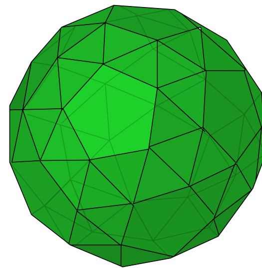
Projects in Mathematics and Applications

NEURAL NETWORK

Hoàng Quốc Thái [*]   [†]Nguyễn Nhật Minh Khôi
Trần Gia Phong [‡]   [§]Phan Thị Mỹ Linh

Ngày 26 tháng 8 năm 2018



* University of Washington
† Trường THPT Chuyên Hoàng Lê Kha
‡ Trường Phổ thông Năng Khiếu
§ Trường THPT Gia Định

Lời cảm ơn

Xin chân thành cảm ơn các anh chị trong Ban tổ chức của Trại hè Toán Khoa học PiMA đã hỗ trợ nhóm trong suốt quá trình hoàn thành dự án. Bên cạnh đó, xin cảm ơn 2 nhà tài trợ chính là Trường Đại học Khoa Học Tự Nhiên ĐHQG-HCM và Trung tâm Giáo dục TITAN Education đã luôn ủng hộ và đồng hành cùng PiMA trong suốt những ngày trại vừa qua.

Tóm tắt nội dung

Mạng Neuron nhân tạo (Artificial Neural Network - ANN) là mô hình xử lý thông tin được mô phỏng theo hệ thống neuron trong bộ não con người, bao gồm số lượng lớn các neuron được gắn kết để xử lý thông tin. ANN giống như bộ não con người, được học bởi kinh nghiệm thông qua huấn luyện, có khả năng lưu giữ những kinh nghiệm đó và sử dụng những nó trong việc dự đoán các dữ liệu chưa biết.
Tài liệu dưới đây sẽ giới thiệu về Hệ thống Neural Network,cụ thể là Multi Layer Perceptron, cách hoạt động cũng như một số cách cải tiến.

Mục lục

1 Đặt vấn đề 1
2 Perceptron 1
2.1 Perceptron 1
2.2 Weights và Bias 1
3 Multi Layers Perceptron (MLP) 2
3.1 Layers 2
3.2 Units (Nodes) 2
3.3 Activation Function 3
3.4 Cost Function (Loss Function) 4
3.5 Gradient Descent 5
3.6 Backpropagation 6
4 Tối ưu hóa (Optimization) 6
4.1 Dừng sớm (Early Stopping) 7
4.2 Sinh dữ liệu 7
4.3 Mini-batch Gradient Descent 7
4.4 L1, L2 Regularization: 7
4.5 Dropout 8
4.6 Cross Entropy Loss Function 8
5 Cách hoạt động của Classification Neural Network: 8
6 Áp dụng mô hình 9
7 Kết luận đánh giá 12

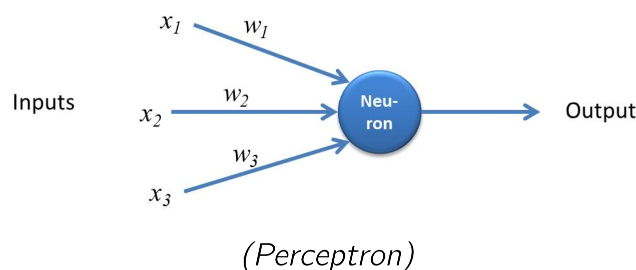
1 Đặt vấn đề

Khi nhìn vào một bức ảnh chụp các kí tự toán học, ta nhận ra được kí hiệu trong bức ảnh là gì một cách dễ dàng. Thế nhưng, chúng ta lại thường không thể tự lý giải được lý do chúng ta nghĩ hình ảnh đó là kí hiệu toán học mà lại không phải là những vật khác? Có thể giải thích rằng, phải có một điểm đặc trưng giúp ta nhìn nhận những kí hiệu này.
Xét góc nhìn từ máy tính, nếu có thể tìm ra những điểm đặc trưng của một vật, thì máy tính cũng có thể phân loại được các sự vật. Người ta gọi việc tìm ra những điểm đặc trưng này để phân loại là Mạng Neuron dùng để phân loại hay Classification Neural Network.

2 Perceptron

2.1 Perceptron

Mạng neuron nhân tạo được tạo nên từ nhiều neuron đơn lẻ, gọi là Perceptron. Cấu tạo của perceptron được mô phỏng theo neuron sinh học trong bộ não con người. Một neuron có thể nhận các dữ liệu vào (Input), xử lý chúng và cho ra một kết quả (Output).



(Perceptron)

Tương tự như hình vẽ, một perceptron sẽ nhận nhiều input là $x_1, x_2, x_3, \ldots$ (0 hoặc 1) và xuất ra kết quả output là 0 hoặc 1. Quá trình xuất ra output được quyết định bởi input, các trọng số (weight) và các độ lệch (bias).

2.2 Weights và Bias

- Trọng số (weight, kí hiệu: w): Là thành phần thể hiện mức độ quan trọng của dữ liệu đầu vào đối với quá trình xử lý thông tin và mức độ quan trọng của layer trước so với layer sau.
- Độ lệch (Bias, kí hiệu: b): Là một ngưỡng giá trị quyết định kết quả đầu ra.

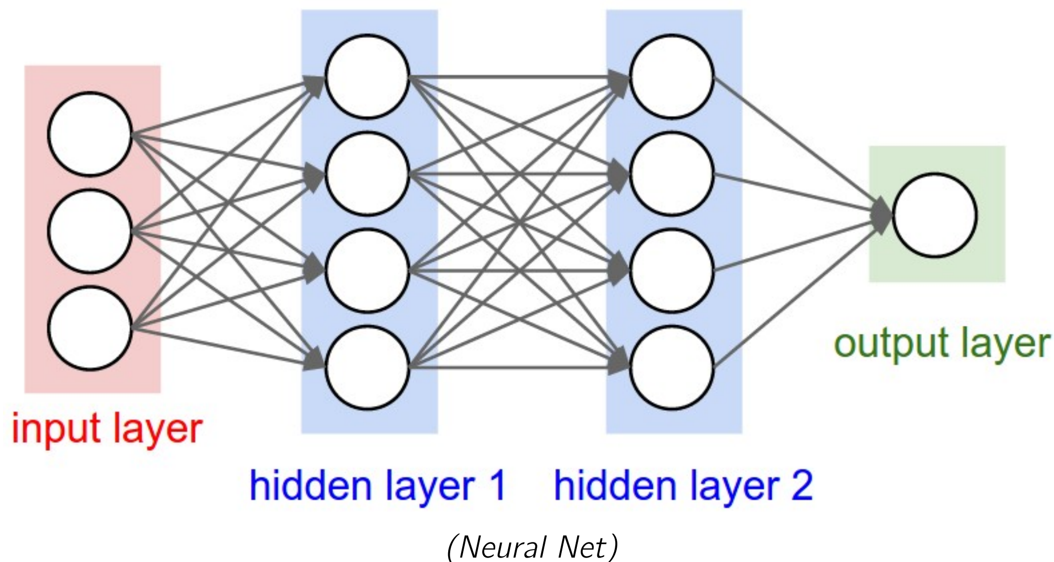
Output sẽ được tính đơn giản như sau:

$$x = \begin{cases} 0, \text{if} \sum_i w_i x_i + b \leq 0 \\ 1, \text{if} \sum_i w_i x_i + b > 0 \end{cases}$$

3 Multi Layers Perceptron (MLP)

3.1 Layers

ANN được cấu tạo từ nhiều tầng perceptron (multilayer perceptron), bao gồm 3 tầng chính: Input layer, Hidden layers và Output layer. Trong đó, chỉ có duy nhất 1 Input layer và 1 Output layer nhưng có thể có nhiều Hidden layers.
Số lượng layer (kí hiệu: L) được tính bằng tổng số hidden layers và output layer, ta không tính input layers.



(Neural Net)

Trong MLP, đầu vào của layer *n* chính là đầu ra của layer *n - 1*. Bởi vậy, ta sẽ có kết quả cuối cùng là một giá trị phức tạp được xây dựng dựa trên kết quả của tất cả các layers. Chính vì thế, đối với những bộ dữ liệu đơn giản thì chỉ cần sử dụng một số lượng nhỏ layer là có thể giải quyết. Nhưng đối với những bộ dữ liệu phức tạp như: time-series hoặc computer vision thì việc có nhiều layer sẽ giúp máy học chi tiết và hiệu quả hơn.
Tuy nhiên, việc có quá nhiều layers sẽ khiến cho mạng neuron khó huấn luyện. Vì khi học quá chi tiết sẽ dẫn tới việc máy học vẹt cả những chi tiết không cần thiết, dẫn đến làm giảm độ chính xác của thuật toán. Hiện tượng này là overfit, sẽ được đề cập ở phần 4.1.

3.2 Units (Nodes)

Một node hình tròn trong một layer được gọi là một unit. Đầu vào của các hidden layer được ký hiệu là z, đầu ra được kí hiệu là a (giá trị sau khi đã áp dụng hàm activation lên z). Đầu vào của unit thứ i trong layer l được kí hiệu là z_i^l. Tương tự, đầu ra được kí hiệu là a_i^l.
Đầu ra của unit được tính như sau:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$

(k: unit thứ k ở layer l; j: unit thứ j ở layer l - 1)

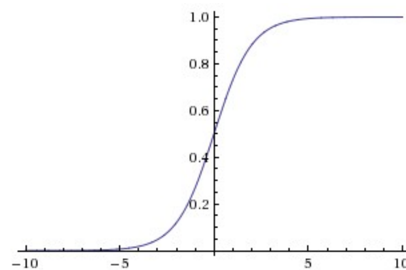
Đầu ra của một layer được tính như sau:

$$z^l = w^l a^{l-1} + b^l$$

3.3 Activation Function

Perceptron chỉ có hai giá trị là 0 và 1. Trong khi đó, thực tế kết quả dự đoán không chỉ có 0 và 1, vì độ chính xác chỉ mang tính tương đối. Do đó, cần một hàm để biểu diễn kết quả một cách chính xác hơn.
Có nhiều hàm activation như Sigmoid, ReLU và nhiều hàm khác.

- **Sigmoid**.



(Sigmoid)

Công thức của hàm Sigmoid:

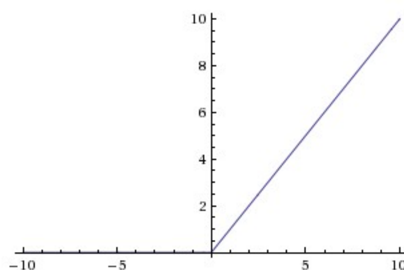
$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid là một hàm phi tuyến tính. Thông qua nó, giá trị đầu vào và đầu ra thay vì chỉ là 0 hoặc 1 sẽ trở thành một giá trị nằm trong khoảng từ 0 đến 1. Hơn nữa, nó có đạo hàm khá đẹp.

$$\sigma[z]' = (\sigma[z]) \cdot (1 - \sigma[z])$$

Tuy nhiên, một nhược điểm dễ nhận thấy là khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương), gradient của hàm số này sẽ rất gần với 0. Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật, đây gọi là hiện tượng Vanshing Gradient. Do đó, thường hàm Sigmoid chỉ được sử dụng ở layer cuối.

- **ReLU (Rectified Linear Unit)**



(ReLU)

Hiện nay hàm activation phổ biến hơn chính là hàm ReLU, có dạng:

$$f(x) = max(0, x)$$

Không như hàm Sigmoid, ReLU không gặp hiện tượng Vanishing Gradient vì đạo hàm của hàm ReLU là một hằng số. Ngoài ra, một tính chất đặc biệt quan trọng để giải thích sự phổ biến của hàm ReLU chính là khả năng "làm thưa" mạng neuron của ReLU: vì khi đầu vào bé hơn 0, output của hàm ReLU là 0, hay neuron đó coi như không hoạt động. Tính chất thưa thớt của mạng neural giúp mô hình hoạt động nhanh và hiệu quả hơn.

Nảy sinh một câu hỏi Tại sao phải sử dụng các hàm activation phi tuyến tính ?
Ta biết, ở mỗi unit của layer nào đó, ta nhận vector đầu vào, sau đó biến đổi để trả về một giá trị đầu ra.
Khi không sử dụng hàm activation thì:

$$\begin{aligned} z^1 &= w^1 z^0 + b^1 \\ z^2 &= w^2 z^1 + b^2 \\ &= w^2(w^1 z^0 + b^1) + b^2 \\ &= w^2 w^1 z^0 + w^2 b^1 + b^2 \end{aligned}$$

$$\text{Đặt: } \begin{cases} W = w^2 w^1 \\ B = w^2 b^1 + b^2 \end{cases}$$

$$\text{Ta có: } z^2 = Wz^0 + B \text{ (hàm tuyến tính)}$$

Từ đó thấy rằng, khi không dùng hàm activation mọi layer sau luôn là kết quả của một tổ hợp tuyến tính của layer trước. Do đó kết quả trả về là một hàm tuyến tính là siêu phẳng phân chia không gian R^n thành 2 không gian con. Vậy nên chỉ giải được những bài toán phân loại đơn giản. Để giải quyết những bài toán phức tạp thì ta cần một hàm phi tuyến tính để phá vỡ tuyến tính, tạo phi tuyến. Đó là lý do ta sử dụng hàm activation.

3.4 Cost Function (Loss Function)

Khi ta cho một mẫu X vào để "huấn luyện" mạng neuron, gọi y_x là kết quả dự đoán và $y_x^{'}$ là kết quả thực tế. Khi đó, thông qua Cost Function ta nhận được Loss Value là C_x thể hiện độ chênh lệch giữa $y_x^{'}$ và y_x (Lưu ý rằng $y_x^{'}$ và y_x không phải là một giá trị thực là một vector thuộc R^n trong khi C_x mà là một giá trị thực).
Có nhiều loại Cost Function được sử dụng, một số loại thông dụng là:

- Mean Squared Error (MSE):

$$C = \frac{1}{2n} \sum_x \parallel y_x^{'} - y_x \parallel^2$$

- Cross Entropy (CE):

$$CE = -\frac{1}{n} \sum_x \left[y_x^{'} ln(y_x) + (1 - y_x^{'}) ln(1 - y_x) \right]$$

Cost Function có những tính chất đặc trưng sau:

- Hàm luôn trả về giá trị thực không âm.
- Khi độ lệch giữa y_x' và y_x càng giảm, giá trị của C càng tiến gần 0.

Mean Squared Error thực chất là trung bình cộng của bình phương khoảng cách của các vector y_x và y_x', biểu diễn cho độ lệch giữa kết quả chuẩn và kết quả do ta huấn luyện. Cross-Entropy Loss tuy tương đối phức tạp nhưng ta cũng có thể chứng minh được hàm đó cũng thỏa hai điều kiện trên. Vai trò của hàm Cross-Entropy sẽ được hiểu sâu hơn khi được áp dụng trong Backpropagation mà ta sẽ đề cập.
Cost Function đóng vai trò to lớn trong việc "huấn luyện" mạng neural. Mục đích của việc "huấn luyện" chính là cực tiểu hóa Loss Value C thông qua điều chỉnh giá trị của các weight và bias trong mạng neural. Tùy theo Cost Function mà ta sử dụng, ta có thể biết được C sẽ thay đổi như thế nào sau khi ta điều chỉnh weight và bias. Đó là điểm mấu chốt của thuật toán Gradient Descent.

3.5 Gradient Descent

Gradient Descent là một thuật toán tìm điểm cực trị của hàm khả vi từ R^n đến R. Ý tưởng của thuật toán là đi ngược hướng của đạo hàm thì sẽ tới được điểm cực tiểu như một viên bi trong không gian n chiều "lăn xuống" đáy thung lũng. Trong trường hợp Neural Network, ta cập nhật các biến Weight và Bias sao cho Loss Value giảm đi nhiều nhất. Về cơ bản, ta thực hiện các bước sau:

- Bước 1: Feedforward training data, có được Loss Value C là trung bình cộng của tất cả Loss Value C_x của các mẫu thử.
- Bước 2: Tính $\dfrac{\partial C}{\partial w_{jk}^l}$ và $\dfrac{\partial C}{\partial b_j^l}$
- Bước 3: Cập nhật weights và biases:

$$w_{jk}^{l*} = w_{jk}^l - \eta \cdot \frac{\partial C}{\partial w_{jk}^l}$$

$$b_j^{l*} = b_j^l - \eta \cdot \frac{\partial C}{\partial b_j^l}$$

- Bước 4: Lặp lại từ bước 1 cho đến khi huấn luyện xong.

Kí hiệu η là một số thực dương và được gọi là tốc độ học. η càng nhỏ, neural network học càng chậm do các weight và bias cập nhật chậm. Nhưng khi η quá lớn, Loss Value đáng lẽ phải hội tụ về cực tiểu thì có thể phân kì và máy không thể học được, có thể ví như viên bi "lăn xuống" quá nhanh dẫn tới lăn vượt qua đáy thung lũng. Do đó việc xác định và điều chỉnh giá trị của η cũng quan trọng để mạng lưới có thể học được và học hiệu quả.

3.6 Backpropagation

Trước hết, ta định nghĩa:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

(δ_j^l là error của neural thứ j của layer l)

Ta có bốn công thức cơ bản của phương pháp Backpropagation:

$$\delta^L = \nabla_a C \odot \sigma^{'}(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma^{'}(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{BP3}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{BP4}$$

(L là layer cuối, l là layer bất kì)

Cả 4 đẳng thức trên đều có thể chứng minh dựa trên nguyên tắc mắt xích (Chain Rule).
Ta có thể cập nhật Gradient Descent theo các bước sau đây:

- Dựa vào công thức của Cost Function, ta có thể dễ dàng tính được $\nabla_a C$ là đạo hàm riêng của C đối với activation a qua đẳng thức (BP1), $\sigma^{'}(z) = \sigma(z) \cdot (1 - \sigma(z))$. Từ đó ta tính được error của các neuron lớp cuối cùng.

- Qua đẳng thức (BP2), ta có thể tính được error của mỗi neuron trên một lớp khi đã có error của tất cả các neuron của lớp liền sau. Qua đó ta có thể tính được error của tất cả neuron trong mạng neuron (ta có thể tưởng tượng các giá trị error lan truyền ngược lại từ layer cuối lên qua các weight và đó cũng là nguồn gốc của tên "Backpropagation").
- Cuối cùng, qua đẳng thức (BP3) và (BP4), ta có thể tính được đạo hàm riêng của Loss Value đối với từng weight và bias.

4 Tối ưu hóa (Optimization)

Khi cho máy học, một vấn đề gọi là Overfitting xuất hiện. Overfitting là một hiện tượng hay xảy ra khi máy học quá hiệu quả, dẫn tới "học vẹt", tức là độ chính xác của máy khi đầu vào là Training Set là rất cao, nhưng khi cho Test Data làm đầu vào thì độ chính xác lại thấp. Nguyên nhân dẫn đến sai lệch là do máy học hết tất cả, bao gồm nhưng sai số của data (noise). Mục đích của tối ưu hóa chính là giảm thiểu vấn đề này, ngoài ra cũng để tăng tốc độ và giảm gánh nặng bộ nhớ cho máy. Nhìn chung, ta có hai phương hướng để tối ưu hóa: cải thiện bộ dữ liệu (sinh dữ liệu, tìm thêm dữ liệu,...) và cải tiến thuật toán (L1, L2 Regularization, Dừng sớm, Dropout,...).

4.1 Dừng sớm (Early Stopping)

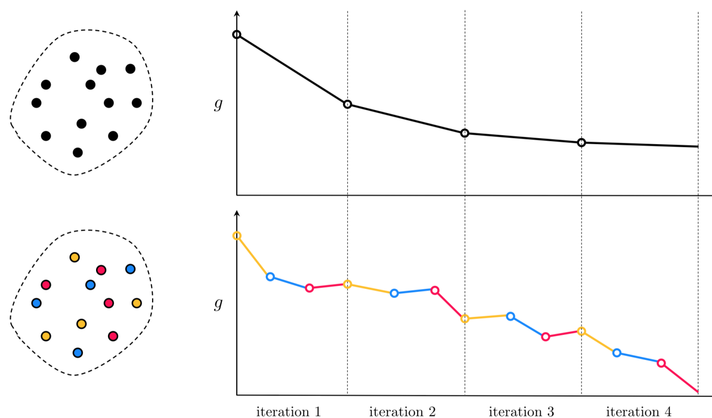
Ở phương pháp này, ta sẽ tách riêng một phần nhỏ từ training data thành một phần gọi là Validation Data. Sau mỗi epoch (một lần train hết toàn bộ Training Data), ta sẽ đo độ chính xác của neural network khi cho vào Validation data. Khi độ chính xác đó dần dần không còn tăng đáng kể nữa thì ta cho dừng quá trình "huấn luyện" để tránh Overfitting.

4.2 Sinh dữ liệu

Trong trường hợp không đủ dữ liệu để "huấn luyện" neural network, ta có thể tạo ra data mới dựa trên các data sẵn có bằng cách biến đổi chúng không đáng kể. Ví dụ như với một tấm ảnh, ta có thể xoay, kéo dãn hay dịch nó một ít. Hơn nữa, những phép biến đổi như thế còn tái hiện lại cho sai số, hay "noise" ngoài thực tiễn.

4.3 Mini-batch Gradient Descent

Điểm khác biệt ở đây là thay vì trong Gradient Descent truyền thống ta truyền toàn bộ training set vào cho máy học trước khi xử lý, ở đây ta sẽ chia training set thành nhiều mẫu nhỏ hơn gọi là mini-batch có số mẫu thử gần giống nhau. Ở trường hợp số lượng mẫu thử mỗi batch là 1 thì ta gọi đây là Stochastic Gradient Descent. Phương pháp này giảm gánh nặng về bộ nhớ vì các mẫu thử đã được chia nhỏ.



4.4 L1, L2 Regularization:

$$C_1 = C_0 + \frac{\lambda}{n}\sum_w |w|$$

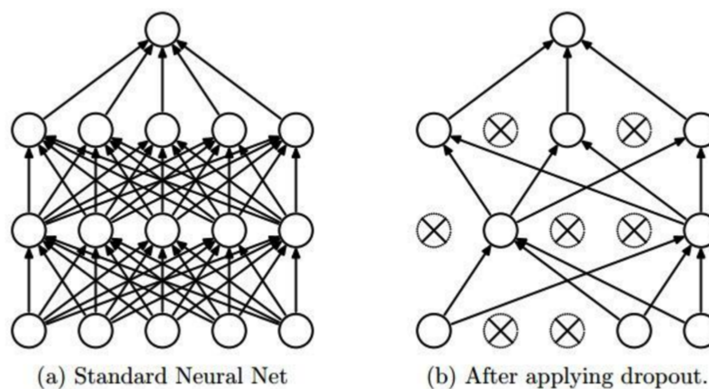
$$C_2 = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

Trong đó C_0 là Loss Function ban đầu như MSE hay Cross Entropy, C_1 và C_2 lần lượt là giá trị của loss function sau khi qua $L1, L2$ Regularization. Đặc điểm của cả hai phương pháp này là chúng đều giảm đi độ tăng của weight trong mỗi lần cập nhật. L1 thì giảm đi một lượng là hằng số dương, trong khi L2 giảm đi một lượng tỉ lệ thuận với độ lớn của weight đang xét. Cả

hai đều có tác dụng làm "tiêu biến" weight, giúp giảm đi tính quan trọng các chi tiết sai số nhỏ.

4.5 Dropout

Trước mỗi lần feedforward, ta vô hiệu hóa một số neuron ngẫu nhiên trong các hidden layers (xem như các weight liên kết tới chúng không tồn tại). Sau khi feedforward, ta backpropagate. Sau đó, tiếp tục thử vô hiệu hóa một tổ hợp neuron ngẫu nhiên khác rồi lại feedforward, backpropagate. Lặp lại quá trình trên nhiều lần. Có thể nói rằng, mỗi lần ta vô hiệu hóa một số neuron bất kì, ta được các mạng neuron khác nhau. Các mạng này có thể gặp tình trạng overfit theo những cách khác nhau nhưng cuối cùng khi hồi phục các neuron, các tình trạng overfit đó có thể trung hòa cho nhau.



4.6 Cross Entropy Loss Function

Ta đã từng đề cập đến hàm này trước đây. Điều khiến cho Loss Function này trở nên thông dụng chính là tính chất đặc biệt của nó: output càng lệch so với nhãn thì Gradient càng lớn, cập nhật càng nhanh, tốc độ hội tụ càng cao. Ta có thể lý giải điều này bằng cách tính error của layer cuối và để ý rằng error này chính bằng độ lệch giữa output và target mà không phụ thuộc vào đạo hàm của hàm Sigmoid. Error này backpropagate cả mạng neuron nên cũng tăng giá trị error tất cả neuron trong cả mạng lưới so với khi sử dụng MSE, từ đó tăng gradient của Cost Function, giúp ta cập nhật weight và bias nhanh hơn.

5 Cách hoạt động của Classification Neural Network:

Về bản chất, quá trình hoạt động của một Classification Neural Network sử dụng thuật toán Backpropagation như sau:

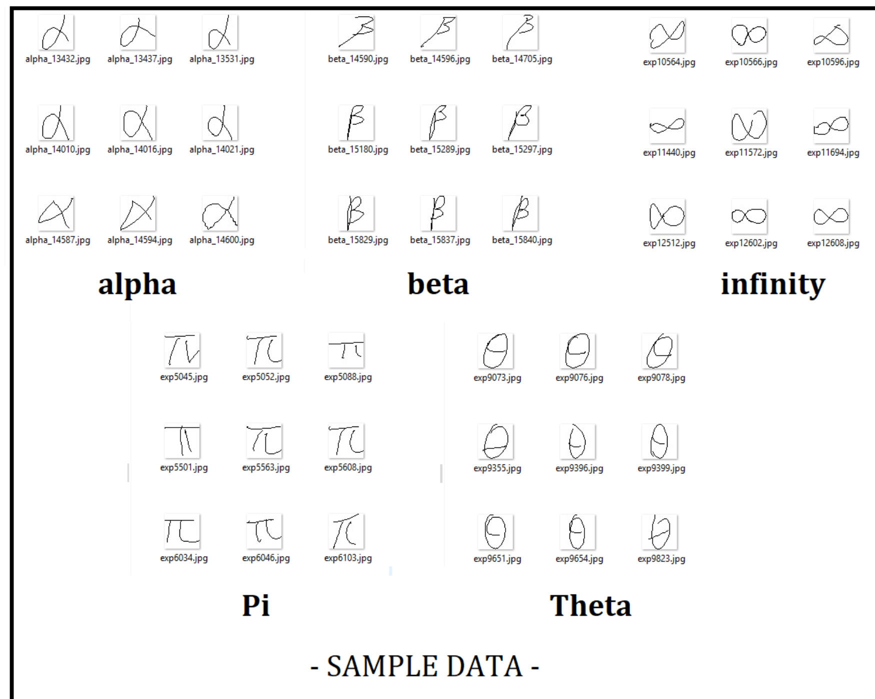
- Bước 1: Chuẩn bị dữ liệu:

  - Ta chia dữ liệu thành 2 phần chính: Dữ liệu huấn luyện (Training Data) và Dữ liệu kiểm tra (Test Data).
  - Mỗi dữ liệu ta phải dán đúng nhãn của nó.
  - Các dạng dữ liệu thường sử dụng: csv, jpg,. . .

- Bước 2: Đọc dữ liệu và đưa nó về dạng vector
- Bước 3: Đi từ layer 1 đến layer cuối (Feedforward):
  - $z^l = w^l a^{l-1} + b^l$
  - $a^l = \sigma(z^l)$
    (z, a, b lần lượt là weighted input, activation, và bias của các neurons tại lớp của nó)
- Bước 4: Backpropagation:
  - Tính error của các neurons layer cuối bằng (BP1)
  - Tính error của các layers trước đó khi biết error của layer liền sau bằng (BP2).
  - Dựa vào các error tính được các đạo hàm riêng của Loss Value với biases và weights theo (BP3) và (BP4).
- Bước 5: Dùng Gradient Descent để cập nhật các weights và biases dựa theo các đạo hàm riêng vừa tính.
  - $w^l_{jk} = w^l_{jk} - \eta \cdot (a^{l-1}_k \cdot \delta^l_j)$
  - $b^l_j = b^l - \eta \cdot \delta^l_j$

6 Áp dụng mô hình

Để minh hoạ cơ chế hoạt động của một Classification Neural Network, trong trường hợp dưới đây, ta sẽ đặt nó vào một mô hình cụ thể: Mô hình phân loại kí tự toán học viết tay (Hand Written Math Symbol Classification).
Để đơn giản hoá vấn đề, mô hình dưới đây sẽ phân loại 5 kí tự thường xuất hiện: $\alpha, \beta, \infty, \theta, \pi$ (trích từ nguồn dữ liệu ở www.kaggle.com).



Quy trình huấn luyện bắt đầu bằng việc đọc dữ liệu. Trong mô hình dưới đây, ta đưa 15000 bức ảnh với kích thước 45 x 45 của 5 kí tự trên về dạng các vector 45 x 45 = 2025 chiều. Để làm được điều này, ta sẽ trích dẫn đường dẫn tới thư mục chứa các bức ảnh.

```
13  directory_list = []
14  foldername="D:/Khoi/[PiMA] Nguyen_Nhat_Minh_Khoi/SymbolRegconitionDataset3"
15  for root, dirs, files in os.walk(foldername, topdown=False):
16      for name in dirs:
17          directory_list.append(os.path.join(root, name))
```

Sau đó, ta bắt đầu đọc dữ liệu và gắn nhãn các dữ liệu đó vào đúng nhãn mác của nó.

```
19  num_symbol = 0
20  # iterate through all folder
21  for directory in directory_list:
22      # create standard output of symbol i
23
24      std_output = np.zeros(5)
25      std_output[num_symbol] = 1
26      num_symbol = num_symbol + 1
27
28      # print(std_output)
29      # iterate through all file in folder
30      for file in os.listdir(directory):
31
32          filename = os.fsdecode(file)
33
34          if filename.endswith(".jpg"):
35
36              # image to vector
37              img = Image.open(os.path.join(directory, filename)).convert('')
38              arr = np.array(img)
39              flat_arr = np.ravel(arr)
40              training_data.append(flat_arr)
41              std_output_data.append(std_output)
```

Sau khi khởi tạo và xử lý để đưa dữ liệu thành các vector, ta sử dụng thư viện Sklearn, tạo ra một mạng neural gồm 2 hidden layer (trong đó mỗi hidden layer gồm 50 neuron) và 1 output layer để huấn luyện.

```
58  #neural network
59
60  clf = MLPClassifier(hidden_layer_sizes=(50,50))
61
62  #train network with 4/5 data
63
64  n=(4*n)/5
65  clf.fit( np.matrix( r_training_data[:int(n)] ) , np.matrix( r_std_output_data[:int(n)]) )
66
67  #score network
68  print( (clf.score( np.matrix(r_training_data[int(n):]), np.matrix(r_std_output_data[int(n):]) )) )
```

Lưu ý ở đây là hàm fit của thư viện Sklearn ở đây sử dụng thuật toán Feed Forward và BackPropagation (có sử dụng hàm activation ReLU và kết hợp các kĩ thuật tối ưu hóa đã trình bày ở trên) để huấn luyện.

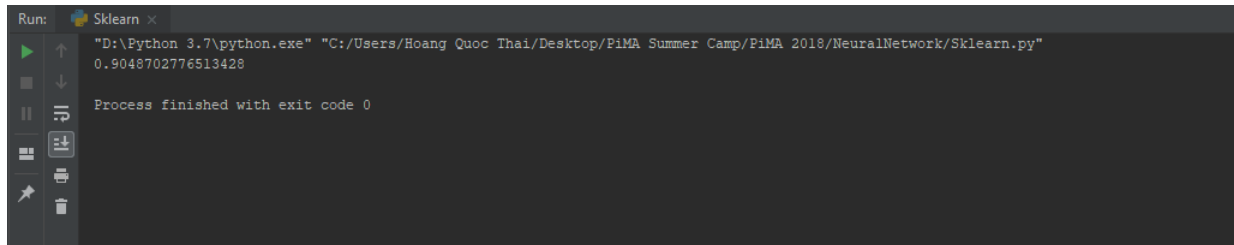
Hàm fit yêu cầu nhập dữ liệu và nhãn của nó. Ở đây ta chia dữ liệu sang $\frac{4}{5}$ dữ liệu cho việc huấn luyện và $\frac{1}{5}$ dữ liệu để kiểm tra quá trình huấn luyện đó.

7 Kết luận đánh giá

Để đánh giá độ hiệu quả của mô hình, ta sử dụng metric. Có nhiều loại metric khác nhau. Ở đây, ta sử dụng metric Accuracy có công thức là:

$$\text{Accuracy} = \frac{\text{Tổng số test đúng}}{\text{Tổng số test thử}}$$

Ở mô hình trên, với training data là 11000 bức ảnh và test data là 1000 bức ảnh, ta có tỉ lệ chính xác: accuracy = 0.89
Hiệu quả huấn luyện với Sklearn:



Bạn có thể xem đoạn code được dùng để thực hiện mô hình tại https://github.com/minhkhoi1026/PiMA20
Hand-Written-Math-Symbol-Classification

Tài liệu

[1] http://kaggle.com.

[2] Michael Nielsen. *Neural Networks and Deep Learning*. http://neuralnetworksanddeeplearning.com, 2017.

[3] Vũ Khắc Tiệp. *Machine Learning cơ bản*. http://machinelearningcoban.com, 2017.