Optimization, Logistic Regression, Softmax Regression

Thai Trung Hieu (hieutt42) - Fsoft QAI Quy Nhon

Ngày 22 tháng 4 năm 2022

Một số topic quan trọng

1. Tối ưu - Optimization (thuật toán Gradient Descent và các biến thể.)
2. Logistic Regression.
3. Softmax Regression.

Why we need to optimize?

Trong Machine Learning, ta thường đi tìm một mô hình $f(x)$ để mô tả dữ liệu. Hiển nhiên ta muốn việc mô tả này phải **tốt nhất có thể**. Để tìm mô hình **tốt nhất có thể**, ta phải giải 1 bài toán tối ưu. Một ví dụ chính là bài toán Maximum likelihood estimation nói ở bài trước.

Optimization problem

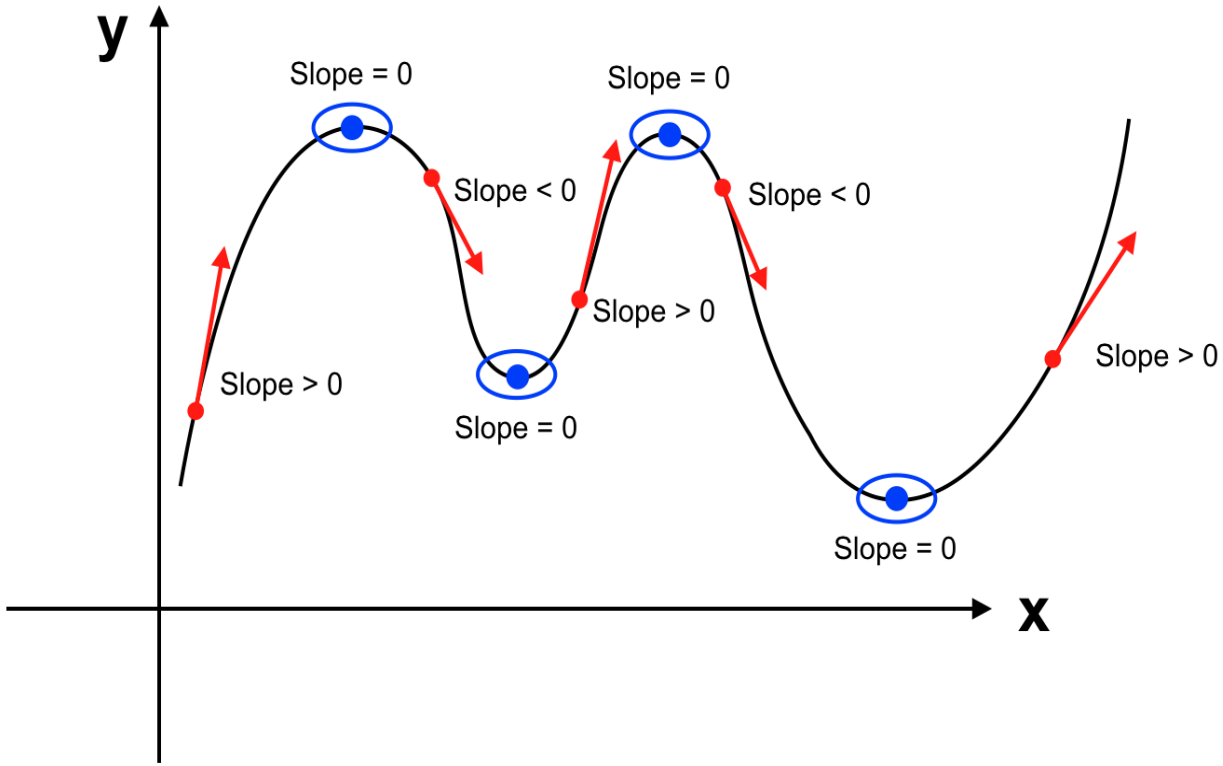
Trong bài này, khi nói tới bài toán tối ưu, ta sẽ luôn hiểu là bài toán đi tìm giá trị bé nhất. Đối với bài toán tìm giá trị lớn nhất của 1 hàm số $f(x)$, ta có thể đặt

$$g(x) = -f(x)$$

và chuyển sang giải bài toán tìm giá trị bé nhất của $g(x)$.

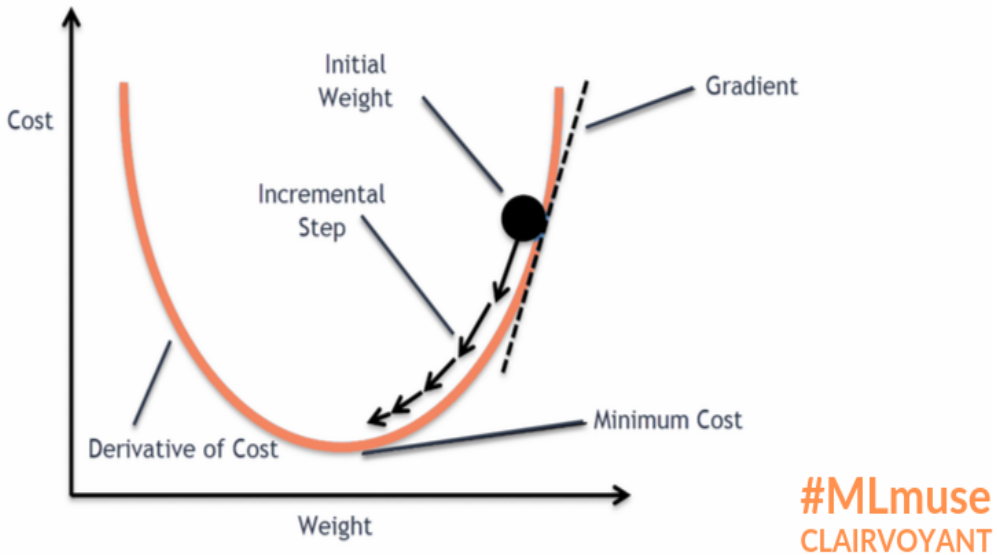
Derivative = Slope

Đạo hàm (hàm chỉ đường) của hàm số $f(x)$ tại điểm x_0 là hệ số góc (slope) của tiếp tuyến tại điểm $(x_0, f(x_0))$.



Gradient

Nhắc lại: đạo hàm (derivative, gradient) của hàm số $f(x)$ tại điểm x_0 cho biết xu hướng di chuyển của $f(x)$ tại x_0: tăng, giảm hay đi ngang.



Gradient Descent

Giả sử x_t là điểm tìm được sau vòng lặp thứ t. Ta cần đưa x_t về càng gần x_* càng tốt.

- Nếu đạo hàm của hàm số x_t dương thì x_t nằm bên phải của x_*. Để điểm x_{t+1} tìm được ở vòng lặp tiếp theo gần x_* hơn, ta cần di chuyển về bên trái, hay $x_{t+1} < x_t$.
- Nếu đạo hàm của hàm số x_t âm thì x_t nằm bên trái của x_*. Để điểm x_{t+1} tìm được ở vòng lặp tiếp theo gần x_* hơn, ta cần di chuyển về bên phải, hay $x_{t+1} > x_t$.

Gradient Descent

Tóm lại, để đi từ x_t đến x_{t+1} ta cần di chuyển ngược dấu với đạo hàm. Có nghĩa là

$$x_{t+1} = x_t + d$$

trong đó d là 1 số âm nếu đạo hàm $f'(x_t) > 0$ và d là 1 số âm nếu đạo hàm $f'(x_t) < 0$.

Trong thuật toán Gradient Descend, d được chọn bằng $-\eta f'(x_t)$. Trong đó η là 1 số dương, được gọi là tốc độ học (learning rate). Vậy

$$x_{t+1} = x_t - \eta f'(x_t).$$

Công thức trên có thể được viết theo kí hiệu của thuật toán như sau:

$$x \leftarrow x - \eta f'(x).$$

Gradient Descend cho hàm nhiều biến

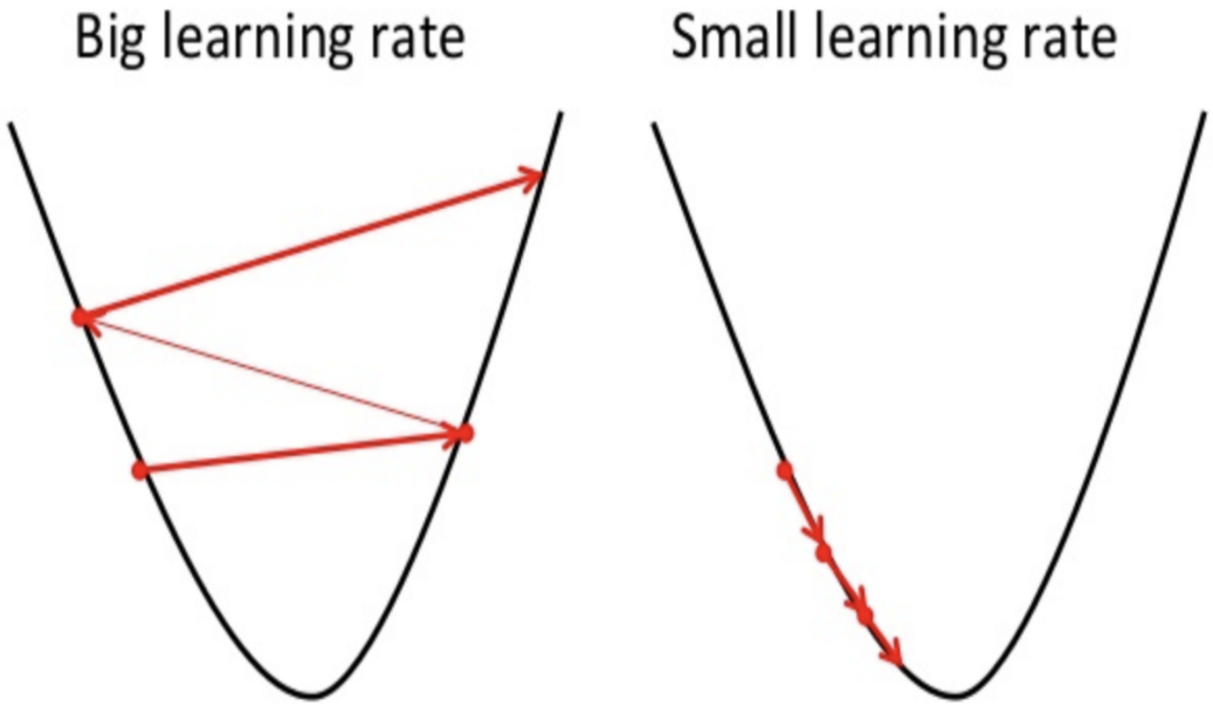
Giả ta cần tìm giá trị bé nhất của hàm $f(\theta)$ trong đó θ là tập hợp các tham số cần tìm. Đạo hàm (gradient) của hàm số tại θ là $\nabla_\theta f$. Tương tự như đối với hàm 1 biến số, thuật toán Gradient descend cho hàm nhiều biến cũng bắt đầu với 1 điểm θ_0 nào đó, ở vòng lặp thứ t, tham số θ được cập nhật bởi công thức:

$$\theta_{t+1} = \theta_t - \eta \cdot [\nabla_\theta f(\theta_t)].$$

Công thức trên có thể được viết theo kí hiệu của thuật toán như sau:

$$\theta \leftarrow \theta - \eta \cdot [\nabla_\theta f(\theta)].$$

Learning rate

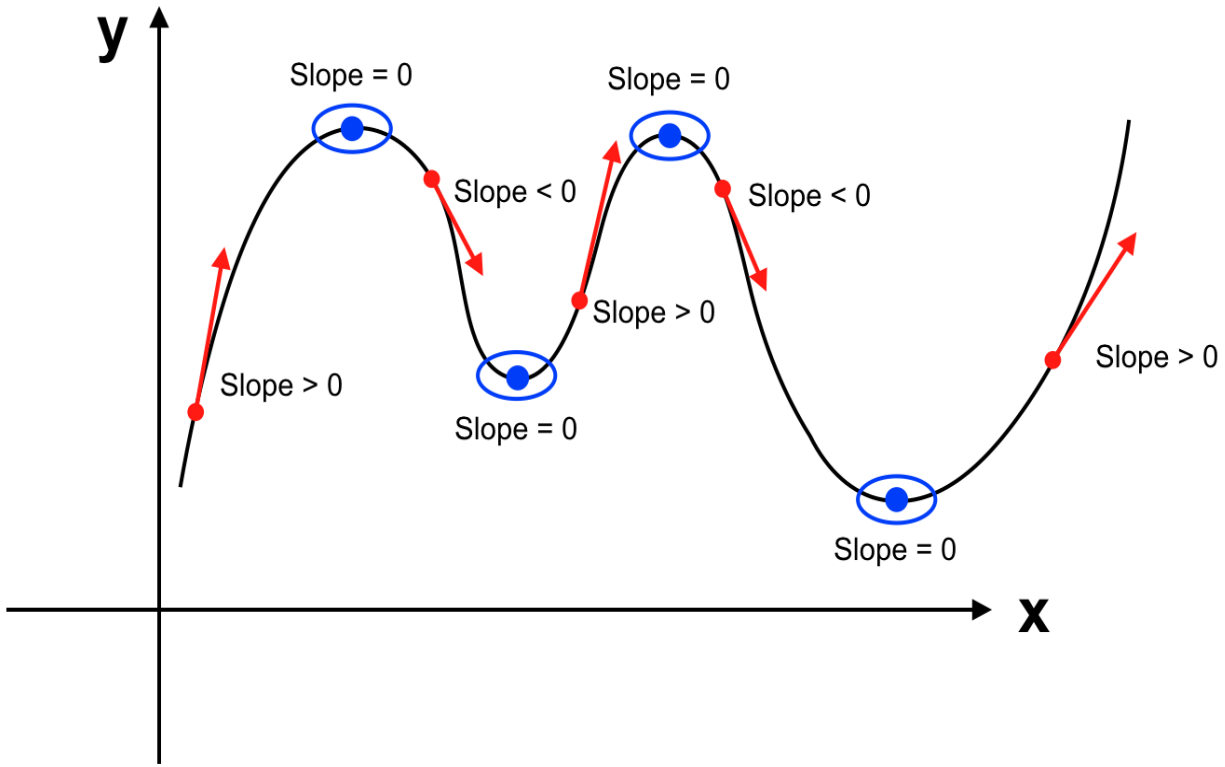


Learning rate

Việc chọn learning rate η ảnh hưởng rất nhiều đến việc tìm ra điểm cực tiểu.

- Nếu η quá nhỏ, tốc độ di chuyển từ x_t đến x_* sẽ chậm. Máy tính sẽ mất nhiều thời gian để tìm đến được x_*.
- Ngược lại, nếu η quá lớn, bước nhảy từ x_t đến x_{t+1} có thể sẽ quá lớn khiến thuật toán không tìm được điểm tối ưu.

Bao giờ thì thuật toán có thể dừng lại? Thuật toán dừng lại khi độ lớn của đạo hàm đủ nhỏ, tức là nó nhỏ hơn 1 giá trị c (thường được gọi là threshold) cho trước. Giá trị c thường được chọn là 1 số rất nhỏ, ví dụ 0.001. Khi đó, ta coi như giá trị của đạo hàm bằng 0 (quá nhỏ, không đáng kể).



Gradient Descend algorithm

Tóm lại, thuật toán Gradient descend để tìm giá trị bé nhất của hàm số $f(x)$ được thực hiện như sau.
Trước khi thực hiện thuật toán Gradient Descend ta cần 1 số sự chuẩn bị.

- Tính đạo hàm $f'(x)$.
- Chọn c. Ví dụ chọn $c = 0.001$.
- Chọn learning rate η. Ví dụ chọn $\eta = 0.1$.

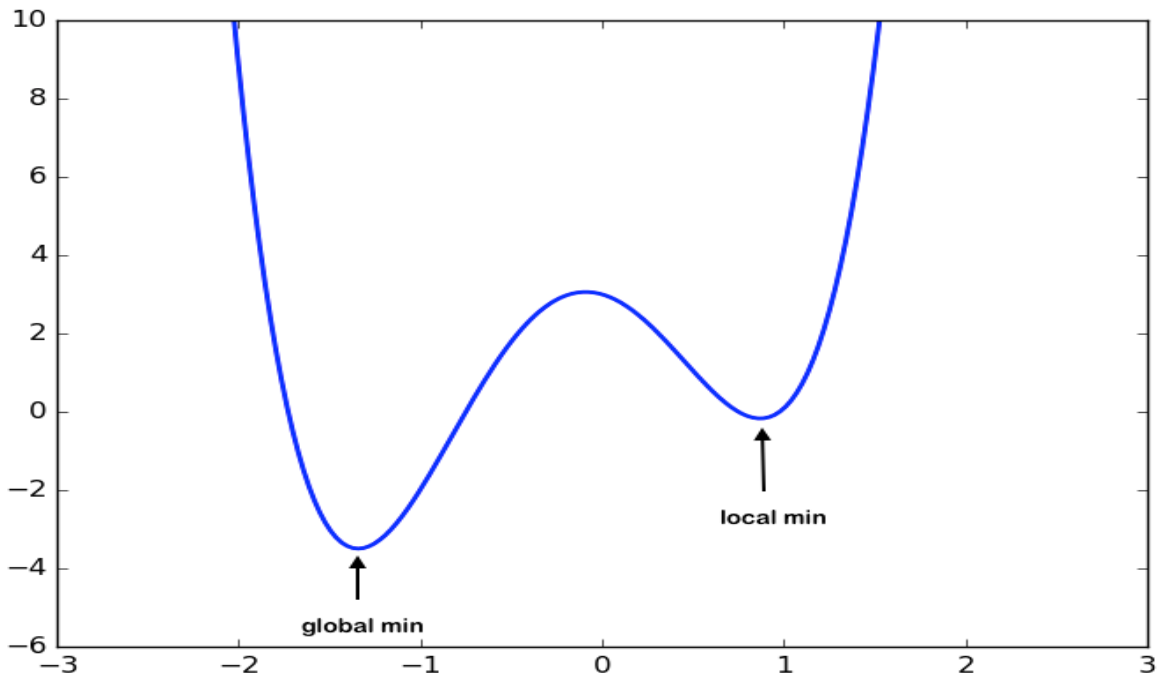
Gradient Descend algorithm

Thuật toán gradient descend được tiến hành như sau.

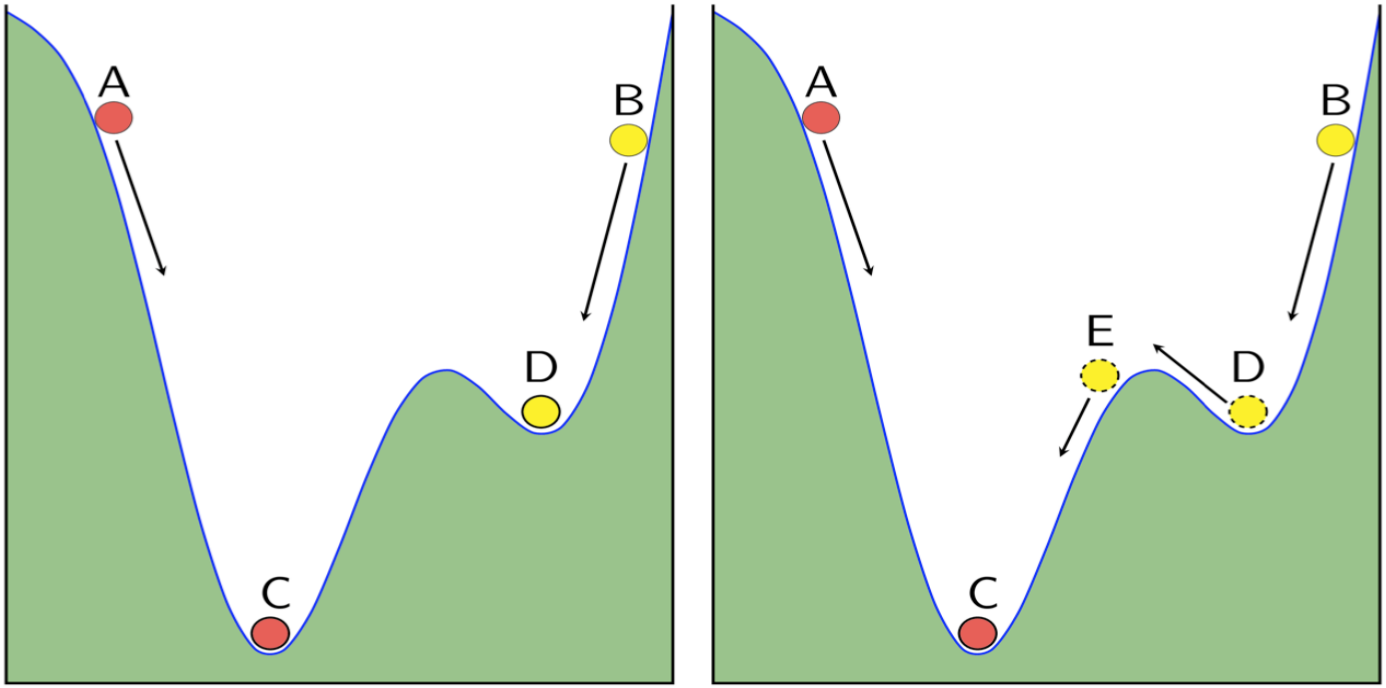
1. Chọn ngẫu nhiên 1 điểm x_0 để bắt đầu. Bước này được xem như vòng lặp thứ 0. Tính $f'(x_0)$.
2. Ở vòng lặp thứ 1, ta cập nhật $x_1 = x_0 - \eta f'(x_0)$. Tính $f'(x_1)$.
3. Ở vòng lặp thứ 2, ta cập nhật $x_2 = x_1 - \eta f'(x_1)$. Tính $f'(x_2)$.
4. Tiếp tục thực hiện vòng lặp thứ 3. Một cách tổng quát, tại vòng lặp thứ t, ta cập nhật $x_{t+1} = x_t - \eta f'(x_t)$.
5. Thuật toán dừng lại khi $|f'(x_t)| < c$. Khi đó nghiệm cần tìm là x_t.

Local minima problem

Thuật toán gradient descend nói trên có thể bị mắc kẹt tại 1 điểm cực tiểu địa phương.



Thuật toán GD bắt đầu bằng việc chọn ngẫu nhiên 1 điểm xuất phát x_0. Nếu may mắn bắt đầu từ điểm A như hình vẽ thì thuật toán sẽ tìm đến điểm cực tiểu C mong muốn. Tuy nhiên, nếu điểm xuất phát là B thì có khả năng thuật toán sẽ dừng ở cực tiểu địa phương D. Đây là điều ta **không mong muốn**.



b) GD

c) GD with momentum

Gradient Descend with momentum

- Làm sao để thuật toán thoát khỏi cực tiểu địa phương?
- Tưởng tượng việc thuật toán đi từ điểm xuất phát B đến cực tiểu địa phương D giống như việc một quả bóng lăn từ B đến D. Nếu quả bóng đủ nặng, quán tính sẽ hất quả bóng văng ra khỏi D để đến E. Khi đó, nó có thể tìm đến C theo thuật toán GD.
- Gradient descend with momentum là 1 cách giúp thuật toán thoát khỏi cực tiểu địa phương. Momentum được hiểu là đà/quán tính.

Gradient Descend with momentum

Công thức (cập nhật tham số) của thuật toán Gradient Descend with momentum như sau.

$$\theta_{t+1} = \theta_t - v_t = \theta_t - \eta \cdot [\nabla_\theta f(\theta_t)] - \gamma v_{t-1}.$$

Trong đó v_t là 1 đại lượng mang thông tin về độ dốc tại thời điểm hiện tại và vận tốc v_{t-1} ở thời điểm kề trước.

GD in Machine Learning

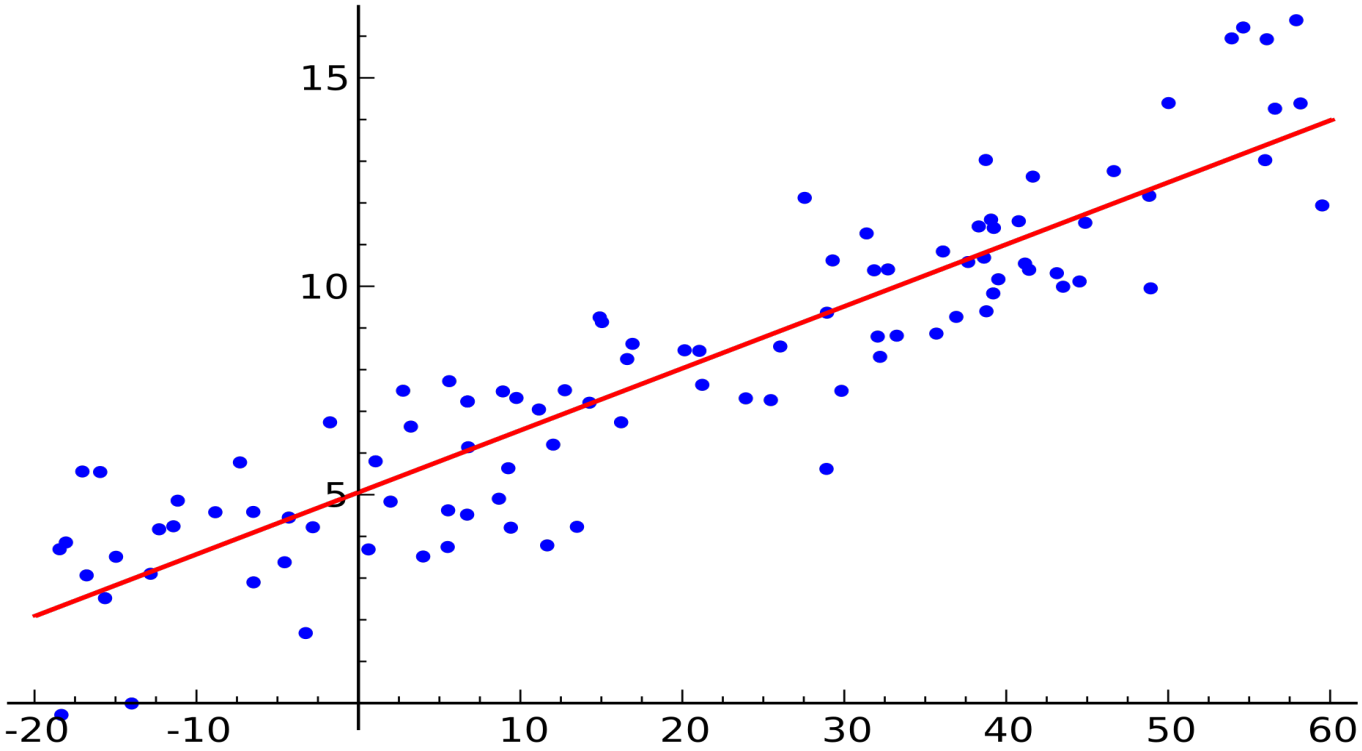
Xét 1 ví dụ với bài toán linear regression (hồi quy tuyến tính). Hàm mất mát của mô hình linear regression và đạo hàm của nó lần lượt là:

$$\mathscr{L}(w) = \frac{1}{2N}\left\|y - X^T w\right\|_2^2$$

$$\mathscr{L}'(w) = \frac{1}{N}X(X^T w - y)$$

Trong đó $X = [x_1, x_2, \ldots, x_N], y = [y_1, y_2, \ldots, y_N]$ và (x_i, y_i) là 1 điểm data (điểm màu xanh, xem slide tiếp theo).

Example: Linear regression



Frame Title

Hãy nhìn vào công thức đạo hàm

$$\mathscr{L}'(w) = \frac{1}{N} X(X^T w - y).$$

Hàm phụ thuộc N

Khi chạy thuật toán gradient descend, ta cần tính gradient để cập nhật tham số θ. Đối với model linear regression, ta dùng công thức ở trên để tính gradient. Nhìn vào công thức, để thuật toán bắt đầu chạy, ta chọn ngẫu nhiên w. (X, y) là data cho trước. Giá trị N là số điểm dữ liệu được sử dụng hay chính là số chiều của vector X. Tuỳ vào cách chọn N mà ta có 1 số thuật toán gradient descend khác nhau.

Stochastic Gradient Descent

Thuật toán GD nói trên còn được gọi là batch GD. Khi cập nhật tham số, ta sử dụng tất cả các điểm dữ liệu x_i. Khi lượng data lớn, việc tính toán thế này tốn rất nhiều thời gian.
Trong SGD, tại 1 thời điểm, ta chỉ tính đạo hàm của hàm mất mát dựa trên chỉ 1 điểm dữ liệu x_i rồi cập nhật θ dựa trên đạo hàm này.

Epoch

Mỗi lần duyệt qua một lượt tất cả các điểm trên toàn bộ tập data được gọi là 1 epoch. Như vậy, trong batch GD, tại mỗi epoch thuật toán cập nhật tham số θ đúng 1 lần. Nhưng trong SGD, tại mỗi epoch, thuật toán cập nhật tham số đến K lần với K là số điểm dữ liệu trong dataset.

Mini batch gradient descent

Mini batch GD là 1 sự cân bằng giữa batch GD và SGD. Để cập nhật tham số θ

- Batch GD dùng toàn bộ K điểm dữ liệu.
- SGD chỉ dùng 1 điểm.
- Mini batch GD dùng k điểm với $1 < k < K$.

Summary on gradient descend

Ở trên, ta nghiên cứu 3 "thuật toán" gradient descend. 3 "thuật toán" này khác nhau ở lượng data cần dùng để tính gradient.

- Batch gd: dùng toàn bộ data.
- Stochastic gd: chỉ dùng 1 điểm data.
- Mini-batch gd: sử dụng nhiều hơn 1 điểm data nhưng không dùng toàn bộ data (như batch gd).

Warning on using gd

- Giới hạn số vòng lặp trước khi chạy thuật toán.
-

Maximum likelihood estimation

Ta nhắc lại Maximum likelihood estiamtion. Giả sử có các điểm dữ liệu $x_1, x_2, \ldots, x_N$. Giả sử ta biết rằng các điểm dữ liệu này tuân theo một phân phối nào đó được mô tả bởi bộ tham số θ. Maximum likelihood estimation là **tìm bộ tham số** θ sao cho xác suất sau đây đạt giá trị lớn nhất:

$$p(x_1, \ldots, x_N|\theta).$$

Trong tối ưu, ta thường diễn đạt việc này bằng kí hiệu sau

$$\theta = \arg\max_k p(x_1, \ldots, x_N|\theta).$$

Maximum likelihood estimation

Chú ý: Xác suất có điều kiện $P(A|B)$ là xác suất xảy ra sự kiện A biết rằng B đã xả ra.
Giá trị **xác suất có điều kiện** $p(x_1|\theta)$ chính là xác suất xảy ra sự kiện x_1 biết rằng phân phối được mô tả bởi bộ tham số θ. Tương tự,

$$p(x_1, \ldots, x_N|\theta)$$

là xác suất để toàn bộ sự kiện $x_1, \ldots, x_N$ đồng thời xảy ra biết rằng phân phối được mô tả bởi bộ tham số θ.

Logistic Regression

Trong các bài toán phân loại, nhiều khi ta không thể nói khi nào thì đầu vào x rơi vào lớp y_1, khi nào thì rơi vào lớp y_2. Cùng lắm, ta chỉ có thể biết xác suất để x rơi vào class y_1, xác suất để rơi vào class y_2. Giá trị xác suất nào lớn hơn thì ta gán x cho lớp tương ứng.

Trong bài này, ta nghiên cứu mô hình logistic regression cho bài toán **phân loại 2 lớp**. Trong mô hình này, đầu ra là xác suất để input x thuộc vào lớp y.

Linear assumption

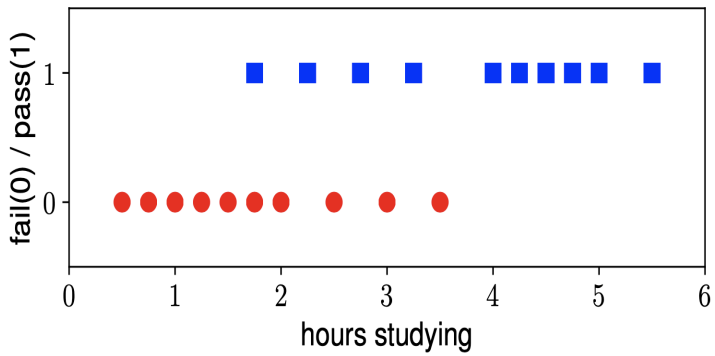
- Xét input $x = (x_1, \ldots, x_n)$, tham số của mô hình $w = (w_1, \ldots, w_n)$. Giả định tuyến tính (linear assumption) giả định rằng mối quan hệ giữa các thành phần khác nhau trong 1 điểm data là tuyến tính.
- Cụ thể, ta sẽ đưa giá trị $z = w^T x = w_1 x_1 + \ldots + w_n x_n$ vào mô hình.

Bảng sau đây là 1 dataset về thời gian ôn thi và kết quả thi của 20 sinh viên, trong đó cột Hours chỉ thời gian dành cho việc ôn thi. Cột Pass chỉ kết quả kì thi 0 = trượt (fail), 1 = đỗ (pass).

Hours	Pass	Hours	Pass	Hours	Pass	Hours	Pass
0.5	0	0.75	0	1	0	1.25	0
1.5	0	1.75	0	1.75	1	2	0
2.25	1	2.5	0	2.75	1	4	0
3.25	1	3.5	0	4	1	4.25	1
4.5	1	4.75	1	5	1	5.5	1

Hãy xây dựng 1 mô hình đánh giá khả năng thi đậu của sinh viên dựa trên thời gian ôn thi.

Đồ thị sau đây minh hoạ data trong bảng nói trên. Trong đó hình tròn màu đỏ là data ứng với trường hợp thi trượt (fail) còn hình vuông màu xanh là data ứng với trường hợp thi đỗ (pass).



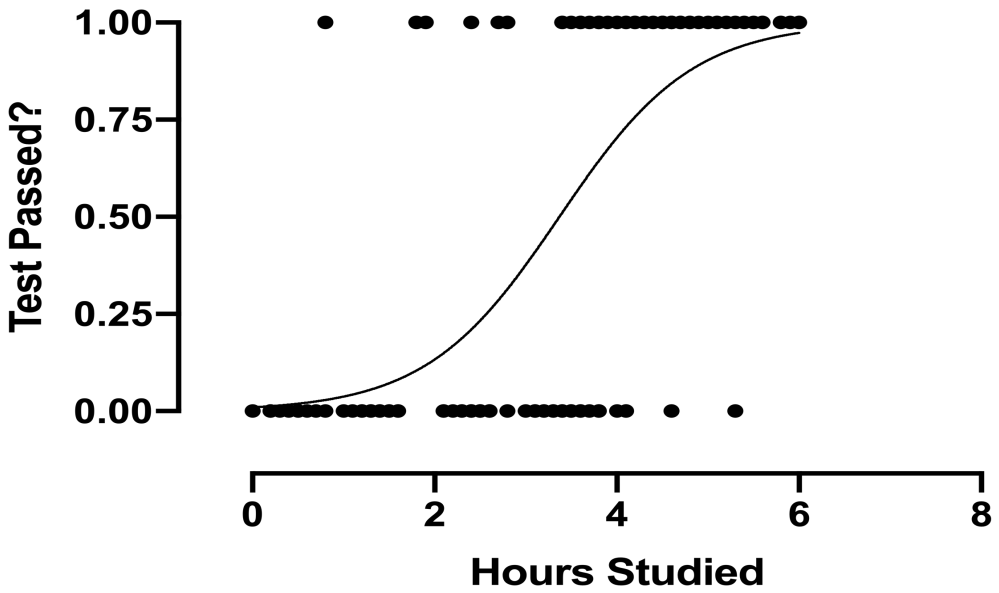
Nhìn vào dữ liệu này ta thấy rằng không có 1 mốc thời gian phân biệt rạch ròi kết quả thi fail/pass. Tức là không thể nói rằng, ví dụ, "nếu ôn thi tối thiểu 2 giờ đồng hồ thì sẽ thi đỗ".

- Điều duy nhất ta có thể nói là "**nói chung**, càng dành nhiều thời gian ôn thi thì **khả năng** đỗ càng cao". Có 1 trường hợp thời gian ôn thi là 3,5 giờ nhưng vẫn trượt trong khi có 1 trường hợp thời gian ôn thi chỉ là 1,75 giờ nhưng vẫn đỗ.
- Ta sẽ xây dựng 1 mô hình xác suất sao cho thời gian ôn thi càng lớn thì xác suất thi đỗ càng cao và ngược lại.

Cụ thể hơn, ta muốn tìm 1 đường cong sao cho:

- Càng đi về phía bên phải (tương ứng với thời gian ôn thi tăng) thì đường cong càng gần với đường thẳng $y = 1$ (tương ứng với pass).
- Ngược lại, càng đi về phía bên trái (tương ứng với thời gian ôn thi giảm) thì đường cong càng gần với trục Ox (tương ứng với fail).
- Đường cong này chỉ nằm giữa Ox và $y = 1$ thì giá trị xác suất chỉ có thể nằm giữa 0 và 1.
- Thêm nữa, ta muốn hàm số (đường cong) này phải có đạo hàm ở mọi nơi để có thể sử dụng gradient descend. (nói nôm na, hàm số có đạo hàm thì đồ thị của nó "cong" và ngược lại.)

Nói chung, 1 đường cong mong muốn sẽ trông giống hình dưới đây.

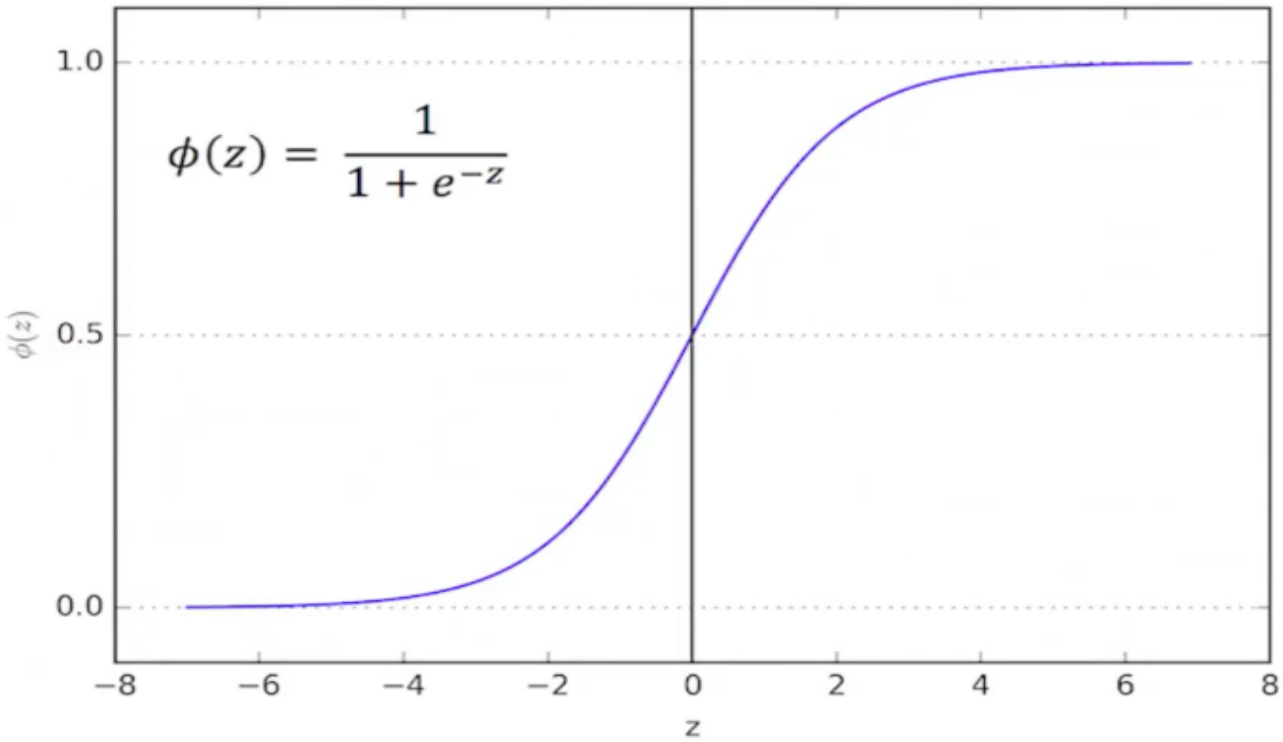


Sigmoid function

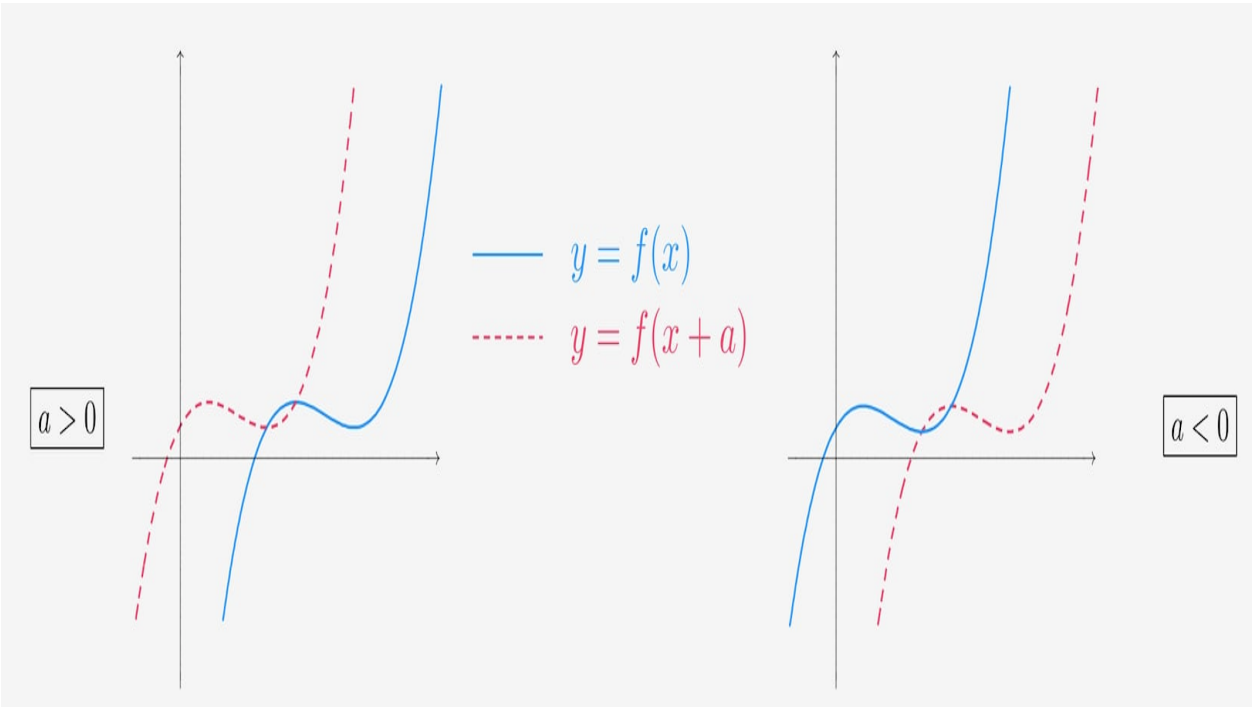
Trong deep learning, người ta thường sử dụng hàm sigmoid để mô hình đường cong nói trên. Hàm sigmoid được định nghĩa như sau:

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

Đồ thị của hàm sigmoid. Đường cong này thoả mãn cả 4 điều kiện đã nêu ở trên.



Nói riêng, đường cong (sigmoid) này vẫn chưa phải là mô hình phù hợp cho bài toán về thời gian ôn thi nói trên. Ta cần đẩy đường cong nói trên sang bên phải. Cụ thể hơn ta xét hàm $\phi(z-a)$ trong đó a là 1 số dương. Khi đó ta sẽ được 1 đường cong phù hợp với bài toán về thời gian ôn thi.



Hàm sigmoid có đạo hàm tại mọi z. Điều này là cơ sở để ta sử dụng thuật toán gradient descend.

$$\sigma'(z) = (1 - \frac{1}{e^z + 1})' = \frac{e^z}{(e^z + 1)^2}.$$

Generalised sigmoid function

Một cách tổng quát hơn, ta dùng hàm sigmoid có dạng

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

trong đó số b chỉ mức độ di chuyển sang trái/phải, số dương a chỉ độ dốc của đường cong. a càng lớn thì đường cong càng dốc và ngược lại a càng nhỏ thì đường cong càng thoải.

Giả sử xác suất để input x rơi vào lớp thứ nhất là

$$p(y = 1|x) = f(x, w)$$

trong đó $w = (a, b)$ là bộ tham số xác định hàm sigmoid đề cập ở slide trước. Khi đó, xác suất để x rơi vào lớp còn lại là

$$p(y = 0|x) = 1 - f(x, w).$$

Để mô hình gần với dữ liệu nhất, ta cần 2 xác suất trên phải lớn nhất có thể. Tuy nhiên, sẽ khó nếu ta phải tối ưu cả 2 hàm số cùng 1 lúc. Ta cần gộp 2 hàm nói trên lại làm 1 hàm số duy nhất.

Đặt $z_i = f(x_i, w)$ với x_i là điểm dữ liệu thứ i. Ta có thể gộp chung 2 xác suất nói trên trong 1 công thức như sau.

$$p(y_i|x_i) = z_i^{y_i}(1 - z_i)^{1-y_i}$$

trong đó y_i (0 hoặc 1) là nhãn đã biết của x_i.

Như vậy ta cần tìm tham số w sao cho xác suất

$$p(y|X;w)$$

đạt giá trị lớn nhất. Trong đó $X = (x_1, x_2, \ldots, x_n)$ là dữ liệu.
$y = (y_1, y_2, \ldots, y_n)$ trong đó y_i (0 hoặc 1) là nhãn của x_i.
Giá trị $p(y|X;w)$ là xác suất để toàn bộ input x_i rơi vào đúng lớp y_i. Xác suất này là 1 hàm số theo biến w.

Lost function

Giả sử rằng các điểm dữ liệu được sinh ra một cách ngẫu nhiên độc lập với nhau, khi đó

$$p(y|X;w) = \prod_{i=1}^{N} p(y_i|x_i;w) = \prod_{i=1}^{N} z_i^{y_i}(1-z_i)^{1-y_i}.$$

Hàm số sau được xem là hàm lost của mô hình và ta cần tìm w sao cho hàm lost này bé nhất.

$$J(w) = -\frac{1}{N}\log p(y|X;w) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log z_i + (1-y_i)\log(1-z_i)).$$

Softmax Regression

- Để giải quyết bài toán phân lớp có nhiều hơn 2 lớp một cách hiệu quả, ta nghiên cứu một phương pháp mở rộng của logistic regression gọi là softmax regression.
- Khi đó logistic regression chỉ còn là 1 trường hợp riêng của softmax khi số lớp cần phân loại là 2.
- Sau đây ta xét bài toán phân loại có C lớp. ($C \geq 2$)

Linear assumption

- Xét input $x = (x_1, \ldots, x_n)$, tham số của mô hình $w = (w_1, \ldots, w_n)$. Giả định tuyến tính (linear assumption) giả định rằng mối quan hệ giữa các thành phần khác nhau trong 1 điểm data là tuyến tính.
- Cụ thể, ta sẽ đưa giá trị $z = w^T x = w_1 x_1 + \ldots + w_n x_n$ vào mô hình.

Chúng ta cần 1 mô hình sao cho với mỗi input x, ta tính được giá trị a_i là xác xuất để x rơi vào lớp thứ i. Như vậy ta cần tìm 1 hàm $f()$ để tính a_i từ $z_i = w_i^T x$ sao cho:

- a_i là các số dương nằm giữa 0 và 1.
- Tổng tất cả các giá trị a_i phải bằng 1.
- Hàm f có đạo hàm. (để có thể dùng gradient descend)

Softmax function

Ta sử dụng hàm nhiều biến sau đây (softmax function) để tính a_i.

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

Ta xem xác suất để input thứ k, x_k, rơi vào lớp thứ i là

$$p(y_k = i | x_k; W) = a_i$$

trong đó W là ma trận tham số của mô hình, hiểu nôm na là 1 tập hợp các tham số cần tìm để xác định được mô hình.

One hot encoding

Ta ôn lại phương pháp one hot encoding.

- Thuật toán Machine Learning chỉ làm việc trên các con số và các vector. Để nói với thuật toán về con mèo, ta cần biểu diễn con mèo bằng 1 con số hay 1 vector nào đó.
- Xét 1 ví dụ. Ta cần xây dựng 1 model nhận diện 3 chữ cái "A", "B" và "C". Một cách đơn giản, ta có thể đặt 1="A", 2="B" và 3="C". Tuy nhiên, chúng ta có đang muốn nói với model rằng sự khác biệt giữa "A" và "C" (3-1=2) lớn hơn sự khác biệt giữa "A" và "B" (2-1=1)?

One hot encoding

Phương pháp one hot encoding đề xuất cách làm như sau.

- Gán "A" với vector $(1,0,0)$.
- Gán "B" với vector $(0,1,0)$.
- Gán "C" với vector $(0,0,1)$.

One hot encoding là 1 ứng dụng đơn giản nhưng rất hiệu quả của vector trong Machine Learning.

Outcome of the model

- Bản thân các công thức toán không trả lời input x rơi vào lớp nào. Nó chỉ cho biết xác suất để x rơi vào lớp đấy. Cụ thể, **đầu ra dự đoán** của mô hình là 1 vector $a = (a_1, \ldots, a_C)$.
- Trong đó a_i là xác suất để input x rơi vào lớp i. Số C là tổng số lớp ta đang xét.
- Tổng $a_1 + \ldots + a_C = 1$. (tổng của xác suất của tất cả khả năng có thể xảy ra bằng 1)

True label via one hot encoding

- Trong bài toán Machine Learning, ta dùng one hot encoding để mô tả data dạng category. Cụ thể, ta mô tả lớp thứ i bằng vector $y = (0, \ldots, 0, 1, 0 \ldots, 0)$ (vị trí thứ i bằng 1, còn lại bằng 0).
- Giả sử x thực sự nằm trong lớp thứ i. Vector y ở trên được gọi là **đầu ra thực sự** (true label) của x.

Cross entropy

- Như vậy ta cần so sánh đầu ra dự đoán a và đầu ra thực sự y. Ta muốn 2 vector này gần nhau nhất có thể.
- Ta dùng hàm sau đây (cross entropy) để đo khoảng cách giữa 2 vector $y = (y_1, \ldots, y_C)$ và $a = (a_1, \ldots, a_C)$

$$H(y, a) = -\sum_{i=1}^{C} y_i log(a_i).$$

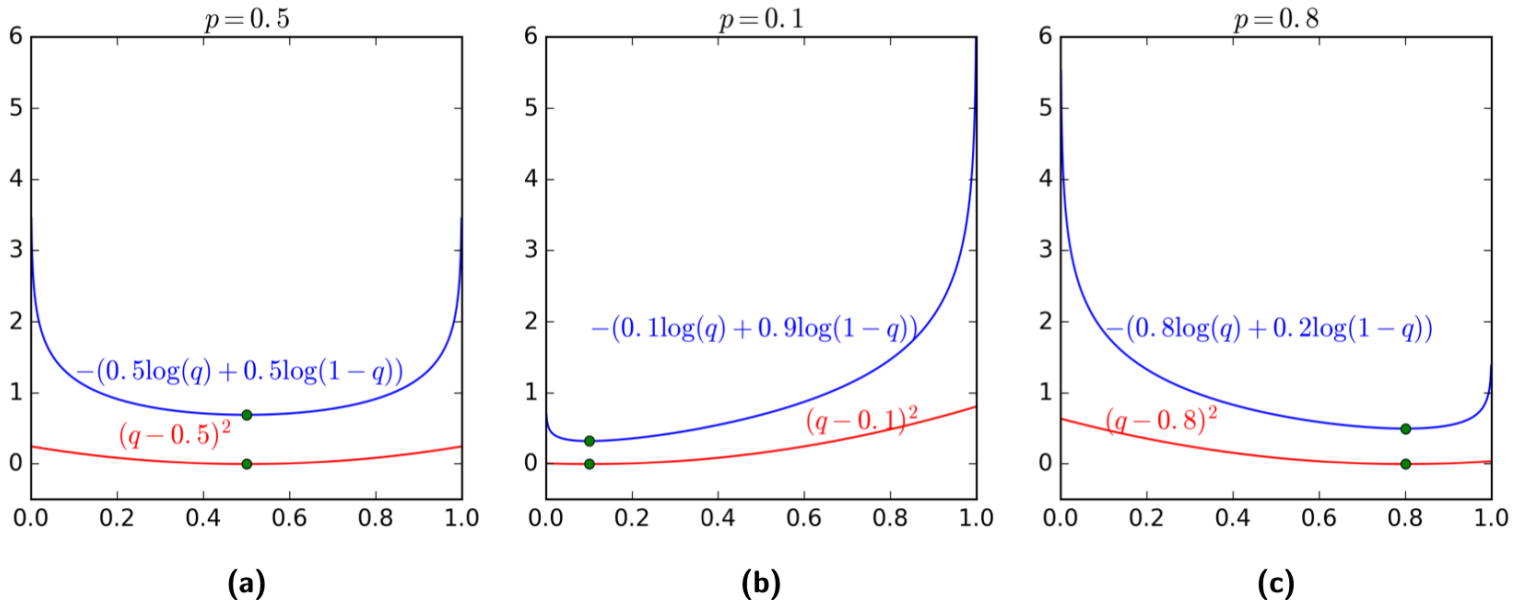
- Trong bài toán của chúng ta, các giá trị $a_i < 1$ nên $log(a_i) < 0$. Dấu trừ trong công thức trên để hàm H nhận giá trị dương.

Cross entropy

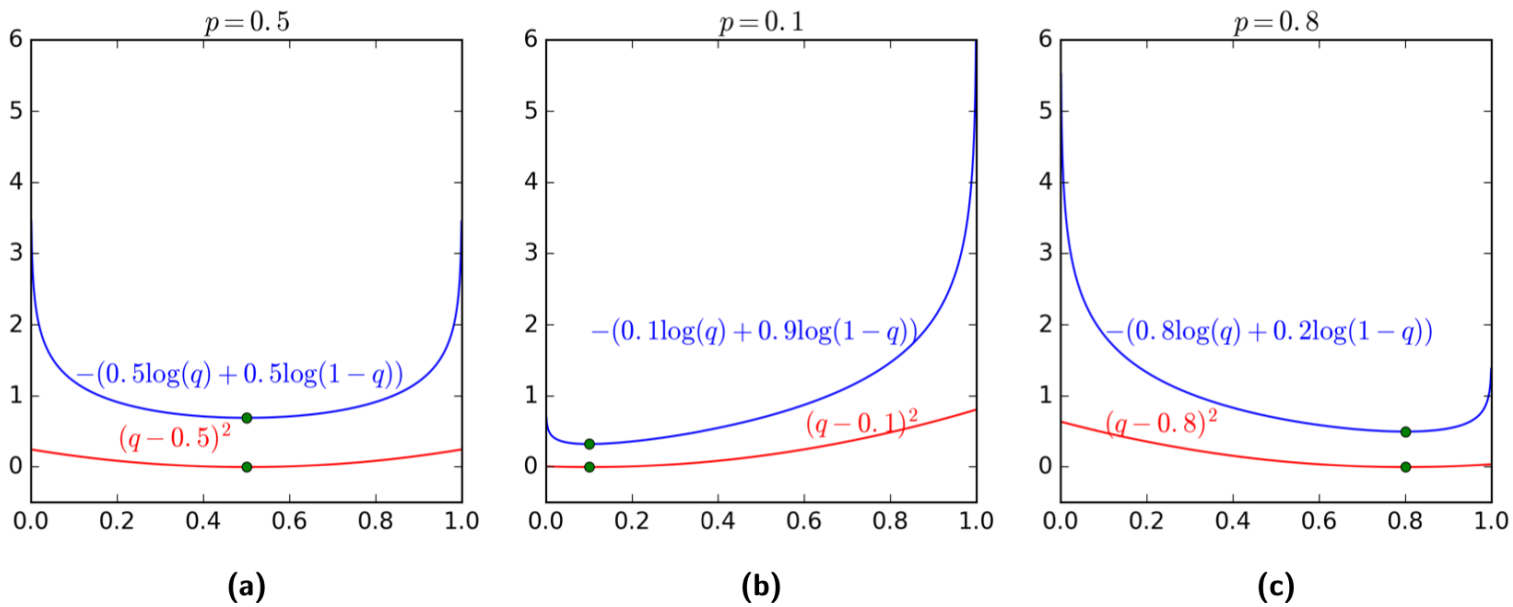
Cố định thành phần thứ nhất y, ta xem $H(y, a)$ là 1 hàm số theo biến a. Hàm cross entropy $H(y, a)$ nói trên có 2 tính chất rất quan trọng, phù hợp với bài toán chúng ta đang xét.

- Giá trị nhỏ nhất của H đạt được khi $a = y$.
- Giá trị của H tăng rất nhanh khi y và a bắt đầu "xa nhau". Đây là lí do quan trọng nhất khiến chúng ta chọn hàm cross entropy chứ không phải metric Euclide thông thường để đo khoảng cách giữa 2 vector y và a.

Ta xét 1 trường hợp đơn giản, khi $C = 2$, để mình hoạ tính chất của hàm cross entropy nói trên. Đường màu xanh là đồ thị của hàm cross entropy. Đường màu đỏ là đồ thị của hàm metric Euclide thông thường. Ta thấy, so với metric Euclide, hàm cross entropy tăng vọt khi q đi xa khỏi p.



Nói cách khác, nếu xem giá trị của hàm loss là mức độ trừng phạt cho việc y và a "xa nhau", so với metric Euclide, hàm cross entropy phạt rất nặng các giá trị q ở xa p. Điều này giúp việc giải bài toán tối ưu hàm loss trở nên dễ dàng hơn.



Loss function

Xét 1 điểm dữ liệu x_i, mất mát giữa đầu ra dự đoán và đầu ra thực sự (true label) y^i của x_i được tính như sau.

$$J^i(W) = -\sum_{j=1}^{C} y_j^i log(a_j^i).$$

Trong đó, a_j^i là xác suất để x_i rơi vào lớp thứ j. y_j^i là phần tử thứ j của vector true label y^i.

Interprettable machine learning ~ ML, không phải deep learning
Explainable AI ~ deep learning

Xin cảm ơn sự chú ý theo dõi!