TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

Final Project

PHƯƠNG PHÁP NGHIÊN CỨU KHOA HỌC

Thành viên nhóm:

Huỳnh Công Sinh 1712724
Đào Thanh Thiện 1712782

Mục Lục

1. Tóm Tắt

Hiện nay, ứng dụng của học máy trong cuộc sống thực tiễn ngày càng rộng rãi.,. càng nhiều mô hình học máy khác nhau xuất hiện cho các nhu cầu các nhau. Cụ thể, em nghiên cứu ứng dụng của mô hình mạng neuron cho việc nhận dạng chữ số viết tay. Nhóm em sử dụng Mạng Neural học sâu thay vi các cách khác như Support Vector Machine... để có thể cải thiện hiệu xuất một cách liên tục. Phương pháp nghiên cứu tập trung vào hướng sử dụng mạng neural để nhận diện tăng độ chính xác khi nhận diện. Kết quả cho ra tỉ lệ nhận dạng chính xác vượt trội so với việc nhận dạng bằng mạng neural thông thường và các phương pháp nhận diện truyền thống khác. Mô hình của chúng tôi sẽ cải thiện khả năng học của mạng neural, khi đó có thể áp dụng vào các ứng dụng nhận dạng khác lớn hơn.

2. Giới Thiệu

Những năm gần đây, thế giới chứng kiến được sự phát triển chóng mặt của trí tuệ nhân tạo nói chung và các thuật toán học máy (machine learning) và học sâu (deep learning) nói riêng. Mặc dù rất nhiều ý tưởng cơ bản của deep learning đã có từ những năm 80-90 của thế kỷ trước, tuy nhiên sự bùng nổ của deep learning chỉ bắt đầu trong khoảng gần 10 gần đây.

Dựa trên cơ chế của bộ não con người, các thuật toán học sâu được tạo nên với mong muốn máy tính cũng có khả năng học hỏi và xử lý như con người. Chỉ trong vài năm, deep learning đã thúc đẩy tiến bộ trong đa dạng các lĩnh vực, giải quyết các vấn đề đôi khi vượt qua cả con người. Nổi tiếng nhất có thể kể đến ví dụ về Alpha Go của google đã chiến thắng kỳ thủ số một thế giới ở bộ môn cờ vây Lee Sedol.

Trong bài viết dưới đây, ta sẽ làm rõ về các định nghĩa, cơ chế và các vấn đề liên quan của mạng học sâu. Và việc sữ dụng mạng học sâu trong việc nhận diện chữ số viết tay.

3. Mạng Neural

3.1 Perceptron

Mạng neural đơn giản nhất là perceptron, chỉ bao gồm một neuron duy nhất. Giống với ý tưởng của neuron sinh học khi nhận tính hiệu từ dendrites xữ lý tính hiệu đầu vào và trả tín hiệu đầu ra.

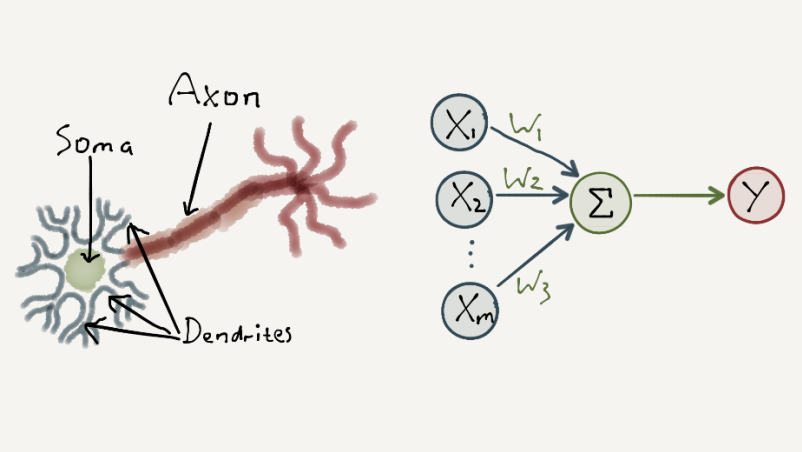


Figure 1: Neural sinh học (trái) và perceptron (phải)

Để diễn tả ý tưởng này dưới dạng thuật toán và các tham số, ta có mô hình neural network bao gồm các đầu vào (input), được nhân với các trọng số (weight) tương ứng và lấy tổng, sau đó đi qua hàm kích hoạt, ta thu được kết quả.

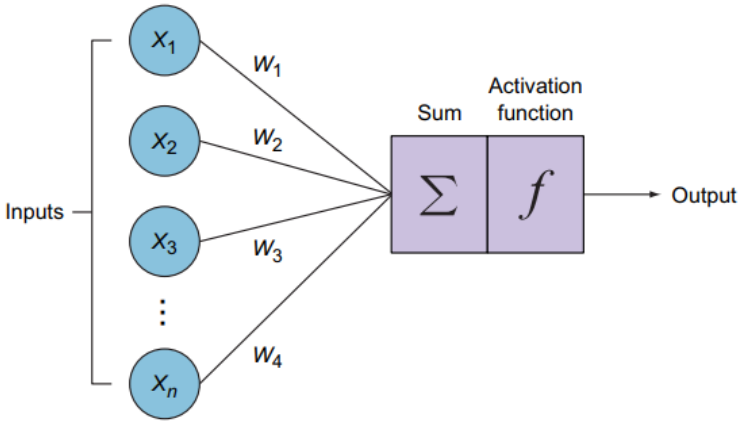


Figure 2: Cấu tạo của một mô hình perceptron

Ta có thể hiểu được quá trình của perceptron với 2 bước chính:

Bước 1: Tính tổng linear (linear combination):

$$z = 1* w_0 + x_1 * w_1 + x_2* w_2 + x_3* w_3 + \ldots.$$

Bước 2: Áp dụng hàm kích hoạt:

$\hat{y} = g(z)$, với g là một hàm kích hoạt.

Với hệ số w_0 được gọi là bias. Với công thức trên, các kết quả $w_0 + x_1$ ta luôn thu được các đường thẳng. tham số Bias được thêm vào này giúp ta di chuyển lên xuống trên trục y để dự đoán kết quả tốt hơn. Nếu không có tham số bias này, đường thẳng tìm ra luôn đi qua điểm gốc O(0, 0) và chắc chắn là kết quả rất tồi.

Quá trình học của perceptron:

Mô hình perceptron sữ dụng phương pháp thử và sai để học. Các trọng số (weight) thay đổi tăng lên và giảm xuống cho đến khi mạng huấn luyện xong.

Bước 1: Dự đoán $\hat{y}$ = activation($\sum x_i * w_i + w_0$).

Bước 2: Tính độ lỗi error = y - $\hat{y}$, nếu error rất nhỏ (gần bằng 0) thì dừng lại.

Bước 3: Cập nhật trọng số và quay lại bước 1.

Mặc dù vậy, ta có thể dễ dàng nhận ra việc mô hình perceptron không có khả năng giải quyết các vấn đề phức tạp. Vì khả năng phân biệt của perceptron chỉ dừng lại ở mức tuyến tính, mà các bài toán cần giải quyết không bao giờ dừng ở mức độ tuyến tính.

3.2 Mạng Neural

Mạng neural là sự kết hợp của nhiều perceptron lại với nhau (nên còn được gọi là Multi-layer Perceptron). Như đã đề cập ở trên, việc xấp xỉ kết quả nếu chỉ dùng một perceptron duy nhất là rất khó khăn và kết quả thường rất tệ. Do đó, việc kết hợp các

perceptron lại với nhau nhằm tăng độ phức tạp tính toán nhằm tìm ra tìm ra cách giải quyết bài toán hiệu quả hơn.

Một mạng neural bao gồm lớp đầu vào (input layer), các lớp ẩn (hidden layer) và lớp đầu ra (output layer). Một mạng có thể có một hay nhiều lớp ẩn, mỗi lớp có thể có một hay nhiều node (tùy thuộc vào người thiết kế). Thông thường, ta có hơn lớp ẩn nên nó được gọi là mạng học sâu.

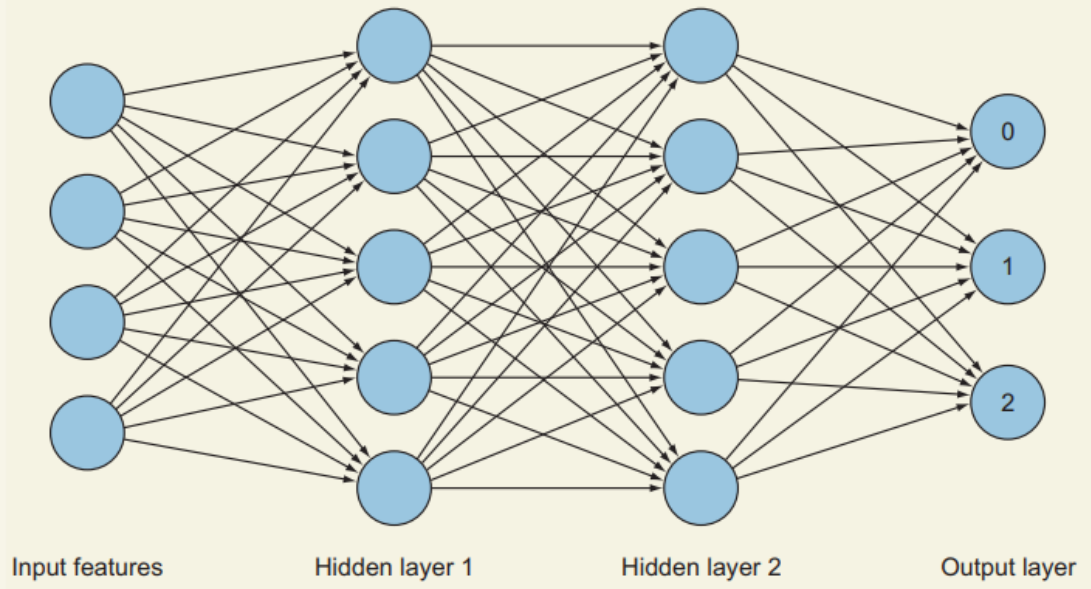


Figure 3: Multi-layer Perceptron

Đơn vị của mỗi

Với mỗi node trong lớp ẩn và lớp đầu ra:

+ liên kết với tất cả các node ở layer trước đó và có trọng số (weight) riêng.

+ mỗi node có 1 hệ số bias riêng.

+ diễn ra 2 bước: tính tổng linear và áp dụng hàm kích hoạt.

$$z_j^{(l)} = \boldsymbol{w}_j^{(l)T} \boldsymbol{a}^{(l-1)} + b_j^{(l)}$$

4. Các hàm kích hoạt (activation function)

4.1 Hàm kích hoạt phi tuyến tính

Hàm kích hoạt chuyển đổi các tổng linear (linear combination) của tổng các trọng số (weight) thành một mô hình phi tuyến tính. Một hàm kích hoạt được đặt ở cuối mỗi perceptron để quyết định có kích hoạt neuron đó hay không. Nếu không có các hàm kích hoạt này thì multi-layer perceptron sẽ thể hiện không khác gì một perceptron thông thường mặc cho ta có thêm bao nhiêu lớp đi chăng nữa.

Tại đây các hàm kích hoạt phải là phi tuyến tính. Kết hợp hai hàm tuyến tính thì ta vẫn thu được một hàm tuyến tính đây không giúp mô hình học thêm được gì cả. Kết quả đạo hàm của một hàm tuyến thì luôn không đổi vì nó không phụ thuộc vào tham số đầu vào. Việc này làm cho gradient không đổi nên độ lỗi mô hình cũng không đổi, chứng tỏ mô hình không được học và cải thiện.

4.2 Các hàm kích hoạt thông dụng

4.2.1 Sigmoid function (Logistic function)

Đây làm hàm kích hoạt thông dụng nhất. Với đầu vào là một con số trong khoảng (-∞, +∞) và trả về một kết quả xác xuất trong khoảng (0, 1).

Công thức hàm sigmoid:

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

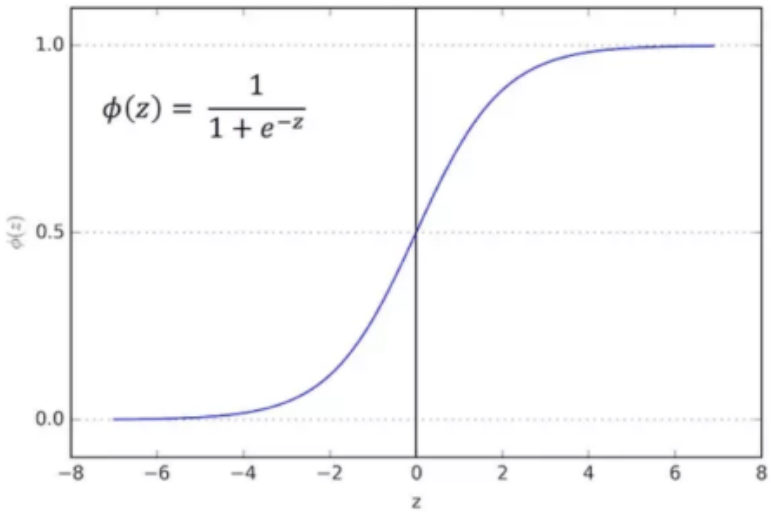


Figure 4: Đồ thị hàm Sigmoid

Hàm sigmoid thường được dùng trong các mô hình phân biệt giữa hai đối tượng. Trong mạng neural, hàm sigmoid thường chỉ được dùng lớp đầu ra nếu yêu cầu của lớp đầu ra là các giá trị nhị phân.

Đạo hàm hàm sigmoid ta thu được:

$$\sigma'^{(x)} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\frac{e^{-x}}{1+e^{-x}} = \sigma^{(x)}\left(1-\sigma^{(x)}\right) \leq \frac{1}{4}$$

Nếu tính gradient sẽ xãy ra hiện tượng vanishing gradient. (sẽ nói rõ hơn ở phần 7)

4.2.2 Softmax function

Đây là hàm tổng quát hơn của hàm sigmoid. Nó được dùng để tính xác xuất phân loại khi có hơn hai lớp cần phân biệt. Được dùng nhiều trong deep learning để dự đoán kết quả phân biệt giữa các lớp khác nhau. x_j

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

4.2.3 Hyperbolic tangent function (tanh)

Đây là phiên bản đã được chỉnh sửa của hàm sigmoid. Với đầu vào vẫn là một con số trong khoảng (-∞, +∞) và trả về một kết quả xác xuất trong khoảng (-1, 1).

$$tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

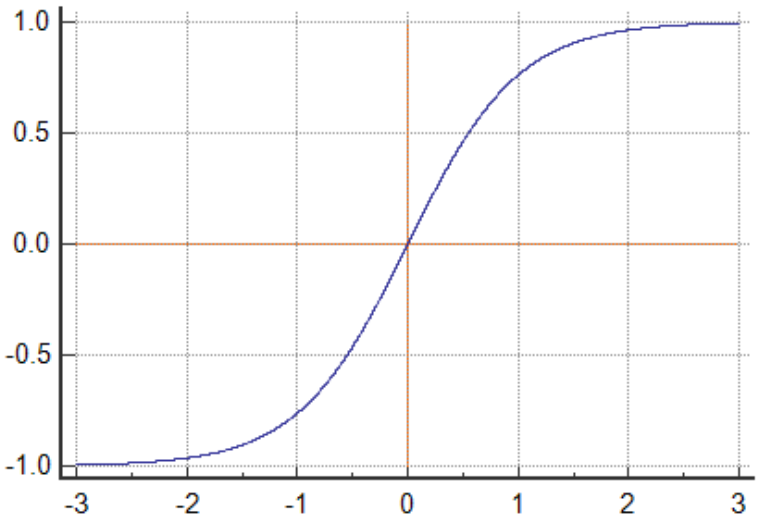


Figure 5: Đồ thị Tanh function

Ta tính đạo hàm hàm softmax:

$$tanh'^{(x)} = \frac{\sinh'(x)}{\cosh'(x)} = \frac{(\sinh(x))'\cosh(x) - \sinh(x)(\cosh(x))'}{cosh^2(x)}$$

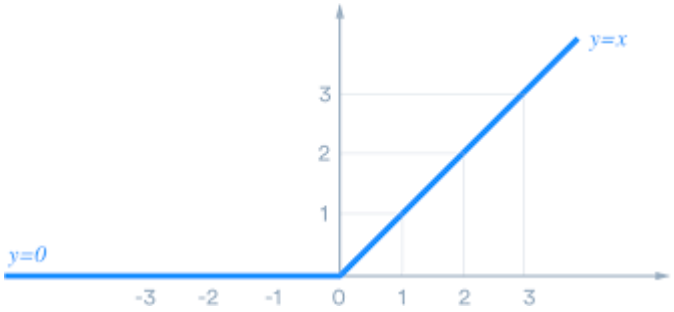
$$= \frac{cosh^2(x) - sinh^2(x)}{cosh^2(x)} = \frac{1}{cosh^2(x)} \leq 1$$

Nếu tính gradient, sẽ xảy ra hiện tượng vanishing gradient. (sẽ nói rõ hơn ở phần 7)

4.2.4 Rectified linear unit (ReLU)

Hàm kích hoạt ReLU chỉ kích hoạt node nếu giá trị đầu vào lớn hơn 0. Nếu giá trị đầu vào nhỏ hơn 0, kết quả đầu ra luôn là 0. Nhưng nếu đầu vào là lớn hơn 0, các giá trị đầu vào và đầu ra có mối quan hệ tuyến tính với nhau.

$$f(\mathrm{x}) = \max(0, x)$$

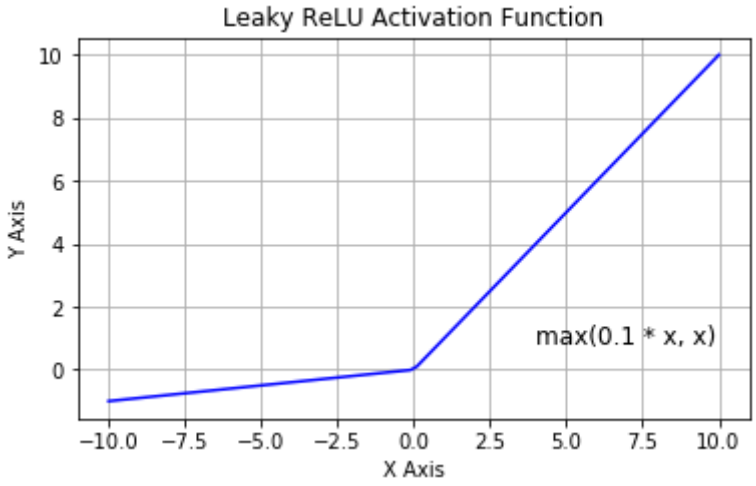


Một nhược điểm của ReLU là với các node có giá trị nhỏ hơn 0, đi qua hàm ReLU activation sẽ trở thành 0, hiện tượng đấy gọi là "Dying ReLU". Nếu các node chuyển thành 0 thì sẽ không còn ý nghĩa với bước tiếp theo cũng như các hệ số tương ứng cũng không được cập nhật với gradient descent.

4.2.5 Leaky ReLU

Để khắc phục hiện tượng Dying ReLU, người ta tạo ra thêm hàm Leaky ReLU.

$$f(x) = \max(0.01x, x)$$



Thay vì triệt tiêu ảnh hưởng của các giá trị nhỏ hơn 0 Hàm Leaky ReLU cho phép nó giữ lại một ít ảnh hưởng . Từ đó nó giữ được các điểm tốt của hàm ReLU ngoài ra còn giải quyết được vấn đề Dying ReLU.

Tại sao là tham số 0.01, tham số này có thể được thay đổi và không ảnh hưởng quá nhiều đến khả năng học của mô hình. Hãy thoải mái sữ dụng với các tham số khác như (0.1, 0.01, 0.02).

5. Quá trình lan truyền tiến (Feedforward process)

Quá trình lan truyền xuôi là quá trình tính tổng linear (linear combination) và áp dụng hàm kích hoạt (activation function) theo chiều tiến, tức là từ lớp đầu vào đến các lớp ẩn cuối cùng truyền đến lớp đầu ra.

Để tổng quát hơn với cả mạng neuron, ta gọi X = [x_1, x_2,….x_n] với các nhãn Y = [y_1, y_2,….y_n], Số layer của mạng là L lớp. Ta cần tìm $\hat{y}$ với input là x.

$$\boldsymbol{a}^{(0)} = x$$

$$z_i^{(l)} = \boldsymbol{w}_i^{(l)T}\boldsymbol{a}^{(l-1)} + b_i^{(l)}$$

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)T}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, l = 1,2,\dots,L$$

$$\boldsymbol{a}^{(l)} = \sigma\left(\boldsymbol{z}^{(l)}\right) = \sigma(\boldsymbol{W}^{(l)T}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)})$$

$$\hat{y} = \boldsymbol{a}^{(L)}$$

Tại đây ta có thể dự đoán được $\hat{y}$ từ input x, nhưng giá trị $\hat{y}$ có thể khác so với nhãn y. Sự sai khác này gọi là độ lỗi. Khi huấn luyện mạng, ta lặp lại quá trình này cho đến khi tìm được tham số độ lỗi là nhỏ nhất, tương đương việc các tham số trong mạng là tối ưu.

6. Hàm mất mát (Loss function)

6.1 Định nghĩa của hàm lỗi

Hàm lỗi (error function) / hàm mất mát (loss function) / hàm chi phí (cost function) là tên chung để chỉ về một độ đo độ sai lệch giữa giá trị dư đoán $\hat{y}$ và nhãn y. Hay nói cách khác hàm lỗi là phương pháp được dùng để đánh giá thuật toán đang được sữ dụng sẽ mô hình hóa dữ liệu tốt đến mức nào. Với tham số độ lỗi lớn, dự đoán của mô hình là không tốt. Ngược lại với độ lỗi càng nhỏ, thì chứng tỏ dự đoán của mô hình càng chuẩn xác.

Độ lỗi phải luôn là số dương, vì nếu tồn tại độ lỗi âm thì có thể dẫn đến tổng độ lỗi là 0 dù cho các điểm lỗi vẫn tồn tại.

Ngoài ra, với mỗi mỗi hàm lỗi sẽ cho một độ lỗi khác nhau trên cùng một dự đoán, vì vậy việc chọn hàm lỗi cần phù hợp với mô hình hoặc được thiết kế chuyên biệt cho một bài toán cụ thể. Mặc dù vẫn có một số hàm lỗi có thể được dùng chung cho nhiều bài toán khác nhau.

6.2 Các hàm lỗi thông dụng

Với y = (y_1, y_2,…, y_n) là các nhãn hay output thực tế, $\hat{y}$ = ($\hat{y}_1$, $\hat{y}_2$,…, $\hat{y}_n$), W và b là các trọng số và bias của mạng

6.2.1 Mean squared error function (MSE)

$$E(W,b) = \frac{1}{\mathrm{N}}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

Hàm MSE tính giá trị trung bình của tổng bình phương phương sai giữa giá trị dự đoán và giá trị thực. Với cách tính này giá trị hàm lỗi thu được luôn dương. Hàm MSE tập

trung trừng phạt những sai lầm lớn của mô hình. Hàm MSE khá nhạy cảm với dữ liệu outliner.

6.2.2 Cross-entropy

$$E(W,b) = -\sum_{k=1}^{K} \mathrm{p}(\hat{y} = \mathrm{k}) \log(\mathrm{q}(y = k))$$

Với: k là số lớp.

p(t) là xác xuất mục tiêu.

q(x) là xác xuất dự đoán.

Hàm lỗi này dùng trong các mô hình phân loại. Cross-entropy có output là một giá trị xác xuất từ 0 đến 1. Giá trị hàm lỗi tăng lên khi xác xuất dự đoán khác với nhãn thực tế.

7. Backpropagation

Backpropagation là phương pháp chủ chốt trong việc học của mạng neuron. Từ lớp đầu ra (output layer) đến lớp đầu vào (input layer) để điều chỉnh trọng số. Các giá trị trọng số được cập nhật theo công thức $W_n = W_{old} - \alpha(\frac{\partial E}{\partial W_x})$.

Ta có:

$$\boldsymbol{z}^{(L)} = \boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)} + \boldsymbol{b}^{(L)}$$
$$\hat{y} = \boldsymbol{a}^{(L)} = \sigma(\boldsymbol{z}^{(L)})$$

$$E(W,b) = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$$\frac{\partial E}{\partial \boldsymbol{W}^{(L)}} = \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{W}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} \quad [7]$$

Với:

$$\frac{\partial E}{\partial \boldsymbol{W}^{(L)}} = 2(\hat{y} - y) = 2(\boldsymbol{a}^{(L)} - y)$$

$$\frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} = \sigma'(\boldsymbol{z}^{(L)})$$

$$\frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{W}^{(L)}} = \boldsymbol{a}^{(L-1)}$$

➔ $\frac{\partial E}{\partial \boldsymbol{W}^{(L)}} = \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{W}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} = \boldsymbol{a}^{(L-1)}\sigma'^{(\boldsymbol{z}^{(L)})}2(\boldsymbol{a}^{(L)} - y)$

Tương tự ta có:

$$\frac{\partial E}{\partial \boldsymbol{b}^{(L)}} = \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{b}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \frac{\partial E}{\partial \boldsymbol{a}^{(L)}} = 1.\sigma'^{(\boldsymbol{z}^{(L)})} 2(\boldsymbol{a}^{(L)} - y)$$

Đây là công thức thể hiện mức độ thay đổi của hàm lỗi khi ta thay đổi trọng số. nhờ cách này ta có thể tìm cách giảm giá trị của hàm lỗi bằng cách thay đổi trọng số W và bias. Từ đó tìm được bộ trọng số W và bias b tối ưu nhất

Ta có toàn bộ quy trình học như sau:

1. Feedforward: tính tổng linear (linear combination) và áp dụng hàm kích hoạt (activation function) để dự đoán $\hat{y}$.

$$\hat{y} = \boldsymbol{a}^{(L)} = \sigma(\boldsymbol{W}^{(L)T}\boldsymbol{a}^{(L-1)} + \boldsymbol{b}^{(L)})$$

2. So sánh dự đoán $\hat{y}$ và nhãn y để tính hàm lỗi (loss function).

$$E(W, b) = \frac{1}{\mathrm{N}} \sum_{i=1}^{N} |\hat{y}_i - y_i|$$

3. Sữ dụng gradient descent để tính đạo hàm hàm lỗi và giảm tối đa giá trị hàm lỗi.

$$W_{new} = W_{old} - \alpha(\frac{\partial E}{\partial W_x})$$

Các bước này lặp lại đến khi ta thu được giá trị hàm lỗi nhỏ nhất

8. Một số cải tiến

8.1 Vanishing gradient và Exploding gradient

Ta biết quá trình học của mô hình là việc tính gradient descent và cập nhật các tham số trong mạng. Trong quá trình backpropagation, gradient được tính từ lớp cuối tính ngược lên lớp đầu, gradient được nhân với ma trận trọng số tại mỗi lớp.

Nếu gradient được tính là quá nhỏ, lúc này khi tính gradient ở các lớp đầu ta sẽ phải nhân tích của rất nhiều số rất nhỏ nên giá trị sẽ tiến dần về giá trị 0, khiến cho bước cập nhật hệ số của gradient descent sẽ trở nên vô nghĩa. Kéo theo là các giá trị gradient không thể hội tụ => **Đây gọi là hiện tượng Vanishing gradient**, vì nó ngăn cản mạng cập nhật hệ số đồng nghĩa ngăn cản khả năng học của mạng. Mạng sẽ cho kết quả không mong muốn.

Ngược lại, nếu gradient được tính là quá lớn, khi tính gradient ở các lớp đầu ta sẽ phải nhân tích của rất nhiều số rất lớn nên giá trị sẽ tiến dần đến vô cực, khiến cho các bước cập nhật hệ số của gradient descent trở nên không chính xác. Kéo theo là giá trị

gradient trở nên phân kỳ => **Đây gọi là hiện tượng Exploring gradient**, vì nó khiến cho mạng cập nhật các hệ số không đúng đồng nghĩa ngăn cản khả năng học của mạng. Mạng sẽ cho kết quả không mong muốn.

Cách giải quyết vaninshing/exproding gradient là lựa chọn các giá trị khởi tạo cho hệ số phù hợp và chọn activation function phù hợp.

8.2 Overfitting và Underfitting

Việc huấn luyện mô hình nhằm giải quyết vấn đề với mọi dữ liệu được đưa vào, hay nói cách khác là mang tính tổng quát. Hiện tượng overfitting và underfitting là các biểu hiện của một mô hình không có tính tổng quát.

Với overfitting là việc mô hình quá phức tạp khi biểu hiện tốt trên tập dữ liệu huấn luyện nhưng lại biểu hiện rất tồi trên tập dữ liệu mà mô hình chưa thấy qua. Thay vì học các đặc trưng dữ liệu, mô hình cố ghi nhớ các dữ liệu huấn luyện.

Underfitting là việc mô hình quá đơn giản và huấn luyện chưa tốt khi thể hiện tồi trên cả tập dữ liệu huấn luyện và tập dữ liệu chưa thấy qua. Có thể giải quyết vấn đề này bằng cách tăng độ phức tạp của mô hình học bằng cách tăng số lượng lớp ẩn và số node trong mỗi lớp ẩn.

Việc của chúng ta là xây dựng một mô hình không quá phức tạp cũng không quá đơn giản mà là "khớp" (fit) với tập dữ liệu. Nhưng trước hết ta cần một độ đo để đánh giá được chất lượng của mô hình. Đó là hàm lỗi, với y là tập nhãn và $\hat{y}$ là tập dữ liệu dự đoán.

Với các mô tả ở trên, ta có một mô hình tốt là hàm thể hiện tốt trên cả dữ liệu huấn luyện và cả tập dữ liệu chưa thấy. ta có:

$$train\ error = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} |\hat{y}_i - y_i|$$

$$test\ error = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} |\hat{y}_i - y_i|$$

Các đánh giá:

+ Nếu train error lớn và test errror lớn: mô hình bị underfitting

+ Nếu train error nhỏ và test errror lớn: mô hình bị overfitting

+ Nếu train error nhỏ và test errror nhỏ: mô hình được xem là fit

Các phương pháp sau sẽ tập trung vào việc giải quyết vấn đề overfitting mà không đề cập đến underfitting.

8.2.1 Tăng độ đa dạng của dữ liệu

Việc mô hình học bị overfitting có khả năng do tập dữ liệu học là không đủ đa dạng, quá ít dẫn đến việc mạng học đi học lại mà không có khả năng tổng quát cao. Để giải quyết vấn đề này ta có thể thu thập thêm dữ liêu. Phương pháp này có thể không phù hợp với một số trường hợp. Ta có thể dùng phương pháp Data augmentation để tạo ra thêm dữ liệu từ tập dữ liệu ta đã có. Đây là phương pháp tiết kiệm hơn để có thêm dữ liệu huấn luyện.

Các phương pháp image-augmentation có thể kể đến như lật (flip), xoay (rotate), mở rộng (scale), phóng to (zoom), tăng giảm sáng (light condition) và nhiều phương pháp khác.

8.2.2 Validation

Trong thực tế, ta không thể tiếp cận được với dữ liệu test, ít nhất là trong quá trình xây dựng mô hình nhằm tránh việc học trên dữ liệu test khiến cho mô hình không được tổng quát hóa tốt. Việc đánh giá trên tập dữ liệu test chỉ được thực hiện 1 lần ở cuối quá trình xây dựng mạng. Tuy nhiên như đã nói ở trên, ta cần phải đánh giá các tham số dữ liệu của mạng trong quá trình xây dựng mô hình nhằm tránh hiện tượng overfitting và underfitting. Vậy nên ta chỉ nhỏ tập dữ liệu huấn luyện làm tập dữ liệu test cho quá trình huấn luyện gọi là tập validation. Kì vọng việc mô hình hoạt động tốt trên tập validation thì cũng hoạt động tốt trên tập test.

8.2.3 Regularization

8.2.3.1 L2 regularization

Ý tưởng cơ bản của L2 regularization là tăng hàm lỗi bằng cách cộng thêm regularization term. Bằng cách này giúp giảm giá trị trọng số (weight value) của các node trong lớp ẩn, biến chúng trở nên cực nhỏ, gần với giá trị 0, việc này giúp làm đơn giản hóa mô hình học.

$$E_{new}(w,b) = E_{old}(w,b) + \frac{\lambda}{2m}\sum(w)^2$$

$$W_{new} = W_{old} - \alpha(\frac{\partial E}{\partial W_x})$$

Vậy nên với với hàm lỗi tăng thì đạo hàm hàm lỗi cũng tăng, kéo theo trọng số W được cập nhật sẽ nhỏ. Từ đó làm đơn giản hóa mô hình học

8.2.3.2 Dropout layer

Thuật toán dropout vô cùng đơn giản: với mỗi vòng lặp, mỗi neuron có xác xuất þ bị tạm thời bỏ qua (dropout) trong suốt vòng huấn luyện. Tham số þ được gọi là tỉ lệ lược bỏ (drop rate). Thường được dùng trong khoảng [0.3, 0.5]. Bắt đầu với 0.3 khi mô hình học có dấu hiệu overfitting và tăng dần theo quá trình học. Nếu sữ þ quá nhỏ thì nó không có tác dụng chống overfitting, còn nếu þ quá có thể dẫn đến underfitting. Về kết quả, nó khá giống với L2 khi bỏ qua sự ảnh hưởng của một số node trong mạng.

8.2.3.3 Batch Normalization

Batch normalization thực hiện chuẩn hóa (normalization) các extracted features trong các node ở các lớp ẩn trước khi đưa qua hàm kích hoạt ở mỗi lớp.

Các bước thực hiện như sau:

Với input lần lược là các mini-batch B, có m phần tử

Bước 1: tính zero-center của input bằng cách tính trung bình và độ lệch chuẩn của tập input

$$\mu_B = \frac{1}{\mathrm{m}}\sum_{i=1}^{m} x_i \qquad \sigma_B = \frac{1}{\mathrm{m}}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

Bước 2: Chuẩn hóa dữ liệu:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{{\sigma_B}^2 + \epsilon}}$$

Với ϵ là con số cực nhỏ nhằm tránh trường hợp chia cho số 0.

Bước 3: Tinh chỉnh đầu ra:

$$z_i = \gamma\hat{x} + \beta$$

Với γ và β là các tham số được học trong quá trình trainning

Batch normalization chuẩn hóa dữ liệu đầu ra của các layer giúp model trong quá trình huấn luyện không bị phụ thuộc và một thành phần trọng số nhất định.

9. Cài đặt trên python và đánh giá kết quả cài đặt

9.1 Cài đặt

Sử dụng thư viện keras để tạo mô hình Neural network**.** Ở đây, mô hình được sử dụng mô hình mạng neuron 3 lớp (không tính lớp input). Bao gồm một lớp input (lớp thứ 0), hai lớp ẩn và một lớp output

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
```

- Lớp đầu tiên là lớp input. Lớp này có nhiệm vụ nhận dữ liệu có 784 cột tương ứng với 784 điểm ảnh của ảnh đầu vào.

- Lớp thứ hai và thứ ba gồm có 256 perceptron để nhận tín hiệu đầu ra từ lớp trước và dùng hàm kích hoạt ReLU để tạo tín hiệu đầu ra.

- Việc sử dụng 256 perceptron là để tối ưu hiệu năng của mô hình. Nếu ta sử dụng 128 perceptron thì tốc độ được cải thiện, tuy nhiên độ chính xác bị giảm đi đáng kể. Còn khi sử dụng 512 hay 1024 perceptron thì độ chính xác không tăng được nhiều mà tốc độ giảm đi đáng kể.

- Hai lớp này sử dụng thêm một tham số nữa là giá trị dropout. Giá trị này được thêm vào để tránh hiện tượng overfitting. Tham số này có thể hiểu như sau: Với mỗi lớp, một phần của các perceptron bị bỏ qua trong việc tham gia vào lớptiếp theo.

- Trong lớp thứ tư, mô hình nhận 256 tín hiệu đầu vào tương ứng với 256 perceptron ở tầng trước, và cho ra 10 tín hiệu đầu ra tương ứng với 10 giá trị số.

- Mô hình sử dụng một vài tham số khác như:

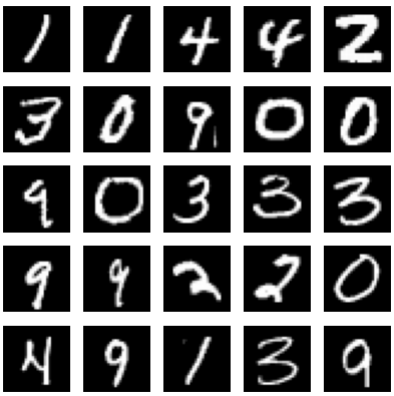
+ Loss function: mô hình sử dụng categorical_crossentropy làm hàm mất mát

+ Optimization: mô hình sử dụng Adaptive Moments để cực tiểu hóa hàm mất mát.

- Trong khi huấn luyện mô hình, tập dữ liệu huấn luyện được chia nhỏ thành từng phần ngẫu nhiên và điều chỉnh thông số theo các phần đó. Lặp lại việc huấn luyện từng phần nhỏ này nhiều lần để cải thiện chất lượng của mô hình. Việc không sử dụng toàn bộ tập dữ liệu cùng lúc để tránh hiện tượng overfitting.

9.2 Đánh giá kết quả

- Dữ liệu được sử dụng cho mô hình này là tập chữ số viết tay MNIST. Tập này gồm có 60000 ảnh, mỗi ảnh gồm có 28 * 28 điểm ảnh, mỗi điểm ảnh mang giá trị từ 0 đến 255. Kèm theo mỗi ảnh là một giá trị từ 0 đến 9, tương ứng với label của ảnh đó.



- Trước khi huấn luyện mô hình, chia tập huấn luyện và tập validation từ bộ dữ liệu trên để có thể đánh giá mô hình sau khi huấn luyện. Sau khi chia, tập huấn luyện sẽ có 54000 ảnh, và tập validation sẽ có 6000.

```
- loss: 0.5936 - accuracy: 0.8115

loss: 0.1988 - accuracy: 0.9405

loss: 0.1564 - accuracy: 0.9538

loss: 0.1339 - accuracy: 0.9605

loss: 0.1192 - accuracy: 0.9643

loss: 0.1075 - accuracy: 0.9669

loss: 0.1068 - accuracy: 0.9677

loss: 0.0983 - accuracy: 0.9698

loss: 0.0936 - accuracy: 0.9712

loss: 0.0909 - accuracy: 0.9715

loss: 0.0906 - accuracy: 0.9716

loss: 0.0834 - accuracy: 0.9755

loss: 0.0767 - accuracy: 0.9770

loss: 0.0799 - accuracy: 0.9748

loss: 0.0719 - accuracy: 0.9783
```

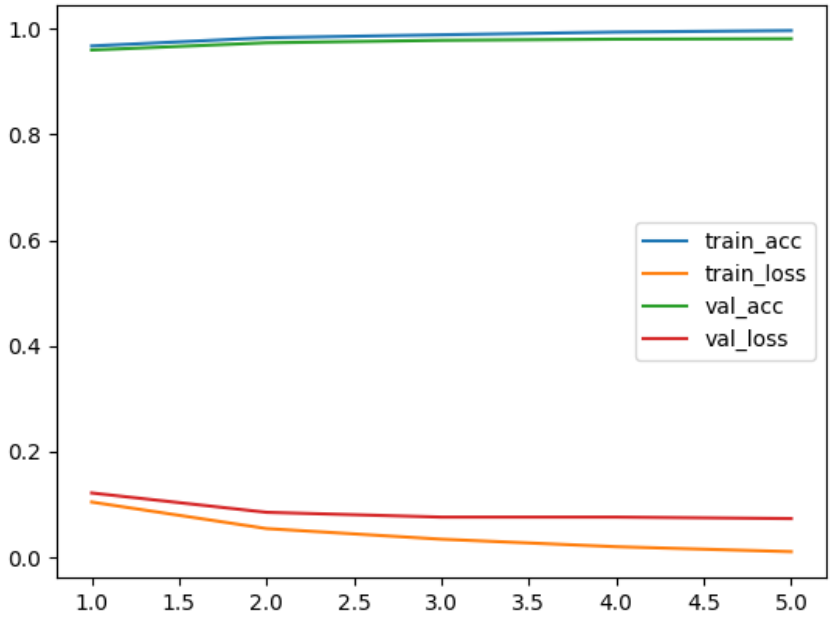
- Trong khi huấn luyện mô hình, mỗi lần ta điều chỉnh mô hình, giá trị loss của mô hình được giảm xuống, đồng thời độ chính xác của mô hình cũng được tăng lên.

- Sau khi kết thúc việc huấn luyện mô hình, giá trị loss là 0.0751 và độ chính xác là 97.73%

- Áp dụng mô hình cho tập validation, có được giá trị loss là 0.0803 và độ chính xác là 97.97%

- Sau khi huấn luyện mô hình, áp dụng vào tập dữ liệu test, mô hình cho ra độ chính xác 98.26% và với 10000 ảnh trong tập này.

- Điều chỉnh giá trị epoch để đánh giá ảnh hưởng của tham số này đối với mô hình



Link github: https://github.com/grant-ward-101/dnn_mnist

10. Tài liệu tham khảo

[1] Blog machinelearningcoban.com

[2] Blog phamdinhkhanh.github.io

[3] Vũ Hữu Tiệp, Multi-layer Perceptron và Backpropagation, Machine Learning cơ bản

[4] Deep Learning for Vision Systems, Mohamed Elgendy

[5] Deep Learning cơ bản, Nguyễn Thanh Tuấn

[6] Vanishing & Exploding Gradients Problems in Deep Neural Networks, Viblo.asia site

[7] Backpropagation calculus, 3Blue1Brown, youtube.com site