Perceptrons

purpose

- Determine whether an input (vector) belongs to a class
- Binary classification
- Linear
- In modern machine learning, neural networks are composed of perceptrons

Model definition

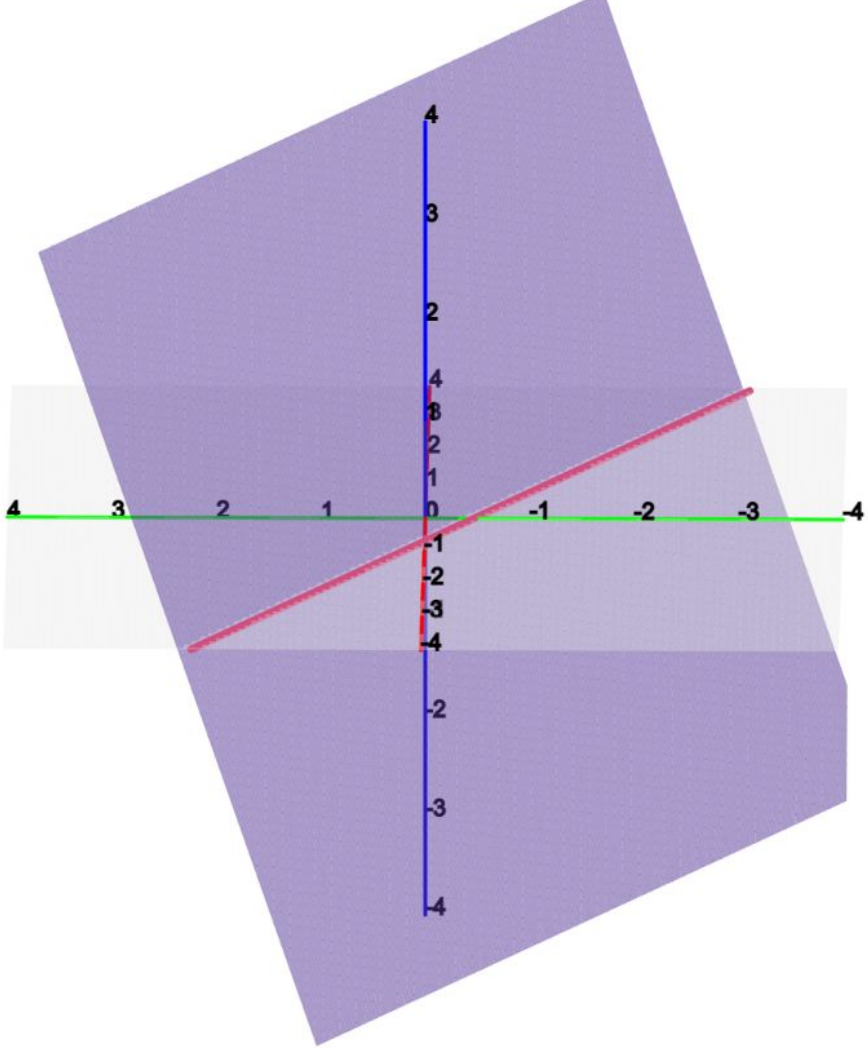
To determine if the model classifies a given input x as 0 or 1, use the following function:

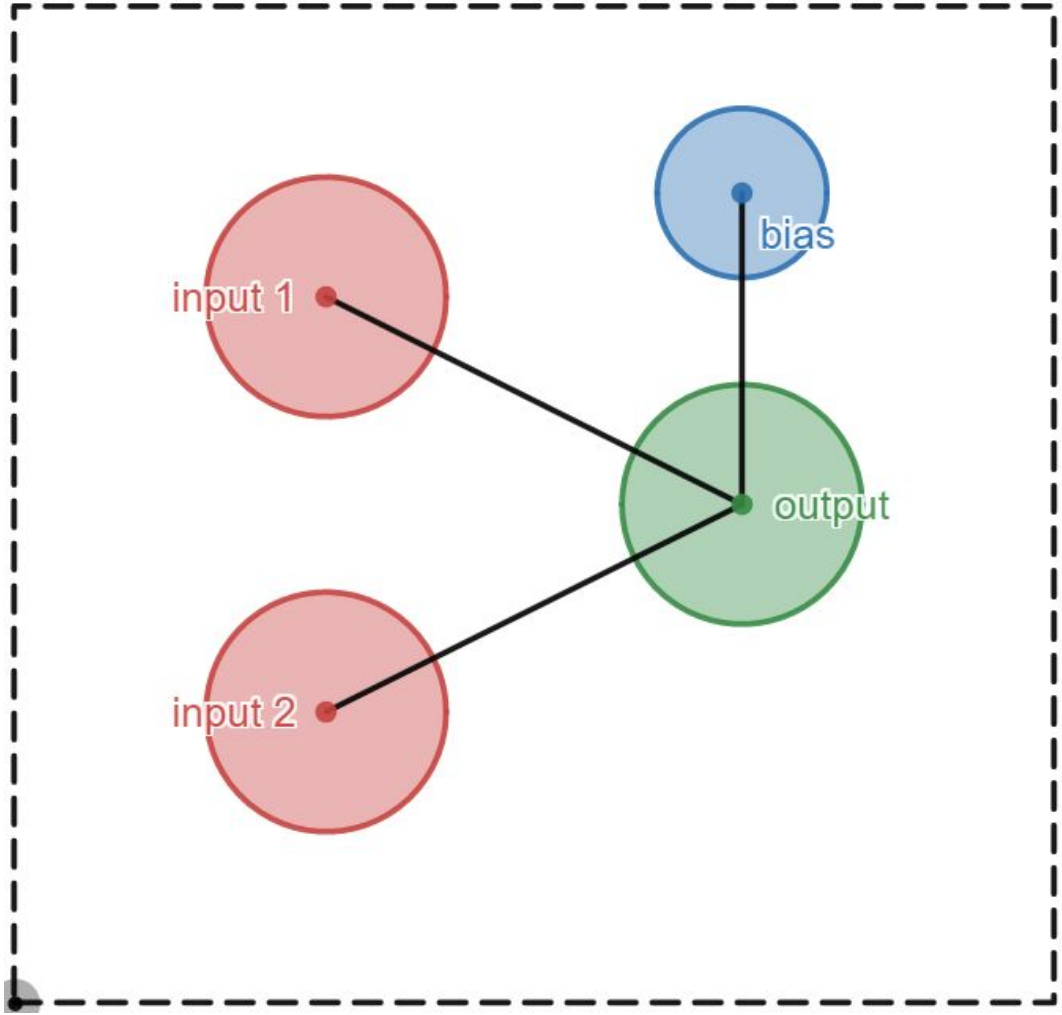
$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Or in this case, simply

$$z(x_1, y_1) = w_1 x_1 + w_2 x_2 + b$$

The classification function defines a plane in space. The plane intersects the xy plane (z=0) at a line, which serves as the “Decision boundary”. The decision boundary is oriented!



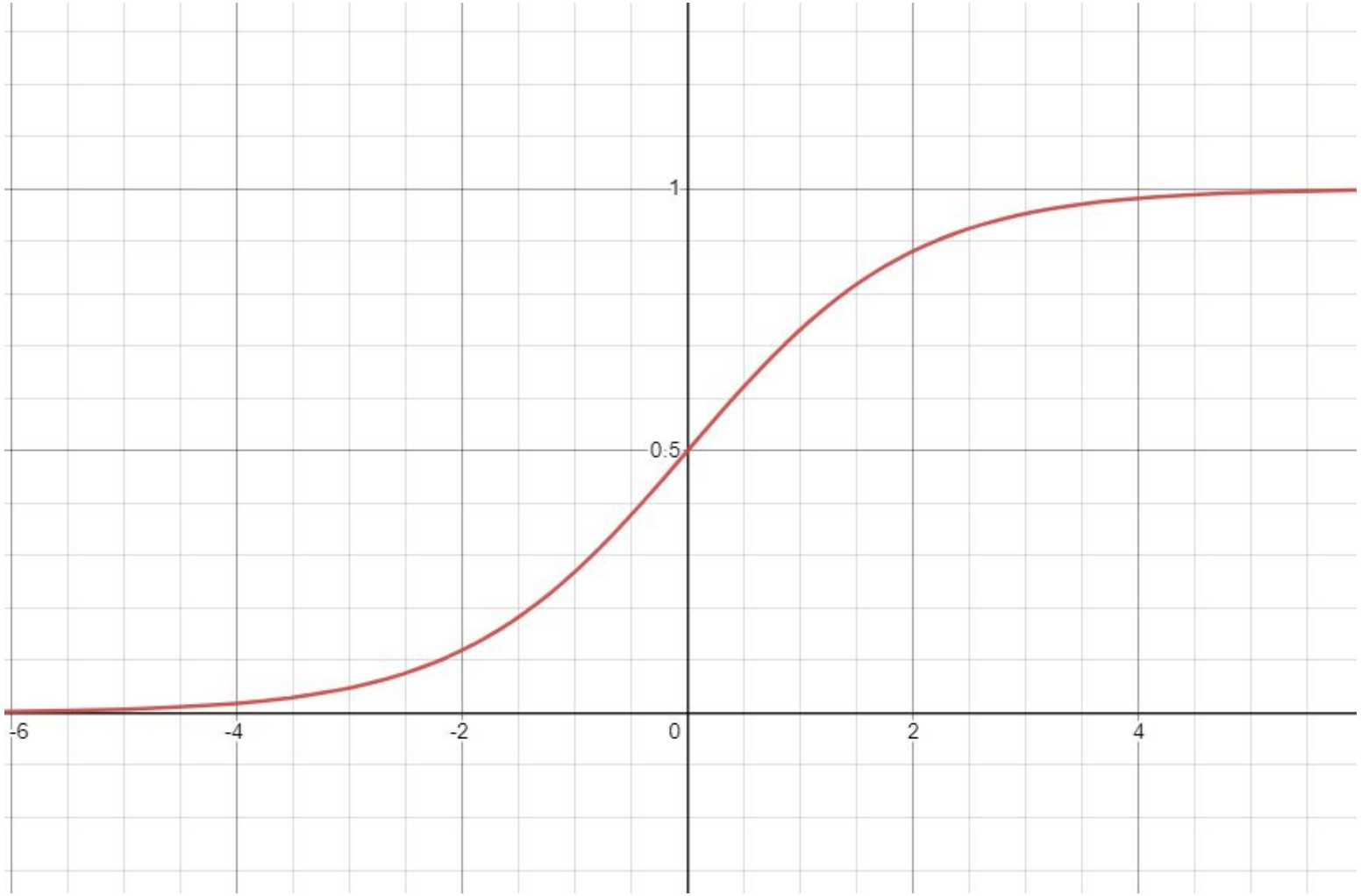

bias
input 1
output
input 2

Probability

So when z is positive, it's classified as 1. Otherwise, it's classified as zero. What if we want a probability (confidence rating) for this classification?

$$a(x) = \frac{1}{1 + \exp(-x)}$$

The greater the magnitude of z (in either direction), the more the confidence increases.


1
0.5
-6
-4
-2
0
2
4

Bernoulli law

So, the probability that an input is a given class is given by

$$P(Y = y) = a(z)^y \cdot (1 - a(z))^{1-y}$$

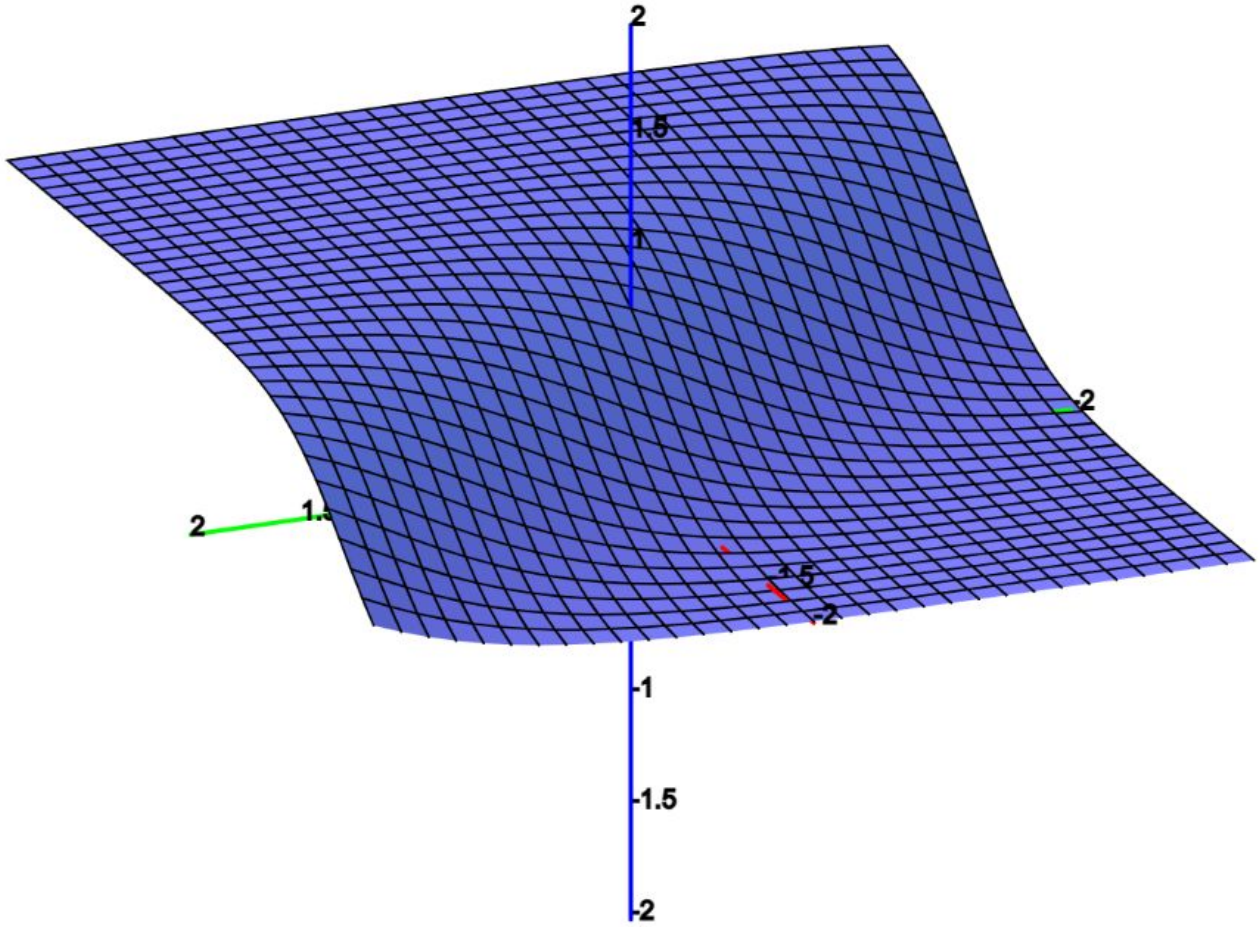
Basically this is a needlessly complicated way of saying P(Y=0) is the complement of P(Y=1).

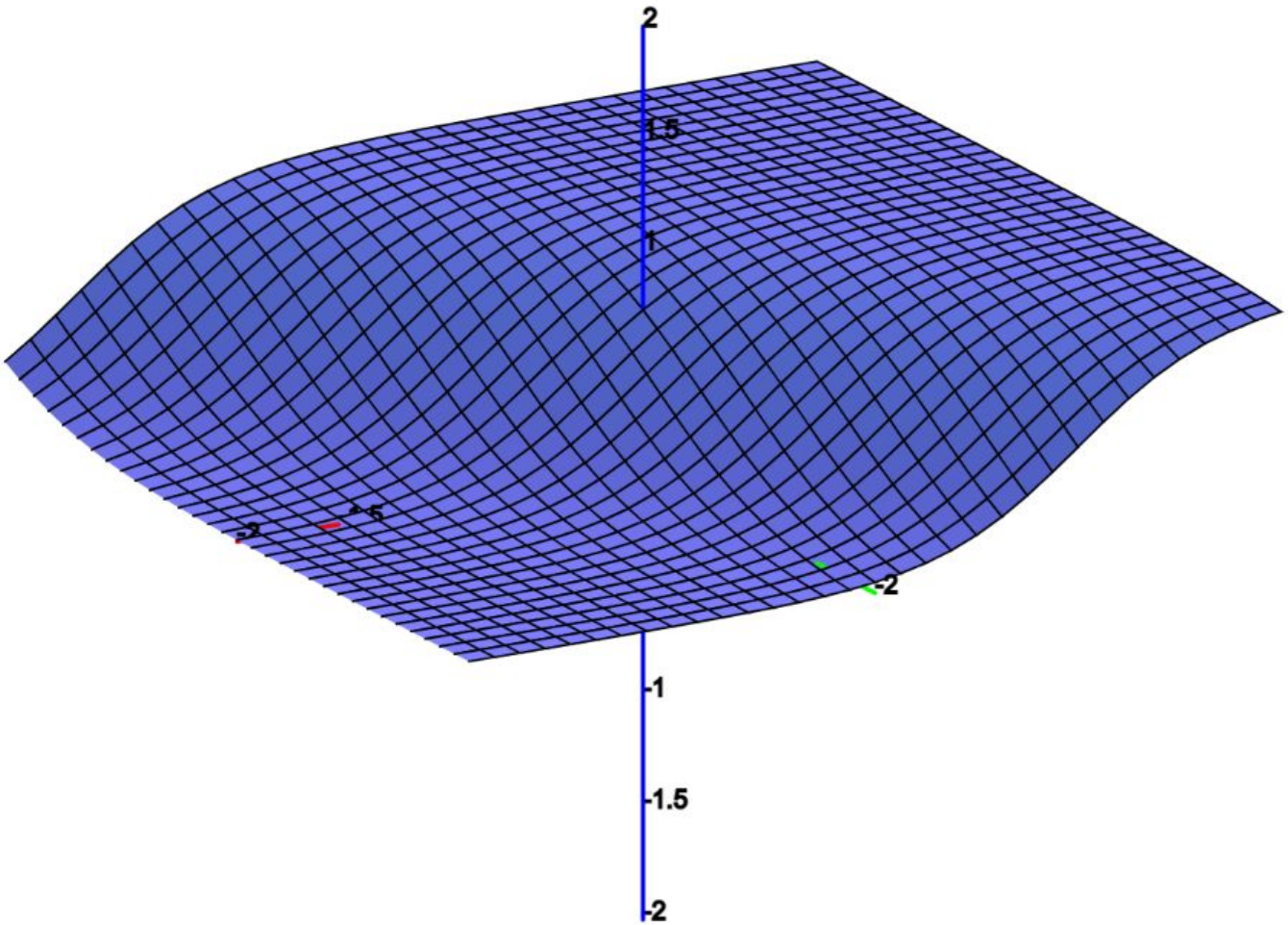
$$P(Y = 0) = 1 - a(z)$$
$$P(Y = 1) = a(z)$$

a(z(x,y))

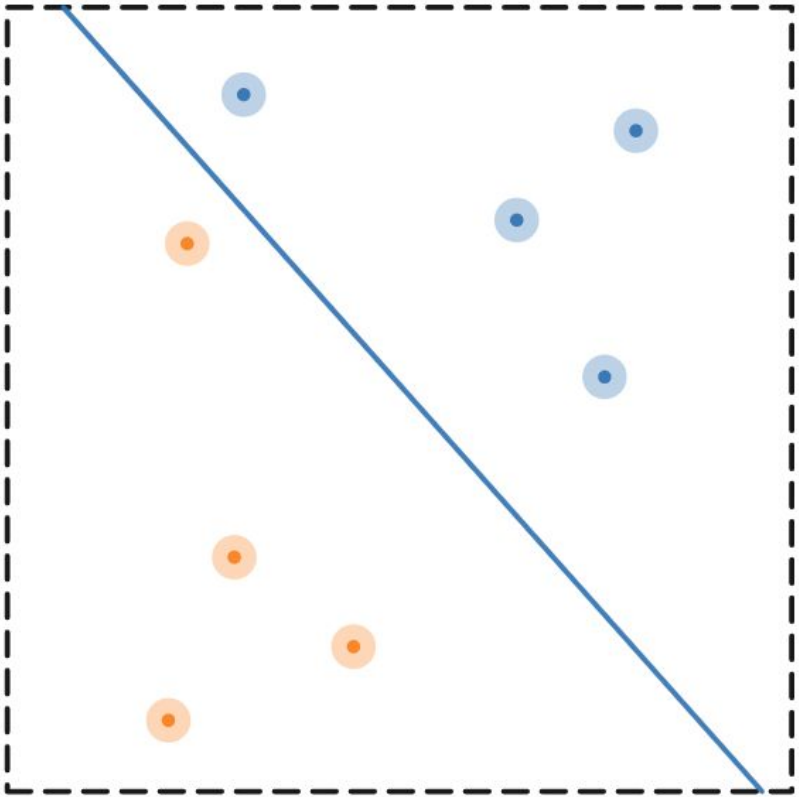
We can apply this probability function to every output of the classification function z(x,y). w1=3, w2=2, and b=1 are example weights.

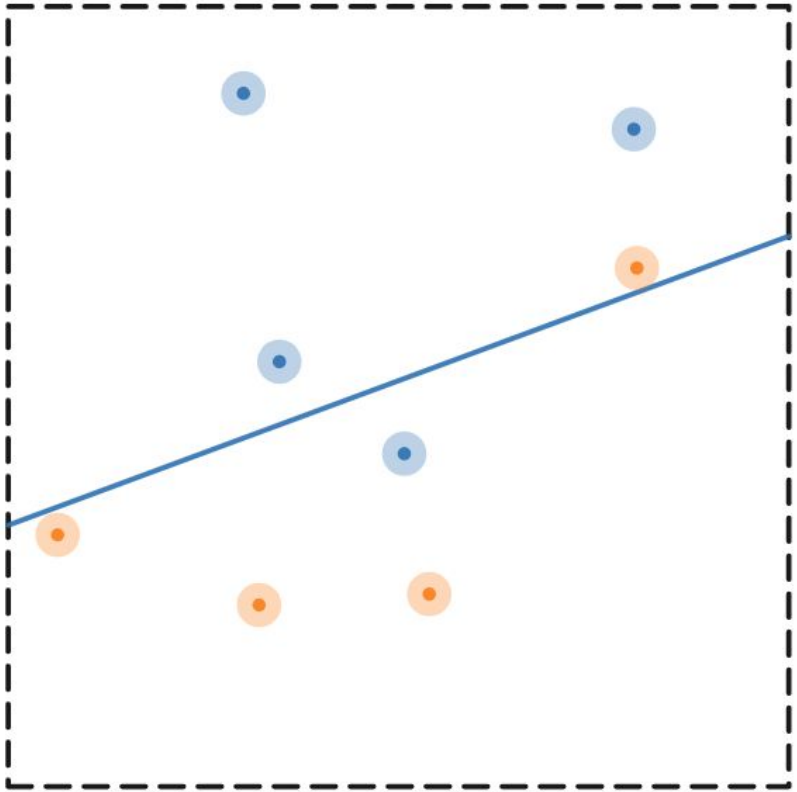
$$\frac{1}{1+\exp(-z(x,y))} = \frac{1}{1+\exp(-(3x+2y+1))}$$


2
1.5
1
2
2
-1.5
-2
-1
-1.5
-2


2
1.5
1
-2
-.5
-2
-1
-1.5
-2

Linear separability





Optimizing weights and bias

What does it even mean for them to be optimal?

Define the following log-loss (cross-entropy) **cost function**:

$$-\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(a(\mathbf{w}\cdot\mathbf{x}+b)) + (1-y_i)\log(1-a(\mathbf{w}\cdot\mathbf{x}+b))\right)$$

In this case:

$$l_{ogloss}(w_{1v}, w_{2v}, b_v) = -\frac{1}{8}\,\text{total}\left(y_5 \ln\left(a\left(z_w(w_{1v}, w_{2v}, b_v, x_4, y_4)\right)\right) + (1-y_5)\ln\left(1-a\left(z_w(w_{1v}, w_{2v}, b_v, x_4, y_4)\right)\right)\right)$$

Motivation for log-loss function

Remember the Bernoulli's law formula. This gives the probability for a certain input. If we want an overall "probability" or verisimilitude of the whole model, we can multiply the probability of all the inputs together:

$$\prod_{i=1}^{N} a(z)^{y_i} \cdot (1 - a(z))^{1-y_i}$$

Motivation for log-loss

However, this function vanishes very quickly to zero, even with high probabilities for each input. So we can take the logarithm of the whole thing, which converts the products into sums. log(x) is monotonically increasing, so this preserves order i.e. argmin(log(loss product)) = argmin(loss product) so the optimal inputs remain unchanged. When simplified, the log of this product (almost) gives the log-loss function shown before. The -1/N term serves to take the average.

$$-\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(a(\mathbf{w}\cdot\mathbf{x}+b)) + (1-y_i)\log(1-a(\mathbf{w}\cdot\mathbf{x}+b))\right)$$

Where z=dot(w,x)+b

Optimizing the weights and biases: gradient descent

The gradient vector of a function at a point gives the direction of steepest ascent i the input space. So the negative of the gradient gives the steepest descent direction. This way, the cost function can be minimized.

$$\nabla L = \begin{pmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \frac{\partial L}{\partial b} \end{pmatrix}$$

Optimizing the weights and biases: gradient descent

So we can update the weights as follows:

$$W_{t+1} = W_t - \alpha \nabla L$$

Where

$$W = \begin{pmatrix} w_1 \\ w_2 \\ b \end{pmatrix}$$

Explicit formulas for these partial derivatives

$$\frac{\partial L}{\partial w_1} = \frac{1}{N}\sum_{i=1}^{N} x_1 \left(a(z) - y_i\right)$$

$$\frac{\partial L}{\partial w_2} = \frac{1}{N}\sum_{i=1}^{N} y_1 \left(a(z) - y_i\right)$$

$$\frac{\partial L}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} \left(a(z) - y_i\right)$$

The derivation of these formulas is straightforward but too involved for the presentation (requires multivariable chain rule). In case anyone is interested, I brought the derivation with on a piece of paper

Desmos graph

https://www.desmos.com/calculator/d5hqcq8dms

thank you for listening!

http://sambrunacini.com/
https://github.com/sam-lb