# Báo cáo tháng 8

[Báo cáo tháng 8 1](#_Toc523266626)

[1. Tài liệu tham khảo 2](#_Toc523266627)

[2. Chuẩn hóa dữ liệu 2](#_Toc523266628)

[2.1. Rescaling 2](#_Toc523266629)

[2.2. Standardization 2](#_Toc523266630)

[2.3. Scaling to unit length 3](#_Toc523266631)

[3. Gradient Descent và một số biến thể 3](#_Toc523266632)

[3.1. Gradient Descent 3](#_Toc523266633)

[3.2. Một số biến thể của Gradient Descent 4](#_Toc523266634)

[3.3. Các thuật toán tối ưu Gradient Descent 5](#_Toc523266635)

[4. Cross Entropy cho bài toán Softmax Regression 7](#_Toc523266636)

[5. Multi-layer Perceptron và Backpropagation 8](#_Toc523266637)

[5.1. Cấu trúc và hoạt động của một nơ-ron 8](#_Toc523266638)

[5.2. Kiến trúc mạng nơ-ron 11](#_Toc523266639)

[5.3. Backpropagation 11](#_Toc523266640)

[5.4. Cài đặt 14](#_Toc523266641)

## Tài liệu tham khảo

1. <https://machinelearningcoban.com/general/2017/02/06/featureengineering/>
2. <https://machinelearningcoban.com/2017/01/16/gradientdescent2/>
3. <https://machinelearningcoban.com/2017/02/24/mlp/>
4. [https://machinelearningcoban.com/2017/02/17/softmax/#-cross-entropy](https://machinelearningcoban.com/2017/02/17/softmax/%23-cross-entropy)
5. Thân Quang Khoát. Slide bài giảng học phần “Học máy”, Lecture 7 – Neural networks.

## Chuẩn hóa dữ liệu

Các điểm dữ liệu đôi khi được đo đạc với những đơn vị khác nhau, m và feet chẳng hạn. Hoặc có hai thành phần (của vector dữ liệu) chênh lệch nhau quá lớn, một thành phần có khoảng giá trị từ 0 đến 1000, thành phần kia chỉ có khoảng giá trị từ 0 đến 1 chẳng hạn. Lúc này, chúng ta cần chuẩn hóa dữ liệu trước khi thực hiện các bước tiếp theo.

Một vài phương pháp chuẩn hóa thường dùng:

### Rescaling

Phương pháp đơn giản nhất là đưa tất cả các thành phần về cùng một khoảng: [0,1] hoặc [-1,1]…

Trong đó x là giá trị ban đầu, x’ là giá trị sau khi chuẩn hóa; min(x), max(x) được tính trên toàn bộ dữ liệu training ở từng thành phần của vector dữ liệu x.

### Standardization

Một phương pháp nữa cũng hay được sử dụng là giả sử mỗi thành phần đều có phân phối chuẩn với kỳ vọng là 0 và phương sai là 1. Khi đó, công thức chuẩn hóa sẽ là:

Với lần lượt là kỳ vọng và độ lệch chuẩn của thành phần đó trên toàn bộ training data.

### Scaling to unit length

Chuẩn hóa các thành phần của mỗi vector dữ liệu sao cho toàn bộ vector có độ lớn (Euclid, tức norm 2) bằng 1. Việc này có thể được thực hiện bằng:

## Gradient Descent và một số biến thể

### Gradient Descent

Trong Machine Learning nói riêng và Toán Tối Ưu nói chung, chúng ta thường xuyên phải tìm giá trị nhỏ nhất (hoặc là lớn nhất) của một hàm số nào đó. Nhìn chung, việc tìm *global minimum* của các hàm mất mát trong Machine Learning là rất phức tạp, thậm chí là bất khả thi. Thay vào đó, người ta thường cố gắng tìm các điểm *local minimum*, và ở một mức độ nào đó, coi đó là nghiệm cần tìm của bài toán.

Các điểm *local minimum* là nghiệm của phương trình đạo hàm bằng 0. Nếu bằng một cách nào đó có thể tìm được toàn bộ (hữu hạn) các điểm cực tiểu, ta chỉ cần thay từng điểm local minimum đó vào hàm số rồi tìm điểm làm cho hàm có giá trị nhỏ nhất. Tuy nhiên, trong hầu hết các trường hợp, việc giải phương trình đạo hàm bằng 0 là bất khả thi. Nguyên nhân có thể đến từ sự phức tạp của dạng của đạo hàm, từ việc các điểm dữ liệu có số chiều lớn, hoặc từ việc có quá nhiều điểm dữ liệu.

Hướng tiếp cận phổ biến nhất là xuất phát từ một điểm mà chúng ta coi là gần với nghiệm của bài toán, sau đó dùng một phép toán lặp để tiến dần đến điểm cần tìm, tức đến khi đạo hàm gần với 0. Gradient Descent (viết gọn là GD) và các biến thể của nó là một trong những phương pháp được dùng nhiều nhất.

Quy tắc của GD: **luôn luôn đi ngược hướng với đạo hàm** để tiến tới đạo hàm của hàm cần tối ưu (hàm mất mát) bằng 0.

Trong đó:

* là tập các tham số của mô hình
* *J()* là hàm mất mát của mô hình
* *η* là tốc độ học (learning rate)
* là đạo hàm của hàm mất mát tại

### Một số biến thể của Gradient Descent

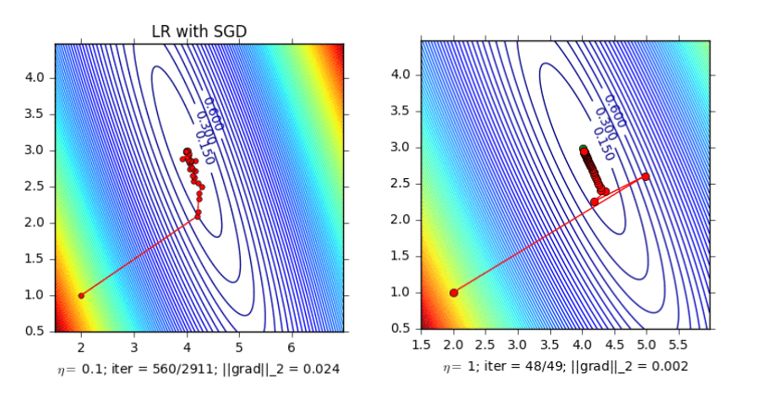
1. Stochastic Gradient Descent

Trong thuật toán này, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ một điểm dữ liệu *xi* rồi cập nhật θdựa trên đạo hàm này. Việc này được thực hiện với từng điểm trên toàn bộ dữ liệu.

SGD có tốc độ hội tụ nhanh so với GD, phù hợp với các bài toán có lượng cơ sở dữ liệu lớn (chủ yếu là Deep Learning) và online learning.

Thứ tự lựa chọn điểm dữ liệu: sau mỗi epoch, chúng ta cần shuffle (xáo trộn) thứ tự của các dữ liệu để đảm bảo tính ngẫu nhiên

Quy tắc cập nhật của SGD là:



Hình 1. Đường đi của nghiệm với SGD (trái) và GD (phải)

Hình bên trái mô tả đường đi của nghiệm. Chúng ta thấy rằng đường đi khá là zigzag chứ không mượt như khi sử dụng GD. Điều này là dễ hiểu vì một điểm dữ liệu không thể đại diện cho toàn bộ dữ liệu được. Tuy nhiên, chúng ta cũng thấy rằng thuật toán hội tụ khá nhanh đến vùng lân cận của nghiệm. Với 1000 điểm dữ liệu, SGD chỉ cần gần 3 epoches (2911 tương ứng với 2911 lần cập nhật, mỗi lần lấy 1 điểm). Nếu so với con số 49 vòng lặp (epoches) như kết quả tốt nhất có được bằng GD, thì kết quả này lợi hơn rất nhiều.

Đây chính là ưu điểm của SGD - **hội tụ rất nhanh**.

1. Mini-batch Gradient Descent

Khác với SGD, mini-batch sử dụng một số lượng *n* lớn hơn 1 (nhưng vẫn nhỏ hơn tổng số dữ liệu *N* rất nhiều). Giống với SGD, Mini-batch Gradient Descent bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu rồi chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu. Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật. Công thức có thể viết dưới dạng:

Mini-batch GD được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong Deep Learning. Giá trị *n* thường được chọn là khoảng từ 50 đến 100.

### Các thuật toán tối ưu Gradient Descent

1. Momentum

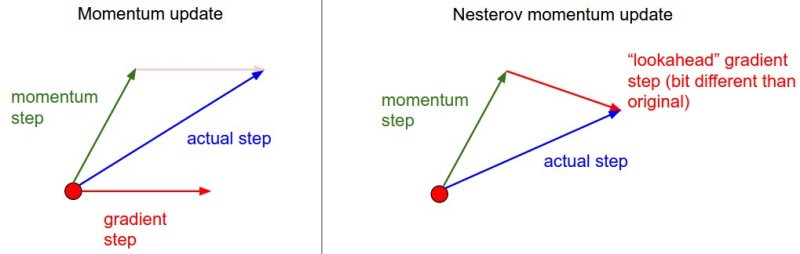
Nhược điểm của GD là nó rất dễ bị mắc kẹt ở điểm *local minimum* (do tại đó gradient bằng 0 và nghiệm không được cập nhật nữa).

Gọi lượng thay đổi để cập nhật vị trí mới tại thời điểm t là *vt.* Momentum tính *vt* sao cho nó vừa mang thông tin của độ dốc (đạo hàm), vừa mang thông tin của đà (vận tốc *vt-1*trước đó):

1. Nesterov accelerated gradient (NAG)

Momentum giúp hòn bi vượt qua được điểm locaminimum, tuy nhiên, có một hạn chế chúng ta có thể thấy: Khi tới gần đích, momemtum vẫn mất khá nhiều thời gian trước khi dừng lại. Lý do lại cũng chính là vì có đà. Có một phương pháp khác tiếp tục giúp khắc phục điều này, phương pháp đó mang tên Nesterov accelerated gradient (NAG), giúp cho thuật toán hội tụ nhanh hơn.

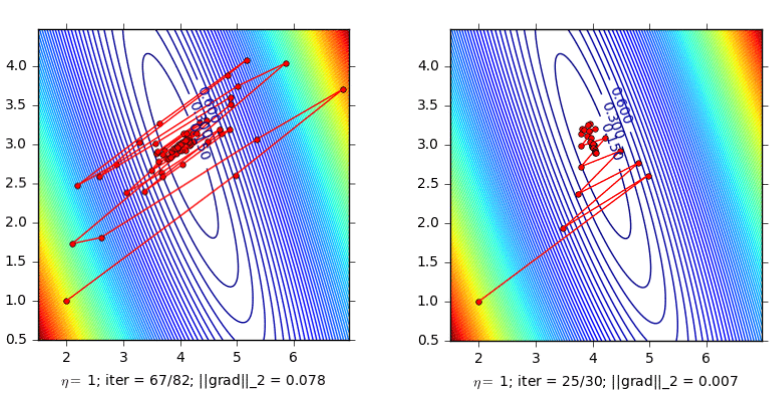
**Ý tưởng chính** là dự đoán hướng đi trong tương lai, tức nhìn trước một bước. Thay vì sử dụng gradient của điểm hiện tại, NAG đi trước một bước, sử dụng gradient của điểm tiếp theo.



Hình 2. Ý tưởng của Momentum và Nesterov accelerated gradient

Công thức cập nhật của NAG:

So sánh Momentum và NAG cho bài toán Linear Regression:



Hình 3. Minh họa thuật toán Momentum và NAG

Hình bên trái là đường đi của nghiệm với phương pháp Momentum. Nghiệm đi khá là zigzag và mất nhiều vòng lặp hơn. Hình bên phải là đường đi của nghiệm với phương pháp NAG, nghiệm hội tụ nhanh hơn, và đường đi ít zigzag hơn.

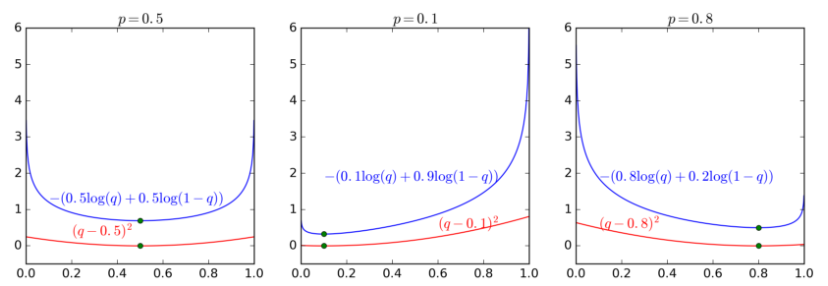
1. Các thuật toán khác

Ngoài hai thuật toán trên, có rất nhiều thuật toán nâng cao khác được sử dụng trong các bài toán thực tế, đặc biệt là các bài toán Deep Learning, như Adagrad, RMSprop, Adam…

## Cross Entropy cho bài toán Softmax Regression

Cross entropy giữa 2 phân phối rời rạc **p** và **q** được định nghĩa là:

Ví dụ trong trường hợp C=2 và p1 lần lượt nhận các giá trị 0.5, 0.1 và 0.8.



Hình 4. So sánh giữa hàm cross entropy và hàm bình phương khoảng cách.

Các điểm màu xanh lục thể hiện các giá trị nhỏ nhất của mỗi hàm.

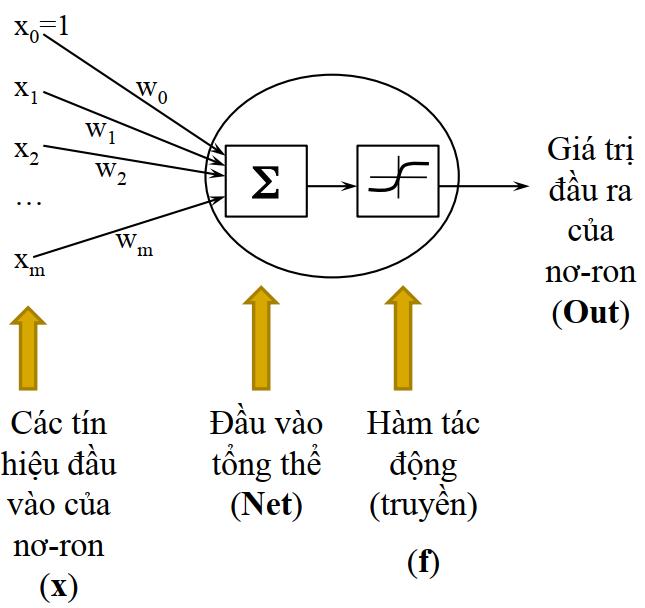
Có hai nhận xét quan trọng sau đây:

* Giá trị nhỏ nhất của cả hai hàm số đạt được khi q=p tại hoành độ của các điểm màu xanh lục.
* Quan trọng hơn, hàm cross entropy nhận giá trị rất cao (tức loss rất cao) khi q ở xa p. Trong khi đó, sự chênh lệch giữa các loss ở gần hay xa nghiệm của hàm bình phương khoảng cách (q−p)2 là không đáng kể. Về mặt tối ưu, hàm cross entropy sẽ cho nghiệm gần với p hơn vì những nghiệm ở xa bị trừng phạt rất nặng.

Hai tính chất trên đây khiến cho cross entropy được sử dụng rộng rãi khi tính khoảng cách giữa hai phân phối xác suất.

## Multi-layer Perceptron và Backpropagation

### Cấu trúc và hoạt động của một nơ-ron



Hình 5. Cấu trúc và hoạt động của một nơ-ron

* **Các tín hiệu đầu vào (input signals)** của nơ-ron (*xi, i=1..m*)
* Mỗi tín hiệu đầu vào *xi* gắn với một trọng số *wi*
* **Trọng số điều chỉnh (bias)** *w0* (với *x0* = 1)
* **Đầu vào tổng thể (Net input)** là một hàm tích hợp của các tín hiệu đầu vào -

*Net(****w, x****)*

* **Hàm kích hoạt (Activation function)** tính giá trị đầu racủa nơ-ron - *f(Net(****w, x****))*
* **Giá trị đầu ra (Output)** của nơ-ron: *Out=f(Net(****w, x****))*

#### Đầu vào tổng thể:

Đầu vào tổng thể (net input) thường được tính toán bởi một hàm tuyến tính:

#### Hàm kích hoạt

1. Hàm *sign* không được sử dụng trong MLP

Hàm *sign* (còn gọi là hard-threshold), không được sử dụng vì hai lý do: đầu ra không liên tục, và đạo hàm tại hầu hết các điểm bằng 0 (trừ điểm 0 không có đạo hàm). Việc đạo hàm bằng 0 này khiến cho các thuật toán gradient-based (ví dụ như Gradient Descent) không hoạt động.

1. Sigmoid và tanh

|  |  |
| --- | --- |
| https://machinelearningcoban.com/assets/14_mlp/sigmoid.jpeg  Hình 6. Hàm sigmoid | https://machinelearningcoban.com/assets/14_mlp/tanh.jpeg  Hình 7. Hàm tanh |

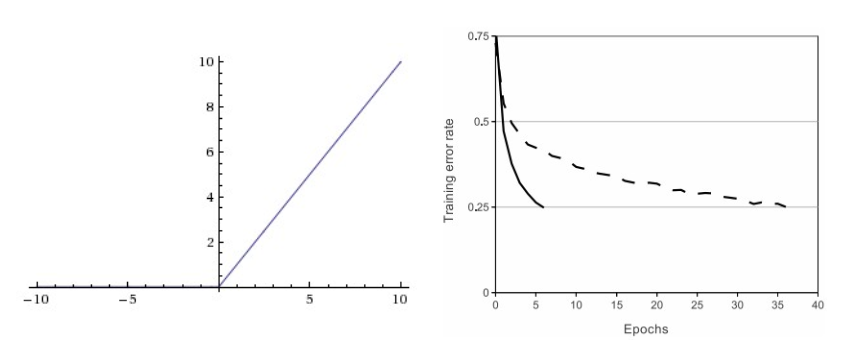
Hàm *sigmoid* và *tanh* được sử dụng nhiều trong quá khứ, có ưu điểm:

* Liên tục và đạo hàm liên tục
* Đạo hàm được biểu diễn bằng một hàm của chính nó:

Những năm gần đây, hai hàm số này ít khi được sử dụng. Nó có một nhược điểm cơ bản:

**Sigmoid saturate and kill gradients**: 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.

1. ReLU



Hình 8. Hàm ReLU và tốc độ hội tụ khi so sánh với hàm tanh

ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Ưu điểm chính của nó là:

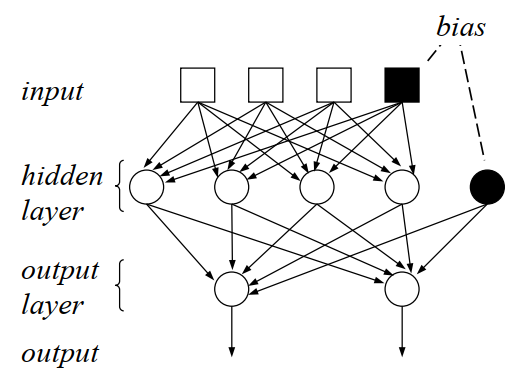
* ReLU được chứng minh giúp cho việc training các Deep Networks nhanh hơn rất. Hình 8 (phải) so sánh sự hội tụ của SGD khi sử dụng hai activation function khác nhau: ReLU và tanh. Sự tăng tốc này được cho là vì ReLU được tính toán gần như tức thời và gradient của nó cũng được tính cực nhanh với gradient bằng 1 nếu đầu vào lớn hơn 0, bằng 0 nếu đầu vào nhỏ hơn 0.
* Mặc dù hàm ReLU không có đạo hàm tại *s =* 0, trong thực nghiệm, người ta vẫn thường định nghĩa ReLU’(0) = 0 và khẳng định thêm rằng, xác suất để input của một unit bằng 0 là rất nhỏ.

Hàm ReLU có nhiều biến thể khác như Noisy ReLU, Leaky ReLu, ELUs.

1. Một vài lưu ý:

* Với các bài toán classification, output layer thường sử dụng hàm *softmax* giúp tính xác suất để một điểm dữ liệu rơi vào mỗi class.
* Mặc dù activation function cho mỗi unit có thể khác nhau, trong cùng một network, activation như nhau thường được sử dụng. Điều này giúp cho việc tính toán được đơn giản hơn.

### Kiến trúc mạng nơ-ron



Hình 9. Kiến trúc mạng nơ-ron

Kiến trúc của một mạng nơ-ron xác định bởi:

* Số lượng các tín hiệu đầu vào và đầu ra
* Số lượng các tầng
* Số lượng các nơ-ron trong mỗi tầng
* Số lượng các trọng số (các liên kết) đối với mỗi nơ-ron
* Cách thức các nơ-ron (trong một tầng, hoặc giữa các tầng) liên kết với nhau
* Những nơ-ron nào nhận các tín hiệu điều chỉnh lỗi

Một mạng nơ-ron phải có:

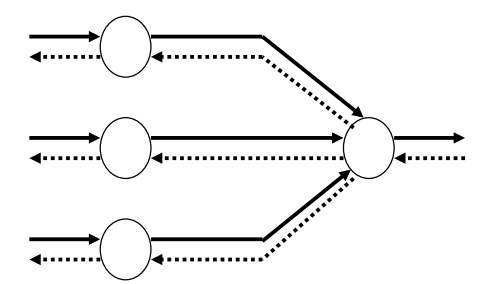
* Một tầng đầu vào (input layer)
* Một tầng đầu ra (output layer)
* Không, một, hoặc nhiều tầng ẩn (hidden layer)

Sau đây, chúng ta sẽ tìm hiểu mạng nơ-ron **liên kết đầy đủ (fully connected):** mọi đầu ra từ một tầng liên kết với mọi nơ-ron của tầng kế tiếp

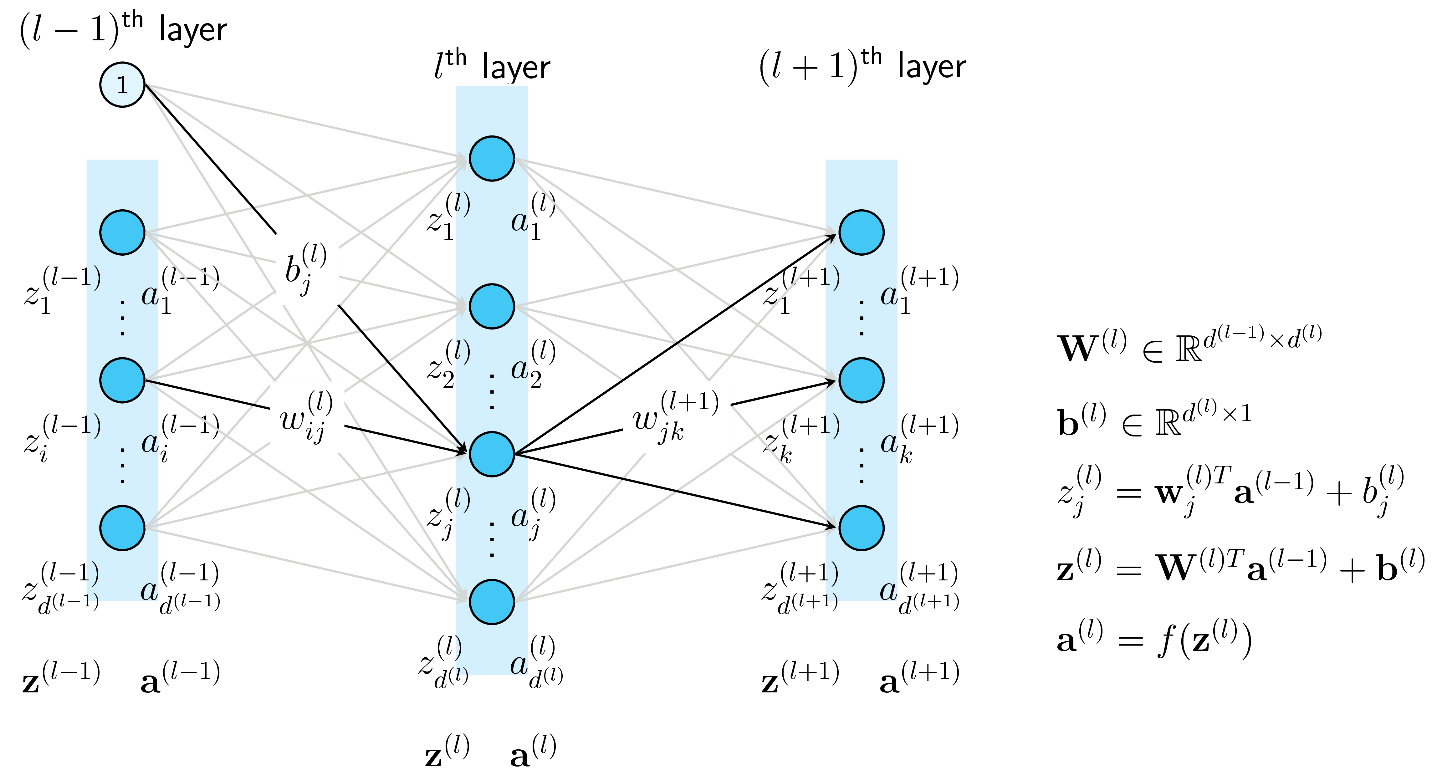
### Backpropagation

Giải thuật Back Propagation bao gồm 2 giai đoạn:

* Giai đoạn **lan truyền tiến tín hiệu (Signal forward)**: Các tín hiệu đầu vào (vectơ các giá trị đầu vào) được lan truyền tiến từ tầng đầu vào đến tầng đầu ra (đi qua các tầng ẩn)
* Giai đoạn **lan truyền ngược lỗi (Error backward)**:
* Căn cứ vào giá trị đầu ra mong muốn của vectơ đầu vào, hệ thống tính toán giá trị lỗi
* Bắt đầu từ tầng đầu ra, giá trị lỗi được lan truyền ngược qua mạng, từ tầng này qua tầng khác (phía trước), cho đến tầng đầu vào
* Việc lan truyền ngược lỗi (error back-propagation) được thực hiện thông qua việc tính toán (một cách truy hồi) giá trị gradient cục bộ của mỗi nơ-ron



Hình 10. Minh họa giai đoạn lan truyền tiến tín hiệu (nét liền) và giai đoạn lan truyền ngược lỗi (nét đứt)



Hình 11. Mô phỏng cách tính Back propagation

Xét 3 layer liên tiếp , và

* là số nơ-ron ở layer
* là ma trận trọng số liên kết giữa layer và
* là vector bias của layer
* lần lượt là vector đầu vào và đầu ra của layer

1. Giai đoạn **lan truyền tiến tín hiệu (Signal forward):**

1. Giai đoạn **lan truyền ngược lỗi (Error backward)**:

Gọi *J*(**W**, **b**; **X**, **Y**) là một hàm mất mát của bài toán, trong đó:

* **W**, **b** là tập tất cả các ma trận trọng số giữa các layers và biases của mỗi layer
* **X, Y** là cặp dữ liệu huấn luyện với mỗi cột tương ứng với một điểm dữ liệu

Ta tính gradient ngược từ layer cuối cùng đến layer đầu tiên. Việc tính toán gradient của các layer trước được thực hiện dựa trên một quy tắc *chain rule*, tức đạo hàm của hàm hợp.

với:

Trong đó và là hàng thứ j của ma trận .

Dấu sigma tính tổng ở biến đổi thứ ba vì đóng góp vào việc tính tất cả các .

**Tóm tắt:** việc tính toán các đào hàm theo được tính như sau:

Đặt .

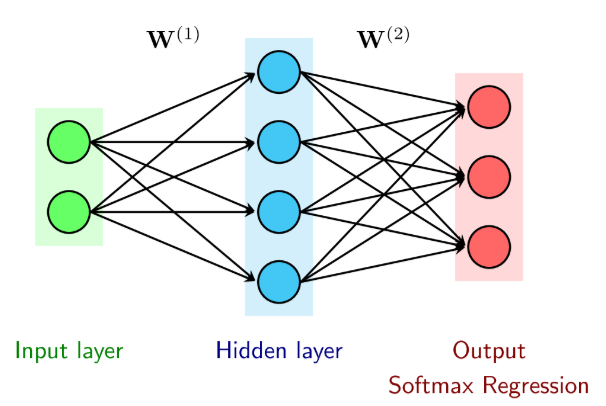
1. Với output layer, tính
2. Từ đó suy ra:
3. Với *l* = *L*-1, *L*-2,…,1, tính:

trong đó ⊙ là *element-wise* *product* hay *Hadamard product* tức lấy từng thành phần của hai vector nhân với nhau để được vector kết quả.

1. Cập nhật đạo hàm cho ma trận trọng số và vector biases:

### Cài đặt

Áp dụng thuật toán Back propagation cho mạng nơ-ron 1 hidden layer:



Hình 12. Mạng nơ-ron với 1 hidden layer

Các hàm sử dụng trong mạng:

* Hàm kích hoạt của hidden layer: ReLU
* Hàm kích hoạt của output layer: Softmax
* Hàm mất mát: Cross-entropy

1. Forward:
2. Backward: