ưu tỉ lệ học tập của từng tham số dựa trên lịch sử của gradient của từng tham số.
 5. **(adadelta)** Adadelta: Tương tự như Adagrad, nhưng điều chỉnh tỉ lệ học tập theo tỉ lệ cập nhật trọng số trước đó.
 6. **(adam)** Nadam: Kết hợp cả Adam và Nesterov Momentum, Nadam là một biến thể hiệu quả của Adam.
 7. **(sgd)** SGD with Momentum: Cải thiện SGD bằng cách tích hợp momentum để giảm thiểu dao động trong quá trình tối ưu.
 8. **(adamax)** Adaptive Moment Estimation (Adamax): Một biến thể của Adam, được thiết kế để hoạt động tốt trên các mạng có lớp ẩn rất sâu.
- **metrics** Can Change to Optimize
 1. **(precision)** Precision: Tỉ lệ giữa số lượng các điểm dự đoán đúng thuộc một lớp cụ thể và tổng số điểm được dự đoán thuộc lớp đó. Precision cao hơn cho thấy rằng mô hình ít phân loại sai các điểm vào lớp đó.
 2. **(recall)** Recall (Sensitivity) : Tỉ lệ giữa số lượng các điểm dự đoán đúng thuộc một lớp cụ thể và tổng số điểm thực sự thuộc lớp đó. Recall cao hơn cho thấy rằng mô hình phát hiện được nhiều điểm thực sự của lớp đó.
 3. **(f1_score)** F1-score : Trung bình điều hòa giữa Precision và Recall. F1-score cao hơn khi cả Precision và Recall đều cao.
 4. **(confusion_matrix)** Confusion Matrix: Bảng hiển thị số lượng các dự đoán đúng và sai cho từng lớp.
 5. ROC Curve (Receiver Operating Characteristic Curve) (Không có trong keras) : Biểu đồ thể hiện mối liên hệ giữa tỷ lệ True Positive Rate (TPR) và tỷ lệ False Positive Rate (FPR) khi ngưỡng quyết định thay đổi.
 6. AUC (Area Under Curve) (Không có trong keras): Diện tích nằm dưới đường ROC Curve, thường được sử dụng để đánh giá hiệu suất của mô hình phân loại.

Huấn luyện model

- **`h = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), batch_size=32, epochs=10, verbose=1)`**
 - **batch_size** : Sau số batch_size này sẽ cập nhật lại trọng số một lần , ví dụ có N mẫu dữ liệu thì sẽ có $\frac{N}{batch_size}$ lần cập nhật trọng số Can Change to Optimize
 - **epochs** : Số epochs làn lặp qua toàn bộ dữ liệu Can Change to Optimize
 - **verbose** : Cách hiển thị quá trình training
 1. `verbose=0` : Không hiển thị bất kỳ thông tin nào về quá trình huấn luyện trên màn hình.
 2. `verbose=1` : Hiển thị tiến trình của quá trình huấn luyện trên màn hình. Thông tin bao gồm số lần lặp lại (epoch), loss function và các metrics đã chọn.
 3. `verbose=2` : Hiển thị thông tin của mỗi epoch trên màn hình, nhưng không hiển thị chi tiết của mỗi lần lặp lại trong epoch.
 4. `verbose=3` : Chỉ hiển thị thông báo sau khi mỗi epoch kết thúc, mà không hiển thị bất kỳ thông tin nào trong quá trình huấn luyện của các epoch.
- **`Expand : history = model.fit(X_train, Y_train, batch_size=32, epochs=10, verbose=1, validation_data=(X_val, Y_val), callbacks=[checkpoint, early_stopping], shuffle=True, initial_epoch=0, validation_freq=1, class_weight=None, sample_weight=None, validation_split=0.2, steps_per_epoch=None, validation_steps=None)`**

- **callbacks**: Một danh sách các callbacks được sử dụng trong quá trình huấn luyện. Callbacks là các hàm được gọi trong mỗi epoch của quá trình huấn luyện, cho phép bạn thực hiện các tác vụ như lưu model, điều chỉnh learning rate, hoặc ghi log.
 - **shuffle**: Xác định xem dữ liệu huấn luyện có được xáo trộn sau mỗi epoch hay không. Mặc định là True.
 - **initial_epoch**: Epoch đầu tiên mà quá trình huấn luyện bắt đầu. Điều này hữu ích khi bạn muốn tiếp tục huấn luyện từ một checkpoint đã lưu trước đó.
 - **validation_freq**: Số lượng epoch giữa mỗi lần kiểm tra dữ liệu validation. Mặc định là 1.
 - **class_weight**: Trọng số được áp dụng cho mỗi lớp trong hàm mất mát. Điều này hữu ích khi một lớp có số lượng mẫu ít hơn so với các lớp khác.
 - **sample_weight**: Trọng số được áp dụng cho mỗi mẫu dữ liệu. Điều này hữu ích khi bạn muốn thay đổi mức độ quan trọng của từng mẫu trong quá trình huấn luyện.
 - **validation_split**: Phần trăm dữ liệu được sử dụng để làm dữ liệu validation từ dữ liệu huấn luyện.
 - **steps_per_epoch**: Số lượng batch mỗi epoch. Mặc định là None, và nếu là None, số lượng batch sẽ được tính dựa trên kích thước của dữ liệu huấn luyện và batch_size.
 - **validation_steps**: Số lượng batch mỗi lần đánh giá dữ liệu validation. Mặc định là None.

EarlyStopping

Tham khảo : <C:\Users\ADMIN\Downloads\DeepLearning\Research\Voice Recognition\nlp-speech-recognition-model-development.ipynb>

```
from keras.callbacks import EarlyStopping, ModelCheckpoint

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, min_delta=0.0001)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', verbose=1, save_best_only=True, mode='max')

history=model.fit(x_tr, y_tr, epochs=100, callbacks=[es,mc], batch_size=32, validation_data=(x_val,y_val))
```

Đánh giá model

- score = model.evaluate(X_test, Y_test, verbose=0)

Dự đoán

- y_predict = model.predict(X_test[100].reshape(1,28,28,1))

Trực quan hóa và lưu model

- model.save('my_model.h5')
- from keras.models import load_model
- loaded_model = load_model('my_model.h5')

```
# 5. Định nghĩa model
model = Sequential()

# Thêm Convolutional layer với 32 kernel, kích thước kernel 3*3
# dùng hàm sigmoid làm activation và chỉ rõ input_shape cho layer đầu tiên
```

```

# model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(28,28,1)))
model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:])))
# 32, (3, 3) chính là 32 kernel w với kích thước là 3*3

# Thêm Convolutional layer
model.add(Conv2D(32, (3, 3), activation='sigmoid'))

# Thêm Max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))
# MaxPooling2D
# kích thước của pooling là 2*2

# Flatten layer chuyển từ tensor sang vector
model.add(Flatten())

# Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid
model.add(Dense(128, activation='sigmoid'))

# Output layer với 10 node và dùng softmax function để chuyển sang xác suất.
model.add(Dense(10, activation='softmax'))

# 6. Compile model, chỉ rõ hàm loss_function nào được sử dụng, phương thức
# dùng để tối ưu hàm loss function.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 7. Thực hiện train model với data
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
               batch_size=32, epochs=10, verbose=1)

# 8. Vẽ đồ thị loss, accuracy của training set và validation set
fig = plt.figure()
numOfEpoch = 10
plt.plot(np.arange(0, numOfEpoch), H.history['loss'], label='training loss')
plt.plot(np.arange(0, numOfEpoch), H.history['val_loss'], label='validation loss')
plt.plot(np.arange(0, numOfEpoch), H.history['accuracy'], label='accuracy') # sử dụng từ khóa accuracy thay vì acc
plt.plot(np.arange(0, numOfEpoch), H.history['val_accuracy'], label='validation accuracy') # sử dụng từ khóa val_accuracy thay
vì val_acc
plt.title('Accuracy and Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss|Accuracy')
plt.legend()

# 9. Đánh giá model với dữ liệu test set
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)

```

```

# 10. Dự đoán ảnh
plt.imshow(X_test[100].reshape(28,28), cmap='gray')

y_predict = model.predict(X_test[100].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

# Lưu model
model.save('my_model.h5')

from keras.models import load_model

# Load model
loaded_model = load_model('my_model.h5')

# 10. Dự đoán ảnh
plt.imshow(X_test[0].reshape(28,28), cmap='gray')

y_predict = loaded_model.predict(X_test[0].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

```

Bỗ sung

Lab4_Self_Driving_Car

Import

```

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, Dropout, Dense, Flatten
from keras.regularizers import l2

from Self_Driving_Car.utils import INPUT_SHAPE, batch_generator

```

Xây dựng model

```

# Xây dựng model
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))

```

```

model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
model.summary()

```

Giải thích

- Lambda Layer:** Lambda layer được sử dụng để thực hiện một phép biến đổi đặc biệt cho dữ liệu đầu vào. Trong trường hợp này, nó được sử dụng để chuẩn hóa dữ liệu đầu vào bằng cách chia mỗi giá trị đi 127.5 và trừ đi 1.0. Điều này giúp đưa dữ liệu về miền [0, 1] thành miền [-1, 1].
- Conv2D Layers:** Đây là các lớp tích chập 2D. Mỗi lớp Conv2D được cấu hình với một số lượng bộ lọc (filters), kích thước kernel (kernel_size), và hàm kích hoạt (activation). Các lớp này được sử dụng để trích xuất đặc trưng từ dữ liệu hình ảnh.
- Subsampling:** Tham số `subsample=(2, 2)` trong các lớp Conv2D chỉ ra rằng mỗi lần lớp tích chập được áp dụng, dữ liệu sẽ được giảm đi một nửa theo cả hai chiều (chiều dọc và chiều ngang). Điều này giúp giảm kích thước của dữ liệu và làm cho mô hình huấn luyện nhanh hơn.
- Dropout Layers:** Lớp Dropout được sử dụng để ngẫu nhiên "tắt" một số đơn vị nơ-ron trong quá trình huấn luyện. Điều này giúp tránh overfitting bằng cách làm cho mô hình trở nên đơn giản hóa và dễ dàng tổng quát hóa.
- Flatten Layer:** Lớp Flatten được sử dụng để chuyển đổi dữ liệu từ dạng tensor 2D (tích chập) sang dạng vector 1D, là bước chuẩn bị cho các lớp Dense tiếp theo.
- Dense Layers:** Các lớp Dense là các lớp nơ-ron kết nối đầy đủ, trong đó mỗi nơ-ron trong lớp trước được kết nối với tất cả các nơ-ron trong lớp tiếp theo. Activation function '**elu'** (**Exponential Linear Unit**) được sử dụng ở đây để tăng tính phi tuyến tính của mô hình.
- Summary:** Lệnh `model.summary()` được sử dụng để hiển thị cấu trúc của mô hình, bao gồm số lượng tham số và kích thước đầu ra của mỗi lớp.

Cấu hình và fit model (training model)

```

nb_epoch = 10
samples_per_epoch = 1000
batch_size = 32
save_best_only = True
learning_rate = 1e-4

# Checkpoint này để nói cho model lưu lại model nếu validation loss thấp nhất
checkpoint = ModelCheckpoint('models/model-{epoch:03d}.h5',
                             monitor='val_loss',
                             verbose=0,
                             save_best_only=save_best_only,
                             mode='auto')

```

```
# Dùng mean_squared_error làm loss function
model.compile(loss='mean_squared_error', optimizer=Adam(lr=learning_rate))

# Train model
H = model.fit_generator(batch_generator(data_dir, X_train, y_train, batch_size, True),
                        steps_per_epoch = samples_per_epoch,
                        epochs = nb_epoch,
                        max_q_size=1,
                        validation_data=batch_generator(data_dir, X_valid, y_valid, batch_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
                        verbose=1)
```

Giải thích

- **ModelCheckpoint** : Hay, Checkpoint này để nói cho model lưu lại model nếu validation loss thấp nhất , `ModelCheckpoint` là một callback trong Keras được sử dụng để lưu trọng số của mô hình sau mỗi epoch.
 1. **'models/model-{epoch:03d}.h5'**: Đây là đường dẫn mẫu để lưu trọng số của mô hình. Trong đó, `{epoch:03d}` là một định dạng chuỗi cho biết số thứ tự của epoch, với ba chữ số, ví dụ như '001', '002',... '100', '101',...
 2. **'monitor='val_loss'**: Đây là chỉ mục mà mô hình sẽ theo dõi để quyết định xem liệu trọng số mới đã được lưu lại có tốt hơn so với trọng số cũ hay không. Trong trường hợp này, mô hình sẽ kiểm tra giá trị của hàm loss trên tập validation.
 3. **'verbose=0'**: Điều này xác định cách thông báo khi mô hình lưu trọng số. Trong trường hợp này, `verbose=0` có nghĩa là không có thông báo nào được hiển thị khi trọng số được lưu lại.
 4. **'save_best_only=save_best_only'**: Tham số này xác định xem liệu chỉ những trọng số tốt nhất (đạt được qua các epoch) có nên được lưu lại hay không. Nếu `save_best_only=True`, chỉ có khi có một sự cải thiện đáng kể trong chỉ số giám sát (ở đây là 'val_loss') so với epoch trước đó mới lưu trọng số.
 5. **'mode='auto'**: Xác định cách quyết định trạng thái tốt nhất. Ở đây, 'auto' tức là sẽ tự động xác định với chỉ số giám sát là 'val_loss', khi giá trị của nó giảm đi, mô hình sẽ được coi là cải thiện.

- **optimizer=Adam(lr=learning_rate) : Có thể tùy chỉnh ở thuật toán tối ưu**

- Expand

```
from keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-8, decay=1e-6, amsgrad=False)

model.compile(loss='mean_squared_error', optimizer=optimizer)
```

1. **'learning_rate'** (float): Tỷ lệ học (learning rate) cho mạng neural.
2. **'beta_1'** (float, $0 < \text{beta} < 1$): Hệ số giảm mômentum thứ nhất.

3. `beta_2` (float, $0 < \text{beta} < 1$): Hệ số giảm mômentum thứ hai.
4. `epsilon` (float): Số nhỏ được thêm vào tên lượng để tránh chia cho 0.
5. `decay` (float): Tỷ lệ giảm học theo thời gian.
6. `amsgrad` (boolean): Có sử dụng phiên bản cố định của Adam hay không.

- **keras.fit() and keras.fit_generator() - GeeksforGeeks** : Toàn bộ tham số và giải thích của hàm đều có ở đây
- **H = model.fit_generator(batch_generator(data_dir, X_train, y_train, batch_size, True),**

```
steps_per_epoch = samples_per_epoch,
epochs = nb_epoch,
max_q_size=1,
validation_data=batch_generator(data_dir, X_valid, y_valid, batch_size, False),
nb_val_samples=len(X_valid),
callbacks=[checkpoint],
verbose=1)
```

 1. **`batch_generator(data_dir, X_train, y_train, batch_size, True)`**: Một hàm sinh dữ liệu đầu vào (input data) và nhãn tương ứng (labels) theo từng batch từ tập huấn luyện (training set).
 2. **`steps_per_epoch`**: Số lượng bước (steps) sẽ được thực hiện trong mỗi epoch. Một bước tương ứng với việc đưa một batch dữ liệu vào mạng và cập nhật trọng số.
 3. **`epochs`**: Số lượng epochs (vòng lặp) sẽ được thực hiện trong quá trình huấn luyện. Mỗi epoch tương ứng với việc sử dụng toàn bộ dữ liệu huấn luyện một lần.
 4. **`max_q_size`**: Số lượng batches tối đa được phép trong hàng đợi (queue). Điều này ảnh hưởng đến việc xử lý song song của các batch.
 5. **`validation_data`**: Dữ liệu dùng để đánh giá hiệu suất của mô hình sau mỗi epoch.
 6. **`nb_val_samples`**: Số lượng mẫu dữ liệu trong tập validation.
 7. **`callbacks`**: Danh sách các callback được gọi trong quá trình huấn luyện, trong trường hợp này là **`checkpoint`** để lưu lại model có performance tốt nhất.
 8. **`verbose`**: Cách hiển thị quá trình huấn luyện, có thể là 0 (im silent), 1 (hiển thị progress bar), hoặc 2 (hiển thị số liệu mỗi epoch).- **classification_report** : In ra dạng bảng số liệu đánh giá một cách nhanh chóng

Đánh giá model
preds = model.predict(x_test) # DỰ ĐOÁN VÀ ĐÁNH GIÁ
print(classification_report(y_test, preds))
2]
precision recall f1-score support
0 0.89 1.00 0.94 17
1 0.91 0.77 0.83 13
2 0.95 0.95 0.95 19
3 0.62 0.91 0.74 11
4 0.87 0.93 0.90 14
5 0.86 0.86 0.86 14
6 1.00 1.00 1.00 11
7 0.92 0.85 0.88 13
8 1.00 0.95 0.97 20
9 1.00 0.94 0.97 18
10 0.91 1.00 0.95 10
11 1.00 0.94 0.97 16
12 0.84 0.94 0.89 17
13 1.00 1.00 1.00 19
14 1.00 0.96 0.98 27
15 0.75 0.69 0.72 13
16 1.00 0.85 0.92 20
avg / total 0.93 0.92 0.92 272

Danh sách các tham số có thể thay đổi để tối ưu model

Ví dụ với bài toán nhận diện chữ số từ 0 đến 9

Danh sách các tham số có thể thay đổi để tối ưu model

- Thay đổi số lượng kernel : k (32) và kích thước kernel : w(3, 3) . model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:])))
- Thay đổi hàm kích hoạt tại các lớp tích chập Conv2D : activation (sigmoid) bằng các hàm khác
- Pooling : MaxPooling2D thay bằng phương pháp khác ví dụ AveragePooling2D (trung bình) và thay đổi kích thước pooling
- Số lượng node trong các lớp Fully Connected (128) có thể thay đổi thành một số nodes khác
- Hàm loss và optimizer: Thủ nghiệm với các hàm loss khác nhau như 'categorical_crossentropy', 'mean_squared_error' và các optimizer khác nhau như 'adam', 'sgd', 'rmsprop' để xem liệu cách lựa chọn này có ảnh hưởng đến hiệu suất của mô hình không.
- Batch_size: Thay đổi batch_size có thể ảnh hưởng đến tốc độ huấn luyện và độ chính xác của mô hình , có thể thay đổi trong tốc độ học và hiệu suất của mô hình.
- Epochs: Thay đổi số lượng epochs có thể giúp bạn xác định khi nào mô hình đã hội tụ và khi nào nó bắt đầu overfitting, có thể giảm số lượng epochs nếu thấy mô hình bắt đầu overfitting trên dữ liệu huấn luyện hoặc tăng số lượng epochs nếu mô hình vẫn chưa hội tụ.
- Metrics, bạn có thể đo lường hiệu suất của mô hình bằng các phương pháp khác nhau, không chỉ là accuracy.
 - o Accuracy: Tỷ lệ dự đoán chính xác trên tất cả các mẫu.
 - o Precision: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số dự đoán dương tính.
 - o Recall: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số mẫu thực sự là dương tính.
 - o F1-score: Trung bình điều hòa giữa Precision và Recall.
 - o ROC-AUC: Diện tích dưới đường cong ROC (Receiver Operating Characteristic), thường được sử dụng cho các bài toán phân loại nhị phân.
 - o Mean Squared Error (MSE): Trung bình của bình phương của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.

- Mean Absolute Error (MAE): Trung bình của giá trị tuyệt đối của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.
- R-squared (R2): Phần trăm phương sai của biến phụ thuộc mà mô hình giải thích được.

Danh sách các tham số không nên thay đổi

- Kích thước dữ liệu đầu vào , đây là các con số gần như cố định , D=1 hay D=3 phụ thuộc vào ảnh xám hay ảnh màu
- Hàm loss của bài toán , ví dụ như đây là bài nhiều hơn 2 nhãn thì dùng categorical cross-entropy , không dùng sigmoid (2 nhãn)
- Thuật toán tối ưu : Adam , thường thì adam là thuật toán tốt nhất trong các thuật toán tối ưu , nên có thay đổi thì có thể accuracy cũng không thay đổi mấy
- Các lớp Dense cuối như 2 lớp cuối : Các hàm kích hoạt của nó thì sigmoid sau đó đến softmax là không nên thay đổi vì nó liên quan đến yêu cầu bài toán
- Số node của lớp Dense cuối (10) phụ thuộc vào số nhãn cần nhận diện của bài toán

CNN Models

NTH	CNN Model Name	Version	Public kiến trúc	Parameter
1	ResNet	ResNet18, ResNet34, ResNet50, ResNet101, ResNet152	Public	25.6M
2	VGGNet	VGG11, VGG13, VGG16, VGG19	Public	138M
3	MobileNet		Public	4.2M
4	Inception	InceptionV1 (GoogLeNet), InceptionV2, InceptionV3, InceptionV4	Public	11.2M
5	AlexNet		Public	61M
6	DenseNet	DenseNet-121, DenseNet-169, DenseNet-201, DenseNet-264	Public	27.2M
7	SqueezeNet		Public	1.2M
8	ShuffleNet	ShuffleNetV1, ShuffleNetV2	Public	1.4M
9	NASNet	NASNet-A, NASNet-B	Public	88.9M
10	EfficientNet	EfficientNetB0, EfficientNetB1, EfficientNetB2, EfficientNetB3, EfficientNetB4, EfficientNetB5, EfficientNetB6, EfficientNetB7	Public	5.3M
11	ConvNeXt		Public	5.2M
12	SeNet	SeNet50(27.6M), SeNet101(48.5M), SeNet154(115.8M)	Public	

Dimensionality Reduction (Giảm chiều dữ liệu)

Có nhiều phương pháp khác nhau để giảm chiều dữ liệu. Dưới đây là một số phương pháp phổ biến:

1. **Principal Component Analysis (PCA)**:

- PCA là một trong những phương pháp phổ biến nhất để giảm chiều dữ liệu.
- Nó tìm ra các thành phần chính của dữ liệu mà giữ lại được phần lớn thông tin của dữ liệu ban đầu.

2. **Linear Discriminant Analysis (LDA):**

- LDA cũng là một phương pháp phổ biến để giảm chiều dữ liệu.
- Khác với PCA, LDA cố gắng tối đa hóa độ phân biệt giữa các lớp trong dữ liệu.

3. **t-distributed Stochastic Neighbor Embedding (t-SNE):**

- t-SNE là một phương pháp không tuyến tính để giảm chiều dữ liệu.
- Nó giữ lại cấu trúc không gian của dữ liệu một cách tốt, đặc biệt là trong việc biểu diễn các dữ liệu có kích thước lớn xuống không gian có kích thước thấp hơn.

4. **Non-negative Matrix Factorization (NMF):**

- NMF là một phương pháp giảm chiều dữ liệu phổ biến trong các ứng dụng như xử lý hình ảnh và xử lý văn bản.
- Nó giảm chiều dữ liệu bằng cách tìm ra các thành phần không âm của ma trận ban đầu.

5. **Autoencoders:**

- Autoencoders là một lớp mô hình học sâu mà có thể được sử dụng để giảm chiều dữ liệu.
- Trong quá trình huấn luyện, chúng cố gắng học cách biểu diễn lại dữ liệu một cách hiệu quả với chiều dữ liệu thấp hơn.

6. **Random Projection:**

- Phương pháp này giảm chiều dữ liệu bằng cách chiếu dữ liệu xuống một không gian con ngẫu nhiên có kích thước nhỏ hơn.
- Mặc dù không cố gắng giữ lại nhiều thông tin như PCA hoặc LDA, nhưng nó có thể nhanh chóng và hiệu quả cho các bộ dữ liệu lớn.
 - ⇒ Đây chỉ là một số trong số rất nhiều phương pháp giảm chiều dữ liệu khác nhau. Mỗi phương pháp có những ưu điểm và hạn chế riêng, và lựa chọn phương pháp thích hợp thường phụ thuộc vào bản chất của dữ liệu và mục tiêu của bài toán.

PCA (Principal Component Analysis)

PCA (*Principal Component Analysis*) - Dimensionality Reduction : Phân tích thành phần chính – Giảm chiều dữ liệu .

About PCA

Về mặt ứng dụng

- Đúng như với mục đích của PCA giảm chiều dữ liệu của một vector m chiều xuống còn k chiều
- Như vậy ta có thể sử dụng PCA vào các trường hợp
 - + Các bài toán học máy đơn giản , ví dụ X có m đặc trưng , ta muốn giảm nó xuống k đặc trưng . Nghĩa là X có m đặc trưng ví dụ loài hoa thì có chiều dài , màu sắc ,... gồm m đặc trưng như thế sau đó ta sẽ giảm chiều nó xuống thành k đặc trưng
 - + Đối với bài toán nhận diện ảnh vì ảnh được chuyển thành một matrix hoặc tensor . Mặc khác với các model Deep learning thì đầu vào của nó cũng yêu cầu là một matrix hoặc tensor (matrix hoặc tensor fixcel của ảnh) , chính vì thế mà nó thường không được áp dụng cho ảnh .

+ Mà nó thường được áp dụng trong Transfer learning . Cụ thể là sau các lớp Conv , trước khi đến các lớp FCs thì ta cho qua PCA để giảm chiều . Ví dụ đầu ra của VGG sau khi qua flatten (làm phẳng) là một vector với số chiều là (4096,1) ta sẽ cho nó qua PCA trở thành (k,1) với k là số chiều được giảm xuống, sau đó mới cho qua các lớp FCs ví dụ như Dense(512,sigmoid) rồi Dense(10,softmax)

Về mặt toán học

- Tham khảo : <https://machinelearningcoban.com/2017/06/15/pca/>

4. Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

1. Tính vector kỳ vọng của toàn bộ dữ liệu:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$$

2. Trừ mỗi điểm dữ liệu đi vector kỳ vọng của toàn bộ dữ liệu:

$$\hat{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}$$

3. Tính ma trận hiệp phương sai:

$$\mathbf{S} = \frac{1}{N} \hat{\mathbf{X}} \hat{\mathbf{X}}^T$$

4. Tính các trị riêng và vector riêng có norm bằng 1 của ma trận này, sắp xếp chúng theo thứ tự giảm dần của trị riêng.
5. Chọn K vector riêng ứng với K trị riêng lớn nhất để xây dựng ma trận \mathbf{U}_K có các cột tạo thành một hệ trực giao. K vectors này, còn được gọi là các thành phần chính, tạo thành một không gian con gần với phân bố của dữ liệu ban đầu đã chuẩn hóa.
6. Chiếu dữ liệu ban đầu đã chuẩn hóa $\hat{\mathbf{X}}$ xuống không gian con tìm được.
7. Dữ liệu mới chính là toạ độ của các điểm dữ liệu trên không gian mới.

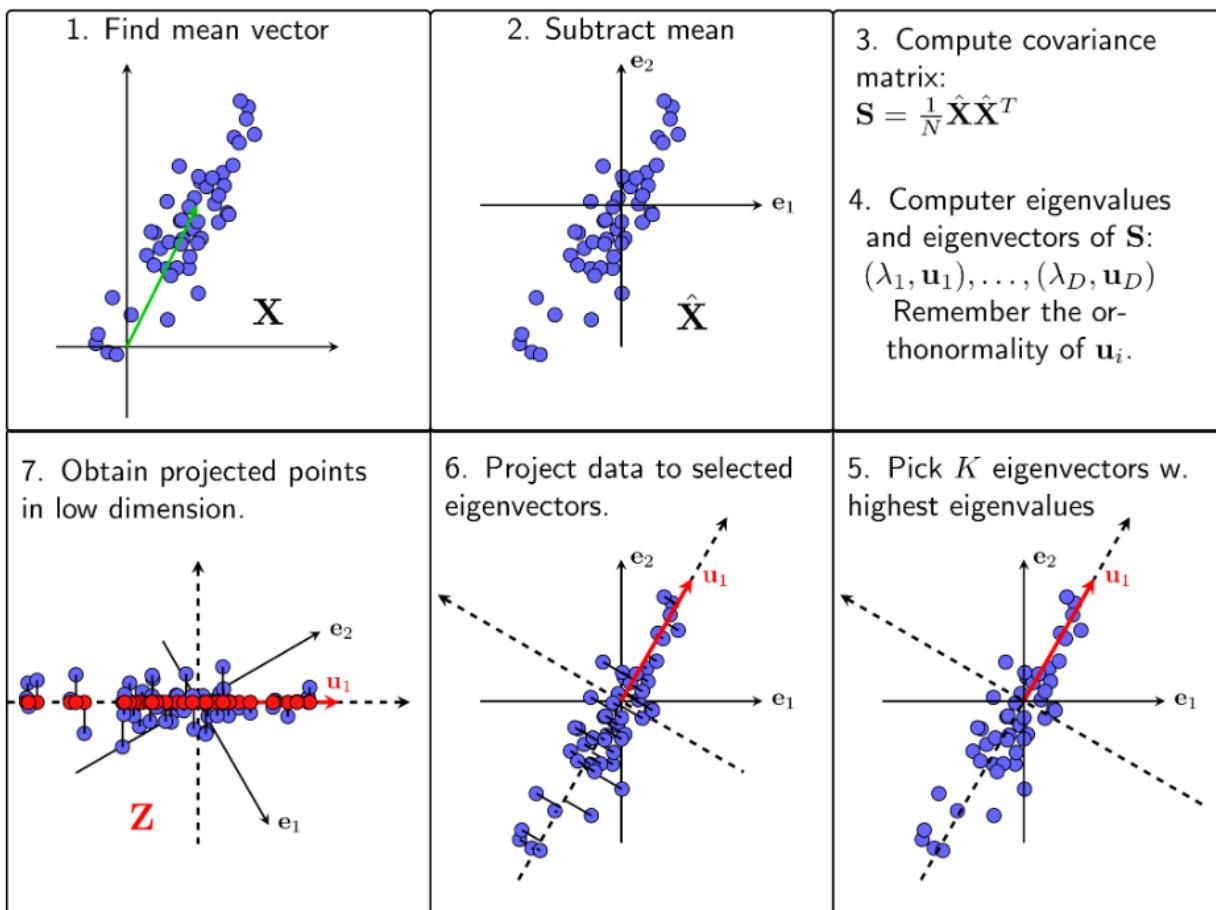
$$\mathbf{Z} = \mathbf{U}_K^T \hat{\mathbf{X}}$$

Dữ liệu ban đầu có thể tính được xấp xỉ theo dữ liệu mới như sau:

$$\mathbf{x} \approx \mathbf{U}_K \mathbf{Z} + \bar{\mathbf{x}}$$

Các bước thực hiện PCA có thể được xem trong Hình dưới đây:

PCA procedure



Hình 5: Các bước thực hiện PCA.

PCA Library

Chi tiết ở file :

C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\PCA - Library.ipynb

Trong `PCA` của `scikit-learn`, hai phương thức `fit_transform()` và `transform()` đều được sử dụng để giảm số chiều của dữ liệu bằng phương pháp phân tích thành phần chính (PCA). Tuy nhiên, chúng có một số khác biệt quan trọng:

1. ***fit_transform():***

- Phương thức `fit_transform()` được sử dụng để tính toán các thành phần chính (principal components) từ dữ liệu đầu vào và sau đó giảm số chiều của dữ liệu xuống theo số lượng thành phần chính được chỉ định.

- Nó kết hợp hai công đoạn: tính toán thành phần chính từ dữ liệu huấn luyện và sử dụng các thành phần chính đó để giảm chiều dữ liệu.

- Thường được sử dụng cho dữ liệu huấn luyện.

2. ***transform():***

- Phương thức `transform()` được sử dụng để giảm số chiều của dữ liệu bằng cách sử dụng các thành phần chính đã được tính toán từ dữ liệu huấn luyện (hoặc từ việc gọi `fit_transform()` trước đó).
- Nó chỉ thực hiện phần giảm chiều của dữ liệu mà không tính toán bất kỳ thứ gì mới từ dữ liệu đầu vào.
- Thường được sử dụng cho dữ liệu kiểm tra hoặc dữ liệu mới mà bạn muốn giảm chiều.

Tóm lại, `fit_transform()` sẽ tính toán các thành phần chính và giảm chiều dữ liệu, trong khi `transform()` chỉ giảm chiều dữ liệu sử dụng các thành phần chính đã được tính toán từ trước.

Một số lưu ý ***Quan Trọng*** :

1. `pca = PCA(n_components=700)` : Giảm xuống k chiều với $k = 700$. Tuy nhiên chưa biết giữ lại được bao nhiêu % phương sai
 - `pca.fit_transform(X)` : Như đã nói, nó có tính toán dựa trên tập dữ liệu nên. Tập dữ liệu X có $X.shape = (n,m)$ thì giảm xuống k chiều với điều kiện *** $k \leq n,m$ *** . Thường được dùng cho tập train. *** fit_transform là huấn luyện cho pca nên trong quá trình giảm chiều thì ít nhất đầu tiên phải fit_transform cho nó***
 - `pca.transform(X)` : Không cần điều kiện gì cả, transform thường được dùng cho tập test và valid. Sử dụng lại fit_transform trước đó để giảm xuống 700 chiều .
⇒ Lưu ý với cách này, ta phải sử dụng : `np.cumsum(pca.explained_variance_ratio_)[-1]` để xem nó giữ lại bao nhiêu % phương sai , thường là phải từ 95% - 99%
2. `pca = PCA(n_components=0.95)` : Giữ lại 95% phương sai và số chiều giảm xuống k sẽ được tự động tính toán cho ta
 - Sử dụng fit_transform thì không dùng lại fit_transform trước đó mà đi huấn luyện lại
 - Sử dụng transform thì sử dụng fit_transform trước đó , lấy số chiều được giảm xuống trước đó của fit_transform .

PCA Implement

Chi tiết ở file :

[C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\PCA - Implement.ipynb](#)

PCA for Transfer learning Feature Extractor

Chi tiết ở file :

[C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\Chapter5_VGG16_feature-extractor.PCA.ipynb](#)

Các cách chuẩn bị dữ liệu để train

Cách 1 : Chia dữ liệu theo batch size : Yêu cầu : Gộp toàn bộ dữ liệu vào một folder '`dataset_new`' , file có đặc điểm là bắt đầu tên file bằng tên label Ex : Bluebell.image_0241.jpg
Sau đó chạy file để chia dữ liệu

Xem file :

[C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\Chapter5_VGG16_fine-tune2_Good.ipynb](#)

```
# root_dir = '/content/drive/MyDrive/5_AIADVANCE/GK_AI_Advance/'
root_dir = 'dataset_new'

# Đường dẫn tới thư mục con 'train', 'val' và 'test'
train_dir = os.path.join(root_dir, 'train')
val_dir = os.path.join(root_dir, 'val')
test_dir = os.path.join(root_dir, 'test')
```

```

img_size = (224 , 224) # resize về đúng kích thước mà VGG 16 cần là (224,224)
batch_size = 8
img_shape = (img_size[0] , img_size[1] , 3)

# làm giàu dữ liệu data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1./255) # Chỉ rescale cho dữ liệu kiểm tra
# làm giàu dữ liệu data augmentation

train_gen = train_datagen.flow_from_directory(train_dir , target_size = img_size , class_mode = 'categorical' ,
                                              color_mode = 'rgb' , shuffle = True , batch_size = batch_size)

valid_gen = test_datagen.flow_from_directory(val_dir , target_size = img_size , class_mode = 'categorical' ,
                                              color_mode = 'rgb' , shuffle = True , batch_size = batch_size)

test_gen = test_datagen.flow_from_directory(test_dir , target_size = img_size , class_mode = 'categorical' ,
                                              color_mode = 'rgb' , shuffle = False , batch_size = batch_size)

g_dict = train_gen.class_indices
classes = list(g_dict.keys())
images, labels = next(train_gen)
num_samples = len(images)

plt.figure(figsize=(20, 20))

for i in range(min(16, num_samples)):
    plt.subplot(4, 4, i + 1)
    image = images[i] # Không cần chia lại cho 255
    plt.imshow(image)
    class_index = np.argmax(labels[i]) # Lấy index của nhãn cao nhất
    class_name = classes[class_index]
    plt.title(class_name, color='blue', fontsize=12)
    plt.axis('off')
plt.show()
# freeze VGG model
for layer in myConvNetModelVGG16.layers:
    layer.trainable = False

```

```

# thuật toán tối ưu
optimizer = RMSprop(0.001)
# optimizer = tf.keras.optimizers.Adam()

# Thiết lập ModelCheckpoint
save_best_only = True
checkpoint = ModelCheckpoint('models/myVGG16_model-{epoch:03d}.keras',
                            monitor='val_loss',
                            verbose=1,
                            save_best_only=save_best_only,
                            mode='auto')

myModel.compile(optimizer, 'categorical_crossentropy', metrics=['accuracy']) # ta có thể ghi optimizer =
optimizer, hoặc optimizer là nó tự hiểu
numOfEpoch = 10
H = myModel.fit(
    train_gen,
    steps_per_epoch = len(train_gen),
    validation_data = valid_gen,
    validation_steps = len(valid_gen),
    epochs=numOfEpoch,
    # callbacks=[checkpoint]
)

```

Cách 2 : Load path (không nên theo cách này) :

[Xem file :](#)

C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\Chapter5_VGG16_fine-tune.ipynb

```

# Lấy các đường dẫn đến ảnh.
image_path = list(paths.list_images('dataset/'))

# Đổi vị trí ngẫu nhiên các đường dẫn ảnh
random.shuffle(image_path)
# Đường dẫn ảnh sẽ là dataset/tên_loài_hoa/tên_ảnh ví dụ dataset/Bluebell/image_0241.jpg nên
p.split(os.path.sep)[-2] sẽ lấy ra được tên loài hoa
labels = [p.split(os.path.sep)[-2] for p in image_path]

# Label name
label_names = np.unique(labels)
label_names = [label.split('/')[-1] for label in label_names]

# Chuyển tên các loài hoa thành số
le = LabelEncoder()

```

```

labels = le.fit_transform(labels)

# One-hot encoding
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
# Load ảnh và resize về đúng kích thước mà VGG 16 cần là (224,224)
list_image = []
for (j, imagePath) in enumerate(image_path):
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)

    image = np.expand_dims(image, 0)
    image = imagenet_utils.preprocess_input(image)

    list_image.append(image)

list_image = np.vstack(list_image)
# Chia traing set, test set tỉ lệ 80-20
X_train, X_test, y_train, y_test = train_test_split(list_image, labels, test_size=0.2, random_state=42)

# Làm giàu dữ liệu
# augmentation cho training data
aug_train = ImageDataGenerator(rescale=1./255, rotation_range=30, width_shift_range=0.1,
height_shift_range=0.1, shear_range=0.2,
        zoom_range=0.2, horizontal_flip=True, fill_mode='nearest')
# augmentation cho test
aug_test= ImageDataGenerator(rescale=1./255)
# freeze VGG model
for layer in myConvNetModelVGG16.layers:
    layer.trainable = False

opt = RMSprop(0.001)
# model.compile(opt, 'categorical_crossentropy', ['accuracy'])
myModel.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
numOfEpoch = 5
H = myModel.fit(aug_train.flow(X_train, y_train, batch_size=32), # thay fit_generator bằng fit (version mới
của keras)
    steps_per_epoch=len(X_train)//32,
    validation_data=(aug_test.flow(X_test, y_test, batch_size=32)),
    validation_steps=len(X_test)//32,
    epochs=numOfEpoch)

```

LSTM và Gộp nhiều đặc trưng của một nhãn thành một đặc trưng

Để biểu diễn 10 đặc trưng của 10 ảnh thành một đặc trưng duy nhất, bạn có thể sử dụng một trong những phương pháp sau:

1. **Trung bình hóa (Mean pooling):** Bạn có thể tính trung bình của các đặc trưng của 10 ảnh. Điều này có thể được thực hiện bằng cách tính trung bình của từng chiều của các vector đặc trưng.
2. **Tích vô hướng (Vector concatenation):** Bạn có thể ghép nối các vector đặc trưng của 10 ảnh thành một vector đặc trưng lớn hơn. Điều này tạo ra một vector đặc trưng dài hơn, kết hợp thông tin từ tất cả các ảnh.
3. **Max pooling:** Thay vì lấy trung bình, bạn có thể chọn giá trị lớn nhất từ mỗi chiều của các vector đặc trưng của 10 ảnh. Điều này giúp giữ lại các thông tin quan trọng nhất từ mỗi vector.
4. **PCA (Principal Component Analysis):** Bạn có thể sử dụng PCA để giảm chiều dữ liệu từ các vector đặc trưng của 10 ảnh xuống thành một vector đặc trưng duy nhất có kích thước mong muốn.
5. **LSTM (Long Short-Term Memory):** Đối với dữ liệu tuần tự như các vector đặc trưng của các ảnh, bạn có thể sử dụng mạng LSTM để học mối quan hệ giữa chúng và tạo ra một vector đặc trưng duy nhất.
⇒ Tùy thuộc vào loại dữ liệu và mô hình mà bạn sử dụng, một phương pháp có thể hoạt động tốt hơn so với các phương pháp khác. Đôi khi, thử nghiệm và so sánh hiệu suất của các phương pháp khác nhau trên tập dữ liệu thử nghiệm có thể giúp bạn chọn ra phương pháp tốt nhất cho bài toán của mình.

Danh sách các thuật toán tối ưu hóa được sử dụng trong quá trình huấn luyện mạng neural network để tối ưu hóa hàm mất mát

SGD, Momentum, Adagrad, Adadelta, RMSprop và Adam không phải là hàm mất mát hoặc hàm kích hoạt, mà chúng là các thuật toán tối ưu hóa được sử dụng trong quá trình huấn luyện mạng neural network để tối ưu hóa hàm mất mát. Cụ thể:

1. **SGD (Stochastic Gradient Descent):** Là thuật toán tối ưu hóa đơn giản, trong đó gradient của hàm mất mát được tính dựa trên một mẫu dữ liệu ngẫu nhiên trong mỗi lần cập nhật.
2. **Momentum:** Cải tiến của SGD bằng cách giữ lại một phần của gradient trước đó để giúp quá trình hội tụ nhanh hơn và tránh trường hợp rơi vào các điểm tối thiểu cục bộ.
3. **Adagrad (Adaptive Gradient Algorithm):** Tối ưu hóa việc cập nhật trọng số bằng cách điều chỉnh tỷ lệ học (learning rate) cho từng tham số của mạng dựa trên tần suất xuất hiện của gradient của tham số đó.
4. **Adadelta:** Là phiên bản cải tiến của Adagrad, thay đổi cách tính toán tỷ lệ học và giảm vấn đề giảm tỷ lệ học quá nhanh trong quá trình huấn luyện.
5. **RMSprop (Root Mean Square Propagation):** Tối ưu hóa bằng cách chuẩn hóa gradient để điều chỉnh tỷ lệ học cho từng tham số theo tỷ lệ của bình phương các gradient trước đó.

6. **Adam (Adaptive Moment Estimation):** Kết hợp cả ý tưởng của Momentum và RMSprop. Nó tính toán gradient với momentum như trong Momentum và cập nhật tỷ lệ học cho từng tham số như trong RMSprop. Adam thường được sử dụng phổ biến trong các mô hình deep learning với hiệu suất tốt và dễ sử dụng.

KNN và K-means

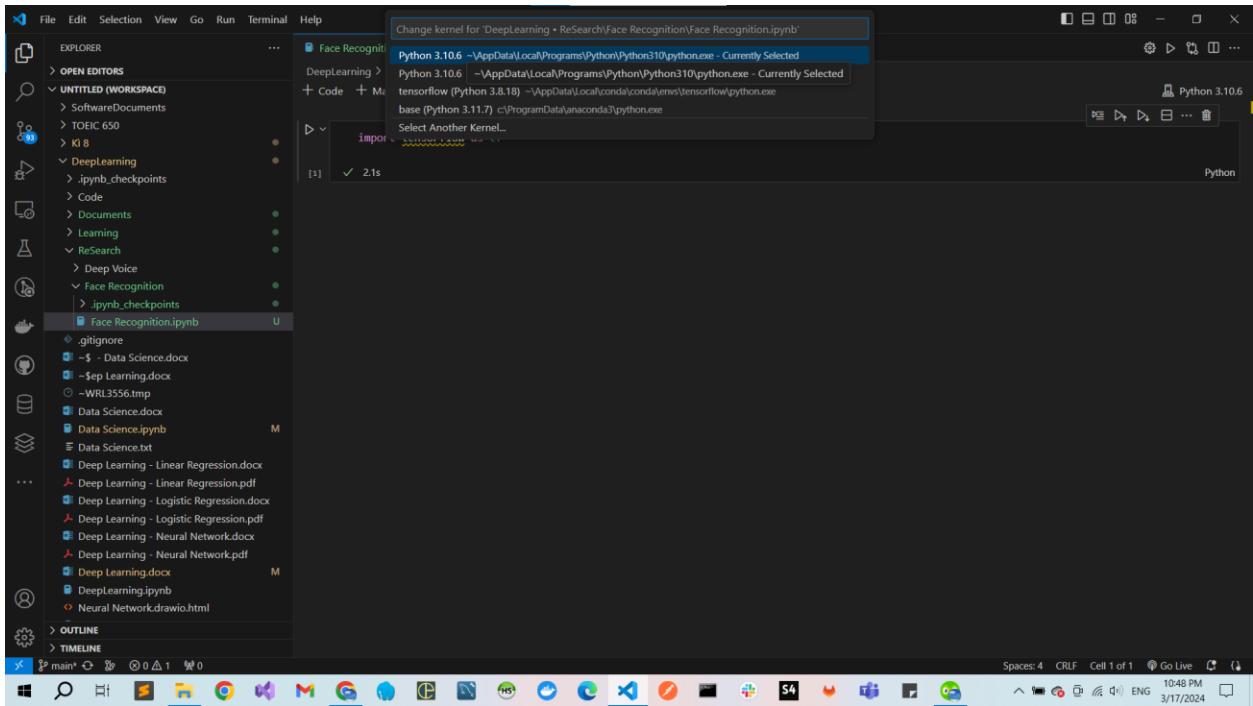
- **K-Nearest Neighbors (KNN)**** là một thuật toán học máy đơn giản được sử dụng cho cả phân loại và dự đoán giá trị. Ý tưởng cơ bản của KNN là dựa vào các điểm dữ liệu gần nhất (*K* láng giềng) để xác định nhãn hoặc giá trị của một điểm dữ liệu mới. Khi có một điểm dữ liệu mới, KNN sẽ tính khoảng cách từ điểm đó đến tất cả các điểm dữ liệu đã biết, sau đó chọn ra *K* điểm gần nhất và dùng đa số hoặc trung bình của nhãn của các điểm láng giềng để dự đoán nhãn cho điểm mới.

- **K-means clustering**** là một thuật toán phân cụm dữ liệu, thường được sử dụng để phân loại dữ liệu không gắn nhãn thành các nhóm dựa trên đặc điểm tương tự của chúng. Thuật toán này bắt đầu bằng cách chọn ngẫu nhiên *K* điểm làm centroids ban đầu, sau đó gán mỗi điểm dữ liệu vào cụm có centroid gần nhất. Tiếp theo, cập nhật centroids bằng cách tính trung bình của tất cả các điểm dữ liệu trong cùng một cụm. Quá trình này được lặp lại cho đến khi centroids không thay đổi đáng kể hoặc số lần lặp đã đạt đến một giới hạn trước định. Kết quả là mỗi điểm dữ liệu sẽ được gán vào một trong *K* cụm, tạo thành các nhóm dữ liệu có đặc điểm tương tự.

Sử dụng Anaconda để cài từng version python tương thích với từng keras và tensorflow
Updating...

Fix & Install

- Sửa lỗi không cài được tensorflow do không cài được wrapt : Cách fix là cài riêng nó <https://stackoverflow.com/questions/59232286/error-cannot-uninstall-wrapt-when-installing-tensorflow-gpu-1-14>
- Cài thành công tensorflow nhưng dùng trong vs code không được : Lí do là chọn khác kernel :



- ⇒ Chọn kernel này là được .
 - Cài deeplake để lấy dữ liệu từ paperwith code

```
Microsoft Windows [Version 10.0.19045.2965]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>pip3 install deeplake
Collecting deeplake
  Downloading deeplake-3.8.25.tar.gz (587 kB)
----- 587.1/587.1 kB 4.6 MB/s eta 0:00:00
    Installing build dependencies ... done
    Getting requirements to build wheel ... done
    Installing backend dependencies ... done
    Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy in c:\users\admin\appdata\local\programs\python\python310\lib\site-packages (from deeplake) (1.26.4)
Requirement already satisfied: pillow in c:\users\admin\appdata\local\programs\python\python310\lib\site-packages (from deeplake) (8.2.0)
Collecting boto3 (from deeplake)
  Downloading boto3-1.34.64-py3-none-any.whl.metadata (6.6 kB)
Collecting click (from deeplake)
  Downloading click-8.1.7-py3-none-any.whl.metadata (3.0 kB)
Collecting pathos (from deeplake)
  Downloading pathos-0.3.2-py3-none-any.whl.metadata (11 kB)
Collecting humbug>=0.3.1 (from deeplake)
  Downloading humbug-0.3.2-py3-none-any.whl.metadata (6.8 kB)
Collecting tqdm (from deeplake)
  Downloading tqdm-4.66.2-py3-none-any.whl.metadata (57 kB)
----- 57.6/57.6 kB 3.2 MB/s eta 0:00:00
Collecting lz4 (from deeplake)
  Downloading lz4-4.3.3-cp310-cp310-win_amd64.whl.metadata (3.8 kB)
Collecting pywt (from deeplake)
  Downloading PyWT-2.8.0-py3-none-any.whl.metadata (4.2 kB)
Collecting pydantic (from deeplake)
  Downloading pydantic-2.6.4-py3-none-any.whl.metadata (85 kB)
----- 85.1/85.1 kB 7 eta 0:00:00
Requirement already satisfied: requests in c:\users\admin\appdata\local\programs\python\python310\lib\site-packages (from deeplake) (2.31.0)
Collecting botocore<1.35.0,>=1.34.64 (from boto3->deeplake)
  Downloading botocore-1.34.64-py3-none-any.whl.metadata (5.7 kB)
Collecting jmespath<2.0.0,>=0.7.1 (from boto3->deeplake)
  Downloading jmespath-2.0.0-py3-none-any.whl.metadata (7.6 kB)
Collecting colorama<1.1.0,>=0.10.0 (from click->deeplake)
  Downloading s3transfer-0.10.1-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: colorama in c:\users\admin\appdata\roaming\python\python310\site-packages (from click->deeplake) (0.4.6)
Collecting pftt<1.7.6.8 (from pathos->deeplake)
  Downloading pftt-1.7.6.8-py3-none-any.whl.metadata (12 kB)
Collecting dill<0.3.8 (from pathos->deeplake)
  Downloading dill-0.3.8-py3-none-any.whl.metadata (10 kB)
Collecting pox<0.3.4 (from pathos->deeplake)
  Downloading pox-0.3.4-py3-none-any.whl.metadata (8.0 kB)
Collecting multiprocess<0.70.10 (from pathos->deeplake)
  Downloading multiprocess-0.70.10-py310-none-any.whl.metadata (7.2 kB)
Collecting annotated-types<0.4.0 (from pydantic->deeplake)
  Downloading annotated-types-0.6.0-py3-none-any.whl.metadata (12 kB)
Collecting pydantic-core<2.16.3 (from pydantic->deeplake)
```

- Reset commit (Reset or delete something commit)

```

PS C:\Users\ADMIN\Downloads\DeepLearning> git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

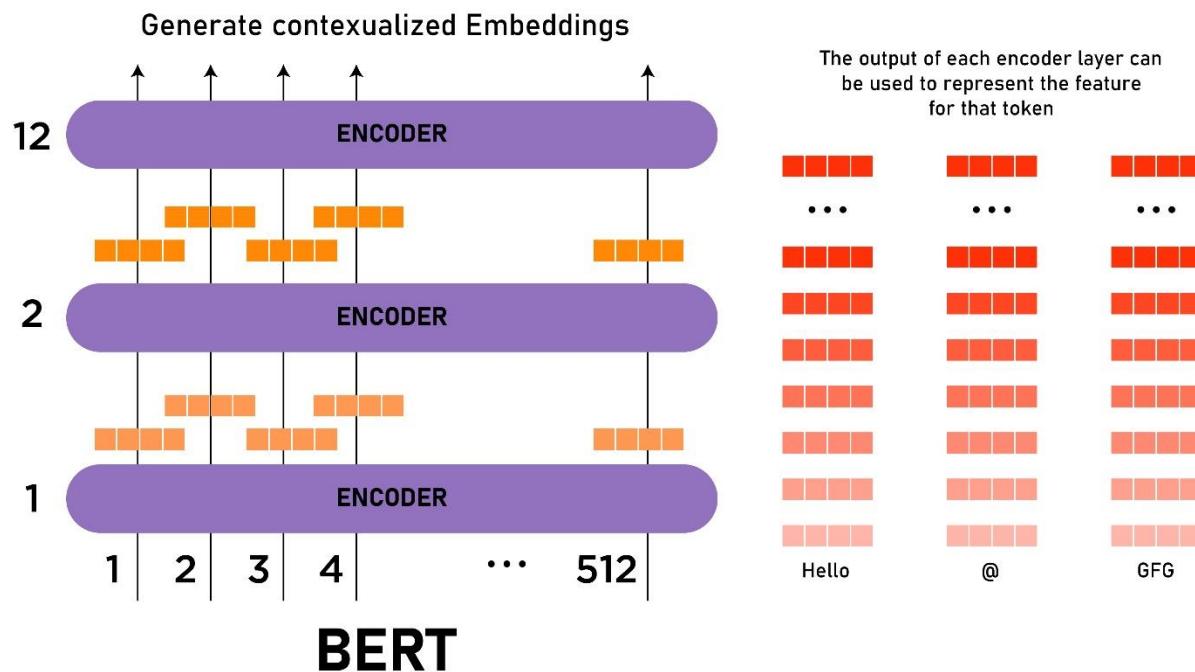
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Research/Flower Classification/VGG16_Flower_Recognition.ipynb

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\ADMIN\Downloads\DeepLearning> git reset --hard HEAD~3
HEAD is now at d63e55f7 update README Face Recognition
PS C:\Users\ADMIN\Downloads\DeepLearning>

```

NLP & BERT

- Embedding text to input



Viết code trong Markdown

- Ta có thể viết một đoạn code của bất kì ngôn ngữ trong markdown bằng cách bọc chúng trong
```[name\_language\_program]  
code...  
```

```

Trong câu lệnh `model.compile()`, bạn đã chỉ định hàm mất mát (loss function) là 'mean_squared_error' và tối ưu hóa mô hình bằng thuật toán Adam với tỷ lệ học (learning rate) là 'learning_rate'. Trong Adam, ngoài tỷ lệ học, còn có một số tham số khác mà bạn có thể tinh chỉnh:
1. 'beta_1': Hỗn số giảm dần cho first moment estimates.
2. 'beta_2': Hỗn số giảm dần cho second moment estimates.
3. 'epsilon': Một số nhỏ được thêm vào tên lượng để tránh chia cho 0.
4. 'decay': Tỷ lệ giảm học theo thời gian (thường được sử dụng với tỷ lệ học không đổi).

Ví dụ:

```python
from keras.optimizers import Adam

Tinh chỉnh tham số beta_1 và beta_2
optimizer = Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999)

Compile model với optimizer được tinh chỉnh
model.compile(loss='mean_squared_error', optimizer=optimizer)
```

Trong trường hợp này, bạn có thể thay đổi 'beta_1', 'beta_2', 'epsilon', và 'decay' nếu cần.

```php
<?php
$1 = 1;

while ($1 < 6) {
 if ($1 == 3) break;
 echo $1;
 $1++;
}
?>
```

```

Kết quả

```

Trong câu lệnh `model.compile()`, bạn đã chỉ định hàm mất mát (loss function) là 'mean_squared_error' và tối ưu hóa mô hình bằng thuật toán Adam với tỷ lệ học (learning rate) là 'learning_rate'. Trong Adam, ngoài tỷ lệ học, còn có một số tham số khác mà bạn có thể tinh chỉnh:
1. 'beta_1': Hỗn số giảm dần cho first moment estimates.
2. 'beta_2': Hỗn số giảm dần cho second moment estimates.
3. 'epsilon': Một số nhỏ được thêm vào tên lượng để tránh chia cho 0.
4. 'decay': Tỷ lệ giảm học theo thời gian (thường được sử dụng với tỷ lệ học không đổi).

Ví dụ:

```python
from keras.optimizers import Adam

Tinh chỉnh tham số beta_1 và beta_2
optimizer = Adam(lr=learning_rate, beta_1=0.9, beta_2=0.999)

Compile model với optimizer được tinh chỉnh
model.compile(loss='mean_squared_error', optimizer=optimizer)
```

Trong trường hợp này, bạn có thể thay đổi `beta_1`, `beta_2`, `epsilon`, và `decay` nếu cần.

```php
<?php
$1 = 1;

while ($1 < 6) {
 if ($1 == 3) break;
 echo $1;
 $1++;
}
?>
```

```

Step by Step Push large file to github

- Tham khảo : <https://git-lfs.com/>
- Tải file : <https://github.com/git-lfs/git-lfs/releases/download/v3.4.1/git-lfs-windows-v3.4.1.exe> và cài đặt nó

- Đi đến folder github chứa các file lớn và chạy lệnh (**Lưu ý : Nên mở git bash dưới quyền admin để chạy các lệnh này**)
- **git lfs track “*.docx”** (nghĩa là thêm tất cả các file docx trong repo github này vào danh sách file lớn) , ta có thể thêm tên file cụ thể . Tương tự với các dạng file khác .
- **git add .gitattributes** (danh sách các file lớn sẽ được thêm vào file này)

```

.gitattributes X id_ed25519.pub
DeepLearning > .gitattributes
You, 14 minutes ago | 1 author (You)
1 *.docx filter=lfs diff=lfs merge=lfs -text
2 *.pdf filter=lfs diff=lfs merge=lfs -text
3 *.h5 filter=lfs diff=lfs merge=lfs -text
4 *.keras filter=lfs diff=lfs merge=lfs -text
5

```

- Với lần đầu tiên có thể cần chạy lệnh này : **git config lfs.https://github.com/NguyenVanManh-AI/Deep-Learning.git/info/lfs.locksverify false** với Deep-Learning là tên repo
- Lúc này sẽ có sự thay đổi
- **git add ...** và **git commit ...** để thêm các thay đổi đó
- **git push origin main** (nếu có yêu cầu nhập mật khẩu thì nhập) , có thể nó bắt nhập đến lần thứ 3 mới được .

```

Git for Windows
Enter passphrase for key '/c/Users/ADMIN/.ssh/id_ed25519':
      g (main)
      519';
done.

*****[REDACTED]*****[REDACTED]

OK Cancel done.

remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:NguyenVanManh-AI/Deep-Learning.git
  8aa6522a..b01d6a6c main -> main

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   Deep_Learning.docx

no changes added to commit (use "git add" and/or "git commit -a")

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git add .

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git commit -m "Step by Step Push Large file to Github"
[main a1079e1f] Step by Step Push Large file to Github
 1 file changed, 2 insertions(+), 2 deletions(-)

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git push origin main
Enter passphrase for key '/c/Users/ADMIN/.ssh/id_ed25519':
Uploading LFS objects:  0% (0/1), 0 B | 0 B/s

```

⇒ KẾT QUẢ : Các file lớn này sẽ được push lên github

```
MINGW64:/c/Users/ADMIN/Downloads/DeepLearning
ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 4 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ git push
Enter passphrase for key '/c/Users/ADMIN/.ssh/id_ed25519':
Uploading LFS objects: 100% (156/156), 536 MB | 10 MB/s, done.
Enumerating objects: 410, done.
Counting objects: 100% (410/410), done.
Delta compression using up to 16 threads
Compressing objects: 100% (211/211), done.
Writing objects: 100% (215/215), 28.18 KiB | 1.08 MiB/s, done.
Total 215 (delta 35), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (35/35), completed with 13 local objects.
To github.com:NguyenVanManh-AI/Deep-Learning.git
  8bcd088a..8aa6522a  main -> main

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ ^C

ADMIN@DESKTOP-CHT52UJ MINGW64 ~/Downloads/DeepLearning (main)
$ |
```

Others

Kim chỉ nam

- **Một đoạn code hay giải thích phức tạp , phải làm cho đến khi nó trở nên đơn giản !**

Hướng dẫn

⇒ Những cái thứ đã có trong sách hay giáo trình thì có thể note đơn giản bằng ảnh hoặc title heading nào đó , để đến lúc cần chỉ cần giờ ra lại đúng trang đó để đọc , không cần phải ghi chép quá chi tiết làm mất thời gian . Hiểu và nhớ mới là điều quan trọng nhất . Chỉ cần note những thứ mà mình hiểu ra được mà sách chưa nói .

- **Có thể mở rộng mục lục ra để tìm cho nhanh**

The screenshot shows a Microsoft Edge browser window with multiple tabs open. The main content area displays a Python script with comments explaining broadcasting and its implementation. To the left, a sidebar shows a 'Table of Contents' for a machine learning tutorial, covering topics from basic Python to neural networks. The bottom of the screen shows the Windows taskbar with various pinned icons.

```

x = np.array([[1,2],[3,4]])
print(np.sum(x)) # Tính tổng tất cả phần tử trong array; prints "10"
print(np.sum(x, axis=0)) # Tính tổng phần tử mỗi hàng; prints "[4 6]"
print(np.sum(x, axis=1)) # Tính tổng phần tử mỗi cột; prints "[3 7]"

2.5.3 Broadcasting
Broadcasting là một kĩ thuật cho phép numpy làm việc với các array có shape khác nhau khi thực hiện các phép toán.

import numpy as np

# Cộng vector v với mỗi hàng của ma trận x, kết quả lưu ở ma trận y.
x = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])
v = np.array([1, 1, 1])
y = np.empty_like(x) # Tạo 1 array có chiều giống x

# Dùng loop để vector v với mỗi hàng của ma trận
for i in range(4):
    y[i, :] = x[i, :] + v

print(y)
# Kết quả của y
# [[ 2  2  4]
# [ 5  5  7]
# [ 8  8 10]
# [11 11 13]]]

# Ở kết quả cho ra giống nhau

2.6 Matplotlib
Thư viện matplotlib dùng để vẽ đồ thị. Phần này tôi đi qua các tính năng cơ bản của module matplotlib.pyplot

```

Con đường học PhD

PhD life 1: Quá trình viết và nhận xét các bài báo khoa học

VS Code show print

- Nếu nội dung dài quá thì có thể sẽ bị ẩn đi vài phần :

The screenshot shows the VS Code interface with a Python workspace. The Explorer sidebar shows various files and folders. In the center, a code editor window displays a Python script. Below the editor, a terminal window shows the output of a command. A status bar at the bottom indicates the output is truncated. A tooltip is visible, pointing to the truncated output area.

```

# Dự đoán giá nhà cho một diện tích là 50 mét vuông
x1 = 50
y1 = w[0] + w[1] * 50
print('Giá nhà cho 50m^2 là : ', y1)

Diện tích Giá
0 30.0000 448.524
1 32.4138 509.248
2 34.8276 535.184
3 37.2414 551.432
4 39.6552 623.418
x [[ 1.  30. ]
   [ 1. 32.4138]
   [ 1. 34.8276]
   [ 1. 37.2414]
   [ 1. 39.6552]
   [ 1. 42.069]
   [ 1. 44.4828]
   [ 1. 46.8966]
   [ 1. 49.3103]
   [ 1. 51.7241]
   [ 1. 54.1379]
   [ 1. 56.5517]
   [ 1. 58.9655]
   [ 1. 61.3793]
   [ 1. 63.7931]
   [ 1. 66.2069]
   [ 1. 68.6297]
   [ 1. 71.0345]
   [ 1. 73.4483]
   [ 1. 75.8621]
   [ 1. 78.2759]
   [ 1. 80.6897]]

```

- ⇒ Click vào show scrollable element để show ra nhiều hơn

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Phần 1 : Giới thiệu

I

Giới thiệu

| | | |
|----------|--|-----------|
| 1 | Cài đặt môi trường | 19 |
| 1.1 | Giới thiệu | 19 |
| 1.2 | Google Colab | 19 |
| 1.2.1 | Tạo file trên google colab | 19 |
| 1.2.2 | Chọn GPU | 20 |
| 1.2.3 | Các thành phần nhỏ | 21 |
| 1.2.4 | Link với google drive | 22 |
| 1.2.5 | Cài thêm thư viện | 24 |
| 1.3 | Hướng dẫn cài đặt anaconda | 24 |
| 1.3.1 | Giới thiệu | 24 |
| 1.3.2 | Yêu cầu phần cứng và phần mềm | 25 |
| 1.3.3 | Cài đặt | 25 |
| 1.3.4 | Hướng dẫn sử dụng Jupyter notebook | 31 |
| 2 | Python cơ bản | 33 |
| 2.1 | Kiểu dữ liệu cơ bản | 33 |
| 2.1.1 | Số | 33 |
| 2.1.2 | Phép tính logic | 34 |
| 2.1.3 | Chuỗi | 34 |
| 2.2 | Containers | 35 |
| 2.2.1 | List | 35 |
| 2.2.2 | Dictionaries | 35 |
| 2.3 | Function | 36 |
| 2.4 | Thư viện trong python | 37 |
| 2.5 | Thư viện Numpy | 37 |
| 2.5.1 | Array indexing | 38 |
| 2.5.2 | Các phép tính trên array | 38 |
| 2.5.3 | Broadcasting | 40 |
| 2.6 | Matplotlib | 40 |

1.1 Giới thiệu

Python là ngôn ngữ được sử dụng phổ biến nhất trong Deep Learning, vậy nên tất cả code trong sách sẽ được viết bằng python và thư viện Deep Learning được chọn để sử dụng là **Keras**. Trong phần này tôi sẽ hướng dẫn cài đặt môi trường. Có 2 dạng là chạy online dùng google colab và cài trên local dùng anaconda và IDE là spyder, VS Code hoặc jupyter notebook. Hiểu đơn giản thì nếu dùng google colab bạn sẽ viết code python và chạy online, không cần cài gì trên máy cả nên sẽ đơn giản hơn và máy cấu hình yêu vẫn chạy được.

[Getting started with Keras](#)

Sử dụng google colab

- Copy path link data set google drive :

The screenshot shows two instances of the Google Colab interface. In the top instance, a context menu is open over a file named 'USA_Housing.csv' in the 'Chapter1_Intro' folder. The menu options include 'Download', 'Rename file', 'Delete file', 'Copy path', and 'Refresh'. The 'Copy path' option is highlighted. In the bottom instance, a Jupyter Notebook cell contains the following Python code:

```
from google.colab import drive
drive.mount('/content/gdrive')

import pandas as pd

# Đọc dữ liệu từ file CSV
file_path = '/content/gdrive/MyDrive/DeepLearning/Chapter1_Intro/USA_Housing.csv'
data = pd.read_csv(file_path)

# Hiển thị một số dòng đầu của dữ liệu để kiểm tra
print(data.head())
```

The output of the code shows the first few rows of the 'USA_Housing.csv' dataset:

| | Avg. Area Income | Avg. Area House Age | Avg. Area Number of Rooms | Address |
|---|------------------|---------------------|---------------------------|--|
| 0 | 79545.458574 | 5.682861 | 7.009188 | 0 208 Michael Ferry Apt. 674\nLaurabury, NE 3701... |
| 1 | 79248.642455 | 6.002900 | 6.730821 | 1 188 Johnson Views Suite 079\nLake Kathleen, CA,... |
| 2 | 61287.067179 | 5.865890 | 8.512727 | 2 9127 Elizabeth Stravenue\nDanieldown, WI 06482... |
| 3 | 63345.240046 | 7.188236 | 5.586729 | |
| 4 | 59982.197226 | 5.040555 | 7.839388 | |



ChatGPT

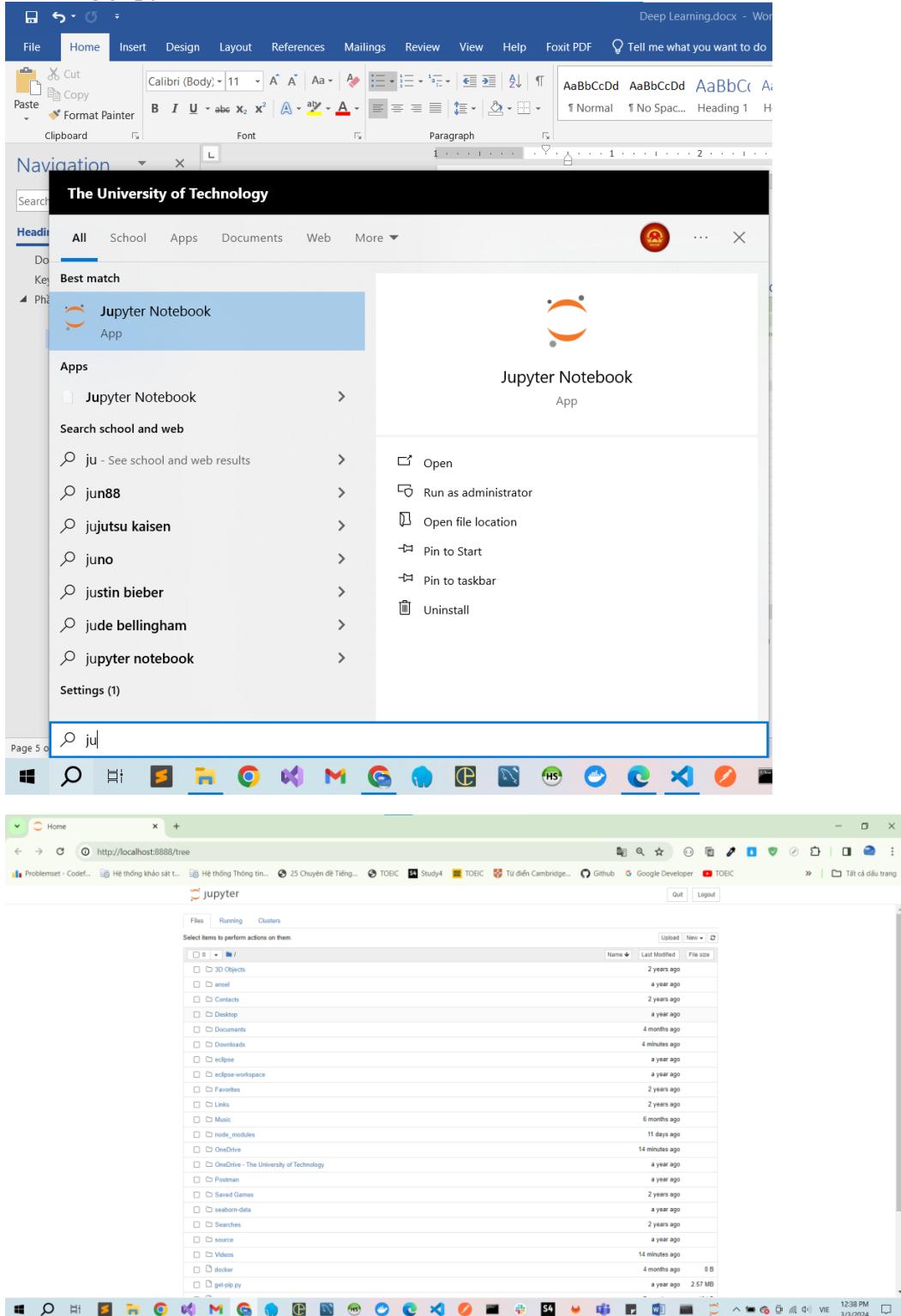
Keras là một thư viện mã nguồn mở được sử dụng rộng rãi trong việc phát triển và triển khai mạng nơ-ron trong lĩnh vực học sâu (deep learning). Nó cung cấp một giao diện dễ sử dụng và linh hoạt để xây dựng các mô hình mạng nơ-ron, giúp người dùng tập trung vào việc thiết kế mô hình mà không cần lo lắng về các chi tiết thấp hơn của triển khai.

Dưới đây là một số đặc điểm của Keras:

- Dễ sử dụng:** Keras được thiết kế để dễ học và dễ sử dụng. Nó cung cấp một API rất trực quan, cho phép người dùng nhanh chóng xây dựng và thử nghiệm các mô hình mạng nơ-ron.
- Modular và linh hoạt:** Keras cho phép bạn xây dựng mô hình từ các lớp (layers) độc lập, và kết hợp chúng lại với nhau để tạo thành các kiến trúc phức tạp hơn. Điều này cho phép bạn tạo ra các mô hình tùy chỉnh và thích nghi với nhu cầu cụ thể của bạn.
- Hỗ trợ nhiều backend:** Keras có thể chạy trên nhiều backend khác nhau, bao gồm TensorFlow, Theano và CNTK. TensorFlow là backend mặc định từ phiên bản TensorFlow 2.0 trở đi.
- Phù hợp với các dự án từ nhỏ đến lớn:** Từ các dự án thử nghiệm nhỏ cho đến các ứng dụng thực tế lớn, Keras có thể được sử dụng trong nhiều tình huống khác nhau mà không đòi hỏi nhiều thời gian hoặc kiến thức chuyên sâu về học máy.

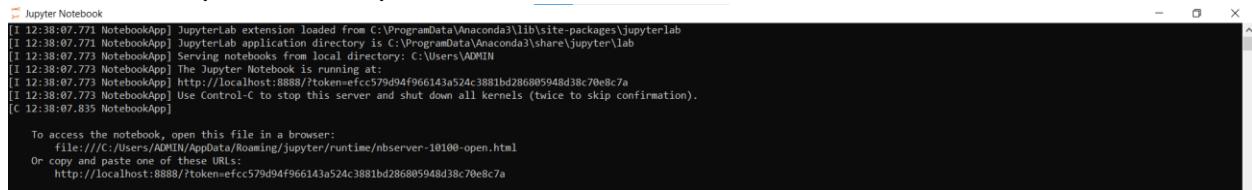
Tóm lại, Keras là một công cụ mạnh mẽ và linh hoạt để phát triển các mô hình mạng nơ-ron trong lĩnh vực học sâu, đặc biệt là cho những người mới bắt đầu và cần một cách tiếp cận đơn giản và dễ dàng.

Sử dụng jupyter notebook của Anaconda



- ❖ Sau khi cài anaconda thì nó tự có sẵn mà ta không cần phải chạy gì cả .

- ❖ Chỉ cần click chọn đê mở là được .



```
[I 12:38:07.771 NotebookApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-packages\jupyterlab
[I 12:38:07.771 NotebookApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\jupyter\lab
[I 12:38:07.773 NotebookApp] Serving notebooks from local directory: C:\Users\ADMIN
[I 12:38:07.773 NotebookApp] The Jupyter Notebook is running at:
[I 12:38:07.773 NotebookApp] http://localhost:8888/?token=efc579d94ff966143a524c3881bd286805948d38c70e8c7a
[I 12:38:07.773 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 12:38:07.835 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/ADMIN/AppData/Roaming/jupyter/runtime/nbserver-10100-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=efc579d94ff966143a524c3881bd286805948d38c70e8c7a
```

- ❖ Nó sẽ mở một cmd như sau . Đừng tắt cái này nếu không ngoài web nó cũng tắt .

Python cơ bản

2.2 Containers

Các kiểu dữ liệu cơ bản chỉ chứa một giá trị mỗi biến (số, chuỗi), tuy nhiên nhiều lúc mình cần chứa nhiều giá trị, ví dụ chứa tên học sinh trong một lớp có 100 bạn. Mình không thể tạo 100 biến để lưu tên 100 bạn. Vậy nên cần các kiểu dữ liệu có thể chứa nhiều giá trị khác nhau. Đó là container (collection). Python có một số container như: list, tuple, set, dictionary. Dưới tôi sẽ trình bày hai kiểu dữ liệu collection mà mọi người hay gặp nhất trong python là list và dictionary.

Nhắc lại về list python

2.2.1 List

List trong Python giống như mảng (array) nhưng không cố định kích thước và có thể chứa nhiều kiểu dữ liệu khác nhau trong 1 list.

```
xs = [3, 1, 2]      # Tao 1 list
print(xs, xs[2])   # Prints "[3, 1, 2] 2"
print(xs[-1])       # Chỉ số âm là đếm phần tử từ cuối list lên; prints "2"
xs[2] = 'foo'        # List có thể chứa nhiều kiểu phần tử khác nhau
print(xs)            # Prints "[3, 1, 'foo']"
xs.append('bar')     # Thêm phần tử vào cuối list
print(xs)            # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()         # Bỏ đi phần tử cuối cùng khỏi list và trả về phần tử đấy
print(x, xs)         # Prints "bar [3, 1, 'foo']"
```

Slicing Thay vì lấy từng phần tử một trong list thì python hỗ trợ truy xuất nhiều phần tử 1 lúc gọi là slicing.

```
nums = list(range(5))      # range sinh ra 1 list các phần tử
print(nums)                 # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])            # Lấy phần tử thứ 2->4, python chỉ số mảng từ 0;
print(nums[2:])              # Lấy từ phần tử thứ 2 đến hết; prints "[2, 3, 4]"
print(nums[:2])              # Lấy từ đầu đến phần tử thứ 2; prints "[0, 1]"
print(nums[:])                # Lấy tất cả phần tử trong list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])             # Lấy từ phần tử đầu đến phần tử gần cuối trong list;
                           # prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]           # Gán giá trị mới cho phần tử trong mảng từ vị trí 2->4
print(nums)                  # Prints "[0, 1, 8, 9, 4]"
```

Loops Để duyệt và in ra các phần tử trong list

```
animals = ['cat', 'dog', 'monkey']
# duyệt giá trị không cần chỉ số
for animal in animals:
    print('%s' % (animal))
# duyệt giá trị kèm chỉ số dùng enumerate
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", in mỗi thành phần trong list 1 dòng
```

Nhắc lại về dictionaries (kiểu từ điển trong python) (object trong một số ngôn ngữ khác)

2.2.2 Dictionaries

Dictionaries lưu thông tin dưới dạng key, value.

```
d = {'cat': 'cute', 'dog': 'furry'} # Tạo dictionary, các phần tử dạng key:value
print(d['cat'])                  # Lấy ra value của key 'cat' trong dictionary prints "cute"
print('cat' in d)                # Kiểm tra key có trong dictionary không; prints "True"
d['fish'] = 'wet'                 # Gán key, value, d[key] = value
print(d['fish'])                 # Prints "wet"
# print(d['monkey'])            # Lỗi vì key 'monkey' không trong dictionary
del d['fish']                   # Xóa phần tử key:value từ dictionary
```

Loop Duyệt qua các phần tử trong dictionary

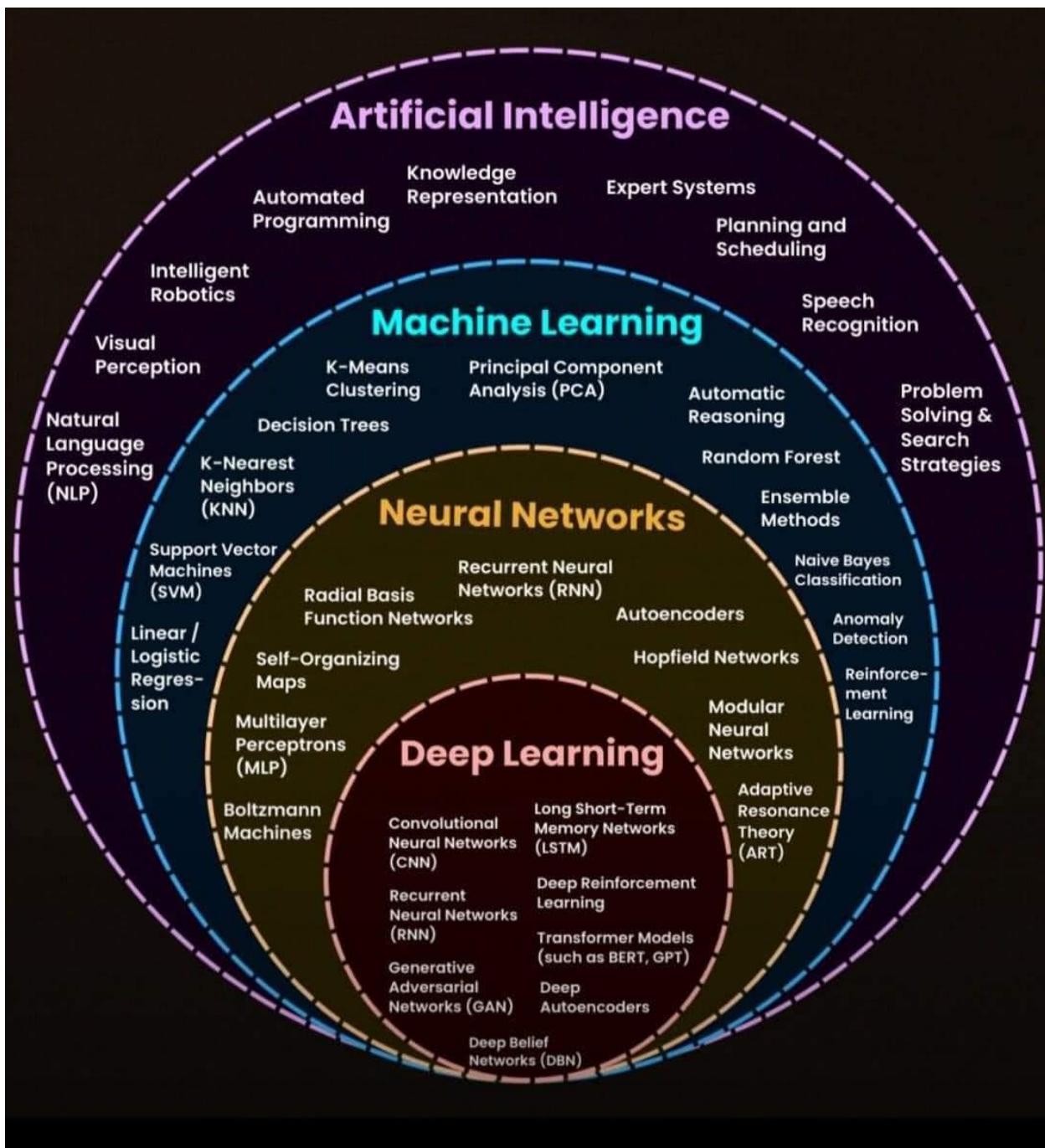
```
d = {'person': 2, 'cat': 4, 'spider': 8}
# Duyệt key
for animal in d:
    print('A %s has %d legs' % (animal, d[animal]))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"

# Duyệt value
for legs in d.values():
    print('%d legs' % (legs))
# Prints "2 legs", "4 legs", "8 legs"

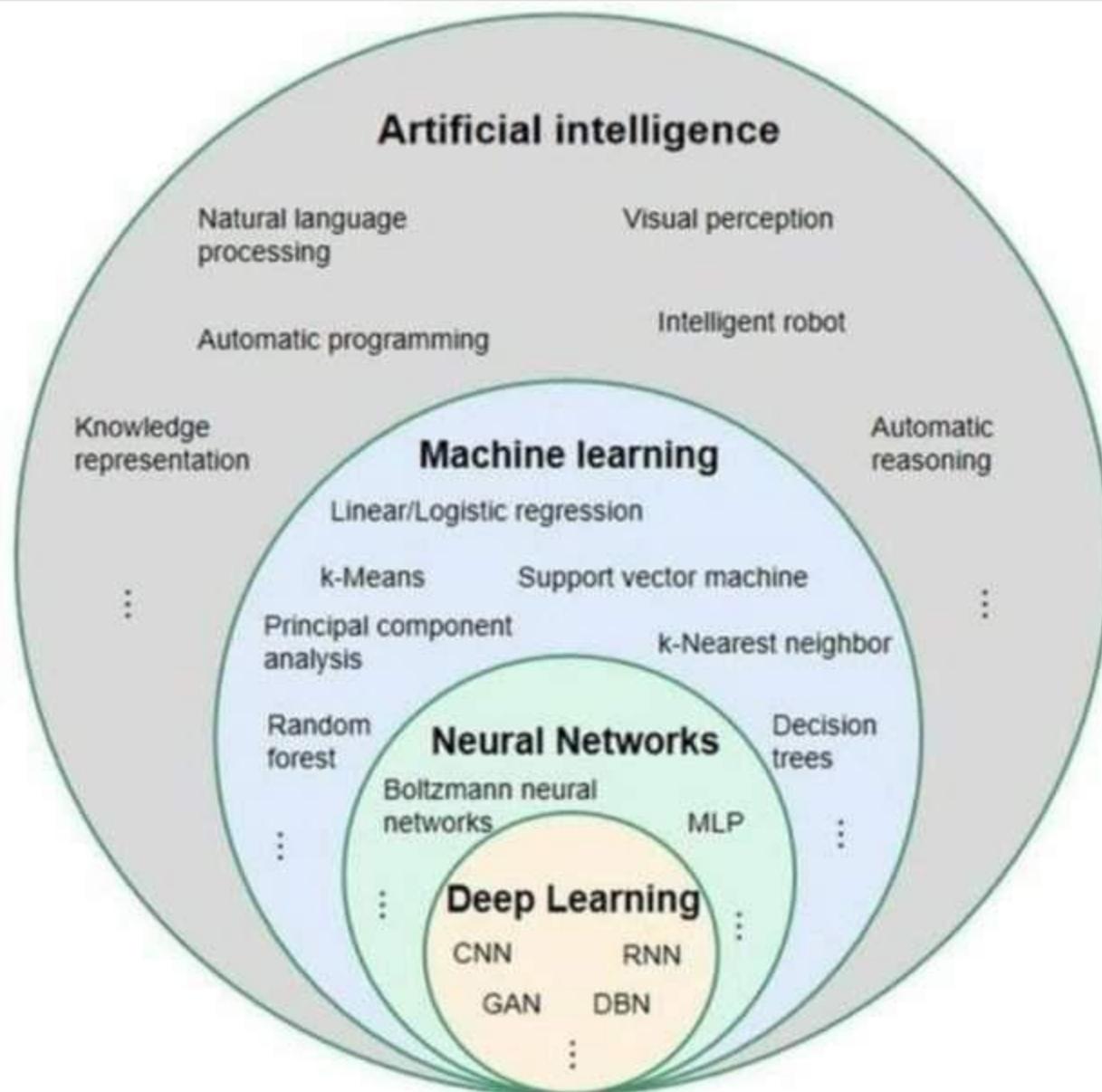
# Duyệt cả key và value
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

Tổng quan (bản đồ) về Machine Learning – Deep Learning

[Classification of deep learning methods – Artofit](#)

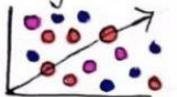


Relationship between artificial intelligence, machine learning, neural network, and deep learning

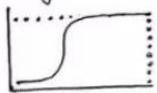


ESSENTIAL MACHINE LEARNING ALGORITHMS :- @codehub.py

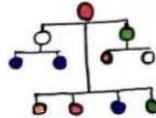
① Linear Regression



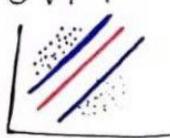
② Logistic Regression



③ Decision Tree



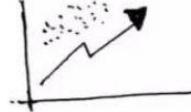
④ SVM



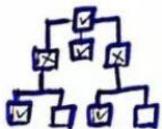
⑤ KNN



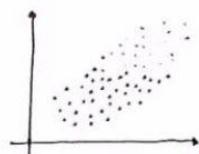
⑥ Dimensionality Reduction



⑦ Random Forest



⑧ K-means



⑨ Naïve Bayes



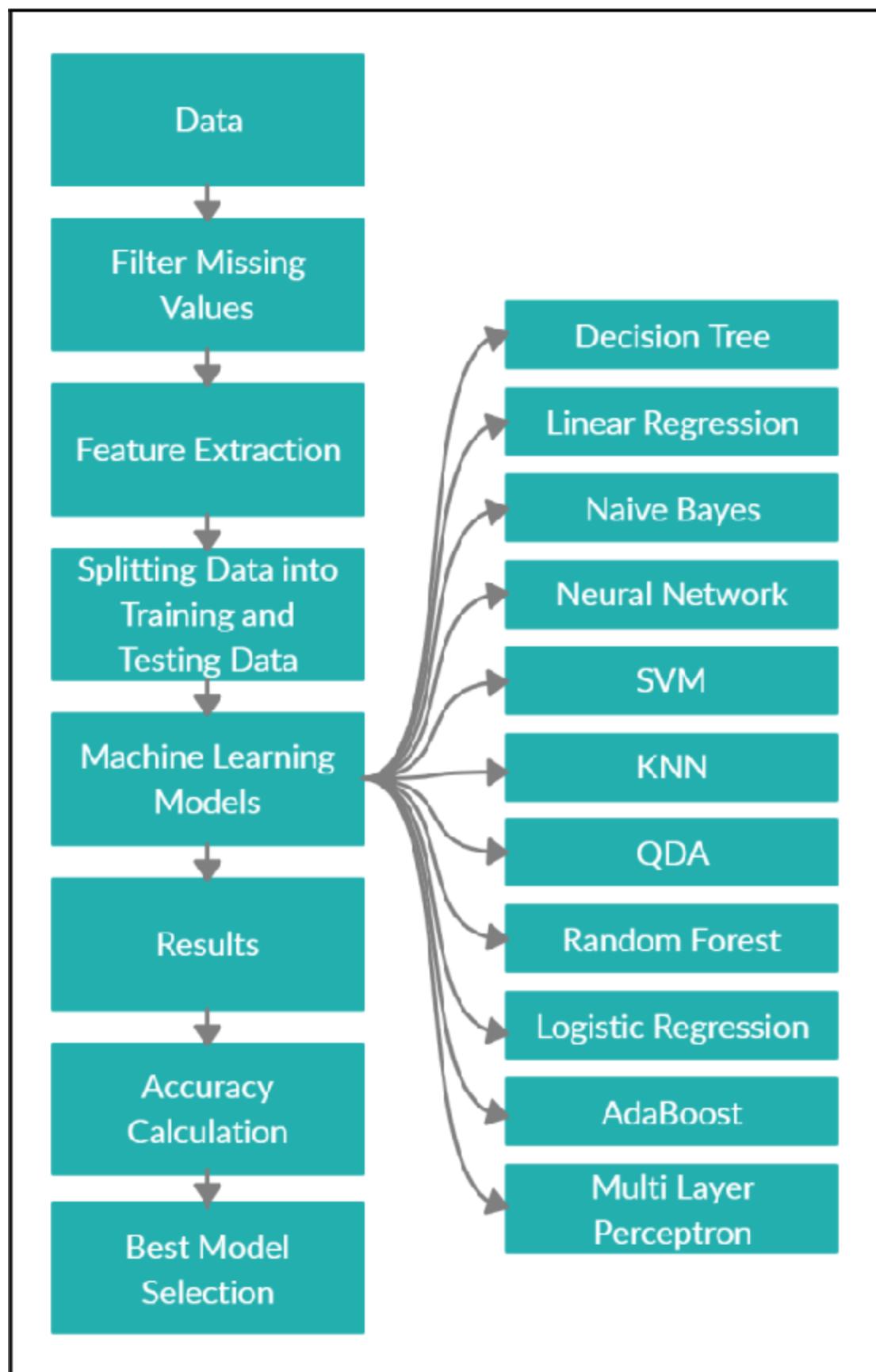
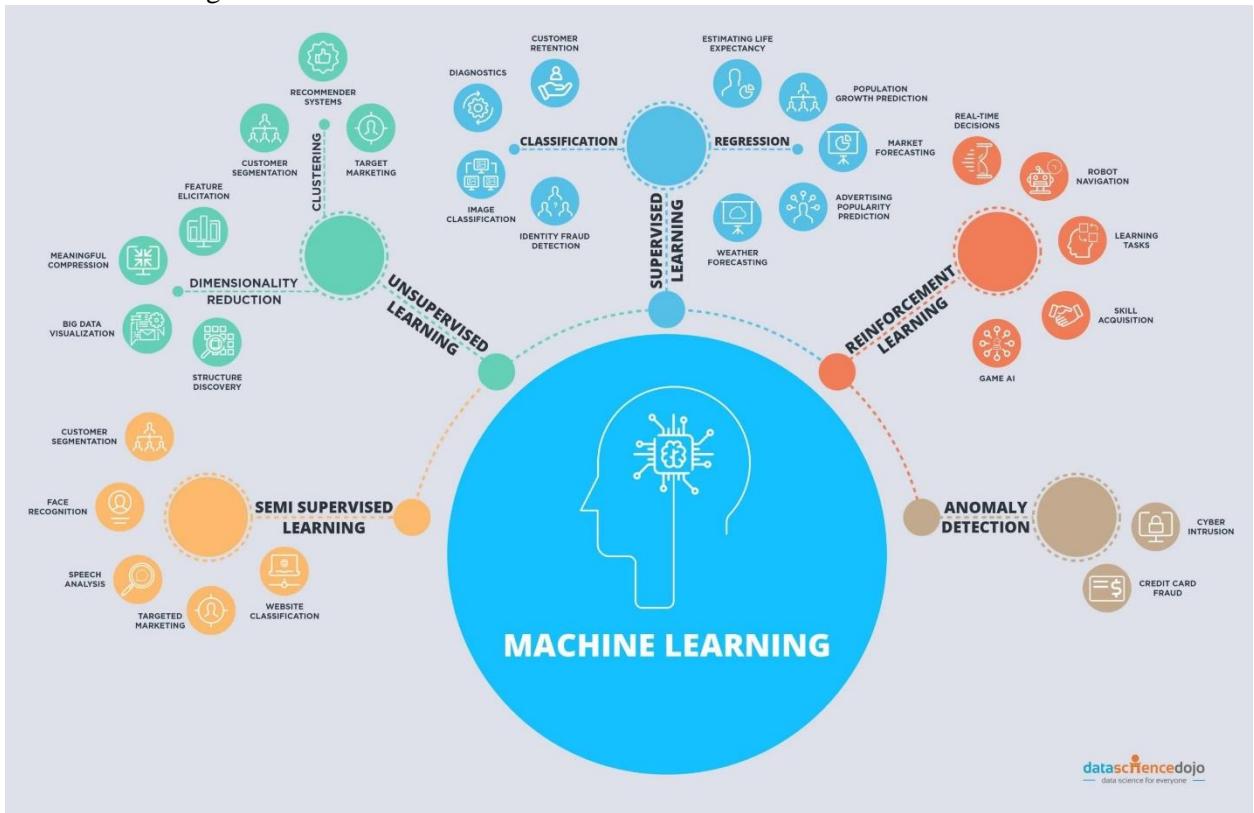


Figure 3. Use case diagram for whole implementation.

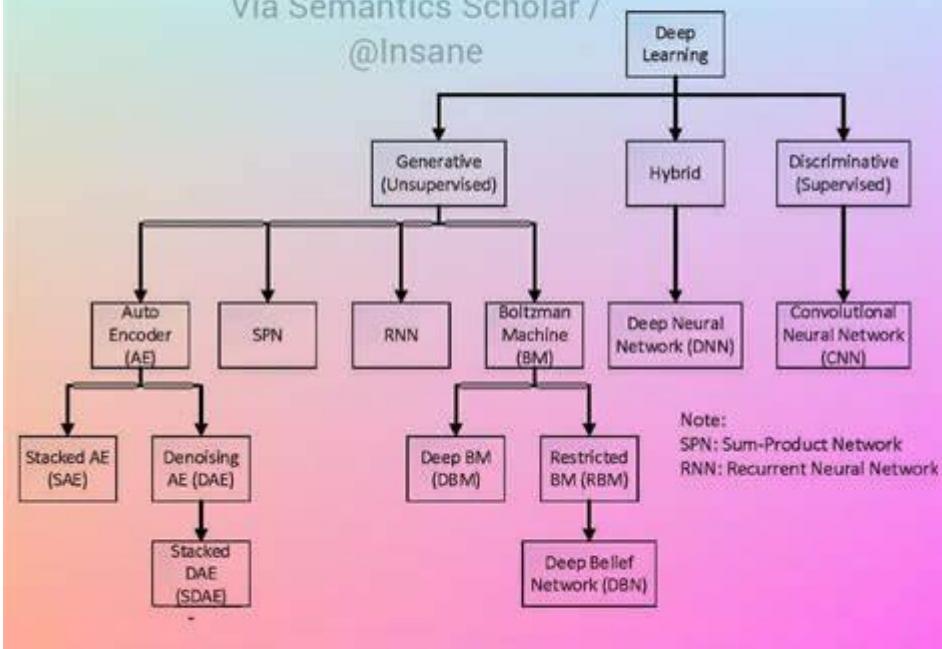
- Machine Learning Overview



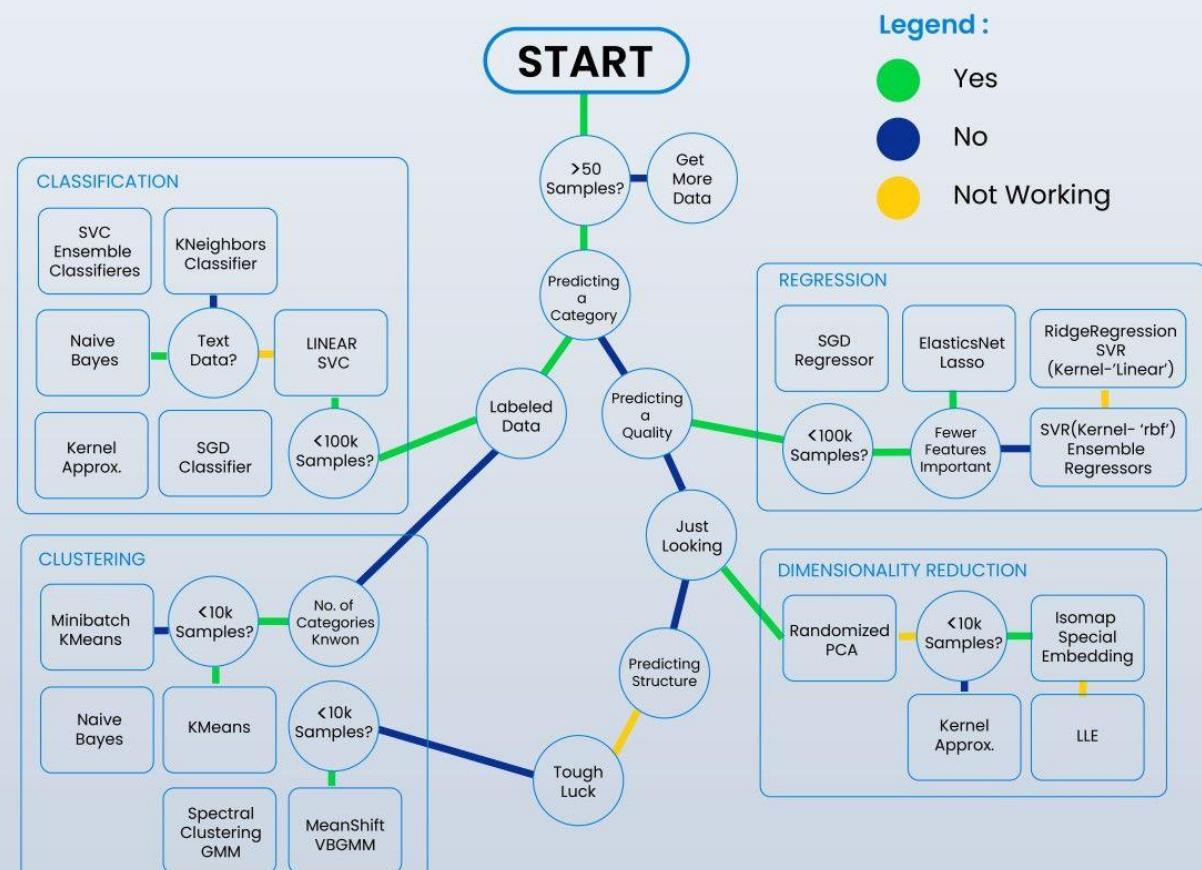
- Deep Learning

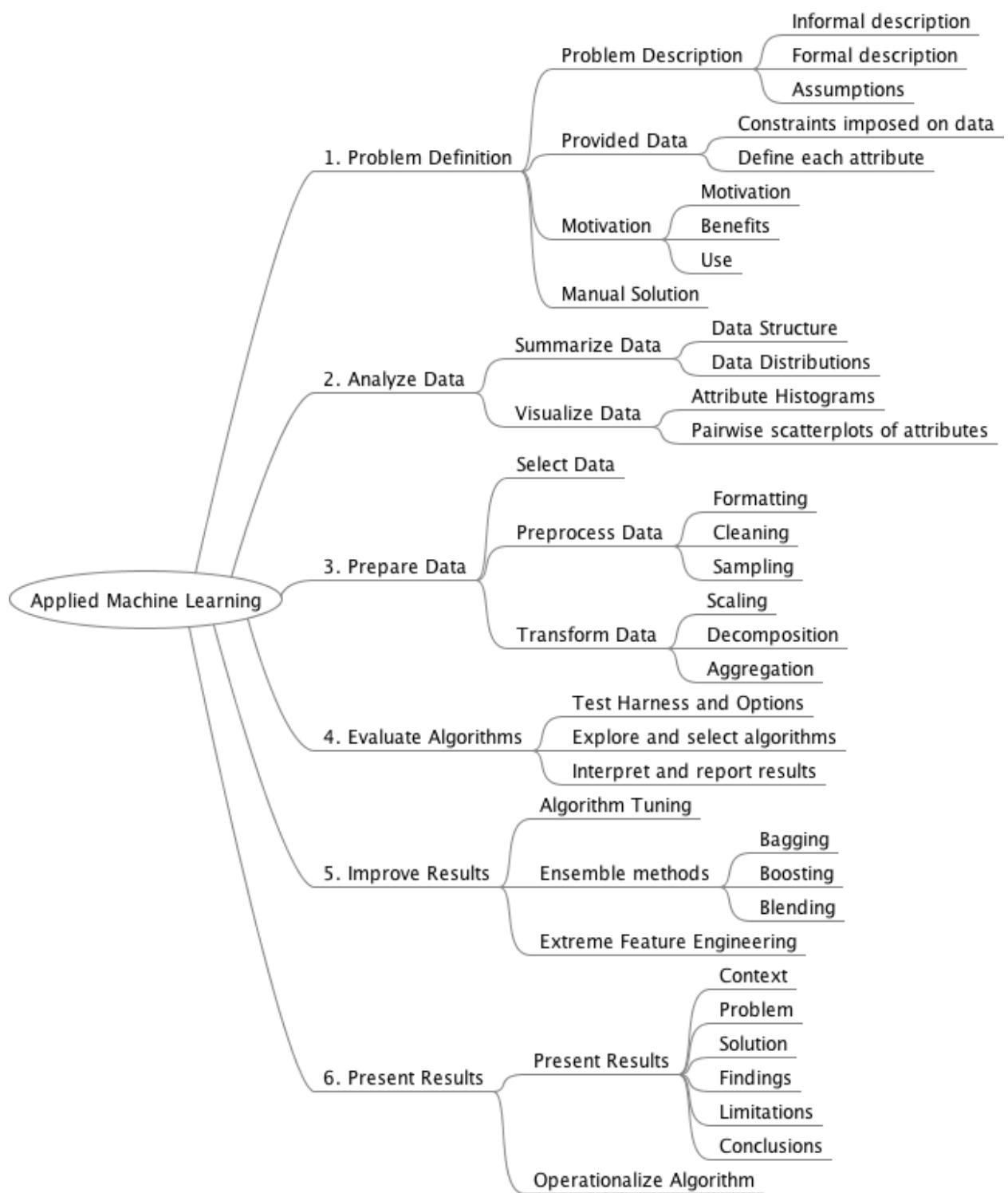
Classification of Deep Learning Methods

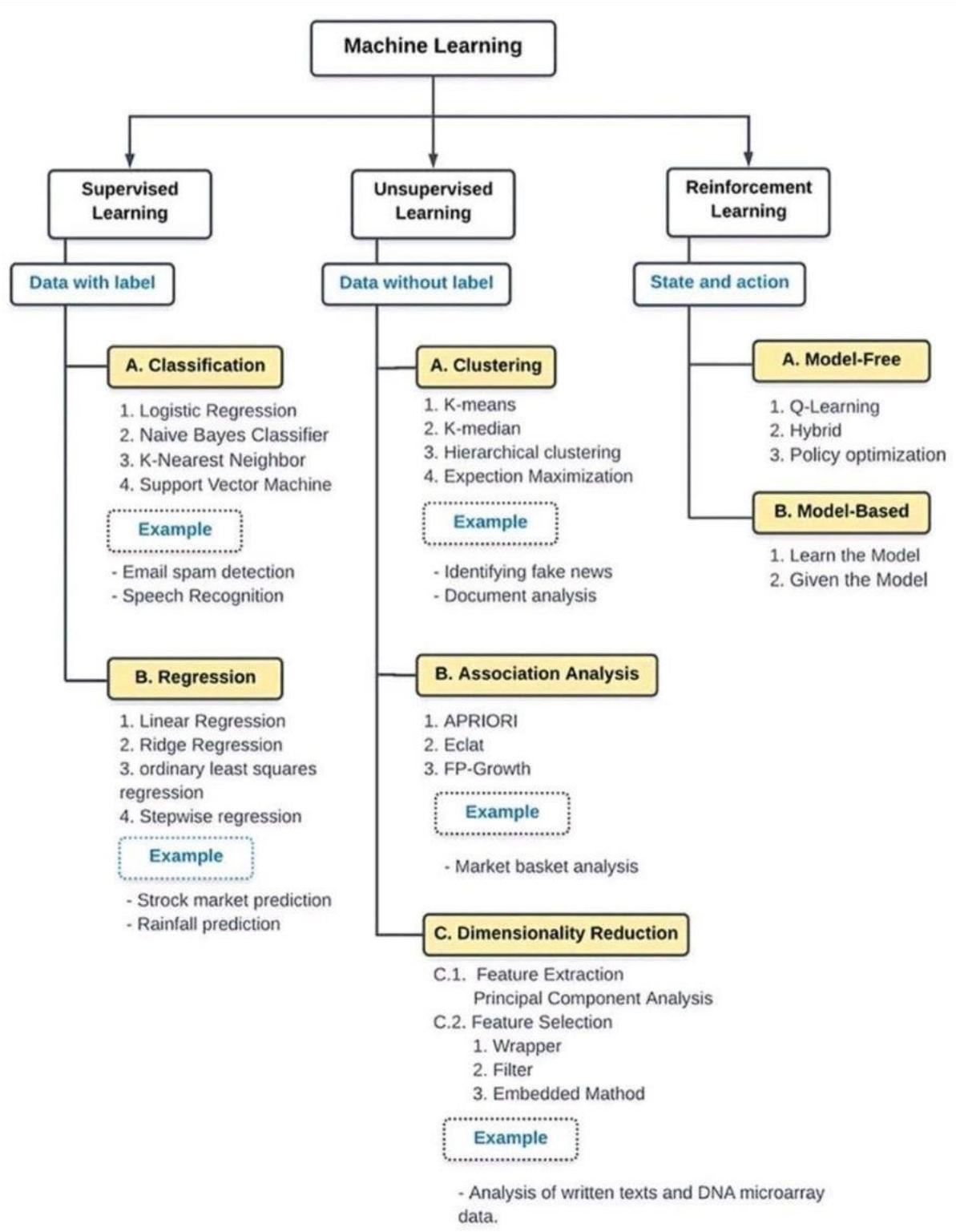
Via Semantics Scholar /
@Insane



CHEATSHEET FOR MACHINE LEARNING

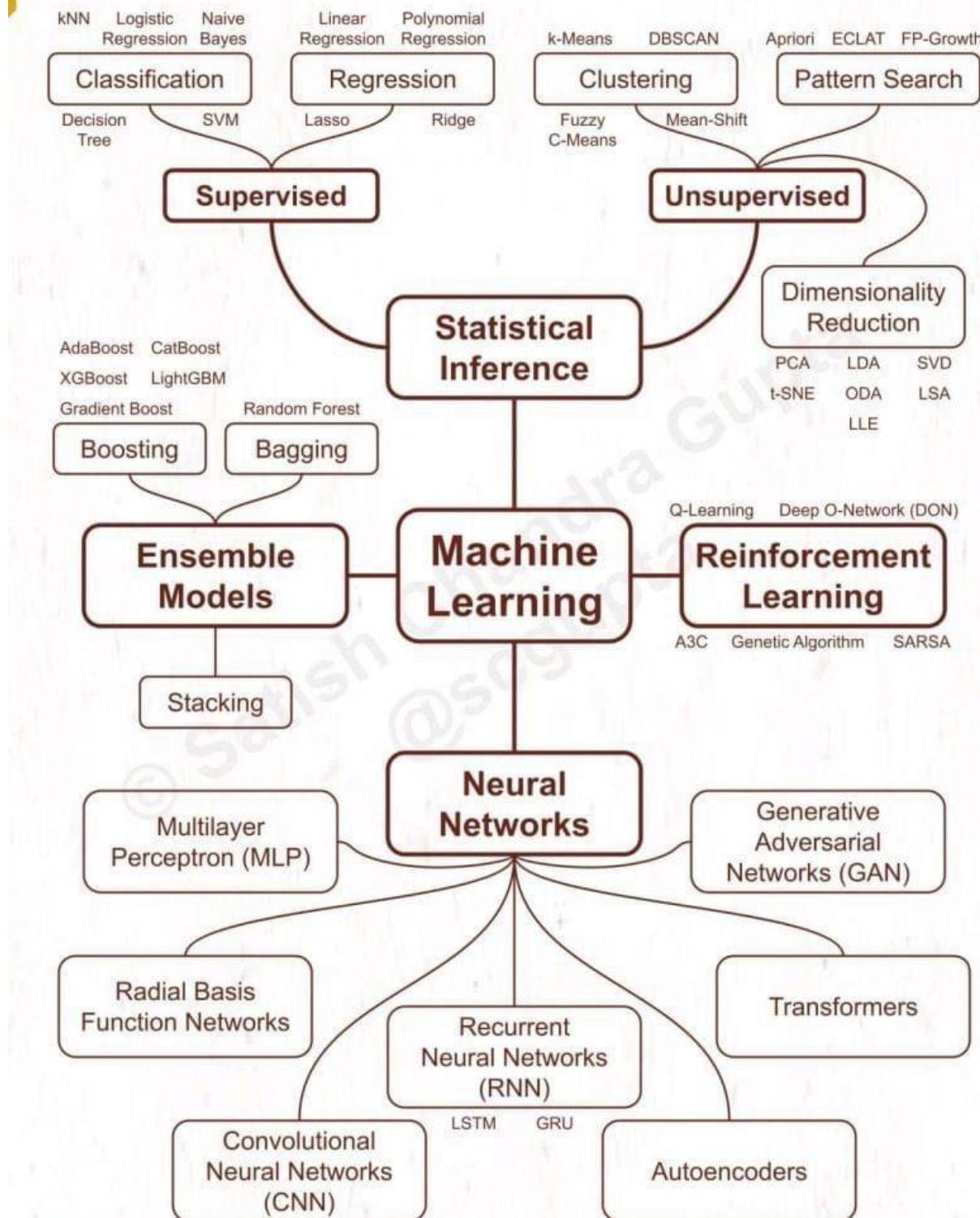






Popular Machine Learning Models

ml4devs.com/ml-intro 



The screenshot shows a Jupyter Notebook interface with three tabs open: 'I1.py', 'L1.ipynb U', and 'Chapter1_Intro.ipynb U'. The 'Chapter1_Intro.ipynb U' tab is active, displaying a list of Artificial Intelligence topics:

- Natural Language processing
- Visual perception
- Automatic programming
- Intelligent robot
- Knowledge representation
- Automatic reasoning
 - Machine Learning
 - Linear regression
 - Logistic regression
 - K-Means
 - Support vector machine
 - Principal component
 - K-Nearest neighbor
 - Random forest
 - Decision trees
 - Neural Networks
 - Boltzmann neural networks
 - MLP
 - Deep Learning
 - CNN
 - RNN
 - GAN
 - DBN

Below the list, there is a detailed description of Artificial Intelligence (Trí tuệ nhân tạo) and its sub-topics:

- **Artificial Intelligence (Trí tuệ nhân tạo):** Lĩnh vực nghiên cứu và phát triển các hệ thống và máy móc có khả năng thực hiện các nhiệm vụ thông thường yêu cầu sự hiểu biết và suy luận tương tự như con người.
 - Natural Language Processing (Xử lý ngôn ngữ tự nhiên): NLP là một lĩnh vực của trí tuệ nhân tạo mà nó tập trung vào tương tác giữa máy tính và con người bằng ngôn ngữ tự nhiên.
 - Visual Perception (Nhận thức hình ảnh): Là khả năng của máy tính hoặc hệ thống máy móc để hiểu và xử lý thông tin từ hình ảnh hoặc dữ liệu thị giác.
 - Automatic Programming (Lập trình tự động): Quá trình tự động tạo ra mã máy hoặc mã ngôn ngữ tự nhiên từ mô tả về yêu cầu và quy tắc.
 - Intelligent Robot (Robot thông minh): Robot có khả năng tương tác với môi trường xung quanh, tự động hoạt động và học từ kinh nghiệm.
 - Knowledge Representation (Biểu diễn kiến thức): Cách thức biểu diễn thông tin và kiến thức trong máy tính để nó có thể sử dụng cho việc suy luận và ra quyết định.
 - Automatic Reasoning (Suy luận tự động): Quá trình suy luận và đưa ra quyết định tự động dựa trên dữ liệu và kiến thức có sẵn.
- **Machine Learning (Học máy):** Phương pháp cho máy tính để học tự động từ dữ liệu mà không cần lập trình cụ thể.
 - Linear Regression (Hồi quy tuyến tính): Phương pháp dự đoán giá trị dựa trên mối quan hệ tuyến tính giữa các biến đầu vào và đầu ra.
 - Logistic Regression (Hồi quy logistic): Phân loại dựa trên mô hình hồi quy tuyến tính để dự đoán xác suất thuộc vào một nhóm phân loại.
 - K-Means (Phân cụm K-Means): Phương pháp phân cụm dữ liệu thành các nhóm có tính chất tương tự nhau.
 - Support Vector Machine (Máy vector hỗ trợ): Một thuật toán học máy dùng để phân loại và hồi quy.

File Edit Selection View Go Run Terminal Help

L1.ipynb U Chapter1_Intro.ipynb Untitled (Workspace)

Learning > Chapter1_Overview.Intro > Chapter1_Intro.ipynb > Quan trọng : Việc lựa chọn Thuật toán học máy nào phụ thuộc vào dữ liệu và yêu cầu của bài toán . Ví dụ như bài này thì dùng hồi quy tuyến tính còn không dùng hồi quy logistic được . Python 3.10.6

+ Code + Markdown | Run All Restart Clear All Outputs Variables Outline ...

- **Machine Learning (Học máy):** Phương pháp cho máy tính để học tự động từ dữ liệu mà không cần lập trình cụ thể.
 - Linear Regression (Hồi quy tuyến tính): Phương pháp dự đoán giá trị dựa trên mối quan hệ tuyến tính giữa các biến đầu vào và đầu ra.
 - Logistic Regression (Hồi quy logistic): Phân loại dựa trên mô hình hồi quy tuyến tính để dự đoán xác suất thuộc vào một nhóm phân loại.
 - K-Means (Phân cụm K-Means): Phương pháp phân cụm dữ liệu thành các nhóm có tính chất tương tự nhau.
 - Support Vector Machine (Máy vector hỗ trợ): Một thuật toán học máy dùng để phân loại và hồi quy.
 - Principal Component Analysis (Thành phần chính): Phương pháp giảm chiều dữ liệu bằng cách tìm các thành phần chính có độ biến thiên lớn nhất.
 - K-Nearest Neighbor (K-Điểm gần nhất): Phương pháp dự đoán nhãn mới dựa trên đặc điểm của các điểm gần nhất trong tập dữ liệu.
 - Decision Trees (Cây quyết định): Mô hình dữ liệu phân loại hoặc hồi quy dựa trên quy tắc quyết định dạng cây.
 - Random Forest (Rừng ngẫu nhiên): Một phương pháp học máy dựa trên việc xây dựng nhiều cây quyết định.
- **Neural Networks (Mạng Nơ-ron):** Một mô hình tính toán lấy cảm hứng từ cấu trúc nơ-ron trong hệ thống thần kinh của con người.
 - Boltzmann Neural Networks (Mạng nơ-ron Boltzmann): Một loại mạng nơ-ron có đặc điểm đặc biệt trong việc học không giám sát.
 - Multilayer Perceptron (Mạng nơ-ron đa lớp): Một loại mạng nơ-ron sâu với ít nhất một lớp ẩn giữa lớp đầu vào và lớp đầu ra.
- **Deep Learning (Học sâu):** Một phương pháp học máy sâu sử dụng các mạng nơ-ron có nhiều lớp.
 - Convolutional Neural Networks (Mạng nơ-ron tích chập): Một loại mạng nơ-ron sử dụng trong việc xử lý và phân tích hình ảnh.
 - Recurrent Neural Networks (Mạng nơ-ron hồi quy): Mạng nơ-ron có khả năng xử lý dữ liệu tuần tự, thường được sử dụng trong xử lý ngôn ngữ tự nhiên.
 - Generative Adversarial Networks (Mạng nơ-ron cốt lõi): Một kiểu mô hình học sâu sử dụng để tạo ra dữ liệu mới có cấu trúc từ dữ liệu mẫu.
 - Deep Belief Networks (Mạng nơ-ron chấn đoán): Một kiểu mạng nơ-ron sâu được sử dụng cho việc phân tích và phân loại dữ liệu không giám sát.

Trong lĩnh vực Học máy (Machine Learning), có nhiều thuật toán khác nhau với mục tiêu giải quyết các vấn đề khác nhau. Sự đa dạng này phản ánh những nỗ lực của cộng đồng nghiên cứu để tìm ra các phương pháp hiệu quả hơn cho các tác vụ cụ thể và để khám phá sâu hơn trong lĩnh vực này. Dưới đây là một số lý do về sự đa dạng của các thuật toán học máy:

Spaces: 4 CRLF Cell 4 of 7 3:00 PM 3/3/2024

File Edit Selection View Go Run Terminal Help

L1.ipynb U Chapter1_Intro.ipynb Untitled (Workspace)

Learning > Chapter1_Overview.Intro > Chapter1_Intro.ipynb > Quan trọng : Việc lựa chọn Thuật toán học máy nào phụ thuộc vào dữ liệu và yêu cầu của bài toán . Ví dụ như bài này thì dùng hồi quy tuyến tính còn không dùng hồi quy logistic được . Python 3.10.6

+ Code + Markdown | Run All Restart Clear All Outputs Variables Outline ...

Trong lĩnh vực Học máy (Machine Learning), có nhiều thuật toán khác nhau với mục tiêu giải quyết các vấn đề khác nhau. Sự đa dạng này phản ánh những nỗ lực của cộng đồng nghiên cứu để tìm ra các phương pháp hiệu quả hơn cho các tác vụ cụ thể và để khám phá sâu hơn trong lĩnh vực này. Dưới đây là một số lý do về sự đa dạng của các thuật toán học máy:

Khác biệt trong cách xử lý dữ liệu: Mỗi thuật toán có cách tiếp cận riêng để xử lý dữ liệu và học từ dữ liệu. Ví dụ, một số thuật toán như Linear Regression tập trung vào việc tìm ra mối quan hệ tuyến tính giữa biến đầu vào và đầu ra, trong khi Support Vector Machine tập trung vào việc tạo ra các ranh giới quyết định phức tạp hơn.

Hiệu suất và độ phức tạp: Mỗi thuật toán có hiệu suất và độ phức tạp riêng biệt. Một số thuật toán có hiệu suất tốt cho dữ liệu lớn và không gian đặc trưng phức tạp, trong khi các thuật toán khác có thể phù hợp hơn cho các vấn đề nhỏ hơn với dữ liệu đơn giản.

Các tác vụ khác nhau: Các thuật toán được thiết kế để giải quyết các loại tác vụ khác nhau trong học máy như phân loại, dự đoán, phân cụm, học tăng cường, và nhiều hơn nữa.

Khám phá và phát triển: Các thuật toán mới thường được phát triển để khám phá và giải quyết các vấn đề mà các thuật toán hiện có chưa giải quyết được một cách hiệu quả. Điều này có thể bao gồm việc xử lý dữ liệu không cân bằng, tạo ra mô hình dự đoán chính xác hơn, hoặc cải thiện khả năng tổng quát hóa của mô hình.

Sắp xếp các thuật toán theo thứ tự ra đời có thể phụ thuộc vào nhiều yếu tố nhưng thường thì các thuật toán được phát triển đầu tiên là những phương pháp cơ bản và sau đó là các phương pháp tiên tiến hơn. Mục đích chính của việc ra đời các thuật toán mới thường là cải thiện hiệu suất, tính linh hoạt và khả năng áp dụng của hệ thống học máy.

Quan trọng : Việc lựa chọn Thuật toán học máy nào phụ thuộc vào dữ liệu và yêu cầu của bài toán . Ví dụ như bài này thì dùng hồi quy tuyến tính còn không dùng hồi quy logistic được .

Hồi quy tuyến tính thích hợp cho các vấn đề trong đó biến phụ thuộc là liên tục và các biến độc lập có thể là liên tục hoặc rời rạc. Trong trường hợp của bạn, dữ liệu giá nhà (Price) là một biến liên tục, do đó hồi quy tuyến tính là lựa chọn hợp lý để dự đoán giá nhà.

Trong khi đó, hồi quy logistic được sử dụng cho các vấn đề phân loại, nơi mục tiêu là dự đoán xác suất của một sự kiện xảy ra hoặc không xảy ra. Trong trường hợp của bạn, mục tiêu là dự đoán giá nhà, một biến liên tục, không phải là một biến phân loại. Do đó, hồi quy logistic không phù hợp cho bài toán này.

Vì vậy, trong trường hợp bạn muốn dự đoán giá nhà (một biến liên tục), hồi quy tuyến tính là phương pháp thích hợp hơn để xử lý dữ liệu và dự đoán.

import pandas as pd

Spaces: 4 CRLF Cell 4 of 7 3:00 PM 3/3/2024

Array indexing

2.5.1 Array indexing

Tương tự như list, numpy array cũng có thể slice. Tuy nhiên vì numpy array có nhiều chiều, nên khi dùng slice phải chỉ định rõ chiều nào.

```
import numpy as np

# Tạo array 2 chiều với kích thước (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Dùng slide để lấy ra subarray gồm 2 hàng đầu tiên (1 & 2) và 2 cột (2 & 3).
# Output là array kích thước 2*2
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

print(a[0, 1])    # Prints "2"
a[0, 1] = 77      # Chính sửa phần tử trong array
print(a[0, 1])    # Prints "77"
```

Bên cạnh đó cũng có thể dùng các chỉ số với slice index. Tuy nhiên số chiều array sẽ giảm đi.

```
import numpy as np

# Tạo array 2 chiều kích thước (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

row_r1 = a[1, :]    # Lấy ra hàng thứ 2 trong a, output array 1 chiều
row_r2 = a[1:2, :]  # Lấy ra hàng thứ 1&2 trong a, output array 2 chiều
print(row_r1, row_r1.shape) # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"
```

Các phép tính trên array – numpy

`+, -, *, /`: Cộng, trừ, nhân, chia các phần tử tương ứng của ma trận

`dot()`: nhân hai ma trận

- ❖ Ta có thể sử dụng các phép tính có sẵn của python hoặc sử dụng hàm của numpy như sau :

2.5.2 Các phép tính trên array

Các phép tính với ma trận được hỗ trợ trên numpy

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Tính tổng
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Phép trừ
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Phép nhân element-wise
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Phép chia element-wise
# [[ 0.2          0.33333333]
#  [ 0.42857143  0.5        ]]
print(x / y)
print(np.divide(x, y))
```

```

# Phép chia element-wise
# [[ 0.2          0.33333333]
#  [ 0.42857143   0.5          ]]
print(x / y)
print(np.divide(x, y))

# Tính căn bậc hai
# [[ 1.              1.41421356]
#  [ 1.73205081   2.              ]]
print(np.sqrt(x))

```

Phép toán * dùng để nhân element-wise chứ không phải nhân ma trận thông thường. Thay vào đó dùng np.dot để nhân 2 ma trận.

```

import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Nhân ma trận, output số 219
print(v.dot(w))
print(np.dot(v, w))

# Nhân ma trận; output ma trận 2*2

```

```

# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))

```

```
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Numpy cũng hỗ trợ tính tổng array theo các chiều khác nhau

```
import numpy as np

x = np.array([[1,2],[3,4]])
```

```
print(np.sum(x)) # Tính tổng tất cả phần tử trong array; prints "10"
print(np.sum(x, axis=0)) # Tính tổng phần tử mỗi hàng; prints "[4 6]"
print(np.sum(x, axis=1)) # Tính tổng phần tử mỗi cột; prints "[3 7]"
```

axis - numpy



ChatGPT

Trong thư viện NumPy, tham số `axis` được sử dụng để xác định hướng tính toán khi thực hiện các phép toán trên mảng đa chiều. Tham số này cho biết hàm nên áp dụng phép toán theo chiều nào trên mảng.

- Nếu `axis=0`, phép toán sẽ được thực hiện theo chiều dọc (theo cột). Khi đó, hàm sẽ tính tổng các phần tử theo từng cột và trả về kết quả là một mảng chứa tổng của từng cột.
- Nếu `axis=1`, phép toán sẽ được thực hiện theo chiều ngang (theo hàng). Khi đó, hàm sẽ tính tổng các phần tử theo từng hàng và trả về kết quả là một mảng chứa tổng của từng hàng.

Dưới đây là ví dụ minh họa:

```
python
Copy code

import numpy as np

x = np.array([[1, 2],
              [3, 4]])

print(np.sum(x, axis=0)) # Tính tổng phần tử mỗi cột; kết quả là "[4 6]"
print(np.sum(x, axis=1)) # Tính tổng phần tử mỗi hàng; kết quả là "[3 7]"
```

Kết quả:

```
csharp
Copy code

[4 6]
[3 7]
```

Trong ví dụ trên, mảng `x` có kích thước (2, 2), nghĩa là có 2 hàng và 2 cột. Khi tính tổng theo `axis=0` (cột), hàm `np.sum()` sẽ tính tổng các phần tử trong mỗi cột và trả về một mảng 1 chiều chứa tổng của từng cột. Khi tính tổng theo `axis=1` (hang), hàm `np.sum()` sẽ tính tổng các phần tử trong mỗi hàng và trả về một mảng 1 chiều chứa tổng của từng hàng.

NG You

axis phụ thuộc vào số nhiều , n chiều thì từ 0 đến n-1

ChatGPT

Đúng, bạn đã hiểu đúng. Trong NumPy, số lượng trực (axes) của một mảng được xác định bởi số chiều (dimensions) của mảng đó. Mỗi trực được đánh số từ 0 đến (số trực - 1).

Ví dụ:

- Một mảng một chiều có một trực, được đánh số là trực 0.
- Một mảng hai chiều có hai trực: trực 0 và trực 1.
- Một mảng ba chiều có ba trực: trực 0, trực 1 và trực 2.
- Và tiếp tục như vậy.

Do đó, tham số `axis` trong NumPy có thể nhận các giá trị từ 0 đến (số trực - 1) của mảng tương ứng.



Broadcasting là một kĩ thuật cho phép numpy làm việc với các array có shape khác nhau khi thực hiện các phép toán.

```
import numpy as np

# Cộng vector v với mỗi hàng của ma trận x, kết quả lưu ở ma trận y.
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)    # Tao 1 array có chiều giống x

# Dùng loop để vector v với mỗi hàng của ma trận
for i in range(4):
    y[i, :] = x[i, :] + v

print(y)
# Kết quả của y
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]

# Cộng kiểu broadcasting trong python
print(x + v)
# Kết quả
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]

# 2 kết quả cho ra giống nhau
```

Phần 2 : Machine learning cơ bản

Phân nhóm các thuật toán Machine Learning

1. 1. Phân nhóm dựa trên phương thức học
 - o Supervised Learning (Học có giám sát)
 - Classification (Phân loại)
 - Regression (Hồi quy)
 - o Unsupervised Learning (Học không giám sát)
 - Clustering (phân nhóm)
 - Association
 - o Semi-Supervised Learning (Học bán giám sát)
 - o Reinforcement Learning (Học Củng Cố)
2. Phân nhóm dựa trên chức năng
 - o Regression Algorithms
 - o Classification Algorithms
 - o Instance-based Algorithms
 - o Regularization Algorithms

- Bayesian Algorithms
- Clustering Algorithms
- Artificial Neural Network Algorithms
- Dimensionality Reduction Algorithms
- Ensemble Algorithms

3. Tài liệu tham khảo

Học có giám sát và Học không giám sát



ChatGPT

Học có giám sát và không giám sát là hai loại chính của các phương pháp học máy:

1. Học có giám sát (Supervised learning):

- Trong học có giám sát, mỗi mẫu dữ liệu đầu vào được gán nhãn (label) hoặc được đánh giá trước.
- Mục tiêu của thuật toán là học cách dự đoán đúng nhãn cho các mẫu dữ liệu mới dựa trên các mẫu dữ liệu đã được gán nhãn.
- Các thuật toán phổ biến trong học có giám sát bao gồm: Linear Regression, Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forests, Neural Networks, và các thuật toán học sâu khác.

2. Học không giám sát (Unsupervised learning):

- Trong học không giám sát, không có nhãn hoặc đánh giá trước cho các mẫu dữ liệu.
- Mục tiêu của thuật toán là khám phá cấu trúc ẩn của dữ liệu, thường là bằng cách phân nhóm dữ liệu hoặc giảm số chiều của dữ liệu.
- Các thuật toán phổ biến trong học không giám sát bao gồm: K-means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), t-SNE (t-distributed Stochastic Neighbor Embedding), và Autoencoders.

Điểm khác biệt chính giữa học có giám sát và không giám sát là sự có hoặc không có sự hướng dẫn từ dữ liệu nhãn trong quá trình học. Trong học có giám sát, mô hình được huấn luyện dựa trên dữ liệu đã được gán nhãn, trong khi trong học không giám sát, mô hình phải tự học từ dữ liệu không có nhãn.



Dưới đây là các chú thích về trường hợp sử dụng của các thuật toán phổ biến trong machine learning

Dưới đây là các chú thích về trường hợp sử dụng của các thuật toán phổ biến trong machine learning:

- Hồi quy Tuyến tính (Linear Regression): Sử dụng khi mối quan hệ giữa biến đầu vào và biến đầu ra là tuyến tính. Đầu ra là các giá trị liên tục. Dễ hiểu và đơn giản để triển khai.
- Hồi quy Logistic (Logistic Regression): Sử dụng khi muốn dự đoán xác suất rơi vào một trong hai lớp. Đầu ra là biến phân loại hoặc biến nhị phân. Thường được sử dụng trong bài toán phân loại nhị phân.
- Máy Vector hỗ trợ (Support Vector Machine - SVM): Sử dụng khi muốn phân loại dữ liệu vào các lớp khác nhau. Hiệu quả trong các không gian chiều cao. Phù hợp với các bài toán có số lượng biến lớn.
- Cây quyết định (Decision Trees): Sử dụng khi muốn dễ dàng diễn giải và hiểu. Có khả năng làm việc với dữ liệu có tính tự nhiên, không cần chuẩn hóa. Phù hợp với bài toán phân loại và hồi quy.
- Random Forest: Sử dụng khi muốn tăng độ chính xác và tránh overfitting so với cây quyết định. Phù hợp với bài toán phân loại và hồi quy. Có thể xử lý các bài toán với số lượng biến lớn.
- K-Nearest Neighbors (KNN): Sử dụng khi không có giả định nào về phân phối của dữ liệu. Đơn giản và dễ triển khai. Phù hợp với các bài toán có dữ liệu ít và không có nhiều biến đầu vào.
- Mạng Nơ-ron (Neural Networks): Sử dụng khi muốn xử lý các bài toán phức tạp và có khả năng học từ dữ liệu. Đặc biệt hiệu quả trong việc xử lý dữ liệu không cấu trúc như hình ảnh, văn bản, âm thanh. Độ hồi số lượng dữ liệu lớn để huấn luyện.
- Mạng nơ-ron hồi tiếp (Recurrent Neural Networks - RNN): Sử dụng khi làm việc với dữ liệu có tính tuần tự như văn bản, âm thanh, chuỗi thời gian. Có khả năng xử lý dữ liệu dạng chuỗi và lưu trữ thông tin lâu dài.
- Mạng nơ-ron tích chập (Convolutional Neural Networks - CNN): Sử dụng khi làm việc với dữ liệu không gian như hình ảnh. Hiệu quả trong việc nhận diện hình ảnh và trích xuất đặc trưng.
- Mạng Generative Adversarial Networks (GAN): Sử dụng khi muốn tạo ra dữ liệu mới từ dữ liệu đã có. Đặc biệt phù hợp trong các lĩnh vực như tạo hình ảnh nhân tạo, tạo văn bản.

Nhớ rằng, lựa chọn thuật toán phù hợp còn phụ thuộc vào đặc điểm của dữ liệu, mục tiêu của bài toán và cách tiếp cận của bạn đối với bài toán cụ thể.

3. Linear regression

- Sử dụng khi mối quan hệ giữa biến đầu vào và biến đầu ra là tuyến tính.
- Đầu ra là các giá trị liên tục.
- Dễ hiểu và đơn giản để triển khai.

Reserch : [\[ML\] Hồi quy tuyến tính \(Linear Regression\) \(dominhhai.github.io\)](#)

Bài toán

Giải thích công thức :

Nên giờ cần 1 hàm để đánh giá là đường thẳng với bộ tham số $(w_0, w_1) = (0, 1)$ có tốt hay không.

Với mỗi điểm dữ liệu (x_i, y_i) độ chênh lệch giữa giá thật và giá dự đoán được tính bằng: $\frac{1}{2} * (\hat{y}_i - y_i)^2$.

Và độ chênh lệch trên toàn bộ dữ liệu tính bằng tổng chênh lệch của từng điểm:

=> Giải thích vì sao không phải là $\hat{y} - y$ mà phải bình phương và chia đôi .

[[Machine Learning cơ bản \(machinelearningcaban.com - linearregression\)](#)]

2. Phân tích toán học

2.1. Dạng của Linear Regression

Trong phương trình (1) phía trên, nếu chúng ta đặt $\mathbf{w} = [w_0, w_1, w_2, w_3]^T$ là vector (cột) hệ số cần phải tối ưu và $\bar{\mathbf{x}} = [1, x_1, x_2, x_3]$ (đọc là *x bar* trong tiếng Anh) là vector (hàng) dữ liệu đầu vào *mở rộng*. Số 1 ở đầu được thêm vào để phép tính đơn giản hơn và thuận tiện cho việc tính toán. Khi đó, phương trình (1) có thể được viết lại dưới dạng:

$$y \approx \bar{\mathbf{x}}\mathbf{w} = \hat{y}$$

Chú ý rằng $\bar{\mathbf{x}}$ là một vector hàng. ([Xem thêm về ký hiệu vector hàng và cột tại đây](#))

2.2. Sai số dự đoán

Chúng ta mong muốn rằng sự sai khác e giữa giá trị thực y và giá trị dự đoán \hat{y} (đọc là *y hat* trong tiếng Anh) là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

$$\frac{1}{2}e^2 = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \bar{\mathbf{x}}\mathbf{w})^2$$

trong đó hệ số $\frac{1}{2}$ (*lại*) là để thuận tiện cho việc tính toán (khi tính đạo hàm thì số $\frac{1}{2}$ sẽ bị triệt tiêu). Chúng ta cần e^2 vì $e = y - \hat{y}$ có thể là một số âm, việc nói e nhỏ nhất sẽ không đúng vì khi $e = -\infty$ là rất nhỏ nhưng sự sai lệch là rất lớn. Bạn đọc có thể tự đặt câu hỏi: **tại sao không dùng trị tuyệt đối $|e|$ mà lại dùng bình phương e^2 ở đây?** Câu trả lời sẽ có ở phần sau.

2.3. Hàm mất mát

Cách gọi: \hat{y} : *y hat*

Trong phương trình (1) phía trên, nếu chúng ta đặt $\mathbf{w} = [w_0, w_1, w_2, w_3]^T$ là vector (cột) hệ số cần phải tối ưu và $\bar{\mathbf{x}} = [1, x_1, x_2, x_3]$ (đọc là *x bar* trong tiếng Anh) là vector (hàng) dữ liệu đầu vào *mở rộng*. Số 1 ở đầu được thêm vào để phép tính đơn giản hơn và thuận tiện cho việc tính toán. Khi đó, phương trình (1) có thể được viết lại dưới dạng:

Chúng ta mong muốn rằng sự sai khác e giữa giá trị thực y và giá trị dự đoán \hat{y} (đọc là *y hat* trong tiếng Anh) là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

- ❖ y hat có nghĩa là y mũ (hat tiếng anh là mũ)

Thuật toán gradient descent (Thuật toán xuồng dốc)

roud

a& Ask Copilot

-

+

↪

57

of 301

?

¶

3.5 So sánh các loss function

57

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$
$$\hat{y} = X * W = \begin{bmatrix} w_0 + w_1 * x_1 \\ w_0 + w_1 * x_2 \\ \dots \\ w_0 + w_1 * x_n \end{bmatrix}$$

Hình 3.14: Biểu diễn bài toán dạng ma trận

$X[:, i]$ hiểu là ma trận kích thước $n*1$ lấy dữ liệu từ cột thứ i của ma trận X , nhưng do trong python chỉ số bắt đầu từ 0, nên cột đầu tiên là cột 0, cột thứ hai là cột 1, Phép tính $\text{sum}(X)$ là tính tổng tất cả các phần tử trong ma trận X .

$$X[:, 1] = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

$$\text{sum}(X[:, 1]) = x_1 + x_2 + \dots + x_n$$

$$\frac{dJ}{dw_0} = \text{sum}(\hat{y} - y)$$

$$\frac{dJ}{dw_1} = \text{sum}(X[:, 1] \otimes (\hat{y} - y))$$

$$\frac{dJ}{dw} = X^T * (\hat{y} - y)$$

❖ Còn chỗ này chưa hiểu !!!

❖ Thật ra ở đây là chỉ một biến đầu vào X , công thức này có thể tổng quát cho đa biến đầu vào X

❖ Xem thêm ở : <https://machinelearningcaban.com/2016/12/28/linearregression/>

❖ X^T là ma trận chuyển vị để có thể nhân lại với $(\hat{y} - y)$

nhưng mình vẫn chưa hiểu tại sao lại có công thức đó . Có thể đây là tổng quát cho nhiều biến đầu vào X

Trước khi đi tìm lời giải, chúng ta đơn giản hóa phép toán trong phương trình hàm măt măt (2). Đặt $\mathbf{y} = [y_1; y_2; \dots; y_N]$ là một vector cột chứa tất cả các *output* của *training data*; $\bar{\mathbf{X}} = [\bar{\mathbf{x}}_1; \bar{\mathbf{x}}_2; \dots; \bar{\mathbf{x}}_N]$ là ma trận dữ liệu đầu vào (mở rộng) mà mỗi hàng của nó là một điểm dữ liệu. Khi đó hàm số măt măt $\mathcal{L}(\mathbf{w})$ được viết dưới dạng ma trận đơn giản hơn:

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N (y_i - \bar{\mathbf{x}}_i \mathbf{w})^2 \\ &= \frac{1}{2} \|\mathbf{y} - \bar{\mathbf{X}}\mathbf{w}\|_2^2 \quad (3)\end{aligned}$$

với $\|\mathbf{z}\|_2$ là Euclidean norm (chuẩn Euclid, hay khoảng cách Euclid), nói cách khác $\|\mathbf{z}\|_2^2$ là tổng của bình phương mỗi phần tử của vector \mathbf{z} . Tới đây, ta đã có một dạng đơn giản của hàm măt măt được viết như phương trình (3).

- ❖ Mỗi \bar{x}_i đã là một mảng (các giá trị chạy từ i đến N), và \bar{X} là toàn bộ ma trận chứa nhiều cột (biến)(đặc trưng) đầu vào .

2.4. Nghiệm cho bài toán Linear Regression

Cách phỗ biến nhất để tìm nghiệm cho một bài toán tối ưu (chúng ta đã biết từ khi học cấp 3) là giải phương trình đạo hàm (gradient) bằng 0! Tất nhiên đó là khi việc tính đạo hàm và việc giải phương trình đạo hàm bằng 0 không quá phức tạp. Thật may mắn, với các mô hình tuyến tính, hai việc này là khả thi.

Đạo hàm theo \mathbf{w} của hàm măt măt là:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \bar{\mathbf{X}}^T (\bar{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

Các bạn có thể tham khảo bảng đạo hàm theo vector hoặc ma trận của một hàm số trong [mục D.2](#) của tài liệu này. Đến đây tôi xin quay lại câu hỏi ở phần [Sai số dự đoán](#) phía trên về việc tại sao không dùng trị tuyệt đối mà lại dùng bình phương. Câu trả lời là hàm bình phương có đạo hàm tại mọi nơi, trong khi hàm trị tuyệt đối thì không (đạo hàm không xác định tại 0).

Phương trình đạo hàm bằng 0 tương đương với:

$$\bar{\mathbf{X}}^T \bar{\mathbf{X}}\mathbf{w} = \bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b} \quad (4)$$

(ký hiệu $\bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b}$ nghĩa là **đặt $\bar{\mathbf{X}}^T \mathbf{y}$ bằng \mathbf{b}**).

Nếu ma trận vuông $\mathbf{A} \triangleq \bar{\mathbf{X}}^T \bar{\mathbf{X}}$ khả nghịch (non-singular hay invertible) thì phương trình (4) có nghiệm duy nhất: $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$.

- ❖ Ở đây có giải thích một phần của cái mà mình khoanh đỏ lại đó .

MSE & MAE Loss function

Loss function

Nên giờ cần 1 hàm để đánh giá là đường thẳng với bộ tham số $(w_0, w_1) = (0, 1)$ có tốt hay không.

Với mỗi điểm dữ liệu (x_i, y_i) độ chênh lệch giữa giá thật và giá dự đoán được tính bằng: $\frac{1}{2} * (\hat{y}_i - y_i)^2$.

Và độ chênh lệch trên toàn bộ dữ liệu tính bằng tổng chênh lệch của từng điểm:

$$J = \frac{1}{2} * \frac{1}{N} * \left(\sum_{i=1}^N (\hat{y}_i - y_i)^2 \right) \quad (N \text{ là số điểm dữ liệu}). \text{ Nhận xét:}$$

- J không âm
- J càng nhỏ thì đường thẳng càng gần điểm dữ liệu. Nếu J = 0 thì đường thẳng đi qua tất cả các điểm dữ liệu.

J được gọi là loss function, hàm để đánh giá xem bộ tham số hiện tại có tốt với dữ liệu không.

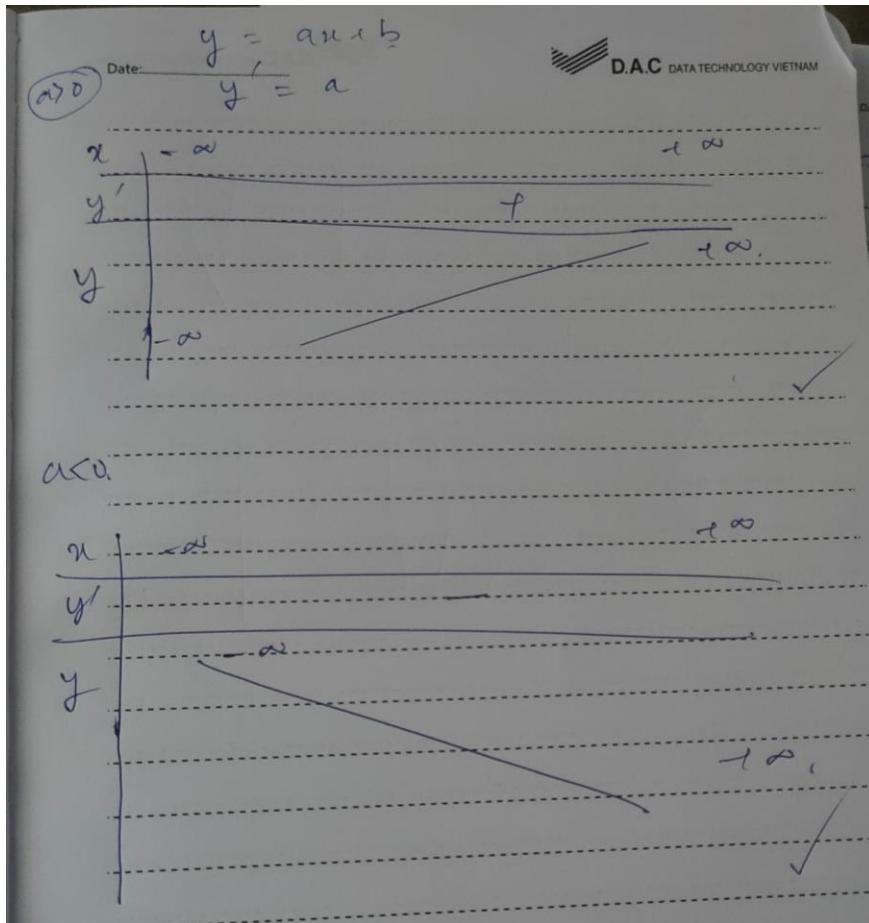
=> Bài toán tìm đường thẳng gần các điểm dữ liệu nhất trở thành tìm w_0, w_1 sao cho hàm J đạt giá trị nhỏ nhất.

Tóm tắt: Cần tìm đường thẳng (model) fit nhất với dữ liệu, tương ứng với việc tìm tham số w_0, w_1 để cực tiểu hóa hàm J.

Giờ cần một thuật toán để tìm giá trị nhỏ nhất của hàm J(w_0, w_1). Đó chính là thuật toán gradient descent.

- ❖ Sử dụng hàm **Loss function (hàm mất mát)** để đánh giá độ chính xác
- ❖ J càng nhỏ thì càng tốt .

Khảo sát hàm bội một :



❖ Hàm số tuyến tính (đường thẳng)

MEA & MSE

- MAE (Mean Absolute Error) (Trung bình Tuyệt đối của lỗi)
- MSE (Mean Square Error) (Trung bình Bình phương lỗi)

3.5.1 Mean Absolute Error, L1 Loss

Mean Absolute Error (MAE) hay còn được gọi là L1 Loss là một loss function được sử dụng cho các mô hình hồi quy, đặc biệt cho các mô hình hồi quy tuyến tính. MAE được tính bằng tổng các trị tuyệt đối của hiệu giữa giá trị thực (y_i : target) và giá trị mà mô hình của chúng ra dự đoán (\hat{y}_i : predicted).

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (3.1)$$

3.5.2 Mean Square Error, L2 Loss

Mean Square Error (MSE) hay còn được gọi là L2 Loss là một loss function cũng được sử dụng cho các mô hình hồi quy, đặc biệt là các mô hình hồi quy tuyến tính. MSE được tính bằng tổng các

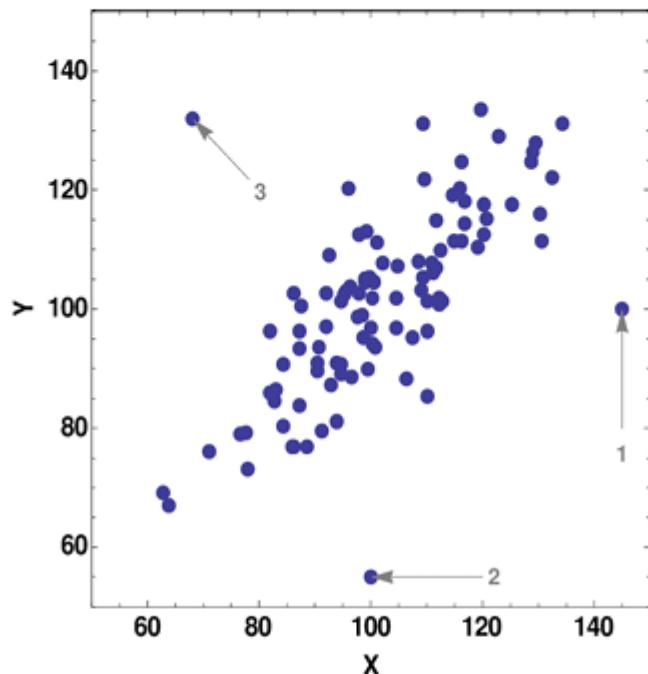
bình phương của hiệu giữa giá trị thực (y_i : target) và giá trị mà mô hình của chúng ra dự đoán (\hat{y}_i : predicted).

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (3.2)$$

So sánh MSE và MAE

So sánh MAE và MSE (L1 Loss và L2 Loss)

Để tìm điểm cực tiểu của các hàm số, thông thường cách đơn giản nhất chúng ta nghĩ đến chính là tìm đạo hàm của hàm số rồi tìm điểm mà tại đó đạo hàm của hàm số bằng 0. Như vậy sẽ có 1 bước đạo hàm, hiển nhiên tìm đạo hàm của MSE sẽ đơn giản hơn rất nhiều so với MAE. Tuy nhiên, MAE thì đem lại kết quả tốt hơn đối với các dữ liệu có outlier (điểm dị biệt - là những điểm có giá trị khác xa so với phần còn lại của dữ liệu).



Hình 3.16: Dữ liệu bị outlier

Đầu tiên hãy để ý đến MSE, ta có $y_i - \hat{y}_i = e$, e^2 sẽ vô cùng lớn nếu $e > 1$. Nếu bậc của e càng lớn thì giá trị hàm Loss cũng càng lớn hơn. Vì vậy nếu như chúng ta có 1 outlier trong bộ dữ liệu, giá trị của e sẽ vô cùng lớn, có thể tiến tới ∞ và khi đó, e^2 có thể sẽ tiến tới ∞ , khi đó giá trị MSE cực kì lớn.

Ngược lại với MSE, nếu ta có giá trị e lớn ($e > 1$) thì $|e|$ vẫn sẽ lớn, nhưng hiển nhiên nhỏ hơn nhiều so với e^2 .

Do đó khi tối ưu loss function, L2 phạt mạnh hơn với các điểm outlier và model sẽ bị kéo về phía outlier hơn. Do đó MSE bị ảnh hưởng bởi outlier và L1 tốt hơn đối với các dữ liệu có outlier.

Tuy vậy đạo hàm của MAE (hay L1 Loss) tính toán khó hơn nhiều so với MSE (hay L2 Loss). Cùng thử tính toán nhé.

Tiêu chí xây dựng hàm loss function là không âm và loss càng nhỏ thì model càng tốt với dữ liệu. Từ đâu đến giờ có lẽ các bạn vẫn thắc mắc tại sao trong các hàm loss chỉ dùng trị tuyệt đối, bậc 2, thế bậc $2n$ thì sao? Câu trả lời là bạn chọn bậc càng cao thì càng bị ảnh hưởng bởi outlier. Ví dụ so sánh các loss function ở hình 3.17. Và 2 loss bạn hay gặp nhất sẽ là L1 (Mean Absolute Error) và L2 (Mean Square Error).

- ❖ MAE khó đạo hàm hơn nhưng sẽ ít bị ảnh hưởng bởi outlier (ngoại lệ)
- ❖ MSE dễ đạo hàm hơn nhưng dễ bị ảnh hưởng bởi giá trị ngoại lệ do nó các mũ $2n$

Mũ $2n$ hay giá trị tuyệt đối đơn giản là làm cho sai số không âm ($y - \hat{y}$) không âm

Khảo sát hàm số bậc 2 sau khi đã có hàm măt măt.

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

- ❖ Ta có
- ❖ Rõ ràng MSE nhỏ nhất khi $(y - \hat{y}) = 0$

Giải bằng đại số tuyến tính

Ma trận nghịch đảo

Điều kiện để có ma trận nghịch đảo

Một ma trận vuông A có thể có ma trận nghịch đảo A^{-1} nếu và chỉ nếu nó là một ma trận không suy biến. Điều này có nghĩa là một ma trận sẽ có ma trận nghịch đảo khi và chỉ khi thỏa mãn các điều kiện sau:

- Kích thước:** Ma trận A phải là một ma trận vuông, nghĩa là số hàng bằng số cột.
- Định thức khác không:** Định thức của ma trận A phải khác không ($\det(A) \neq 0$). Nếu định thức bằng không, tức là ma trận là suy biến và không có ma trận nghịch đảo.
- Hạng (Rank) đầy đủ:** Hạng của ma trận A phải bằng số hàng hoặc số cột của nó. Trong trường hợp hạng không đầy đủ (không đủ để tạo ra một hệ thống đủ điều kiện), ma trận sẽ không có ma trận nghịch đảo.
- Cột tuyến tính độc lập:** Các cột của ma trận A phải tạo thành một hệ cột tuyến tính độc lập. Điều này đảm bảo rằng không có một cột nào có thể được biểu diễn dưới dạng tổ hợp tuyến tính của các cột khác, và do đó có thể tìm được một ma trận nghịch đảo.

Nếu ma trận A thỏa mãn các điều kiện trên, tức là nó không suy biến, thì có thể tính toán ma trận nghịch đảo A^{-1} của nó. Ma trận nghịch đảo sẽ cho phép giải hệ phương trình tuyến tính và thực hiện nhiều phép toán khác trong đại số tuyến tính.

Ma trận nghịch đảo (Inverse Matrix):

- Cho một ma trận vuông không suy biến A , ma trận nghịch đảo của nó, thường được ký hiệu là A^{-1} , là một ma trận sao cho tích của nó với ma trận ban đầu là ma trận đơn vị I .
- Cụ thể, nếu $A \cdot A^{-1} = A^{-1} \cdot A = I$, thì A^{-1} được gọi là ma trận nghịch đảo của A .

Ma trận khả nghịch (Ma trận không suy biến)

Ma trận khả nghịch (Invertible Matrix):

- Một ma trận vuông A được gọi là khả nghịch (hoặc không suy biến) nếu nó có ma trận nghịch đảo.
- Nếu một ma trận là không suy biến, nghĩa là bạn có thể tìm được một ma trận nghịch đảo cho nó.

Định thức và cách tính ma trận nghịch đảo

Chúng ta cùng xem xét một ví dụ cụ thể về một ma trận và ma trận nghịch đảo, sao cho tích của ma trận với ma trận nghịch đảo và ngược lại bằng ma trận đơn vị.

Giả sử chúng ta có một ma trận A như sau:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

Để tính ma trận nghịch đảo A^{-1} , chúng ta có thể sử dụng một số phương pháp tính toán như phương pháp đối xứng-Gauss hoặc phương pháp sử dụng định thức. Tuy nhiên, ở đây, chúng ta sẽ sử dụng công thức cụ thể:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A)$$

Trong đó, $\text{adj}(A)$ là ma trận phụ hợp của A , và $\det(A)$ là định thức của A .

Đầu tiên, chúng ta tính định thức của A :

$$\det(A) = 2 \cdot 3 - 1 \cdot 1 = 6 - 1 = 5$$

Tiếp theo, tính ma trận phụ hợp của A :

$$\text{adj}(A) = \begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix}$$

Sau đó, chúng ta tính ma trận nghịch đảo A^{-1} :

$$A^{-1} = \frac{1}{5} \cdot \begin{bmatrix} 3 & -1 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} \frac{3}{5} & -\frac{1}{5} \\ -\frac{1}{5} & \frac{2}{5} \end{bmatrix}$$

Bây giờ, chúng ta sẽ kiểm tra xem phép nhân của A với A^{-1} và của A^{-1} với A có bằng ma trận đơn vị không.

1. $A \cdot A^{-1}$:

$$\begin{aligned} A \cdot A^{-1} &= \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \cdot \begin{bmatrix} \frac{3}{5} & -\frac{1}{5} \\ -\frac{1}{5} & \frac{2}{5} \end{bmatrix} \\ &= \begin{bmatrix} \frac{6}{5} - \frac{1}{5} & -\frac{2}{5} + \frac{2}{5} \\ \frac{3}{5} - \frac{3}{5} & -\frac{1}{5} + \frac{6}{5} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \end{aligned}$$

2. $A^{-1} \cdot A$:

$$\begin{aligned} A^{-1} \cdot A &= \begin{bmatrix} \frac{3}{5} & -\frac{1}{5} \\ -\frac{1}{5} & \frac{2}{5} \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \\ &= \begin{bmatrix} \frac{6}{5} - \frac{1}{5} & -\frac{3}{5} + \frac{3}{5} \\ \frac{5}{5} - \frac{5}{5} & -\frac{1}{5} + \frac{6}{5} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \end{aligned}$$

Do đó, ta đã chứng minh được rằng $A \cdot A^{-1} = A^{-1} \cdot A = I$, trong đó I là ma trận đơn vị. Điều này chỉ ra rằng A^{-1} là ma trận nghịch đảo của A .

Ma trận chuyển vị



ChatGPT

Ma trận chuyển vị là một phép biến đổi ma trận bằng cách đổi vị trí các hàng thành các cột (và ngược lại). Để biểu diễn ma trận chuyển vị, chúng ta sử dụng ký hiệu A^T .

Cho một ma trận A có dạng:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Ma trận chuyển vị của A , ký hiệu là A^T , sẽ có dạng:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}$$

Nói cách khác, các phần tử ở hàng i của ma trận A sẽ trở thành các phần tử ở cột i của ma trận A^T , và ngược lại.

Ví dụ, nếu chúng ta có ma trận A như sau:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Thì ma trận chuyển vị của A , ký hiệu là A^T , sẽ là:

$$A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

Đây là phép biến đổi đơn giản nhưng rất hữu ích trong nhiều ứng dụng, đặc biệt trong việc xử lý dữ liệu và tính toán đại số tuyến tính.

Các phép toán trên ma trận chuyển vị



ChatGPT

Các phép toán trên ma trận chuyển vị cung cấp một số tính chất quan trọng và tiện lợi trong đại số tuyến tính. Dưới đây là một số phép toán phổ biến được thực hiện trên ma trận chuyển vị:

1. Chuyển vị của chuyển vị:

- Phép chuyển vị của một ma trận chuyển vị sẽ trở lại với ma trận ban đầu:

$$(A^T)^T = A$$

2. Cộng và trừ ma trận chuyển vị:

- Chuyển vị của tổng của hai ma trận là tổng của các chuyển vị của chúng:

$$(A + B)^T = A^T + B^T$$

- Tương tự, chuyển vị của hiệu của hai ma trận là hiệu của các chuyển vị của chúng:

$$(A - B)^T = A^T - B^T$$

3. Nhân ma trận với một số vô hướng:

- Chuyển vị của tích của một ma trận với một số vô hướng là tích của chuyển vị của ma trận với số vô hướng đó:

$$(kA)^T = k \cdot A^T$$

4. Nhân ma trận:

- Chuyển vị của tích của hai ma trận là tích của chuyển vị của ma trận thứ hai với chuyển vị của ma trận thứ nhất, nhưng thứ tự ngược lại:

$$(AB)^T = B^T \cdot A^T$$

5. Ma trận chuyển vị của ma trận đơn vị:

- Ma trận chuyển vị của ma trận đơn vị sẽ vẫn là ma trận đơn vị:

$$(I)^T = I$$

6. Chuyển vị của ma trận nghịch đảo:

- Chuyển vị của ma trận nghịch đảo của một ma trận chuyển vị là nghịch đảo của chuyển vị của ma trận ban đầu:

$$(A^{-1})^T = (A^T)^{-1}$$

Các tính chất trên rất hữu ích trong việc phân tích và tính toán trên ma trận chuyển vị, đặc biệt trong các ứng dụng thống kê, xử lý tín hiệu, và trong học máy.

Các phép toán trên ma trận nghịch đảo



ChatGPT

Các phép toán trên ma trận nghịch đảo cung cấp nhiều tính chất hữu ích trong đại số tuyến tính và được sử dụng trong nhiều ứng dụng khác nhau. Dưới đây là một số phép toán phổ biến trên ma trận nghịch đảo:

1. Nhân ma trận với ma trận nghịch đảo:

- Khi nhân một ma trận với ma trận nghịch đảo của nó, ta sẽ thu được ma trận đơn vị:

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

- Đây là một trong những tính chất quan trọng nhất của ma trận nghịch đảo và thường được sử dụng để giải hệ phương trình tuyến tính và thực hiện các phép toán khác.

2. Nhân ma trận với ma trận nghịch đảo của ma trận khác:

- Khi nhân một ma trận với ma trận nghịch đảo của một ma trận khác, ta cũng có thể thu được kết quả tương tự như khi nhân với ma trận nghịch đảo của chính nó:

$$A \cdot B^{-1} = (B^{-1})^T \cdot A^T$$

3. Chuyển vị của ma trận nghịch đảo:

- Chuyển vị của ma trận nghịch đảo là nghịch đảo của chuyển vị của ma trận ban đầu:

$$(A^{-1})^T = (A^T)^{-1}$$

- Điều này có thể hữu ích trong một số tình huống khi ta cần phải làm việc với chuyển vị của ma trận nghịch đảo.

4. Ma trận nghịch đảo của ma trận nghịch đảo:

- Ma trận nghịch đảo của ma trận nghịch đảo là chính ma trận ban đầu:

$$(A^{-1})^{-1} = A$$

- Điều này cho thấy rằng ma trận nghịch đảo của một ma trận nghịch đảo là chính ma trận đó.

5. Nhân ma trận với một số vô hướng và ma trận nghịch đảo:

- Khi nhân một ma trận với một số vô hướng và sau đó nhân với ma trận nghịch đảo của chính nó, ta cũng thu được ma trận đơn vị:

$$k \cdot A \cdot (A^{-1}) = A \cdot (A^{-1}) \cdot k = I$$

Các tính chất trên cung cấp một cái nhìn tổng quan về cách làm việc với ma trận nghịch đảo và làm cho nó trở thành một công cụ quan trọng trong việc giải các hệ phương trình tuyến tính, tối ưu hóa, và nhiều ứng dụng khác trong toán học và khoa học máy.

Định nghĩa của Euclidean norm



ChatGPT

Định nghĩa của Euclidean norm, còn được gọi là độ dài Euclidean hoặc độ dài Euclidean, là một khái niệm toán học quan trọng trong không gian vector Euclidean. Euclidean norm được sử dụng để đo lường "độ lớn" của một vector trong không gian Euclidean nhiều chiều.

Trong không gian Euclidean n chiều, một vector \mathbf{x} được biểu diễn bằng n thành phần số thực:

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$

Độ dài Euclidean của vector \mathbf{x} , ký hiệu là $\|\mathbf{x}\|$ hoặc $\|\mathbf{x}\|_2$, được tính bằng căn bậc hai của tổng bình phương các thành phần của vector:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

Tính chất quan trọng của Euclidean norm là nó thể hiện khoảng cách từ điểm gốc đến điểm được đại diện bởi vector \mathbf{x} trong không gian n chiều. Độ lớn của vector được định nghĩa bằng cách tính khoảng cách Euclidean từ điểm này đến điểm gốc, giống như việc đo độ dài của một đoạn thẳng trong không gian 2D hoặc 3D bằng công thức khoảng cách Euclidean.

Euclidean norm là một trong những loại norm phổ biến nhất trong toán học và có nhiều ứng dụng trong lĩnh vực học máy, đặc biệt là trong việc đo lường độ lớn của vector và trong các thuật toán như clustering, regularization trong huấn luyện mô hình, và các thuật toán tối ưu hóa.

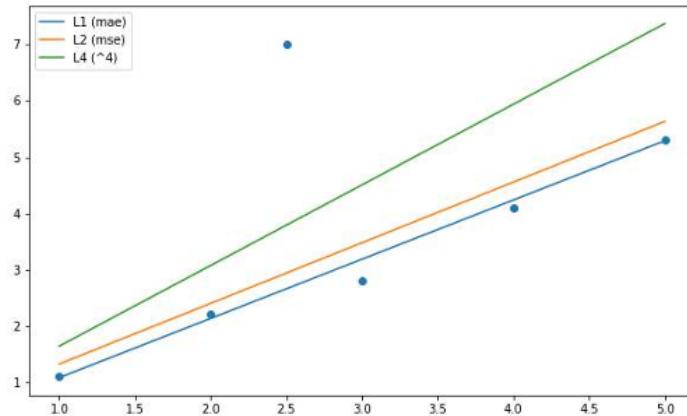


3.6 Giải bằng đại số tuyến tính

$w = [w_0, w_1, w_2, \dots, w_m]^T$ là vecto hệ số cần phải tối ưu, w_0 thường hay được gọi là bias.
 $x^{(i)} = [1, x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}]$ là dữ liệu thứ i trong bộ n số lượng dữ liệu quan sát được, mỗi dữ liệu có m giá trị.
 $\implies \hat{y}^{(i)} = \tilde{x}^{(i)}w$

Biểu diễn tổng quát bài toán Linear Regression:

$$X \in \mathbb{R}^{n*(m+1)}, w \in \mathbb{R}^{(m+1)*1}, y \in \mathbb{R}^{n*1}$$



Hình 3.17: So sánh các loss function

$$X = \begin{pmatrix} x^{(1)} \\ x^{(2)} \\ \dots \\ x^{(n)} \end{pmatrix}, w = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{pmatrix}, \hat{y} = Xw = \begin{pmatrix} w_0 + w_1x_1^{(1)} + \dots + w_mx_m^{(1)} \\ w_0 + w_1x_1^{(2)} + \dots + w_mx_m^{(2)} \\ \dots \\ w_0 + w_1x_1^{(n)} + \dots + w_mx_m^{(n)} \end{pmatrix}, y - \hat{y} = \begin{pmatrix} y^{(1)} - (w_0 + w_1x_1^{(1)} + \dots + w_mx_m^{(1)}) \\ y^{(2)} - (w_0 + w_1x_1^{(2)} + \dots + w_mx_m^{(2)}) \\ \dots \\ y^{(n)} - (w_0 + w_1x_1^{(n)} + \dots + w_mx_m^{(n)}) \end{pmatrix}$$

$$\text{Hàm loss: } L = \frac{1}{2} * \frac{1}{n} \sum_{i=1}^n \left(y^{(i)} - \hat{y}^{(i)} \right)^2 = \frac{1}{2} * \frac{1}{n} \sum_{i=1}^n \left(y^{(i)} - x^{(i)}w \right)^2$$

$$\text{Định nghĩa Euclidean norm: } \|z\|_2 = (z_1^2 + z_2^2 + \dots + z_n^2)^{\frac{1}{2}} \implies \|z\|_2^2 = (z_1^2 + z_2^2 + \dots + z_n^2)$$

$$\implies L = \frac{1}{2} * \frac{1}{n} \|y - \hat{y}\|_2^2 = \frac{1}{2} * \frac{1}{n} \|y - Xw\|_2^2 = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y})$$

=> Mũ ½ ở đây chính là căn (theo định nghĩa Euclidean norm) thì chính là căn ...

Giải thích chi tiết:

$$\begin{aligned}
 + &> (Xw)^T = w^T X^T \\
 + &> x^{(i)}w = \begin{bmatrix} 1 & x_1^{(i)} & \dots & x_m^{(i)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{bmatrix} = w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)} \\
 \implies & \frac{d(x^{(i)}w)}{dw} = \begin{bmatrix} \frac{d(x^{(i)}w)}{dw_0} & \frac{d(x^{(i)}w)}{dw_1} & \dots & \frac{d(x^{(i)}w)}{dw_m} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(i)} & \dots & x_m^{(i)} \end{bmatrix} = x^{(i)} \\
 \implies & \frac{d(Xw)}{dw} = X
 \end{aligned}$$

3.7 Python code

$$\begin{aligned}
 + &> wx^{(i)} = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{bmatrix} \begin{bmatrix} 1 & x_1^{(i)} & \dots & x_m^{(i)} \end{bmatrix} = w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)} \\
 \implies & \frac{d(wx^{(i)})}{dw} = \begin{bmatrix} \frac{d(wx^{(i)})}{dw_0} \\ \frac{d(wx^{(i)})}{dw_1} \\ \dots \\ \frac{d(wx^{(i)})}{dw_m} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \dots \\ x_m^{(i)} \end{bmatrix} = x^{(i)T} \\
 \implies & \frac{d(wX)}{dw} = X^T
 \end{aligned}$$

$$\implies \frac{d(wX)}{dw} = X^T$$

$$+ > \frac{d(w^T X w)}{dw} = w^T X + \frac{d((Xw)^T w)}{dw} = w^T X + w^T X^T = w^T (X + X^T) \quad (\text{Nếu } X \text{ đối xứng})$$

$$= 2w^T X$$

*Để ý $X^T X$ đối xứng

$$\implies \frac{dX}{dw} = 0, \frac{d(Xw)}{dw} = X, \frac{d(wX)}{dw} = X^T, \frac{d(w^T X w)}{dw} = w^T (X + X^T)$$

$$A = (y - \hat{y})^T * (y - \hat{y}) = (y - Xw)^T * (y - Xw)$$

$$= (y^T - w^T X^T) (y - Xw) = y^T y - y^T Xw - w^T X^T y + w^T X^T Xw$$

$$\begin{aligned} \implies A'_w &= 0 - y^T X - ((y^T X) w)'_w + 2w^T X^T X \\ &= -y^T X - ((y^T X) w)'_w + 2w^T X^T X \\ &= -y^T X - y^T X + 2w^T X^T X \\ &= -y^T X - y^T X + 2w^T X^T X \\ &= 0 \\ \Leftrightarrow 2w^T X^T X &= 2y^T X \\ \Leftrightarrow w^T X^T X &= y^T X \\ \Leftrightarrow (X^T X) * w &= X^T y \\ \Leftrightarrow w &= (X^T X)^{-1} * X^T y \end{aligned}$$

Nếu $X^T X$ khả nghịch, thì L có nghiệm duy nhất: $w = (X^T X)^{-1} X^T y$

Nếu không khả nghịch, ta có thể sử dụng khái niệm giả nghịch đảo, bạn đọc có thể tự tìm hiểu thêm ở đây.

Linear regression Da biến

- ❖ Hết sức bình thường . Vẫn là thuật toán đó . Chỉ là ma trận X sẽ mở rộng ra thêm số cột

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix}, y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$$

$$\hat{y} = X * W = \begin{bmatrix} w_0 + w_1 * x_1 \\ w_0 + w_1 * x_2 \\ \dots \\ w_0 + w_1 * x_n \end{bmatrix}$$

Hình 3.14: Biểu diễn bài toán dạng ma trận

$X[:, i]$ hiểu là ma trận kích thước $n*1$ lấy dữ liệu từ cột thứ i của ma trận X , nhưng do trong python chỉ số bắt đầu từ 0, nên cột đầu tiên là cột 0, cột thứ hai là cột 1, Phép tính $\text{sum}(X)$ là tính tổng tất cả các phần tử trong ma trận X .

- ❖ Nếu một biến đầu vào x thì sẽ chỉ là 2 cột : cột có giá trị toàn 1 và cột một cột có giá trị từ x_i đến x_n như tài liệu
- ❖ Còn nếu nhiều biến đầu vào thì , ví dụ như này thì $x_{1,i}$ là số phòng , $x_{2,i}$ là diện tích và ta sẽ có
- ❖ $W = [w_0, w_1, w_2]$
- ❖ $\begin{bmatrix} 1 & x_{1,1} & \dots & x_{m,1} \\ \dots & \dots & \dots & \dots \\ 1 & x_{1,n} & \dots & x_{m,n} \end{bmatrix}$



ChatGPT

Nếu bạn có nhiều biến đầu vào, vẫn có thể sử dụng thuật toán hồi quy tuyến tính. Tuy nhiên, nếu có nhiều biến đầu vào, việc sử dụng hồi quy tuyến tính có thể đưa đến một số vấn đề, bao gồm:

1. **Overfitting:** Khi có quá nhiều biến đầu vào so với số lượng mẫu dữ liệu, mô hình có thể trở nên quá phức tạp và dễ bị overfitting.
2. **Multicollinearity:** Nếu các biến đầu vào có mức độ tương quan cao với nhau, có thể xảy ra tình trạng multicollinearity, làm cho ước lượng của các hệ số trở nên không ổn định hoặc không chính xác.

Trong trường hợp bạn đang xử lý nhiều biến đầu vào, có một số thuật toán khác có thể phù hợp hơn:

1. **Ridge Regression:** Ridge Regression thêm một thành phần điều chuẩn vào hàm mất mát để giảm overfitting, đặc biệt là trong trường hợp có multicollinearity.
2. **Lasso Regression:** Lasso Regression cũng thêm một thành phần điều chuẩn, nhưng nó có thể được sử dụng để thực hiện lựa chọn biến đầu vào (feature selection), giúp loại bỏ các biến không quan trọng.
3. **ElasticNet Regression:** ElasticNet kết hợp cả Ridge và Lasso Regression, làm giảm overfitting và lựa chọn biến đầu vào đồng thời.
4. **Principal Component Regression (PCR):** PCR là một phương pháp giảm số lượng biến đầu vào bằng cách sử dụng phân tích thành phần chính (PCA) trước khi áp dụng hồi quy tuyến tính.
5. **Partial Least Squares Regression (PLS):** PLS cũng giảm số lượng biến đầu vào nhưng tối ưu hóa trực tiếp trên biến phụ thuộc.

Khi vẽ biểu đồ, bạn có thể sử dụng các biểu đồ scatter plot để hiển thị mối quan hệ giữa biến đầu vào và biến đầu ra. Đối với trường hợp có nhiều biến đầu vào, bạn cũng có thể sử dụng các biểu đồ scatter plot 3D hoặc heatmap để hiển thị mức quan hệ giữa biến đầu vào và biến đầu ra.





ChatGPT

Trong trường hợp bạn có nhiều hơn một biến đầu vào (feature) cho mô hình hồi quy tuyến tính, bạn vẫn có thể sử dụng thuật toán hồi quy tuyến tính. Trong trường hợp này, mô hình của bạn sẽ trở thành một mô hình hồi quy tuyến tính đa biến.

Ví dụ, nếu bạn có hai biến đầu vào là diện tích và số phòng, và bạn muốn dự đoán giá nhà, mô hình hồi quy tuyến tính đa biến sẽ có dạng:

$$y = w_0 + w_1 \times \text{Diện tích} + w_2 \times \text{Số phòng}$$

Trong đó:

- y là giá nhà dự đoán.
- w_0 là hệ số chặc (intercept).
- w_1 là hệ số tương ứng với diện tích.
- w_2 là hệ số tương ứng với số phòng.

Công thức trên vẫn là một hàm tuyến tính với các biến đầu vào. Do đó, thuật toán hồi quy tuyến tính vẫn có thể được áp dụng một cách hiệu quả trong trường hợp này. Điều quan trọng là đảm bảo rằng dữ liệu của bạn đáp ứng các giả định của mô hình hồi quy tuyến tính và các điều kiện về độc lập tuyến tính, phân phối đồng nhất và homoscedasticity.



ChatGPT

Trong trường hợp bạn có nhiều hơn một biến đầu vào (feature) cho mô hình hồi quy tuyến tính, bạn vẫn có thể sử dụng thuật toán hồi quy tuyến tính. Trong trường hợp này, mô hình của bạn sẽ trở thành một mô hình hồi quy tuyến tính đa biến.

Ví dụ, nếu bạn có hai biến đầu vào là diện tích và số phòng, và bạn muốn dự đoán giá nhà, mô hình hồi quy tuyến tính đa biến sẽ có dạng:

$$y = w_0 + w_1 \times \text{Diện tích} + w_2 \times \text{Số phòng}$$

Trong đó:

- y là giá nhà dự đoán.
- w_0 là hệ số chận (intercept).
- w_1 là hệ số tương ứng với diện tích.
- w_2 là hệ số tương ứng với số phòng.

Công thức trên vẫn là một hàm tuyến tính với các biến đầu vào. Do đó, thuật toán hồi quy tuyến tính vẫn có thể được áp dụng một cách hiệu quả trong trường hợp này. Điều quan trọng là đảm bảo rằng dữ liệu của bạn đáp ứng các giả định của mô hình hồi quy tuyến tính và các điều kiện về độc lập tuyến tính, phân phối đồng nhất và homoscedasticity.





ChatGPT

Trong bài toán hồi quy tuyến tính hoặc mô hình hồi quy tuyến tính đa biến, $\mathbf{w} = [w_0, w_1, w_2, \dots, w_m]^T$ thường được sử dụng để biểu diễn vectơ tham số của mô hình. Đây là một vectơ dòng (row vector) kích thước $(m + 1) \times 1$, trong đó m là số lượng biến đầu vào (hoặc số lượng đặc trưng) và w_0 thường là hệ số chặn (intercept) hoặc bias của mô hình.

- w_0 thường tương ứng với hệ số chặn của đường thẳng (intercept) trong không gian m chiều.
- w_1, w_2, \dots, w_m là các hệ số tương ứng với các biến đầu vào hoặc đặc trưng (features).

Ví dụ, trong một mô hình hồi quy tuyến tính đơn giản với một biến đầu vào x , vectơ tham số \mathbf{w} có thể được biểu diễn như sau:

$$\mathbf{w} = [w_0, w_1]^T$$

Trong đó:

- w_0 là hệ số chặn của đường thẳng.
- w_1 là hệ số tương ứng với biến đầu vào x .

Trong các mô hình phức tạp hơn với nhiều biến đầu vào, vectơ tham số \mathbf{w} sẽ có số chiều lớn hơn và chứa các hệ số cho mỗi biến đầu vào.

7 Phương pháp đánh giá độ chính xác cho mô hình hồi quy

1. Mean Squared Error (MSE)
2. Mean Absolute Error (MAE)
3. R squared (R^2)
4. Adjusted R squared
5. Root Mean Squared Error (RMSE)
6. Mean Absolute Percentage Error (MAPE)
7. Mean Percentage Error (MPE)



ChatGPT

Dưới đây là công thức và mô tả của hai phương pháp đánh giá độ chính xác trong mô hình hồi quy:

1. Mean Squared Error (MSE):

- Công thức: $MSE = \frac{1}{n} \sum_{i=1}^n (y_{true,i} - y_{pred,i})^2$
- MSE là trung bình của bình phương của sai số giữa giá trị thực tế ($y_{true,i}$) và giá trị dự đoán ($y_{pred,i}$) cho tất cả các mẫu trong tập dữ liệu. MSE càng nhỏ thì mô hình càng tốt.

2. Mean Absolute Error (MAE):

- Công thức: $MAE = \frac{1}{n} \sum_{i=1}^n |y_{true,i} - y_{pred,i}|$
- MAE là trung bình của giá trị tuyệt đối của sai số giữa giá trị thực tế ($y_{true,i}$) và giá trị dự đoán ($y_{pred,i}$) cho tất cả các mẫu trong tập dữ liệu. MAE cũng đo lường độ lớn của sai số giữa giá trị dự đoán và giá trị thực tế, nhưng nó không cộng hai sai số âm và dương lại với nhau. MAE cũng càng nhỏ thì mô hình càng tốt.

Cả hai độ đo này đều sử dụng để đánh giá hiệu suất của mô hình hồi quy, nhưng MSE thường được ưa chuộng hơn vì nó có tính chất tính toán dễ dàng và độ lớn của nó phản ánh độ lớn của sai số bình phương. Trong khi đó, MAE thường được ưa chuộng hơn trong các tình huống mà các điểm dữ liệu có thể chứa các giá trị ngoại lai (outliers) vì nó ít nhạy cảm hơn với các giá trị ngoại lai.



ChatGPT

Dưới đây là công thức toán học của các phương pháp đánh giá độ chính xác cho mô hình hồi quy:

1. **R squared (R^2):**

- Công thức: $R^2 = 1 - \frac{SS_{\text{residual}}}{SS_{\text{total}}}$
- Trong đó:
 - SS_{residual} là tổng bình phương của sai số dự đoán.
 - SS_{total} là tổng bình phương của sự chênh lệch giữa giá trị thực tế và giá trị trung bình của biến phụ thuộc.

2. **Adjusted R squared:**

- Công thức: Adjusted $R^2 = 1 - \frac{(1-R^2) \times (n-1)}{n-k-1}$
- Trong đó:

- n là số lượng mẫu.
- k là số lượng biến đầu vào.

3. **Root Mean Squared Error (RMSE):**

- Công thức: $\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_{\text{true},i} - y_{\text{pred},i})^2}$

4. **Mean Absolute Percentage Error (MAPE):**

- Công thức: $\text{MAPE} = \frac{1}{n} \sum_{i=1}^n \frac{|y_{\text{true},i} - y_{\text{pred},i}|}{|y_{\text{true},i}|} \times 100$

5. **Mean Percentage Error (MPE):**

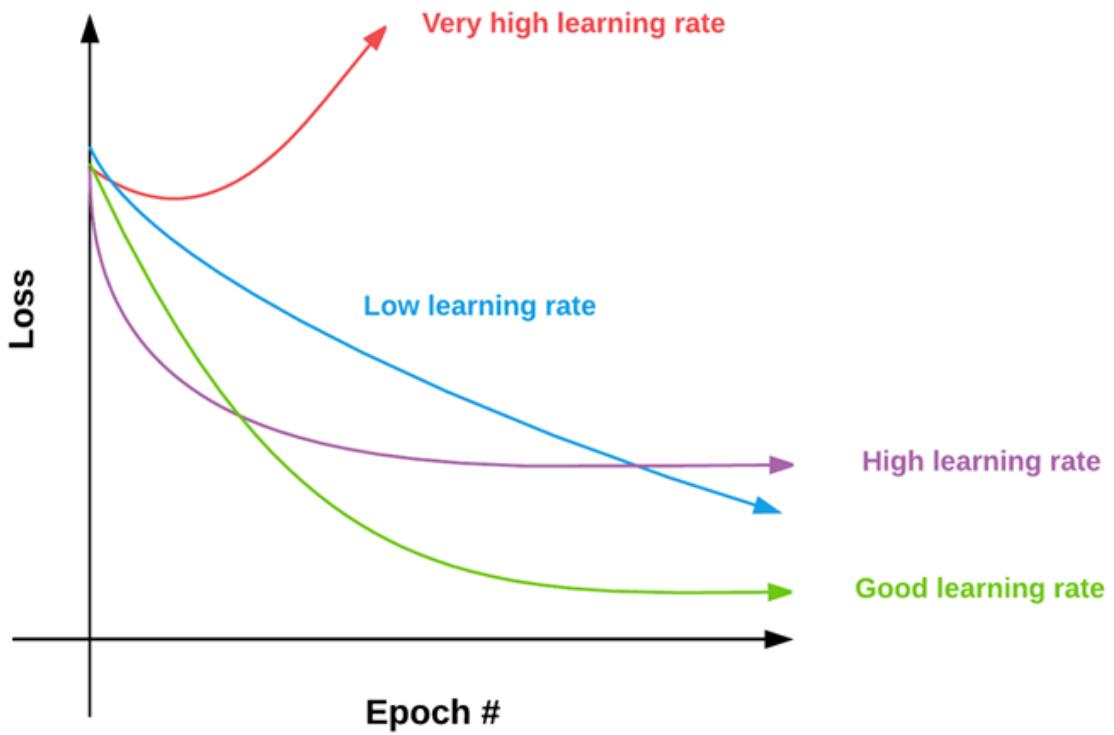
- Công thức: $\text{MPE} = \frac{1}{n} \sum_{i=1}^n \frac{y_{\text{true},i} - y_{\text{pred},i}}{y_{\text{true},i}} \times 100$

Trong đó:

- $y_{\text{true},i}$ là giá trị thực tế của mẫu thứ i .
- $y_{\text{pred},i}$ là giá trị dự đoán của mẫu thứ i .
- n là số lượng mẫu.

1. **Mean Squared Error (MSE)**: Đây là giá trị trung bình của bình phương của sự chênh lệch giữa giá trị dự đoán và giá trị thực tế. MSE đo lường sự lớn của sai số dự đoán và càng thấp thì mô hình càng chính xác. Giá trị MSE càng gần 0 thì mô hình càng tốt. MSE không phải là phần trăm và không có giới hạn trên hoặc dưới.
2. **Mean Absolute Error (MAE)**: Đây là giá trị trung bình của sự chênh lệch tuyệt đối giữa giá trị dự đoán và giá trị thực tế. MAE cũng đo lường sự chính xác của mô hình, nhưng nó tính toán sự chênh lệch tuyệt đối thay vì bình phương như MSE. MAE càng thấp thì mô hình càng chính xác. Giá trị MAE không phải là phần trăm và không có giới hạn trên hoặc dưới.
3. **R squared (R^2)**: Đây là một phép đo thống kê cho biết mức độ biến thiên của mô hình có thể giải thích được bởi các biến đầu vào. Giá trị R^2 nằm trong khoảng từ 0 đến 1 và càng gần 1 thì mô hình càng tốt. $R^2 = 1$ nghĩa là mô hình phù hợp hoàn hảo với dữ liệu.
4. **Adjusted R squared**: R^2 được điều chỉnh để tính toán sự phù hợp của mô hình với số lượng biến đầu vào. Điều chỉnh này thường được sử dụng để tránh việc tăng giá trị R^2 khi thêm các biến đầu vào vào mô hình, ngay cả khi các biến này không đóng góp thực sự vào sự giải thích của mô hình.
5. **Root Mean Squared Error (RMSE)**: Đây là căn bậc hai của MSE, nó cũng đo lường sự chênh lệch giữa giá trị dự đoán và giá trị thực tế như MSE nhưng dưới dạng đơn vị của y. RMSE càng thấp thì mô hình càng chính xác.
6. **Mean Absolute Percentage Error (MAPE)**: Đây là giá trị trung bình của tỷ lệ tuyệt đối giữa sự chênh lệch giữa giá trị dự đoán và giá trị thực tế, tính theo phần trăm. MAPE càng thấp thì mô hình càng chính xác. Giá trị MAPE không phải là phần trăm và không có giới hạn trên hoặc dưới.
7. **Mean Percentage Error (MPE)**: Đây là giá trị trung bình của sự chênh lệch giữa giá trị dự đoán và giá trị thực tế, tính theo phần trăm. MPE là một phép đo về sự chênh trung bình của mô hình và nếu dương thì mô hình thường đoán cao hơn giá trị thực tế, và nếu âm thì ngược lại. Giá trị MPE không phải là phần trăm và không có giới hạn trên hoặc dưới.

Còn "epoch" là một khái niệm trong quá trình huấn luyện mô hình máy học. Mỗi "epoch" tương ứng với một lần duyệt qua toàn bộ dữ liệu huấn luyện một lần. Trong mỗi epoch, thuật toán sẽ cập nhật các trọng số của mô hình dựa trên dữ liệu huấn luyện để giảm thiểu hàm mất mát. Số lượng epoch được chọn phụ thuộc vào kích thước của dữ liệu và độ phức tạp của mô hình. Đôi khi cần thiết phải thử nghiệm với nhiều giá trị của epoch để tìm ra giá trị tốt nhất cho mô hình.



Nhận xét:

- Thuật toán hoạt động rất tốt trong trường hợp không thể tìm giá trị nhỏ nhất bằng đại số tuyến tính.
- Việc quan trọng nhất của thuật toán là tính đạo hàm của hàm số theo từng biến sau đó lặp lại bước 2.

Việc chọn hệ số learning_rate cực kì quan trọng, có 3 trường hợp:

- Nếu learning_rate nhỏ: mỗi lần hàm số giảm rất ít nên cần rất nhiều lần thực hiện bước 2 để hàm số đạt giá trị nhỏ nhất.
- Nếu learning_rate hợp lý: sau một số lần lặp bước 2 vừa phải thì hàm sẽ đạt giá trị đủ nhỏ.
- Nếu learning_rate quá lớn: sẽ gây hiện tượng overshoot (như trong hình 3.8) và không bao giờ đạt được giá trị nhỏ nhất của hàm.



- [Machine Learning cơ bản \(machinelearningcoban.com\)](http://Machine Learning cơ bản (machinelearningcoban.com)) tham khảo

2.4. Nghiệm cho bài toán Linear Regression

Cách phổ biến nhất để tìm nghiệm cho một bài toán tối ưu (chúng ta đã biết từ khi học cấp 3) là giải phương trình đạo hàm (gradient) bằng 0! Tất nhiên đó là khi việc tính đạo hàm và việc giải phương trình đạo hàm bằng 0 không quá phức tạp. Thật may mắn, với các mô hình tuyến tính, hai việc này là khả thi.

Đạo hàm theo \mathbf{w} của hàm mất mát là:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \bar{\mathbf{X}}^T (\bar{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

Các bạn có thể tham khảo bảng đạo hàm theo vector hoặc ma trận của một hàm số trong [mục D.2](#) của tài liệu này. Đến đây tôi xin quay lại câu hỏi ở phần [Sai số dự đoán](#) phía trên về việc tại sao không dùng trị tuyệt đối mà lại dùng bình phương. Câu trả lời là hàm bình phương có đạo hàm tại mọi nơi, trong khi hàm trị tuyệt đối thì không (đạo hàm không xác định tại 0).

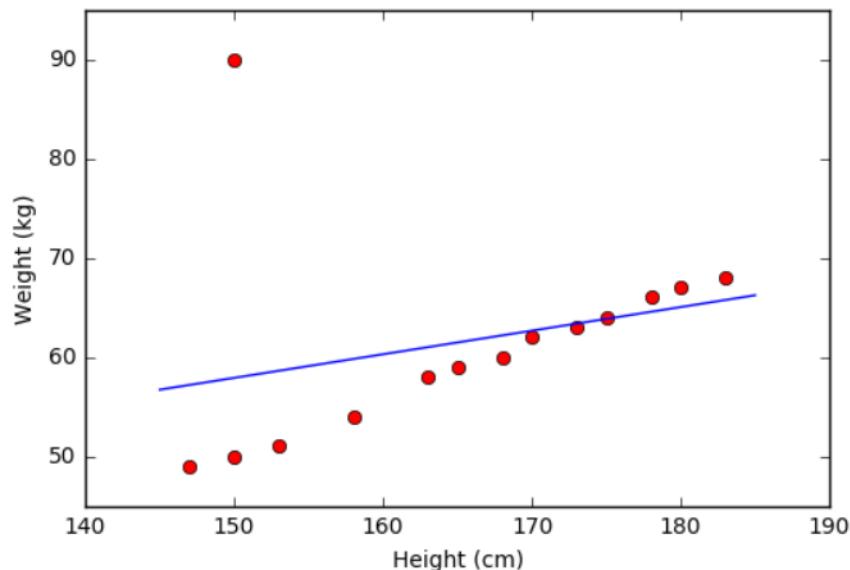
Phương trình đạo hàm bằng 0 tương đương với:

$$\bar{\mathbf{X}}^T \bar{\mathbf{X}}\mathbf{w} = \bar{\mathbf{X}}^T \mathbf{y} \triangleq \mathbf{b} \quad (4)$$

Hạn chế của Linear Regression

4.2. Hạn chế của Linear Regression

Hạn chế đầu tiên của Linear Regression là nó rất **nhạy cảm với nhiễu** (sensitive to noise). Trong ví dụ về mối quan hệ giữa chiều cao và cân nặng bên trên, nếu có chỉ một cặp dữ liệu *nhiễu* (150 cm, 90kg) thì kết quả sẽ sai khác đi rất nhiều. Xem hình dưới đây:



Vì vậy, trước khi thực hiện Linear Regression, các nhiễu (*outlier*) cần phải được loại bỏ. Bước này được gọi là **tiền xử lý** (pre-processing).

Hạn chế thứ hai của Linear Regression là nó **không biểu diễn được các mô hình phức tạp**. Mặc dù trong phần trên, chúng ta thấy rằng phương pháp này có thể được áp dụng nếu quan hệ giữa *outcome* và *input* không nhất thiết phải là tuyến tính, nhưng mối quan hệ này vẫn đơn giản nhiều so với các mô hình thực tế. Hơn nữa, chúng ta sẽ tự hỏi: làm thế nào để xác định được các hàm $x_1^2, \sin(x_2), x_1x_2$ như ở trên?

Code và Giải thích – Bài toán dự đoán giá nhà với một tham số đầu vào

Code thuần thuật toán Linear Regression (Hồi quy tuyến tính)

$$X[:, 1] = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

$$\sum(X[:, 1]) = x_1 + x_2 + \dots + x_n$$

$$\frac{dJ}{dw_0} = \sum(\hat{y} - y)$$

$$\frac{dJ}{dw_1} = \sum(X[:, 1] \otimes (\hat{y} - y))$$

$$\frac{dJ}{dw} = X^T * (\hat{y} - y)$$

Hình 3.15: Biểu diễn đạo hàm dạng ma trận

Gradient descent là thuật toán tìm giá trị nhỏ nhất của hàm số $f(x)$ dựa trên đạo hàm. Thuật toán:

1. Khởi tạo giá trị $x = x_0$ tùy ý
2. $\hat{x} = x - \text{learning_rate} * f'(x)$ (learning_rate là hằng số dương ví dụ $\text{learning_rate} = 0.001$)
3. Tính lại $f(x)$: Nếu $f(x)$ đủ nhỏ thì dừng lại, ngược lại tiếp tục bước 2

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('data_linear.csv').values # Đọc dữ liệu từ tệp CSV và lưu vào biến data
print(pd.read_csv('data_linear.csv').head())

# Số lượng mẫu
N = data.shape[0]
x = data[:, 0].reshape(-1, 1) # Lấy dữ liệu của biến đầu vào và đầu ra từ data
y = data[:, 1].reshape(-1, 1)

plt.scatter(x, y) # Vẽ biểu đồ scatter plot
plt.xlabel('mét vuông') # Đặt tên trục x
plt.ylabel('giá') # Đặt tên trục y

x = np.hstack((np.ones((N, 1)), x)) # Thêm cột 1 vào dữ liệu biến đầu vào x để tính toán hệ số điều chỉnh
# (intercept)
w = np.array([0., 1.]).reshape(-1, 1) # Khởi tạo giá trị hệ số hồi quy ban đầu
numOfIteration = 100 # Số lượng vòng lặp
cost = np.zeros((numOfIteration, 1)) # Mảng lưu trữ giá trị hàm mất mát trong quá trình huấn luyện
learning_rate = 0.000001 # Tốc độ học (learning rate)

# Huấn luyện mô hình
for i in range(1, numOfIteration): # THUẬT TOÁN Gradient descent và xem hình 3.15
    r = np.dot(x, w) - y # Tính residual # y hay trừ y (độ chênh lệch)

```

```

w[0] -= learning_rate*np.sum(r) # Cập nhật hệ số w0 (hệ số điều chỉnh) .
# Để tìm f(x) min ta phải tìm được x , tìm bằng cách
# Ta có x = x - learning_rate*f(x) . sau đó tính lại x
# f(w[0]) min thì tìm w[0] . Ta có w[0] = w[0] - learning_rate*f(w[0]) . Mặt khác ta cũng có : f(w[0]) =
sum(x.w - y)

# Tương tự với dòng dưới ,từ công thức ta cũng suy ra được
w[1] -= learning_rate*np.sum(np.multiply(r, x[:,1].reshape(-1,1))) # Cập nhật hệ số w1 (hệ số của biến đầu
vào)
cost[i] = 0.5*np.sum(r*r) # này đang dùng công thức MSE # Tính toán giá trị hàm mất mát
# Sau khi w[0] và w[1] được cập nhật lại thì w chứa w[0] và w[1] cũng được update lại tính ra được độ
chênh lệch mới : r = np.dot(x, w) - y
# Từ giá trị mới này tính ra được cost[i] mới (hàm mất mát) . Giá trị mất mát càng nhỏ càng tốt .
print(cost[i]) # Quang sát giá trị mất mát , càng ngày càng giảm dần cho đến khi gần như đạt cực tiểu có thể
giảm tiếp nhung giảm với số lượng rất nhỏ thì dừng

predict = np.dot(x, w) # Dự đoán giá trị của y dựa trên x . Sau khi tính toán được w[0] và w[1] của w thì tính ra
giá trị dự đoán 'y hat' là y predict
# từ ma trận x sau khi qua 2 hệ số w0 và w1 thì ra được ma trận y predict (y hat)

# Tọa độ điểm đầu (x[0][1] ; predict[0])
# Tọa độ điểm cuối (x[N-1][1] ; predict[N-1])
plt.plot((x[0][1], x[N-1][1]),(predict[0], predict[N-1]), 'r') # Vẽ đường hồi quy . Nối từ điểm đầu cho đến điểm
cuối dự đoán được .
plt.show()

x1 = 50 # Dự đoán giá nhà cho một diện tích là 50 mét vuông
y1 = w[0] + w[1] * 50
print('Giá nhà cho 50m^2 là : ', y1)

```

```
File Edit Selection View Go Run Terminal Help Untitled (Workspace)
Lab1_LinearRegression.ipynb L1.ipynb M
Al - Data Science > Documents > DL_Tutorial > L1 > L1.ipynb > Code thuật toán Linear Regression (Hồi quy tuyến tính) > import numpy as np
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
Python 3.10.6
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('data_linear.csv').values # Đọc dữ liệu từ tệp CSV và lưu vào biến data
print(pd.read_csv('data_linear.csv').head())

# Số lượng mẫu
N = data.shape[0]
x = data[:, 0].reshape(-1, 1) # Lấy dữ liệu của biến đầu vào và đầu ra từ data
y = data[:, 1].reshape(-1, 1)

plt.scatter(x, y) # Vẽ biểu đồ scatter plot
plt.xlabel('mét vuông') # Đặt tên trục x
plt.ylabel('giá') # Đặt tên trục y

x = np.hstack((np.ones((N, 1)), x)) # Thêm cột 1 vào dữ liệu biến đầu vào x để tính toán hệ số điều chỉnh (intercept)
w = np.array([0., 1.]).reshape(-1, 1) # Khởi tạo giá trị hệ số hồi quy ban đầu
numOfIteration = 100 # Số lượng vòng lặp
cost = np.zeros((numOfIteration, 1)) # Mảng lưu trữ giá trị hàm mất mát trong quá trình huấn luyện
learning_rate = 0.000001 # Tốc độ học (learning rate)

# Huấn luyện mô hình
for i in range(1, numOfIteration): # THUẬT TOÁN Gradient descent và xem hình 3.15
    r = np.dot(x, w) - y # Tính residual y hay trừ y (độ chênh lệch)
    w[0] -= learning_rate*np.sum(r) # Cập nhật hệ số w0 (hệ số điều chỉnh).
    # Bé tim f(x) min ta phai tim duoc x , tim bang cach
    # Ta co x = x - learning_rate*f'(x) . sau do tinh lai x |
    # # f(w[0]) min thi tim w[0] . Ta co w[0] = w[0] - learning_rate*f'(w[0]) . Mat khac ta cung co : f'(w[0]) = sum(x.w - y)

    # Tương tự với dòng dưới , từ công thức ta cũng suy ra được
    w[1] -= learning_rate*np.sum(r*x[:, 1].reshape(-1, 1))) # Cập nhật hệ số w1 (hệ số của biến đầu vào)
    cost[i] = 0.5*np.sum(r**2) # này đang dùng công thức MSE # Tính toán giá trị hàm mất mát
    # Sau khi w[0] và w[1] được cập nhật lại thì w chính w[0] và w[1] cũng được update lại tinh ra được độ chênh lệch mới : r = np.dot(x, w) - y
    # Từ giá trị mới này tính ra được cost[i] mới (hàm mất mát) . Giá trị mất mát càng nhỏ càng tốt .
    print(cost[i]) # Quan sát giá trị mất mát , càng ngày càng giảm dần cho đến khi gần như đạt cực tiểu có thể giảm tiếp nhưng giảm với số lượng rất nhỏ thi dừng

predict = np.dot(x, w) # Dự đoán giá trị của y dựa trên x . Sau khi tính toán được w[0] và w[1] của w thi tính ra giá trị dự đoán 'y hat' là y predict
# từ ma trận x sau khi qua 2 hệ số w0 và w1 thi ra được ma trận y predict (y hat)

# Tạo độ diểm đầu (x[0][1] ; predict[0])
# Tạo độ diểm cuối (x[N-1][1] ; predict[N-1])
plt.plot((x[0][1], x[N-1][1]), (predict[0], predict[N-1]), 'r') # Vẽ đường hồi quy . Nối từ diểm đầu cho đến diểm cuối dự đoán được .
plt.show()

x1 = 50 # Dự đoán giá nhà cho một diện tích là 50 mét vuông
y1 = w[0] + w[1] * 50
print('Giá nhà cho 50m^2 là : ', y1)
```

```
File Edit Selection View Go Run Terminal Help Untitled (Workspace)
Lab1_LinearRegression.ipynb L1.ipynb M
Al - Data Science > Documents > DL_Tutorial > L1 > L1.ipynb > Code thuật toán Linear Regression (Hồi quy tuyến tính) > import numpy as np
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ...
Python 3.10.6
x = np.hstack((np.ones((N, 1)), x)) # Thêm cột 1 vào dữ liệu biến đầu vào x để tính toán hệ số điều chỉnh (intercept)
w = np.array([0., 1.]).reshape(-1, 1) # Khởi tạo giá trị hệ số hồi quy ban đầu
numOfIteration = 100 # Số lượng vòng lặp
cost = np.zeros((numOfIteration, 1)) # Mảng lưu trữ giá trị hàm mất mát trong quá trình huấn luyện
learning_rate = 0.000001 # Tốc độ học (learning rate)

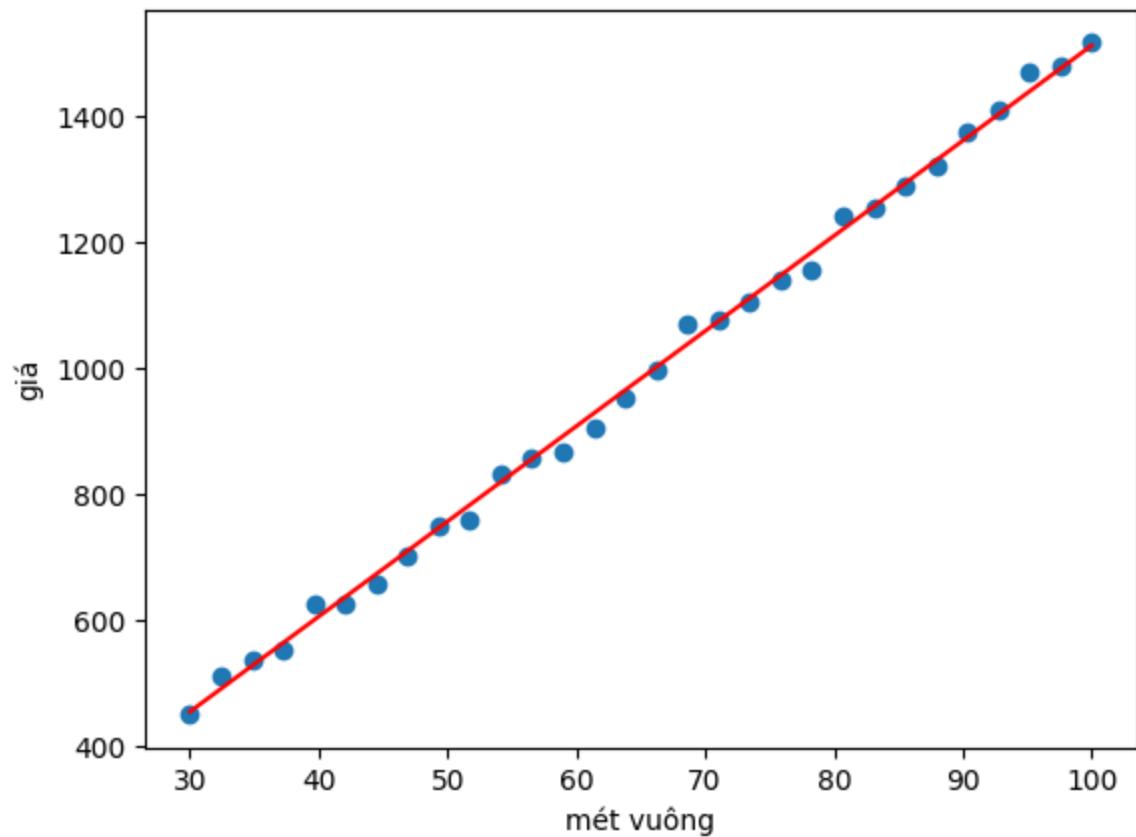
# Huấn luyện mô hình
for i in range(1, numOfIteration): # THUẬT TOÁN Gradient descent và xem hình 3.15
    r = np.dot(x, w) - y # Tính residual y hay trừ y (độ chênh lệch)
    w[0] -= learning_rate*np.sum(r) # Cập nhật hệ số w0 (hệ số điều chỉnh).
    # Bé tim f(x) min ta phai tim duoc x , tim bang cach
    # Ta co x = x - learning_rate*f'(x) . sau do tinh lai x |
    # # f(w[0]) min thi tim w[0] . Ta co w[0] = w[0] - learning_rate*f'(w[0]) . Mat khac ta cung co : f'(w[0]) = sum(x.w - y)

    # Tương tự với dòng dưới , từ công thức ta cũng suy ra được
    w[1] -= learning_rate*np.sum(r*x[:, 1].reshape(-1, 1))) # Cập nhật hệ số w1 (hệ số của biến đầu vào)
    cost[i] = 0.5*np.sum(r**2) # này đang dùng công thức MSE # Tính toán giá trị hàm mất mát
    # Sau khi w[0] và w[1] được cập nhật lại thi w chính w[0] và w[1] cũng được update lại tinh ra được độ chênh lệch mới : r = np.dot(x, w) - y
    # Từ giá trị mới này tính ra được cost[i] mới (hàm mất mát) . Giá trị mất mát càng nhỏ càng tốt .
    print(cost[i]) # Quan sát giá trị mất mát , càng ngày càng giảm dần cho đến khi gần như đạt cực tiểu có thể giảm tiếp nhưng giảm với số lượng rất nhỏ thi dừng

predict = np.dot(x, w) # Dự đoán giá trị của y dựa trên x . Sau khi tính toán được w[0] và w[1] của w thi tính ra giá trị dự đoán 'y hat' là y predict
# từ ma trận x sau khi qua 2 hệ số w0 và w1 thi ra được ma trận y predict (y hat)

# Tạo độ diểm đầu (x[0][1] ; predict[0])
# Tạo độ diểm cuối (x[N-1][1] ; predict[N-1])
plt.plot((x[0][1], x[N-1][1]), (predict[0], predict[N-1]), 'r') # Vẽ đường hồi quy . Nối từ diểm đầu cho đến diểm cuối dự đoán được .
plt.show()

x1 = 50 # Dự đoán giá nhà cho một diện tích là 50 mét vuông
y1 = w[0] + w[1] * 50
print('Giá nhà cho 50m^2 là : ', y1)
```



[HAY] Tóm tắt Linear Regression

Mô hình chung cho bài toán trong Deep Learning.

1. Visualize dữ liệu

2. Thiết lập model (Linear regression là hàm đường thẳng : $\hat{y}_i = w_0 + w_1x_1^{(i)} + \dots + w_mx_m^{(i)}$)

Logistic regression là hàm sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$

$\hat{y}_i = \sigma(w_0 + w_1x_1^{(i)} + \dots + w_mx_m^{(i)}) = \frac{1}{1+e^{-(w_0+w_1x_1^{(i)}+\dots+w_mx_m^{(i)})}}$ (cho \hat{y}_i về khoảng [0,1])

3. Thiết lập loss function (MSE) (Linear regression là hàm MSE = $\frac{\sum_{i=1}^n(y - \hat{y})^2}{n}$)

Logistic regression là hàm **binary crossentropy** : $L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$

4. Tìm tham số bằng việc tối ưu loss function (Gradient descent) Linear là $\frac{dJ}{dw} = X^T * (\hat{y} - y)$

Logistic regression có hàm gradient descent là $\frac{dJ}{dw} = \frac{1}{N} * X^T * (\hat{y} - y)$

5. Dự đoán dữ liệu mới bằng model vừa tìm được

Giải thích :

- Ta có $f(x)$, để tìm cực trị của nó ta đi đạo hàm và cho $f'(x) = 0$ để tìm ra x , tuy nhiên nhiều hàm $f(x)$ phức tạp ta chỉ đạo hàm ra được $f'(x)$ còn việc tìm ra được x là rất khó. Chính vì thế nên ta dùng Gradient descent để đi tìm giá trị gần giống với nghiệm x nhất.

- Tóm lại :

Model + Gradient descent : Đi tìm w mới sau mỗi vòng lặp

Loss function : MSE hay binary crossentropy để đánh giá w mới đã tối ưu hay chưa thông qua việc xem giá trị mất mát mà w mới này mang lại sau mỗi vòng lặp từ đó điều chỉnh hệ số learning rate và epochs cho hợp lý để tìm ra được w chính xác mà không bị các tình trạng sóm, trễ hay không chính xác.

- Ở trong bài này điều đặc biệt là ta có thể tìm ra được nghiệm của hàm mất mát sau khi đạo hàm. Nhưng ở Logistic thì ta không thể tìm ra được nghiệm. Mặc khác việc giải toán và tìm ra nghiệm này cũng không quan trọng vì như đã nói ở trên.

Thiết lập Model

$$\hat{y}_i = w_0 + w_1x_1^{(i)} + w_2x_2^{(i)} + \dots + w_mx_m^{(i)}$$

Công thức Gradient Descent

$$\frac{dJ}{dw} = X^T * (\hat{y} - y)$$

MSE Loss function

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

Giải bài toán bằng đại số tuyến tính

$w = [w_0, w_1, w_2, \dots, w_m]^T$ là vector hệ số cần phải tối ưu, w_0 thường được gọi là bias.

❖ Giải thích : w là ma trận $(m+1)$ hàng và 1 cột để lưu các hệ số cần tìm

$x^{(i)} = [1, x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}]$ là hàng dữ liệu thứ i trong bộ n số lượng dữ liệu quan sát được, mỗi dữ liệu có m giá trị.

❖ Giải thích : X là ma trận đầu vào có n mẫu dữ liệu và m biến đầu vào (feature) thì ta sẽ tạo ra ma trận n hàng $(m+1)$ cột (do thêm cột 1 ở đầu cho tham số bias w_0)

❖ Ta sẽ có ma trận $X = \begin{bmatrix} 1 & x_1^1 & \dots & x_m^1 \\ 1 & x_1^2 & \dots & x_m^2 \\ \dots & \dots & \dots & \dots \\ 1 & x_1^n & \dots & x_m^n \end{bmatrix}$ (ma trận n hàng và $(m+1)$ cột) (i chạy từ 1 đến n)

Ta có : Giá trị dự đoán y hat của thàng dữ liệu thứ i là $\hat{y}^{(i)} = x^{(i)}.w$

$X \in R^{n*(m+1)}$, $w \in R^{(m+1)*1}$, $y \in R^{n*1}$

$$Ta\ có : X = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \dots \\ x^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & \dots & x_m^1 \\ 1 & x_1^2 & \dots & x_m^2 \\ \dots & \dots & \dots & \dots \\ 1 & x_1^n & \dots & x_m^n \end{bmatrix}, w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix}, \hat{y} = X.w = \begin{bmatrix} w_0 + w_1x_1^1 + w_2x_2^1 + \dots + w_mx_m^1 \\ w_0 + w_1x_1^2 + w_2x_2^2 + \dots + w_mx_m^2 \\ \dots \\ w_0 + w_1x_1^n + w_2x_2^n + \dots + w_mx_m^n \end{bmatrix}$$

$$y = \begin{bmatrix} y^1 \\ y^2 \\ \dots \\ y^n \end{bmatrix}, y - \hat{y} = \begin{bmatrix} y^1 - (w_0 + w_1x_1^1 + w_2x_2^1 + \dots + w_mx_m^1) \\ y^2 - (w_0 + w_1x_1^2 + w_2x_2^2 + \dots + w_mx_m^2) \\ \dots \\ y^n - (w_0 + w_1x_1^n + w_2x_2^n + \dots + w_mx_m^n) \end{bmatrix}$$

❖ Giải thích : X là ma trận $n \times (m+1)$, w là ma trận $(m+1) \times 1$, $\hat{y} = X.w$ sẽ là ma trận $n \times 1$

Theo như hàm mất mát MSE Loss function :

$$MSE = \frac{\sum_{i=1}^n (y - \hat{y})^2}{n} \text{ hoặc } MSE = \frac{1}{2} * \frac{\sum_{i=1}^n (y - \hat{y})^2}{n}$$

$$Ta\ có : L = \frac{1}{2} * \frac{1}{n} * \sum_{i=1}^n (y^i - \hat{y}^i)^2 = \frac{1}{2} * \frac{1}{n} * \sum_{i=1}^n (y^i - x^{(i)}.w)^2$$

❖ Giải thích : MES cho ma trận X là mất mát của n hàng dữ liệu

Định nghĩa Euclidean norm : $\|z\|_2 = (z_1^2 + z_2^2 + \dots + z_n^2)^{\frac{1}{2}} \Rightarrow \|z\|_2^2 = (z_1^2 + z_2^2 + \dots + z_n^2)$

❖ Giải thích : Trong công thức Euclidean norm thì $\|z\|_2^2$, số 2 ở dưới là chuẩn 2 tức là căn bậc 2 và số 2 ở trên là mũ 2 và mũ hai của căn bậc 2 tức là hết căn có nghĩa là đơn giản ta chỉ bình phương của căn thôi .

Mặc khác ta có :

$$L = \frac{1}{2} * \frac{1}{n} * \sum_{i=1}^n (y^i - \hat{y}^i)^2 = \frac{1}{2} * \frac{1}{n} * [(y^1 - \hat{y}^1)^2 + (y^2 - \hat{y}^2)^2 + \dots + (y^n - \hat{y}^n)^2]$$

$$\text{Đặt } (y^i - \hat{y}^i)^2 = z_i^2 \text{ Ta được } L = \frac{1}{2} * \frac{1}{n} * (z_1^2 + z_2^2 + \dots + z_n^2) = \frac{1}{2} * \frac{1}{n} * \|z\|_2^2$$

Viết lại

$$L = \frac{1}{2} * \frac{1}{n} * \sum_{i=1}^n (y^i - \hat{y}^i)^2 = \frac{1}{2} * \frac{1}{n} * [(y^1 - \hat{y}^1)^2 + (y^2 - \hat{y}^2)^2 + \dots + (y^n - \hat{y}^n)^2]$$

$$= \frac{1}{2} * \frac{1}{n} * \|y - \hat{y}\|_2^2 = \frac{1}{2} * \frac{1}{n} * \|y - Xw\|_2^2 = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y})$$

❖ Giải thích : $\|y - \hat{y}\|_2^2 = (y^1 - \hat{y}^1)^2 + (y^2 - \hat{y}^2)^2 + \dots + (y^n - \hat{y}^n)^2$

❖ Ta thấy : $y - \hat{y} = \begin{bmatrix} y^1 - \hat{y}^1 \\ y^2 - \hat{y}^2 \\ \dots \\ y^n - \hat{y}^n \end{bmatrix} \Rightarrow (y - \hat{y})^T = [y^1 - \hat{y}^1 \quad y^2 - \hat{y}^2 \quad \dots \quad y^n - \hat{y}^n]$

❖ Mà $(y - \hat{y})^T * (y - \hat{y}) = [y^1 - \hat{y}^1 \quad y^2 - \hat{y}^2 \quad \dots \quad y^n - \hat{y}^n] * \begin{bmatrix} y^1 - \hat{y}^1 \\ y^2 - \hat{y}^2 \\ \dots \\ y^n - \hat{y}^n \end{bmatrix}$
 $= (y^1 - \hat{y}^1) * (y^1 - \hat{y}^1) + (y^2 - \hat{y}^2) * (y^2 - \hat{y}^2) + \dots + (y^n - \hat{y}^n) * (y^n - \hat{y}^n)$
 $= (y^1 - \hat{y}^1)^2 + (y^2 - \hat{y}^2)^2 + \dots + (y^n - \hat{y}^n)^2$ Nhân 2 ma trận $I \times n$ và $n \times I$

Từ đó suy ra :

$$L = \frac{1}{2} * \frac{1}{n} * \|y - \hat{y}\|_2^2 = \frac{1}{2} * \frac{1}{n} * \|y - Xw\|_2^2 = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y})$$

Bây giờ tìm đi tìm w sao cho L (giá trị mất mát) là nhỏ nhất , như thường lệ ta sẽ khảo sát , nghĩa là ta sẽ đạo hàm L theo w và tìm nghiệm w để cho $L' = 0$

Lưu ý :

+ X là một ma trận bất kì thì ta có $A = X^T * X$ thì A sẽ luôn là một ma trận đối xứng nghĩa là $A^T = A$

+ $(A * B)^T = B^T * A^T$ (công thức liên quan đến ma trận chuyển vị trong đại số tuyến tính)

+ Quy tắc đạo hàm tích $[f(x) \cdot g(x)]' = [f(x)]' \cdot g(x) + f(x) \cdot [g(x)]'$

Giải bài toán $L' = 0$

$$L = \frac{1}{2} * \frac{1}{n} * (y - \hat{y})^T * (y - \hat{y}) . Ta có thể cho L = (y - \hat{y})^T * (y - \hat{y}) vì \frac{1}{2} * \frac{1}{n} là hằng$$

$$L = (y - \hat{y})^T * (y - \hat{y}) = (y - Xw)^T * (y - Xw)$$

$$= (y^T - w^T X^T) * (y - Xw) = y^T y - y^T Xw - w^T X^T y + w^T X^T Xw$$

Đạo hàm :

$$\begin{aligned} L'_w &= -y^T X - [(Xw)^T y + w^T X^T Xw]'_w = -y^T X - [(Xw)^T (y^T)^T + w^T X^T Xw]'_w \\ &= -y^T X - [(y^T Xw)^T + w^T X^T Xw]'_w = -y^T X - [(y^T Xw)^T + w^T X^T Xw]'_w \\ &= -y^T X - [(y^T Xw)^T]'_w + [w^T X^T Xw]'_w \\ &= -y^T X - [(y^T X)w]'_w + [w^T X^T Xw]'_w \\ &= -y^T X - y^T X + [w^T X^T Xw]'_w \\ &= -2y^T X + [w^T X^T Xw]'_w \end{aligned}$$

❖ Cần đi tìm $[w^T X^T Xw]'_w$

Tìm $[w^T X^T Xw]'_w$:

$$+ x^{(i)}w = [1 \ x_1^i \ \dots \ x_m^i] * \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix} = w_0 + w_1x_1^i + w_2x_2^i + \dots + w_mx_m^i$$

$$\Rightarrow \frac{d(x^{(i)}w)}{dw} = \left[\frac{d(x^{(i)}w)}{dw_0} \ \ \frac{d(x^{(i)}w)}{dw_1} \ \ \dots \ \ \frac{d(x^{(i)}w)}{dw_m} \right] = [1 \ x_1^i \ \dots \ x_m^i] = x^{(i)}$$

$$\Rightarrow \frac{d(Xw)}{dw} = \begin{bmatrix} \frac{d(x^{(1)}w)}{dw_0} & \frac{d(x^{(1)}w)}{dw_1} & \dots & \frac{d(x^{(1)}w)}{dw_m} \\ \frac{d(x^{(2)}w)}{dw_0} & \frac{d(x^{(2)}w)}{dw_1} & \dots & \frac{d(x^{(2)}w)}{dw_m} \\ \dots & \dots & \dots & \dots \\ \frac{d(x^{(n)}w)}{dw_0} & \frac{d(x^{(n)}w)}{dw_1} & \dots & \frac{d(x^{(n)}w)}{dw_m} \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & \dots & x_m^1 \\ 1 & x_1^2 & \dots & x_m^2 \\ \dots & \dots & \dots & \dots \\ 1 & x_1^n & \dots & x_m^n \end{bmatrix} = X$$

$$+ wx^{(i)} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \\ w_m \end{bmatrix} * [1 \ x_1^i \ \dots \ x_m^i] = \color{red}{w_0 + w_1x_1^i + w_2x_2^i + \dots + w_mx_m^i}$$

$$= \begin{bmatrix} w_0 & w_0x_1^i & \dots & w_0x_m^i \\ w_1 & w_1x_1^i & \dots & w_1x_m^i \\ \dots & \dots & \dots & \dots \\ w_m & w_mx_1^i & \dots & w_mx_m^i \end{bmatrix}$$

$$\Rightarrow \frac{d(wx^{(i)})}{dw} = \begin{bmatrix} \frac{d(wx^{(i)})}{dw_0} \\ \frac{d(wx^{(i)})}{dw_1} \\ \dots \\ \frac{d(wx^{(i)})}{dw_m} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^i \\ \dots \\ x_m^i \end{bmatrix} = (x^{(i)})^T$$

$$\Rightarrow \frac{d(Xw)}{dw} = \begin{bmatrix} \frac{d(wx^{(1)})}{dw_0} & \frac{d(wx^{(2)})}{dw_0} & \dots & \frac{d(wx^{(n)})}{dw_0} \\ \frac{d(wx^{(1)})}{dw_1} & \frac{d(wx^{(2)})}{dw_1} & \dots & \frac{d(wx^{(n)})}{dw_1} \\ \dots & \dots & \dots & \dots \\ \frac{d(wx^{(1)})}{dw_m} & \frac{d(wx^{(2)})}{dw_m} & \dots & \frac{d(wx^{(n)})}{dw_m} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^1 & x_1^2 & \dots & x_1^n \\ \dots & \dots & \dots & \dots \\ x_m^1 & x_m^2 & \dots & x_m^n \end{bmatrix} = X^T$$

$$+ \frac{d(w^T X w)}{dw} = w^T X + \frac{d((Xw)^T w)}{dw} = w^T X + w^T X^T = w^T (X + X^T) = 2w^T X \quad (\text{Nếu } X \text{ đối xứng thì } X^T = X)$$

Quay lại bài toán $L'_w = 0$

Giải thích : Đặt $X^T X = H$. Mà H là ma trận đối xứng.

Áp dụng kết quả đã được chứng minh ở trên $\frac{d(w^T X w)}{dw} = 2w^T X \leftrightarrow \frac{d(w^T H w)}{dw} = 2w^T H = 2w^T X^T X$

Ta có :

$$L'_w = -2y^T X + [w^T X^T X w]'_w = -2y^T X + \frac{d(w^T X^T X w)}{dw} = -2y^T X + 2w^T X^T X = 0$$

- ❖ $2w^T X^T X = 2y^T X$
- ❖ $w^T X^T X = y^T X$
- ❖ $(Xw)^T X = (X^T y)^T$
- ❖ $[X^T (Xw)]^T = (X^T y)^T$
- ❖ $X^T (Xw) = X^T y$
- ❖ $(X^T X)w = X^T y$
- ❖ $w = (X^T X)^{-1} X^T y$

Kết luận : Nếu $X^T X$ khả nghịch (có ma trận nghịch đảo), thì L có nghiệm duy nhất : $w = (X^T X)^{-1} X^T y$

Nếu không khả nghịch, ta có thể sử dụng khái niệm giả nghịch đảo.

Như vậy ta đã có hàm Gradient descent $\frac{dJ}{dw} = X^T * (\hat{y} - y)$ và nghiệm $w = (X^T X)^{-1} X^T y$

- ❖ Sử dụng công thức $w = w - learning_rate * gradient$ sẽ thu được w tối ưu sau **epochs** lần lặp
- ❖ Hoặc có thể cho nó dừng sớm nếu w_{old} và w_{new} chênh lệch không quá **epsilon**.

Code Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

data = pd.read_csv("USA_Housing.csv")

def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_filtered = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_filtered

def preprocess_data(data):
    data = data.dropna()
    data = data.drop(["Address"], axis=1)

    cols_to_check = ['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population', 'Price']

    for col in cols_to_check:
        data = remove_outliers(data, col)
```

```

for col in cols_to_check:
    data[col] = (data[col] - data[col].min()) / (data[col].max() - data[col].min())

return data

data = preprocess_data(data)
data.head()

def matrix_transformation(x, y):
    x = np.concatenate((np.ones((x.shape[0], 1)), x), axis=1)
    y = np.array(y).reshape(-1, 1)
    return x, y

def linear_regression(X, y, lr=0.05, epsilon=1e-10, epochs=100000):
    ep = 0
    N = X.shape[0]
    w = np.zeros((X.shape[1], 1))
    while ep < epochs:
        gradient = np.dot(X.T, (np.dot(X, w) - y)) / N

        w = w - lr * gradient

        if np.linalg.norm(gradient) < epsilon:
            break
        ep += 1

    return w

def predict(X, w):
    predictions = X.dot(w)

    return predictions

X = data[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms', 'Area Population']]
y = data['Price']

np.random.seed(42)
random_indices = np.random.permutation(len(X))
train_size = int(0.7 * len(X))

X_train = X.iloc[random_indices[:train_size]]
y_train = y.iloc[random_indices[:train_size]].values

```

```

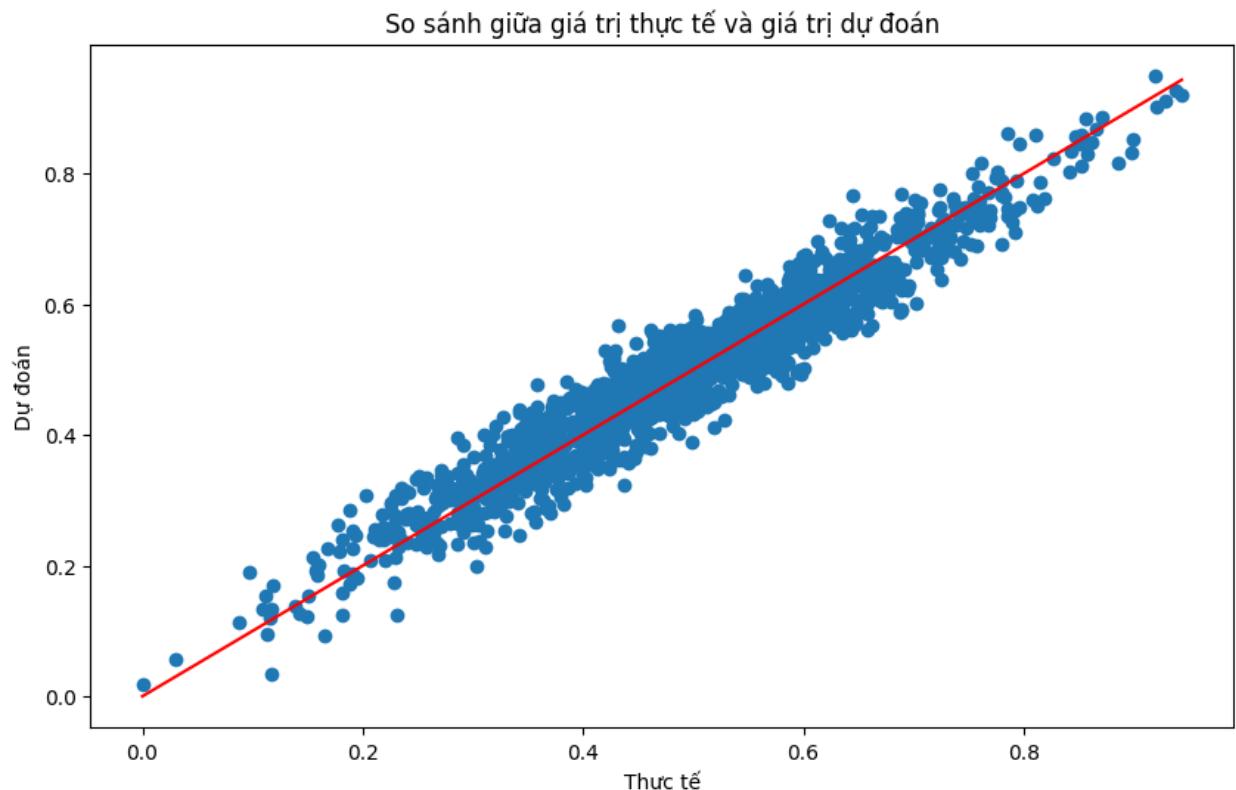
X_test = X.iloc[random_indices[train_size:]]
y_test = y.iloc[random_indices[train_size:]].values

X_train, y_train = matrix_transformation(X_train, y_train)
X_test, y_test = matrix_transformation(X_test, y_test)

w = linear_regression(X_train, y_train, lr=0.05, epsilon=1e-10, epochs=100000)
predictions = predict(X_test, w)

plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red')
plt.xlabel('Thực tế')
plt.ylabel('Dự đoán')
plt.title('So sánh giữa giá trị thực tế và giá trị dự đoán')
plt.show()

```



- Tiền xử lý dữ liệu
 - Chuẩn hóa các cột có giá trị lớn về khoảng 0 -> 1 . Vì có quá nhiều số lớn nhau lại với nhau khi tính toán sẽ dẫn đến NAN
 - Số quá lớn nhưng learning rate chỉ có bước nhảy là 0.05 => quá nhỏ không ảnh hưởng đến w (không phù hợp)
 - Luôn cần phải có tiền xử lý dữ liệu trước khi train (Xử lý dữ liệu null , trống , ngoại lệ . Chuẩn hóa dữ liệu ...)

Code Dự đoán giá nhà bằng Linear Regression

```

File Edit Selection View Go Run Terminal Help ⏎ ⏎ Untitled (Workspace)
Lab1_LinearRegression.ipynb M Lab1.ipynb M
AI - Data Science > Learning > Chapter2_Machine learning_basic > Lab1_LinearRegression > Lab1_LinearRegression.ipynb > LinearRegression Pure (thuần) > import pandas as pd
+ Code + Markdown | Run All Restart Clear All Outputs Variables Outline ...
Python 3.10.6
D return w
if __name__ == "__main__":
    X = []
    # code you have previously used to load data
    iowa_file_path = './USA_Housing.csv'
    data = pd.read_csv(iowa_file_path)
    df = pd.DataFrame(data)
    # xử lý dữ liệu
    data = df.dropna()
    data = data.drop(['Address'], axis=1)
    data = data.drop(data[data['Avg. Area Income'] == 0].index)
    data = data.drop(data[data['Avg. Area House Age'] == 0].index)
    data = data.drop(data[data['Avg. Area Number of Rooms'] == 0].index)
    data = data.drop(data[data['Avg. Area Number of Bedrooms'] == 0].index)
    data = data.drop(data[data['Area Population'] == 0].index)
    data = data.drop(data[data['Price'] == 0].index)

    # min và max hóa dữ liệu
    data['Avg. Area Income'] = (data['Avg. Area Income'] - data['Avg. Area Income'].min()) / (data['Avg. Area Income'].max() - data['Avg. Area Income'].min())
    data['Avg. Area House Age'] = (data['Avg. Area House Age'] - data['Avg. Area House Age'].min()) / (data['Avg. Area House Age'].max() - data['Avg. Area House Age'].min())
    data['Avg. Area Number of Rooms'] = (data['Avg. Area Number of Rooms'] - data['Avg. Area Number of Rooms'].min()) / (data['Avg. Area Number of Rooms'].max() - data['Avg. Area Number of Rooms'].min())
    data['Avg. Area Number of Bedrooms'] = (data['Avg. Area Number of Bedrooms'] - data['Avg. Area Number of Bedrooms'].min()) / (data['Avg. Area Number of Bedrooms'].max() - data['Avg. Area Number of Bedrooms'].min())
    data['Area Population'] = (data['Area Population'] - data['Area Population'].min()) / (data['Area Population'].max() - data['Area Population'].min())
    data['Price'] = (data['Price'] - data['Price'].min()) / (data['Price'].max() - data['Price'].min())

    X1 = np.array(data['Avg. Area Income']).reshape(-1, 1)
    X2 = np.array(data['Avg. Area House Age']).reshape(-1, 1)
    X3 = np.array(data['Avg. Area Number of Rooms']).reshape(-1, 1)
    X4 = np.array(data['Avg. Area Number of Bedrooms']).reshape(-1, 1)
    X5 = np.array(data['Area Population']).reshape(-1, 1)
    y = np.array(data['Price']).reshape(-1, 1)
    X = np.concatenate((X1, X2, X3, X4, X5), axis=1)
    print("len(X) = ", len(X))
    X_tanh = X * 0.0001 - 1
Spaces: 4 Cell 2 of 28 7:32 AM 3/5/2024

```

- ❖ Chính vì Linear Regression dễ bị ảnh hưởng bởi các giá trị ngoại lệ nên việc tiền xử lí dữ liệu hết sức quan trọng .

C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter2_Machine_learning_basic\Lab1_LinearRegression\Lab1_LinearRegression.ipynb

4. Logistic regression

Mô hình chung cho bài toán trong Deep Learning.

1. Visualize dữ liệu

2. Thiết lập model (Linear regression là hàm đường thẳng : $\hat{y}_i = w_0 + w_1x_1^{(i)} + \dots + w_mx_m^{(i)}$)

Logistic regression là hàm sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$

$\hat{y}_i = \sigma(w_0 + w_1x_1^{(i)} + \dots + w_mx_m^{(i)}) = \frac{1}{1+e^{-(w_0+w_1x_1^{(i)}+\dots+w_mx_m^{(i)})}}$ (cho \hat{y}_i về khoảng [0,1])

3. Thiết lập loss function (MSE) (Linear regression là hàm MSE = $\frac{\sum_{i=1}^n(y_i - \hat{y}_i)^2}{n}$)

Logistic regression là hàm **binary crossentropy** : $L = -(y_i * \log(\hat{y}_i)) + (1 - y_i) * \log(1 - \hat{y}_i)$

4. Tìm tham số bằng việc tối ưu loss function (Gradient descent) Linear là $\frac{dL}{dw} = X^T * (\hat{y} - y)$

Logistic regression có hàm gradient descent là $\frac{dL}{dw} = \frac{1}{N} * X^T * (\hat{y} - y)$

5. Dự đoán dữ liệu mới bằng model vừa tìm được

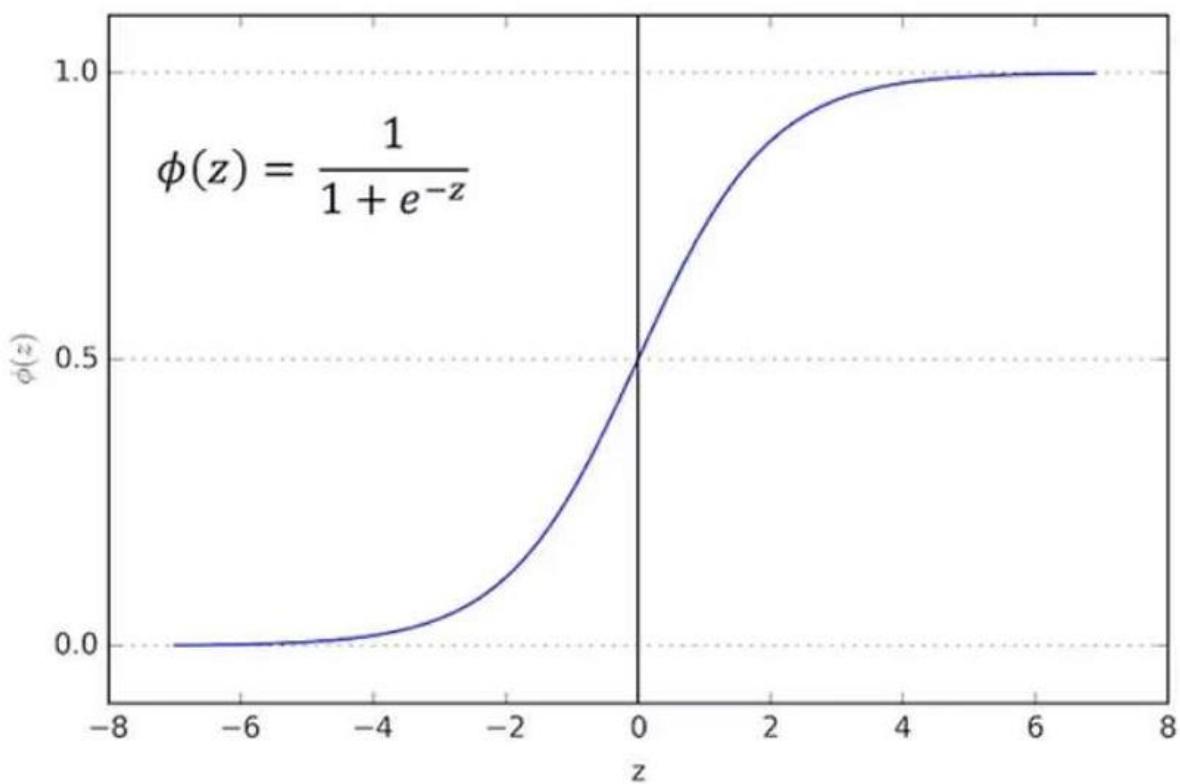
Hàm Sigmoid

- Xác suất của 1 sự kiện trong khoảng [0,1]
- Sự kiện bạn càng chắc chắn xảy ra thì xác suất càng cao. Ví dụ bạn lương cao và còn đi làm lâu lăm thì xác suất bạn được vay mua chung cư là cao.
- Tổng xác suất của sự kiện A và sự kiện phủ định của A là 100% (hay 1). Ví dụ sự kiện A: tung đồng xu mặt ngửa, xác suất 50%; phủ định của sự kiện A: tung đồng xu mặt sấp, xác suất 50% => tổng 100%.

Bạn sẽ thấy xác suất quan trọng hơn là chỉ 0 hay 1, ví dụ trước mỗi ca mổ khó, bác sĩ không thể chắc chắn là sẽ thất bại hay thành công mà chỉ có thể nói xác suất thành công là bao nhiêu (ví dụ 80%).

4.3 Hàm sigmoid

Giờ ta cần tìm xác suất của hồ sơ mới nên cho vay. Hay giá trị của hàm cần trong khoảng [0,1]. Rõ ràng là giá trị của phương trình đường thẳng như bài trước có thể ra ngoài khoảng [0,1] nên cần một hàm mới luôn có giá trị trong khoảng [0,1]. Đó là hàm sigmoid.



Hình 4.3: Đồ thị hàm sigmoid

Nhận xét:

- Hàm số liên tục, nhận giá trị thực trong khoảng (0,1).
- Hàm có đạo hàm tại mọi điểm (để áp dụng gradient descent).

Thiết lập bài toán

4.4 Thiết lập bài toán

Mọi người có để ý các bước trong bài linear regression không nhỉ, các bước bao gồm:

1. Visualize dữ liệu
2. Thiết lập model
3. Thiết lập loss function
4. Tìm tham số bằng việc tối ưu loss function
5. Dự đoán dữ liệu mới bằng model vừa tìm được

Đây là mô hình chung cho bài toán trong Deep Learning.



ChatGPT

Đúng vậy, các bước bạn đã liệt kê là những bước cơ bản trong quá trình huấn luyện một mô hình trong machine learning, bao gồm cả linear regression và deep learning. Dưới đây là cách thiết lập bài toán linear regression theo các bước mà bạn đã nêu:

1. Visualize dữ liệu:

Trước khi bắt đầu xây dựng mô hình, việc hiểu rõ dữ liệu thông qua việc visualize giúp ta có cái nhìn tổng quan về mối quan hệ giữa các biến. Điều này giúp phát hiện ra các điểm ngoại lệ, tương quan giữa các biến và hình dung ra một mô hình có thể phù hợp.

2. Thiết lập model:

Trong trường hợp linear regression, mô hình được thiết lập dưới dạng một phương trình tuyến tính:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

Trong đó, y là biến phụ thuộc, x_1, x_2, \dots, x_n là các biến độc lập, $\theta_0, \theta_1, \dots, \theta_n$ là các tham số cần được tối ưu.

3. Thiết lập loss function:

Trong linear regression, loss function thường là mean squared error (MSE), được biểu diễn như sau:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Trong đó, m là số lượng mẫu dữ liệu, $h_\theta(x^{(i)})$ là giá trị dự đoán của mô hình cho mẫu thứ i , $y^{(i)}$ là giá trị thực của mẫu thứ i .

4. Tìm tham số bằng việc tối ưu loss function:

Để tìm tham số tối ưu cho mô hình, chúng ta cần sử dụng một thuật toán tối ưu hóa như gradient descent. Quá trình này nhằm tối thiểu hóa giá trị của loss function bằng cách điều chỉnh các tham số θ .

5. Dự đoán dữ liệu mới bằng model vừa tìm được:

Sau khi đã tìm được các tham số tối ưu, ta có thể sử dụng mô hình để dự đoán giá trị của biến phụ thuộc y cho các mẫu dữ liệu mới bằng cách đưa vào các giá trị của các biến độc lập x vào mô hình đã huấn luyện.

Các bước này cũng tương tự được áp dụng trong deep learning, tuy nhiên, mô hình trong deep learning thường có cấu trúc phức tạp hơn và quy trình huấn luyện cũng đòi hỏi nhiều công đoạn hơn.

Model

4.4.1 Model

Với dòng thứ i trong bảng dữ liệu, gọi $x_1^{(i)}$ là lương và $x_2^{(i)}$ là thời gian làm việc của hồ sơ thứ i .

$p(x^{(i)} = 1) = \hat{y}_i$ là xác suất mà model dự đoán hồ sơ thứ i được cho vay.

$p(x^{(i)} = 0) = 1 - \hat{y}_i$ là xác suất mà model dự đoán hồ sơ thứ i không được cho vay.

$$\Rightarrow p(x^{(i)} = 1) + p(x^{(i)} = 0) = 1$$

Hàm sigmoid: $\sigma(x) = \frac{1}{1 + e^{-x}}$.

4.4 Thiết lập bài toán

69

Như bài trước công thức của linear regression là: $\hat{y}_i = w_0 + w_1 * x_i$ thì giờ công thức của logistic regression là:

$$\hat{y}_i = \sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)}) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)})}}$$

Ở phần cuối mọi người sẽ thấy được quan hệ giữa xác suất và đường thẳng.

Tóm lại

Logistic regresion là hàm sigmoid : $\sigma(x) = \frac{1}{1 + e^{-x}}$

$$\hat{y}_i = \sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)})}}$$

Mục đích là cho \hat{y}_i về khoảng giá trị [0,1]

Loss function

Nhắc lại công thức Logarit

| | |
|--|---|
| 1. $\log_a 1 = 0$ | 8. $\log_a N^2 = 2 \cdot \log_a N $ |
| 2. $\log_a a = 1$ | 9. $\log_a N = \log_a b \cdot \log_b N$ |
| 3. $\log_a a^m = m$ | 10. $\log_a N = \frac{\log_a N}{\log_a b}$ |
| 4. $a^{\log_a n} = n$ | 11. $\log_a b = \frac{1}{\log_a b}$ |
| 5. $\log_a(n_1 \cdot n_2) = \log_a n_1 + \log_a n_2$ | 12. $\log_a^\alpha N = \frac{1}{\alpha} \cdot \log_a N$ |
| 6. $\log_a \left(\frac{n_1}{n_2} \right) = \log_a n_1 - \log_a n_2$ | 13. $a^{\log_b c} = c^{\log_b a}$ |
| 7. $\log_a N^\alpha = \alpha \cdot \log_a N$ | 14. $\log_a x = \frac{\log_b x}{\log_b a}$ |

Hàm Loss function

$$L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

❖ log ở đây chính là logarit cơ số e (hằng số tự nhiên)

$$\begin{aligned} L &= -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i)) = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i)) \\ &= -(y_i * \ln(\hat{y}_i) + (1 - y_i) * \ln(1 - \hat{y}_i)) \end{aligned}$$

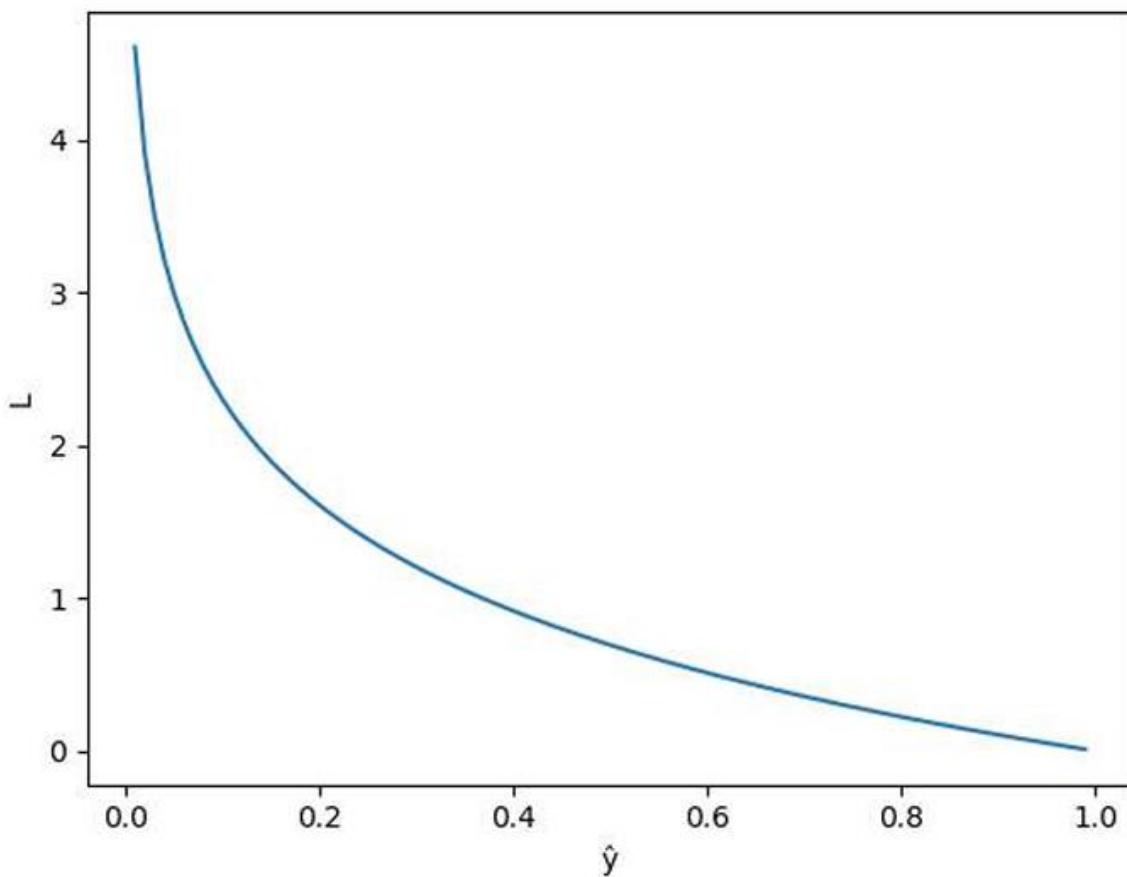
Giờ cũng cần một hàm để đánh giá độ tốt của model. Như bài trước là \hat{y} càng gần y càng tốt, giờ cũng vậy:

- Nếu hồ sơ thứ i là cho vay, tức $y_i = 1$ thì ta cũng mong muốn \hat{y}_i càng gần 1 càng tốt hay model dự đoán xác suất người thứ i được vay vốn càng cao càng tốt.
- Nếu hồ sơ thứ i không được vay, tức $y_i = 0$ thì ta cũng mong muốn \hat{y}_i càng gần 0 càng tốt hay model dự đoán xác suất người thứ i được vay vốn càng thấp càng tốt.

Với mỗi điểm $(x^{(i)}, y_i)$, gọi hàm loss function $L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$, loss function này có tên gọi là **binary_crossentropy**

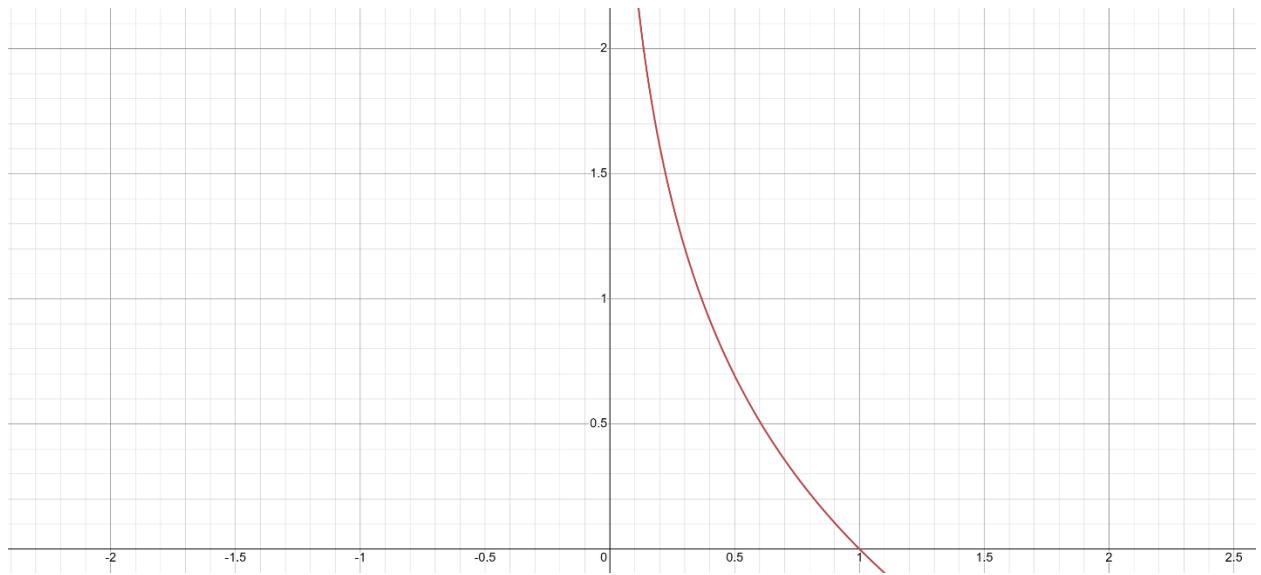
Mặc định trong machine learning nói chung hay deep learning thì viết log hiểu là ln

Thử đánh giá hàm L nhé. Nếu $y_i = 1 \Rightarrow L = -\log(\hat{y}_i)$



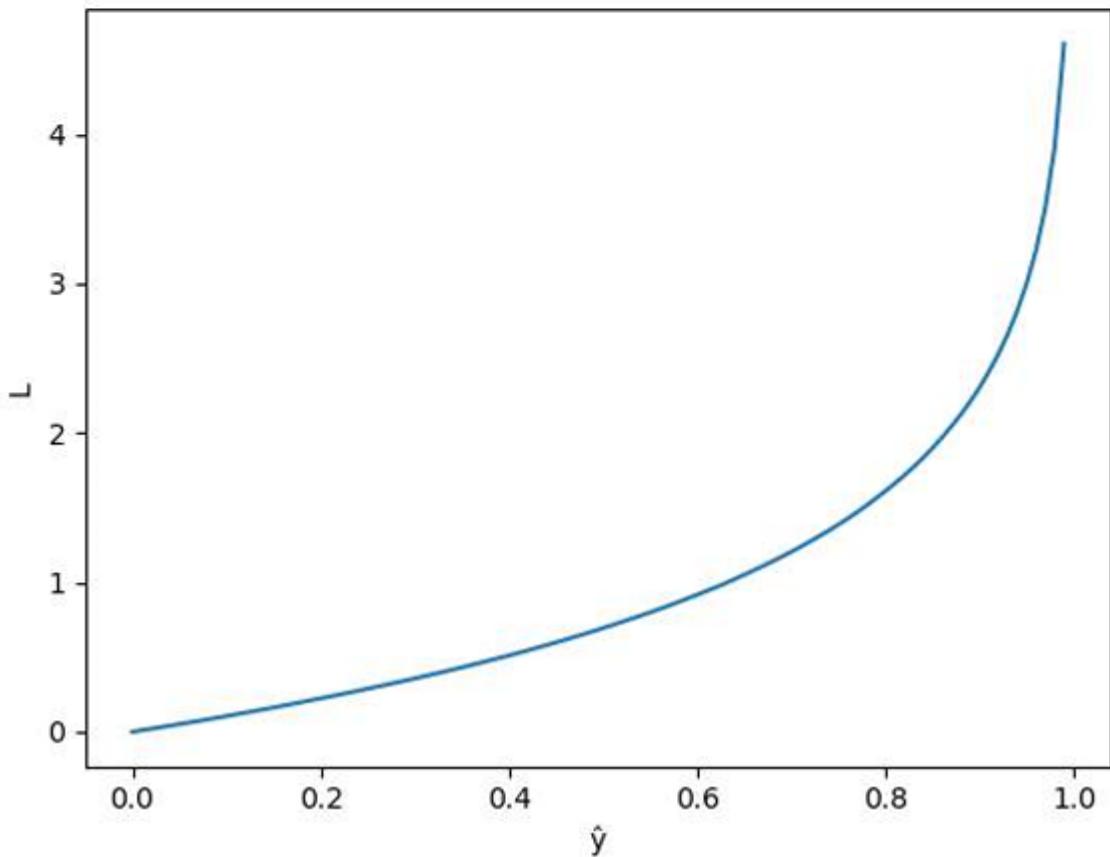
Quan sát đồ thị

- Do hàm sigmoid nên \hat{y}_i sẽ có giá trị trong khoảng [0,1]
- $L = -\log(\hat{y}_i) = -\log_e(\hat{y}_i)$
- Nếu \hat{y}_i tiến về 1 thì L tiến về 0
- Nếu \hat{y}_i tiến về 0 thì L tiến về dương vô cùng



Nhận xét:

- Hàm L giảm dần từ 0 đến 1.
 - Khi model dự đoán \hat{y}_i gần 1, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0
 - Khi model dự đoán \hat{y}_i gần 0, tức giá trị dự đoán ngược lại giá trị thật y_i thì L rất lớn
- Ngược lại, nếu $y_i = 0 \Rightarrow L = -\log(1 - \hat{y}_i)$



Nhận xét:

- Hàm L tăng dần từ 0 đến 1
- Khi model dự đoán \hat{y}_i gần 0, tức giá trị dự đoán gần với giá trị thật y_i thì L nhỏ, xấp xỉ 0
- Khi model dự đoán \hat{y}_i gần 1, tức giá trị dự đoán ngược lại giá trị thật y_i thì L rất lớn

=> Hàm L nhỏ khi giá trị model dự đoán gần với giá trị thật và rất lớn khi model dự đoán sai, hay nói cách khác L càng nhỏ thì model dự đoán càng gần với giá trị thật. => Bài toán tìm model trở thành tìm giá trị nhỏ nhất của L

$$\text{Hàm loss function trên toàn bộ dữ liệu } J = -\frac{1}{N} * \sum_{i=1}^N (y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$$

Tóm lại : Hàm loss function trên **toàn bộ dữ liệu**

$$\text{Loss function } J = -\frac{1}{N} * \sum_{i=1}^n -(y_i * \ln(\hat{y}_i) + (1 - y_i) * \ln(1 - \hat{y}_i))$$

Chain Rule

Quy tắc chain Rule

Nếu $z = f(y)$ và $y = g(x)$ hay $z = f(g(x))$ thì $\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx}$

Nhắc lại về công thức đạo hàm

Giới hạn, nếu có, của tỉ số giữa số gia của hàm số và số gia của đối số tại x_0 , khi số gia của đối số tiến dần tới 0, được gọi là đạo hàm của hàm số $y = f(x)$ tại điểm x_0 .

Cho hàm số $y = f(x)$ xác định trên $(a; b)$ và $x_0 \in (a; b)$:

$$f'(x_0) = \lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0} = \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} \quad (\Delta x = x - x_0, \Delta y = f(x_0 + \Delta x) - f(x_0))$$

Nếu hàm số $y = f(x)$ có đạo hàm tại x_0 thì nó liên tục tại điểm đó.

Công thức đạo hàm của tổng, hiệu, tích, thương

Giả sử $u = u(x)$ và $v = v(x)$ là các hàm số có đạo hàm tại điểm x thuộc khoảng xác định. Ta có:

$$(u + v)' = u' + v'$$

$$\left(\frac{u}{v}\right)' = \frac{u'v - uv'}{v^2}, \quad (v(x) \neq 0)$$

$$(u - v)' = u' - v'$$

$$\left(\frac{k}{u'}\right)' = -\frac{k \cdot u'}{u^2}$$

$$(u \cdot v)' = u' \cdot v + u \cdot v'$$

$$y'_x = y'_u \cdot u'_x$$

$$(ku)' = ku'$$

Đạo hàm các hàm số sơ cấp

$$(C)' = 0$$

$$(x^a)' = a \cdot x^{a-1} \forall a \in \mathbb{R}$$

$$(u^a)' = a \cdot u^{a-1} \cdot u' \forall a \in \mathbb{R}$$

$$(\sqrt{x})' = \frac{1}{2\sqrt{x}}$$

$$\left(\frac{1}{x}\right)' = -\frac{1}{x^2}$$

$$(\sqrt{u})' = \frac{u'}{2\sqrt{u}}$$

$$\left(\frac{1}{u}\right)' = -\frac{u'}{u^2}$$

$$(\sqrt[n]{x})' = \frac{1}{n\sqrt[n]{x^{n-1}}} \quad n \in N^*, n > 1$$

$$(\sqrt[n]{u})' = \frac{u'}{n\sqrt[n]{u^{n-1}}} \quad n \in N^*, n > 1$$

$$(\sin x)' = \cos x$$

$$(\sin u)' = u' \cdot \cos u$$

$$(\cos x)' = -\sin x$$

$$(\cos u)' = -u' \cdot \sin u$$

$$(\tan x)' = 1 + \tan^2 x = \frac{1}{\cos^2 x}$$

$$(\tan u)' = u' \cdot (1 + \tan^2 u) = \frac{u'}{\cos^2 u}$$

$$\left(\frac{a_1x^2 + b_1x + c_1}{a_2x^2 + b_2x + c_2} \right) = \frac{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \cdot x^2 + 2 \begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix} \cdot x + \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{(a_2x^2 + b_2x + c_2)^2}$$

$$(\cot x)' = - (1 + \cot^2 x) = -\frac{1}{\sin^2 x}$$

$$(\cot u)' = -u' \cdot (1 + \cot^2 u) = -\frac{u'}{\sin^2 u}$$

$$(e^x)' = e^x$$

$$(e^u)' = u' \cdot e^u$$

$$(a^x)' = a^x \cdot \ln a$$

$$(a^u)' = u' \cdot a^u \cdot \ln a$$

$$(\ln x)' = \frac{1}{x}$$

$$(\ln u)' = \frac{u'}{u}$$

$$(\log_a x)' = \frac{1}{x \cdot \ln a}$$

$$(\log_a u)' = \frac{u'}{u \cdot \ln a}$$

$$\left(\frac{ax + b}{cx + d} \right)' = \frac{ad - bc}{(cx + d)^2}$$

$$\left(\frac{ax^2 + bx + c}{ex + f} \right)' = \frac{aex^2 + 2af \cdot x + (bf - ce)}{(ex + f)^2}$$

Đạo hàm cấp cao

$$(x^m)^{(n)} = m(m-1)\dots(m-n+1) \cdot x^{(m-n)}$$

$$(\ln x)^{(n)} = \frac{(-1)^{n-1}(n-1)!}{x^n}$$

$$(a^x)^{(n)} = a^x \cdot \ln^n a \quad \text{với } a > 0.$$

$$(\sin x)^{(n)} = \sin(x + n\frac{\pi}{2})$$

$$(\cos x)^{(n)} = \cos(x + n\frac{\pi}{2})$$

$$(e^x)^{(n)} = e^x$$

$$(\frac{1}{x})^{(n)} = (-1)^n \cdot n! \cdot x^{-n-1}$$

Quy tắc cơ bản của đạo hàm

| | |
|----------------------------|---|
| | $(u+v)' = u' + v'$ |
| $(C') = 0$ | $(u-v)' = u' - v'$ |
| Đạo hàm của hằng số bằng 0 | $(u_1 + u_2 + \dots + u_n)' = (u_1)' + (u_2)' + \dots + (u_n)'$ |
| | $(u_1 - u_2 - \dots - u_n)' = (u_1)' - (u_2)' - \dots - (u_n)'$ |
| | Đạo mà của một tổng bằng tổng các đạo hàm |
| $(uv)' = (u)'v + (v)'u$ | $\left(\frac{u}{v}\right)' = \frac{(u)'v - (v)'u}{v^2}$ |

Quy tắc đạo hàm của hàm số hợp

Nếu $y = y(u(x))$ thì $y'(x) = y'(u) \times u'(x)$

Công thức đạo hàm cơ bản

Đạo hàm của $f(x)$ với x là biến số

| |
|--|
| $(kx)' = k$ |
| $(x^n)' = nx^{n-1}$ |
| $\left(\frac{1}{x}\right)' = -\frac{1}{x^2}$ |
| $(\sqrt{x})' = \frac{1}{2\sqrt{x}}$ |
| $(\sin x)' = \cos x$ |
| $(\cos x)' = -\sin x$ |
| $(\tan x)' = 1 + \tan^2 x = \frac{1}{\cos^2 x}$ |
| $(\cot x)' = -\left(1 + \cot^2 x\right) = -\frac{1}{\sin^2 x}$ |
| $(e^x)' = e^x$ |
| $(a^x)' = a^x \times \ln a$ |
| $(\ln x)' = (\ln x)' = \frac{1}{x}$ |

Đạo hàm của $f(u)$ với u là một hàm số

| |
|---|
| $(k \times u)' = k \times u'$ |
| $(u^n)' = n \times u^{n-1} \times (u)'$ |
| $\left(\frac{1}{u}\right)' = -\frac{(u)'}{u^2}$ |
| $(\sqrt{u})' = -\frac{(u)'}{2\sqrt{u}}$ |
| $(\sin u)' = \cos u \times (u)'$ |
| $(\cos u)' = -\sin u \times (u)'$ |
| $(\tan u)' = (1 + \tan^2 u) \times (u)' = \frac{(u)'}{\cos^2 u}$ |
| $(\cot u)' = -\left(1 + \cot^2 u\right) \times (u)' = -\frac{(u)'}{\sin^2 u}$ |
| $(e^u)' = e^u \times (u)'$ |
| $(a^u)' = a^u \times \ln a \times (u)'$ |

$$(\log a)' = (\log_a |x|)' = \frac{1}{x \times \ln a}$$

$$\begin{aligned} (\ln u)' &= (\ln |u|)' = \frac{(u)'}{u} \\ (\log_a u)' &= (\log_a |u|)' = \frac{(u)}{u \times \ln a} \end{aligned}$$

Công thức đạo hàm lượng giác

$$(\sin x)' = \cos x$$

$$(\cos x)' = -\sin x$$

$$(\tan x)' = \left(\frac{\cos x}{\sin x}\right)' = \frac{\cos^2 x + \cos^2 x}{\sin^2 x} = \frac{1}{\cos^2 x}$$

$$(\cot x)' = \left(\frac{\sin x}{\cos x}\right)' = \frac{-\sin^2 x - \sin 2x}{\cos^2 x} = -(1 + \cot^2 x)$$

$$(\sec(x))' = \frac{1}{(\cos x)'} = \frac{\sin x}{\cos^2 x} = \frac{1}{\cos x} \times \frac{\sin x}{\cos x} = \sec(x) \times \tan x$$

$$(\csc(x))' = \left(\frac{1}{\sin x}\right)' = -\frac{\cos x}{\sin^2 x} = -\frac{1}{\sin x} \times \frac{\cos x}{\sin x} = -\csc(x) \cot(x)$$

$$(\arcsin(x))' = \frac{1}{\sqrt{1-x^2}}$$

$$(\arccos(x))' = \frac{-1}{\sqrt{1-x^2}}$$

$$(\arctan(x))' = \frac{1}{x^2+1}$$

Bảng đạo hàm

| | |
|---|---|
| $x^{a'} = ax^{a-1}$ | $(u^a)' = a \cdot u' \cdot u^{a-1}$ |
| $(\sin x)' = \cos x$ | $(\sin u)' = u' \cdot \cos u$ |
| $(\cos x)' = -\sin x$ | $(\cos u)' = -u' \cdot \sin u$ |
| $(\tan x)' = \frac{1}{\cos^2 x} = 1 + \tan^2 x$ | $(\tan u)' = \frac{u'}{\cos^2 u} = u' \cdot (1 + \tan^2 u)$ |
| $(\cot x)' = \frac{-1}{\sin^2 x} = -(1 + \cot^2 x)$ | $(\cot u)' = \frac{-u'}{\sin^2 u} = -u' \cdot (1 + \cot^2 u)$ |
| $\log_a x' = \frac{1}{x \ln a}$ | $\log_a u' = \frac{u'}{u \cdot \ln a}$ |
| $\ln x' = \frac{1}{x}$ | $\ln u' = \frac{u'}{u}$ |
| $a^{x'} = a^x \cdot \ln a$ | $a^{u'} = a^u \cdot u' \cdot \ln a$ |
| $e^{x'} = e^x$ | $(e^u)' = u' \cdot e^u$ |

Bảng nguyên hàm

$$\int x^a dx = \frac{x^{a+1}}{a+1} + c, (a \neq -1)$$

$$\int (ax+b)^a dx = \frac{1}{a} \cdot \frac{(ax+b)^{a+1}}{a+1} + c$$

$$\int \sin x dx = -\cos x + c$$

$$\int \sin(ax+b) dx = -\frac{1}{a} \cos(ax+b) + c$$

$$\int \cos x dx = \sin x + c$$

$$\int \cos(ax+b) dx = \frac{1}{a} \sin(ax+b) + c$$

$$\int \frac{1}{\cos^2 x} dx = \tan x + c$$

$$\int \frac{1}{\cos^2(ax+b)} dx = \frac{1}{a} \tan(ax+b) + c$$

$$\int \frac{1}{\sin^2 x} dx = -\cot x + c$$

$$\int \frac{1}{\sin^2(ax+b)} dx = -\frac{1}{a} \cot(ax+b) + c$$

$$\int \frac{1}{x} dx = \ln|x| + c$$

$$\int \frac{1}{ax+b} dx = \frac{1}{a} \ln|ax+b| + c$$

$$\int a^x dx = \frac{a^x}{\ln a} + c$$

$$\int a^{ax+\beta} dx = \frac{a^{ax+\beta}}{\alpha \cdot \ln a} + c$$

$$\int e^x dx = e^x + c$$

$$\int e^{ax+b} dx = \frac{1}{a} e^{ax+b} + c$$

Gradient descent trong Logistic regression

Logistic regression có hàm gradient descent là $\frac{dJ}{dw} = \frac{1}{N} * \mathbf{X}^T * (\hat{\mathbf{y}} - \mathbf{y})$

Để áp dụng thuật toán gradient descent tìm tối ưu loss function mình cần tính đạo hàm của loss function với w.

Với mỗi điểm $(x^{(i)}, y_i)$, gọi hàm loss function

$L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$ trong đó $\hat{y}_i = \sigma(w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)}) = \sigma(z)$ là giá trị mà model dự đoán, còn y_i là giá trị thật của dữ liệu, $z = w_0 + w_1 * x_1^{(i)} + w_2 * x_2^{(i)}$.

Nhắc lại đạo hàm cơ bản, $\frac{d(\log(x))}{dx} = \frac{1}{x}$

Áp dụng chain rule ta có: $\frac{dL}{dw_0} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{dz} * \frac{dz}{dw_0}$

$$\frac{dL}{d\hat{y}_i} = -\frac{d(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))}{d\hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right) = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)}$$

Giải thích : Áp dụng quy tắc chain rule ta có

$$\frac{d(\ln(1 - \hat{y}_i))}{d\hat{y}_i} = \frac{d(\ln(1 - \hat{y}_i))}{d(1 - \hat{y}_i)} * \frac{d(1 - \hat{y}_i)}{d\hat{y}_i} = \frac{1}{1 - \hat{y}_i} * -1 = \frac{-1}{1 - \hat{y}_i}$$

[HAY] Tóm tắt Logistic Regression

Mô hình chung cho bài toán trong Deep Learning.

1. Visualize dữ liệu

2. Thiết lập model Logistic regression là hàm sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\hat{y}_i = \sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}) = \frac{1}{1+e^{-(w_0+w_1x_1^{(i)}+\dots+w_mx_m^{(i)})}} \text{ (cho } \hat{y}_i \text{ về khoảng [0,1])}$$

3. Thiết lập loss function **binary crossentropy** : $L = -(y_i * \log(\hat{y}_i) + (1 - y_i) * \log(1 - \hat{y}_i))$

4. Tìm tham số bằng việc tối ưu loss function (Gradient descent) : $\frac{dJ}{dw} = \frac{1}{N} * X^T * (\hat{y} - y)$

5. Dự đoán dữ liệu mới bằng model vừa tìm được

Giải thích :

- Ta có $f(x)$, để tìm cực trị của nó ta đi đạo hàm và cho $f'(x) = 0$ để tìm ra x , tuy nhiên nhiều hàm $f(x)$ phức tạp ta chỉ đạo hàm ra được $f'(x)$ còn việc tìm ra được x là rất khó . Chính vì thế nên ta dùng Gradient descent để đi tìm giá trị gần giống với nghiệm x nhất .

- **Tóm lại :**

Model + Gradient descent : Đi tìm w mới sau mỗi vòng lặp

Loss function : MSE hay binary crossentropy để đánh giá w mới đã tối ưu hay chưa thông qua việc xem giá trị mất mát mà w mới này mang lại sau mỗi vòng lặp từ đó điều chỉnh hệ số learning rate và epochs cho hợp lý để tìm ra được w chính xác mà không bị các tình trạng sóm , trễ hay không chính xác .

- Ở trong Linear regression điều đặc biệt là ta có thể tìm ra được nghiệm của hàm mất mát sau khi đạo hàm . Nhưng ở Logistic thì ta không thể tìm ra được nghiệm . Mặc khác việc giải toán và tìm ra nghiệm này cũng không quan trọng vì như đã nói ở trên .

Hàm Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} ; \forall x \in \mathbb{R} \rightarrow \sigma(x) \in (0, 1)$$

Giải thích : Mục đích là có đạo hàm với mọi x nhưng giá trị của $f(x)$ chỉ ở phạm vi $(0, 1)$

Thiết lập Model

$$\hat{y}_i = \sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}) = \frac{1}{1+e^{-(w_0+w_1x_1^{(i)}+\dots+w_mx_m^{(i)})}}$$

Gradient Descent

$$\frac{dJ}{dw} = \frac{1}{N} * X^T * (\hat{y} - y)$$

Binary Crossentropy Loss function

$$L = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

Giải thích :

- Hàm này bắt nguồn từ công thức trong thuyết Entropy (information theory) . Tham khảo [Cách xây dựng hàm Binary Cross Entropy](#) .
- Ta có thể dùng cơ số 2 , 10 , e . Tuy nhiên để cho phù hợp với hàm Sigmoid sau này thì ta dùng cơ số e .

Tóm lại :

- Sigmoid , Model và Gradient Descent để tìm ra và tối ưu dần w
- Binary Crossentropy Loss function để đánh giá mỗi khi tìm ra w mới thông qua giá trị mất mát mà w mới đó mang lại

Giải bài toán bằng đại số tuyến tính

Quy tắc Chain Rule : Nếu $z = f(y)$ và $y = g(x)$ hay $z = f(g(x))$ thì $\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx}$

Áp dụng Gradient descent

$$\text{Ta có : } L = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

$$\hat{y}_i = \sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}) = \frac{1}{1 + e^{-(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)})}}$$

Như vậy ta đã có hàm loss function theo biến w như thường lệ ta sẽ đi tìm đạo hàm của hàm loss function nhằm mục đích sau này kết hợp với Gradient descent để đi tìm nghiệm w .

Áp dụng quy tắc Chain rule

$$\frac{dL}{dw} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{w}$$

Ta có

$$\frac{dL}{d\hat{y}_i} = \frac{d(-(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i)))}{d\hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right) = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)}$$

$$\frac{d(\sigma(x))}{dx} = \frac{d\left(\frac{1}{1 + e^{-x}}\right)}{dx} = \frac{d\left(\frac{1}{1 + e^{-x}}\right)}{d(1 + e^{-x})} * \frac{d(1 + e^{-x})}{d(-x)} * \frac{d(-x)}{x} = -\frac{1}{(1 + e^{-x})^2} * e^{-x} * -1$$

$$= \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} * \frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^{-x}} * \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma(x) * (1 - \sigma(x))$$

$$\text{Đặt } z^i = w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}$$

$$\frac{d\hat{y}_i}{w_0} = \frac{d(\sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}))}{dw_0} = \frac{d(\sigma(z^i))}{dz^i} * \frac{dz^i}{dw_0} = \sigma(z^i) * (1 - \sigma(z^i)) * 1 = \hat{y}_i * (1 - \hat{y}_i)$$

$$\frac{d\hat{y}_i}{w_1} = \frac{d(\sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}))}{dw_1} = \frac{d(\sigma(z^i))}{dz^i} * \frac{dz^i}{dw_1} = \sigma(z^i) * (1 - \sigma(z^i)) * x_1^{(i)} = \hat{y}_i * (1 - \hat{y}_i) * x_1^{(i)}$$

...

$$\frac{d\hat{y}_i}{w_m} = \frac{d(\sigma(w_0 + w_1 x_1^{(i)} + \dots + w_m x_m^{(i)}))}{dw_m} = \frac{d(\sigma(z^i))}{dz^i} * \frac{dz^i}{dw_m} = \sigma(z^i) * (1 - \sigma(z^i)) * x_m^{(i)} = \hat{y}_i * (1 - \hat{y}_i) * x_m^{(i)}$$

Do đó

$$\frac{dL}{dw_0} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{w_0} = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)} * \hat{y}_i * (1 - \hat{y}_i) = \hat{y}_i - y_i$$

$$\frac{dL}{dw_1} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{w_1} = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)} * \hat{y}_i * (1 - \hat{y}_i) * x_1^{(i)} = x_1^{(i)} * (\hat{y}_i - y_i)$$

...

$$\frac{dL}{dw_m} = \frac{dL}{d\hat{y}_i} * \frac{d\hat{y}_i}{w_m} = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)} * \hat{y}_i * (1 - \hat{y}_i) * x_m^{(i)} = x_m^{(i)} * (\hat{y}_i - y_i)$$

Đạo hàm trên **toàn bộ dữ liệu** ta sẽ có

$$J = \frac{1}{N} * \sum_{i=1}^N L = -\frac{1}{N} * \sum_{i=1}^N x_m^{(i)} * (y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

$$\frac{dJ}{dw_0} = \frac{1}{N} * \sum_{i=1}^N \hat{y}_i - y_i$$

$$\frac{dJ}{dw_1} = \frac{1}{N} * \sum_{i=1}^N x_1^{(i)} * (\hat{y}_i - y_i)$$

...

$$\frac{dJ}{dw_m} = \frac{1}{N} * \sum_{i=1}^N x_m^{(i)} * (\hat{y}_i - y_i)$$

Nếu để ý quan sát ta sẽ thấy

$$\frac{dJ}{dw} = \frac{dJ}{dw_0} + \frac{dJ}{dw_1} + \dots + \frac{dJ}{dw_m} = \frac{\mathbf{1}}{N} * \mathbf{X}^T * (\hat{\mathbf{y}} - \mathbf{y})$$

Vậy Tóm lại ta đã có :

- Sigmoid , Model và Gradient Descent để tìm ra và tối ưu dần w
- Đạo hàm của Binary Crossentropy Loss function để đánh giá mỗi khi tìm ra w mới thông qua giá trị mất mát mà w mới đó mang lại

Code dự đoán pass hay không pass môn học

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
def train_test_split(X, y, train_ratio=0.7):
    m = len(X)
    train_size = int(train_ratio * m)
    indices = np.random.permutation(m)
    train_indices = indices[:train_size]
    test_indices = indices[train_size:]
    X_train, y_train = X[train_indices], y[train_indices]
    X_test, y_test = X[test_indices], y[test_indices]
    return X_train, y_train, X_test, y_test

# Hàm sigmoid
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

```

# Hàm loss binary crossentropy
def binary_crossentropy(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Huấn luyện mô hình bằng gradient descent
def logistic_regression(X, y, lr=0.01, epochs=10000, epsilon=1e-6):
    m, n = X.shape
    w = np.zeros((n, 1))
    losses = [] # List để lưu giá trị loss function qua các epoch
    for epoch in range(epochs):
        z = np.dot(X, w)
        y_pred = sigmoid(z)
        loss = binary_crossentropy(y, y_pred)
        losses.append(loss)
        gradient = np.dot(X.T, (y_pred - y)) / m
        w -= lr * gradient
        if np.linalg.norm(gradient) < epsilon:
            break
    return w, losses

# Dự đoán và tính các độ đo chất lượng mô hình
def evaluate_model(X, y, w):
    y_pred = sigmoid(np.dot(X, w))
    y_pred_labels = (y_pred > 0.5).astype(int)
    accuracy = np.mean(y_pred_labels == y)
    TP = np.sum((y_pred_labels == 1) & (y == 1))
    FN = np.sum((y_pred_labels == 0) & (y == 1))
    FP = np.sum((y_pred_labels == 1) & (y == 0))
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    f1_score = 2 * (precision * recall) / (precision + recall)
    return accuracy, recall, f1_score

# Đọc dữ liệu từ file
data = pd.read_csv('data_logistic.csv').values
N, d = data.shape
X = data[:, 0:d-1].reshape(-1, d-1)
y = data[:, 2].reshape(-1, 1)
X = np.hstack((np.ones((N, 1)), X)) # thêm cột 1 vào cho X

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
X_train, y_train, X_test, y_test = train_test_split(X, y)

# Huấn luyện mô hình

```

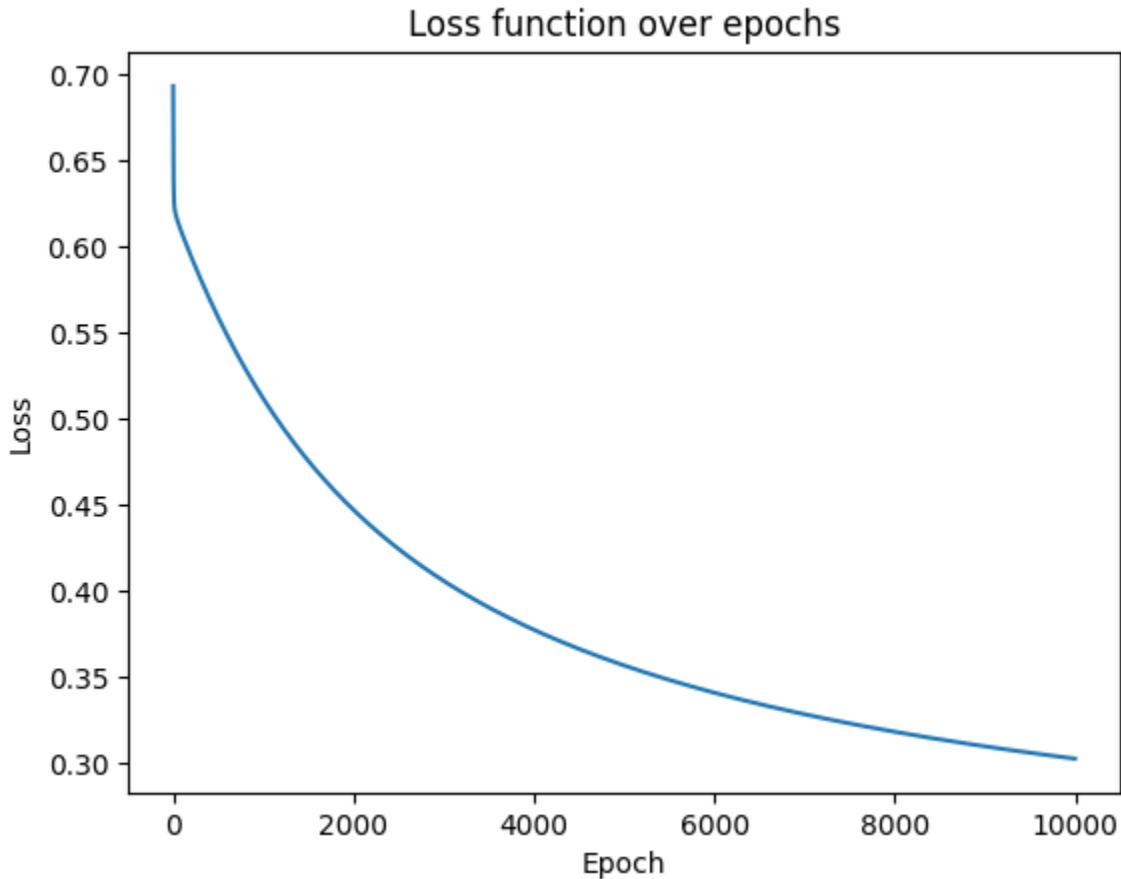
```
w, losses = logistic_regression(X, y)

# Đánh giá mô hình trên tập kiểm tra
accuracy, recall, f1_score = evaluate_model(X_test, y_test, w)

print("Accuracy:", accuracy)
print("Recall:", recall)
print("F1-score:", f1_score)

# Vẽ biểu đồ hàm loss function qua các epoch
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss function over epochs')
plt.show()
```

Accuracy: 0.9333333333333333
 Recall: 1.0
 F1-score: 0.9411764705882353



```
# Load data từ file csv
```

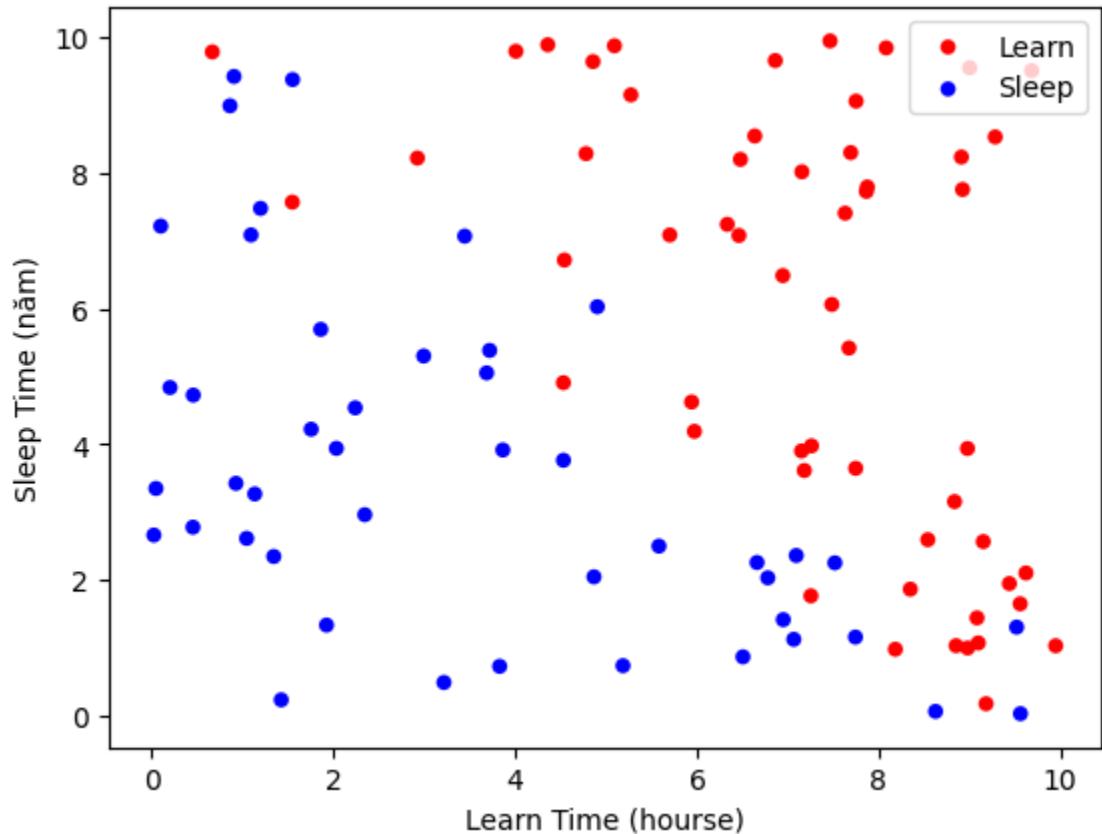
```

data = pd.read_csv('data_logistic.csv').values
N, d = data.shape
x = data[:, 0:d-1].reshape(-1, d-1)
y = data[:, 2].reshape(-1, 1)

# Vẽ data bằng scatter
x_learn = x[y[:,0]==1]
x_sleep = x[y[:,0]==0]

plt.scatter(x_learn[:, 0], x_learn[:, 1], c='red', edgecolors='none', s=30, label='Learn')
plt.scatter(x_sleep[:, 0], x_sleep[:, 1], c='blue', edgecolors='none', s=30, label='Sleep')
plt.legend(loc=1)
plt.xlabel('Learn Time (hourse)')
plt.ylabel('Sleep Time (năm)')

```



```

# Tính giá trị đầu vào cho mô hình logistic regression
learn_time , sleep_time = 6 , 4
X_new = np.array([[1, learn_time, sleep_time]])

# Dự đoán kết quả sử dụng mô hình đã huấn luyện
y_pred = sigmoid(np.dot(X_new, w))

```

```

print(y_pred)

# Kiểm tra kết quả dự đoán
if y_pred > 0.5:
    print("Bạn có thể qua.")
else:
    print("Bạn không thể qua.")

```

[[0.59147665]]

Bạn có thể qua.

[Đánh giá model AI với Precision, Recall và F1 Score - Mì AI \(miai.vn\)](#)

| | | Model dự đoán | |
|---------|----------|---|---|
| | | Positive | Negative |
| Thực tế | Positive | True Positive
(Dự đoán Đúng là Positive) | False Negative
(Dự đoán Sai là Negative) |
| | Negative | False Positive
(Dự đoán Sai là Positive) | True Negative
(Dự đoán Đúng là Negative) |

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$F1 = 2 \frac{Precision * Recall}{Precision + Recall}$$

$$TruePositiveRate = \frac{TruePositive}{TruePositive + FalseNegative}$$

$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative}$$

Phần 3 : Neural Network

Tổng quan

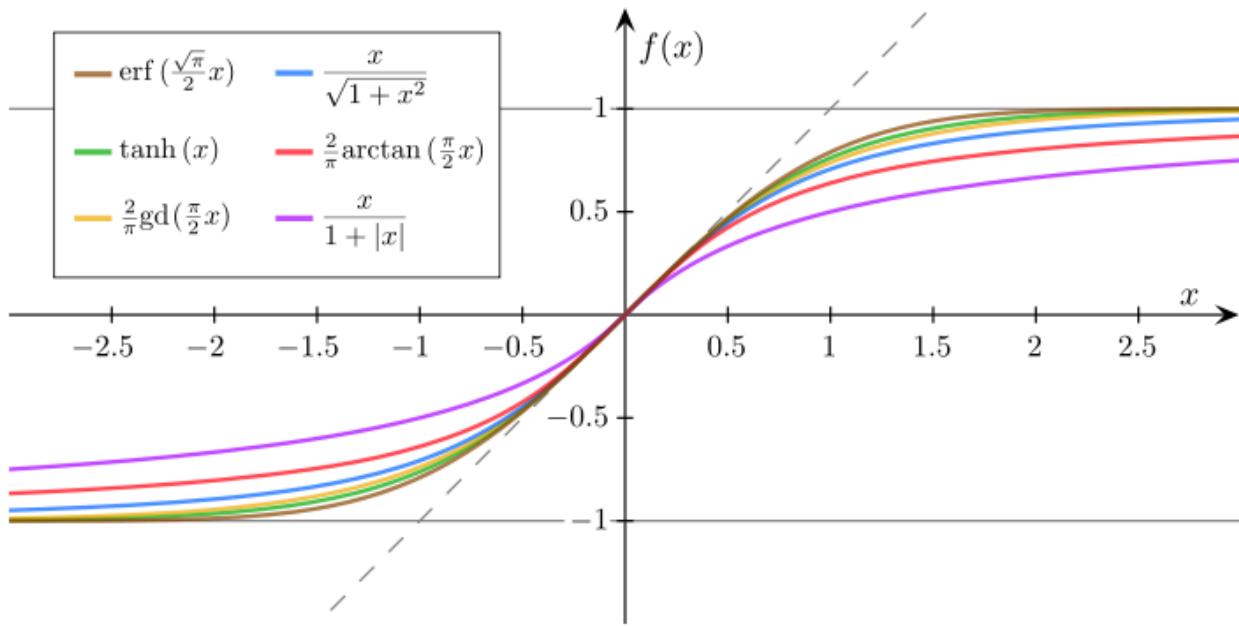
Mỗi node trong hidden layer và output layer :

- Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.
- Mỗi node có 1 hệ số bias b riêng.
- Diễn ra 2 bước: tính tổng linear và áp dụng activation function.

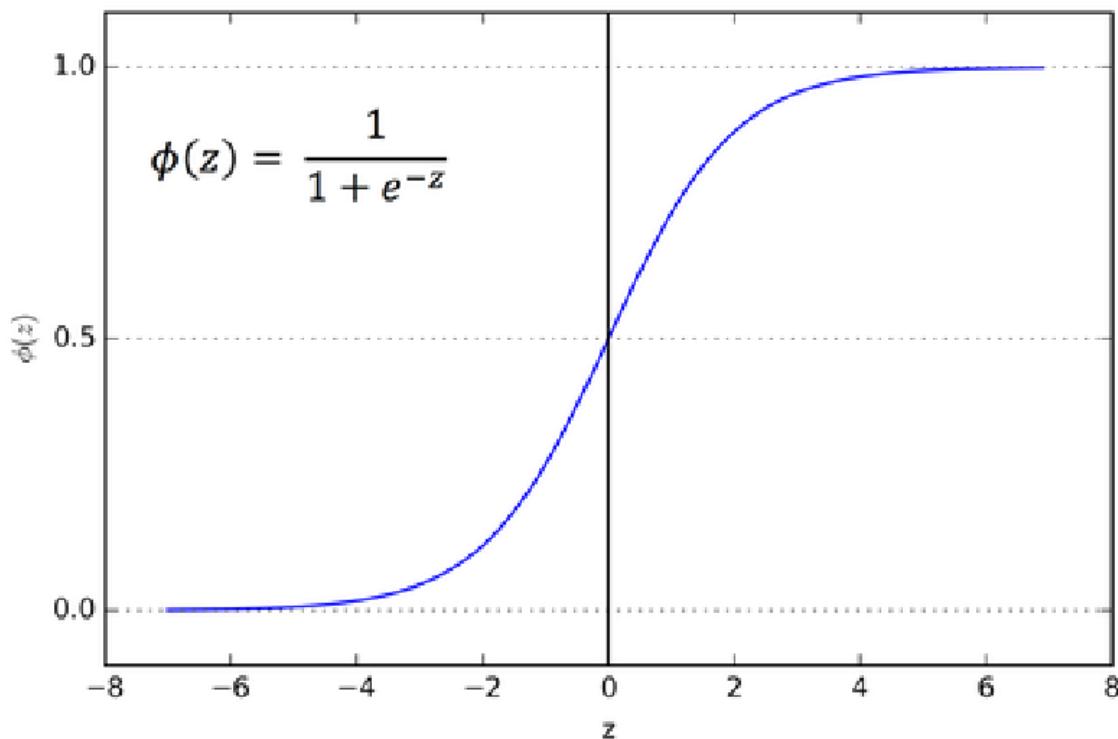
Nhắc lại

Hàm kích hoạt (Activation function)

⇒ Việc lựa chọn hàm kích hoạt nào phụ thuộc vào mong muốn dữ liệu đầu ra của ta và bài toán muốn làm gì .

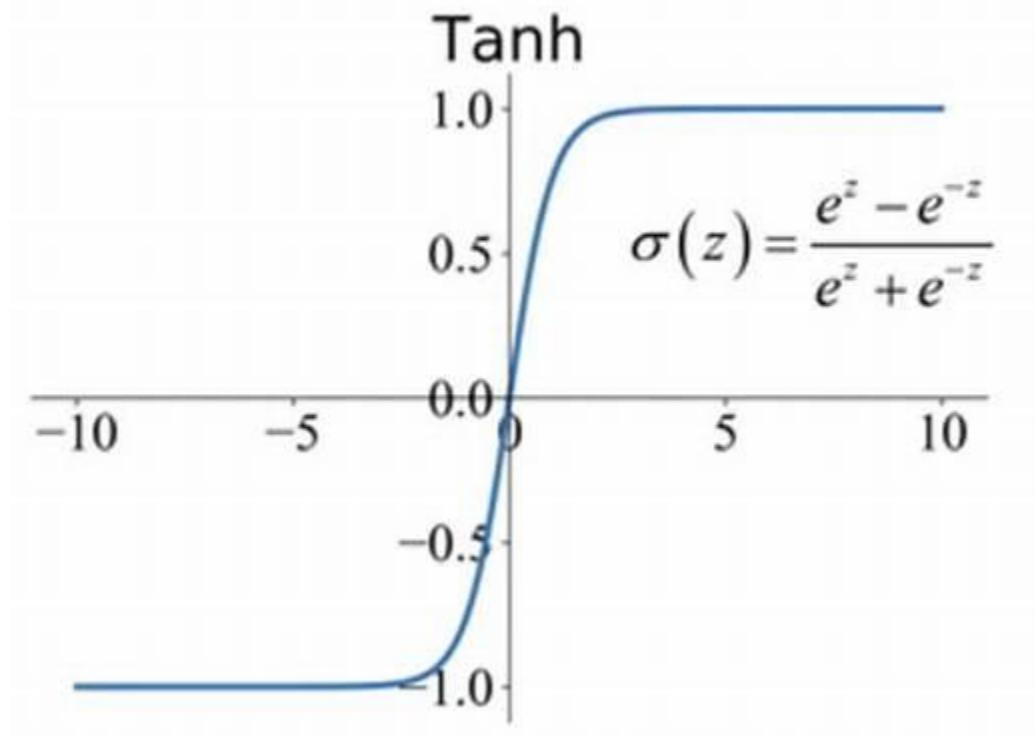


Sigmoid (đầu ra từ 0 -> 1)



⇒ Thường dùng cho bài toán có nhãn là 0 hoặc 1 , nếu giá trị từ 0 đến 0.5 thì nhãn là 0 còn nếu từ 0.5 đến 1 thì nhãn là 1 . Và đặt biệt các giá trị đầu ra là luôn dương , ta thấy nó có giá trị từ 0 đến 1 .

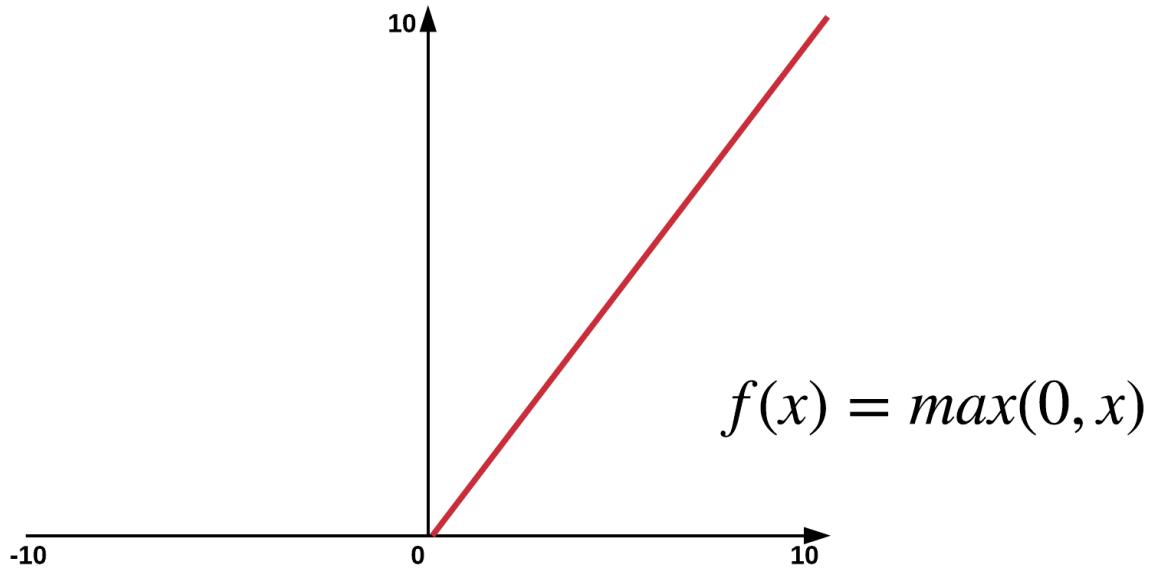
Tanh (-1 -> 1)



- ⇒ Đặt thù của giá trị đầu ra là có thể âm , ta thấy nó có giá trị từ -1 đến 1 .
Hàm kích hoạt này cũng có thể dùng cho bài toán 2 nhãn là 0 hoặc 1 , nếu từ -1 đến 0 thì
nhãn là 0 , còn từ 0 đến 1 thì nhãn là 1 .

Relu

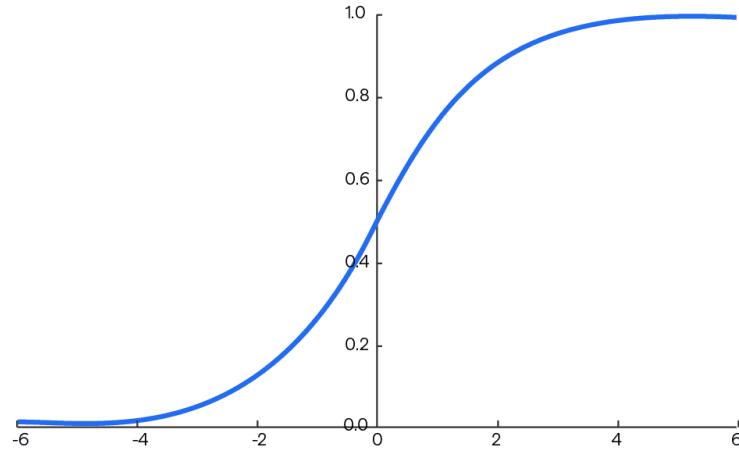
ReLU Activation Function



- ⇒ **Đặt thù của dữ liệu đầu ra là luồng dương , các giá trị âm sau khi qua ReLU đều cho bằng 0 và các giá trị dương thì giữ nguyên , miền giá trị chạy từ 0 đến dương vô cùng .**

Softmax

Softmax Function



- ⇒ **Đặt thù của dữ liệu là luôn dương và có miền giá trị từ 0 đến 1 . Hàm kích hoạt softmax thường được sử dụng trong các bài toán nhận dạng nhiều nhãn , tổng % các nhãn là 100% , nhãn nào có % cao nhất thì nhãn dự đoán được chọn chính là nhãn đó .**

Loss function – Linear Regression

MSE

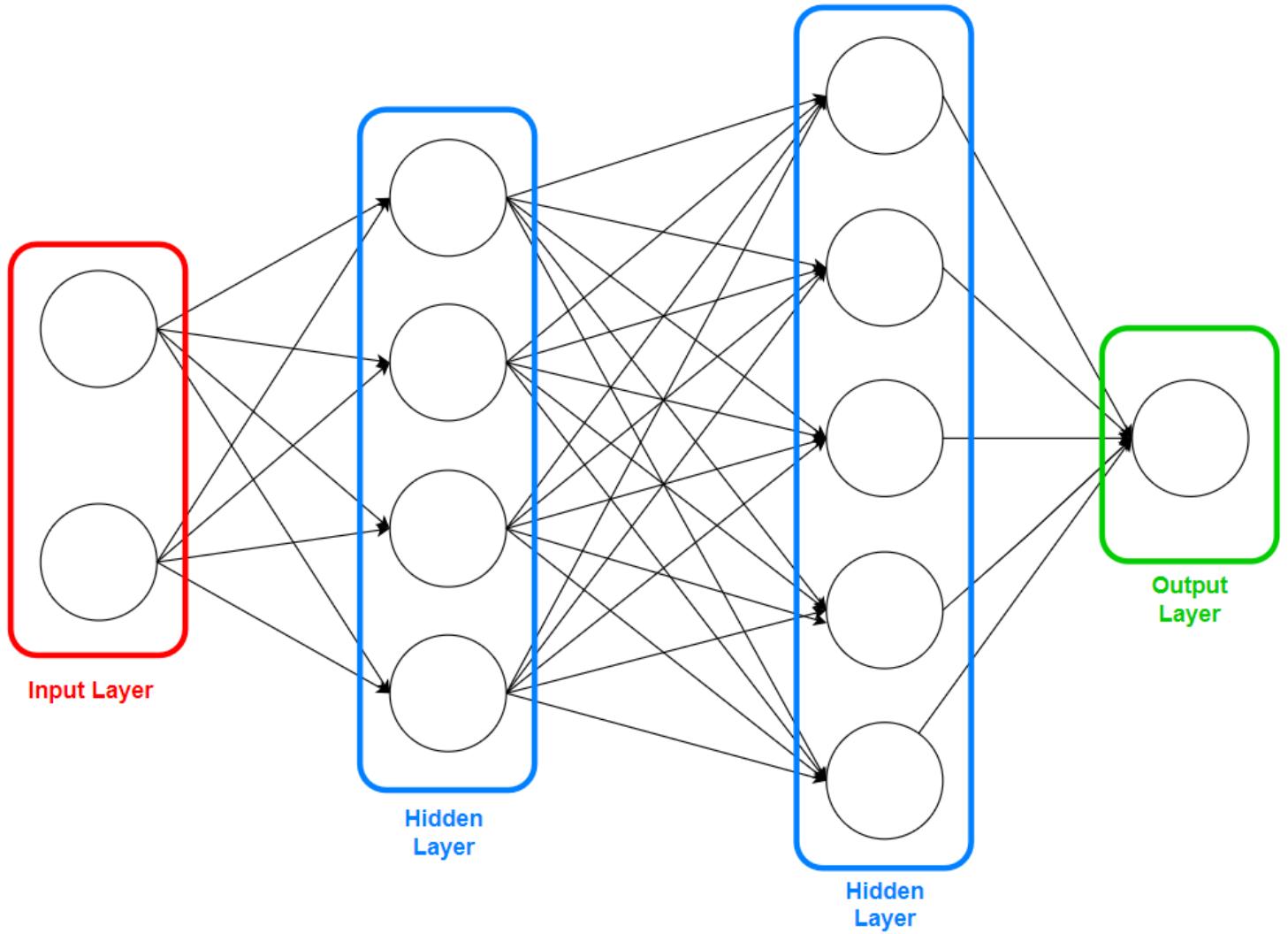
MAE

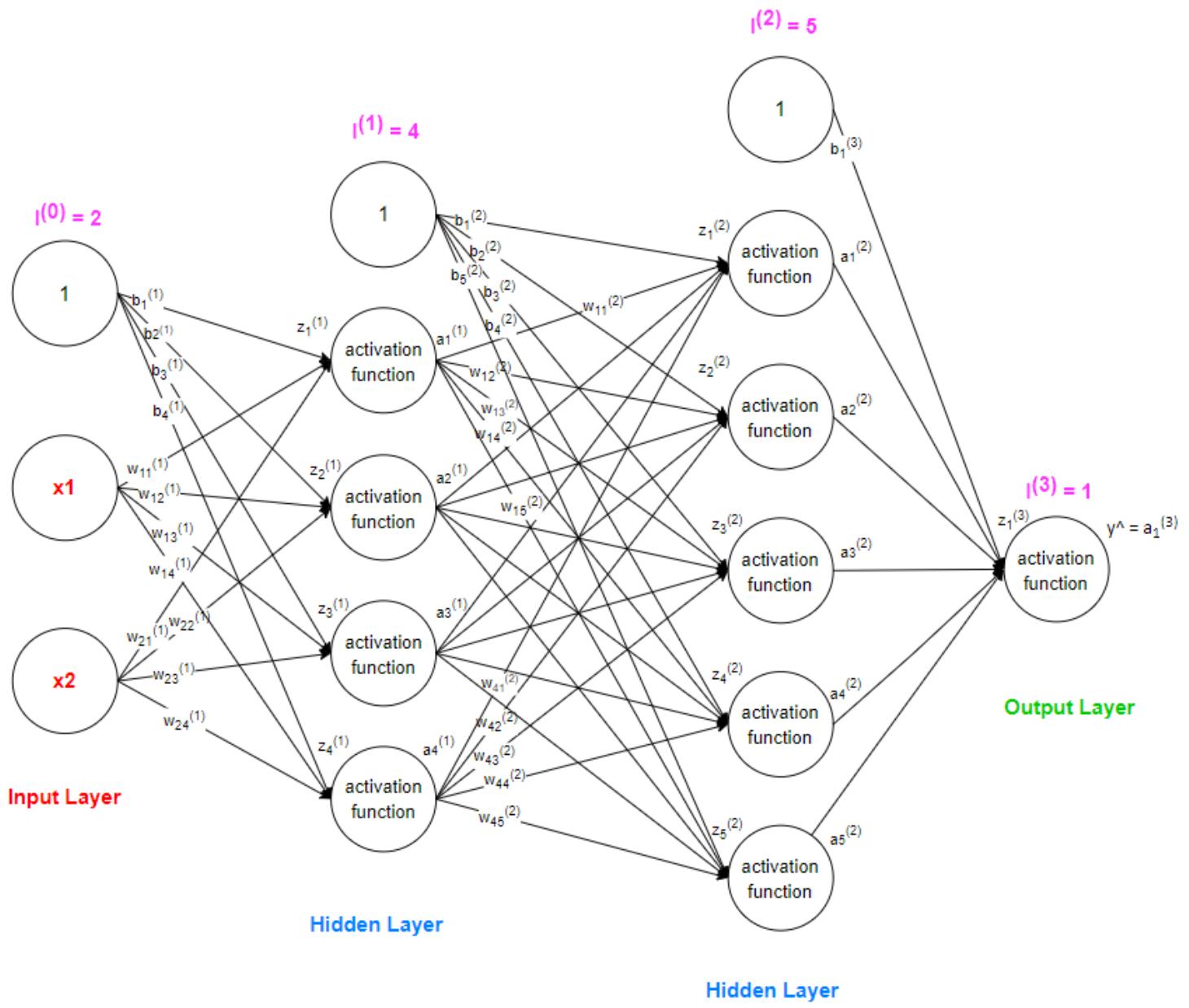
Loss function – Logistic Regression

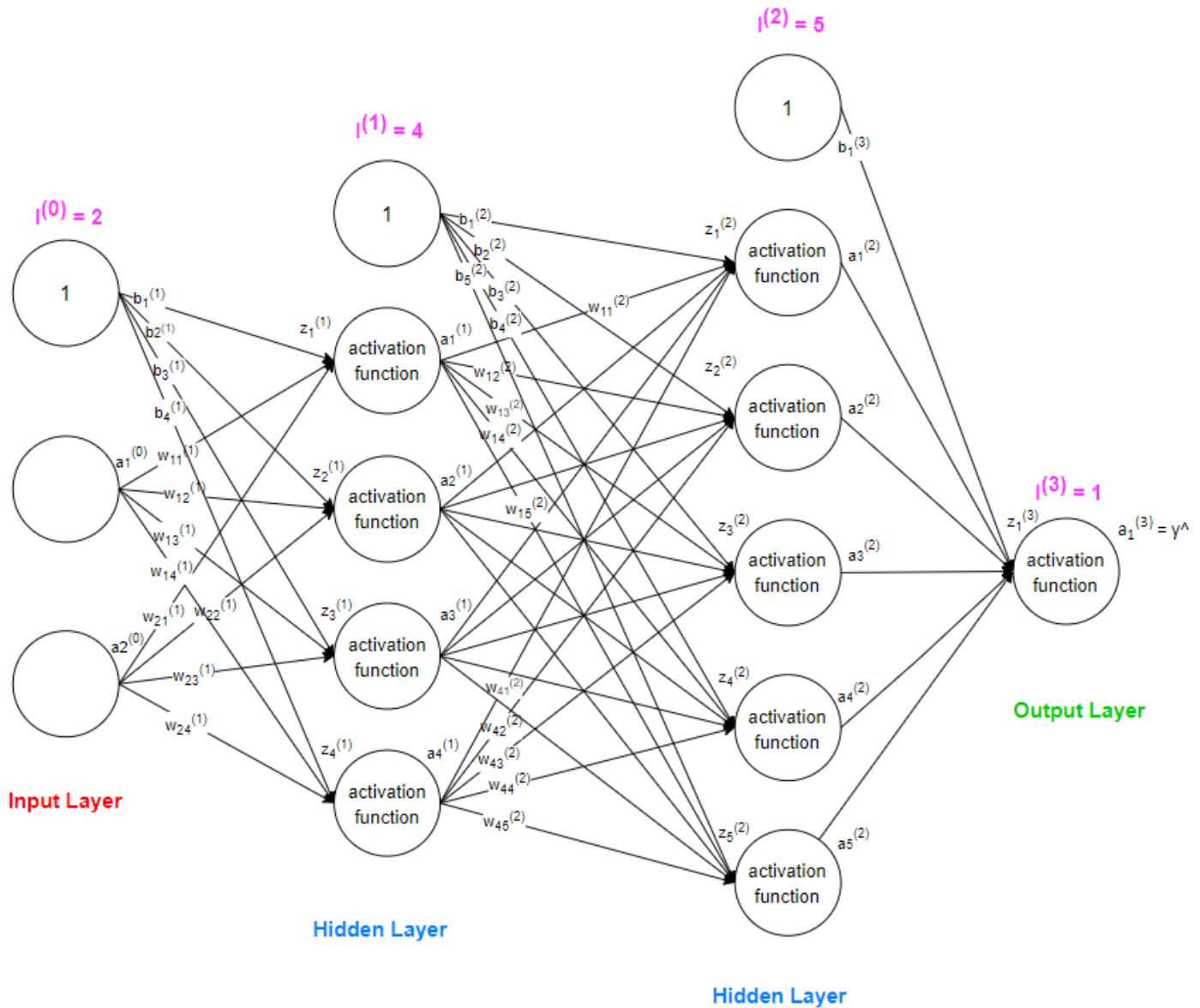
Log function

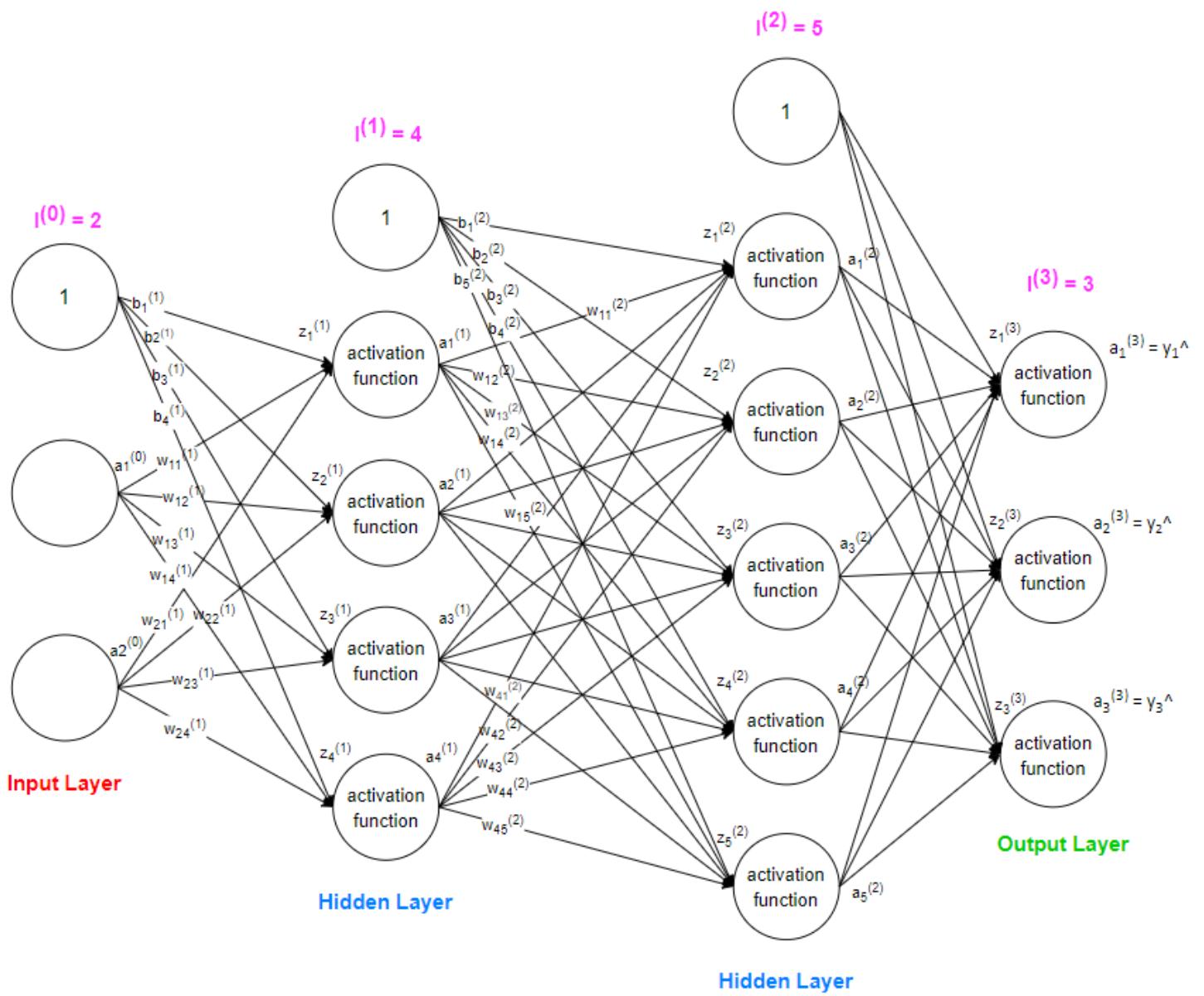
Neural Network

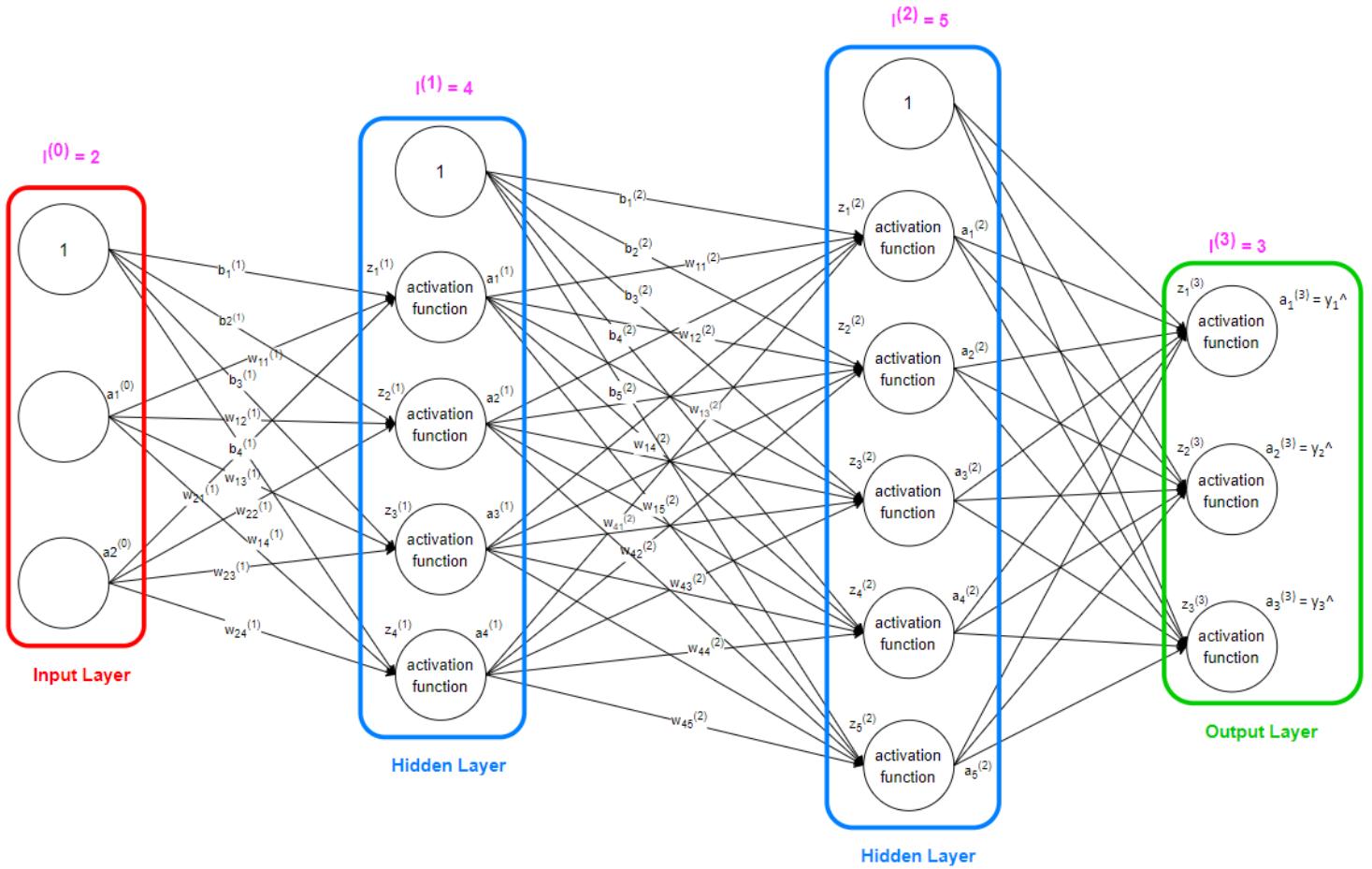
Quá trình hình thành Neural network đầy đủ

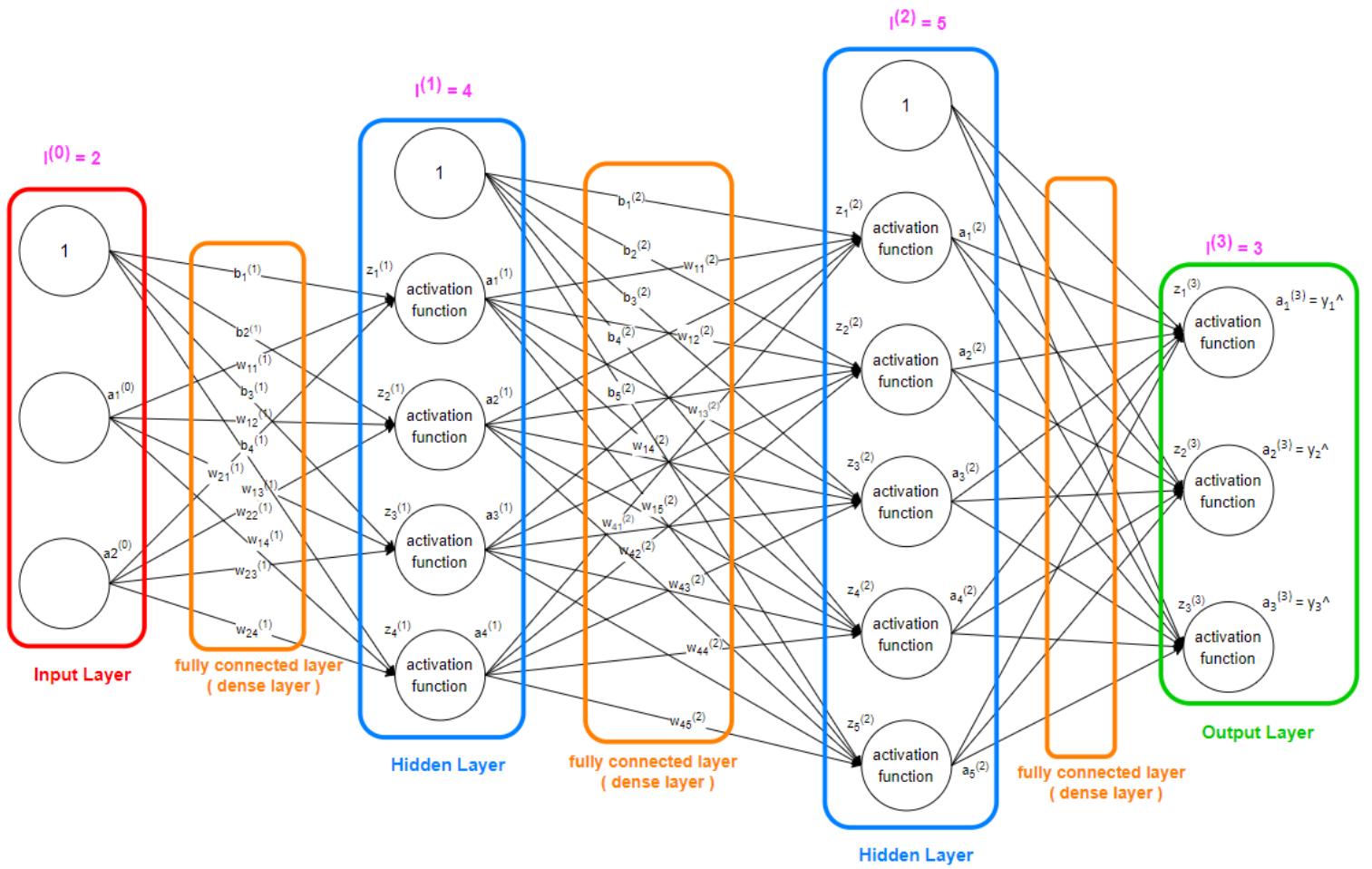


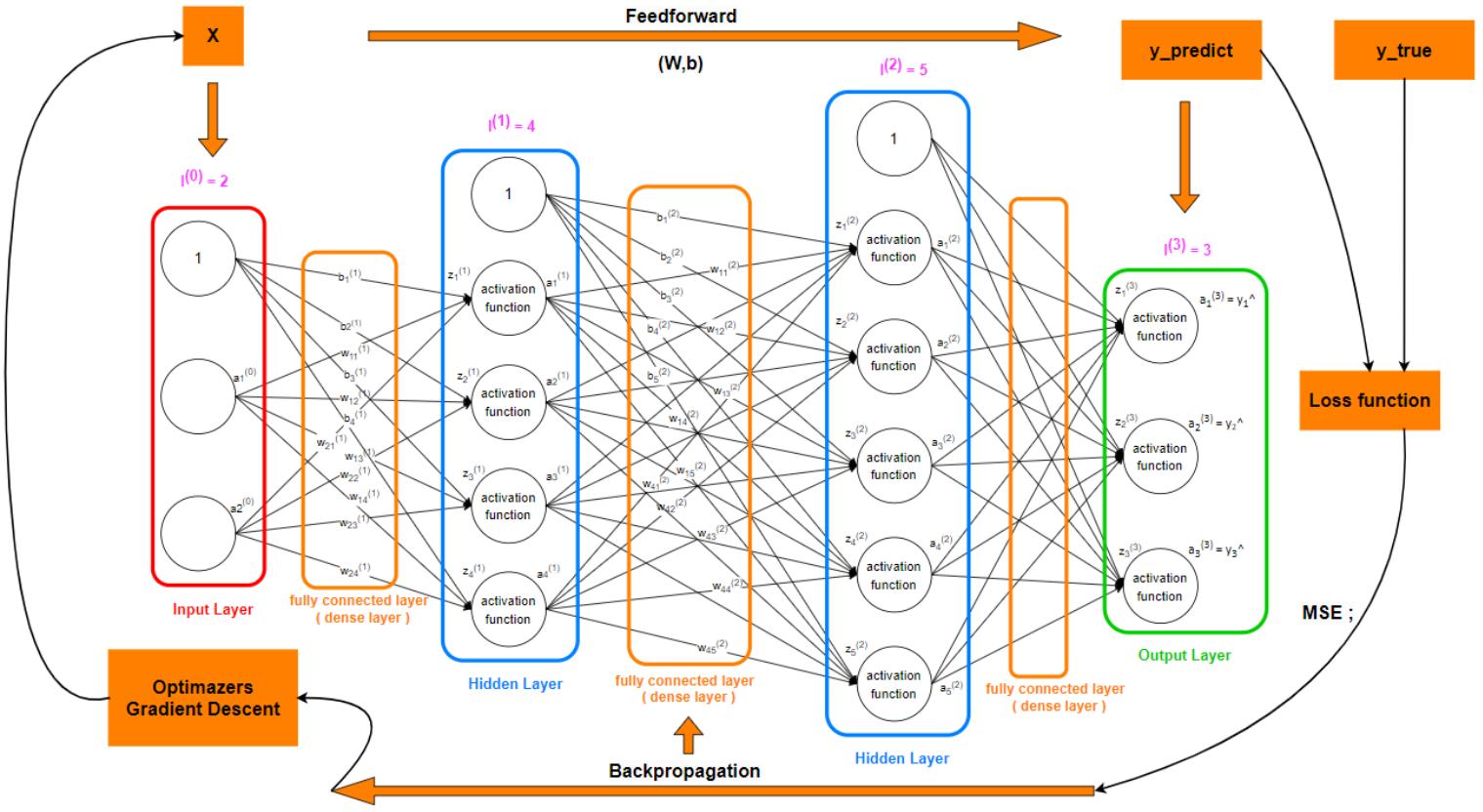


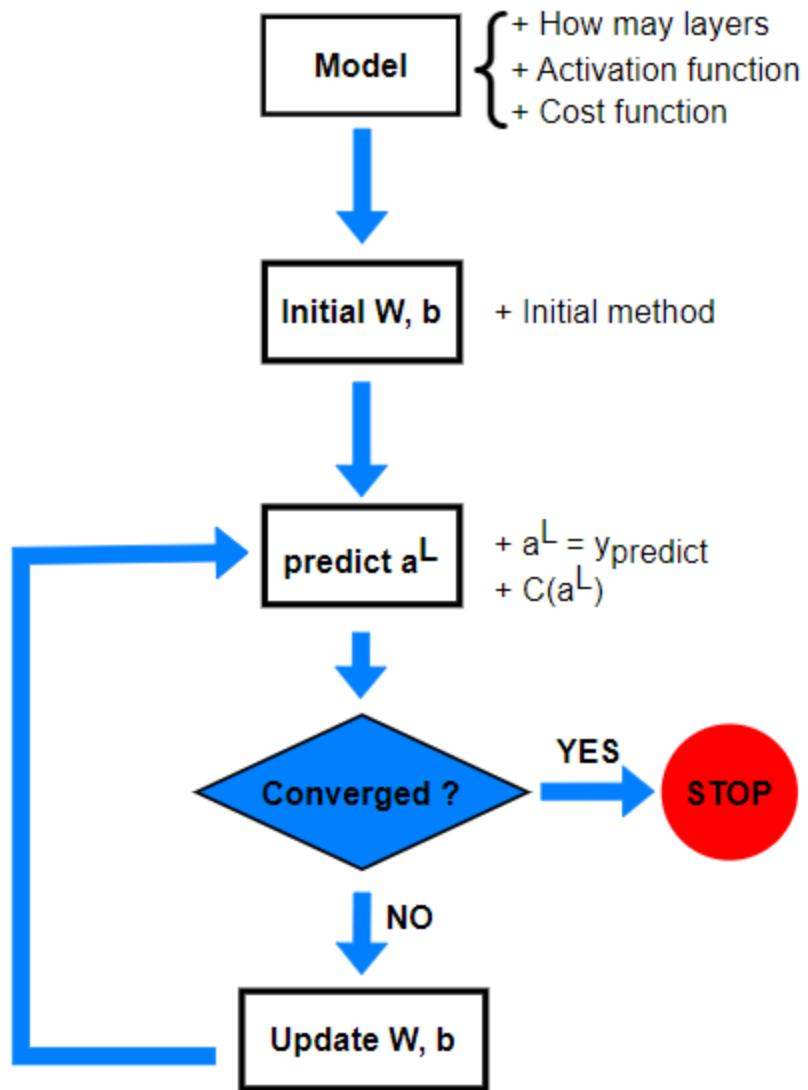


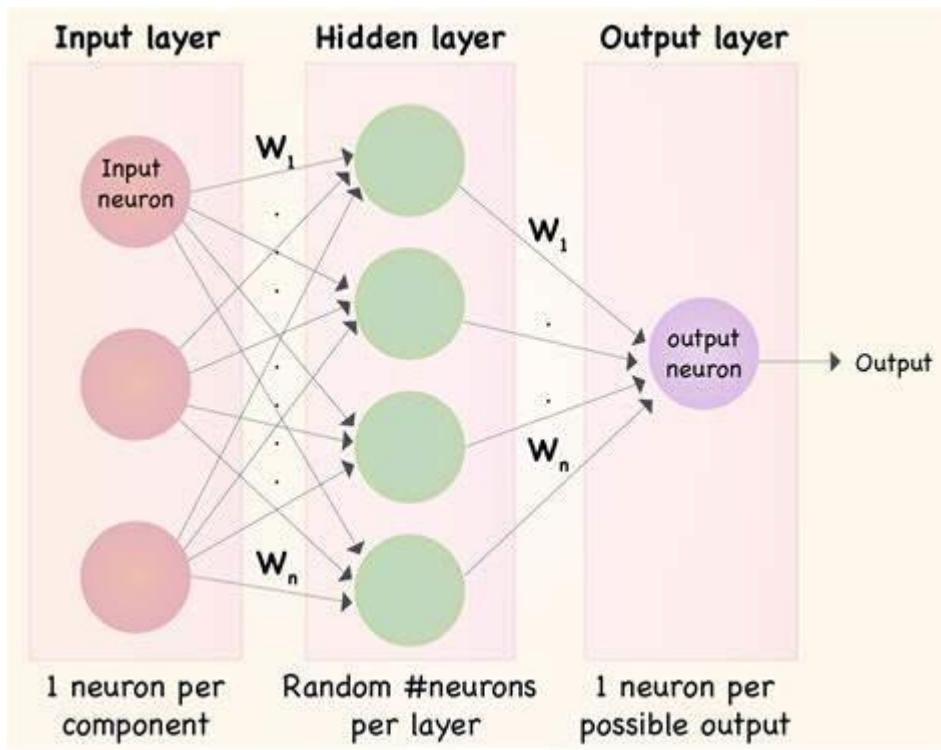












- ❖ Các tầng ở giữa (w_1, \dots, w_n) gọi là tầng kết nối
- ❖ Input neuron (X_1, X_2, \dots, X_n) gọi là các đặc trưng
- ❖ Lan truyền tiên dựa vào các activation
- ❖ Lan truyền ngược để cập nhật lại các trọng số

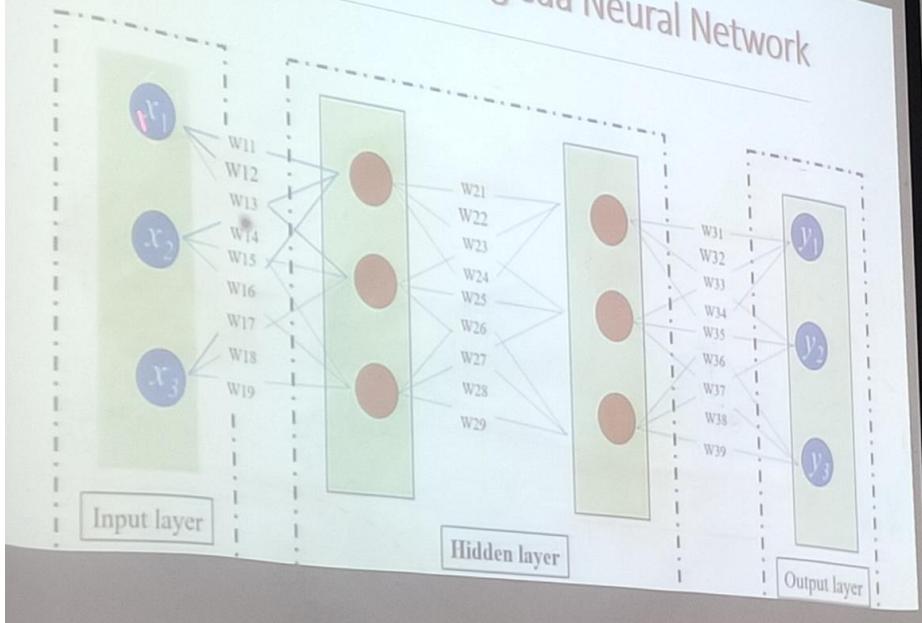
CNN , RNN => đều bắt nguồn và có core từ Mạnh nơ ron này

Feed forward algorithm

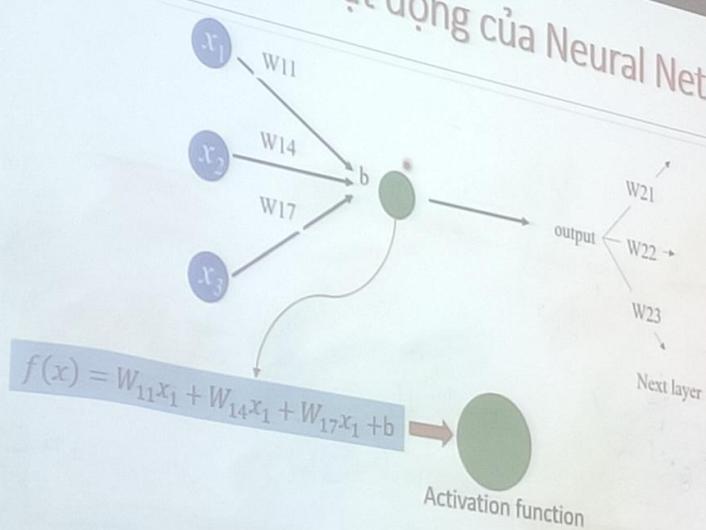
X_1, \dots, X_n là các đặc trưng đầu vào

y_1, \dots, y_n là các giá trị đầu ra

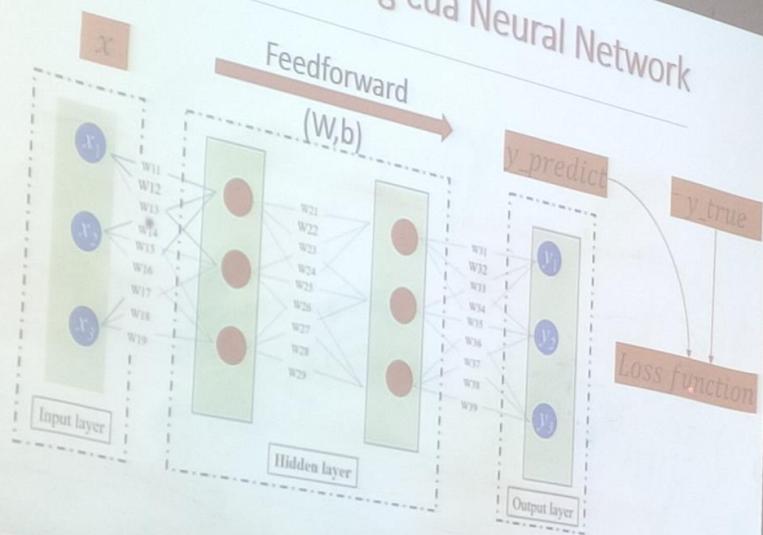
1.4 Nguyên tắc hoạt động của Neural Network



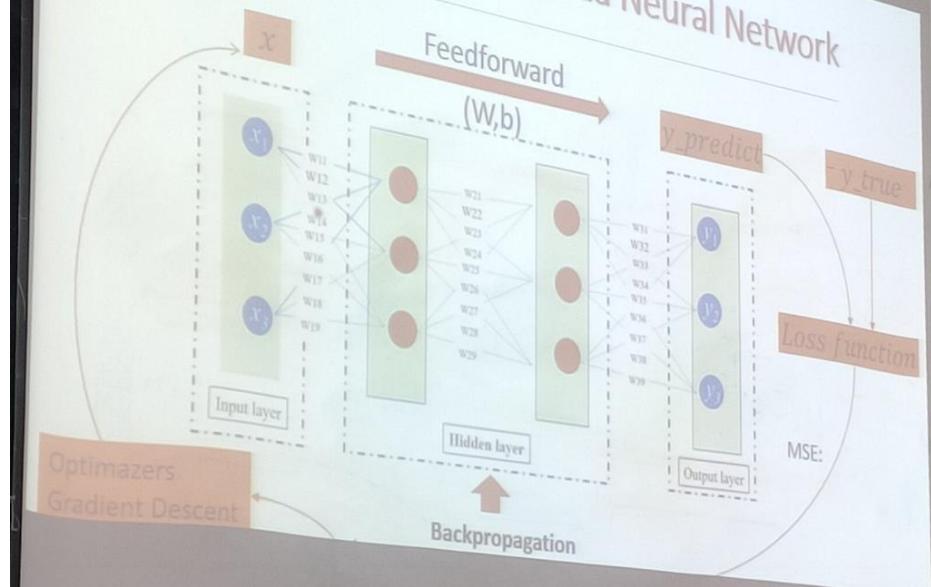
1.4 Nguyên tắc hoạt động của Neural Network



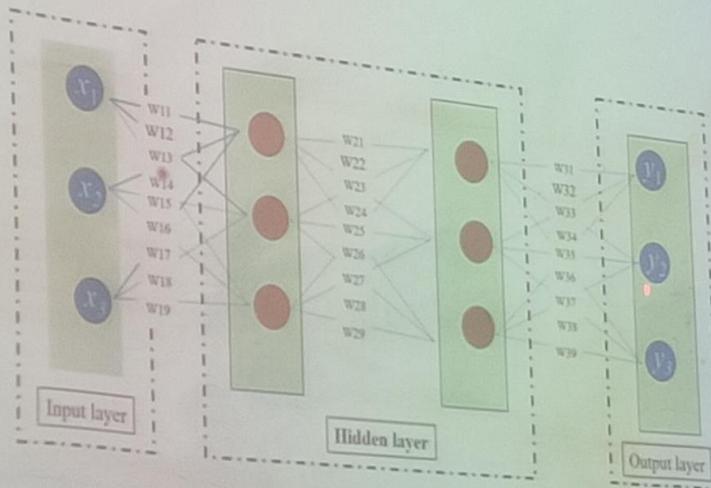
1.4 Nguyên tắc hoạt động của Neural Network



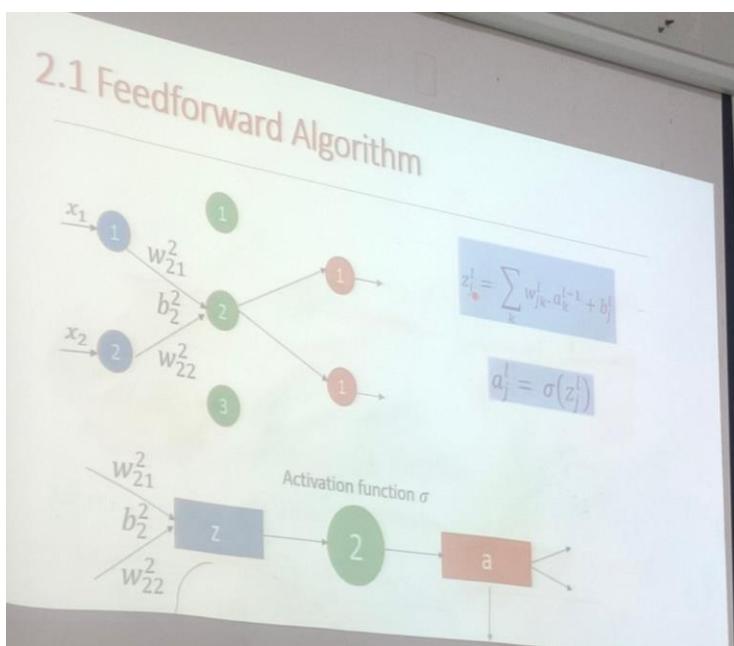
1.4 Nguyên tắc hoạt động của Neural Network



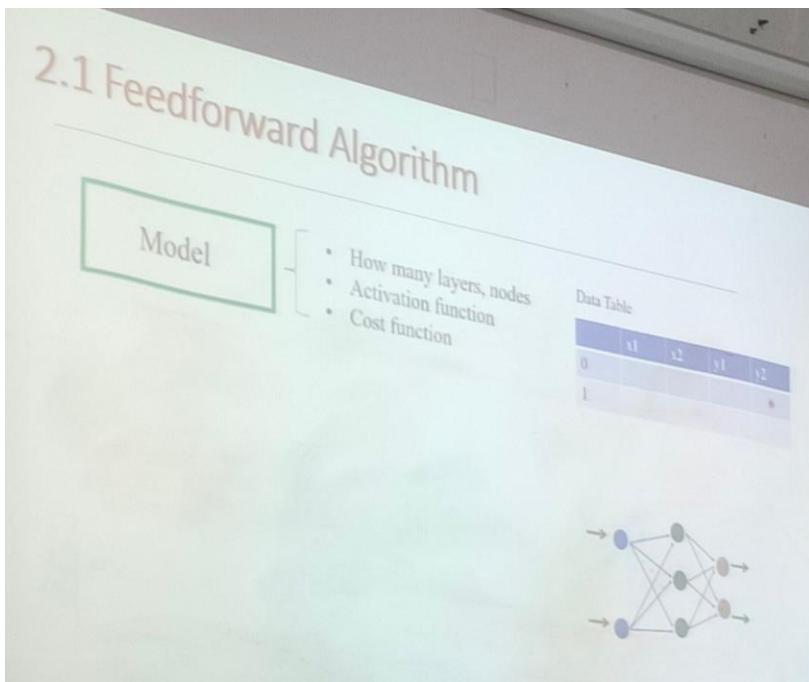
1.4 Nguyên tắc hoạt động của Neural Network



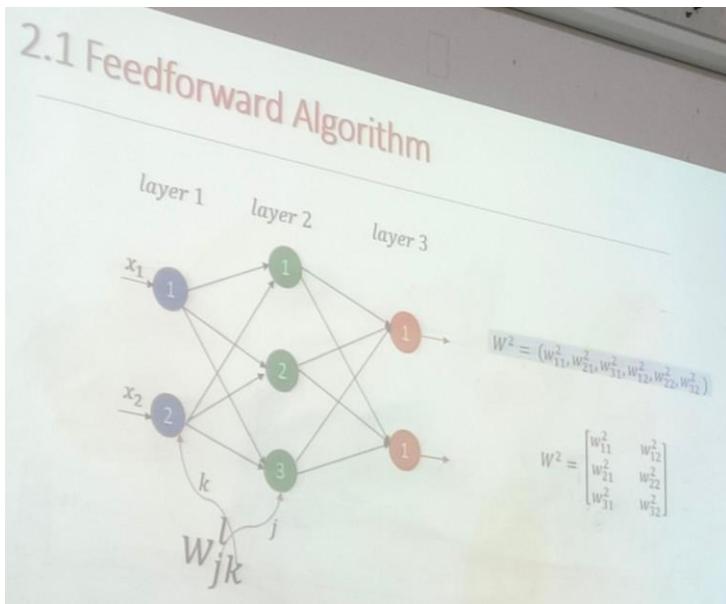
2.1 Feedforward Algorithm



- ❖ NOTE : Ở CÔNG THỨC TÍNH Z LÀ W.X (SỬA A LẠI THÀNH X)
- ❖ CÔNG THỨC K SAI : VVV
- ❖ VÌ TẦNG ĐẦU TIÊN CÓ X , CÒN TẦNG THỨ 2 trở đi thì X chính là A đã được tính từ tầng trước đó.
- ❖ Có thể khởi tạo Ở TẦNG 1 là các giá trị a bằng các giá trị X .



dot([n,2],[2,3]) , n là số mẫu dữ liệu đầu vào .



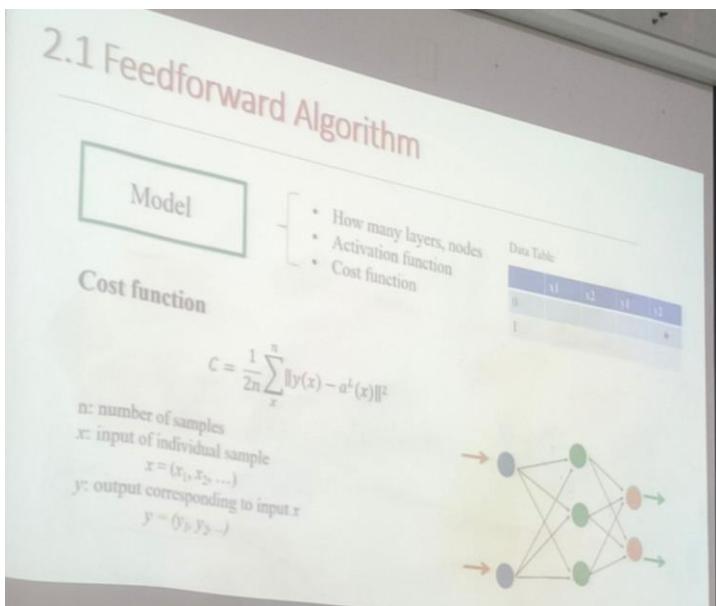
❖ Ví dụ ntn : lớp thứ 2 w là ma trận 3×2 .

Activation function

❖ Tất cả các tầng phải cùng một hàm kích hoạt (ví dụ dùng sigmoid thì dùng sigmoid hết) .

Loss function . Cross function

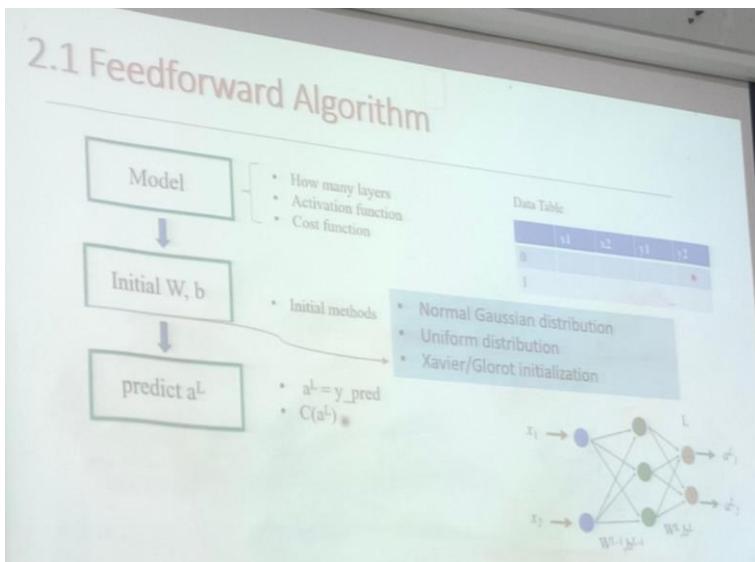
❖ Hàm mất mát .



- ❖ a^L chính là giá trị dự đoán . (nó vừa là giá trị đầu vào của lớp sau và là giá trị đầu ra của lớp trước)
- ❖ **n là số mẫu dữ liệu**
- ❖ **y là giá trị đầu ra cuối cùng**

Khởi tạo W, b

- Biểu đồ phân bố Gaussian .

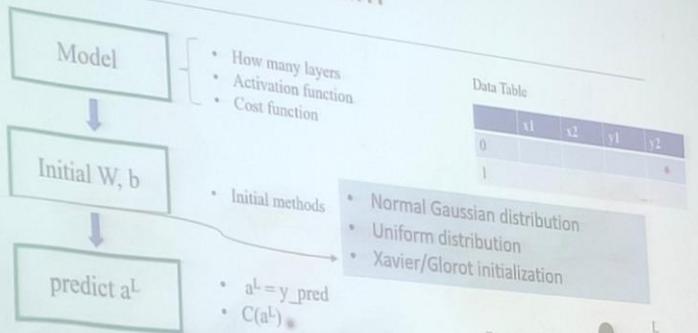


Predict a^L

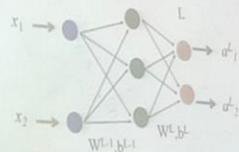
$$a^L = y_{pred}$$

$$C(a^L)$$

2.1 Feedforward Algorithm



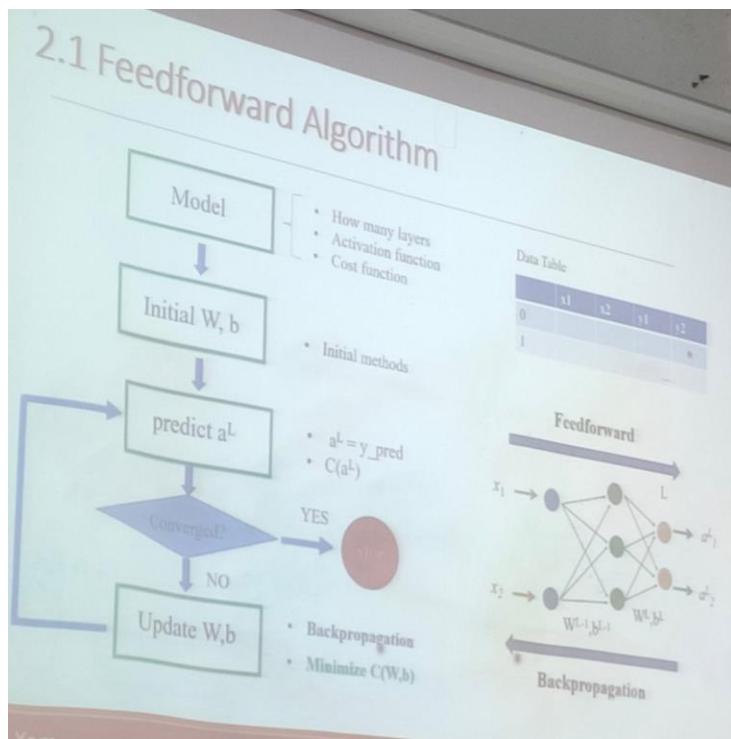
| | x1 | x2 | y1 | y2 |
|---|----|----|----|----|
| 0 | | | | |
| 1 | | | | * |



$$z_j^l = \sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l \quad a_j^l = \sigma(z_j^l)$$

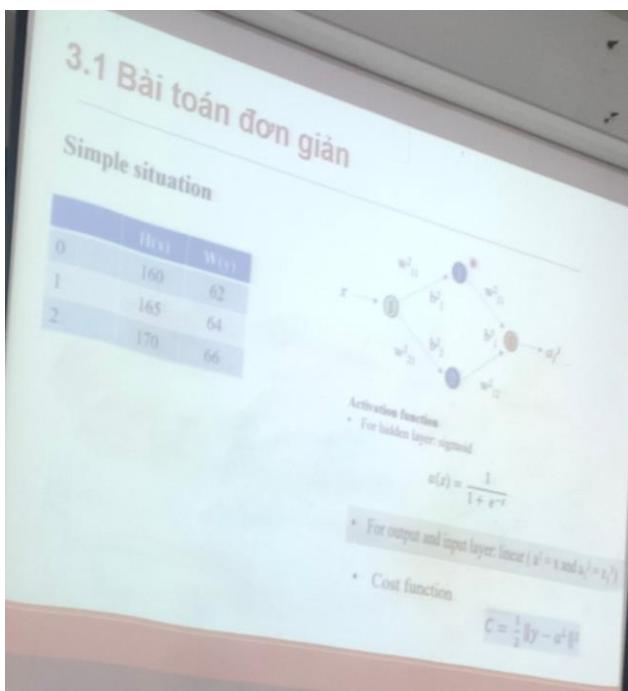
Feedforward W and b are available and constant, we computed a^L and C

Xem



Backpropagation algorithm

- CNN Pure Implement



Vì $f(x) = x$

❖ $\sigma(z) = a$

❖ $z = a$ (do $f(x) = x$)

3.1 Bài toán đơn giản

Simple situation

| | H(x) | W(y) |
|---|------|------|
| 0 | 160 | 62 |
| 1 | 165 | 64 |
| 2 | 170 | 66 |

Activation function

- For hidden layer: sigmoid

$$a(z) = \frac{1}{1 + e^{-z}}$$

- For output and input layer: linear ($a^1 = x$ and $a^L = z_i$)

Cost function

$$C = \frac{1}{2} \|y - a^L\|^2$$

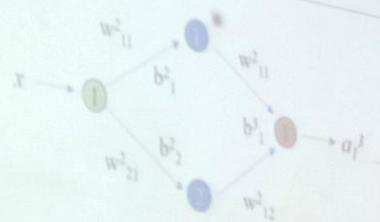
- Trong w, b thì b chính là bias
- $2 w$ sau là $w^3..$, trong slide $w^2..$ là nhầm . \Rightarrow sửa lại .

3.1 Bài toán đơn giản

Simple situation

| | $H(x)$ | $W(y)$ |
|---|--------|--------|
| 0 | 160 | 62 |
| 1 | 165 | 64 |
| 2 | 170 | 66 |

| w^1_{11} | w^1_{21} | b^1_1 | b^1_2 | w^2_{11} | w^2_{12} | b^2_1 |
|------------|------------|---------|---------|------------|------------|---------|
| 0.5 | 1 | 0 | 0 | 1 | 0.5 | 1 |



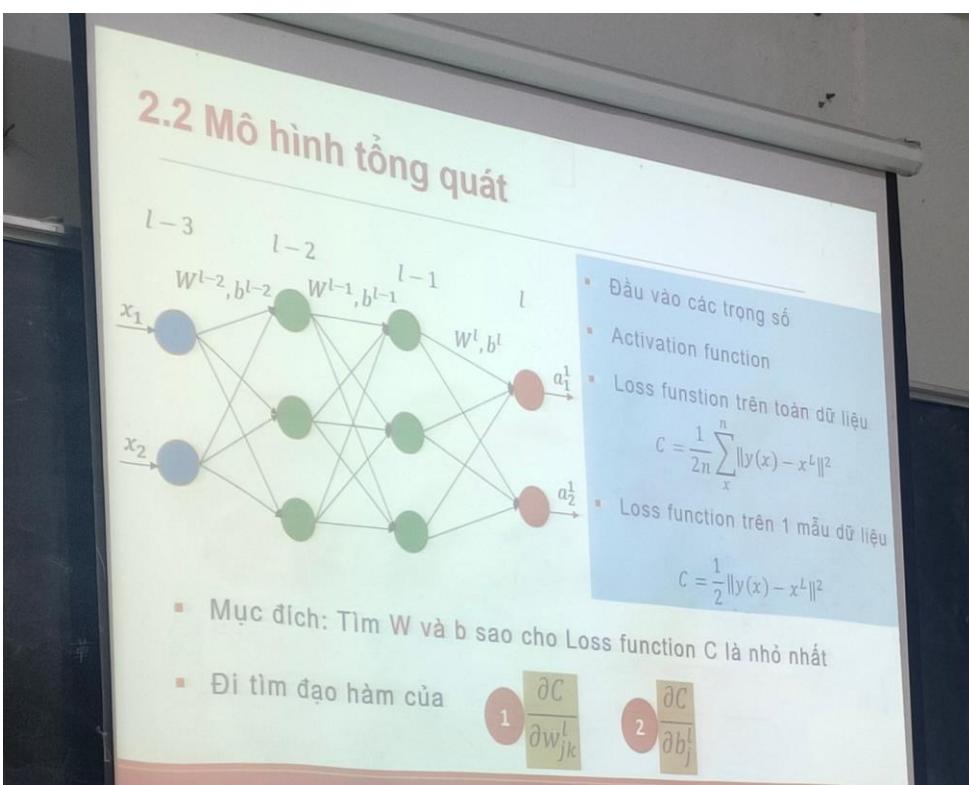
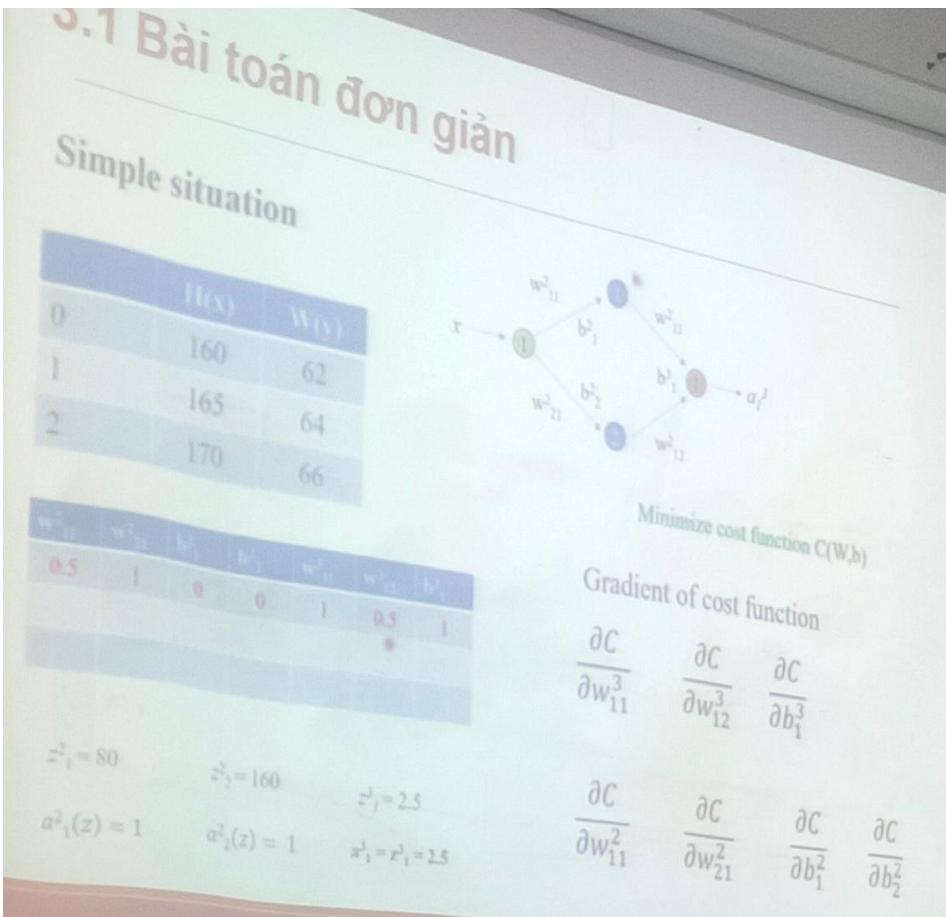
$$z^2_1 = w^2_{11} * x + b^2_1 = 0.5 * 160 + 0 = 80$$

$$z^2_2 = w^2_{21} * x + b^2_2 = 1 * 160 + 0 = 160$$

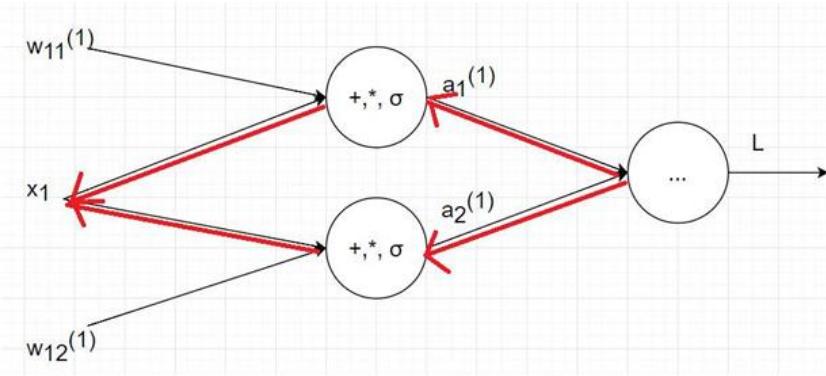
$$a^2_1 = \sigma(z^2_1) = \frac{1}{1 + e^{-80}} \approx 1$$

$$a^2_2 = \sigma(z^2_2) = \frac{1}{1 + e^{-160}} \approx 1$$

$$w = w - \text{learning_rate} * \text{gradient}$$



- ❖ Tìm W, b sao cho C nhỏ nhất , càng về 0 càng tốt



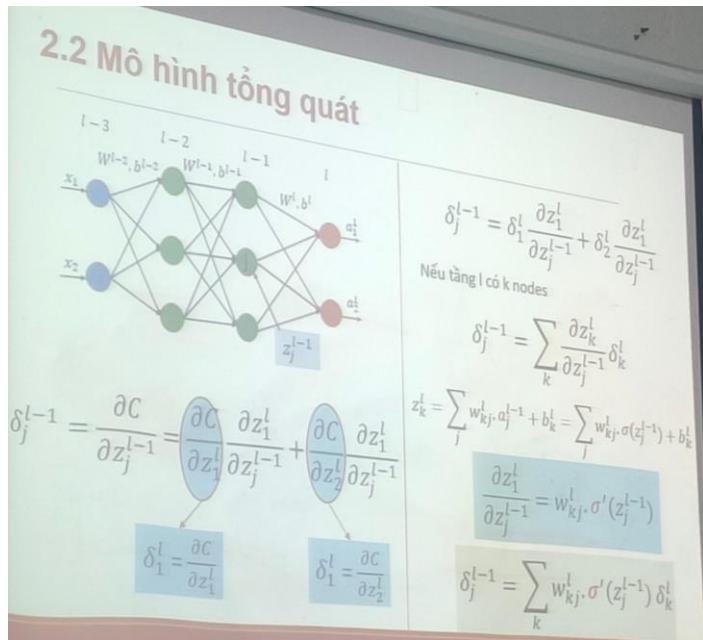
Do đó:

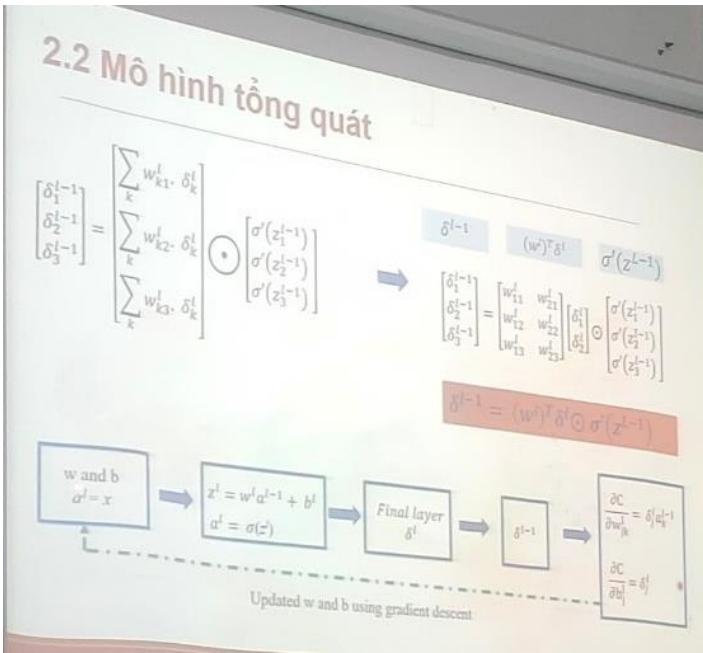
6.2 Mô hình tổng quát

103

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial a_1^{(1)}} * \frac{\partial a_1^{(1)}}{\partial x_1} + \frac{\partial L}{\partial a_2^{(1)}} * \frac{\partial a_2^{(1)}}{\partial x_1} = w_{11}^{(1)} * a_1^{(1)} * (1 - a_1^{(1)}) * w_{11}^{(2)} * (y_i - \hat{y}_i) + w_{12}^{(1)} * a_2^{(1)} * (1 - a_2^{(1)}) * w_{21}^{(2)} * (y_i - \hat{y}_i)$$

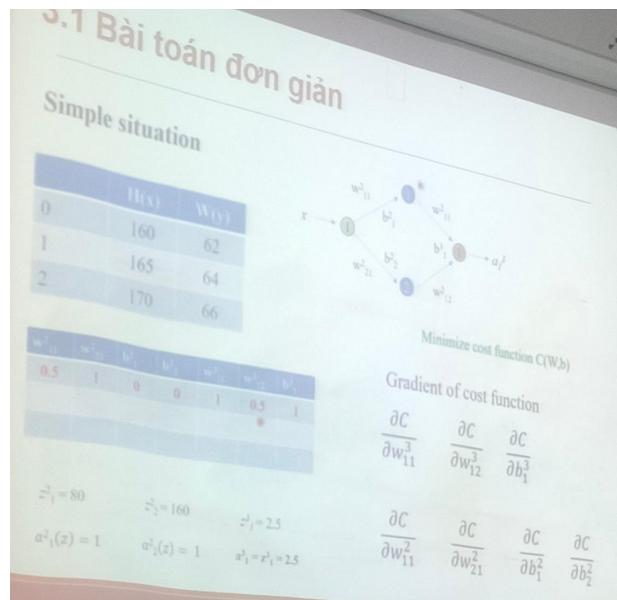
$\frac{\partial(L)}{\partial(x_1)}$ ở đây chính là $L-1$ (lớp trước)





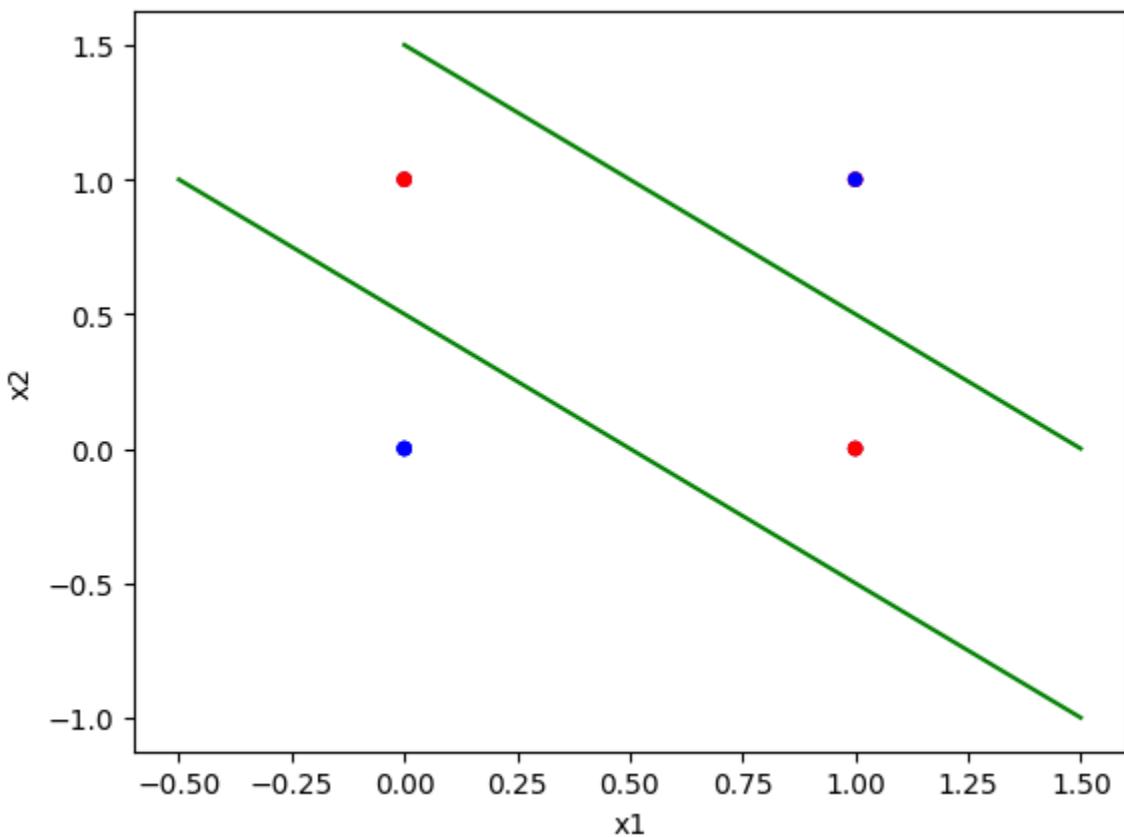
BTVN :

Tính tay :



- Viết RNN theo hướng đối tượng (Layer, FullyConnected)
- Sử dụng lại data của bài logistic => cho mạng nơ ron => nhận dạng
- ❖ So sánh độ chính xác

Logistic regression với toán tử XOR



❖ Xem chi tiết ở sách.

[HAY] Tóm tắt Neural Network

Tóm tắt quá trình hoạt động của mạng neural network

1. **Dữ liệu đầu vào:** Dữ liệu đầu vào là các cặp giá trị (Learn, Sleep) và nhãn tương ứng (Is_Pass). Mỗi mẫu dữ liệu được biểu diễn bởi hai đặc trưng: Learn và Sleep. Mục tiêu là dự đoán nhãn Is_Pass dựa trên các đặc trưng này.
2. **Layer input:** Layer input có 2 node tương ứng với 2 đặc trưng của dữ liệu đầu vào: Learn và Sleep.
3. **Layer ẩn:** Trong một mạng neural network đơn giản, có thể sử dụng một hoặc nhiều layer ẩn. Trong trường hợp này, chúng ta sẽ sử dụng một layer ẩn. Số node trong layer ẩn là một siêu tham số mà bạn có thể điều chỉnh để tối ưu hiệu suất mạng. Trong ví dụ này, chúng ta có thể chọn 4 node cho layer ẩn.
4. **Layer đầu ra:** Layer đầu ra có 1 node, tương ứng với dự đoán Is_Pass.
5. **Quá trình Feedforward:** Mỗi mẫu dữ liệu được đưa vào mạng neural network. Mỗi node trong layer input là một đầu vào cho mạng. Các trọng số và hệ số bias được áp dụng và kết hợp để tạo ra giá trị tại các node trong layer ẩn. Giá trị đầu ra từ layer ẩn được truyền qua hàm kích hoạt (ví dụ: sigmoid) để tạo ra giá trị tại các node trong layer đầu ra. Giá trị dự đoán từ layer đầu ra được sử dụng để tính toán hàm mất mát (ví dụ: MSE) với nhãn thực tế.
6. **Quá trình Backpropagation:** Hàm mất mát được sử dụng để tính toán độ lỗi của mạng.

Độ lỗi được lan truyền ngược từ layer đầu ra trở lại layer ẩn, dựa trên đạo hàm của hàm kích hoạt và các trọng số của mạng. Các trọng số và hệ số bias được cập nhật bằng cách sử dụng gradient descent để giảm thiểu độ lỗi.

Quá trình này được lặp lại cho đến khi độ lỗi giảm đến mức chấp nhận được hoặc đạt đến số lần lặp được chỉ định (epochs).

7. **Lặp qua dữ liệu:** Mỗi lần lặp lại (epoch), mỗi mẫu dữ liệu trong tập huấn luyện được sử dụng để cập nhật trọng số của mạng.
 - ❖ Quá trình này tiếp tục cho đến khi mạng hội tụ hoặc cho đến khi đạt đến số lần lặp được chỉ định. Quá trình này được lặp lại cho đến khi mạng hội tụ hoặc cho đến khi đạt đến số lần lặp được chỉ định. Sau khi mạng đã được huấn luyện, bạn có thể sử dụng nó để dự đoán nhãn cho các dữ liệu mới.

Kích thước các ma trận liên quan, ví dụ gồm các lớp :

- Input layer : 2 node
 - Hidden layer 1 : 3 node
 - Hidden layer 2 : 4 node
 - Output layer : 1 node
1. Layer Input:
 - o Ma trận trọng số $W^{(1)}$ có kích thước $2 * 3$ (2 node input và 2 node ẩn)
 - o Hệ số bias: $b^{(1)}$ có kích thước $1 * 3$
 - o $Z^{(1)}$ có kích thước $n * 3$ (với n là số mẫu trong tập dữ liệu)
 - o $A^{(1)}$ có kích thước $n * 3$
 2. Layer ẩn:
 - o Ma trận trọng số $W^{(2)}$ có kích thước $3 * 4$ (3 node ẩn và 4 node ẩn tiếp theo)
 - o Hệ số bias: $b^{(2)}$ có kích thước $1 * 4$
 - o $Z^{(2)}$ có kích thước $n * 4$
 - o $A^{(2)}$ có kích thước $n * 4$
 3. Layer Output:
 - o Ma trận trọng số $W^{(3)}$ có kích thước $4 * 1$ (4 node ẩn và 1 node output)
 - o Hệ số bias: $b^{(3)}$ có kích thước $1 * 4$
 - o $Z^{(3)}$ có kích thước $n * 1$
 - o $A^{(3)}$ có kích thước $n * 1$
- ❖ Trong đó n là số mẫu dữ liệu có trong tập dữ liệu

Quá trình hoạt động của neural network bằng đại số tuyến tính . Với số lớp trên .

1. Feedforward:
 - o Tính cho layer ẩn thứ 1:
 - $Z^{(1)} = X \cdot W^{(1)} + b^{(1)}$
 - $A^{(1)} = \sigma(Z^{(1)})$
 - o Tương tự, tính $Z^{(2)}$ và $A^{(2)}$ cho layer ẩn thứ 2:
 - $Z^{(2)} = A^{(1)} \cdot W^{(2)} + b^{(2)}$
 - $A^{(2)} = \sigma(Z^{(2)})$
 - o Cuối cùng, tính $Z^{(3)}$ và $A^{(3)}$ cho layer output:
 - $Z^{(3)} = A^{(2)} \cdot W^{(3)} + b^{(3)}$
 - $A^{(3)} = \sigma(Z^{(3)})$
2. Backpropagation
 - o Tính độ lỗi của layer output: $\delta^{(3)}$:
 - $\delta^{(3)} = A^{(3)} - Y$

- Tính độ lỗi của layer ẩn thứ 2: $\delta^{(2)}$:
 - $\delta^{(2)} = (\delta^{(3)} * (W^{(3)})^T) * \sigma'(Z^{(2)})$
- Tính độ lỗi của layer ẩn thứ 1: $\delta^{(1)}$:
 - $\delta^{(1)} = (\delta^{(2)} * (W^{(2)})^T) * \sigma'(Z^{(1)})$
- Cập nhật trọng số và bias:
 - Cập nhật trọng số và bias cho layer output:
 - $W^{(3)} = W^{(3)} - \alpha * (A^{(2)})^T * \delta^{(3)}$
 - $b^{(3)} = b^{(3)} - \alpha * \sum \delta^{(3)}$
 - Cập nhật trọng số và bias cho layer ẩn thứ 2 :
 - $W^{(2)} = W^{(2)} - \alpha * (A^{(1)})^T * \delta^{(2)}$
 - $b^{(2)} = b^{(2)} - \alpha * \sum \delta^{(2)}$
 - Cuối cùng, cập nhật trọng số và bias cho layer ẩn thứ 1 :
 - $W^{(1)} = W^{(1)} - \alpha * (X)^T * \delta^{(1)}$
 - $b^{(1)} = b^{(1)} - \alpha * \sum \delta^{(1)}$
- ❖ Trong đó
 - α : learning rate (tốc độ học)
 - σ : sigmoid activation function
 - σ' : đạo hàm hàm sigmoid
 - X là ma trận đầu vào , Y là ma trận nhãn thực tế
 - $\delta^{(i)}$: độ lỗi của layer thứ i

Nói thêm về Neural network

1. **Số lớp ẩn và số node của mỗi lớp ẩn trong một mạng neural network phụ thuộc vào nhiều yếu tố khác nhau, bao gồm:**
 - Phức tạp của bài toán: Độ phức tạp của bài toán có thể ảnh hưởng đến cần phải có bao nhiêu lớp ẩn và số node trong mỗi lớp. Các bài toán phức tạp thường cần nhiều lớp ẩn và nhiều node hơn để mạng có khả năng học được các mô hình phức tạp.
 - Số lượng và đa dạng của dữ liệu đầu vào: Số lượng và đa dạng của các đặc trưng đầu vào cũng có thể ảnh hưởng đến cấu trúc của mạng neural network. Dữ liệu đa dạng hoặc có nhiều đặc trưng có thể đòi hỏi một số lượng lớn hơn các node trong các lớp ẩn để mạng có thể học được các biểu diễn phức tạp của dữ liệu.
 - Kích thước của tập dữ liệu: Kích thước của tập dữ liệu đầu vào cũng có thể ảnh hưởng đến cấu trúc của mạng. Trong một số trường hợp, khi có ít dữ liệu, việc sử dụng một mạng quá phức tạp có thể dẫn đến overfitting. Ngược lại, khi có nhiều dữ liệu, có thể sử dụng các mạng lớn hơn mà không gặp vấn đề overfitting.
 - Tài nguyên tính toán: Số lượng node trong mỗi lớp ẩn cũng phụ thuộc vào tài nguyên tính toán có sẵn. Mạng lớn với nhiều node yêu cầu nhiều tài nguyên tính toán hơn.
 - Thủ nghiệm và điều chỉnh: Thường xuyên cần thử nghiệm với các kiến trúc mạng khác nhau và điều chỉnh số lượng lớp ẩn và số node trong mỗi lớp để tìm ra cấu trúc hiệu quả nhất cho bài toán cụ thể.
2. **Sử dụng Neural network**

Mạng neural network có thể áp dụng cho nhiều loại giá trị dự đoán khác nhau và chỉ cần thay đổi hàm kích hoạt (activation function) để phù hợp với loại giá trị y mà bạn muốn dự đoán.

- **Hàm kích hoạt tuyến tính (Linear Activation Function):** Thích hợp cho các bài toán dự đoán giá trị số, không giới hạn về phạm vi giá trị.

- **Hàm kích hoạt nhị phân (Sigmoid Activation Function):** Thích hợp cho các bài toán phân loại nhị phân, với đầu ra dự đoán gần 0 hoặc 1.
- **Hàm kích hoạt softmax (Softmax Activation Function):** Thích hợp cho các bài toán phân loại nhiều lớp (multiclass classification), với đầu ra dự đoán là xác suất của mỗi lớp và tổng các xác suất bằng 1.
- **Hàm kích hoạt ReLU (Rectified Linear Unit):** Thường được sử dụng cho các mạng neural network sâu (deep neural networks) với hiệu suất tốt, đặc biệt trong các mô hình học sâu (deep learning).
- ❖ Và còn nhiều hàm kích hoạt khác nữa mà bạn có thể sử dụng tùy thuộc vào bài toán cụ thể và đặc tính của dữ liệu. Điều này làm cho mạng neural network trở thành một công cụ mạnh mẽ và linh hoạt cho nhiều loại bài toán khác nhau trong học máy.

3. Số lượng node đầu ra trong mạng neural network

Số lượng node đầu ra trong mạng neural network phụ thuộc vào loại bài toán mà bạn đang giải quyết và yêu cầu cụ thể của tập dữ liệu. Dưới đây là một số ví dụ cụ thể:

1. **Phân loại nhị phân (Binary Classification):** Trong trường hợp này, chỉ có hai lớp được phân loại, ví dụ: "có bệnh" hoặc "không có bệnh", "spam" hoặc "không spam". Do đó, bạn chỉ cần một node ở output layer để đưa ra dự đoán nhị phân.
 2. **Phân loại đa lớp (Multi-Class Classification):** Khi bạn phải phân loại dữ liệu thành nhiều lớp hơn hai, ví dụ: phân loại chủ đề của email thành "thể thao", "chính trị", "kinh doanh", "giải trí", và "khoa học". Trong trường hợp này, số lượng node đầu ra sẽ bằng số lượng lớp cần phân loại.
 3. **Dự đoán giá trị số (Regression):** Trong bài toán dự đoán giá trị số, ví dụ: dự đoán giá nhà dựa trên diện tích và số phòng ngủ. Trong trường hợp này, bạn cũng chỉ cần một node ở output layer để dự đoán giá trị liên tục.
 4. **Một số bài toán đặc biệt khác:** Có những bài toán yêu cầu sử dụng nhiều node đầu ra hơn, như bài toán Object Detection trong Computer Vision, nơi mỗi node đại diện cho một loại đối tượng có thể xuất hiện trong ảnh.
- ❖ Tóm lại, số node đầu ra phụ thuộc vào yêu cầu cụ thể của bài toán và loại dữ liệu bạn đang làm việc.

4. Softmax activation function cho bài toán Phân loại nhiều lớp (Multi-Class Classification) (Nhiều nhãn)

- Trong phân loại đa lớp (Multi-Class Classification), hàm kích hoạt thích hợp cho lớp đầu ra là softmax. Hàm softmax thường được sử dụng để chuyển đổi các điểm số đầu ra thành các xác suất. Nó là một dạng nâng cao của hàm sigmoid, được thiết kế để xử lý các tác vụ phân loại có nhiều lớp.
- Hàm softmax được định nghĩa như sau:

$$\text{softmax}(\mathbf{z}_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

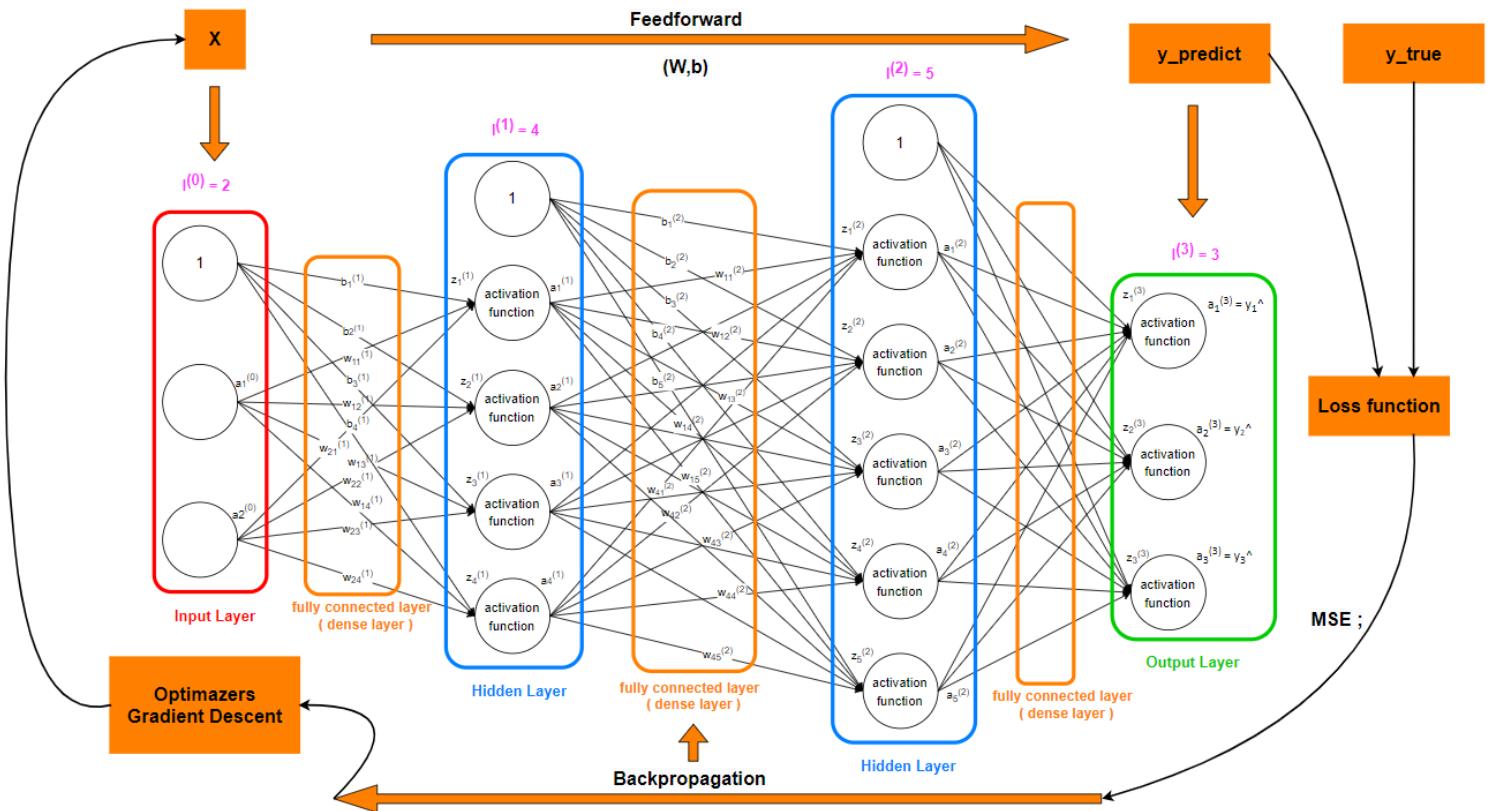
Trong đó :

- z_i là điểm số (score) của lớp i (đầu ra của mạng tại node i)
- K là số lớp đầu ra

Hàm softmax thực hiện hai công việc chính:

- Chuyển đổi các điểm số đầu ra thành các giá trị xác suất bằng cách biến đổi chúng sao cho tổng của tất cả các xác suất bằng 1.
 - Tăng cường sự khác biệt giữa các xác suất, làm cho xác suất cao nhất trở nên rất gần 1 trong khi các xác suất khác gần 0.
- ❖ Việc sử dụng softmax trong lớp đầu ra cho phép mô hình phản ánh chính xác xác suất của dữ liệu thuộc vào từng lớp, giúp ta dễ dàng tạo ra dự đoán đúng cho nhiều lớp khác nhau trong bài toán phân loại đa lớp.

Ví dụ về Sơ đồ đồ Neural network



Trong đó :

- Neural network*
 - + Cấu trúc lớp (layer)
 - + Lớp đầu vào (Input layer)
 - + Các lớp ẩn (Hidden Layers)
 - + Lớp đầu ra (Output layer)
 - + Kí hiệu
 - + $l^{(i)}$ là số node của lớp thứ i
- Ban đầu :*
 - + Input đầu vào là mảng $x^{(0)} = [x_1^{(0)} \quad x_2^{(0)} \quad \dots \quad x_l^{(0)}]$ với kích thước là $1 * l^0$
 - + Ở Linear và Logistic regression thì ta có w_0 nhưng ở neural thì ta sẽ có b_0 là hệ số bias tách ra để để tính toán. $w^{(1)}$ và $b^{(1)}$ là lớp fully connected kết nối giữa 2 layer .

Với $w^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \dots & w_{1l^1}^{(1)} \\ \dots & \dots & \dots \\ w_{l^0 1}^{(1)} & \dots & w_{l^0 l^1}^{(1)} \end{bmatrix}$ với ma trận là $l^0 * l^1$ và $b^{(1)} = [b_1^{(1)} \ b_2^{(1)} \ \dots \ b_{l^1}^{(1)}]$ với ma trận $1 * l^1$
 Ta có $z^{(1)} = x^{(0)} * w^{(1)} + b^{(1)} = [z_1^{(1)} \ z_2^{(1)} \ \dots \ z_{l^1}^{(1)}]$

Áp dụng activation function, ví dụ là sigmoid cho z ta có: $a^{(1)} = \sigma(z^{(1)}) = [a_1^{(1)} \ a_2^{(1)} \ \dots \ a_{l^1}^{(1)}]$

Ở đây: Trên mỗi layer ta chỉ áp dụng một hàm kích hoạt, nhiều layer có thể áp dụng nhiều hàm kích hoạt ví dụ: layer_1 dùng sigmoid, layer_2 dùng relu, layer_i dùng tanh,...

❖ Sau đó $a^{(1)}$ lại trở thành input đầu vào cho lớp tiếp theo...

Chính vì thế nên đổi kí hiệu $a^{(0)} = [x^{(0)}]^T$ với ma trận là $l^0 * 1$ cho tiện tính toán sau này.

❖ Tổng quát ra ta sẽ có $a^{(i+1)}$ với ma trận là $l^{(i+1)} * 1$ với $l^{(i+1)}$ là số node trên lớp thứ i và $a^{(i+1)}$ là kết quả đầu ra của lớp $a^{(i)}$ và sẽ là đầu vào của lớp $a^{(i+2)}$, trong đó $i \in [0, N - 3]$, mạng neural network có N lớp (gồm cả input, hidden, output) ($N \geq 3$)

4. Sau đó :

+ Vì $a^{(i)}$ đã được thay đổi nên $w^{(i)}$ cũng phải được thay đổi ta sẽ có

$$a. \quad z^{(i+1)} = a^{(i)} * [w^{(i+1)}]^T + b^{(i)}$$

$$b. \quad a^{(i+1)} = \text{activation function}(z^{(i+1)})$$

ex: $a^{(i+1)} = \sigma(z^{(i+1)})$; if activation function = sigmoid function

List activation function

1. Sigmoid Activation Function

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

2. Tanh Activation Function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3. ReLU (Rectified Linear Unit) Activation Function

$$f(x) = \max(0, x)$$

4. Leaky ReLU Activation Function

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \text{constant} * x & \text{others} \end{cases}$$

5. Softmax Activation Function (used in output layer for multi-class classification)

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^N e^{x_j}}$$

6. Linear Activation Function

$$f(x) = x$$

7. Swish Activation Function

$$f(x) = x * \sigma(x)$$

8. ELU (Exponential Linear Unit) Activation Function

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha * (e^x - 1), & \text{others} \end{cases}$$

Trong đó, α là một số dương cố định.

Biểu diễn dưới dạng ma trận

Feedforward (Lan truyền xuôi)

Tất nhiên trên thực tế ta sẽ không làm việc với một dòng dữ liệu. Đầu vào dữ liệu X sẽ có n hàng và d là số trường dữ liệu hay còn gọi là số tham số đầu vào. Ta có X với ma trận $n * d$. Ta có x_j^i là phần tử ở hàng i cột j.

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_d^1 \\ x_1^2 & x_2^2 & \dots & x_d^2 \\ \dots & \dots & \dots & \dots \\ x_1^n & x_2^n & \dots & x_d^n \end{bmatrix} = \begin{bmatrix} -(x^{(1)})^T & - \\ -(x^{(2)})^T & - \\ \dots & \\ -(x^{(n)})^T & - \end{bmatrix}$$

Giải thích : Như đã nói ở trên mỗi hàng dữ liệu $x^{(i)}$ đã được chuyển vị lại thành $[x^{(i)}]^T$

*Bởi vì mỗi điểm dữ liệu $x^{(i)}$ là một vector cột với kích thước $d * 1$, chuyển vị để kích thước $1 * d$*

Lưu ý : $[x^{(i)}]^T$, i ở đây chính là hàng dữ liệu thứ i. còn i ở Z và A là layer thứ i

Gọi ma trận $Z^{(i)}$ kích thước $N * l^{(i)}$ trong đó $z_j^{(i)[k]}$ là giá trị thứ j trong layer i sau bước tính tổng linear của dữ liệu thứ k trong dataset. **Lưu ý : Ở trong X thì i chính là hàng dữ liệu thứ i. Ở trong Z và A thì i chính là layer thứ i và k chính là hàng dữ liệu thứ k trong dataset, tránh nhầm lẫn.**

Tương tự, gọi $A^{(i)}$ kích thước $N * l^{(i)}$ trong đó $a_j^{(i)[k]}$ là giá trị thứ j trong layer i sau khi áp dụng activation function của dữ liệu thứ k trong dataset

*Giải thích : $Z^{(i)}$ và $A^{(i)}$ có kích thước $N * l^{(i)}$ trong đó N là số hàng và $l^{(i)}$ là số cột. N sẽ là cố định, N chính là số hàng dữ liệu của dataset, lặp qua N hàng dữ liệu và ta có được N hàng $Z^{(i)}$, còn $l^{(i)}$ sẽ thay đổi vì nó phụ thuộc vào số Node của các lớp layer. Tương tự với $A^{(i)}$.*

Ta có :

$$\begin{aligned} Z^{(i)} &= \begin{bmatrix} z_1^{(i)[1]} & z_2^{(i)[1]} & \dots & z_{l^{(i)}}^{(i)[1]} \\ z_1^{(i)[2]} & z_2^{(i)[2]} & \dots & z_{l^{(i)}}^{(i)[2]} \\ \dots & \dots & \dots & \dots \\ z_1^{(i)[n]} & z_2^{(i)[n]} & \dots & z_{l^{(i)}}^{(i)[n]} \end{bmatrix} = \begin{bmatrix} -(z^{(i)[1]})^T & - \\ -(z^{(i)[2]})^T & - \\ \dots & \\ -(z^{(i)[n]})^T & - \end{bmatrix} = \begin{bmatrix} (x^{[1]})^T * w^{(i)} + (b^{(i)})^T \\ (x^{[2]})^T * w^{(i)} + (b^{(i)})^T \\ \dots \\ (x^{[n]})^T * w^{(i)} + (b^{(i)})^T \end{bmatrix} \\ &= \begin{bmatrix} (x^{[1]})^T * w^{(i)} + (b^{(i)})^T \\ (x^{[2]})^T * w^{(i)} + (b^{(i)})^T \\ \dots \\ (x^{[n]})^T * w^{(i)} + (b^{(i)})^T \end{bmatrix} = X * W^{(i)} + \begin{bmatrix} (b^{(i)})^T \\ (b^{(i)})^T \\ \dots \\ (b^{(i)})^T \end{bmatrix} = X * W^{(i)} + b^{(i)} \end{aligned}$$

Giải thích :

5. Có N hàng dữ liệu và M layer, lắp qua N hàng dữ liệu ta sẽ có $Z^{(i)}$ và lắp qua M layer ta sẽ có ma trận Z gồm nhiều $Z^{(i)}$, $i \in [1, M - 1]$; $M \geq 3$. (layer luôn bắt đầu bằng 0 nên input layer sẽ là $i - 1$)
6. $(x^{[k]})^T$ có kích thước $1 * d$ với d là số đặc trưng (tham số đầu vào), mặc khác d cũng là $l^{(0)}$, là số node của lớp input đầu vào
7. $w^{(i)}$ có kích thước là $l^{(0)} * l^{(1)}$
8. $(b^{(i)})^T$ có kích thước là $1 * l^{(1)}$
9. Giả sử ta khai triển $z^{(i)[1]}$: Với $i = 1$ (lớp đầu tiên sau khi cho input $x^{(k)}$ vào

$$z^{(1)[1]} = [x_1^1 \quad x_2^1 \quad \dots \quad x_{l^{(0)}}^1] * \begin{bmatrix} w_{11}^{(1)} & \dots & w_{1l^{(1)}}^{(1)} \\ \dots & \dots & \dots \\ w_{l^{(0)}1}^{(1)} & \dots & w_{l^{(0)}l^{(1)}}^{(1)} \end{bmatrix} + [b_1^1 \quad b_2^1 \quad \dots \quad b_{l^{(1)}}^1]$$

$$z_1^{(1)[1]} = x_1^1 * w_{11}^1 + x_2^1 * w_{21}^1 + \dots + x_{l^{(0)}}^1 * w_{l^{(0)}1}^1 + b_1^1$$

$$z_2^{(1)[1]} = x_1^1 * w_{12}^1 + x_2^1 * w_{22}^1 + \dots + x_{l^{(0)}}^1 * w_{l^{(0)}2}^1 + b_2^1$$

...

$$z_{l^{(1)}}^{(1)[1]} = x_1^1 * w_{1l^{(1)}}^1 + x_2^1 * w_{2l^{(1)}}^1 + \dots + x_{l^{(0)}}^1 * w_{l^{(0)}l^{(1)}}^1 + b_{l^{(1)}}^1$$

$$\text{Cuối cùng ta được: } z^{(1)[1]} = [z_1^{(1)[1]} \quad z_2^{(1)[1]} \quad \dots \quad z_{l^{(1)}}^{(1)[1]}] \text{ với kích thước } 1 * l^{(1)}$$

- ❖ Sau khi có $z^{(i)[k]}$ ta chỉ cần cho qua activation function nữa là xong có được $a^{(i)[k]}$ làm đầu vào cho lớp tiếp theo. $a^{(i)[k]} = \sigma(z^{(i)[k]})$ hay $A^{(i)} = \sigma(Z^{(i)})$. Và $A^{(i)} = \hat{Y}$ nếu i là lớp cuối cùng.

Tổng quát :

Ta sẽ có với $i = 0$ (input layer)

$$z^{(i+1)[k]} = [x_1^k \quad x_2^k \quad \dots \quad x_{l^{(i)}}^k] * \begin{bmatrix} w_{11}^{(i+1)} & \dots & w_{1l^{(i+1)}}^{(i+1)} \\ \dots & \dots & \dots \\ w_{l^{(i)}1}^{(i+1)} & \dots & w_{l^{(i)}l^{(i+1)}}^{(i+1)} \end{bmatrix} + [b_1^1 \quad b_2^1 \quad \dots \quad b_{l^{(i+1)}}^1]$$

$$z_1^{(i+1)[k]} = x_1^k * w_{11}^{(i+1)} + x_2^k * w_{21}^{(i+1)} + \dots + x_{l^{(i)}}^k * w_{l^{(i)}1}^{(i+1)} + b_1^1$$

$$z_2^{(i+1)[k]} = x_1^k * w_{12}^{(i+1)} + x_2^k * w_{22}^{(i+1)} + \dots + x_{l^{(i)}}^k * w_{l^{(i)}2}^{(i+1)} + b_2^1$$

...

$$z_{l^{(i+1)}}^{(i+1)[k]} = x_1^k * w_{1l^{(i+1)}}^{(i+1)} + x_2^k * w_{2l^{(i+1)}}^{(i+1)} + \dots + x_{l^{(i)}}^k * w_{l^{(i)}l^{(i+1)}}^{(i+1)} + b_{l^{(i+1)}}^1$$

$$\text{Cuối cùng ta được: } z^{(i+1)[k]} = [z_1^{(i+1)[k]} \quad z_2^{(i+1)[k]} \quad \dots \quad z_{l^{(i+1)}}^{(i+1)[k]}] \text{ với kích thước } 1 * l^{(i+1)}$$

$$a^{(i+1)[k]} = [\sigma(z_1^{(i+1)[k]}) \quad \sigma(z_2^{(i+1)[k]}) \quad \dots \quad \sigma(z_{l^{(i+1)}}^{(i+1)[k]})]$$

Ta sẽ có với $i \geq 1$ (hidden layer + output layer)

$$z^{(i+1)[k]} = [a_1^{(i)[k]} \quad a_2^{(i)[k]} \quad \dots \quad a_{l^{(i)}}^{(i)[k]}] * \begin{bmatrix} w_{11}^{(i+1)} & \dots & w_{1l^{(i+1)}}^{(i+1)} \\ \dots & \dots & \dots \\ w_{l^{(i)}1}^{(i+1)} & \dots & w_{l^{(i)}l^{(i+1)}}^{(i+1)} \end{bmatrix} + [b_1^{(i+1)} \quad b_2^{(i+1)} \quad \dots \quad b_{l^{(i+1)}}^{(i+1)}]$$

$$z_1^{(i+1)[k]} = a_1^{(i)[k]} * w_{11}^{(i+1)} + a_2^{(i)[k]} * w_{21}^{(i+1)} + \dots + a_{l^{(i)}}^{(i)[k]} * w_{l^i 1}^{(i+1)} + b_1^{(i+1)}$$

$$z_2^{(i+1)[k]} = a_1^{(i)[k]} * w_{12}^{(i+1)} + a_2^{(i)[k]} * w_{22}^{(i+1)} + \dots + a_{l^{(i)}}^{(i)[k]} * w_{l^i 2}^{(i+1)} + b_2^{(i+1)}$$

...

$$z_{l^{i+1}}^{(i+1)[k]} = a_1^{(i)[k]} * w_{1l^{i+1}}^{(i+1)} + a_2^{(i)[k]} * w_{2l^{i+1}}^{(i+1)} + \dots + a_{l^{(i)}}^{(i)[k]} * w_{l^i l^{i+1}}^{(i+1)} + b_{l^{(i+1)}}^{(i+1)}$$

Cuối cùng ta được : $z^{(i+1)[k]} = [z_1^{(i+1)[k]} \ z_2^{(i+1)[k]} \ \dots \ z_{l^{i+1}}^{(i+1)[k]}]$ với kích thước $1 * l^{i+1}$

$$a^{(i+1)[k]} = [\sigma(z_1^{(i+1)[k]}) \ \sigma(z_2^{(i+1)[k]}) \ \dots \ \sigma(z_{l^{i+1}}^{(i+1)[k]})]$$

- ❖ Từ input X ta có \hat{Y} , tuy nhiên như 2 bài trước, cần phải tối ưu W và b bằng Gradient descent. Ta sẽ đi tìm đạo hàm của các hệ số đối với hàm loss function. Việc tính đạo hàm của các hệ số trong neural network được thực hiện bởi thuật toán **backpropagation**.

Backpropagation (Lan truyền ngược)

Model

10. Mô hình $2 - 2 - 1$ có nghĩa là 2 node trong input layer, 1 hidden layer có 2 node và output layer có 1 node.
11. Input layer và hidden layer luôn thêm node 1 để tính bias cho layer sau, nhưng không tính vào số lượng node trong layer.
12. Ở mỗi node trong hidden layer và output layer đều thực hiện 2 bước: tính tổng linear và áp dụng activation function.

Loss function

Hàm loss function vẫn dùng giống như trong bài 2, với mỗi điểm $(x^{(i)}, y_i)$, gọi hàm loss function

$$L = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

$$J = \frac{1}{N} * \sum_{i=1}^N L = -\frac{1}{N} * \sum_{i=1}^N x_m^{(i)} * (y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

Lưu ý : $[x^{(i)}]^T$, i ở đây chính là hàng dữ liệu thứ i. còn i ở Z và A là layer thứ i

Gradient descent

Lưu ý :

Khi hàm f(x) là hàm 1 biến x, ví dụ $f(x) = 2 * x + 1$. Đạo hàm f đối với biến x là $\frac{df}{dx}$

Khi hàm f(x,y) là hàm nhiều biến, ví dụ $f(x,y) = x^2 + y^2$. Đạo hàm f đối với biến x kí hiệu là $\frac{\partial f}{\partial x}$

$$L = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

$$\text{Hay } L = -(y_{i+1} * \log_e(\hat{y}_{i+1}) + (1 - y_{i+1}) * \log_e(1 - \hat{y}_{i+1}))$$

Trong đó :

$$\hat{y}_{i+1} = a^{(i+1)[k]} = \sigma(z_{l^{i+1}}^{(i+1)[k]}) = \sigma(a_1^{(i)[k]} * w_{1l^{i+1}}^{(i+1)} + a_2^{(i)[k]} * w_{2l^{i+1}}^{(i+1)} + \dots + a_{l^{(i)}}^{(i)[k]} * w_{l^i l^{i+1}}^{(i+1)} + b_{l^{(i+1)}}^{(i+1)})$$

Chuẩn bị

$$\frac{\partial L}{\partial \hat{y}_i} = \frac{\partial(-(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i)))}{\partial \hat{y}_i} = -\left(\frac{y_i}{\hat{y}_i} - \frac{1 - y_i}{1 - \hat{y}_i}\right) = \frac{\hat{y}_i - y_i}{\hat{y}_i * (1 - \hat{y}_i)}$$

$$\begin{aligned} \frac{d(\sigma(x))}{dx} &= \frac{d\left(\frac{1}{1+e^{-x}}\right)}{dx} = \frac{d\left(\frac{1}{1+e^{-x}}\right)}{d(1+e^{-x})} * \frac{d(1+e^{-x})}{d(-x)} * \frac{d(-x)}{x} = -\frac{1}{(1+e^{-x})^2} * e^{-x} * -1 \\ &= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} * \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}} * \left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x) * (1 - \sigma(x)) \end{aligned}$$

Tính đạo hàm L với W^{i+1} và b^{i+1}

$$\text{Áp dụng quy tắc Chain rule : } \frac{\partial L}{\partial b^{i+1}} = \frac{\partial L}{\partial \widehat{y_{i+1}}} * \frac{\partial \widehat{y_{i+1}}}{\partial b^{i+1}}$$

$$\begin{aligned} 1. \quad \frac{\partial \widehat{y_{i+1}}}{\partial b_l^{(i+1)}} &= \frac{\partial \left(\sigma(a_1^{(i)[k]} * w_{1l^{i+1}}^{(i+1)} + a_2^{(i)[k]} * w_{2l^{i+1}}^{(i+1)} + \dots + a_l^{(i)[k]} * w_{ll^{i+1}}^{(i+1)} + b_l^{(i+1)}) \right)}{\partial b_l^{(i+1)}} = \frac{\partial \left(\sigma(z_{l^{i+1}}^{(i+1)[k]}) \right)}{\partial z_{l^{i+1}}^{(i+1)[k]}} * \frac{\partial z_{l^{i+1}}^{(i+1)[k]}}{\partial b_l^{(i+1)}} \\ &= \sigma(z_{l^{i+1}}^{(i+1)[k]}) * (1 - \sigma(z_{l^{i+1}}^{(i+1)[k]})) * 1 = \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) \\ 2. \quad \frac{\partial \widehat{y_{i+1}}}{\partial a_l^{(i)[k]}} &= \frac{\partial \left(\sigma(a_1^{(i)[k]} * w_{1l^{i+1}}^{(i+1)} + a_2^{(i)[k]} * w_{2l^{i+1}}^{(i+1)} + \dots + a_l^{(i)[k]} * w_{ll^{i+1}}^{(i+1)} + b_l^{(i+1)}) \right)}{\partial a_l^{(i)[k]}} = \frac{\partial \left(\sigma(z_{l^{i+1}}^{(i+1)[k]}) \right)}{\partial z_{l^{i+1}}^{(i+1)[k]}} * \frac{\partial z_{l^{i+1}}^{(i+1)[k]}}{\partial a_l^{(i)[k]}} \\ &= \sigma(z_{l^{i+1}}^{(i+1)[k]}) * (1 - \sigma(z_{l^{i+1}}^{(i+1)[k]})) * w_{ll^{i+1}}^{(i+1)} = w_{ll^{i+1}}^{(i+1)} * \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) \\ 3. \quad \frac{\partial \widehat{y_{i+1}}}{\partial w_{ll^{i+1}}^{(i+1)}} &= \frac{\partial \left(\sigma(a_1^{(i)[k]} * w_{1l^{i+1}}^{(i+1)} + a_2^{(i)[k]} * w_{2l^{i+1}}^{(i+1)} + \dots + a_l^{(i)[k]} * w_{ll^{i+1}}^{(i+1)} + b_l^{(i+1)}) \right)}{\partial w_{ll^{i+1}}^{(i+1)}} = \frac{\partial \left(\sigma(z_{l^{i+1}}^{(i+1)[k]}) \right)}{\partial z_{l^{i+1}}^{(i+1)[k]}} * \frac{\partial z_{l^{i+1}}^{(i+1)[k]}}{\partial w_{ll^{i+1}}^{(i+1)}} \\ &= \sigma(z_{l^{i+1}}^{(i+1)[k]}) * (1 - \sigma(z_{l^{i+1}}^{(i+1)[k]})) * a_{l^{(i)}}^{(i)[k]} = a_{l^{(i)}}^{(i)[k]} * \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) \end{aligned}$$

Do đó :

$$\begin{aligned} 1. \quad \frac{\partial L}{\partial b_l^{(i+1)}} &= \frac{\partial L}{\partial \widehat{y_{i+1}}} * \frac{\partial \widehat{y_{i+1}}}{\partial b_l^{(i+1)}} = \frac{\widehat{y_{i+1}} - y_{i+1}}{\widehat{y_{i+1}} * (1 - \widehat{y_{i+1}})} * \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) = \widehat{y_{i+1}} - y_{i+1} \\ 2. \quad \frac{\partial L}{\partial a_{l^{(i)}}^{(i)[k]}} &= \frac{\partial L}{\partial \widehat{y_{i+1}}} * \frac{\partial \widehat{y_{i+1}}}{\partial a_{l^{(i)}}^{(i)[k]}} = \frac{\widehat{y_{i+1}} - y_{i+1}}{\widehat{y_{i+1}} * (1 - \widehat{y_{i+1}})} * w_{ll^{i+1}}^{(i+1)} * \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) = w_{ll^{i+1}}^{(i+1)} * (\widehat{y_{i+1}} - y_{i+1}) \\ 3. \quad \frac{\partial L}{\partial w_{ll^{i+1}}^{(i+1)}} &= \frac{\partial L}{\partial \widehat{y_{i+1}}} * \frac{\partial \widehat{y_{i+1}}}{\partial w_{ll^{i+1}}^{(i+1)}} = \frac{\widehat{y_{i+1}} - y_{i+1}}{\widehat{y_{i+1}} * (1 - \widehat{y_{i+1}})} * a_{l^{(i)}}^{(i)[k]} * \widehat{y_{i+1}} * (1 - \widehat{y_{i+1}}) = a_{l^{(i)}}^{(i)[k]} * (\widehat{y_{i+1}} - y_{i+1}) \end{aligned}$$

Biểu diễn dưới dạng ma trận

Lưu ý : Đạo hàm của L đối với ma trận W kích thước m * n cũng là một ma trận cùng kích thước m * n .

$$\frac{\partial L}{\partial W} \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \dots & \frac{\partial L}{\partial w_{1n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{m1}} & \dots & \frac{\partial L}{\partial w_{mn}} \end{bmatrix}$$

Do đó : $\frac{\partial J}{\partial w^{i+1}} = (a^{(i)[k]})^T * (\hat{Y} - Y)$; $\frac{\partial J}{\partial b^{i+1}} = (sum(\hat{Y} - Y))^T$; $\frac{\partial J}{\partial a^{(i)[k]}} = (w^{i+1})^T * (\hat{Y} - Y)$

Phép tính **Sum** tính tổng trên các cột của ma trận

$$W = \begin{bmatrix} w_{11}^{(i+1)} & \dots & w_{1l^{(i+1)}}^{(i+1)} \\ \vdots & \ddots & \vdots \\ w_{l^i 1}^{(i+1)} & \dots & w_{l^i l^{(i+1)}}^{(i+1)} \end{bmatrix} \Rightarrow sum(W) = (w_{11}^{(i+1)} + \dots + w_{l^i 1}^{(i+1)}, \dots, w_{1l^{(i+1)}}^{(i+1)} + \dots + w_{l^i l^{(i+1)}}^{(i+1)})$$

$$(sum(W))^T = \begin{bmatrix} w_{11}^{(i+1)} + \dots + w_{l^i 1}^{(i+1)} \\ \vdots \\ w_{1l^{(i+1)}}^{(i+1)} + \dots + w_{l^i l^{(i+1)}}^{(i+1)} \end{bmatrix}$$

Tính đạo hàm **L** với W^i và b^i . Ta sẽ có với $i = 0$ (input layer)

Lưu ý : Ở trên ta đạo hàm như sau : $\frac{\partial L}{\partial b^{i+1}} = \frac{\partial L}{\partial \widehat{y}_{i+1}} * \frac{\partial \widehat{y}_{i+1}}{\partial b^{i+1}}$

Còn sau đây ta sẽ đạo hàm theo : $\frac{\partial L}{\partial b^{i+1}} = \frac{\partial L}{\partial a^{(i+1)}} * \frac{\partial a^{(i+1)}}{\partial b^{i+1}}$

Ta có :

$$z_{l^{i+1}}^{(i+1)[k]} = x_1^k * w_{1l^{i+1}}^{(i+1)} + x_2^k * w_{2l^{i+1}}^{(i+1)} + \dots + x_{l^i}^k * w_{l^i l^{i+1}}^{(i+1)} + b_{l^{i+1}}^1$$

Cuối cùng ta được : $z^{(i+1)[k]} = [z_1^{(i+1)[k]} \ z_2^{(i+1)[k]} \ \dots \ z_{l^{i+1}}^{(i+1)[k]}]$ với kích thước $1 * l^{i+1}$

$$a^{(i+1)[k]} = [\sigma(z_1^{(i+1)[k]}) \ \sigma(z_2^{(i+1)[k]}) \ \dots \ \sigma(z_{l^{i+1}}^{(i+1)[k]})]$$

$$1. \quad \frac{\partial a^{(i+1)}}{\partial b^{i+1}} = \frac{\partial (\sigma(z_{l^{i+1}}^{(i+1)[k]}))}{\partial z_{l^{i+1}}^{(i+1)[k]}} * \frac{\partial z_{l^{i+1}}^{(i+1)[k]}}{\partial b_{l^{i+1}}^1} = a^{(i+1)} * (1 - a^{(i+1)})$$

$$2. \quad \frac{\partial a^{(i+1)}}{\partial w_{l^i l^{i+1}}} = \frac{\partial (\sigma(z_{l^{i+1}}^{(i+1)[k]}))}{\partial z_{l^{i+1}}^{(i+1)[k]}} * \frac{\partial z_{l^{i+1}}^{(i+1)[k]}}{\partial w_{l^i l^{i+1}}} = x_{l^i}^k * a^{(i+1)} * (1 - a^{(i+1)})$$

Mặc khác ta có ở trên

$$\frac{\partial L}{\partial a^{(i)}} = W_{l^i l^{i+1}}^{(i+1)} * (\widehat{y}_{i+1} - y_{i+1}). Thay i bằng i + 1 ta sẽ được tương tự$$

Do đó :

$$1. \quad \frac{\partial L}{\partial b^{i+1}} = \frac{\partial L}{\partial a^{(i+1)}} * \frac{\partial a^{(i+1)}}{\partial b^{i+1}} = a^{(i+1)} * (1 - a^{(i+1)}) * w_{l^{i+2} l^{i+1}}^{(i+2)} * (\widehat{y}_{i+1} - y_{i+1})$$

$$2. \frac{\partial L}{\partial w_{l^i l^{i+1}}^{(i+1)}} = \frac{\partial L}{\partial a^{(i+1)}} * \frac{\partial a^{(i+1)}}{\partial w_{l^i l^{i+1}}^{(i+1)}} = x_l^k * a^{(i+1)} * (1 - a^{(i+1)}) * w_{l^{i+2} l^{i+1}}^{(i+2)} * (\widehat{y_{i+1}} - y_{i+1})$$

Biểu diễn dưới dạng ma trận

Có thể viết dưới dạng Chain rule : $\frac{\partial J}{\partial W^{(i+1)}} = \frac{\partial J}{\partial A^{(i+1)}} * \frac{\partial A^{(i+1)}}{\partial Z^{(i+1)}} * \frac{\partial Z^{(i+1)}}{\partial W^{(i+1)}}$ (1)

Từ trên ta đã tính được : $\frac{\partial J}{\partial A^{(i+1)}} = (\widehat{Y} - Y) * (W^{(i+2)})^T$

Đạo hàm của hàm sigmoid $\frac{d\sigma(x)}{dx} = \sigma(x) * (1 - \sigma(x))$ và $A^{(i+1)} = \sigma(Z^{(i+1)})$, nên trong (1) có thể hiểu là $\frac{\partial A^{(i+1)}}{\partial Z^{(i+1)}} = A^{(i+1)} * (1 - A^{(i+1)})$.

Cuối cùng $Z^{(i+1)} = X * W^{(i+1)} + b^{(i+1)}$, nên có thể hiểu $\frac{\partial Z^{(i+1)}}{\partial W^{(i+1)}} = X$

Kết hợp lại tất cả ta có : $\frac{\partial J}{\partial W^{(i+1)}} = X^T * \left(((\widehat{Y} - Y) * (W^{(i+2)})^T) \otimes A^{(i+1)} \otimes (1 - A^{(i+1)}) \right)$

Thế khi nào thì dùng element-wise(\otimes), khi nào dùng nhân ma trận($*$) ?

4. Khi tính đạo hàm ngược lại quá trình activation thì dùng \otimes
5. Khi có phép tính nhân ma trận thì dùng $(*)$, nhưng đặc biệt chú ý đến kích thước ma trận và dùng transpose nếu cần thiết. Ví dụ ma trận X kích thước $N*3$, W kích thước $3*4$, $Z = X * W$ sẽ có kích thước $N * 4$ thì

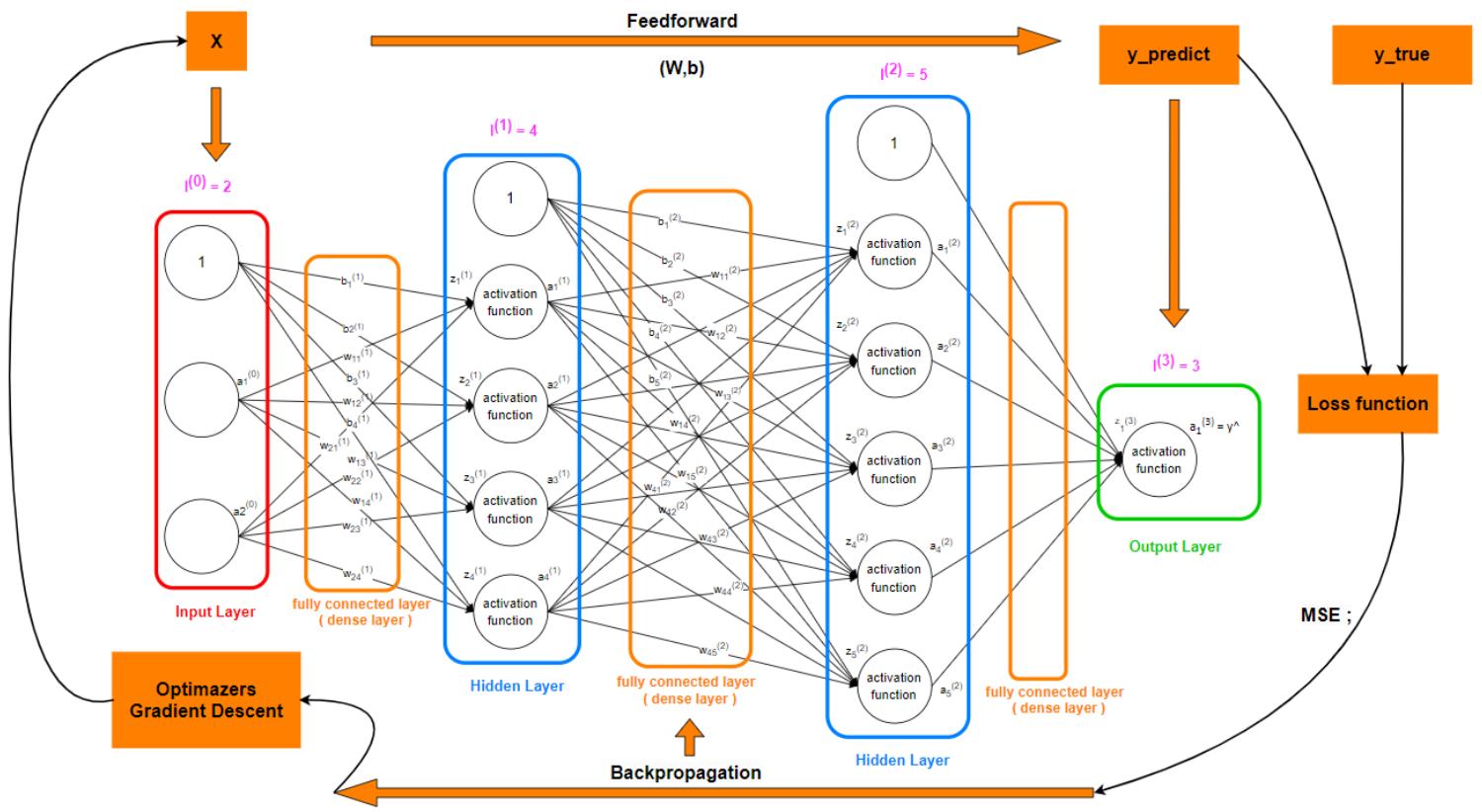
$$\frac{\partial J}{\partial W} = X^T * \frac{\partial J}{\partial Z} \text{ và } \frac{\partial J}{\partial X} = \frac{\partial J}{\partial Z} * W^T. \text{ Tương tự } \frac{\partial J}{\partial b^{(i+1)}} = \text{sum} \left(\left((\widehat{Y} - Y) * (W^{(i+2)})^T \right) \otimes A^{(i+1)} \right)^T$$

❖ **Vậy là đã tính xong hết đạo hàm của loss function với các hệ số W và bias b, giờ có thể áp dụng gradient descent để giải bài toán .**

Tóm tắt toàn bộ quá trình

Giả sử ta có mạng neural network với số lớp như sau :

- Input layer : 2 node
- Hidden layer 1 : 4 node
- Hidden layer 2 : 5 node
- Output layer : 1 node



1. Bước 1 : $\frac{\partial J}{\partial \hat{Y}}$, trong đó $\hat{Y} = A^{(3)}$

2. Bước 2 :

$$\begin{aligned}\frac{\partial J}{\partial \widehat{W^{(3)}}} &= (A^{(2)})^T * \left(\frac{\partial J}{\partial \hat{Y}} \otimes \frac{\partial A^{(3)}}{\partial Z^{(3)}} \right) \\ \frac{\partial J}{\partial \widehat{b^{(3)}}} &= \left(\text{sum} \left(\frac{\partial J}{\partial \hat{Y}} \otimes \frac{\partial A^{(3)}}{\partial Z^{(3)}} \right) \right)^T \\ \frac{\partial J}{\partial \widehat{A^{(2)}}} &= \left(\frac{\partial J}{\partial \hat{Y}} \otimes \frac{\partial A^{(3)}}{\partial Z^{(3)}} \right) * (W^{(3)})^T\end{aligned}$$

3. Bước 3 :

$$\begin{aligned}\frac{\partial J}{\partial \widehat{W^{(2)}}} &= (A^{(1)})^T * \left(\frac{\partial J}{\partial A^{(2)}} \otimes \frac{\partial A^{(2)}}{\partial Z^{(2)}} \right) \\ \frac{\partial J}{\partial \widehat{b^{(2)}}} &= \left(\text{sum} \left(\frac{\partial J}{\partial A^{(2)}} \otimes \frac{\partial A^{(2)}}{\partial Z^{(2)}} \right) \right)^T \\ \frac{\partial J}{\partial \widehat{A^{(1)}}} &= \left(\frac{\partial J}{\partial A^{(2)}} \otimes \frac{\partial A^{(2)}}{\partial Z^{(2)}} \right) * (W^{(2)})^T\end{aligned}$$

4. Bước 4 :

$$\frac{\partial J}{\partial \widehat{W^{(1)}}} = (A^{(0)})^T * \left(\frac{\partial J}{\partial A^{(1)}} \otimes \frac{\partial A^{(1)}}{\partial Z^{(1)}} \right)$$

$$\frac{\partial J}{\partial \widehat{b^{(1)}}} = \left(\text{sum} \left(\frac{\partial J}{\partial A^{(1)}} \otimes \frac{\partial A^{(1)}}{\partial Z^{(1)}} \right) \right)^T$$

$$A^{(0)} = X$$

- Nếu network có nhiều layer hơn thì cứ tiếp tục cho đến khi tính được đạo hàm của loss function J với tất cả hệ số W và bias b.

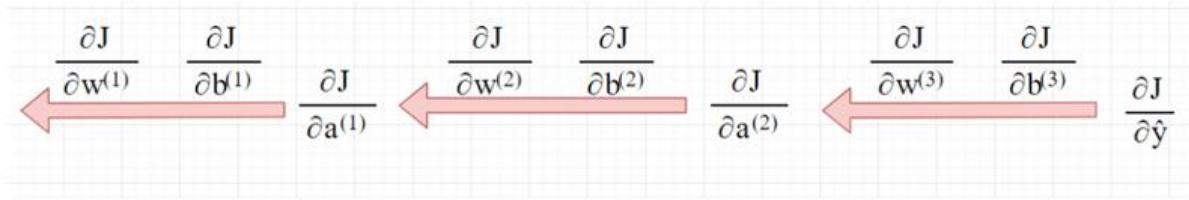
Nếu hàm activation là sigmoid thì $\frac{\partial A^{(i)}}{\partial Z^{(i)}} = A^{(i)} \otimes (1 - A^{(i)})$

Ở bài trước quá trình feedforward



Hình 6.3: Quá trình feedforward

Thì ở bài này quá trình tính đạo hàm ngược lại



Hình 6.4: Quá trình backpropagation

Đây là vì sao thuật toán được gọi là backpropagation (làn truyền ngược)

Nói thêm

Mạng nơ-ron tích chập (CNN) thường được sử dụng trong xử lý ảnh và nhận diện vì các đặc điểm sau:

- Cấu trúc chia sẻ trọng số: CNN sử dụng các lớp tích chập để trích xuất đặc trưng từ dữ liệu đầu vào. Các lớp này có các bộ lọc nhỏ di chuyển trên toàn bộ hình ảnh để phát hiện các đặc điểm cụ thể như cạnh, góc, hoặc đường cong. Việc sử dụng trọng số chia sẻ giúp giảm bớt số lượng tham số cần học và tăng tốc độ tính toán.
- Dữ liệu không gian: Ảnh có cấu trúc không gian, nghĩa là các pixel ở gần nhau thường có mối quan hệ về mặt không gian. CNN có thể hiệu quả trong việc xử lý dữ liệu không gian như ảnh bằng cách giữ lại thông tin về không gian và mối quan hệ giữa các pixel.
- Scale invariance: CNN có khả năng nhận biết các đặc trưng ở nhiều tỷ lệ khác nhau trong ảnh. Điều này có ý nghĩa là nó có thể nhận diện đối tượng trong ảnh dù chúng có kích thước khác nhau.

CNN và RNN (Mạng nơ-ron hồi tiếp) khác nhau ở cấu trúc và cách chúng xử lý dữ liệu. Trong khi CNN thích hợp cho dữ liệu không gian như ảnh, RNN thích hợp cho dữ liệu tuần tự như văn bản hoặc âm

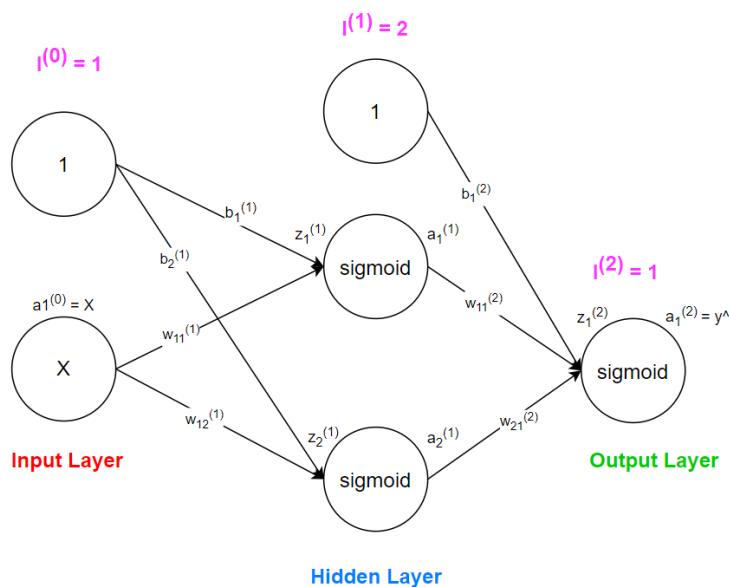
thanh. CNN thường được sử dụng cho việc phát hiện đặc trưng cục bộ trong dữ liệu không gian, trong khi RNN thích hợp cho việc mô hình chuỗi dữ liệu với các phụ thuộc thời gian.

KNN (K-nearest neighbors) là một thuật toán học máy không được huấn luyện, nghĩa là nó không học ra một mô hình từ dữ liệu huấn luyện mà thay vào đó so sánh các điểm dữ liệu mới với các điểm dữ liệu đã được gán nhãn trong tập huấn luyện và dự đoán dựa trên việc tìm kiếm các điểm gần nhất. KNN không có cấu trúc mạng nơ-ron như CNN hoặc RNN, và nó không yêu cầu việc huấn luyện mô hình.

Neural network calculates by hand

Author : Nguyen Van Manh

Sơ đồ lớp



Tập dữ liệu đầu vào : Chuẩn hóa x và y bằng phương pháp Min-Max Scaling hoặc Standardization (z-score normalization)

| nth | H(x) | W(y) | $x_{scaling}$ | $y_{scaling}$ |
|-----|------|------|---------------|---------------|
| 0 | 160 | 62 | 0 | 0 |
| 1 | 165 | 64 | 0.5 | 0.5 |
| 2 | 170 | 66 | 1 | 1 |

- ❖ Ta thấy x và y quá lớn so với trọng số w chính vì thế khi tính ra z rồi cho qua sigmoid thì a gần như bằng 1 , điều này dẫn đến kết quả sau đạo hàm sẽ có đoạn $a * (1 - a) = 0$

Dẫn đến việc không thể cập nhật trọng số . Nên chuẩn hóa cho x , y nhỏ lại .

$$x_{min} = 160 ; x_{max} = 170 ; y_{min} = 62 ; y_{max} = 66$$

$$x_{\text{scaling}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}; \quad y_{\text{scaling}} = \frac{y - y_{\min}}{y_{\max} - y_{\min}}$$

Khởi tạo trọng số W, b

| $w_{11}^{(1)}$ | $w_{12}^{(1)}$ | $b_1^{(1)}$ | $b_{12}^{(1)}$ | $w_{11}^{(2)}$ | $w_{21}^{(2)}$ | $b_1^{(2)}$ |
|----------------|----------------|-------------|----------------|----------------|----------------|-------------|
| 0.5 | 1 | 0 | 0 | 1 | 0.5 | 1 |

Learning rate : 0.1

Activation function : $\sigma(x) = \frac{1}{1+e^{-x}}$ (Sigmoid)

Loss function : $L = -(y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$ (Log loss – Cross Entropy)

$$J = \frac{1}{N} * \sum_{i=1}^N L = -\frac{1}{N} * \sum_{i=1}^N x_m^{(i)} * (y_i * \log_e(\hat{y}_i) + (1 - y_i) * \log_e(1 - \hat{y}_i))$$

Tóm tắt quá trình

- Lặp qua epochs lần để tối ưu W và b
 - Lặp qua n bảng ghi có trong tập dữ liệu
 - o Qua mỗi bảng ghi
 - Khởi tạo W, b cho các lớp fully connected kết nối giữa các layer
 - Thực hiện quá trình Feedforward để tính z, a của mỗi layer và cuối cùng được \hat{y}
 - Thực hiện quá trình Backpropagation : Mục đích là để cập nhật lại các W, b, hay nói cách khác là tối ưu W, b
 - Lưu ý : Ta đã biết ở các bài trước để tối ưu một giá trị bằng gradient sẽ thực hiện .
 - Ex : $w = w - \text{learning_rate} * \frac{dL}{dw}$ với L là một hàm mất mát
 - Vậy có thể hiểu đơn giản là tính $\frac{dL}{dw}$ và $\frac{dL}{db}$ sau đó cập nhật lại W và b .
 - o Sau n bảng ghi ta thu được ma trận W, b và các ma trận liên quan khác, từ đây ta sẽ tiến hành tính toán độ mất mát cho toàn bộ tập dữ liệu để biết được có dừng lại hay không với mỗi ma trận W và b mới .

Thực hiện quá trình tính toán trên bảng ghi thứ hai của tập dữ liệu

- Tương tự với các bảng ghi khác
- Lặp qua số epochs lần cũng tương tự

Input : x = 0.5 , y = 0.5

Feedforward

$$z_1^{(1)} = x * w_{11}^{(1)} + b_1^{(1)} = 0.5 * 0.5 + 0 = 0.25$$

$$z_2^{(1)} = x * w_{12}^{(1)} + b_2^{(1)} = 0.5 * 1 + 0 = 0.5$$

$$a_1^{(1)} = \sigma(z_1^{(1)}) = \frac{1}{1+e^{-z_1^{(1)}}} \approx 0.562$$

$$a_2^{(1)} = \sigma(z_2^{(1)}) = \frac{1}{1 + e^{-z_2^{(1)}}} \approx 0.622$$

$$z_1^{(2)} = a_1^{(1)} * w_{11}^{(2)} + a_2^{(1)} * w_{21}^{(2)} + b_1^{(2)} = 0.562 * 1 + 0.622 * 0.5 + 1 = 1.873$$

$$a_1^{(2)} = \widehat{y}_1 = \sigma(z_1^{(2)}) = \frac{1}{1 + e^{-z_1^{(2)}}} \approx 0.867$$

Backpropagation (Xem đạo hàm chi tiết ở cuối file)

$$\text{Ta có } \widehat{y}_1 = \sigma(z_1^{(2)}) = \sigma(a_1^{(1)} * w_{11}^{(2)} + a_2^{(1)} * w_{21}^{(2)} + b_1^{(2)}) ; L = -(y_i * \log_e(\widehat{y}_i) + (1 - y_i) * \log_e(1 - \widehat{y}_i))$$

$$\frac{\partial L}{\partial b_1^{(2)}} = \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial b_1^{(2)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * 1 = (\widehat{y}_1 - y_1) = 0.867 - 0.5 \approx 0.367$$

$$\frac{\partial L}{\partial w_{11}^{(2)}} = \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial w_{11}^{(2)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (a_1^{(1)}) = (\widehat{y}_1 - y_1) * a_1^{(1)} = 0.367 * 0.562 \approx 0.206$$

$$\frac{\partial L}{\partial w_{21}^{(2)}} = \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial w_{21}^{(2)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (a_2^{(1)}) = (\widehat{y}_1 - y_1) * a_2^{(1)} = 0.367 * 0.622 \approx 0.228$$

Ta có : Cú tính lùi theo như các biểu thức trên Feedforward

$$\begin{aligned} \frac{\partial L}{\partial b_1^{(1)}} &= \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} * \frac{\partial a_1^{(1)}}{\partial z_1^{(1)}} * \frac{\partial z_1^{(1)}}{\partial b_1^{(1)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (w_{11}^{(2)}) * (a_1^{(1)} (1 - a_1^{(1)})) * 1 \\ &= (\widehat{y}_1 - y_1) * w_{11}^{(2)} * (a_1^{(1)} (1 - a_1^{(1)})) = 0.367 * 1 * 0.562 * (1 - 0.562) = 0.09 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial b_2^{(1)}} &= \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} * \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} * \frac{\partial z_2^{(1)}}{\partial b_2^{(1)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (w_{21}^{(2)}) * (a_2^{(1)} (1 - a_2^{(1)})) * 1 \\ &= (\widehat{y}_1 - y_1) * w_{21}^{(2)} * (a_2^{(1)} (1 - a_2^{(1)})) = 0.367 * 0.5 * 0.622 * (1 - 0.622) = 0.043 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial w_{11}^{(1)}} &= \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} * \frac{\partial a_1^{(1)}}{\partial w_{11}^{(1)}} * \frac{\partial w_{11}^{(1)}}{\partial b_1^{(1)}} \\ &= \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (w_{11}^{(2)}) * (a_1^{(1)} (1 - a_1^{(1)})) * x \\ &= (\widehat{y}_1 - y_1) * w_{11}^{(2)} * (a_1^{(1)} (1 - a_1^{(1)})) * x = 0.367 * 1 * 0.562 * (1 - 0.562) * 0.5 = 0.045 \end{aligned}$$

$$\begin{aligned} \frac{\partial L}{\partial w_{12}^{(1)}} &= \frac{\partial L}{\partial \widehat{y}_1} * \frac{\partial \widehat{y}_1}{\partial z_1^{(2)}} * \frac{\partial z_1^{(2)}}{\partial a_2^{(1)}} * \frac{\partial a_2^{(1)}}{\partial z_2^{(1)}} * \frac{\partial z_2^{(1)}}{\partial w_{12}^{(1)}} = \left(\frac{\widehat{y}_1 - y_1}{\widehat{y}_1 * (1 - \widehat{y}_1)} \right) * (\widehat{y}_1 * (1 - \widehat{y}_1)) * (w_{21}^{(2)}) * (a_2^{(1)} (1 - a_2^{(1)})) * x \\ &= (\widehat{y}_1 - y_1) * w_{21}^{(2)} * (a_2^{(1)} (1 - a_2^{(1)})) * x = 0.367 * 0.5 * 0.622 * (1 - 0.622) * 0.5 = 0.0215 \end{aligned}$$

Áp dụng Gradient descent để cập nhật trọng số

$$\frac{\partial L}{\partial b_1^{(2)}} = 0.367 ; \frac{\partial L}{\partial w_{11}^{(2)}} = 0.206 ; \frac{\partial L}{\partial w_{21}^{(2)}} = 0.228 ; \frac{\partial L}{\partial b_1^{(1)}} = 0.09 ; \frac{\partial L}{\partial b_2^{(1)}} = 0.043 ; \frac{\partial L}{\partial w_{11}^{(1)}} = 0.045 ; \frac{\partial L}{\partial w_{12}^{(1)}} = 0.0215$$

$$w_{11}^{(1)} = w_{11}^{(1)} - learning_rate * \frac{\partial L}{\partial w_{11}^{(1)}} = 0.5 - 0.1 * 0.045 = 0.4955$$

$$w_{12}^{(1)} = w_{12}^{(1)} - learning_rate * \frac{\partial L}{\partial w_{12}^{(1)}} = 1 - 0.1 * 0.0215 = 0.99785$$

$$b_1^{(1)} = b_1^{(1)} - learning_rate * \frac{\partial L}{\partial b_1^{(1)}} = 0 - 0.1 * 0.09 = -0.009$$

$$b_2^{(1)} = b_2^{(1)} - learning_rate * \frac{\partial L}{\partial b_2^{(1)}} = 0 - 0.1 * 0.043 = -0.0043$$

$$w_{11}^{(2)} = w_{11}^{(2)} - learning_rate * \frac{\partial L}{\partial w_{11}^{(2)}} = 1 - 0.1 * 0.206 = 0.9794$$

$$w_{21}^{(2)} = w_{21}^{(2)} - learning_rate * \frac{\partial L}{\partial w_{21}^{(2)}} = 0.5 - 0.1 * 0.228 = 0.4772$$

$$b_1^{(2)} = b_1^{(2)} - learning_rate * \frac{\partial L}{\partial b_1^{(2)}} = 1 - 0.1 * 0.367 = 0.9633$$

Sau epochs thứ nhất

| $w_{11}^{(1)}$ | $w_{12}^{(1)}$ | $b_1^{(1)}$ | $b_{12}^{(1)}$ | $w_{11}^{(2)}$ | $w_{21}^{(2)}$ | $b_1^{(2)}$ |
|----------------|----------------|-------------|----------------|----------------|----------------|-------------|
| 0.4955 | 0.99785 | -0.009 | -0.0043 | 0.9794 | 0.4772 | 0.9633 |

Áp dụng tương tự với các bảng ghi còn lại của tập dữ liệu .

Các phương pháp chuẩn hóa dữ liệu



ChatGPT

Chuẩn hóa dữ liệu là một phương pháp phổ biến trong việc xử lý dữ liệu trước khi đưa vào mạng neural network. Mục đích của việc chuẩn hóa là để đảm bảo rằng các biến đầu vào có phạm vi tương tự, giúp mạng neural network học hiệu quả hơn. Đối với trường hợp của bạn, nếu x và y quá lớn so với các trọng số w , việc chuẩn hóa x và y có thể cần thiết.

Có một số phương pháp để chuẩn hóa dữ liệu, hai phương pháp phổ biến nhất là Min-Max Scaling và Standardization (z-score normalization).

1. Min-Max Scaling:

- Phương pháp này chuyển đổi dữ liệu sao cho giá trị của các biến nằm trong khoảng $[0, 1]$ hoặc khoảng $[-1, 1]$.
- Công thức chuẩn hóa Min-Max: $x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$
- Công thức này đảm bảo rằng giá trị nhỏ nhất sẽ được chuyển thành 0 và giá trị lớn nhất sẽ được chuyển thành 1.

2. Standardization (z-score normalization):

- Phương pháp này chuyển đổi dữ liệu sao cho giá trị trung bình của mỗi biến là 0 và độ lệch chuẩn là 1.
- Công thức chuẩn hóa z-score: $x_{standardized} = \frac{x - \mu}{\sigma}$
- Trong đó, μ là giá trị trung bình của biến và σ là độ lệch chuẩn.

Để áp dụng chuẩn hóa, bạn có thể tính toán giá trị min, max, trung bình và độ lệch chuẩn của các biến x và y trên toàn bộ tập dữ liệu và sau đó áp dụng các công thức chuẩn hóa tương ứng. Đảm bảo rằng việc chuẩn hóa được thực hiện trên tập dữ liệu huấn luyện và sau đó áp dụng cùng các tham số (min, max, trung bình, độ lệch chuẩn) đã tính được cho tập dữ liệu kiểm tra hoặc dữ liệu mới.





ChatGPT

Để chuẩn hóa dữ liệu của biến H (chiều cao) và W (cân nặng), bạn có thể sử dụng phương pháp Min-Max Scaling hoặc Standardization. Dưới đây là cách thực hiện chuẩn hóa sử dụng cả hai phương pháp:

1. Min-Max Scaling:

- Tính min và max của mỗi biến:
 - $\min(H) = 160$
 - $\max(H) = 170$
 - $\min(W) = 62$
 - $\max(W) = 66$
- Áp dụng công thức chuẩn hóa Min-Max:

$$H_{\text{scaled}} = \frac{H - \min(H)}{\max(H) - \min(H)}$$

$$W_{\text{scaled}} = \frac{W - \min(W)}{\max(W) - \min(W)}$$

2. Standardization (z-score normalization):

- Tính mean và standard deviation của mỗi biến:
 - $\mu_H = \frac{160+165+170}{3} = 165$
 - $\sigma_H = \sqrt{\frac{(160-165)^2 + (165-165)^2 + (170-165)^2}{3}}$
 - $\mu_W = \frac{62+64+66}{3} = 64$
 - $\sigma_W = \sqrt{\frac{(62-64)^2 + (64-64)^2 + (66-64)^2}{3}}$
- Áp dụng công thức chuẩn hóa z-score:

$$H_{\text{standardized}} = \frac{H - \mu_H}{\sigma_H}$$

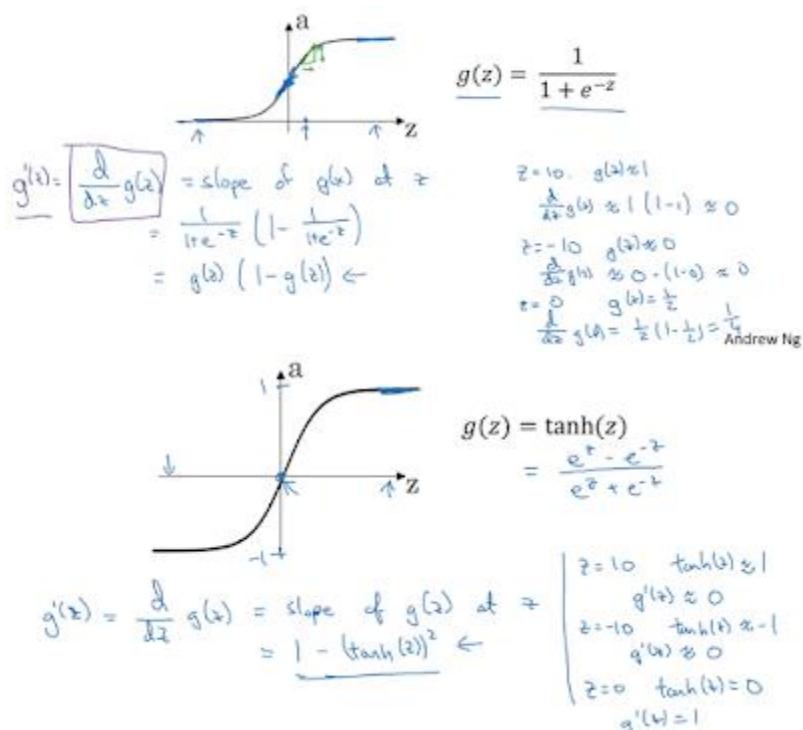
$$W_{\text{standardized}} = \frac{W - \mu_W}{\sigma_W}$$

Sau khi tính toán, bạn sẽ thu được dữ liệu đã được chuẩn hóa cho cả hai biến H và W. Bạn có thể sử dụng dữ liệu này để đưa vào mạng neural network để huấn luyện mô hình của mình.

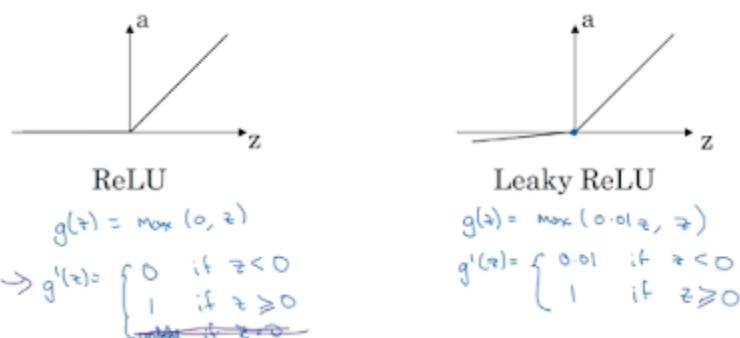
Code Demo

Ö>: [Machine Learning] Đạo hàm của các activation functions (dexterspurs.blogspot.com)

[Machine Learning] Đạo hàm của các activation functions

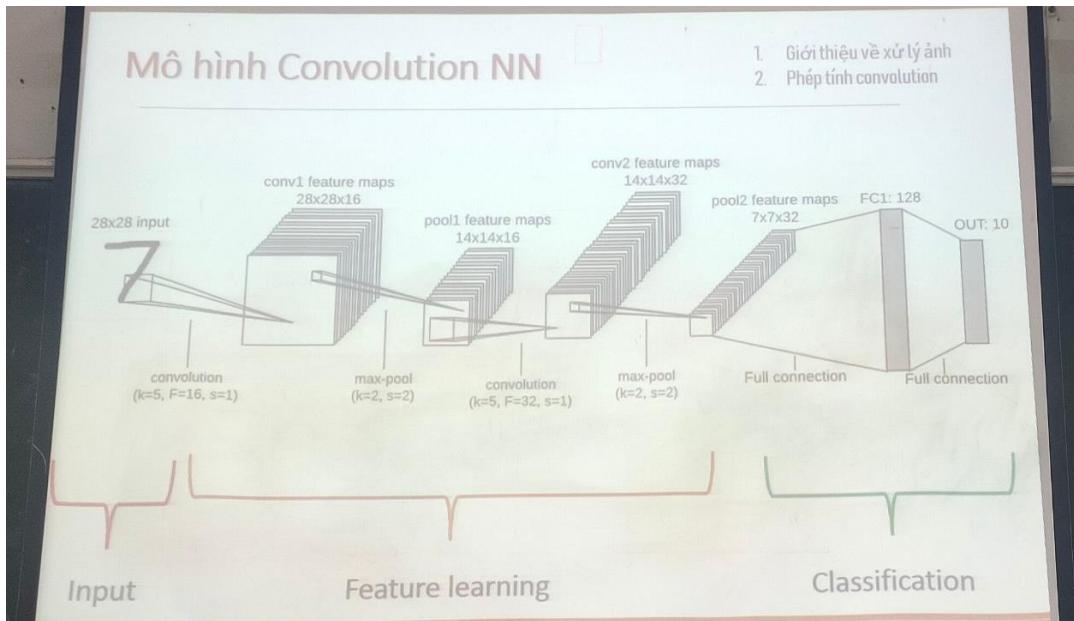


Cuối cùng là hàm RELU và Leaky RELU.



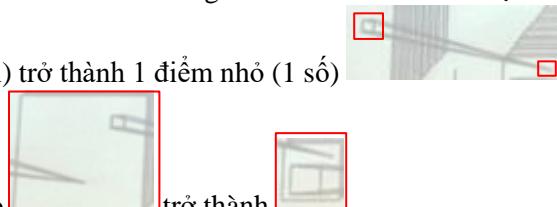
Phần 4 : Convolutional Neural Network (Mạng nơ ron tích chập)

Overview CNN



Giải thích kiến trúc :

- Input**
 - Hình ảnh kích thước 28*28*1 (ảnh grey nên nó là 1 ma trận 28*28 nên không phải là ảnh màu tensor 3 channel) (ma trận 28*28 hoặc có thể gọi là tensor 28*28*1)
- Tích chập với ($k=5, F=16, s=1$)
 - Lưu ý :** Ở đây ta có một lưu ý là input đầu vào 28 và đi qua kernel vẫn giữ nguyên 28 => có sử dụng padding . Công thức là : $p = \frac{k-1}{2}$, vậy padding ở đây là $p = 2$
 - $K=5 \Rightarrow$ chính là k của kernel : $k*k*3 = 5*5*3$ (ở đây ta có X là $m*n = 28*28$ nên $k*k = 5*5$ nhỏ hơn 28*28 là hoàn toàn được)
 - $F=16$ (số lượng kernel là 16) F ở đây chính là W (kernel) (một số tài liệu gọi kernel W chính là feature F)(Tuy nhiên kernel F này chưa tối ưu , vẫn đang còn random)(Ở đây ta cũng có thể thêm một bước tối ưu kernel)
 - Cho X này qua 16 kernel ($5*5*3$) sẽ thu được 16 X mới ta thấy trong hình là 28*28*16
 - S trong hình là stride = 1 => bước nhảy khi cho qua kernel là 1
 - Những thanh này là tích chập cho nó nhỏ xuống => Chính là nhân ma trận W cho nó gộp lại . Từ một ô vuông (ma trận) trở thành 1 điểm nhỏ (1 số)

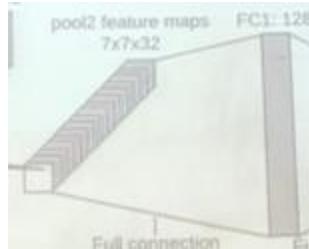


- Dẫn đến ma trận to thành nhỏ
- Vậy theo nhu công thức ở dưới thay vào ta có

Input : 28*28*1

Qua kernel ($k=5, F=16, s=1, p = \frac{k-1}{2} = 2$) thì thu được tensor có kích thước
 $(\frac{28-5+2*2}{1} + 1) * (\frac{28-5+2*2}{1} + 1) * F(kernel) = 28 * 28 * 16$

- Max-pool ($k=2, s=2$)
 - Sau đó cho qua max-pool để thu nhỏ kích thước ma vẫn giữ được đặt trung của ảnh
 - Kích thước pooling là $k=2, k*k = 2*2$ và số bước nhảy stride là $2 \Rightarrow$ giải kích thước ban đầu xuống $\frac{1}{2} \Rightarrow 28$ xuống còn 14
- Tích chập lần 2 ($k=5, F=32, S=1$)
 - Tương tự, lần này ta thấy cho qua 32 Kernel \Rightarrow số lượng tạo ra gấp đôi so với $F=16$ (như hình)
 - Input là : $14*14*16$
 - Qua kernel ($k=5, F=32, s=1, p = \frac{k-1}{2} = 2$) thì thu được tensor có kích thước
 $(\frac{14-5+2*2}{1} + 1) * (\frac{14-5+2*2}{1} + 1) * F(kernel) = 14 * 14 * 32$
- Cho qua max-pool ($k=2, s=2$)
 - Kích thước pooling là $k=2, k*k = 2*2$ và số bước nhảy stride là $2 \Rightarrow$ giải kích thước ban đầu xuống $\frac{1}{2} \Rightarrow 14$ xuống còn 7
- **Flattening (trong hình thiêu)**
 - Thông qua Flattening kích thước $H*W*D$, sẽ được chuyển về 1 vector kích thước $(H*W*D, 1)$. Ta có $7*7*32$ là 32 ma trận $7*7$ trở thành vector có kích thước $(1568, 1)$



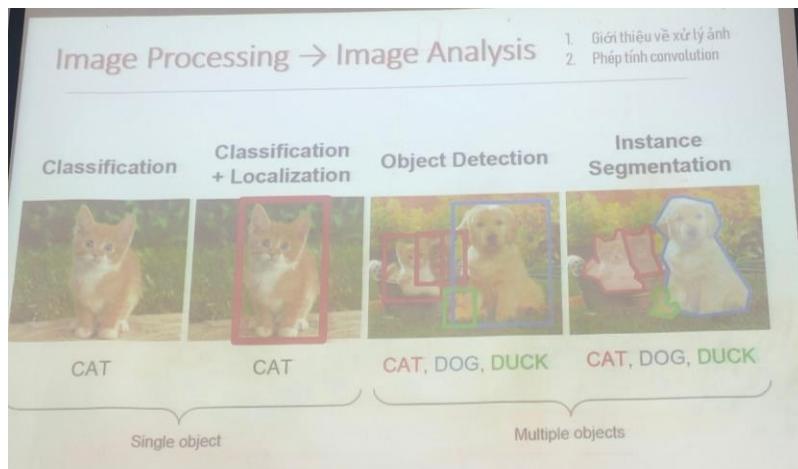
◦ Các hình vuông sẽ bẻ thành một đường thẳng và là đầu vào của mạng neural network

- Fully connected 1
 - Input là vector $(1568, 1)$ và output là vector $(128, 1)$
- Fully connected 2
 - Input là vector $(128, 1)$ và output là vector $(10, 1)$
- ❖ Công thức tổng quát :

Công thức tổng quát cho phép tính convolution của ma trận X kích thước $m*n$ với kernel kích thước $k*k$, stride = s , padding = p ra ma trận Y kích thước $(\frac{m-k+2p}{s} + 1) * (\frac{n-k+2p}{s} + 1)$.

p : padding , k : kích thước của kernel : $k*k$, m,n là ma trận đầu vào, s là stride .

- ❖ Lưu ý : Đối với ảnh màu thì như trên ví dụ sẽ là : $28*28*3$ (3 channel r,g,b trên 1 ảnh màu)
 Từ đó dẫn đến các xử lí ở phía sau đều $*3$ lên .



- Detect object (phát hiện đối tượng)
- Phân lớp (Classification)
- Tầng trước là đầu vào của tầng sau

Vector đặc trưng và One-host

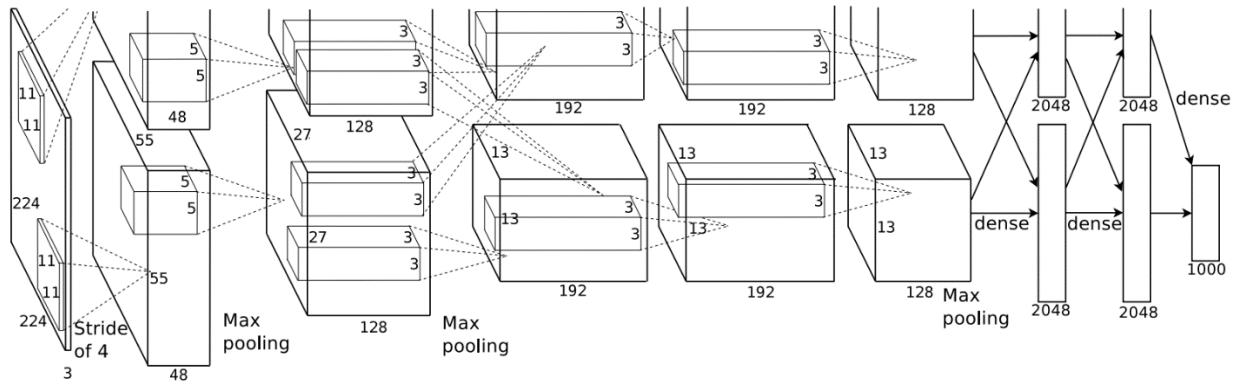
- **Khái niệm**
 - o **Vector đặc trưng :**
 - Đây là kích thước của đầu ra sau khi đã làm phẳng (flattening) các đặc trưng từ lớp tích chập và pooling trong mạng CNN. Ex : (1568,1) như ví dụ trên.
 - Vector đặc trưng (feature vector) là một vector chứa các giá trị biểu diễn đặc trưng của đối tượng trong ảnh. Trong CNN, sau khi thông qua các lớp tích chập và pooling, đầu ra của mỗi nút ẩn (hidden node) sẽ tạo ra một phần của vector đặc trưng. Khi đi qua các lớp fully connected, các phần tử của vector đặc trưng này được kết hợp lại để tạo ra một đầu ra cuối cùng dưới dạng one-hot vector, thường là để dự đoán lớp của đối tượng trong ảnh.
 - o **One-host :**
 - Là vector thu được khi cho Vector đặc trưng qua tiếp các lớp Fully connected .
 - Sau khi thông qua lớp fully connected (hoặc dense layer), các đầu ra được biểu diễn dưới dạng các one-hot vector. Mỗi one-hot vector có độ dài bằng số lớp đầu ra của mạng, và chỉ có một phần tử trong vector có giá trị là 1, các phần tử còn lại đều là 0. Phần tử có giá trị 1 đại diện cho lớp được dự đoán là lớp của đối tượng trong ảnh.
 - Ví dụ, nếu chúng ta có một mạng CNN để nhận diện chữ số từ 0 đến 9, và đầu ra của mạng có 10 lớp (do có 10 chữ số), thì sau khi đi qua các lớp fully connected, đầu ra sẽ là một vector có độ dài 10. Trong vector này, phần tử thứ i sẽ có giá trị 1 nếu mạng dự đoán đối tượng trong ảnh là số i, và tất cả các phần tử khác sẽ có giá trị 0.
- **Trường hợp sử dụng**
 - o **Vector đặc trưng :**
 - So khớp các vector bằng phương pháp như **Support Vector Machine (SVM)**
 - **Sử dụng trong bài toán phân loại nhị phân (Có hoặc Không)**

- Trong nhận diện khuôn mặt, mỗi khuôn mặt được coi là một thực thể duy nhất và việc nhận diện đúng khuôn mặt đó có thể coi là một vấn đề phân loại nhị phân (có hoặc không có khuôn mặt). Trong trường hợp này, bạn không cần sử dụng one-hot encoding cho vector đặc trưng của khuôn mặt. Thay vào đó, bạn có thể sử dụng các phương pháp như Support Vector Machine (SVM) hoặc kỹ thuật so sánh vector đặc trưng của khuôn mặt đang xét với các vector đặc trưng trong cơ sở dữ liệu để nhận diện.
 - **One-host**
 - **Sử dụng trong bài toán nhận diện thuộc lớp nào (Chắc chắn có nhưng phải xác định nó thuộc lớp nào) . Nghĩa là ta biết nó là một số thuộc từ 0 đến 9 nhưng chưa biết nó thuộc số nào , việc ta cần làm là phân nó và lớp nào từ 0 đến 9. Trong khi nhận diện khuôn mặt thì ta chưa biết khuôn mặt đó có trong database hay chưa .**
 - Trong bài toán nhận diện chữ số từ 0 đến 9, mỗi chữ số được coi là một lớp riêng biệt. Do đó, để huấn luyện một mô hình phân loại chữ số, bạn cần sử dụng one-hot encoding cho vector đặc trưng của mỗi chữ số. Khi dùng one-hot encoding, mỗi chữ số sẽ được biểu diễn bằng một vector có độ dài bằng số lượng chữ số (tức là 10), trong đó chỉ có một thành phần có giá trị 1, tương ứng với lớp của chữ số đó.
- ⇒ Tóm lại, việc sử dụng one-hot encoding hay không phụ thuộc vào bài toán cụ thể bạn đang giải quyết và cách bạn định nghĩa các lớp hoặc thực thể trong bài toán đó. Trong trường hợp nhận diện khuôn mặt, việc sử dụng one-hot encoding thường không cần thiết, trong khi đối với bài toán nhận diện chữ số, one-hot encoding là một phần quan trọng của quá trình phân loại.

Nói thêm :

- **Các thuật ngữ mà bạn đề cập là các loại mô hình học máy và mạng nơ-ron phổ biến. Dưới đây là ý nghĩa của mỗi loại:**
 - 1. **NN (Neural Network - Mạng Nơ-ron):**** Một mô hình toán học được lấy cảm hứng từ cách hoạt động của não người. Nó bao gồm các lớp nơ-ron được tổ chức thành các lớp, trong đó mỗi nơ-ron kết nối với các nơ-ron trong các lớp liền kề.
 - 2. **RNN (Recurrent Neural Network - Mạng Nơ-ron Tái phát):**** Một loại mạng nơ-ron mà có khả năng xử lý dữ liệu chuỗi, vì chúng giữ lại trạng thái trước đó thông qua các chu kỳ lặp. Điều này làm cho chúng phù hợp cho việc xử lý dữ liệu có cấu trúc chuỗi như ngôn ngữ tự nhiên hoặc dữ liệu thời gian.
 - 3. **KNN (K-Nearest Neighbors - K-Hàng Xóm Gần Nhất):**** Một thuật toán học máy đơn giản được sử dụng cho cả phân loại và hồi quy. Nó hoạt động bằng cách tính khoảng cách giữa mỗi điểm dữ liệu và các điểm dữ liệu đã biết, sau đó chọn ra K điểm gần nhất và dùng chúng để dự đoán nhãn hoặc giá trị của điểm dữ liệu đang xét.
 - 4. **CNN (Convolutional Neural Network - Mạng Nơ-ron Tích Chập):**** Một loại mạng nơ-ron thường được sử dụng cho việc phân tích ảnh và video. Nó sử dụng các lớp tích chập để trích xuất các đặc trưng cấp thấp từ dữ liệu đầu vào, giúp nó nhận biết các mẫu và cấu trúc phức tạp trong dữ liệu ảnh. CNN đã chứng minh được hiệu quả trong nhiều ứng dụng như nhận diện ảnh, nhận dạng ký tự và xe, nhận dạng khuôn mặt, và nhiều ứng dụng khác.

AlexNet Model Architecture



Tham khảo :

[Calculator-CNN-Archilecture.xlsx](#)

Quá trình tính toán

Công thức

- Convolutitional layer

| Conv : F kernel (k*k*D) | | | | |
|-------------------------|---|---|------------|-------------|
| F | k | D | s (stride) | p (padding) |
| | | | | |

- Max-Pooling

| Max-pooling : k*k | | |
|-------------------|------------|-------------|
| k | s (stride) | p (padding) |
| | | |

- Input X (W,H,D) sau khi cho qua Conv

$$W_{new} = \frac{W - k + 2 * p}{s} + 1$$

$$H_{new} = \frac{H - k + 2 * p}{s} + 1$$

$$D_{new} = F$$

Lưu ý

- o $p = 0$ if p is “valid”

- o $p = “same”$

$$W_{new} = W$$

$$H_{new} = H$$

$$D_{new} = F$$

Trong đó

W: Chiều rộng của ma trận đầu vào.

H: Chiều cao của ma trận đầu vào.

D: Số lượng kênh (channel) của ma trận đầu vào.

F: Số lượng kernel (filter).

k: Kích thước của kernel (cùng chiều rộng và chiều cao).

- s: Stride của max pooling.
- p: Padding của max pooling.
- o Lưu ý : Ta sẽ thấy khi cho X qua Conv : $k*k*D$ thì D thường sẽ bằng với D của input X (chính là F của Conv trước đó)

- Input X (W,H,D) sau khi cho qua Pooling

$$W_{new} = \frac{W - k + 2 * p}{s} + 1$$

$$H_{new} = \frac{H - k + 2 * p}{s} + 1$$

$$D_{new} = D$$

Lưu ý

- o $p = 0$ if p is “valid”

- o $p = “same”$

$$W_{new} = W$$

$$H_{new} = H$$

$$D_{new} = D$$

- Công thức tính tham số

+ Input X (W,H,D_x) cho qua Conv : F kernel (k,k,D_{conv}) . Thường thì D_x = D_{conv}

Bias = F

$$\text{Parameters} = \text{weight} + \text{bias} = k * k * D * F + F = (k * k * D + 1) * F$$

+ Input khi cho Vector(N,1) qua Fully connected(M)

Bias = M

$$\text{Parameters} = \text{weight} + \text{bias} = N * M + M$$

- **Nói thêm :**

Ta thấy : $k * k * D * F \Rightarrow$ đây chính là kích thước của kernel Conv nếu $D_x = D_{conv}$ (thường là $D_x = D_{conv}$)

Tính toán

- Input tensor : 224*224*3
- Cho Input X (W, H, D) = (224,224,3) qua Conv1(F=96,k*k=11*11,D=3,s=4,p=valid)

$$\left(\frac{W - k + 2 * p}{s} + 1, \frac{W - k + 2 * p}{s} + 1, F \right) = \left(\frac{224 - 11 + 2 * 0}{4} + 1, \frac{224 - 11 + 2 * 0}{4} + 1, 96 \right) = (55, 55, 96)$$

$$\text{Parameters} = \text{weight} + \text{bias} = k * k * D * F + F = 11 * 11 * 3 * 96 + 96 = 34944$$
- Cho Input X (W, H, D) = (55, 55, 96) qua Max-Pooling (k=3,s=2,p=valid)

$$\left(\frac{W - k + 2 * p}{s} + 1, \frac{W - k + 2 * p}{s} + 1, D \right) = \left(\frac{55 - 3 + 2 * 0}{2} + 1, \frac{55 - 3 + 2 * 0}{2} + 1, 96 \right) = (27, 27, 96)$$
- Cho Input X (W, H, D) = (27, 27, 96) qua Conv2(F=256,k*k=5*5,D=96,s=1,p=same)

$$(W, H, F) = (27, 27, 256)$$

$$\text{Parameters} = \text{weight} + \text{bias} = k * k * D * F + F = 5 * 5 * 96 * 256 + 256 = 614656$$
- Cho Input X (W, H, D) = (27, 27, 256) qua Max-Pooling2 (k=3,s=2,p=valid)

$$\left(\frac{W - k + 2 * p}{s} + 1, \frac{W - k + 2 * p}{s} + 1, D \right) = \left(\frac{27 - 3 + 2 * 0}{2} + 1, \frac{27 - 3 + 2 * 0}{2} + 1, 256 \right) = (13, 13, 256)$$

- Cho Input X (W, H, D) = (13, 13, 256) qua Conv3(F=384,k*k=3*3,D=256,s=1,p=same)
 $(W, H, F) = \mathbf{(13, 13, 384)}$
Parameters = weight + bias = $k * k * D * F + F = 3 * 3 * 256 * 384 + 384 = \mathbf{885120}$
- Cho Input X (W, H, D) = (13, 13, 384) qua Conv4(F=384,k*k=3*3,D=384,s=1,p=same)
 $(W, H, F) = \mathbf{(13, 13, 384)}$
Parameters = weight + bias = $k * k * D * F + F = 3 * 3 * 384 * 384 + 384 = \mathbf{1327488}$
- Cho Input X (W, H, D) = (13, 13, 384) qua Conv2(F=256,k*k=3*3,D=384,s=1,p=same)
 $(W, H, F) = \mathbf{(13, 13, 256)}$
Parameters = weight + bias = $k * k * D * F + F = 3 * 3 * 384 * 256 + 256 = \mathbf{884992}$
- Cho Input X (W, H, D) = (13, 13, 256) qua Max-Pooling3 (k=3,s=2,p=valid)
 $\left(\frac{W - k + 2 * p}{s} + 1, \frac{W - k + 2 * p}{s} + 1, D\right) = \left(\frac{13 - 3 + 2 * 0}{2} + 1, \frac{13 - 3 + 2 * 0}{2} + 1, 256\right) = \mathbf{(6, 6, 256)}$
- Cho Input X (W, H, D) = (6, 6, 256) qua Flatten thu được Vector (6*6*256,1) = **(9216,1)**
- Cho Input Vector X(9216,1) qua Fully connected1 (4096) thu được Vector **(4096,1)**
Parameters = weight + bias = $9216 * 4096 + 4096 = \mathbf{37752832}$
- Cho Input Vector X(4096,1) qua Fully connected2 (4096) thu được Vector **(4096,1)**
Parameters = weight + bias = $4096 * 4096 + 4096 = \mathbf{16781312}$
- Cho Input Vector X(4096,1) qua Fully connected3 (1000) thu được Vector **(1000,1)**
Parameters = weight + bias = $4096 * 1000 + 1000 = \mathbf{4097000}$

Tổng số parameters = $34944 + 614656 + 885120 + 1327488 + 884992 + 37752832 + 16781312 + 4097000$
= **62387344**

Kiến trúc thu gọn

| Layer (type) | Output Shape | Param # |
|-------------------------------|---------------------|----------|
| conv2d_1 (Conv2D) | (None, 55, 55, 96) | 34944 |
| max_pooling2d_1 (MaxPooling2) | (None, 27, 27, 96) | 0 |
| conv2d_2 (Conv2D) | (None, 27, 27, 256) | 614656 |
| max_pooling2d_2 (MaxPooling2) | (None, 13, 13, 256) | 0 |
| conv2d_3 (Conv2D) | (None, 13, 13, 384) | 885120 |
| conv2d_4 (Conv2D) | (None, 13, 13, 384) | 1327488 |
| conv2d_5 (Conv2D) | (None, 13, 13, 256) | 884992 |
| max_pooling2d_3 (MaxPooling2) | (None, 6, 6, 256) | 0 |
| flatten_1 (Flatten) | (None, 9216) | 0 |
| dense_1 (Dense) | (None, 4096) | 37752832 |

| | | |
|---------------------|--------------|----------|
| dropout_1 (Dropout) | (None, 4096) | 0 |
| dense_2 (Dense) | (None, 4096) | 16781312 |
| dropout_2 (Dropout) | (None, 4096) | 0 |
| dense_3 (Dense) | (None, 1000) | 4097000 |

Chi tiết kiến trúc

| Layer (type) | Conv/Pooling | | | | | Output Shape | Param # |
|---------------------------------|--------------|---------|-----------------|------------------|--------------------------|---------------------|----------|
| conv2d_1 (Conv2D) | F
96 | k
11 | D
3 | s (stride)
4 | p (padding)
0 (valid) | (None, 55, 55, 96) | 34944 |
| max_pooling2d_1
(MaxPooling2 | | k
3 | s (stride)
2 | p (padding)
0 | | (None, 27, 27, 96) | 0 |
| conv2d_2 (Conv2D) | F
256 | k
5 | D
96 | s (stride)
1 | p (padding)
same | (None, 27, 27, 256) | 614656 |
| max_pooling2d_2
(MaxPooling2 | | k
3 | s (stride)
2 | p (padding)
0 | | (None, 13, 13, 256) | 0 |
| conv2d_3 (Conv2D) | F
384 | k
3 | D
256 | s (stride)
1 | p (padding)
same | (None, 13, 13, 384) | 885120 |
| conv2d_4 (Conv2D) | F
384 | k
3 | D
384 | s (stride)
1 | p (padding)
same | (None, 13, 13, 384) | 1327488 |
| conv2d_5 (Conv2D) | F
256 | k
3 | D
384 | s (stride)
1 | p (padding)
same | (None, 13, 13, 256) | 884992 |
| max_pooling2d_3
(MaxPooling2 | | k
3 | s (stride)
2 | p (padding)
0 | | (None, 6, 6, 256) | 0 |
| flatten_1 (Flatten) | | | | | | (None, 9216) | 0 |
| dense_1 (Dense) | | | | | | (None, 4096) | 37752832 |
| dropout_1 (Dropout) | | | | | | (None, 4096) | 0 |
| dense_2 (Dense) | | | | | | (None, 4096) | 16781312 |
| dropout_2 (Dropout) | | | | | | (None, 4096) | 0 |
| dense_3 (Dense) | | | | | | (None, 1000) | 4097000 |

Total params: 62,378,344

Trainable params: 62,378,344

Non-trainable params: 0

Parameters (Tham số)

| Layer Name | Tensor Size | Weights | Biases | Parameters |
|--------------|-------------|------------|--------|-------------------|
| Input Image | 227x227x3 | 0 | 0 | 0 |
| Conv-1 | 55x55x96 | 34,848 | 96 | 34,944 |
| MaxPool-1 | 27x27x96 | 0 | 0 | 0 |
| Conv-2 | 27x27x256 | 614,400 | 256 | 614,656 |
| MaxPool-2 | 13x13x256 | 0 | 0 | 0 |
| Conv-3 | 13x13x384 | 884,736 | 384 | 885,120 |
| Conv-4 | 13x13x384 | 1,327,104 | 384 | 1,327,488 |
| Conv-5 | 13x13x256 | 884,736 | 256 | 884,992 |
| MaxPool-3 | 6x6x256 | 0 | 0 | 0 |
| FC-1 | 4096×1 | 37,748,736 | 4,096 | 37,752,832 |
| FC-2 | 4096×1 | 16,777,216 | 4,096 | 16,781,312 |
| FC-3 | 1000×1 | 4,096,000 | 1,000 | 4,097,000 |
| Output | 1000×1 | 0 | 0 | 0 |
| Total | | | | 62,378,344 |

| Layer (type) | Output Shape | Param # |
|--------------------------------|---------------------|----------|
| <hr/> | | |
| conv2d_1 (Conv2D) | (None, 55, 55, 96) | 34944 |
| <hr/> | | |
| max_pooling2d_1 (MaxPooling2D) | (None, 27, 27, 96) | 0 |
| <hr/> | | |
| conv2d_2 (Conv2D) | (None, 27, 27, 256) | 614656 |
| <hr/> | | |
| max_pooling2d_2 (MaxPooling2D) | (None, 13, 13, 256) | 0 |
| <hr/> | | |
| conv2d_3 (Conv2D) | (None, 13, 13, 384) | 885120 |
| <hr/> | | |
| conv2d_4 (Conv2D) | (None, 13, 13, 384) | 1327488 |
| <hr/> | | |
| conv2d_5 (Conv2D) | (None, 13, 13, 256) | 884992 |
| <hr/> | | |
| max_pooling2d_3 (MaxPooling2D) | (None, 6, 6, 256) | 0 |
| <hr/> | | |
| flatten_1 (Flatten) | (None, 9216) | 0 |
| <hr/> | | |
| dense_1 (Dense) | (None, 4096) | 37752832 |
| <hr/> | | |
| dropout_1 (Dropout) | (None, 4096) | 0 |
| <hr/> | | |
| dense_2 (Dense) | (None, 4096) | 16781312 |
| <hr/> | | |
| dropout_2 (Dropout) | (None, 4096) | 0 |
| <hr/> | | |
| dense_3 (Dense) | (None, 1000) | 4097000 |
| <hr/> | | |
| Total params: | 62,378,344 | |
| Trainable params: | 62,378,344 | |
| Non-trainable params: | 0 | |

7. Giới thiệu về xử lý ảnh

Ảnh màu

- Hệ ảnh gồm
 - Màu rgb (red,green,blue)

- Màu rgb-alpha (red,green,blue,opacity) (độ trong suốt) (ngoài ra còn có màu **hsl**)
- Các giá trị rgb nằm trong khoảng [0,255]



Vậy bức ảnh trên kích thước 800 pixel * 600 pixel, có thể biểu diễn dưới dạng một ma trận kích thước 600 * 800 (vì định nghĩa ma trận là số hàng nhân số cột).

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,800} \\ w_{2,1} & w_{2,2} & \dots & w_{2,800} \\ \dots & \dots & \dots & \dots \\ w_{600,1} & w_{600,2} & \dots & w_{600,800} \end{bmatrix}$$

Trong đó mỗi phần tử w_{ij} là một pixel. Như vậy có thể hiểu là mỗi pixel thì biểu diễn một màu và bức ảnh trên là sự kết hợp rất nhiều pixel. Hiểu đơn giản thì in bức ảnh ra, kẻ ô vuông như chơi cờ ca rô với 800 đường thẳng ở chiều dài, 600 đường ở chiều rộng, **thì mỗi ô vuông là một pixel**, biểu diễn một chấm màu.

Tuy nhiên để biểu diễn 1 màu ta cần 3 thông số (r,g,b) nên gọi $w_{ij} = (r_{ij}, g_{ij}, b_{ij})$ để biểu diễn dưới dạng ma trận thì sẽ như sau:

$$\begin{bmatrix} (100, 100, 50) & (101, 112, 3) & (131, 20, 80) \\ (150, 210, 130) & (10, 120, 130) & (111, 120, 130) \\ (10, 260, 30) & (200, 20, 30) & (100, 20, 3) \end{bmatrix}$$

Để tiện lưu trữ và xử lý không thể lưu trong 1 ma trận như thế kia mà sẽ tách mỗi giá trị màu trong mỗi pixel ra một ma trận riêng.

$$\begin{bmatrix} 100 & 101 & 131 \\ 150 & 10 & 111 \\ 10 & 200 & 100 \end{bmatrix}, \begin{bmatrix} 100 & 112 & 20 \\ 210 & 120 & 120 \\ 260 & 20 & 20 \end{bmatrix}, \begin{bmatrix} 50 & 3 & 80 \\ 130 & 130 & 130 \\ 30 & 30 & 3 \end{bmatrix}$$

R **G** **B**

Tổng quát

$$\begin{bmatrix} (r_{1,1}, g_{1,1}, b_{1,1}) & (r_{1,2}, g_{1,2}, b_{1,2}) & \dots & (r_{1,800}, g_{1,800}, b_{1,800}) \\ (r_{2,1}, g_{2,1}, b_{2,1}) & (r_{2,2}, g_{2,2}, b_{2,2}) & \dots & (r_{2,800}, g_{2,800}, b_{2,800}) \\ \dots & \dots & \dots & \dots \\ (r_{600,1}, g_{600,1}, b_{600,1}) & (r_{600,2}, g_{600,2}, b_{600,2}) & \dots & (r_{600,800}, g_{600,800}, b_{600,800}) \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} r_{1,1} & r_{1,2} & \dots & r_{1,800} \\ r_{2,1} & r_{2,2} & \dots & r_{2,800} \\ \dots & \dots & \dots & \dots \\ r_{600,1} & r_{600,2} & \dots & r_{600,800} \end{bmatrix}, \begin{bmatrix} g_{1,1} & g_{1,2} & \dots & g_{1,800} \\ g_{2,1} & g_{2,2} & \dots & g_{2,800} \\ \dots & \dots & \dots & \dots \\ g_{600,1} & g_{600,2} & \dots & g_{600,800} \end{bmatrix}, \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,800} \\ b_{2,1} & b_{2,2} & \dots & b_{2,800} \\ \dots & \dots & \dots & \dots \\ b_{600,1} & b_{600,2} & \dots & b_{600,800} \end{bmatrix},$$

Hình 7.6: Tách ma trận biểu diễn màu ra 3 ma trận, mỗi ma trận lưu giá trị 1 màu.

Tensor là gì ?

- Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột.
- Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng * số cột.
- Khi dữ liệu nhiều hơn 2 chiều thì sẽ được gọi là tensor, ví dụ như dữ liệu có 3 chiều.

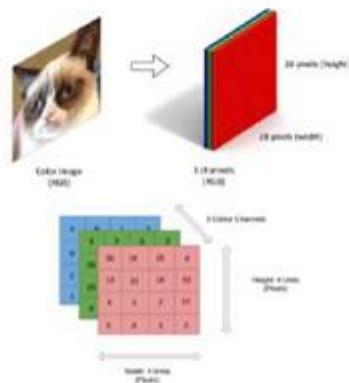
Thì tensor 3 chiều cũng là sự kết hợp của các ma trận cùng kích thước, xếp k ma trận kích thước m*n lên nhau sẽ được tensor kích thước m*n*k.

Tưởng tượng mặt đáy là một ma trận kích thước $a * b$, được tạo bởi b vector kích thước a . Cả hình hộp là tensor 3 chiều kích thước $a*b*h$, được tạo bởi xếp h ma trận kích thước $a*b$ lên nhau.

Do đó biểu diễn ảnh màu trên máy tính ở phần trên sẽ được biểu diễn dưới dạng tensor 3 chiều kích thước $600*800*3$ do có 3 ma trận (channel) màu red, green, blue kích thước $600*800$ chồng lên nhau.

Ví dụ biểu diễn ảnh màu kích thước $28*28$, biểu diễn dưới dạng tensor $28*28*3$

color image is 3rd-order tensor



Hình 7.9: Ảnh màu biểu diễn dưới dạng tensor [1]

Ex : tensor có : $600*800*3 \Rightarrow$ có 3 ma trận r , g , b chồng lên nhau . Mỗi ma trận có kích thước là $600*800$ (đây cũng là kích thước pixcel) . Mỗi ma trận là tập hợp $600*800$ phần tử trên 600 hàng và 800 cột .

Ảnh xám



Tương tự ảnh màu, ảnh xám cũng có kích thước 800 pixel * 600 pixel, có thể biểu diễn dưới dạng một ma trận kích thước 600 * 800 (vì định nghĩa ma trận là số hàng nhân số cột).

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,800} \\ w_{2,1} & w_{2,2} & \dots & w_{2,800} \\ \dots & \dots & \dots & \dots \\ w_{600,1} & w_{600,2} & \dots & w_{600,800} \end{bmatrix}$$

- **Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ [0,255] thay vì (r,g,b) như trong ảnh màu. Do đó khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ**

$$\begin{bmatrix} 0 & 215 & \dots & 250 \\ 12 & 156 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ 244 & 255 & \dots & 12 \end{bmatrix}$$

Hình 7.11: Biểu diễn ảnh xám

- **Giá trị 0 là màu đen, 255 là màu trắng và giá trị pixel càng gần 0 thì càng tối và càng gần 255 thì càng sáng.**

Chuyển hệ màu của ảnh

- Mỗi pixel trong ảnh màu được biểu diễn bằng 3 giá trị (r,g,b) còn trong ảnh xám chỉ cần 1 giá trị x để biểu diễn.
- Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng công thức: $x = r * 0.299 + g * 0.587 + b * 0.114$.
- Tuy nhiên khi chuyển ngược lại, bạn chỉ biết giá trị x và cần đi tìm r,g,b nên sẽ không chính xác.

Phép tính convolution

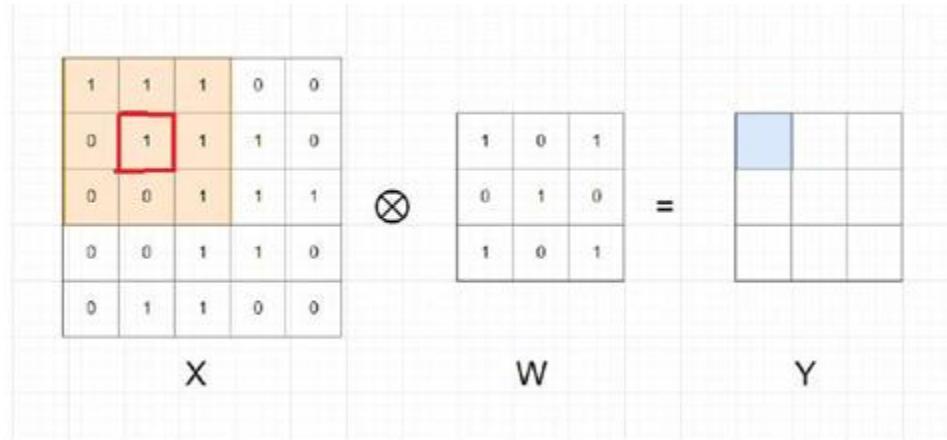
Convolution

Giả sử ta xử lý trên : ẢNH GREY (ảnh xám) (ma trận m*n)

Ta định nghĩa kernel là một ma trận vuông kích thước $k*k$ trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9,...

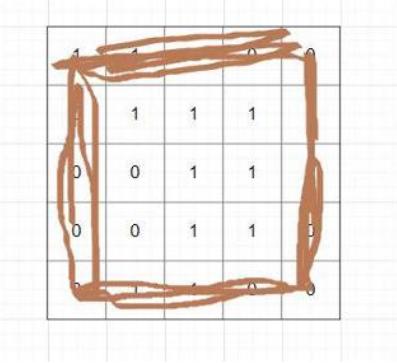
Ví dụ kernel kích thước $3*3$

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



Ví dụ khi tính tại x_{22} (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W , có x_{22} làm trung tâm có màu nền da cam như trong hình. Sau đó tính $y_{11} = \text{sum}(A \otimes W) = x_{11}*w_{11}+x_{12}*w_{12}+x_{13}*w_{13}+x_{21}*w_{21}+x_{22}*w_{22}+x_{23}*w_{23}+x_{31}*w_{31}+x_{32}*w_{32}+x_{33}*w_{33} = 4$. Và làm tương tự với các phần tử còn lại trong ma trận.

Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như x_{11} ? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X.



Nên bạn để ý thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là $(m-k+1) * (n-k+1)$.

- ❖ W có kích thước là $k*k$
- ❖ X có kích thước là $m*n$
- ❖ Đến đến ma trận Y có kích thước là $(m-k+1) * (n-k+1)$.

| | | | | |
|---|---|---|---|---|
| 1
<small>$\times 1$</small> | 1
<small>$\times 0$</small> | 1
<small>$\times 1$</small> | 0 | 0 |
| 0
<small>$\times 0$</small> | 1
<small>$\times 1$</small> | 1
<small>$\times 0$</small> | 1 | 0 |
| 0
<small>$\times 1$</small> | 0
<small>$\times 0$</small> | 1
<small>$\times 1$</small> | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |

Image

| | | |
|---|--|--|
| 4 | | |
| | | |
| | | |
| | | |

Convolved Feature

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

W

Ví dụ :

- Lưu ý : Có tài liệu thì gọi kernel là feature

Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X \Rightarrow Tìm cách giải quyết cho các

phản tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Hình 7.13: Ma trận X khi thêm viền 0 bên ngoài

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phản tử x_{11} , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu.

- ❖ Phép tính này gọi là **convolution** với padding=1. Padding=k nghĩa là thêm k vector 0 vào mỗi phía (trên, dưới, trái, phải) của ma trận.

Stride

Như ở trên ta thực hiện tuần tự các phản tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Hình 7.14: stride=1, padding=1

Tuy nhiên nếu $\text{stride}=k$ ($k > 1$) thì ta chỉ thực hiện phép tính convolution trên các phần tử $x_{1+i*k, 1+j*k}$. Ví dụ $k=2$.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Hình 7.15: padding=1, stride=2

Hiểu đơn giản là bắt đầu từ vị trí x_{11} sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X.

Kích thước của ma trận Y là 3*3 đã giảm đi đáng kể so với ma trận X.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước $m*n$ với kernel kích thước $k*k$, stride = s, padding = p ra ma trận Y kích thước $(\frac{m-k+2p}{s}+1) * (\frac{n-k+2p}{s}+1)$.

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

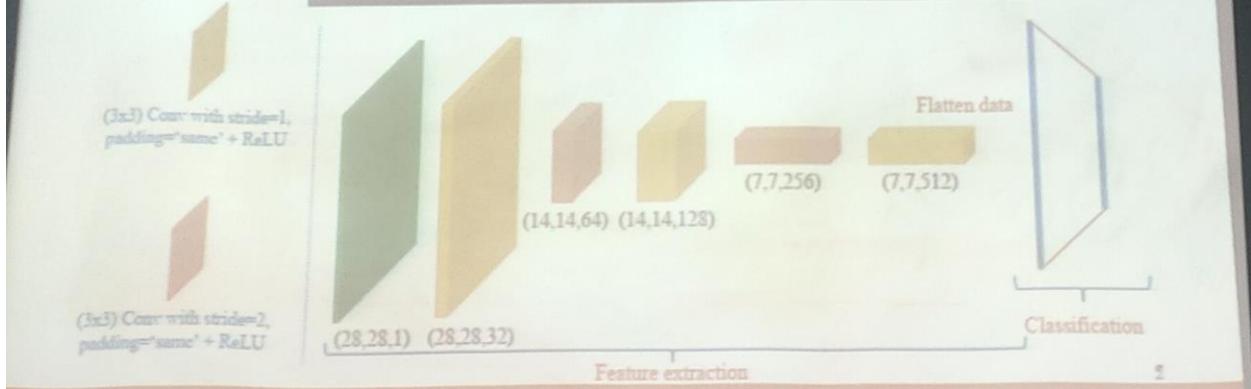
Mọi người có thể xem thêm trực quan hơn ở [đây](#).

2. Phép tính convolution

1. Giới thiệu về xử lý ảnh
2. Phép tính convolution

Example

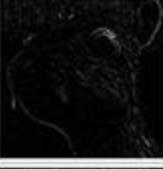
```
model = nn.Sequential(  
    nn.Conv2d(1, 32, kernel_size=3, padding='same', stride=1), nn.ReLU(),  
    nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2), nn.ReLU(),  
    nn.Conv2d(64, 128, kernel_size=3, padding=1, stride=1), nn.ReLU(),  
    nn.Conv2d(128, 256, kernel_size=3, padding=1, stride=2), nn.ReLU(),  
    nn.Conv2d(256, 512, kernel_size=3, padding=1, stride=1), nn.ReLU(),  
    nn.Flatten(), nn.Linear(7*7*512, 10)  
)
```



⇒ Nếu dùng thư viện thì same chính là thêm padding ở ngoài

Ý nghĩa của phép tính convolution

⇒ Mục đích của phép tính convolution trên ảnh là làm mờ, làm nét ảnh; xác định các đường;... Mỗi kernel khác nhau thì sẽ phép tính convolution sẽ có ý nghĩa khác nhau. Ví dụ:

| Operation | Kernel ω | Image result $g(x,y)$ |
|--------------------------|---|--|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Box blur
(normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |

Hình 7.16: Một số kernel phổ biến [12]

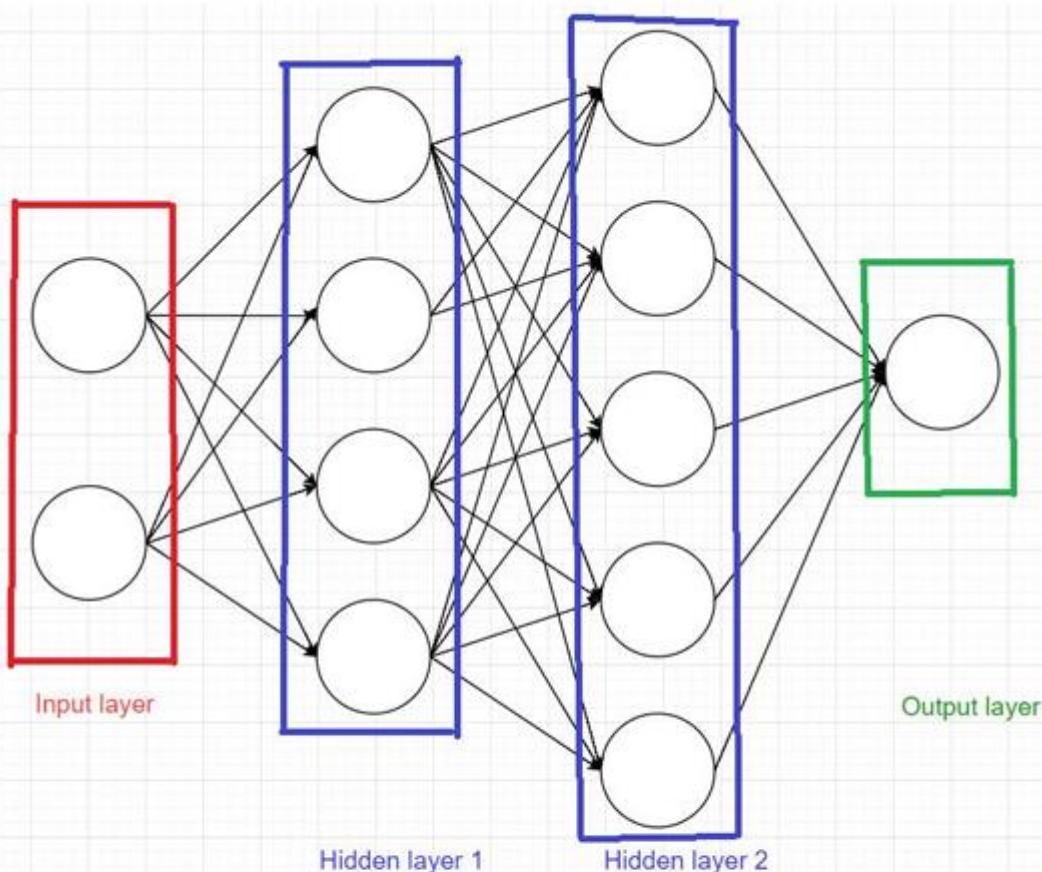
8. Convolutional Neural Network

Convolutional neural network

Tiền xử lí : Các ảnh phải cùng kích thước

Convolutional layer

Mô hình neural network từ những bài trước

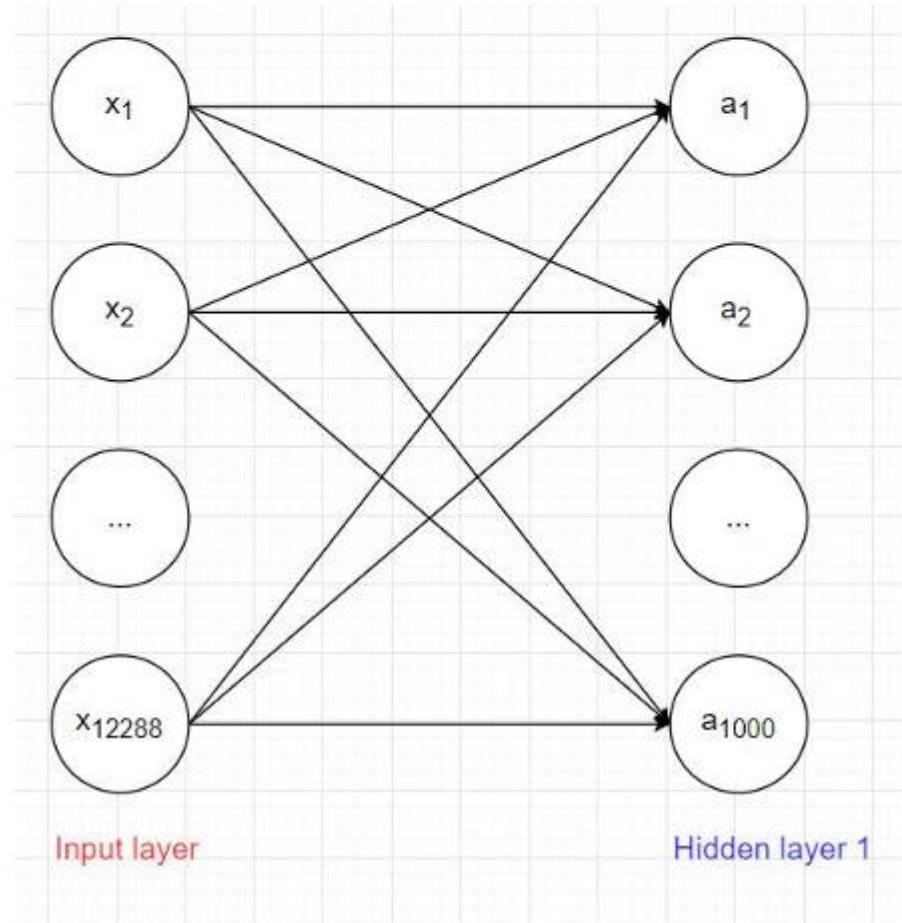


Hình 8.2: Mô hình neural network.

Mỗi hidden layer được gọi là **fully connected layer**, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là **fully connected neural network (FCN)**.

Vấn đề của fully connected neural network với xử lý ảnh

Như bài trước về xử lý ảnh, thì ảnh màu 64×64 được biểu diễn dưới dạng 1 tensor $64 \times 64 \times 3$. Nên để biểu thị hết nội dung của bức ảnh thì cần truyền vào input layer tất cả các pixel ($64 \times 64 \times 3 = 12288$). Nghĩa là input layer giờ có 12288 nodes.



Hình 8.3: Input layer và hidden layer 1

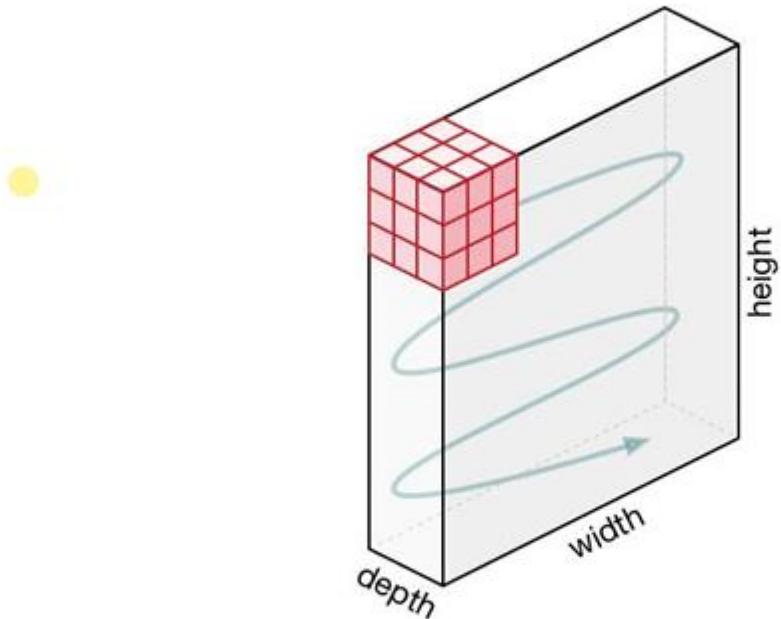
Giả sử số lượng node trong hidden layer 1 là 1000. Số lượng weight W giữa input layer và hidden layer 1 là $12288 \times 1000 = 12288000$, số lượng bias là 1000 \Rightarrow tổng số parameter là: 12289000. Đây mới chỉ là số parameter giữa input layer và hidden layer 1, trong model còn nhiều layer nữa, và nếu kích thước ảnh tăng, ví dụ 512×512 thì số lượng parameter tăng cực kì nhanh \Rightarrow Cần giải pháp tốt hơn !!!

Nhận xét:

- Trong ảnh các pixel ở cạnh nhau thường có liên kết với nhau hơn là những pixel ở xa. Ví dụ như phép tính convolution trên ảnh ở bài trước. Để tìm các đường trong ảnh, ta áp dụng sobel kernel trên mỗi vùng kích thước 3×3 . Hay làm nét ảnh ta áp dụng sharpen kernel cũng trên vùng có kích thước 3×3 .
- Với phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh chia sẻ hệ số với nhau.

\Rightarrow Áp dụng phép tính convolution vào layer trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.

Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước $k*k*3$.



Hình 8.4: Phép tính convolution trên ảnh màu với $k=3$.

Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.

- ⇒ Giải thích : Nghĩa là ta sẽ có ma trận $m*n*3$ là ma trận của ảnh và kernel sẽ là $k*k*3$ (vì khác với ảnh grey chỉ có 1 ma trận) thì ảnh màu có đến 3 ma trận . Ma trận kernel sẽ di chuyển như trong hình , theo chiều dài và chiều cao của khối . Ta có $k*k*3 \Rightarrow$ số 3 này sẽ luôn khớp với độ sâu của tensor ảnh là 3 (3 ma trận red , green , blue) chính vì thế kernel sẽ luôn trượt theo chiều dài và chiều cao của ma trận tức là $k*k$ sẽ luôn trượt theo $m*n$. Chính vì thế k chỉ giới hạn bởi m và n . Nên k có thể là số lớn , miễn sao $k*k$ không vượt ra ngoài kích thước của $m*n$
- ⇒ Xử lí trên ảnh màu

RED

| | | | | | |
|---|-----|-----|-----|-----|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 156 | 155 | 156 | 158 | 0 |
| 0 | 153 | 154 | 157 | 159 | 0 |
| 0 | 149 | 151 | 155 | 159 | 0 |
| 0 | 146 | 146 | 149 | 153 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

GREEN

| | | | | | |
|---|-----|-----|-----|-----|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 167 | 166 | 167 | 158 | 0 |
| 0 | 164 | 165 | 168 | 159 | 0 |
| 0 | 160 | 162 | 166 | 159 | 0 |
| 0 | 146 | 146 | 149 | 153 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

BLUE

| | | | | | |
|---|-----|-----|-----|-----|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 163 | 162 | 163 | 158 | 0 |
| 0 | 160 | 161 | 164 | 159 | 0 |
| 0 | 156 | 158 | 162 | 159 | 0 |
| 0 | 146 | 146 | 149 | 153 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

\otimes

+

=

+

X

W

Y

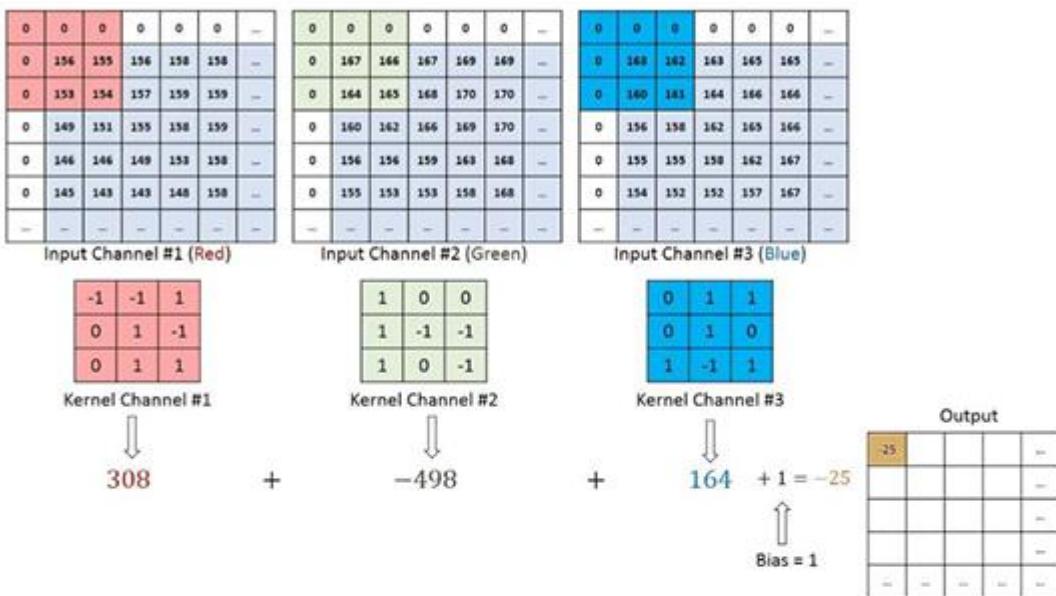
\Rightarrow Giải thích : Ví dụ như hình là $k = 3$, $k*k*3 = 3*3*3$. Còn tensor X là $m*n*3 = 6*6*3$
 \Rightarrow 3 Channel .

Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là x_{ijk} .

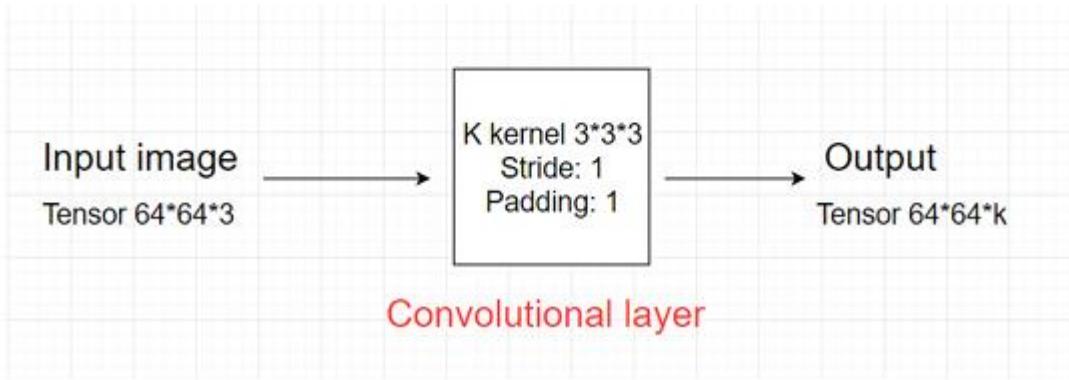
$$y_{11} = b + (x_{111} * w_{111} + x_{121} * w_{121} + x_{131} * w_{131} + x_{211} * w_{211} + x_{221} * w_{221} + x_{231} * w_{231} + x_{311} * w_{311} + x_{321} * w_{321} + x_{331} * w_{331}) + (x_{112} * w_{112} + x_{122} * w_{122} + x_{132} * w_{132} + x_{212} * w_{212} + x_{222} * w_{222} + x_{232} * w_{232} + x_{312} * w_{312} + x_{322} * w_{322} + x_{332} * w_{332}) + (x_{113} * w_{113} + x_{123} * w_{123} + x_{133} * w_{133} + x_{213} * w_{213} + x_{223} * w_{223} + x_{233} * w_{233} + x_{313} * w_{313} + x_{323} * w_{323} + x_{333} * w_{333}) = -25$$

Nhận xét:

- Output Y của phép tính convolution trên ảnh màu là 1 matrix.
- Có 1 hệ số bias được cộng vào sau bước tính tổng các phần tử của phép tính element-wise



Hình 8.6: Thực hiện phép tính convolution trên ảnh màu.



Hình 8.7: Convolutional layer đầu tiên

Output của convolutional layer đầu tiên sẽ thành input của convolutional layer tiếp theo.

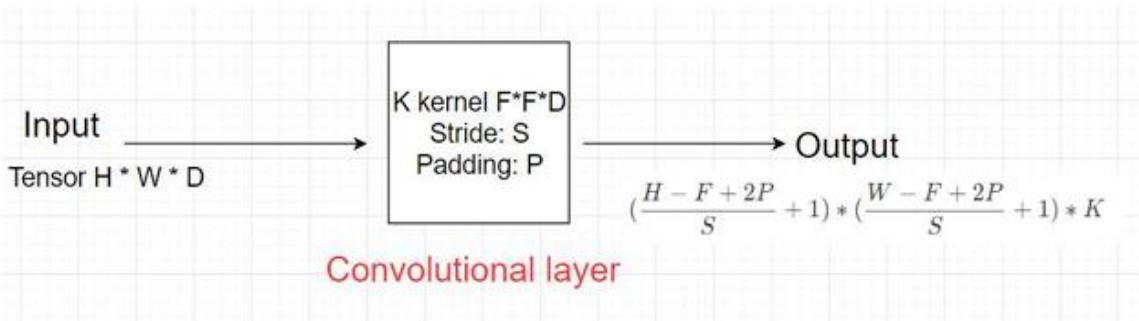
Convolutional layer tổng quát

Giả sử input của 1 convolutional layer tổng quát là tensor kích thước $H * W * D$.

Kernel có kích thước $F * F * D$ (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S , padding: P .

Convolutional layer áp dụng K kernel.

=> Output của layer là tensor 3 chiều có kích thước: $(\frac{H - F + 2P}{S} + 1) * (\frac{W - F + 2P}{S} + 1) * K$



Lưu ý:

- Output của convolutional layer sẽ qua hàm non-linear activation function trước khi trở thành input của convolutional layer tiếp theo.
- Tổng số parameter của layer: Mỗi kernel có kích thước $F * F * D$ và có 1 hệ số bias, nên tổng parameter của 1 kernel là $F * F * D + 1$. Mà convolutional layer áp dụng K kernel => Tổng số parameter trong layer này là $K * (F * F * D + 1)$.

- ⇒ Xem lại phần Overview CNN ở phần đầu tiên .
- ⇒ Kí hiệu khác thôii nhưng cách tính vẫn thế .

Ở phía trên

- F : Số lượng feature (số lượng kernel)
- k : kích thước kernel ($k * k$)

Công thức hiện tại

- $H = \text{height}$
- $W = \text{Weight}$
- $D = \text{Depth}$ (độ sâu)
 - Ở đây ta có $H * W * D$
 - Kernel : $F * F * D$
- ⇒ Có một lưu ý là depth của kernel luôn bằng depth của input
- $F = \text{kích thước kernel}$

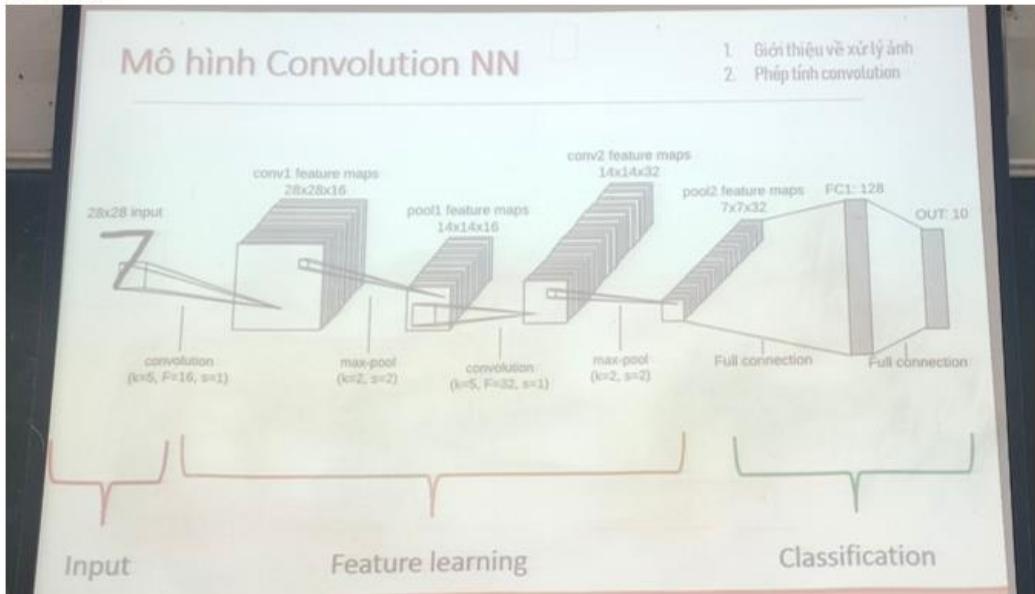
- K : Số lượng kernel F^*F^*D
- Ví dụ từ input qua kernel ở tích chập thứ nhất theo hình vẽ ở trên ta sẽ có

$$\frac{H - F + 2P}{S} + 1 = \frac{28 - 5 + 2 * 2}{1} + 1 = 28$$

$$\frac{W - F + 2P}{S} + 1 = \frac{28 - 5 + 2 * 2}{1} + 1 = 28$$

$$K = 16$$
- $\Rightarrow Output = H * W * D = 28 * 28 * 16$
- \Rightarrow Tương tự với tích chập lần 2

Overview CNN



Giải thích kiến trúc :

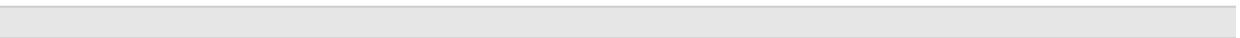
- Input
 - Hình ảnh kích thước $28*28*1$ (ảnh grey nên nó là 1 ma trận $28*28$ nên không phải là ảnh màu tensor 3 channel) (ma trận $28*28$ hoặc có thể gọi là tensor $28*28*1$)
- Tích chập với ($k=5, F=16, s=1$)
 - **Lưu ý :** Ở đây ta có một lưu ý là input đầu vào 28 và đi qua kernel vẫn giữ nguyên
 $28 \Rightarrow$ có sử dụng padding . Công thức là : $p = \frac{k-1}{2}$, vậy padding ở đây là $p = 2$
 - $K=5 \Rightarrow$ chính là k của kernel : $k*k*3 = 5*5*3$ (ở đây ta có X là $m*n = 28*28$ nên $k*k = 5*5$ nhỏ hơn $28*28$ là hoàn toàn được)
 - $F=16$ (số lượng kernel là 16) F ở đây chính là W (kernel) (một số tài liệu gọi kernel W chính là feature F)(Tuy nhiên kernel F này chưa tối ưu , vẫn đang còn random)(Ở đây ta cũng có thể thêm một bước tối ưu kernel)
 - Cho X này qua 16 kernel ($5*5*3$) sẽ thu được 16 X mới ta thấy trong hình là $28*28*16$
 - S trong hình là stride = 1 \Rightarrow bước nhảy khi cho qua kernel là 1

- Những thanh này là tích chập cho nó nhỏ xuống => Chính là nhân ma trận W cho nó gọn

lại . Từ một ô vuông (ma trận) trở thành 1 điểm nhỏ (1 số)



- Dẫn đến ma trận to thành nhỏ
- Vậy theo như công thức ở dưới thay vào ta có
Input : $28*28*1$



Qua kernel ($k=5, F=16, s=1, p = \frac{k-1}{2} = 2$) thì thu được tensor có kích thước
 $\left(\frac{28-5+2*2}{1} + 1\right) * \left(\frac{28-5+2*2}{1} + 1\right) * F(kernel) = 28 * 28 * 16$

Max-pool ($k=2, s=2$)

- Kích thước pooling là $k=2$, $k*k = 2*2$ và số bước nhảy stride là 2 => giải kích thước ban đầu xuống $\frac{1}{2}$ => 28 xuống còn 14
- Sau đó cho qua max-pool để thu nhỏ kích thước ma vẫn giữ được đặt trưng của ảnh

Tích chập lần 2 ($k=5, F=32, S=1$)

- Tương tự , lần này ta thấy cho qua 32 Kernel => số lượng tạo ra gấp đôi so với $F=16$ (như hình)
- Input là : $14*14*16$
- Qua kernel ($k=5, F=32, s=1, p = \frac{k-1}{2} = 2$) thì thu được tensor có kích thước
- $\left(\frac{14-5+2*2}{1} + 1\right) * \left(\frac{14-5+2*2}{1} + 1\right) * F(kernel) = 14 * 14 * 32$

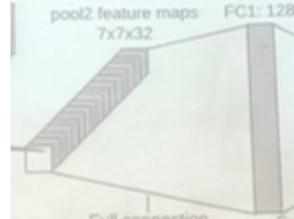
Cho qua max-pool ($k=2, s=2$)

- Kích thước pooling là $k=2$, $k*k = 2*2$ và số bước nhảy stride là 2 => giải kích thước ban đầu xuống $\frac{1}{4}$ => 14 xuống còn 7

- Cho qua max-pool ($k=2, s=2$)
 - Kích thước pooling là $k=2$, $k*k = 2*2$ và số bước nhảy stride là 2 => giải kích thước ban đầu xuống $\frac{1}{2} \Rightarrow 14$ xuống còn 7

- **Flattening (trong hình thiếp)**

- Thông qua Flattening kích thước $H*W*D$, sẽ được chuyển về 1 vector kích thước $(H*W*D, 1)$. Ta có $7*7*32$ là 32 ma trận $7*7$ trở thành vector có kích thước $(1568, 1)$



- Các hình vuông sẽ bẻ thành một đường thẳng và là đầu vào của mạng neural network

- Fully connected 1
 - Input là vector $(1568, 1)$ và output là vector $(128, 1)$
- Fully connected 2
 - Input là vector $(128, 1)$ và output là vector $(10, 1)$

- ❖ Công thức tổng quát :

Công thức tổng quát cho phép tính convolution của ma trận X kích thước $m*n$ với kernel kích thước $k*k$, stride = s , padding = p ra ma trận Y kích thước $(\frac{m-k+2p}{s} + 1) * (\frac{n-k+2p}{s} + 1)$.

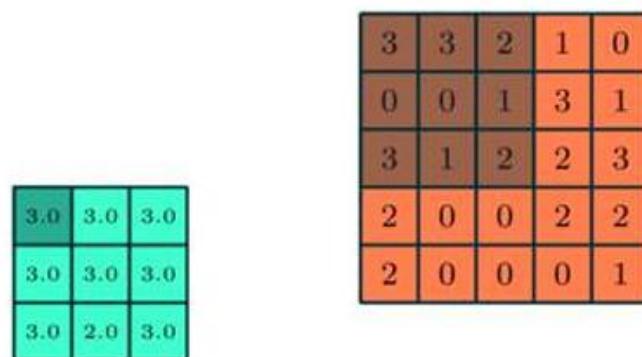
p : padding , k : kích thước của kernel : $k*k$, m,n là ma trận đầu vào, s là stride .

- ❖ Lưu ý : Đối với ảnh màu thì như trên ví dụ sẽ là : $28*28*3$ (3 channel r,g,b trên 1 ảnh màu)
Từ đó dẫn đến các xử lý sau đều $*3$ lên .

Pooling layer

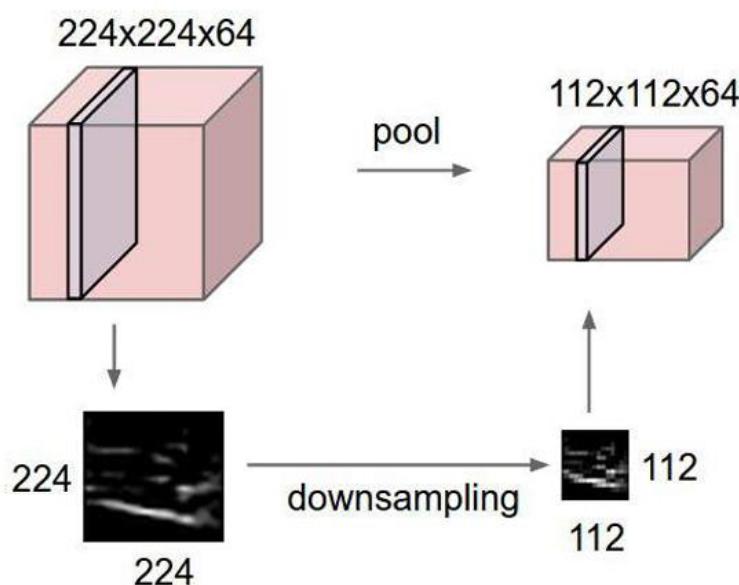
- Pooling layer (Lớp gộp) thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu giúp giảm các phép tính toán trong model.
- Cách tính pooling (Giải quyết vấn đề thu hẹp tensor nhưng vẫn giữ được đặc trưng của ảnh)
 - Max pooling (lấy max)
 - Min pooling (lấy min)
 - Average pooling (lấy trung bình)

Gọi pooling size kích thước $K \times K$. Input của pooling layer có kích thước $H \times W \times D$, ta tách ra làm D ma trận kích thước $H \times W$. Với mỗi ma trận, trên vùng kích thước $K \times K$ trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.



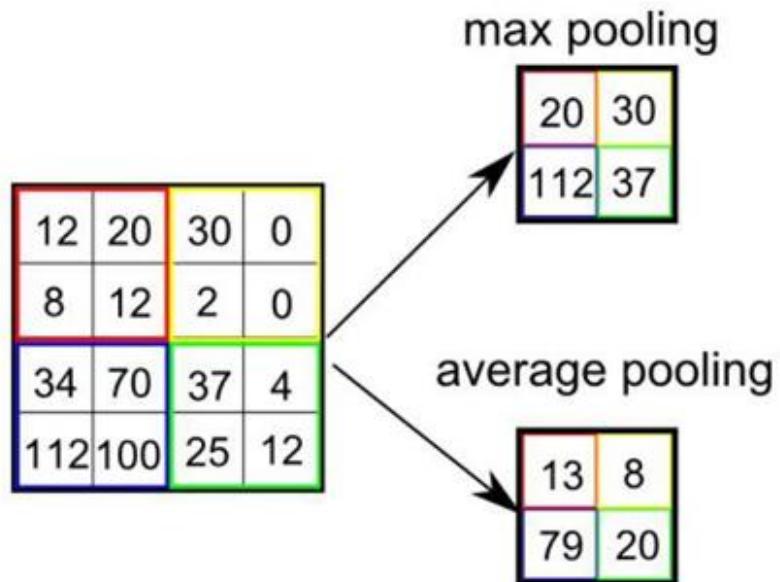
Hình 8.8: max pooling layer với size=(3,3), stride=1, padding=0

Nhưng hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên.



Hình 8.9: Sau pooling layer (2*2) [4]

Có 2 loại pooling layer phổ biến là: max pooling và average pooling.



Hình 8.10: Ví dụ về pooling layer

Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.

=> Ví dụ ở đây $4 \times 4 \Rightarrow$ nếu dùng max $2 \times 2 \Rightarrow$ chia đủ

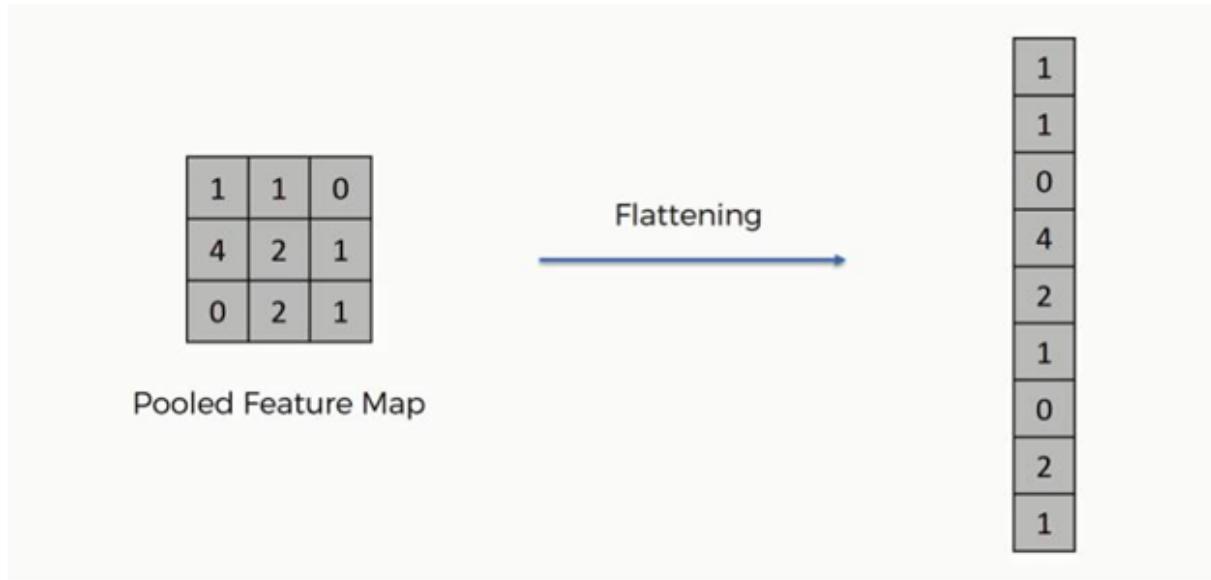
=> Nhưng nếu dùng max $3 \times 3 \Rightarrow$ thêm các 0 vào bên phải và các hàng 0 vào bên dưới \Rightarrow sẽ chia đủ .



=> Thường sẽ chọn max pooling .

Fully connected layer

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,...) thì tensor của output của layer cuối cùng, kích thước $H*W*D$, sẽ được chuyển về 1 vector kích thước $(H*W*D, 1)$



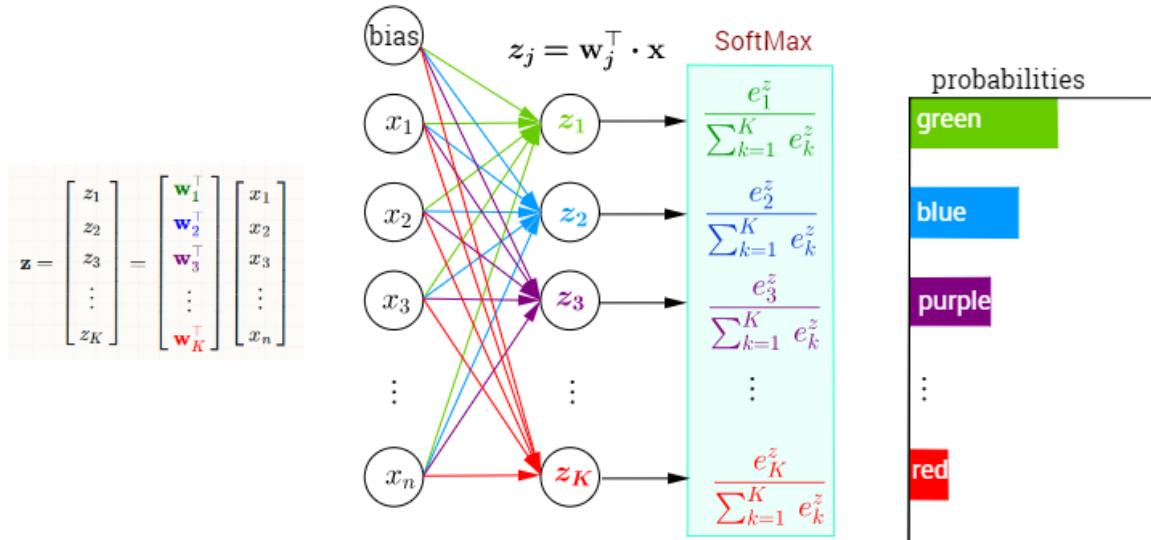
Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

=> Chuyển ma trận thành vector cột (x_0 đến x_n)

Sau này nó sẽ trở thành đầu vào của RNN

Hàm softmax

Multi-Class Classification with NN and SoftMax Function



Softmax activation function cho bài toán Phân loại nhiều lớp (Multi-Class Classification) (Nhiều nhãn)

- Trong phân loại đa lớp (Multi-Class Classification), hàm kích hoạt thích hợp cho lớp đầu ra là softmax. Hàm softmax thường được sử dụng để chuyển đổi các điểm số đầu ra thành các xác suất. Nó là một dạng nâng cao của hàm sigmoid, được thiết kế để xử lý các tác vụ phân loại có nhiều lớp.
- Hàm softmax được định nghĩa như sau:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Trong đó :

- z_i là điểm số (score) của lớp i (đầu ra của mạng tại node i)
- K là số lớp đầu ra

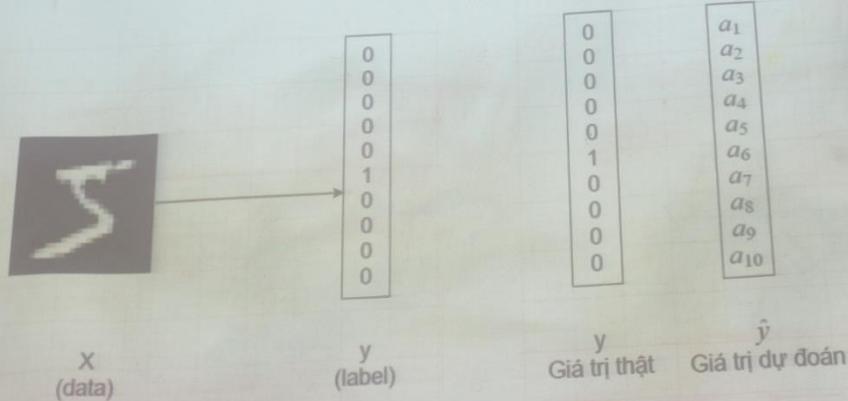
Hàm softmax thực hiện hai công việc chính:

1. Chuyển đổi các điểm số đầu ra thành các giá trị xác suất bằng cách biến đổi chúng sao cho tổng của tất cả các xác suất bằng 1.
 2. Tăng cường sự khác biệt giữa các xác suất, làm cho xác suất cao nhất trở nên rất gần 1 trong khi các xác suất khác gần 0.
- ❖ Việc sử dụng softmax trong lớp đầu ra cho phép mô hình phản ánh chính xác xác suất của dữ liệu thuộc vào từng lớp, giúp ta dễ dàng tạo ra dự đoán đúng cho nhiều lớp khác nhau trong bài toán phân loại đa lớp.

Mô hình CNN

1. Convolutional layer
2. Convolutional layer tổng quát
3. Pooling layer
4. Fully Connected layer

- One-hot encoding: chuyển đổi label của ảnh từ giá trị số sang vector cùng kích thước với output của model



- ⇒ Ví vụ về nhận dạng chữ số viết tay .
- ⇒ Mỗi chữ số sẽ cho ra một one-hot , ví dụ số 5 là one-hot [0,0,...,1,0] ; số 9 là [0,1,0,...,0,0]
- ⇒ **Xem phần : Vector đặc trưng và One-hot đã nói ở phía trên .**

Mô hình CNN

1. Convolutional layer
2. Convolutional layer tổng quát
3. Pooling layer
4. Fully Connected layer

Nhắc lại: hàm softmax để xác định đầu ra

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

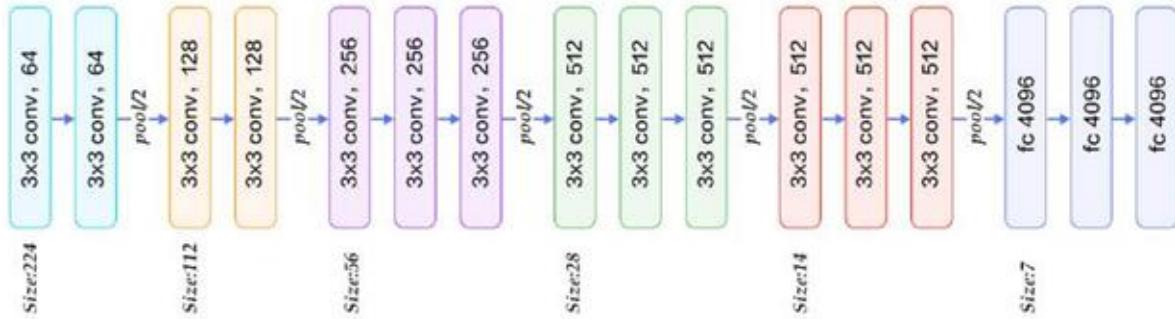
Z: vector đầu vào của hàm

Z_i : các phần tử vector đầu vào

K: số lớp để phân loại vector đầu vào

Mạng VGG16

VGG16 là mạng convolutional neural network được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Giờ áp dụng kiến thức ở trên để phân tích mạng VGG 16.



Hình 8.11: Kiến trúc VGG16 conv: convolutional layer, pool: pooling layer, fc: fully connected layer

Phân tích:

- Convolutional layer: kích thước 3×3 , padding=1, stride=1. Tại sao không ghi stride, padding mà vẫn biết? Vì mặc định sẽ là stride=1 và padding để cho output cùng width và height với input.
- Pool/2 : max pooling layer với size 2×2
- 3×3 conv, 64: thì 64 là số kernel áp dụng trong layer này, hay depth của output của layer này.
- Càng các convolutional layer sau thì kích thước width, height càng giảm nhưng depth càng tăng.
- Sau khá nhiều convolutional layer và pooling layer thì dữ liệu được flatten và cho vào fully connected layer.
- Bài sau tôi sẽ giới thiệu về keras và hướng dẫn dùng keras để áp dụng convolutional neural vào các ứng dụng như nhận diện số viết, dự đoán góc di chuyển trong ô tô tự lái.

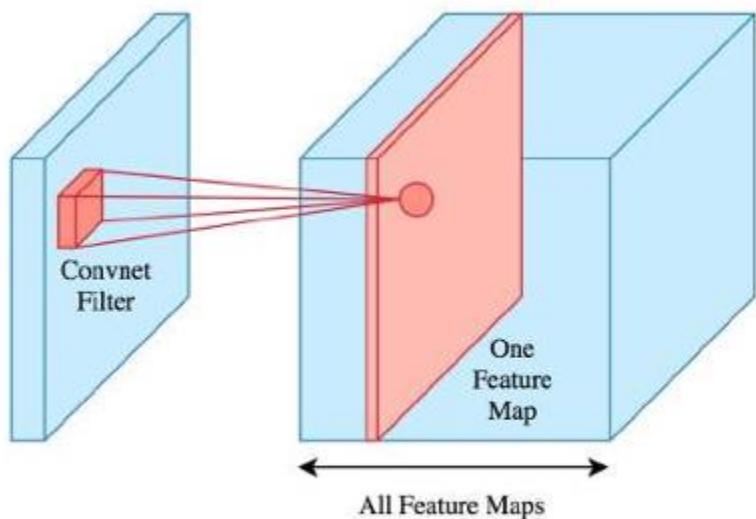
Visualizing Convolutional Neural Network

Mục đích chính của việc visualizing Convolutional Neural Network (CNN) là hiểu rõ cách mà mạng CNN hoạt động và cách nó học cách phân loại hình ảnh. Cụ thể, việc visualizing CNN có các mục đích sau:

1. **Hiểu cách hoạt động của mạng**: Bằng cách visualizing CNN, chúng ta có thể xem xét các lớp và phương pháp kích hoạt để hiểu cách mạng thực hiện các phép tính và trích xuất đặc trưng từ hình ảnh đầu vào.
2. **Phát hiện các đặc trưng quan trọng**: Visualizing CNN có thể giúp chúng ta nhận biết các đặc trưng quan trọng mà mạng học được từ dữ liệu. Điều này có thể giúp chúng ta hiểu tại sao mạng đưa ra các quyết định phân loại như vậy và giúp cải thiện hiệu suất của mô hình.
3. **Debug và điều chỉnh mô hình**: Bằng cách xem xét các hình ảnh đầu vào và đầu ra của các lớp khác nhau trong mạng, chúng ta có thể phát hiện ra các vấn đề như overfitting, underfitting hoặc các vấn đề khác trong quá trình huấn luyện. Điều này giúp chúng ta debug và điều chỉnh mô hình để tối ưu hóa hiệu suất.

4. ****Trực quan hóa quá trình học**:** Visualizing CNN có thể cung cấp một cái nhìn trực quan về quá trình học của mạng, bao gồm việc theo dõi cách mà các trọng số được điều chỉnh và cách độ chính xác của mạng thay đổi qua từng vòng lặp huấn luyện.
 5. ****Giảng dạy và giải thích**:** Visualizing CNN cũng là một cách hiệu quả để giảng dạy và giải thích về cách hoạt động của mạng neural network cho người mới học hoặc những người không chuyên trong lĩnh vực này.
- ⇒ Nghĩa là để ta có thể quan sát được kiến trúc của model đó , để có thể hiểu và tùy chỉnh trên nó được .

Visualizing Feature Maps



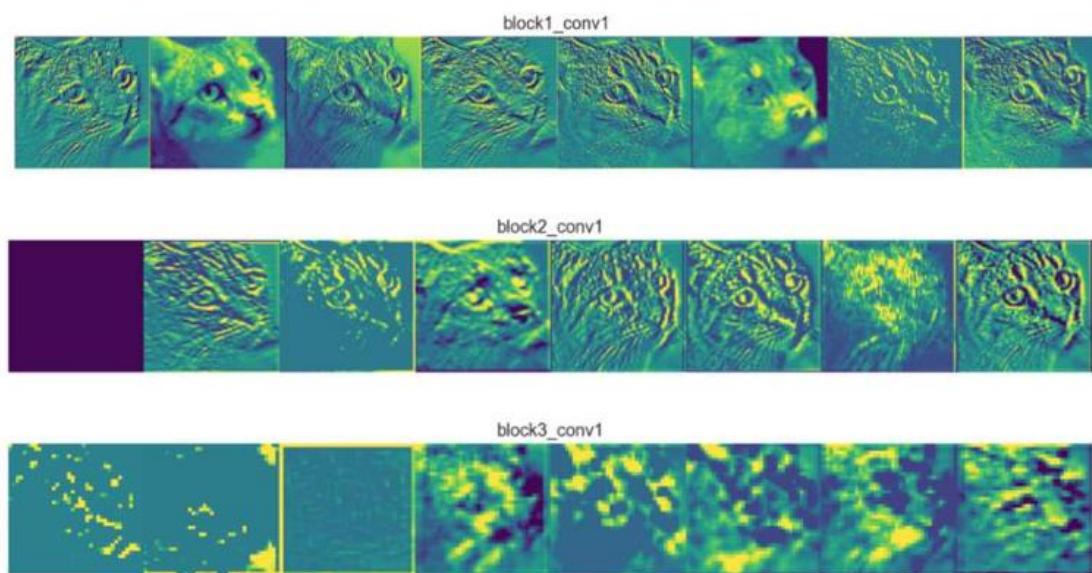
Hình 8.12: Tensor gồm các Feature Maps



Hình 8.13: Visualize block1-conv1 của mạng VGG16

Phần có màu sáng hơn được gọi là vùng kích hoạt, nghĩa là vùng đó đã được filter lọc ra thứ mà nó cần tìm. Như hình trên, có vẻ filter này đang muốn trích xuất ra phần mắt và mũi của chú mèo.

Mỗi tensor của conv layer gồm nhiều feature maps khác nhau, hình bên dưới ta thực hiện visualize 8 feature maps của 5 block khác nhau trong mạng VGG16.



Khá thú vị phải không. Để ý hơn ta có thể đưa ra vài nhận xét như sau:

- Các lớp đầu tiên (block1-conv1) chủ yếu giữ lại hầu hết thông tin của ảnh. Trong mạng CNN, các lớp đầu tiên thường là trích xuất viền cạnh của ảnh.

- Càng đi sâu, các lớp khó có thể nhận ra là hình gì. Như hình ở block2-conv1, ta vẫn có thể nhận ra đó là con mèo. Tuy nhiên đến block thứ 3 thì rất khó đoán được đó là hình gì và block thứ 4 thì không thể đoán nổi. Lý do là càng về sâu, các feature map biểu diễn cho những hình trừu tượng hơn. Những thông tin càng cụ thể hơn thì ta càng khó hình dung nó là gì. Chẳng hạn như các lớp về sau chỉ giữ lại thông tin của tai, mắt hay mũi của mèo mà thôi.
- Các thông tin càng về sau càng thưa thớt hơn. Điều này hợp lý vì những lớp đầu giữ lại các viền, cạnh của ảnh, thứ mà bức ảnh nào cũng có. Tuy nhiên càng đi sâu, mô hình tìm những thứ phức tạp hơn ví dụ như cái đuôi của con mèo - thứ mà không phải hình nào cũng có. Vậy nên đó là lý do tại sao ta thấy một số hình ở trên bị trống.

⇒ [HAY]

Để ý hơn ta có thể đưa ra vài nhận xét như sau:

- Các lớp đầu tiên (block1-conv1) **chủ yếu giữ lại hầu hết thông tin của ảnh**. Trong mạng CNN, các lớp đầu tiên thường là trích xuất viền cạnh của ảnh. 134 Chương 8. Convolutional neural network
- **Càng đi sâu, các lớp khó có thể nhận ra là hình gì.** Như hình ở block2-conv1, ta vẫn có thể nhận ra đó là con mèo. Tuy nhiên đến block thứ 3 thì rất khó đoán được đó là hình gì và block thứ 4 thì không thể đoán nổi. Lý do là càng về sâu, các feature map biểu diễn cho những hình trừu tượng hơn. Những thông tin càng cụ thể hơn thì ta càng khó hình dung nó là gì. Chẳng hạn như các lớp về sau chỉ giữ lại thông tin của tai, mắt hay mũi của mèo mà thôi.
- Các thông tin càng về sau càng thưa thớt hơn. **Điều này hợp lý vì những lớp đầu giữ lại các viền, cạnh của ảnh, thứ mà bức ảnh nào cũng có. Tuy nhiên càng đi sâu, mô hình tìm những thứ phức tạp hơn ví dụ như cái đuôi của con mèo- thứ mà không phải hình nào cũng có. Vậy nên đó là lý do tại sao ta thấy một số hình ở trên bị trống.**

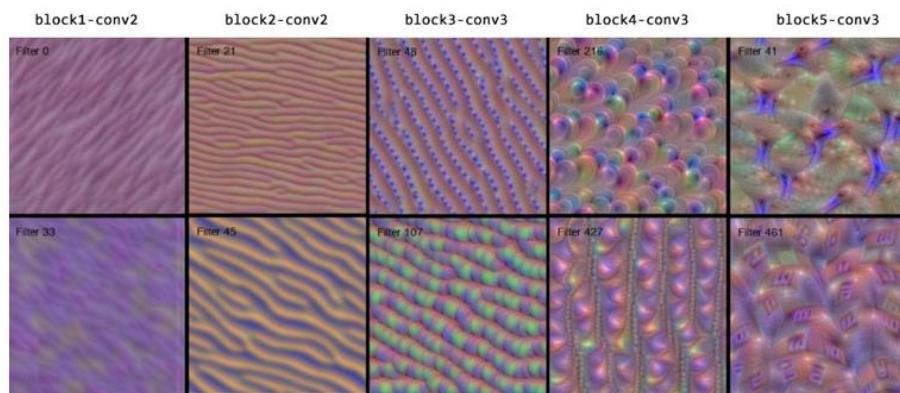
Visualizing Convolutional Filters

Thay vào đó, ta sẽ thử biểu diễn các filters thông qua một hình đầu vào mà khi qua hàm tích chập với filters tương ứng đạt được giá trị kích hoạt trung bình cực đại. Có một số kỹ thuật nâng cao, được đề xuất bởi [Erhan et al. 2009](#), nhưng tựu chung gồm các bước cơ bản như sau:

- Một hình ảnh với giá trị các pixel ngẫu nhiên và một pre-trained model CNN.
- Ta sẽ tìm các giá trị của pixel sao cho hàm loss là để cực đại trung bình giá trị kích hoạt của filter bằng cách thực hiện gradient ascent.
- Lặp lại đến khi đạt giá trị cực đại

Lưu ý rằng, trong phần trước ta visualize đầu ra của ảnh khi đã đi qua filter, phần này là visualize xem filter đó dùng để làm gì bằng cách tìm một hình khiến cho filter đó kích hoạt tối đa các giá trị pixel.

Lưu ý rằng, trong phần trước ta visualize đầu ra của ảnh khi đã đi qua filter, phần này là visualize xem filter đó dùng để làm gì bằng cách tìm một hình khiến cho filter đó kích hoạt tối đa các giá trị pixel.



Hình 8.16: Visualize 2 filters với mỗi layer tương ứng

Càng về các layer cuối, hình ảnh càng trông rõ ràng hơn:

- Các filter đầu tiên chủ yếu dùng để phát hiện những cạnh, viền, màu sắc và những hình dạng đơn giản.
- Các filter về sau mã hoá những hình ảnh phức tạp hơn. Như ta thấy filter 41 ở layer block5-conv3 có vẻ dùng để nhận ra các con chim. Khi mà rất nhiều hình con chim với dáng khác nhau ở đó, nó khiến cho giá trị trung bình hàm kích hoạt gần đến cực đại.

Có thể thấy mô hình CNN cũng giống con người. Khi còn là trẻ sơ sinh, ta tập trung vào những hình đơn giản trước rồi qua quá trình luyện tập những hình ảnh phức tạp dễ dàng được nhận ra bởi não bộ con người.

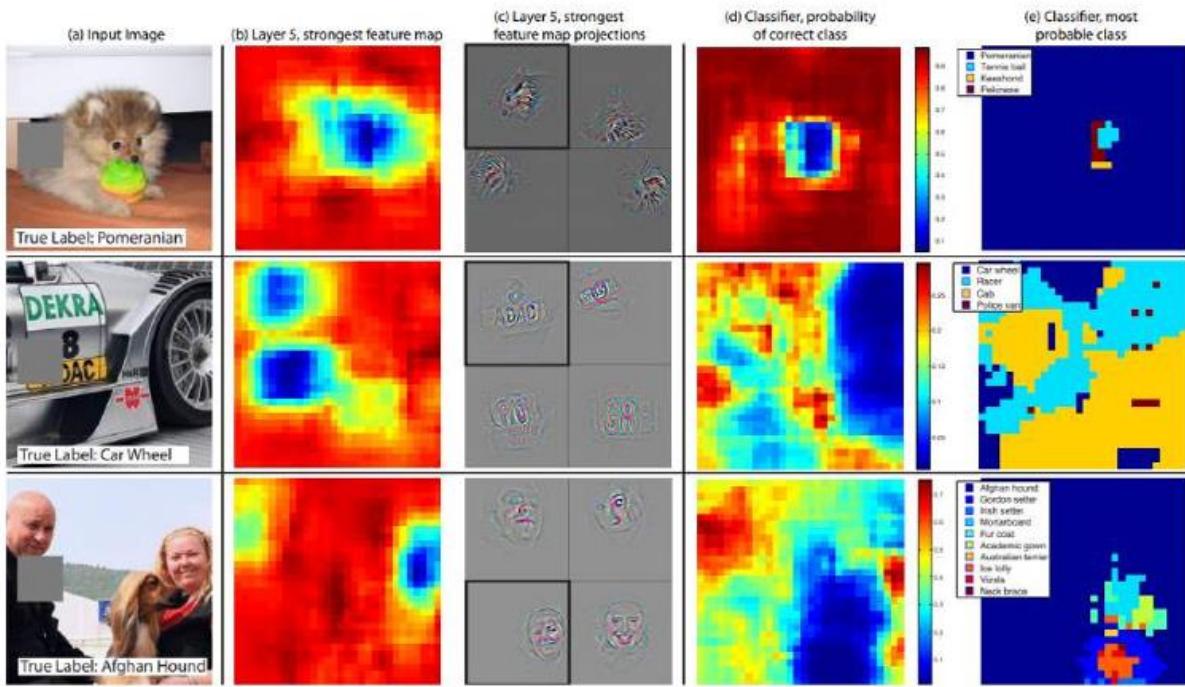
Visualizing Class Outputs

[Home - Keras-vis Documentation \(raghakot.github.io\)](#)

Visualizing Attention Map

8.4.4 Visualizing Attention Map

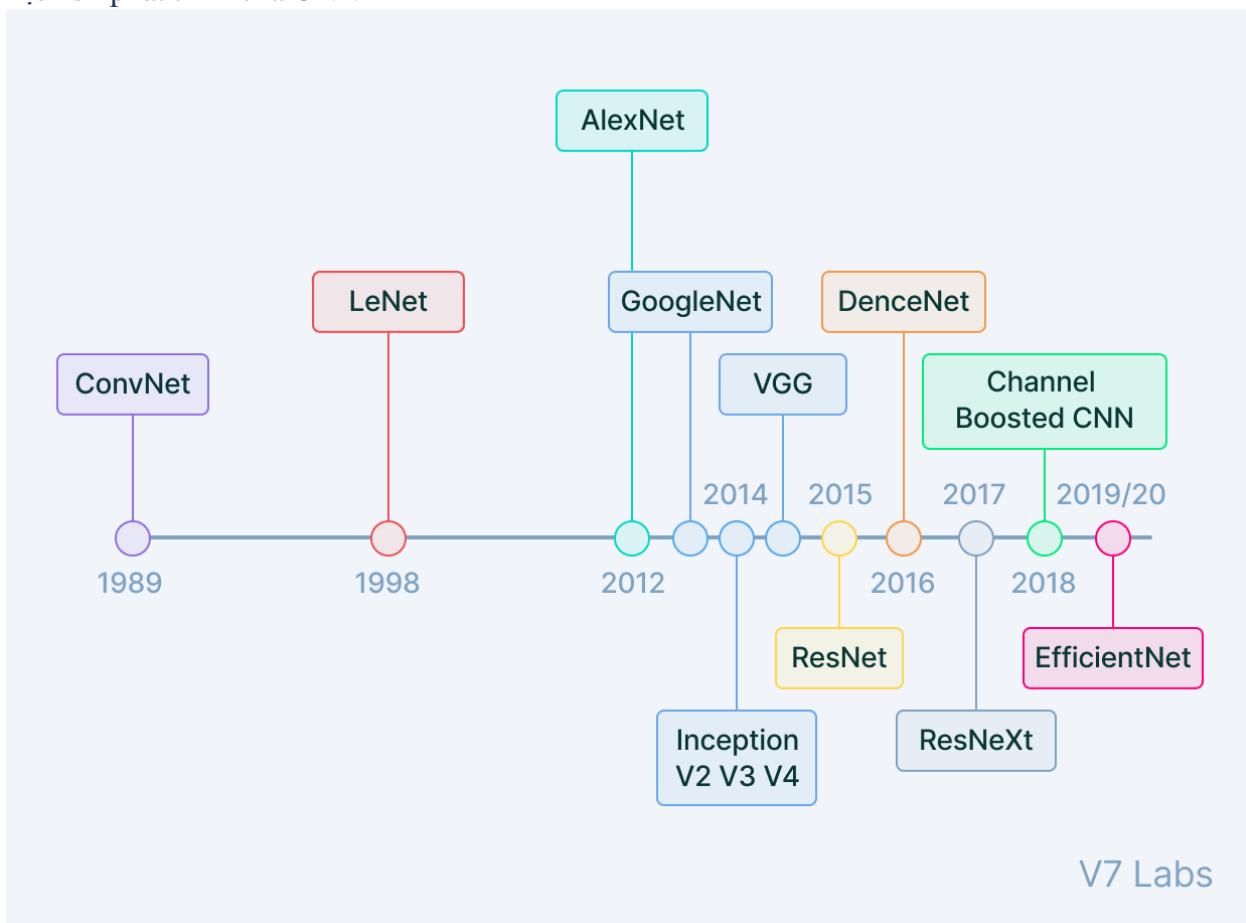
Một cách rất trực quan để visualize một mạng CNN dùng phân loại ảnh là Attention Map. Giả sử ta có mạng CNN để phân loại ảnh, làm thế nào và tại sao mạng CNN lại đưa ảnh đó vào lớp chó? Liệu có phải mạng CNN có chú ý hơn vào những pixel liên quan đến chó? Để biết điều này ta xoá đi một phần của ảnh, sau đó đưa vào mạng CNN phân loại ảnh để xem xác suất rơi vào class nào và bao nhiêu. Khi lặp lại đến hết toàn bộ ảnh, ta có thể vẽ được heatmap với vùng càng nóng thì xác suất mô hình dự đoán đúng càng cao, hay phần bị che đi đó càng ít quan trọng. Kỹ thuật này được đề xuất bởi Matthew Zeiler



Hình 8.18: Ba ví dụ áp dụng kỹ thuật nêu trên, phần bị che đi dùng một hình vuông màu xám (a) và xem feature map ở layer 5 chú ý đến gì (b & c), cột thứ 4 là attention map, ví dụ khi mặt của con chó Pom bị che thì tỉ lệ rơi vào class chó Pom gần như bằng không và mô hình CNN dự đoán là bóng tennis hoặc loài chó khác (e)

9. Giới thiệu keras và bài toán phân loại ảnh

Lịch sử phát triển của CNN

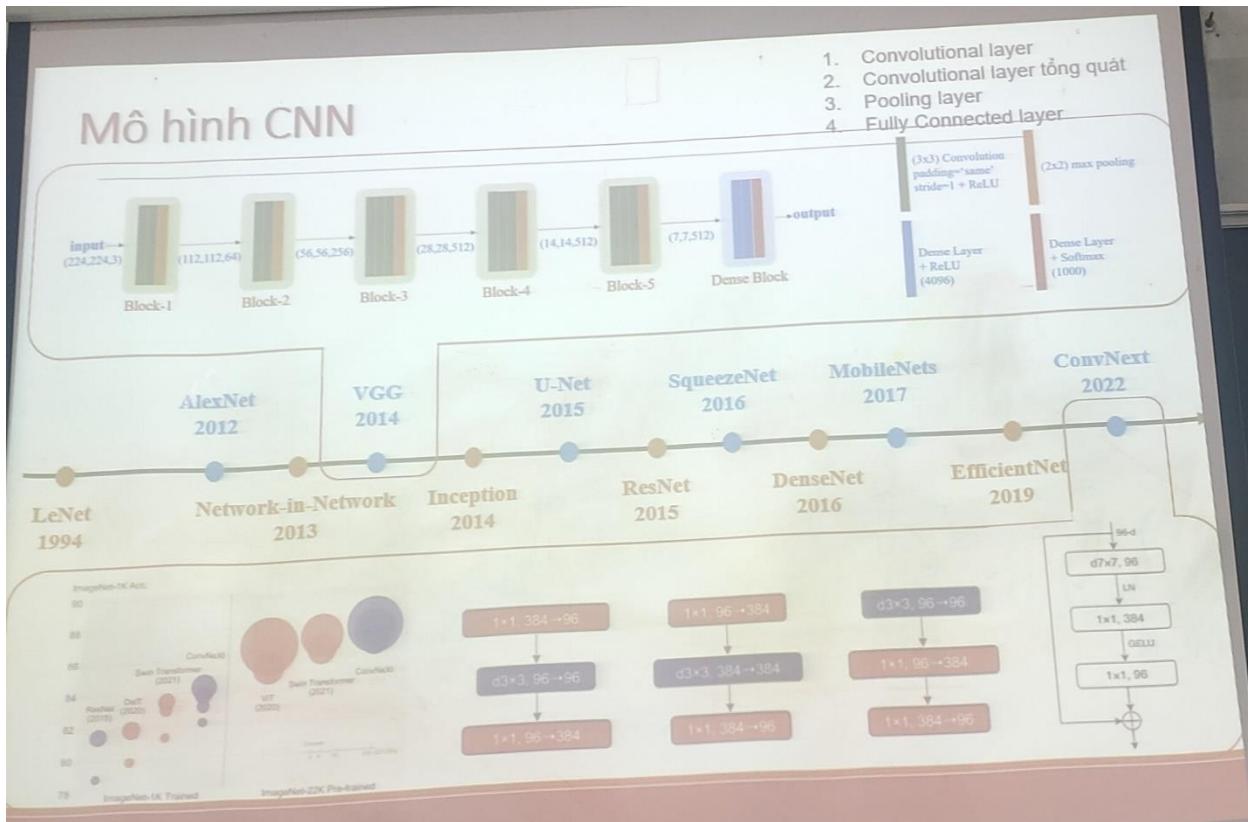


Dưới đây là một số mô hình CNN quan trọng đã được công bố cùng với lịch sử phát triển của chúng:

1. **LeNet-5:** Được giới thiệu bởi Yann LeCun vào năm 1998, LeNet-5 là một trong những mô hình CNN đầu tiên được áp dụng rộng rãi trong **việc nhận diện ký tự viết tay**.
2. **AlexNet:** Được giới thiệu bởi Alex Krizhevsky, Ilya Sutskever và Geoffrey Hinton vào năm 2012, AlexNet là một trong những mô hình đầu tiên áp dụng CNN với quy mô lớn trên tập dữ liệu **ImageNet**, giúp tăng cường sự phát triển của deep learning.
3. **VGGNet:** Mô hình này được phát triển bởi Karen Simonyan và Andrew Zisserman vào năm 2014. VGGNet có kiến trúc đơn giản nhưng sâu, với **nhiều lớp convolution và pooling**, đã đạt được **hiệu suất tốt** trên ImageNet.
4. **GoogLeNet (Inception):** Được giới thiệu bởi Christian Szegedy và nhóm nghiên cứu Google vào năm 2014, GoogLeNet sử dụng kiến trúc Inception, với **nhiều module Inception khác nhau** để giảm số lượng tham số và tăng hiệu suất.

5. **ResNet:** ResNet, hoặc Residual Network, được giới thiệu bởi Kaiming He, Xiangyu Zhang, Shaoqing Ren và Jian Sun của Microsoft Research vào năm 2015. ResNet sử dụng các khối residual để xử lý vấn đề vanishing gradient và giúp huấn luyện các mô hình sâu hơn mà không gặp vấn đề quá tải.
6. **MobileNet:** Được giới thiệu bởi Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto và Hartwig Adam của Google vào năm 2017, MobileNet được thiết kế đặc biệt để có thể chạy trên các thiết bị có tài nguyên hạn chế như điện thoại di động.
7. **EfficientNet:** Được giới thiệu bởi Mingxing Tan và Quoc V. Le của Google vào năm 2019, EfficientNet sử dụng một phương pháp tự động điều chỉnh kích thước mô hình để đạt được hiệu suất cao với tài nguyên hạn chế.

Những mô hình này đã được công bố kiến trúc của chúng và có thể được xây dựng lại bằng TensorFlow hoặc Keras.



⇒ NN thông qua tích chập (convolution) thành CNN => sau đó tất cả các mô hình này đều phát triển từ CNN

NG

You

trong việc nhận diện , phân loại hình ảnh thì model nào cho độ chính xác cao nhất (2024)



ChatGPT

Hiện tại, không có một mô hình cụ thể nào được xác định là có độ chính xác cao nhất trong việc nhận diện và phân loại hình ảnh vào năm 2024. Điều này phụ thuộc vào nhiều yếu tố như loại dữ liệu, kích thước mô hình, độ phức tạp của bài toán, và nhiều yếu tố khác.

Tuy nhiên, một số mô hình phổ biến và có hiệu suất tốt trong nhận diện và phân loại hình ảnh vào năm 2024 bao gồm:

1. EfficientNet: Mô hình này được thiết kế để đạt được hiệu suất cao với tài nguyên hạn chế, và đã chứng tỏ sức mạnh của mình trong nhiều bài toán phân loại hình ảnh.
2. ResNet và các biến thể của nó (ví dụ: ResNet-50, ResNet-101): ResNet vẫn là một trong những lựa chọn phổ biến và mạnh mẽ trong việc phân loại hình ảnh.
3. MobileNet và các biến thể như MobileNetV2 và MobileNetV3: Các mô hình này được thiết kế để chạy trên các thiết bị có tài nguyên hạn chế và có hiệu suất tốt trong việc nhận diện hình ảnh.
4. InceptionNet và các biến thể của nó: Mô hình InceptionNet, đặc biệt là phiên bản mới nhất như InceptionV3 và InceptionResNetV2, cũng có hiệu suất tốt trong việc phân loại hình ảnh.

Tùy thuộc vào yêu cầu cụ thể của bài toán và tài nguyên có sẵn, mỗi mô hình có thể phù hợp với các trường hợp sử dụng khác nhau. Để chọn mô hình phù hợp nhất, cần xem xét cẩn thận về yêu cầu của dự án cụ thể và thử nghiệm các mô hình khác nhau để đánh giá hiệu suất.

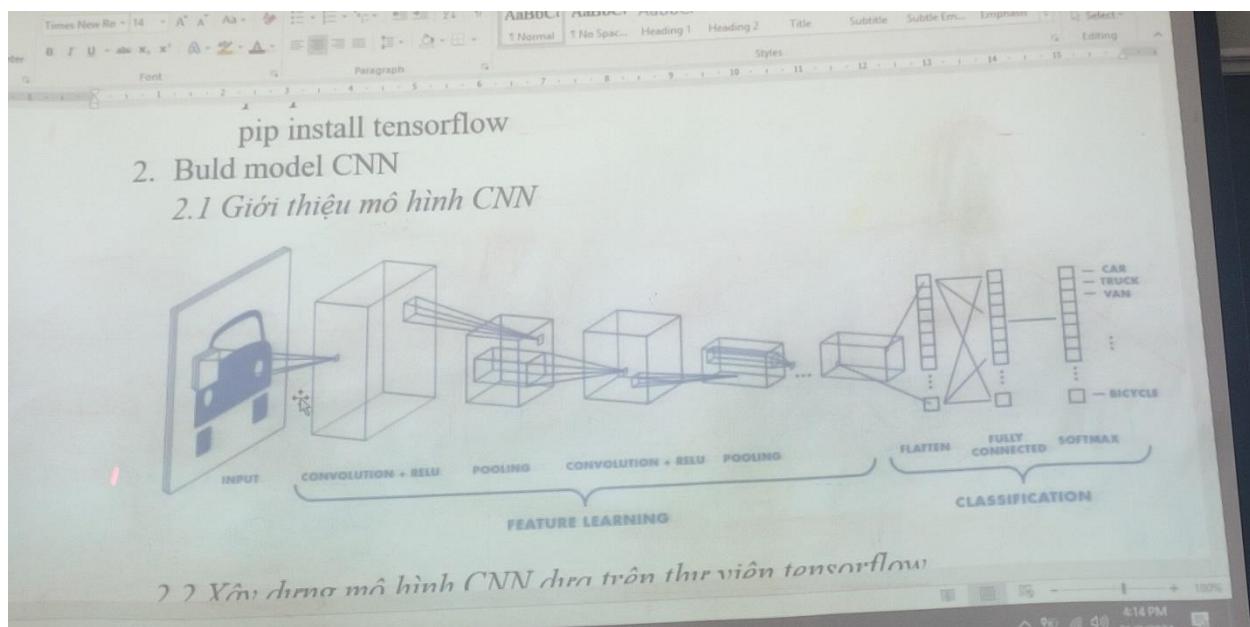


⇒ Đây là các mô hình chung , ví dụ YOLO5 sử dụng để nhận diện đối tượng nói chung , chính vì thế khi chúng ta sử dụng lại để nhận diện một đối tượng cụ thể ví dụ : Con người ,... thì phải điều chỉnh lại cho nó phù hợp để nhận diện người . Ví dụ nếu nhận diện đối tượng nói chung độ chính xác có thể là 80% là tối đa nhưng khi ta điều chỉnh lại để cho nó chỉ nhận diện người thôi thì độ chính xác có thể cao hơn .

Thực hành

- Từ bộ dữ liệu hình ảnh
 1. Cat, Dog
 2. Cifar-10
 3. Fashion-MNIST
- ⇒ Chia train , test
- ⇒ Sử dụng CNN

Sử dụng keras Tham khảo : <https://www.kaggle.com/code/vanmanhbk/checkpoint-keras/edit>



6. Xây model

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
# Building the CNN
model = Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
    input_shape=(28, 28, 1), activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
    activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(300, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax),
])
model.summary()
```

3. Xây dựng mô hình CNN để training và predict nhãn dạng chữ số viết tay

- ⇒ Softmax , relu : các hàm kích hoạt
- ⇒ 10 (10 classifier)(10 nhãn là 10 số từ 0 đến 9 chữ số viết tay)
- ⇒ Filter=32 => 32 lớp kernel với kích thước 3*3

7. Train

- Thêm hàm tối ưu (log loss,...)

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

model.fit(x_train, y_train, epochs=5)
# model.save('model.h5')

img_test = cv2.imread('anh4.png', 0) #ảnh của số 9
# cv2.imshow('anh test', img_test)
# cv2.waitKey(0)
img_test = img_test.reshape(1, 28, 28, 1)
print(model.predict(img_test))
```

4. Ứng dụng mô hình để phân loại Mèo và Chó

- ⇒ Optimizer = sgd (gradient descent) (hàm tối ưu)
- ⇒ Img_test = cv2.... (đọc ảnh đầu vào)
- ⇒ Cho qua predict (kết quả in ra sẽ là one-host) => từ one-host này ta sẽ biết nó là số gì (trong các số từ 0 -> 9) (tùy mình cài đặt mà nó ra nhãn 0 -> 9 hay là một vector one-host)
- **Giải thích : One-host**
 - + Nếu có 10 số thì không dùng nhã từ 0 -> 9
 - + Mà phải là dùng nhã là một one-host vector nhị phân để cho các nhãn bình đẳng với nhau
 - + ...

3. Xây dựng mô hình CNN để training và predict nhận dạng chữ số viết tay

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)

x_train = x_train.astype('float32')

x_train /= 255
y_train = np_utils.to_categorical(y_train, 10)

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3),
```

- ...[0],28 ,28, 1 => 1 là ảnh xám , 3 là ảnh màu (chỉ có 2 giá trị này thôi)

- 1 là 1 ma trận , 3 là 3 ma trận (red, green, blue)

```
import cv2
import tensorflow as tf
from keras.utils import np_utils
from tensorflow.layers import Conv2D, MaxPooling2D, Flatten
```

```
model.fit(x_train, y_train, epochs=5)
# model.save('model.h5')

img_test = cv2.imread('anh4.png', 0) #anh của số 9
# cv2.imshow('anh test', img_test)
# cv2.waitKey(0)
img_test = img_test.reshape(1, 28, 28, 1)
print(model.predict(img_test))

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)

x_train = x_train.astype('float32')


---


x_train /= 255
y_train = np_utils.to_categorical(y_train, 10)

model = tf.keras.models.Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), input_shape=(28, 28, 1),
activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=32, kernel_size=(3, 3), activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(300, activation=tf.nn.relu),
    Dense(10, activation=tf.nn.softmax),
])
model.compile(optimizer='sgd',
loss='mean_squared_error',metrics=['accuracy'])
# model.summary()

model = tf.keras.models.Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), input_shape=(28, 28, 1),
activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(filters=32, kernel_size=(3, 3), activation=tf.nn.relu),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(300, activation=tf.nn.relu),
    Dense(10, activation=tf.nn.softmax),
])
model.summary()
```

Giới thiệu về keras

9.1 Giới thiệu về keras

Bản chất của bài toán Deep learning: Bạn có dữ liệu, bạn muốn máy tính học được các mô hình (model) từ dữ liệu, sau đó dùng mô hình đấy để dự đoán được các dữ liệu mới. Các bước cơ bản làm một bài toán deep learning :

1. Xây dựng bài toán
2. Chuẩn bị dữ liệu (dataset)
3. Xây dựng model
4. Định nghĩa loss function
5. Thực hiện backpropagation và áp dụng gradient descent để tìm các parameter gồm weight và bias để tối ưu loss function.
6. Dự đoán dữ liệu mới bằng model với các hệ số tìm được ở trên

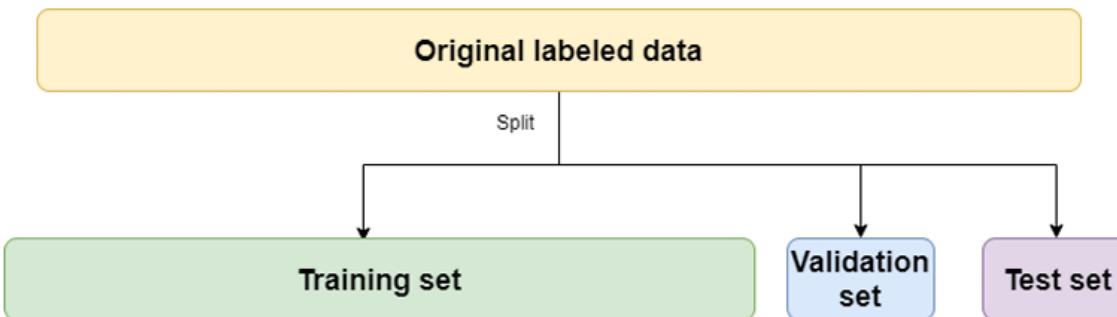
Bước xây dựng model thì áp dụng các kiến thức được trình bày trong bài neural network và convolutional neural network ta có thể xây dựng model hoàn chỉnh từ đầu bằng python. Tuy nhiên bước backpropagation trở nên phức tạp hơn rất nhiều. Khó để implement và tối ưu được tốc độ tính toán. **Đây là lý do các framework về deep learning ra đời với các đặc điểm:**

- Người dùng chỉ cần định nghĩa model và loss function, framework sẽ lo phần backpropagation.
- Việc định nghĩa layer, activation function, loss function đơn giản hơn cho người dùng. Ví dụ để thêm layer trong neural network chỉ cần báo là layer có bao nhiêu node và dùng hàm activation gì.

Có thể thấy tensorflow là framework phổ biến nhất tuy nhiên tensorflow khá khó sử dụng cho người mới bắt đầu. Nên tôi sẽ giới thiệu về keras: dễ sử dụng, thân thiện với người dùng nhưng đủ tốt để làm các bài toán về deep learning.

Keras - MNIST Dataset

Chuẩn bị dữ liệu



Cấu trúc chia dữ liệu

- Ta có một tập dữ liệu , ta sẽ tiến hành chia nó ra như sau
 - Dữ liệu để training
 - Dữ liệu để training **để đào tạo mô hình**
 - Dữ liệu để validation việc training **cho ra hiệu suất để chọn mô hình tốt nhất**

- Dữ liệu để test để kiểm tra xem model có chạy tốt với dữ liệu thực tế không

Ví dụ : MNIST là bộ cơ sở dữ liệu về chữ số viết tay, bao gồm 2 tập con: training set gồm 60.000 ảnh các chữ số viết tay và test set gồm 10.000 ảnh các chữ số. Trong MNIST dataset có 60.000 dữ liệu ở training set ở trong MNIST, ta sẽ chia ra 50.000 dữ liệu cho training set và 10.000 dữ liệu cho validation set. Vẫn giữ nguyên 10.000 dữ liệu của test set.

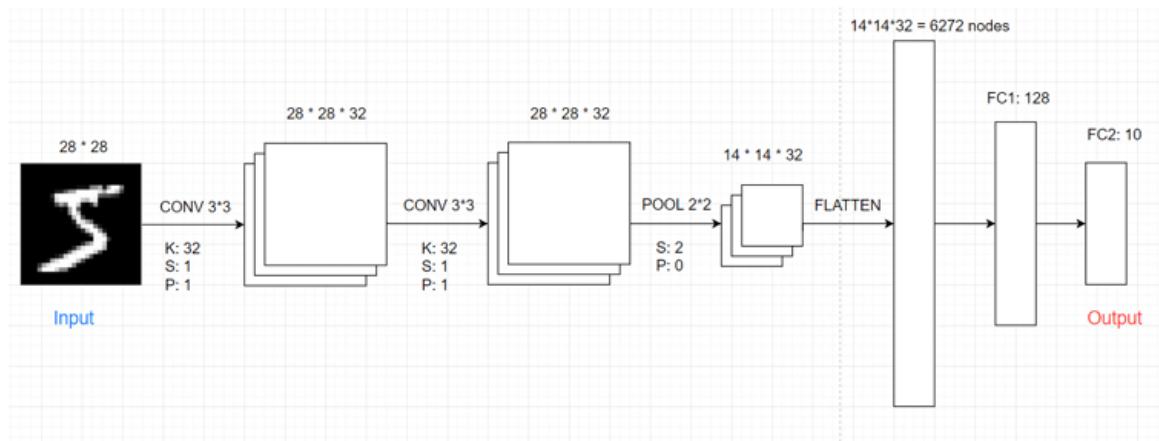
- Training set : 60.000 ảnh trở thành
 - Training set : 50.000 ảnh
 - Validation set : 10.000 ảnh
- Test set : 10.000 ảnh

- Training set, test set là gì? **Mục đích cuối cùng của machine learning không phải là predict tốt với dữ liệu mình đang có mà là để predict tốt các dữ liệu mới khi đi vào thực tế, vậy nên mình mong muốn model học được tổng quan hóa dữ liệu (generalization) thay vì chỉ nhớ các dữ liệu trong dataset.** Để đánh giá xem model có học không hay chỉ nhớ cũng như khi dùng ngoài thực tế thì performance sẽ thế nào, người ta chia dataset làm 3 tập training set, validation set và test set. Mình sẽ cho model học trên tập training set và đánh giá model trên tập validation set. **Nếu có nhiều hơn 1 mô hình (ví dụ VGG16, VGG19,...) thì mô hình nào cho performance tốt hơn trên tập validation set sẽ được chọn.** Và cuối cùng model tốt nhất sẽ được đánh giá trên tập test set làm hiệu suất của model khi dùng thực tế. Nhận thấy là tập test set không được dùng trong cả quá trình training chỉ đến cuối dùng để đánh giá.

Xây dựng model

Vì input của model là ảnh nên nghĩ ngay đến convolutional neural network (CNN).

Mô hình chung bài toán CNN: Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.



Hình 9.4: Model cho bài toán

Các lợi ích của việc sử dụng CNN cho việc xử lý ảnh bao gồm:

1. **Kết nối cục bộ:** Các tầng tích chập trong CNN cho phép mô hình học được các đặc trưng cục bộ từ ảnh, giúp nó hiểu được các mảnh nhỏ trong ảnh mà không cần phải biết vị trí cụ thể của chúng.
2. **Chia sẻ trọng số:** Các trọng số của bộ lọc tích chập có thể được chia sẻ, giúp giảm lượng tham số cần học trong mô hình và ngăn chặn việc quá mức hóa.
3. **Invariant với dịch chuyển:** CNN có khả năng học các đặc trưng có tính dịch chuyển, tức là các đặc trưng được phát hiện không thay đổi khi vị trí của chúng trong ảnh thay đổi.

Nhưng không chỉ dành riêng cho ảnh, CNN cũng có thể được áp dụng cho các loại dữ liệu không gian khác nhau như âm thanh, video, và dữ liệu không gian 3D.

Softmax function : Hàm kích hoạt dùng cho bài toán dữ đoán nhiều nhãn (kéo lên phía trên để xem) , thay vì chỉ dự đoán nhãn 0 hoặc 1 như sigmoid của logistic thì ta tính % của các nhãn , tổng % của các nhãn là 100 % , nhãn nào có % cao nhất thì lấy nhãn đó . **Hàm kích hoạt này cũng là một trong danh sách các hàm của Machine learning .** Còn nhiều hàm khác nữa , nếu có thời gian ra để đọc và tìm hiểu chi tiết cũng khá là hay .

- ❖ **Lưu ý :** Với các bài toán classification (phân loại) thì **nếu có 2 lớp** thì hàm activation ở output layer là **hàm sigmoid** và hàm loss function là **binary_crossentropy**, còn **nhiều hơn 2 lớp** thì hàm activation ở ouput layer là **hàm softmax** với loss function là **hàm categorical_crossentropy** .

Giảm chiều PCA ?



ChatGPT

Giảm chiều PCA (Principal Component Analysis) là quá trình giảm số chiều của dữ liệu bằng cách chuyển đổi dữ liệu từ không gian ban đầu sang không gian con ít chiều hơn, trong đó các thành phần chính (principal components) được chọn sao cho giữ lại một phần lượng thông tin quan trọng nhất có thể từ dữ liệu gốc.

Cụ thể, giảm chiều PCA thực hiện các bước sau:

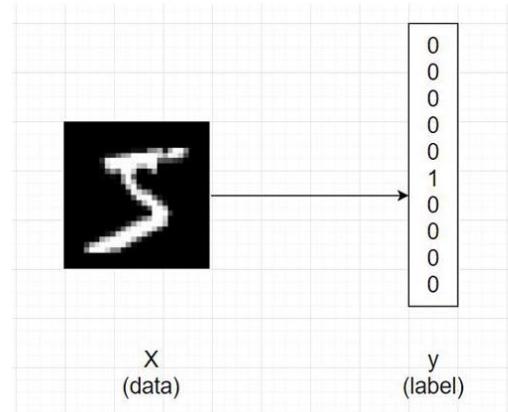
1. **Chuẩn bị dữ liệu:** Chuẩn bị dữ liệu bằng cách trừ đi giá trị trung bình của từng đặc trưng và chuẩn hóa dữ liệu nếu cần thiết.
2. **Tính toán ma trận hiệp phương sai:** Tính toán ma trận hiệp phương sai của dữ liệu, biểu thị mức độ biến thiên tương quan giữa các đặc trưng.
3. **Phân tích thành phần chính:** Sử dụng phương pháp phân tích giá trị suy biến (Singular Value Decomposition - SVD) hoặc phương pháp covaraince để tính toán các thành phần chính của dữ liệu.
4. **Chọn số lượng thành phần chính:** Chọn số lượng thành phần chính cần giữ lại, thường dựa trên mức độ giữ lại phần trăm lượng thông tin hoặc dựa trên tiêu chí kỹ thuật.
5. **Chiếu dữ liệu vào không gian mới:** Chiếu dữ liệu gốc vào không gian mới được tạo ra bởi các thành phần chính đã chọn.
6. **Giảm chiều:** Giữ lại số lượng thành phần chính đã chọn để tạo ra một phiên bản giảm chiều của dữ liệu.

Việc giảm chiều PCA có thể giúp giảm chi phí tính toán và tăng hiệu suất của các mô hình máy học, đặc biệt là khi làm việc với dữ liệu có số chiều lớn và không cần thiết. Tuy nhiên, việc giảm chiều cũng có thể dẫn đến mất mát thông tin nếu không thực hiện một cách cẩn thận và đúng đắn.

Trong một kiến trúc CNN, việc giảm chiều PCA thường không được thực hiện trực tiếp bằng cách sử dụng thuật toán PCA truyền thống. Thay vào đó, giảm chiều được thực hiện thông qua các lớp Convolutional và Pooling.

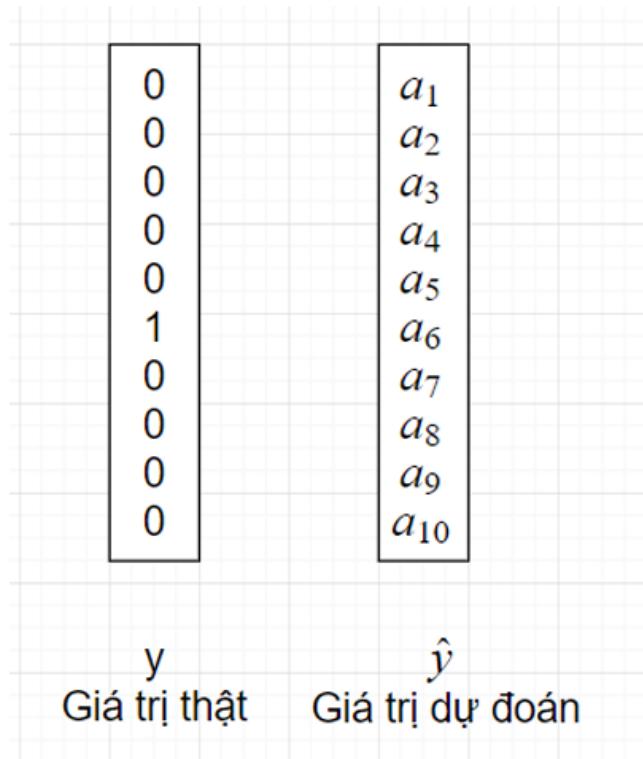
Loss function

Để định nghĩa loss function, trước hết ta dùng one-hot encoding chuyển đổi label của ảnh từ giá trị số sang vector cùng kích thước với output của model. Ví dụ:



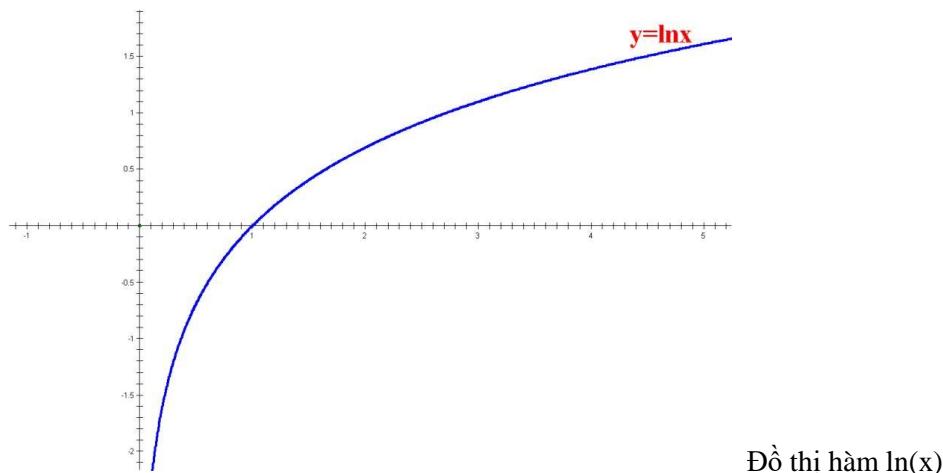
Để ý là label của data là số i là vector v kích thước 10×1 với $v_{i+1} = 1$ và các giá trị khác bằng 0. So với quy ước về phần trăm ở trên thì one-hot encoding có ý nghĩa là ta chắc chắn 100% ảnh này là số 5.

Giờ ta có giá trị thật (label) dạng one-hot encoding giá trị dự đoán ở output layer sau hàm softmax function cùng kích thước 10×1 . Ta cần định nghĩa hàm loss function để đánh giá độ tốt của model.

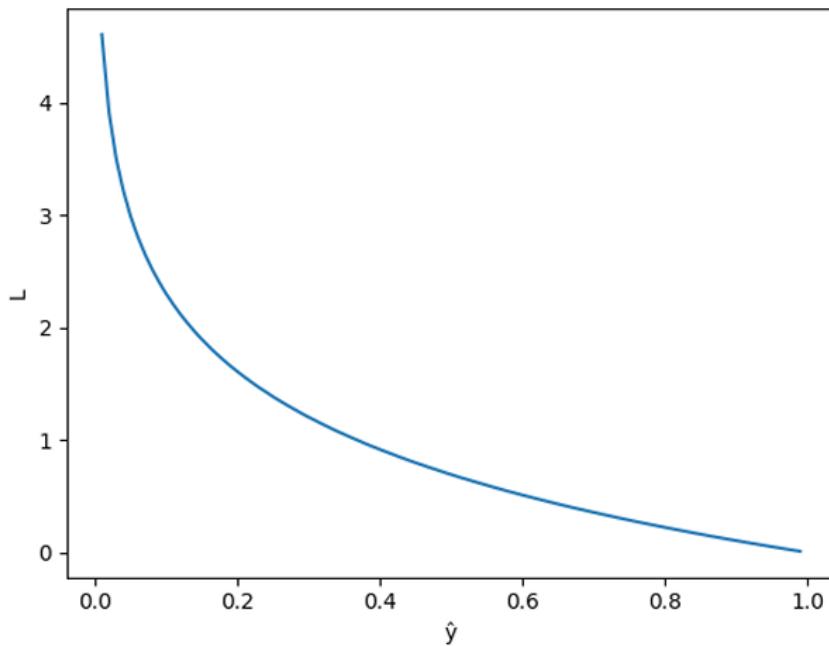


Mong muốn là a_6 gần 1 còn các giá trị a khác gần 0 vì như thế nghĩa là model dự đoán đúng được ảnh đầu vào là ảnh số 5. Ta định nghĩa loss function:

$$L = - \sum_{i=1}^{10} y_i * \log(\hat{y}_i)$$



Thử đánh giá hàm L. Giả sử ảnh là số 5 thì $L = -\log(\hat{y}_6)$.



Nhận xét:

- Hàm L giảm dần từ 0 đến 1
 - Khi model dự đoán \hat{y}_6 gần 1, tức giá trị dự đoán gần với giá trị thật y_6 thì L nhỏ, xấp xỉ 0
 - Khi model dự đoán \hat{y}_6 gần 0, tức giá trị dự đoán ngược lại giá trị thật y_6 thì L rất lớn
- => Hàm L nhỏ khi giá trị model dự đoán gần với giá trị thật và rất lớn khi model dự đoán sai, hay nói cách khác L càng nhỏ thì model dự đoán càng gần với giá trị thật. => Bài toán tìm model trở thành tìm giá trị nhỏ nhất của L.

Hàm loss function định nghĩa như trên trong keras gọi là "categorical_crossentropy"

Code Keras [HAY] – Nhận dạng chữ số viết tay

```
# 1. Thêm các thư viện cần thiết
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.utils import np_utils # thay cho from keras.utils import np_utils (error)
from keras.datasets import mnist

# 2. Load dữ liệu MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_val, y_val = X_train[50000:60000,:], y_train[50000:60000]
X_train, y_train = X_train[:50000,:], y_train[:50000]
print(X_train.shape)

# 3. Reshape lại dữ liệu cho đúng kích thước mà keras yêu cầu
```

```

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)

# 4. One hot encoding label (Y)
Y_train = np_utils.to_categorical(y_train, 10)
Y_val = np_utils.to_categorical(y_val, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print('Dữ liệu y ban đầu ', y_train[0])
print('Dữ liệu y sau one-hot encoding ', Y_train[0])

# 5. Định nghĩa model
model = Sequential()

# Thêm Convolutional layer với 32 kernel, kích thước kernel 3*3
# dùng hàm sigmoid làm activation và chỉ rõ input_shape cho layer đầu tiên
model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(28,28,1)))

# Thêm Convolutional layer
model.add(Conv2D(32, (3, 3), activation='sigmoid'))

# Thêm Max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))

# Flatten layer chuyển từ tensor sang vector
model.add(Flatten())

# Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid
model.add(Dense(128, activation='sigmoid'))

# Output layer với 10 node và dùng softmax function để chuyển sang xác suất.
model.add(Dense(10, activation='softmax'))

# 6. Compile model, chỉ rõ hàm loss_function nào được sử dụng, phương thức
# để tối ưu hàm loss function.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 7. Thực hiện train model với data
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
               batch_size=32, epochs=10, verbose=1)

# 8. Vẽ đồ thị loss, accuracy của training set và validation set
fig = plt.figure()
numOfEpoch = 10
plt.plot(np.arange(0, numOfEpoch), H.history['loss'], label='training loss')
plt.plot(np.arange(0, numOfEpoch), H.history['val_loss'], label='validation loss')
plt.plot(np.arange(0, numOfEpoch), H.history['accuracy'], label='accuracy') # sử dụng từ khóa accuracy thay vì
acc

```

```

plt.plot(np.arange(0, numOfEpoch), H.history['val_accuracy'], label='validation accuracy') # sử dụng từ khóa
val_accuracy thay vì val_acc
plt.title('Accuracy and Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss|Accuracy')
plt.legend()
# 9. Đánh giá model với dữ liệu test set
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)
# 10. Dự đoán ảnh
plt.imshow(X_test[100].reshape(28,28), cmap='gray')

y_predict = model.predict(X_test[100].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

```

Note

- Bước này chuyển đổi one-hot encoding label Y của ảnh ví dụ số 5 thành vector [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

HAY

```

# 4. One hot encoding label (Y)
y_train = np_utils.to_categorical(y_train, 10)
y_val = np_utils.to_categorical(y_val, 10)
y_test = np_utils.to_categorical(y_test, 10)
print('Dữ liệu y ban đầu ', y_train[0])
print('Dữ liệu y sau one-hot encoding ', y_train[0])
[22]   ✓ 0.0s
...
Dữ liệu y ban đầu 5
Dữ liệu y sau one-hot encoding [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

Python

- Trong Machine Learning và Deep Learning, "Sequential" thường được sử dụng để chỉ một kiểu mô hình hoặc kiến trúc mạng neural đơn giản, gồm các layer liên tiếp nhau theo một thứ tự cố định. Trong ngữ cảnh của thư viện Keras (một thư viện phổ biến cho Deep Learning), "**Sequential**" là **một lớp mô hình cho phép bạn xây dựng một mạng neural theo kiểu tuần tự.**

Một mạng Sequential trong Keras cho phép bạn xây dựng một mạng neural từng layer một, mà mỗi layer đều được kết nối với layer tiếp theo theo một thứ tự nhất định. Điều này làm cho việc xây dựng và huấn luyện mạng neural trở nên đơn giản và dễ dàng, đặc biệt là đối với các mạng neural có cấu trúc đơn giản như các mạng feedforward.

Cấu trúc của mạng Sequential rất phổ biến trong nhiều ứng dụng Deep Learning, như nhận diện hình ảnh, dự đoán chuỗi, và xử lý ngôn ngữ tự nhiên.

```

# 5. Định nghĩa model
model = Sequential()

# Thêm Convolutional layer với 32 kernel, kích thước kernel 3*3
# dùng hàm sigmoid làm activation và chỉ rõ input_shape cho layer đầu tiên
model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(28,28,1)))

# Thêm Convolutional layer
model.add(Conv2D(32, (3, 3), activation='sigmoid'))

# Thêm Max pooling layer
model.add(MaxPooling2D(pool_size=(2,2)))

# Flatten layer chuyển từ tensor sang vector
model.add(Flatten())

# Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid
model.add(Dense(128, activation='sigmoid'))

# Output layer với 10 node và dùng softmax function để chuyển sang xác suất.
model.add(Dense(10, activation='softmax'))

[23] ✓ 0.0s Python

```

- Load dữ liệu từ MNIST dataset, bao gồm 60.000 training set và 10.000 test set. Sau đó chia bộ traning set thành 2: 50.000 cho training set và 10.000 dữ liệu cho validation set.

```

# 1. Thêm các thư viện cần thiết
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from tensorflow.python.keras.utils import np_utils # thay cho from keras.utils import np_utils (error)
from keras.datasets import mnist

[32] ✓ 0.0s Python

Load dữ liệu từ MNIST dataset, bao gồm 60.000 training set và 10.000 test set. Sau đó chia bộ traning set thành 2: 50.000 cho training set và 10.000 dữ liệu cho validation set.

# 2. Load dữ liệu MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_val, y_val = X_train[50000:60000,:], y_train[50000:60000]
X_train, y_train = X_train[:50000,:], y_train[:50000]
print(X_train.shape) # (50000, 28, 28)
# 50.000 ảnh => cho ra 50.000 ma trận , mỗi ma trận có kích thước là W*H = 28*28 , đây là ảnh xám nên chỉ có W và H (D=1)
# Nếu là ảnh màu thì W*H*D với D = 3
[33] ✓ 0.1s Python
... (50000, 28, 28)

```

- ##### Trong keras dữ liệu đầu vào với yêu cầu là : Dữ liệu input cho mô hình convolutional neural network là 1 tensor 4 chiều (N, W, H, D)
- Dữ liệu input cho mô hình convolutional neural network là 1 tensor 4 chiều (N, W, H, D), trong bài này **là ảnh xám** nên W = H = 28, D = 1, N là số lượng ảnh cho mỗi lần training. Do dữ liệu ảnh ở trên có kích thước là (N, 28, 28) tức là (N, W, H) **nên cần reshape lại thành kích thước N * 28 * 28 * 1 để giống kích thước mà keras yêu cầu.**
- Bước này chuyển đổi one-hot encoding label Y của ảnh ví dụ số 5 thành vector [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
- ##### **HAY**
- Cấu trúc của mạng Sequential rất phổ biến trong nhiều ứng dụng Deep Learning, như nhận diện hình ảnh, dự đoán chuỗi, và xử lý ngôn ngữ tự nhiên.

```

... (50000, 28, 28)

Trong keras dữ liệu đầu vào với yêu cầu là : Dữ liệu input cho mô hình convolutional neural network là 1 tensor 4 chiều (N, W, H, D)

Dữ liệu input cho mô hình convolutional neural network là 1 tensor 4 chiều (N, W, H, D), trong bài này là ảnh xám nên W = H = 28, D = 1, N là số lượng ảnh cho mỗi lần training. Do dữ liệu ảnh ở trên có kích thước là (N, 28, 28) tức là (N, W, H) nên cần reshape lại thành kích thước N * 28 * 28 * 1 để giống kích thước mà keras yêu cầu.

# 3. Reshape lại dữ liệu cho đúng kích thước mà keras yêu cầu
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1) # (N, W, H, D) = (50.000, 28, 28, 1)
X_val = X_val.reshape(X_val.shape[0], 28, 28, 1) # (N, W, H, D) = (10.000, 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1) # (N, W, H, D) = (10.000, 28, 28, 1)

[34] ✓ 0.0s
Python

```

Bước này chuyển đổi one-hot encoding label Y của ảnh ví dụ số 5 thành vector [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

- Bước này định nghĩa model:
- 1. Model = Sequential() để nói cho keras là ta sẽ xếp các layer lên nhau để tạo model. Ví dụ input -> CONV -> POOL -> CONV -> POOL -> FLATTEN -> FC -> OUTPUT
- 2. Ở layer đầu tiên cần chỉ rõ input_shape của ảnh, input_shape = (W, H, D), ta dùng ảnh xám kích thước (28,28) nên input_shape = (28, 28, 1) . Nếu ảnh màu thì D = 3 , nên nếu ảnh màu sẽ là (28, 28, 3)
- 3. Khi thêm Convolutional Layer ta cần chỉ rõ các tham số: K (số lượng layer), kernel size (W, H), hàm activation sử dụng. cấu trúc: model.add(Conv2D(K, (W, H), activation='tên_hàm_activation'))
- 4. Khi thêm Maxpooling Layer cần chỉ rõ size của kernel, model.add(MaxPooling2D(pool_size=(W, H)))
- 5. Bước Flatten chuyển từ tensor sang vector chỉ cần thêm flatten layer.
- 6. Để thêm Fully Connected Layer (FC) cần chỉ rõ số lượng node trong layer và hàm activation sử dụng trong layer, cấu trúc: model.add(Dense(số_lượng_node activation='tên_hàm_activation'))
- Trong Machine Learning và Deep Learning, "Sequential" thường được sử dụng để chỉ một kiểu mô hình hoặc kiến trúc mạng neural đơn giản, gồm các layer liên tiếp nhau theo một thứ tự cố định. Trong ngữ cảnh của thư viện Keras (một thư viện phổ biến cho Deep Learning), "Sequential" là một lớp mô hình cho phép bạn xây dựng một mạng neural theo kiểu tuần tự.
- Một mạng Sequential trong Keras cho phép bạn xây dựng một mạng neural từng layer một, mà mỗi layer đều được kết nối với layer tiếp theo theo một thứ tự nhất định. Điều này làm cho việc xây dựng và huấn luyện mạng neural trở nên đơn giản và dễ dàng, đặc biệt là đối với các mạng neural có cấu trúc đơn giản như các mạng feedforward.

HAY

```
# 4. One hot encoding label (y)
Y_train = np_utils.to_categorical(y_train, 10)
Y_val = np_utils.to_categorical(y_val, 10)
Y_test = np_utils.to_categorical(y_test, 10)
print('Đữ liệu y ban đầu ', y_train[0])
print('Đữ liệu y sau one-hot encoding ', Y_train[0])

[5] ✓ 0.0s
.. Đữ liệu y ban đầu 5
Đữ liệu y sau one-hot encoding [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Python

Bước này định nghĩa model:

1. Model = Sequential() để nói cho keras là ta sẽ xếp các layer lên nhau để tạo model. Ví dụ input -> CONV -> POOL -> CONV -> POOL -> FLATTEN -> FC -> OUTPUT
2. Ở layer đầu tiên cần chỉ rõ input_shape của ảnh, input_shape = (W, H, D), ta dùng ảnh xám kích thước (28,28) nên input_shape = (28, 28, 1) . Nếu ảnh màu thì D = 3 , nên nếu ảnh màu sẽ là (28, 28, 3)
3. Khi thêm Convolutional Layer ta cần chỉ rõ các tham số: K (số lượng layer), kernel size (W, H), hàm activation sử dụng. cấu trúc: model.add(Conv2D(K, (W, H), activation='tên_hàm_activation'))
4. Khi thêm Maxpooling Layer cần chỉ rõ size của kernel, model.add(MaxPooling2D(pool_size=(W, H)))
5. Bước Flatten chuyển từ tensor sang vector chỉ cần thêm flatten layer.
6. Để thêm Fully Connected Layer (FC) cần chỉ rõ số lượng node trong layer và hàm activation sử dụng trong layer, cấu trúc: model.add(Dense(số_lượng_node activation='tên_hàm_activation'))

Trong Machine Learning và Deep Learning, "Sequential" thường được sử dụng để chỉ một kiểu mô hình hoặc kiến trúc mạng neural đơn giản, gồm các layer liên tiếp nhau theo một thứ tự cố định. Trong ngữ cảnh của thư viện Keras (một thư viện phổ biến cho Deep Learning), "Sequential" là một lớp mô hình cho phép bạn xây dựng một mạng neural theo kiểu tuần tự.

Một mạng Sequential trong Keras cho phép bạn xây dựng một mạng neural từng layer một, mà mỗi layer đều được kết nối với layer tiếp theo theo một thứ tự nhất định. Điều này làm cho việc xây dựng và huấn luyện mạng neural trở nên đơn giản và dễ dàng, đặc biệt là đối với các mạng neural có cấu trúc đơn giản như các mạng feedforward.

Cấu trúc của mạng Sequential rất phổ biến trong nhiều ứng dụng Deep Learning, như nhận diện hình ảnh, dự đoán chuỗi, và xử lý ngôn ngữ tự nhiên.

- Conv2D là hàm để tạo tích chập với kernel w .
- Với công thức : Conv2D(k,w,input_shape=(H,W,D)) . K là số kernel cần áp dụng , w là kích thước kernel . W và H là kích thước của ma trận ảnh . D là độ sâu , D=1 nếu ảnh xám và D=3 nếu là ảnh màu .
- Ví dụ : Conv2D(32, (3, 3), input_shape=(height, width, 3)) nghĩa là 32 kernel kích thước 3*3 và đầu vào là height , width với ảnh màu nên D=3 (3 lớp red, green, blue)
- pooling có 3 phương pháp : Max , min , Trung bình
- MaxPooling2D => Phương pháp pooling được sử dụng ở đây là Max
- model.add(MaxPooling2D(pool_size=(2,2))) => kích thước của pooling là 2*2
- model.add(Flatten()) => layer flatten để chuyển tensor về Vector đặc trưng
- model.add(Flatten()) => layer flatten để chuyển tensor về Vector đặc trưng

Cấu trúc của mạng Sequential rất phổ biến trong nhiều ứng dụng Deep Learning, như nhận diện hình ảnh, dự đoán chuỗi, và xử lý ngôn ngữ tự nhiên.

```
X_train.shape[1:]
```

Python

- Conv2D là hàm để tạo tích chập với kernel w .
- Với công thức : Conv2D(k,w,input_shape=(H,W,D)) . K là số kernel cần áp dụng , w là kích thước kernel . W và H là kích thước của ma trận ảnh . D là độ sâu , D=1 nếu ảnh xám và D=3 nếu là ảnh màu .
- Ví dụ : Conv2D(32, (3, 3), input_shape=(height, width, 3)) nghĩa là 32 kernel kích thước 3*3 và đầu vào là height , width với ảnh màu nên D=3 (3 lớp red, green, blue)

markdown

- pooling có 3 phương pháp : Max , min , Trung bình
- MaxPooling2D => Phương pháp pooling được sử dụng ở đây là Max
- model.add(MaxPooling2D(pool_size=(2,2))) => kích thước của pooling là 2*2
- model.add(Flatten()) => layer flatten để chuyển tensor về Vector đặc trưng
- model.add(Dense(128, activation='sigmoid')) => để thêm các lớp Fully connected
- model.add(Dense(10, activation='softmax')) => Vì là bài toán nhận dạng nhiều nhãn nên sẽ là activation softmax

[HAY]

Giải thích cho :

Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid

```
model.add(Dense(128, activation='sigmoid'))
```

Output layer với 10 node và dùng softmax function để chuyển sang xác suất.

```
model.add(Dense(10, activation='softmax'))
```

1. Trong mạng neural network, các lớp fully connected layer (hay còn gọi là dense layer)

2. Như đã nói đầu ra của y dự đoán phải là one-host (1 vector có chiều dài là 10 và gồm các giá trị 0 và 1)
=> Chính vì thế ta sử dụng hàm activation sigmoid với đặc thù là dữ liệu đầu ra của nó sẽ là 0 hoặc 1 .
Với số nodes là 128 .

=> Ta có thể hiểu đơn giản là :

- Sau khi qua Flatten thu được vector đặc trưng thì vector này sẽ trở thành input cho lớp input 128 node thu được z

- Sau khi có được z thì cho qua sigmoid và ra được giá trị a là 0 hoặc 1 , các giá trị này sẽ là input cho lớp tiếp theo gồm 10 node

- Như vậy ta sẽ có giá trị output là 10 giá trị 0 hoặc 1 => tạo thành one-host chiều dài là 10 gồm các giá trị 0 hoặc 1

- LUU Ý : 10 ở lớp dense cuối không được điều chỉnh vì bài toán ta nhận diện 10 chữ số nên có 10 nhãn

- Tuy nhiên số 128 là có thể điều chỉnh thành số khác , nếu hợp lý thì có thể thu được độ chính xác cao

3. Ta biết rằng đầu ra là các one-host có chiều dài là 10 gồm các giá trị 0 và 1 => Ta sử dụng Dense với 10 nghĩa là 10 node => nó sẽ cho ra vector one-host với chiều dài là 10 với hàm activation softmax cho dự đoán nhiều nhãn

[HAY]

Giải thích cho :

```
# Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid  
model.add(Dense(128, activation='sigmoid'))  
  
# Output layer với 10 node và dùng softmax function để chuyển sang xác suất.  
model.add(Dense(10, activation='softmax'))
```

1. Trong mạng neural network, các lớp fully connected layer (hay còn gọi là dense layer)
2. Như đã nói đầu ra của y dự đoán phải là one-host (1 vector có chiều dài là 10 và gồm các giá trị 0 và 1) => Chính vì thế ta sử dụng hàm activation sigmoid với đặc thù là dữ liệu đầu ra của nó sẽ là 0 hoặc 1 . Với số nodes là 128 . => Ta có thể hiểu đơn giản là :
 - o Sau khi qua Flatten thu được vector đặc trưng thì vector này sẽ trở thành input cho lớp input 128 node thu được z
 - o Sau khi có được z thì cho qua sigmoid và ra được giá trị a là 0 hoặc 1 , các giá trị này sẽ là input cho lớp tiếp theo gồm 10 node
 - o Như vậy ta sẽ có giá trị output là 10 giá trị 0 hoặc 1 => tạo thành one-host chiều dài là 10 gồm các giá trị 0 hoặc 1
 - o LUU Ý : 10 ở lớp dense cuối không được điều chỉnh vì bài toán ta nhận diện 10 chữ số nên có 10 nhãn
 - o Tuy nhiên số 128 là có thể điều chỉnh thành số khác , nếu hợp lý thì có thể thu được độ chính xác cao
3. Ta biết rằng đầu ra là các one-host có chiều dài là 10 gồm các giá trị 0 và 1 => Ta sử dụng Dense với 10 nghĩa là 10 node => nó sẽ cho ra vector one-host với chiều dài là 10 với hàm activation softmax do dự đoán nhiều nhãn

5. Định nghĩa model

```
model = Sequential()
```

```
# Thêm Convolutional layer với 32 kernel, kích thước kernel 3*3
```

```
# dùng hàm sigmoid làm activation và chỉ rõ input_shape cho layer đầu tiên
```

```
# model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(28,28,1)))
```

```
model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:])))
```

```
# 32, (3, 3) chính là 32 kernel w với kích thước là 3*3
```

```
# Thêm Convolutional layer
```

```
model.add(Conv2D(32, (3, 3), activation='sigmoid'))
```

```
# Thêm Max pooling layer
```

```
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
# MaxPooling2D
```

```
# kích thước của pooling là 2*2
```

```
# Flatten layer chuyển từ tensor sang vector
```

```
model.add(Flatten())
```

```
# Thêm Fully Connected layer với 128 nodes và dùng hàm sigmoid
```

```
model.add(Dense(128, activation='sigmoid'))
```

```
# Output layer với 10 node và dùng softmax function để chuyển sang xác suất.
```

```
model.add(Dense(10, activation='softmax'))
```

- Như đã nói : Đối với bài toán nhận dạng nhiều nhãn thì dùng hàm loss là categorical_crossentropy

- Ở đây đang dùng thuật toán tối ưu là : adam (không yêu cầu phải dùng adam , có thể đổi sang thuật toán khác cũng được)

Trong lệnh `model.compile` , tham số `metrics` được sử dụng để chỉ định các độ đo được sử dụng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện và kiểm tra.

Trong trường hợp `metrics=['accuracy']`, nó chỉ định rằng độ chính xác (accuracy) sẽ được tính toán và hiển thị trong quá trình huấn luyện và kiểm tra mô hình. Độ chính xác là tỷ lệ phần trăm của các dự đoán chính xác trên tổng số mẫu được dự đoán. Điều này giúp đánh giá khả năng của mô hình trong việc phân loại các mẫu vào các lớp được xác định.

The screenshot shows a Jupyter Notebook cell with the following content:

- Như đã nói : Đối với bài toán nhận dạng nhiều nhãn thì dùng hàm loss là categorical_crossentropy
- Ở đây đang dùng thuật toán tối ưu là : adam (không yêu cầu phải dùng adam , có thể đổi sang thuật toán khác cũng được)

```
# 6. Compile model, chỉ rõ hàm loss function nào được sử dụng, phương thức
# dùng để tối ưu hàm loss function.
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Trong lệnh `model.compile`, tham số `metrics` được sử dụng để chỉ định các độ đo được sử dụng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện và kiểm tra.

Trong trường hợp `metrics=['accuracy']`, nó chỉ định rằng độ chính xác (accuracy) sẽ được tính toán và hiển thị trong quá trình huấn luyện và kiểm tra. Độ chính xác là tỷ lệ phần trăm của các dự đoán chính xác trên tổng số mẫu được dự đoán. Điều này giúp đánh giá khả năng của mô hình trong việc phân loại các mẫu vào các lớp được xác định.

Trong đoạn code trên:

1. `model.fit` : Đây là phương thức để huấn luyện mô hình. Nó nhận đầu vào là các dữ liệu huấn luyện (`X_train`, `Y_train`) và dữ liệu validation (`X_val`, `Y_val`). Trong quá trình huấn luyện, mô hình sẽ được cập nhật dựa trên dữ liệu huấn luyện và sẽ được đánh giá trên dữ liệu validation để kiểm tra hiệu suất và tránh việc overfitting.
2. `batch_size=32` : Đây là kích thước của các batch dữ liệu được sử dụng trong mỗi lần cập nhật trọng số của mô hình. Trong trường hợp này, kích thước batch là 32, tức là 32 mẫu dữ liệu sẽ được sử dụng để cập nhật trọng số mỗi lần.
3. `epochs=10` : Số lần mà toàn bộ tập dữ liệu sẽ được sử dụng để huấn luyện mô hình. Trong trường hợp này, mô hình sẽ được huấn luyện qua 10 epochs.
4. `verbose=1` : Điều này chỉ định cách mà quá trình huấn luyện sẽ được hiển thị. Trong trường hợp này, `verbose=1` chỉ định rằng thông tin về quá trình huấn luyện sẽ được hiển thị, bao gồm thông tin về số lần lặp lại (epoch), loss function và các metrics được chọn.

Trong thư viện Keras, tham số `verbose` trong phương thức `model.fit` có thể nhận các giá trị sau:

1. `verbose=0` : Không hiển thị bất kỳ thông tin nào về quá trình huấn luyện trên màn hình.
2. `verbose=1` : Hiển thị tiến trình của quá trình huấn luyện trên màn hình. Thông tin bao gồm số lần lặp lại (epoch), loss function và các metrics đã chọn.
3. `verbose=2` : Hiển thị thông tin của mỗi epoch trên màn hình, nhưng không hiển thị chi tiết của mỗi lần lặp lại trong epoch.
4. `verbose=3` : Chỉ hiển thị thông báo sau khi mỗi epoch kết thúc, mà không hiển thị bất kỳ thông tin nào trong quá trình huấn luyện của các epoch.

Trong cả bốn trường hợp, quá trình huấn luyện đều được thực hiện nhưng cách hiển thị thông tin khác nhau sẽ giúp người dùng có thể theo dõi và kiểm soát quá trình huấn luyện một cách linh hoạt.

```
# 6. Compile model, chỉ rõ hàm loss function nào được sử dụng, phương thức
# dùng để tối ưu hàm loss function.
```

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

Trong lệnh `model.compile`, tham số `metrics` được sử dụng để chỉ định các độ đo được sử dụng để đánh giá hiệu suất của mô hình trong quá trình huấn luyện và kiểm tra.

Trong trường hợp `metrics=['accuracy']`, nó chỉ định rằng độ chính xác (accuracy) sẽ được tính toán và hiển thị trong quá trình huấn luyện và kiểm tra mô hình. Độ chính xác là tỷ lệ phần trăm của các dự đoán chính xác trên tổng số mẫu được dự đoán. Điều này giúp đánh giá khả năng của mô hình trong việc phân loại các mẫu vào các lớp được xác định.

```
# 7. Thực hiện train model với data
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
               batch_size=32, epochs=10, verbose=1)

[39] ✓ 1m 36.7s
```

... Epoch 1/10
1563/1563 10s 6ms/step - accuracy: 0.7791 - loss: 0.7187 - val_accuracy: 0.9793 - val_loss: 0.0801
Epoch 2/10
1563/1563 9s 6ms/step - accuracy: 0.9796 - loss: 0.0698 - val_accuracy: 0.9854 - val_loss: 0.0538
Epoch 3/10
1563/1563 10s 6ms/step - accuracy: 0.9884 - loss: 0.0404 - val_accuracy: 0.9852 - val_loss: 0.0551
Epoch 4/10
1563/1563 9s 6ms/step - accuracy: 0.9924 - loss: 0.0278 - val_accuracy: 0.9866 - val_loss: 0.0474
Epoch 5/10
1563/1563 9s 6ms/step - accuracy: 0.9943 - loss: 0.0204 - val_accuracy: 0.9863 - val_loss: 0.0472
Epoch 6/10
1563/1563 10s 6ms/step - accuracy: 0.9959 - loss: 0.0143 - val_accuracy: 0.9875 - val_loss: 0.0424
Epoch 7/10
1563/1563 9s 6ms/step - accuracy: 0.9974 - loss: 0.0103 - val_accuracy: 0.9895 - val_loss: 0.0403
Epoch 8/10
1563/1563 10s 6ms/step - accuracy: 0.9984 - loss: 0.0070 - val_accuracy: 0.9884 - val_loss: 0.0424
Epoch 9/10
1563/1563 10s 6ms/step - accuracy: 0.9988 - loss: 0.0060 - val_accuracy: 0.9887 - val_loss: 0.0425
Epoch 10/10
1563/1563 10s 6ms/step - accuracy: 0.9991 - loss: 0.0050 - val_accuracy: 0.9892 - val_loss: 0.0426

7. Thực hiện train model với data

```
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
               batch_size=32, epochs=10, verbose=1)
```

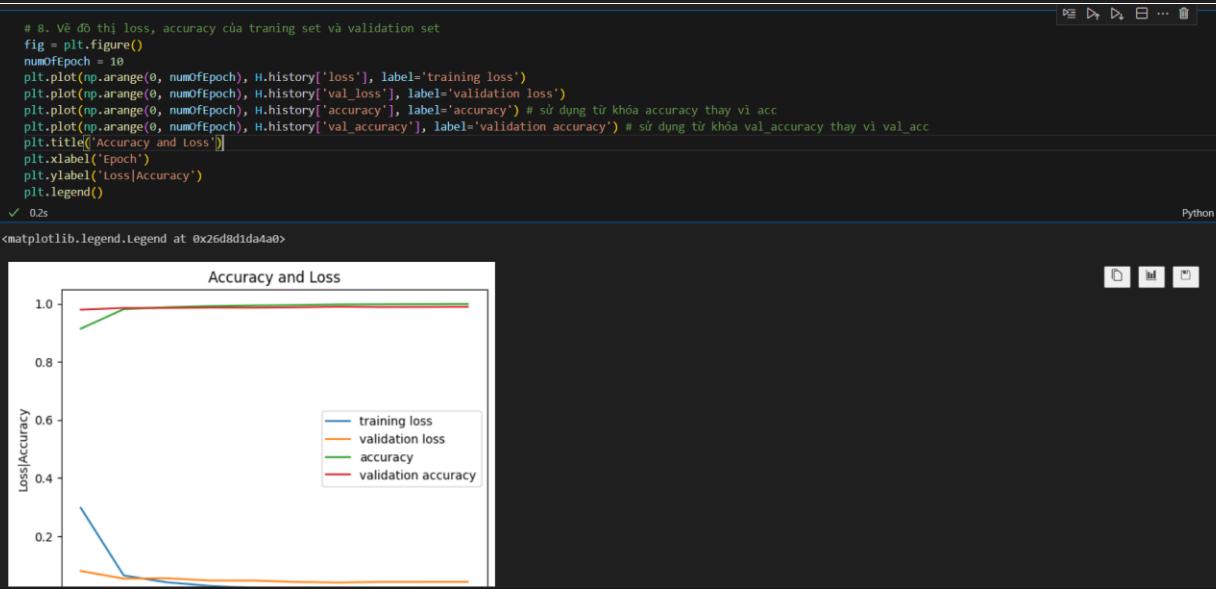
Trong đoạn code trên:

1. `model.fit`: Đây là phương thức để huấn luyện mô hình. Nó nhận đầu vào là các dữ liệu huấn luyện (`X_train`, `Y_train`) và dữ liệu validation (`X_val`, `Y_val`). Trong quá trình huấn luyện, mô hình sẽ được cập nhật dựa trên dữ liệu huấn luyện và sẽ được đánh giá trên dữ liệu validation để kiểm tra hiệu suất và tránh việc overfitting.
2. `batch_size=32`: Đây là kích thước của các batch dữ liệu được sử dụng trong mỗi lần cập nhật trọng số của mô hình. Trong trường hợp này, kích thước batch là 32, tức là 32 mẫu dữ liệu sẽ được sử dụng để cập nhật trọng số mỗi lần.
3. `epochs=10`: Số lần mà toàn bộ tập dữ liệu sẽ được sử dụng để huấn luyện mô hình. Trong trường hợp này, mô hình sẽ được huấn luyện qua 10 epochs.
4. `verbose=1`: Điều này chỉ định cách mà quá trình huấn luyện sẽ được hiển thị. Trong trường hợp này, `verbose=1` chỉ định rằng thông tin về quá trình huấn luyện sẽ được hiển thị, bao gồm thông tin về số lần lặp lại (epoch), loss function và các metrics được chọn.

Trong thư viện Keras, tham số `verbose` trong phương thức `model.fit` có thể nhận các giá trị sau:

1. `verbose=0`: Không hiển thị bất kỳ thông tin nào về quá trình huấn luyện trên màn hình.
2. `verbose=1`: Hiển thị tiến trình của quá trình huấn luyện trên màn hình. Thông tin bao gồm số lần lặp lại (epoch), loss function và các metrics đã chọn.
3. `verbose=2`: Hiển thị thông tin của mỗi epoch trên màn hình, nhưng không hiển thị chi tiết của mỗi lần lặp lại trong epoch.
4. `verbose=3`: Chỉ hiển thị thông báo sau khi mỗi epoch kết thúc, mà không hiển thị bất kỳ thông tin nào trong quá trình huấn luyện của các epoch.

Trong cả bốn trường hợp, quá trình huấn luyện đều được thực hiện nhưng cách hiển thị thông tin khác nhau sẽ giúp người dùng có thể theo dõi và kiểm soát quá trình huấn luyện một cách linh hoạt.



9. Đánh giá model với dữ liệu test set

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)
```

Kết quả trả về `[0.035371359437704086, 0.9887999892234802]` trong trường hợp này là kết quả của việc đánh giá model trên tập dữ liệu test. Cụ thể:

- Phần tử đầu tiên (`0.035371359437704086`) là giá trị của hàm loss trên tập dữ liệu test. Đây là giá trị trung bình của loss function trên các mẫu dữ liệu trong tập test.
- Phần tử thứ hai (`0.9887999892234802`) là độ chính xác của model trên tập dữ liệu test, được tính dưới dạng tỷ lệ giữa số lượng dự đoán đúng và tổng số lượng mẫu trong tập test.

Kết quả này cho thấy model có độ chính xác khoảng 98.88% trên tập dữ liệu test, và giá trị của hàm loss trên tập test là khoảng 0.035. Đây là các thông số quan trọng để đánh giá hiệu suất của model trên dữ liệu mới.

Ta sẽ dùng kết quả đánh giá của model với test set để làm kết quả cuối cùng của model. Tức model của chúng ta dự đoán chữ số có độ chính xác 98.92% với MNIST dataset. Nghĩa là dự đoán khoảng 100 ảnh thì sai 1 ảnh.

```

# 9. Đánh giá model với dữ liệu test set
score = model.evaluate(X_test, Y_test, verbose=0)
print(score)

[41] ✓ 0.7s
... [0.035371359437704086, 0.9887999892234802]

Python

```

Kết quả trả về **[0.035371359437704086, 0.9887999892234802]** trong trường hợp này là kết quả của việc đánh giá model trên tập dữ liệu test. Cụ thể:

- Phần tử đầu tiên (**0.035371359437704086**) là giá trị của hàm loss trên tập dữ liệu test. Đây là giá trị trung bình của loss function trên các mẫu dữ liệu trong tập test.
- Phần tử thứ hai (**0.9887999892234802**) là độ chính xác của model trên tập dữ liệu test, được tính dưới dạng tỷ lệ giữa số lượng dự đoán đúng và tổng số lượng mẫu trong tập test.

Kết quả này cho thấy model có độ chính xác khoảng 98.88% trên tập dữ liệu test, và giá trị của hàm loss trên tập test là khoảng 0.035. Đây là các thông số quan trọng để đánh giá hiệu suất của model trên dữ liệu mới.

Ta sẽ dùng kết quả đánh giá của mode với test set để làm kết quả cuối cùng của model. Tức model của chúng ta đã đoán chữ số có độ chính xác 98.92% với MNIST dataset. Nghĩa là dự đoán khoảng 100 ảnh thì sai 1 ảnh.

```

# 10. Dự đoán ảnh
plt.imshow(X_test[100].reshape(28,28), cmap='gray')

y_predict = model.predict(X_test[100].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

[42] ✓ 0.1s
1/1 ━━━━━━ 0s 52ms/step
Giá trị dự đoán: 6

```

Lưu lại model và load ra lại để sử dụng khi cần (không cần phải chạy lại file)

```

# Lưu model
model.save('my_model.h5')

from keras.models import load_model

# Load model
loaded_model = load_model('my_model.h5')

# 10. Dự đoán ảnh
plt.imshow(X_test[0].reshape(28,28), cmap='gray')

y_predict = loaded_model.predict(X_test[0].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

[44] ✓ 0.2s
... WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native TensorFlow format.
... WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
1/1 ━━━━━━ 0s 50ms/step
Giá trị dự đoán: 7

```

```
# Lưu model
model.save('my_model.h5')
```

```

from keras.models import load_model

# Load model
loaded_model = load_model('my_model.h5')

# 10. Dự đoán ảnh
plt.imshow(X_test[0].reshape(28,28), cmap='gray')

y_predict = loaded_model.predict(X_test[0].reshape(1,28,28,1))
print('Giá trị dự đoán: ', np.argmax(y_predict))

```

Danh sách các tham số có thể thay đổi để tối ưu model

Với bài toán nhận diện chữ số từ 0 đến 9

Danh sách các tham số có thể thay đổi để tối ưu model

- Thay đổi số lượng kernel : k (32) và kích thước kernel : w(3, 3) . model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:])))
- Thay đổi hàm kích hoạt tại các lớp tích chập Conv2D : activation (sigmoid) bằng các hàm khác
- Pooling : MaxPooling2D thay bằng phương pháp khác ví dụ AveragePooling2D (trung bình) và thay đổi kích thước pooling
- Số lượng node trong các lớp Fully Connected (128) có thể thay đổi thành một số nodes khác
- Hàm loss và optimizer: Thủ nghiệm với các hàm loss khác nhau như 'categorical_crossentropy', 'mean_squared_error' và các optimizer khác nhau như 'adam', 'sgd', 'rmsprop' để xem liệu cách lựa chọn này có ảnh hưởng đến hiệu suất của mô hình không.
- Batch_size: Thay đổi batch_size có thể ảnh hưởng đến tốc độ huấn luyện và độ chính xác của mô hình , có thể thay đổi trong tốc độ học và hiệu suất của mô hình.
- Epochs: Thay đổi số lượng epochs có thể giúp bạn xác định khi nào mô hình đã hội tụ và khi nào nó bắt đầu overfitting, có thể giảm số lượng epochs nếu thấy mô hình bắt đầu overfitting trên dữ liệu huấn luyện hoặc tăng số lượng epochs nếu mô hình vẫn chưa hội tụ.
- Metrics, bạn có thể đo lường hiệu suất của mô hình bằng các phương pháp khác nhau, không chỉ là accuracy.
 - o Accuracy: Tỷ lệ dự đoán chính xác trên tất cả các mẫu.
 - o Precision: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số dự đoán dương tính.
 - o Recall: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số mẫu thực sự là dương tính.
 - o F1-score: Trung bình điều hòa giữa Precision và Recall.
 - o ROC-AUC: Diện tích dưới đường cong ROC (Receiver Operating Characteristic), thường được sử dụng cho các bài toán phân loại nhị phân.
 - o Mean Squared Error (MSE): Trung bình của bình phương của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.
 - o Mean Absolute Error (MAE): Trung bình của giá trị tuyệt đối của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.
 - o R-squared (R2): Phần trăm phương sai của biến phụ thuộc mà mô hình giải thích được.

Danh sách các tham số không nên thay đổi

- Kích thước dữ liệu đầu vào , đây là các con số gần như cố định , D=1 hay D=3 phụ thuộc vào ảnh xám hay ảnh màu
- Hàm loss của bài toán , ví dụ như đây là bài nhiều hơn 2 nhãn thì dùng categorical cross-entropy , không dùng sigmoid (2 nhãn)
- Thuật toán tối ưu : Adam , thường thì adam là thuật toán tốt nhất trong các thuật toán tối ưu , nên có thay đổi thì có thể accuracy cũng không thay đổi mấy
- Các lớp Dense cuối như 2 lớp cuối : Các hàm kích hoạt của nó thì sigmoid sau đó đến softmax là không nên thay đổi vì nó liên quan đến yêu cầu bài toán
- Số node của lớp Dense cuối (10) phụ thuộc vào số nhãn cần nhận diện của bài toán

Với bài toán nhận diện chữ số từ 0 đến 9

Danh sách các tham số có thể thay đổi để tối ưu model

- Thay đổi số lượng kernel : k (32) và kích thước kernel : w(3, 3) . model.add(Conv2D(32, (3, 3), activation='sigmoid', input_shape=(X_train.shape[1:])))
- Thay đổi hàm kích hoạt tại các lớp tích chập Conv2D : activation ('sigmoid' bằng các hàm khác
- Pooling : MaxPooling2D thay bằng phương pháp khác ví dụ AveragePooling2D (trung bình) và thay đổi kích thước pooling
- Số lượng node trong các lớp Fully Connected (128) có thể thay đổi thành một số nodes khác
- Hàm loss và optimizer: Thử nghiệm với các hàm loss khác nhau như 'categorical_crossentropy', 'mean_squared_error' và các optimizer khác nhau như 'adam', 'sgd', 'rmsprop' để xem liệu cách lựa chọn này có ảnh hưởng đến hiệu suất của mô hình không.
- Batch_size: Thay đổi batch_size có thể giúp bạn xác định khi nào mô hình đã hội tụ và khi nào nó bắt đầu overfitting, có thể giảm số lượng epochs nếu thấy mô hình bắt đầu overfitting trên dữ liệu huấn luyện hoặc tăng số lượng epochs nếu mô hình vẫn chưa hội tụ.
- Metrics, bạn có thể đếm lượng hiệu suất của mô hình bằng các phương pháp khác nhau, không chỉ là accuracy.
 - Accuracy: Tỷ lệ dự đoán chính xác trên tất cả các mẫu.
 - Precision: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số dự đoán dương tính.
 - Recall: Tỷ lệ giữa số lượng dự đoán dương tính đúng và tổng số mẫu thực sự là dương tính.
 - F1-score: Trung bình điều hòa giữa Precision và Recall.
 - ROC-AUC: Diện tích dưới đường cong ROC (Receiver Operating Characteristic), thường được sử dụng cho các bài toán phân loại nhị phân.
 - Mean Squared Error (MSE): Trung bình của bình phương của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.
 - Mean Absolute Error (MAE): Trung bình của giá trị tuyệt đối của sự khác biệt giữa các giá trị dự đoán và giá trị thực tế.
 - R-squared (R2): Phần trăm phương sai của biến phụ thuộc mà mô hình giải thích được.

Danh sách các tham số không nên thay đổi

- Kích thước dữ liệu đầu vào , đây là các con số gần như cố định , D=1 hay D=3 phụ thuộc vào ảnh xám hay ảnh màu
- Hàm loss của bài toán , ví dụ như đây là bài nhiều hơn 2 nhãn thì dùng categorical cross-entropy , không dùng sigmoid (2 nhãn)
- Thuật toán tối ưu : Adam , thường thì adam là thuật toán tốt nhất trong các thuật toán tối ưu , nên có thay đổi thì có thể accuracy cũng không thay đổi mấy
- Các lớp Dense cuối như 2 lớp cuối : Các hàm kích hoạt của nó thì sigmoid sau đó đến softmax là không nên thay đổi vì nó liên quan đến yêu cầu bài toán
- Số node của lớp Dense cuối (10) phụ thuộc vào số nhãn cần nhận diện của bài toán

Nhận xét transfer learning

Như đã code ở trên , ta thấy rằng các lớp cuối liên quan nhiều đến bài toán ta cần làm gì , số nhãn sẽ thay đổi với bài toán và còn nhiều thứ khác ,...

Chính vì thế như đã nói ở trên , ví dụ ta có được mô hình đã được xây dựng và training sẵn ví dụ để nhận diện đối tượng nào đó , giờ ta muốn dùng lại để nhận diện riêng cho mình một cái gì đó ví dụ nhận diện các loài hoa , nhận diện 10 loài chẳng hạn , thì ta phải re train , có nghĩa là đào tạo lại , sử dụng kỹ thuật transferlearning để mở các lớp cuối của mô hình đã được đào tạo đó rồi đào tạo lại các mô hình cuối đó để tương thích với bài toán của ta .

Đúng, bạn đã hiểu đúng về việc sử dụng transfer learning. Transfer learning là một kỹ thuật phổ biến trong deep learning, trong đó chúng ta sử dụng các mô hình đã được huấn luyện trước trên một tập dữ liệu lớn và sử dụng lại các trọng số đã học để giúp huấn luyện mô hình mới cho một nhiệm vụ tương tự hoặc liên quan.

Khi sử dụng transfer learning, bạn có thể thực hiện các bước sau:

1. Chọn một mô hình đã được huấn luyện trước, chẳng hạn như một trong những mô hình đã được public như VGG, ResNet, hoặc Inception.

2. Loại bỏ hoặc đóng băng (freeze) các lớp cuối cùng của mô hình (các lớp fully connected) để không cập nhật trọng số của chúng trong quá trình huấn luyện.
3. Thay thế các lớp cuối cùng bằng các lớp mới phù hợp với nhiệm vụ của bạn, ví dụ như một lớp fully connected với số lượng đầu ra tương ứng với số lượng lớp của bài toán của bạn.
4. Tiến hành huấn luyện lại mô hình trên dữ liệu của bạn, thường là chỉ huấn luyện các lớp mới mà bạn đã thêm vào.

Như vậy, bạn có thể sử dụng transfer learning để tận dụng tri thức đã học được từ các mô hình đã được huấn luyện trước và áp dụng nó vào bài toán cụ thể của mình mà không cần phải huấn luyện từ đầu trên một tập dữ liệu lớn. Điều này giúp tiết kiệm thời gian và công sức trong việc xây dựng và huấn luyện mô hình.

Ứng dụng của việc phân loại ảnh

- Chuẩn đoán ảnh X-ray của bệnh nhân có bị ung thư hay không
- Phân loại, nhận diện được các chữ, số viết tay => tự động đọc được biển số xe, văn bản.
- Phân loại được các biển báo giao thông => hỗ trợ cho ô tô tự lái

Bài tập

1. Tại sao cần dùng softmax activation ở layer cuối cùng?
2. (a) Thiết kế và training model CNN phân loại chó, mèo, nguồn dữ liệu <https://www.kaggle.com/c/dogs-vs-cats/data>
 (b) Thay đổi learning rate, epoch, batch size xem accuracy thay đổi thế nào? Thủ giải thích tại sao?
 (c) Thay đổi model bằng cách thêm lần lượt các layer pooling, dropout, batch normalization, activation function và thay đổi kernel size bằng (5x5), (7x7) xem số lượng parameter, accuracy thay đổi thế nào? Thủ giải thích tại sao?
3. Xây dựng model CNN cho bài toán phân loại ảnh với dữ liệu CIFAR10 dataset bao gồm 50,000 training set và 10.000 test set ảnh màu kích thước 32x32 cho 10 thể loại khác nhau (máy bay, ô tô, thuyền, chim, chó, mèo, ngựa,...).

```
# Load dữ liệu cifar10
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Keras - Dog and Cat Classification

- Thiết kế và training model CNN phân loại chó, mèo, nguồn dữ liệu : [Dogs vs. Cats | Kaggle](https://www.kaggle.com/competitions/dogs-vs-cats/data) (<https://www.kaggle.com/competitions/dogs-vs-cats/data>)

Dogs vs. Cats

train.zip (569.55 MB)

Unable to show preview

Previews for binary data are not supported

Data Explorer

853.96 MB

- sampleSubmission.csv
- test1.zip
- train.zip

Summary

- 3 files
 - .zip
 - .csv
- 2 columns
 - Id
 - # Integer

Download All

>_ kaggle competitions download -c dogs-vs-cats

Notebooks

DogsCats-TransferLearningResnet

Dog_vs_Cat_Classifier_CNN

Cat VS Dog in ML

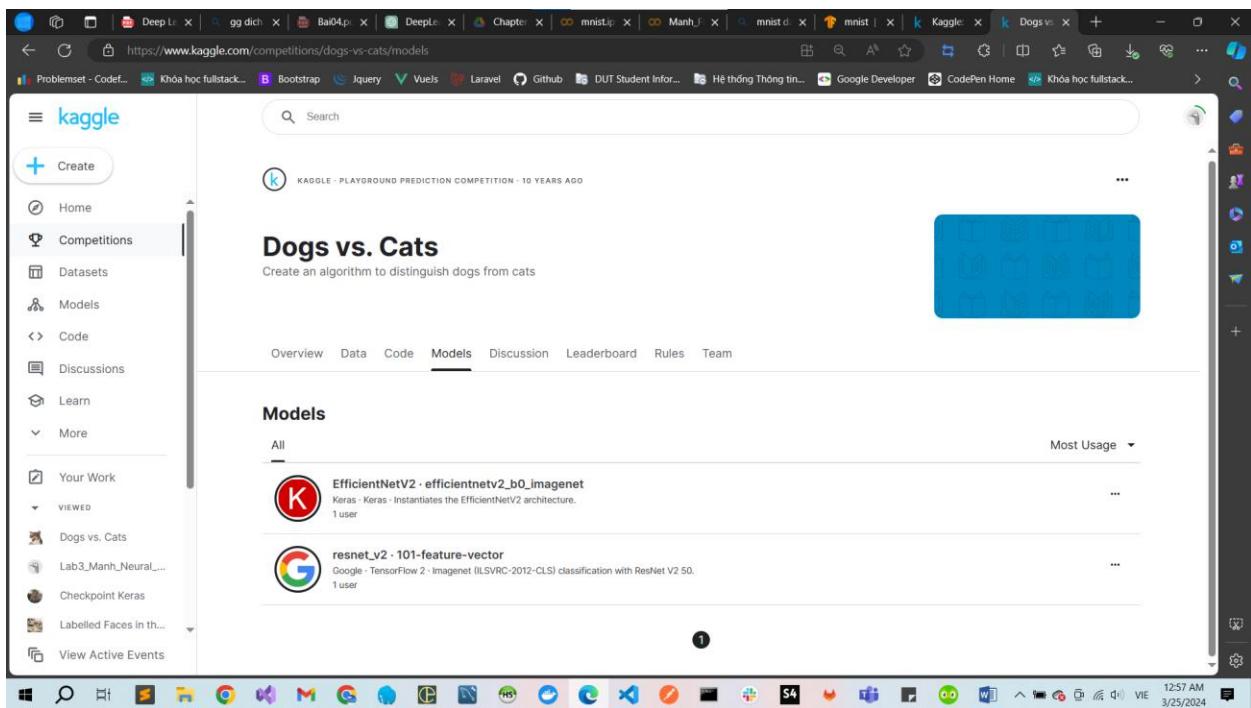
Dog and Cat classification 2024

Image Classification with Customise Model

⇒ Có thể tham khảo những code này : <https://www.kaggle.com/competitions/dogs-vs-cats/code>

Danh sách model tốt nhất cho bài toán với data này :

- EfficientNetV2 · efficientnetv2_b0_imagenet
- resnet_v2 · 101-feature-vector



Chuẩn bị dữ liệu

- Chuyển dữ liệu ảnh sang dữ liệu dạng ma trận

Xây dựng model

10. Keras - Ứng dụng CNN cho ô tô tự lái - self-driving-car-sim

Giới thiệu mô phỏng ô tô tự lái

Bài toán ô tô tự lái

Xây dựng bài toán

Bạn muốn dự đoán góc lái của vô lăng bằng ảnh ở camera trên ô tô. Vì input là ảnh nên nghĩ ngay đến việc dùng CNN. Dữ liệu training set sẽ được lấy từ training mode của phần mềm mô phỏng.

Chuẩn bị dữ liệu

Sau khi bạn chọn training mode thì hãy làm quen với việc di chuyển của ô tô bằng các phím mũi tên. Đến khi bạn lái mượt rồi thì chọn nút record. Chọn folder bạn muốn lưu dữ liệu và chọn select.

Bạn lái xe khoảng 10 phút sẽ ra khoảng 18000 ảnh (6000 ảnh từ mỗi camera).

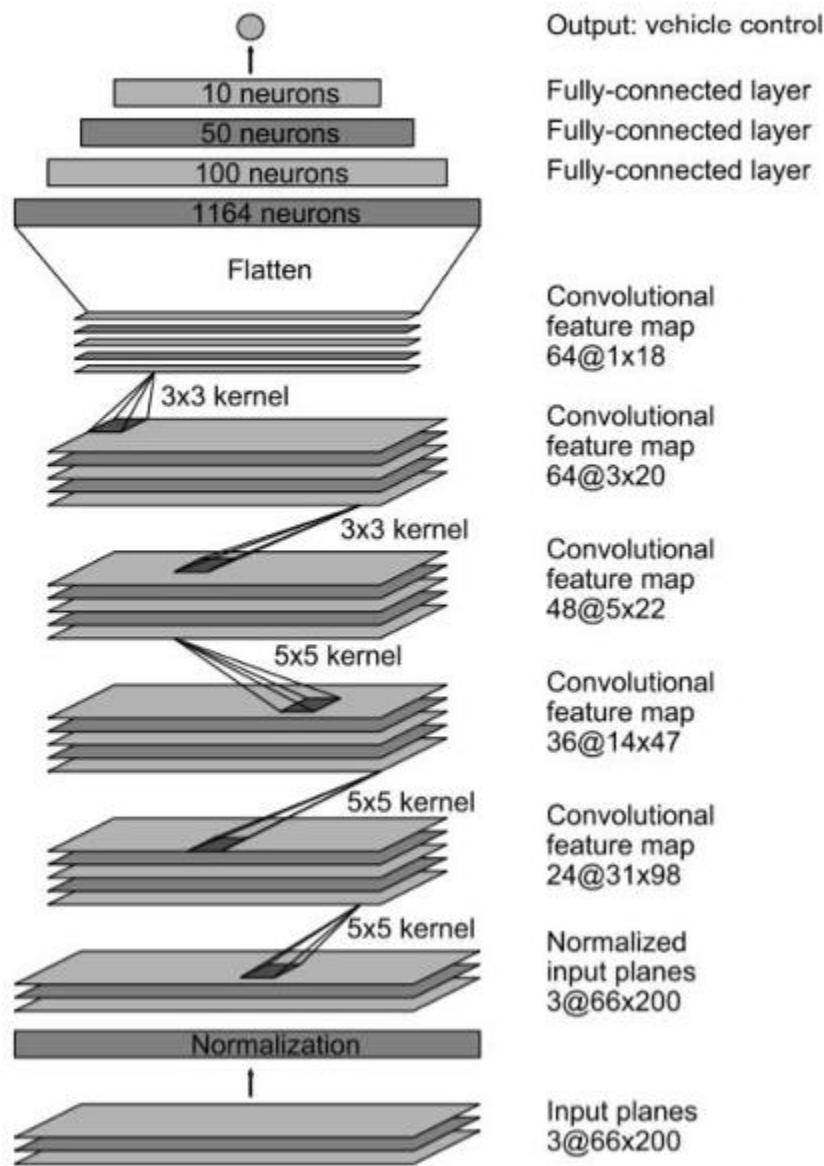
Bạn cũng thấy file driving_log.csv để mô tả dữ liệu

| | | | | | | |
|---------------------------|-----------------|-----------------|---------|-------------|-------|--------|
| đường dẫn ảnh camera giữa | ảnh camera trái | ảnh camera phải | góc lái | độ giảm tốc | phanh | tốc độ |
|---------------------------|-----------------|-----------------|---------|-------------|-------|--------|

Tiền xử lý dữ liệu (Preprocessing)

- Chi tiết ở file : [utils.py](#)
- Ảnh màu từ camera ở ô tô có kích thước 320*160
- Tăng cường dữ liệu (sinh dữ liệu)(làm giàu dữ liệu)
 1. Lật ngược ảnh
 2. Thêm sáng
 3. Thêm tối
 4. Dịch chuyển ảnh
 5. Others (còn nhiều ảnh khác nữa : Đục lỗ ảnh ,...)

Xây dựng model



Input layer: Ảnh màu kích thước 66*200

Output layer: 1 node dự đoán góc lái của vô lăng

Normalization : Quá trình normalization dữ liệu input trước khi đi qua các lớp tích chập trong mạng neural là một phần quan trọng trong quá trình xử lý dữ liệu. Quá trình này thường được thực hiện để đảm bảo rằng dữ liệu đầu vào có phân phối đồng đều và ổn định, giúp tăng tốc độ hội tụ và hiệu suất của mạng neural.

Khi dữ liệu input được chuẩn hóa, mỗi điểm dữ liệu có thể được điều chỉnh sao cho có giá trị trung bình gần bằng 0 và phương sai gần bằng 1. Trong trường hợp của ảnh màu, dữ liệu input thường được chuẩn hóa bằng cách trừ đi giá trị trung bình và chia cho độ lệch chuẩn của từng kênh màu (R, G, B) của ảnh.

Dữ liệu input sau quá trình chuẩn hóa sẽ được gọi là "Normalized input planes". Sau khi qua quá trình này, dữ liệu được chuẩn hóa sẽ được đưa vào các lớp tích chập để trích xuất đặc trưng từ ảnh. Các lớp tích chập sẽ thực hiện các phép tích chập trên dữ liệu input để tìm ra **các đặc trưng cụ thể của ảnh, như cạnh, góc, hoặc các đặc điểm quan trọng khác**.

Tóm lại, quá trình normalization dữ liệu input trước khi đi qua các lớp tích chập trong mạng neural giúp cải thiện hiệu suất của mô hình và đảm bảo rằng mô hình học được các đặc trưng chính xác từ dữ liệu.

Loss function

- Vì giá trị dự đoán là giá trị thực nên ta sẽ dùng **mean square error** giống như trong bài 1 khi dự đoán giá nhà.

Python code

Import

```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, Dropout, Dense, Flatten
from keras.regularizers import l2

from Self_Driving_Car.utils import INPUT_SHAPE, batch_generator
```

Xây dựng model

```
# Xây dựng model
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dropout(0.5))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
model.summary()
```

Giải thích

8. **Lambda Layer:** Lambda layer được sử dụng để thực hiện một phép biến đổi đặc biệt cho dữ liệu đầu vào. Trong trường hợp này, nó được sử dụng để chuẩn hóa dữ liệu đầu vào bằng cách chia mỗi giá trị đi 127.5 và trừ đi 1.0. Điều này giúp đưa dữ liệu về miền [0, 1] thành miền [-1, 1].
9. Conv2D Layers: Đây là các lớp tích chập 2D. Mỗi lớp Conv2D được cấu hình với một số lượng bộ lọc (filters), kích thước kernel (kernel_size), và hàm kích hoạt (activation). Các lớp này được sử dụng để trích xuất đặc trưng từ dữ liệu hình ảnh.
10. **Subsampling:** Tham số subsample=(2, 2) trong các lớp Conv2D chỉ ra rằng mỗi lần lớp tích chập được áp dụng, dữ liệu sẽ được giảm đi một nửa theo cả hai chiều (chiều dọc và chiều ngang). Điều này giúp giảm kích thước của dữ liệu và làm cho mô hình huấn luyện nhanh hơn.
11. **Dropout Layers:** Lớp Dropout được sử dụng để ngẫu nhiên "tắt" một số đơn vị neuron trong quá trình huấn luyện. Điều này giúp tránh overfitting bằng cách làm cho mô hình trở nên đơn giản hóa và dễ dàng tổng quát hóa.
12. Flatten Layer: Lớp Flatten được sử dụng để chuyển đổi dữ liệu từ dạng tensor 2D (tích chập) sang dạng vector 1D, là bước chuẩn bị cho các lớp Dense tiếp theo.
13. Dense Layers: Các lớp Dense là các lớp neuron kết nối đầy đủ, trong đó mỗi neuron trong lớp trước được kết nối với tất cả các neuron trong lớp tiếp theo. Activation function '**elu**' (**Exponential Linear Unit**) được sử dụng ở đây để tăng tính phi tuyến tính của mô hình.
14. **Summary:** Lệnh model.summary() được sử dụng để hiển thị cấu trúc của mô hình, bao gồm số lượng tham số và kích thước đầu ra của mỗi lớp.

Cấu hình và fit model (training model)

```
nb_epoch = 10
samples_per_epoch = 1000
batch_size = 32
save_best_only = True
learning_rate = 1e-4

# Checkpoint này để nói cho model lưu lại model nếu validation loss thấp nhất
checkpoint = ModelCheckpoint('models/model-{epoch:03d}.h5',
                             monitor='val_loss',
                             verbose=0,
                             save_best_only=save_best_only,
                             mode='auto')

# Dùng mean_squared_error làm loss function
model.compile(loss='mean_squared_error', optimizer=Adam(lr=learning_rate))

# Train model
H = model.fit_generator(batch_generator(data_dir, X_train, y_train, batch_size, True),
                        steps_per_epoch = samples_per_epoch,
                        epochs = nb_epoch,
                        max_q_size=1,
                        validation_data=batch_generator(data_dir, X_valid, y_valid, batch_size, False),
                        nb_val_samples=len(X_valid),
                        callbacks=[checkpoint],
```

verbose=1)

Giải thích

- **ModelCheckpoint** : Hay, Checkpoint này để nói cho model lưu lại model nếu validation loss thấp nhất , `ModelCheckpoint` là một callback trong Keras được sử dụng để lưu trọng số của mô hình sau mỗi epoch.
 1. **'models/model-{epoch:03d}.h5'**: Đây là đường dẫn mẫu để lưu trọng số của mô hình. Trong đó, `{epoch:03d}` là một định dạng chuỗi cho biết số thứ tự của epoch, với ba chữ số, ví dụ như '001', '002',... '100', '101',...
 8. **'monitor='val_loss'**: Đây là chỉ mục mà mô hình sẽ theo dõi để quyết định xem liệu trọng số mới đã được lưu lại có tốt hơn so với trọng số cũ hay không. Trong trường hợp này, mô hình sẽ kiểm tra giá trị của hàm loss trên tập validation.
 9. **'verbose=0'**: Điều này xác định cách thông báo khi mô hình lưu trọng số. Trong trường hợp này, `verbose=0` có nghĩa là không có thông báo nào được hiển thị khi trọng số được lưu lại.
 10. **'save_best_only=save_best_only'**: Tham số này xác định xem liệu chỉ những trọng số tốt nhất (đạt được qua các epoch) có nên được lưu lại hay không. Nếu `save_best_only=True` , chỉ có khi có một sự cải thiện đáng kể trong chỉ số giám sát (ở đây là 'val_loss') so với epoch trước đó mới lưu trọng số.
 11. **'mode='auto'** : Xác định cách quyết định trạng thái tốt nhất. Ở đây, 'auto' tức là sẽ tự động xác định với chỉ số giám sát là 'val_loss', khi giá trị của nó giảm đi, mô hình sẽ được coi là cải thiện.

- **optimizer=Adam(lr=learning_rate) : Có thể tùy chỉnh ở thuật toán tối ưu**

- Expand

```
from keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-8, decay=1e-6, amsgrad=False)

model.compile(loss='mean_squared_error', optimizer=optimizer)
```

1. **'learning_rate'** (float): Tỷ lệ học (learning rate) cho mạng neural.
2. **'beta_1'** (float, $0 < \beta < 1$): Hệ số giảm mômentum thứ nhất.
3. **'beta_2'** (float, $0 < \beta < 1$): Hệ số giảm mômentum thứ hai.
4. **'epsilon'** (float): Số nhỏ được thêm vào tên lượng để tránh chia cho 0.
5. **'decay'** (float): Tỷ lệ giảm học theo thời gian.
6. **'amsgrad'** (boolean): Có sử dụng phiên bản cố định của Adam hay không.

- **keras.fit() and keras.fit_generator() - GeeksforGeeks** : Toàn bộ tham số và giải thích của hàm đều có ở đây
- **H = model.fit_generator(batch_generator(data_dir, X_train, y_train, batch_size, True),**

```

steps_per_epoch = samples_per_epoch,
epochs = nb_epoch,
max_q_size=1,
validation_data=batch_generator(data_dir, X_valid, y_valid, batch_size, False),
nb_val_samples=len(X_valid),
callbacks=[checkpoint],
verbose=1)

```

9. **`batch_generator(data_dir, X_train, y_train, batch_size, True)`**: Một hàm sinh dữ liệu đầu vào (input data) và nhãn tương ứng (labels) theo từng batch từ tập huấn luyện (training set).

10. **`steps_per_epoch`**: Số lượng bước (steps) sẽ được thực hiện trong mỗi epoch. Một bước tương ứng với việc đưa một batch dữ liệu vào mạng và cập nhật trọng số.

11. **`epochs`**: Số lượng epochs (vòng lặp) sẽ được thực hiện trong quá trình huấn luyện. Mỗi epoch tương ứng với việc sử dụng toàn bộ dữ liệu huấn luyện một lần.

12. **`max_q_size`**: Số lượng batches tối đa được phép trong hàng đợi (queue). Điều này ảnh hưởng đến việc xử lý song song của các batch.

13. **`validation_data`**: Dữ liệu dùng để đánh giá hiệu suất của mô hình sau mỗi epoch.

14. **`nb_val_samples`**: Số lượng mẫu dữ liệu trong tập validation.

15. **`callbacks`**: Danh sách các callback được gọi trong quá trình huấn luyện, trong trường hợp này là **`checkpoint`** để lưu lại model có performance tốt nhất.

16. **`verbose`**: Cách hiển thị quá trình huấn luyện, có thể là 0 (im silent), 1 (hiển thị progress bar), hoặc 2 (hiển thị số liệu mỗi epoch).

Steps_per_epoch

Sau khi chia train, val, test gen thì ta được train_gen có samples là 1399 (đây là số ảnh đầu vào chuyển thành tensor thì có 1399 tensor) . Ta đặt batch_size là 8 (số mẫu dữ liệu trong mỗi batch) . Như vậy ta sẽ có $1399 // 8 = 174$ batch và mỗi batch chứa 8 mẫu dữ liệu .

Tiếp tục . Tham số Steps_per_epoch , như ta đã biết thì trong mỗi epoch toàn bộ dữ liệu sẽ được lấy ra hết và cho qua mạng nơ ron , thì **số bước** này cũng chính là **số batch** của toàn bộ dữ liệu .

Trong mỗi epoch sẽ có 174 bước và trong mỗi bước sẽ lấy ra một batch gồm 8 mẫu , như vậy qua hết 174 bước thì lấy ra được 174 batch mà mỗi batch 8 mẫu thì qua hết 174 sẽ lấy ra 1399 mẫu là toàn bộ dữ liệu

Dựa trên thông tin bạn cung cấp, giải thích `steps_per_epoch` như sau:

Trong quá trình huấn luyện mô hình học sâu, bạn thường chia dữ liệu huấn luyện thành các batch, mỗi batch có số lượng mẫu là **`batch_size`**. Trong ví dụ của bạn, **`batch_size`** được thiết lập là 8.

Đối với mỗi epoch trong quá trình huấn luyện, mỗi bước (hoặc step) đại diện cho việc đưa một batch dữ liệu (trong trường hợp của bạn, mỗi batch có 8 mẫu) vào mạng nơ-ron để tính toán gradient của hàm mất mát và cập nhật trọng số.

Số lượng bước (**`steps_per_epoch`**) cần thực hiện trong mỗi epoch được tính bằng cách chia tổng số mẫu trong tập huấn luyện cho **`batch_size`**. Trong ví dụ của bạn, tổng số mẫu trong tập huấn luyện là 1399 và **`batch_size`** là 8, vì vậy **`steps_per_epoch`** sẽ là **`1399 // 8 = 174`**.

Điều này có nghĩa là trong mỗi epoch của quá trình huấn luyện, mạng sẽ thực hiện 174 bước, mỗi bước đưa vào một batch dữ liệu kích thước 8 mẫu để huấn luyện.

Ví dụ

```
img_size = (32 , 32)
batch_size = 8
img_shape = (img_size[0] , img_size[1] , 3)

tr_gen = ImageDataGenerator()
ts_gen = ImageDataGenerator()

train_gen = tr_gen.flow_from_directory(train_dir , target_size = img_size , class_mode =
'categorical' ,
                                         color_mode = 'rgb' , shuffle = True , batch_size = batch_size)

valid_gen = ts_gen.flow_from_directory(val_dir , target_size = img_size , class_mode =
'categorical' ,
                                         color_mode = 'rgb' , shuffle = True , batch_size = batch_size)

test_gen = ts_gen.flow_from_directory(test_dir , target_size = img_size , class_mode =
'categorical' ,
                                         color_mode = 'rgb' , shuffle = False , batch_size = batch_size)
```

Ta có

```
print('train_gen.samples : ',train_gen.samples) # 1399
print('train_gen.batch_size : ',train_gen.batch_size) # 8
print('steps_per_epoch : ',train_gen.samples // train_gen.batch_size) # 174
```

Và

```
    steps_per_epoch = train_gen.samples // train_gen.batch_size,
```

Sử dụng model sau khi đã training

```
import argparse
import base64
from datetime import datetime
import os
import shutil
import numpy as np
import socketio
import eventlet
import eventlet.wsgi
from PIL import Image
from flask import Flask
```

```
from io import BytesIO

from keras.models import load_model

import utils

#initialize our server
sio = socketio.Server()
#our flask (web) app
app = Flask(__name__)
#init our model and image array as empty
model = None
prev_image_array = None

# Tốc độ tối thiểu và tối đa của xe
MAX_SPEED = 25
MIN_SPEED = 10

# Tốc độ thời điểm ban đầu
speed_limit = MAX_SPEED

@registering event handler for the server
@sio.on('telemetry')
def telemetry(sid, data):
    if data:
        # Lấy giá trị throttle hiện tại
        throttle = float(data["throttle"])
        # Góc lái hiện tại của ô tô
        steering_angle = float(data["steering_angle"])
        # Tốc độ hiện tại của ô tô
        speed = float(data["speed"])
        # Ảnh từ camera giữa
        image = Image.open(BytesIO(base64.b64decode(data["image"])))
        try:
            # Tiền xử lý ảnh, cắt, reshape
            image = np.asarray(image)
            image = utils.preprocess(image)
            image = np.array([image])
            print('*'*50)
            steering_angle = float(model.predict(image, batch_size=1))

            # Tốc độ ta đê trong khoảng từ 10 đến 25
            global speed_limit
            if speed > speed_limit:
                speed_limit = MIN_SPEED # giảm tốc độ
        except Exception as e:
            print(e)
```

```

        else:
            speed_limit = MAX_SPEED
            throttle = 1.0 - steering_angle**2 - (speed/speed_limit)**2

            print('{} {} {}'.format(steering_angle, throttle, speed))

            # Gửi lại dữ liệu về góc lái, tốc độ cho phần mềm để ô tô tự lái
            send_control(steering_angle, throttle)
        except Exception as e:
            print(e)

        # save frame
        if args.image_folder != "":
            timestamp = datetime.utcnow().strftime('%Y_%m_%d_%H_%M_%S_%f')[:-3]
            image_filename = os.path.join(args.image_folder, timestamp)
            image.save('{}.jpg'.format(image_filename))
    else:

        sio.emit('manual', data={}, skip_sid=True)

@sio.on('connect')
def connect(sid, environ):
    print("connect ", sid)
    send_control(0, 0)

def send_control(steering_angle, throttle):
    sio.emit(
        "steer",
        data={
            'steering_angle': steering_angle.__str__(),
            'throttle': throttle.__str__()
        },
        skip_sid=True)

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Remote Driving')
    parser.add_argument(
        'model',
        type=str,
        help='Path to model h5 file. Model should be on the same path.')
    parser.add_argument(

```

```
'image_folder',
type=str,
nargs='?',
default="",
help='Path to image folder. This is where the images from the run will be saved.'
)
args = parser.parse_args()

# Load model mà ta đã train được từ bước trước
model = load_model(args.model)

if args.image_folder != "":
    print("Creating image folder at {}".format(args.image_folder))
    if not os.path.exists(args.image_folder):
        os.makedirs(args.image_folder)
    else:
        shutil.rmtree(args.image_folder)
        os.makedirs(args.image_folder)
    print("RECORDING THIS RUN ...")
else:
    print("NOT RECORDING THIS RUN ...")

# wrap Flask application with engineio's middleware
app = socketio.Middleware(sio, app)

# deploy as an eventlet WSGI server
eventlet.wsgi.server(eventlet.listen(("", 4567)), app)
```

Dòng code sau chuyển từ ảnh về dữ liệu tensor để dự đoán:

```
python
image = Image.open(BytesIO(base64.b64decode(data["image"])))
```

Dòng này đọc dữ liệu ảnh từ gói tin nhận được thông qua socket, sau đó giải mã ảnh từ base64, và cuối cùng mở ảnh bằng thư viện PIL (Pillow).

Dòng tiếp theo thực hiện tiền xử lý ảnh để chuẩn bị cho việc dự đoán:

```
python
image = np.asarray(image)
image = utils.preprocess(image)
image = np.array([image])
```

Ở đây, ảnh được chuyển đổi thành mảng numpy và được tiền xử lý bằng hàm `preprocess` từ module `utils`. Cuối cùng, ảnh được đưa vào một mảng numpy có shape phù hợp cho việc dự đoán.

```
image = Image.open(BytesIO(base64.b64decode(data["image"])))
image = np.asarray(image)
image = utils.preprocess(image)
image = np.array([image])
```

Phần 5 : Deep Learning Tips

11. Transfer learning và data augmentation

Transfer learning (Học chuyển giao)

"Transfer learning" có thể được dịch ra tiếng Việt là "**học chuyển giao**." Đây là một kỹ thuật trong lĩnh vực máy học mà một mô hình đã được huấn luyện trước trên một tập dữ liệu lớn có thể được sử dụng lại cho một nhiệm vụ tương tự hoặc liên quan mà không cần phải huấn luyện lại từ đầu.

Transfer learning là một kỹ thuật trong machine learning và deep learning, trong đó kiến thức được học từ một tác vụ có sẵn được chuyển đến một tác vụ mới. Thay vì bắt đầu từ việc huấn luyện một mô hình từ đầu với dữ liệu lớn, chúng ta có thể sử dụng một mô hình đã được huấn luyện trước đó trên một tác vụ tương tự và sau đó điều chỉnh lại cho phù hợp với tác vụ mới.

[QUAN TRỌNG]

Trong quá trình transfer learning, có một số cách thức để điều chỉnh lại mô hình:

1. **Đóng băng (Freezing)** các lớp cốt lõi (Core layers): **Các lớp cốt lõi là những lớp gần với đầu vào** và được coi là phần **chứa thông tin phổ quát** về dữ liệu (ví dụ: các lớp CNN trong mô hình học sâu cho việc phân loại ảnh). Các lớp này thường được đóng băng, nghĩa là tham số của chúng không được cập nhật trong quá trình huấn luyện trên tác vụ mới.

2. **Thay đổi tham số của các lớp đóng băng:** Một số lớp có thể được thay đổi tham số, nhưng thường là với tốc độ học (learning rate) thấp hơn so với các lớp không đóng băng. Điều này giúp giữ lại các thông tin học được từ tác vụ cũ.

3. **Thêm lớp mới (Adding new layers):** Các lớp mới thường được thêm vào cuối của mô hình để phù hợp với tác vụ mới. Các lớp này thường được huấn luyện từ đầu và thường là các lớp fully connected layers (hoặc các lớp tương tự phù hợp với tác vụ).

4. **Tinh chỉnh (Fine-tuning) các lớp trên cùng:** Một khi các lớp mới đã được thêm vào, có thể mở đóng băng một số lớp ở gần đỉnh của mô hình (**gần với lớp đầu ra**) và tiếp tục huấn luyện chúng với tốc độ học thấp. Điều này giúp cho mô hình có thể học được thông tin cụ thể cho tác vụ mới mà không làm mất đi kiến thức đã học được từ tác vụ cũ.

Tóm lại, quá trình transfer learning là một quá trình điều chỉnh lại mô hình đã được huấn luyện trước đó để phù hợp với tác vụ mới bằng cách đóng băng, thêm và tinh chỉnh các lớp trong mô hình.

Thứ tự từ input đến output trong một mô hình học sâu thường là:

1. **Các lớp Convolutional (Convolutional layers):** Các lớp này thường được đặt ở phía đầu của mô hình và chịu trách nhiệm trích xuất các đặc trưng cơ bản từ dữ liệu đầu vào. Trong các mô hình dựa trên CNN, **nhiều thông tin chung** về đối tượng trong dữ liệu đầu vào. Chẳng hạn, nếu đầu vào là hình ảnh khuôn mặt, các lớp convolutional có thể học được các đặc điểm chung của khuôn mặt như cạnh, góc, hoặc các đặc trưng hình học cơ bản của khuôn mặt.

2. **Các lớp Pooling (Pooling layers):** Các lớp này thường được sử dụng sau các lớp convolutional để giảm kích thước của các đặc trưng và **giúp giảm thiểu overfitting**. Các lớp pooling không thêm **nhiều thông tin mới** mà chỉ chọn lựa thông tin quan trọng từ các đặc trưng đã được trích xuất. **Do đó, thông tin chung về đối tượng thường vẫn được giữ lại sau các lớp pooling.**

3. **Các lớp fully connected (hoặc lớp gần cuối cùng):** Các lớp này thường được sử dụng để kết nối các đặc trưng được trích xuất từ các lớp trước và tạo ra đầu ra cuối cùng. Trong các mô hình phân loại, **các lớp này thường chứa thông tin chi tiết hoặc thông tin đặc thù về các lớp đối tượng cụ thể**. Ví dụ, trong trường hợp phân loại khuôn mặt, các lớp này có thể học được các đặc điểm đặc thù của khuôn mặt người Việt Nam như hình dạng mắt, mũi, miệng, hoặc các đặc trưng về phong cách tóc và trang phục.

Tuy nhiên, cần lưu ý rằng việc phân biệt rõ ràng giữa các lớp chứa thông tin chung và thông tin chi tiết không phải lúc nào cũng đơn giản và phụ thuộc vào cả dữ liệu và kiến trúc của mô hình. Trong nhiều trường hợp, thông tin chung và thông tin chi tiết có thể được phản ánh qua các lớp của mạng.

- ⇒ *Như vậy có nghĩa là đóng băng các lớp đầu (các lớp chứa thông tin chung)*
- ⇒ *Mở ra các lớp cuối để tinh chỉnh, hoặc thêm các lớp mới vào cuối kiến trúc để thực hiện bài toán riêng , chi tiết hơn , phù hợp hơn với yêu cầu bài toán .*

Sự khác biệt giữa pre-training và freeze model trong phần transfer learning:

1. ****Pre-training**:** Khi sử dụng pre-training, bạn sẽ sử dụng một mô hình đã được đào tạo trước đó trên một tập dữ liệu lớn. Sau đó, bạn sẽ sử dụng mô hình này làm một loạt trọng số ban đầu cho mô hình của mình. Việc này giúp cho mô hình của bạn khởi đầu với các trọng số gần với tối ưu hơn và có thể giúp tăng tốc quá trình học.

2. ****Freeze Model**:** Khi bạn freeze một số lớp cuối của mô hình, bạn ngừng cập nhật các trọng số của các lớp này trong quá trình huấn luyện. Thay vào đó, bạn chỉ huấn luyện các lớp mới mà bạn thêm vào phía trên của mô hình (với các trọng số ban đầu từ pre-training hoặc được khởi tạo ngẫu nhiên). Bằng cách này, các lớp dưới của mô hình vẫn giữ nguyên các trọng số đã được học từ dữ liệu lớn trước đó, trong khi các lớp mới sẽ được huấn luyện để phù hợp với nhiệm vụ cụ thể của bạn.

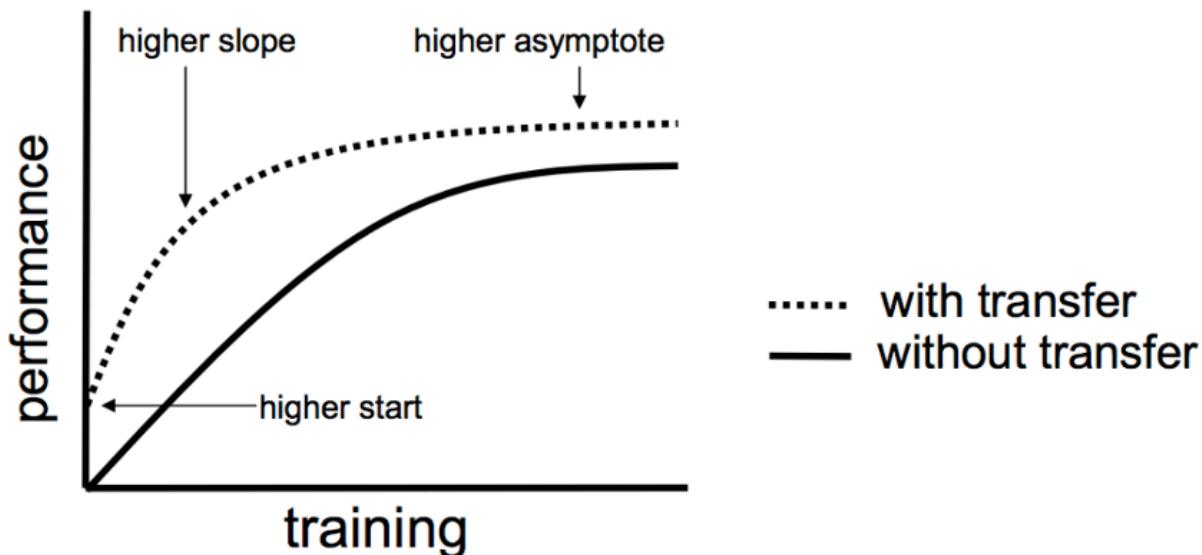
Sử dụng kỹ thuật này, bạn có thể tận dụng kiến thức học được từ các mô hình đã đào tạo sẵn để giúp cải thiện hiệu suất của mô hình của mình trên tập dữ liệu nhỏ hoặc mới.

Bạn nhớ ra là trong convolutional neural network, convolutional layer có tác dụng lấy ra các đặc trưng của ảnh, và sau hàng loạt các convolutional layer + pooling layer (ConvNet) thì model sẽ học được các đặc điểm của ảnh, trước khi được cho vào fully connected layer => ConvNet trong VGGFace2 model cũng lấy ra được các đặc điểm của mặt người (tai, mũi, tóc,...) => Ta cũng có thể áp dụng phần ConvNet của VGGFace2 model vào bài toán nhận diện mặt người nói tiếng ở Việt Nam để lấy ra các đặc điểm của mặt.

Quá trình sử dụng pre-trained model như trên gọi là transfer learning. Các pre-trained model được sử dụng thường là các bài toán được train với dữ liệu lớn ví dụ ImageNet, dữ liệu chứa 1.2 triệu ảnh với 1000 thể loại khác nhau.

Có 2 loại transfer learning:

- **Feature extractor (Trích xuất đặc trưng)** : Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ dùng linear classifier (**linear SVM, softmax classifier,..**) để phân loại ảnh. Hiểu đơn giản thì các đặc điểm ảnh (tai, mũi, tóc,...) giờ như input của bài toán linear regression hay logistic regression.
- **Fine tuning (Tinh chỉnh)**: Sau khi lấy ra các đặc điểm của ảnh bằng việc sử dụng ConvNet của pre-trained model, thì ta sẽ coi đây là input của 1 CNN mới bằng cách thêm các ConvNet và Fully Connected layer. Lý do là ConvNet của VGGFace 2 model có thể lấy ra được các thuộc tính của mặt người nói chung nhưng người Việt Nam có những đặc tính khác nên cần thêm 1 số Convnet mới để học thêm các thuộc tính của người Việt Nam.



Bài toán: Ta muốn nhận diện ảnh của 17 loài hoa, mỗi loài hoa có khoảng 80 ảnh.

Sử dụng pre-trained model là VGG 16 của ImageNet. Mô hình VGG 16 mọi người có thể xem lại bài CNN. Mô hình VGG16 của ImageNet dataset, phân loại ảnh thuộc 1000 thể loại khác nhau. Nên có thể hiểu là nó đủ tổng quát để tách ra các đặc điểm của bức ảnh, cụ thể ở đây là hoa.

- ⇒ Các mô hình nhận diện đối tượng sẽ nhận diện hết được tất cả các đối tượng, tuy nhiên độ chính xác thấp .
- ⇒ Vì thế ta cần sử dụng transfer learning , lúc này độ chính xác của nó sẽ cao lên . So với việc nhận diện một đối tượng chung chung , ta tập trung vào nhận diện một đối tượng cụ thể thì độ chính xác của nó sẽ cao lên .

Nói thêm : Dropout là một kỹ thuật regularization phổ biến được sử dụng trong quá trình huấn luyện mạng nơ-ron để giảm overfitting. Ý tưởng cơ bản của dropout là tạm thời loại bỏ một số lượng ngẫu nhiên các nơ-ron khỏi mạng nơ-ron trong quá trình huấn luyện.

Trong quá trình huấn luyện, mỗi lần forward hoặc backward pass, dropout sẽ loại bỏ một số nơ-ron ngẫu nhiên từ lớp trước và lớp sau. Cụ thể, một tỷ lệ dropout (thường là một giá trị từ 0.2 đến 0.5) sẽ được áp dụng cho mỗi lớp. Khi một nơ-ron bị loại bỏ, nó sẽ không tham gia vào quá trình forward hoặc backward pass trong lần đó.

Việc loại bỏ ngẫu nhiên các nơ-ron này giúp mạng nơ-ron trở nên ít nhạy cảm hơn với các đặc điểm cụ thể của tập huấn luyện và làm giảm khả năng overfitting. Nó tương tự như việc huấn luyện nhiều mạng con khác nhau và kết hợp chúng lại, tạo ra một mô hình tổng quát hóa tốt hơn.

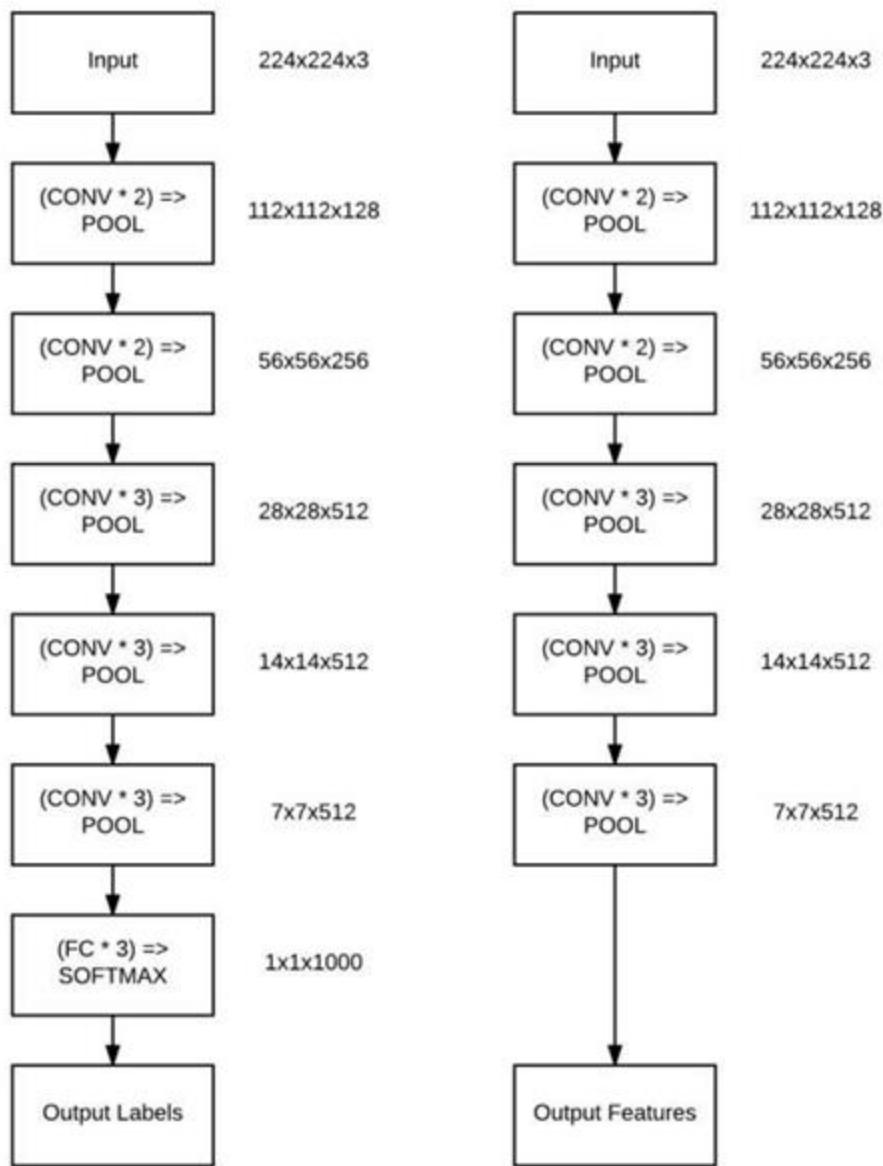
Sau quá trình huấn luyện, tất cả các nơ-ron sẽ được sử dụng trong quá trình dự đoán, không có dropout được áp dụng.

2 Dạng transfer learning

Feature extractor (Trích xuất đặc trưng)

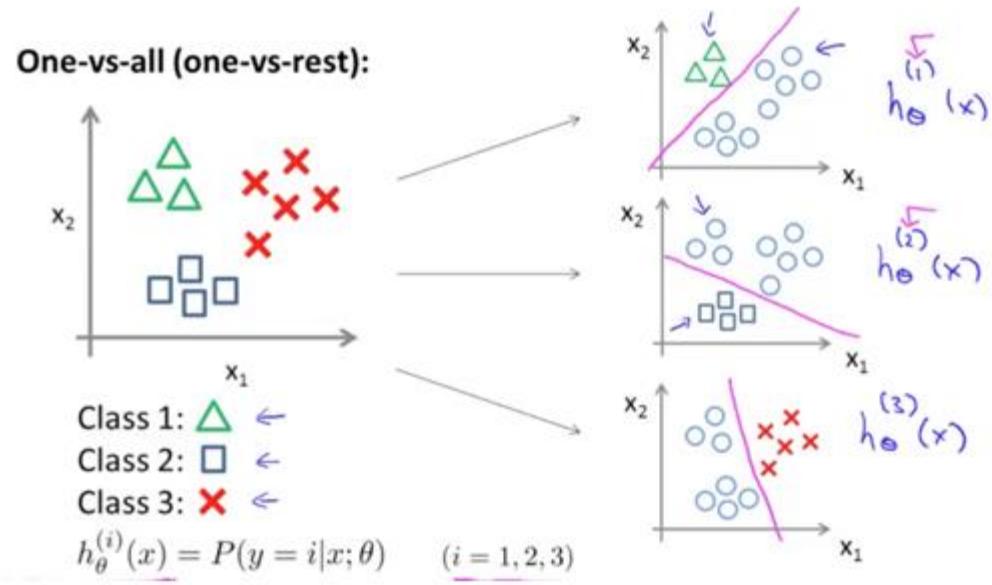
Ta chỉ giữ lại phần ConvNet (Convolution & Pooling) trong CNN và bỏ đi FCs (Fully connected) .

Sau đó dùng output của ConvNet còn lại để làm input cho Logistic Regression với nhiều output, như trong hình 11.2.



Mô hình logistic regression với nhiều output có 2 dạng:

1. Dạng thứ nhất là một neural network, không có hidden layer, hàm activation ở output layer là softmax function, loss function là hàm categorical-cross entropy, giống như bài phân loại ảnh.
2. Dạng thứ hai giống như bài logistic regression, tức là model chỉ phân loại 2 class. Mỗi lần ta sẽ phân loại 1 class với tất cả các class còn lại.



Hình 11.3: 1 vs all classification [13]

Keras – Transfer learning

```
# Thêm thư viện
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from imutils import paths
from keras.applications import VGG16
from keras.applications import imagenet_utils
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import numpy as np
import random
import os
# Lấy các đường dẫn đến ảnh.
image_path = list(paths.list_images('dataset/'))

# Đổi vị trí ngẫu nhiên các đường dẫn ảnh
random.shuffle(image_path)
# Đường dẫn ảnh sẽ là dataset/tên loài hoa/tên ảnh ví dụ dataset/Bluebell/image_0241.jpg nên
p.split(os.path.sep)[-2] sẽ lấy ra được tên loài hoa
labels = [p.split(os.path.sep)[-2] for p in image_path]

# Chuyển tên các loài hoa thành số
le = LabelEncoder()
```

```

labels = le.fit_transform(labels)
# Load model VGG 16 của ImageNet dataset, include_top=False để bỏ phần Fully connected layer ở cuối.
model = VGG16(weights='imagenet', include_top=False)
# Load ảnh và resize về đúng kích thước mà VGG 16 cần là (224,224)
list_image = []
for (j, imagePath) in enumerate(image_path):
    image = load_img(imagePath, target_size=(224, 224))
    image = img_to_array(image)

    image = np.expand_dims(image, 0)
    image = imagenet_utils.preprocess_input(image)

    list_image.append(image)

list_image = np.vstack(list_image)
# Dùng pre-trained model để lấy ra các feature của ảnh
features = model.predict(list_image)

# Giống bước flatten trong CNN, chuyển từ tensor 3 chiều sau ConvNet sang vector 1 chiều
features = features.reshape((features.shape[0], 512*7*7))
# Chia training set, test set tỉ lệ 80-20
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)
# Grid search để tìm các parameter tốt nhất cho model. C = 1/lamda, hệ số trong regularisation. Solver là kiểu optimize
# https://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html
params = {'C' : [0.1, 1.0, 10.0, 100.0]}
#model = GridSearchCV(LogisticRegression(solver='lbfgs', multi_class='multinomial'), params)
model = GridSearchCV(LogisticRegression(), params)
model.fit(X_train, y_train)
print('Best parameter for the model {}'.format(model.best_params_))
# Đánh giá model
preds = model.predict(X_test)
print(classification_report(y_test, preds))

```

Một vài điểm hay cần lưu ý :

- **In đánh giá ra dạng bảng nhanh chóng**

```
# Đánh giá model
preds = model.predict(X_test) # DỰ ĐOÁN VÀ ĐÁNH GIÁ
print(classification_report(y_test, preds))
```

| | precision | recall | f1-score | support |
|-------------|-----------|--------|----------|---------|
| 0 | 0.89 | 1.00 | 0.94 | 17 |
| 1 | 0.91 | 0.77 | 0.83 | 13 |
| 2 | 0.95 | 0.95 | 0.95 | 19 |
| 3 | 0.62 | 0.91 | 0.74 | 11 |
| 4 | 0.87 | 0.93 | 0.90 | 14 |
| 5 | 0.86 | 0.86 | 0.86 | 14 |
| 6 | 1.00 | 1.00 | 1.00 | 11 |
| 7 | 0.92 | 0.85 | 0.88 | 13 |
| 8 | 1.00 | 0.95 | 0.97 | 20 |
| 9 | 1.00 | 0.94 | 0.97 | 18 |
| 10 | 0.91 | 1.00 | 0.95 | 10 |
| 11 | 1.00 | 0.94 | 0.97 | 16 |
| 12 | 0.84 | 0.94 | 0.89 | 17 |
| 13 | 1.00 | 1.00 | 1.00 | 19 |
| 14 | 1.00 | 0.96 | 0.98 | 27 |
| 15 | 0.75 | 0.69 | 0.72 | 13 |
| 16 | 1.00 | 0.85 | 0.92 | 20 |
| avg / total | 0.93 | 0.92 | 0.92 | 272 |

- Bỏ đi Fully connected layer ở cuối

```
# Load model VGG 16 của ImageNet dataset, include_top=False để bỏ phần Fully connected layer ở cuối.
model = VGG16(weights='imagenet', include_top=False)
```

Transfer learning – Feature Extractor for Face Recognition

1. Lấy ra toàn bộ ảnh trong dataset , chuyển ảnh về ma trận 224x224x3 là kích thước đầu vào của VGG16
2. Trích xuất đặc trưng của các ảnh bằng cách cho tập ma trận của ảnh đi qua "model VGG16 của thư viện , bỏ đi lớp FC ở cuối"
3. Sau khi có được tập đặc trưng và label tương ứng , chia tập thành train , test , val
4. Tạo ra model của riêng mình và huấn luyện nó :
 - model = GridSearchCV(LogisticRegression(), params)
 - model.fit(X_train, y_train)
5. Sau khi có model huấn luyện đánh giá lại bằng tập test

Dưới đây là các bước cơ bản để triển khai phương án sử dụng vector đặc trưng và transfer learning cho việc nhận diện khuôn mặt:

1. **Chuẩn bị dữ liệu:**

- Thu thập dữ liệu về khuôn mặt và phân loại chúng thành các nhóm (ví dụ: người, không phải người).
- Chia dữ liệu thành tập huấn luyện và tập kiểm tra.

2. **Chọn mô hình pre-trained:**

- Chọn một mô hình deep learning đã được huấn luyện trước trên một tập dữ liệu lớn, thường là mô hình nhận diện khuôn mặt như VGG16, ResNet, hoặc densenet.

- Tải mô hình và loại bỏ phần fully connected layer ở cuối (nếu có).

3. **Trích xuất vector đặc trưng:**

- Sử dụng mô hình pre-trained để trích xuất vector đặc trưng từ các ảnh khuôn mặt trong tập dữ liệu.

- Các vector này sẽ được sử dụng để biểu diễn các đặc điểm quan trọng của khuôn mặt.

4. **Huấn luyện mô hình:**

- Sử dụng các vector đặc trưng đã trích xuất và nhãn tương ứng để huấn luyện một mô hình máy học như Support Vector Machine (SVM), Random Forest, hoặc Neural Network.

- Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra.

5. **Đánh giá mô hình:**

- Đánh giá hiệu suất của mô hình trên tập kiểm tra bằng cách sử dụng các phương pháp đánh giá như accuracy, precision, recall, và F1-score.

- Tinh chỉnh các siêu tham số nếu cần thiết để cải thiện hiệu suất của mô hình.

6. **Kiểm tra và triển khai:**

- Kiểm tra mô hình trên dữ liệu mới để đảm bảo rằng nó hoạt động đúng như mong đợi.

- Triển khai mô hình vào ứng dụng hoặc hệ thống cuối cùng nếu nó đạt được hiệu suất mong muốn.

Nhớ rằng, mỗi bước trong quy trình này đều có thể đòi hỏi sự tinh chỉnh và điều chỉnh để đảm bảo rằng mô hình đáp ứng được yêu cầu của bài toán nhận diện khuôn mặt cụ thể của bạn.

Khi bạn thêm một khuôn mặt mới vào hệ thống, bạn có thể làm như sau:

1. **Trích xuất vector đặc trưng:** Sử dụng mô hình đã được huấn luyện trước để trích xuất vector đặc trưng từ khuôn mặt mới này. Điều này không đòi hỏi huấn luyện lại mô hình, chỉ cần chạy khuôn mặt mới qua mô hình để lấy vector đặc trưng.

2. **So sánh với cơ sở dữ liệu:** Sau khi bạn có vector đặc trưng của khuôn mặt mới, bạn có thể so sánh nó với các vector đặc trưng của các khuôn mặt đã biết trong cơ sở dữ liệu. Bạn có thể sử dụng các phương pháp như tính khoảng cách Euclidean hoặc cosine similarity để đo độ tương đồng giữa các vector đặc trưng.

3. **Quyết định:** Dựa trên kết quả của việc so sánh, bạn có thể quyết định xem khuôn mặt mới có tương tự với bất kỳ khuôn mặt nào trong cơ sở dữ liệu hay không. Nếu tương đồng đủ lớn, bạn có thể xác định khuôn mặt mới là một người đã biết hoặc một người mới.

Tóm lại, bạn không cần phải huấn luyện lại mô hình mỗi khi thêm một khuôn mặt mới vào hệ thống. Thay vào đó, bạn chỉ cần trích xuất vector đặc trưng từ khuôn mặt mới và so sánh nó với cơ sở dữ liệu đã biết.

Khi bạn đã có một mô hình được huấn luyện cho nhận diện khuôn mặt và muốn thêm khuôn mặt mới vào, có một số hướng giải quyết phổ biến:

1. **Tái huấn luyện mô hình:**

- Thu thập dữ liệu mới cho khuôn mặt mới và kết hợp với dữ liệu đã có.
- Sử dụng dữ liệu mới này để tái huấn luyện mô hình nhận diện khuôn mặt.
- Điều này đảm bảo rằng mô hình có thể nhận diện được khuôn mặt mới.

2. **Transfer Learning:**

- Sử dụng phương pháp transfer learning để cập nhật một phần của mô hình đã được huấn luyện với dữ liệu mới.
 - Bạn có thể giữ lại các lớp phía trước của mô hình (ví dụ: các lớp Convolutional), và chỉ cập nhật các lớp cuối cùng hoặc thêm các lớp mới để phù hợp với dữ liệu mới.
 - Điều này giúp tiết kiệm thời gian và tài nguyên huấn luyện so với việc huấn luyện lại toàn bộ mô hình.

3. **Fine-tuning:**

- Tinh chỉnh (fine-tuning) một mô hình đã được huấn luyện sẵn để thích ứng với dữ liệu mới.
- Bắt đầu bằng cách sử dụng một mô hình đã được huấn luyện cho nhiệm vụ nhận diện khuôn mặt (ví dụ: mô hình có sẵn trên thị trường hoặc một mô hình mà bạn đã huấn luyện trước đó).
- Tiếp tục huấn luyện mô hình này với dữ liệu mới để cập nhật trọng số và cải thiện hiệu suất nhận diện khuôn mặt.

4. **Thêm dữ liệu vào tập kiểm tra:**

- Nếu số lượng khuôn mặt mới không quá lớn và không cần phải tái huấn luyện hoặc tinh chỉnh mô hình, bạn có thể đơn giản là thêm dữ liệu mới vào tập kiểm tra.
- Điều này giúp mô hình học được các đặc trưng của khuôn mặt mới mà không cần phải thay đổi cấu trúc hoặc trọng số của mô hình.

5. **Sử dụng kỹ thuật Ensemble:**

- Kết hợp (ensemble) mô hình đã được huấn luyện sẵn với một số mô hình mới được huấn luyện trên dữ liệu mới.
 - Kỹ thuật ensemble giúp cải thiện độ chính xác và độ tin cậy của hệ thống nhận diện khuôn mặt.
- Mỗi hướng giải quyết có ưu nhược điểm riêng, và sự lựa chọn phụ thuộc vào ngữ cảnh cụ thể của vấn đề và tài nguyên có sẵn.

LSTM - Để biểu diễn 10 đặc trưng của 10 ảnh thành một đặc trưng duy nhất, bạn có thể sử dụng một trong những phương pháp sau:

1. **Trung bình hóa (Mean pooling)**: Bạn có thể tính trung bình của các đặc trưng của 10 ảnh. Điều này có thể được thực hiện bằng cách tính trung bình của từng chiều của các vector đặc trưng.
2. **Tích vô hướng (Vector concatenation)**: Bạn có thể ghép nối các vector đặc trưng của 10 ảnh thành một vector đặc trưng lớn hơn. Điều này tạo ra một vector đặc trưng dài hơn, kết hợp thông tin từ tất cả các ảnh.
3. **Max pooling**: Thay vì lấy trung bình, bạn có thể chọn giá trị lớn nhất từ mỗi chiều của các vector đặc trưng của 10 ảnh. Điều này giúp giữ lại các thông tin quan trọng nhất từ mỗi vector.

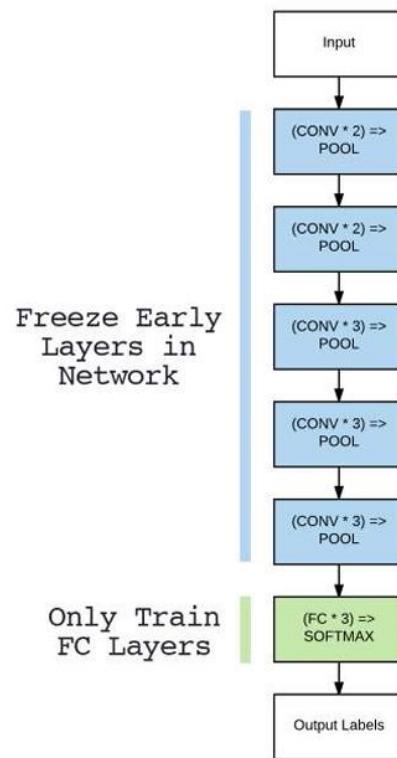
4. **PCA (Principal Component Analysis)**: Bạn có thể sử dụng PCA để giảm chiều dữ liệu từ các vector đặc trưng của 10 ảnh xuống thành một vector đặc trưng duy nhất có kích thước mong muốn.

5. **LSTM (Long Short-Term Memory)**: Đôi với dữ liệu tuần tự như các vector đặc trưng của các ảnh, bạn có thể sử dụng mạng LSTM để học mối quan hệ giữa chúng và tạo ra một vector đặc trưng duy nhất.

Tùy thuộc vào loại dữ liệu và mô hình mà bạn sử dụng, một phương pháp có thể hoạt động tốt hơn so với các phương pháp khác. Đôi khi, thử nghiệm và so sánh hiệu suất của các phương pháp khác nhau trên tập dữ liệu thử nghiệm có thể giúp bạn chọn ra phương pháp tốt nhất cho bài toán của mình.

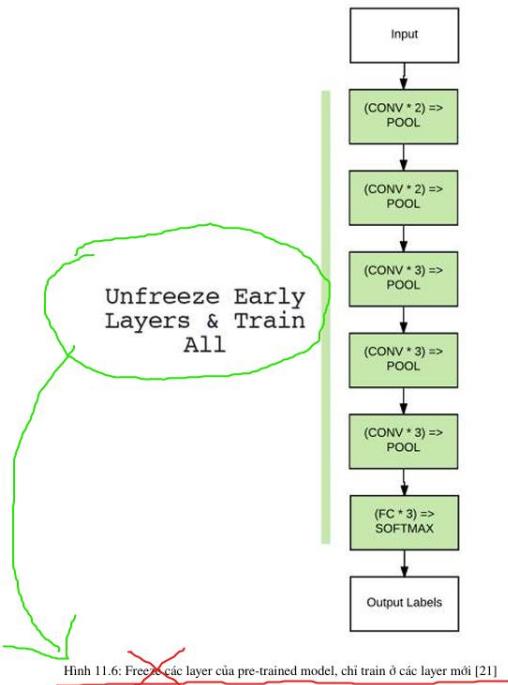
Fine tuning [Quan trọng – Đọc kỹ]

- Ta chỉ giữ lại phần ConvNet trong CNN và bỏ đi FCs. Sau đó thêm các Fully Connected layer mới vào output của ConvNet như trong hình 11.4.
- Khi train model ta chia làm 2 giai đoạn
 - Giai đoạn 1:** Vì các Fully Connected layer ta mới thêm vào có các hệ số được khởi tạo ngẫu nhiên tuy nhiên các layer trong ConvNet của pre-trained model đã được train với ImageNet dataset nên ta sẽ không train (đóng băng/freeze) trên các layer trong ConvNet của model VGG16. Sau khoảng 20-30 epoch thì các hệ số ở các layer mới đã được học từ dữ liệu thì ta chuyển sang giai đoạn 2.



Hình 11.5: Freeze các layer của pre-trained model, chỉ train ở các layer mới [21]

Giai đoạn 2: Ta sẽ unfreeze các layer trên ConvNet của pre-trained model và train trên các layer của ConvNet của pre-trained model và các layer mới. Bạn có thể unfreeze tất cả các layer trong ConvNet của VGG16 hoặc chỉ unfreeze một vài layer cuối tùy vào thời gian và GPU bạn có.



Hình 11.6: Freeze các layer của pre-trained model, chỉ train ở các layer mới [21]

Keras – Fine tuning

File :

[C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\Chapter5_VGG16_fine-tune.ipynb](#)

File :

[C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter5_Deep_Learning_Tips\Chapter5_VGG16_fine-tune2_Good.ipynb](#)

Cách bước tiến hành

1. Chỉ lấy các lớp ConvNet của Model VGG16 bằng cách bỏ đi các lớp FC
2. Tự xây dựng FC của mình bằng cách lấy ra lớp cuối cùng của các lớp Conv của Model VGG16 sau khi bỏ FC, thêm các lớp mình muốn vào để tạo thành FC hoàn chỉnh
3. Nối Conv của VGG16 và FC của mình tự xây lại thành 1 model hoàn chỉnh myModel
4. Đóng băng các lớp Conv của VGG16 chỉ train các lớp FC của mình
5. Mở đóng băng hoặc Mở đóng băng một phần các lớp ConvNet của VGG16 và train lại myModel(các lớp đã mở đóng băng + FC)

Mã nguồn

```

# pre_train model : Là model được train sẵn trước đó , chính là VGG16,..
# Load model VGG 16 của ImageNet dataset, include_top=False để bỏ phần Fully connected layer ở cuối.
myConvNetModelVGG16 = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

#### Lấy lớp cuối (xem như lớp output) của "Model VGG16 sau khi bỏ đi FC" rồi thêm các lớp của mình vào
# Xây thêm các layer

```

```

# Lấy output của ConvNet trong VGG16
myFCHead = myConvNetModelVGG16.output # (None, 7, 7, 512)
# Flatten trước khi dùng FCs
myFCHead = Flatten(name='flatten')(myFCHead) # 7x7x512 = 25088
# Thêm FC
myFCHead = Dense(256, activation='relu')(myFCHead) # 256
myFCHead = Dropout(0.5)(myFCHead) # 256
# Output layer với softmax activation
myFCHead = Dense(len(classes), activation='softmax')(myFCHead) # len(classes) = 17 số nhãn cần nhận diện

# Xây dựng model bằng việc nối ConvNet của VGG16 và fcHead
myModel = Model(inputs=myConvNetModelVGG16.input, outputs=myFCHead)
# freeze VGG model
for layer in myConvNetModelVGG16.layers:
    layer.trainable = False

# thuật toán tối ưu
optimizer = RMSprop(0.001)

myModel.compile(optimizer, 'categorical_crossentropy', metrics=['accuracy'])
numOfEpoch = 10
H = myModel.fit(
    train_gen,
    steps_per_epoch = len(train_gen),
    validation_data = valid_gen,
    validation_steps = len(valid_gen),
    epochs=numOfEpoch,
)
# unfreeze some last CNN layer:
for layer in myConvNetModelVGG16.layers[15:]:
    layer.trainable = True

numOfEpoch = 20
opt = SGD(0.001)
myModel.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
H = myModel.fit(
    train_gen,
    steps_per_epoch = len(train_gen),
    validation_data = valid_gen,
    validation_steps = len(valid_gen),
    epochs=numOfEpoch,
)

```

Giải thích

1. Tham chiếu trong python

```
# Ở đây ta lấy ra các ConvNet của VGG16 sau đó cho vào myModel
myConvNetModelVGG16 = VGG16(weights='imagenet', include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
myModel = Model(inputs=myConvNetModelVGG16.input, outputs=myFCHead)

# Do tính chất tham chiếu , nên ta chỉ cần đóng băng myConvNetModelVGG16 thì myModel cũng tham
chiếu và thay đổi trên myModel
for layer in myConvNetModelVGG16.layers:
    layer.trainable = False

opt = RMSprop(0.001)
myModel.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
numOfEpoch = 5
H = myModel.fit(...)
```

2. Nói thêm về kiến trúc

a. Kiến trúc gốc của VGG16

```
model_vgg16 = VGG16(weights='imagenet')
model_vgg16.summary()
```

| Layer (type) | Output Shape | Param # |
|-----------------------------|-----------------------|-----------|
| input_layer_20 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1,792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36,928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73,856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147,584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295,168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1,180,160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |

| | | |
|----------------------------|-------------------|-------------|
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| flatten (Flatten) | (None, 25088) | 0 |
| fc1 (Dense) | (None, 4096) | 102,764,544 |
| fc2 (Dense) | (None, 4096) | 16,781,312 |
| predictions (Dense) | (None, 1000) | 4,097,000 |

- ⇒ Kiến trúc gốc là có lớp Fullyconected ở cuối
 b. Kiến trúc sau khi bỏ đi Fullyconected ở cuối

```
myConvNetModelVGG16.summary()
```

| Layer (type) | Output Shape | Param # |
|-----------------------------|-----------------------|-----------|
| input_layer_19 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1,792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36,928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73,856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147,584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295,168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1,180,160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |

- ⇒ Ta sẽ lấy lớp cuối để tự xây dựng Fullyconected cho mình

| | | |
|----------------------------|-------------------|---|
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
|----------------------------|-------------------|---|

- c. Kiến trúc của model sau khi gộp ConvNet của VGG16 và FC của mình tự xây

| Layer (type) | Output shape | Params # |
|-----------------------------|-----------------------|-----------|
| input_layer_19 (InputLayer) | (None, 224, 224, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1,792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36,928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73,856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147,584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295,168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590,080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1,180,160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2,359,808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2,359,808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |

| | | |
|----------------------|---------------|-----------|
| flatten (Flatten) | (None, 25088) | 0 |
| dense_28 (Dense) | (None, 256) | 6,422,784 |
| dropout_14 (Dropout) | (None, 256) | 0 |
| dense_29 (Dense) | (None, 17) | 4,369 |

⇒ Với phương pháp này Fine-Tuning cho ra được model có độ chính xác cao hơn .

Full Source code

```
# root_dir = '/content/drive/MyDrive/5_AIADVANCE/GK_AI_Advance/'
root_dir = 'dataset_new'

# Đường dẫn tới thư mục con 'train', 'val' và 'test'
train_dir = os.path.join(root_dir, 'train')
val_dir = os.path.join(root_dir, 'val')
test_dir = os.path.join(root_dir, 'test')
img_size = (224, 224) # resize về đúng kích thước mà VGG 16 cần là (224,224)
batch_size = 8
img_shape = (img_size[0], img_size[1], 3)

# làm giàu dữ liệu data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1./255) # Chỉ rescale cho dữ liệu kiểm tra
# làm giàu dữ liệu data augmentation

train_gen = train_datagen.flow_from_directory(train_dir, target_size = img_size, class_mode = 'categorical',
                                              color_mode = 'rgb', shuffle = True, batch_size = batch_size)

valid_gen = test_datagen.flow_from_directory(val_dir, target_size = img_size, class_mode = 'categorical',
                                              color_mode = 'rgb', shuffle = True, batch_size = batch_size)

test_gen = test_datagen.flow_from_directory(test_dir, target_size = img_size, class_mode = 'categorical',
                                              color_mode = 'rgb', shuffle = False, batch_size = batch_size)

g_dict = train_gen.class_indices
classes = list(g_dict.keys())
images, labels = next(train_gen)
num_samples = len(images)
```

```

plt.figure(figsize=(20, 20))

for i in range(min(16, num_samples)):
    plt.subplot(4, 4, i + 1)
    image = images[i] # Không cần chia lại cho 255
    plt.imshow(image)
    class_index = np.argmax(labels[i]) # Lấy index của nhãn cao nhất
    class_name = classes[class_index]
    plt.title(class_name, color='blue', fontsize=12)
    plt.axis('off')
plt.show()

# pre_train model : Là model được train sẵn trước đó , chính là VGG16,..
# Load model VGG 16 của ImageNet dataset, include_top=False để bỏ phần Fully connected layer ở cuối.
myConvNetModelVGG16 = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(224, 224, 3)))

#### Lấy lớp cuối (xem như lớp output) của "Model VGG16 sau khi bỏ đi FC" rồi thêm các lớp của mình vào
# Xây thêm các layer
# Lấy output của ConvNet trong VGG16
myFCHead = myConvNetModelVGG16.output # (None, 7, 7, 512)
# Flatten trước khi dùng FCs
myFCHead = Flatten(name='flatten')(myFCHead) # 7x7x512 = 25088
# Thêm FC
myFCHead = Dense(256, activation='relu')(myFCHead) # 256
myFCHead = Dropout(0.5)(myFCHead) # 256
# Output layer với softmax activation
myFCHead = Dense(len(classes), activation='softmax')(myFCHead) # len(classes) = 17 số nhãn cần nhận diện

# Xây dựng model bằng việc nối ConvNet của VGG16 và fcHead
myModel = Model(inputs=myConvNetModelVGG16.input, outputs=myFCHead)
model_vgg16 = VGG16(weights='imagenet')
model_vgg16.summary()
myConvNetModelVGG16.summary()
myModel.summary()
# freeze VGG model
for layer in myConvNetModelVGG16.layers:
    layer.trainable = False

# thuật toán tối ưu
optimizer = RMSprop(0.001)
# optimizer = tf.keras.optimizers.Adam()

```

```

numOfEpoch = 10
H = myModel.fit(
    train_gen,
    steps_per_epoch = len(train_gen),
    validation_data = valid_gen,
    validation_steps = len(valid_gen),
    epochs=numOfEpoch,
)
# unfreeze some last CNN layer:
for layer in myConvNetModelVGG16.layers[15:]:
    layer.trainable = True

save_best_only = True
checkpoint = ModelCheckpoint('models/myVGG16_model-{epoch:03d}.keras',
                            monitor='val_loss',
                            verbose=1,
                            save_best_only=save_best_only,
                            mode='auto')

myModel.compile(optimizer, 'categorical_crossentropy', metrics=['accuracy']) # ta có thể ghi optimizer =
optimizer, hoặc optimizer là nó tự hiểu

```

```

numOfEpoch = 20
opt = SGD(0.001)
myModel.compile(opt, 'categorical_crossentropy', metrics=['accuracy'])
H = myModel.fit(
    train_gen,
    steps_per_epoch = len(train_gen),
    validation_data = valid_gen,
    validation_steps = len(valid_gen),
    epochs=numOfEpoch,
    # callbacks=[checkpoint]
)
myModel.save('Model_VGG16_fine_tune.h5')
train_acc = H.history['accuracy']
train_loss = H.history['loss']

val_acc = H.history['val_accuracy']
val_loss = H.history['val_loss']

index_loss = np.argmin(val_loss)
index_acc = np.argmax(val_acc)

val_lowest = val_loss[index_loss]
val_highest = val_acc[index_acc]

```

```

Epochs = [i+1 for i in range(len(train_acc))]

loss_label = f'Best Epoch = {str(index_loss + 1)}'
acc_label = f'Best Epoch = {str(index_acc + 1)}'

plt.figure(figsize= (20,8))
plt.style.use('fivethirtyeight')

plt.subplot(1,2,1)
plt.plot(Epochs , train_loss , 'r', label = 'Training Loss')
plt.plot(Epochs , val_loss , 'g' , label = 'Validation Loss')
plt.scatter(index_loss +1 , val_lowest , s = 150 , c = 'blue' , label = loss_label)
plt.title('Training vs Validation (loss)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1,2,2)
plt.plot(Epochs , train_acc , 'r', label= 'Training Accuracy')
plt.plot(Epochs , val_acc , 'g' , label = 'Validation Accuracy')
plt.scatter(index_acc + 1 , val_highest , s= 150 , c = 'blue' , label= acc_label)
plt.title('Training vs Validation (Accuracy)')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout
plt.show()
from keras.models import load_model
Model_VGG16_fine_tune = load_model('Model_VGG16_fine_tune.h5')

train_score = Model_VGG16_fine_tune.evaluate(train_gen , steps= 32 , verbose = 1)
val_score = Model_VGG16_fine_tune.evaluate(valid_gen , steps = 32 , verbose = 1)
test_score = Model_VGG16_fine_tune.evaluate(test_gen , steps = 32 , verbose = 1)

print(f'Train loss = {train_score[0]}')
print(f'Train Accuracy = {train_score[1]}')
print(f'Validation loss = {val_score[0]}')
print(f'Validation Accuracy = {val_score[1]}')
print(f'Test loss = {test_score[0]}')
print(f'Test Accuracy = {test_score[1]}')
from sklearn.metrics import classification_report, confusion_matrix

# Dự đoán

```

```

preds = Model_VGG16_fine_tune.predict(test_gen)
preds = np.argmax(preds, axis=1)

# Lấy nhãn thật
true_labels = test_gen.classes

# Tính các chỉ số
report = classification_report(true_labels, preds, output_dict=True)

# In chỉ số f1, precision, recall
print("F1:", report['weighted avg']['f1-score'])
print("Precision:", report['weighted avg']['precision'])
print("Recall:", report['weighted avg']['recall'])
print(classification_report(true_labels, preds))
# Lấy 1 batch dữ liệu từ generator
batch = next(test_gen)

# In hình ảnh đầu tiên và nhãn tương ứng
img = batch[0][0]
label_index = np.argmax(batch[1][0]) # Lấy index của nhãn cao nhất
label_name = classes[label_index]
plt.imshow(img)
plt.title(f"Nhãn thực: {label_name}") # Hiển thị nhãn thực trên tiêu đề
plt.axis('off')
plt.show()

# Chuyển đổi thành 4D tensor
img = np.expand_dims(img, axis=0)

# Dự đoán
pred = Model_VGG16_fine_tune.predict(img)
print('Dự đoán:', classes[np.argmax(pred)])
# Tính confusion matrix
conf_mat = confusion_matrix(true_labels, preds)
fig, ax = plt.subplots(figsize=(20,20))
ax.matshow(conf_mat, cmap=plt.cm.Blues, alpha=0.3)

# Hiển thị các nhãn trên trục Ox và Oy
ax.set_xticks(np.arange(len(classes)))
ax.set_yticks(np.arange(len(classes)))
ax.set_xticklabels(classes)
ax.set_yticklabels(classes)

# Hiển thị giá trị của ma trận nhầm lẫn trong từng ô
for i in range(conf_mat.shape[0]):

```

```

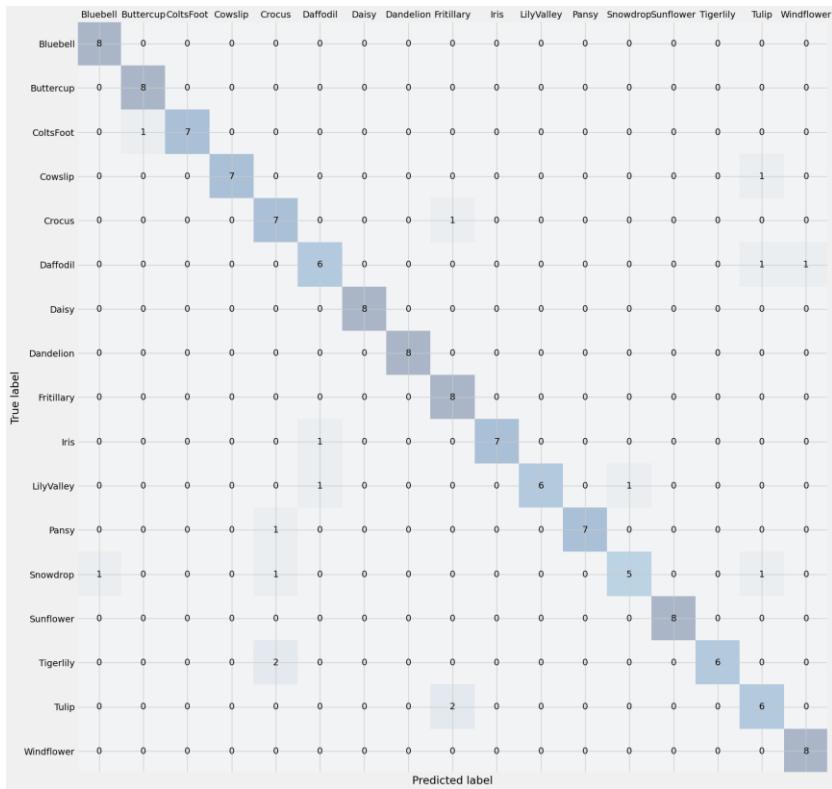
for j in range(conf_mat.shape[1]):
    ax.text(x=j, y=i, s=conf_mat[i,j], va='center', ha='center')

# Đặt tiêu đề và nhãn cho trục Ox và Oy
plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.show()

```

Ma trận nhầm lẫn



Khi nào nên dùng transfer learning

Có 2 yếu tố quan trọng nhất để dùng transfer learning đó là kích thước của dữ liệu bạn có và sự tương đồng của dữ liệu giữa mô hình bạn cần train và pre-trained model.

- Dữ liệu bạn có nhỏ và tương tự với dữ liệu ở pre-trained model. Vì dữ liệu nhỏ nên nếu dùng fine-tuning thì model sẽ bị overfitting. Hơn nữa là dữ liệu tương tự nhau nên là ConvNet của pre-trained model cũng lấy ra các đặc điểm ở dữ liệu của chúng ta. Do đó nên dùng feature extractor.
- Dữ liệu bạn có lớn và tương tự với dữ liệu ở pre-trained model. Giờ có nhiều dữ liệu ta không sợ overfitting do đó nên dùng fine-tuning.
- Dữ liệu bạn có nhỏ nhưng khác với dữ liệu ở pre-trained model. Vì dữ liệu nhỏ nên ta nên dùng feature extractor để tránh overfitting. Tuy nhiên do dữ liệu ta có và dữ liệu ở pre-trained model khác nhau, nên không nên dùng feature extractor với toàn bộ ConvNet của pre-trained model mà chỉ dùng các layer đầu. Lý do là vì các layer ở phía trước sẽ học các đặc điểm chung chung hơn (cạnh, góc,...), còn các layer phía sau trong ConvNet sẽ học các đặc điểm cụ thể hơn trong dataset (ví dụ mắt, mũi,...).
- Dữ liệu bạn có lớn và khác với dữ liệu ở pre-trained model. Ta có thể train model từ đầu, tuy nhiên sẽ tốt hơn nếu ta khởi tạo các giá trị weight của model với giá trị của pre-trained model và sau đó train bình thường.

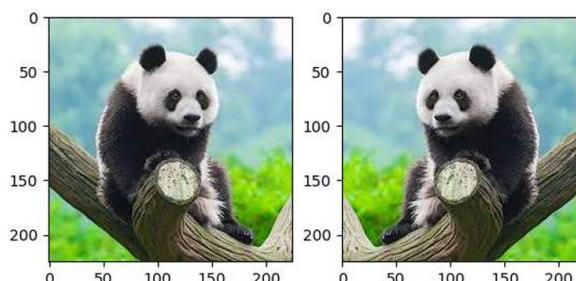
Lưu ý

- Vì pre-trained model đã được train với kích thước ảnh cố định, nên khi dùng pre-trained model ta cần resize lại ảnh có kích thước bằng kích thước mà ConvNet của pre-trained model yêu cầu.
- Hệ số learning rate của ConvNet của pre-trained model nên được đặt với giá trị nhỏ vì nó đã được học ở pre-trained model nên ít cần cập nhật hơn so với các layer mới thêm.

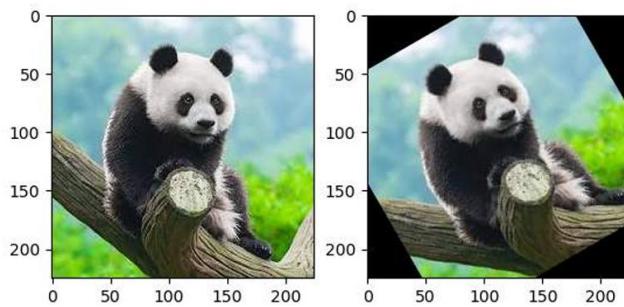
Data augmentation

Ngoài transfer learning, có một kỹ thuật nữa giải quyết vấn đề có ít dữ liệu cho việc training model, đó là data augmentation. Augmentation là kỹ thuật tạo ra dữ liệu training từ dữ liệu mà ta đang có. Cùng xem một số kỹ thuật augmentation phổ biến với ảnh nhé

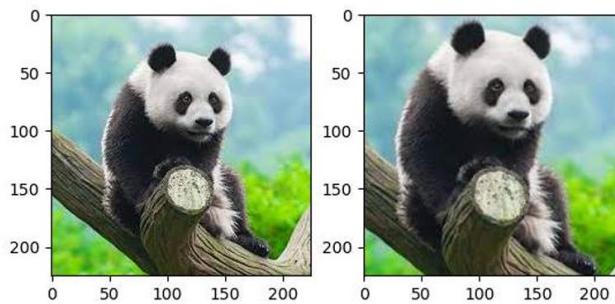
1. Flip: Lật ngược ảnh theo chiều dọc hoặc chiều ngang



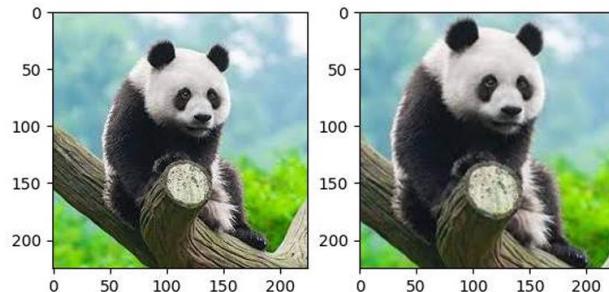
2. Rotation: Quay ảnh theo nhiều góc khác nhau



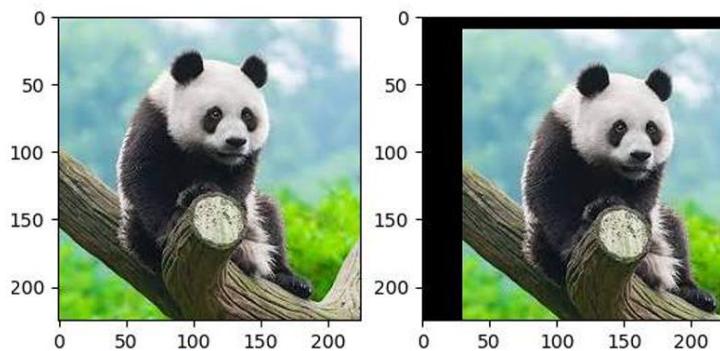
3. Scale: Phóng to hoặc thu nhỏ ảnh



4. Crop: Cắt một vùng ảnh sau đó resize vùng ảnh đáy về kích thước ảnh ban đầu



5. Translation: dịch chuyển ảnh theo chiều x, y.



Tuy nhiên khi rotate hoặc translation thì ảnh bị những khoảng đen mà thường ảnh thực tế không có các khoảng đen đây nên có một số cách để xử lý như: lấy giá trị từ cạnh của ảnh mới để cho các pixel bị đen, gán các giá trị đen bằng giá trị của ảnh đối xứng qua cạnh,...

Nên áp dụng kiểu augmentation nào thì tùy thuộc vào ngữ nghĩa của ảnh trong bài toán bạn đang giải quyết.

Generative Adversarial Networks (GANs) là một loại mô hình deep learning được đề xuất bởi Ian Goodfellow và các đồng nghiệp vào năm 2014. Ý tưởng cơ bản của GANs là huấn luyện hai mạng neuron, được gọi là Generator và Discriminator, cùng một lúc thông qua việc cạnh tranh với nhau.

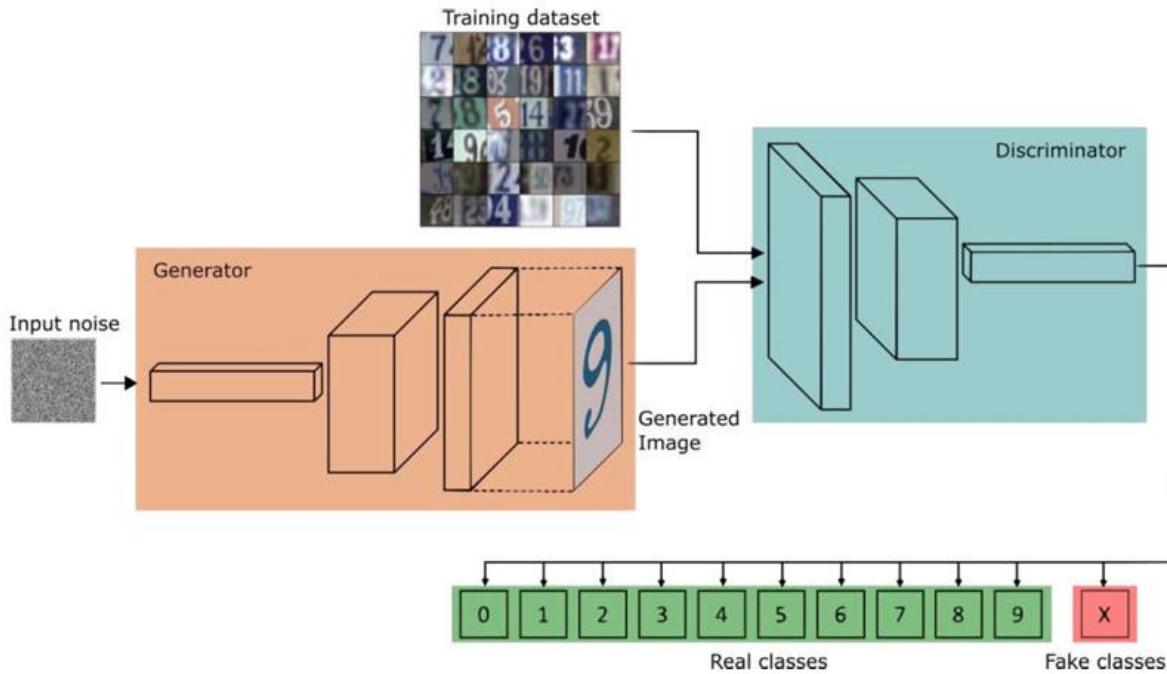
Tham khảo : <https://towardsdatascience.com/semi-supervised-learning-with-gans-9f3cb128c5e>

1. **Generator:** Mạng này nhận một vector ngẫu nhiên làm đầu vào và cố gắng sinh ra dữ liệu mới, thường là hình ảnh, âm nhạc hoặc văn bản, từ các vector ngẫu nhiên này.
2. **Discriminator:** Mạng này nhận dữ liệu từ hai nguồn: (a) dữ liệu thực tế từ tập huấn luyện và (b) dữ liệu được tạo ra bởi Generator. Nhiệm vụ của Discriminator là phân biệt giữa dữ liệu thực và dữ liệu giả mạo.

Trong quá trình huấn luyện, Generator cố gắng tạo ra dữ liệu giả mạo sao cho Discriminator không thể phân biệt được giữa dữ liệu thực và dữ liệu giả. Trong khi đó, Discriminator cố gắng phân biệt chính xác giữa dữ liệu thực và dữ liệu giả. Quá trình này tạo ra một mạng Generator ngày càng tinh vi trong việc tạo ra dữ liệu mới, cũng như một mạng Discriminator ngày càng khó phân biệt giữa dữ liệu thực và giả mạo.

GANs được sử dụng rộng rãi trong nhiều ứng dụng như tạo ra hình ảnh giả mạo, tạo ra âm nhạc mới, tạo ra văn bản tự động, và nhiều ứng dụng khác trong lĩnh vực tạo nội dung sáng tạo.

*** Ngoài ra người ta còn dùng GAN cho semi-supervised learning



Hình 11.12: Ứng dụng GAN cho semi-supervised learning [24]

Ý tưởng là generator giống như một nguồn cung cấp ảnh mới, discriminator giờ có 2 việc:

- nhận diện ảnh fake và ảnh thật
- phân loại ảnh

Tài liệu cho ai muốn đọc thêm ở [đây](#).

12. Các kỹ thuật cơ bản trong deep learning (Xem chi tiết các kỹ thuật ở sách , dưới đây là tóm tắt)

Vectorization

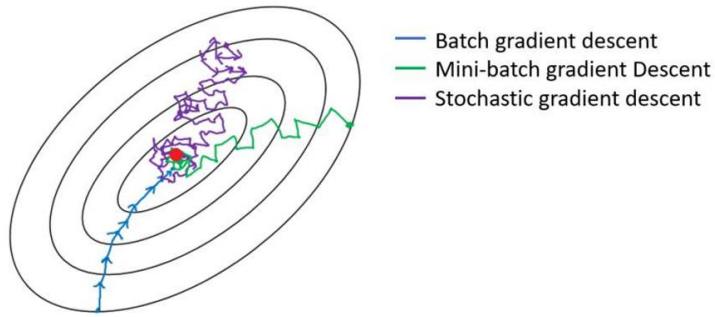
⇒ Tóm lại biểu diễn bài toán dưới dạng vector sẽ dễ tính toán hơn .

Mini-batch gradient descent

Mini-batch gradient descent

Dựa vào số lượng dữ liệu cho mỗi lần thực hiện bước 2 trong gradient descent là người ta chia ra làm 3 loại:

- Batch gradient descent: Dùng tất cả dữ liệu trong training set cho mỗi lần thực hiện bước tính đạo hàm. 12.2 Mini-batch gradient descent 179
- Mini-batch gradient descent: Dùng một phần dữ liệu trong training set cho mỗi lần thực hiện bước tính đạo hàm.
- Stochastic gradient descent: Chỉ dùng một dữ liệu trong training set cho mỗi lần thực hiện bước tính đạo hàm.



Hình 12.2: Cập nhật loss function đến minimum point của các thuật toán [13]

⇒ Batch gradient descent thường được dùng khi số lượng dữ liệu trong training set nhỏ hơn 2000. Với lượng dữ liệu lớn thì mini-batch gradient descent được sử dụng.

Các thông số trong mini-batch gradient descent

7. Thực hiện train model với data

```
H = model.fit(X_train, Y_train, validation_data=(X_val, Y_val),
               batch_size=32, epochs=10, verbose=1)
```

X_train, Y_train là dữ liệu và label cho training set. Tương tự X_val, Y_val là dữ liệu cho validation set.

batch_size: Là size trong mini-batch gradient descent, nghĩa là dùng bao nhiêu dữ liệu cho mỗi lần tính và cập nhật hệ số.

steps_per_epoch: Là bao nhiêu lần thực hiện bước 2 trong gradient descent trong mỗi epoch. **Mặc định** sẽ là số lượng dữ liệu chia cho batch_size. Hiểu đơn giản là mỗi epoch sẽ dùng hết các dữ liệu để tính gradient descent.

epochs: số lượng epoch thực hiện trong quá trình training.

Vậy thực sự số lần thực hiện bước 2 trong gradient descent trong quá trình training là: epochs * steps_per_epoch.

Lời khuyên:

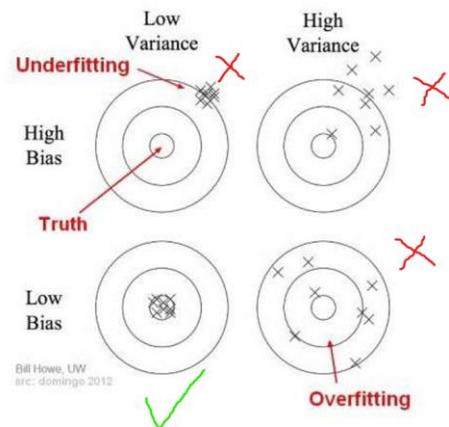
- Batch_size nên được chọn là số mũ của 2 ví dụ 16, 32, 64, 128 để CPU/GPU tính toán tốt hơn. Giá trị mặc định là 32.
- Nên vẽ đồ thị loss/epoch để chọn batch_size phù hợp.

Bias và variance

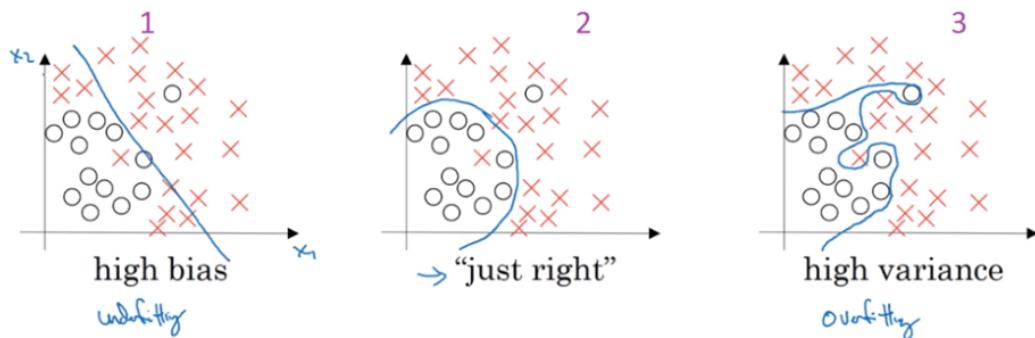
Bias, variance

Bias: nghĩa là độ lệch, biểu thị sự chênh lệch giữa giá trị trung bình mà mô hình dự đoán và giá trị thực tế của dữ liệu.

Variance: nghĩa là phương sai, biểu thị độ phân tán của các giá trị mà mô hình dự đoán so với giá trị thực tế.

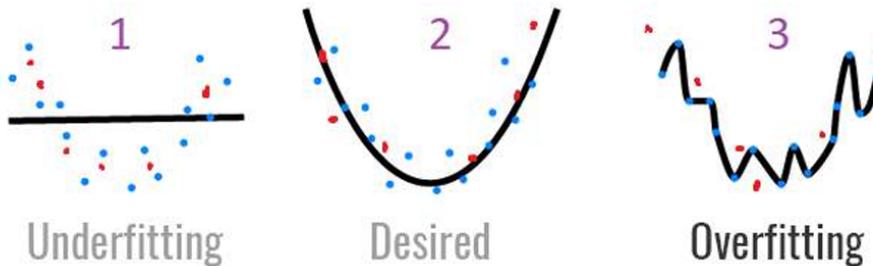


Giá trị thật dữ liệu (ground truth) ở giữa tâm các đường tròn. Các dấu X là các giá trị dự đoán. Ta thấy nếu high bias thì giá trị dự đoán rất xa tâm. Tuy nhiên nếu high variance thì các giá trị dự đoán phân tán rộng dẫn đến việc ra giá trị thực tế. => **Ta mong muốn low bias và low variance.**



Đây là bài toán logistic regression, cần tìm đường phân chia dữ liệu.

- Ở hình 1, đường phân chia có khá nhiều điểm bị lỗi => sự chênh lệch giữa mô hình dự đoán và giá trị thực tế của dữ liệu cao => **high bias**, hay còn được gọi là **underfitting**, ý hiểu là mô hình hiện tại đơn giản hơn và chưa mô tả được mô hình của dữ liệu thực tế.
 - Ở hình 2, đường phân chia vẫn có lỗi nhưng ở mức chấp nhận được và nó có thể mô tả dữ liệu => **low bias, low variance**.
 - Ở hình 3, đường phân chia có thể dự đoán đúng tất cả các điểm trong training set nhưng vì nó không tổng quát hóa mô hình dữ liệu thực sự nên khi áp dụng dự đoán vào validation set thì sẽ có rất nhiều lỗi => **high variance** hay còn được gọi là **overfitting**, ý hiểu là mô hình hiện tại thực hiện tốt với dữ liệu trong training set nhưng dự đoán không tốt với validation set.
- Thực ra khái niệm high bias và high variance khá trùng tượng và nhiều lúc dùng nhầm lẫn giữa thống kê và machine learning. Nên khái niệm hay được dùng hơn là underfitting và overfitting.



Ví dụ khi luyện thi đại học, nếu bạn chỉ luyện khoảng 1-2 đề trước khi thi thì bạn sẽ bị underfitting vì bạn chưa hiểu hết cấu trúc, nội dung của đề thi. Tuy nhiên nếu bạn chỉ luyện kĩ 50 đề thầy cô giáo bạn soạn và đưa cho thì khả năng bạn sẽ bị overfitting với các đề mà thầy cô giáo các bạn soạn mà khi thi đại học có thể điểm số của các bạn vẫn tệ.

Bias, variance tradeoff

Nếu model quá đơn giản thì ta sẽ bị high bias và low variance. Tuy nhiên nếu model quá phức tạp thì sẽ bị high variance và low bias. Đây là bias, variance tradeoff. Do đó để train được model tốt ta cần cân bằng giữa bias và variance.

Fix underfitting và overfitting

Có 2 thông số thường được sử dụng để đánh giá bias and variance của mô hình là training set error và validation set error. Ví dụ error (1-accuracy) trong logistic regression.

| | | | | |
|-----------------|---------------|-----------|-------------------------|-----------------------|
| Train set error | 1% | 15% | 15% | 0.5% |
| Val set error | 11% | 16% | 30% | 1% |
| | High variance | High bias | High bias High variance | Low bias Low variance |

Ta mong muốn model là low bias và low variance. Cùng xem một số cách để giải quyết vấn đề high bias hoặc high variance nhé.

Giải quyết high bias (underfitting): Ta cần tăng độ phức tạp của model

- Tăng số lượng hidden layer và số node trong mỗi hidden layer.
- Dùng nhiều epoch hơn để train model.

Giải quyết high variance (overfitting):

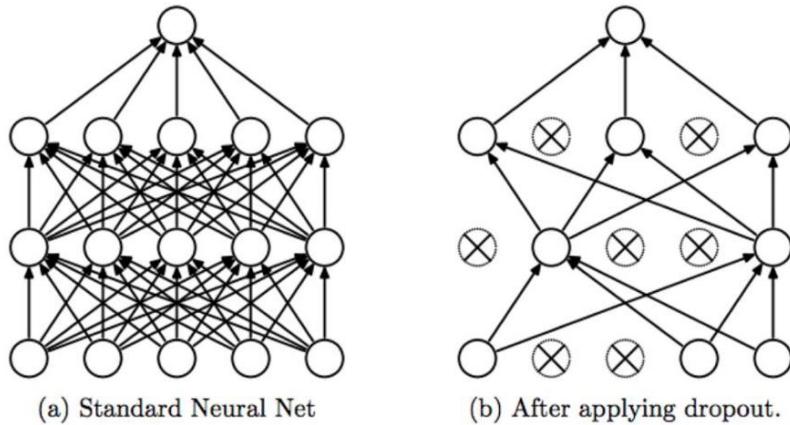
- Thu thập thêm dữ liệu hoặc dùng data augmentation

- Dùng regularization như: L1, L2, dropout

Dropout

Dropout là gì

Dropout với hệ số p nghĩa là trong quá trình train model, với mỗi lần thực hiện cập nhật hệ số trong gradient descent ta ngẫu nhiên loại bỏ $p\%$ số lượng node trong layer đấy, hay nói cách khác là giữ lại $(1-p)\%$ node. Mỗi layer có thể có các hệ số dropout p khác nhau.



Hình 12.4: So sánh model dropout và neural network thông thường [26]

Ví dụ mô hình neural network 1-2-1: 1 input layer, 2 hidden layer và 1 output layer. Ví dụ như hidden layer 1, ta dùng dropout với $p = 0.6$, nên chỉ giữ lại 2 trên 5 node cho mỗi lần cập nhật.

Dropout hạn chế việc overfitting

Overfitting là mô hình đang dùng quá phức tạp so với mô hình thật của dữ liệu. Khi ta dùng dropout như hình trên thì rõ ràng mô hình bên phải đơn giản hơn => tránh overfitting.Thêm vào đó, vì mỗi bước khi train model thì ngẫu nhiên $(1-p\%)$ các node bị loại bỏ nên model không thể phụ thuộc vào bất kì node nào của layer trước mà thay vào đó có xu hướng trái đều weight, **giống như trong L2 regularization => tránh được overfitting.**

Lời khuyên khi dùng dropout [Quan trọng]

1. **Hệ số p nên ở khoảng [0.2, 0.5].** Nếu p quá nhỏ thì không có tác dụng chống overfitting, tuy nhiên nếu p quá lớn thì gần như loại bỏ layer đấy và có thể dẫn đến underfitting.
2. **Nên dùng model lớn, phức tạp hơn** vì ta có dropout chống overfitting.
3. **Dropout chỉ nên dùng cho Fully Connected layer, ít khi được dùng cho ConvNet layer**
4. **Hệ số p ở các layer nên tỉ lệ với số lượng node trong FC layer đó.**

Activation function

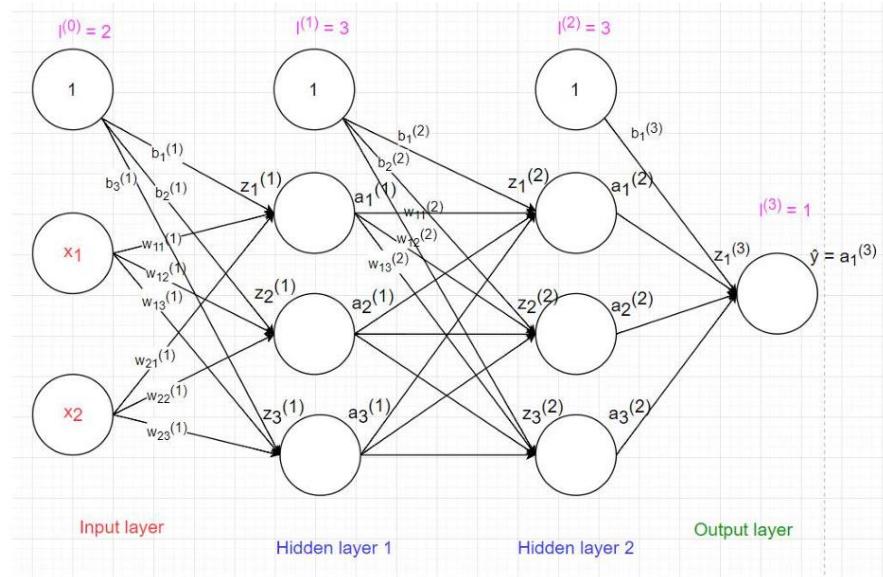
Non-linear activation function

Hàm activation function được dùng sau bước tính tổng linear trong neural network hoặc sau convolutional layer trong CNN. Vì hàm activation là non-linear function.

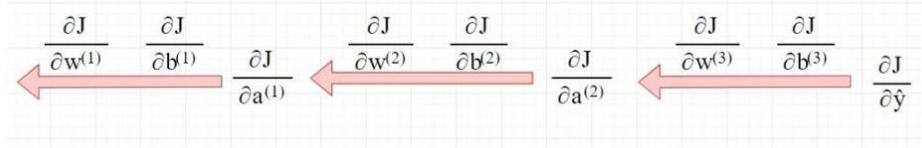
Tóm lại $y = x*a+b$ hay nói cách khác mô hình neural network chỉ là mô hình linear regression đơn giản => Hàm activation function phải là non-linear function.

Vanishing và exploding gradient

Backpropagation là thuật toán được dùng để tính đạo hàm các hệ số trong neural network với loss function để rồi áp dụng gradient descent để tìm các hệ số.



Hình 12.6: Mô hình neural network 2-3-3-1



Hình 12.7: Quá trình backpropagation

Ta có:

$$\frac{\partial J}{\partial A^{(2)}} = \left(\frac{\partial J}{\partial \hat{Y}} \otimes \frac{\partial A^{(3)}}{\partial Z^{(3)}} \right) * (W^{(3)})^T \text{ và } \frac{\partial J}{\partial A^{(1)}} = \left(\frac{\partial J}{\partial A^{(2)}} \otimes \frac{\partial A^{(2)}}{\partial Z^{(2)}} \right) * (W^{(2)})^T$$

$$\text{Do đó } \frac{\partial J}{\partial A^{(1)}} = \left(\left(\frac{\partial J}{\partial \hat{Y}} \otimes \frac{\partial A^{(3)}}{\partial Z^{(3)}} \right) * (W^{(3)})^T \otimes \frac{\partial A^{(2)}}{\partial Z^{(2)}} \right) * (W^{(2)})^T$$

$$\text{Ta tạm kí hiệu đạo hàm của biến qua hàm activation } \frac{\partial A^{(i)}}{\partial Z^{(i)}} = D^{(i)}$$

$$\text{Có thể tạm hiểu là: } \frac{\partial J}{\partial A^{(1)}} = \frac{\partial J}{\partial \hat{Y}} * D^{(3)} * D^{(2)} * W^{(3)} * W^{(2)}$$

$$\text{Nếu neural network có n layer thì } \frac{\partial J}{\partial A^{(l)}} = \frac{\partial J}{\partial \hat{Y}} * \prod_{i=l+1}^n D^{(i)} * \prod_{i=l+1}^n W^{(i)}. \quad (1)$$

Nhận xét

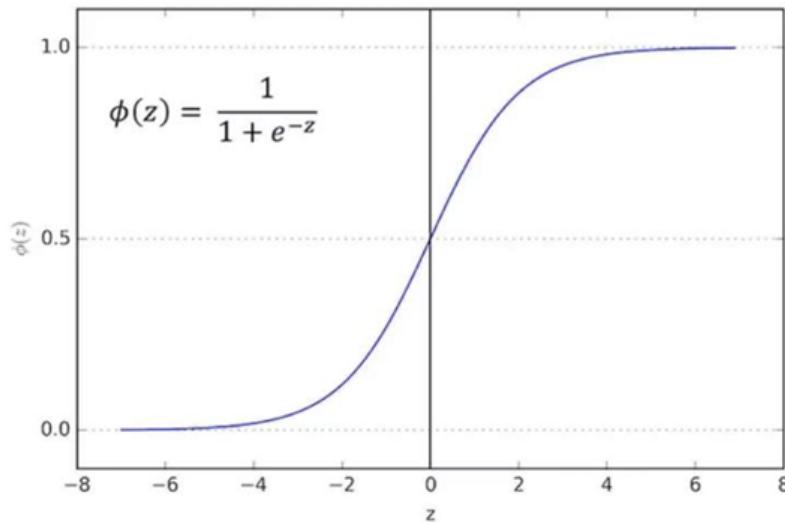
1. Nếu các hệ số W và D đều nhỏ hơn 1 thì khi tính gradient ở các layer đầu ta sẽ phải nhân tích của rất nhiều số nhỏ hơn 1 nên giá trị sẽ tiến dần về 0 và bước cập nhật hệ số trong gradient descent trở nên vô nghĩa và các hệ số neural network sẽ không học được nữa. \Rightarrow Vanishing gradient
2. Nếu các hệ số W và D đều lớn hơn 1 thì khi tính gradient ở các layer đầu ta sẽ phải nhân tích của rất nhiều số lớn hơn 1 nên giá trị sẽ tiến dần về vô cùng và bước cập nhật hệ số trong gradient

descent trở nên không chính xác và các hệ số neural network sẽ không học được nữa. => Exploding gradient

⇒ **Cách giải quyết vanishing/exploding gradient là lựa chọn các giá trị khởi tạo cho hệ số phù hợp và chọn activation function phù hợp.**

Một số activation thông dụng

1. Sigmoid activation function



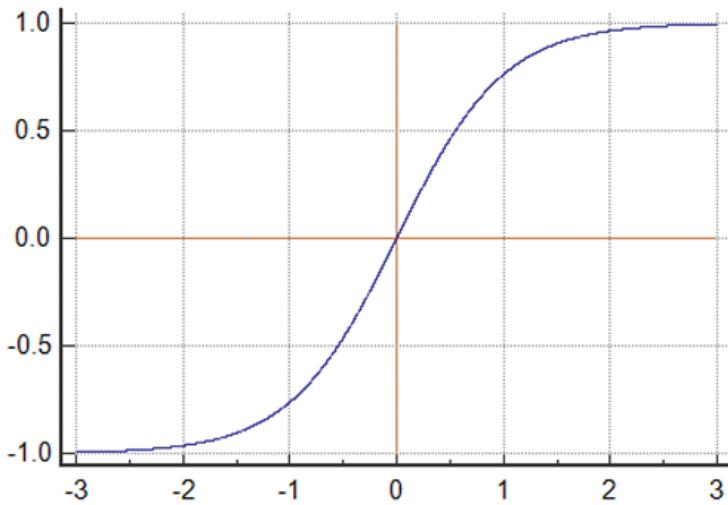
Hình 12.8: Hàm sigmoid

Đạo hàm hàm sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \Rightarrow \frac{d(\sigma(x))}{dx} = \sigma(x) * (1 - \sigma(x)), \text{ do } \sigma(x) > 0 \Rightarrow \sigma(x) * (1 - \sigma(x)) <= \frac{1}{4}$$

Ví dụ $(\frac{1}{4})^{20} = 9 * 10^{-13}$ nên nếu bạn nhìn vào công thức (1) ở trên thì ở những layer đầu tiên sẽ bị **vanishing gradient**.

2. Tanh activation function



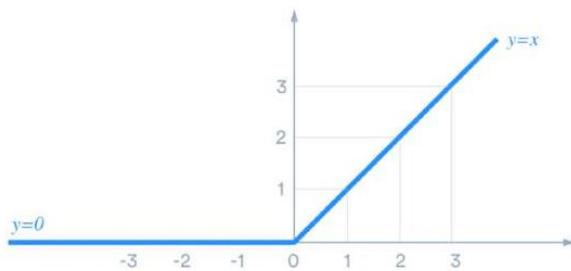
Hình 12.9: Hàm tanh

Hàm tanh: $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, giá trị g(x) trong đoạn (-1,1)

Đạo hàm hàm tanh: $\frac{d(g(x))}{dx} = 1 - g(x)^2 \leq 1$. Do đó khi dùng tanh activation function sẽ bị vanishing gradient.

3. ReLU activation function

ReLU activation function



Hình 12.10: Hàm tanh

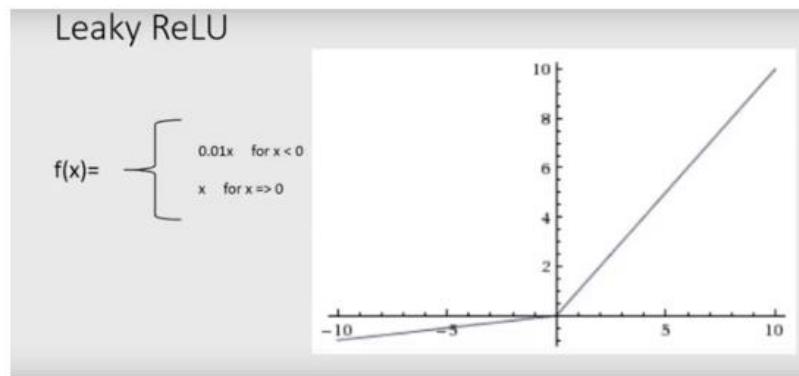
Hàm relu (rectified linear unit): $y = \max(0, x)$

Nhận xét:

- Hàm ReLU activation đơn giản để tính \Rightarrow thời gian train model nhanh hơn.
- Đạo hàm là 1 với $x \geq 0$ nên không bị vanishing gradient.

Tuy nhiên với các node có giá trị nhỏ hơn 0, qua ReLU activation sẽ thành 0, hiện tượng đấy gọi là "Dying ReLU". Nếu các node bị chuyển thành 0 thì sẽ không có ý nghĩa với bước linear activation ở lớp tiếp theo và các hệ số tương ứng từ node đấy cũng không được cập nhật với gradient descent. \Rightarrow Leaky ReLU ra đời.

4. Leaky ReLU



Hình 12.11: Hàm Leaky ReLU

Hàm Leaky ReLU có các điểm tốt của hàm ReLU và giải quyết được vấn đề Dying ReLU bằng cách xét một độ dốc nhỏ cho các giá trị âm thay vì để giá trị là 0.

Lời khuyên: Mặc định nên dùng ReLU làm hàm activation. Không nên dùng hàm sigmoid.

Lời khuyên: Mặc định nên dùng ReLU làm hàm activation. Không nên dùng hàm sigmoid.

Batch Normalize (Chuẩn hóa theo Batch)

- Vanishing gradient : Biến mất gradient

Batch Normalization ra đời

Batch normalization thực hiện việc chuẩn hóa (normalizing) và zero centering (mean subtracting) dữ liệu trước khi đưa qua hàm activation (giá trị trung bình (mean) sẽ được đưa về 0 và phương sai (variance) sẽ được đưa về 1). Để thực hiện 2 công việc trên, batch normalization tính toán phương sai và độ lệch chuẩn của dữ liệu dựa trên các batches, rồi sử dụng 2 tham số γ và β tinh chỉnh đầu ra.

Batch normalization:

$$\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x^{(i)}$$

$$\sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x^{(i)} - \mu_B)^2$$

$$\hat{x}^{(i)} = (x^{(i)} - \mu_B) / (\sqrt{\sigma_B^2 + \xi})$$

$$z_{(i)} = \gamma \hat{x} + \beta$$

Trong đó:

μ_B là giá trị trung bình của batch B

σ_B^2 là phương sai của batch B

$\hat{x}^{(i)}$ là giá trị của mẫu thứ i trong batch B sau khi được normalize và zero centering

$z_{(i)}$ là đầu ra của mẫu thứ i trong batch B

γ là scaling parameter của lớp

β là shifting parameter của lớp

ξ là smoothing parameter, tránh xảy ra việc chia cho 0, giá trị rất nhỏ

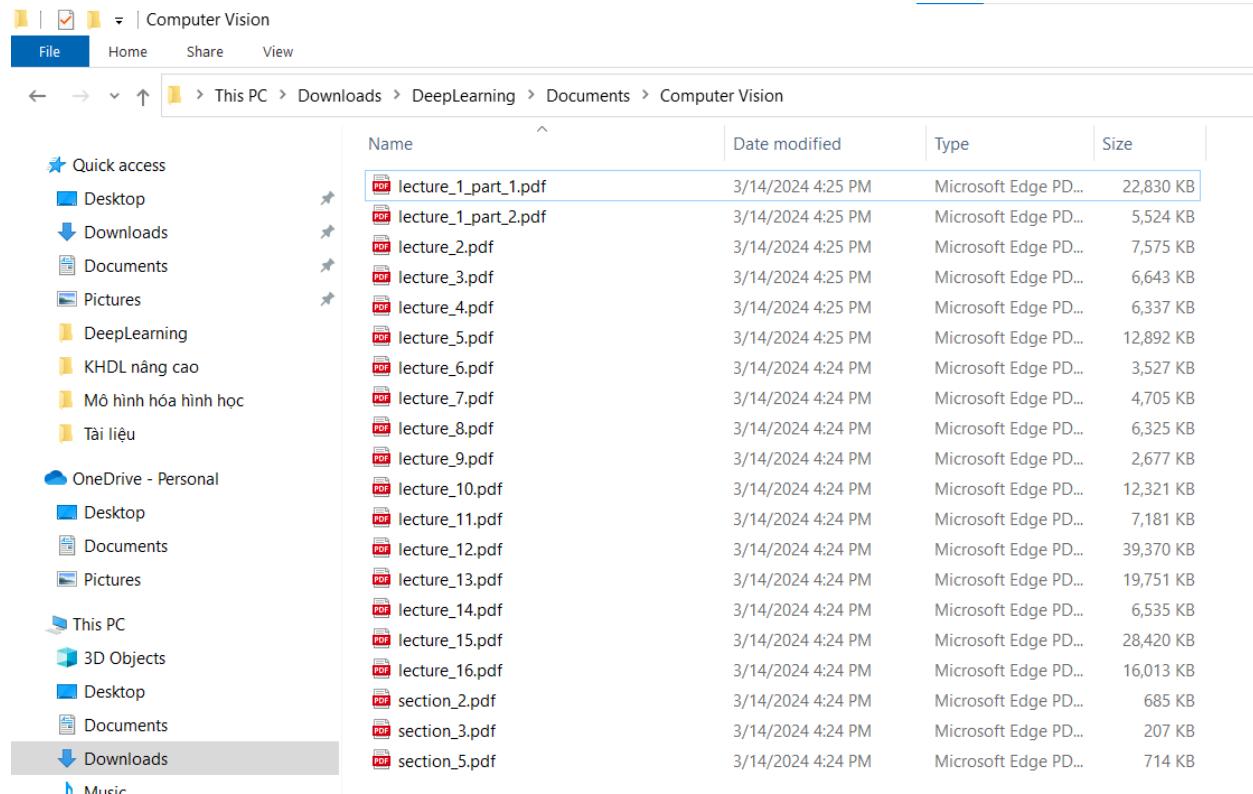
Chú ý: γ và β là 2 tham số được học trong quá trình training.

Hiệu quả của batch normalization

- Batch normalization đưa dữ liệu về zero mean và chuẩn hóa variance trước khi đưa qua activation function nhờ đó giải quyết các vấn đề vanishing gradient hay exploding gradient.
- Batch normalization cho phép learning rate lớn trong quá trình huấn luyện.
- Batch-Norm giảm thiểu sự ảnh hưởng của quá trình khởi tạo trọng số ban đầu.
- Batch-Norm chuẩn hóa dữ liệu đầu ra của các layer giúp model trong quá trình huấn luyện không bị phụ thuộc vào một thành phần trọng số nhất định. Do đó, Batch-norm còn được sử dụng như một regularizer giúp giảm overfitting

Phần 6 : Computer Vision Task

Tài liệu



- The K-Nearest Neighbors (KNN) (K láng giềng gần nhất)
- K-means clustering (K cụm)

13. Object detection với Faster R-CNN

R-CNN (Region Convolution Neural Network)(Mạng nơ ron tích chập khu vực)

Bài toán object detection

Trong bài 7, sách đã giới thiệu về ứng dụng mô hình CNN cho bài toán phân loại ảnh, tuy nhiên các ảnh input của bài toán phân loại chỉ bao gồm 01 đối tượng cụ thể như chữ số hay 01 loài hoa.

⇒ **Tuy nhiên là ảnh trong cuộc sống bình thường thì không chỉ có 01 đối tượng mà thường chứa nhiều các đối tượng khác. Từ đó nảy sinh vấn đề cần tìm vị trí của từng đối tượng trong ảnh. Đó là bài toán object detection.**

Bài toán object detection có **input là ảnh màu và output là vị trí của các đối tượng trong ảnh**. Ta thấy nó bao gồm 2 bài toán nhỏ:

- Xác định các bounding box (hình chữ nhật) quanh đối tượng.
- Với mỗi bounding box thì cần phân loại xem đây là đối tượng gì (chó, ngựa, ô tô,...) với bao nhiêu phần trăm chắc chắn.

Việc lựa chọn có bao nhiêu loại đối tượng thì phụ thuộc vào bài toán mà ta đang giải quyết.

Bạn tự hỏi liệu mô hình CNN có giải quyết được bài toán object detection không? Vấn đề chính là vì không biết trước có bao nhiêu đối tượng trong ảnh, nên không thiết kế được output layer hiệu quả => **mô hình CNN truyền thống không giải quyết được => R-CNN** (regional convolutional neural network) ra đời.

Faster R-CNN

Ta sẽ có lịch sử phát triển là :

R-CNN → Fast R-CNN → Faster R-CNN

Fast R-CNN để cải thiện tốc độ bằng cách : Fast R-CNN khác với R-CNN là nó thực hiện feature map với cả ảnh sau đó với lấy các region proposal ra từ feature map, còn R-CNN thực hiện tách các region proposal ra rồi mới thực hiện CNN trên từng region proposal. Do đó Fast R-CNN nhanh hơn đáng kể nhờ tối ưu việc tính toán bằng Vectorization.

(R-CNN xác định các ROI Ngay trên ảnh rồi mới cho qua Conv khác với Fast R-CNN thì cho ảnh qua Conv thành Feature extractor sau đó mới tách thành các ROI)

Tuy nhiên nhìn hình trên ở phần test time với mục Fast R-CNN thì thời gian tính region proposal rất lâu và làm chậm thuật toán => Cần thay thế thuật toán selective search. Giờ người ta nghĩ đến việc dùng deep learning để tạo ra region proposal => Faster R-CNN ra đời.

Faster R-CNN để cải thiện test time

R-CNN (Region with CNN feature)

- R-CNN xác định các ROI Ngay trên ảnh rồi mới cho qua Conv khác với Fast R-CNN thì cho ảnh qua Conv thành Feature extractor sau đó rồi mới tách thành các ROI
- **Region proposal : Khu vực đề xuất**

Ý tưởng thuật toán R-CNN khá đơn giản

Bước 1: Dùng Selective Search algorithm để lấy ra khoảng 2000 bounding box trong input mà có khả năng chứa đối tượng.

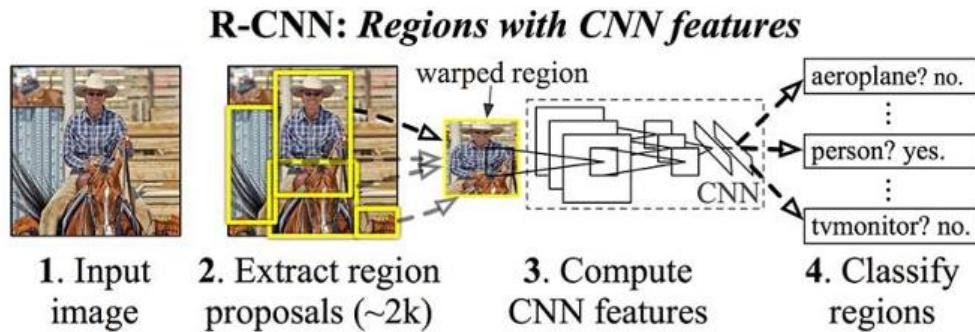
Bước 2: Với mỗi bounding box ta xác định xem nó là đối tượng nào (người, ô tô, xe đạp,...)

Selective search algorithm

Input của thuật toán là ảnh màu, output là khoảng 2000 region proposal (bounding box) mà có khả năng chứa các đối tượng.

Phân loại region proposal

Bài toán trở thành phân loại ảnh cho các region proposal. Do thuật toán selective search cho tới 2000 region proposal nên có rất nhiều region proposal không chứa đối tượng nào. Vậy nên ta cần thêm 1 lớp background (không chứa đối tượng nào). Ví dụ như hình dưới ta có 4 region proposal, ta sẽ phân loại mỗi bounding box là người, ngựa hay background.



Hình 13.4: Các bước trong RCNN [8]

Sau đó các region proposal được resize lại về cùng kích thước và thực hiện transfer learning với feature extractor, sau đó các extracted feature được cho vào thuật toán SVM để phân loại ảnh.

Bên cạnh đó thì extracted feature cũng được dùng để dự đoán 4 offset values cho mỗi cạnh. Ví dụ như khi region proposal chứa người nhưng chỉ có phần thân và nửa mặt, nửa mặt còn lại không có trong region proposal đó thì offset value có thể giúp mở rộng region proposal để lấy được toàn bộ người.

Vấn đề với R-CNN

Hồi mới xuất hiện thì thuật toán hoạt động khá tốt cho với các thuật toán về computer vision trước đó nhờ vào CNN, tuy nhiên nó vẫn có khá nhiều hạn chế:

- Vì với mỗi ảnh ta cần phân loại các class cho 2000 region proposal nên thời gian train rất lâu.
- Không thể áp dụng cho real-time vì mỗi ảnh trong test set mất tới 47s để xử lý.

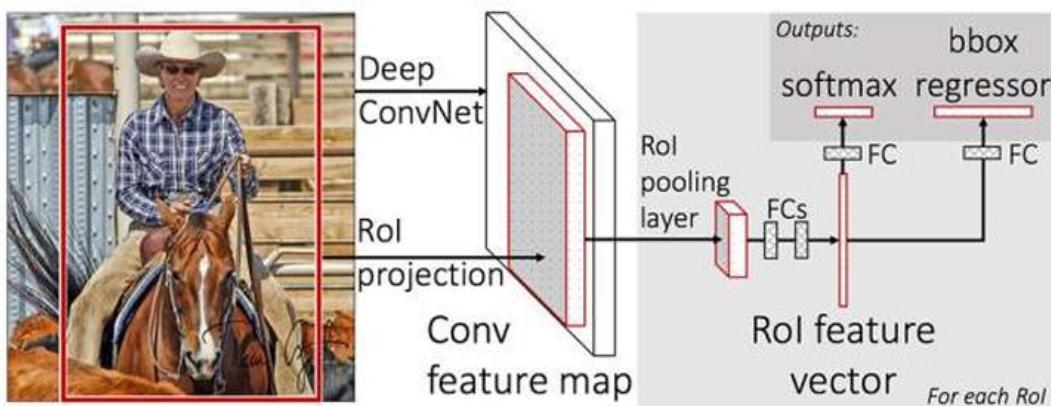
Fast R-CNN

- R-CNN xác định các ROI Ngay trên ảnh rồi mới cho qua Conv khác với Fast R-CNN thì cho ảnh qua Conv thành Feature extractor sau đó rồi mới tách thành các ROI

Khoảng 1.5 năm sau đó, Fast R-CNN được giới thiệu bởi cùng tác giả của R-CNN, nó giải quyết được một số hạn chế của R-CNN để cải thiện tốc độ.

Tương tự như R-CNN thì Fast R-CNN vẫn dùng selective search để lấy ra các region proposal. Tuy nhiên là nó không tách 2000 region proposal ra khỏi ảnh và thực hiện bài toán image classification cho mỗi ảnh. Fast R-CNN cho cả bức ảnh vào ConvNet (một vài convolutional layer + max pooling layer) để tạo ra convolutional feature map.

Sau đó các vùng region proposal được lấy ra tương ứng từ convolutional feature map. Tiếp đó được Flatten và thêm 2 Fully connected layer (FCs) để dự đoán lớp của region proposal và giá trị offset values của bounding box.



Tuy nhiên là kích thước của các region proposal khác nhau nên khi Flatten sẽ ra các vector có kích thước khác nhau nên không thể áp dụng neural network được. Thủ nhỉn lại xem ở trên R-CNN đã xử lý như thế nào? Nó đã resize các region proposal về cùng kích thước trước khi dùng transfer learning. Tuy nhiên ở feature map ta không thể resize được, nên ta phải có cách gì đấy để chuyển các region proposal trong feature map về cùng kích thước => Region of Interest (ROI) pooling ra đời.

Region of Interest (ROI) pooling

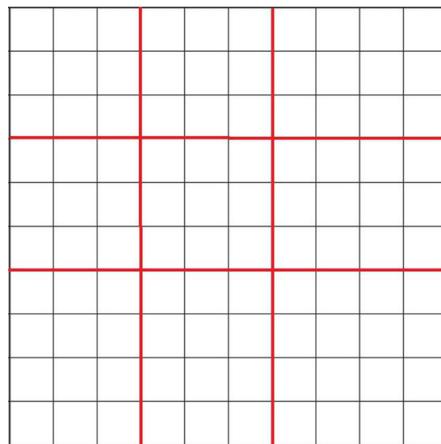
ROI pooling là một dạng của **pooling layer**. Điểm khác so với **max pooling** hay **average pooling** là bất kể kích thước của tensor input, ROI pooling luôn cho ra output có kích thước cố định được định nghĩa trước.

Ta kí hiệu a/b là phần nguyên của a khi chia cho b và $a\%b$ là phần dư của a khi chia cho b . Ví dụ: $10/3 = 3$ và $10\%3 = 1$.

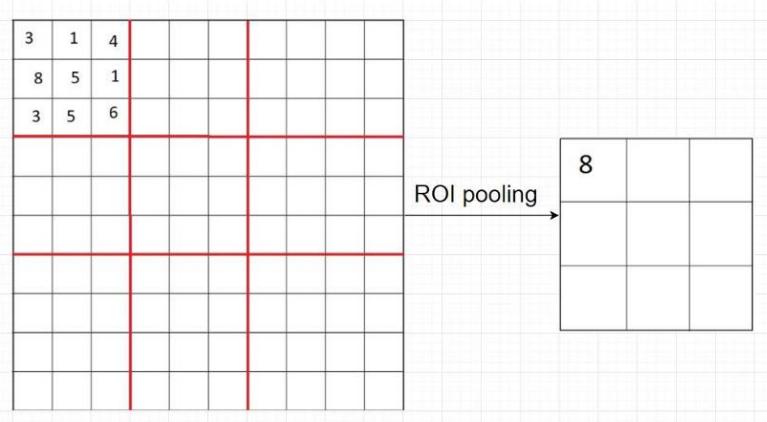
Gọi input của ROI pooling có kích thước $m*n$ và output có kích thước $h*k$ (thông thường h, k nhỏ ví dụ $7*7$).

- Ta chia chiều rộng thành h phần, $(h-1)$ phần có kích thước m/h , phần cuối có kích thước $m/h + m\%h$.
- Tương tự ta chia chiều dài thành k phần, $(k-1)$ phần có kích thước n/k , phần cuối có kích thước $n/k + n\%k$.

Ví dụ $m=n=10$, $h=k=3$, do $m/h = 3$ và $m\%h = 1$, nên ta sẽ chia chiều rộng thành 3 phần, 2 phần có kích thước 3, và 1 phần có kích thước 4.



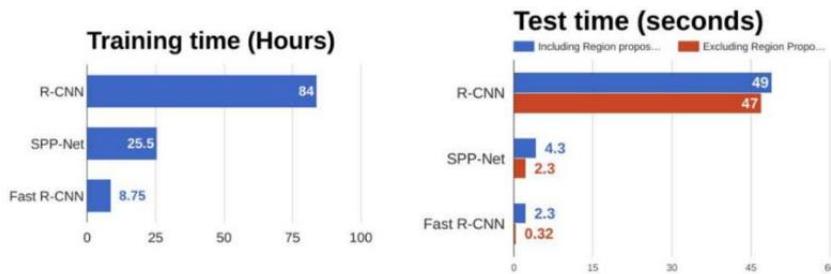
Sau đó với mỗi khối được tạo ra bằng các đường đỏ và cạnh, ta thực hiện max pooling lấy ra 1 giá trị.



Hình 13.6: Thực hiện ROI pooling

Ta có thể thấy là kích thước sau khi thực hiện ROI pooling về đúng $h*k$ như ta mong muốn.

Đánh giá Fast R-CNN



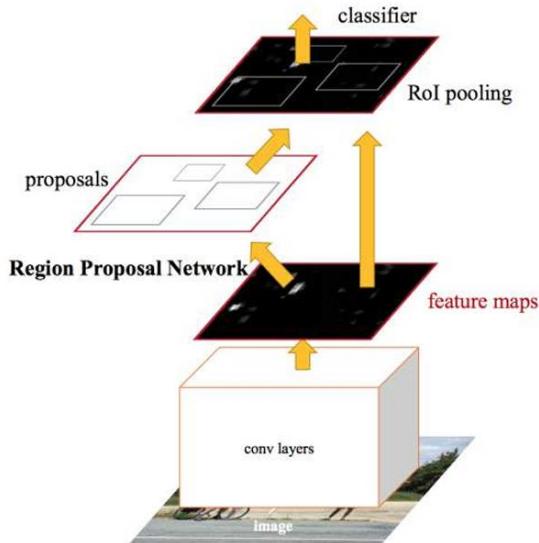
Hình 13.7: So sánh thời train train và test giữa R-CNN và Fast R-CNN [17]

Fast R-CNN khác với R-CNN là nó thực hiện feature map với cả ảnh sau đó với lấy các region proposal ra từ feature map, còn R-CNN thực hiện tách các region proposal ra rồi mới thực hiện CNN trên từng region proposal. Do đó Fast R-CNN nhanh hơn đáng kể nhờ tối ưu việc tính toán bằng Vectorization.

Tuy nhiên nhìn hình trên ở phần test time với mục Fast R-CNN thì thời gian tính region proposal rất lâu và làm chậm thuật toán => Cần thay thế thuật toán selective search. Giờ người ta nghĩ đến việc dùng deep learning để tạo ra region proposal => Faster R-CNN ra đời.

Faster R-CNN

Region Proposal Network (RPN) : Mạng lưới để xuất khu vực

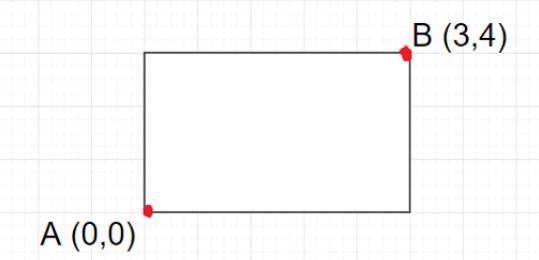


Hình 13.8: Kiến trúc mới Faster R-CNN [18]

Đầu tiên cả bức ảnh được cho qua pre-trained model để lấy feature map. Sau đó feature map được dùng cho Region Proposal Network để lấy được các region proposal. Sau khi lấy được vị trí các region proposal thì thực hiện tương tự Fast R-CNN.

Region Proposal Network (RPN) : Mạng khu vực đề xuất

Input của RPN là feature map và output là các region proposal. Ta thấy các region proposal là hình chữ nhật.



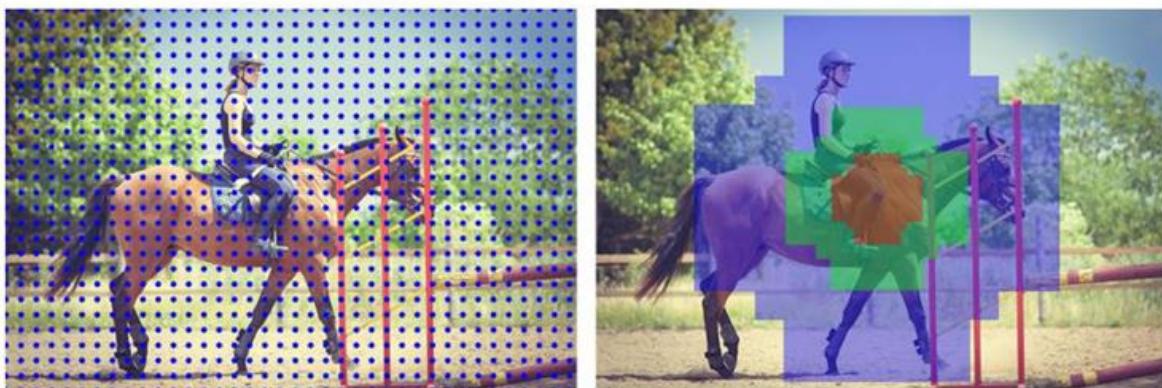
Mà một hình chữ nhật được xác định bằng 2 điểm ở 2 góc, ví dụ A(x_min, y_min) và B(x_max, y_max). Nhận xét:

- Khi RPN dự đoán ta phải rồng buộc $x_{min} < x_{max}$ và $y_{min} < y_{max}$.
- Hơn nữa các giá trị x,y khi dự đoán có thể ra ngoài khung bức ảnh

=> Cần một kĩ thuật mới để biểu diễn region proposal => Anchor ra đời.

Ý tưởng là thay vì dự đoán 2 góc ta sẽ dự đoán **điểm trung tâm (x_center, y_center)** và width, height của hình chữ nhật. Như vậy mỗi anchor được xác định bằng 4 tham số (x_center, y_center, width, height).

Vì không sử dụng Selective search nên RPN ban đầu cần xác định các anchor box có thể là region proposal, sau đó qua RPN thì chỉ output những anchor box chắc chắn chứa đối tượng.



Hình 13.9: Ví dụ về anchor [21]

Ảnh bên trái kích thước $400 * 600$ pixel, tâm của anchor box màu xanh, cách nhau 16 pixel => có khoảng $(400*600)/(16*16) = 938$ tâm. Do các object trong ảnh có thể có kích thước và tỉ lệ khác nhau nên với mỗi tâm ta định nghĩa 9 anchors với kích thước 64×64 , 128×128 , 256×256 , mỗi kích thước có 3 tỉ lệ tương ứng: $1 : 1$, $1 : 2$ và $2 : 1$.

⇒ Mỗi hình bên phải là các anchor chồng lên nhau , ví dụ như trong hình là có 3 anchor

Giống như hình bên phải với tâm ở giữa 3 kích thước ứng với màu da cam, xanh lam, xanh lục và với mỗi kích thước có 3 tỉ lệ.

=> Số lượng anchor box giờ là $938 * 9 = 8442$ anchors. Tuy nhiên sau RPN ta chỉ giữ lại khoảng

1000 anchors box để thực hiện như trong Fast R-CNN.

Việc của RPN là lấy ra các region proposal giống như selective search thôi chứ không phải là phân loại ảnh.

Mô hình RPN khá đơn giản, feature map được cho qua Conv layer 3*3, 512 kernels. Sau đó với mỗi anchor lấy được ở trên, RPN thực hiện 2 bước:

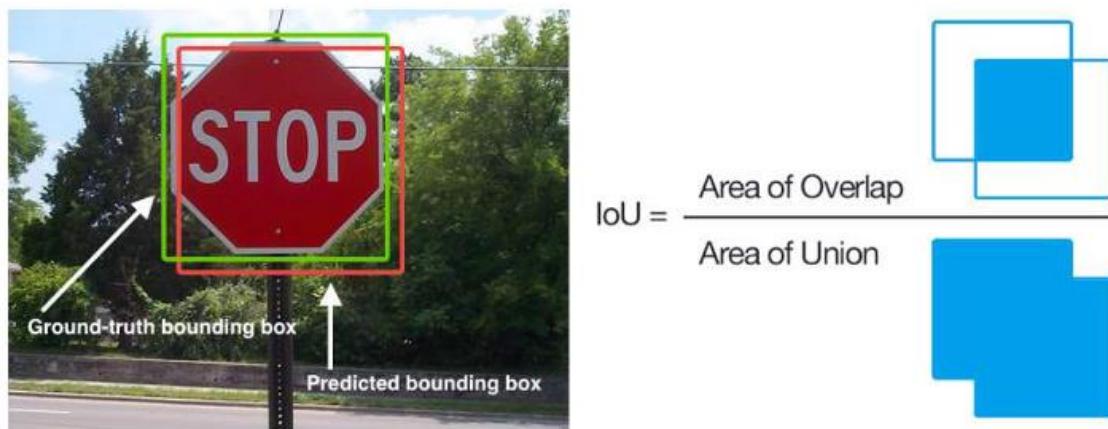
1. Dự đoán xem anchor đây là foreground (chứa object) hay background (không chứa object)
2. Dự đoán 4 offset value cho x_center, y_center, width, height cho các anchor.

Nhận xét: có rất nhiều anchor bị chồng lên nhau nên non-maxima suppression được dùng để loại bỏ các anchor chồng lên nhau.

Sau cùng dựa vào phần trăm dự đoán background RPN sẽ lấy N anchor (N có thể 2000, 1000, thậm chí 100 vẫn chạy tốt) để làm region proposal.

Intersection over Union (IoU)

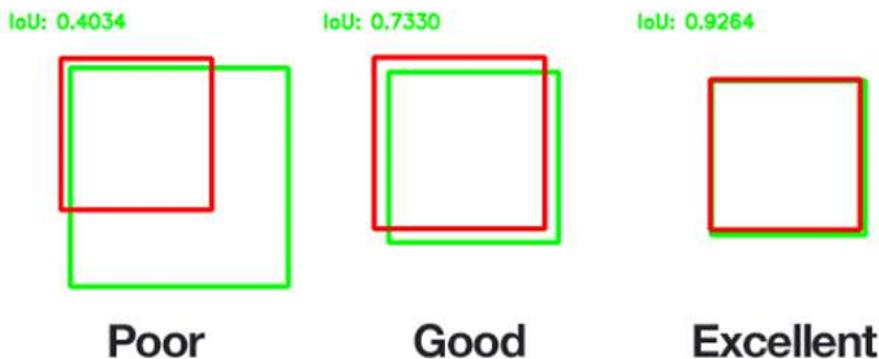
IoU được sử dụng trong bài toán object detection, để đánh giá xem bounding box dự đoán đối tượng khớp với ground truth thật của đối tượng.



Hình 13.10: Ví dụ về IoU [11]

Ví dụ về hệ số IoU, nhận xét:

- Chỉ số IoU trong khoảng [0,1]
- IoU càng gần 1 thì bounding box dự đoán càng gần ground truth



Hình 13.11: Chỉ số IoU [11]

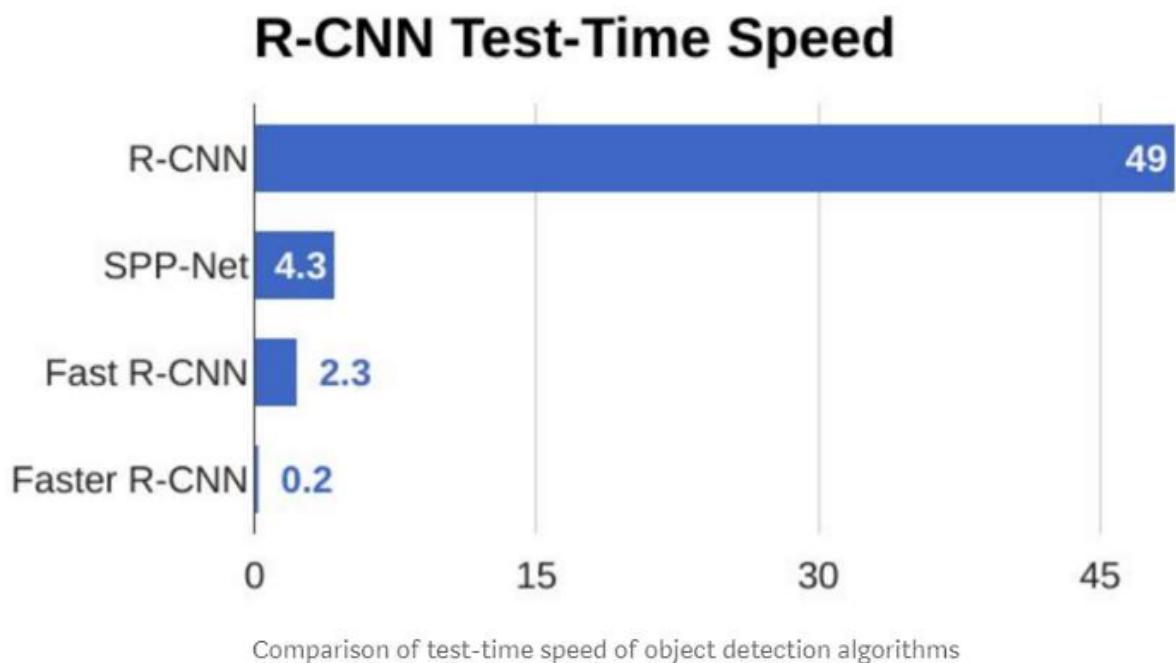
Ngoài ra thì hệ số IoU cũng được dùng để đánh giá độ khớp của 2 hình chữ nhật.

Non-maxima suppression

Ở trong Region Proposal Network đầu tiên ta có khoảng 9000 anchor box (tập Input) tuy nhiên ta chỉ muốn giữ lại 100 anchor (tập Ouput) làm region proposal. Ta sẽ làm như sau:

- Chọn ra anchor box (A) có xác suất là foreground lớn nhất trong tập Input.
- Thêm A vào tập Ouput.
- Loại bỏ A và các anchor box trong tập Input mà có hệ số IoU với A lớn hơn 0.5 ra khỏi tập Input.
- Kiểm tra nếu tập Input rỗng hoặc tập Output đủ 100 anchor thì dừng lại, nếu không quay lại bước 1.

Kết quả của Faster R-CNN



- ⇒ Nhìn ở hình trên ta thấy Faster R-CNN nhanh hơn hẳn các dòng R-CNN trước đó, vì vậy có thể dùng cho real-time object detection

Ứng dụng object detection

- **Tự động điểm danh:** Xác định được vị trí mặt của các học sinh và phân loại các học sinh trong lớp.
- **Hỗ trợ ô tô tự lái:** Xác định được vị trí và phân loại được các phương tiện giao thông, người đi bộ.
- **Dự đoán hành vi:** Xác định được vị trí và phân loại người => track được người => dùng RNN để dự đoán hành vi.
- ⇒ Ví dụ như bài toán nhận diện khuôn mặt , điểm danh và chấm công bằng camera thì có 2 phần . Phần detect (phát hiện) vùng chứa khuôn mặt và phần recognition .

14. Bài toán phát hiện biển số xe máy Việt Nam [HAY]

Full source and Dataset : [phamvanhanh6720/LicensePlateDetection \(github.com\)](https://github.com/phamvanhanh6720/LicensePlateDetection)

Tóm tắt quá trình :

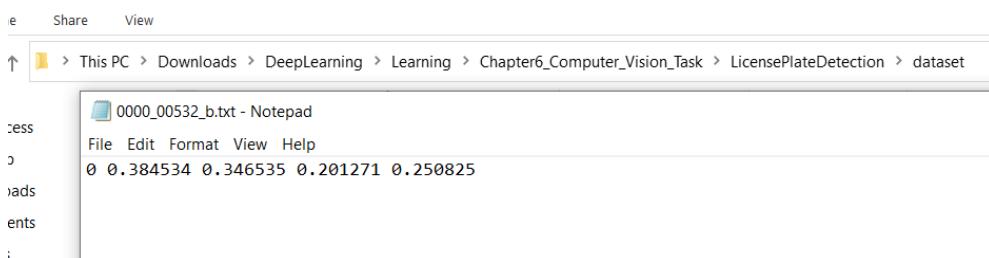
1. Thu thập dữ liệu về biển số xe
2. Cấu hình darknet và training model detect bằng file trọng số có sẵn

Mỗi dòng trong một file annotation bao gồm: <object-class> <x> <y> <width> <height>.

Trong đó: <x> <y> <width> <height> tương ứng là tọa độ trung tâm và kích thước của đối tượng. Các giá trị này đã được chuẩn hóa lại, do vậy giá trị luôn nằm trong đoạn [0,1]. object-class là chỉ số đánh dấu các classes.

Lưu ý: Với bài toán có nhiều nhãn, nhiều người cùng gán nhãn thì cần thống nhất với nhau trước về thứ tự nhãn. Nguyên nhân do trong file annotation chỉ lưu chỉ số (0,1,3,4,...) của nhãn chứ không lưu tên nhãn.

| | | | |
|------------------|--------------------|---------------|-------|
| 0000_00532_b.jpg | 7/28/2020 11:35 PM | JPG File | 17 KB |
| 0000_00532_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0000_02187_b.jpg | 7/28/2020 11:35 PM | JPG File | 18 KB |
| 0000_02187_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0000_05696_b.jpg | 7/28/2020 11:35 PM | JPG File | 16 KB |
| 0000_05696_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0000_06886_b.jpg | 7/28/2020 11:35 PM | JPG File | 16 KB |
| 0000_06886_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0000_08244_b.jpg | 7/28/2020 11:35 PM | JPG File | 17 KB |
| 0000_08244_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0001_05318_b.jpg | 7/28/2020 11:35 PM | JPG File | 18 KB |
| 0001_05318_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 1.jpg | 7/28/2020 11:35 PM | JPG File | 33 KB |
| 1.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0002_02183_b.jpg | 7/28/2020 11:35 PM | JPG File | 18 KB |
| 0002_02183_b.txt | 7/28/2020 11:35 PM | Text Document | 1 KB |
| 0002_02554_b.jpg | 7/28/2020 11:35 PM | JPG File | 17 KB |



⇒ Vị trí tương ứng của các ảnh

3. Sau khi train xong thì lấy model này để detect tập ảnh của mình

⇒ Chỉ cần có file trọng số này ta sẽ vẽ ra được đường bao của biển số xe

Giới thiệu

Bài toán nhận diện biển số Việt Nam là một bài toán không còn mới, đã được phát triển dựa trên các phương pháp xử lý ảnh truyền thống và cả những kỹ thuật mới sử dụng Deep Learning. Trong bài toán này tôi chỉ phát triển bài toán phát hiện biển số (một phần trong bài toán nhận diện biển số) dựa trên thuật toán YOLO-Tinyv4 với mục đích:

- Hướng dẫn chuẩn bị dữ liệu cho bài toán Object Detection.
- Hướng dẫn huấn luyện YOLO-TinyV4 dùng darknet trên Google Colab.

Chuẩn bị dữ liệu

Đánh giá bộ dữ liệu

Dataset : [DataThiGiacMayTinh/GreenParking.zip at main · thigiacmaytinh/DataThiGiacMayTinh \(github.com\)](https://github.com/thigiacmaytinh/DataThiGiacMayTinh)

Các phương pháp tăng sự đa dạng của bộ dữ liệu

Đa dạng kích thước của biển số

Đa dạng kích thước bằng 2 cách:

- Cách 1: Thu nhỏ kích thước biển bằng cách thêm biên kích thước ngẫu nhiên vào ảnh gốc, sau đó resize ảnh bằng kích thước ảnh ban đầu.
- Cách 2: Crop ảnh chứa biển số với kích thước ngẫu nhiên, sau đó resize ảnh bằng kích thước ảnh ban đầu.

```
import cv2
import random
import matplotlib.pyplot as plt

def show_img(image1, image2):
    plt.figure(figsize=(20, 20))
    plt.subplot(1, 2, 1) # plt.subplot(nrows, ncols, index)
    image = cv2.imread(image1)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.subplot(1, 2, 2)
    image = cv2.imread(image2)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

# Cách1

def add_border(image_path, output_path, low, high):
    # random các kích thước biên trong khong (low, high)
    top = random.randint(low, high)
```

```

bottom = random.randint(low, high)

left = random.randint(low, high)

right = random.randint(low, high)

image = cv2.imread(image_path)

original_width, original_height = image.shape[1], image.shape[0]

#sdng hàm ca opencv đ thêm biên

image = cv2.copyMakeBorder(image, top, bottom, left, right, cv2.BORDER_REPLICATE)

#sau đó resize nh bng kích thuc ban đầu ca nh

image = cv2.resize(image, (original_width, original_height))

cv2.imwrite(output_path, image)

add_border("input.jpg", "output.jpg", 100, 100)

show_img("input.jpg", "output.jpg")

```



```

# Cách2

def random_crop(image_path, out_path):

    image = cv2.imread(image_path)

    original_width, original_height = image.shape[1], image.shape[0]

    x_center,y_center = original_height//2, original_width//2

    x_left = random.randint(0, x_center//2)

    x_right = random.randint(original_width-x_center//2, original_width)

    y_top = random.randint(0, y_center//2)

    y_bottom = random.randint(original_height-y_center//2, original_width)

```

```

# crop ra vùng nh vi kích thước ngu nhiên
cropped_image = image[y_top:y_bottom, x_left:x_right]

# resize nh bng kích thu c nh ban đù
cropped_image = cv2.resize(cropped_image, (original_width, original_height))

cv2.imwrite(out_path, cropped_image)

# Sử dụng hàm để cắt ảnh và lưu
random_crop('input.jpg', 'output_image_c2.jpg')
show_img("input.jpg", "output_image_c2.jpg")

```



Thay đổi độ sáng của ảnh

```

def change_brightness(image_path, output_path, value):
    """
    value: độ sáng thay đổi
    """
    img=cv2.imread(image_path)

    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    v = cv2.add(v, value)
    v[v > 255] = 255
    v[v < 0] = 0

    final_hsv = cv2.merge((h, s, v))

    img = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)
    cv2.imwrite(output_path, img)

```

```
# Sử dụng hàm để thay đổi độ sáng của ảnh và lưu lại
change_brightness('input.jpg', 'output_image_brightness.jpg', 100) # Thay đổi độ sáng lên 50
show_img("input.jpg", "output_image_brightness.jpg")
```



Xoay ảnh

```
import imutils
def rotate_image(image_path, range_angle, output_path):
    image = cv2.imread(image_path)
    #là chn ngu nhiên góc quay
    angle = random.randint(-range_angle, range_angle)
    img_rot = imutils.rotate(image, angle)
    cv2.imwrite(output_path, img_rot)
rotate_image('input.jpg', 60, 'output_image_rotate.jpg') # Phạm vi góc quay từ -30 đến 30 độ
show_img("input.jpg", "output_image_rotate.jpg")
```



Gán nhãn dữ liệu

Tool : <https://github.com/HumanSignal/labelImg>

LabelImg hỗ trợ gán nhãn trên cả 2 định dạng PASCAL VOC và YOLO với phần mở rộng file annotation tương ứng là .xml và .txt.

Trong bài toán sử dụng mô hình YOLO, tôi lưu file annotation dưới dạng .txt.

```
0 0.384534 0.346535 0.201271 0.250825
```

Hình 14.7: Nội dung trong một file annotation

Mỗi dòng trong một file annotation bao gồm: **<object-class> <x> <y> <width> <height>**.

Trong đó: **<x> <y> <width> <height>** tương ứng là tọa độ trung tâm và kích thước của đối tượng. Các giá trị này đã được chuẩn hóa lại, do vậy giá trị luôn nằm trong đoạn [0,1]. **object-class** là chỉ số đánh dấu các classes.

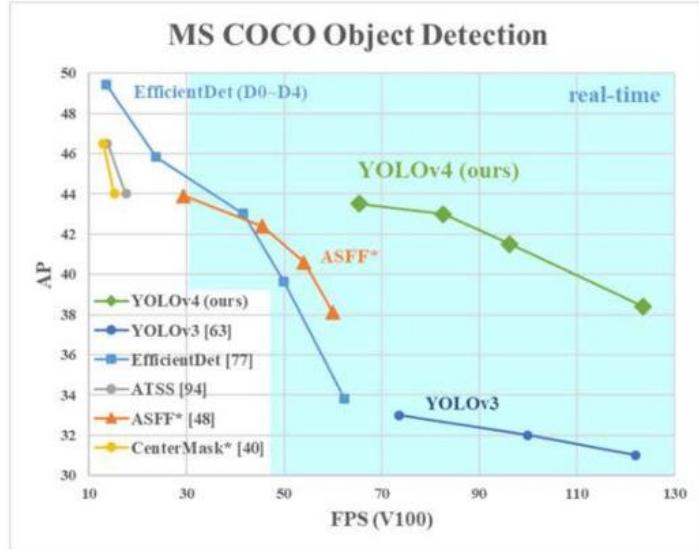
Lưu ý: Với bài toán có nhiều nhãn, nhiều người cùng gán nhãn thì cần thống nhất với nhau trước về thứ tự nhãn. Nguyên nhân do trong file annotation chỉ lưu chỉ số (0,1,3,4,...) của nhãn chứ không lưu tên nhãn.

Sau khi gán nhãn xong các bạn để file annotation và ảnh tương ứng **vào cùng một thư mục**.

Huấn luyện mô hình

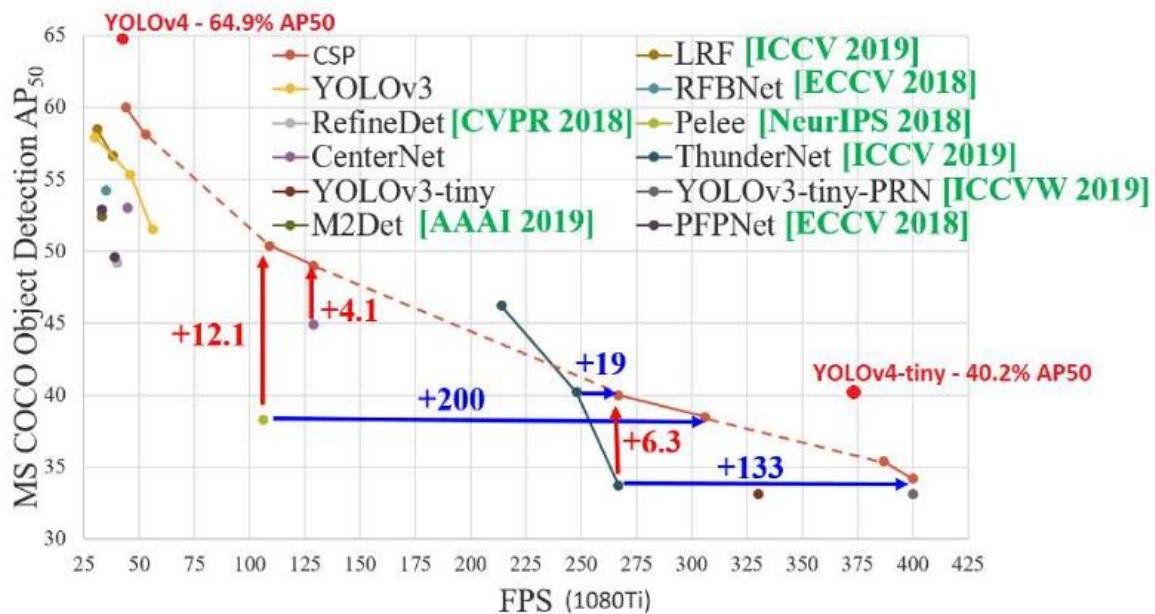
Giới thiệu về YOLO-Tinyv4 và darknet

YOLOv4 là thuật toán **Object Detection**, mới được công bố trong thời gian gần đây với sự cải thiện về kết quả đáng kể so với YOLOv3.



Hình 14.8: Sự cải thiện của YOLOv4 ([source](#))

YOLOv4 cho kết quả real-time khi chạy trên các nền tảng GPU cao cấp. Với mục đích trade-off giữa độ chính xác và tốc độ để có thể chạy trên các nền tảng CPU và GPU thấp hơn thì YOLO-Tinyv4 được ra đời.



Hình 14.9: YOLOv4 với YOLO-Tinyv4 ([source](#))

[YOLOv4-tiny released: 40.2% AP50, 371 FPS \(GTX 1080 Ti\), 1770 FPS tkDNN/TensorRT · Issue #6067 · AlexeyAB/darknet \(github.com\)](#)

YOLOv4-tiny released: 40.2% AP50, 371 FPS (GTX 1080 Ti) / 330 FPS (RTX 2070)

- 1770 FPS - on GPU RTX 2080Ti - (416x416, fp16, batch=4) tkDNN/TensorRT [ceccocats/tkDNN#59](#) (comment)
- 1353 FPS - on GPU RTX 2080Ti - (416x416, fp16, batch=4) OpenCV 4.4.0 (including: transferring CPU->GPU and GPU->CPU) (excluding: nms, pre/post-processing) [#6067](#) (comment)
- 39 FPS - 25ms latency - on Jetson Nano - (416x416, fp16, batch=1) tkDNN/TensorRT [ceccocats/tkDNN#59](#) (comment)
- 290 FPS - 3.5ms latency - on Jetson AGX - (416x416, fp16, batch=1) tkDNN/TensorRT [ceccocats/tkDNN#59](#) (comment)
- 42 FPS - on CPU Core i7 7700HQ (4 Cores / 8 Logical Cores) - (416x416, fp16, batch=1) OpenCV 4.4.0 (compiled with OpenVINO backend) [#6067](#) (comment)
- 20 FPS on CPU ARM Kirin 990 - Smartphone Huawei P40 [#6091](#) (comment) - Tencent/NCNN library <https://github.com/Tencent/ncnn>
- 120 FPS on nVidia Jetson AGX Xavier - MAX_N - Darknet framework
- 371 FPS on GPU GTX 1080 Ti - Darknet framework

Hình 14.10: YOLO-Tinyv4 trên các nền tảng (source)

Darknet

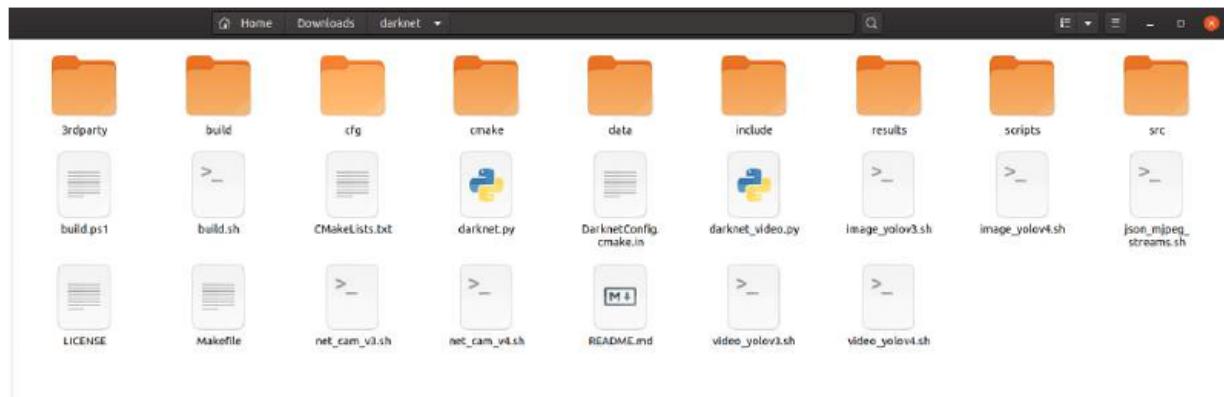
Darknet là một framework open source chuyên về Object Detection được viết bằng ngôn ngữ C và CUDA. Darknet dùng để huấn luyện các mô hình YOLO một cách nhanh chóng, dễ sử dụng.

Cấu hình darknet

Việc cấu hình trực tiếp trên Google Colab tương đối khó khăn với những bạn chưa quen làm việc với linux, bên cạnh đó vẫn đề delay kết nối giữa Google Colab với drive trong một số trường hợp. Vì vậy tôi sẽ hướng dẫn các bạn cấu hình trên máy cá nhân, xong nén thành file .zip, sau đó đẩy lên Google Drive.

Việc cấu hình trải qua các bước:

Bước 1: Clone **darknet** về máy.



Hình 14.11: Nội dung trong thư mục vừa clone về.

Bước 2: Sửa file Makefile trong thư mục vừa clone về.

[AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection \(Windows and Linux version of Darknet\) \(github.com\)](https://github.com/AlexeyAB/darknet)

```
GPU=1
CUDNN=1
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
```

Hình 14.12: Sửa một vài dòng đầu trong file.

Trong trường hợp huấn luyện mô hình trên GPU có TensorCores thì bạn đọc có thể sửa "**CUDNN_HALF=1**" để tăng tốc độ huấn luyện. Trên Google Colab có hỗ trợ GPU Tesla T4, là GPU có kiến trúc mới nhất trên Colab và có TensorCores. Vì vậy, trong trường hợp chắc chắn sẽ có được GPU Tesla T4 thì bạn đọc có thể sửa "**CUDNN_HALF=1**".

Bước 3: Tạo file yolo-tinyv4-obj.cfg.

Tạo file yolo-tinyv4-obj.cfg với nội dung tương tự file yolov4-tiny.cfg trong thư mục darknet/cfg, sau đó chỉnh sửa một số dòng:

- Dòng 6: Thay đổi batch=64. Nghĩa là: batch = số ảnh (cả file annotation) được đưa vào huấn luyện trong một batch.
- Dòng 7: Thay đổi subdivisions=16. Trong một batch được chia thành nhiều block, mỗi block chứa batch/subdivisions ảnh được đưa vào GPU xử lý tại một thời điểm. Weights của mô hình được update sau mỗi batch.
- Dòng 20: Thay đổi max_batches=classes*2000, không nhỏ hơn số ảnh trong tập huấn luyện, và không nhỏ hơn 6000 (theo [đây](#)). VD: max_batches=6000
- Dòng 22: Thay đổi steps= 80%, 90% max_batches. VD: steps=4800,5400. Sau khi huấn luyện được 80%, 90% max_batches, learning_rate sẽ được nhân với một tỷ lệ (dòng 23 trong file), mặc định là 0.1.
- Thay đổi classes=1 trong mỗi layer [yolo], dòng 217, 266.
- Thay đổi filters trong mỗi layer [convolutional] trước layer [yolo] theo công thức filters=(số class+5)*3. Trong bài toán này filters=18.

Chi tiết ý nghĩa của các tham số trong file cfg, bạn đọc có thể xem tại [đây](#).

Bước 4: Tạo file obj.names chứa tên của các class, sau đó lưu trong thư mục darknet/data.



Bước 5: Tạo file obj.data, sau đó lưu trong thư mục darknet/data.

```
classes= 1
train  = data/train.txt
valid  = data/valid.txt
names  = data/obj.names
backup = backup/
```

Hình 14.14: Nội dung file obj.data

Bước 6: Đưa toàn bộ thư mục chứa ảnh và file annotation ở trên vào thư mục darknet/data.

Bước 7: Download pre-trained weights của YOLO-Tinyv4 tại [đây](#), lưu trong thư mục darknet.

Bước 8: Nén thư mục darknet thành file darknet.zip, sau đó đưa lên Google Drive.

https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29

Huấn luyện model trên colab

Để thực thi các lệnh command line trong colab sử dụng thêm ! trước mỗi câu lệnh.

Kiểm tra cấu hình GPU dùng command line: !nvidia-smi

```
[1] !nvidia-smi

[Mon Jul 27 16:06:42 2020]
+-----+
| NVIDIA-SMI 450.51.05      Driver Version: 418.67      CUDA Version: 10.1 |
+-----+
GPU  Name      Persistence-M	Bus-Id     Disp.A  Volatile Uncorr. ECC		
Fan  Temp  Perf  Pwr:Usage/Cap	Memory-Usage	GPU-Util  Compute M.	
			MIG M.
+-----+			
0  Tesla P100-PCIE... Off  00000000:00:04.0 Off   0MiB / 16280MiB	0%      Default		
N/A   42C    P0    27W / 250W			ERR!
+-----+			
+-----+			
Processes:			
GPU  GI  CI      PID  Type  Process name        GPU Memory			
ID  ID              ID               Usage			
+-----+			
No running processes found			
+-----+
```

Hình 14.15: GPU được cung cấp là Tesla P100

```
#Sau khi mount với drive. Chuyển đến thư mục chứa file darknet.zip vừa tải lên
#Ví dụ tôi để ở thư mục gốc của Google Drive
cd drive/My\ Drive
```

```
#Giải nén file darknet.zip
!unzip darknet.zip
```

```
#Chuyển đến thư mục darknet
cd darknet
```

```
#Tạo thư mục backup để lưu lại weights khi huấn luyện
#Tên thư mục phải trùng với link folder backup trong file obj.data trên
!mkdir backup

#Tạo file train.txt, valid.txt theo đoạn code
import os
import numpy as np
#"obj" là tên thư mục chứa cả ảnh và file annotation.
lst_files = os.listdir("data/obj/")
lst_images = []

for file in lst_files:
    if ".txt" not in file:
        lst_images.append(file)

#Tách 200 ảnh ra làm tập validation
random_idx = np.random.randint(0, len(lst_images), 200)

#Tạo file train.txt được đặt trong thư mục darknet/data
with open("data/train.txt", "w") as f:
    for idx in range(len(lst_images)):
        if idx not in random_idx:
            f.write("data/obj/" + lst_images[idx] + "\n")

#Tạo file valid.txt được đặt trong thư mục darknet/data
with open("data/valid.txt", "w") as f:
    for idx in random_idx:
        f.write("data/obj/" + lst_images[idx] + "\n")

#Biên dịch darknet (chỉ cần biên dịch một lần, lần sau dùng bỏ qua bước này)
!make
```

```
#Phân quyền thực thi module darknet  
!chmod +x ./darknet
```

Bắt đầu quá trình huấn luyện sử dụng command line:

```
!./darknet detector train data/obj.data yolo-tinyv4-obj.cfg yolov4-tiny.conv.29 -map \  
-dont_show > yolotinyv4_lisenceplate.log
```

Cú pháp tổng quát để huấn luyện:

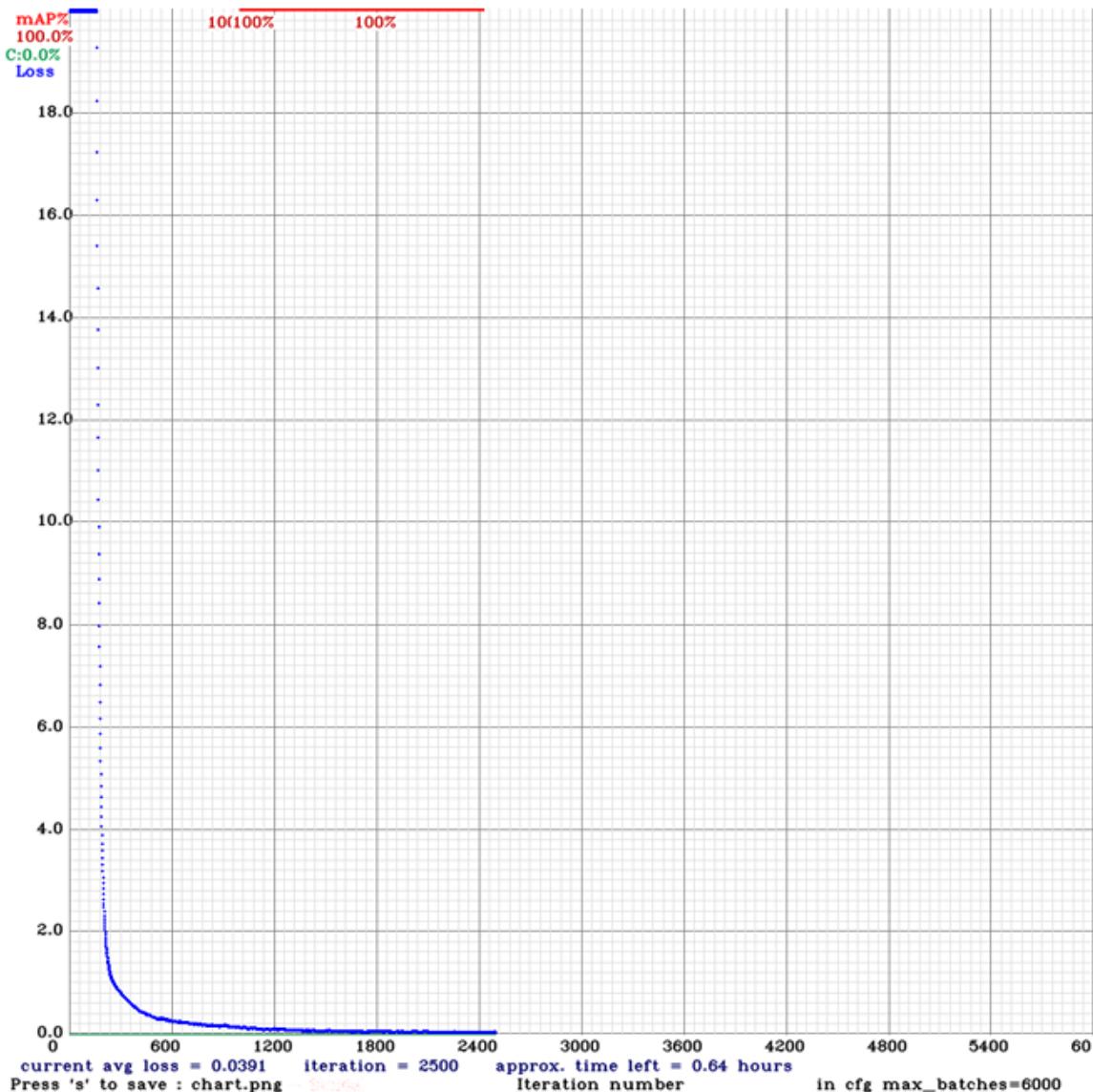
```
!./darknet detector train [data config file] [model config file] [pre-trained weights]  
-map > [file log saved]
```

-map: Dùng để hiển thị mAP được tính trên tập validation.

Nếu bạn gặp lỗi: **CUDA Error: out of memory: File exists** thì hãy quay lại sửa subdivisions=32 trong file yolo-tinyv4-obj.cfg

Theo dõi quá trình huấn luyện

Quá trình huấn luyện sẽ được lưu vào file yolotinv4_lisenceplate.log, ngoài ra darknet tự động tạo ra ảnh chart.png lưu trong thư mục darknet và được cập nhật liên tục để theo dõi trực tiếp thông số của quá trình huấn luyện.



Hình 14.16: chart.png

Nhận xét: Quá trình huấn luyện hội tụ rất nhanh. Vì vậy có thể dừng sớm sau 2000 batches.

Dự đoán

Sau khi huấn luyện xong, toàn bộ weights sẽ được lưu trong folder backup.

#Danh sách các weights được lưu

```
!ls backup/
```

```
yolo-tinyv4-obj_1000.weights  yolo-tinyv4-obj_best.weights  
yolo-tinyv4-obj_2000.weights  yolo-tinyv4-obj_last.weights
```

Để dự báo một bức ảnh sử dụng cú pháp:

```
!./darknet detector test [data config file] [model config file] [best-weights]  
[image path]  
#cụ thể như sau  
!./darknet detector test data/obj.data yolo-tinyv4-obj.cfg \  
backup/yolo-tinyv4-obj_best.weights test1.jpg
```

Kết quả dự đoán được lưu thành file predictions.jpg

```
#Hàm sau được dùng để hiển thị kết quả dự đoán lên colab  
def show(path):  
    import cv2  
    import matplotlib.pyplot as plt  
  
    image = cv2.imread(path)  
    original_width, original_height = image.shape[1], image.shape[0]  
    resized_image = cv2.resize(image, (2*original_width, 2*original_height))\  
    , interpolation = cv2.INTER_CUBIC)  
  
    resized_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)  
    plt.figure(figsize=(20,10))  
    plt.axis("off")  
    plt.imshow(resized_image)  
    plt.show()  
  
show("predictions.jpg")
```



Hình 14.17: Kết quả dự đoán

Code và dataset đã gán nhãn mọi người có thể lấy ở [đây](#).

15. Image segmentation với U-Net (Phân đoạn hình ảnh với U-Net)

Bài toán image segmentation

Bài trước bạn đã được giới thiệu về object detection, đi tìm các bounding box quanh các đối tượng trong ảnh và sau đó phân loại các bounding box. Tuy nhiên là các bounding box thì không biểu thị được đúng hình dạng của đối tượng và có nhiều nhiễu ở trong bounding box đấy ví dụ như trong bounding box màu đỏ có cả một phần của cây thông cũng như cái gối => Image segmentation ra đời để chia ảnh thành nhiều vùng khác nhau hay tìm được đúng hình dạng của các đối tượng.

Object Detection



Instance Segmentation



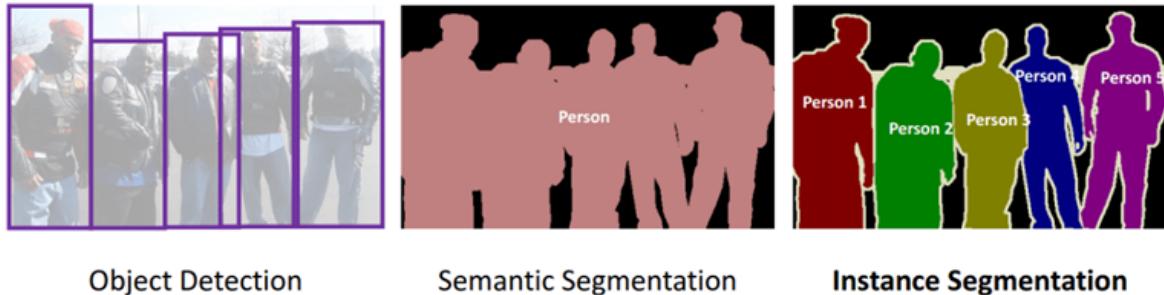
Phân loại bài toán image segmentation

Semantic segmentation (Phân đoạn ngữ nghĩa)

Instance segmentation (Phân đoạn phiên bản)

Bài toán image segmentation được chia ra làm 2 loại:

- **Semantic segmentation:** Thực hiện segment với từng lớp khác nhau, ví dụ: tất cả người là 1 lớp, tất cả ô tô là 1 lớp.
- **Instance segmentation:** Thực hiện segment với từng đối tượng trong một lớp. Ví dụ có 3 người trong ảnh thì sẽ có 3 vùng segment khác nhau cho mỗi người.



Hình 15.3: Phân loại semantic segmentation và instance segmentation [27]

Cần áp dụng kiểu segmentation nào thì phụ thuộc vào bài toán. Ví dụ: cần segment người trên đường cho ô tô tự lái, thì có thể dùng semantic segmentation vì không cần thiết phải phân biệt ai với ai, nhưng nếu cần theo dõi mọi hành vi của mọi người trên đường thì cần instance segmentation thì cần phân biệt mọi người với nhau.

Ứng dụng bài toán segmentation

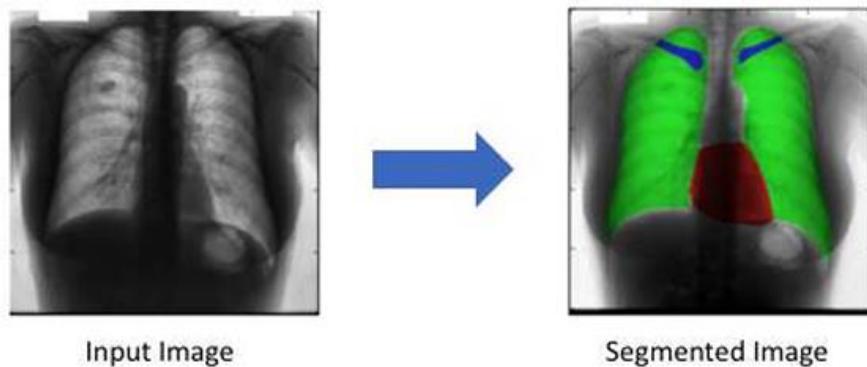
Ô tô tự lái

Segmentation dùng để xác định đường, các xe ô tô, người đi bộ,... để hỗ trợ cho ô tô tự lái



Chẩn đoán trong y học

Segmentation được ứng dụng rất nhiều trong y học để hỗ trợ việc chẩn đoán bệnh. Ví dụ phân tích ảnh X-quang.



Hình 15.4: Ứng dụng segmentation [16]

Mạng U-Net với bài toán semantic segmentation

Như trong bài xử lý ảnh ta đã biết thì ảnh bản chất là một ma trận của các pixel. Trong bài toán image segmentation, ta cần phân loại mỗi pixel trong ảnh. Ví dụ như trong hình trên với semantic segmentation, với mỗi pixel trong ảnh ta cần xác định xem nó là background hay là người. Thêm nữa là ảnh input và output có cùng kích thước.

U-Net được phát triển bởi Olaf Ronneberger et al. để dùng cho image segmentation trong y học. Kiến trúc có 2 phần đối xứng nhau được gọi là encoder (phần bên trái) và decoder (phần bên phải).

Kiến trúc mạng U-Net

2

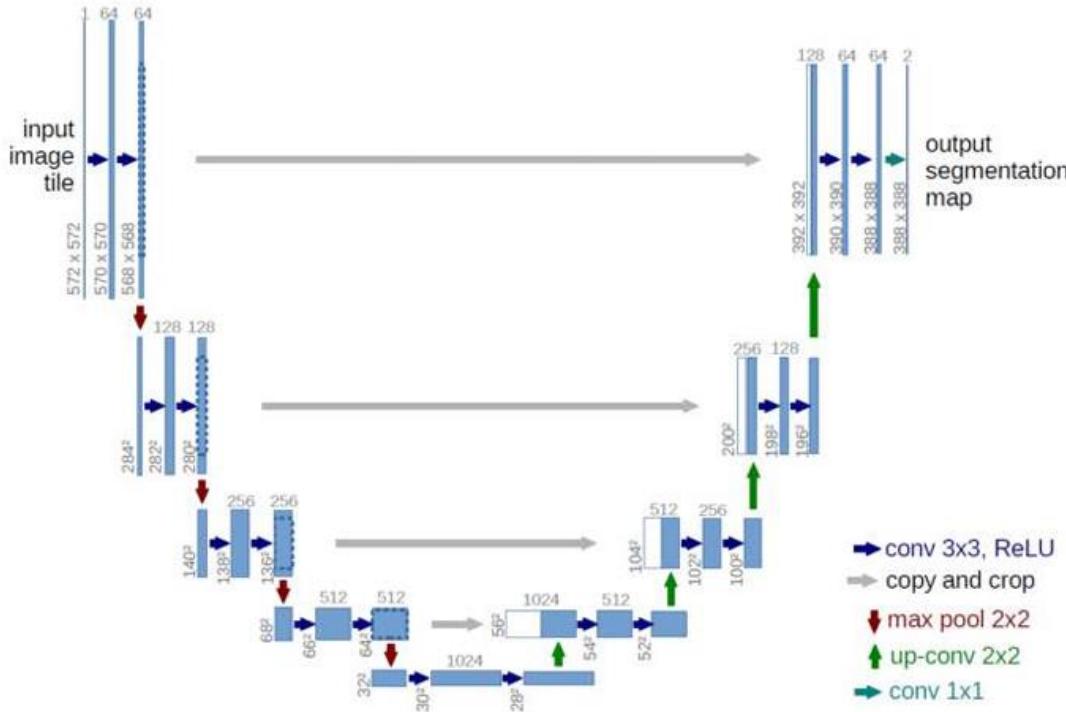


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Nhận xét:

- Thực ra phần encoder chỉ là ConvNet bình thường (conv, max pool) với quy tắc quen thuộc từ bài VGG, các layer sau thì width, height giảm nhưng depth tăng.
- Phần decoder có mục đích là khôi phục lại kích thước của ảnh gốc, ta thấy có up-conv là Conv với stride > 1 để giảm kích thước của ảnh giống như max pool, thì up-conv dùng để tăng kích thước của ảnh.
- Bạn thấy các đường màu xám, nó nối layer trước với layer hiện tại được dùng rất phổ biến

trong các CNN ngày nay như DenseNet để tránh vanishing gradient cũng như mang được các thông tin cần thiết từ layer trước tới layer sau.

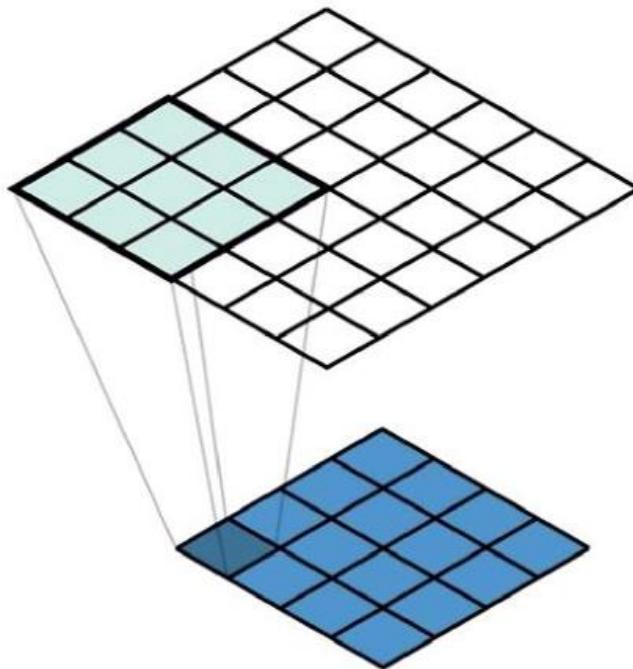
Loss function

Vì bài toán là phân loại cho mỗi pixel nên loss function sẽ là tổng cross-entropy loss cho mỗi pixel trong toàn bộ bức ảnh.

Transposed convolution

Hình ở trên có kích thước là 6*6, hình ở dưới có kích thước là 4*4, kernel có kích thước 3*3.

Nếu ta thực hiện phép tính convolution với input là hình ở trên, padding = 0, stride = 1 và kernel 3*3 thì output sẽ là hình ở dưới.



Phép tính transposed convolution thì sẽ ngược lại, input là hình ở dưới, padding = 0, stride = 1 và kernel 3*3 thì output sẽ là hình ở trên. Các ô vuông ở hình trên bị đè lên nhau thì sẽ được cộng dồn. Các quy tắc về stride và padding thì tương tự với convolution.

Mọi người có thể xem thêm ở [đây](#).

[vdumoulin/conv_arithmetic: A technical report on convolution arithmetic in the context of deep learning \(github.com\)](#)

Code

Nhận xét:

- Đây là phần code để tạo model, mô hình encoder và decoder đối xứng nhau hoàn toàn.
 - Các conv layer đều dùng 3*3 và padding là same để giữ nguyên kích thước.
 - Max pooling kích thước 2*2, mỗi lần pooling width, height giảm 1 nửa, nhưng layer sau đó depth tăng gấp đôi.
 - Up sampling kích thước 2*2, mỗi lần pooling width, height tăng gấp đôi, nhưng layer sau đó depth giảm một nửa.
 - Concatenate dùng để nối layer đối xứng ở encoder với layer hiện tại có cùng kích thước.
 - Dropout cũng được sử dụng để tránh overfitting
-
- Relu activation được dùng trong các layer trừ output layer dùng sigmoid.
 - Như bạn biết sigmoid dùng cho binary classification tuy nhiên nếu bạn output ra nhiều ra giá trị (ví dụ cả 1 ma trận trong bài này) và để loss là binary_crossentropy thì Keras sẽ hiểu loss function là tổng của binary_crossentropy của từng pixel trong ảnh.

```
import numpy as np
import os
import skimage.io as io
import skimage.transform as trans
import numpy as np
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras

def unet(pretrained_weights = None,input_size = (256,256,1)):
    inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs)
    conv1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1)
```

```

conv2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2)
conv3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3)
pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3)
conv4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4)
drop4 = Dropout(0.5)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4)
conv5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5)
drop5 = Dropout(0.5)(conv5)

up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(drop5))
merge6 = concatenate([drop4,up6], axis = 3)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6)
conv6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6)

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv6))
merge7 = concatenate([conv3,up7], axis = 3)
conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7)
conv7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7)

up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv7))
merge8 = concatenate([conv2,up8], axis = 3)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8)
conv8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8)

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv8))
merge9 = concatenate([conv1,up9], axis = 3)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9)
conv9 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv10 = Conv2D(1, 1, activation = 'sigmoid')(conv9)

model = Model(input = inputs, output = conv10)

model.compile(optimizer = Adam(lr = 1e-4), loss = 'binary_crossentropy', metrics = ['accuracy'])

#model.summary()

```

```
if(pretrained_weights):
    model.load_weights(pretrained_weights)

return model
```

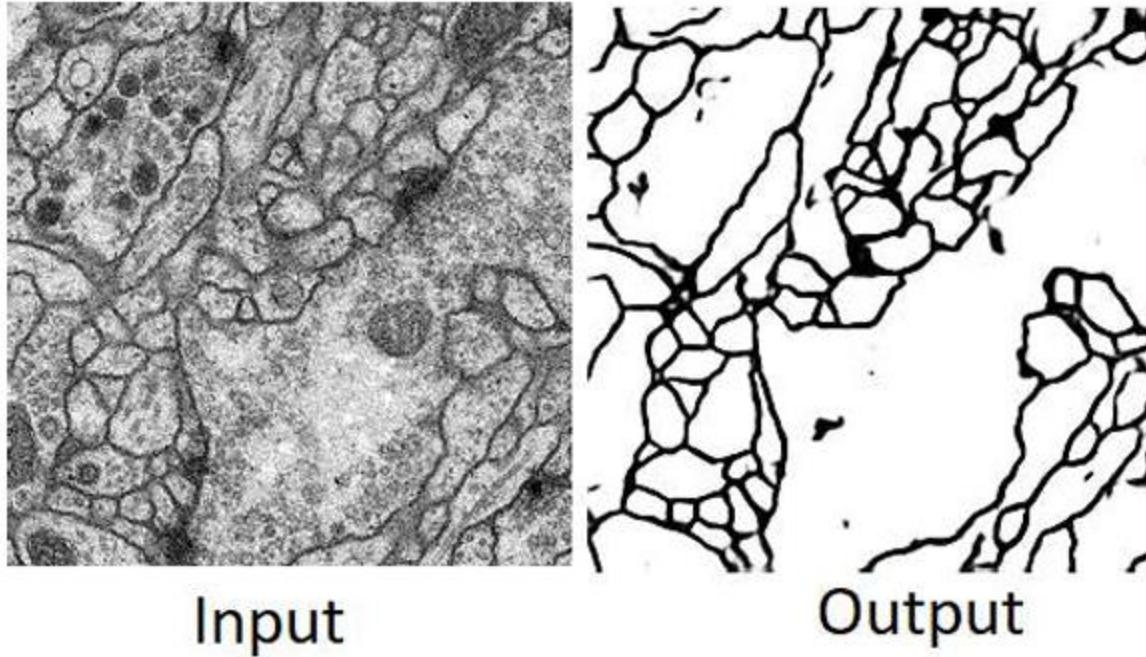
Ngoài ra thì ảnh cũng được scale về [0,1] và mask cũng đưa về 0 và 1 với bài toán segmentation chỉ gồm 1 lớp và background

```
img = img / 255
mask = mask /255
mask[mask > 0.5] = 1
mask[mask <= 0.5] = 0
```

Bên cạnh đó thì data augmentation cũng được sử dụng để tăng lượng ảnh cho training set

```
data_gen_args = dict(rotation_range=0.2,
                      width_shift_range=0.05,
                      height_shift_range=0.05,
                      shear_range=0.05,
                      zoom_range=0.05,
                      horizontal_flip=True,
                      fill_mode='nearest')
```

Kết quả sau khi train model



Phần 7 : Recurrent Neural Network (RNN) (Mạng nơ ron tái phát)

Deep learning có 2 mô hình lớn là Convolutional Neural Network (CNN) cho bài toán có input là ảnh và Recurrent neural network (RNN) cho bài toán dữ liệu dạng chuỗi (sequence). Tôi đã giới thiệu về Convolutional Neural Network (CNN) và các ứng dụng của deep learning trong computer vision bao gồm: classification, object detection, segmentation. Có thể nói là tương đối đầy đủ các dạng bài toán liên quan đến CNN. Bài này tôi sẽ giới thiệu về RNN.

16. Recurrent neural network

Đọc tài liệu : [lecture_8-Recurrent Neural Networks.pdf](#) (chi tiết , hay)

Recurrent Neural Network là gì?

Bài toán: Cần phân loại hành động của người trong video, input là video 30s, output là phân loại hành động, ví dụ: đứng, ngồi, chạy, đánh nhau, bắn súng....

Khi xử lý video ta hay gặp khái niệm FPS (frame per second) tức là bao nhiêu frame (ảnh) mỗi giây. Ví dụ 1 FPS với video 30s tức là lấy ra từ video 30 ảnh, mỗi giây một ảnh để xử lý.

Ta dùng 1 FPS cho video input ở bài toán trên, tức là lấy ra 30 ảnh từ video, ảnh 1 ở giây 1, ảnh 2 ở giây 2,... ảnh 30 ở giây 30. Bây giờ input là 30 ảnh: ảnh 1, ảnh 2,... ảnh 30 và output là phân loại hành động. Nhận xét:

- Các ảnh có thứ tự: ảnh 1 xảy ra trước ảnh 2, ảnh 2 xảy ra trước ảnh 3,... **Nếu ta đảo lộn các ảnh thì có thể thay đổi nội dung của video.** Ví dụ: nội dung video là cảnh bắn nhau, thứ tự đúng là A bắn trúng người B và B chết, nếu ta đảo thứ tự ảnh thành người B chết xong A mới bắn thì rõ ràng bây giờ A không phải là kẻ giết người => nội dung video bị thay đổi.
- Ta có thể dùng CNN để phân loại 1 ảnh trong 30 ảnh trên, nhưng rõ ràng là 1 ảnh không thể mô tả được nội dung của cả video. Ví dụ: Cảnh người cướp điện thoại, nếu ta chỉ dùng 1 ảnh là người đấy cầm điện thoại lúc cướp xong thì ta không thể biết được cả hành động cướp.

=> Cần một mô hình mới có thể giải quyết được bài toán với input là sequence (chuỗi ảnh 1->30)
=> Recurrent Neural Network (RNN) ra đời.

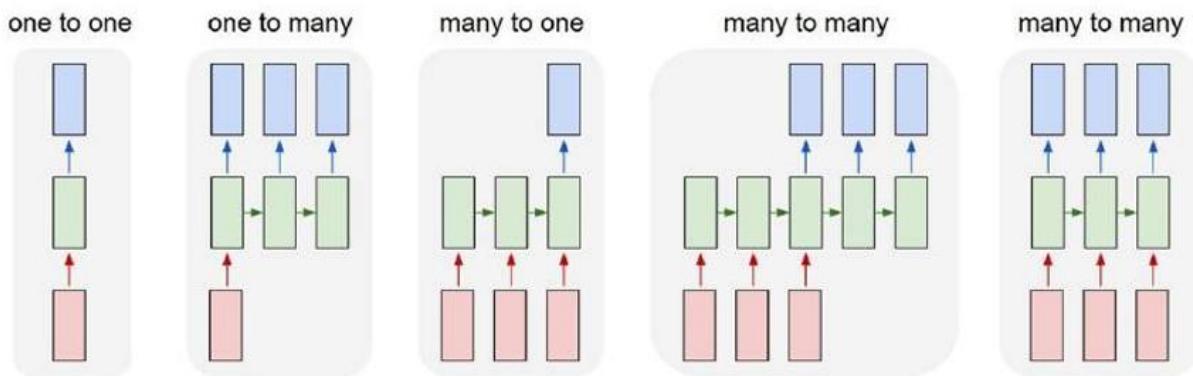
Dữ liệu dạng sequence

Dữ liệu có thứ tự như các ảnh tách từ video ở trên được gọi là sequence, time-series data.

Trong bài toán dự đoán đột quy tim cho bệnh nhân bằng các dữ liệu tim mạch khám trước đó. Input là dữ liệu của những lần khám trước đó, ví dụ i1 là lần khám tháng 1, i2 là lần khám tháng 2,... i8 là lần khám tháng 8. (i1,i2,..i8) **được gọi là sequence data.** RNN sẽ học từ input và dự đoán xem bệnh nhân có bị đột quy tim hay không.

Ví dụ khác là trong bài toán dịch tự động với input là 1 câu, ví dụ "tôi yêu Việt Nam" thì vị trí các từ và sự sắp xếp cực kì quan trọng đến nghĩa của câu và dữ liệu input các từ ['tôi', 'yêu', 'việt', 'nam'] được gọi là **sequence data.** **Trong bài toán xử lý ngôn ngữ (NLP) thì không thể xử lý cả câu được và người ta tách ra từng từ (chữ) làm input, giống như trong video người ta tách ra các ảnh (frame) làm input.**

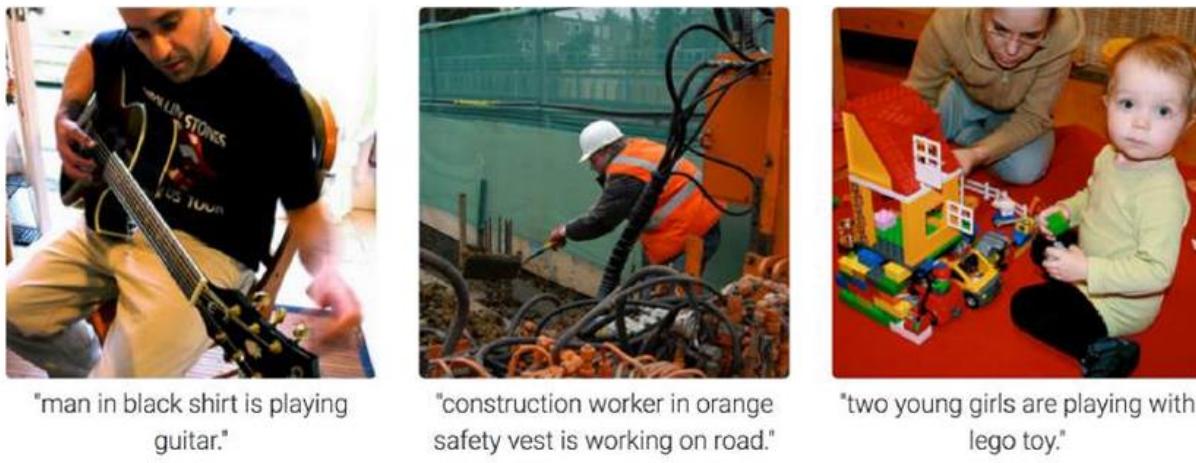
Phân loại bài toán RNN



Hình 16.1: Các dạng bài toán RNN

One to one: mẫu bài toán cho Neural Network (NN) và Convolutional Neural Network (CNN), 1 input và 1 output, ví dụ với bài toán phân loại ảnh MNIST input là ảnh và output là ảnh đấy là số nào.

One to many: bài toán có 1 input nhưng nhiều output, ví dụ với bài toán caption cho ảnh, input là 1 ảnh nhưng output là nhiều chữ mô tả cho ảnh đấy, dưới dạng một câu.



Hình 16.2: Ví dụ image captioning [10]

Many to one: bài toán có nhiều input nhưng chỉ có 1 output, ví dụ bài toán phân loại hành động trong video, input là nhiều ảnh (frame) tách ra từ video, output là hành động trong video

Many to many: bài toán có nhiều input và nhiều output, ví dụ bài toán dịch từ tiếng anh sang tiếng việt, input là 1 câu gồm nhiều chữ: "I love Vietnam" và output cũng là 1 câu gồm nhiều chữ "Tôi yêu Việt Nam". Để ý là độ dài sequence của input và output có thể khác nhau.

⇒ **Many to Many** (ví dụ bài toán seq2seq)

- ⇒ **Bài toán nhận dạng khuôn mặt có thể là one to one (ảnh và nhãn)**
- ⇒ **Hoặc Many to one (video và nhãn)**

Ứng dụng bài toán RNN

Về cơ bản nếu bạn thấy sequence data hay time-series data và bạn muốn áp dụng deep learning thì bạn nghĩ ngay đến RNN. Dưới đây là một số ứng dụng của RNN:

- **Speech to text:** Chuyển giọng nói sang text.
- **Sentiment classification:** Phân loại bình luận của người dùng, tích cực hay tiêu cực.
- **Machine translation:** Bài toán dịch tự động giữa các ngôn ngữ.
- **Video recognition:** Nhận diện hành động trong video.
- **Heart attack:** Dự đoán đột quỵ tim.

Mô hình bài toán RNN [Quan trọng]

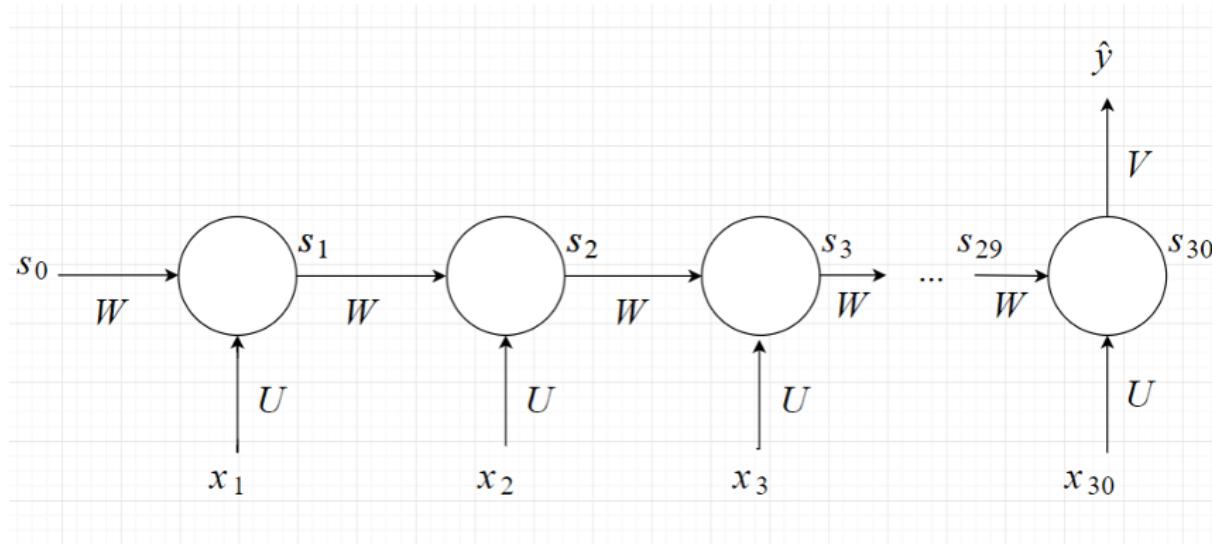
Mô hình RNN [Quan trọng]

Lưu ý phần này

Bài toán: Nhận diện hành động trong video 30s. Đây là dạng bài toán many to one trong RNN, tức nhiều input và 1 output.

Input ta sẽ tách video thành 30 ảnh (mỗi giây một ảnh). Các ảnh sẽ được cho qua pretrained model CNN để lấy ra các feature (feature extraction) vector có kích thước $n \times 1$. Vector tương ứng với ảnh ở giây thứ i là x_i .

Output là vector có kích thước $d \times 1$ (d là số lượng hành động cần phân loại), softmax function được sử dụng như trong bài phân loại ảnh.



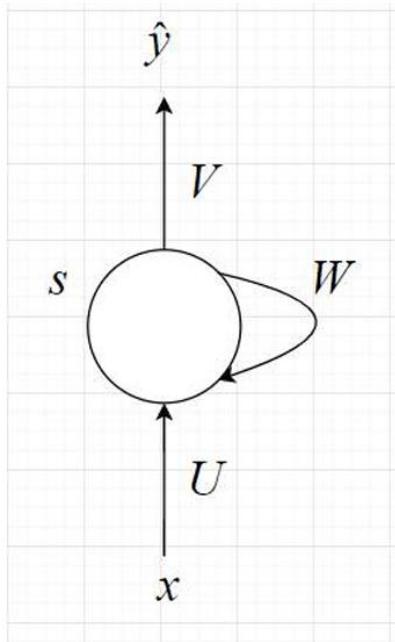
Hình 16.3: Mô hình RNN cho bài toán.

Ta có:

- Mô hình có 30 input và 1 output, các input được cho vào model đúng với thứ tự ảnh trong video x_1, x_2, \dots, x_{30} .

- Mỗi hình tròn được gọi là 1 state, state t có input là x_t và s_{t-1} (output của state trước); output là $s_t = f(U * x_t + W * s_{t-1})$. f là activation function thường là Tanh hoặc ReLU.
- Có thể thấy s_t mang cả thông tin từ state trước (s_{t-1}) và input của state hiện tại $\Rightarrow s_t$ giống như memory nhớ các đặc điểm của các input từ x_1 đến x_t .
- s_0 được thêm vào chỉ cho chuẩn công thức nên thường được gán bằng 0 hoặc giá trị ngẫu nhiên. Có thể hiểu là ban đầu chưa có dữ liệu gì để học thì memory rỗng.
- Do ta chỉ có 1 output, nên sẽ được đặt ở state cuối cùng, khi đó s_{30} học được thông tin từ tất cả các input. $\hat{y} = g(V * s_{30})$. g là activation function, trong bài này là bài toán phân loại nên sẽ dùng softmax.

Ta thấy là ở mỗi state các hệ số W, U là giống nhau nên model có thể được viết lại thành:



Hình 16.4: Mô hình RNN rút gọn

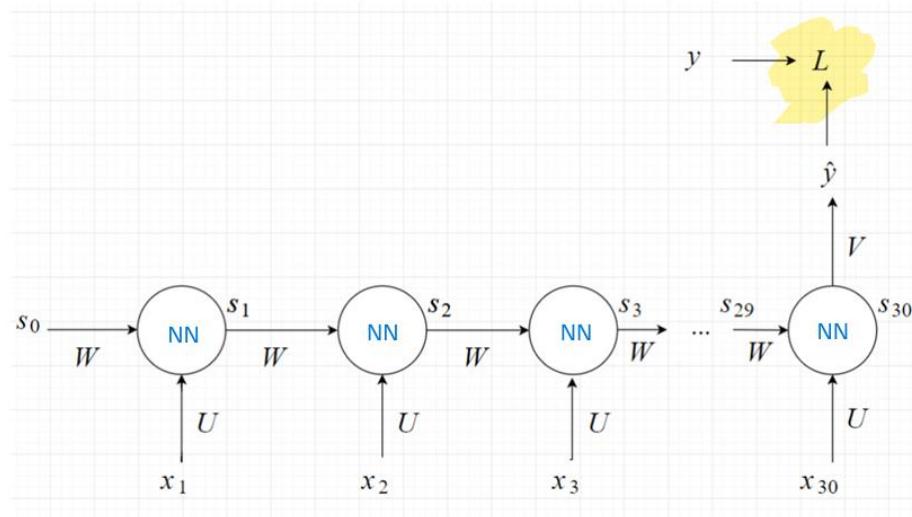
⇒ Do quá trình cập nhật vào vào lại như nhau , quá trình này lặp lại với W và U như nhau nên rút gọn lại ta có hình như trên .

Tóm lại:

- x_t là vector có kích thước $n*1$, s_t là vector có kích thước $m*1$, y_t là vector có kích thước $d*1$.
U là ma trận có kích thước $m*n$, W là ma trận có kích thước $m*m$ và V là ma trận có kích thước $d*m$.
- $s_0 = 0, s_t = f(U * x_t + W * s_{t-1})$ với $t \geq 1$
- $\hat{y} = g(V * s_{30})$

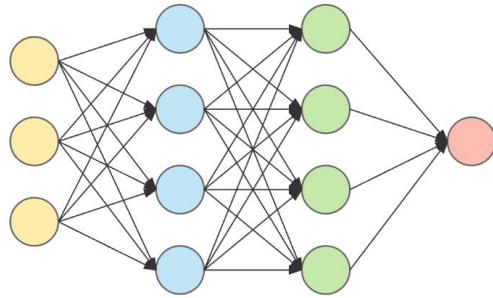
Loss function

Loss function của cả mô hình bằng tổng loss của mỗi output, tuy nhiên ở mô hình trên chỉ có 1 output và là bài toán phân loại nên categorical cross entropy loss sẽ được sử dụng.



Hình 16.5: Loss function

⇒ Mỗi node trong mạng RNN này chính là mỗi mạng nơ ron network (**NN**)



Backpropagation Through Time (BPTT)

Có 3 tham số ta cần phải tìm là W, U, V. Để thực hiện gradient descent, ta cần tính: $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$.

Tính đạo hàm với V thì khá đơn giản:

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial V}$$

Tuy nhiên với U, W thì lại khác.

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial W}$$

Do $s_{30} = f(W * s_{29} + U * x_{30})$ có s_{29} phụ thuộc vào W. Nên áp dụng công thức hồi cấp 3 bạn học: $(f(x) * g(x))' = f'(x) * g(x) + f(x) * g'(x)$. Ta có

$\frac{\partial s_{30}}{\partial W} = \frac{\partial s'_{30}}{\partial W} + \frac{\partial s_{30}}{\partial s_{29}} * \frac{\partial s_{29}}{\partial W}$, trong đó $\frac{\partial s'_{30}}{\partial W}$ là đạo hàm của s_{30} với W khi coi s_{29} là constant với W.

Tương tự trong biểu thức s_{29} có s_{28} phụ thuộc vào W, s_{28} có s_{27} phụ thuộc vào W ... nên áp dụng công thức trên và chain rule:

$\frac{\partial L}{\partial W} = \sum_{i=0}^{30} \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial s_i} * \frac{\partial s'_i}{\partial W}$, trong đó $\frac{\partial s_{30}}{\partial s_i} = \prod_{j=i}^{29} \frac{\partial s_{j+1}}{\partial s_j}$ và $\frac{\partial s'_i}{\partial W}$ là đạo hàm của s_i với W khi coi s_{i-1} là constant với W.

Lưu ý những ý sau :

Nhìn vào công thức tính đạo hàm của L với W ở trên ta có thể thấy hiện tượng vanishing gradient ở các state đầu nên ta cần mô hình tốt hơn để giảm hiện tượng vanishing gradient => Long short term memory (LSTM) ra đời và sẽ được giới thiệu ở bài sau. Vì trong bài toán thực tế liên quan đến

time-series data thì LSTM được sử dụng phổ biến hơn là mô hình RNN thuần nên bài này không có code, bài sau sẽ có code ứng dụng với LSTM.

Bài tập

[Bitcoin Historical Data \(kaggle.com\)](https://www.kaggle.com)

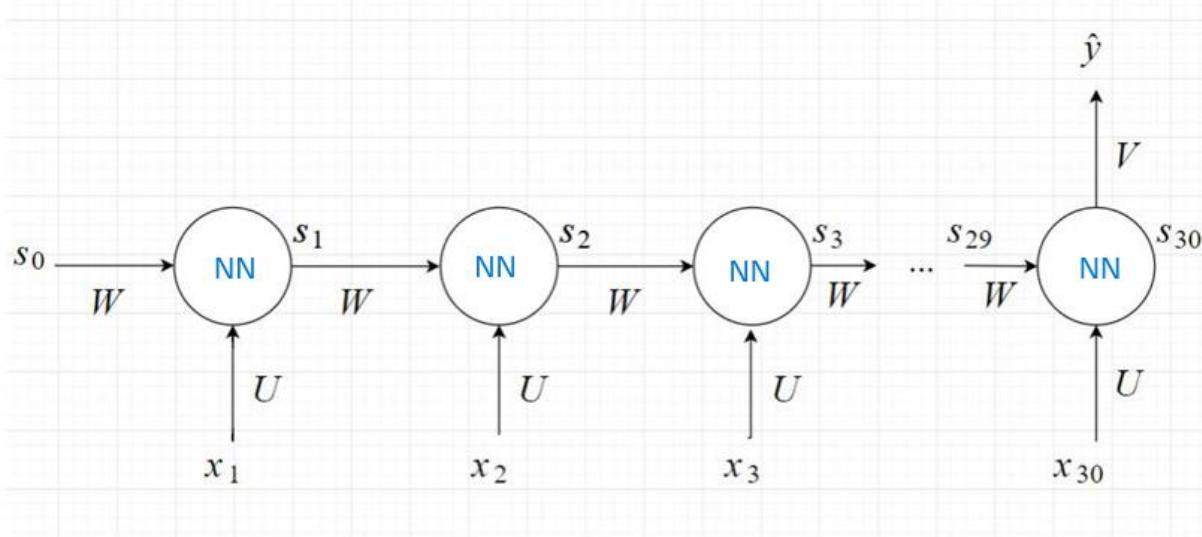
1. Hệ số trong RNN là gì?
2. Thiết kế và train model RNN dự báo giá Bitcoin, tải dữ liệu ở [đây](#).
3. Tự tìm hiểu và sử dụng mô hình Bidirectional cho bài toán trên.

17. Long short term memory (LSTM)

Giới thiệu về LSTM

LSTM (Long short term memory) (Bộ nhớ dài ngắn hạn)

Bài trước tôi đã giới thiệu về recurrent neural network (RNN). RNN có thể xử lý thông tin dạng chuỗi (sequence/time-series). Như ở bài dự đoán hành động trong video ở bài trước, RNN có thể mang thông tin của frame (ảnh) từ state trước tới các state sau, rồi ở state cuối là sự kết hợp của tất cả các ảnh để dự đoán hành động trong video.

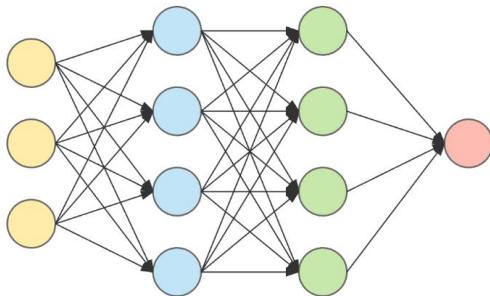


Hình 17.1: Mô hình RNN

Đạo hàm của L với W ở state thứ i: $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s_{30}} * \frac{\partial s_{30}}{\partial s_i} * \frac{\partial s'_i}{\partial W}$, trong đó $\frac{\partial s_{30}}{\partial s_i} = \prod_{j=i}^{29} \frac{\partial s_{j+1}}{\partial s_j}$

Giả sử activation là tanh function, $s_t = \tanh(U * x_t + W * s_{t-1})$

\Rightarrow Mỗi node trong mạng RNN này chính là mỗi mạng nơ ron network (**NN**)



$$\frac{\partial s_t}{\partial s_{t-1}} = (1 - s_t^2) * W \Rightarrow \frac{\partial s_{30}}{\partial s_i} = W^{30-i} * \prod_{j=i}^{29} (1 - s_j^2).$$

Ta có $s_j < 1, W < 1 \Rightarrow$ Ở những state xa thì $\frac{\partial s_{30}}{\partial s_i} \approx 0$ hay $\frac{\partial L}{\partial W} \approx 0$, hiện tượng vanishing gradient

Ta có thể thấy là các state càng xa ở trước đó thì càng bị vanishing gradient và các hệ số không được update với các frame ở xa. Hay nói cách khác là RNN không học được từ các thông tin ở trước đó xa do vanishing gradient.

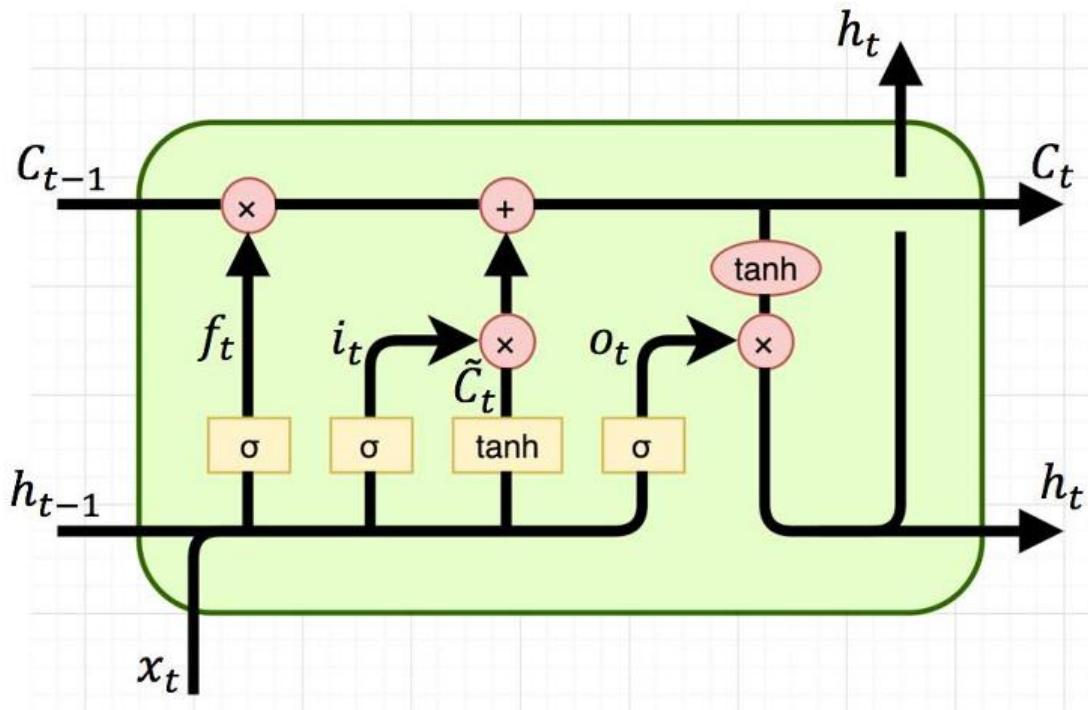
Như vậy về lý thuyết là RNN có thể mang thông tin từ các layer trước đến các layer sau, nhưng thực tế là thông tin chỉ mang được qua một số lượng state nhất định, sau đó thì sẽ bị vanishing gradient, hay nói cách khác là model chỉ học được từ các state gần nó \Rightarrow short term memory.

Cùng thử lấy ví dụ về short term memory nhé. Bài toán là dự đoán từ tiếp theo trong đoạn văn. Đoạn đầu tiên "Mặt trời mọc ở hướng ...", ta có thể chỉ sử dụng các từ trước trong câu để đoán là đông. Tuy nhiên, với đoạn, "Tôi là người Việt Nam. Tôi đang sống ở nước ngoài. Tôi có thể nói trôi chảy tiếng ..." thì rõ ràng là chỉ sử dụng từ trong câu đấy hoặc câu trước là không thể dự đoán được từ cần điền là Việt. Ta cần các thông tin từ state ở trước đó rất xa \Rightarrow cần long term memory điều mà RNN không làm được \Rightarrow Cần một mô hình mới để giải quyết vấn đề này \Rightarrow Long short term memory (LSTM) ra đời.

Mô hình LSTM

Ở state thứ t của mô hình LSTM:

- Output: c_t, h_t , ta gọi c là cell state, h là hidden state.
- Input: c_{t-1}, h_{t-1}, x_t . Trong đó x_t là input ở state thứ t của model. c_{t-1}, h_{t-1} là output của layer trước. h đóng vai trò khá giống như s ở RNN, trong khi c là điểm mới của LSTM.



Hình 17.2: Mô hình LSTM [25]

Trong đó :

- + Phép nhân là phép nhân element ma trận
- + Phép cộng là cộng ma trận

Cách đọc biểu đồ trên: bạn nhìn thấy kí hiệu σ , tanh ý là bước đầy dùng sigma, tanh activation function. Phép nhân ở đây là element-wise multiplication, phép cộng là cộng ma trận.

f_t, i_t, o_t tương ứng với forget gate, input gate và output gate.

- Forget gate: $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$
- Input gate: $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$
- Output gate: $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$

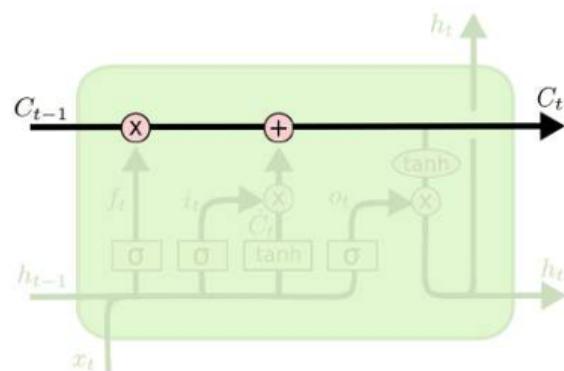
Nhận xét: $0 < f_t, i_t, o_t < 1$; b_f, b_i, b_o là các hệ số bias; hệ số W, U giống như trong bài RNN.

$\tilde{c}_t = \tanh(U_c * x_t + W_c * h_{t-1} + b_c)$, bước này giống hệt như tính s_t trong RNN.

$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$, **forget gate** quyết định xem cần lấy bao nhiêu từ cell state trước và **input gate** sẽ quyết định lấy bao nhiêu từ input của state và hidden layer của layer trước.

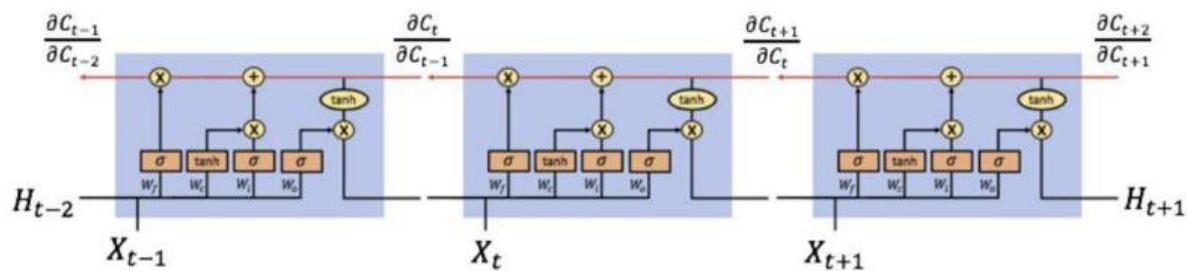
$h_t = o_t * \tanh(c_t)$, **output gate** quyết định xem cần lấy bao nhiêu từ cell state để trở thành output của hidden state. Ngoài ra h_t cũng được dùng để tính ra output y_t cho state t.

Nhận xét: h_t, \tilde{c}_t khá giống với RNN, nên model có short term memory. Trong khi đó c_t giống như một băng chuyền ở trên mô hình RNN vậy, thông tin nào cần quan trọng và dùng ở sau sẽ được gửi vào và dùng khi cần => có thể mang thông tin từ đi xa => long term memory. Do đó mô hình LSTM có cả short term memory và long term memory.



Hình 17.3: cell state trong LSTM

LSTM chống vanishing gradient



Hình 17.4: Mô hình LSTM [9]

Ta cũng áp dụng thuật toán back propagation through time cho LSTM tương tự như RNN.

NG

You

tại sao trong LSTM thì C_t lại fix cứng hàm tanh

∅

ChatGPT

Trong mạng LSTM (Long Short-Term Memory), việc sử dụng hàm kích hoạt tanh để định hình giá trị của bộ nhớ dài hạn C_t là một trong những tính năng chính của kiến trúc này. Có một số lý do cho quyết định này:

- Tính giới hạn của hàm tanh:** Hàm tanh có miền giá trị giữa -1 và 1, điều này giúp kiểm soát giá trị của C_t trong khoảng này, giảm thiểu vấn đề gradient exploding và vanishing trong quá trình lan truyền ngược.
- Xử lý dữ liệu đầu vào:** Dữ liệu đầu vào thường nằm trong khoảng giá trị tương tự như hàm tanh (-1 đến 1 hoặc 0 đến 1), việc sử dụng hàm tanh giúp mạng có khả năng học được mối quan hệ phức tạp giữa dữ liệu đầu vào và giá trị của C_t .
- Biểu diễn dữ liệu không bị giới hạn:** Hàm tanh không bị giới hạn trên đầu vào, điều này có nghĩa là mạng LSTM có khả năng học được biểu diễn của dữ liệu một cách tự nhiên mà không bị hạn chế bởi việc chọn phạm vi đầu vào.
- Thuận tiện cho việc kết hợp:** Việc sử dụng hàm tanh giúp cho việc tích hợp các phân tử trong LSTM trở nên dễ dàng hơn, bởi vì các phép toán nhân và cộng thường đi kèm với hàm tanh.

Tóm lại, việc sử dụng hàm tanh trong LSTM là một phần quan trọng của kiến trúc này, giúp nó có khả năng học được các mối quan hệ phức tạp trong dữ liệu và giải quyết vấn đề gradient exploding và vanishing.

+ Gradient exploding : Hội tụ chậm

+ Vanishing : Biến mất (mất mát) (dẫn đến overfitting)

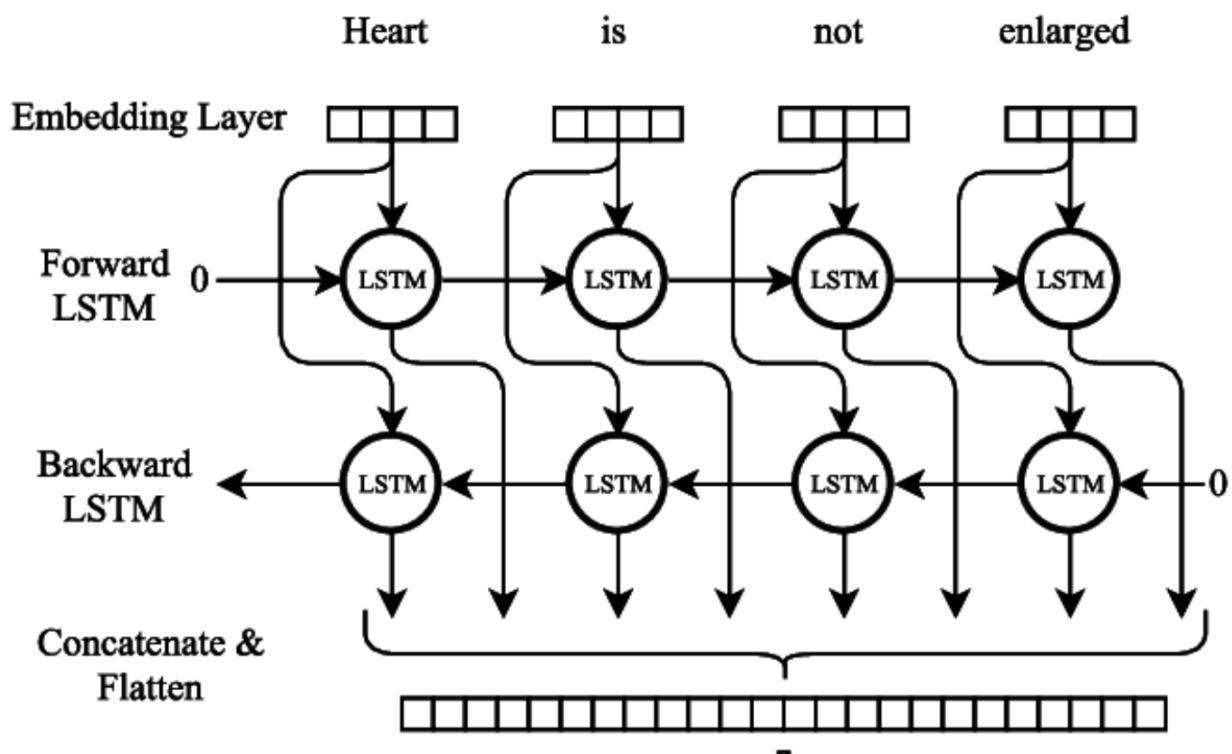
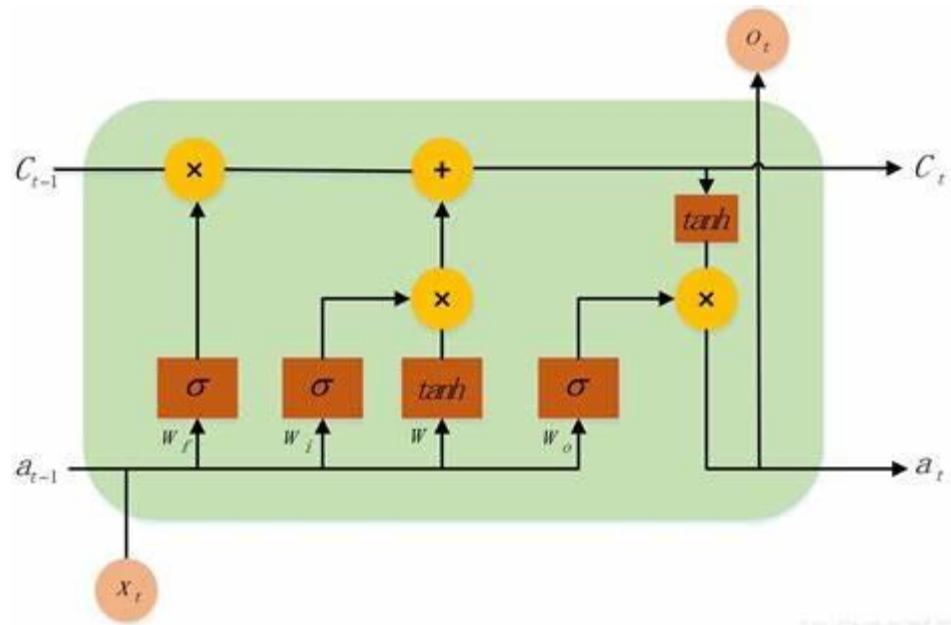
Thành phần chính gây ra vanishing gradient trong RNN là $\frac{\partial s_{t+1}}{\partial s_t} = (1 - s_t^2) * W$, trong đó $s_t, W < 1$.

Tương tự trong LSTM ta quan tâm đến $\frac{\partial c_t}{\partial c_{t-1}} = f_t$. Do $0 < f_t < 1$ nên về cơ bản thì LSTM vẫn bị vanishing gradient nhưng bị ít hơn so với RNN. Hơn thế nữa, khi mang thông tin trên cell state thì ít khi cần phải quên giá trị cell cũ, nên $f_t \approx 1 \Rightarrow$ Tránh được vanishing gradient.

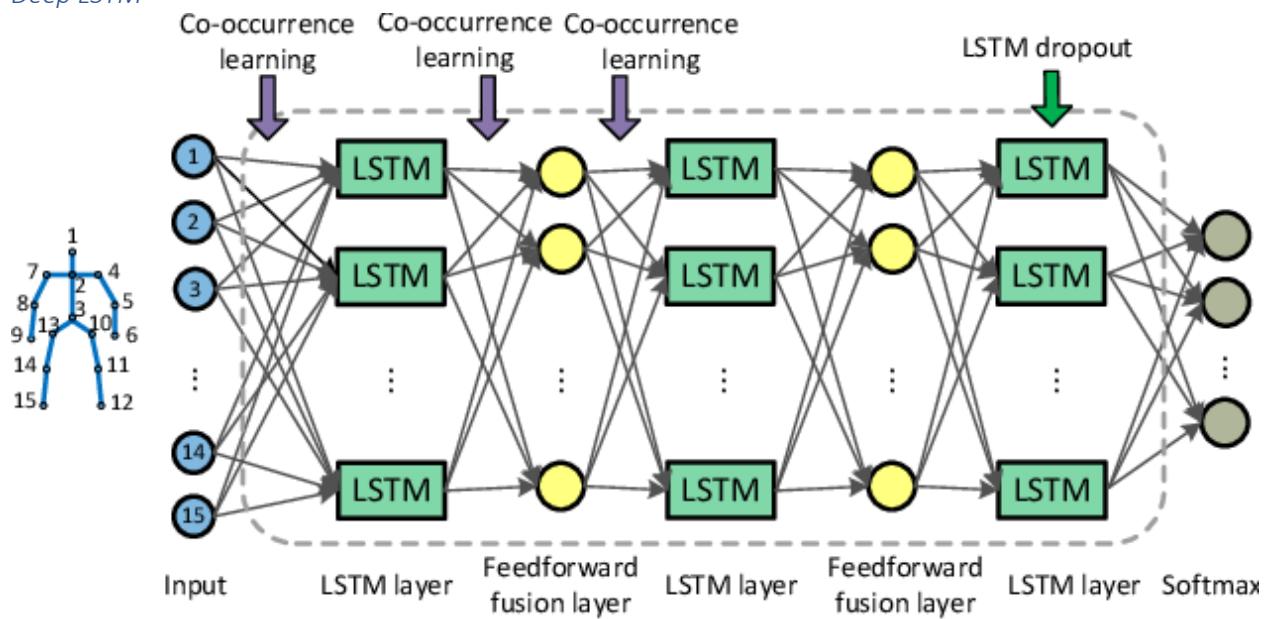
Do đó LSTM được dùng phổ biến hơn RNN cho các toán thông tin dạng chuỗi. Bài sau tôi sẽ giới thiệu về ứng dụng LSTM cho image captioning.

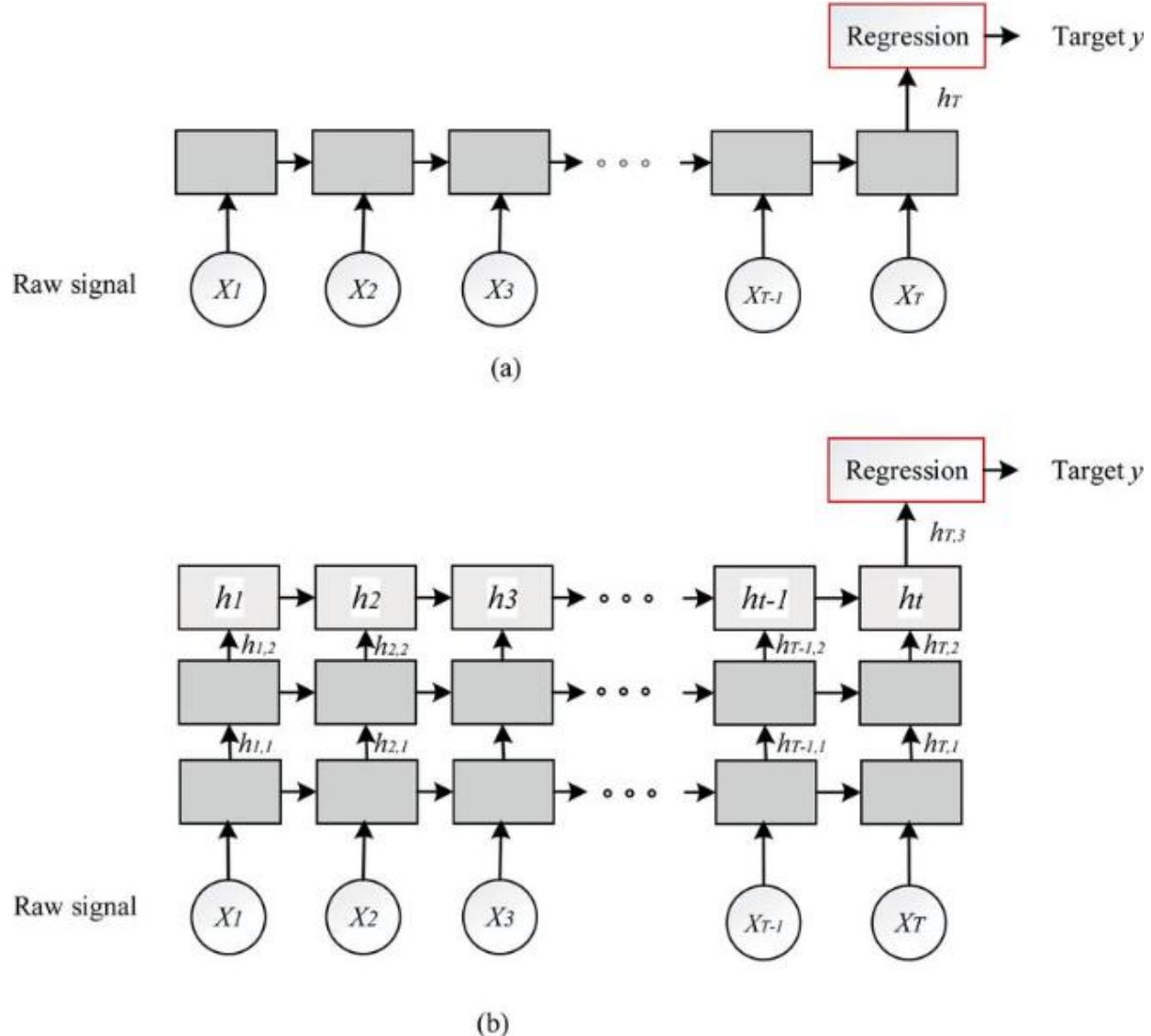
Readmore

Bi LSTM



Deep LSTM





18. Ứng dụng thêm mô tả cho ảnh

C:\Users\ADMIN\Downloads\DeepLearning\Learning\Chapter7_Recurrent Neural Network>New folder\RNN - Photo Description.ipynb

Dataset : [Flickr 8k Dataset](#)

Ứng dụng



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."

Hình 18.1: Ví dụ image captioning [10]

Ta có thể thấy ngay 2 ứng dụng của image captioning:

- Để giúp những người già mắt kém hoặc người mù có thể biết được cảnh vật xung quanh hay hỗ trợ việc di chuyển. Quy trình sẽ là: Image -> text -> voice.
- Giúp google search có thể tìm kiếm được hình ảnh dựa vào caption.

Dataset

[Flickr8k Dataset - Academic Torrents](#)

Dữ liệu dùng trong bài này là Flickr8k Dataset. Mọi người tải ở [đây](#). Dữ liệu gồm 8000 ảnh, 6000 ảnh cho training set, 1000 cho dev set (validation set) và 1000 ảnh cho test set.

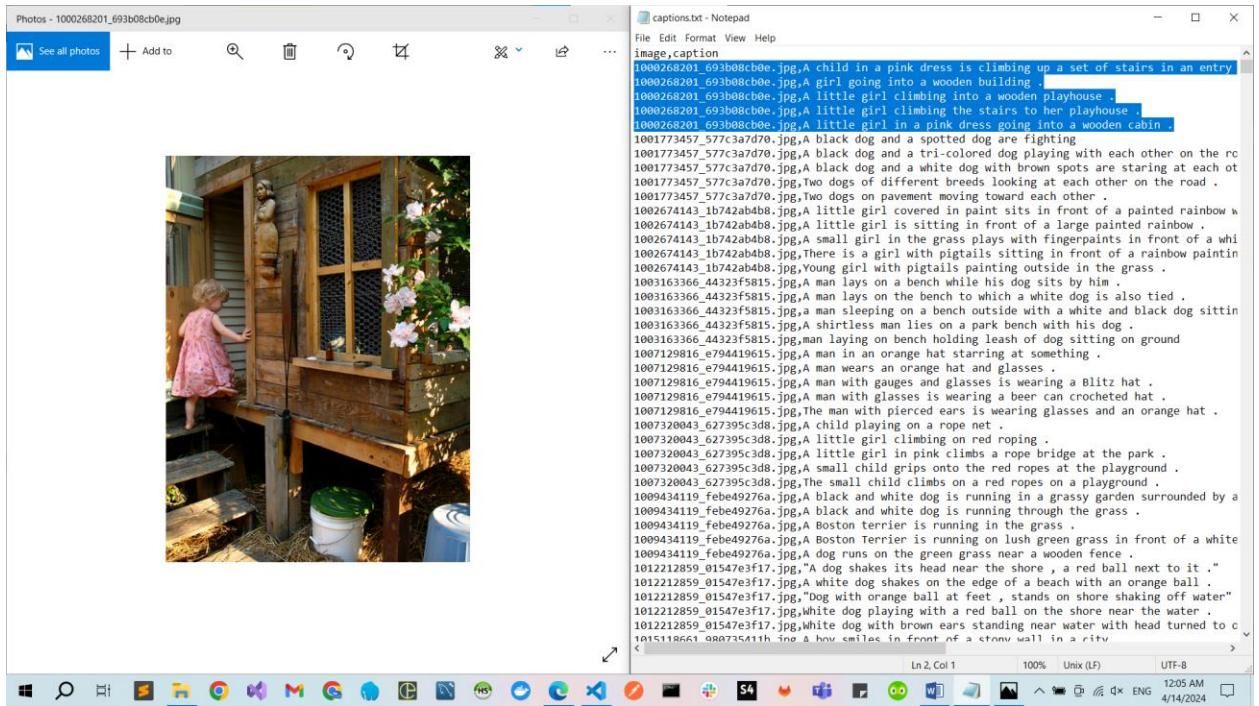
Bạn tải về có 2 folder: Flicker8k_Dataset và Flicker8k_Text. Flicker8k_Dataset chứa các ảnh với tên là các id khác nhau. Flicker8k_Text chứa:

- Flickr_8k.testImages, Flickr_8k.devImages, Flickr_8k.trainImages, Flickr_8k.devImages
chứa id các ảnh dùng cho việc test, train, validation.
- Flickr8k.token chứa các caption của ảnh, mỗi ảnh chứa 5 captions.

Ví dụ ảnh ở hình 18.2 có 5 captions:

- A child in a pink dress is climbing up a set of stairs in an entry way.
- A girl going into a wooden building.
- A little girl climbing into a wooden playhouse.
- A little girl climbing the stairs to her playhouse.
- A little girl in a pink dress going into a wooden cabin.

Thực ra 1 ảnh nhiều caption cũng hợp lý vì bức ảnh có thể được mô tả theo nhiều cách khác nhau. Một ảnh 5 caption sẽ cho ra 5 training set khác nhau: (ảnh, caption 1), (ảnh, caption 2), (ảnh, caption 3), (ảnh, caption 4), (ảnh, caption 5). Như vậy training set sẽ có $6000 * 5 = 40000$ dataset.



Phân tích bài toán

Input là ảnh và output là text, ví dụ "man in black shirt is playing guitar".

Nhìn chung các mô hình machine learning hay deep learning đều không xử lý trực tiếp với text như 'man', 'in', 'black',... mà thường phải quy đổi (encode) về dạng số. Từng từ sẽ được encode sang dạng vector với độ dài số định, phương pháp đấy gọi là word embedding.

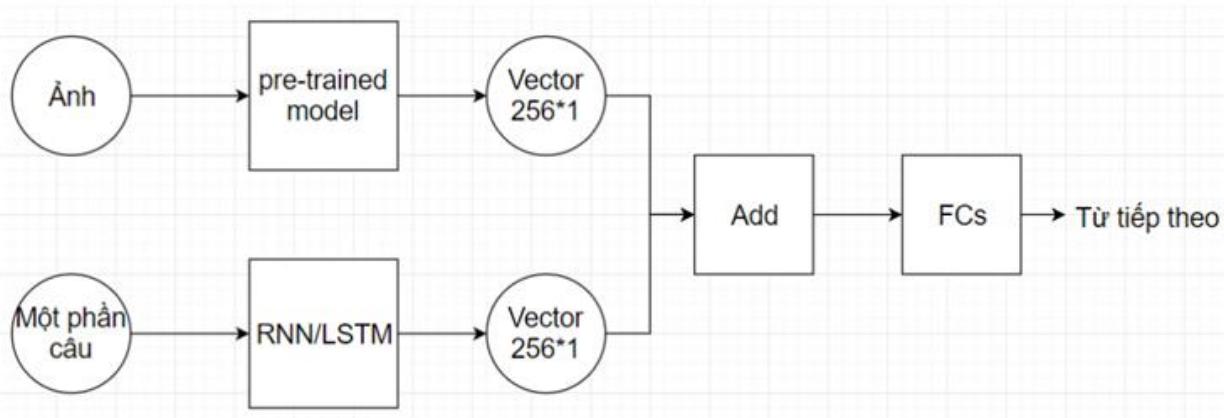
Nhìn thấy output là text nghĩ ngay đến RNN và sử dụng mô hình LSTM.

Input là ảnh thường được extract feature qua pre-trained model với dataset lớn như ImageNet và model phổ biến như VGG16, ResNet, quá trình được gọi là embedding và output là 1 vector.

Ý tưởng sẽ là dùng embedding của ảnh và dùng các từ phía trước để dự đoán từ tiếp theo trong caption.

Ví dụ:

- Embedding vector + A -> girl
- Embedding vector + A girl -> going
- Embedding vector + A girl going -> into
- Embedding vector + A girl going into -> a.
- Embedding vector + A girl going into a -> wooden building .
- Embedding vector + A girl going into a wooden -> building .



Hình 18.3: Mô hình của bài toán

Để dự đoán từ tiếp theo ta sẽ xây dựng từ điển các từ xuất hiện trong training set (ví dụ 2000 từ) và bài toán trở thành bài toán phân loại từ, xem từ tiếp theo là từ nào, khá giống như bài phân loại ảnh.

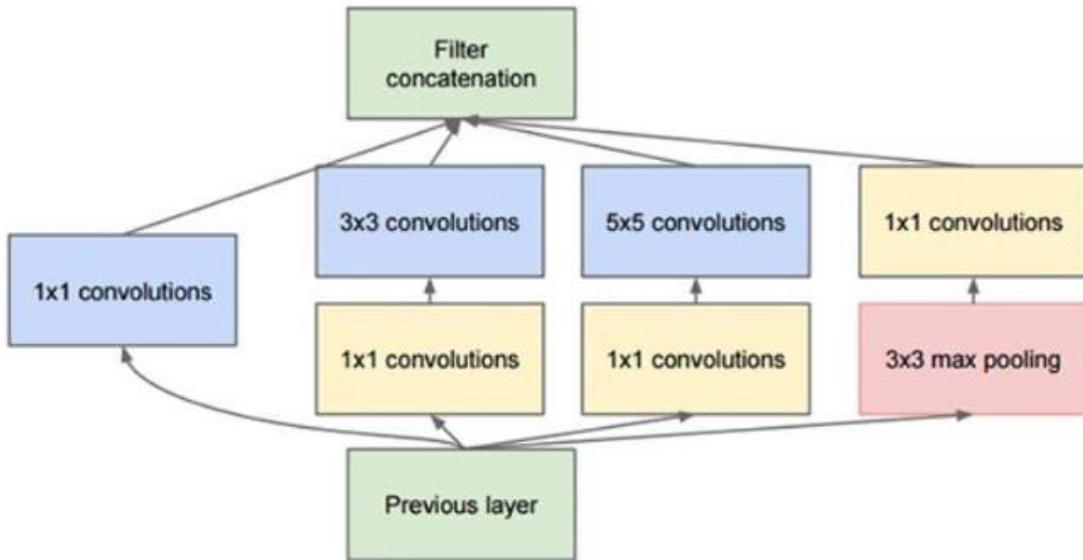
Chi tiết các bước

Image embedding với inception

Inception (khởi đầu)

Exception (ngoại lệ)

Có lẽ cái tên GoogLeNet sẽ quen thuộc hơn và gấp nhiều hơn so với Inception, GoogLeNet là version 1 của Inception, hiện giờ mô hình phổ biến là Inception v3.



Hình 18.4: Mô hình Googlenet, Going Deeper with Convolutions, Szegedy et al

Thay vì trong mỗi Conv layer chỉ dùng 1 kernel size nhất định như 3×3 , 5×5 , thì giờ ở một layer có nhiều kernel size khác nhau, do đó mô hình có thể học được nhiều thuộc tính khác nhau của ảnh trong mỗi layer.

Cụ thể hơn mọi người xem thêm ở [đây](#).

Ta sẽ sử dụng pre-trained model Inception v3 với dataset Imagenet. Do là pre-trained model yêu cầu ảnh đầu vào là 229×229 nên ta sẽ resize ảnh về kích thước này. Sau khi qua pre-trained model ta sẽ lấy được embedding vector của ảnh, kích thước 256×1

Text preprocessing

Ta xử lý text qua một số bước cơ bản.

- Chuyển chữ hoa thành chữ thường, "Hello" -> "hello"
- Bỏ các kí tự đặc biệt như "
- Loại bỏ các chữ có số như hey199

Sau đó ta sẽ thêm 2 từ "startseq" và "endseq" để biểu thị sự bắt đầu và kết thúc của caption. Ví dụ: "startseq a girl going into a wooden building endseq". "endseq" dùng khi test ảnh thì biết kết thúc của caption.

Ta thấy có 8763 chữ khác nhau trong số 40000 caption. Tuy nhiên ta không quan tâm lắm những từ mà chỉ xuất hiện 1 vài lần, vì nó giống như là nhiễu vậy và không tốt cho việc học và dự đoán từ của model, nên ta chỉ giữ lại những từ mà xuất hiện trên 10 lần trong số tất cả các caption. Sau khi bỏ những từ xuất hiện ít hơn 10 lần ta còn 1651 từ.

Tuy nhiên do độ dài các sequence khác nhau, ví dụ: "A", " A girl going", " A girl going into a wooden", nên ta cần padding thêm để các chuỗi có cùng độ dài bằng với độ dài của chuỗi dài nhất là 34. Do đó số tổng số từ (từ điển) ta có là $1651 + 1$ (từ dùng để padding).

Word embedding [HAY] [Quan trọng] [Đọc kỹ]

Để có thể đưa text vào mô hình deep learning, việc đầu tiên chúng ta cần **làm là số hóa các từ đầu vào** (embedding). Ở phần này chúng ta sẽ thảo luận về các mô hình nhúng từ (word embedding) và sự ra đời của mô hình **word2vec** rất nổi tiếng được google giới thiệu vào năm 2013.

Các phương pháp trước đây

One hot encoding

Phương pháp này là phương pháp đơn giản nhất để đưa từ về dạng số hóa vector với chiều bằng với kích thước bộ từ điển. Mỗi từ sẽ được biểu diễn bởi 1 vector mà giá trị tại vị trí của từ đó trong từ điển bằng 1 và giá trị tại các vị trí còn lại đều bằng 0.

Ví dụ: Ta có 3 câu đầu vào: "Tôi đang đi học", "Mình đang bận nhé", "Tôi sẽ gọi lại sau". Xây dựng bộ từ điển: "Tôi, đang, đi, học, Mình, bận, nhé, sẽ, gọi, lại, sau". Ta có các biểu diễn one hot encoding của từng từ như sau:

Tôi: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],

đang: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],

...

Mình: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

...

sau: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1].

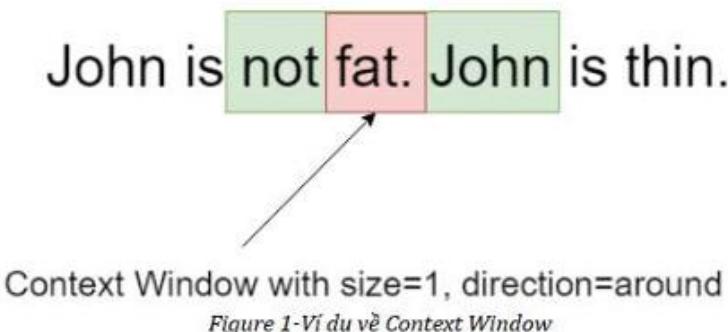
Cách biểu diễn này rất đơn giản, tuy nhiên ta có thể nhận thấy ngay các hạn chế của phương pháp này. Trước hết, one hot encoding không thể hiện được thông tin về ngữ nghĩa của từ, ví dụ như khoảng cách($\text{vector}(\text{Tôi}) - \text{vector}(\text{Mình})$) = khoảng cách($\text{vector}(\text{Tôi}) - \text{vector}(\text{đang})$), trong khi rõ ràng từ "Tôi" và từ "Mình" trong ngữ cảnh như trên có ý nghĩa rất giống nhau còn từ "Tôi" và từ "đang" lại khác nhau hoàn toàn. Tiếp nữa, mỗi từ đều được biểu diễn bằng một vector có độ dài bằng kích thước bộ từ điển, như bộ từ điển của google gồm 13 triệu từ, thì mỗi one hot vector sẽ dài 13 triệu chiều. Cách biểu diễn này tồn tại nhiều tài nguyên nhưng thông tin biểu diễn được lại rất

hạn hẹp.

=> Cần một cách biểu diễn từ ít chiều hơn và mang nhiều thông tin hơn.

Co-occurrence Matrix

Năm 1957, nhà ngôn ngữ học J.R. Firth phát biểu rằng: "Bạn sẽ biết nghĩa của một từ nhờ những từ đi kèm với nó.". Điều này cũng khá dễ hiểu. Ví dụ nhắc đến Việt Nam, người ta thường có các cụm từ quen thuộc như "Chiến tranh Việt Nam", "Cafe Việt Nam", "Việt Nam rừng vàng biển bạc", dựa vào những từ xung quanh ta có thể hiểu hoặc mường tượng ra được "Việt Nam" là gì, như thế nào. Co-occurrence Matrix được xây dựng dựa trên nhận xét trên, co-occurrence đảm bảo quan hệ ngữ nghĩa giữa các từ, dựa trên số lần xuất hiện của các cặp từ trong "context window". Một context window được xác định dựa trên kích thước và hướng của nó, ví dụ của context window:



Hình 18.5: Ví dụ về context window

Co-occurrence matrix là một ma trận vuông đối xứng, mỗi hàng, mỗi cột sẽ làm vector đại diện cho từ tương ứng. Từ ví dụ trên ta tiếp tục xây dựng co-occurrence matrix:

| | John | is | not | fat | thin |
|------|------|----|-----|-----|------|
| John | 0 | 2 | 0 | 1 | 0 |
| is | 2 | 0 | 1 | 0 | 1 |
| not | 0 | 1 | 0 | 1 | 0 |
| fat | 1 | 0 | 1 | 0 | 0 |
| thin | 0 | 1 | 0 | 0 | 0 |

Figure 2-Ví dụ về Co-occurrence Matrix

Hình 18.6: Ví dụ về co-ocurrence matrix

Trong đó, giá trị tại ô $[i, j]$ là số lần xuất hiện của từ i nằm trong context window của từ j .
Cách biểu diễn trên mặc dù đã giữ được thông tin về ngữ nghĩa của một từ, tuy vẫn còn các hạn chế như sau:

- Khi kích thước bộ từ điển tăng, chiều vector cũng tăng theo.
- Lưu trữ co-occurrence matrix cần rất nhiều tài nguyên về bộ nhớ.
- Các mô hình phân lớp bị gặp vấn đề với biểu diễn thưa (có rất nhiều giá trị 0 trong ma trận).

Để làm giảm kích thước của co-occurrence matrix người ta thường sử dụng **phép SVD (Singular Value Decomposition)** để giảm chiều ma trận. Ma trận thu được sau SVD có chiều nhỏ hơn, dễ lưu trữ hơn và ý nghĩa của từ cũng cô đọng hơn. Tuy nhiên, SVD có độ phức tạp tính toán cao, tăng nhanh cùng với chiều của ma trận ($O(mn^2)$ với m là chiều của ma trận trước SVD, n là chiều của ma trận sau SVD và $n < m$), ngoài ra phương pháp này cũng gặp khó khăn khi thêm các từ vựng mới vào bộ từ điển.

=> Cần phương pháp khác lưu trữ được nhiều thông tin và vector biểu diễn nhỏ.

Word to vec (Word2vec)

Với tư tưởng rằng ngữ cảnh và ý nghĩa của một từ có sự tương quan mật thiết đến nhau, năm 2013 nhóm của Mikolov đề xuất một phương pháp mang tên Word2vec.

Ý tưởng chính của Word2vec

- Thay thế việc lưu thông tin số lần xuất hiện của các từ trong context window như co-occurrence matrix, word2vec học cách dự đoán các từ lân cận.
- Tính toán nhanh hơn và có thể transfer learning khi thêm các từ mới vào bộ từ điển.

Phương pháp:

Với mỗi từ t trong bộ từ điển ta dự đoán các từ lân cận trong bán kính m của nó.

Hàm mục tiêu nhằm tối ưu xác suất xuất hiện của các từ ngữ cảnh (context word) đối với từ đang

$$\text{xét hiện tại: } J(\theta) = -\frac{1}{T} \prod_{t=1}^T \prod_{j=-m, j \neq 0}^m p(w_{t+j} | w_t; \theta)$$

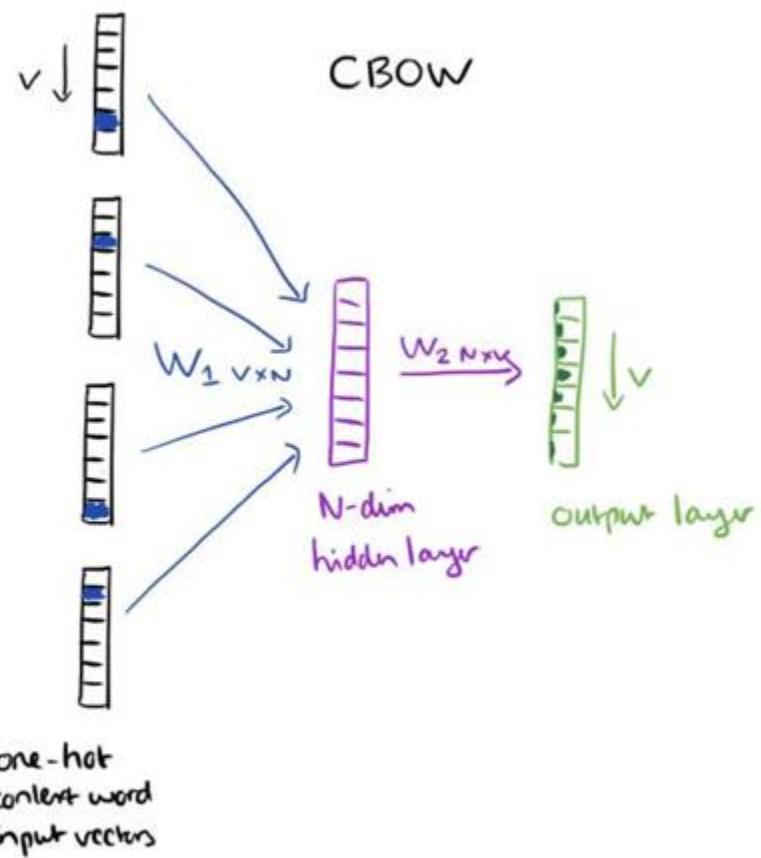
Có 2 kiến trúc khác nhau của word2vec, là CBoW và Skip-Gram:

- Cbow: Cho trước ngữ cảnh ta dự đoán xác suất từ đích. Ví dụ: "I ... you", với đầu vào là 2 từ "I" và "you" ta cố gắng dự đoán từ còn thiếu, chẳng hạn "love".
- Skip-Gram: Cho từ đích ta dự đoán xác suất các từ ngữ cảnh (nằm trong context window) của nó. Ví dụ: "... love ...", cho từ "love" ta dự đoán các từ là ngữ cảnh của nó, chẳng hạn "I", "you".

Trong bài báo giới thiệu word2vec, Mikolov và cộng sự có so sánh và cho thấy 2 mô hình này cho kết quả tương đối giống nhau.

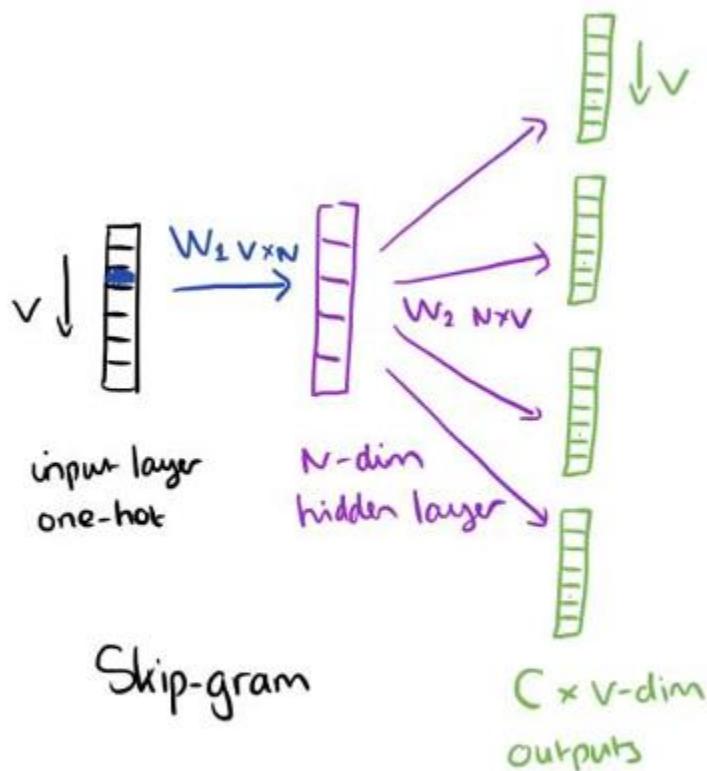
Chi tiết mô hình

Cbow



Hình 18.7: Mô hình Cbow

Skip – Gram



Hình 18.8: Mô hình Skip-Gram

Do 2 kiến trúc khá giống nhau nên ta chỉ thảo luận về Skip-Gram.

Mô hình Skip-Gram sẽ input từ đích và dự đoán ra các từ ngữ cảnh. Thay vì input từ đích và output ra nhiều từ ngữ cảnh trong 1 mô hình, họ xây dựng model để input từ đích và output ra 1 từ ngữ cảnh.

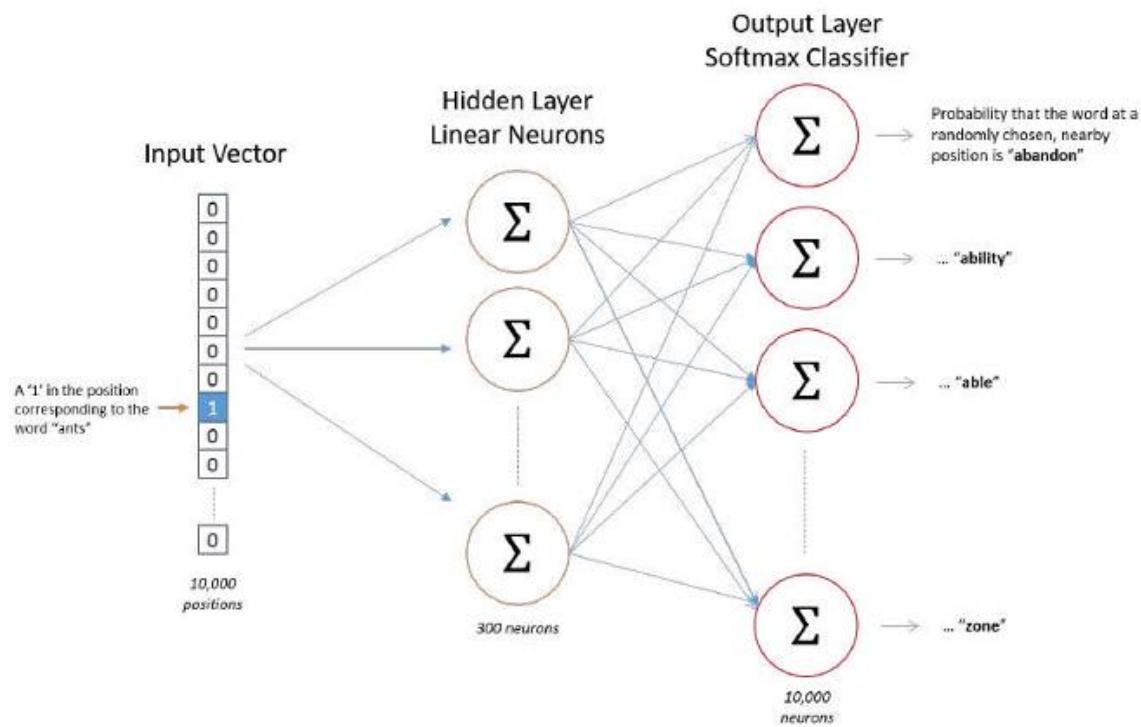
Mô hình là một mạng neural network 2 lớp, với chỉ 1 hidden layer. Input là một từ trong từ điển đã được mã hóa thành dạng one hot vector chiều $V * 1$ với V là kích thước từ điển. Hidden layer không sử dụng activation function có N node, trong đó N chính là độ dài vector embedding của mỗi từ. Output layer có V node, sau đó softmax activation được sử dụng để chuyển về dạng xác suất. Categorical cross entropy loss function được học để dự đoán được từ ngữ cảnh với input là từ đích.

Ví dụ của xây dựng training data:

| Source Text | Training Samples |
|--|--|
| The quick brown fox jumps over the lazy dog. → | (the, quick)
(the, brown) |
| The quick brown fox jumps over the lazy dog. → | (quick, the)
(quick, brown)
(quick, fox) |
| The quick brown fox jumps over the lazy dog. → | (brown, the)
(brown, quick)
(brown, fox)
(brown, jumps) |
| The quick brown fox jumps over the lazy dog. → | (fox, quick)
(fox, brown)
(fox, jumps)
(fox, over) |

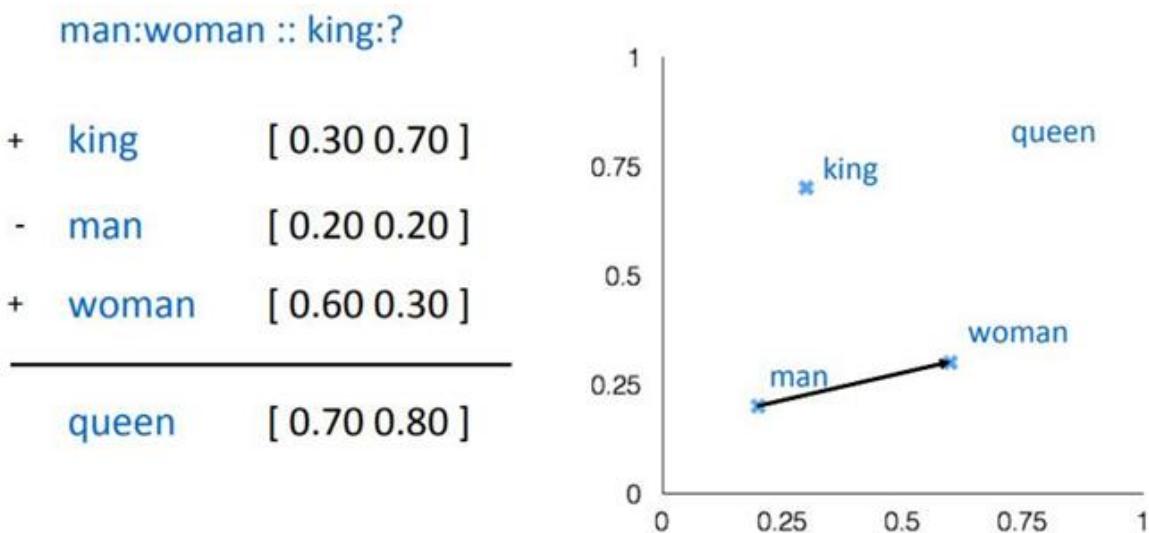
Hình 18.9: Ví dụ của xây dựng training data, window size = 2, tức là lấy 2 từ bên trái và 2 từ bên phải mỗi từ trung tâm làm từ ngữ cảnh (context word)

Ví dụ của model Skip-gram:



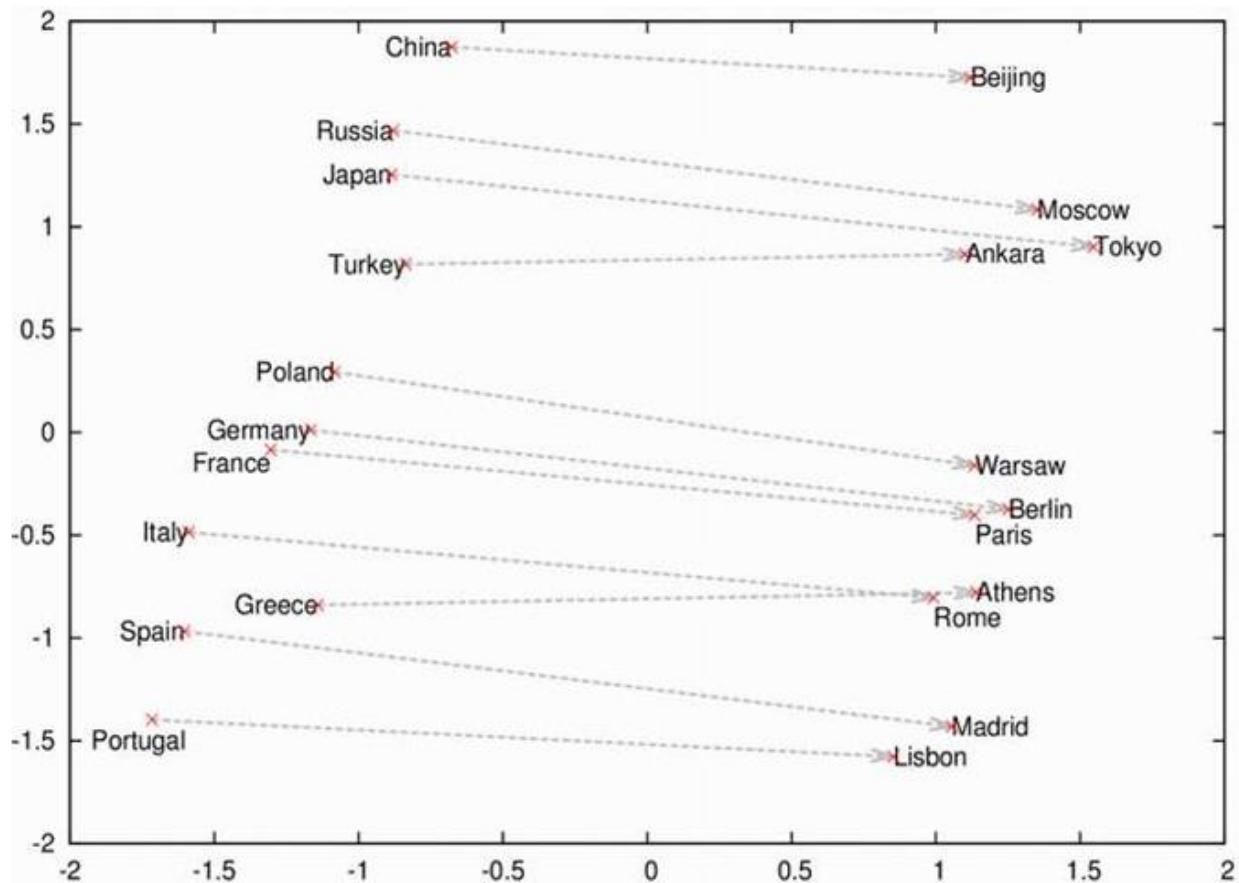
Hình 18.10: Ví dụ của xây dựng Skip-gram

Một số kết quả của Word2vec:



Hình 18.11: Vị trí các word vector trong không gian

Ví dụ trên là ví dụ kinh điển của Word2vec cho thấy các vector biểu diễn tốt quan hệ về mặt ngữ nghĩa của từ vựng như thế nào.



Hình 18.12: Mối quan hệ của đất nước và thủ đô tương ứng

Pre-trained GLOVE Model được sử dụng cho quá trình word embedding.

Mọi người vào link [này](#) để tải file glove.6B.zip

Từng dòng trong file sẽ lưu text và encoded vector khích thước 200*1

[GloVe: Global Vectors for Word Representation \(stanford.edu\)](http://nlp.stanford.edu/glove)

GloVe: Global Vectors for Word Representation

Jeffrey Pennington, Richard Socher, Christopher D. Manning

Introduction

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

Getting started (Code download)

- Download the latest [latest code](#) (licensed under the [Apache License, Version 2.0](#)).
- Look for "Clone or download".
- Unpack the files: `unzip master.zip`
- Compile the source: `cd GloVe-master && make`
- Run the demo script: `./demo.sh`
- Consult the included README for further usage details, or ask a [question](#).

Download pre-trained word vectors

- Pre-trained word vectors. This data is made available under the [Public Domain Dedication and License v1.0](#) whose full text can be found at: <http://www.opendatacommons.org/licenses/pddl/1.0/>
 - [Wiktionary 2014 + Gigaword 5](#) (6B tokens, 400K vocab, uncased, 500, 1000, 2000, & 3000 vectors, 822 MB download): [glove.42B.300d.zip](#)
 - Common Crawl (42B tokens, 19M vocab, uncased, 3000 vectors, 1.75 GB download): [glove.42B.300d.zip](#)
 - Common Crawl (840B tokens, 2.2M vocab, cased, 3000 vectors, 2.03 GB download): [glove.840B.300d.zip](#)
 - Twitter (2B tweets, 2.7B tokens, 1.2M vocab, uncased, 250, 500, 1000, & 2000 vectors, 1.42 GB download): [glove.twitter.27B.zip](#)
- Ruby [script](#) for preprocessing Twitter data

Citing GloVe

Jeffrey Pennington, Richard Socher, and Christopher D. Manning, 2014. [GloVe: Global Vectors for Word Representation](#). [pdf] [bib]

Highlights

Output

Bài toán là dự đoán từ tiếp theo trong chuỗi ở input với ảnh hiện tại, nên output là từ nào trong số 1652 từ trong từ điển mà ta có. Với bài toán phân loại thì softmax activation và categorical_crossentropy loss function được sử dụng.

Model

| Layer (type) | Output Shape | Param # | Connected to |
|-------------------------|-----------------|---------|-------------------|
| input_3 (InputLayer) | (None, 34) | 0 | |
| input_2 (InputLayer) | (None, 2048) | 0 | |
| embedding_1 (Embedding) | (None, 34, 200) | 330400 | input_3[0][0] |
| dropout_1 (Dropout) | (None, 2048) | 0 | input_2[0][0] |
| dropout_2 (Dropout) | (None, 34, 200) | 0 | embedding_1[0][0] |

```
dense_1 (Dense)           (None, 256)      524544    dropout_1[0][0]
-----
lstm_1 (LSTM)             (None, 256)      467968    dropout_2[0][0]
-----
add_1 (Add)               (None, 256)      0         dense_1[0][0]
                                         lstm_1[0][0]
-----
dense_2 (Dense)           (None, 256)      65792     add_1[0][0]
-----
dense_3 (Dense)           (None, 1652)     424564    dense_2[0][0]
=====
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
```

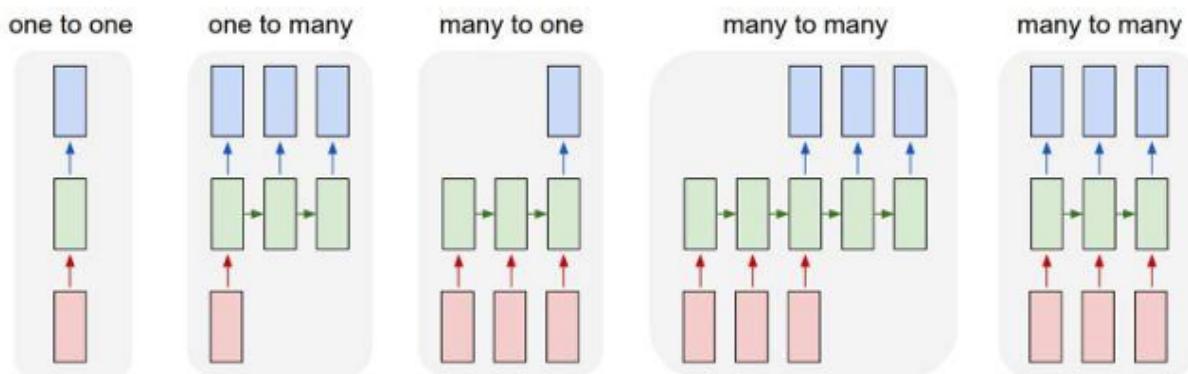
Python code

C:\Users\ADMIN\Downloads\DeepLearning\Learning\DL_Tutorial\L15\Automatic Image Captioning.ipynb

19. Seq2seq và attention

Giới thiệu

Mô hình RNN ra đời để xử lý các dữ liệu dạng chuỗi (sequence) như text, video.



Hình 19.1: Các dạng bài toán RNN

Bài toán RNN được phân làm một số dạng:

- **One to one:** mẫu bài toán cho Neural Network (NN) và Convolutional Neural Network (CNN), 1 input và 1 output, ví dụ với bài toán phân loại ảnh MNIST input là ảnh và output ảnh đấy là số nào.
- **One to many:** bài toán có 1 input nhưng nhiều output, ví dụ với bài toán caption cho ảnh, input là 1 ảnh nhưng output là nhiều chữ mô tả cho ảnh đấy, dưới dạng một câu.
- **Many to one:** bài toán có nhiều input nhưng chỉ có 1 output, ví dụ bài toán phân loại hành động trong video, input là nhiều ảnh (frame) tách ra từ video, output là hành động trong video.
- **Many to many:** bài toán có nhiều input và nhiều output, ví dụ bài toán dịch từ tiếng anh sang tiếng việt, input là 1 câu gồm nhiều chữ: "I love Vietnam" và output cũng là 1 câu gồm nhiều chữ "Tôi yêu Việt Nam". Để ý là độ dài sequence của input và output có thể khác nhau.

Mô hình sequence to sequence (seq2seq) sinh ra để giải quyết bài toán many to many và rất thành công trong các bài toán: dịch, tóm tắt đoạn văn. Bài này mình sẽ cùng tìm hiểu về mô hình seq2seq với bài toán dịch từ tiếng anh sang tiếng việt.

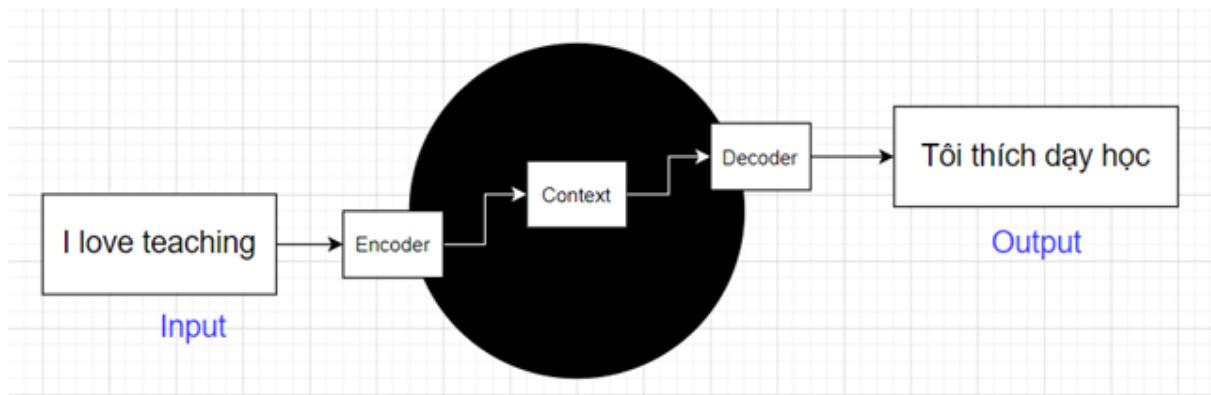
Mô hình seq2seq

Input của mô hình seq2seq là một câu tiếng anh và output là câu dịch tiếng việt tương ứng, độ dài hai câu này có thể khác nhau. Ví dụ: input: I love teaching -> output: Tôi thích dạy học, input 1 câu 3 từ, output 1 câu 4 từ.



Hình 19.2: Seq2seq model

Mô hình seq2seq gồm 2 thành phần là encoder và decoder.

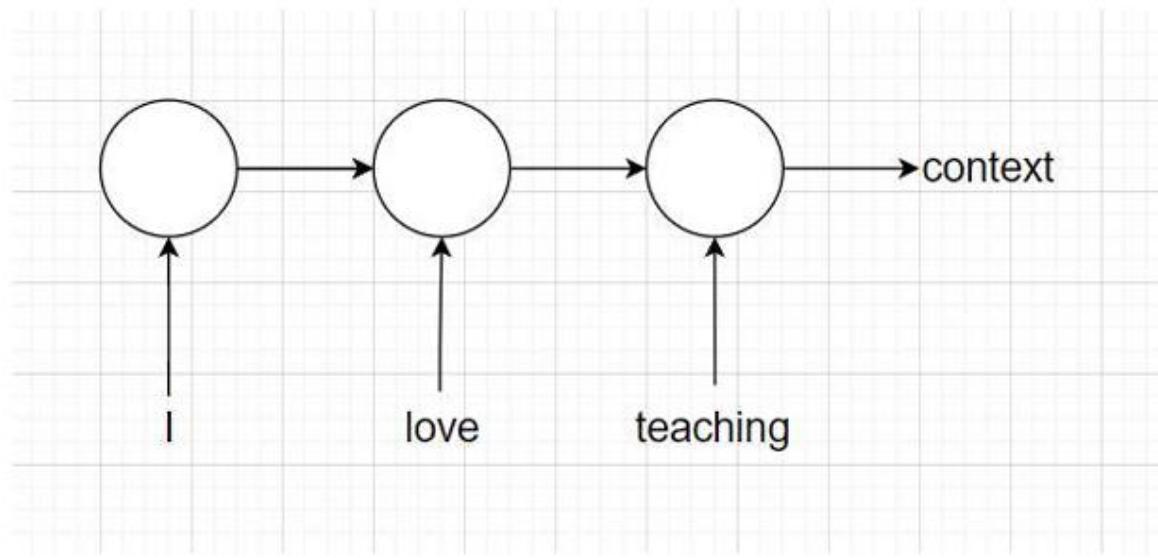


Hình 19.3: Seq2seq model

Encoder nhận input là câu tiếng anh và output ra context vector, còn decoder nhận input là context vector và output ra câu tiếng việt tương ứng. Phần encoder sử dụng mô hình RNN (nói là mô hình RNN nhưng có thể là các mô hình cải tiến như GRU, LSTM) và context vector được dùng là hidden states ở node cuối cùng. Phần decoder cũng là một mô hình RNN với s_0 chính là context vector rồi dần dần sinh ra các từ ở câu dịch.

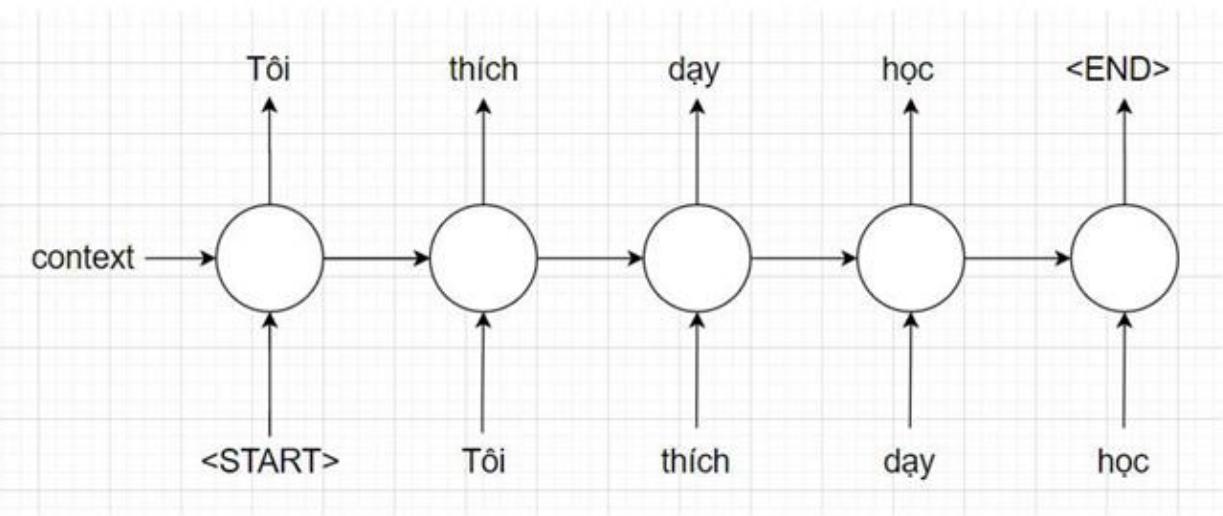
Phần decoder này giống với bài toán image captioning. Ở bài image captioning mình cũng cho ảnh qua pre-trained model để lấy được embedding vector, sau đó cho embedding vector làm s_0 của mô hình RNN rồi sinh ra caption tương ứng với ảnh.

Bài trước mình đã biết model RNN chỉ nhận input dạng vector nên dữ liệu ảnh (từ) sẽ được encode về dạng vector trước khi cho vào model.



Hình 19.4: Mô hình encoder

Các từ trong câu tiếng anh sẽ được embedding thành vector và cho vào mô hình RNN, hidden state ở node cuối cùng sẽ được dùng làm context vector. Về mặt lý thuyết thì context vector sẽ mang đủ thông tin của câu tiếng anh cần dịch và sẽ được làm input cho decoder.



Hình 19.5: Mô hình decoder

2 tag `<START>` và `<END>` được thêm vào câu output để chỉ từ bắt đầu và kết thúc của câu dịch. Mô hình decoder nhận input là context vector. Ở node đầu tiên context vector và tag `<START>` sẽ output ra chữ đầu tiên trong câu dịch, rồi tiếp tục mô hình sinh chữ tiếp theo cho đến khi gặp tag `<END>` hoặc đến max_length của câu output thì dừng lại.

Ví dụ code seq2seq cho bài toán dịch với mô hình LSTM mọi người tham khảo ở [đây](#).

Vấn đề: Mô hình seq2seq encode cả câu tiếng anh thành 1 context vector, rồi dùng context vector để sinh ra các từ trong câu dịch ứng tiếng việt. Như vậy khi câu dài thì rất khó cho decoder chỉ dùng 1 context vector có thể sinh ra được câu output chuẩn. Thêm vào đó các mô hình RNN đều bị mất ít nhiều thông tin ở các node ở xa nên bản thân context vector cũng khó để học được thông tin ở các từ ở phần đầu của encoder.

[keras-io/examples/nlp/lstm_seq2seq.py at master · keras-team/keras-io \(github.com\)](https://github.com/keras-team/keras/blob/master/examples/nlp/lstm_seq2seq.py)

=> Cần có cơ chế để lấy được thông tin các từ ở input cho mỗi từ cần dự đoán ở ouput thay vì chỉ dựa vào context vector => **Attention** ra đời.

Attention (Chú ý)

Cơ chế attention

Motivation

Attention tiếng anh nghĩa là chú ý, hay tập trung. Khi dịch ở mỗi từ tiếng việt ta cần chú ý đến 1 vài từ tiếng anh ở input, hay nói cách khác là có 1 vài từ ở input có ảnh hưởng lớn hơn để dịch từ đấy.



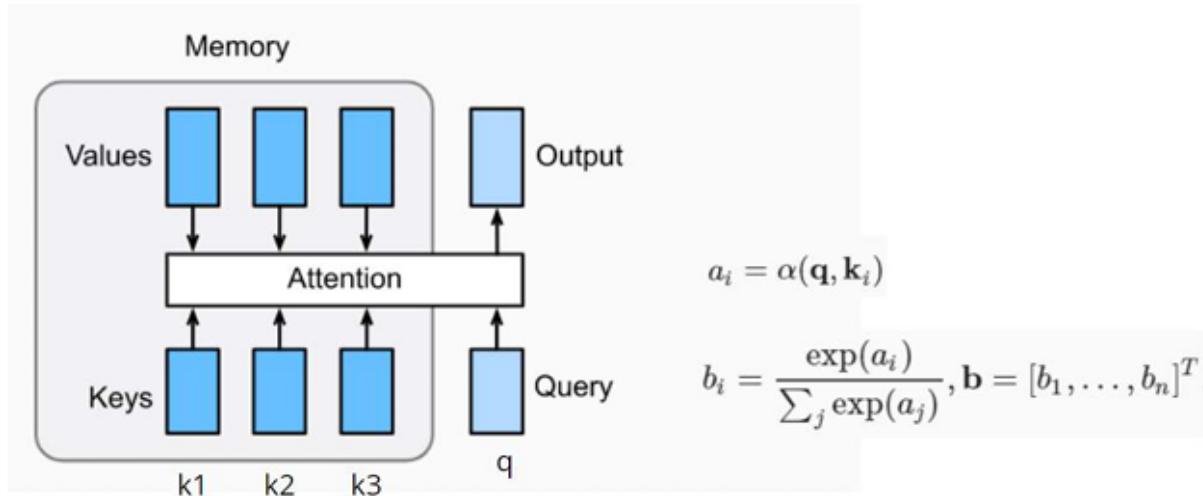
Hình 19.6: Dịch tiếng anh sang tiếng việt, độ quan trọng các từ khi dịch

Ta thấy từ I có trọng số ảnh hưởng lớn tới việc dịch từ tôi, hay từ teaching có ảnh hưởng nhiều tới việc dịch từ dạy và từ học.

=> Do đó khi dịch mỗi từ ta cần chú ý đến các từ ở câu input tiếng anh và đánh trọng số khác nhau cho các từ để dịch chuẩn hơn.

Cách hoạt động

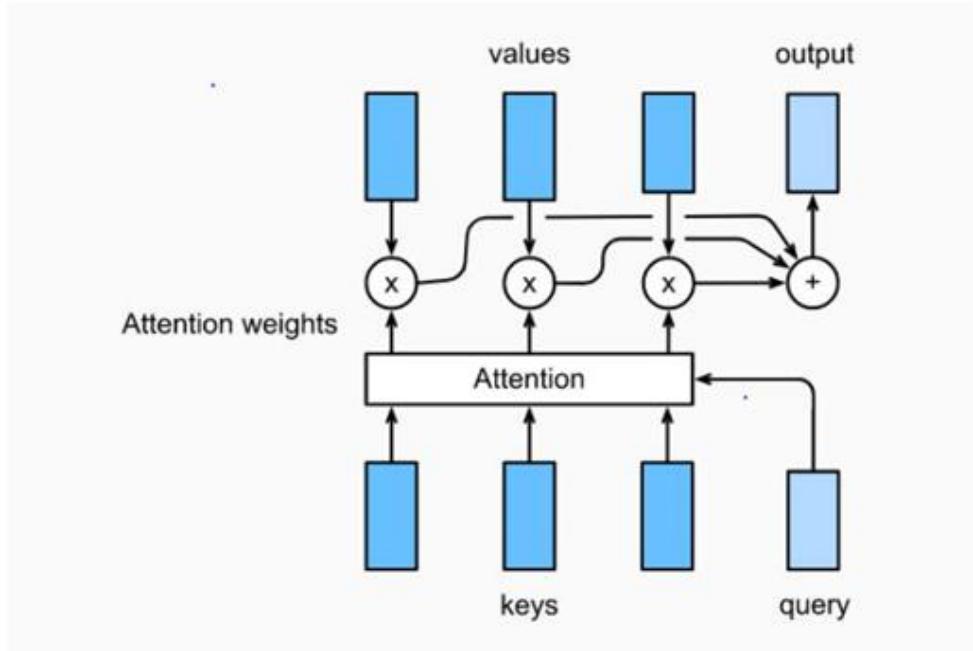
Attention sẽ định nghĩa ra 3 thành phần query, key, value.



Hình 19.7: Các thành phần attention

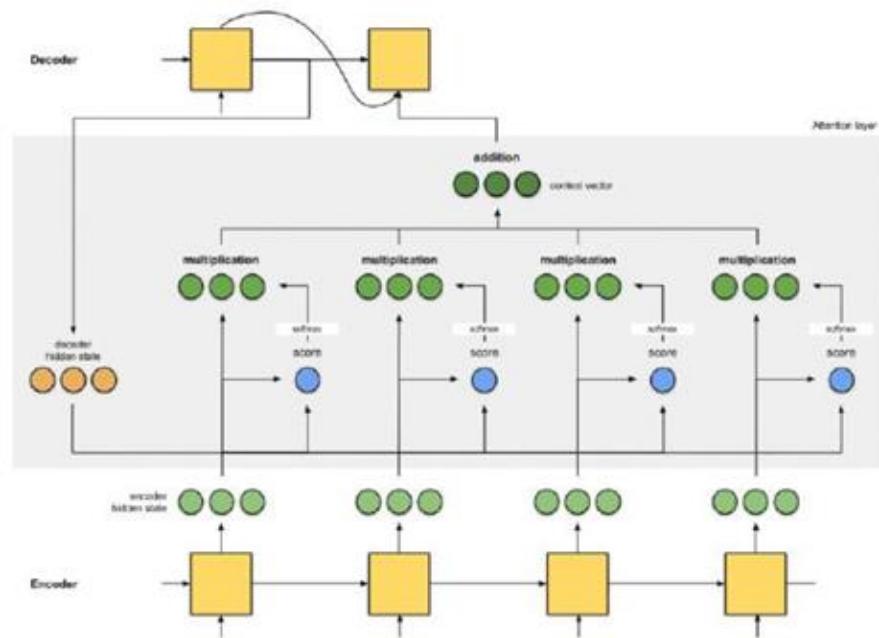
Query (q) lấy thông tin từ từ tiếp theo cần dịch (ví dụ từ dạy). Mỗi từ trong câu input tiếng anh sẽ cho ra 2 thành phần tương ứng là key và value, từ thứ i kí hiệu là k_i, v_i .

Mỗi bộ q, k_i qua hàm α sẽ cho ra a_i tương ứng, a_i chính là độ ảnh hưởng của từ thứ i trong input lên từ cần dự đoán. Sau đó các giá trị a_i được normalize theo hàm softmax được b_i .



Hình 19.8: Các thành phần attention

Cuối cùng các giá trị v_i được tính tổng lại theo hệ số b_i , $\text{output} = \sum_{i=1}^N b_i * v_i$, trong đó N là số từ trong câu input. Việc normalize các giá trị a_i giúp output cùng scale với các giá trị value.



Hình 19.9: Các bước trong attention

Ở phần encoder, thông thường mỗi từ ở input thì hidden state ở mỗi node được lấy làm cả giá trị key và value của từ đấy. Ở phần decoder, ở node 1 gọi input là x_1 , output y_1 và hidden state s_1 ; ở node 2 gọi input là x_2 , output y_2 . Query là hidden state của node trước của node cần dự đoán từ tiếp theo (s_1). Các bước thực hiện:

- Tính score: $a_i = \alpha(q, k_i)$
- Normalize score: b_i

- Tính output: $\text{output_attention} = \sum_{i=1}^N b_i * v_i$
- Sau đó kết hợp hidden state ở node trước s_1 , input node hiện tại x_2 và giá trị output_attention để dự đoán từ tiếp theo y_2 .

| Name | Alignment score function |
|------------------------|---|
| Content-base attention | $\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]$ |
| Additive(*) | $\text{score}(s_t, h_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; h_i])$ |
| Location-Base | $\alpha_{t,i} = \text{softmax}(\mathbf{W}_a s_t)$
Note: This simplifies the softmax alignment to only depend on the target position. |
| General | $\text{score}(s_t, h_i) = s_t^\top \mathbf{W}_a h_i$
where \mathbf{W}_a is a trainable weight matrix in the attention layer. |
| Dot-Product | $\text{score}(s_t, h_i) = s_t^\top h_i$ |
| Scaled Dot-Product(^) | $\text{score}(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$
Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state. |

Hình 19.10: Một số hàm α hay được sử dụng

Nhận xét: Cơ chế attention không chỉ dùng context vector mà còn sử dụng hidden state ở từng từ trong input với trọng số ảnh hưởng tương ứng, nên việc dự đoán từ tiếp theo sẽ tốt hơn cũng như không sợ tình trạng từ ở xa bị mất thông tin ở context vector.

Ngoài ra các mô hình deep learning hay bị nói là hộp đen (black box) vì mô hình không giải thích được, attention phần nào giúp visualize được kết quả dự đoán, ví dụ từ nào ở output ảnh hưởng nhiều bởi từ nào trong input. Do đó model học được quan hệ giữa các từ trong input và output để đưa ra kết quả dự đoán.

Lúc đầu cơ chế attention được dùng trong bài toán seq2seq, về sau do ý tưởng attention quá hay nên được dùng trong rất nhiều bài toán khác, ví dụ như trong CNN người ta dùng cơ chế attention để xem pixel nào quan trọng đến việc dự đoán, feature map nào quan trọng hơn trong CNN layer... Giống như resnet, attention cũng là 1 đột phá trong deep learning. Mọi người để ý thì các mô hình mới hiện tại đều sử dụng cơ chế attention.

Phần 8 : Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) nghĩa tiếng việt là **Mạng tạo sinh cạnh tranh**

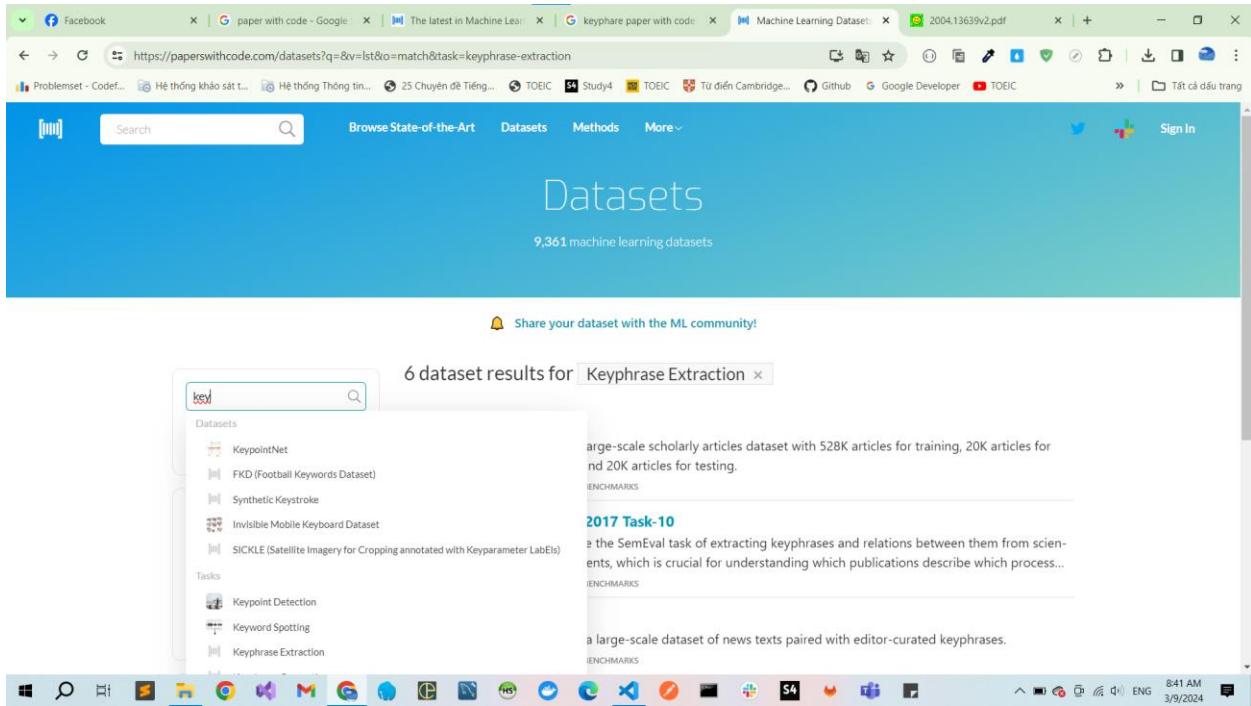
20. Giới thiệu về GAN

21. Deep Convolutional GAN (DCGAN)

22. Conditional GAN (cGAN)

Bibliography

PBL7



Mạng nơ-ron tích chập (CNN) thường được sử dụng trong xử lý ảnh và nhận diện vì các đặc điểm sau:

1. **Cấu trúc chia sẻ trọng số:** CNN sử dụng các lớp tích chập để trích xuất đặc trưng từ dữ liệu đầu vào. Các lớp này có các bộ lọc nhỏ di chuyển trên toàn bộ hình ảnh để phát hiện các đặc điểm cụ thể như cạnh, góc, hoặc đường cong. Việc sử dụng trọng số chia sẻ giúp giảm bớt số lượng tham số cần học và tăng tốc độ tính toán.
2. **Dữ liệu không gian:** Ảnh có cấu trúc không gian, nghĩa là các pixel ở gần nhau thường có mối quan hệ về mặt không gian. CNN có thể hiệu quả trong việc xử lý dữ liệu không gian như ảnh băng cách giữ lại thông tin về không gian và mối quan hệ giữa các pixel.
3. **Scale invariance:** CNN có khả năng nhận biết các đặc trưng ở nhiều tỷ lệ khác nhau trong ảnh. Điều này có ý nghĩa là nó có thể nhận diện đối tượng trong ảnh dù chúng có kích thước khác nhau.

CNN và RNN (Mạng nơ-ron hồi tiếp) khác nhau ở cấu trúc và cách chúng xử lý dữ liệu. Trong khi CNN thích hợp cho dữ liệu không gian như ảnh, RNN thích hợp cho dữ liệu tuần tự như văn bản hoặc âm thanh. CNN thường được sử dụng cho việc phát hiện đặc trưng cục bộ trong dữ liệu không gian, trong khi RNN thích hợp cho việc mô hình chuỗi dữ liệu với các phụ thuộc thời gian.

KNN (K-nearest neighbors) là một thuật toán học máy không được huấn luyện, nghĩa là nó không học ra một mô hình từ dữ liệu huấn luyện mà thay vào đó so sánh các điểm dữ liệu mới với các điểm dữ liệu đã được gán nhãn trong tập huấn luyện và dự đoán dựa trên việc tìm kiếm các điểm gần nhất. KNN không có cấu trúc mạng nơ-ron như CNN hoặc RNN, và nó không yêu cầu việc huấn luyện mô hình.