TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA ĐÀO TẠO CHẤT LƯỢNG CAO
NGÀNH CÔNG NGHỆ THÔNG TIN

----🕮----

MACHINE LEARNING

Đề tài: Handwritten Digit Recognition using Convolutional Neural Networks

GVHD : Vũ Quang Huy
SVTH :
Huỳnh Kiến Minh 15110079
Tôn Nữ Minh Châu 15110019

Tp. Hồ Chí Minh, 04 tháng 12 năm 2018

Mục lục

1. Introduction – Giới thiệu

- Đồ án: Handwritten Digit Recognition using Convolutional Neural Networks
- Limitation:
  - Đề tài: nhận diện chữ số từ 1 đến 10.
  - Kiến thức: áp dụng những kiến thức cơ bản của Neural Networks, kiến thức của một số thuật toán, activation function. cost functions để phục vụ cho bài toán nhận dạng chữ số viết.
  - Sản phẩm: sản phẩm là chương trình được viết bằng ngôn ngữ lập trình Python. Sử dụng bộ data set MNIST để đào tạo và đánh giá.

1.1. Reason – Lý do

Việc lựa chọn đề tài dựa trên 2 yếu tố chính

1.1.1. Scope của đề tài :

Do thời gian chỉ giới hạn trong một học kỳ và phải tiếp thu nhiều kiến thức mới nên nhóm quyết định chọn một đề tài vừa sức, phù hợp với thời gian và độ phức tạp nhưng vẫn có thể áp dụng những kiến thức cơ bản đã học và tự tìm hiểu những kiến thức mới.

Với đề tài Handwritten Digit Recognition không quá phức tạp về mặt nghiệp vụ nên nhóm sẽ có nhiều thời gian hơn để chú trọng vào lý thuyết và tìm hiểu kiến thức kỹ những kiến thức mới để áp dụng vào bài toán.

1.1.2. Đề tài có thể áp dụng với Convolutional Neural Network

Convolutional Neural Network là kiến thức mà nhóm có thể áp dụng cho đề tài , đây cũng là nền tảng để tiếp cận với kiến thức sâu hơn về Machine Learning như Deep Learning nên nhóm sẵn có củng cố kiên thức và tự tìm hiểu thêm về mảng kiến thức mới này.

1.2. Meaning – Ý Nghĩa

- Nắm được khái quát mối quan hệ giữa kiến thức rời rạt để hoàn thành được đề tài.
- Hiểu được bản chất của các thuật toán: ưu điểm và nhược điểm, khi nào và áp dụng như thế nào?
- Rèn luyện khả năng đọc hiểu và áp dụng những kiến thức mới vào đề tài.
- Hiểu và áp dụng Convolutional Neural Network cho bài toán phân loại đơn giản.
- Có kiên thức về Python và tìm hiểu các thư viện hỗ trợ như Keras.
- Thực hành được những kiến thức tự tìm hiểu và là cơ sở để tìm hiểu sâu về lĩnh vực Artificial Intelligence, Data Analyst.

2. Goal – Kết quả

Xây dựng được một model nhận diện được chữ số viết tay đạt tỷ lệ chính xác trên 98%.

3. Objectives – Các mục tiêu cần đạt được

- Tìm hiểu tiến trình của một bài toán Machine Learning, cụ thể là bài toán Supervied Learning (Classification)
  - Traning phase
  - Predict phase
- Tìm hiểu cách mà neural network learn (traning process)
  - Perceptron
  - Multiple-layer Perceptron
    - Forward Propagation
    - Back-Propagation
  - Activation Function: Sigmoid, Rectified linear unit (ReLU), Softmax
    - Cách áp dụng
    - Cách chọn activation phù hợp
  - Cost Function: sự ảnh hưởng của weight, bias trong quá trình học model.
  - Thuật toán tối ưu: Gradient Descent
- Tìm hiểu cách xử lý dữ liệu đầu vào, cụ thể là hình ảnh (image processing) – Convolution Neural Network (CNN)
  - Convolution layer
  - Pooling Layer
  - Fully-connected Layer
- Áp dụng lý thuyết vào chương trình nhận diện chữ viết số viết tay.
  - Tìm hiểu thư viện Keras.
  - Viết chương trình nhận dạng chữ số viết tay bằng Keras.

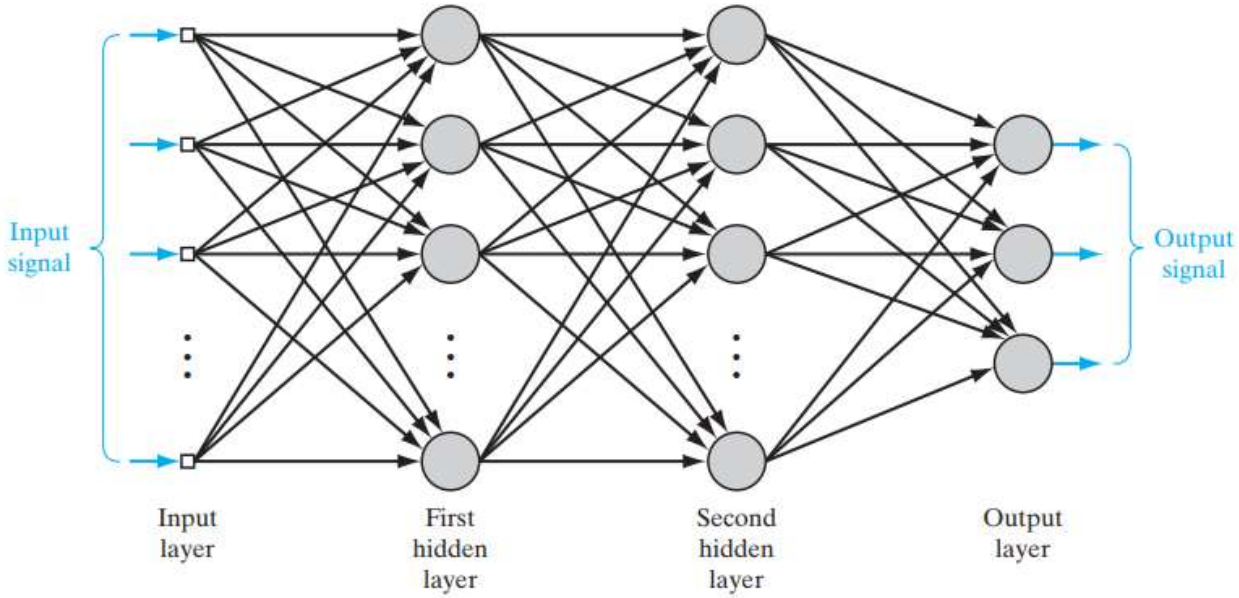
4. Theory – Cơ sở lý thuyết

4.1. Neutral Network

Dựa trên mô hình liên kết giữa các tế bào thần kinh trong não bộ, mô hình neural network là tập hợp các đơn vị kết nối gọi là neuron, các neuron liên kết truyền tín hiệu cho nhau.

Neutral Network được phát triển dựa trên nền tảng Perceptron Learning Algorithm (PLA) hay còn gọi là Perceptron.

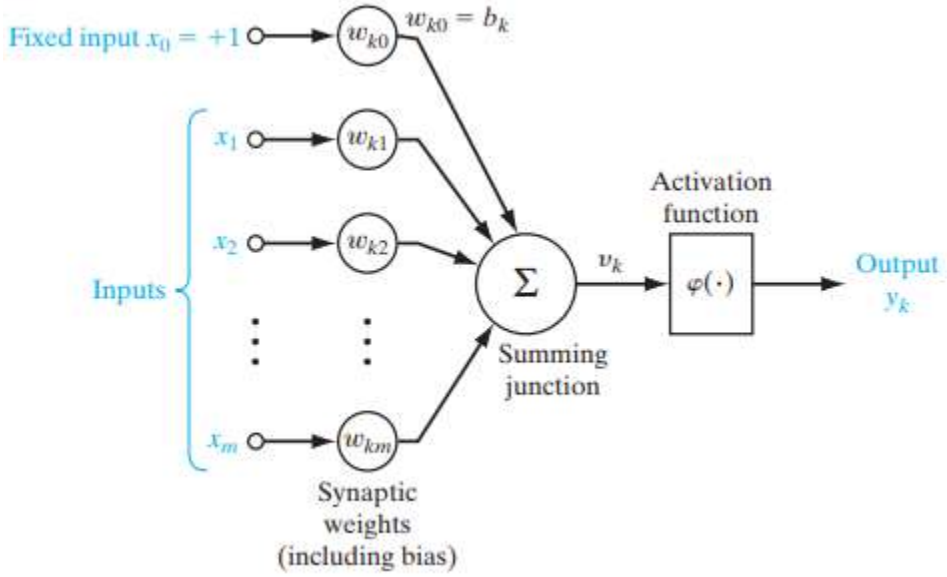
Neural network có cấu trúc Multi-layer Perceptrons. Perceptron có thể hiểu tương đương single-layer neural network.



Hình 4.1.1: Neural network structure

Neuron là một đơn vị xử lý thông tin cơ bản cho hoạt động của neural network

Một neuron trong bất kỳ layer nào trong network đều được kết nối với tất cả các neuron (node) trong layer trước đó – fully connected.



Hình 4.1.2: Mô hình non-linear của một neuron, có nhãn k

Phương trình viết lại từ mô hình:

$$v_k = \sum_{j=1}^{m} w_{kj} x_j$$

và

$$y_k = \varphi(v_k)$$

Với

- $x_1, x_2, \dots, x_m$: input
- $w_{k1}, w_{k2}, \dots, w_{km}$: trọng số tương ứng của neuron k
- b_k : bias
- $\varphi(.)$: activation function
- y_k : output

4.2. Activation Function

Mục đích: giới hạn phạm vi giá trị tới một giá trị hữu hạn cho phép.

Một số hàm activation thường dùng : Sigmoid, Rectified linear unit (ReLU), Softmax.

4.2.1. Sigmoid function

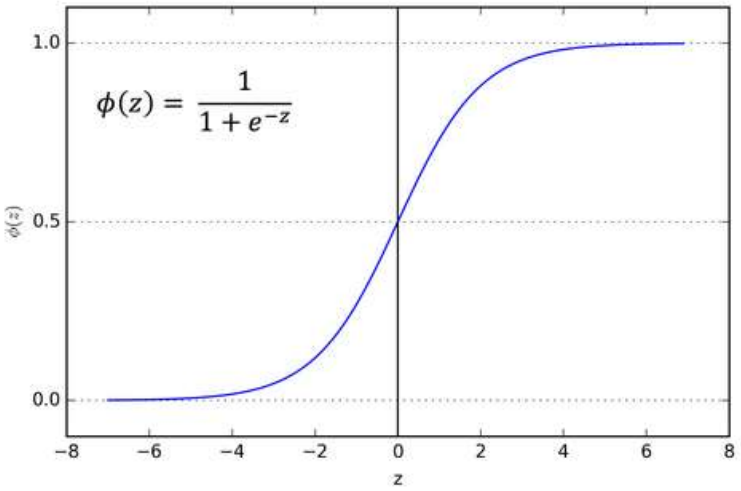
Sigmoid function là một logistic function.

Phạm vi (0, 1). Do đó, nó đặc biệt được sử dụng cho các mô hình mà phải dự đoán xác suất. Vì vậy xác suất của bất cứ điều gì chỉ tồn tại giữa phạm vi 0 và 1.

Áp dụng tốt nhất có bài toán classification với chỉ hai class.

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

$$f'(z) = f(z)(1 - f(z))$$

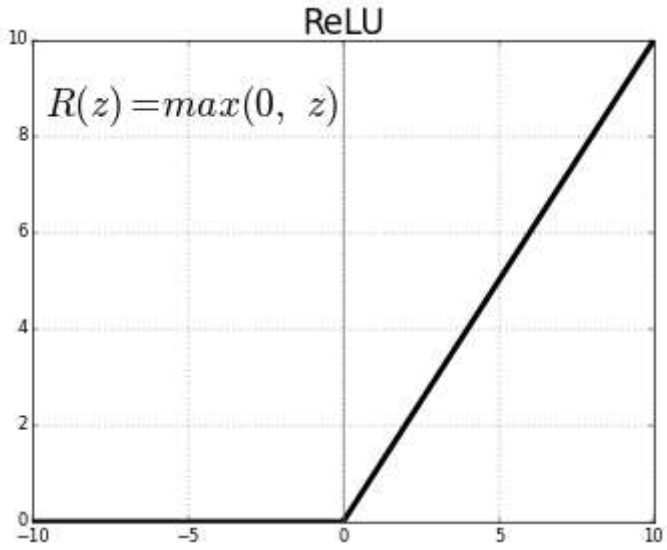


Hình 4.2.1: Đồ thị của Sigmoid Function

4.2.2. Rectified linear unit (ReLU)

ReLU là activation function được sử dụng nhiều nhất trên thế giới hiện nay, nó được sử dụng trong Convolutional Neural Networks (CNN) hay Deep Learning.

$$f(z) = \max(0, z)$$



Hình 4.2.2: Đồ thị của ReLU Function

Phạm vi: $[0, \infty)$

Với bất kỳ input có giá trị âm đưa vào cho hàm ReLU sẽ biến giá trị thành 0, do đó ảnh hưởng đến biểu đồ kết quả khi không ánh xạ các giá trị âm một cách thích hợp.

4.2.3. Softmax Function

Hàm softmax có output nằm trong khoảng từ 0 đến 1, giống như hàm sigmoid. Nhưng nó chia mỗi output sao cho tổng số output bằng 1.

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1,2,..,K$$

Với z: K-dimensional vector

4.3. Cost Function

Cost function là độ sai lệch

$$L(w,b) = \frac{1}{2n}\sum_{x} \|y(x) - a\|^2$$

Với

- n: số lượng traning input
- a: output (vector) với input x
- x: input (vector)

$L(w,b) \approx 0$ khi giá trị output gần như sấp xỉ bằng $y(x)$. Từ đó cho thấy thuật toán training hoạt động tốt hay không dựa vào việc tìm được weights and biases để $L(w,b) \approx 0$.

Để tránh sai số cao cần làm tối thiểu chi phí mất mát thấp nhất có thể

➔ để kiểm tra lại độ chính xác của việc học (phân loại).

Sử dụng thuật toán như Gradient Descent, SGD, … để tối ưu hàm mất mát.

4.4. Thuật toán tối ưu

* Local minimum: điểm cực tiểu (nghiệm của đạo hàm $f'(x) = 0$).

* Global minimum: điểm mà tại đó hàm số đạt cực tiểu.

Việc tối ưu hàm mất mát $L(\theta)$ là đi tìm giá trị của tham số θ sao cho hàm L đạt giá trị nhỏ nhất, nói cách khác là đi tìm Global minimum. Nhưng việc giải phương trình đạo hàm bằng 0 để tìm nghiệm là bất khả thi. Do đó cần một phương pháp tổng quát để tính được xấp xỉ giá trị của θ mà tại đó L đạt giá trị nhỏ nhất.

Gradient Descent và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

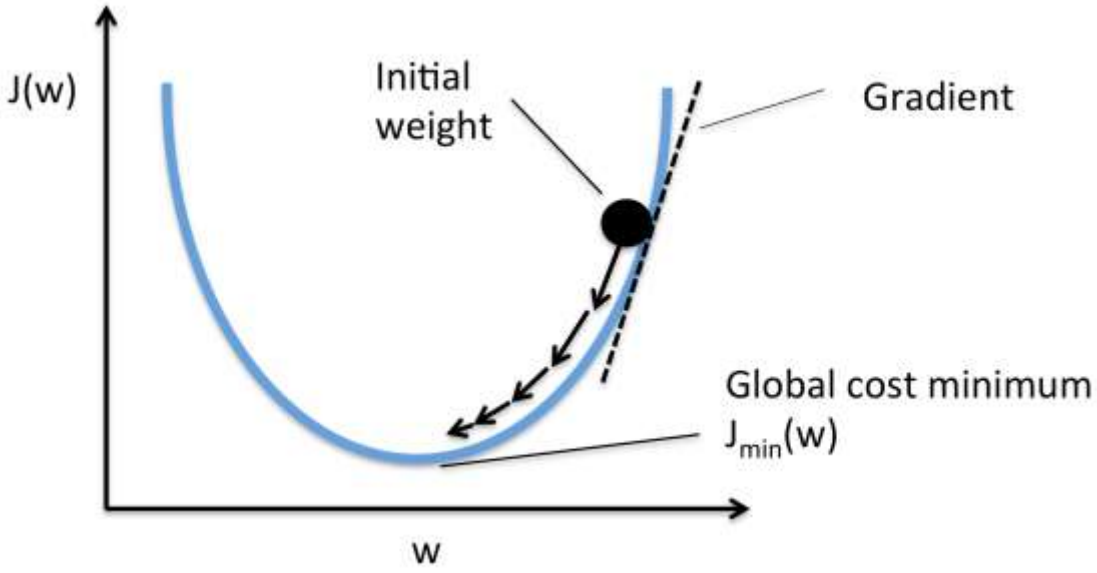
4.4.1. Gradient Desent

Giả sử cần tìm global minimum cho hàm $f(\theta)$, với θ (khởi tạo $\theta_0 = 0$) là một vector.

Ta có đạo hàm của hàm mất mát tại θ bất kỳ bằng: $\nabla_\theta f(\theta)$.

Ta có quy tắc cập nhập di chuyển:

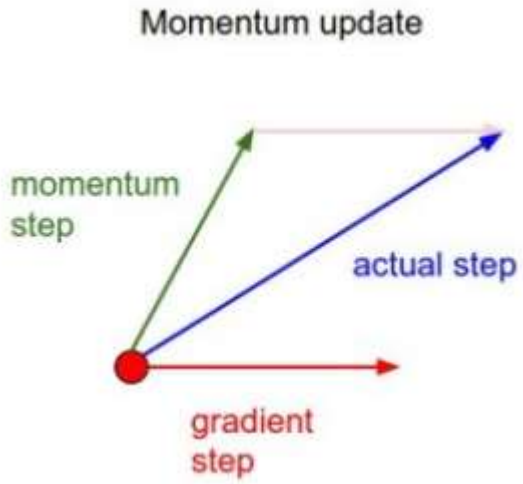
$$\theta = \theta - \eta \nabla_\theta f(\theta)$$



Hình 4.4.1: Đồ thị mô phỏng thuật toán Gradient Desent

4.4.2. Các thuật toán tối ưu cho Gradient Densent

- **Momentum**



Lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm hiện tại.

Gọi lượng thay đổi ở thời điểm t là v_t, ta có vị trí mới được cập nhập:

$$\theta = \theta - v_t$$

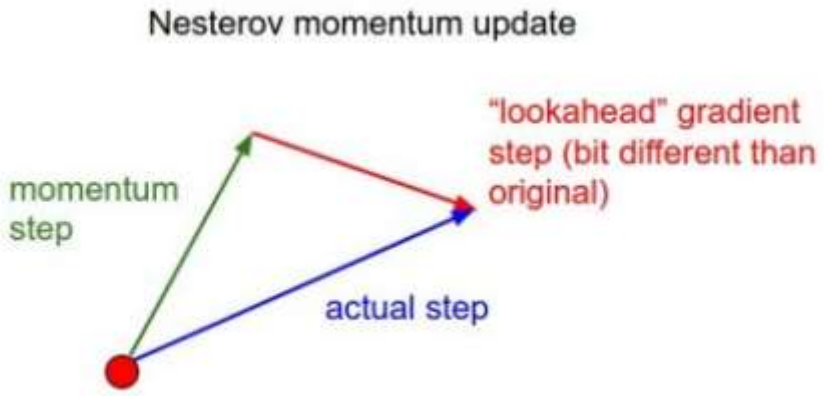
Với

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta f(\theta)$$

Trong đó:

- $\gamma = 0.9$
- v_{t-1}: vận tốc tại thời điểm trước đó (đà).
- $\nabla_\theta f(\theta)$: là độ dốc của điểm trước đó.

- **Nesterov accelerated gradient (NAG)**



Lượng thay đổi là tổng của hai vector: momentum vector và gradient ở thời điểm được xấp xỉ là điểm tiếp theo

Công thức cập nhập của NAG như sau:

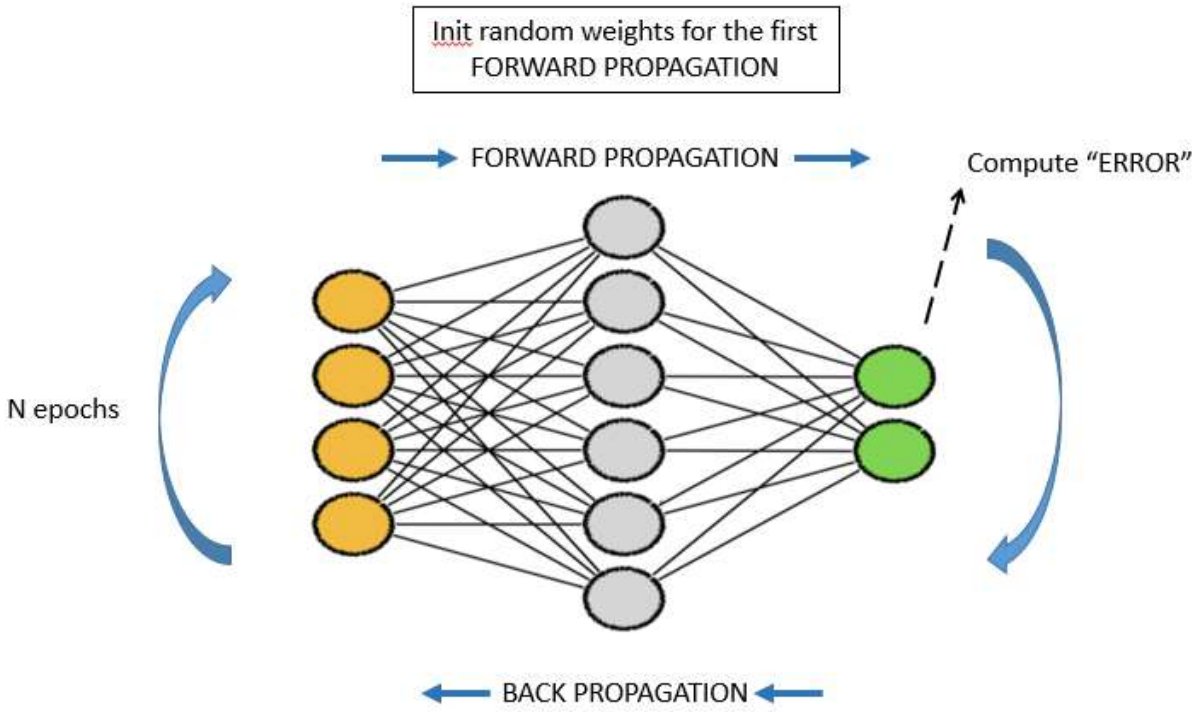
$$\theta = \theta - v_t$$

Với

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta f(\theta - \gamma v_{t-1})$$

4.5. How neural network learn

Quá trình học của neural network có 2 phần chính là Forward Propagation và Back Propagation



Hình 4.5.1: Mô hình thể hiện quá trình học của neural network

*Một Epoch là khi tất cả dữ liệu được đưa vào mạng neural network 1 lần.

Chu kỳ hoạt động của quá trình trên được cụ thể với mẫu training $\{(x(n), d(n))\}_{n=1}^{N}$:

1) **Initialization:** khởi tạo random các trọng số (w) trong khoảng (0, 1)
2) **Training data**: đưa dữ liệu vào input layer
3) **Forward Computation**:

- Ta có input vector $x(n)$, desired output vector $d(n)$
- Neuron j layer l có:

$$v_j^l(n) = \sum_i w_{ji}^l(n).y_i^{l-1}(n) \quad (4.5.1)$$

Với

- $y_i^{l-1}(n)$: output của neuron i layer $(l-1)$ tại lần lặp thứ n
- $w_{ji}^l(n)$: trọng số của neuron j layer l
- Với $i = 0$, ta có $y_0^{l-1}(n) = 1$ và $w_{j0}^l(n) = b_j^l$ (bias áp dụng cho neuron j layer l)

- Qua activation function, output của neuron j layer l bằng:

$$y_j^l(n) = \varphi_j\left(v_j(n)\right) \quad (4.5.2)$$

- Đối với neuron j thuộc hidden layer ($l = 1$) đầu tiên thì đặt:

$$y_j^0(n) = x_j(n)$$

với $x_j(n)$ là phần tử thứ j của input vector $x(n)$.

- Đối với neuron j thuộc output layer ($l = L$) đầu tiên thì:

$$y_j^L(n) = y_j(n)$$

- Tính độ sai lệch của neuron j (với neuron j thuộc output layer, $l = L$)

$$e_j(n) = d_j(n) - y_j(n) \quad (4.5.3)$$

với $d_j(n)$ là phần tử thứ j trong desired output vector $d(n)$.

- Lost function (hàm mất mát)

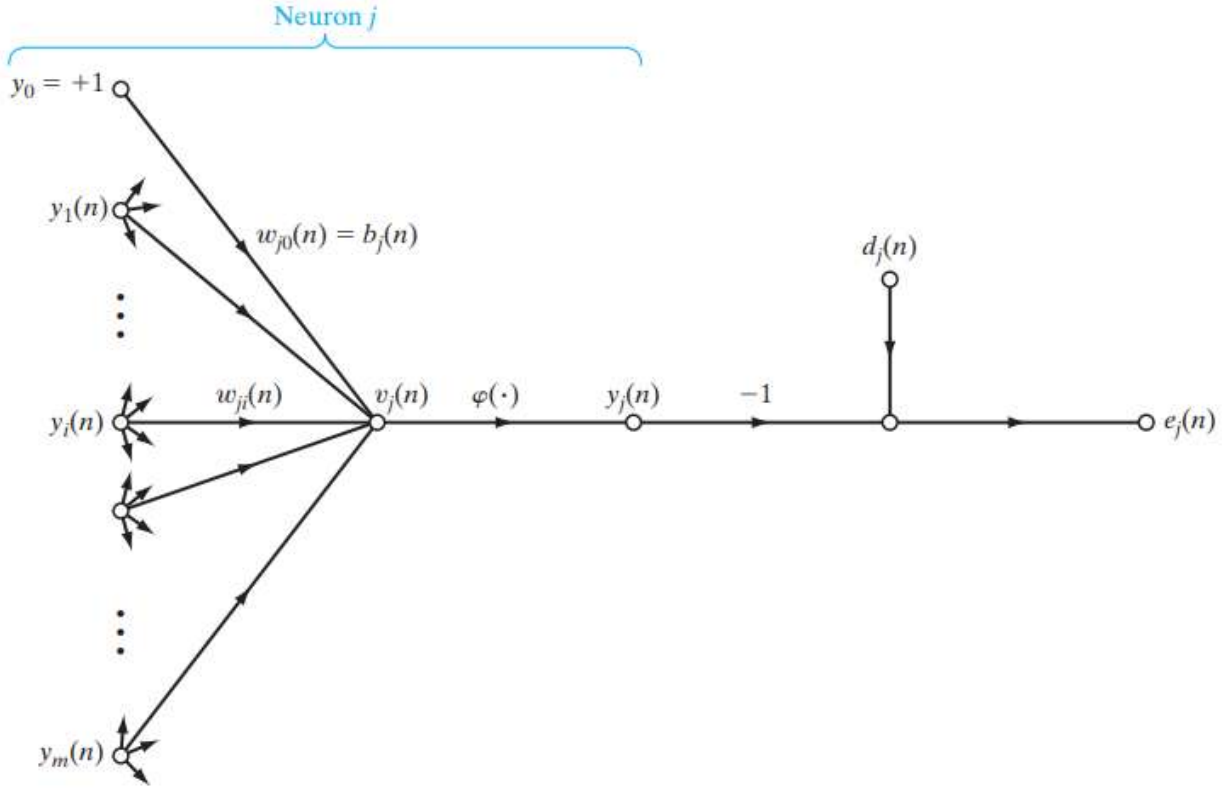
$$E(n) = \frac{1}{2}\sum_{j\epsilon C} e_j^2(n) \quad (4.5.4)$$

Với C là tập các neuron của output layer.

4) **Backward Computation**:

Thuật toán backpropagation áp dụng để hiệu chỉnh $\Delta w_{ji}(n)$ của trọng số $w_{ji}(n)$. Ta có 2 trường hợp tính toán xảy ra, được giải thích rõ dưới đây.

Neuron j là một Output Node



Hình 4.5.2: Biểu đồ luồng tín hiệu làm nổi bật các chi tiết của neuron j (thuộc output layer)

- Ta có quy tắc tính hiệu chỉnh các trọng số:

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)} \quad (4.5.5)$$

Theo quy tắc chuỗi tính toán (chain rule), ta có đạo hàm từng phần của $\frac{\partial E(n)}{\partial w_{ji}(n)}$ như sau:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} = -e_j(n).\varphi_j'\left(v_j(n)\right) y_j(n) (4.5.6)$$

Với :

- Đạo hàm của phương trình (4.5.4)

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n)$$

- Đạo hàm của phương trình (4.5.3)

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

- Đạo hàm của phương trình (4.5.2)

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)$$

- Đạo hàm của phương trình (4.5.1)

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_j(n)$$

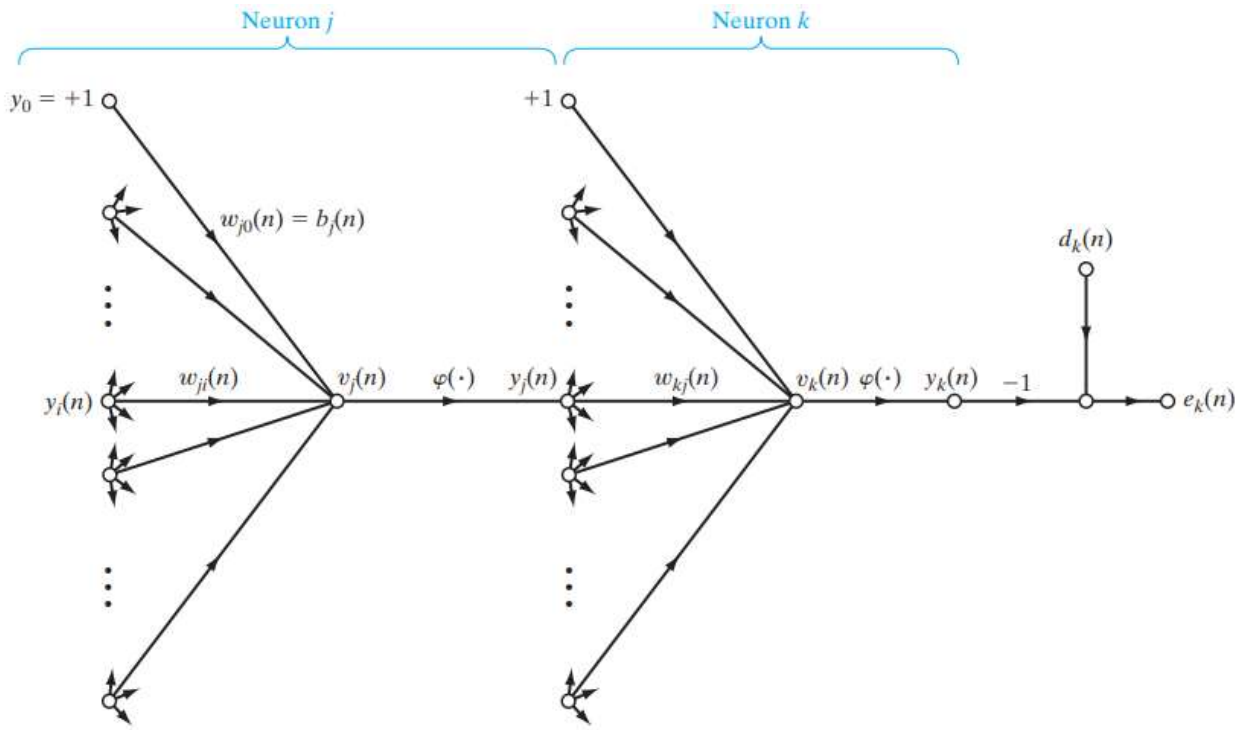
- Lại có phương trình tính local gradient $\delta_j(n)$

$$\delta_j(n) = \frac{\partial E(n)}{\partial v_j(n)} = \frac{\partial E(n)}{\partial e_j(n)} . \frac{\partial e_j(n)}{\partial y_j(n)} . \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) . \varphi_j'\left(v_j(n)\right) \quad (4.5.7)$$

Từ 3 phương trình (4.5.5),(4.5.6) và (4.5.7), suy ra:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_j(n)$$

Neuron j là một Hidden Node



Hình 4.5.3 : Biểu đồ luồng tín hiệu làm nổi bật các chi tiết của neuron k (thuộc output layer) kết nối với neuron j (thuộc hidden layer).

Dựa theo hình trên, các phương trình được định nghĩa lại như sau:

- Với neuron k là một output node, ta có

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n)\, y_j(n)$$

Với:

- m là tổng số input được áp dụng cho neuron k ,bao gồm bias
- bias : $b_k(n) = w_{k0} = +1$

$$\Rightarrow \frac{\partial v_j(n)}{\partial y_j(n)} = w_{kj}(n) \quad (4.5.8)$$

- Độ sai lệch của neuron k

$$e_k(n) = d_k(n) - y_k(n)$$

$$= d_k(n) - \varphi_k\big(v_k(n)\big)$$

$$\Rightarrow \frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k'\big(v_k(n)\big) \quad (4.5.9)$$

- Lost function (hàm mất mát) tại output layer

$$E(n) = \frac{1}{2}\sum_{k\epsilon C} e_k^2(n)$$

Với C là tập các neuron của output layer.

$$\Rightarrow \frac{\partial E(n)}{\partial e_k(n)} = \sum_{k} e_k(n) \quad (4.5.10)$$

Từ (4.5.8), (4.5.9) và (4.5.10), áp dụng quy tắc chuỗi tính toán (chain rule), ta có đạo hàm của Lost function :

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\partial E(n)}{\partial e_k(n)}\frac{\partial e_k(n)}{\partial k_j(n)}\frac{\partial v_j(n)}{\partial y_j(n)}$$

$$= -\sum_{k} e_k(n)\, \varphi_k'\big(v_k(n)\big) w_{kj}(n) \qquad (4.5.11)$$

Từ phương trình (4.5.7), phương trình (4.5.11) được viết lại như sau:

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_{k} \delta_k(n)\, w_{kj}(n)$$

- Local gradient tại neuron j

$$\delta_j(n) = -\frac{\partial \mathrm{E(n)}}{\partial \mathrm{y_j(n)}}\frac{\partial \mathrm{y_j(n)}}{\partial \mathrm{v_j(n)}} = -\frac{\partial \mathrm{E(n)}}{\partial \mathrm{y_j(n)}}\varphi_j'\left(\mathrm{v_j(n)}\right)$$
$$= \varphi_j'\left(\mathrm{v_j(n)}\right)\sum_k \delta_k(n)\,\mathrm{w_{kj}(n)}$$

Tổng kết lại:

- Tính hiệu lượng hiệu chỉnh $\Delta w_{ji}(n)$ có công thức như sau

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning\ rate \\ parameter \\ \eta \end{pmatrix} \times \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} input\ signal \\ of\ neuron\ j \\ \mathrm{y_j(n)} \end{pmatrix}$$

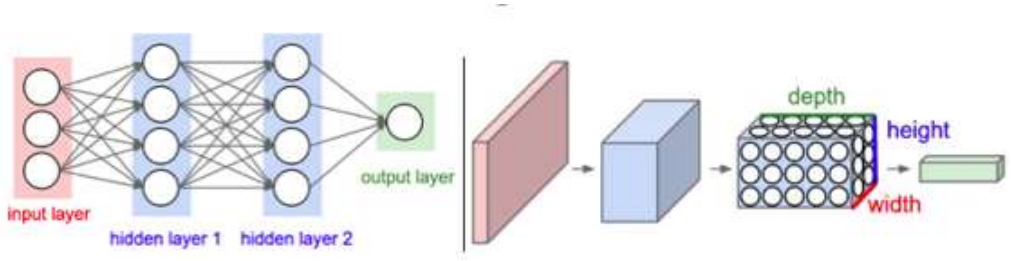
- Tính toán các local gradient của network:

$$\delta_j^l(n) = \begin{cases} e_j^L(\mathrm{n}).\varphi_j'\left(v_j^L(\mathrm{n})\right) & ,neuron\ j\ tại\ output\ layer\ L \\ \varphi_j'\left(v_j^L(\mathrm{n})\right)\sum_k \delta_k^{l+1}(n).w_{kj}^{l+1}(n) & ,neuron\ j\ tại\ hidden\ layer\ l \end{cases}$$

- Tùy theo thuật toán chọn (GD or SGD, …), ta thực hiện việc cập nhập trọng số tương ứng.

5) **Iteration**: Lặp lại công việc tính toán Forward và Backward (mục 3,4 phía trên) trong mỗi epoch cho đến khi hết n epoch (n là số lần lặp được cho trước). Bộ dữ liệu traning phải trộn ngẫu nhiên sau mỗi epoch. Các thông số momentum, learning-rate đều được chỉnh khi số lần lặp tăng lên, thường là giảm.

4.6. Convolution Neural Network – CNN



Hình: Mô hình Neural network (trái) và CNN (phải)

Convolutional Neural Network –CNN là một loại Artificial Neural Network.CNN dùng để xử lý các dữ liệu 2D,3D như: image,audio, text. Cũng như

Neural Network truyền thống, CNN gồm có: input layer, các hidden layer và outut layer.

Nhưng tập hidden layer trong CNN bao gồm thêm convolutional layers, pooling layers, fully connected layers.

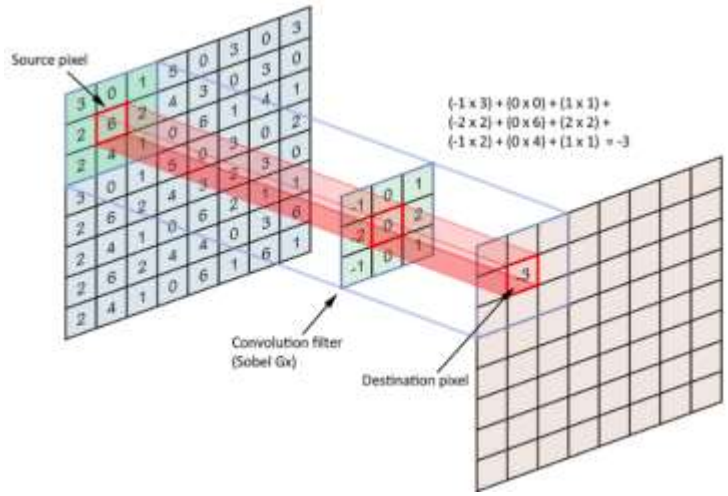
4.6.1. Convolutional Layer

Mục đích: trích xuất ra các feature/đặc tính cụ thể.

Tập các feature map, mỗi feature map là 1 bản scan thu nhỏ đặc tính cụ thể của dữ liệu input.

Quá trình scan: dựa vào convolution filter/kernel, ma trận sẽ quét qua ma trận dữ liệu đầu vào, từ trái qua phải, trên xuống dưới để lấy những đặc tính cụ thể.

Kết quả đưa qua activation funciton (sigmoid, relu,…), kết quả sẽ là một con số cụ thể. Tập hợp các con số này lại là 1 ma trận nữa, chính là **feature map**



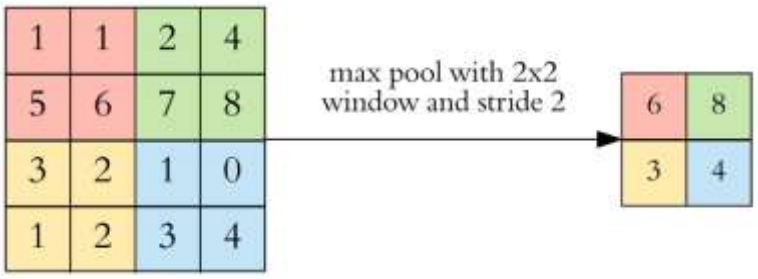
Hình: Mô tả tính toán Convolution

4.6.2. Pooling Layer

Mục đích: giảm số hyperparameter cần phải tính toán, từ đó giảm thời gian tính toán, tránh overfitting.

Có 2 loại Pooling: max pooling (dùng nhiều nhất), average pooling.

Max pooling: lấy giá trị lớn nhất trong một *pooling window*.



Hình: Mô tả tính toán max pooling

4.6.3. Fully Connected Layer

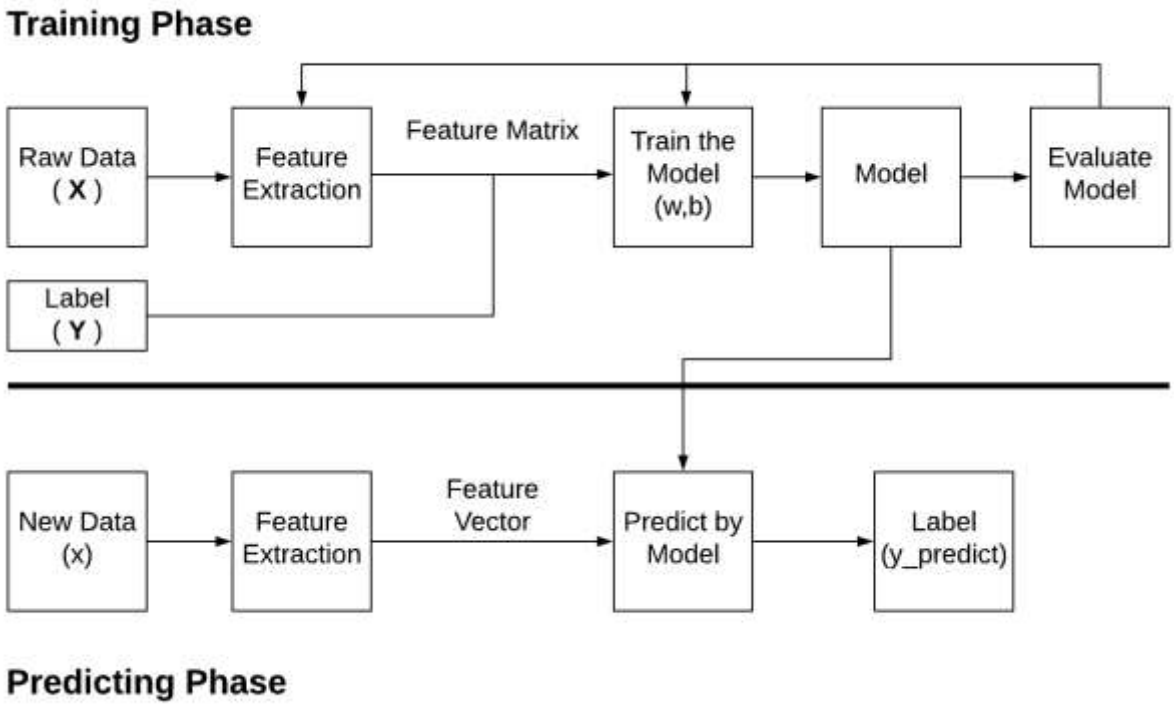
Mục đích: kết nối mọi neuron trong một lớp với mọi neuron trong một lớp khác.

Về nguyên tắc, nó giống như multi-layer perceptron neural network truyền thống.

Thực hiên sau convolutional layers + pooling layers. Vì ouput của 2 layer này có kích thước 3D, còn output của fully connected layer là mảng 1D nên phải Flatten output của pooling layer cuối cùng.

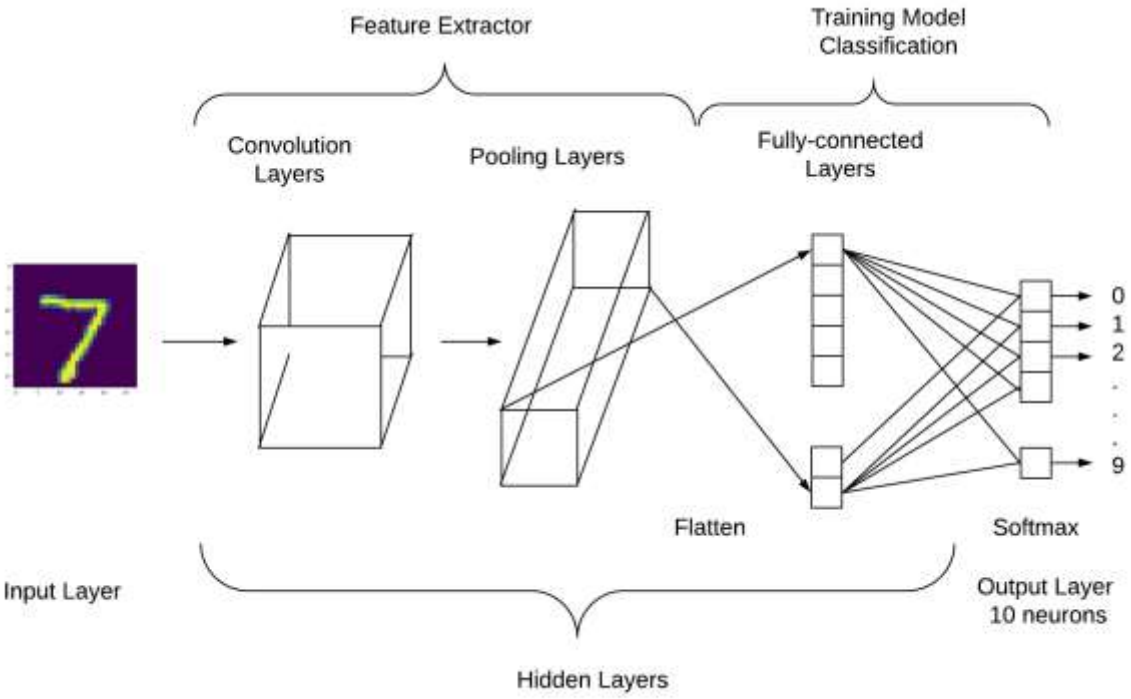
Flatten chỉ đơn giản là sắp xếp khối 3D của các số thành một vector 1D (n hàng ,1 cột).

5. Diagram cho bài toán Supervised Learning (Classification)



6. Áp dụng đề tài

6.1. Mô hình Convolution Neural Network cụ thể áp dụng cho đề bài



- Số neuron của Output Layer bằng số class phân loại. Đề bài nhận dạng chữ số viết tay, ta có 10 số : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

➔ Số neuron của Output Layer: 10 (neuron)

6.2. Thư viện Keras

6.2.1. Giới thiệu

Keras là một neural networks API cấp cao, được viết bằng Python và có khả năng chạy trên TensorFlow, CNTK hoặc Theano. Nó được phát triển với trọng tâm là cho phép thử nghiệm nhanh. Có thể đi từ ý tưởng đến kết quả với sự chậm trễ ít nhất có thể là chìa khóa để thực hiện nghiên cứu tốt.

Trang chủ: https://keras.io/

6.2.2. Một số chức năng lưu ý (áp dụng khi code)

6.2.2.1. Sequential model methods

Compile : cấu hình model để training

```
compile(optimizer, loss=None, metrics=None )
```

- Optimizer : đều được xây dựng dựa trên thuật toán Gradient Descent (vd: SGD, RMSprop, Adam, Adamax)
- Loss: hàm loss function để áp dụng cho quá trình training, mean squared error với bài toán regression, binary crossentropy cho bài toán classification, categorical crossentropy với bài toán multiclass-classification và sparse categorical crossentropy cũng cho bài toán multiclass-classification nhưng với label chưa được đưa về dạng one-hot
- Metrics: Danh sách các số liệu được mô hình đánh giá trong quá trình training và kiểm tra, thường dùng metrics=['accuracy']

Fit: training model với số lần epoch (các lần lặp trên tập dữ liệu)

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, validation_data=None)
```

- X: Mảng dữ liệu training
- Y: Mảng label của dữ liệu training
- Batch_size: Số lượng samples cho mỗi cập nhật gradient
- Epochs: Số lượng epochs để training model. Epochs là một lần lặp trên toàn bộ dữ liệu x và y được cung cấp
- Verbose: là một số nguyên gồm 0, 1, hoặc 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch
- Validation_data: đánh giá model ở giai đoạn cuối của mỗi epochs. Samples được đánh giá phải khác samples training (x_valuation, y_valuation)

Evaluate: Trả về giá trị mất mát (loss value) và giá trị số liệu cho mô hình ở giai đoạn đánh giá model

```
evaluate(x=None, y=None)
```

- X: Mảng dữ liệu để đánh giá (khác dữ liệu training)
- Y: Mảng label dữ liệu để đánh giá

Predict : Tạo dự đoán đầu ra cho các input samples.

```
predict(x)
```

- X: Dữ liệu để predict

6.2.2.2. Convolutional Layers

Conv2D: 2D convolution layer

```
keras.layers.Conv2D(filters, kernel_size, data_format=None, activation=None, use_bias=True)
```

Lớp này tạo ra a kernel convolution được kết hợp với the layer input trên một kích thước không gian duy nhất để tạo ra một tensor đầu ra.

- Filters: kích thước của không gian đầu ra
- Kernel_size; chỉ định chiều cao và chiều rộng của 2D convolution window
- Data_format: kích thước trong các đầu vào
- Activation: Activation function để sử dụng

6.2.2.3. Available activations (Một số activation dùng trong projects)

Softmax:

```
keras.activations.softmax(x, axis=-1)
```

- **x**: Input tensor.
- **axis**: là số nguyên, trục dọc theo đó chuẩn hóa softmax được áp dụng

relu

```
keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0.0)
```

- **x**: Input tensor.
- **alpha**: là kiểu float. Độ dốc của phần tiêu cực. Mặc định là 0
- **max_value**: là kiểu float. Ngưỡng bão hòa
- **threshold**: float. Threshold value for thresholded activation.

Tài liệu tham khảo

1. Simon Haykin (2009), *Neural Networks and Learning Machines Third Edition*, Pearson
2. https://github.com/tnmchau/TLCN2018-NeuralNetwork
3. https://en.wikipedia.org/wiki/Convolutional_neural_network
4. http://cs231n.github.io/convolutional-networks/
5. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2
6. https://keras.io/