Deep Learning

Perceptron to sigmoid neuron

Puneet Kumar Jain

CSE Department

National Institute of Technology Rourkela

References:

The Slides are prepared from the following major source:

- **"CS7105-Deep Learning" by Mitesh M. Khapra, IIT Madras.** **http://www.cse.iitm.ac.in/~miteshk/CS7015_2018.html**

Perceptron to sigmoid neuron

Enough about boolean functions!

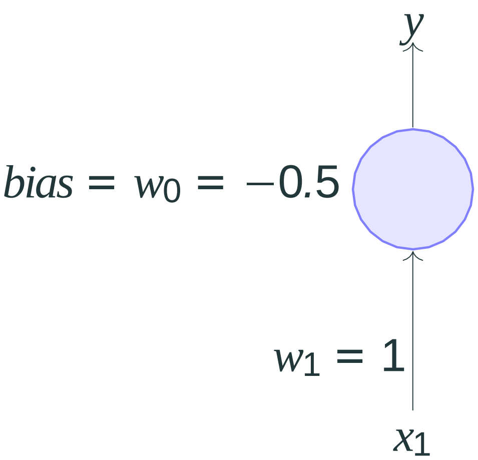
What about arbitrary functions of the form

$y = f(x)$ where $x \in R^n$ (instead of $\{0, 1\}^n$) and

$y \in R$ (instead of $\{0, 1\}$) ?

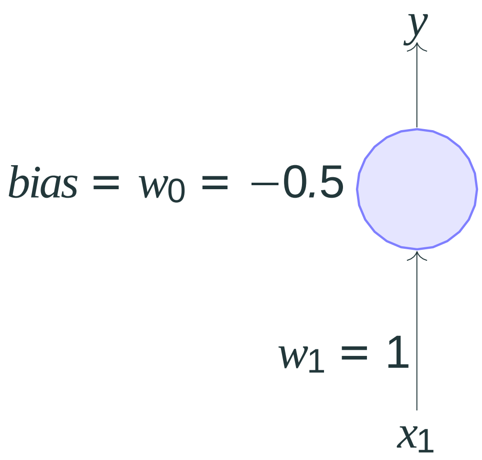
Can we have a network which can (approximately) represent such functions ?

Perceptron



The thresholding logic used by a perceptron is very harsh !

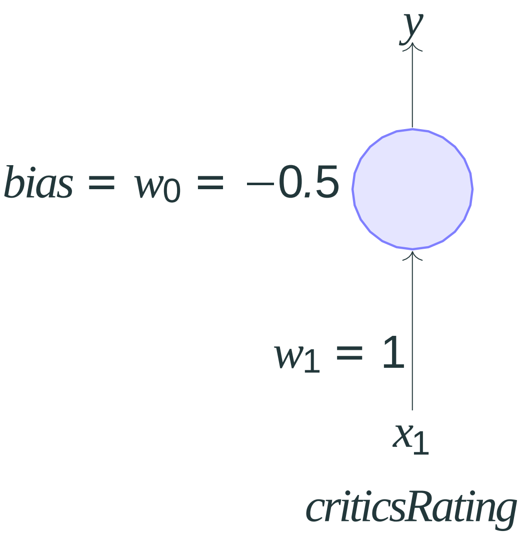
Perceptron



The thresholding logic used by a perceptron is very harsh !

For example, let us return to our problem of deciding whether we will like or dislike a movie

Perceptron

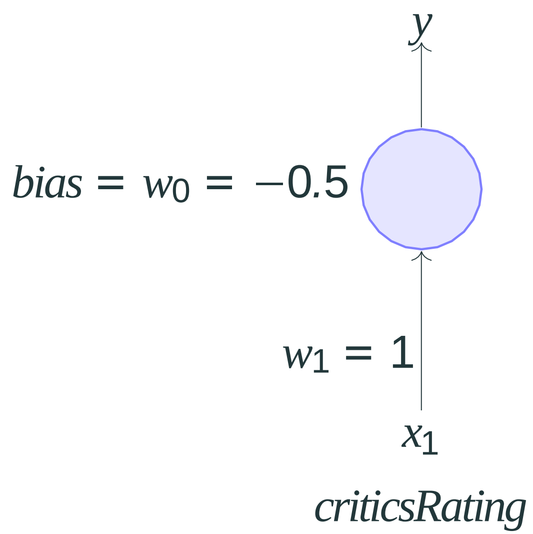


The thresholding logic used by a perceptron is very harsh !

For example, let us return to our problem of deciding whether we will like or dislike a movie

Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)

Perceptron



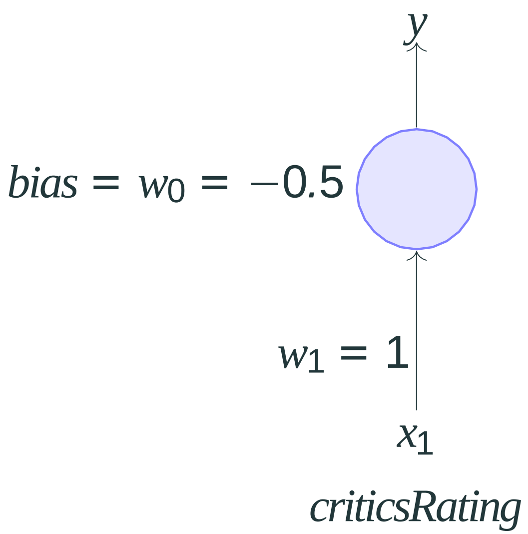
The thresholding logic used by a perceptron is very harsh !

For example, let us return to our problem of deciding whether we will like or dislike a movie

Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)

If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$?

Perceptron



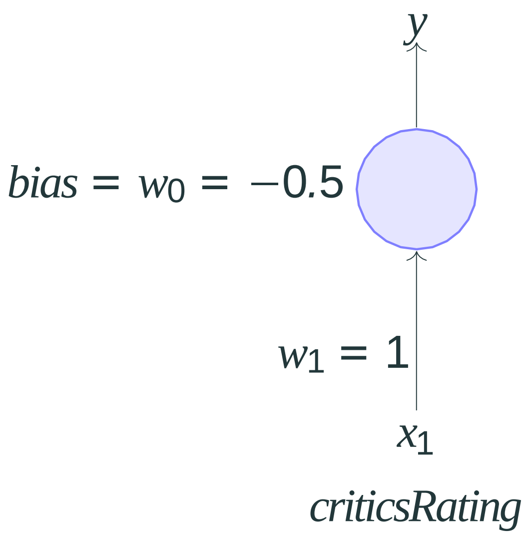
The thresholding logic used by a perceptron is very harsh !

For example, let us return to our problem of deciding whether we will like or dislike a movie

Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)

If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)

Perceptron



The thresholding logic used by a perceptron is very harsh !

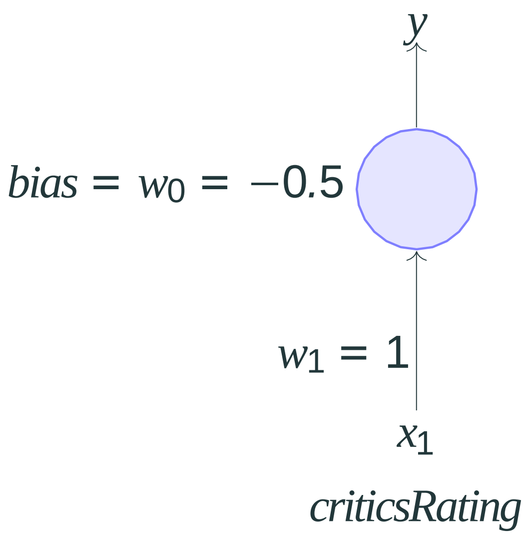
For example, let us return to our problem of deciding whether we will like or dislike a movie

Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)

If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)

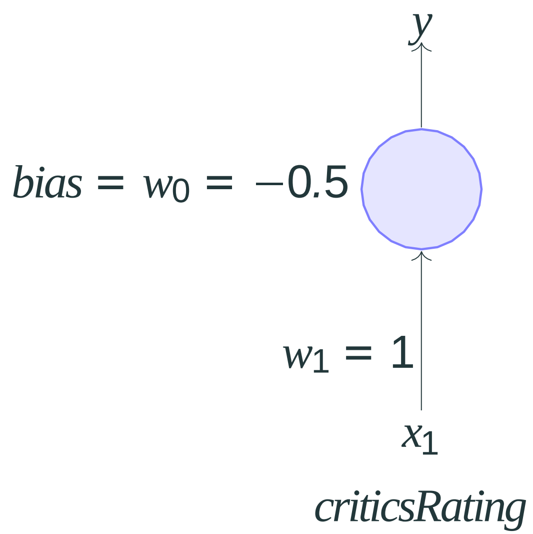
What about a movie with $criticsRating = 0.49$?

Perceptron



- The thresholding logic used by a perceptron is very harsh !
- For example, let us return to our problem of deciding whether we will like or dislike a movie
- Consider that we base our decision only on one input ($x_1 = criticsRating$ which lies between 0 and 1)
- If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)
- What about a movie with $criticsRating = 0.49$? (dislike)

Perceptron



The thresholding logic used by a perceptron is very harsh !

For example, let us return to our problem of deciding whether we will like or dislike a movie

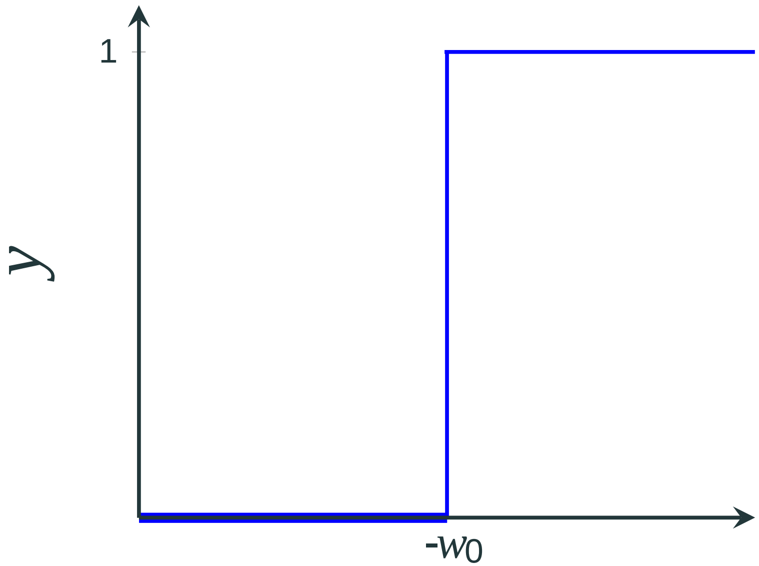
Consider that we base our decision only on one input ($x_1 =$ $criticsRating$ which lies between 0 and 1)

If the threshold is 0.5 ($w_0 = -0.5$) and $w_1 = 1$ then what would be the decision for a movie with $criticsRating = 0.51$? (like)

What about a movie with $criticsRating = 0.49$? (dislike)

It seems harsh that we would like a movie with rat- ing 0.51 but not one with a rating of 0.49

Perceptron



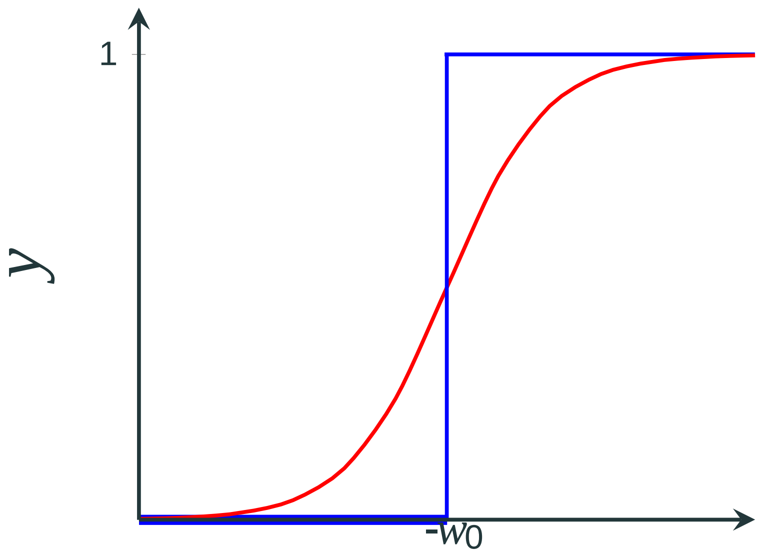
$$z = \sum_{i=1}^{n} w_i x_i$$

It is a characteristic of the perceptron function itself which behaves like a step function

There will always be this sudden change in the decision (from 0 to 1) when $\sum_{i=1}^{n} w_i x_i$ crosses the threshold ($-w_0$)

For most real world applications we would expect a smoother decision function which gradually changes from 0 to 1

Sigmoid neuron



$$z = \sum_{i=1}^{n} w_i x_i$$

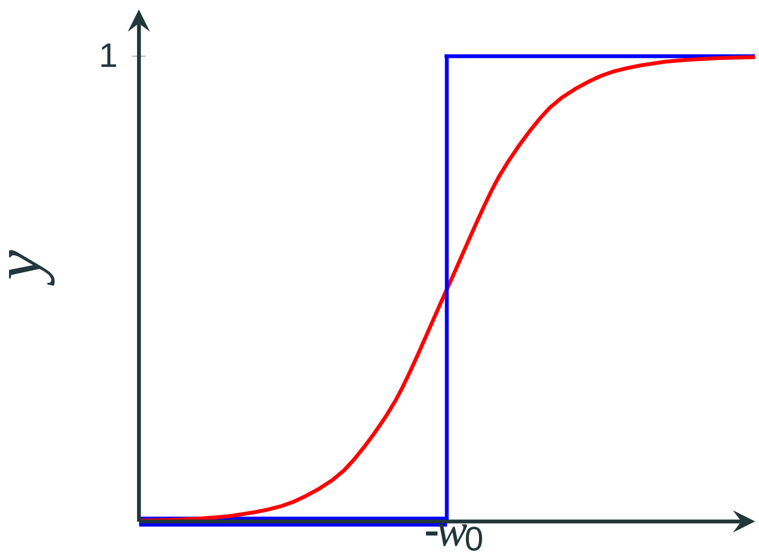
Introducing sigmoid neurons where the output function is much smoother than the step function

Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

We no longer see a sharp transition around the threshold -w_0

Sigmoid neuron



$$z = \sum_{i=1}^{n} w_i x_i$$

Introducing sigmoid neurons where the output function is much smoother than the step function

Here is one form of the sigmoid function called the logistic function

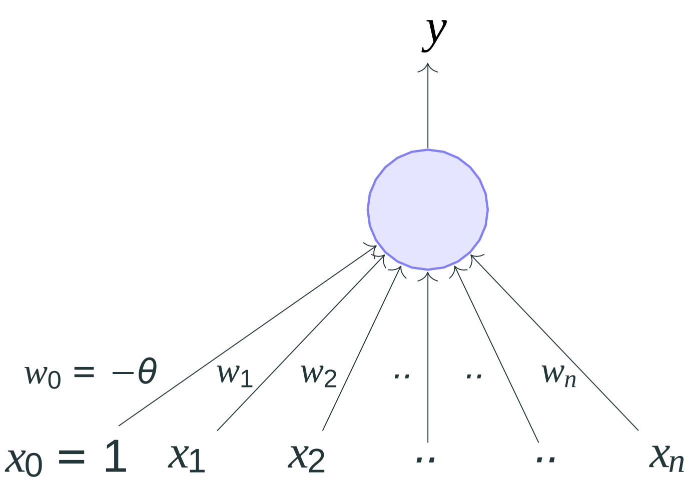
$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

We no longer see a sharp transition around the threshold $-w_0$

Also the output y is no longer binary but a real value between 0 and 1 which can be interpreted as a probability

Instead of a like/dislike decision we get the probability of liking the movie

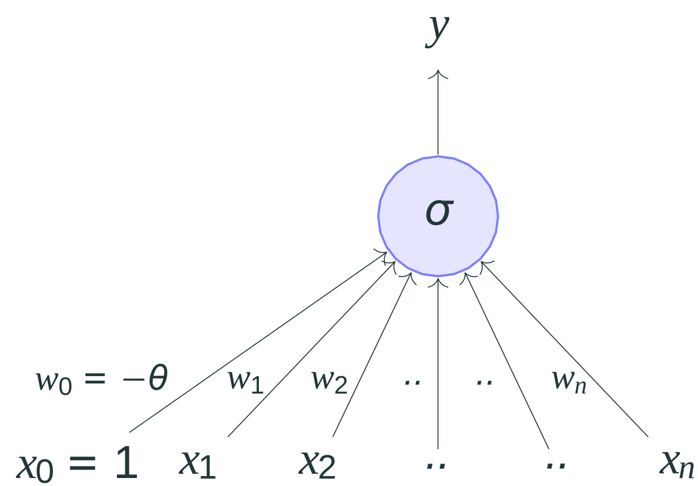
Perceptron vs sigmoid neuron



$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

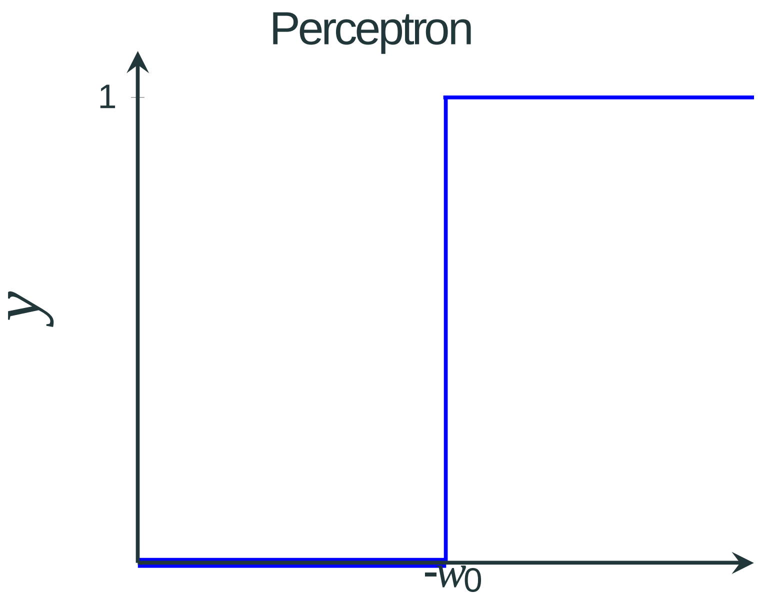
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

Sigmoid (logistic) Neuron



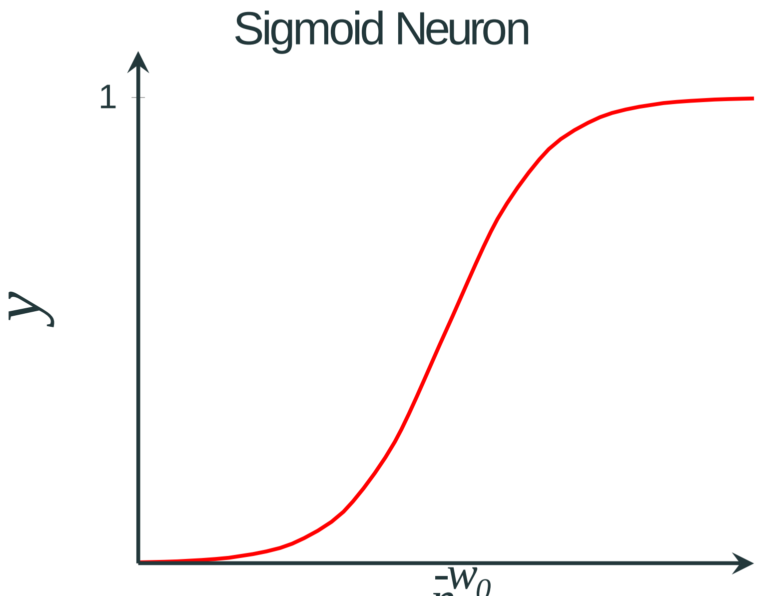
$$y = \frac{1}{1 + e^{-\left(\sum_{i=0}^{n} w_i x_i\right)}}$$

Perceptron vs sigmoid neuron



$$z = \sum_{i=1}^{n} w_i x_i$$

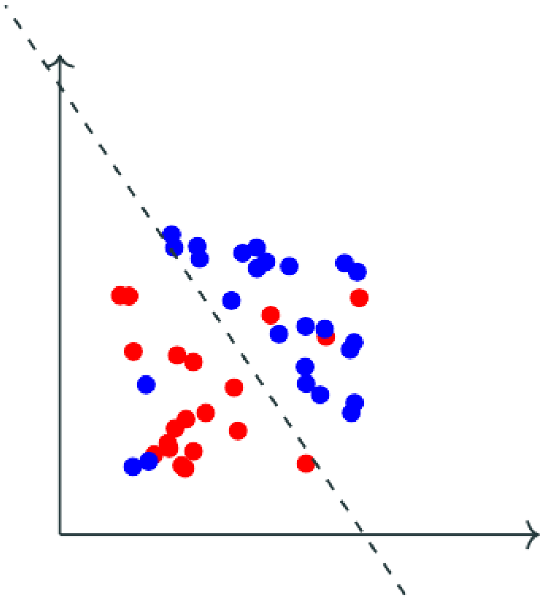
Not smooth, not continuous (at $w0$), **not differentiable**



$$z = \sum_{i=1}^{n} w_i x_i$$

Smooth, continuous, **differentiable**

Typical Supervised ML Setup



Earlier we mentioned that a single perceptron cannot deal with this data because it is not linearly separable

We would probably end up with a line like this ...

This line doesn't seem to be too bad

Sure, it misclassifies 3 blue points and 3 red points but we could live with this error in **most** real world applications

From now on, we will accept that it is hard to drive the error to 0 in most cases and will instead aim to reach the minimum possible error

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

$$or \quad \hat{y} = \mathbf{w}^T\mathbf{x}$$

$$or \quad \hat{y} = \mathbf{x}^T\mathbf{W}\mathbf{x}$$

or just about any function

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between $\mathbf{x}$ and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

$$or \quad \hat{y} = \mathbf{w}^T\mathbf{x}$$

$$or \quad \hat{y} = \mathbf{x}^T\mathbf{W}\mathbf{x}$$

or just about any function

Parameters: In all the above cases, w is a parameter which needs to be learned from the data

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

$$or \quad \hat{y} = \mathbf{w}^T\mathbf{x}$$

$$or \quad \hat{y} = \mathbf{x}^T\mathbf{W}\mathbf{x}$$

or just about any function

Parameters: In all the above cases, w is a parameter which needs to be learned from the data

Learning algorithm: An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

$$or \quad \hat{y} = \mathbf{w}^T\mathbf{x}$$

$$or \quad \hat{y} = \mathbf{x}^T\mathbf{W}\mathbf{x}$$

or just about any function

Parameters: In all the above cases, w is a parameter which needs to be learned from the data

Learning algorithm: An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

Objective/Loss/Error function: To guide the learning algorithm

Supervised ML setup

Data: $\{x_i, y_i\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

$$or \quad \hat{y} = \mathbf{w}^T\mathbf{x}$$

$$or \quad \hat{y} = \mathbf{x}^T\mathbf{W}\mathbf{x}$$

or just about any function

Parameters: In all the above cases, w is a parameter which needs to be learned from the data

Learning algorithm: An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

Objective/Loss/Error function: To guide the learning algorithm

the learning algorithm should aim to minimize the loss function

Supervised ML setup: example

As an illustration, consider our movie example

Data: $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

Supervised ML setup: example

As an illustration, consider our movie example

Data: $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

Model: Our approximation of the relation between $\mathbf{x}$ and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

Supervised ML setup: example

As an illustration, consider our movie example

Data: $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

Parameter: w

Supervised ML setup: example

As an illustration, consider our movie example

Data: $\{x_i = movie, y_i = like/dislike\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

Parameter: **w**

Learning algorithm: Gradient Descent [we will see soon]

Supervised ML setup: example

As an illustration, consider our movie example

Data: $\{ x_i = movie, y_i = like/\ dislike\}_{i=1}^{n}$

Model: Our approximation of the relation between **x** and y (the probability of liking a movie).

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x})}}$$

Parameter: **w**

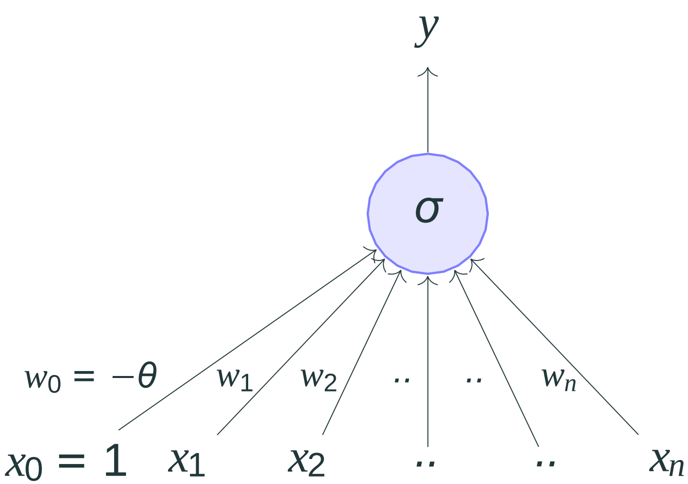
Learning algorithm: Gradient Descent [we will see soon]

Objective/Loss/Error function: One Possibility is

$$L(\mathbf{w}) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

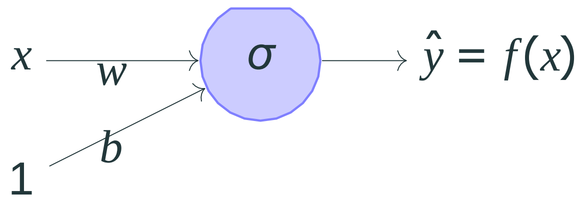
The learning algorithm should aim to find a w which minimizes the above function (squared error between y and $\hat{y}$)

Learning Parameters: guess work



$$f(x) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

Keeping this supervised ML setup in mind, we will now focus on this **model** and discuss an **algorithm** for learning the **parameters** of this model from some given **data** using an appropriate **objective function**

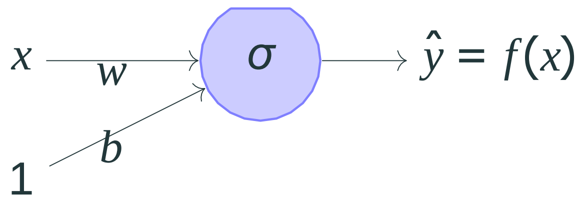


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

For ease of explanation, we will consider a very simplified version of the model having just 1 input

Further to be consistent with the literature, from now on, we will refer to w_0 as b (bias)

Lastly, instead of considering the problem of predicting like/dislike, we will assume that we want to predict $criticsRating(y)$ given $imdbRating(x)$ (for no particular reason)



$$f(x) = \frac{1}{1+e^{-(w\cdot x+b)}}$$

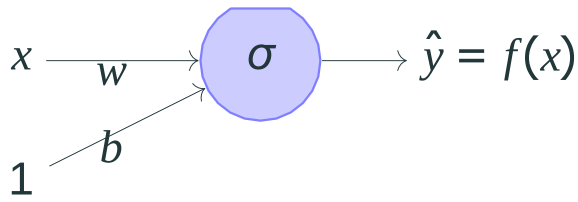
Input for training

$\{x_i, y_i\}_{i=1}^{N} \rightarrow N$ pairs of (x, y)

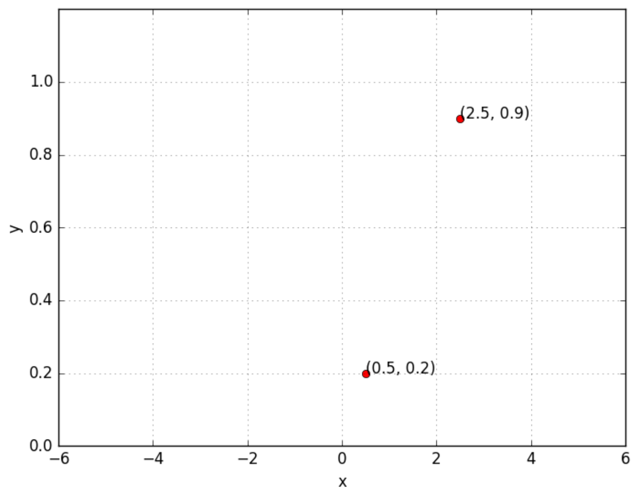
Training objective

Find w and b such that:

$$\underset{w,b}{\text{minimize}}\, L\ (w, b) = \sum_{i=1}^{N} (y_i - f(x_i))^2$$

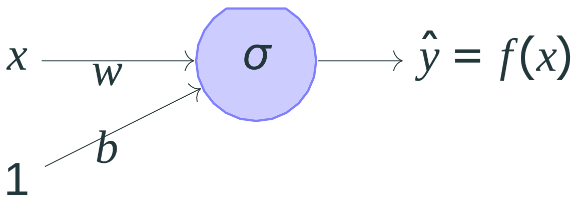


$$f(x) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

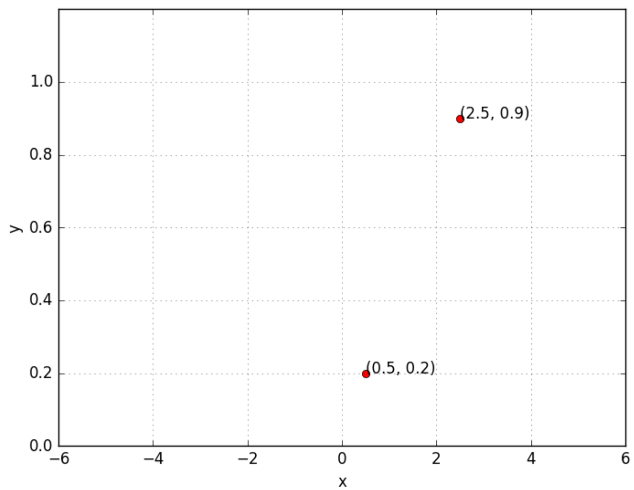


What does it mean to train the network?

Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$



$$f(x) = \frac{1}{1+e^{-(w\cdot x+b)}}$$



What does it mean to train the network?

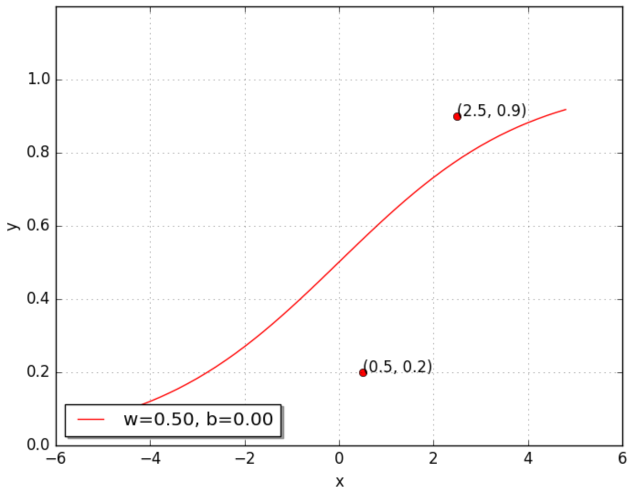
Suppose we train the network with $(x, y) = (0.5, 0.2)$ and $(2.5, 0.9)$

At the end of training we expect to find w*, b* such that:

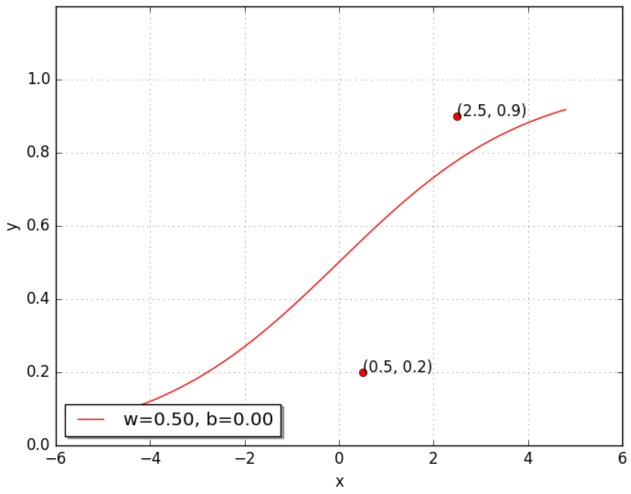
$f(0.5) \rightarrow 0.2$ and $f(2.5) \rightarrow 0.9$

Can we try to find such a w^*, b^* manually

Let us try a random guess.. (say, $w = 0.5, b = 0$)



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



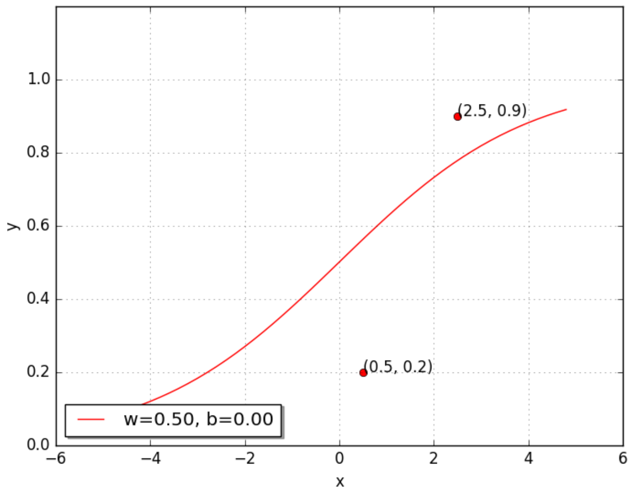
Can we try to find such a w^*, b^* manually

Let us try a random guess.. (say, $w = 0.5, b = 0$)

Clearly not good, but how bad is it ?

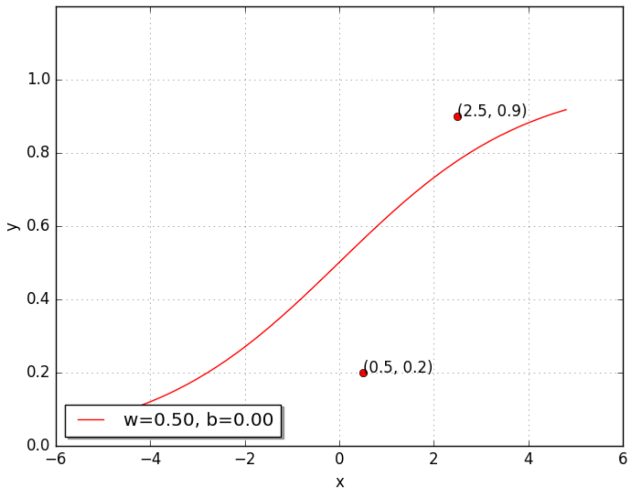
Let us revisit $\mathscr{L}(w, b)$ to see how bad it is ...

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



$$L\ (w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$

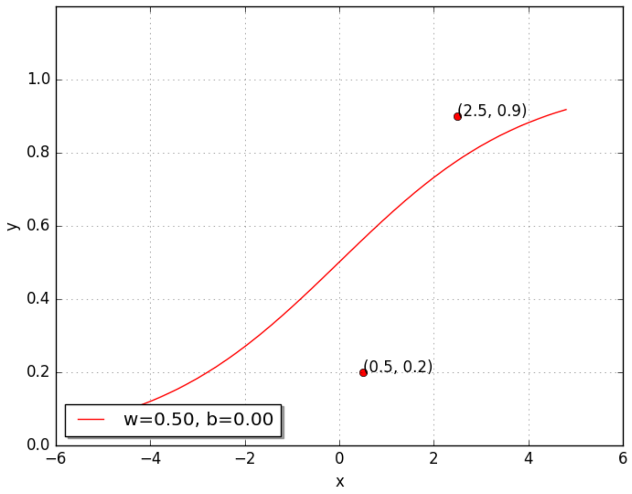
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$



$$L(w,b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$

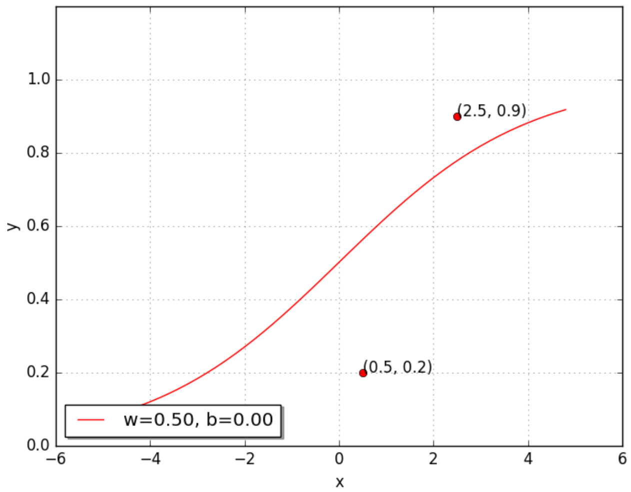
$$= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2$$

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



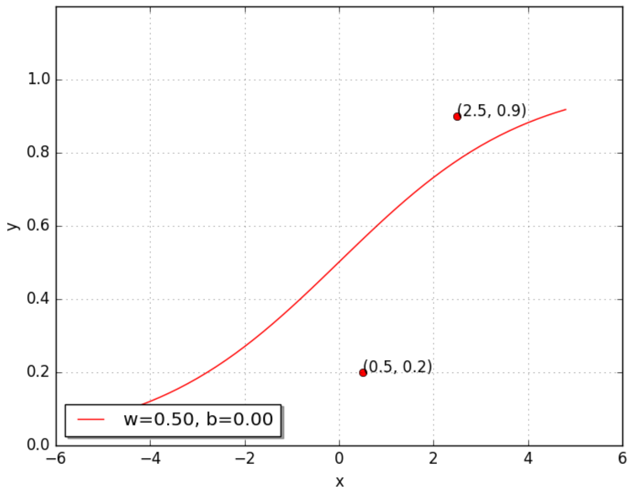
$$L\ (w, b) = \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2$$
$$= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2$$
$$= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2$$

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



$$
\begin{aligned}
L\ (w,b) &= \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2 \\
&= \frac{1}{2} * (y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\
&= \frac{1}{2} * (0.9 - f(2.5))^2 + (0.2 - f(0.5))^2 \\
&= 0.073
\end{aligned}
$$

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

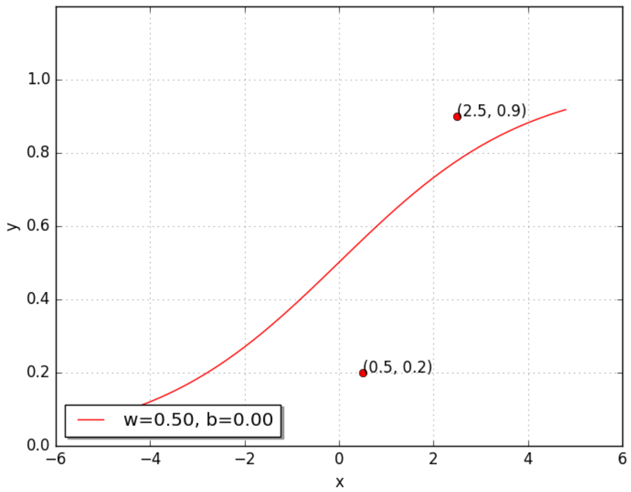


$$
\begin{aligned}
L\ (w, b) &= \frac{1}{2} * \sum_{i=1}^{N} (y_i - f(x_i))^2 \\
&= \frac{1}{2} *(y_1 - f(x_1))^2 + (y_2 - f(x_2))^2 \\
&= \frac{1}{2} *(0.9 - f(2.5))^2 + (0.2 - f(0.5))^2 \\
&= 0.073
\end{aligned}
$$

We want $L\ (w, b)$ to be as close to 0 as possible

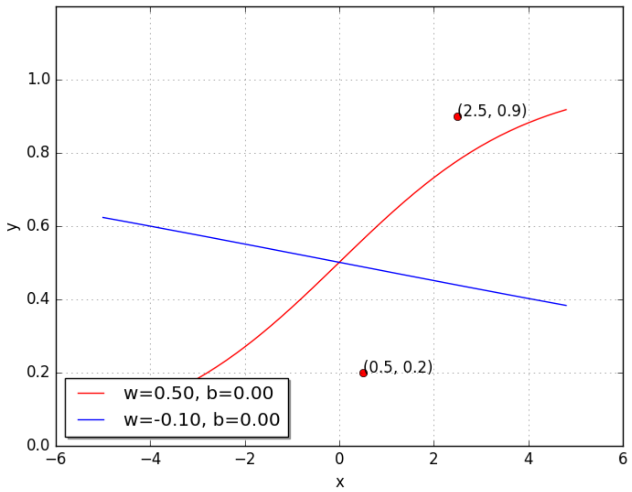
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$

Let us try some other values of w, b



w	b	$\mathcal{L}(w, b)$
0.50	0.00	0.0730

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



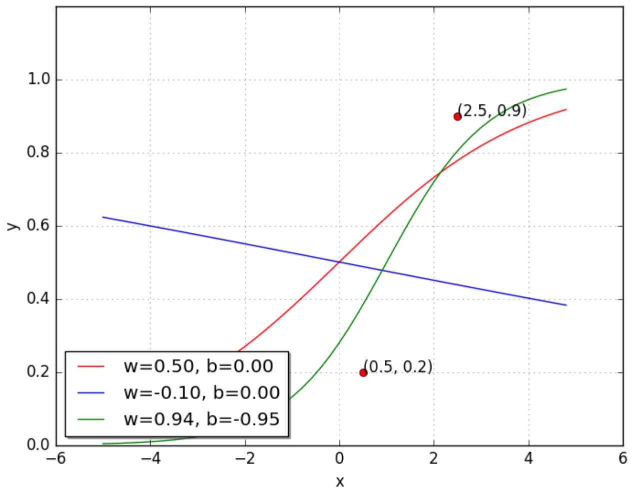
Let us try some other values of w, b

w	b	$\mathscr{L}(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481

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

Oops!! this made things even worse...

Let us try some other values of w, b

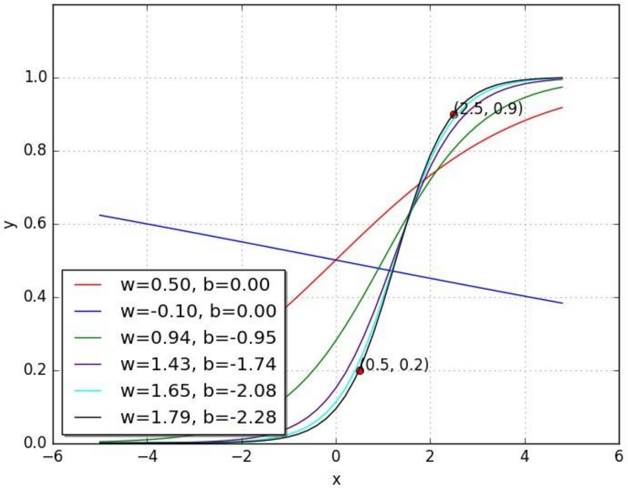


w	b	$L(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214

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

Perhaps it would help to push w and b in the other direction...

Let us try some other values of w, b

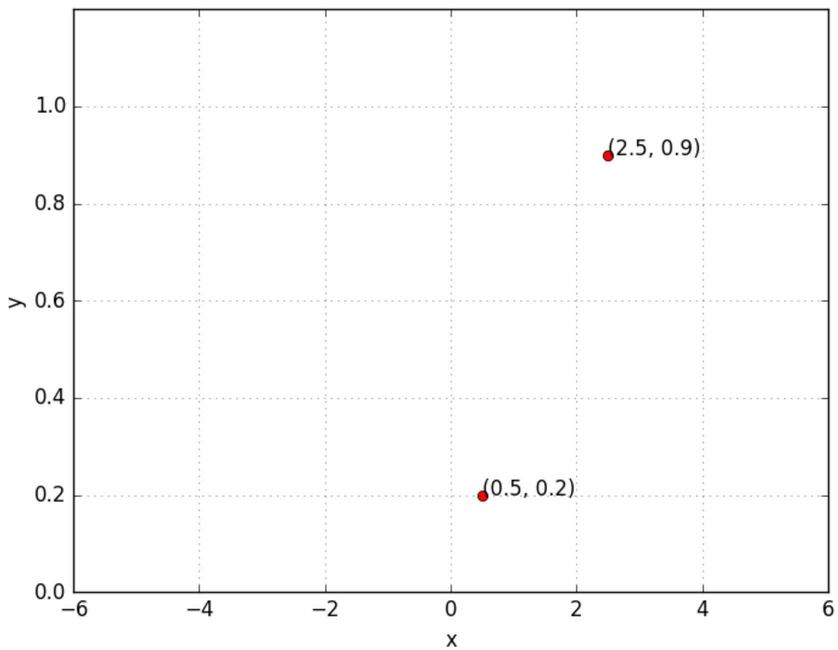


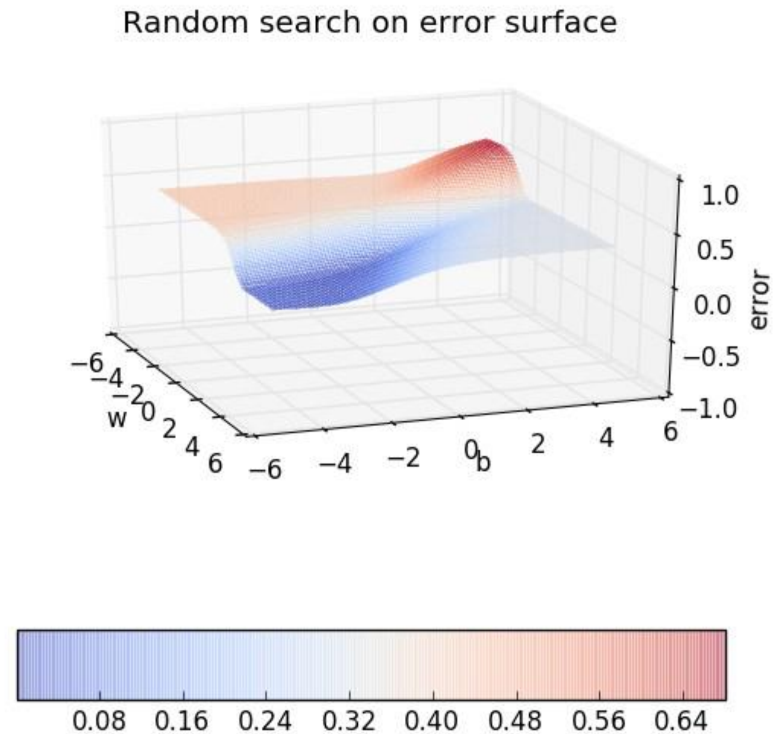
w	b	$L(w, b)$
0.50	0.00	0.0730
-0.10	0.00	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

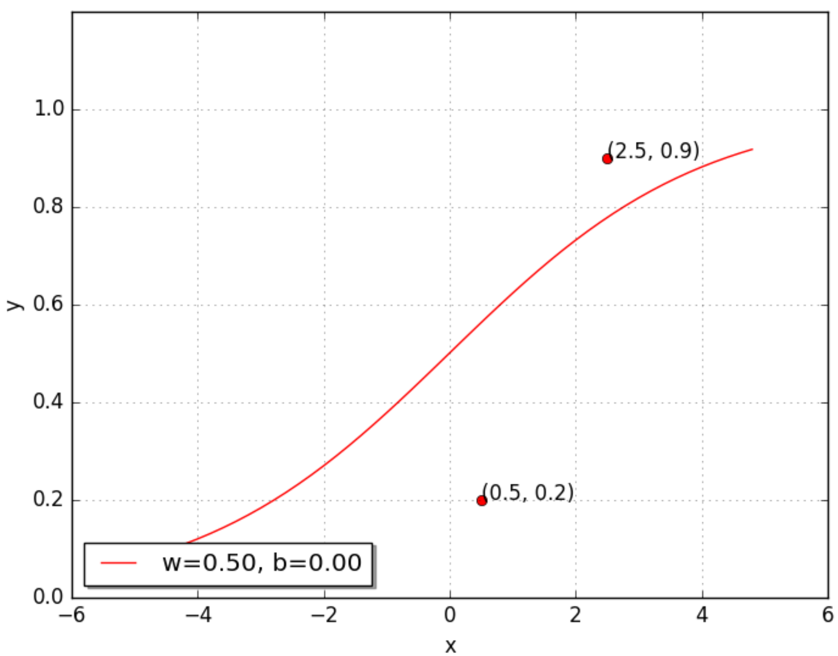
$$\sigma(x) = \frac{1}{1 + e^{-(wx+b)}}$$

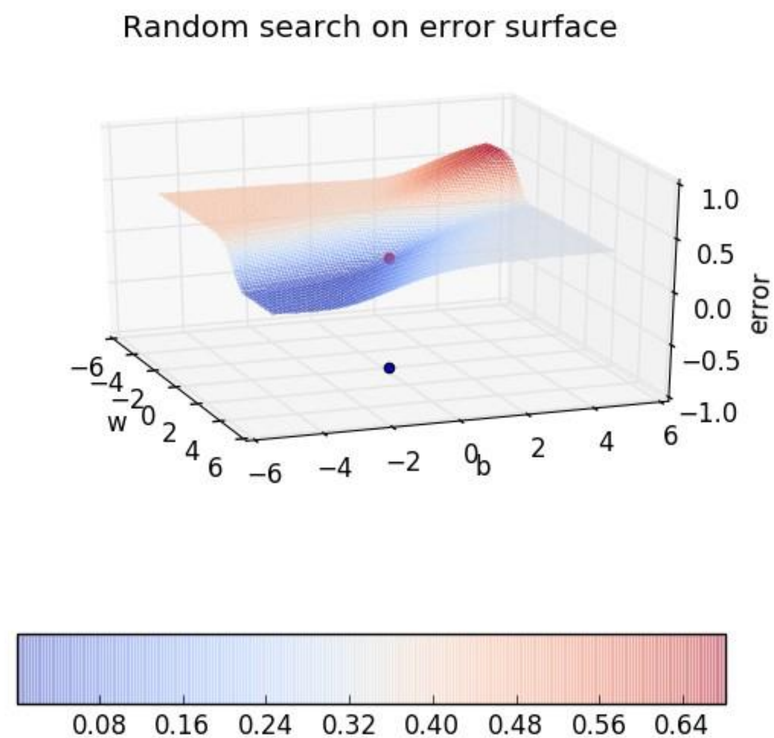
With some guess work and intuition we were able to find the right values for w and b

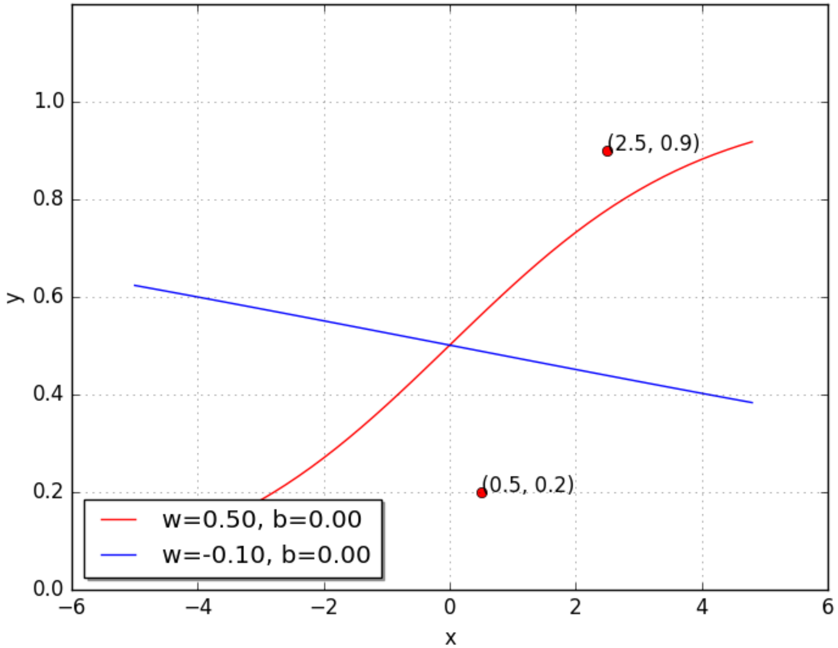
Let us look at the geometric interpretation of our "guess work" algorithm in terms of this error surface


(2.5, 0.9)
(0.5, 0.2)
x
y

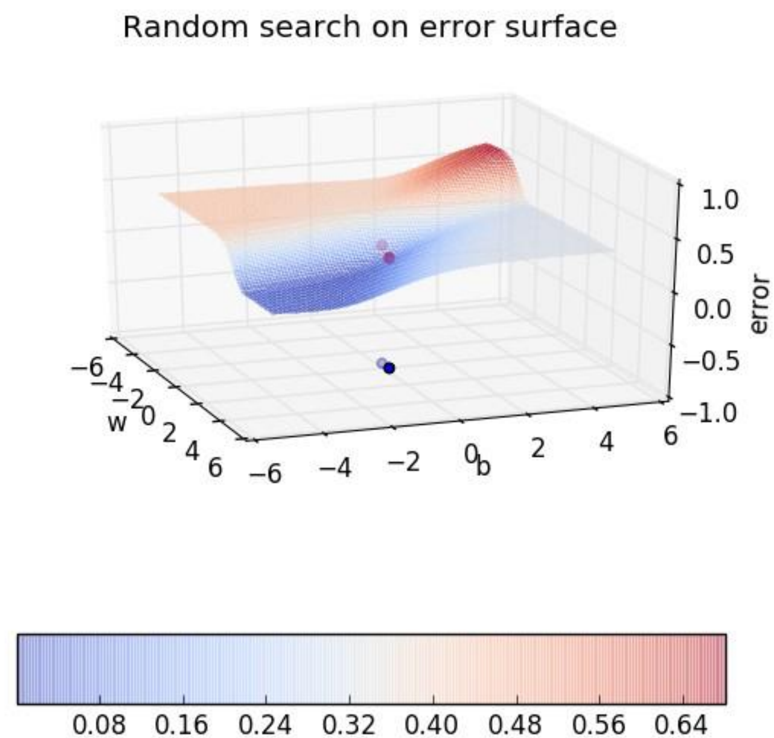

Random search on error surface
w
b
error


(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
x
y

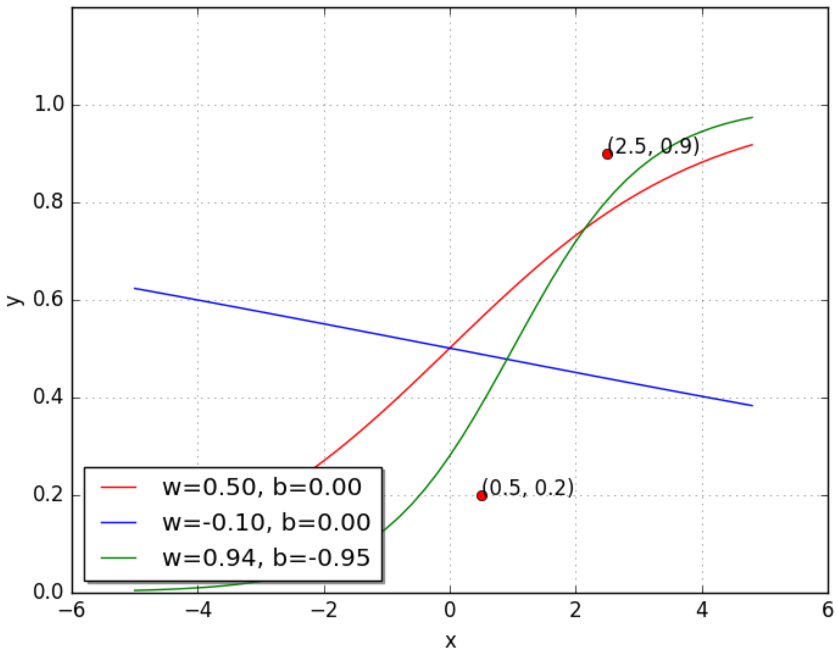

Random search on error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



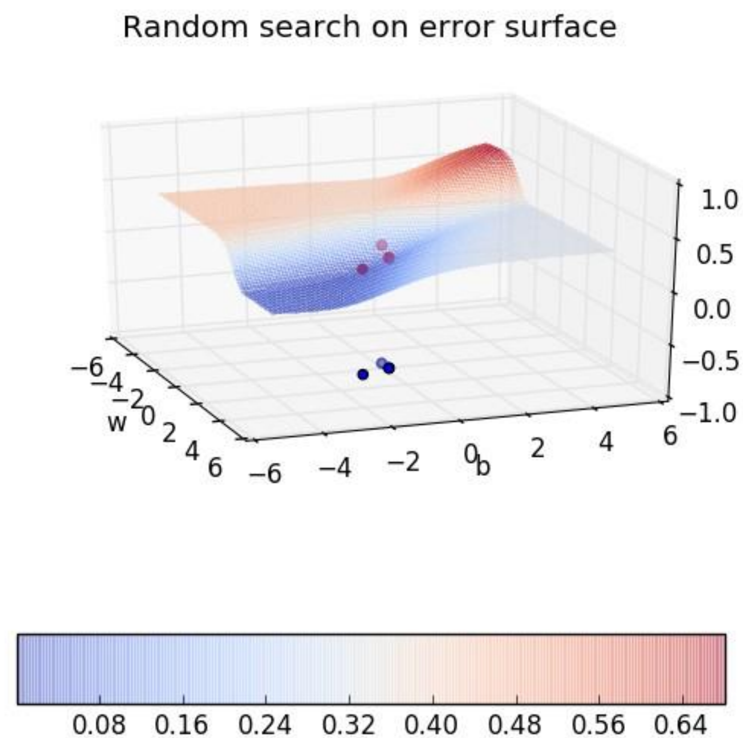
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
y
x



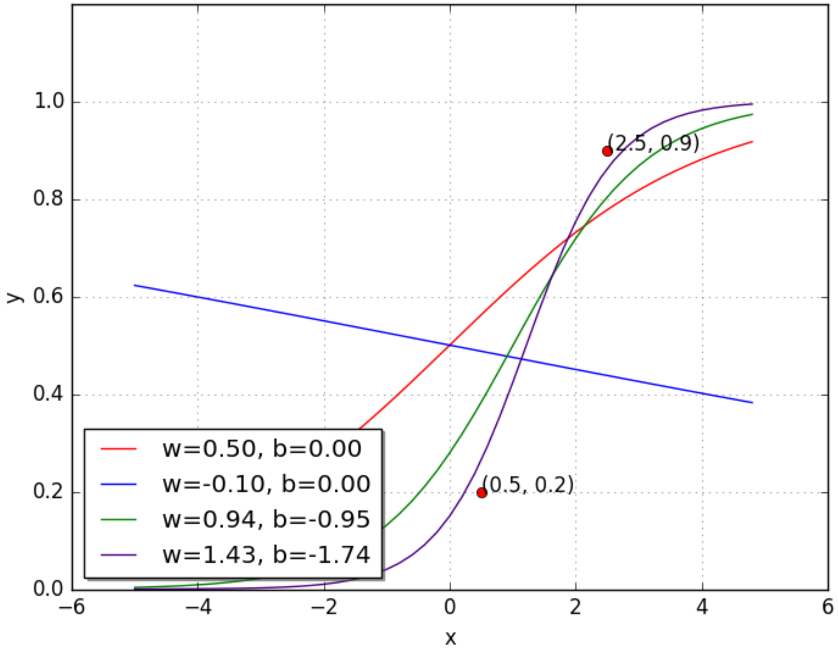
Random search on error surface
error
w
b
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



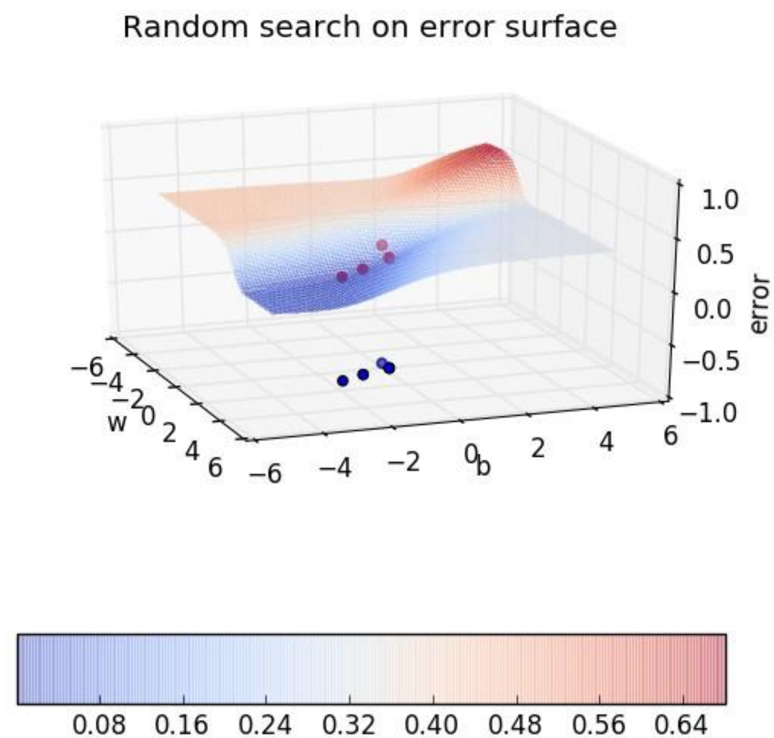
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
y
x



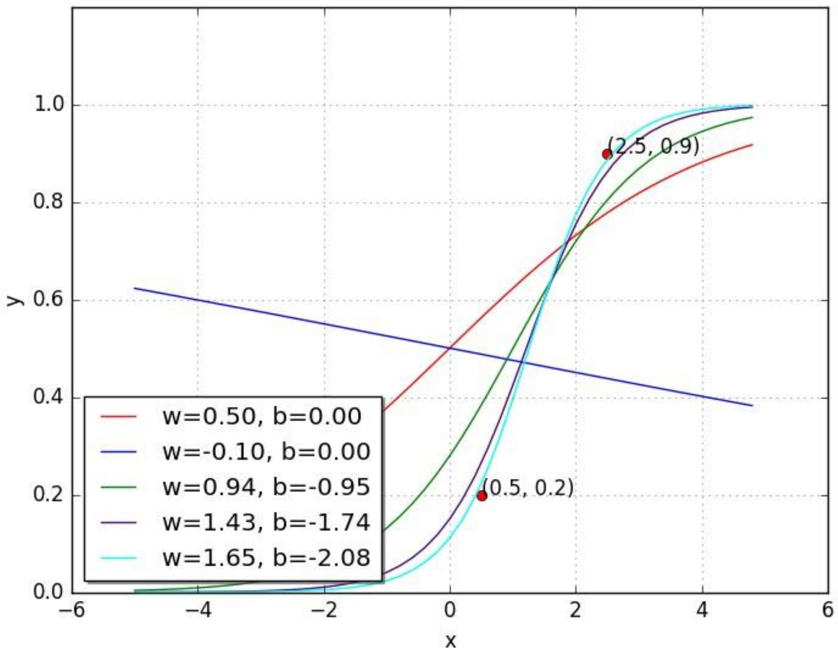
Random search on error surface
w
b
error
0.08 0.16 0.24 0.32 0.40 0.48 0.56 0.64



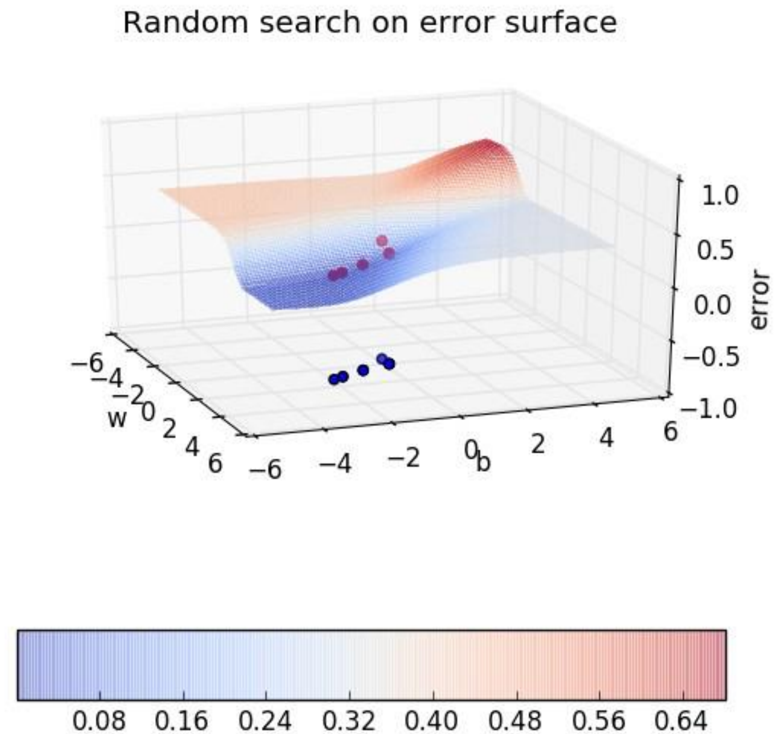
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
y
x



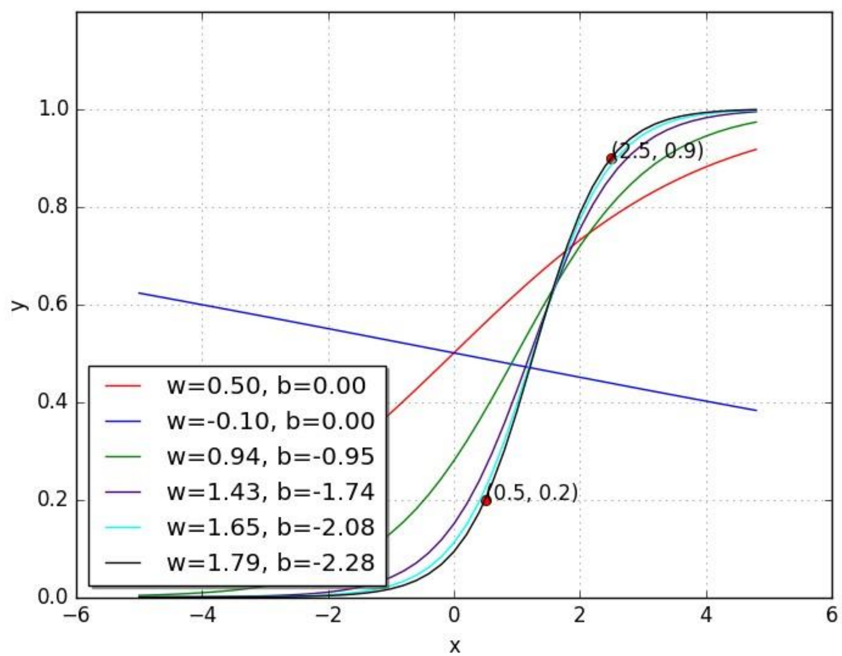
Random search on error surface
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64

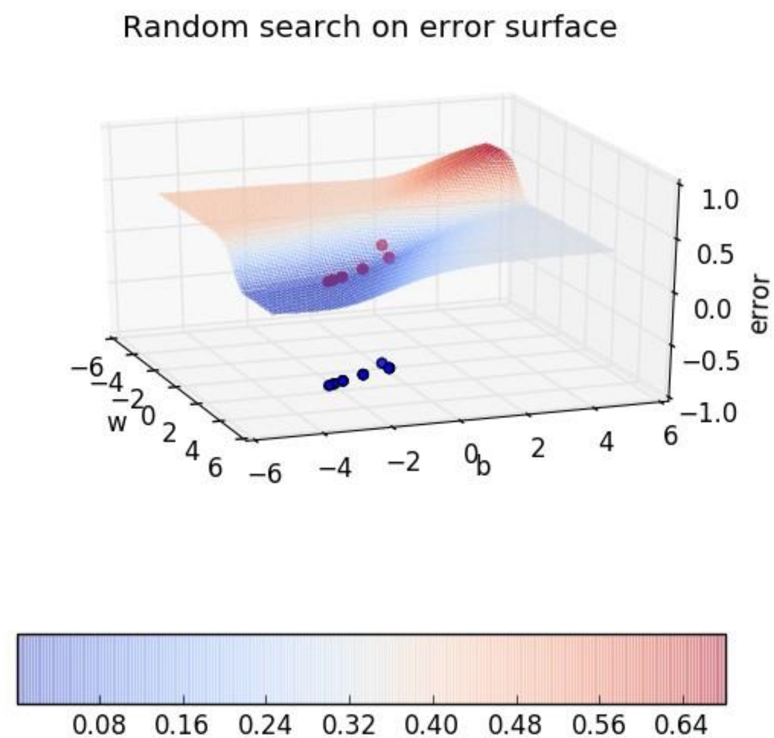


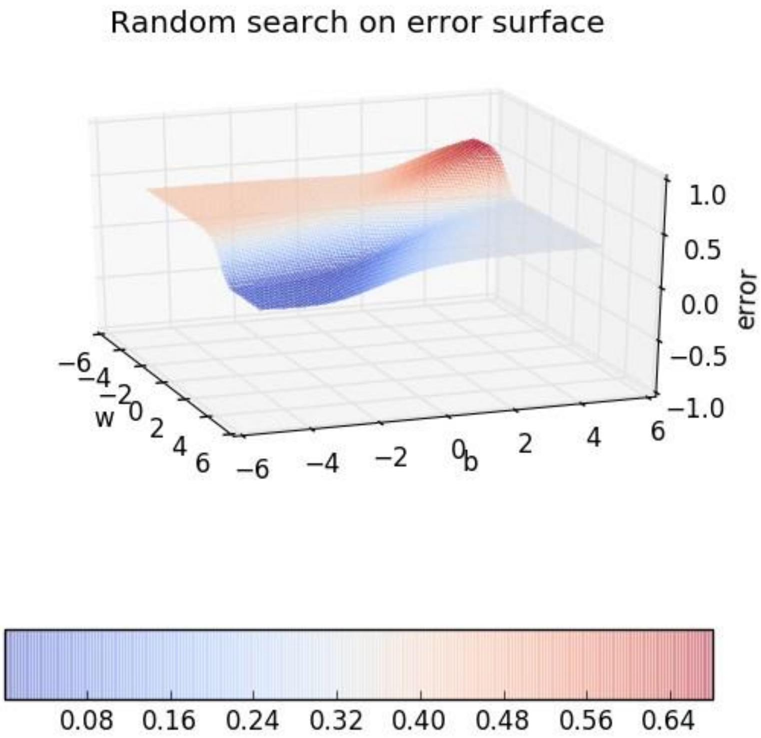
1.0
0.8
0.6
0.4
0.2
0.0
y
−6
−4
−2
0
2
4
6
x
(2.5, 0.9)
(0.5, 0.2)
w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
w=1.65, b=-2.08



Random search on error surface
1.0
0.5
0.0
−0.5
−1.0
error
w
b
0.08
0.16
0.24
0.32
0.40
0.48
0.56
0.64


w=0.50, b=0.00
w=-0.10, b=0.00
w=0.94, b=-0.95
w=1.43, b=-1.74
w=1.65, b=-2.08
w=1.79, b=-2.28
(2.5, 0.9)
(0.5, 0.2)
x
y


Random search on error surface
error
w
b



Since we have only 2 points and 2 parameters (w, b) we can easily plot $\mathscr{L}(w, b)$ for different values of (w, b) and pick the one where $\mathscr{L}(w, b)$ is minimum

But of course this becomes intractable once you have many more data points and many more parameters !!

Further, even here we have plotted the error surface only for a small range of (w, b) [from $(-6, 6)$ and not from $(-\inf, \inf)$]

- **End of topic**