« Bài 8: Gradient Descent (phần 2/2) (/2017/01/16/gradientdescent2/)
Bài 10: Logistic Regression » (/2017/01/27/logisticregression/)

Bài 9: Perceptron Learning Algorithm

Neural-nets (/tags#Neural-nets) Supervised-learning (/tags#Supervised-learning) Classification (/tags#Classification) Linear-models (/tags#Linear-models) GD (/tags#GD)

Jan 21, 2017

Cứ làm đi, sai đâu sửa đấy, cuối cùng sẽ thành công!

Đó chính là ý tưởng chính của một thuật toán rất quan trọng trong Machine Learning - thuật toán Perceptron Learning Algorithm hay PLA.

Trong trang này:

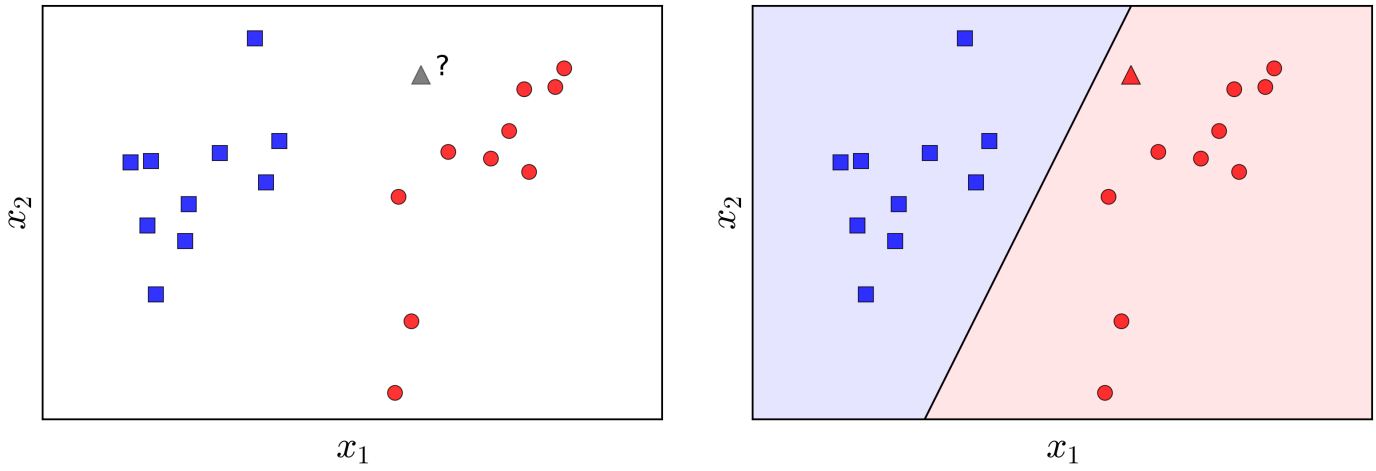
1. Giới thiệu

Trong bài này, tôi sẽ giới thiệu thuật toán đầu tiên trong Classification có tên là Perceptron Learning Algorithm (PLA) hoặc đôi khi được viết gọn là Perceptron.

Perceptron là một thuật toán Classification cho trường hợp đơn giản nhất: chỉ có hai class (lớp) (*bài toán với chỉ hai class được gọi là binary classification*) và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Tuy nhiên, nó là nền tảng cho một mảng lớn quan trọng của Machine Learning là

Neural Networks và sau này là Deep Learning. (Tại sao lại gọi là Neural Networks - tức mạng dây thần kinh - các bạn sẽ được thấy ở cuối bài).

Giả sử chúng ta có hai tập hợp dữ liệu đã được gán nhãn được minh hoạ trong Hình 1 bên trái dưới đây. Hai class của chúng ta là tập các điểm màu xanh và tập các điểm màu đỏ. Bài toán đặt ra là: từ dữ liệu của hai tập được gán nhãn cho trước, hãy xây dựng một *classifier* (bộ phân lớp) để khi có một điểm dữ liệu hình tam giác màu xám mới, ta có thể dự đoán được màu (nhãn) của nó.



Hình 1: Bài toán Perceptron

Hiểu theo một cách khác, chúng ta cần tìm *lãnh thổ* của mỗi class sao cho, với mỗi một điểm mới, ta chỉ cần xác định xem nó nằm vào lãnh thổ của class nào rồi quyết định nó thuộc class đó. Để tìm *lãnh thổ* của mỗi class, chúng ta cần đi tìm biên giới (boundary) giữa hai *lãnh thổ* này. Vậy bài toán classification có thể coi là bài toán đi tìm boundary giữa các class. Và boundary đơn giản nhất trong không gian hai chiều là một đường thẳng, trong không gian ba chiều là một mặt phẳng, trong không gian nhiều chiều là một siêu mặt phẳng (hyperplane) (tôi gọi chung những boundary này là *đường phẳng*). Những boundary phẳng này được coi là đơn giản vì nó có thể biểu diễn dưới dạng toán học bằng một hàm số đơn giản có dạng tuyến tính, tức linear. Tất nhiên, chúng ta đang giả sử rằng tồn tại một đường phẳng để có thể phân định *lãnh thổ* của hai class. Hình 1 bên phải minh họa một đường thẳng phân chia hai class trong mặt phẳng. Phần có nền màu xanh được coi là *lãnh thổ* của lớp xanh, phần có nên màu đỏ được coi là *lãnh thổ* của lớp đỏ. Trong trường hợp này, điểm dữ liệu mới hình tam giác được phân vào class đỏ.

Bài toán Perceptron

Bài toán Perceptron được phát biểu như sau: *Cho hai class được gán nhãn, hãy tìm một đường phẳng sao cho toàn bộ các điểm thuộc class 1 nằm về 1 phía, toàn bộ các điểm thuộc class 2 nằm về phía còn lại của đường phẳng đó. Với giả định rằng tồn tại một đường phẳng như thế.*

Nếu tồn tại một đường phẳng phân chia hai class thì ta gọi hai class đó là *linearly separable*. Các thuật toán classification tạo ra các boundary là các đường phẳng được gọi chung là Linear Classifier.

2. Thuật toán Perceptron (PLA)

Cũng giống như các thuật toán lặp trong K-means Clustering (/2017/01/01/kmeans/) và Gradient Descent (/2017/01/12/gradientdescent/), ý tưởng cơ bản của PLA là xuất phát từ một nghiệm dự đoán nào đó, qua mỗi vòng lặp, nghiệm sẽ được cập nhật tới một ví trí tốt hơn. Việc cập nhật này dựa trên việc giảm giá trị của một hàm mất mát nào đó.

Một số ký hiệu

Giả sử $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ là ma trận chứa các điểm dữ liệu mà mỗi cột $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$ là một điểm dữ liệu trong không gian d chiều. (*Chú ý: khác với các bài trước tôi thường dùng các vector hàng để mô tả dữ liệu, trong bài này tôi dùng vector cột để biểu diễn. Việc biểu diễn dữ liệu ở dạng hàng hay cột tùy thuộc vào từng bài toán, miễn sao cách biểu diễn toán học của nó khiến cho người đọc thấy dễ hiểu*).

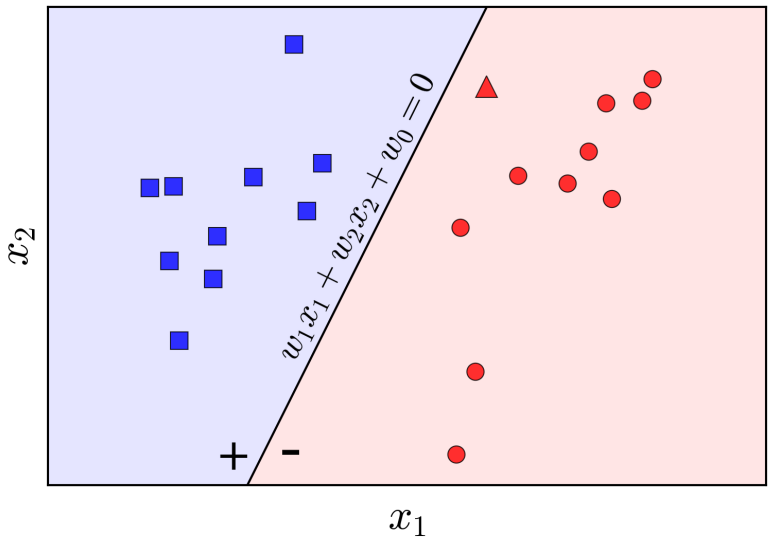
Giả sử thêm các nhãn tương ứng với từng điểm dữ liệu được lưu trong một vector hàng $\mathbf{y} = [y_1, y_2, \ldots, y_N] \in \mathbb{R}^{1 \times N}$, với $y_i = 1$ nếu $\mathbf{x}_i$ thuộc class 1 (xanh) và $y_i = -1$ nếu $\mathbf{x}_i$ thuộc class 2 (đỏ).

Tại một thời điểm, giả sử ta tìm được boundary là đường phẳng có phương trình:

$$\begin{aligned} f_{\mathbf{w}}(\mathbf{x}) &= w_1 x_1 + \cdots + w_d x_d + w_0 \\ &= \mathbf{w}^T \bar{\mathbf{x}} = 0 \end{aligned}$$

với $\bar{\mathbf{x}}$ là điểm dữ liệu mở rộng bằng cách thêm phần tử $x_0 = 1$ lên trước vector $\mathbf{x}$ tương tự như trong Linear Regression (/2016/12/28/linearregression/). Và từ đây, khi nói $\mathbf{x}$, tôi cũng ngầm hiểu là điểm dữ liệu mở rộng.

Để cho đơn giản, chúng ta hãy cùng làm việc với trường hợp mỗi điểm dữ liệu có số chiều $d = 2$. Giả sử đường thẳng $w_1 x_1 + w_2 x_2 + w_0 = 0$ chính là nghiệm cần tìm như Hình 2 dưới đây:



Hình 2: Phương trình đường thẳng boundary.

Nhận xét rằng các điểm nằm về cùng 1 phía so với đường thẳng này sẽ làm cho hàm số $f_{\mathbf{w}}(\mathbf{x})$ mang cùng dấu. Chỉ cần đổi dấu của $\mathbf{w}$ nếu cần thiết, ta có thể giả sử các điểm nằm trong nửa mặt phẳng nền xanh mang dấu dương (+), các điểm nằm trong nửa mặt phẳng nền đỏ mang dấu âm (-). Các dấu này cũng tương đương với nhãn y của mỗi class. Vậy nếu $\mathbf{w}$ là một nghiệm của bài toán Perceptron, với một điểm dữ liệu mới $\mathbf{x}$ chưa được gán nhãn, ta có thể xác định class của nó bằng phép toán đơn giản như sau:

$$\text{label}(\mathbf{x}) = 1 \text{ if } \mathbf{w}^T\mathbf{x} \geq 0, \text{otherwise} -1$$

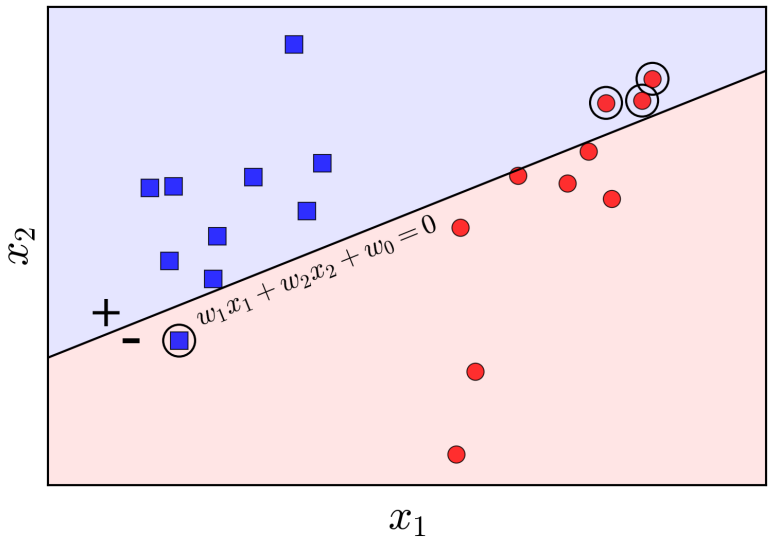
Ngắn gọn hơn:

$$\text{label}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x})$$

trong đó, sgn là hàm xác định dấu, với giả sử rằng $\text{sgn}(0) = 1$.

Xây dựng hàm mất mát

Tiếp theo, chúng ta cần xây dựng hàm mất mát với tham số $\mathbf{w}$ bất kỳ. Vẫn trong không gian hai chiều, giả sử đường thẳng $w_1x_1 + w_2x_2 + w_0 = 0$ được cho như Hình 3 dưới đây:



Hình 3: Đường thẳng bất kỳ và các điểm bị misclassified được khoanh tròn.

Trong trường hợp này, các điểm được khoanh tròn là các điểm bị misclassified (phân lớp lỗi). Điều chúng ta mong muốn là không có điểm nào bị misclassified. Hàm mất mát đơn giản nhất chúng ta nghĩ đến là hàm *đếm* số lượng các điểm bị misclassied và tìm cách tối thiểu hàm số này:

$$J_1(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \text{sgn}(\mathbf{w}^T\mathbf{x_i}))$$

trong đó $\mathcal{M}$ là tập hợp các điểm bị misclassifed (*tập hợp này thay đổi theo* $\mathbf{w}$). Với mỗi điểm $\mathbf{x}_i \in \mathcal{M}$, vì điểm này bị misclassified nên y_i và $\text{sgn}(\mathbf{w}^T\mathbf{x})$ khác nhau, và vì thế $-y_i\text{sgn}(\mathbf{w}^T\mathbf{x_i}) = 1$. Vậy $J_1(\mathbf{w})$ chính là hàm *đếm* số lượng các điểm bị misclassified. Khi hàm số

này đạt giá trị nhỏ nhất bằng 0 thì ta không còn điểm nào bị misclassified.

Một điểm quan trọng, hàm số này là rời rạc, không tính được đạo hàm theo $\mathbf{w}$ nên rất khó tối ưu. Chúng ta cần tìm một hàm mất mát khác để việc tối ưu khả thi hơn.

Xét hàm mất mát sau đây:

$$J(\mathbf{w}) = \sum_{\mathbf{x}_i \in \mathcal{M}} (-y_i \mathbf{w}^T \mathbf{x_i})$$

Hàm $J()$ khác một chút với hàm $J_1()$ ở việc bỏ đi hàm sgn. Nhận xét rằng khi một điểm misclassified $\mathbf{x}_i$ nằm càng xa boundary thì giá trị $-y_i \mathbf{w}^T \mathbf{x}_i$ sẽ càng lớn, nghĩa là sự sai lệch càng lớn. Giá trị nhỏ nhất của hàm mất mát này cũng bằng 0 nếu không có điểm nào bị misclassifed. Hàm mất mát này cũng được cho là tốt hơn hàm $J_1()$ vì nó *trừng phạt* rất nặng những điểm *lấn sâu sang lãnh thổ của class kia*. Trong khi đó, $J_1()$ *trừng phạt* các điểm misclassified như nhau (đều = 1), bất kể chúng xa hay gần với đường biên giới.

Tại một thời điểm, nếu chúng ta chỉ quan tâm tới các điểm bị misclassified thì hàm số $J(\mathbf{w})$ khả vi (tính được đạo hàm), vậy chúng ta có thể sử dụng Gradient Descent (/2017/01/12/gradientdescent/) hoặc Stochastic Gradient Descent (SGD) (/2017/01/16/gradientdescent2/#-stochastic-gradient-descent) để tối ưu hàm mất mát này. Với ưu điểm của SGD cho các bài toán large-scale (/2017/01/12/gradientdescent/#large-scale), chúng ta sẽ làm theo thuật toán này.

Với *một* điểm dữ liệu $\mathbf{x}_i$ bị misclassified, hàm mất mát trở thành:

$$J(\mathbf{w}; \mathbf{x}_i; y_i) = -y_i \mathbf{w}^T \mathbf{x}_i$$

Đạo hàm tương ứng:

$$\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{x}_i; y_i) = -y_i \mathbf{x}_i$$

Vậy quy tắc cập nhật là:

$$\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$$

với η là learning rate được chọn bằng 1. Ta có một quy tắc cập nhật rất gọn là: $\mathbf{w}_{t+1} = \mathbf{w}_t + y_i \mathbf{x}_i$. Nói cách khác, với mỗi điểm $\mathbf{x}_i$ bị misclassifed, ta chỉ cần nhân điểm đó với nhãn y_i của nó, lấy kết quả cộng vào $\mathbf{w}$ ta sẽ được $\mathbf{w}$ mới.

Ta có một quan sát nhỏ ở đây:

$$\begin{aligned}\mathbf{w}_{t+1}^T \mathbf{x}_i &= (\mathbf{w}_t + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= \mathbf{w}_t^T \mathbf{x}_i + y_i ||\mathbf{x}_i||_2^2\end{aligned}$$

Nếu $y_i = 1$, vì $\mathbf{x}_i$ bị misclassifed nên $\mathbf{w}_t^T\mathbf{x}_i < 0$. Cũng vì $y_i = 1$ nên $y_i||\mathbf{x}_i||_2^2 = ||\mathbf{x}_i||_2^2 \geq 1$ (chú ý $x_0 = 1$), nghĩa là $\mathbf{w}_{t+1}^T\mathbf{x}_i > \mathbf{w}_t^T\mathbf{x}_i$. Lý giải bằng lời, $\mathbf{w}_{t+1}$ tiến về phía làm cho $\mathbf{x}_i$ được phân lớp đúng. Điều tương tự xảy ra nếu $y_i = -1$.

Đến đây, cảm nhận của chúng ta với thuật toán này là: cứ chọn đường boundary đi. Xét từng điểm một, nếu điểm đó bị misclassified thì tiến đường boundary về phía làm cho điểm đó được classifed đúng. Có thể thấy rằng, khi di chuyển đường boundary này, các điểm trước đó được classified đúng có thể lại bị misclassified. Mặc dù vậy, PLA vẫn được đảm bảo sẽ hội tụ sau một số hữu hạn bước (tôi sẽ chứng minh việc này ở phía sau của bài viết). Tức là cuối cùng, ta sẽ tìm được đường phẳng phân chia hai lớp, miễn là hai lớp đó là linearly separable. Đây cũng chính là lý do câu đầu tiên trong bài này tôi nói với các bạn là: “Cứ làm đi, sai đâu sửa đấy, cuối cùng sẽ thành công!”.

Tóm lại, thuật toán Perceptron có thể được viết như sau:

Tóm tắt PLA

1. Chọn ngẫu nhiên một vector hệ số $\mathbf{w}$ với các phần tử gần 0.
2. Duyệt ngẫu nhiên qua từng điểm dữ liệu $\mathbf{x}_i$:
    - Nếu $\mathbf{x}_i$ được phân lớp đúng, tức $\text{sgn}(\mathbf{w}^T\mathbf{x}_i) = y_i$, chúng ta không cần làm gì.
    - Nếu $\mathbf{x}_i$ bị misclassifed, cập nhật $\mathbf{w}$ theo công thức:

$$\mathbf{w} = \mathbf{w} + y_i\mathbf{x}_i$$

3. Kiểm tra xem có bao nhiêu điểm bị misclassifed. Nếu không còn điểm nào, dừng thuật toán. Nếu còn, quay lại bước 2.

3. Ví dụ trên Python

Như thường lệ, chúng ta sẽ thử một ví dụ nhỏ với Python.

Load thư viện và tạo dữ liệu

Chúng ta sẽ tạo hai nhóm dữ liệu, mỗi nhóm có 10 điểm, mỗi điểm dữ liệu có hai chiều để thuận tiện cho việc minh họa. Sau đó, tạo dữ liệu mở rộng bằng cách thêm 1 vào đầu mỗi điểm dữ liệu.

```
# generate data
# list of points
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(2)

means = [[2, 2], [4, 2]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N).T
X1 = np.random.multivariate_normal(means[1], cov, N).T

X = np.concatenate((X0, X1), axis = 1)
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1)
# Xbar
X = np.concatenate((np.ones((1, 2*N)), X), axis = 0)
```

Sau khi thực hiện đoạn code này, biến `X` sẽ chứa dữ liệu input (mở rộng), biến `y` sẽ chứa nhãn của mỗi điểm dữ liệu trong `X`.

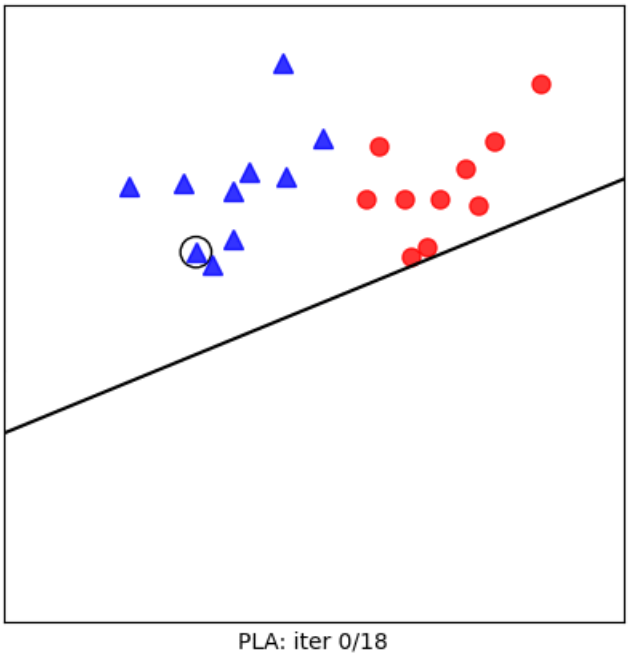
Các hàm số cho PLA

Tiếp theo chúng ta cần viết 3 hàm số cho PLA:

1. `h(w, x)`: tính đầu ra khi biết đầu vào `x` và weights `w`.
2. `has_converged(X, y, w)`: kiểm tra xem thuật toán đã hội tụ chưa. Ta chỉ cần so sánh `h(w, X)` với *ground truth* `y`. Nếu giống nhau thì dừng thuật toán.
3. `perceptron(X, y, w_init)`: hàm chính thực hiện PLA.

```
def h(w, x):
    return np.sign(np.dot(w.T, x))

def has_converged(X, y, w):
    return np.array_equal(h(w, X), y)

def perceptron(X, y, w_init):
    w = [w_init]
    N = X.shape[1]
    d = X.shape[0]
    mis_points = []
    while True:
        # mix data
        mix_id = np.random.permutation(N)
        for i in range(N):
            xi = X[:, mix_id[i]].reshape(d, 1)
            yi = y[0, mix_id[i]]
            if h(w[-1], xi)[0] != yi: # misclassified point
                mis_points.append(mix_id[i])
                w_new = w[-1] + yi*xi
                w.append(w_new)

        if has_converged(X, y, w[-1]):
            break
    return (w, mis_points)

d = X.shape[0]
w_init = np.random.randn(d, 1)
(w, m) = perceptron(X, y, w_init)
```

Dưới đây là hình minh họa thuật toán PLA cho bài toán nhỏ này:



Hình 4: Minh họa thuật toán PLA

Sau khi cập nhật 18 lần, PLA đã hội tụ. Điểm được khoanh tròn màu đen là điểm misclassified tương ứng được chọn để cập nhật đường boundary.

Source code cho phần này (bao gồm hình động) có thể được tìm thấy ở đây (https://github.com/tiepvupsu/tiepvupsu.github.io/blob/master/assets/pla/perceptron.py).

4. Chứng minh hội tụ

Giả sử rằng $\mathbf{w}^*$ là một nghiệm của bài toán (ta có thể giả sử việc này được vì chúng ta đã có giả thiết hai class là linearly separable - tức tồn tại nghiệm). Có thể thấy rằng, với mọi $\alpha > 0$, nếu $\mathbf{w}^*$ là nghiệm, $\alpha\mathbf{w}^*$ cũng là nghiệm của bài toán. Xét dãy số không âm $u_\alpha(t) = ||\mathbf{w}_t - \alpha\mathbf{w}^*||_2^2$. Với $\mathbf{x}_i$ là một điểm bị misclassified nếu dùng nghiệm $\mathbf{w}_t$ ta có:

$$
\begin{aligned}
&u_\alpha(t+1) = ||\mathbf{w}_{t+1} - \alpha\mathbf{w}^*||_2^2 \\
&= ||\mathbf{w}_t + y_i\mathbf{x}_i - \alpha\mathbf{w}^*||_2^2 \\
&= ||\mathbf{w}_t - \alpha\mathbf{w}^*||_2^2 + y_i^2||\mathbf{x}_i||_2^2 + 2y_i\mathbf{x}_i^T(\mathbf{w} - \alpha\mathbf{w}^*) \\
&< u_\alpha(t) \ + ||\mathbf{x}_i||_2^2 - 2\alpha y_i\mathbf{x}_i^T\mathbf{w}^*
\end{aligned}
$$

Dấu nhỏ hơn ở dòng cuối là vì $y_i^2 = 1$ và $2y_i\mathbf{x}_i^T\mathbf{w}_t < 0$. Nếu ta đặt:

$$\beta^2 = \max_{i=1,2,\ldots,N} ||\mathbf{x}_i||_2^2$$
$$\gamma = \min_{i=1,2,\ldots,N} y_i \mathbf{x}_i^T \mathbf{w}^*$$

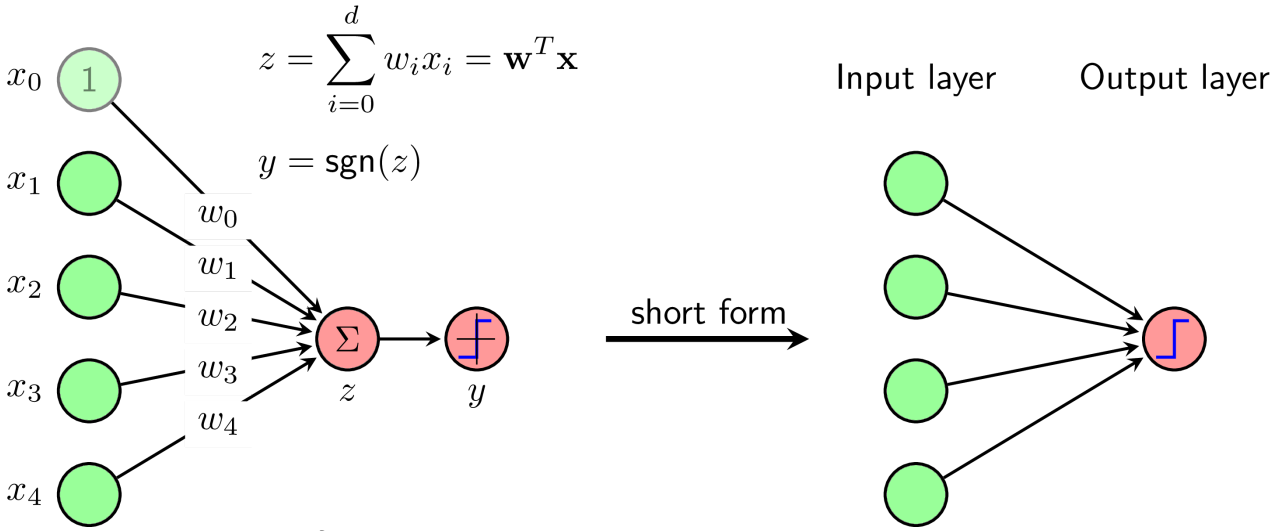
và chọn $\alpha = \frac{\beta^2}{\gamma}$, ta có:

$$0 \leq u_\alpha(t+1) < u_\alpha(t) + \beta^2 - 2\alpha\gamma = u_\alpha(t) - \beta^2$$

Điều này nghĩa là: nếu luôn luôn có các điểm bị misclassified thì dãy $u_\alpha(t)$ là dãy giảm, bị chặn dưới bởi 0, và phần tử sau kém phần tử trước ít nhất một lượng là $\beta^2 > 0$. Điều vô lý này chứng tỏ đến một lúc nào đó sẽ không còn điểm nào bị misclassified. Nói cách khác, thuật toán PLA hội tụ sau một số hữu hạn bước.

5. Mô hình Neural Network đầu tiên

Hàm số xác định class của Perceptron $\text{label}(\mathbf{x}) = \text{sgn}(\mathbf{w}^T\mathbf{x})$ có thể được mô tả như hình vẽ (được gọi là network) dưới đây:



Hình 5: Biểu diễn của Perceptron dưới dạng Neural Network.

Đầu vào của network $\mathbf{x}$ được minh họa bằng các node màu xanh lục với node x_0 luôn luôn bằng 1. Tập hợp các node màu xanh lục được gọi là *Input layer*. Trong ví dụ này, tôi giả sử số chiều của dữ liệu $d = 4$. Số node trong input layer luôn luôn là $d + 1$ với một node là 1 được thêm vào. Node $x_0 = 1$ này đôi khi được ẩn đi.

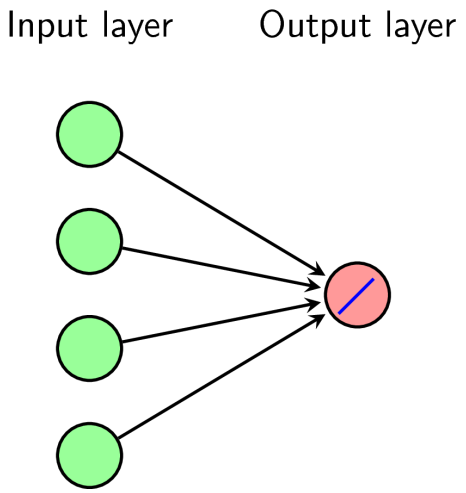
Các trọng số (*weights*) $w_0, w_1, \ldots, w_d$ được gán vào các mũi tên đi tới node $z = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^T\mathbf{x}$. Node $y = \text{sgn}(z)$ là *output* của network. Ký hiệu hình chữ Z ngược màu xanh trong node y thể hiện đồ thị của hàm số sgn.

Trong thuật toán PLA, ta phải tìm các weights trên các mũi tên sao cho với mỗi $\mathbf{x}_i$ ở tập các điểm dữ liệu đã biết được đặt ở Input layer, output của network này trùng với nhãn y_i tương ứng.

Hàm số $y = \text{sgn}(z)$ còn được gọi là *activation function*. Đây chính là dạng đơn giản nhất của Neural Network.

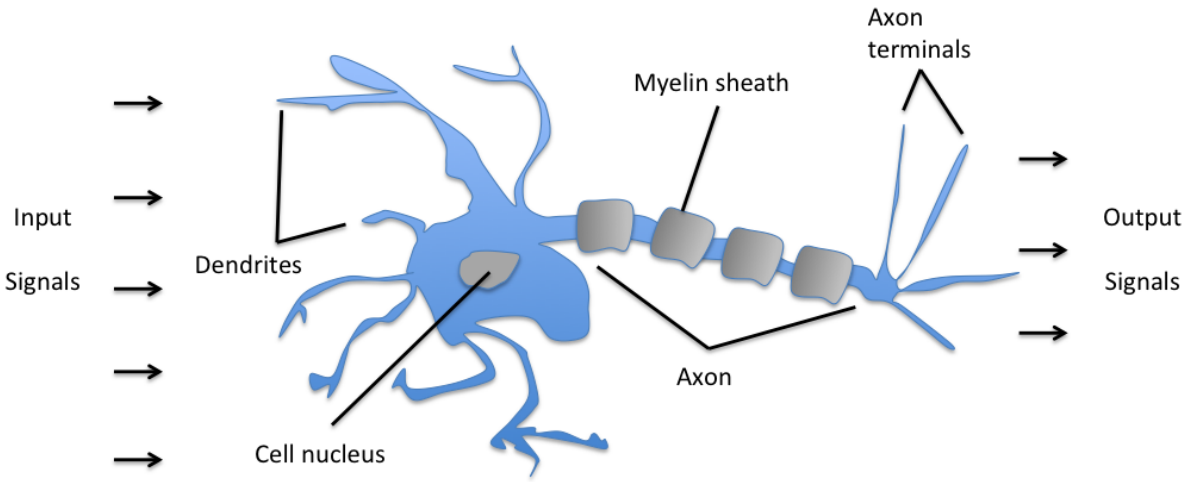
Các Neural Networks sau này có thể có nhiều node ở output tạo thành một *output layer*, hoặc có thể có thêm các layer trung gian giữa *input layer* và *output layer*. Các layer trung gian đó được gọi là *hidden layer*. Khi biểu diễn các Networks lớn, người ta thường giản lược hình bên trái thành hình bên phải. Trong đó node $x_0 = 1$ thường được ẩn đi. Node z cũng được ẩn đi và viết gộp vào trong node y. Perceptron thường được vẽ dưới dạng đơn giản như Hình 5 bên phải.

Để ý rằng nếu ta thay *activation function* bởi $y = z$, ta sẽ có Neural Network mô tả thuật toán Linear Regression như hình dưới. Với đường thẳng chéo màu xanh thể hiện đồ thị hàm số $y = z$. Các trục tọa độ đã được lược bỏ.



Hình 6: Biểu diễn của Linear Regression dưới dạng Neural Network.

Mô hình perceptron ở trên khá giống với một node nhỏ của dây thần kinh sinh học như hình sau đây:



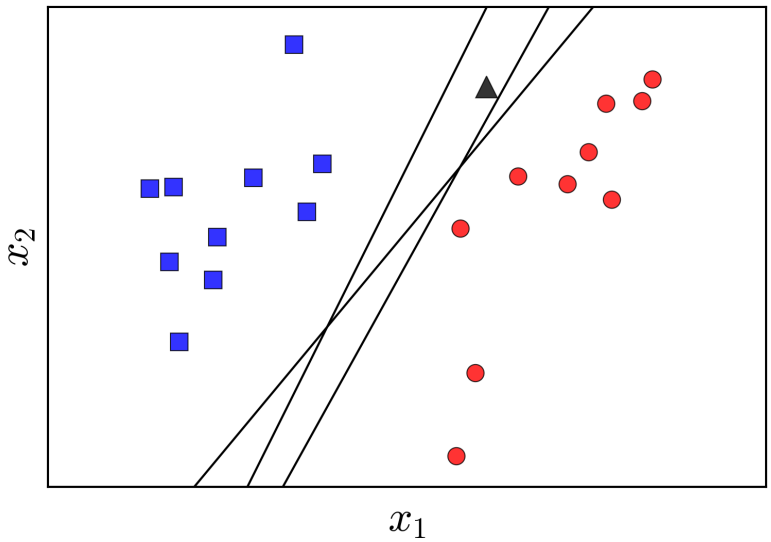
Hình 7: Mô tả một neuron thần kinh sinh học. (Nguồn: Single-Layer Neural Networks and Gradient Descent (http://sebastianraschka.com/Articles/2015_singlelayer_neurons.html))

Dữ liệu từ nhiều dây thần kinh đi về một *cell nucleus*. Thông tin được tổng hợp và được đưa ra ở output. Nhiều bộ phận như thế này kết hợp với nhau tạo nên hệ thần kinh sinh học. Chính vì vậy mà có tên Neural Networks trong Machine Learning. Đôi khi mạng này còn được gọi là Artificial Neural Networks (ANN) tức *hệ neuron nhân tạo*.

6. Thảo Luận

PLA có thể cho vô số nghiệm khác nhau

Rõ ràng rằng, nếu hai class là linearly separable thì có vô số đường thẳng phân cách 2 class đó. Dưới đây là một ví dụ:

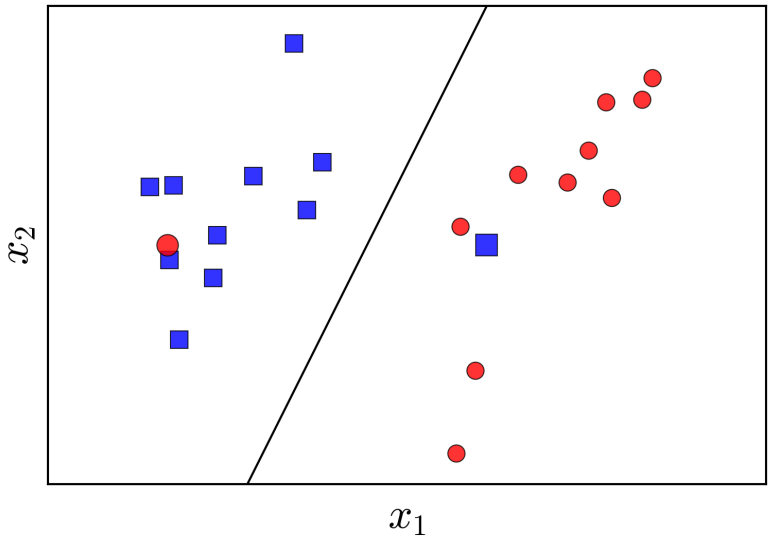


Hình 8: PLA có thể cho vô số nghiệm khác nhau.

Tất cả các đường thẳng màu đen đều thỏa mãn. Tuy nhiên, các đường khác nhau sẽ quyết định điểm hình tam giác thuộc các lớp khác nhau. Trong các đường đó, đường nào là tốt nhất? Và định nghĩa "tốt nhất" được hiểu theo nghĩa nào? Có một thuật toán khác định nghĩa và tìm đường tốt nhất như thế, tôi sẽ giới thiệu trong 1 vài bài tới. Mời các bạn đón đọc.

PLA đòi hỏi dữ liệu linearly separable

Hai class trong ví dụ dưới đây *tương đối* linearly separable. Mỗi class có 1 điểm coi như *nhiễu* nằm trong khu vực các điểm của class kia. PLA sẽ không làm việc trong trường hợp này vì luôn luôn có ít nhất 2 điểm bị misclassified.



Hinhf 9: PLA không làm việc nếu chỉ có một nhiễu nhỏ.

Trong một chừng mực nào đó, đường thẳng màu đen vẫn có thể coi là một nghiệm tốt vì nó đã giúp phân loại chính xác hầu hết các điểm. Việc không hội tụ với dữ liệu *gần* linearly separable chính là một nhược điểm lớn của PLA.

Để khắc phục nhược điểm này, có một cải tiến nhỏ như thuật toán Pocket Algorithm dưới đây:

Pocket Algorithm

Một cách tự nhiên, nếu có một vài *nhiễu*, ta sẽ đi tìm một đường thẳng phân chia hai class sao cho có ít điểm bị misclassified nhất. Việc này có thể được thực hiện thông qua PLA với một chút thay đổi nhỏ như sau:

1. Giới hạn số lượng vòng lặp của PLA.
2. Mỗi lần cập nhật nghiệm $\mathbf{w}$ mới, ta đếm xem có bao nhiêu điểm bị misclassified. Nếu là lần đầu tiên, giữ lại nghiệm này trong *pocket* (túi quần). Nếu không, so sánh số điểm misclassified này với số điểm misclassified của nghiệm trong *pocket*, nếu nhỏ hơn thì *lôi* nghiệm cũ ra, đặt nghiệm mới này vào.

Thuật toán này giống với thuật toán tìm phần tử nhỏ nhất trong 1 mảng.

7. Kết luận

Hy vọng rằng bài viết này sẽ giúp các bạn phần nào hiểu được một số khái niệm trong Neural Networks. Trong một số bài tiếp theo, tôi sẽ tiếp tục nói về các thuật toán cơ bản khác trong Neural Networks trước khi chuyển sang phần khác.

Trong tương lai, nếu có thể, tôi sẽ viết tiếp về Deep Learning và chúng ta sẽ lại quay lại với Neural Networks.

8. Tài liệu tham khảo

[1] F. Rosenblatt. The perceptron, a perceiving and recognizing automaton Project Para. Cornell Aeronautical Laboratory, 1957.

[2] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

[3] B. Widrow et al. Adaptive ”Adaline” neuron using chemical ”memistors”. Number Technical Report 1553-2. Stanford Electron. Labs., Stanford, CA, October 1960.

[3] Abu-Mostafa, Yaser S., Malik Magdon-Ismail, and Hsuan-Tien Lin. Learning from data. Vol. 4. New York, NY, USA:: AMLBook, 2012. (link to course (http://work.caltech.edu/telecourse.html))

[4] Bishop, Christopher M. “Pattern recognition and Machine Learning.”, Springer (2006). (book (http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf))

[5] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern classification. John Wiley & Sons, 2012.

Nếu có câu hỏi, Bạn có thể để lại comment bên dưới hoặc trên Forum (https://www.facebook.com/groups/257768141347267/) để nhận được câu trả lời sớm hơn.
Bạn đọc có thể ủng hộ blog qua 'Buy me a cofee' (/buymeacoffee/) ở góc trên bên trái của blog.
Tôi vừa hoàn thành cuốn ebook 'Machine Learning cơ bản', bạn có thể đặt sách tại đây (/ebook/). Cảm ơn bạn.

« Bài 8: Gradient Descent (phần 2/2) (/2017/01/16/gradientdescent2/)
Bài 10: Logistic Regression » (/2017/01/27/logisticregression/)
